Home Credit Default Risk Part 1: Introduction and EDA

Introduction

Many people struggle to get loans due to insufficient or non-existent credit histories. And, unfortunately, this population is often taken advantage of by untrustworthy lenders.

Home Credit strives to broaden financial inclusion for the unbanked population by providing a positive and safe borrowing experience. In order to make sure this underserved population has a positive loan experience, Home Credit makes use of a variety of alternative data--including telco and transactional information--to predict their clients' repayment abilities.

This dataset is available on Kaggle: Home Credit Default Risk.

Table of contents

Data Description

There are 10 tables:

`application_{train|test}.csv`

- This is the main table, broken into two files for Train (with TARGET) and Test (without TARGET).
- Static data for all applications. One row represents one loan in our data sample.

`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).
- For every loan in our sample, there are as many rows as number of credits the client had in Credit Bureau before the application date.

`bureau_balance.csv`

- Monthly balances of previous credits in Credit Bureau.
- This table has one row for each month of history of every previous credit reported to Credit Bureau – i.e the table has (#loans in sample *# of relative previous credits* # of months where we have some history observable for the previous credits) rows.

`POS_CASH_balance.csv`

- Monthly balance snapshots of previous POS (point of sales) and cash loans that the applicant had with Home Credit.
- This table has one row for each month of history of every previous credit in Home Credit (consumer credit and cash loans) related to loans in our sample – i.e. the table has (#loans in sample *# of relative previous credits* # of months in which we have some history observable for the previous credits) rows.

`credit_card_balance.csv`

- Monthly balance snapshots of previous credit cards that the applicant has with Home Credit.
- This table has one row for each month of history of every previous credit in Home Credit (consumer credit and cash loans) related to loans in our sample – i.e. the table has (#loans in sample *# of relative previous credit cards* # of months where we have some history observable for the previous credit card) rows.

`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.

`installments_payments.csv`

- Repayment history for the previously disbursed credits in Home Credit related to the loans in our sample.
- There is a) one row for every payment that was made plus b) one row each for missed payment.
- 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.

`HomeCredit_columns_description.csv`

- This file contains descriptions for the columns in the various data files.

Structure of relational tables

Loading libries

In [2]:
```
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
import seaborn as sns
import phik

import warnings
warnings.filterwarnings('ignore')

import plotly
import plotly.express as px
import plotly.graph_objs as go
from plotly.subplots import make_subplots
from datetime import datetime

#pd.DataFrame columns and rows limits
pd.set_option('max_columns', None)
pd.set_option('max_rows', None)
```

1. Defining utility functions

In [3]:
```
def load_all_tables(data_path='./data/', verbose=True):
    '''
    Function to load all tables
    Input:
        data_path: str, default='./data/'
        verbose: bool, default=True
    '''
    if verbose:
        print('Loading all tables...')
        start = datetime.now()
```

```
    #making all the variables global
    global app_train, app_test, bureau, bureau_balance, cc_balance, install_pay, \
            POS_CASH_balance, pre_app

    app_train = pd.read_csv(data_path+'application_train.csv')
    if verbose: print('1 load app_train.csv')

    app_test = pd.read_csv(data_path+'application_test.csv')
    if verbose: print('2 load 2 app_test.csv')

    bureau = pd.read_csv(data_path+'bureau.csv')
    if verbose: print('3 load bureau.csv')

    bureau_balance = pd.read_csv(data_path+'bureau_balance.csv')
    if verbose: print('4 load bureau_balance.csv')

    cc_balance = pd.read_csv(data_path+'credit_card_balance.csv')
    if verbose: print('5 load credit_card_balance.csv')

    install_pay = pd.read_csv(data_path+'installments_payments.csv')
    if verbose: print('6 load installments_payments.csv')

    POS_CASH_balance = pd.read_csv(data_path+'POS_CASH_balance.csv')
    if verbose: print('7 load POS_CASH_balance.csv')

    pre_app = pd.read_csv(data_path+'previous_application.csv')
    if verbose:
        print('8 load previous_application.csv')
        print('Done')
        print('Time to load files: ', datetime.now()-start)
```

In [4]:

```
def nan_df_create(data):
    '''
    Function to create a dataframe of percentage of NaN values for each column of the d

    Inputs:
        data: DataFrame

    Return:
        DataFrame of NaN percentages
    '''
    nan_percentages = data.isna().sum() * 100 / len(data)
    df = pd.DataFrame({'column': nan_percentages.index, 'percent': nan_percentages.valu
    df.sort_values(by = 'percent', ascending=False, inplace=True)

    return df
```

In [5]:

```
def plot_nan_percent(df_nan, title_name, tight_layout=True, figsize=(20,8),
                     grid=False, rotation=90):
    '''
    Function to plot Bart plots of NaN percentages for each column with missing values

    Inputs:
        df_nan:
            DataFrame of NaN percentages
        title_name:
            Name of table to be displayed in title of plot
```

```
        tight_layout: bool, default=True
            Whether to keep tight layout or not
        figsize: tuple, default=(20,8)
            Figure size of plot
        grid: bool, default=False
            Whether to draw gridlines to plot or not
        rotation: int, default = 0
            Degree of rotation for x-tick labels
    '''
    #checking if there is any column with NaNs or not
    if df_nan.percent.sum() != 0:
        print(f"Number of columns having NaN values: {df_nan[df_nan['percent'] != 0].sh

        #plotting bar-plot for NaN percentages (only for columns with non-zeros NaN per
        plt.figure(figsize=figsize, tight_layout=tight_layout)
        sns.barplot(x = 'column', y = 'percent', data = df_nan[df_nan['percent'] > 0])
        plt.xticks(rotation = rotation)
        plt.xlabel('Column name')
        plt.ylabel('Percentage of NaN values')
        plt.title(f'Percentage of NaN values in {title_name}')
        if grid:
            plt.grid()
        plt.show()
    else:
        print(f'The dataframe {title_name} does not contain any NaN values')
```

In [6]:

```
class correlation_matrix:
    '''
    Class to plot heatmap of Correlation Matrix and print top correlated features with
    Contains three methods:
        1. init method
        2. plot_correlation_matrix_method
        3. target_top_corr_method
    '''
    def __init__(self, data, columns_to_drop, figsize=(25,23), mask_upper=True,
                 tight_layout=True, linewidth=0.1, fontsize=10, cmap='Blues'):
        '''
        Function to initialize the class members

        Inputs:
            data: DataFrame
                The DataFrame from which to build correlation matrix
            columns_to_drop: list
                Columns which have to be dropped while building the correlation matrix
            figsize: tuple, defaul=(25,23)
                Size of the figure to be plotted
            mask_upper: bool, default=True
                Whether to plot only the lower triangle of heatmap or plot full
            tight_layout: bool, default=True
                Whether to keep tight layout or not
            linewidth: float/int, default=0.1
                The linewidth to use for heatmap
            fontsize: int, default=10
                The font size for the X and Y tick labels
            cmap: str, default='Blue'
                The colormap to be used for heatmap
        Returns:
            None
```

```
        '''
        self.data = data
        self.columns_to_drop = columns_to_drop
        self.figsize = figsize
        self.mask_upper = mask_upper
        self.tight_layout = tight_layout
        self.linewidth = linewidth
        self.fontsize = fontsize
        self.cmap = cmap

    def plot_correlation_matrix(self):
        '''
        Function to plot the correlation matrix heatmap

        Inputs:
            self
        Returns:
            None
        '''
        print('-' * 100)
        #building the correlation dataframe
        self.corr_data = self.data.drop(self.columns_to_drop + ['TARGET'], axis=1).corr

        if self.mask_upper:
            #masking the heatmap to show only lower triangle, this is to save RAM
            mask_array = np.ones(self.corr_data.shape)
            mask_array = np.triu(mask_array) #
        else:
            mask_array = np.zeros(self.corr_data.shape)

        plt.figure(figsize = self.figsize, tight_layout = self.tight_layout)
        sns.heatmap(self.corr_data, annot=False, mask=mask_array,
                    linewidth=self.linewidth, cmap=self.cmap)
        #mask=mask_array, don't show where mask is true
        plt.xticks(rotation=90, fontsize=self.fontsize)
        plt.yticks(fontsize=self.fontsize)
        plt.title('Correlation heatmap for numerical features')
        plt.show()
        print('-' * 100)

    def target_top_corr(self, target_top_columns=10):
        '''
        Function to show the top correlated features with the target

        Inputs:
            self
            target_top_columns: int, default=10
                The number of top correlated features with target to display

        Returns:
            Top correlated features DataFrame
        '''

        phik_target_arr = np.zeros(self.corr_data.shape[1])
        #calculating the Phik-Correlation with target
        for index, column in enumerate(self.corr_data.columns):
            phik_target_arr[index] = self.data[['TARGET', column]].phik_matrix().iloc[0
        #getting the top correlated columns and their values
        top_corr_target_df = pd.DataFrame({'Column Name': self.corr_data.columns,
                                           'Phik-Correlation': phik_target_arr})
```

```
        top_corr_target_df = top_corr_target_df.sort_values(by = 'Phik-Correlation',
                                                            ascending = False)

        return top_corr_target_df.iloc[:target_top_columns]
```

In [7]:

```
def plot_phik_matrix(data, categorical_columns, figsize=(20,20), mask_upper=True,
                     tight_layout=True, linewidth=0.1, fontsize=10, cmap='Blues',
                     show_target_top_corr=True, target_top_columns=10):
    '''
    Function to plot Phi_k matrix for categorical features

    Inputs:
        data: DataFrame
            The DataFrame from which to build correlation matrix
        categorical_columns: list
            List of categorical columns whose PhiK values are to be plotted
        figsize: tuple, default = (25,23)
            Size of the figure to be plotted
        mask_upper: bool, default = True
            Whether to plot only the lower triangle of heatmap or plot full.
        tight_layout: bool, default = True
            Whether to keep tight layout or not
        linewidth: float/int, default = 0.1
            The linewidth to use for heatmap
        fontsize: int, default = 10
            The font size for the X and Y tick labels
        cmap: str, default = 'Blues'
            The colormap to be used for heatmap
        show_target_top_corr: bool, default = True
            Whether to show top/highly correlated features with Target.
        target_top_columns: int, default = 10
            The number of top correlated features with target to display
    '''
    ###fetching only the categorical collumns
    data_for_phik = data[categorical_columns].astype('object')
    phik_matrix = data_for_phik.phik_matrix()

    print('-' * 100)

    if mask_upper:
        #masking the heatmap to show only lower triangle, this is to save RAM
        mask_array = np.ones(phik_matrix.shape)
        mask_array = np.triu(mask_array) #
    else:
        mask_array = np.zeros(phik_matrix.shape)

    plt.figure(figsize = figsize, tight_layout = tight_layout)
    sns.heatmap(phik_matrix, annot=False, mask=mask_array, linewidth=linewidth, cmap=cm
    #mask=mask_array, don't show where mask is true
    plt.xticks(rotation=90, fontsize=fontsize)
    plt.yticks(rotation=0, fontsize=fontsize)
    plt.title('Phi-K correlation heatmap for categorical features')
    plt.show()
    print('-' * 100)

    if show_target_top_corr:
        #display the top columns with highest correlation with the target variable in a
        print('Categorise with highest values of Phi-K correlation value with target va
```

```
        phik_df = pd.DataFrame({'Column Name': phik_matrix.TARGET.index[1:],
                                'Phik-Correlation': phik_matrix.TARGET.values[1:]})
        phik_df =phik_df.sort_values(by='Phik-Correlation', ascending=False)
        display(phik_df.head(target_top_columns))
        print('-' * 100)
```

In [8]:

```
def plot_categorical_variables_bar(data, column_name, figsize=(18,6), percentage_displa
                                   rotation=0, horizontal_adjust=0, fontsize_percent='xx
    '''
    Function to plot Categorical Variables Bar Plots

    Inputs:
        data: DataFrame
            The DataFrame from which to plot
        column_name: str
            Column's name whose distribution is to be plotted
        figsize: tuple, default = (18,6)
            Size of the figure to be plotted
        percentage_display: bool, default = True
            Whether to display the percentages on top of Bars in Bar-Plot
        plot_defaulter: bool
            Whether to plot the Bar Plots for Defaulters or not
        rotation: int, default = 0
            Degree of rotation for x-tick labels
        horizontal_adjust: int, default = 0
            Horizontal adjustment parameter for percentages displayed on the top of Bar
        fontsize_percent: str, default = 'xx-small'
            Fontsize for percentage Display

    '''
    print(f'Total number of unique categories of {column_name} = {len(data[column_name]

    plt.figure(figsize=figsize, tight_layout=False)
    sns.set(style='whitegrid', font_scale=1.2)

    #plotting overal distribution of category
    plt.subplot(1,2,1)
    data_to_plot = data[column_name].value_counts().sort_values(ascending=False)
    ax = sns.barplot(x=data_to_plot.index, y=data_to_plot, palette='Set1')

    if percentage_display:
        total_datapoints = len(data[column_name].dropna())
        for p in ax.patches:
            ax.text(p.get_x() + horizontal_adjust, p.get_height() + 0.005 * total_datap
                    '{:1.02f}'.format(p.get_height() * 100 / total_datapoints), fontsize
    plt.xlabel(column_name, labelpad = 10)
    plt.title(f'Distribution of {column_name}', pad = 20)
    plt.xticks(rotation = rotation)
    plt.ylabel('Counts')

    #plotting distribution of category for defaulters
    if plot_defaulter:
        percentage_defaulter_per_category = (data[column_name][data.TARGET == 1].value_
                                             data[column_name].value_counts()).dropna().
        plt.subplot(1,2,2)
        sns.barplot(x = percentage_defaulter_per_category.index,
                    y = percentage_defaulter_per_category, palette='Set2')
        plt.ylabel('Percentage of defaulter per category')
```

```
        plt.xlabel(column_name, labelpad = 10)
        plt.xticks(rotation = rotation)
        plt.title(f'Percentage of defaulters for each category of {column_name}', pad =
    plt.show()
```

In [9]:

```
def plot_categorical_variables_pie(data, column_name, plot_defaulter=True, hole=0):
    '''
    Function to plot categorical variables Pie Plots

    Inputs:
        data: DataFrame
            The DataFrame from which to plot
        column_name: str
            Column's name whose distribution is to be plotted
        plot_defaulter: bool
            Whether to plot the Pie Plot for Defaulters or not
        hole: int, default = 0
            Radius of hole to be cut out from Pie Chart
    '''
    if plot_defaulter:
        cols = 2
        specs = [[{'type' : 'domain'}, {'type' : 'domain'}]]
        titles = [f'Distribution of {column_name} for all Targets',
                  f'Percentage of Defaulters for each category of {column_name}']
    else:
        cols = 1
        specs = [[{'type': 'domain'}]]
        titles = [f'Distribution of {column_name} for all Targets']

    values_categorical = data[column_name].value_counts()
    labels_categorical = values_categorical.index

    fig = make_subplots(rows=1, cols=cols, specs=specs, subplot_titles=titles)

    fig.add_trace(go.Pie(values=values_categorical, labels=labels_categorical, hole=hol
                         textinfo='label+percent', textposition='inside'), row=1, col=1)
    if plot_defaulter:
        percentage_defaulter_per_category = data[column_name][data.TARGET == 1].value_c
                                            * 100 / data[column_name].value_counts()
        percentage_defaulter_per_category.dropna(inplace = True)
        percentage_defaulter_per_category = percentage_defaulter_per_category.round(2)

        fig.add_trace(go.Pie(values = percentage_defaulter_per_category,
                             labels = percentage_defaulter_per_category.index, hole = h
                             textinfo = 'label+value', hoverinfo = 'label+value'),
                      row = 1, col = 2)

    fig.update_layout(title = f'Distribution of {column_name}')
    fig.show()
```

In [10]:

```
def print_unique_categories(data, column_name, show_counts=False):
    '''
    Function to print the basic stats such as unique categories and their counts for
    categorical variables

        Inputs:
```

```
        data: DataFrame
            The DataFrame from which to print statistics
        column_name: str
            Column's name whose stats are to be printed
        show_counts: bool, default = False
            Whether to show counts of each category or not

    '''
    print('-'*100)
    print(f"The unique categories of '{column_name}' are:\n{data[column_name].unique()}
    print('-'*100)

    if show_counts:
        print(f"Counts of each category are:\n{data[column_name].value_counts()}")
        print('-'*100)
```

In [11]:

```
def plot_cdf(data, column_name, log_scale=False, figsize=(12,8)):
    '''
    Function to plot CDF of a continuour variable

    Inputs:
        data: DataFrame
            The DataFrame from which to plot
        column_name: str
            Column's name whose CDF is to be plotted
        log_scale: bool, default = True
            Whether to use log-scale (for widely varying values) or not
        figsize: tuple, default = (12,8)
            The size of figure to be plotted
    '''
    percentile_values = data[[column_name]].dropna().sort_values(by = column_name)
    percentile_values['Percentile'] = [ele / (len(percentile_values) - 1)
                                       for ele in range(len(percentile_values))]

    plt.figure(figsize = figsize)
    if log_scale:
        plt.xscale('log')
        plt.xlabel(column_name + ' - (log-scale)')
    else:
        plt.xlabel(column_name)

    plt.plot(percentile_values[column_name], percentile_values['Percentile'], color = '
    plt.ylabel('Probability')
    plt.title('CDF of {}'.format(column_name))
    plt.show()
```

In [12]:

```
def print_percentiles(data, column_name, percentiles=None):
    '''
    Function to print percentile values for given column

    Inputs:
        data: DataFrame
            The DataFrame from which to print percentiles
        column_name: str
            Column's name whose percentiles are to be printed
```

```
        percentiles: list, default = None
            The list of percentiles to print, if not given, default are printed
    '''

    print('-' * 100)
    if not percentiles:
        percentiles = list(range(0,80,25)) + list(range(90,101,2))
    for i in percentiles:
        print(f'The {i}th percentile value of {column_name} is
              {np.percentile(data[column_name].dropna(), i)}')
    print('-' * 100)
```

In [42]:

```
def plot_continuous_variables(data, column_name, plots = ['distplot', 'CDF', 'box', 'vi
                scale_limits = None, figsize = (20,8), histogram = True, log_scale = Fa

    '''
    Function to plot continuous variables distribution

    Inputs:
        data: DataFrame
            The DataFrame from which to plot.
        column_name: str
            Column's name whose distribution is to be plotted.
        plots: list, default = ['distplot', 'CDF', box', 'violin']
            List of plots to plot for Continuous Variable.
        scale_limits: tuple (left, right), default = None
            To control the limits of values to be plotted in case of outliers.
        figsize: tuple, default = (20,8)
            Size of the figure to be plotted.
        histogram: bool, default = True
            Whether to plot histogram along with distplot or not.
        log_scale: bool, default = False
            Whether to use log-scale for variables with outlying points.
    '''


    data_to_plot = data.copy()
    if scale_limits:
        data_to_plot[column_name] = data[column_name][(data[column_name] > scale_limits
                                                       (data[column_name] < scale_limits

    number_of_subplots = len(plots)
    plt.figure(figsize=figsize)
    sns.set_style('whitegrid')

    for i, ele in enumerate(plots):
        plt.subplot(1, number_of_subplots, i + 1)
        plt.subplots_adjust(wspace = 0.25)

        if ele == 'CDF':
            #making the percentile DataFrame for both positive and negative class label
            percentile_values_0 = data_to_plot[data_to_plot.TARGET == 0][[column_name]]
            percentile_values_0['Percentile'] = [ele / (len(percentile_values_0) - 1) f

            percentile_values_1 = data_to_plot[data_to_plot.TARGET == 1][[column_name]]
            percentile_values_1['Percentile'] = [ele / (len(percentile_values_1) - 1) f

            plt.plot(percentile_values_0[column_name], percentile_values_0['Percentile'
```

```
            plt.plot(percentile_values_1[column_name], percentile_values_1['Percentile'
            plt.xlabel(column_name)
            plt.ylabel('Probability')
            plt.title(f'CDF of {column_name}')
            plt.legend(fontsize = 'medium')
            if log_scale:
                plt.xscale('log')
                plt.xlabel(column_name + ' - (log-scale)')

        if ele == 'distplot':
            sns.distplot(data_to_plot[column_name][data.TARGET == 0].dropna(),
                         label='Non-Defaulters', hist=False, color='red')
            sns.distplot(data_to_plot[column_name][data.TARGET == 1].dropna(),
                         label='Defaulters', hist=False, color='black')
            plt.xlabel(column_name)
            plt.ylabel('Probability Density')
            plt.legend(fontsize='medium')
            plt.title(f'Dist-Plot of {column_name}')
            if log_scale:
                plt.xscale('log')
                plt.xlabel(column_name + ' - (log-scale)')

        if ele == 'violin':
            sns.violinplot(x='TARGET', y=column_name, data=data_to_plot)
            plt.title(f'Violin-Plot of {column_name}')
            if log_scale:
                plt.xscale('log')
                plt.xlabel(column_name + ' - (log-scale)')

        if ele == 'box':
            sns.boxplot(x='TARGET', y=column_name, data=data_to_plot)
            plt.title(f'Box-Plot of {column_name}')
            if log_scale:
                plt.xscale('log')
                plt.xlabel(column_name + ' - (log-scale)')

    plt.show()
```

2. Exploratory Data Analysis (EDA)

For the data analysis, we will follow following steps:

1. For each table, we will first check basic stats like the number of records in tables, number of features, number of NaN values, etc.
2. Next, we will explore some of the features with respect to the target variable for each table. We will be employing the following plot:
   - For `categorical features` , we will mostly use **Bar Plots** and **Pie Charts**.
   - For `continuous/numeric features` , we will use **Box Plots**, **PDFs**, **CDF** and **Violin Plots**.
3. We will drawing observations from each plot and note important insights generated from the plots.

Note 1:

For `Categorical Variables` , we will be plotting the `Bar Plots` and `Pie Plots` . The use of Bar and Pie Plot will be based on the Number of unique categories present in a feature. If a feature has too many categories, displaying them on Pie Plots can be cumbersome, and Bar Plot does a better job of showing each category. Also Bar Plot will be preferred when the proportions of all the categories are more or less the same to identify small differences

We will be following the below mentioned strategy for plotting for Categorical Features in the whole notebook:

- First, we will be plotting the distribution of each category in the whole data, in the first subplot.
- Next, in the second subplot, we will be plotting the Percentage of Defaulters from each category, i.e. with Target = 1.
- For example, say if a feature contains Gender, Male and Female.
  - For first subplot we will plot the number of occurrences of each of Male and Female in our dataset.
  - In the second subplot, we will be plotting that out of the counts of Male present in the dataset, how many or what percentage of Males were found to Default. Similarly we will do this for Female.
  - This is being done because there will be few categories which will be dominant over others, and their Default characteristics would not be identifiable if we look at just the counts.

Note 2:

For analysing the `continuous variables` , we will use four kinds of plots as and when needed, which are `Distplot, CDF, Box-plots and Violin Plots` .

DistPlots:

The distplot will be used when we want to see the PDFs of the continuous variable. This PDF will help us to analyze where most of our data is lying.

CDF:

CDFs can be used as an extention to PDFs to see what percentage of points lie below a certain threshold value. This would give us a good estimate of the distribution of majority of data.

Box-Plots:

Box-plots are helpful when we want to analyze the whole range of values that our continuous variable has. It shows the 25th, 50th and 75th percentile in a single plot. Moreover, it also gives some ideas related to presence of outliers in a given set of values.

Violin-Plots:

Violin-plots tend to combine the features of both Distplots and Box-Plots. Vertically they mimic the box-plot and show the quantiles, range of values, and horizontally they show the PDF of the continuous variable.

Loading all tables

In [14]:
```
load_all_tables()
```

```
Loading all tables...
1 load app_train.csv
2 load 2 app_test.csv
3 load bureau.csv
4 load bureau_balance.csv
5 load credit_card_balance.csv
6 load installments_payments.csv
7 load POS_CASH_balance.csv
8 load previous_application.csv
Done
Time to load files:  0:00:27.188027
```

In [14]:
```
#app_train, app_test, bureau, bureau_balance, cc_balance, install_pay, POS_CASH_balance
```

2.1 application_train.csv and application_test.csv

Description:

- The `application_train.csv` table consists of static data relating to the borrowers with labels. Each row represents one loan application.
- The `application_test.csv` contains the tesing dataset, and is similar to *application_train.csv*, except that the TARGET column has been removed, which has to be predicted with the help of statistical and machine learning predictive models.

2.1.1 Basic Stats

In [15]:
```
print('-' * 100)
print(f'The shape of application_train.csv is : {app_train.shape}')
print('-' * 100)
print(f'Number of duplicated rows in application_train.csv: {app_train.duplicated().sum
print('-' * 100)
# print(app_train.dtypes)
app_train.head()
```

```
----------------------------------------------------------------------------------------------------
The shape of application_train.csv is : (307511, 122)
----------------------------------------------------------------------------------------------------
Number of duplicated rows in application_train.csv: 0
----------------------------------------------------------------------------------------------------
```

Out[15]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY
0	100002	1	Cash loans	M	N	Y
1	100003	0	Cash loans	F	N	N
2	100004	0	Revolving loans	M	Y	Y

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY
3	100006	0	Cash loans	F	N	Y
4	100007	0	Cash loans	M	N	Y

In [16]:
```
print('-' * 100)
print(f'The shape of application_test.csv is : {app_test.shape}')
print('-' * 100)
print(f'Number of duplicated rows in application_test.csv: {app_test.duplicated().sum()}')
print('-' * 100)
app_test.head()
```

```
----------------------------------------------------------------------------------------------------
The shape of application_test.csv is : (48744, 121)
----------------------------------------------------------------------------------------------------
Number of duplicated rows in application_test.csv: 0
----------------------------------------------------------------------------------------------------
```

Out[16]:

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHI
0	100001	Cash loans	F	N	Y	
1	100005	Cash loans	M	N	Y	
2	100013	Cash loans	M	Y	Y	
3	100028	Cash loans	F	N	Y	
4	100038	Cash loans	M	Y	N	

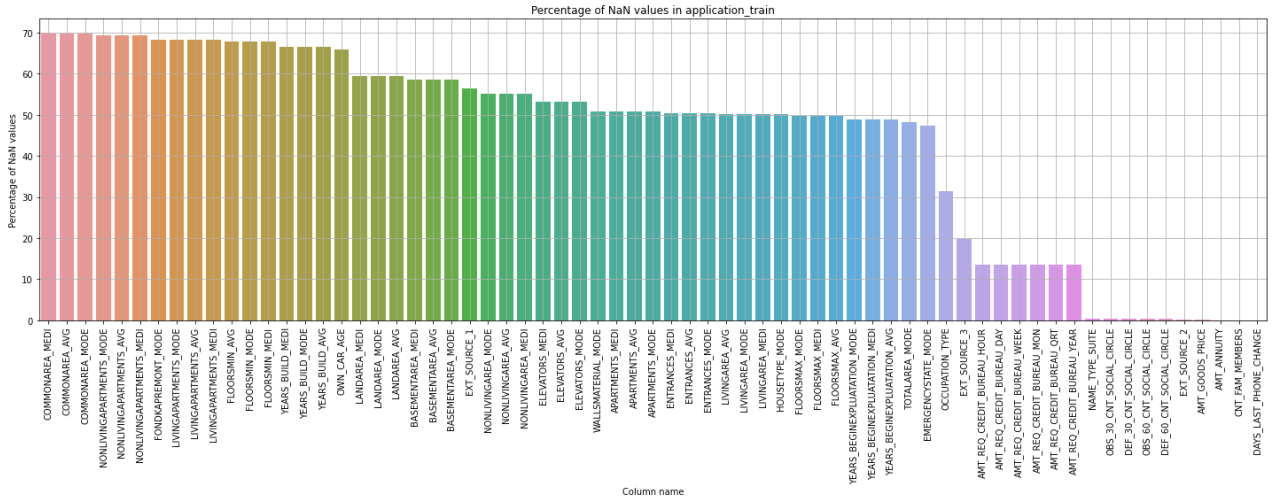
Observations and conclusions:

1. `application_train.csv`
   - The *application_train.csv* file has 307511 records and 122 features. These features contain the personal statistics belonging to a particular customer such as Age, Income, Type of Loan, Apartment Stats, etc.
   - There are 307511 unique SK_ID_CURR which represent unique loan applications.
   - The TARGET column represents the load-default status, 0 stands for Non-Defaulter, and 1 for Defaulter.
2. `application_test.csv`
   - The *application_test.csv* file has 48733 records and 121 features.These features are exactly those which are in application_train.csv, except the TARGET column is removed.

2.1.2 NaN columns and percentages

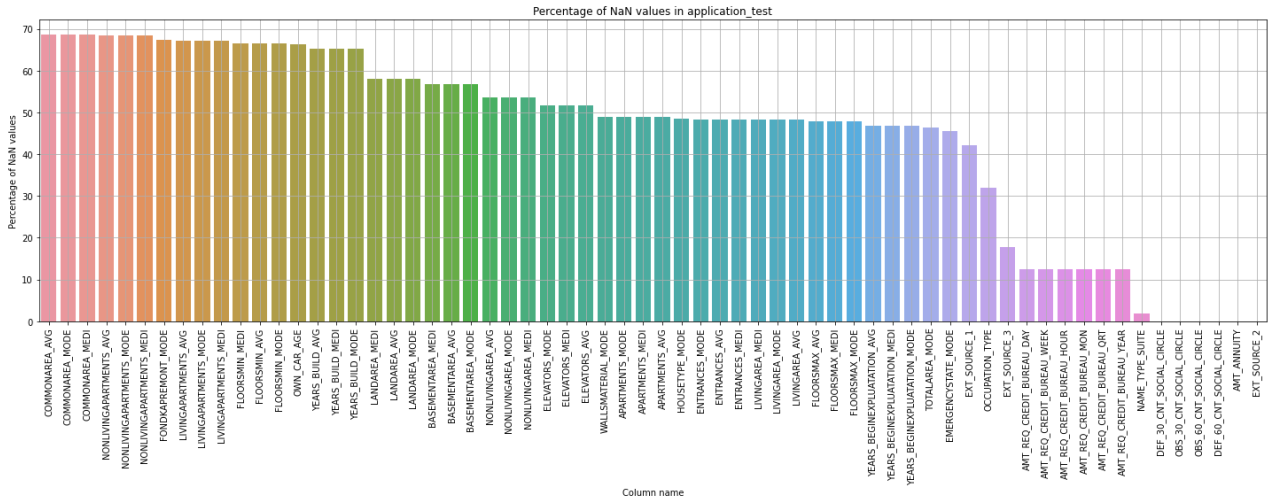
In [21]:
```
plot_nan_percent(nan_df_create(app_train), 'application_train', grid=True)
```

```
Number of columns having NaN values: 67 columns
```



In [22]:
```
plot_nan_percent(nan_df_create(app_test), 'application_test', grid=True)
```

```
Number of columns having NaN values: 64 columns
```



Observations and conclusions:

1. `application_train.csv`

   - It can be seen that there are 67 columns out of 122 features which contain NaN value. Since the number of columns which contain NaN values is so big, we cannot just emilinate those columns.
   - We see that some columns like relating to "COMMONAREA", "NONLIVINGAPARTMENT", etc. have close to 70% missing values.We have to come up with techniques to handle these many missing values.
   - Another thing to note here is that most of the columns which have more than 50% missing values are related to Apartment Statistics. It's very likely that these values were not recorded during data entry, and could be optional.

2. `application_test.csv`

    - There are very similar number of columns with NaN values (64) as were with the application_train.
    - The percentages of NaN values are also quite similar to the ones present in training dataset. This means that the training and test sets are pretty much of similar distribution.

2.1.3 Distribution of target variable

In [23]:
```
target_distribution = app_train.TARGET.value_counts()
display(target_distribution)
labes = ['Non_Defaulter', 'Defaulter']

fig = go.Figure(data = [go.Pie(values=target_distribution, labels=labes,
                               textinfo='label+percent+value', pull=[0, 0.04])],
                layout=go.Layout(title='Distribution of target variable'))
fig.show(render='svg')
```

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

Observations and conclusions:

- From the distribution of target variable, one thing that we can quickly notice is the **data imbalance**. There are only 8.07% of the total loans that had actually been defaulted.
- For imbalance data, duting building the model, we cannot feed the data as is to some algorithms, which are imbalance sensitive.
- Similar is the case with the preformance metrics. For such a dataset, `accuracy` is usually not the right metric as the accuracy would generally be biased to majority class. We can use other metrics such as `ROC-AUC Score`, `Log-Loss`, `F1-Score`, `Confusion Matrix` for better evaluation of model.
- One more important thing to note here is that there are very few people who actually default, and they tend to show some sort of different behaviour. Thus in such cases of Fraud, Default and Anamoly Detection, we need to focus on outliers too, and we cannot remove them, as they could be the differentiating factor between Defaulter and Non-Defaulter.

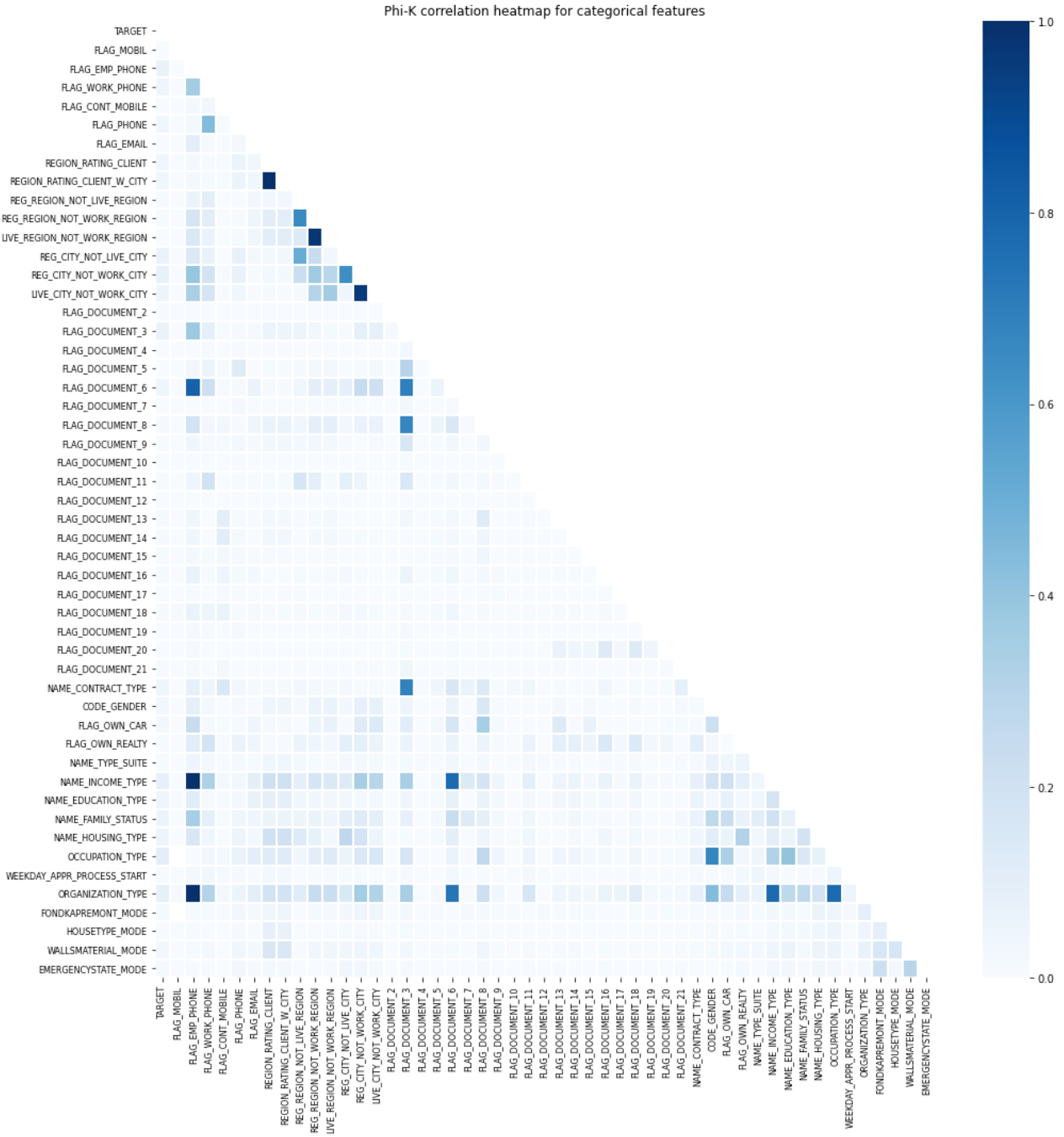
2.1.4 Phi-K matrix

We will plot a heatmap of the values of Phi-K correlation coefficient between each of the feature with the other.

The Phi-K coefficient is similar to correlation coefficient except that it can be used with a pair of categorical features to check if one feature shows some sort of association with the other categorical feature. It's max value can be 1 which would show a maximum association between two categorical variable.

In [26]:
```
categorical_columns = ['TARGET','FLAG_MOBIL', 'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE',
                       'FLAG_CONT_MOBILE','FLAG_PHONE', 'FLAG_EMAIL','REGION_RATING_CLI
                       'REGION_RATING_CLIENT_W_CITY','REG_REGION_NOT_LIVE_REGION',
                       'REG_REGION_NOT_WORK_REGION', 'LIVE_REGION_NOT_WORK_REGION',
                       'REG_CITY_NOT_LIVE_CITY', 'REG_CITY_NOT_WORK_CITY',
                       'LIVE_CITY_NOT_WORK_CITY'] + \
                     ['FLAG_DOCUMENT_' + str(i) for i in range(2,22)] + \
                     app_train.dtypes[app_train.dtypes == 'object'].index.tolist()
plot_phik_matrix(app_train, categorical_columns, figsize = (15,15), fontsize = 8)
```

```
--------------------------------------------------------------------------------
------------
```



Categorise with highest values of Phi-K correlation value with target variable are:

	Column Name	Phik-Correlation
43	OCCUPATION_TYPE	0.102846
45	ORGANIZATION_TYPE	0.089164
39	NAME_INCOME_TYPE	0.084831
12	REG_CITY_NOT_WORK_CITY	0.079946
1	FLAG_EMP_PHONE	0.072087
11	REG_CITY_NOT_LIVE_CITY	0.069588
15	FLAG_DOCUMENT_3	0.069525

	Column Name	Phik-Correlation
41	NAME_FAMILY_STATUS	0.056043
42	NAME_HOUSING_TYPE	0.051107
13	LIVE_CITY_NOT_WORK_CITY	0.050956

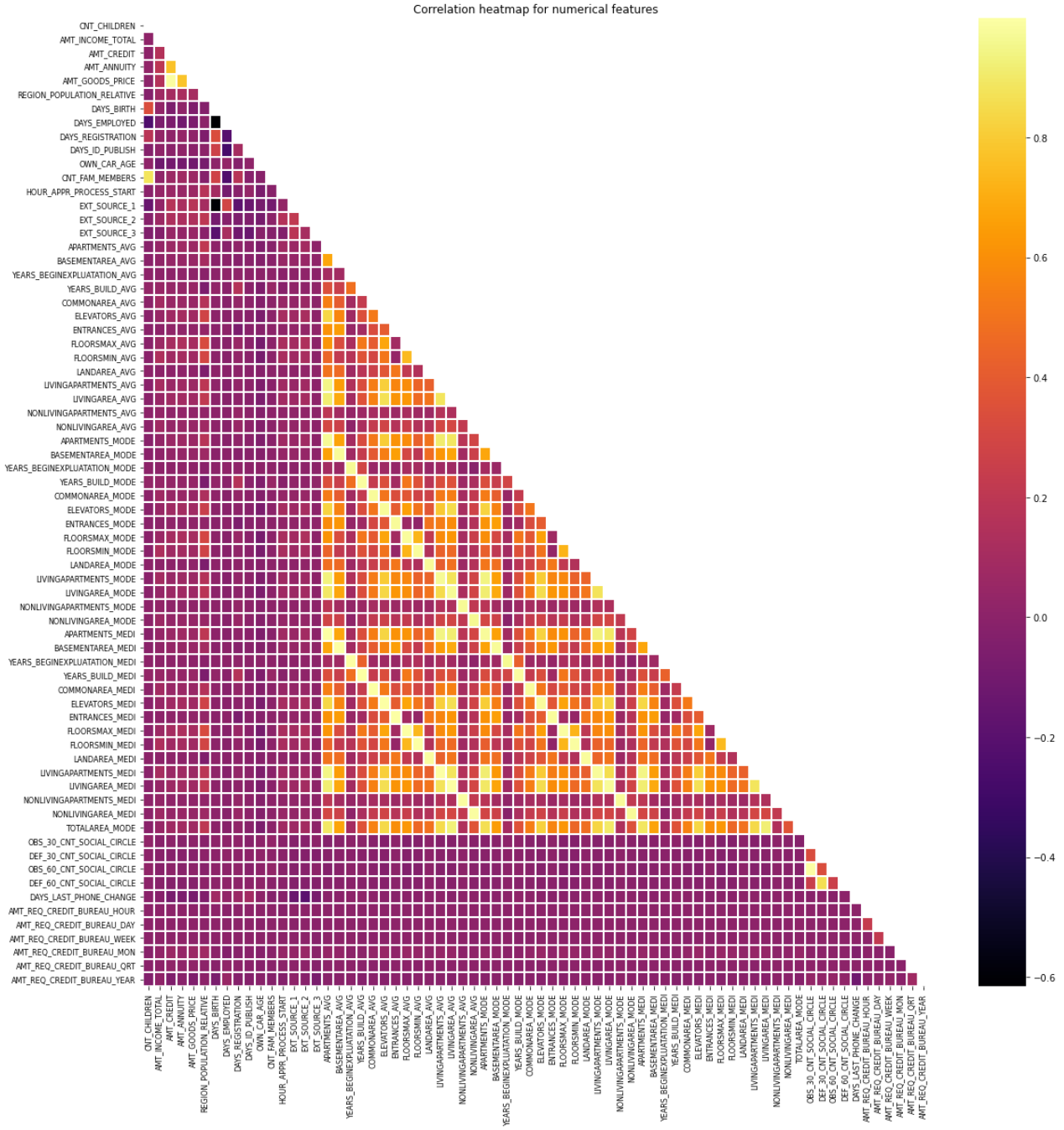
Observations and conclusions:

1. From the above heatmap of Phi-K Correlation, we see that most of the categorical features are not correlated to each other, however some of them show strong correlation.
2. Some of the highly correlated Category pairs are:
   - REGION_RATING_CLIENT_W_CITY and REGION_RATING_CLIENT - This is understandable as they would more or less tell a similar story.
   - LIVE_REGION_NOT_WORK_REGION and REG_REGION_NOT_WORK_REGION
   - NAME_INCOME_TYPE, ORGANIZATION_TYPE and FLAG_EMP_PHONE
3. We can also see some correlation between the Organization type and the income type of a client. Similarly we see a correlation between the Occupation Type and the Organization Type too.
4. We find that the category OCCUPATION_TYPE, ORGANIZATION_TYPE, NAME_INCOME_TYPE, REG_CITY_NOT_WORK_CITY are some of the highest correlated categories with the TARGET variable. These maybe important in the classification task, and would need further EDA

2.1.5 Correlation matrix of numerical features

We will plot a heatmap of the correlation of each numerical feature with respect to other features. We have excluded the column SK_ID_CURR, as it does not have any relevance. This heatmap will help us identify the highly correlated numeric features and will also help us to identify features which are highly correlated with Target Variable.

In [27]:
```
columns_to_drop = ['SK_ID_CURR'] + list(set(categorical_columns) - set(['TARGET']))
corr_mat = correlation_matrix(app_train, columns_to_drop, figsize=(17,17),
                              fontsize=8, cmap='inferno')
corr_mat.plot_correlation_matrix()
```



In [28]:

```
#Seeing the top columns with hightest phik-correlation with the TARGET variable
#in application_train.csv
top_corr_target_df = corr_mat.target_top_corr()
print('-' * 100)
print("Columns with highest values of Phik-correlation with Target Variable are:")
display(top_corr_target_df)
print('-' * 100)
```

```
interval columns not set, guessing: ['TARGET', 'CNT_CHILDREN']
interval columns not set, guessing: ['TARGET', 'AMT_INCOME_TOTAL']
interval columns not set, guessing: ['TARGET', 'AMT_CREDIT']
interval columns not set, guessing: ['TARGET', 'AMT_ANNUITY']
interval columns not set, guessing: ['TARGET', 'AMT_GOODS_PRICE']
interval columns not set, guessing: ['TARGET', 'REGION_POPULATION_RELATIVE']
```

```
interval columns not set, guessing: ['TARGET', 'DAYS_BIRTH']
interval columns not set, guessing: ['TARGET', 'DAYS_EMPLOYED']
interval columns not set, guessing: ['TARGET', 'DAYS_REGISTRATION']
interval columns not set, guessing: ['TARGET', 'DAYS_ID_PUBLISH']
interval columns not set, guessing: ['TARGET', 'OWN_CAR_AGE']
interval columns not set, guessing: ['TARGET', 'CNT_FAM_MEMBERS']
interval columns not set, guessing: ['TARGET', 'HOUR_APPR_PROCESS_START']
interval columns not set, guessing: ['TARGET', 'EXT_SOURCE_1']
interval columns not set, guessing: ['TARGET', 'EXT_SOURCE_2']
interval columns not set, guessing: ['TARGET', 'EXT_SOURCE_3']
interval columns not set, guessing: ['TARGET', 'APARTMENTS_AVG']
interval columns not set, guessing: ['TARGET', 'BASEMENTAREA_AVG']
interval columns not set, guessing: ['TARGET', 'YEARS_BEGINEXPLUATATION_AVG']
interval columns not set, guessing: ['TARGET', 'YEARS_BUILD_AVG']
interval columns not set, guessing: ['TARGET', 'COMMONAREA_AVG']
interval columns not set, guessing: ['TARGET', 'ELEVATORS_AVG']
interval columns not set, guessing: ['TARGET', 'ENTRANCES_AVG']
interval columns not set, guessing: ['TARGET', 'FLOORSMAX_AVG']
interval columns not set, guessing: ['TARGET', 'FLOORSMIN_AVG']
interval columns not set, guessing: ['TARGET', 'LANDAREA_AVG']
interval columns not set, guessing: ['TARGET', 'LIVINGAPARTMENTS_AVG']
interval columns not set, guessing: ['TARGET', 'LIVINGAREA_AVG']
interval columns not set, guessing: ['TARGET', 'NONLIVINGAPARTMENTS_AVG']
interval columns not set, guessing: ['TARGET', 'NONLIVINGAREA_AVG']
interval columns not set, guessing: ['TARGET', 'APARTMENTS_MODE']
interval columns not set, guessing: ['TARGET', 'BASEMENTAREA_MODE']
interval columns not set, guessing: ['TARGET', 'YEARS_BEGINEXPLUATATION_MODE']
interval columns not set, guessing: ['TARGET', 'YEARS_BUILD_MODE']
interval columns not set, guessing: ['TARGET', 'COMMONAREA_MODE']
interval columns not set, guessing: ['TARGET', 'ELEVATORS_MODE']
interval columns not set, guessing: ['TARGET', 'ENTRANCES_MODE']
interval columns not set, guessing: ['TARGET', 'FLOORSMAX_MODE']
interval columns not set, guessing: ['TARGET', 'FLOORSMIN_MODE']
interval columns not set, guessing: ['TARGET', 'LANDAREA_MODE']
interval columns not set, guessing: ['TARGET', 'LIVINGAPARTMENTS_MODE']
interval columns not set, guessing: ['TARGET', 'LIVINGAREA_MODE']
interval columns not set, guessing: ['TARGET', 'NONLIVINGAPARTMENTS_MODE']
interval columns not set, guessing: ['TARGET', 'NONLIVINGAREA_MODE']
interval columns not set, guessing: ['TARGET', 'APARTMENTS_MEDI']
interval columns not set, guessing: ['TARGET', 'BASEMENTAREA_MEDI']
interval columns not set, guessing: ['TARGET', 'YEARS_BEGINEXPLUATATION_MEDI']
interval columns not set, guessing: ['TARGET', 'YEARS_BUILD_MEDI']
interval columns not set, guessing: ['TARGET', 'COMMONAREA_MEDI']
interval columns not set, guessing: ['TARGET', 'ELEVATORS_MEDI']
interval columns not set, guessing: ['TARGET', 'ENTRANCES_MEDI']
interval columns not set, guessing: ['TARGET', 'FLOORSMAX_MEDI']
interval columns not set, guessing: ['TARGET', 'FLOORSMIN_MEDI']
interval columns not set, guessing: ['TARGET', 'LANDAREA_MEDI']
interval columns not set, guessing: ['TARGET', 'LIVINGAPARTMENTS_MEDI']
interval columns not set, guessing: ['TARGET', 'LIVINGAREA_MEDI']
interval columns not set, guessing: ['TARGET', 'NONLIVINGAPARTMENTS_MEDI']
interval columns not set, guessing: ['TARGET', 'NONLIVINGAREA_MEDI']
interval columns not set, guessing: ['TARGET', 'TOTALAREA_MODE']
interval columns not set, guessing: ['TARGET', 'OBS_30_CNT_SOCIAL_CIRCLE']
interval columns not set, guessing: ['TARGET', 'DEF_30_CNT_SOCIAL_CIRCLE']
interval columns not set, guessing: ['TARGET', 'OBS_60_CNT_SOCIAL_CIRCLE']
interval columns not set, guessing: ['TARGET', 'DEF_60_CNT_SOCIAL_CIRCLE']
interval columns not set, guessing: ['TARGET', 'DAYS_LAST_PHONE_CHANGE']
interval columns not set, guessing: ['TARGET', 'AMT_REQ_CREDIT_BUREAU_HOUR']
interval columns not set, guessing: ['TARGET', 'AMT_REQ_CREDIT_BUREAU_DAY']
```

```
interval columns not set, guessing: ['TARGET', 'AMT_REQ_CREDIT_BUREAU_WEEK']
interval columns not set, guessing: ['TARGET', 'AMT_REQ_CREDIT_BUREAU_MON']
interval columns not set, guessing: ['TARGET', 'AMT_REQ_CREDIT_BUREAU_QRT']
interval columns not set, guessing: ['TARGET', 'AMT_REQ_CREDIT_BUREAU_YEAR']
--------------------------------------------------------------------------------
------------
Columns with highest values of Phik-correlation with Target Variable are:
```

	Column Name	Phik-Correlation
15	EXT_SOURCE_3	0.247680
13	EXT_SOURCE_1	0.217846
14	EXT_SOURCE_2	0.213965
6	DAYS_BIRTH	0.102378
63	DAYS_LAST_PHONE_CHANGE	0.073218
7	DAYS_EMPLOYED	0.072095
9	DAYS_ID_PUBLISH	0.067766
4	AMT_GOODS_PRICE	0.059094
23	FLOORSMAX_AVG	0.058826
51	FLOORSMAX_MEDI	0.058595

```
--------------------------------------------------------------------------------
------------
```

Observations and conclusions:

1. From the heatmap, we can see that most of the features are almost not correlated to others.
2. However, at the middle of the heatmap, these shades depict a high value of correlation between the features. These features are related to the state of the apartments.
3. If we look at the features of application_train, we can clearly see that the statistics of apartments are given in terms of Mean, Median and Mode, so it can be expected for the mean, median and mode to be correlated with each other. One more thing to note is that the features among particular category, for example Mean are also correlated with other mean features, such as Number of Elevators, Living Area, Non-Living Area, Basement Area, etc.
4. We also see some high correlation between AMT_GOODS_PRICE and AMT_CREDIT, between DAYS_EMPLOYED and DAYS_BIRTH.
5. We would not want highly correlated features as they increase the time complexity of the model without adding much value to it. Hence, we would be removing the inter-correlated features.
6. Among all the features, we see some high correlation for EXT_SOURCE features with respect to Target Variable. These might be important for our classification task.

2.1.6 Plotting distribution of categorical variables

2.1.6.1 NAME_CONTRACT_TYPE

`NAME_CONTRACT_TYPE` is about the type of the loan for the given applicant. According to the ducumentation provided by Home Credit, there are two types of loans, i.e. *Revolving Loans* and *Cash Loans*.

In [29]:
```
#let's first check the unique categories of 'NAME_CONTRCT_TYPE'
print_unique_categories(app_train, 'NAME_CONTRACT_TYPE')

#plot the pie plot for the column
plot_categorical_variables_pie(app_train, 'NAME_CONTRACT_TYPE', hole=0.5)
print('-' * 100)
```

```
----------------------------------------------------------------------------------------------------
The unique categories of 'NAME_CONTRACT_TYPE' are:
['Cash loans' 'Revolving loans']
----------------------------------------------------------------------------------------------------
```

```
----------------------------------------------------------------------------------------------------
```

Observations and conclusions: From the above plot, we can draw following observations and conclusions:

1. From the first subplot, i.e. the overall distribution:

- It can be seen that most of the loans that the customers take are Cash Loans.
- Only 9.52% of the people have taken Revolving Loans.

2. From the second subplot, i.e. Percentage of Defaulters:
   - We see is that there are more percentage of people who have defaulted with Cash Loans (8.35%) as compared to those who defaulted with Revolving Loans (5.48%).

2.1.6.2 CODE_GENDER

`CODE_GENDER` contains information about the gender of the applicant. Here **M** stands for male, **F** stands for female.

In [32]:
```
#let's first check the unique categories of 'CODE_GENDER'
print_unique_categories(app_train, 'CODE_GENDER', show_counts=True)

#plot the pie plot for the column
plot_categorical_variables_pie(app_train, 'CODE_GENDER', hole=0.5)
print('-' * 100)
```

```
----------------------------------------------------------------------------------------------------
The unique categories of 'CODE_GENDER' are:
['M' 'F' 'XNA']
----------------------------------------------------------------------------------------------------
Counts of each category are:
F      202448
M      105059
XNA         4
Name: CODE_GENDER, dtype: int64
----------------------------------------------------------------------------------------------------
```

```
------------------------------------------------------------------------------------------
------------
```

Observations and conclusions:

1. There are 4 rows in the application_train.csv which have 'XNA' gender, which don't make sence, can be counted as NaN or elimite these 4 records.
2. From the subplot 1 we see that for the given dataset, there are more number of Female applicants (65.8%) than Male applicants (34.2%).
3. However, contrary to the number of Female applicants, from the second plot we note that it has been seen that Male applicants tend to default more (10.14%) as compared to Female applicants (7%).

Thus, it can be said that Male have more tendency to default than Female as per the given dataset.

2.1.6.3 FLAG_EMP_PHONE

`FLAG_EMP_PHONE` tells us whether the client provided his work phone number or not. Here **1** stands for **Yes**, **0** stands for **No**.

In [33]:
```
#let's first check the unique categories of 'FLAG_EMP_PHONE'
print_unique_categories(app_train, 'FLAG_EMP_PHONE', show_counts=True)

#plot the pie plot for the column
plot_categorical_variables_pie(app_train, 'FLAG_EMP_PHONE', hole=0.5)
print('-' * 100)
```

```
------------------------------------------------------------------------------------------
------------
The unique categories of 'FLAG_EMP_PHONE' are:
[1 0]
------------------------------------------------------------------------------------------
------------
Counts of each category are:
1    252125
0     55386
Name: FLAG_EMP_PHONE, dtype: int64
------------------------------------------------------------------------------------------
------------
```

```
------------------------------------------------------------------------------------------
------------
```

Observations and conclusions:

1. From the first subplot we see that most of the applicants provided their Work Phone Number (82%) and only 18% didn't provide their Work Phone Number.
2. The Default tendency for those who do provide Work Phone Number is more than those who do not provide Work Phone Number.This characteristic could be attributed to the fact that the Defaulters might be providing their Work Phone Numbers so that they don't get disturbed on their personal phone.

2.1.6.4 REGION_RATING_CLIENT_W_CITY ¶

REGION_RATING_CLIENT_W_CITY is the rating provided by the Home Credit to each client's region based on the surveys that they might have done. This rating also takes into account the City in which the client lives.

Taking City into account is important because even if some regions have a good rating in a particular City, but that City doesn't have high rating, then applicant would be given a medium rating and not a high rating. It contains values in the range from 1 to 3.

In [34]:
```
#Let's first check the unique categories of 'REGION_RATING_CLIENT_W_CITY'
print_unique_categories(app_train, 'REGION_RATING_CLIENT_W_CITY', show_counts=True)

#plot the pie plot for the column
plot_categorical_variables_pie(app_train, 'REGION_RATING_CLIENT_W_CITY', hole=0.5)
print('-' * 100)
```

```
------------------------------------------------------------------------------------------
------------
The unique categories of 'REGION_RATING_CLIENT_W_CITY' are:
```

```
[2 1 3]
------------------------------------------------------------------------------------------
------------
Counts of each category are:
2    229484
3     43860
1     34167
Name: REGION_RATING_CLIENT_W_CITY, dtype: int64
------------------------------------------------------------------------------------------
------------
```

```
------------------------------------------------------------------------------------------
------------
```

Observations and conclusions:

1. From the first subplot, we see that most of the clients (74.6%) have a region rating of 2. This is the middle value which is for most of the applicants.Very few applicants have a region rating of 1 (only 11.1%) and some have a rating of 3 (14.3%).
2. Among the Defaulters, it is seen that most of the defaulters have a region rating of 3 (11.4%) which is comparably higher to the other two ratings, i.e. clients with rating of 1 have a Defaulting percentage of just 4.84% and with rating 2 have a percentage of 7.92%.

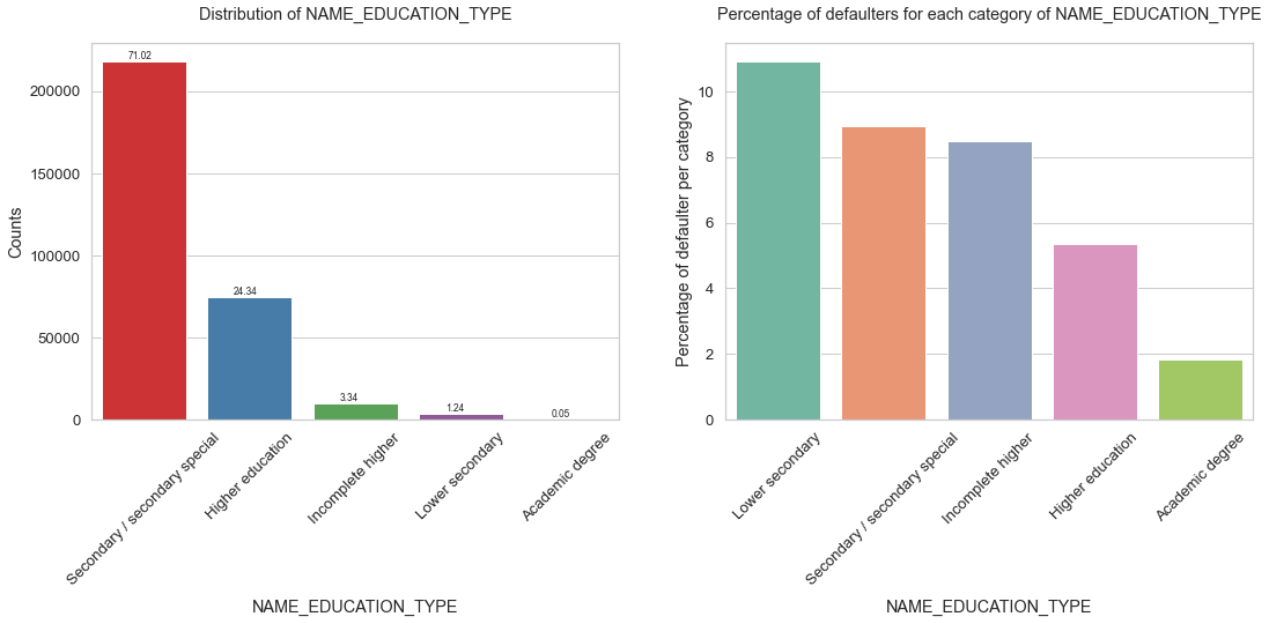
This shows that the rating 3 could be an important attribute for making a decision on Defaulting Characteristics.

2.1.6.5 NAME_EDUCAtION_TYPE

NAME_EDUCAtION_TYPE descibes/enlists the Highest Education that the client had achieved.

In [35]:
```
#Let us first see the unique categories of 'NAME_EDUCATION_TYPE'
print_unique_categories(app_train, 'NAME_EDUCATION_TYPE', show_counts = True)

#plotting the Bar Plot for the Column
plot_categorical_variables_bar(app_train, column_name = 'NAME_EDUCATION_TYPE', rotation
                               horizontal_adjust = 0.25)
print('-'*100)
```

```
----------------------------------------------------------------------------------------------------
------------
The unique categories of 'NAME_EDUCATION_TYPE' are:
['Secondary / secondary special' 'Higher education' 'Incomplete higher'
 'Lower secondary' 'Academic degree']
----------------------------------------------------------------------------------------------------
------------
Counts of each category are:
Secondary / secondary special    218391
Higher education                  74863
Incomplete higher                 10277
Lower secondary                    3816
Academic degree                     164
Name: NAME_EDUCATION_TYPE, dtype: int64
----------------------------------------------------------------------------------------------------
------------
Total number of unique categories of NAME_EDUCATION_TYPE = 5
```




```
----------------------------------------------------------------------------------------------------
------------
```

Observations and conclusions:

1. About 71% of people have had their education only till Secondary/Secondary Special, along with 24.34% clients having done Higher Education. This suggests that most of the clients/borrowers don't have a high education level.

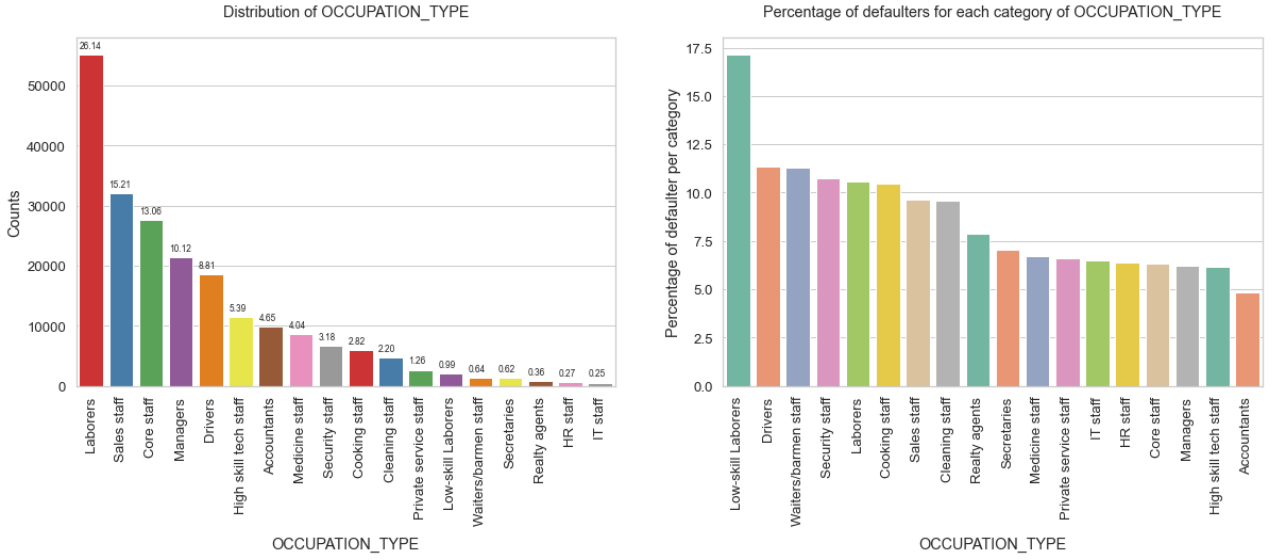
2. From the second plot, we see that the people who have had their studies only Lower Secondary have the highest Defaulting Characterists, with Secondary and Incomplete higher having similar defaulting tendencies.
3. The group of people with Higher Education have comparably lower defaulting tendency, which is logical too. Also, people with Academic Degree show the least Defaulting Rate. However, the Academic Degree group are very few in numbers, so it might not be very useful.

2.1.6.6 OCCUPATION_TYPE

`OCCUPATION_TYPE` tells about the type of Occupation that the client has. This can be a very important feature which could describe the Defaulting Characteristics of a client. Let us see the plots for them.

In [36]:
```
#Let us first see the unique categories of 'OCCUPATION_TYPE'
print_unique_categories(app_train, 'OCCUPATION_TYPE')

#plotting the Bar Plot for the Column
plot_categorical_variables_bar(app_train, column_name = 'OCCUPATION_TYPE', figsize = (2
print('-'*100)
```

```
----------------------------------------------------------------------------------------------------
------------
The unique categories of 'OCCUPATION_TYPE' are:
['Laborers' 'Core staff' 'Accountants' 'Managers' nan 'Drivers'
 'Sales staff' 'Cleaning staff' 'Cooking staff' 'Private service staff'
 'Medicine staff' 'Security staff' 'High skill tech staff'
 'Waiters/barmen staff' 'Low-skill Laborers' 'Realty agents' 'Secretaries'
 'IT staff' 'HR staff']
----------------------------------------------------------------------------------------------------
------------
Total number of unique categories of OCCUPATION_TYPE = 19
```




```
----------------------------------------------------------------------------------------------------
------------
```

Observations and conclusions:

1. Among the applicants, the most common type of Occupation is Laborers contributing to close to 26% applications. The next most frequent occupation is Sales Staff, followed by Core Staff

and Managers.

2. The Defaulting Rate for Low-Skill Laborers is the highest among all the occupation types (~17.5%). This is followed by Drivers, Waiters, Security Staff, Laborers, Cooking Staff, etc. All the jobs are low-level jobs. This shows that low-level Jobs people tend to have higher default rate.
3. The lowest Defaulting Rate are among Accountants, Core Staff, Managers, High skill tech staff, HR staff, etc. which are from medium to high level jobs.

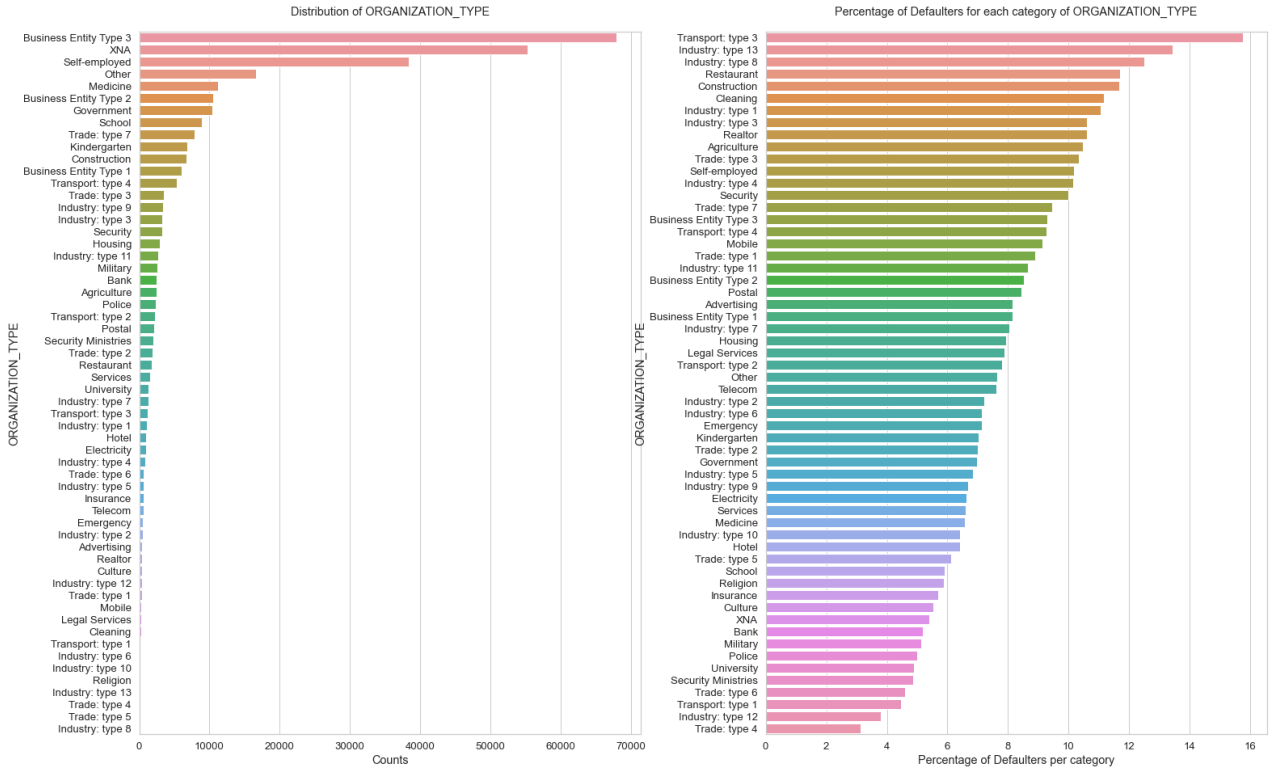
Thus it can be concluded that Low-level job workers tend to have a higher defaulting tendency compared to medium-high level jobs.

2.1.6.7 ORGANIZATION_TYPE

`ORGANIZATION_TYPE` that the client belongs to could also be an important feature for predicting the Default Risk of that client. Let us visualize this feature in more detail.

In [37]:
```
print(f"Total Number of categories of ORGANIZATION_TYPE =
      {len(app_train.ORGANIZATION_TYPE.unique())}")

plt.figure(figsize = (25,16))
sns.set(style = 'whitegrid', font_scale = 1.2)
plt.subplots_adjust(wspace=0.25)

plt.subplot(1,2,1)
count_organization = app_train.ORGANIZATION_TYPE.value_counts().sort_values(ascending =
sns.barplot(x = count_organization, y = count_organization.index)
plt.title('Distribution of ORGANIZATION_TYPE', pad = 20)
plt.xlabel('Counts')
plt.ylabel('ORGANIZATION_TYPE')

plt.subplot(1,2,2)
percentage_default_per_organization = app_train[app_train.TARGET == 1].ORGANIZATION_TYP
percentage_default_per_organization = percentage_default_per_organization.dropna().sort
sns.barplot(x = percentage_default_per_organization, y = percentage_default_per_organiz
plt.title('Percentage of Defaulters for each category of ORGANIZATION_TYPE', pad = 20)
plt.xlabel('Percentage of Defaulters per category')
plt.ylabel('ORGANIZATION_TYPE')

plt.show()
```

```
Total Number of categories of ORGANIZATION_TYPE = 58
```



Observations and conclusions:

There are 58 organization types which the client belongs to. The plots above give the following observations:

1. From the first plot we see that most of the applicants work in Organizations of Type 'Business Entity Type3', 'XNA' or 'Self Employed'. The Organization Type 'XNA' could probably denote unclassified Organization Type.
2. From the second plot, we notice that the applicants belonging to 'Transport: type 3' have the highest defaulting tendency as compared to the rest. They are followed by organizations of types: 'Industry: type 13', 'Industry: type 8', 'Restaurant', 'Construction', etc.
3. The organizations which show lowest default rates are 'Trade: type 4', 'Industry: type 12', etc.

These type numbers also would say something more about the Organization, however, we don't have any information related to that, so we will stick with the naming provided to us only.

2.1.6.8 `REG_CITY_NOT_LIVE_CITY`, `REG_CITY_NOT_WORK_CITY`, `LIVE_CITY_NOT_WORK_CITY` ¶

`REG_CITY_NOT_LIVE_CITY`, `REG_CITY_NOT_WORK_CITY` : These columns include flags whether if the the client's permanent address matches with his Contact Address or Work Address or not at region level.

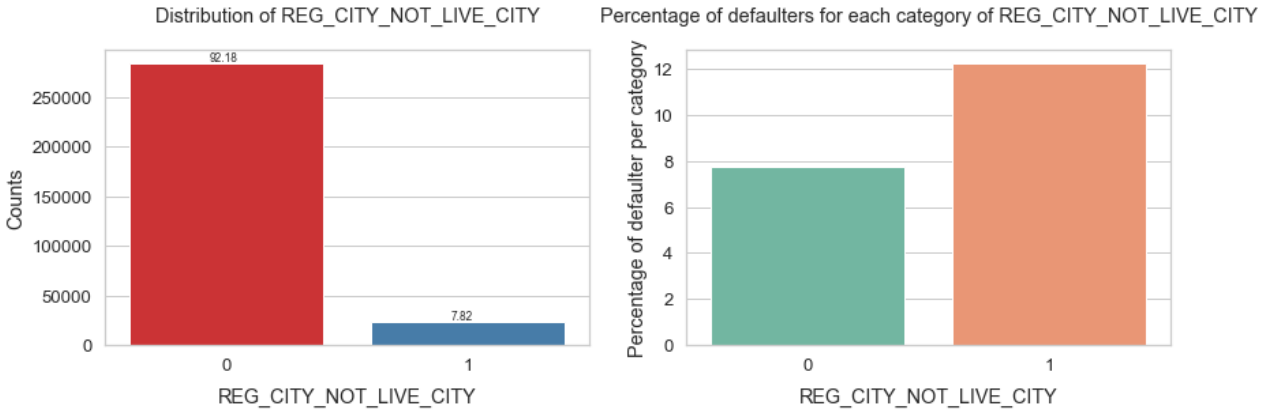
`LIVE_CITY_NOT_WORK_CITY` ¶: This column indicates whether if the client's permanent address matches with his Contact Address at city level or not.

Here **1** stands for **different** address, **0** stands for **same** address.

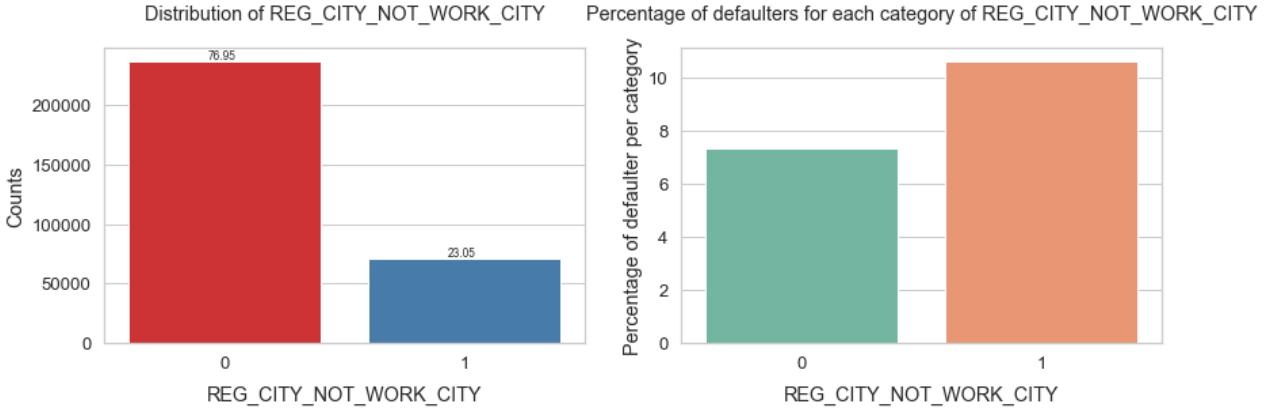
In [38]:

```
print('-'*100)
plot_categorical_variables_bar(app_train, column_name = 'REG_CITY_NOT_LIVE_CITY', figsi
print('-'*100)
plot_categorical_variables_bar(app_train, column_name = 'REG_CITY_NOT_WORK_CITY', figsi
print('-'*100)
plot_categorical_variables_bar(app_train, column_name = 'LIVE_CITY_NOT_WORK_CITY', figs
print('-'*100)
```

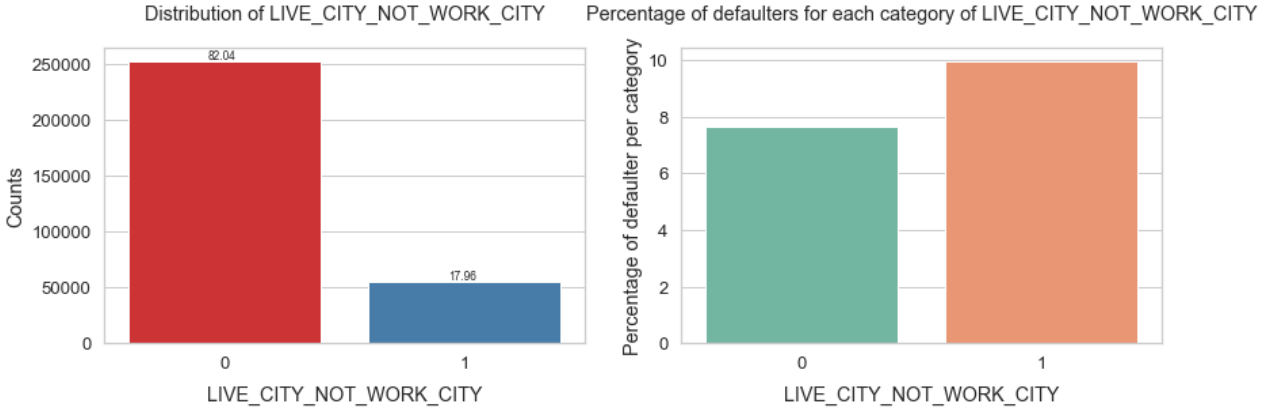
```
----------------------------------------------------------------------------------------------------
Total number of unique categories of REG_CITY_NOT_LIVE_CITY = 2
```




```
----------------------------------------------------------------------------------------------------
Total number of unique categories of REG_CITY_NOT_WORK_CITY = 2
```




```
----------------------------------------------------------------------------------------------------
Total number of unique categories of LIVE_CITY_NOT_WORK_CITY = 2
```




```
----------------------------------------------------------------------------------------------------
```

Observations and conclusions:

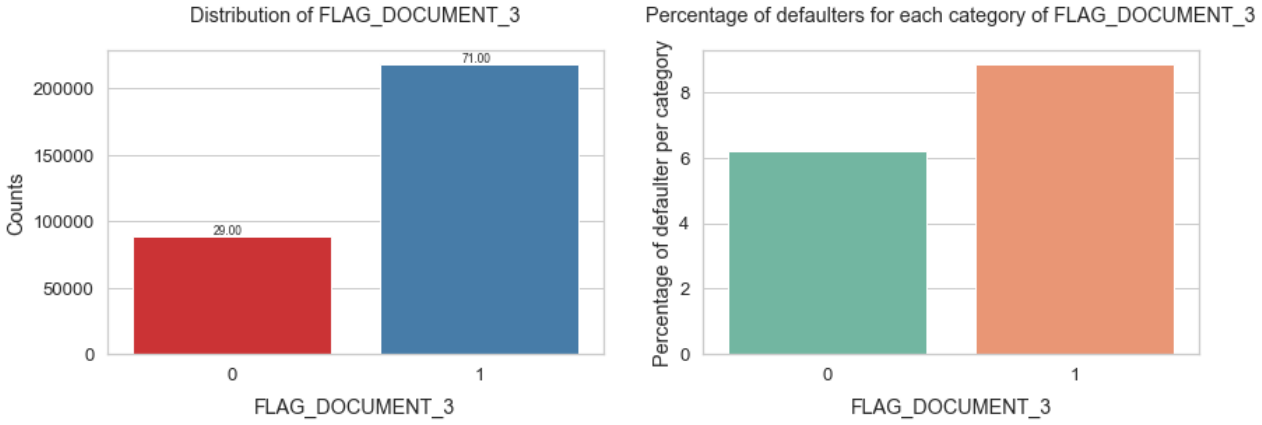
1. Of all the applicants there are only a minority of applicants whose addresses do not match.
    - Firstly, there are only 7.52% people who have different permanent address from their contact address at region level.
    - Secondly, there are around 23.05% people who have different permanent address from their work address at region level. This higher number is explainable, because it is possible that they work in different region as compared to their permanent address.
    - Lastly, there are around 17.96% people who have different permanent address from their contact address at city level.
2.
    - If we look at the defaulting characteristics, we find that there is maximum defaulting tendency of those people who have their permanent and contact addresses different at region level, which is followed by different permanent and work address and lastly different permanent and contact address at city level.
    - For all the cases it is seen that the Defaulting tendency of those people who have different addresses is higher than those who have same address. This means that somewhere, this difference in address may suggest Defaulting Risk.

2.1.6.9 FLAG_DOCUMENT_3

FLAG_DOCUMENT_3 contains the flag about a document that was to be submitted by the applicant. It's value is 0 if the client didn't provid the document and 1 if provided.

In [40]:
```
plot_categorical_variables_bar(app_train, column_name = 'FLAG_DOCUMENT_3',
                               figsize = (14, 4), horizontal_adjust = 0.33)
```

```
Total number of unique categories of FLAG_DOCUMENT_3 = 2
```



Observations and conclusions: Most of the applicant provided documnet 3, those people also have high tendency to default.

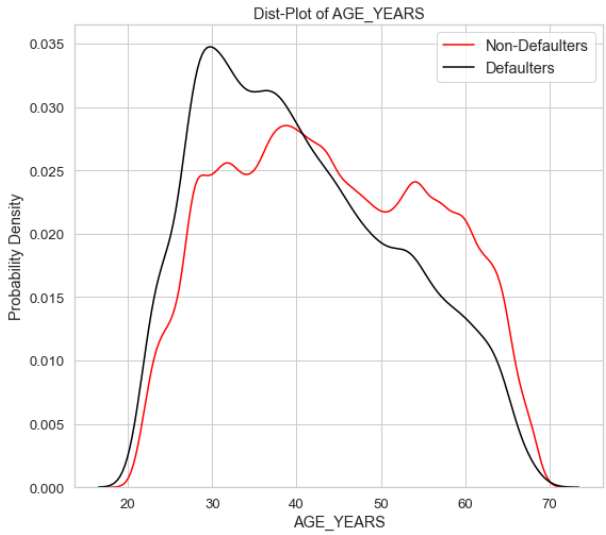
2.1.7 Plotting distribution of Continuous Variables

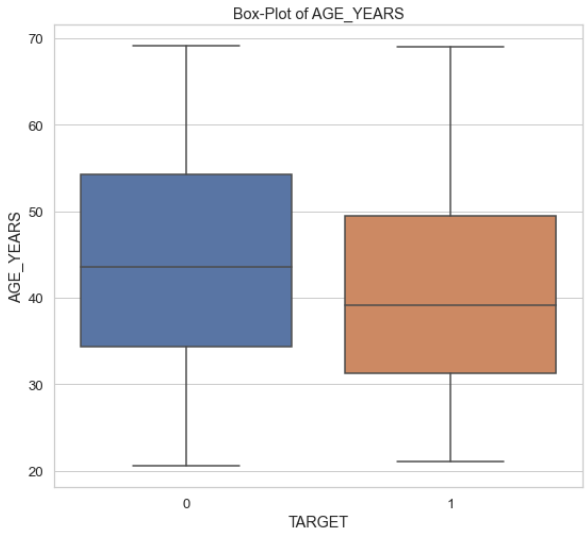
2.1.7.1 Age of Applicant

For the given dataset, the age is given in Days, which can be harder to interpret. Thus, we will create a latent variable to store the ages in Years, which would be easier to analyse and interpret..

In [43]:
```
app_train['AGE_YEARS'] = app_train['DAYS_BIRTH'] * -1 / 365
```

```
plot_continuous_variables(app_train, 'AGE_YEARS', plots = ['distplot','box'])
_ = app_train.pop('AGE_YEARS')
```





Observations and conclusions:

1. From the distplot, we can observe the peak of Age of people who Default to be close to 30 years. Also, at this point, the Non-Defaulters have a quite smaller PDF. One more thing to note is that the PDF of Age for Defaulters starts a bit left from the Non-Defaulters, and also is a bit throughtout the range. This means that the Defaulters are usually younger than Non-Defaulters.
2. From the box-plot too, we can better visualize the same thing. The Age of Defaulters is usually lesser than the Non-Defaulters. All the quantiles of ages of Defaulters is lesser than Non-Defaulters. The 75th percentile value of Non-Defaulters is around 54 years while for Defaulters it is near to 49 years.

These observations imply that the Defaulters are usually younger than Non-Defaulters.

2.1.7.2 DAYS features

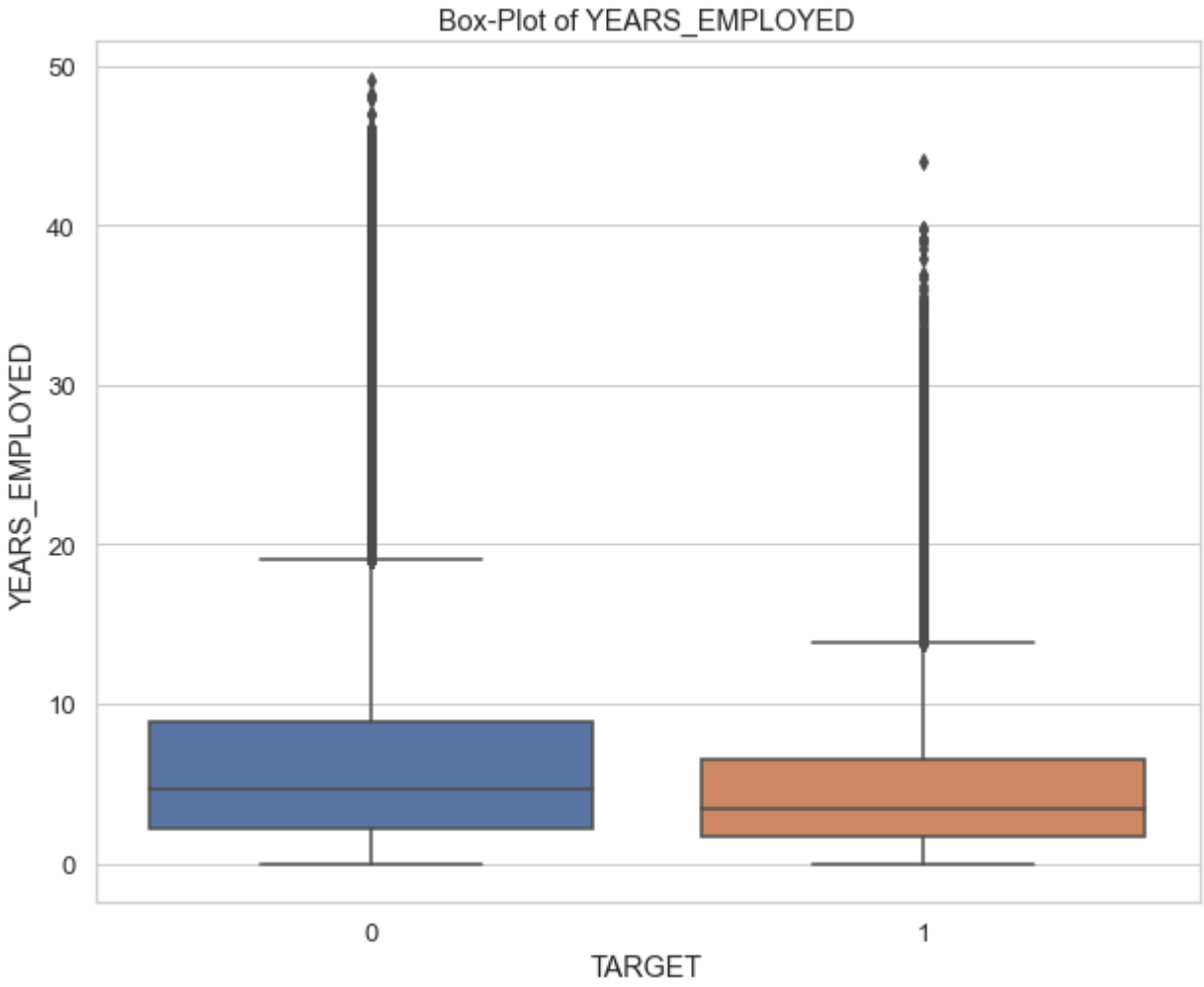
DAYS_EMPLOYED

This feature tells about the number of days from the day of application the applicant had been employed. For easy interpretation, we will convert the days to years.

In [44]:
```
app_train['YEARS_EMPLOYED'] = app_train.DAYS_EMPLOYED * -1 / 365
print_percentiles(app_train, 'DAYS_EMPLOYED')
plot_continuous_variables(app_train, 'YEARS_EMPLOYED', plots = ['box'], scale_limits =
_ = app_train.pop('YEARS_EMPLOYED')
```

```
--------------------------------------------------------------------------------
------------
The 0th percentile value of DAYS_EMPLOYED is -17912.0
The 25th percentile value of DAYS_EMPLOYED is -2760.0
The 50th percentile value of DAYS_EMPLOYED is -1213.0
The 75th percentile value of DAYS_EMPLOYED is -289.0
The 90th percentile value of DAYS_EMPLOYED is 365243.0
The 92th percentile value of DAYS_EMPLOYED is 365243.0
```

```
The 94th percentile value of DAYS_EMPLOYED is 365243.0
The 96th percentile value of DAYS_EMPLOYED is 365243.0
The 98th percentile value of DAYS_EMPLOYED is 365243.0
The 100th percentile value of DAYS_EMPLOYED is 365243.0
------------------------------------------------------------------------------------------
------------
```



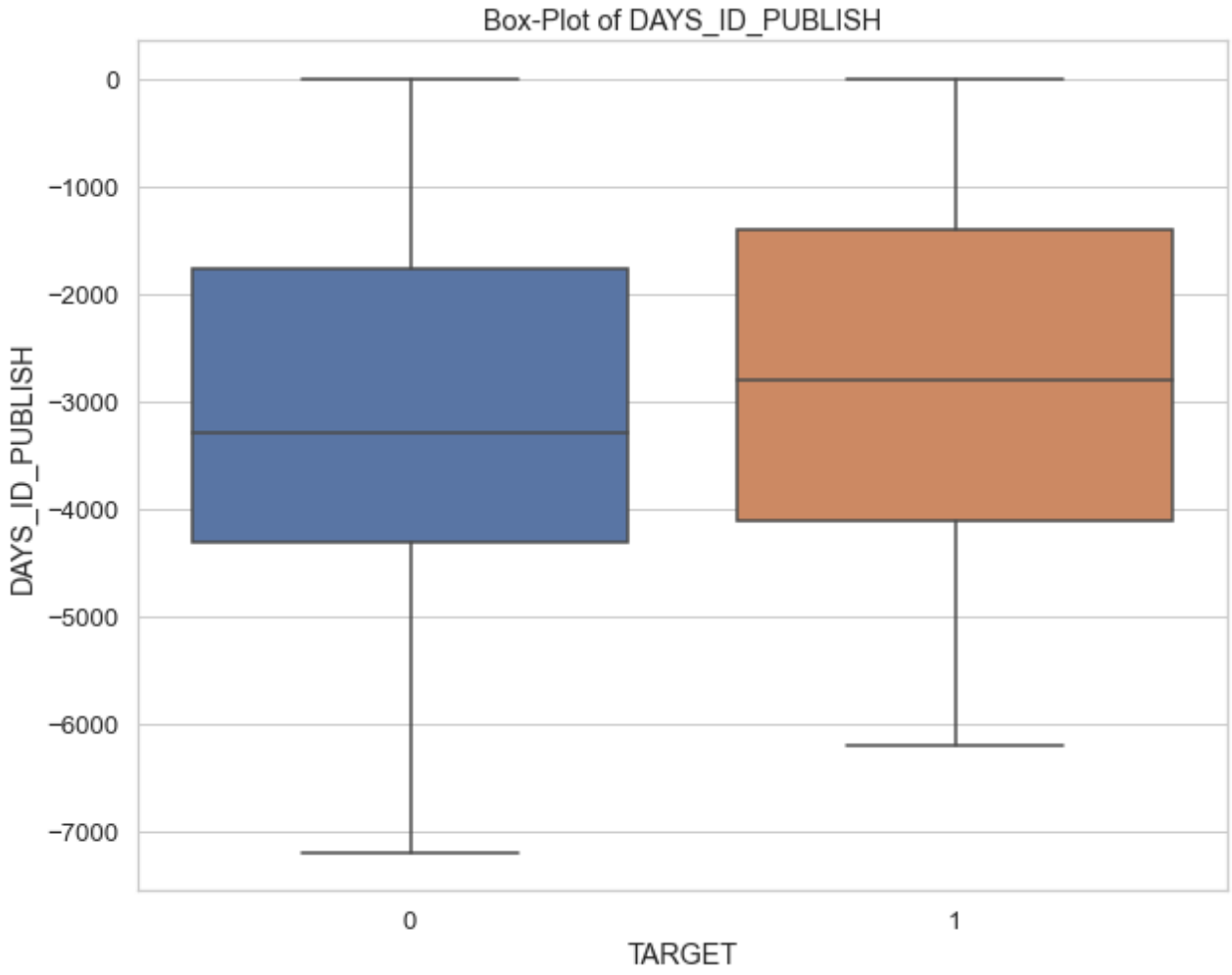
Observations and conclusions:

1. We see that the DAYS_EMPLOYED column contains some erroneous datapoints with values 365243. These seem like some erroneous/non-sensicle values.
2. From the box plot we observe that the Defaulters seem to have less number of years being employed as compared to Non-Defaulters. All the 25th, 50th and 75th quantile for Defaulters are lesser than those of Non-Defaulters.

DAYS_ID_PUBLISH

This columns tells about how many days ago from the day of registration did the client change his Identity Document with which he applied for loan.

In [45]:
```
plot_continuous_variables(app_train, 'DAYS_ID_PUBLISH', plots = ['box'], figsize = (10,
```



Observations and conclusions:

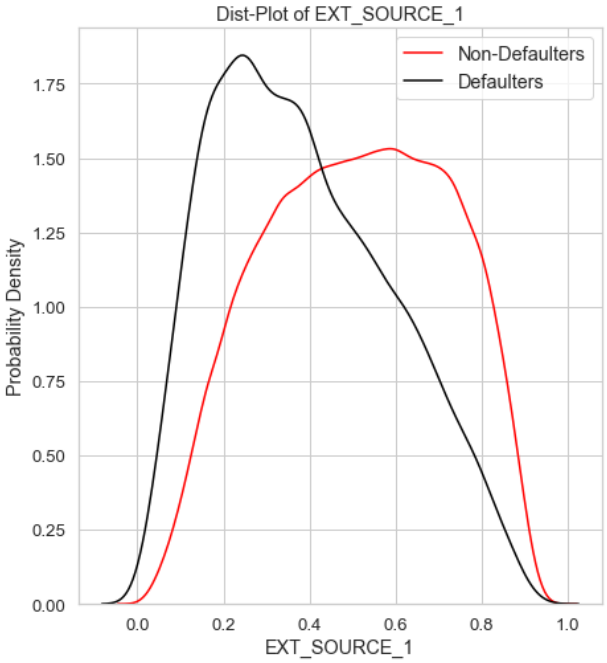
From the above box plot, we see a similar trend as seen with DAYS_REGISTRATION, in which the Defaulters usually had lesser number of days since they changed their identity. The Non-Defaulters show to have more number of days for all the quantiles since they changed their identity document.

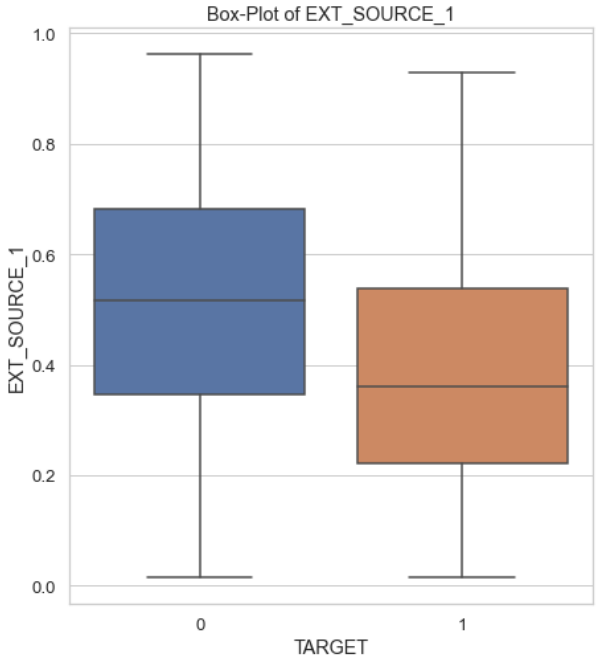
2.1.7.3 EXT_SOURCES

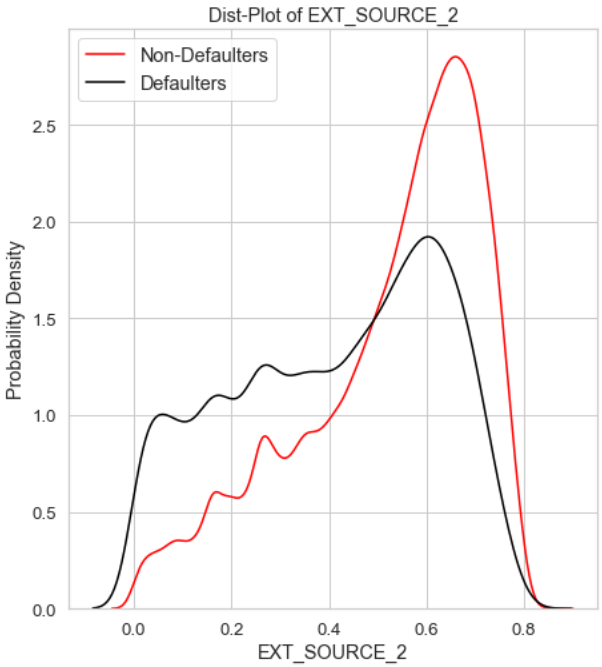
There are three EXT_SOURCES columns, which contain values between 0 and 1. They are normalized scores from different sources

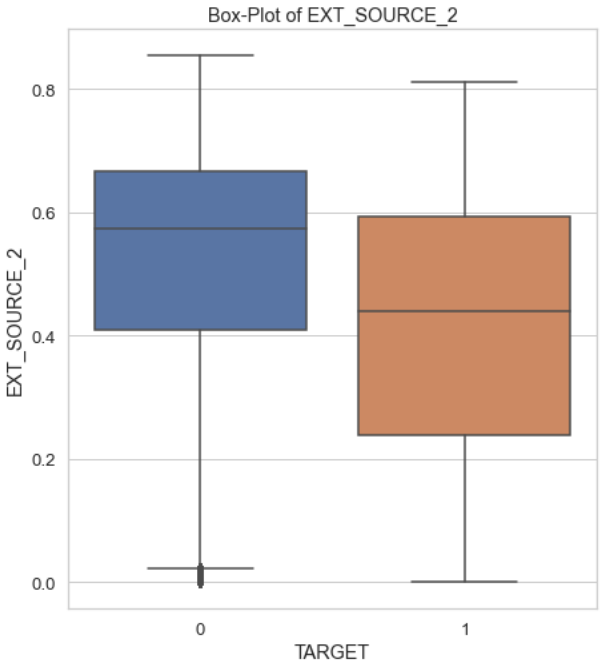
In [46]:
```
print('-'*100)
plot_continuous_variables(app_train, 'EXT_SOURCE_1', plots = ['distplot', 'box'], figsi
print('-'*100)
plot_continuous_variables(app_train, 'EXT_SOURCE_2', plots = ['distplot', 'box'], figsi
print('-'*100)
plot_continuous_variables(app_train, 'EXT_SOURCE_3', plots = ['distplot', 'box'], figsi
print('-'*100)
```

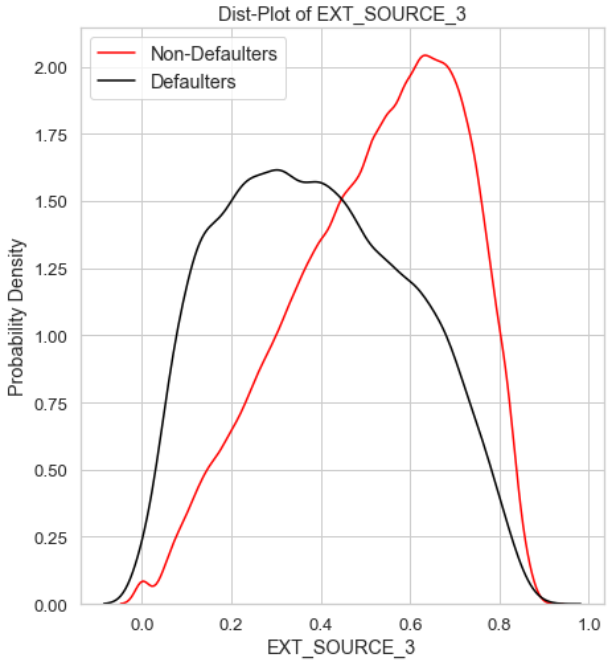
```
----------------------------------------------------------------------------------------
------------
```

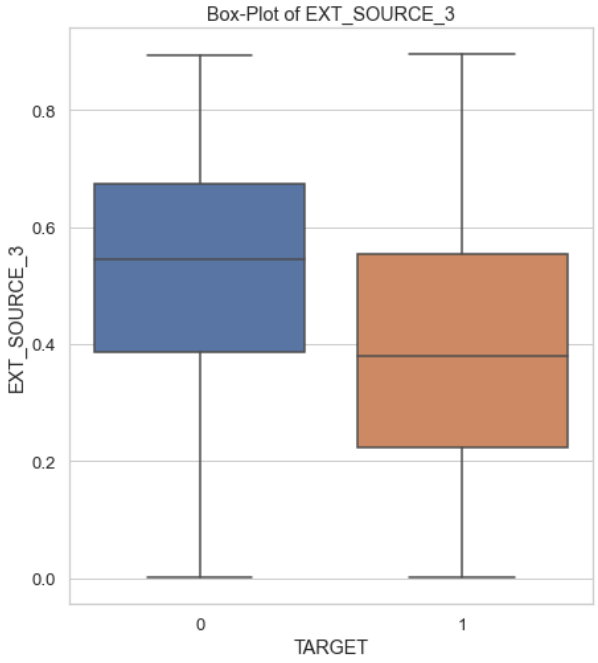

Dist-Plot of EXT_SOURCE_1
Non-Defaulters
Defaulters
Probability Density
EXT_SOURCE_1


Box-Plot of EXT_SOURCE_1
EXT_SOURCE_1
TARGET


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


Box-Plot of EXT_SOURCE_2
EXT_SOURCE_2
TARGET





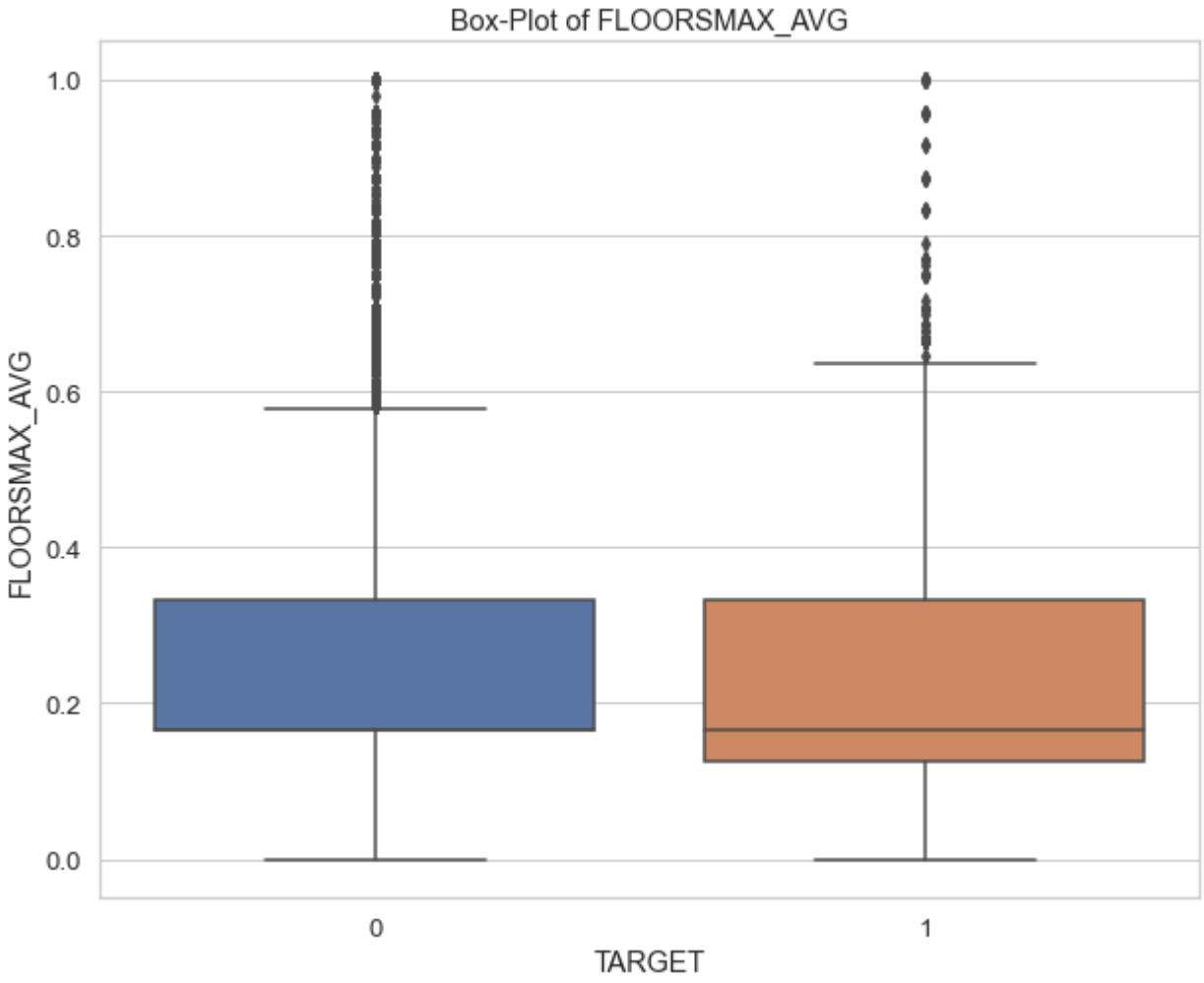
Observations and conclusions:

1. If we look at the box-plots, we can clearly see a similar trend for all three EXT_SOURCE columns, which is that the Defaulters tend to have considerably lower values.
2. This trend can also be seen with the PDFs. The Non-Defaulters show a higher peak at high EXT_SOURCE values, and the Probability Densities are very low for low values. This implies that Non-Defaulters generally have high values of these scores.
3. It is interesting to note that the median value for defaulters is almost equal to or lower than 25th percentile values of Non-Defaulters.
4. EXT_SOURCE_1 and EXT_SOURCE_3 columns tend to show better discrimination/separability as compared to EXT_SOURCE_2.
5. These 3 features look to be best separating the Defaulters and Non-Defaulters linearly among all the features so far.

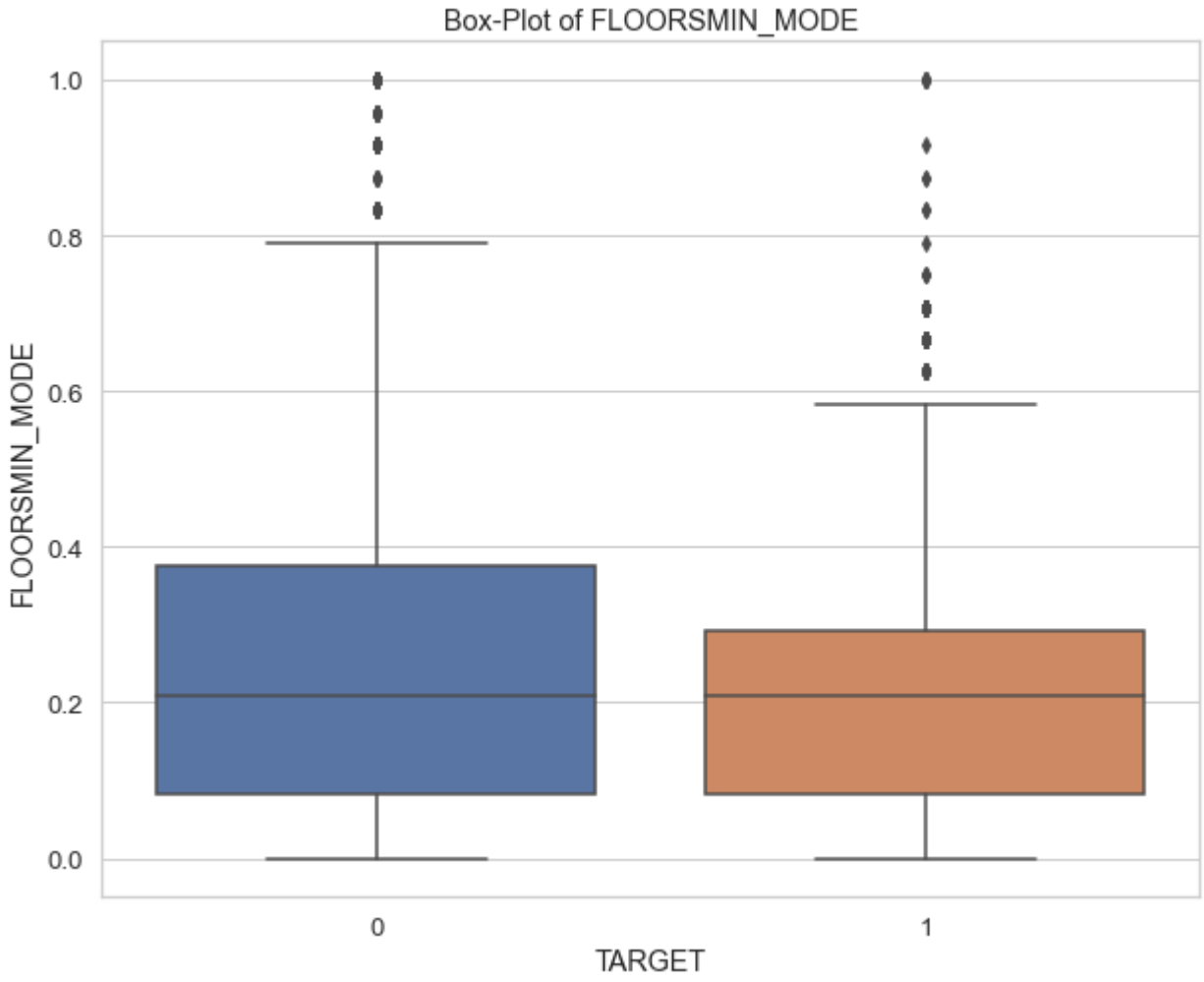
2.1.7.4 FLOORSMAX_AVG and FLOORSMIN_MODE

These columns describe the normalized scores of Average of Maximum number of Floors and Mode of Minimum number of Floors in applicant's building

In [47]:
```
plot_continuous_variables(app_train, 'FLOORSMAX_AVG', plots = ['box'], figsize = (10,8)
```



In [48]:
```
plot_continuous_variables(app_train, 'FLOORSMIN_MODE', plots = ['box'], figsize = (10,8
```



Observations and conclusions:

1. The defaulters have lower median value of FLOORSMAX_AVG feature as compared to Non-Defaulters. The 75th percentile values of both the Defaulters and Non-Defaulters is more or less the same, but the 25th percentile value of Non-Defaulters is almost more than the median of Defaulters, thus this could be an important feature.
2. The Non-Defaulters also tend to show a higher value of FLLORSMIN_MODE as compared to Defaulters. The 75th percentile value of Non-Defaulters is significantly higher than the 75th percentile value of Defaulters.

2.2 bureau.csv

Description:

This table consists of all client's previous credit records with financial institutions other than Home Credit Group which were reported by the the Credit Bureau.

2.2.1 Basic Stats

In [49]:
```
print(f'The shape of bureau.csv is: {bureau.shape}')
print('-'*100)
print(f'Number of unique SK_ID_BUREAU in bureau.csv are: {len(bureau.SK_ID_BUREAU.uniqu
```

```
print(f'Number of unique SK_ID_CURR in bureau.csv are: {len(bureau.SK_ID_CURR.unique())
print(f'Number of overlapping SK_ID_CURR in application_train.csv and bureau.csv are:
      {len(set(app_train.SK_ID_CURR.unique()).intersection(set(bureau.SK_ID_CURR.unique
print(f'Number of overlapping SK_ID_CURR in application_test.csv and bureau.csv are:
      {len(set(app_test.SK_ID_CURR.unique()).intersection(set(bureau.SK_ID_CURR.unique(
print('-'*100)
print(f'Number of duplicate values in bureau: {bureau.shape[0] - bureau.duplicated().sh
print('-'*100)
display(bureau.head(5))
```

```
The shape of bureau.csv is: (1716428, 17)
----------------------------------------------------------------------------------------
------------
Number of unique SK_ID_BUREAU in bureau.csv are: 1716428
Number of unique SK_ID_CURR in bureau.csv are: 305811
Number of overlapping SK_ID_CURR in application_train.csv and bureau.csv are: 263491
Number of overlapping SK_ID_CURR in application_test.csv and bureau.csv are: 42320
----------------------------------------------------------------------------------------
------------
Number of duplicate values in bureau: 0
----------------------------------------------------------------------------------------
------------
```

	SK_ID_CURR	SK_ID_BUREAU	CREDIT_ACTIVE	CREDIT_CURRENCY	DAYS_CREDIT	CREDIT_DAY_OVERDU
0	215354	5714462	Closed	currency 1	-497	
1	215354	5714463	Active	currency 1	-208	
2	215354	5714464	Active	currency 1	-203	
3	215354	5714465	Active	currency 1	-203	
4	215354	5714466	Active	currency 1	-629	

Observations and conclusions:

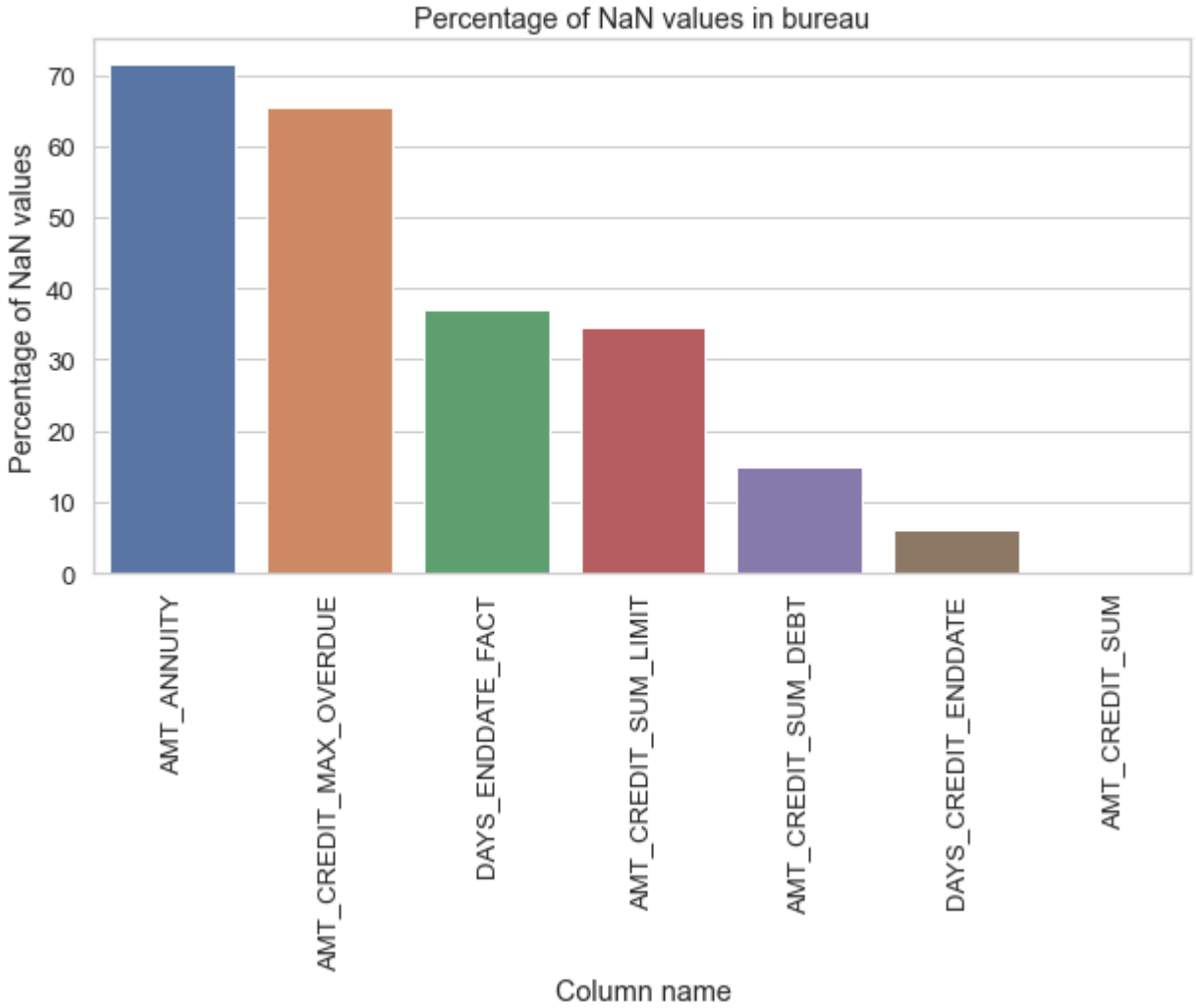
1. The bureau.csv file contains close to 1.7M datapoints, with 17 features. Out of these 17 features, two are SK_ID_CURR and SK_ID_BUREAU
   - SK_ID_BUREAU is the loan ID of the client's previous loan from other financial institutions. There may be multiple previous loans corresponding to a single SK_ID_CURR which depends on client's borrowing pattern.
   - SK_ID_CURR is the loan ID of client's current loan with Home Credit.
   - The rest of the features contain other stats such as DAYS_CREDIT, AMT_CREDIT_SUM, CREDIT_TYPE, etc.
2. There are 305k unique SK_ID_CURR in bureau out of which:
   - There are 263k SK_ID_CURR in bureau which are present in application_train out of total of total 307k of application_train's unique SK_ID_CURR. This means that some of the applicants in current loan application with Home Credit Group do not have any previous Credit history with Credit Bureau Department.

- Similarly, there are 42.3k SK_ID_CURR in bureau which are present in application_test, out of total 48k of application_test's unique SK_ID_CURR.

2.2.2 NaN columns and percentages

In [52]:
```
nan_df_bureau = nan_df_create(bureau)
print("-"*100)
plot_nan_percent(nan_df_bureau, 'bureau', tight_layout = False, figsize = (10,5))
print('-'*100)
```

```
----------------------------------------------------------------------------------------------------
Number of columns having NaN values: 7 columns
```




```
----------------------------------------------------------------------------------------------------
```

Observations and conclusions:

1. Out of 17 features, there are 7 features which contain NaN values.
2. The highest NaN values are observed with the column AMT_ANNUITY which has over 70% missing values.

2.2.3 Merging the TARGETS from application_train to bureau table.

In [54]:
```
print("-"*100)
print("Merging TARGET with bureau Table")
bureau_merged = app_train.iloc[:,:2].merge(bureau, on = 'SK_ID_CURR', how = 'left')
print("-"*100)
```

```
----------------------------------------------------------------------------------------------------
Merging TARGET with bureau Table
----------------------------------------------------------------------------------------------------
```

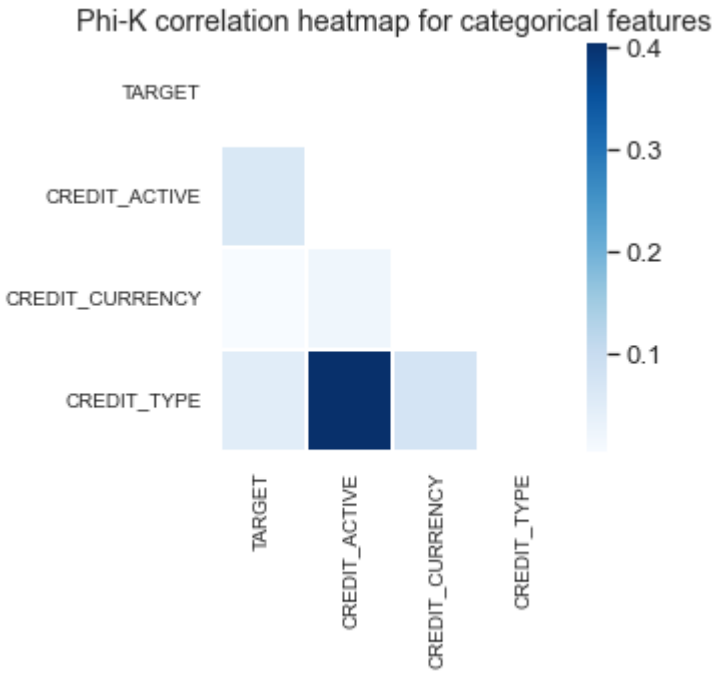
In [55]:
```
print('app_train shape: ', app_train.shape)
print('bureau shape: ', bureau.shape)
print('bureau_merged shape', bureau_merged.shape)
```

```
app_train shape:  (307511, 122)
bureau shape:  (1716428, 17)
bureau_merged shape (1509345, 18)
```

2.2.4 Phi-K matrix

In [56]:
```
cols_for_phik = ['TARGET','CREDIT_ACTIVE','CREDIT_CURRENCY','CREDIT_TYPE']
plot_phik_matrix(bureau_merged, cols_for_phik,  figsize = (5,5))
```

```
----------------------------------------------------------------------------------------------------
```




```
----------------------------------------------------------------------------------------------------
Categorise with highest values of Phi-K correlation value with target variable are:
```

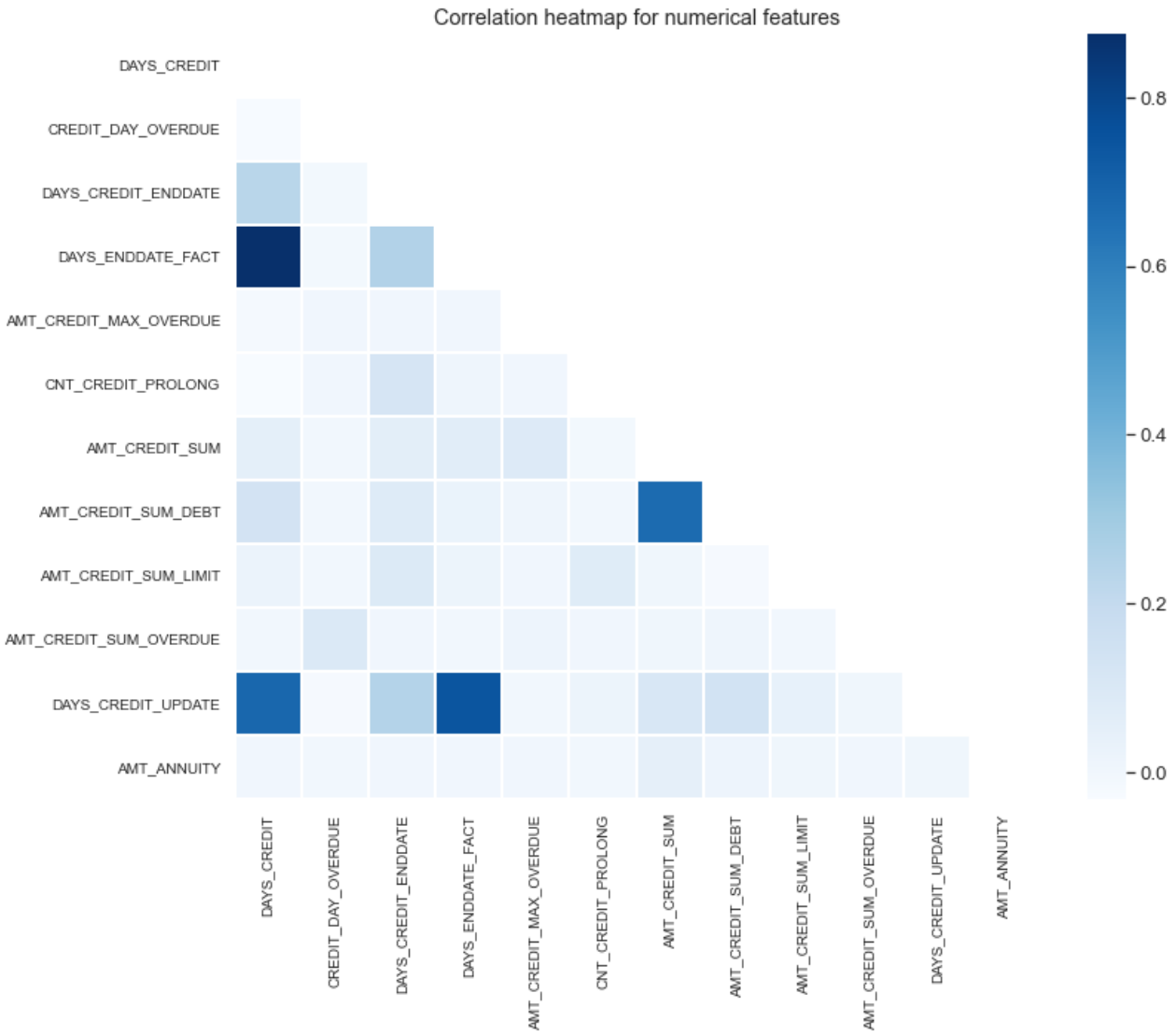
	Column Name	Phik-Correlation
0	CREDIT_ACTIVE	0.064481
2	CREDIT_TYPE	0.049954
1	CREDIT_CURRENCY	0.004993

Observations and conclusions:

1. From the Phi-K Correlation Coefficient, we see that the variable CREDIT_TYPE shows some association with the variable CREDIT_ACTIVE.
2. We see that the Categorical Variables don't really have a high association with TARGET variable, especially the CREDIT_CURRENCY feature.

2.2.5 Correlation Matrix of Features

In [57]:
```
corr_mat = correlation_matrix(bureau_merged, ['SK_ID_CURR','SK_ID_BUREAU'], cmap = 'Blu
                              figsize = (12,10))
corr_mat.plot_correlation_matrix()
```



In [58]:
```
#Seeing the top columns with highest phik-correlation with the target variable in burea
```

```
top_corr_target_df = corr_mat.target_top_corr()
print("-" * 100)
print("Columns with highest values of Phik-correlation with Target Variable are:")
display(top_corr_target_df)
print("-"*100)
```

```
interval columns not set, guessing: ['TARGET', 'DAYS_CREDIT']
interval columns not set, guessing: ['TARGET', 'CREDIT_DAY_OVERDUE']
interval columns not set, guessing: ['TARGET', 'DAYS_CREDIT_ENDDATE']
interval columns not set, guessing: ['TARGET', 'DAYS_ENDDATE_FACT']
interval columns not set, guessing: ['TARGET', 'AMT_CREDIT_MAX_OVERDUE']
interval columns not set, guessing: ['TARGET', 'CNT_CREDIT_PROLONG']
interval columns not set, guessing: ['TARGET', 'AMT_CREDIT_SUM']
interval columns not set, guessing: ['TARGET', 'AMT_CREDIT_SUM_DEBT']
interval columns not set, guessing: ['TARGET', 'AMT_CREDIT_SUM_LIMIT']
interval columns not set, guessing: ['TARGET', 'AMT_CREDIT_SUM_OVERDUE']
interval columns not set, guessing: ['TARGET', 'DAYS_CREDIT_UPDATE']
interval columns not set, guessing: ['TARGET', 'AMT_ANNUITY']
----------------------------------------------------------------------------------------------------
Columns with highest values of Phik-correlation with Target Variable are:
```

	Column Name	Phik-Correlation
0	DAYS_CREDIT	0.088651
2	DAYS_CREDIT_ENDDATE	0.018980
9	AMT_CREDIT_SUM_OVERDUE	0.005654
8	AMT_CREDIT_SUM_LIMIT	0.005192
4	AMT_CREDIT_MAX_OVERDUE	0.004280
5	CNT_CREDIT_PROLONG	0.003862
1	CREDIT_DAY_OVERDUE	0.002528
10	DAYS_CREDIT_UPDATE	0.002219
7	AMT_CREDIT_SUM_DEBT	0.001695
6	AMT_CREDIT_SUM	0.000670

```
----------------------------------------------------------------------------------------------------
```

Observations and conclusions:

1. It can be observed that most of the heatmap has light colors, which shows little to no correlation.
2. However, we can see some dark shades which represent high correlation.
3. The high correlation is particularly observed for features:
   - DAYS_CREDIT and DAYS_CREDIT_UPDATE
   - DAYS_ENDDATE_FACT and DAYS_CREDIT_UPDATE
   - AMT_CREDIT_SUM and AMT_CREDIT_SUM_DEBT
   - DAYS_ENDDATE_FACT and DAYS_CREDIT
4. We can also see that the features don't particularly show good/high correlation with Target as such, except for DAYS_CREDIT feature. This implies that there isn't a direct linear relation

between Target and the features.

2.2.6 Plotting distribution of Categorical Variables

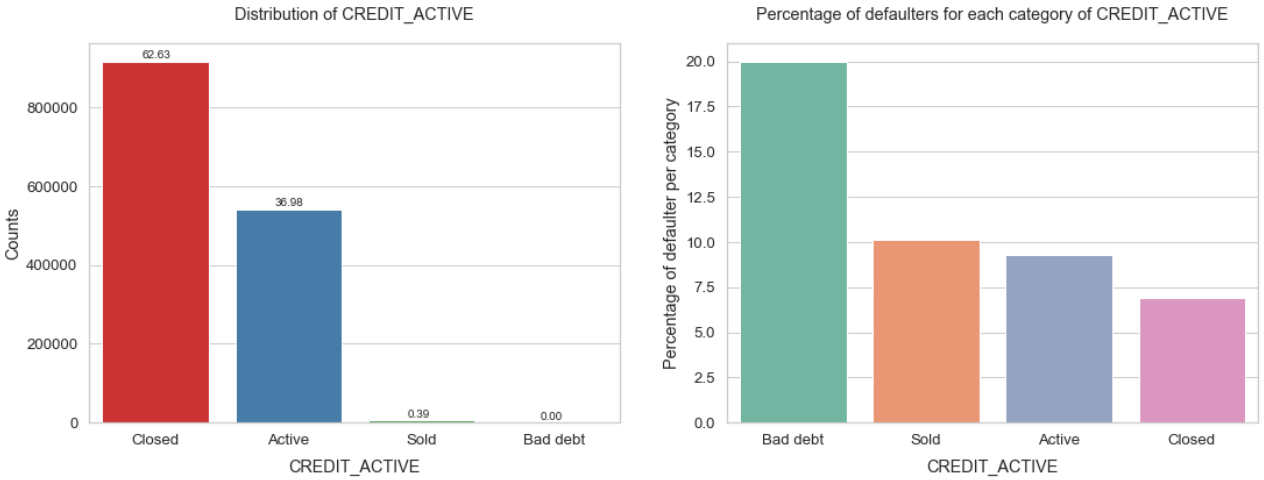
We will now plot some of the Categorical Variables of the table bureau, and see their impact on the Target Variable.

2.2.6.1 CREDIT_ACTIVE

This column describes the Status of the previous loan reported from Credit Bureau.

In [59]:
```
#Let us first see the unique categories of 'CREDIT_ACTIVE'
print_unique_categories(bureau_merged, 'CREDIT_ACTIVE', show_counts = True)

# plotting the Bar Plot for the Column
plot_categorical_variables_bar(bureau_merged, column_name = 'CREDIT_ACTIVE', horizontal
print('-'*100)
```

```
----------------------------------------------------------------------------------------------------
The unique categories of 'CREDIT_ACTIVE' are:
['Closed' 'Active' nan 'Sold' 'Bad debt']
----------------------------------------------------------------------------------------------------
Counts of each category are:
Closed      917733
Active      541919
Sold          5653
Bad debt        20
Name: CREDIT_ACTIVE, dtype: int64
----------------------------------------------------------------------------------------------------
Total number of unique categories of CREDIT_ACTIVE = 5
```




```
----------------------------------------------------------------------------------------------------
```

Observations and conclusions:

1. From the first subplot, we see that a majority of the previous loans from other financial institutions are Closed Loans (62.63%), followed by 36.98% active loans. The sold and Bad-Debt Loans are very less in number.

2. If we look at the Defaulters Percentage per category, we see that about 20% of people from Bad-Debt defaulted, which is the highest default rate. This is followed by Sold loans and Active Loans. The lowest default rate is for Closed Loans, which show a good history about a client. Thus the patten observed here is quite logical and expected.

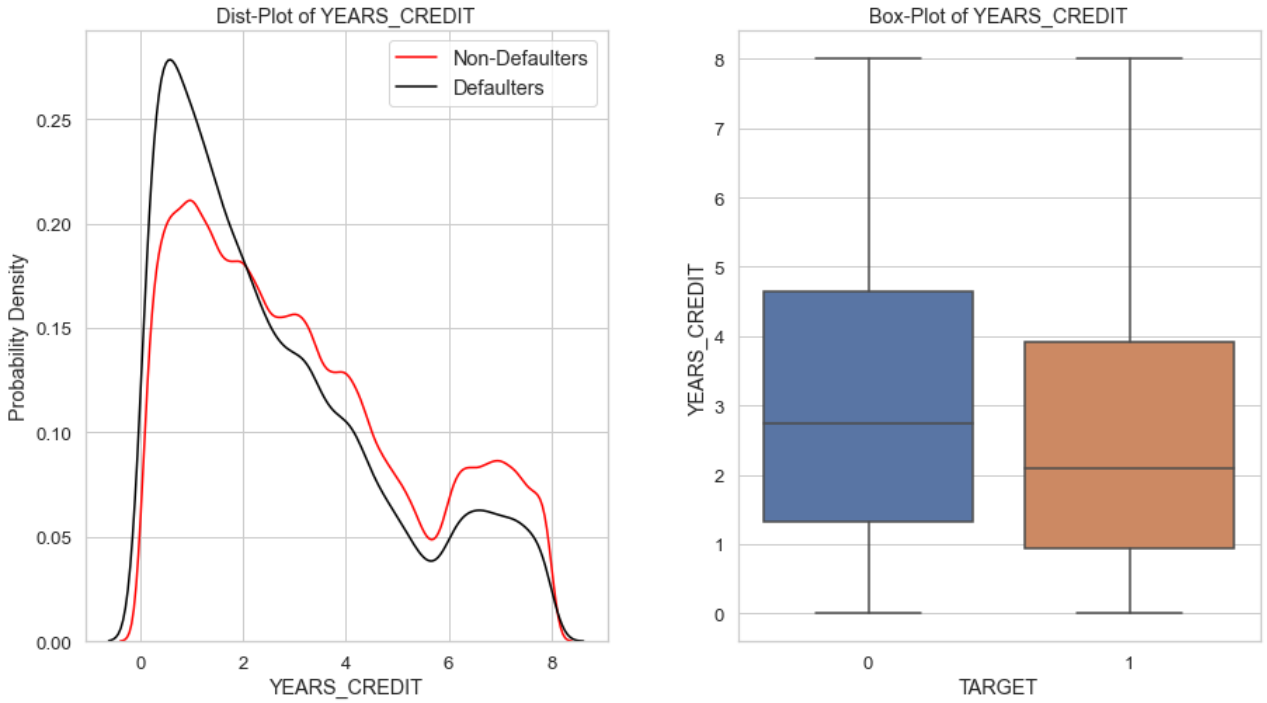
2.2.7 Plotting distribution of Continuous Variables

2.2.7.1 DAYS features

DAYS_CREDIT

This column describes about the number of days before current application when the client applied for Credit Bureau Credit. For ease of interpretability, we will convert these days to years.

In [60]:
```
bureau_merged['YEARS_CREDIT'] = bureau_merged['DAYS_CREDIT'] / -365
plot_continuous_variables(bureau_merged, 'YEARS_CREDIT', plots = ['distplot', 'box'],
                          figsize = (15,8))
_ = bureau_merged.pop('YEARS_CREDIT')
```



Observations and conclusions:

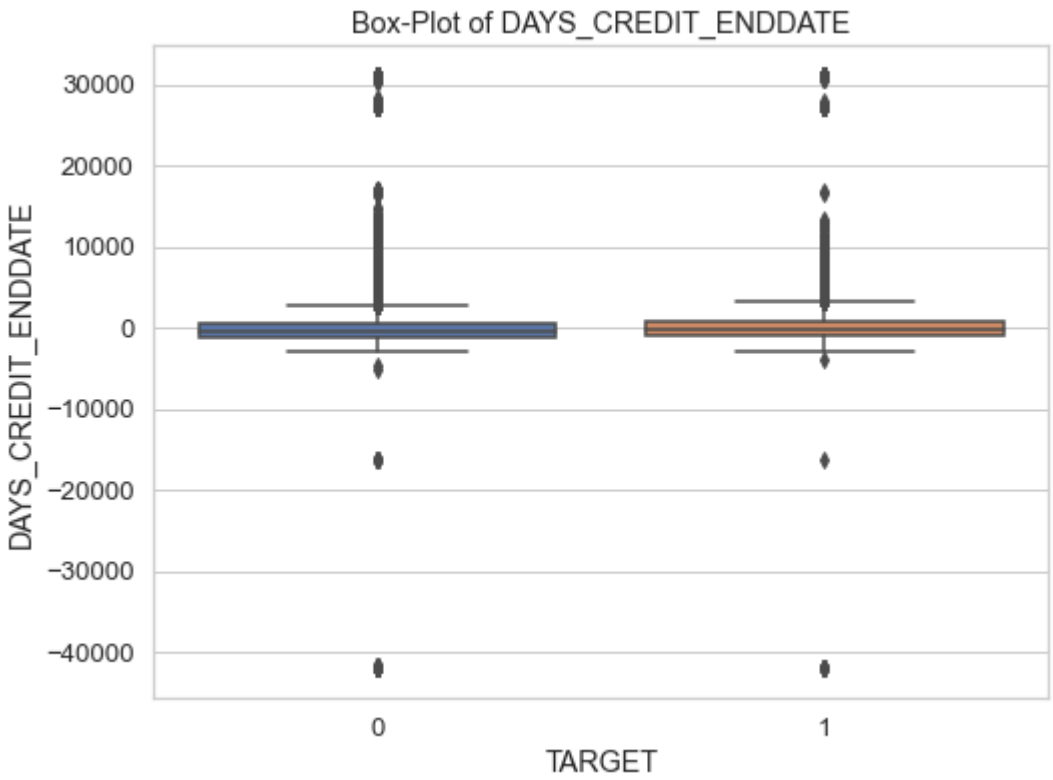
1. From the PDF, we see that the Defaulters tend to have higher peaks compared to Non-Defaulters when the number of years are less.. This implies that the applicants who had applied for loans from Credit Bureau recently showed more defaulting tendency than those who had applied long ago. The PDF of Defaulters is also a bit towards left as compared to Non-Defaulters.
2. Fro the box-plot as well, we see that Defaulters usually had less YEARS_CREDIT as compared to Non-Defaulters.

DAYS_CREDIT_ENDDATE

This column tells about the remaining duration of Credit Bureau credit at the time of application for loan in Home Credit.

In [61]:
```
print_percentiles(bureau_merged, 'DAYS_CREDIT_ENDDATE',
                  percentiles = list(range(0,11,2)) + [25,50,75,100])
plot_continuous_variables(bureau_merged, 'DAYS_CREDIT_ENDDATE', plots = ['box'],
                          figsize = (8,6))
print('-'*100)
```

```
----------------------------------------------------------------------------------------------------
The 0th percentile value of DAYS_CREDIT_ENDDATE is -42060.0
The 2th percentile value of DAYS_CREDIT_ENDDATE is -2487.0
The 4th percentile value of DAYS_CREDIT_ENDDATE is -2334.0
The 6th percentile value of DAYS_CREDIT_ENDDATE is -2202.0
The 8th percentile value of DAYS_CREDIT_ENDDATE is -2073.9199999999983
The 10th percentile value of DAYS_CREDIT_ENDDATE is -1939.0
The 25th percentile value of DAYS_CREDIT_ENDDATE is -1144.0
The 50th percentile value of DAYS_CREDIT_ENDDATE is -334.0
The 75th percentile value of DAYS_CREDIT_ENDDATE is 473.0
The 100th percentile value of DAYS_CREDIT_ENDDATE is 31199.0
----------------------------------------------------------------------------------------------------
```




```
----------------------------------------------------------------------------------------------------
```

Observations and conclusions:

From the above percentile values, and looking at the box-plot, we see that there seems to be erroneous value for DAYS_CREDIT_ENDDATE, where the 0th percentile value dates back to as long as 42060 days or 115 years. This does not make much sense as this implies that the previous loan the

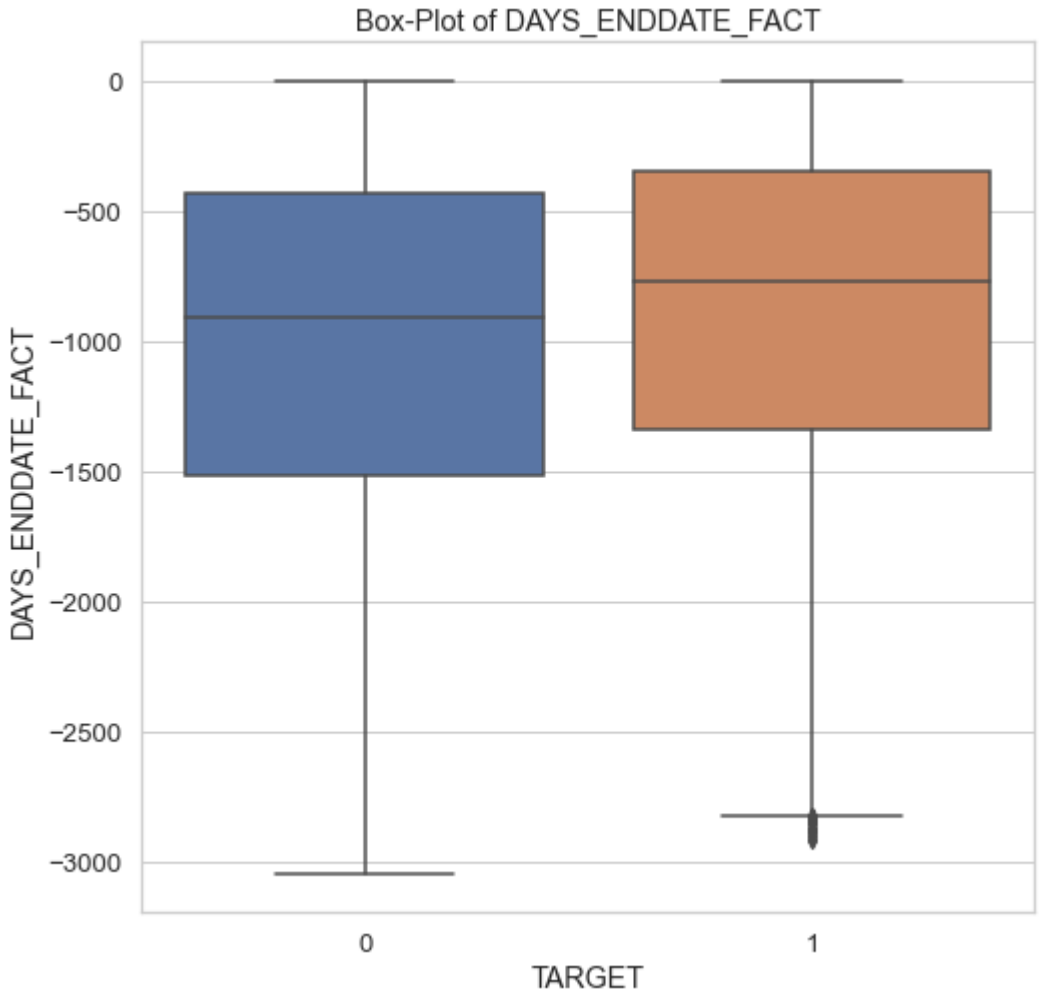
client had dates back to 115 years ago. This could be inherited loan too, but we cannot comment so surely about that. We would try to remove these values in the data preprocessing stage.

DAYS_ENDDATE_FACT

This column tells about the the number of days ago that the Credit Bureau credit had ended at the time of application for loan in Home Credit. These values are only for Closed Credits.

In [62]:
```
print_percentiles(bureau_merged, 'DAYS_ENDDATE_FACT',
                  percentiles = list(range(0,11,2)) + [25,50,75,100])
plot_continuous_variables(bureau_merged, 'DAYS_ENDDATE_FACT', plots = ['box'],
                          figsize = (8,8), scale_limits = [-40000, 0])
print('-'*100)
```

```
----------------------------------------------------------------------------------------------------
The 0th percentile value of DAYS_ENDDATE_FACT is -42023.0
The 2th percentile value of DAYS_ENDDATE_FACT is -2561.0
The 4th percentile value of DAYS_ENDDATE_FACT is -2450.0
The 6th percentile value of DAYS_ENDDATE_FACT is -2351.0
The 8th percentile value of DAYS_ENDDATE_FACT is -2265.0
The 10th percentile value of DAYS_ENDDATE_FACT is -2173.0
The 25th percentile value of DAYS_ENDDATE_FACT is -1503.0
The 50th percentile value of DAYS_ENDDATE_FACT is -900.0
The 75th percentile value of DAYS_ENDDATE_FACT is -427.0
The 100th percentile value of DAYS_ENDDATE_FACT is 0.0
----------------------------------------------------------------------------------------------------
```



Observations and conclusions:

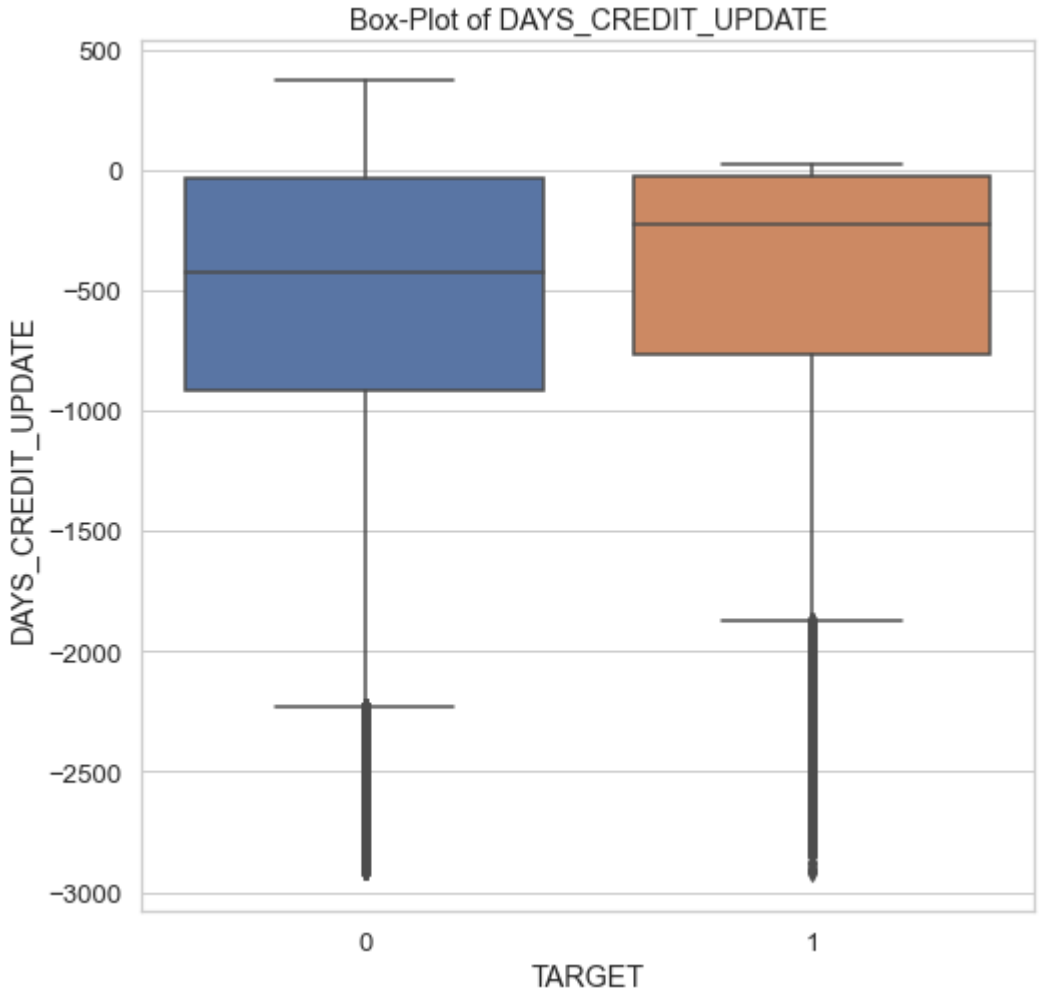
1. Just like previous column, we see that the 0th percentile for this column also seems erroneous, which is 42023 days or ~115 years. We would have to remove these values, as they don't make much sense.
2. Looking at the box-plot, we see that the Defaulters tend to have lesser number of days since their Credit Bureau credit had ended. The Non-Defaulters usually have their previous credits ended longer before than Defaulters.

DAYS_CREDIT_UPDATE

This column tells about the number of days ago that the information from Credit Bureau credit had come at the time of application for loan in Home Credit.

In [63]:
```
print_percentiles(bureau_merged, 'DAYS_CREDIT_UPDATE', percentiles = list(range(0,11,2))
                  + [25,50,75,100])
plot_continuous_variables(bureau_merged, 'DAYS_CREDIT_UPDATE', plots = ['box'],
                          figsize = (8,8), scale_limits = [-40000, 400])
print('-'*100)
```

```
The 0th percentile value of DAYS_CREDIT_UPDATE is -41947.0
The 2th percentile value of DAYS_CREDIT_UPDATE is -2415.0
The 4th percentile value of DAYS_CREDIT_UPDATE is -2213.0
The 6th percentile value of DAYS_CREDIT_UPDATE is -2002.0
The 8th percentile value of DAYS_CREDIT_UPDATE is -1766.0
The 10th percentile value of DAYS_CREDIT_UPDATE is -1582.0
The 25th percentile value of DAYS_CREDIT_UPDATE is -904.0
The 50th percentile value of DAYS_CREDIT_UPDATE is -406.0
The 75th percentile value of DAYS_CREDIT_UPDATE is -33.0
The 100th percentile value of DAYS_CREDIT_UPDATE is 372.0
--------------------------------------------------------------------------------
------------
```




```
--------------------------------------------------------------------------------
------------
```

Observations and conclusions:

1. The trend of erroneious values is again very similar to the other days column where this 0th percentile value seems to be erroneous. Also since only the 0th percentile value is so odd, and the rest seem to be fine, thus this value is definitely erroneous. We will be removing this value too.
2. From the box-plot, we can say that the Defaulters tend to have a lesser number of days since their Information about the Credit Bureau Credit were received. Their median, 75th percentile values all are lesser than those for Non-Defaulters.

2.3 bureau_balance.csv

Description:

This table consists of Monthly balance of each credit for each of the previous credit that the client had with financial institutions other than Home Credit.

2.3.1 Basic Stats

In [64]:
```
print(bureau_balance.shape)
print(bureau_balance.duplicated().shape)
```

```
(27299925, 3)
(27299925,)
```

In [65]:
```
print(f'The shape of bureau_balance.csv is: {bureau_balance.shape}')
print('-'*100)
print(f'Number of duplicate values in bureau_balance: {bureau_balance.shape[0] - bureau
print('-'*100)
display(bureau_balance.head(5))
```

```
The shape of bureau_balance.csv is: (27299925, 3)
----------------------------------------------------------------------------------------------------
Number of duplicate values in bureau_balance: 0
----------------------------------------------------------------------------------------------------
```

	SK_ID_BUREAU	MONTHS_BALANCE	STATUS
0	5715448	0	C
1	5715448	-1	C
2	5715448	-2	C
3	5715448	-3	C
4	5715448	-4	C

In [66]:
```
print("-"*100)
print(f'Number of unique SK_ID_BUREAU in bureau_balance.csv are: {len(bureau_balance.SK
print('-'*100)
print(f'Number of unique values for STATUS are: {len(bureau_balance.STATUS.unique())}')
print(f"Unique values of STATUS are:\n{bureau_balance.STATUS.unique()}")
print('-'*100)
print(f"Max number of months for Months Balance: {np.abs(bureau_balance.MONTHS_BALANCE.
print('-'*100)
```

```
----------------------------------------------------------------------------------------------------
Number of unique SK_ID_BUREAU in bureau_balance.csv are: 817395
----------------------------------------------------------------------------------------------------
Number of unique values for STATUS are: 8
Unique values of STATUS are:
```

```
['C' '0' 'X' '1' '2' '3' '5' '4']
------------------------------------------------------------------------------------------
------------
Max number of months for Months Balance: 96
------------------------------------------------------------------------------------------
------------
```

Observations and conclusions:

1. The bureau_balance.csv table contains approximately 27.29M rows, and 3 columns.
2. This table contains the monthly status for each of the previous loan for a particular applicant reported by the Credit Bureau Department.
3. There are 8 unique values for the STATUS which are encoded. Each of them have a special meaning. C means closed, X means status unknown, 0 means no DPD, 1 means maximal did during month between 1-30, 2 means DPD 31-60,... 5 means DPD 120+ or sold or written off.
4. The most earliest month's balance that we have is the 96 months back status, i.e. the Status has been provided upto 8 years of history for loans for which those exist.

2.3.2 NaN Columns and Percentages

In [67]:
```
plot_nan_percent(nan_df_create(bureau_balance), 'bureau_balance')
```

```
The dataframe bureau_balance does not contain any NaN values
```

2.4 previous_application.csv

Description:

This table contains the static data of the previous loan which the client had with Home Credit.

2.4.1 Basic Stats

In [68]:
```
print(f'The shape of previous_application.csv is: {pre_app.shape}')
print('-'*100)
print(f'Number of unique SK_ID_PREV in previous_application.csv are:
      {len(pre_app.SK_ID_PREV.unique())}')
print(f'Number of unique SK_ID_CURR in previous_application.csv are:
      {len(pre_app.SK_ID_CURR.unique())}')
print('-'*100)
print(f'Number of overlapping SK_ID_CURR in application_train.csv and previous_applicat
      {len(set(app_train.SK_ID_CURR.unique()).intersection(set(pre_app.SK_ID_CURR.uniqu
print(f'Number of overlapping SK_ID_CURR in application_test.csv and previous_applicati
      {len(set(app_test.SK_ID_CURR.unique()).intersection(set(pre_app.SK_ID_CURR.unique
print('-'*100)
print(f'Number of duplicate values in previous_application:
      {pre_app.shape[0] - pre_app.duplicated().shape[0]}')
print('-'*100)
display(pre_app.head(5))
```

```
The shape of previous_application.csv is: (1670214, 37)
------------------------------------------------------------------------------------------
------------
Number of unique SK_ID_PREV in previous_application.csv are: 1670214
```

```
Number of unique SK_ID_CURR in previous_application.csv are: 338857
------------------------------------------------------------------------------------------
------------
Number of overlapping SK_ID_CURR in application_train.csv and previous_application.csv a
re: 291057
Number of overlapping SK_ID_CURR in application_test.csv and previous_application.csv ar
e: 47800
------------------------------------------------------------------------------------------
------------
Number of duplicate values in previous_application: 0
------------------------------------------------------------------------------------------
------------
```

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT
0	2030495	271877	Consumer loans	1730.430	17145.0	17145.0
1	2802425	108129	Cash loans	25188.615	607500.0	679671.0
2	2523466	122040	Cash loans	15060.735	112500.0	136444.5
3	2819243	176158	Cash loans	47041.335	450000.0	470790.0
4	1784265	202054	Cash loans	31924.395	337500.0	404055.0

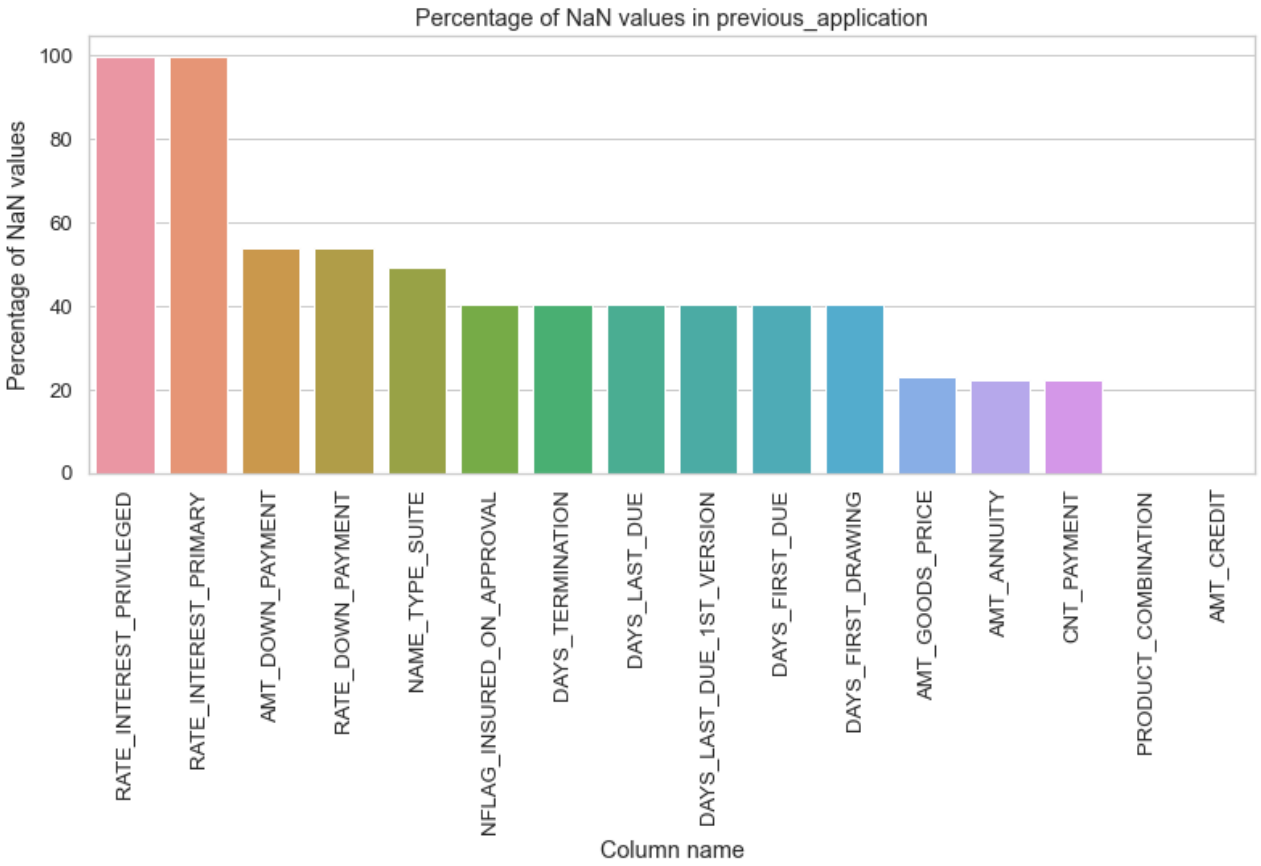
Observations and conclusions:

1. The table previous_application.csv consists of 1.67M rows in total. Each row corresponds to each of the previous loan that the client had with previously with Home Credit Group. It is possible for a single client of current application to have multiple previous loans with Home Credit Group.
2. There are 37 columns in previous_application.csv, which contain the details about the previous loan.
3. There are 338k unique SK_ID_CURR in previous_application, of which 291k correspond to the application_train SK_ID_CURRs and 47.8k correspond to application_test SK_ID_CURRs.

2.4.2 NaN Columns and Percentages

In [69]:
```
previous_application_nan = nan_df_create(pre_app)
print('-' * 100)
plot_nan_percent(previous_application_nan, 'previous_application', tight_layout = False
print('-' * 100)
del previous_application_nan
```

```
------------------------------------------------------------------------------------------
------------
Number of columns having NaN values: 16 columns
```



Observations and conclusions:

1. There are 16 columns out of the 37 columns which contain NaN values.
2. Two of these columns have 99.64% missing values, which is very high, and we will have to come up with some smart way to handle such high NaN values. We cannot directly discard any feature at this point.
3. Other than these two columns, rest of the columns also contain > 40% NaN values, except for 5 columns.

2.4.3 Merging the TARGETS from application_train to previous_application table.

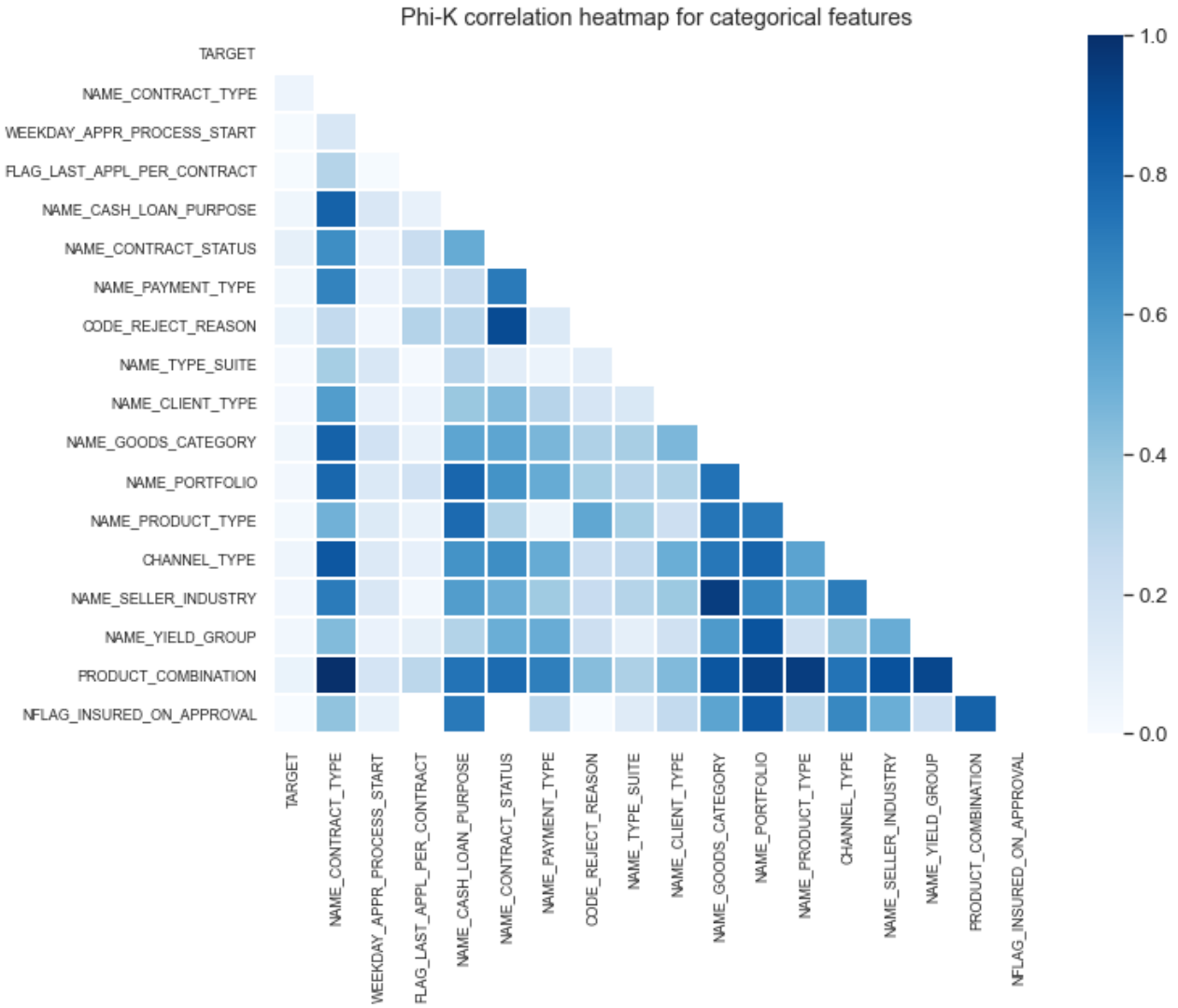
In [70]:
```
print("-"*100)
print("Merging TARGET with previous_application Table")
prev_merged = app_train.iloc[:,:2].merge(pre_app, on = 'SK_ID_CURR', how = 'left')
print("-"*100)
```

```
----------------------------------------------------------------------------------------------------
Merging TARGET with previous_application Table
----------------------------------------------------------------------------------------------------
```

2.4.4 Phi-K Matrix

In [71]:

```
cols_for_phik = ['TARGET'] + prev_merged.dtypes[prev_merged.dtypes == 'object'].index.t
plot_phik_matrix(prev_merged, cols_for_phik, cmap = 'Blues', figsize = (11,9), fontsize
```



Categorise with highest values of Phi-K correlation value with target variable are:

	Column Name	Phik-Correlation
4	NAME_CONTRACT_STATUS	0.088266
15	PRODUCT_COMBINATION	0.063839
6	CODE_REJECT_REASON	0.062771
0	NAME_CONTRACT_TYPE	0.050859
12	CHANNEL_TYPE	0.050302
9	NAME_GOODS_CATEGORY	0.042951
3	NAME_CASH_LOAN_PURPOSE	0.040305
5	NAME_PAYMENT_TYPE	0.039752
13	NAME_SELLER_INDUSTRY	0.038077

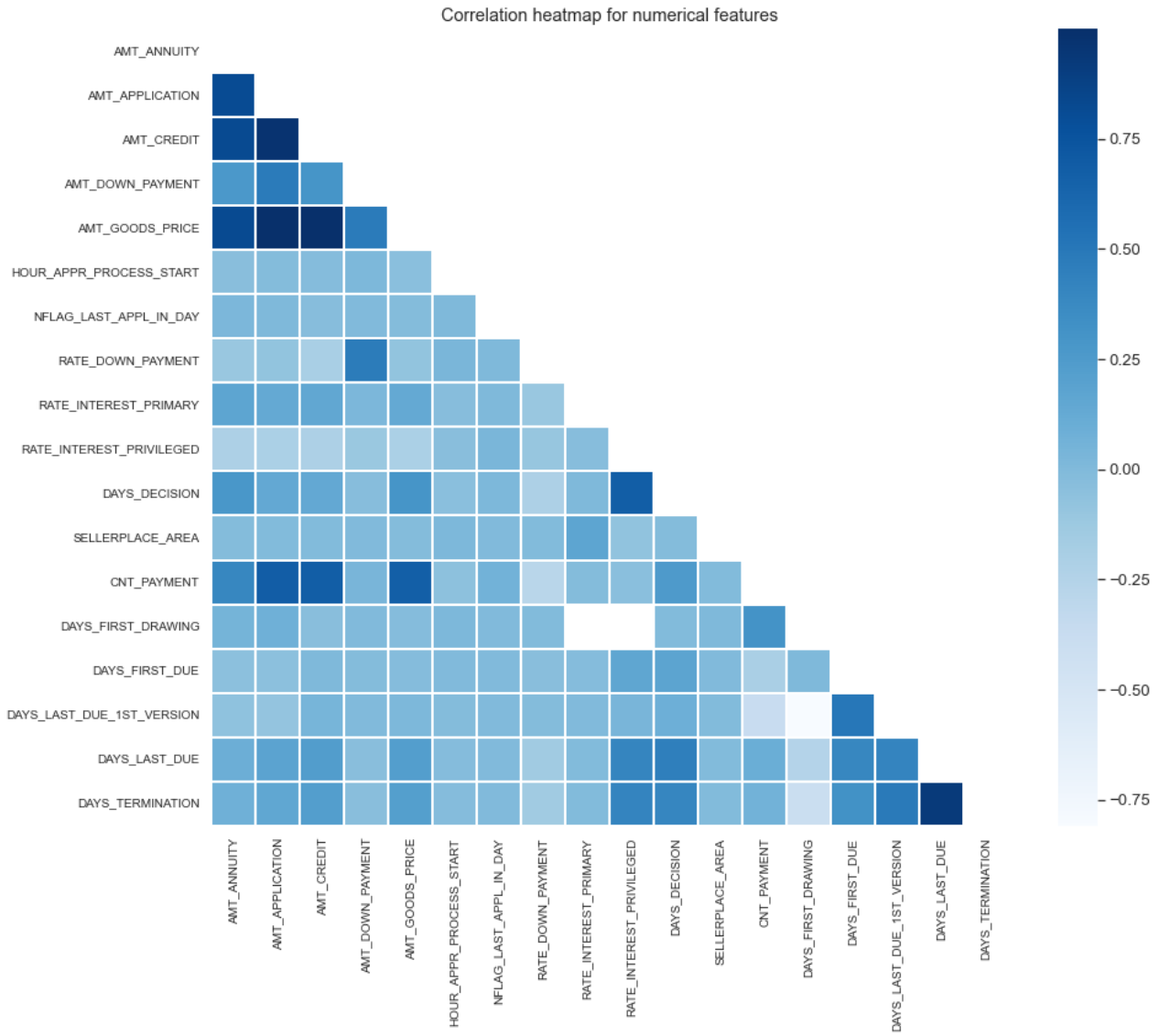
	Column Name	Phik-Correlation
14	NAME_YIELD_GROUP	0.034626

Observations and conclusions:

1. The feature PRODUCT_COMBINATION shows association with lots of other features such as NAME_CONTRACT_TYPE, NAME_PRODUCT_TYPE, NAME_PORTFOLIO, etc.
2. The feature NAME_GOODS_CATEGORY is also highly associated with NAME_SELLER_INDUSTRY
3. If we look at the association with TARGET variable, we see that the features NAME_CONTRACT_STATUS, PRODUCT_COMBINATION, CODE_REJECT_REASON are some of the highest associated features, and would need further investigation

2.4.5 Correlation Matrix of Features

In [72]:
```
corr_mat = correlation_matrix(prev_merged, ['SK_ID_CURR','SK_ID_PREV',
                                            'NFLAG_INSURED_ON_APPROVAL'],
                              cmap = 'Blues', figsize = (14,12))
corr_mat.plot_correlation_matrix()
```



In [73]:

```
#Seeing the top columns with highest phik-correlation with the target variable in previ
top_corr_target_df = corr_mat.target_top_corr()
print("-" * 100)
print("Columns with highest values of Phik-correlation with Target Variable are:")
display(top_corr_target_df)
print("-"*100)
```

```
interval columns not set, guessing: ['TARGET', 'AMT_ANNUITY']
interval columns not set, guessing: ['TARGET', 'AMT_APPLICATION']
interval columns not set, guessing: ['TARGET', 'AMT_CREDIT']
interval columns not set, guessing: ['TARGET', 'AMT_DOWN_PAYMENT']
interval columns not set, guessing: ['TARGET', 'AMT_GOODS_PRICE']
interval columns not set, guessing: ['TARGET', 'HOUR_APPR_PROCESS_START']
interval columns not set, guessing: ['TARGET', 'NFLAG_LAST_APPL_IN_DAY']
interval columns not set, guessing: ['TARGET', 'RATE_DOWN_PAYMENT']
interval columns not set, guessing: ['TARGET', 'RATE_INTEREST_PRIMARY']
interval columns not set, guessing: ['TARGET', 'RATE_INTEREST_PRIVILEGED']
interval columns not set, guessing: ['TARGET', 'DAYS_DECISION']
interval columns not set, guessing: ['TARGET', 'SELLERPLACE_AREA']
interval columns not set, guessing: ['TARGET', 'CNT_PAYMENT']
interval columns not set, guessing: ['TARGET', 'DAYS_FIRST_DRAWING']
```

```
interval columns not set, guessing: ['TARGET', 'DAYS_FIRST_DUE']
interval columns not set, guessing: ['TARGET', 'DAYS_LAST_DUE_1ST_VERSION']
interval columns not set, guessing: ['TARGET', 'DAYS_LAST_DUE']
interval columns not set, guessing: ['TARGET', 'DAYS_TERMINATION']
--------------------------------------------------------------------------------
------------
Columns with highest values of Phik-correlation with Target Variable are:
```

	Column Name	Phik-Correlation
12	CNT_PAYMENT	0.056639
10	DAYS_DECISION	0.053694
13	DAYS_FIRST_DRAWING	0.048993
7	RATE_DOWN_PAYMENT	0.039592
5	HOUR_APPR_PROCESS_START	0.038121
9	RATE_INTEREST_PRIVILEGED	0.028204
15	DAYS_LAST_DUE_1ST_VERSION	0.027878
16	DAYS_LAST_DUE	0.027320
17	DAYS_TERMINATION	0.026479
0	AMT_ANNUITY	0.013808

```
--------------------------------------------------------------------------------
------------
```

Observations and conclusions:

1. The high correlation is particularly observed for features:
   - DAYS_TERMINATION and DAYS_LAST_DUE
   - AMT_CREDIT and AMT_APPLICATION
   - AMT_APPLICATION and AMT_GOODS_PRICE
   - AMT_CREDIT and AMT_ANNUITY
   - AMT_ANNUITY and AMT_CREDIT
   - AMT_CREDIT and AMT_GOODS_PRICE
2. We can also see that the features don't particularly show good/high correlation with Target as such. This implies that there isn't much of a direct linear relation between Target and the features.

2.4.6 Plotting Distribution of Categorical Variables

Let us now plot some of the Categorical Variables of table previous_application and see how they impact the Target Variable.

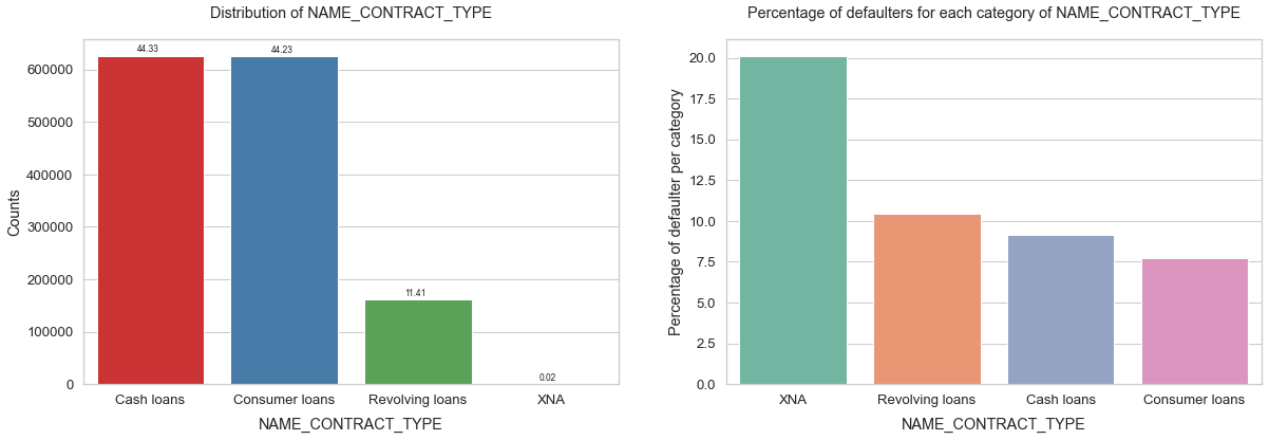
2.4.6.1 NAME_CONTRACT_TYPE

This column descibes the type of the Contract of the previous loan with the Home Credit Group.

In [74]:
```
#Let us first see the unique categories of 'NAME_CONTRACT_TYPE'
print_unique_categories(prev_merged, 'NAME_CONTRACT_TYPE', show_counts = True)
```

```
# plotting the Bar Plot for the Column
plot_categorical_variables_bar(prev_merged, 'NAME_CONTRACT_TYPE', horizontal_adjust = 0
                               figsize = (20, 6))
print('-'*100)
```

```
----------------------------------------------------------------------------------------
------------
The unique categories of 'NAME_CONTRACT_TYPE' are:
['Consumer loans' 'Cash loans' 'Revolving loans' nan 'XNA']
----------------------------------------------------------------------------------------
------------
Counts of each category are:
Cash loans         626764
Consumer loans     625256
Revolving loans    161368
XNA                   313
Name: NAME_CONTRACT_TYPE, dtype: int64
----------------------------------------------------------------------------------------
------------
Total number of unique categories of NAME_CONTRACT_TYPE = 5
```




```
----------------------------------------------------------------------------------------
------------
```

Observations and conclusions:

1. From the first subplot, we see that most of the previous loans have been either Cash Loans or Consumer Loans, which correspond to roughly 44% of loans each. The remaining 11.41% corresponds to Revolving Loans, and there are some loans named XNA whose types are actually not known, but they are very few in numbers.
2. Looking at the second subplot, we see that the Percentage of Defaulters for XNA type of loan are the highest, at 20% Default rate. The next highest Default Rate is among Revolving Loans, which is close to 10.5%.
3. The Cash Loans have lesser default rates, roughly 9% while the consumer loans tend to have the lowest Percentage of Defaulters, which is close to 7.5%.

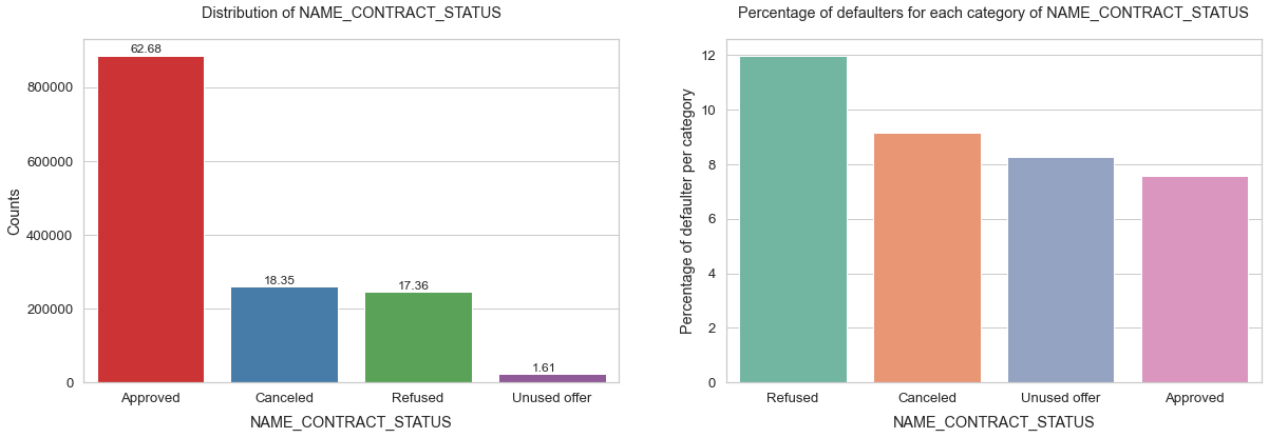
2.4.6.2 NAME_CONTRACT_STATUS

This column describes the status of the contract of the previous loan with Home Credit, i.e. whether it is active or closed, etc.

In [75]:

```
#let us first see the unique categories of 'NAME_CONTRACT_STATUS'
print_unique_categories(prev_merged, 'NAME_CONTRACT_STATUS')

# plotting the Bar Plot for the Column
plot_categorical_variables_bar(prev_merged, 'NAME_CONTRACT_STATUS', horizontal_adjust =
                               figsize = (20, 6), fontsize_percent = 'small')
print('-'*100)
```

```
----------------------------------------------------------------------------------------
------------
The unique categories of 'NAME_CONTRACT_STATUS' are:
['Approved' 'Canceled' 'Refused' nan 'Unused offer']
----------------------------------------------------------------------------------------
------------
Total number of unique categories of NAME_CONTRACT_STATUS = 5
```




```
----------------------------------------------------------------------------------------
------------
```

Observations and conclusions:

1. The most common type of Contract Status is the Approved Status. About 63% of the previous Credits have an Approved Status. The next two common status are Canceled and Refused, which both correspond to about 18% of the loans. This implies that most of the loans get approved and only some fraction of them do not. The least occurring type of contract status is Unused Offer which corresponds to just 1.61% of all the loans.
2. Looking at the second subplot for percentage of defaulters, we see that the those loans which previously had Refused Status tend to have defaulted the highest in the current loans. They correspond to about 12% of Defaulters from that category. These are followed by Canceled Status which correspond to close to 9% of Default Rate. This behavious is quite expected logically, as these people must have been refused due to not having adequate profile. The least default rate is observed for Contract Status of Approved.

2.4.6.3 CODE_REJECT_REASON

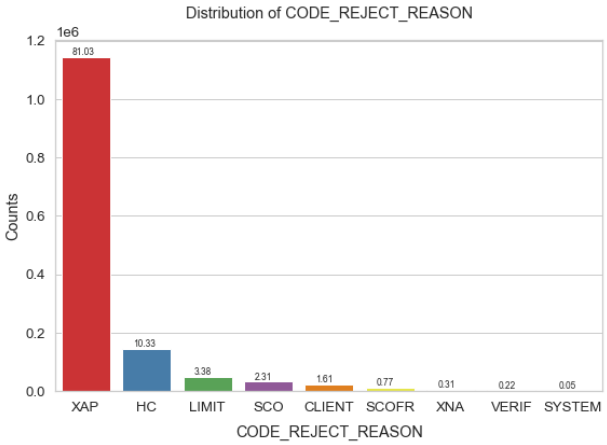
This column describes the reason of the rejection of previously applied loan in Home Credit Group.

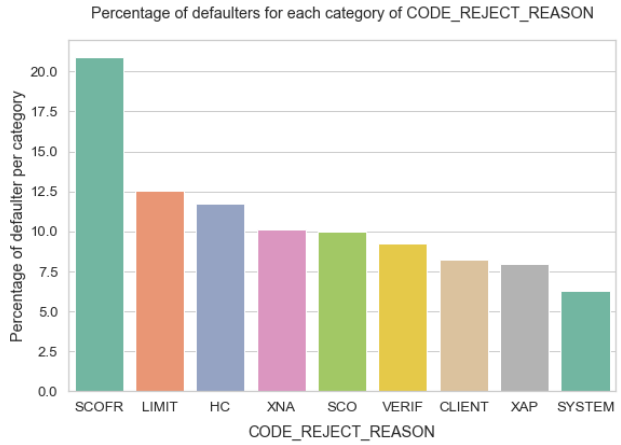
In [76]:

```
#let us first see the unique categories of 'CODE_REJECT_REASON'
print_unique_categories(prev_merged, 'CODE_REJECT_REASON', show_counts = True)

# plotting the Bar Plot for the Column
plot_categorical_variables_bar(prev_merged, 'CODE_REJECT_REASON', horizontal_adjust = 0
```

```
                                    figsize = (20, 6))
print('-'*100)
```

```
----------------------------------------------------------------------------------------------------
The unique categories of 'CODE_REJECT_REASON' are:
['XAP' 'LIMIT' nan 'HC' 'SCO' 'SCOFR' 'VERIF' 'CLIENT' 'XNA' 'SYSTEM']
----------------------------------------------------------------------------------------------------
Counts of each category are:
XAP       1145533
HC         145984
LIMIT       47773
SCO         32636
CLIENT      22771
SCOFR       10875
XNA          4378
VERIF        3079
SYSTEM        672
Name: CODE_REJECT_REASON, dtype: int64
----------------------------------------------------------------------------------------------------
Total number of unique categories of CODE_REJECT_REASON = 10
```






```
----------------------------------------------------------------------------------------------------
```

Observations and conclusions:

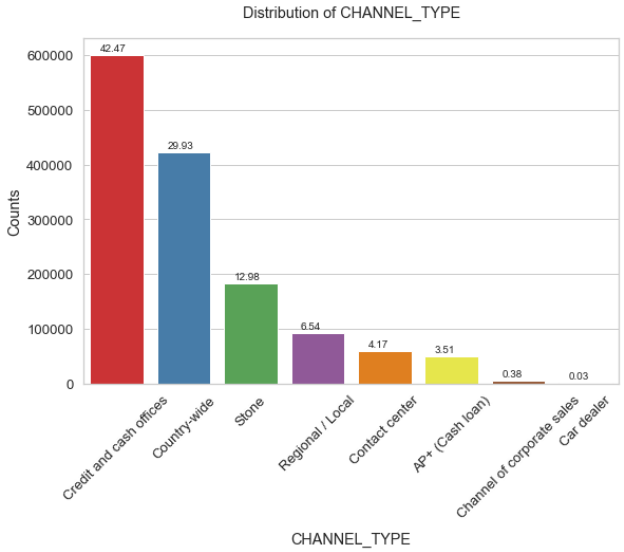
1. The most common type of reason of rejection is XAP, which is about ~81%. The other reasons form only a small part of the rejection reasons. HC is the second highest rejection reason with just 10.33% of occurrences.
2. The distribution of percentage of defaulters for each category of CODE_REJECT_REASON in quite interesting. Those applicants who had their previous applications rejected by Code SCOFT have the highest percentage of Defaulters among them (~21%). This is followed by LIMIT and HC which have around 12.5% and 12% of Defaulters.
3. The most common occurring rejection reason XAP corresponds to only 7.5% of Defaulters of all, and is the second lowest percentage of Defaulters after SYSTEM code.

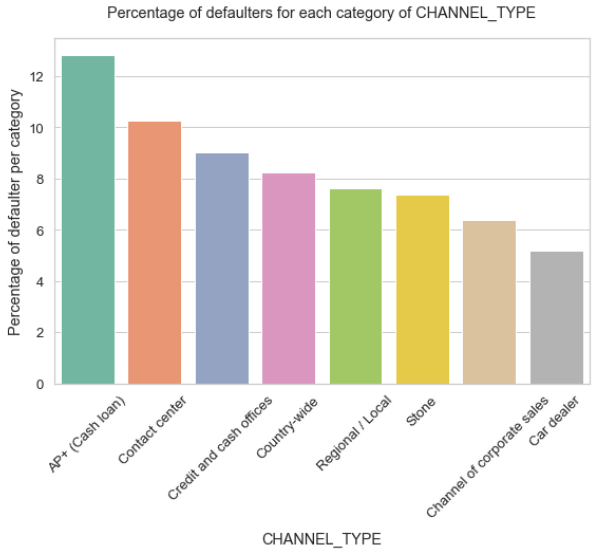
2.4.6.4 CHANNEL_TYPE

This column describes the channel through which the client was acquired for the previous loan in Home Credit.

In [77]:
```
#Let us first see the unique categories of 'CHANNEL_TYPE'
print_unique_categories(prev_merged, 'CHANNEL_TYPE')

# plotting the Bar Plot for the Column
plot_categorical_variables_bar(prev_merged, 'CHANNEL_TYPE', horizontal_adjust = 0.15,
                               rotation = 45, figsize = (20, 6), fontsize_percent = 'x-
print('-'*100)
```

```
----------------------------------------------------------------------------------------
-----------
The unique categories of 'CHANNEL_TYPE' are:
['Stone' 'Credit and cash offices' 'Country-wide' 'Regional / Local'
 'AP+ (Cash loan)' 'Contact center' nan 'Channel of corporate sales'
 'Car dealer']
----------------------------------------------------------------------------------------
-----------
Total number of unique categories of CHANNEL_TYPE = 9
```






```
----------------------------------------------------------------------------------------
-----------
```

Observations and conclusions:

1. From the first subplot we see that most of the applications were acquired through the Credit and cash offices which were roughly 42.47% applications, which were followed by Country-wide channel corresponding to 29.93% applications. Rest of the channel types corresponded to only a select number of applications.
2. The highest Defaulting Percentage was seen among applications who had a channel type of AP+ (Cash loan) which corresponded to about 13% defaulters in that category. The rest of the channels had lower default percentages than this one. The channel Car Dealer showed a lowest Percentage of Defaulters in that category (only 5%).

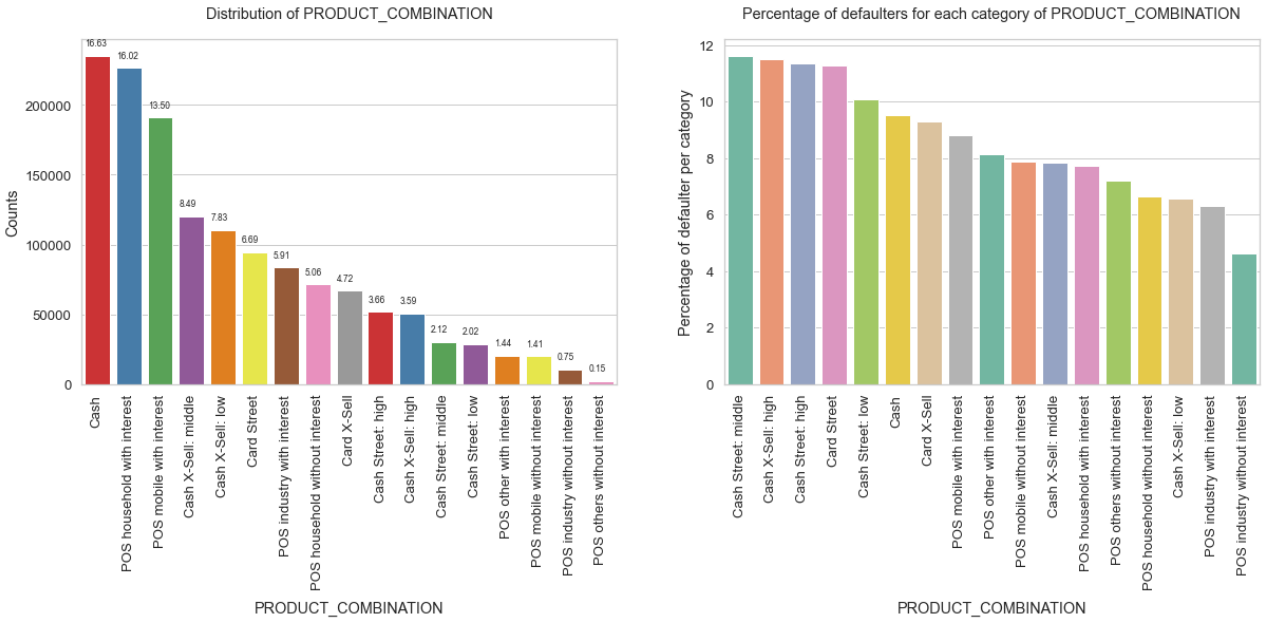
2.4.6.5 PRODUCT_COMBINATION

This column gives details about the product combination of the previous applications.

In [78]:
```
#Let us first see the unique categories of 'PRODUCT_COMBINATION'
print_unique_categories(prev_merged, 'PRODUCT_COMBINATION')
```

```
# plotting the Bar Plot for the Column
plot_categorical_variables_bar(prev_merged, 'PRODUCT_COMBINATION', rotation = 90,
                               figsize = (20, 6))
print('-'*100)
```

```
----------------------------------------------------------------------------------------------------
------------
The unique categories of 'PRODUCT_COMBINATION' are:
['POS other with interest' 'Cash X-Sell: low' 'POS industry with interest'
 'POS household with interest' 'POS mobile without interest' 'Card Street'
 'Card X-Sell' 'Cash X-Sell: high' 'Cash' 'Cash Street: high'
 'Cash X-Sell: middle' 'POS mobile with interest'
 'POS household without interest' 'POS industry without interest'
 'Cash Street: low' nan 'Cash Street: middle'
 'POS others without interest']
----------------------------------------------------------------------------------------------------
------------
Total number of unique categories of PRODUCT_COMBINATION = 18
```




```
----------------------------------------------------------------------------------------------------
------------
```

Observations and conclusions:

1. The 3 most common types of Product Combination are Cash, POS household with interest and POS mobile with interest. They correspond to roughly 50% of all the applications.
2. Looking at the Percentage of Defaulters per category plot, we see a highest defaulting tendency among Cash Street: mobile category, Cash X-sell: high, Cash Street: high and Card Street which all are near about 11-11.5% defaulters per category. The lowest Percentage of Defaulters are in the POS Industry without interest Category, which correspond to about 4.5% Defaulters.

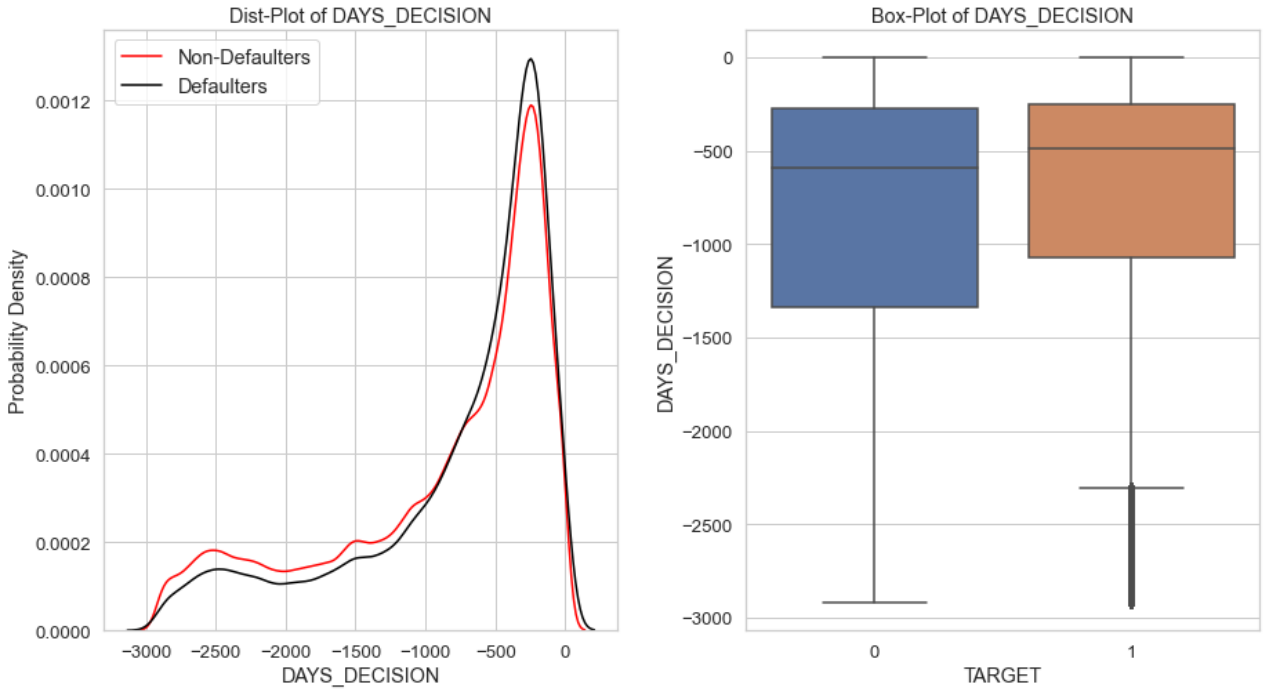
2.4.7 Plotting Distribution of Continuous Variables

2.4.7.1 DAYS Features

DAYS_DECISION

This column tells about the number of days relative to the current application when the decision was made about previous application.

In [79]:
```
plot_continuous_variables(prev_merged, 'DAYS_DECISION', plots = ['distplot', 'box'], fi
```



Observations and conclusions:

From the above plot, we notice that for Defaulters, the number of days back when the decision was made is a bit lesser than that for Non-Defaulters. This implies that the Defaulters usually had the decision on their previous applications made more recently as compared to Non-Defaulters.

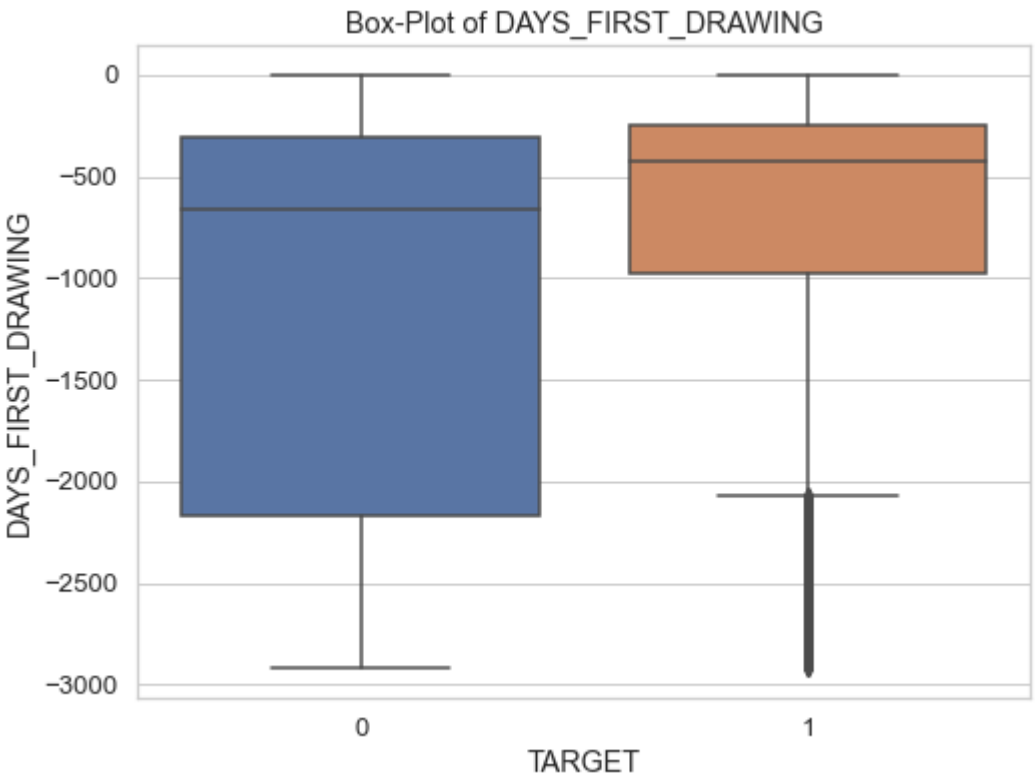
DAYS_FIRST_DRAWING

This column tells about the number of days back from current application that the first disbursement of the previous application was made.

In [80]:
```
print_percentiles(prev_merged, 'DAYS_FIRST_DRAWING', percentiles = list(range(0,11)) +
                  list(range(20,101,20)))
plot_continuous_variables(prev_merged, 'DAYS_FIRST_DRAWING', plots = ['box'], figsize =
                          scale_limits = [-3000,0])
print('-'*100)
```

```
----------------------------------------------------------------------------------------------------
------------
The 0th percentile value of DAYS_FIRST_DRAWING is -2922.0
The 1th percentile value of DAYS_FIRST_DRAWING is -2451.0
The 2th percentile value of DAYS_FIRST_DRAWING is -1179.0
The 3th percentile value of DAYS_FIRST_DRAWING is -674.0
The 4th percentile value of DAYS_FIRST_DRAWING is -406.0
The 5th percentile value of DAYS_FIRST_DRAWING is -262.0
The 6th percentile value of DAYS_FIRST_DRAWING is -156.0
The 7th percentile value of DAYS_FIRST_DRAWING is 365243.0
The 8th percentile value of DAYS_FIRST_DRAWING is 365243.0
The 9th percentile value of DAYS_FIRST_DRAWING is 365243.0
```

```
The 10th percentile value of DAYS_FIRST_DRAWING is 365243.0
The 20th percentile value of DAYS_FIRST_DRAWING is 365243.0
The 40th percentile value of DAYS_FIRST_DRAWING is 365243.0
The 60th percentile value of DAYS_FIRST_DRAWING is 365243.0
The 80th percentile value of DAYS_FIRST_DRAWING is 365243.0
The 100th percentile value of DAYS_FIRST_DRAWING is 365243.0
--------------------------------------------------------------------------------
------------
```




```
--------------------------------------------------------------------------------
------------
```

Observations and conclusions:

1. Looking at the percentile values of DAYS_FIRST_DRAWING, it seems like most of the values are erroneous, starting from 7th percentile values itself. These erroneous values will needed to be dopped.
2. If we try to analyze the distribution of this column by removing the erroneous ponts, we see that most of the Defaulters had their First Drawing on previous credit more recently as compared to Non-Defaulters. The 75th percentile value for Defaulters is also significantly lesser than that of Non-Defaulters.

DAYS_FIRST_DUE , DAYS_LAST_DUE_1ST_VERSION , DAYS_LAST_DUE , and DAYS_TERMINATION

These columns also decribe about the number of days ago from the current application that certain activities happened.

In [81]:

```
print('-'*100)
print("Percentile Values for DAYS_FIRST_DUE")
print_percentiles(prev_merged, 'DAYS_FIRST_DUE', percentiles = list(range(0,11,2)) +
                  [20,40,60,80,100])
print("Percentile Values for DAYS_LAST_DUE_1ST_VERSION")
print_percentiles(prev_merged, 'DAYS_LAST_DUE_1ST_VERSION', percentiles = list(range(0,
```

```
                    [20,40,60,80,100])
print("Percentile Values for DAYS_LAST_DUE")
print_percentiles(prev_merged, 'DAYS_LAST_DUE', percentiles = list(range(0,11,2)) +
                    [20,40,60,80,100])
print("Percentile Values for DAYS_TERMINATION")
print_percentiles(prev_merged, 'DAYS_TERMINATION', percentiles = list(range(0,11,2)) +
                    [20,40,60,80,100])
```

```
------------------------------------------------------------------------------------------
------------
Percentile Values for DAYS_FIRST_DUE
------------------------------------------------------------------------------------------
------------
The 0th percentile value of DAYS_FIRST_DUE is -2892.0
The 2th percentile value of DAYS_FIRST_DUE is -2759.0
The 4th percentile value of DAYS_FIRST_DUE is -2648.0
The 6th percentile value of DAYS_FIRST_DUE is -2555.0
The 8th percentile value of DAYS_FIRST_DUE is -2471.0
The 10th percentile value of DAYS_FIRST_DUE is -2388.0
The 20th percentile value of DAYS_FIRST_DUE is -1882.0
The 40th percentile value of DAYS_FIRST_DUE is -1070.0
The 60th percentile value of DAYS_FIRST_DUE is -647.0
The 80th percentile value of DAYS_FIRST_DUE is -329.0
The 100th percentile value of DAYS_FIRST_DUE is 365243.0
------------------------------------------------------------------------------------------
------------
Percentile Values for DAYS_LAST_DUE_1ST_VERSION
------------------------------------------------------------------------------------------
------------
The 0th percentile value of DAYS_LAST_DUE_1ST_VERSION is -2801.0
The 2th percentile value of DAYS_LAST_DUE_1ST_VERSION is -2516.0
The 4th percentile value of DAYS_LAST_DUE_1ST_VERSION is -2380.0
The 6th percentile value of DAYS_LAST_DUE_1ST_VERSION is -2267.0
The 8th percentile value of DAYS_LAST_DUE_1ST_VERSION is -2159.0
The 10th percentile value of DAYS_LAST_DUE_1ST_VERSION is -2045.0
The 20th percentile value of DAYS_LAST_DUE_1ST_VERSION is -1498.0
The 40th percentile value of DAYS_LAST_DUE_1ST_VERSION is -644.0
The 60th percentile value of DAYS_LAST_DUE_1ST_VERSION is -146.0
The 80th percentile value of DAYS_LAST_DUE_1ST_VERSION is 273.0
The 100th percentile value of DAYS_LAST_DUE_1ST_VERSION is 365243.0
------------------------------------------------------------------------------------------
------------
Percentile Values for DAYS_LAST_DUE
------------------------------------------------------------------------------------------
------------
The 0th percentile value of DAYS_LAST_DUE is -2889.0
The 2th percentile value of DAYS_LAST_DUE is -2534.0
The 4th percentile value of DAYS_LAST_DUE is -2400.0
The 6th percentile value of DAYS_LAST_DUE is -2290.0
The 8th percentile value of DAYS_LAST_DUE is -2186.0
The 10th percentile value of DAYS_LAST_DUE is -2079.0
The 20th percentile value of DAYS_LAST_DUE is -1554.0
The 40th percentile value of DAYS_LAST_DUE is -784.0
The 60th percentile value of DAYS_LAST_DUE is -333.0
The 80th percentile value of DAYS_LAST_DUE is 365243.0
The 100th percentile value of DAYS_LAST_DUE is 365243.0
------------------------------------------------------------------------------------------
------------
Percentile Values for DAYS_TERMINATION
------------------------------------------------------------------------------------------
```

```
------------
The 0th percentile value of DAYS_TERMINATION is -2874.0
The 2th percentile value of DAYS_TERMINATION is -2523.0
The 4th percentile value of DAYS_TERMINATION is -2383.0
The 6th percentile value of DAYS_TERMINATION is -2270.0
The 8th percentile value of DAYS_TERMINATION is -2161.0
The 10th percentile value of DAYS_TERMINATION is -2048.0
The 20th percentile value of DAYS_TERMINATION is -1508.0
The 40th percentile value of DAYS_TERMINATION is -743.0
The 60th percentile value of DAYS_TERMINATION is -297.0
The 80th percentile value of DAYS_TERMINATION is 365243.0
The 100th percentile value of DAYS_TERMINATION is 365243.0
------------------------------------------------------------------------------------------
------------
```

Observations and conclusions:

From all of the above percentile values, we realise that all the Days columns have these erroneous values somewhere or the other. Thus these values need to be replaced so that our model doesn't get affected by these.

2.5 installments_payments.csv

Description

This table lists out the repayment history of each of the loan that the applicant had with Home Credit Group. The table contains features like the amount of instalment, how much did the client pay for each instalments, etc.

2.5.1 Basic Stats

In [82]:

```
print(f'The shape of installments_payments.csv is: {install_pay.shape}')
print('-'*100)
print(f'Number of unique SK_ID_PREV in installments_payments.csv are:
      {len(install_pay.SK_ID_PREV.unique())}')
print(f'Number of unique SK_ID_CURR in installments_payments.csv are:
      {len(install_pay.SK_ID_CURR.unique())}')
print('-'*100)
print(f'Number of overlapping SK_ID_CURR in application_train.csv and installments_paym
      {len(set(app_train.SK_ID_CURR.unique()).intersection(set(install_pay.SK_ID_CURR.u
print(f'Number of overlapping SK_ID_CURR in application_test.csv and installments_payme
      {len(set(app_test.SK_ID_CURR.unique()).intersection(set(install_pay.SK_ID_CURR.un
print('-'*100)
print(f'Number of duplicate values in installments_payments:
      {install_pay.shape[0] - install_pay.duplicated().shape[0]}')
print('-'*100)
display(install_pay.head(5))
```

```
The shape of installments_payments.csv is: (13605401, 8)
------------------------------------------------------------------------------------------
------------
Number of unique SK_ID_PREV in installments_payments.csv are: 997752
Number of unique SK_ID_CURR in installments_payments.csv are: 339587
------------------------------------------------------------------------------------------
------------
```

```
Number of overlapping SK_ID_CURR in application_train.csv and installments_payments.csv
are: 291643
Number of overlapping SK_ID_CURR in application_test.csv and installments_payments.csv a
re: 47944
-------------------------------------------------------------------------------------
------------
Number of duplicate values in installments_payments: 0
-------------------------------------------------------------------------------------
------------
```

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER	DAYS_INSTALME
0	1054186	161674	1.0	6	-11
1	1330831	151639	0.0	34	-21
2	2085231	193053	2.0	1	-
3	2452527	199697	1.0	3	-24
4	2714724	167756	1.0	2	-13

Observations and conclusions:

1. There are about 13.6M datapoints in the table installments_payments.csv. Each row represents each installment history related to the a particular loan that the client previously had with Home Credit Group.
2. There are 997k unique previous loans in the installments_payments. These belong to 339k unique SK_ID_CURR, which are ID of applicants of current loan.
3. Out of these 339k SK_ID_CURR, 291k belong to the training dataset, and 47.9k belong to the test dataset. This implies that almost out of 307k unique SK_ID_CURR in application_train, 291k previously had some form of loan with Home Credit. Similarly for 48.7k of those in test dataset, 47.9k had loan previously with Home Credit.
4. The table has 8 unique features, 6 of which describe the statistics of each installment for previous loan.

2.5.2 NaN Columns and Percentages

In [83]:

```
print('-'*100)
print("Columns with NaN values and their percentages:")
installments_payments_nan = nan_df_create(install_pay)
display(installments_payments_nan[installments_payments_nan.percent != 0])
print('-'*100)
del installments_payments_nan
```

```
-------------------------------------------------------------------------------------
------------
Columns with NaN values and their percentages:
```

	column	percent
5	DAYS_ENTRY_PAYMENT	0.021352
7	AMT_PAYMENT	0.021352

```
-------------------------------------------------------------------------------------
```

```
------------
```

Observations and conclusions:

1. There are only 2 columns which contain NaN values of the 8 columns from installments_payments.
2. These columns also contain very minimal proportion of NaN values, i.e only 0.02%, so it is not of much concern.

2.5.3 Merging the TARGETS from application_train to installments_payments table

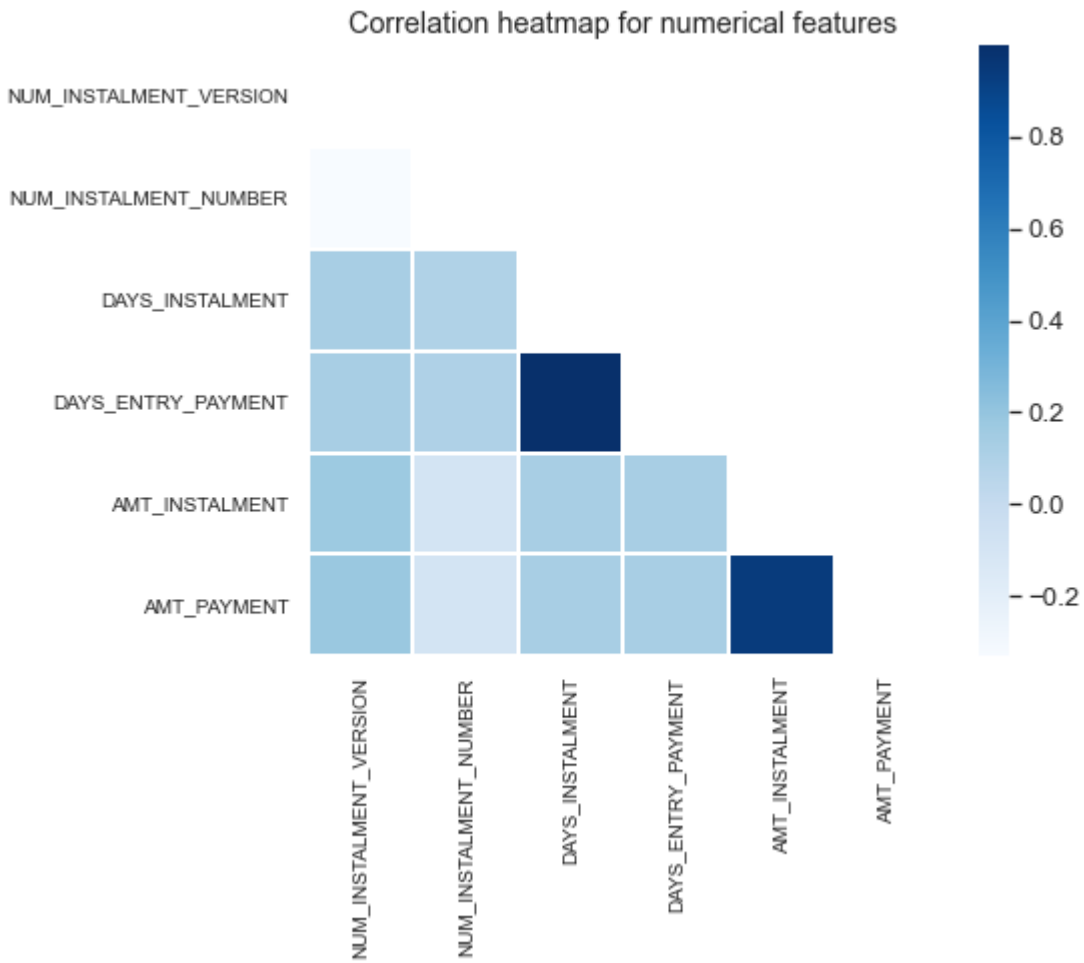
In [84]:
```
print("-"*100)
print("Merging TARGET with installments_payments Table")
installments_merged = app_train.iloc[:,:2].merge(install_pay, on = 'SK_ID_CURR', how =
print("-"*100)
```

```
----------------------------------------------------------------------------------------------------
------------
Merging TARGET with installments_payments Table
----------------------------------------------------------------------------------------------------
------------
```

2.5.4 Correlation Matrix of Features

In [85]:
```
corr_mat = correlation_matrix(installments_merged, ['SK_ID_CURR','SK_ID_PREV'], figsize
corr_mat.plot_correlation_matrix()
```

```
----------------------------------------------------------------------------------------------------
------------
```



In [86]:
```
#Seeing the top columns with highest phik-correlation with the target variable in insta
top_corr_target_df = corr_mat.target_top_corr()
print("-" * 100)
print("Columns with highest values of Phik-correlation with Target Variable are:")
display(top_corr_target_df)
print("-"*100)
```

```
interval columns not set, guessing: ['TARGET', 'NUM_INSTALMENT_VERSION']
interval columns not set, guessing: ['TARGET', 'NUM_INSTALMENT_NUMBER']
interval columns not set, guessing: ['TARGET', 'DAYS_INSTALMENT']
interval columns not set, guessing: ['TARGET', 'DAYS_ENTRY_PAYMENT']
interval columns not set, guessing: ['TARGET', 'AMT_INSTALMENT']
interval columns not set, guessing: ['TARGET', 'AMT_PAYMENT']
----------------------------------------------------------------------------------------------------
Columns with highest values of Phik-correlation with Target Variable are:
```

	Column Name	Phik-Correlation
2	DAYS_INSTALMENT	0.046824
3	DAYS_ENTRY_PAYMENT	0.033128
1	NUM_INSTALMENT_NUMBER	0.022993
4	AMT_INSTALMENT	0.004125
5	AMT_PAYMENT	0.003084

	Column Name	Phik-Correlation
0	NUM_INSTALMENT_VERSION	0.002198

Observations and conclusions:

1. From the heatmap of correlation matrix, we see a couple of highly correlated features. These are:
   - AMT_INSTALMENT and AMT_PAYMENT
   - DAYS_INSTALMENT and DAYS_ENTRY_PAYMENT
2. These two sets of correlated features are understandable, as they are actually the features as to when the installment was due to be paid vs when it was paid and also the amount that was due vs the amount that was paid.
3. These features will be useful for creating new sets of completely uncorrelated features.
4. The correlation of features with Target isn't noticeable, this shows the absence of a linear relationship between the feature and the target variable.

2.5.5 Plotting Distribution of Continuous Variables

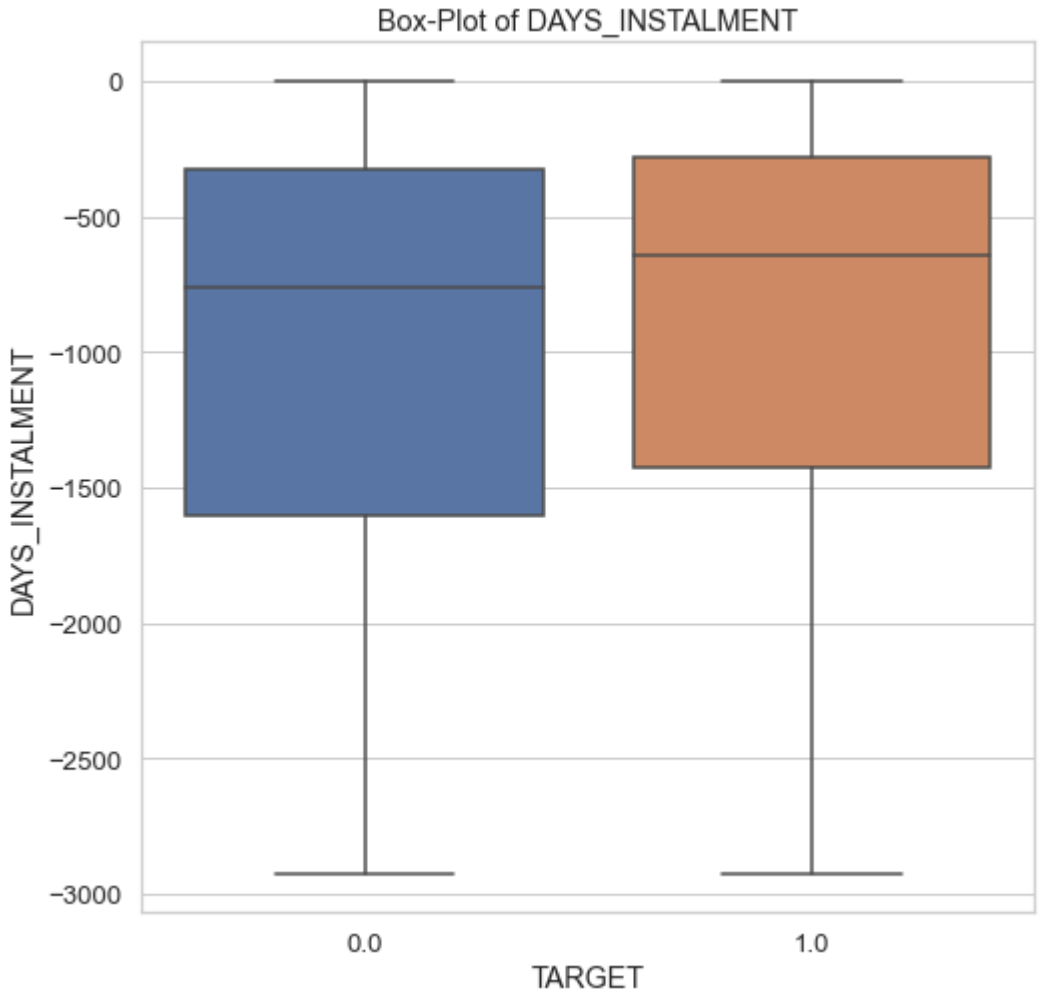
Firstly we will group by the 'SK_ID_PREV' field and aggregate with mean, so that we get an averaged row for each of the previous loan that the client had.

In [87]:
```
installments_merged = installments_merged.groupby('SK_ID_PREV').mean()
```

2.5.5.1 DAYS_INSTALMENT

This column lists the days when the installment of previous credit was to be paid.

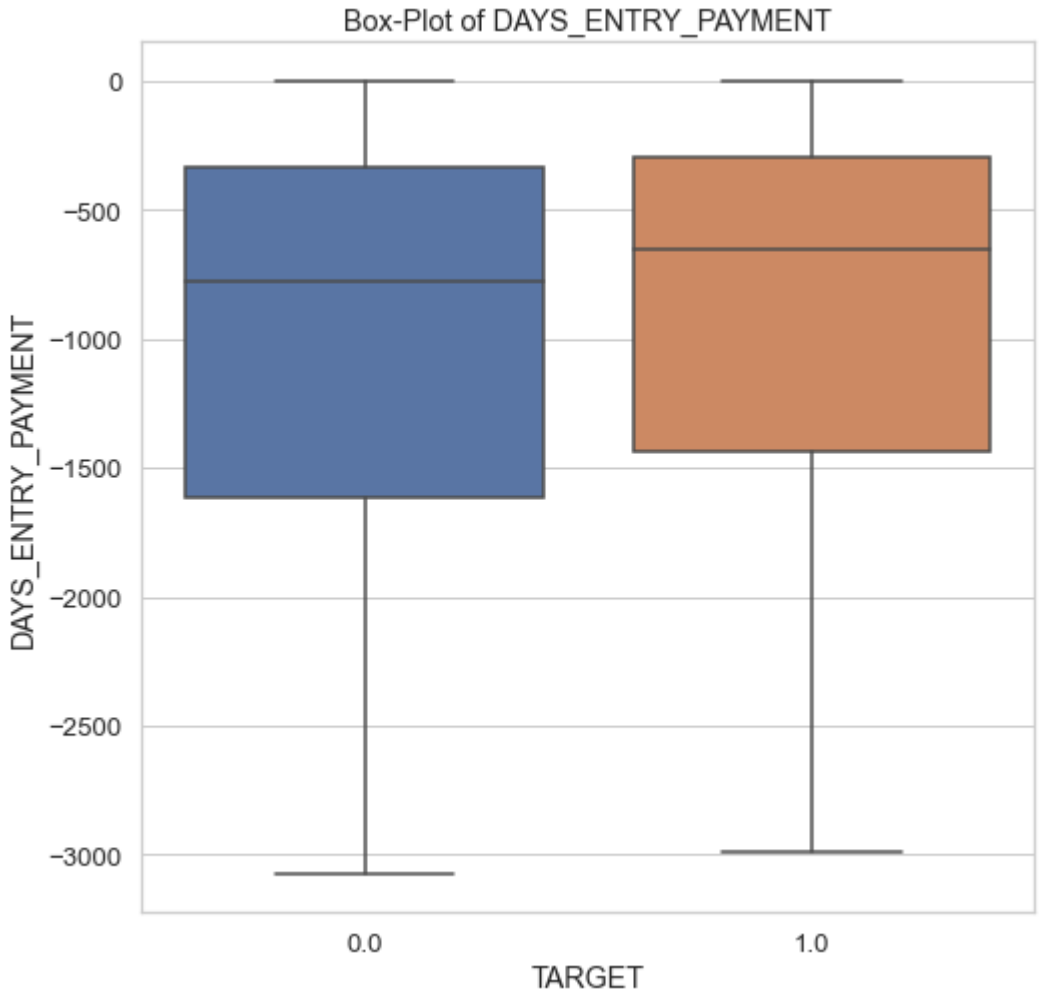
In [89]:
```
plot_continuous_variables(installments_merged, 'DAYS_INSTALMENT', plots = ['box'],
                          figsize = (8,8))
```



2.5.5.2 DAYS_ENTRY_PAYMENT

This column lists the days when the installment of previous credit was actually paid.

In [90]:
```
plot_continuous_variables(installments_merged, 'DAYS_ENTRY_PAYMENT', plots = ['box'],
                          figsize = (8,8))
del installments_merged
```



Observations and conclusions:

From the above two plots, we can see a similar pattern, where the Defaulters tend to have lesser number of days since their last payment, while Non-Defaulters have more number of days since their last payments. All quantiles of Defaulters have more recent days than those of Non-Defaulters. Thus, Non-Defaulters usually have more gap in their payments from the day of application as compared to Defaulters.

2.6 POS_CASH_balance.csv

Description

This table contains the Monthly Balance Snapshots of previous Point of Sales and Cash Loans that the applicant had with Home Credit Group. The table contains columns like the status of contract, the number of installments left, etc.

2.6.1 Basic Stats

In [91]:
```
print(f'The shape of POS_CASH_balance.csv is: {POS_CASH_balance.shape}')
print('-'*100)
print(f'Number of unique SK_ID_PREV in POS_CASH_balance.csv are:
      {len(POS_CASH_balance.SK_ID_PREV.unique())}')
```

```
print(f'Number of unique SK_ID_CURR in POS_CASH_balance.csv are:
      {len(POS_CASH_balance.SK_ID_CURR.unique())}')
print('-'*100)
print(f'Number of overlapping SK_ID_CURR in application_train.csv and POS_CASH_balance.
      {len(set(app_train.SK_ID_CURR.unique()).intersection(set(POS_CASH_balance.SK_ID_C
print(f'Number of overlapping SK_ID_CURR in application_test.csv and POS_CASH_balance.c
      {len(set(app_test.SK_ID_CURR.unique()).intersection(set(POS_CASH_balance.SK_ID_CU
print('-'*100)
print(f'Number of duplicate values in POS_CASH_balance:
      {POS_CASH_balance.shape[0] - POS_CASH_balance.duplicated().shape[0]}')
print('-'*100)
display(POS_CASH_balance.head())
```

```
The shape of POS_CASH_balance.csv is: (10001358, 8)
----------------------------------------------------------------------------------------
------------
Number of unique SK_ID_PREV in POS_CASH_balance.csv are: 936325
Number of unique SK_ID_CURR in POS_CASH_balance.csv are: 337252
----------------------------------------------------------------------------------------
------------
Number of overlapping SK_ID_CURR in application_train.csv and POS_CASH_balance.csv are:
289444
Number of overlapping SK_ID_CURR in application_test.csv and POS_CASH_balance.csv are: 4
7808
----------------------------------------------------------------------------------------
------------
Number of duplicate values in POS_CASH_balance: 0
----------------------------------------------------------------------------------------
------------
```

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	CNT_INSTALMENT	CNT_INSTALMENT_FUTURE	NAME_
0	1803195	182943	-31	48.0	45.0	
1	1715348	367990	-33	36.0	35.0	
2	1784872	397406	-32	12.0	9.0	
3	1903291	269225	-35	48.0	42.0	
4	2341044	334279	-35	36.0	35.0	

Observations and conclusions:

1. This table contains around 10M datapoints, where each row corresponds to the monthly snapshot of the status of the previous POS and Cash Loan that the client had with Home Credit Group. It consists of 8 columns, two of which are SK_ID_CURR and SK_ID_PREV.
2. There are 936k unique previous loan IDs in the table, which correspond to 337k unique current applicants (SK_ID_CURR).
3. Out of these 337k SK_ID_CURR, 289k belong to training set and 47.8k belong to test set.

2.6.2 NaN Columns and Percentages

In [92]:

```
print('-'*100)
print("Columns with NaN values and their percentages:")
POS_CASH_nan = nan_df_create(POS_CASH_balance)
display(POS_CASH_nan[POS_CASH_nan.percent != 0])
```

```
print('-'*100)
del POS_CASH_nan
```

```
----------------------------------------------------------------------------------------------------
Columns with NaN values and their percentages:
```

	column	percent
4	CNT_INSTALMENT_FUTURE	0.260835
3	CNT_INSTALMENT	0.260675

```
----------------------------------------------------------------------------------------------------
```

Observations and conclusions:

1. There are only 2 columns which contain NaN values of the 8 columns from POS_CASH_balance. These columns are the Counts of Installments remaining and the term of the loan.
2. These columns also contain very minimal proportion of NaN values, i.e only 0.26%%, so it is also not of much concern.

2.6.3 Merging the TARGETS from application_train to POS_CASH_balance table

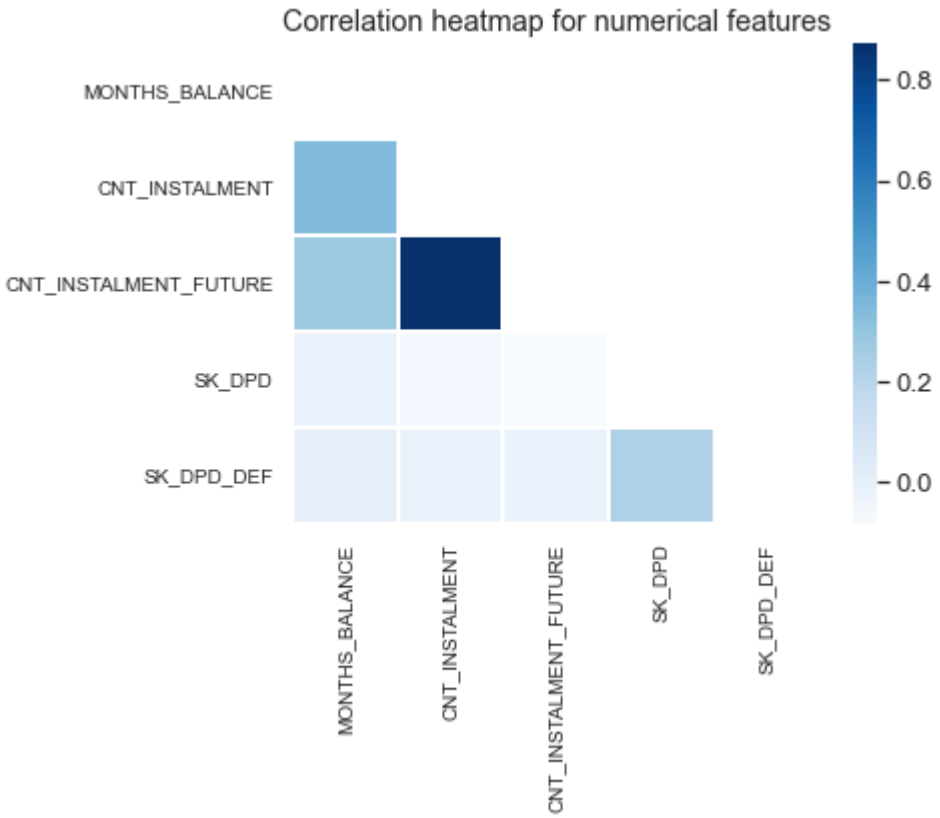
In [93]:
```
print("-"*100)
print("Merging TARGET with POS_CASH_balance Table")
pos_cash_merged = app_train.iloc[:,:2].merge(POS_CASH_balance, on = 'SK_ID_CURR', how =
print("-"*100)
```

```
----------------------------------------------------------------------------------------------------
Merging TARGET with POS_CASH_balance Table
----------------------------------------------------------------------------------------------------
```

2.6.4 Correlation Matrix of Features

In [94]:
```
corr_mat = correlation_matrix(pos_cash_merged, ['SK_ID_CURR','SK_ID_PREV'], figsize = (
corr_mat.plot_correlation_matrix()
```

```
----------------------------------------------------------------------------------------------------
```



In [95]:
```
#Seeing the top columns with highest phik-correlation with the target variable in POS_C
top_corr_target_df = corr_mat.target_top_corr()
print("-" * 100)
print("Columns with highest values of Phik-correlation with Target Variable are:")
display(top_corr_target_df)
print("-"*100)
```

```
interval columns not set, guessing: ['TARGET', 'MONTHS_BALANCE']
interval columns not set, guessing: ['TARGET', 'CNT_INSTALMENT']
interval columns not set, guessing: ['TARGET', 'CNT_INSTALMENT_FUTURE']
interval columns not set, guessing: ['TARGET', 'SK_DPD']
interval columns not set, guessing: ['TARGET', 'SK_DPD_DEF']
----------------------------------------------------------------------------------------------------
Columns with highest values of Phik-correlation with Target Variable are:
```

	Column Name	Phik-Correlation
2	CNT_INSTALMENT_FUTURE	0.033194
1	CNT_INSTALMENT	0.030947
0	MONTHS_BALANCE	0.027391
3	SK_DPD	0.012773
4	SK_DPD_DEF	0.010539

```
----------------------------------------------------------------------------------------------------
```

Observations and conclusions:

1. From the heatmap of correlation matrix, we one set of moderately correlated features, which are: CNT_INSTALMENT and CNT_INSTALMENT_FUTURE.
2. The correlation of features with Target is very low, this shows the absence of a linear relationship between the feature and the target variable.

2.6.5 Plotting Distribution of Continuous Variables

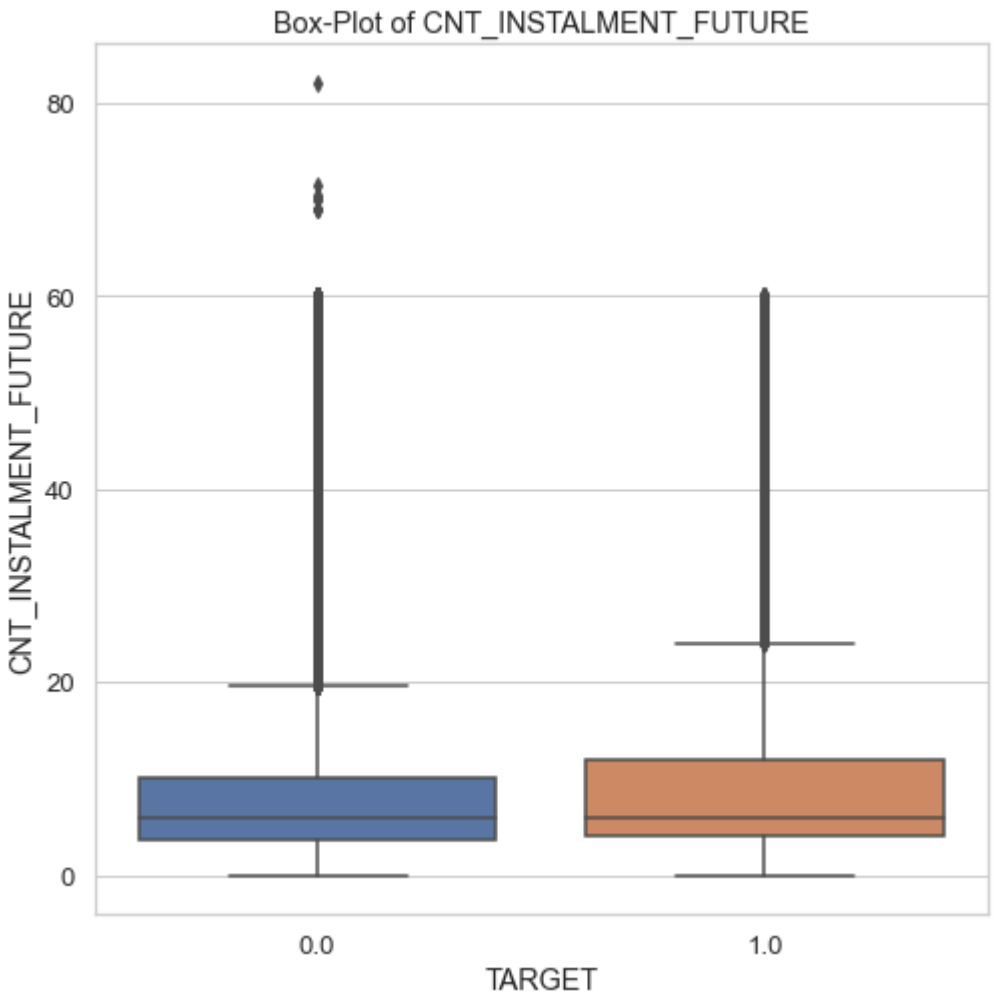
Firstly we will group by the 'SK_ID_PREV' field and aggregate with mean, so that we get an averaged row for each of the previous loan that the client had.

In [96]:
```
pos_cash_merged = pos_cash_merged.groupby('SK_ID_PREV').mean()
```

2.6.5.1 CNT_INSTALMENT_FUTURE

This column describes the number of installments left to pay on the previous credit.

In [97]:
```
plot_continuous_variables(pos_cash_merged, 'CNT_INSTALMENT_FUTURE', plots = ['box'],
                          figsize = (8,8))
del pos_cash_merged
```



Observations and conclusions:

Looking at the above box-plot for CNT_INSTALMENT_FUTURE, we see that the percentile values>50% for Defaulters are usually higher than those of Non-Defaulters. Even the upper limit whisker for Defaulters is higher than that of Non-Defaulters. This suggests that the Defaulters tend to have more number of Installments remaining on their previous credits as compared to Non-Defaulters.

2.7 credit_card_balance.csv

Description

This table consists of the monthly data related to any or multiple Credit Cards that the applicant had with the Home Credit Group. The table contains fields like balance, the credit limit, amount of drawings, etc. for each month of the credit card.

2.7.1 Basic Stats

In [98]:
```
print(f'The shape of credit_card_balance.csv is: {cc_balance.shape}')
print('-'*100)
print(f'Number of unique SK_ID_PREV in credit_card_balance.csv are:
      {len(cc_balance.SK_ID_PREV.unique())}')
print(f'Number of unique SK_ID_CURR in credit_card_balance.csv are:
      {len(cc_balance.SK_ID_CURR.unique())}')
print('-'*100)
print(f'Number of overlapping SK_ID_CURR in application_train.csv and credit_card_balan
      {len(set(app_train.SK_ID_CURR.unique()).intersection(set(cc_balance.SK_ID_CURR.un
print(f'Number of overlapping SK_ID_CURR in application_test.csv and credit_card_balanc
      {len(set(app_test.SK_ID_CURR.unique()).intersection(set(cc_balance.SK_ID_CURR.uni
print('-'*100)

print(f'Number of duplicate values in credit_card_balance:
      {cc_balance.shape[0] - cc_balance.duplicated().shape[0]}')
print('-'*100)
display(cc_balance.head(5))
```

```
The shape of credit_card_balance.csv is: (3840312, 23)
----------------------------------------------------------------------------------------
------------
Number of unique SK_ID_PREV in credit_card_balance.csv are: 104307
Number of unique SK_ID_CURR in credit_card_balance.csv are: 103558
----------------------------------------------------------------------------------------
------------
Number of overlapping SK_ID_CURR in application_train.csv and credit_card_balance.csv ar
e: 86905
Number of overlapping SK_ID_CURR in application_test.csv and credit_card_balance.csv ar
e: 16653
----------------------------------------------------------------------------------------
------------
Number of duplicate values in credit_card_balance: 0
----------------------------------------------------------------------------------------
------------
```

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	AMT_CREDIT_LIMIT_ACTUAL	AMT_DR
0	2562384	378907	-6	56.970	135000	

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	AMT_CREDIT_LIMIT_ACTUAL	AMT_DR
1	2582071	363914	-1	63975.555	45000	
2	1740877	371185	-7	31815.225	450000	
3	1389973	337855	-4	236572.110	225000	
4	1891521	126868	-1	453919.455	450000	

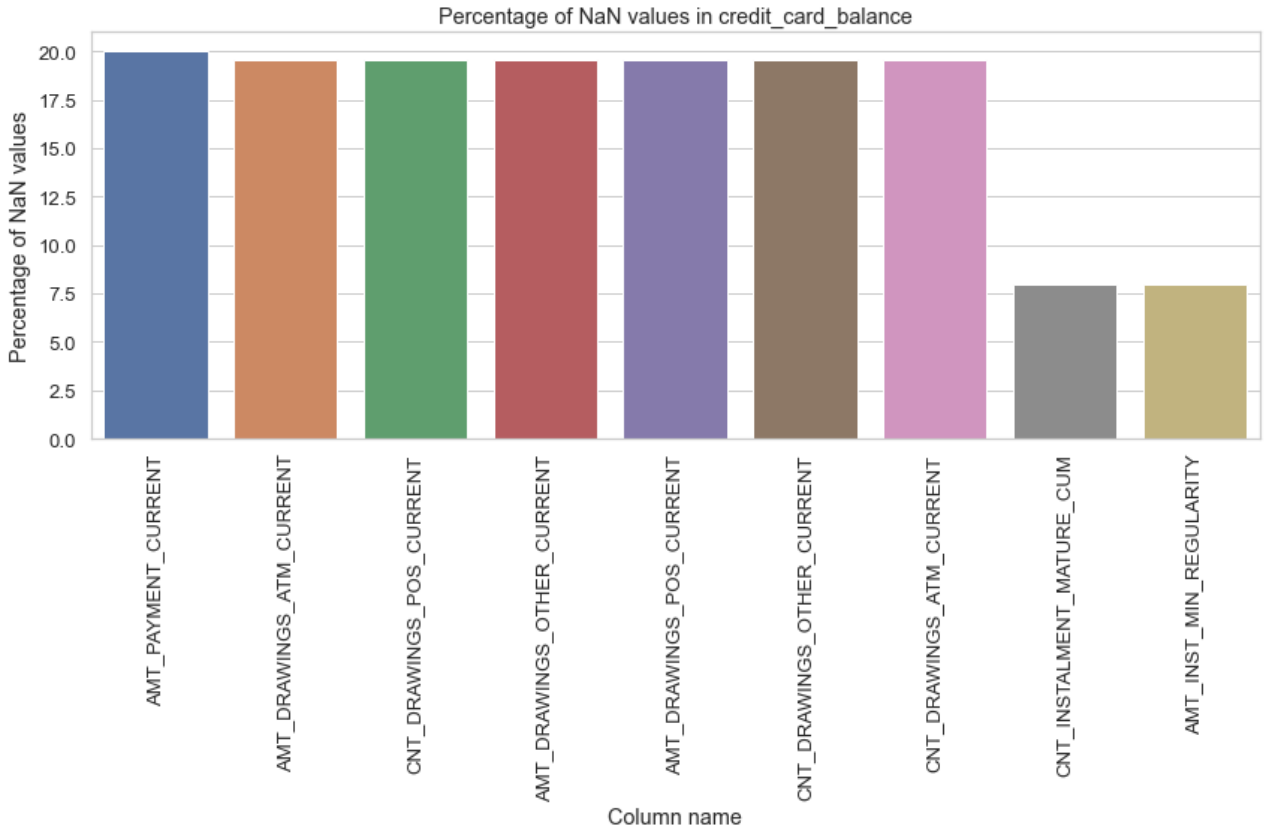
Observations and conclusions:

1. There are around 3.84M rows in the table credit_card_balance.csv, each of which corresponds to the monthly status of the Credit Card which the applicant had with Home Credit Group. This table contains 23 features which contain the statistics about each month's Credit Card status, such as Balance amount, Amount of Drawings, Number of drawings, status, etc.
2. There are 104.3k unique Credit Cards whose details are in this table.
3. Out of these 104.3k there are 103.5k unique SK_ID_CURR. What this means is that most of the applicants had just 1 credit card with them, and only few of them had more than 1. These SK_ID_CURR are the ID of the applicants who have currently applied for loan.
4. Out of the 103k unique SK_ID_CURR, 86.9k of these applicants belong to the training set, and 16.6k belong to test application set.
5. Out of 307k applicants in application_train table, only 86.9k of those had a credit card previously with Home Credit Group.

2.7.2 NaN Columns and Percentages

In [99]:
```
cc_balance_nan = nan_df_create(cc_balance)
print('-'*100)
plot_nan_percent(cc_balance_nan, 'credit_card_balance', tight_layout = False, rotation
                 figsize = (14,5))
print('-'*100)
del cc_balance_nan
```

```
----------------------------------------------------------------------------------------------------
------------
Number of columns having NaN values: 9 columns
```



Observations and conclusions:

1. Out of the 23 features, 9 of these features contain some NaN values.
2. If we look at the percentages of NaN values, they are considerably lower than the rest of the tables we have seen so far.
3. 7 of these features have close to 20% NaN values. These features are mostly related to the Amounts of Drawing and Counts of Drawings. Other two of the features are related to the installments statistics.

2.7.3 Merging the TARGETS from application_train to credit_card_balance table

In [100…

```
print("-"*100)
print("Merging TARGET with credit_card_balance Table")
cc_balance_merged = app_train.iloc[:,:2].merge(cc_balance, on = 'SK_ID_CURR', how = 'le
print("-"*100)
```

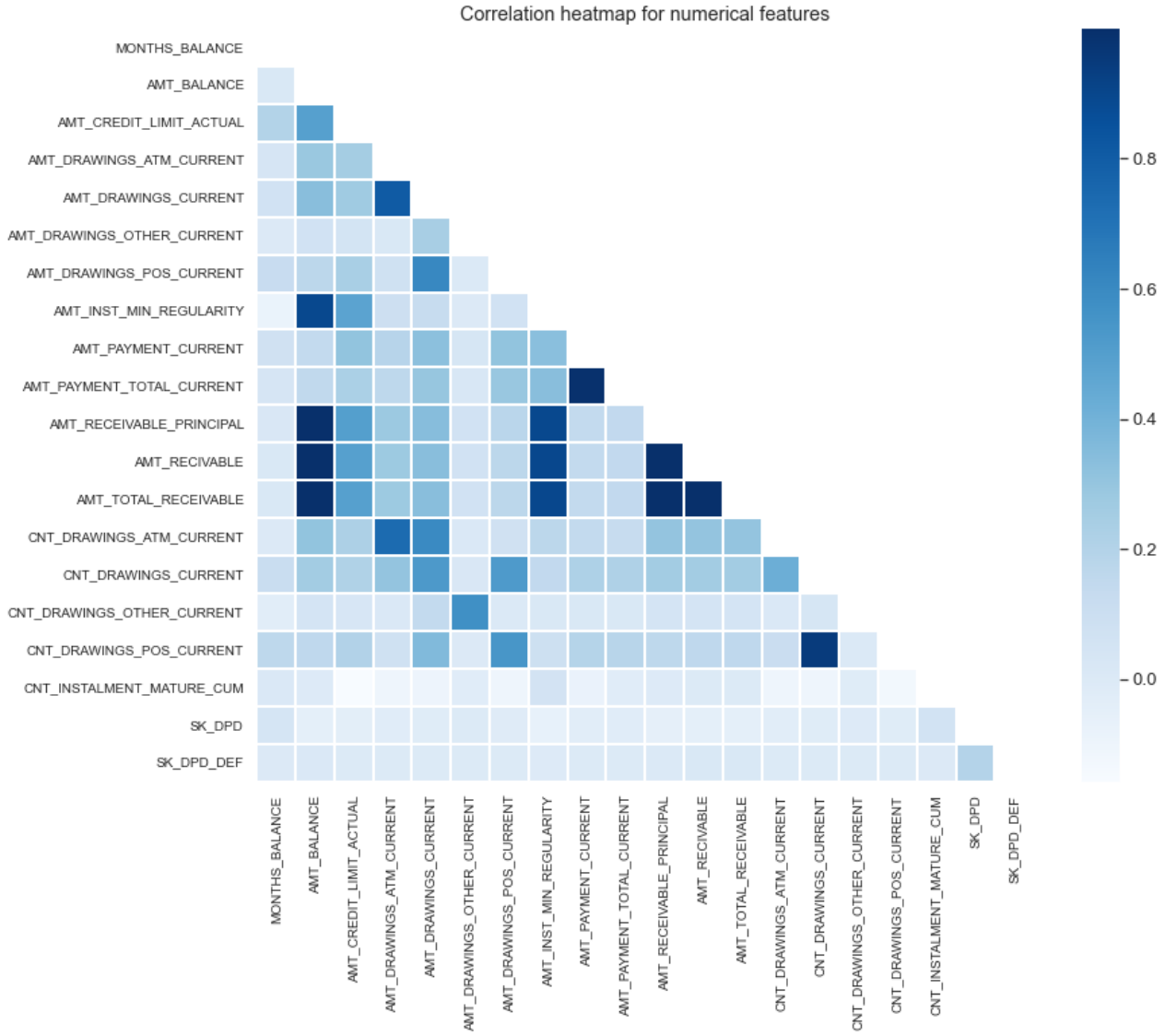
```
----------------------------------------------------------------------------------------
------------
Merging TARGET with credit_card_balance Table
----------------------------------------------------------------------------------------
------------
```

2.7.4 Correlation Matrix of Features

In [101…

```
corr_mat = correlation_matrix(cc_balance_merged, ['SK_ID_CURR','SK_ID_PREV'], figsize =
```

```
corr_mat.plot_correlation_matrix()
```



In [102…

```
#Seeing the top columns with highest phik-correlation with the target variable in credi
top_corr_target_df = corr_mat.target_top_corr()
print("-" * 100)
print("Columns with highest values of Phik-correlation with Target Variable are:")
display(top_corr_target_df)
print("-" * 100)
```

```
interval columns not set, guessing: ['TARGET', 'MONTHS_BALANCE']
interval columns not set, guessing: ['TARGET', 'AMT_BALANCE']
interval columns not set, guessing: ['TARGET', 'AMT_CREDIT_LIMIT_ACTUAL']
interval columns not set, guessing: ['TARGET', 'AMT_DRAWINGS_ATM_CURRENT']
interval columns not set, guessing: ['TARGET', 'AMT_DRAWINGS_CURRENT']
interval columns not set, guessing: ['TARGET', 'AMT_DRAWINGS_OTHER_CURRENT']
interval columns not set, guessing: ['TARGET', 'AMT_DRAWINGS_POS_CURRENT']
interval columns not set, guessing: ['TARGET', 'AMT_INST_MIN_REGULARITY']
interval columns not set, guessing: ['TARGET', 'AMT_PAYMENT_CURRENT']
interval columns not set, guessing: ['TARGET', 'AMT_PAYMENT_TOTAL_CURRENT']
```

```
interval columns not set, guessing: ['TARGET', 'AMT_RECEIVABLE_PRINCIPAL']
interval columns not set, guessing: ['TARGET', 'AMT_RECIVABLE']
interval columns not set, guessing: ['TARGET', 'AMT_TOTAL_RECEIVABLE']
interval columns not set, guessing: ['TARGET', 'CNT_DRAWINGS_ATM_CURRENT']
interval columns not set, guessing: ['TARGET', 'CNT_DRAWINGS_CURRENT']
interval columns not set, guessing: ['TARGET', 'CNT_DRAWINGS_OTHER_CURRENT']
interval columns not set, guessing: ['TARGET', 'CNT_DRAWINGS_POS_CURRENT']
interval columns not set, guessing: ['TARGET', 'CNT_INSTALMENT_MATURE_CUM']
interval columns not set, guessing: ['TARGET', 'SK_DPD']
interval columns not set, guessing: ['TARGET', 'SK_DPD_DEF']
--------------------------------------------------------------------------------
------------
Columns with highest values of Phik-correlation with Target Variable are:
```

	Column Name	Phik-Correlation
1	AMT_BALANCE	0.059838
11	AMT_RECIVABLE	0.059311
12	AMT_TOTAL_RECEIVABLE	0.059287
10	AMT_RECEIVABLE_PRINCIPAL	0.058895
0	MONTHS_BALANCE	0.050360
7	AMT_INST_MIN_REGULARITY	0.042174
17	CNT_INSTALMENT_MATURE_CUM	0.038261
13	CNT_DRAWINGS_ATM_CURRENT	0.030052
2	AMT_CREDIT_LIMIT_ACTUAL	0.028752
14	CNT_DRAWINGS_CURRENT	0.027868

```
--------------------------------------------------------------------------------
------------
```

Observations and conclusions:

1. From the heatmap of correlation matrix, we see a few couples of highly correlated features. These are:
   - AMT_RECEIVABLE_PRINCIPLE, AMT_RECIVABLE, AMT_TOTAL_RECEIVABLE and AMT_BALANCE
   - We also observe high correlation between these 3 AMT_RECEIVABLE columns
   - AMT_PAYMENT_TOTAL_CURRENT and AMT_PAYMENT_CURRENT
2. The sets of 2nd and 3rd correlating features are understandable because they more or less the same tale.
3. The correlation of features with Target isn't noticeable, this shows the absence of a linear relationship between the feature and the target variable.

2.7.5 Plotting Distribution of Continuous Variables

Firstly we will group by the 'SK_ID_PREV' field and aggregate with mean, so that we get an averaged row for each of the previous loan that the client had.

In [103…

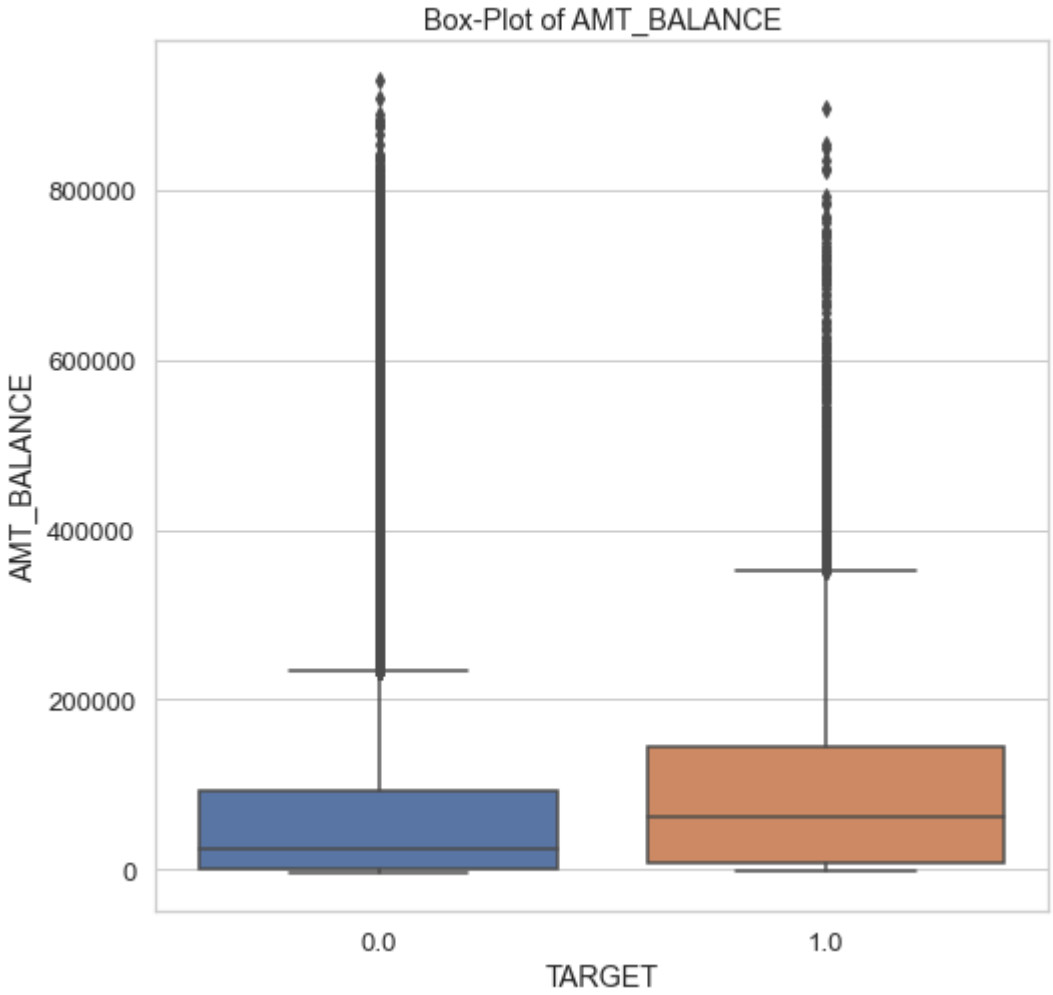
```
cc_balance_merged = cc_balance_merged.groupby('SK_ID_PREV').mean()
```

2.7.5.1 AMT_BALANCE

This column provided the average amount of balance that a person usually had on his credit card loan account for previous loan.

In [104…

```
plot_continuous_variables(cc_balance_merged, 'AMT_BALANCE', plots = ['box'], figsize =
```



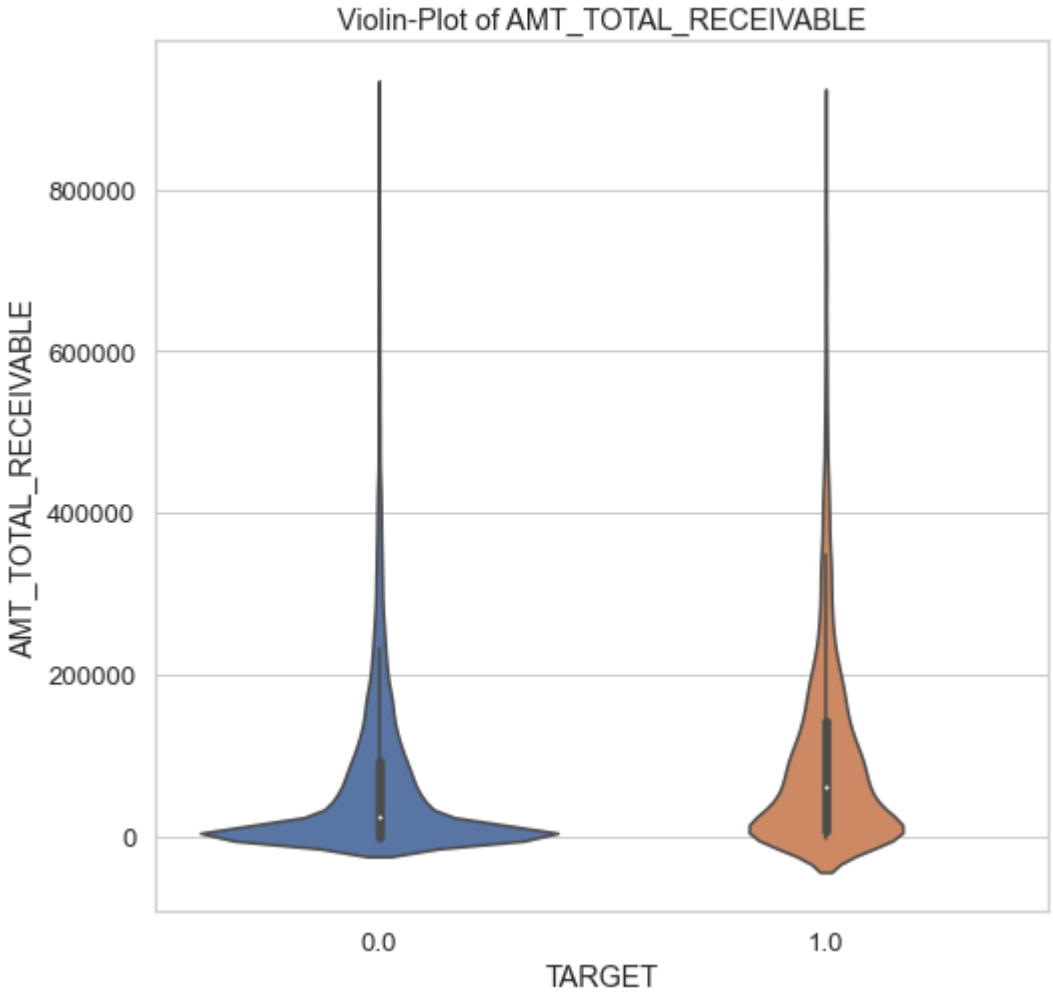
Observations and conclusions:

1. From the above plot, it can be seen that the Defaulters have a higher value of AMT_BALANCE as compared to Non-Defaulters. They show a higher values of all the quantiles and even the whiskers. This could imply that the Credit amount for Defaulters could also be relatively higher as compared to Non-Defaulters.
2. We see that the Defaulters here too appeared to have a higher minimum installment each month as compared to Non-Defaulters. This usually tells about the spending and borrowing habbit of the people. The defaulters show a higher spending and borrowing habits as compared to Non-Defaulters.

2.7.5.2 AMT_TOTAL_RECEIVABLE

This column describes the average of total amount receivable on the previous credit.

In [105…
```
plot_continuous_variables(cc_balance_merged, 'AMT_TOTAL_RECEIVABLE', plots = ['violin']
                          figsize = (8,8))
```



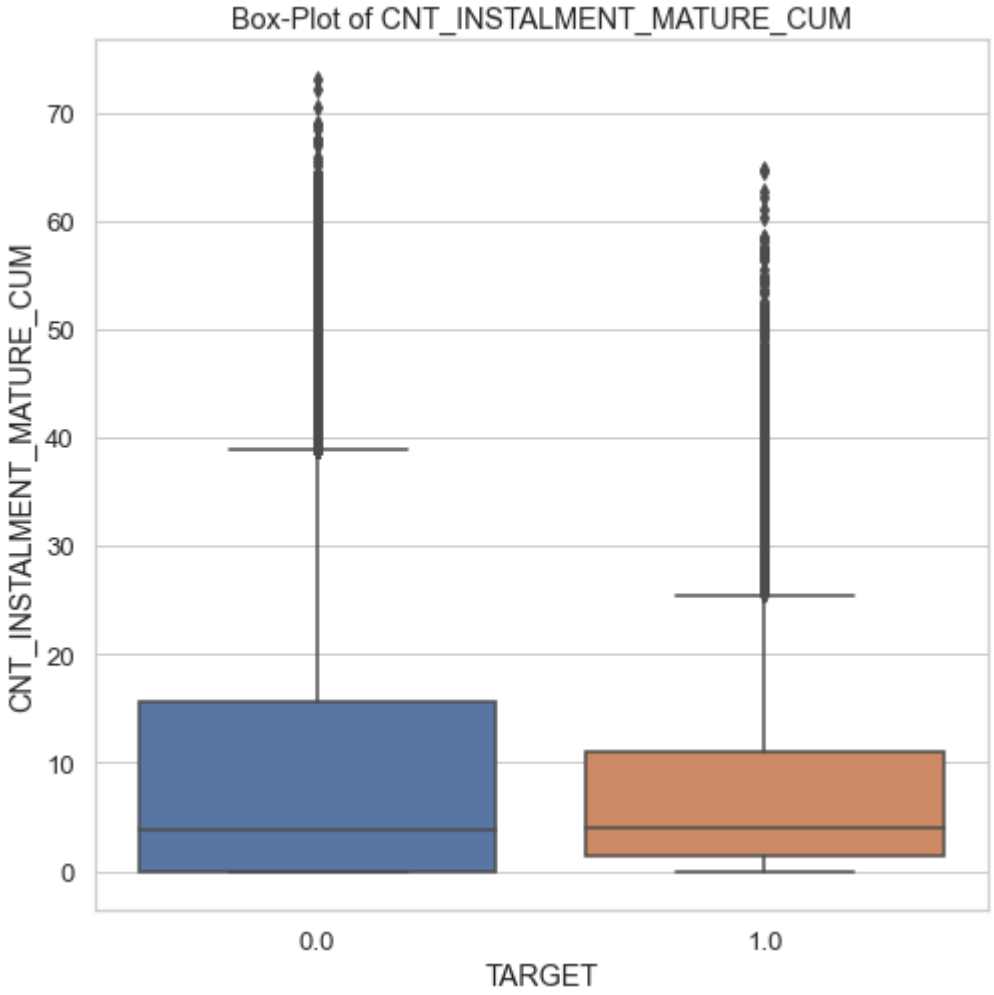
Observations and conclusions:

Looking at the violin plot of AMT_TOTAL_RECEIVABLE, we see a similar behavious as seen with other amounts as well, which is that the Defaulters usually had higher Amount Receivable on their previous credit, which may imply the higher amounts of credits that they may have taken. The PDF also shows a very higher peak at lower amounts for Non-Defaulters as compared to Defaulters.

2.7.5.3 CNT_INSTALMENT_MATURE_CUM

The column describes about the average number of installments paid on the previous credits.

In [106…
```
plot_continuous_variables(cc_balance_merged, 'CNT_INSTALMENT_MATURE_CUM', plots = ['box
                          figsize = (8,8))
```



Observations and conclusions:

From the above plot, we see a very interesting behaviour. This plot shows that the Non-Defaulters usually had higher range of values for the number of installments paid as compared to Defaulters. This might show the defaulting behaviour, where in the defaulters usually would pay fewer number of installments on their previous credit.

3 Conclusions From EDA

From the Exhaustive Exporatory Data Analysis that we performed, we can draw some high level conclusions of our given dataset.

1. Firstly, the whole dataset will need to be merged together with some ingenious way for the merged data to make sense.
2. Some categories are very well discriminatory between the Defaulters and Non-Defaulters, which could be important for the purpose of classification.
3. There are few Continuous Numerical Variables which contain Erroneous points, we would have to handle those points.
4. We also noticed some correlated features, which would just be increasing the dimensionality of data, and not add much value. We would want to remove such features.

5. Overall the dataset is Imbalanced, and we would need to come up with techniques to handle such imbalance.
6. For Default Risk prediction, the Defaulters usually tend to have some behaviour which is not normal, and thus, we cannot remove outliers or far-off points, as they may suggest some important Defaulting tendency.
7. With all these insights, we will move to Data Clearning and Feature Engineering task.

In [ ]: