

Phase 4: Neural Network Implementation and Final Results

Home Credit Default Risk

Group 10

Mark Green

margree@iu.edu



Pragat Wagle

pwagle@iupui.edu



Sahil Dhingra

sahdhin@iu.edu



Kunal Singh

singhku@iu.edu

Table of Contents

1. Phase Leader Table
 2. Credit Assignment Plan
 3. Abstract
 4. Introduction
 5. Data Description
 6. Exploratory Data Analysis
 7. Visual EDA
 8. Data Preparation
 9. Hyperparameter Tuning
 10. Baseline Machine Learning Pipelines
 11. Leakage Analysis
 12. Results and Discussion
 13. Gap Analysis
 14. Conclusion
 15. Bibliography
 16. Attachment 1 - Data Dictionary
-

NOTE

A note on the styling of this report. After Section 5, many sections will be broken up by inline code blocks as the aim of this report is to highlight the exploratory data analysis, machine learning models, and the work we are performing.

Phase Leader Table

Phase	Leader	Phase Description	Completed
1	Mark	Project Proposal: Project Plan, Data description, algorithms and metrics, baseline models, pipelines	11/15/22
2	Kunal	Baseline Report: EDA, baseline pipeline, feature engineering, video presentation	11/29/22
3	Pragat	Finetuning Model: Hyperparameter tuning, feature selection, ensemble methods, video presentation	12/06/22
4	Sahil	Final Report: Implement neural network, advanced models and loss functions, video presentation	12/13/22

Credit Assignment Table

Phase	Person	Hours	Effort
4	Mark	24	Mark's goal was (1) build a pytorch neural network model using multilayer perceptrons and apply it to the existing pipeline infrastructure, (2) to hyperparameter tune MLP models on Collinearity Reduction, Target Class Rebalancing, and optimizers, neuron architectures and other Multilayer Perceptron parameters (3) to facilitate production of the document deliverable, and (4) to help develop the summary video. Mark achieved his goal to complete these tasks by 12/13/2022. Mark accomplished these goals by adopting the HW11 Section 9 pytorch code to the existing pipelines, running and hyperparameter tuning MLP models, recording a video outside of the group meeting time, and contributing to the report document. Accomplishing these goals allowed us to test a large variety of MLP models and tuning combinations contributing to collective knowledge.
4	Sahil	12	Sahil's goal for this phase was to look into MLP implementation run few experiments and find optimal result. In addition, Sahil also had to work on balancing data, create data flow diagrams, phase diagrams, examine and write details on leakage, write abstract and perform gap analysis. All this has to be completed by Dec-13. Sahil was able to utilize the MLP pipelines and run few experiments. Along with that, Sahil was able to verify leakage, implement balancing imbalance data and accomplish all the writing tasks. Concluding this project and all assigned tasks was important to understand end to end ML implemtation of real world data.
4	Pragat	26	Pragat's goal was to attempt to get pytorch lightning working, work on the discussion and results, write the powerpoint presentation, put together the video, help with the conclusion and clean up sections of the notebook. Pragat will accomplish the goal by looking at the existing code and reading python documentation. This goal will help with completing the report portion of the project and presentation portions of the project to help arrange our report findings in a well formatted manner.
4	Kunal	20	Kunal's goal was to study and implement building multilayer perceptron model in pytorch . Kunal also worked for preparing the documentation for team and plan updates, proejct abstract, description, and conclusion. He also helped develop the summary video.
3	Mark	32	Mark's goal was (1) to select appropriate features and transform (IE log- and root- transform) the bureau and bureau_balance datasets, (2) to hyperparameter tune models on Collinearity Reduction, Engineered Features, Logistic Regression, Random Forests, and Gradient Boosted Trees, (3) to facilitate production of the document deliverable, and (4) to help develop the summary video. Mark achieved his goal to complete these tasks by 12/6/2022. Mark accomplished these goals by evaluating the bureau and bureau_bal tables, implementing the CollinearityReducer() class and reduce_mem_usage function , running and hyperparameter tuning logistic regression models, random forest and xgb models, recording a video outside of the group meeting time, and contributing to the report document. Accomplishing these goals allowed us to vet a large variety of models and tuning combinations contributing to collective knowledge which ultimately resulted in our current improved model.
3	Sahil	24	Sahil's goal for this phase was to do detailed feature engineering for POS Cash Balance, Previous application, and Installments payments by adding new attributes, doing transformation using distribution plots and skewness measure and doing naming convention on elements to avoid duplicate names as many tables had same name for the features. Another goal was to implement decision tree and also look into neural network starter code to be used for next phase. Apart from technical. another doal was to actively participate in documenting

abstract, feature engineering, hyper-parameter tuning and concluding all the results. All of this needs to be accomplished by 12-06-2022. Sahil was able to perform all the duties well in time and was also active participant in discussion related to project progression, documentation, and presentation. This has helped Sahil gain better understanding on importance of detailed EDA and plotting data to better understand to data.

Pragat's goal was to feature transform the credit card balance and the application train and finally the application test to test with the model that was trained on the transformed data. Also to do visualizations on those tables to determine which columns skewness improved after log transformation and to log transform them if they did. Pragat is also to work on some of the writing portions of the notebook. Also another goal is to run the some of the pipeline and to submit it to kaggle. Pragat will accomplish this goal by looking at the existing code, reading python documentation, and by using the web to find feature transformations. This goal will help with Feature Engineering to eventually create a more improved model from phase 2 to improve test set accuracy on the dataset provided by kaggle.

3 Pragat 26

3 Kunal 20

Mark's goal was (1) to transform (IE denormalize) and perform EDA on the bureau and bureau_balance datasets, (2) to explore multicollinearity between variables in the aggregated dataset, (3) to facilitate production of the document deliverable, and (4) to help develop the summary video. Mark achieved his goal to complete these tasks by 11/29/2022. Mark accomplished these goals by writing code to aggregate and explore the bureau and bureau_bal tables, combining work from multiple documents into a singular report document and editing/formatting to produce a final draft document in html, pdf, and ipynb formats, and engaging in zoom meetings to facilitate workflow and develop the video. Accomplishing these goals completes a significant piece of the data preparation and provides insights into the bureau data and the "rolled-up" denormalized data. Accomplishing these goals also ensures we have high-quality deliverables that are suitable for reading by audiences with varied ML backgrounds.

2 Mark 24

2 Sahil 42

Sahil's goal was to write reusable functions like aggregation, summarization, ETL-transformations for EDA and functions like Num plots, categorical plots, NULL count, Dendograms, histograms, and unique feature plots for EDA visualization. These goals were required to facilitate creation of new features, pre-processing, remove data below a threshold value(0.7), aggregation of hundreds of attributes for POS (POS-CASH Balance) and PREVAPP (previous applications) tables, and then rolling them to the application train and test tables. Helper functions were used for both EDA and visual EDA. Another goal was to publish details which could be added w.r.t functions. Last goal was to have the results and provide the discussion considering the feature engineering and score recorded in the last phase and doing a comparative analysis. This is all to be done before 11/29/2022. Sahil accomplished these goals by running and understanding the starter notebook, research on helpful EDA, EDA visualization, and feature engineering techniques which could be used across tables. Accomplishing these goals will enable Sahil to have better understanding of data visually and statistically. Accomplishing these goals will also provide insight for the next phase of targeted feature engineering and hyper-parameter tuning.

Pragat's goal was to create visualizations including correlation plots and numerical plots such as a histogram, barplot, and kernel density tables. Another goal was to roll up the credit card balance table by summarizing and aggregating that data and then merging into the train data. Lastly, to run, train the pipeline on the final aggregated table and test it on the final aggregated test table to measure accuracy of the model on the test set, and submit to Kaggle.

2 Pragat 34

This is all to be done before 11/29/2022. Pragat will accomplish this goal by looking at the existing code, reading python documentation, and by using the web to find existing implementations. This goal will help with EDA to eventually create a more improved model from phase 1 to improve test set accuracy on the

dataset provided by kaggle.

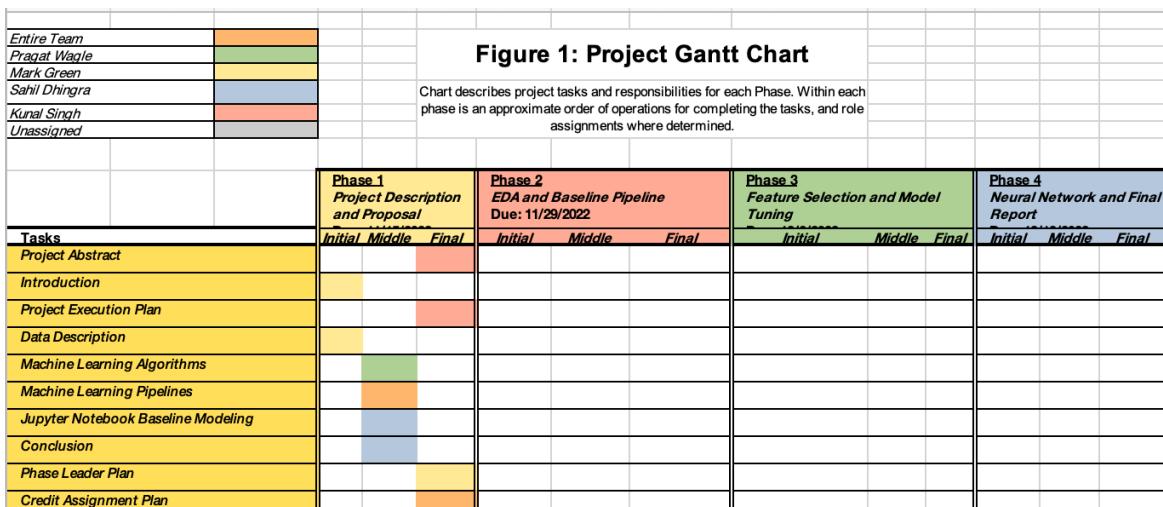
Kunals goal was to load , transform and perform EDA on “installments payments” table. He explored the collinearity between the variables in the dataset. He also worked for preparing the documentation for Team and Plan updates, Project Abstract and description. He also helped develop the summary video.

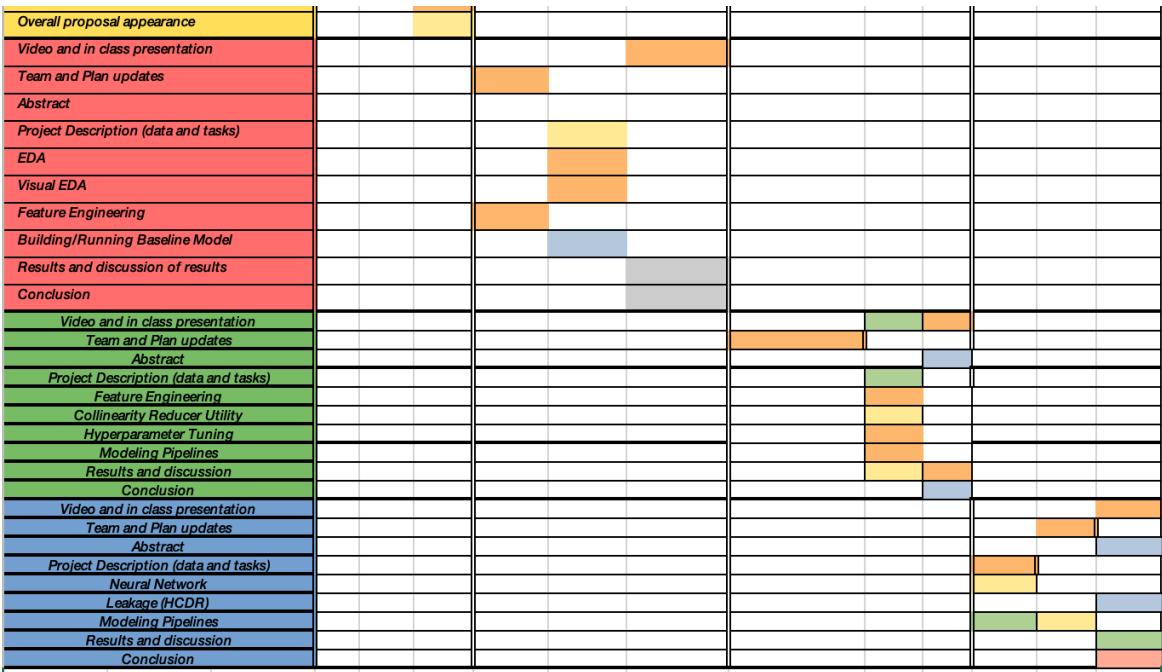
Mark's goals were to accomplish exploration and write about the data description, combine project elements developed by team members into a singular document, organize phase leader table, edit document appearance and content, guide discussion, goals, and organize team and resources as Phase 1 Leader, and to collaborate with team members where needed by November 15, 2022. Mark accomplished these goals by stitching together elements into a jupyter notebook and formatting the markdown into a report style, setting up a project Github resource and downloading/democratizing datasource, exploring and describing the data structure, editing the jupyter notebook submittal, being an active leader on team discussions, consulting other team members on writing in the Project Execution Plan, ML Algorithms, and ML Pipelines sections. Accomplishing these goals is key to project delivery in Phase 1 because these tasks complete the logistical elements of report development. They also contribute to the group understanding of the data structure and context. This work also sets precedents and tools for delivering future reports.

Sahil's goal for this phase was to do EDA, feature engineering, create pipelines, create Block Diagram, conclude results and publish results for the same in Kaggle by 15-NOV-2022. To achieve this, Sahil went through the initial notebook and worked from there to do EDA, feature selection and create pipeline with different attribute list using the knowledge from prior assignments. Accomplishing this goal will help us get us to baseline test score and AUC score which will give us a good idea on our predictions and what more we can do to make better predictions.

Pragat's goal was to describe the algorithms to be used and work on the getting it done prior to the November 15th deadline. Pragat will accomplish this goal by looking at the implementations on sklearn and the prioritize the most important parameters for each algorithm by looking at existing implementations seen in the documentation. Also by looking at existing implementations he will determine the loss functions and metrics and analysis to be used to measure the accuracy and overall quality of the algorithm. This goal will help to prioritize the most important arguments to consider for hyper-parameter tuning, optimizing the model, and in measuring the overall quality of results.

Kunal's was to develop the Abstract and Project Execution plan. He accomplished this by collaborating with team members on regular team calls. This is important for the project as it helps to outline a plan of expectations and summarizes work done this week.





Abstract

The Home Credit Group needs a machine learning-based classification model to make accurate lending decisions for individuals by predicting loan default risk using a scope of data beyond just traditional credit history. Previously EDA and feature engineering, we did 7 set of aggregations on normalized supporting tables. We also added 20 new features, 14 features were square-root transformed while 13 features were log-transformed. The data fed into a custom collinearity reducer which removed elements based on threshold correlations. We performed various experiments using Linear regression, Decision tree, Random Forest, XGBoost, and MultiLayer Perceptron on imbalanced and balanced data. Multiple metrics were used such as AUC-ROC, Recall, and Balanced accuracy. Our best model was Logistic Regression model with parameters {C:'0.01', penalty:'L1', solver:'liblinear'} with Kaggle private score as 0.7640 and public score was 0.7689. For MLP the private score was 0.7552 and public score was 0.76174.

Introduction

Loans have always been an important part of people's lives. Each individual has different reasons for borrowing a loan. It could be to buy a dream car or a home, to set up a business, or to buy some products. A large part of the population finds it difficult to get their home loans approved due to insufficient or absent credit history. It is a major challenge for banks and other finance lending agencies to decide for which candidates to approve housing loans.

The Home Credit organization's mission is to provide responsible lending solutions to people with little to no credit history. Home Credit wants to determine their customer's eligibility for their financial products by determining their risk of default - the risk they will miss payments.

Group 10 has been tasked to evaluate their data and develop a machine learning ("ML") classification tool to predict loan default risk which uses more features than just the traditional credit history. Once developed, the *Wagle-Dhingra-Singh-Green model* will eventually be used as a tool to evaluate future loan candidates.

This report discusses the integration of the full normalized dataset into a machine learning model. The Exploratory Data Analysis (*EDA*) performed throughout this process is also presented as a means to familiarize the data and to make informed decisions throughout the data engineering and model integration process. Feature engineering and colliearity reduction hopes to build up the process developed from *EDA*. Many existing model utilize Logistic Regression due to it's capabilities to classify linearly separable data and it's ease of use. Decision Tree is another algorithm that builds a decision making process for classification. Xgboost is another model that utilizes gradient boosting and building up errors in the previous models to reduce errors in future models. Multilayer perceptron models are also implemented to explore a deep learning approach to solve this classification problem. Utilization of different models provide insight to the differing performances between models and allows selection of the best performing model for deployment.

Tasks To be Tackled

The focus of Phase 4 is to develop and test a multilayer perceptron neural network model to solve the classification task. The best multilayer perceptron model is compared to the best parametric model to evaluate both methods. Another task to be tackled is handling the target class imbalance in the original dataset - the class '0' is significantly overrepresented compared to class '1' in the `application_train|test` datasets.

A generalized machine learning project pipeline is outlined in figure 3 below and each step is described in detail in the Phase 1 Project report. The steps relevant to this phase (7, 8, 9) are outlined in detail below.

1. Data Logging and ETL

- The data is segregated in multiple csv files related to each other by primary and

foreign keys. Extract-Transform-Load operations must be performed on the dataset to consolidate the datasources into single denormalized table for efficient analysis.

2. Exploratory Data Analysis

- While integrating the data, we explore the data with the goal of answering questions like:
 - How many features are present?
 - How are they correlated to one another and the target?
 - How is the data quality? Are there any problems such as high leverage points or multicollinearity?
 - What are the data types, missing values, or non-numeric features?
 - Are there any patterns between the predictor and response features?
- The denormalized data is explored to find features which have the most impact on predicting the default risk of a loan applicant. Additional feature characteristics such as correlation between features and skewness are also observed and noted at this stage.

3. Feature Engineering

- Using unnecessary features to train a machine learning model reduces the overall model accuracy and increases the model complexity and bias. To counter these issues, Feature Selection is implemented to reduce the number of input variables for the model by using only features with a relevant effect on predicting the target variable. Some feature selection methods include finding correlation coefficients, forward selection, backward selections, P-value tests, regularization, gradient boosting, and principal components analysis.
- The features also need to be re-engineered for optimal learning. This is accomplished by restructuring the input data such that the gradient surface of the model's objective function is sufficiently convex and of a form sympathetic to gradient descent optimization. Such feature engineering tasks include transforming categorical features into a numerical space, transforming skewed features into a normalized space, or extracting relevant information "hidden" within text strings.
- Comparing distributions and skewness with original columns to root transformed and log transformed columns was a part of the decision process to transform on leave columns untransformed.

1. Hyper-parameter Tuning

- Implement multiple machine learning models including but not limited to Logistic regression, Ensemble methods, Decision Trees, XGBoost etc along with hyper-parameter tuning. When choosing parameters, need to choose wisely and test with multiple round of value to check for best parameters.

2. Evaluation Metrics

- Compare the results before and after doing feature engineering and hyper-parameter tuning. Also, check for **recall score** and see **balanced accuracy**.

- Evaluate why we got the values as result, for both the score and accuracy.

PHASE 4

1. Balancing skewed Classification

- There are multiple ways that we have taken care of balancing the Target classification. As we have seen, data is skewed 92:8 towards people who can repay the loans. Therefore, it affects the prediction as well. Ways which we have used to balance classification are:
 - + Random under-sampling
 - + Random over-sampling
 - + Using other metrics for accuracy mainly AUC-ROC, Recall, Balanced Accuracy.
 - + Using Trees ensembles (Random Forests, XGBOOST, etc.)

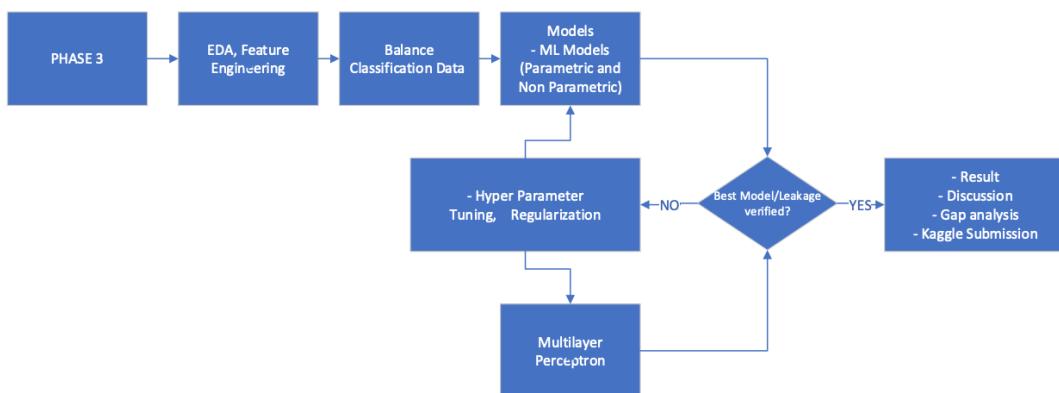
2. MLP (multi-layer perception) Pytorch Implementation

- Using the started notebook and course material, we were able to implement the multi layer perceptron on the HCDR data. Multiple experiments were done for hidden layers, multiple optimizers like Adam, Adagrad, Adamax, epochs (5,10,15) and learning rates (0.001, 0.0001, 0.004, 0.0005 etc).

9) Data Leakage exploration

- From a thorough review of how the data is transformed and the train set and test attained, it needs to be verified that no leakages occurred during that process or if so where it occurred.

Diagram of some of the major tasks that were tackled throughout the 4 phases:



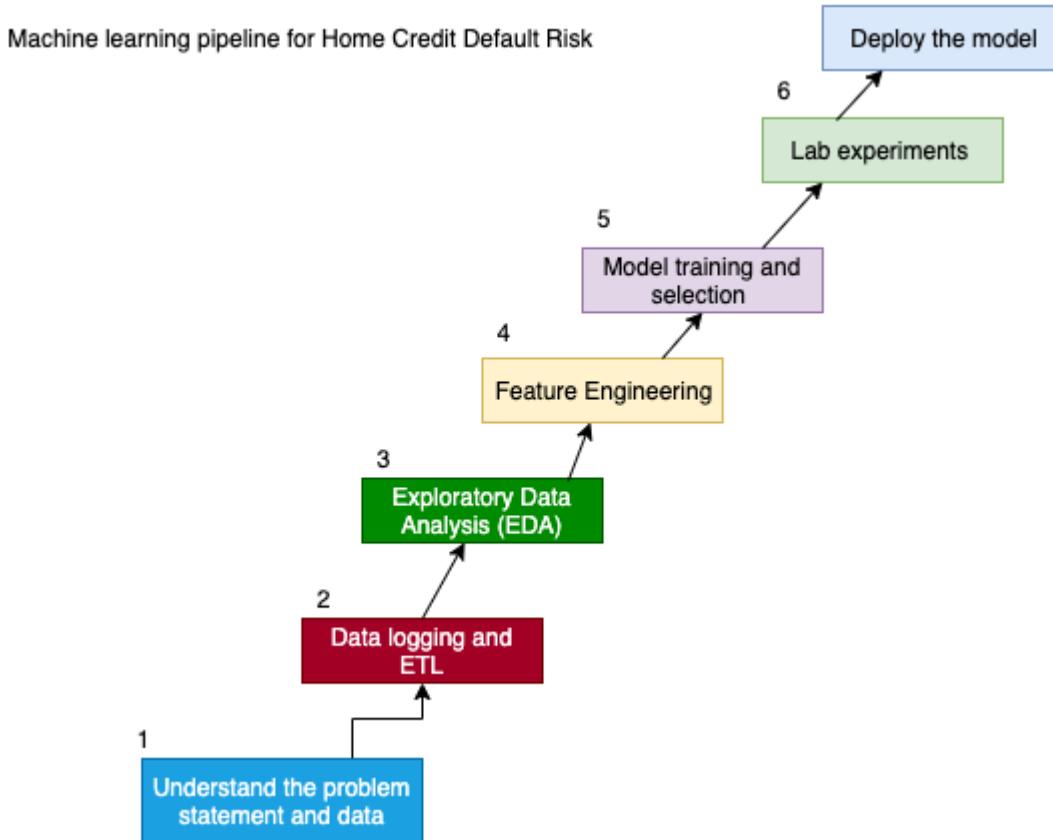


Figure 3: Visualization of key project steps to successfully build a machine learning model application.

Challenges

Challenges encountered during this phase include the following:

1. Challenge during feature engineering was to align everyone to have consistent feature engineering which includes creating new features and transformations. We implemented thenaming changes for attributes very late as we later thought that it was an important task to do for feature tracking when we have duplicate feature names and once we roll up to Train and test data.
2. Collinearity reducer function was doing a great job but was taking some good 30-40 mins each time. This does not include running pipelines.
3. Running the model was challenging given the full raw dataset is approximately 2.7 gb of data. The process of denormalizing this dataset results in a high-dimensional dataset which can be cumbersome for complex models to work on. Local machines were having difficulty running the pipeline on the unrefined denormalized dataset.
4. Hyper-parameter tuning was an intensive job which broke many a times. We had to limit the number of hyper-parameters to consider as more parameters meant more and more execution time which was ultimately leading to kernel failure.

Data Description

Although the Data Description is also discussed in the Phase 1 Proposal Report, a review of the underlying data structure will aid understanding the process by which the data is denormalized and prepared for the ML pipeline.

The data provided in the loan applications give insights into an applicant's current demographic and social status. Data from past credit accounts also give insight into an applicant's past financial behaviors. Notably, the primary key for each sample (loan application) is the Loan ID `SK_ID_CURR`. This relates the current loan application to associated loans from Home Credit's past account records, and to associated loans from other institution's that were reported to the Credit Bureaus. The data are stored in 7 different table schemas shown on Figure 2 below. The specific attributes in each table, along with their descriptions, are stored in an addendum table named `HomeCredit_columns_description.csv`.

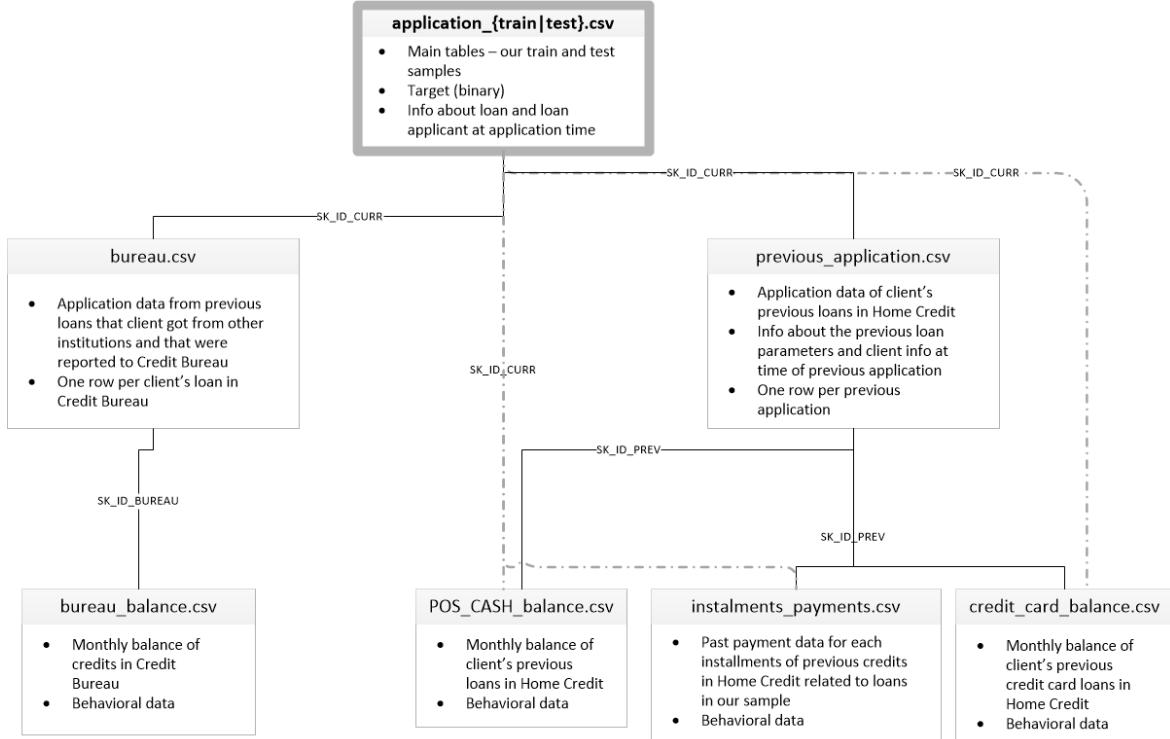


Figure 2: Entity Relation Diagram for Home Credit Applicant Datatables

Home Credit provides three main sources of data upon which to learn the ML model: applicant information, credit bureau information, and Home Credit's account records. The sections

below describe the data from these various sources, and some of the key variables.

Background on the dataset

Home Credit is a non-banking financial institution, founded in 1997 in the Czech Republic.

The company operates in 14 countries (including United States, Russia, Kazakhstan, Belarus, China, India) and focuses on lending primarily to people with little or no credit history which will either not obtain loans or became victims of untrustworthy lenders.

Home Credit group has over 29 million customers, total assets of 21 billions Euro, over 160 millions loans, with the majority in Asia and almost half of them in China (as of 19-05-2018).

While Home Credit is currently using various statistical and machine learning methods to make these predictions, they're challenging Kagglers to help them unlock the full potential of their data. Doing so will ensure that clients capable of repayment are not rejected and that loans are given with a principal, maturity, and repayment calendar that will empower their clients to be successful.

-- *From original Jupyter Notebook*

Data files overview

There are 7 different sources of data:

- **application_train/application_test:** the main training and testing data with information about each loan application at Home Credit. Every loan has its own row and is identified by the feature SK_ID_CURR. The training application data comes with the TARGET indicating **0: the loan was repaid** or **1: the loan was not repaid**. The target variable defines if the client had payment difficulties meaning he/she had late payment more than X days on at least one of the first Y installments of the loan. Such case is marked as 1 while other all other cases as 0.
- **bureau:** data concerning client's previous credits from other financial institutions. Each previous credit has its own row in bureau, but one loan in the application data can have multiple previous credits.
- **bureau_balance:** monthly data about the previous credits in bureau. Each row is one month of a previous credit, and a single previous credit can have multiple rows, one for each month of the credit length.
- **previous_application:** previous applications for loans at Home Credit of

- **previous_application**: previous applications for loans at Home Credit of clients who have loans in the application data. Each current loan in the application data can have multiple previous loans. Each previous application has one row and is identified by the feature `SK_ID_PREV`.
- **POS_CASH_BALANCE**: monthly data about previous point of sale or cash loans clients have had with Home Credit. Each row is one month of a previous point of sale or cash loan, and a single previous loan can have many rows.
- **credit_card_balance**: monthly data about previous credit cards clients have had with Home Credit. Each row is one month of a credit card balance, and a single credit card can have many rows.
- **installments_payment**: payment history for previous loans at Home Credit. There is one row for every made payment and one row for every missed payment.

-- From original Jupyter Notebook

Loan Application Information

This is a denormalized pair of tables `application_test|train.csv` with attribute information about the current loan. These attributes describe both the financial instrument itself (the contract type, rate, size of loan, annuity, etc.) and also the demographic information for the applicant (age, gender, income, education, etc.). In addition to these basic demographic data on the applicant, a suite of other very detailed information about the applicant are provided including details on the building/property in which they live, differences in work and home addresses, car ownership, if the applicant's friends recently defaulted on any loans, which documents were turned in, and many more. This table also contains the target variable `TARGET`. This is a binary variable indicating whether or not the client had payment difficulties on the loan.

These data are explored and a baseline model is created in the supplementary Baseline Jupyter Notebook in [Section 10](#).

Credit Bureau Information

Some applicants have had loans through other financial institutions which were reported to the Credit Bureaus. Data from these loan applications are available from `bureau.csv` and are related to the current loan application IDs. A monthly time series describing the payment status through loan term is available for each of these related outside institution loans in `bureau_balance.csv`, related to `bureau.csv` by `SK_ID_BUREAU`. The `STATUS` attribute in this table categorically indicates whether a loan has past-due payments on a given statement. These data can be used to evaluate an applicant's past payment behaviors.

Home Credit Previous Records

Some applicants have had loans or other financial products through Home Credit in the past. These related financial product applications are made available under `previous_application.csv`. The `SK_ID_PREV` relates these applications to payment and balance information depending on the type of financial product (credit card balances `credit_card_balance.csv`, installment payments `installments_payments.csv`, and loan balances `POS_CASH_balance.csv`). These data can be used to evaluate an applicant's past payment behaviors.

The credit card balances table shows detailed statement accounts of an applicants credit card withdrawals, credit limit, and days past due. Similarly, the installment payments and loan balances tables detail the amount prescribed and paid for various loan installments as well as the days past due for any late payments.

Data Analysis Environment Setup

This section is composed of code that sets up the data for exploration, denormaliation, and integration with an ML pipeline. It assumes the `*.csv` files from the Kaggle dataset are stored in a folder at the following directory relative to the path of this notebook:

```
DATA_DIR = "../Data/"
```

Import Required Packages

```
In [7]: # import packages

import os
import time
import warnings
import zipfile
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import Pipeline, FeatureUnion, make_pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.feature_selection import SelectKBest
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.linear_model import LogisticRegression
from pandas.plotting import scatter_matrix
# import missingno as msno
# !pip install msno

warnings.filterwarnings('ignore')
```

```
In [8]: !pwd
```

```
/root/shared/Assignments/Unit-Project-Home-Credit-Default-Risk/HCDR_Phase_1_baseline_submission
```

Loading Data

Creating dictionary for datasets so that we can keep track of datasets and also make them callable to functions.

```
In [9]:
```

```
DATA_DIR = "/../Data/"

ds_names = ("application_train", "application_test", "bureau", "bureau_balance",
            "credit_card_balance", "installments_payments", "previous_application",
            "POS_CASH_balance")

datasets = {}
datasets_transformed = {}
datasets_agg = {}

for ds_name in ds_names:
    datasets[ds_name] = pd.read_csv(os.getcwd() + DATA_DIR + f'{ds_name}.csv')
```

Data Dictionary, Size, and Overview

As part of the data download comes a *Data Dictionary* named `HomeCredit_columns_description.csv` and it is summarized in [Section 11](#) as an attachment to this report. The subsections below give brief table descriptions, variable name overviews, and head views for the attributes in the normalized data tables. Additionally included are information on the size (memory usage) of each table. All together summed the dataset is approximately 2.7 gigabytes large.

Application train

- **application_train/application_test:** the main training and testing data with information about each loan application at Home Credit. Every loan has its own row and is identified by the feature `SK_ID_CURR`. The training application data comes with the TARGET indicating **0: the loan was repaid** or **1: the loan was not repaid**. The target variable defines if the client had payment difficulties meaning he/she had late payment more than X days on at least one of the first Y installments of the loan. Such case is marked as 1 while other all other cases as 0.

```
In [5]:
```

```
def load_data(df,name):
    #df = pd.read_csv(in_path)
    print(f'{name}: shape is {df.shape}')
    print(df.info())
    display(df.head(5))
    return df

ds_name = 'application_train'
datasets[ds_name]= load_data(datasets[ds_name],ds_name)
```

```

application_train: shape is (307511, 122)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Columns: 122 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(41), object(16)
memory usage: 286.2+ MB
None
   SK_ID_CURR  TARGET  NAME_CONTRACT_TYPE  CODE_GENDER  FLAG_OWN_CAR  FLAG_O
0      100002      1    Cash loans           M            N
1      100003      0    Cash loans           F            N
2      100004      0  Revolving loans         M            Y
3      100006      0    Cash loans           F            N
4      100007      0    Cash loans           M            N

```

5 rows × 122 columns

Application test

In [6]:

```

ds_name = 'application_test'
datasets[ds_name]= load_data(datasets[ds_name],ds_name)

```

```

application_test: shape is (48744, 121)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48744 entries, 0 to 48743
Columns: 121 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(40), object(16)
memory usage: 45.0+ MB
None
   SK_ID_CURR  NAME_CONTRACT_TYPE  CODE_GENDER  FLAG_OWN_CAR  FLAG_OWN_REALI
0      100001    Cash loans           F            N
1      100005    Cash loans           M            N
2      100013    Cash loans           M            Y
3      100028    Cash loans           F            N
4      100038    Cash loans           M            Y

```

5 rows × 121 columns

The application dataset has the most information about the client: Gender, Income, Family Status, Education, and others.

The Other Datasets

- **bureau**: data concerning client's previous credits from other financial institutions. Each previous credit has its own row in bureau but one loan in the application data can have multiple previous credits.
- **bureau_balance**: monthly data about previous credits in bureau. Each row is one month of a previous credit and a single previous credit can have multiple rows, one for each month of the credit length.
- **previous_application**: previous applications for loans at Home Credit of clients who have loans in the application train data. Each current loan in the application train data can have multiple previous loans. Each previous application has one row and is identified by the feature SK_ID_PREV.
- **POS_CASH_BALANCE**: monthly data about previous point of sale or cash loans clients have had with Home Credit. Each row is one month of a previous point of sale or cash loan and a single previous loan can have many rows.
- **Credit_card_balance**: monthly data about previous credit cards clients have had with Home Credit. Each row is one month of a credit card balance and a single credit card can have many rows.
- **installments_payment**: payment history for previous loans at Home Credit. There is one row for every made payment and one row for every missed payment.

In [10]:

```
%time
ds_names = ("application_train", "application_test", "bureau", "credit_card_balance",
             "previous_application", "POS_CASH_balance")

for ds_name in ds_names:
    datasets[ds_name] = load_data(datasets[ds_name], ds_name)
```

```
application_train: shape is (307511, 122)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Columns: 122 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(41), object(16)
memory usage: 286.2+ MB
None
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_O'
0	100002	1	Cash loans	M	N	
1	100003	0	Cash loans	F	N	
2	100004	0	Revolving loans	M	Y	
3	100006	0	Cash loans	F	N	
4	100007	0	Cash loans	M	N	

5 rows × 122 columns

```

application_test: shape is (48744, 121)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48744 entries, 0 to 48743
Columns: 121 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(40), object(16)
memory usage: 45.0+ MB
None

```

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

5 rows × 121 columns

```

bureau: shape is (1716428, 17)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1716428 entries, 0 to 1716427
Data columns (total 17 columns):
 #   Column           Dtype    
--- 
 0   SK_ID_CURR       int64    
 1   SK_ID_BUREAU     int64    
 2   CREDIT_ACTIVE     object   
 3   CREDIT_CURRENCY   object   
 4   DAYS_CREDIT      int64    
 5   CREDIT_DAY_OVERDUE int64    
 6   DAYS_CREDIT_ENDDATE float64  
 7   DAYS_ENDDATE_FACT float64  
 8   AMT_CREDIT_MAX_OVERDUE float64 
 9   CNT_CREDIT_PROLONG int64    
 10  AMT_CREDIT_SUM    float64  
 11  AMT_CREDIT_SUM_DEBT float64  
 12  AMT_CREDIT_SUM_LIMIT float64 
 13  AMT_CREDIT_SUM_OVERDUE float64 
 14  CREDIT_TYPE      object   
 15  DAYS_CREDIT_UPDATE int64    
 16  AMT_ANNUITY      float64  
dtypes: float64(8), int64(6), object(3)
memory usage: 222.6+ MB
None

```

	SK_ID_CURR	SK_ID_BUREAU	CREDIT_ACTIVE	CREDIT_CURRENCY	DAYS_CREDIT	CREDIT_
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	

```
credit_card_balance: shape is (3840312, 23)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3840312 entries, 0 to 3840311
Data columns (total 23 columns):
 #   Column           Dtype  
 --- 
 0   SK_ID_PREV      int64  
 1   SK_ID_CURR      int64  
 2   MONTHS_BALANCE int64  
 3   AMT_BALANCE     float64 
 4   AMT_CREDIT_LIMIT_ACTUAL int64  
 5   AMT_DRAWINGS_ATM_CURRENT float64 
 6   AMT_DRAWINGS_CURRENT  float64 
 7   AMT_DRAWINGS_OTHER_CURRENT float64 
 8   AMT_DRAWINGS_POS_CURRENT float64 
 9   AMT_INST_MIN_REGULARITY float64 
 10  AMT_PAYMENT_CURRENT float64 
 11  AMT_PAYMENT_TOTAL_CURRENT float64 
 12  AMT_RECEIVABLE_PRINCIPAL float64 
 13  AMT_RECVABLE      float64 
 14  AMT_TOTAL_RECEIVABLE float64 
 15  CNT_DRAWINGS_ATM_CURRENT float64 
 16  CNT_DRAWINGS_CURRENT  int64  
 17  CNT_DRAWINGS_OTHER_CURRENT float64 
 18  CNT_DRAWINGS_POS_CURRENT float64 
 19  CNT_INSTALMENT_MATURE_CUM float64 
 20  NAME_CONTRACT_STATUS object  
 21  SK_DPD          int64  
 22  SK_DPD_DEF      int64  
dtypes: float64(15), int64(7), object(1)
memory usage: 673.9+ MB
```

None

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	AMT_CREDIT_LIMIT_ACTUAL
0	2562384	378907		-6	56.970
1	2582071	363914		-1	63975.555
2	1740877	371185		-7	31815.225
3	1389973	337855		-4	236572.110
4	1891521	126868		-1	453919.455

5 rows × 23 columns

```
installments_payments: shape is (13605401, 8)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13605401 entries, 0 to 13605400
Data columns (total 8 columns):
 #   Column           Dtype  
--- 
 0   SK_ID_PREV      int64  
 1   SK_ID_CURR      int64  
 2   NUM_INSTALMENT_VERSION  float64
 3   NUM_INSTALMENT_NUMBER  int64  
 4   DAYS_INSTALMENT  float64
 5   DAYS_ENTRY_PAYMENT float64
 6   AMT_INSTALMENT    float64
 7   AMT_PAYMENT       float64
dtypes: float64(5), int64(3)
memory usage: 830.4 MB
None
```

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER	DAYS_
0	1054186	161674		1.0	6
1	1330831	151639		0.0	34
2	2085231	193053		2.0	1
3	2452527	199697		1.0	3
4	2714724	167756		1.0	2

```

previous_application: shape is (1670214, 37)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1670214 entries, 0 to 1670213
Data columns (total 37 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   SK_ID_PREV      1670214 non-null  int64  
 1   SK_ID_CURR      1670214 non-null  int64  
 2   NAME_CONTRACT_TYPE 1670214 non-null  object  
 3   AMT_ANNUITY     1297979 non-null  float64 
 4   AMT_APPLICATION 1670214 non-null  float64 
 5   AMT_CREDIT       1670213 non-null  float64 
 6   AMT_DOWN_PAYMENT 774370  non-null   float64 
 7   AMT_GOODS_PRICE  1284699 non-null  float64 
 8   WEEKDAY_APPR_PROCESS_START 1670214 non-null  object  
 9   HOUR_APPR_PROCESS_START 1670214 non-null  int64  
 10  FLAG_LAST_APPL_PER_CONTRACT 1670214 non-null  object  
 11  NFLAG_LAST_APPL_IN_DAY    1670214 non-null  int64  
 12  RATE_DOWN_PAYMENT    774370  non-null   float64 
 13  RATE_INTEREST_PRIMARY 5951   non-null   float64 
 14  RATE_INTEREST_PRIVILEGED 5951   non-null   float64 
 15  NAME_CASH_LOAN_PURPOSE 1670214 non-null  object  
 16  NAME_CONTRACT_STATUS 1670214 non-null  object  
 17  DAYS_DECISION      1670214 non-null  int64  
 18  NAME_PAYMENT_TYPE   1670214 non-null  object  
 19  CODE_REJECT_REASON 1670214 non-null  object  
 20  NAME_TYPE_SUITE     849809 non-null  object  
 21  NAME_CLIENT_TYPE    1670214 non-null  object  
 22  NAME_GOODS_CATEGORY 1670214 non-null  object  
 23  NAME_PORTFOLIO      1670214 non-null  object  
 24  NAME_PRODUCT_TYPE   1670214 non-null  object  
 25  CHANNEL_TYPE        1670214 non-null  object  
 26  SELLERPLACE_AREA    1670214 non-null  int64  
 27  NAME_SELLER_INDUSTRY 1670214 non-null  object  
 28  CNT_PAYMENT         1297984 non-null  float64 
 29  NAME_YIELD_GROUP    1670214 non-null  object  
 30  PRODUCT_COMBINATION 1669868 non-null  object  
 31  DAYS_FIRST_DRAWING 997149  non-null   float64 
 32  DAYS_FIRST_DUE      997149  non-null   float64 
 33  DAYS_LAST_DUE_1ST_VERSION 997149  non-null   float64 
 34  DAYS_LAST_DUE       997149  non-null   float64 
 35  DAYS_TERMINATION    997149  non-null   float64 
 36  NFLAG_INSURED_ON_APPROVAL 997149  non-null   float64 

dtypes: float64(15), int64(6), object(16)
memory usage: 471.5+ MB
None

```

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

5 rows × 37 columns

```
POS_CASH_balance: shape is (10001358, 8)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10001358 entries, 0 to 10001357
Data columns (total 8 columns):
 #   Column           Dtype  
 --- 
 0   SK_ID_PREV      int64  
 1   SK_ID_CURR      int64  
 2   MONTHS_BALANCE int64  
 3   CNT_INSTALMENT float64 
 4   CNT_INSTALMENT_FUTURE float64
 5   NAME_CONTRACT_STATUS object 
 6   SK_DPD           int64  
 7   SK_DPD_DEF      int64  
dtypes: float64(2), int64(5), object(1)
memory usage: 610.4+ MB
None
```

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	CNT_INSTALMENT	CNT_INSTALMENT_FUTURE	
0	1803195	182943		-31	48.0	45
1	1715348	367990		-33	36.0	35
2	1784872	397406		-32	12.0	9
3	1903291	269225		-35	48.0	42
4	2341044	334279		-35	36.0	35

```
CPU times: user 6.69 s, sys: 170 ms, total: 6.86 s
Wall time: 6.7 s
```

In [8]:

```
for ds_name in datasets.keys():
    print(f'dataset {ds_name}: {24}: [{datasets[ds_name].shape[0]}:{10}, {datasets[ds_name].shape[1]}]')
```

dataset application_train	:	[307,511, 122]
dataset application_test	:	[48,744, 121]
dataset bureau	:	[1,716,428, 17]
dataset bureau_balance	:	[27,299,925, 3]
dataset credit_card_balance	:	[3,840,312, 23]
dataset installments_payments	:	[13,605,401, 8]
dataset previous_application	:	[1,670,214, 37]
dataset POS_CASH_balance	:	[10,001,358, 8]

Exploratory Data Analysis

Exploratory Data Analysis (*EDA*) on the input tables is necessary to understand the data and to make decisions on transforming the data into a format that is ready for the machine learning pipeline. Custom functions are designed and used for each normalized table to explore the data distribution and transform the table. The tables are transformed by aggregating the data into summary statistics for the table so the child tables can be joined to their parent tables, and ultimately rolled into the application_test_train data based on their ID keys

(SK_ID_CURR , SK_ID_PREV , SK_ID_BUREAU). The subsections below outline the main functions designed to accomplish the EDA and data transformation, and also the individual steps applied to each table to accomplish this process.

EDA Functions

A number of custom functions were developed to explore and transform the data. Names of the functions and brief descriptions of their usefulness are summarized here:

1. Summary Statistics
 - This provides an overview of summary statistics on each variable of each table
2. Feature Type Extraction
 - This function describes numerical and categorical feature types and provides reference lists for other functions in the EDA process
3. Missing Data Analysis
 - Looks at the missing data rate for a table's columns
4. Feature Aggregating
 - A function to group the tables by their ID key and aggregate them into their summary statistics
5. One-Hot-Encoding Extract-Transform-Load
 - An ETL function that one-hot encodes categorical variables so they can be aggregated, and applies the transformation to the table

Summary Statistics for All tables

Function Feature Summary:

A summary of each table is provided using the described function below. This function will take a dataframe as input and provide summary statistics for the table. This generic function will help us evaluate each table with just one line of code, improving the reusability of the code. Summary statistics will show statistics on each dataframe and through that meaningful interpretations were made about the data to further plan EDA on each dataset. The function was reutilized from the Kaggle notebook

Reference link for code used:

[Reference link](#)

Function name: feature_summary

Input: Dataframe

Output: Dataframe Summary

Function output:

1. Provide table name for which analysis in progress.
2. Shape - provides rows and columns in the data set.
3. NULLS - Number of nulls in the column.
4. %_NULL - NULL percentage. (nulls/attributes count).
5. Unique values - No of unique values for the feature.
6. Data Type - Id column numerical or categorical.
7. Max/Min - Max and min for the column. This provides idea about scale. helps if need log transformation.
8. Mean - Mean of the attribute.
9. Std - Standard Deviation of the attribute.
10. Skewness - Skewness, data distribution of transformations.
11. Sample values - Sample values for the attribute.

In [6]:

```
def feature_summary(df_fa):
    print('DataFrame shape')
    print('Rows:', df_fa.shape[0])
    print('Cols:', df_fa.shape[1])
    print("-----")
    col_list=['Null','%_Null','Unique_Count','Data_type','Max/Min','Mean','Std','Skewness']
    df=pd.DataFrame(index=df_fa.columns,columns=col_list)
    df['Null']=list([len(df_fa[col][df_fa[col].isnull()]) for i,col in enumerate(df_fa.columns)])
    df['%_Null']=round((df_fa.isnull().sum()/df_fa.isnull().count()*100),3)
    df['Unique_Count']=list([len(df_fa[col].unique()) for i,col in enumerate(df_fa.columns)])
    df['Data_type']=list([df_fa[col].dtype for i,col in enumerate(df_fa.columns)])
    for i,col in enumerate(df_fa.columns):
        if 'float' in str(df_fa[col].dtype) or 'int' in str(df_fa[col].dtype):
            df.at[col,'Max/Min']=str(round(df_fa[col].max(),2))+'/'+str(round(df_fa[col].min(),2))
            df.at[col,'Mean']=df_fa[col].mean()
            df.at[col,'Std']=df_fa[col].std()
            df.at[col,'Skewness']=df_fa[col].skew()
    print("Table Statistics")
    print("-----")
    display(df)
```

In [7]:

```
for i,ds_name in enumerate(datasets.keys()):
    print("Table under consideration:",ds_name.upper())
    print("-----")
    ds = feature_summary(datasets[ds_name])
    print("-----")
```

Table under consideration: APPLICATION_TRAIN

DataFrame shape
Rows: 307511
Cols: 122

Table Statistics

	Null	%_Null	Unique_Count	Data_type	Max/Min
SK_ID_CURR	0	0.000	307511	int64	456255/100002 271
TARGET	0	0.000	2	int64	1/0
NAME_CONTRACT_TYPE	0	0.000	2	object	-
CODE_GENDER	0	0.000	3	object	-
FLAG_OWN_CAR	0	0.000	2	object	-
...
AMT_REQ_CREDIT_BUREAU_DAY	41519	13.502	10	float64	9.0/0.0
AMT_REQ_CREDIT_BUREAU_WEEK	41519	13.502	10	float64	8.0/0.0
AMT_REQ_CREDIT_BUREAU_MON	41519	13.502	25	float64	27.0/0.0
AMT_REQ_CREDIT_BUREAU_QRT	41519	13.502	12	float64	261.0/0.0
AMT_REQ_CREDIT_BUREAU_YEAR	41519	13.502	26	float64	25.0/0.0

122 rows × 9 columns

Table under consideration: APPLICATION_TEST

DataFrame shape
Rows: 48744
Cols: 121

Table Statistics

		Null	%_Null	Unique_Count	Data_type	Max/Min	
	SK_ID_CURR	0	0.00	48744	int64	456250	100001 277
	NAME_CONTRACT_TYPE	0	0.00	2	object		-
	CODE_GENDER	0	0.00	2	object		-
	FLAG_OWN_CAR	0	0.00	2	object		-
	FLAG_OWN_REALTY	0	0.00	2	object		-

	AMT_REQ_CREDIT_BUREAU_DAY	6049	12.41	4	float64	2.0	0.0
	AMT_REQ_CREDIT_BUREAU_WEEK	6049	12.41	4	float64	2.0	0.0
	AMT_REQ_CREDIT_BUREAU_MON	6049	12.41	8	float64	6.0	0.0
	AMT_REQ_CREDIT_BUREAU_QRT	6049	12.41	9	float64	7.0	0.0
	AMT_REQ_CREDIT_BUREAU_YEAR	6049	12.41	17	float64	17.0	0.0

121 rows × 9 columns

Table under consideration: BUREAU

DataFrame shape
Rows: 1716428
Cols: 17

Table Statistics

	Null	%_Null	Unique_Count	Data_type	Max/M
SK_ID_CURR	0	0.000	305811	int64	456255/1000000
SK_ID_BUREAU	0	0.000	1716428	int64	6843457/500000
CREDIT_ACTIVE	0	0.000	4	object	
CREDIT_CURRENCY	0	0.000	4	object	
DAYS_CREDIT	0	0.000	2923	int64	0/-2923
CREDIT_DAY_OVERDUE	0	0.000	942	int64	2792
DAYS_CREDIT_ENDDATE	105553	6.150	14097	float64	31199.0/-42060
DAYS_ENDDATE_FACT	633653	36.917	2918	float64	0.0/-42023
AMT_CREDIT_MAX_OVERDUE	1124488	65.513	68252	float64	115987185.0/0
CNT_CREDIT_PROLONG	0	0.000	10	int64	9
AMT_CREDIT_SUM	13	0.001	236709	float64	585000000.0/0
AMT_CREDIT_SUM_DEBT	257669	15.012	226538	float64	170100000.0/-4705600.0
AMT_CREDIT_SUM_LIMIT	591780	34.477	51727	float64	4705600.32/-586406.1

	Null	%_Null	Unique_Count	Data_type	Max/M
AMT_CREDIT_SUM_OVERDUE	0	0.000	1616	float64	3756681.0/0
CREDIT_TYPE	0	0.000	15	object	
DAYS_CREDIT_UPDATE	0	0.000	2982	int64	372/-4194
AMT_ANNUITY	1226791	71.473	40322	float64	118453423.5/0

Table under consideration: BUREAU_BALANCE

DataFrame shape
Rows: 27299925
Cols: 3

Table Statistics

	Null	%_Null	Unique_Count	Data_type	Max/Min	Mean
SK_ID_BUREAU	0	0.0	817395	int64	6842888/5001709	6036297.332974
MONTHS_BALANCE	0	0.0	97	int64	0/-96	-30.741687
STATUS	0	0.0	8	object	-	-

Table under consideration: CREDIT_CARD_BALANCE

DataFrame shape
Rows: 3840312
Cols: 23

Table Statistics

		Null	%_Null	Unique_Count	Data_type	Max
	SK_ID_PREV	0	0.000	104307	int64	2843496/100
	SK_ID_CURR	0	0.000	103558	int64	456250/10
	MONTHS_BALANCE	0	0.000	96	int64	-
	AMT_BALANCE	0	0.000	1347904	float64	1505902.19/-42025
	AMT_CREDIT_LIMIT_ACTUAL	0	0.000	181	int64	13500
	AMT_DRAWINGS_ATM_CURRENT	749816	19.525	2268	float64	2115000.0/-682
	AMT_DRAWINGS_CURRENT	0	0.000	187005	float64	2287098.31/-621
	AMT_DRAWINGS_OTHER_CURRENT	749816	19.525	1833	float64	1529847.
	AMT_DRAWINGS_POS_CURRENT	749816	19.525	168749	float64	2239274.1
	AMT_INST_MIN_REGULARITY	305236	7.948	312267	float64	202882.0
	AMT_PAYMENT_CURRENT	767988	19.998	163210	float64	4289207.4

	Null	%_Null	Unique_Count	Data_type	Max	
AMT_PAYMENT_TOTAL_CURRENT	0	0.000	182957	float64	4278315.6	
AMT_RECEIVABLE_PRINCIPAL	0	0.000	1195839	float64	1472316.79/-42330	
AMT_RECEIVABLE	0	0.000	1338878	float64	1493338.19/-42025	
AMT_TOTAL_RECEIVABLE	0	0.000	1339008	float64	1493338.19/-42025	
CNT_DRAWINGS_ATM_CURRENT	749816	19.525		45	float64	51.
CNT_DRAWINGS_CURRENT	0	0.000	129	int64	1	
CNT_DRAWINGS_OTHER_CURRENT	749816	19.525		12	float64	12.
CNT_DRAWINGS_POS_CURRENT	749816	19.525		134	float64	165.
CNT_INSTALMENT_MATURE_CUM	305236	7.948		122	float64	120.
NAME_CONTRACT_STATUS	0	0.000	7	object		
SK_DPD	0	0.000	917	int64	32	
SK_DPD_DEF	0	0.000	378	int64	32	

Table under consideration: INSTALLMENTS_PAYMENTS

DataFrame shape
Rows: 13605401
Cols: 8

Table Statistics

	Null	%_Null	Unique_Count	Data_type	Max/Min	
SK_ID_PREV	0	0.000	997752	int64	2843499/1000001	1.90336
SK_ID_CURR	0	0.000	339587	int64	456255/100001	2.78444
NUM_INSTALMENT_VERSION	0	0.000	65	float64	178.0/0.0	8.56637
NUM_INSTALMENT_NUMBER	0	0.000	277	int64	277/1	1.88709
DAYS_INSTALMENT	0	0.000	2922	float64	-1.0/-2922.0	-1.04227
DAYS_ENTRY_PAYMENT	2905	0.021	3040	float64	-1.0/-4921.0	-1.05111
AMT_INSTALMENT	0	0.000	902539	float64	3771487.85/0.0	1.70509
AMT_PAYMENT	2905	0.021	944236	float64	3771487.85/0.0	1.72382

Table under consideration: PREVIOUS_APPLICATION

DataFrame shape
Rows: 1670214
Cols: 37

Table Statistics

		Null	%_Null	Unique_Count	Data_type	Max/Min
	SK_ID_PREV	0	0.000	1670214	int64	2845382/1000001
	SK_ID_CURR	0	0.000	338857	int64	456255/100001
	NAME_CONTRACT_TYPE	0	0.000	4	object	.
	AMT_ANNUITY	372235	22.287	357960	float64	418058.15/0.0
	AMT_APPLICATION	0	0.000	93885	float64	6905160.0/0.0
	AMT_CREDIT	1	0.000	86804	float64	6905160.0/0.0
	AMT_DOWN_PAYMENT	895844	53.636	29279	float64	3060045.0/-0.5
	AMT_GOODS_PRICE	385515	23.082	93886	float64	6905160.0/0.0
	WEEKDAY_APPR_PROCESS_START	0	0.000	7	object	.
	HOUR_APPR_PROCESS_START	0	0.000	24	int64	23/0
	FLAG_LAST_APPL_PER_CONTRACT	0	0.000	2	object	.
	NFLAG_LAST_APPL_IN_DAY	0	0.000	2	int64	1/0
	RATE_DOWN_PAYMENT	895844	53.636	207034	float64	1.0/-0.0
	RATE_INTEREST_PRIMARY	1664263	99.644	149	float64	1.0/0.03
	RATE_INTEREST_PRIVILEGED	1664263	99.644	26	float64	1.0/0.37
	NAME_CASH_LOAN_PURPOSE	0	0.000	25	object	.
	NAME_CONTRACT_STATUS	0	0.000	4	object	.

	Null	%_Null	Unique_Count	Data_type	Max/Min
DAY_S_DECISION	0	0.000	2922	int64	-1/-2922
NAME_PAYMENT_TYPE	0	0.000	4	object	
CODE_REJECT_REASON	0	0.000	9	object	
NAME_TYPE_SUITE	820405	49.120	8	object	
NAME_CLIENT_TYPE	0	0.000	4	object	
NAME_GOODS_CATEGORY	0	0.000	28	object	
NAME_PORTFOLIO	0	0.000	5	object	
NAME_PRODUCT_TYPE	0	0.000	3	object	
CHANNEL_TYPE	0	0.000	8	object	
SELLERPLACE_AREA	0	0.000	2097	int64	4000000/-1
NAME_SELLER_INDUSTRY	0	0.000	11	object	
CNT_PAYMENT	372230	22.286	50	float64	84.0/0.0
NAME_YIELD_GROUP	0	0.000	5	object	
PRODUCT_COMBINATION	346	0.021	18	object	
DAY_FIRST_DRAWING	673065	40.298	2839	float64	365243.0/-2922.0
DAY_FIRST_DUE	673065	40.298	2893	float64	365243.0/-2892.0
DAY_LAST_DUE_1ST_VERSION	673065	40.298	4606	float64	365243.0/-2801.0
DAY_LAST_DUE	673065	40.298	2874	float64	365243.0/-2889.0

	Null	%_Null	Unique_Count	Data_type	Max/Min
DAY_S_TERMINATION	673065	40.298	2831	float64	365243.0/-2874.0
NFLAG_INSURED_ON_APPROVAL	673065	40.298	3	float64	1.0/0.0

Table under consideration: POS_CASH_BALANCE

DataFrame shape

Rows: 10001358

Cols: 8

Table Statistics

	Null	%_Null	Unique_Count	Data_type	Max/Min	
SK_ID_PREV	0	0.000	936325	int64	2843499/1000001	1903216.
SK_ID_CURR	0	0.000	337252	int64	456255/100001	278403.
MONTHS_BALANCE	0	0.000	96	int64	-1/-96	-35.
CNT_INSTALMENT	26071	0.261	74	float64	92.0/1.0	1.
CNT_INSTALMENT_FUTURE	26087	0.261	80	float64	85.0/0.0	10.
NAME_CONTRACT_STATUS	0	0.000	9	object	-	-
SK_DPD	0	0.000	3400	int64	4231/0	11.
SK_DPD_DEF	0	0.000	2997	int64	2525/0	0.

Feature extraction based on Type

Function id_num_cat_feature:

This function will take a dataframe as input and return 4 lists that contain ID columns, numerical features, categorical features, and numerical features without the ID cols.

Function name: id_num_cat_feature Input : Dataframe Output : 4 Lists

Function output:

1. ID columns
2. Numerical features
3. Categorical features
4. Numerical features excluding the ID columns.

Using this function gives us multiple lists of the column types which are then used in other functions. The function returns clear separations on what columns are categorical and numerical. By having the distinctive categorical columns and numerical columns lists, different transformations are performed on them based on if they are categorical or numerical.

In [3]:

```
def id_num_cat_feature(df, text = True):
    numerical = df.select_dtypes(include=['int64', 'float64']).columns
    categorical = df.select_dtypes(include=['object', 'bool']).columns
    feat_num = list(numerical)
    feat_cat = list(categorical)

    id_cols = ['SK_ID_CURR', 'SK_ID_BUREAU']

    id_cols = [cols for cols in list(df.columns.intersection(id_cols))]
    features = list(set(df.columns) - set(id_cols))

    if text == True:
        # print eda
        print('-----')
        print(f"# of ID's: {len(id_cols)}")
        print(f" ID's:")
        print(id_cols)
        print('')
        print('-----')
        print(f"# All features: {len(features)}")
        print(f"All features:")
        print(features)
        print('')
        print(f"Missing data:")
        print(missing_data(df[features]))
        print('')
        print('-----')
        print(f"# of Numerical features: {len(feat_num)}")
        print(f"Numerical features:")
        print(feat_num)
        print('')
        print(f"Numerical Statistical Summary:")
        print('')
        print(df[feat_num].describe())
        print('')
        print('-----')
        print(f"# of Categorical features: {len(feat_cat)}")
        print(f"Categorical features:")
        print(feat_cat)
        print('')
        print(f"Categorical Statistical Summary:")
        print('')
        #print(df[feat_cat].describe(include='all'))
        print('')
        print("Categories:")
        print('')
        print(df[feat_cat].apply(lambda col: col.unique()))
        print('')
        print('-----')
    return id_cols, feat_num, feat_cat, features
```

Missing Count and percentage

Function missing_data:

This function will take a dataframe as input and provide null count and % of nulls for a dataframe.

Function name: missing_data

Input: Dataframe

Output: NULL count and NULL %

Function output:

1. NULL Count
2. NULL Percentage.

In [12]:

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

Feature Aggregating

Class: FeatureSummarizer

Class FeatureSummarizer has following aggregation parameters:

1. "min"
2. "max"
3. "count"
4. "sum"
5. "median"
6. "mean"
7. "var"

Based on the keys of the dataframe, transformation function will enable grouping of feature variables on ID and then aggregate the feature variables into their predefined statistical summaries for each grouping.

Function: runFeatureSummarizer

This function will take dataframe and features as input and using the class above, get aggregated features. The output of the above function will be the transformed aggregated features for the dataframe passed into the function argument.

In [13]:

```
class FeatureSummarizer(BaseEstimator, TransformerMixin):
    def __init__(self, features=None):
        self.features = features
        self.agg_ops = ["min", "max", "count", "sum", "median", "mean", "var"]

    def fit(self, X, y=None):
        return self

    def transform(self, X, y=None):
        keys = list(set(X.columns) - set(self.features))

        result = X.groupby(keys, as_index=False).agg({ft: self.agg_ops[ft] for ft in self.agg_ops})
        result.columns = result.columns.map(lambda ct: '_' .join([x for x in ct if x != '_']))
        result.reset_index()
        return result
```

In [14]:

```
def runFeatureSummarizer(df, features):
    print(f"df.shape: {df.shape}\n")
    print(f"Aggregated Features:\n{df[features][0:5]}\n{df[features][0:5].columns}")
    pipeline = make_pipeline(FeatureSummarizer(features))
    return(pipeline.fit_transform(df))
```

One-Hot-Encoded Extract-Transform-Load

Function eda_transformation:

Function input: Dataframe, n. Where n is a parameter for feature selection.

- This function calls id_num_cat_feature to put all features types id columns, numerical features, categorical features, and numerical features excluding id columns into 4 respectively lists.
- Categorical variables are one hot encoded into some numerical value, to allow the pipeline to make interpretations from categorical variables more easily.
- run FeatureSummarizer function is called to get all aggregated features.
- Final features are selected through feature selection from the transformed tables.
- Output of this function will be a dataframe with all aggregated features selected, which will eventually be used in feature selection.
- The function will also prints aggregated features and aggregated data.

Feature Selection:

- Once we have completed OHE, we can remove some attributes which are redundant, for example:
- count of previous application is good to have, but mean,max is not required.
- Features with DAYS, count suffice the aggregated feature, we can remove all other min, max, etc. feature for that column.
- Based on above logic, we removed certain columns from the supporting tables to remove the redundant attributes and save on execution time.

In [15]:

```
def eda_transformation(df,n):

    id_cols, feat_num, feat_cat, features = id_num_cat_feature(df)

    df = pd.get_dummies(data=df, columns=feat_cat)

    features = list(set(df.columns) - set(id_cols))
    feat_ohe = list(set(features) - set(feat_num))

    print(f"# of OHE categorical features: {len(feat_ohe)}")
    print(f'OHE Categorical features: {feat_ohe}')
    print('-----')

    df = runFeatureSummarizer(df, features)

    if n == 0:
        # bureau_balance
        feature_selection = [
            df[id_cols],
            df[[column for column in df.columns if column.startswith('MONTH_')]],
            df[[column for column in df.columns if column.startswith('STATUS_')]]
        ]
    elif n == 1:
        # bureau
        feature_selection = [
            df[[column for column in df.columns if not column.startswith(tuple(['SK_ID_BUREAU', 'SK_ID_PREV']))]],
            df[[column for column in df.columns if column.startswith('DAYS_')]],
            df[[column for column in df.columns if column.startswith(tuple(['SK_ID_CURR', 'SK_ID_CURR']))]]
        ]
    elif n == 3:
        feature_selection = [
            df[[column for column in df.columns if not column.startswith('SK_ID_CURR')]],
            df[[column for column in df.columns if column.startswith('DAYS')]],
            df[[column for column in df.columns if column.startswith('SK_ID')]],
            df[[column for column in df.columns if column.startswith(tuple(['SK_ID_CURR', 'SK_ID_CURR']))]]
        ]
    elif n == 4:
        feature_selection = [
            df[[column for column in df.columns if not column.startswith('SK_ID_CURR')]],
            df[[column for column in df.columns if column.startswith('SK_ID')]],
            df[[column for column in df.columns if column.startswith(tuple(['SK_ID_CURR', 'SK_ID_CURR']))]]
        ]
    else:
        print('ERROR: Invalid feature method. ')

    df = pd.concat(feature_selection, axis=1)

    features = list(set(df.columns) - set(id_cols))

    print('-----')
    print('Aggregated Features:')
    print('\n'.join(map(str, sorted(features))))
    print('')
    print('Aggregated Data:')
    print('')
```

```
print(df[features].describe().T)
return df
```

EDA and Feature Engineering on Tables

Since each table is structured slightly differently and they are all composed of separate variables, It is necessary to perform different steps on each of the tables as necessary. While all the tables can be explored and transformed by the `eda_transformation()` function, this function included both the eda and also feature engineering.

Feature Engineering

For the phase 3, Our task is to perform targetted feature engineering as well as hyper parameter tuning, i.e. look for the attributes which could be transformed and then use them for the pipelines.

We are taking below approach to see if we need to perform transformation.

1. CREATE NEW FEATURES:

Based on our data analysis many new columns were created. List is attached along with.

1. TRANSFORM NUMERICAL FEATURES BASED ON DISTRIBUTION

- Take all numerical features, create 3 histograms.
 - Normal data.
 - SQRT Transformed Data
 - Log transformed Data.
- Print the Skewness as well along with to see how much transformation is helping to normalize the data.

2. Collinearity reduction:

CollinearityReducer() transformer class combats multicollinearity by removing the most (multi-) collinear columns which are least correlated to the target variable. The algorithm steps are described below:

1. Given input variable X and target variable y, calculate the correlation matrix.
2. Pivot the correlation matrix into a long dataframe of correlation pairs and values (input variable 1, input variable 2, absolute correlation) and drop the following variable pairs:
 - any pair including the target variable
 - any pair with two of the same input variable
 - any pair with an absolute correlation value below a specified threshold value (default is 0.5)
3. For each variable pair, compare to see which variable is more correlated to the target variable. These input variables are given a 'win' while the input variable in the pair which is less correlated to the target variable is given a 'loss'.

Feature Tracking:

For the purpose of knowing feature origin or transformation, we have taken steps to rename the feature such that we can readily know everythign about the feature.

Here is how renames columns will look like (Consider Previous application table.):

All columns will be renamed to PA_N_VV, where PA_N

- All columns will be renamed to PA_U_XX, where PA_U_ is a prefix and XX is the actual column name.
- Revert renaming of ID columns. Because ID's are used across tables for comparison.
- If any transformation is done on features:
 - Log : PA_O_L_XX
 - SQRT: PA_O_R_XX
- Any New feature created will be named as : PA_N_XX.
- All New Features with transformations:
 - Log : PA_N_L_XX
 - SQRT: PA_N_R_XX
- Now if we have a attribute months balance which is present in both Bureau and POS Cash balance, they will be renamed to BR_O_MONTHS_BALANCE and PA_O_MONTHS_BALANCE respectively. And if we do sqrt transformation, we will add "_R after the O to show that they have root transformation.

Using string manipulation, we can readily create multiple lists to include or exclude such elements from the pipeline.

Benefits:

- 1. There are features which are common between tables, like monthly balance. This will help us know monthly balance from which table is useful for prediction.**
- 2. Based on the column name, we know the origin, transformation, new features created just by looking at the column name. We can quickly create list each like an example given below:**

All new cols ip_new_cols = [col for col in ip.columns if "IP_N" in col]

Cols on which transformation took place ip_unt_cols = [col for col in ip.columns if "IP_O_DAYS_ENTRY_PAYMENT" in col]

Transformed cols ip_r_cols = [col for col in ip.columns if "IP_N_R" in col] ip_l_cols = [col for col in ip.columns if "IP_N_L" in col]

Feature Engineering And EDA

Feature Engineering and EDA were done together in one comprehensive function for each table.

Please refer to section 6 as well for details on specific tables.

Data Type Optimization

Data Type Optimization:

When aggregated, the dataset is very large, approximately 2.5 gb of space. When reading these data in as-is, the large memory required to work with the dataset made it a cumbersome object on which to operate ML pipelines. The `reduce_mem_usage()` function is used immediately upon data read-in to counter this large memory requirement. Where possible, this function changes a given column's default datatype to a datatype with a smaller memory footprint. For example, an `int64` column composed of only 1s and 0s (IE OHE columns...) might be converted to an `int8` column with no data loss, and significant memory reduction.

This operation typically led to a ~70% memory size reduction of the imported dataset making it much easier to work with. The function is provided below with an example output. Credit to the publisher: <https://pythonsimplified.com/how-to-handle-large-datasets-in-python-with-pandas/>

Insert IMAGE

Please refer to section 6.2.1 for details as well.

Phase I: In phase 1, we did not create any features and used the application train data to train the model as-is.

Phase II:

In phase 2, we analyzed all provided tables. In addition, we used 7 aggregation parameters to aggregate the data and roll them up to train and test. Those 7 parameters were "min", "max", "count", "sum", "median", "mean", "var". We got best Test AUC ROC score as XX and XX was our Kaggle submission ROC AUC score.

In phase II, we also laid the bricks for our custom collinearity Reduction function.

Phase III:

Rigorous analysis was done on elements for phase 3 with respect to feature engineering. We initially created more columns which would make sense to describe the data more. Then, we passed all columns (numerical) through a visualization function which will create 3 set of histograms and pop out skewness of each one as well. The histograms were for Data (untransformed), Log, and Root transformed. Based on plots and skewness, elements were transformed. Finally, to cater to these massive amount of elements (aggregated, created, transformed), Numerical data goes through collinearity reduction function. This function will find the collinearity among all possible pairs. The variables with the lowest target variable correlation are dropped from the input X. The process is repeated until there are no more colinear pairs with absolute correlations above the threshold or max_iter.

We chose these 3 methods as they work in conjunction with each other and provide value to each step. Transforming is one of the best possible way to normalize the data, which will lead to better results. And as we are dealing with thousands of features, we need a solid utility which can contain the amount of features we want to use for our model and drop the ones which are least collinear with target.

Below are all 3 feature engineering methods in detail:

Created new Features

- Created new Features:** Based on our analysis, we created a series of new features and rolled them up to the application train table. Here are the details of Features and their corresponding tables:

```

POS      'PS_N_PERC_INSTL_PNDNG' Percentage of
installments pending
POS      'PS_N_CNT_INSTAL_PNDNG' Number of installments
pending
POS      'PS_N_DAYS_WTHT_TOLRNC' Days with Tolerance
PrevApp 'PA_N_APRA_CNT' Count of approved previous
application
PrevApp 'PA_N_REJ_CNT' Count of Rejected previous
applications
PrevApp 'PA_N_APCTN_CRDT_DIFF' Difference: Amount
requested in application - Actual credit amount
PrevApp 'PA_N_APCTN_CRDT_RATIO' Ratio of application
amount to amount credited
PrevApp 'PA_N_CRDT_ANNUITY_RATIO' Ratio of amount
credited to amount annuity
PrevApp 'PA_N_DWN_PYMNT_CRDT_RATIO' Ratio of down
payment to amount credited
IP      'IP_N_DIFF_PAYMNT_INSTMNT' Difference between
Payment and installment.
AT      'EXT_MEAN' Mean of External Sources, EXT 1,
EXT 2, EXT 3.
AT      'EXT_SOURCES_SUM' Sum of all EXT source
values
AT      'AT_N_R_DAYS_EMPLOYED_PERC' Ratio of days
employed to days birth.
AT      'AT_N_R_INCOME_CREDIT_PERC' Ratio of Income to
credit.
AT      'AT_N_R_INCOME_PER_PERSON' Ratio of Total
income to total family members.
AT      'AT_N_R_ANNUITY_INCOME_PERC' Ratio of
Amount annuity to total income
AT      'AT_N_R_PAYMENT_RATE' Ratio of Annuity
amount to Credit.
AT      'AT_N_R_GOODS_TO_CREDIT' Ratio of goods
price to credit
AT      'AT_N_R_PAYMENT_RATE' Ratio of Annuity
amount to Credit
AT      'AT_N_R_LOAN_GOOD_RATIO' Ratio of Credit
amount to goods price.

```

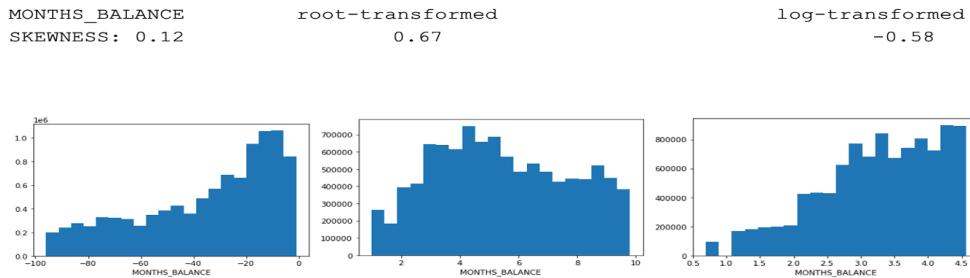
Transformations

1. Transformations: Based upon plotting distribution for attribute, log transformed attribute, and square root transformed attribute, and numerical skewness value, attributes were transformed. Below is the list of attributes transformed.

FEATURE	TABLE SOURCE TRANSFORMATION	ADDED
L: LOG		
R: ROOT		
	BUREAU	
'BR_N_R_MONTHS_BALANCE_count'	BUREAU	R
R	'BR_N_R_DAYS_CREDIT'	
	BUREAU	
'BR_N_R_DAYS_ENDDATE_FACT'	BUREAU	R
	'BR_N_L_CREDIT_DAY_OVERDUE'	L
	POS CASH BALANCE	
PS_N_R_CNT_INSTALMENT_FUTURE	POS CASH BALANCE	R
R	POS CASH BALANCE	PS_N_R_MONTHS_BALANCE
PS_N_R_CNT_INSTAL_PNDNG	POS CASH BALANCE	R
L	Previous application	PS_N_L_CNT_INSTALMENT
PA_N_R_RATE_DOWN_PAYMENT	Previous application	R
R	PA_N_R_AMT_CREDIT	
R	Previous application	PA_N_R_AMT_ANNUITY
R	Previous application	PA_N_R_CNT_PAYMENT
R	Previous application	PA_N_R_AMT_GOODS_PRICE
PA_N_R_DWN_PYMNT_CRDT_RATIO	Previous application	R
R	PA_N_R_DAYS_DECISION	
PA_N_L_CRDT_ANNUITY_RATIO	Previous application	L
PA_N_L_DAYS_TERMINATION	Previous application	L
PA_N_L_AMT_DOWN_PAYMENT	Previous application	L
PA_N_L_SELLERPLACE_AREA	Previous application	L
L	PA_N_L_AMT_APPLICATION	

	Previous application	
PA_N_L_PREV_APCTN_CRDT_DIFF		L
	Installment Payments	
IP_N_R_DAYS_ENTRY_PAYMENT		R
	Application Train	AT_L_0_ AMT_ANNUITY
L		
	Application Train	AT_L_0_ DAYS_BIRTH
L		
	Application Train	AT_L_0_ DAYS_EMPLOYED
L		
	Application Train	AT_L_0_
DAYS_REGISTRATION		L
	Application Train	AT_L_0_
DAYS_ID_PUBLISH		L

Sample Code and plot screenshots



```
for i,ds_name in enumerate(datasets.keys()):
    if ds_name.upper() in ("POS_CASH_BALANCE"):
        ps = datasets[ds_name]
        ps['POS_PERC_INSTL_PNDNG']=ps['CNT_INSTALMENT_FUTURE']/ps['CNT_INSTALMENT']
        ps['POS_CNT_INSTAL_PNDNG']=ps['CNT_INSTALMENT']-ps['CNT_INSTALMENT_FUTURE']
        ps['POS_DAYS_WTHT_TOLRNC']=ps['SK_DPD']+ps['SK_DPD_DEF']
        print("-----")
        id_cols,num_cols,_,_ = id_num_cat_feature(ps, text = False)
        id_cols.append('SK_ID_PREV')
        new_cols = ['POS_PERC_INSTL_PNDNG','POS_CNT_INSTAL_PNDNG','POS_DAYS_WTHT_TOLRNC']
        cols = list(set(num_cols) - set(id_cols) )
        #cols = ['PS_N_PERC_INSTL_PNDNG','PS_N_CNT_INSTAL_PNDNG','PS_N_DAYS_WTHT_TOLRNC']
        ps.replace([np.inf, -np.inf], 0, inplace=True)
        for i in cols:
            print('---')
            print(f'{i}')
            print(((ps[i].abs()).skew()))
            plt.hist(ps[i], bins=20)
            plt.xlabel(i)
            plt.show()
            print('')
            print(f'{i} :: root-transformed')
            print((np.sqrt(ps[i].abs())).skew())
            plt.hist(np.sqrt(ps[i].abs()), bins=20)
            plt.xlabel(i)
            plt.show()
            print('')
            print(f'{i} :: log-transformed')
            print((np.log(ps[i].abs() + 1)).skew())
            plt.hist(np.log(ps[i].abs() + 1), bins=20)
            plt.xlabel(i)
            plt.show()
```

Collinearity Reducer

1. Collinearity Reducer:

During Phase 2 exploratory data analysis, multicollinearity between input variables was shown to be prevalent in most of the data tables. Multicollinearity was further increased during the aggregation process as numeric input variables were proliferated into their mean, median, variance counterparts (amongst other aggregation types). This introduction of new (possibly) redundant variables to the ML algorithms significantly increases computation time and often reduces algorithm accuracy/effectiveness.

Used in conjunction with the DataframeSelector() transformer class, the CollinearityReducer() transformer class combats multicollinearity by removing the most (multi-) collinear columns which are least correlated to the target variable. The algorithm steps are described below:

1. Given input variable X and target variable y, calculate the correlation matrix
2. Pivot the correlation matrix into a long dataframe of correlation pairs and values (input variable 1, input variable 2 absolute correlation) and drop the following variable pairs:
 - any pair including the target variable
 - any pair with two of the same input variable
 - any pair with an absolute correlation value below a specified threshold value (default is 0.5)
3. For each variable pair, compare to see which variable is more correlated to the target variable. These input variables are given a 'win' while the input variable in the pair which is less correlated to the target variable is given a 'loss'.
4. Count the total 'wins' and 'losses' for each variable and drop from the original dataframe any variable with 0 wins.
5. Repeat steps 1 to 4 until there are no more input variable pairs with correlations above the threshold, no more input variables scoring 0 wins, or the specified maximum number of iterations has been reached.

The CollinearityReducer() thus applies a common multicollinearity solution - drop the variable which is least correlated to the target (Introduction to Statistical Learning Chapter 3). This is no simple task for human to perform on a high-dimensional dataset, but this class provides a mechanical solution so it does not have to be done manually. The transform method of this class creates a list of attribute names from the original dataframe which are to be kept - multicollinear classes culled by the CollinearityReducer() are not included in the output. This output list is then used as the input for the DataframeSelector() class in the pipeline. Thus, the CollinearityReducer() selects columns based only on the training data and there is no leakage from the validation or test data.

Notably, this algorithm is only applied to numerical variables as it is assumed there should naturally be some elevated degree of collinearity between one-hot-encoded categorical variables. Furthermore, the algorithm should be applied to numerical data which has been subjected to the same scaling and imputation strategy as applied in the actual pipeline.

Both the CollinearityReducer() and DataframeSelector() classes are shown below along with a basic example of their usage.

```

# transformer reduces the list of columns by a subset
class DataframeSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attribute_names):
        self.attribute_names = attribute_names
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.attribute_names].values

# transformer produces a reduced column list by collinearity reduction
class CollinearityReducer(BaseEstimator, TransformerMixin):

    """
    This class reduces features by measuring collinearity between the input variables and target.
    Works on numerical features based on the correlations between each variable pair.
    Of the variable pairs with absolute correlations above the threshold value the variables with the lowest target variable correlation are dropped from the input X.
    Repeat until no more collinear pairs with absolute correlations above the threshold or max_iter.
    """

    Inputs:
        X (numpy array) - input variables
        y (numpy array) - target variable
        attribute_names (list) - column names of the input variables (from original dataframe order)
        threshold (int) - the absolute correlation threshold above which variable pairs are subject to the 'correlation competition'
        max_iter (int) - the maximum number of iterations to cut off the algorithm

    Output:
        list - attribute_names to be used by DataframeSelector class
    ...

    def __init__(self, attribute_names, threshold=0.5, max_iter=None):
        self.attribute_names = attribute_names
        self.threshold = threshold
        self.max_iter = max_iter

    def fit(self, X, y):
        return self

    def transform(self, X, y=None):
        dataframe = pd.concat([y, pd.DataFrame(X)], axis=1)

        i = 0
        while i <= self.max_iter:

            # read-in and assign columns
            # gets correlation matrix between variables and pivots to a longer df
            # identify target variable
            # drop same-name and target correlations pairs

            df = dataframe
            features = df.iloc[:,1:].columns
            target_name = df.iloc[:,0].name

            df = pd.melt(abs(df.corr()).reset_index(), id_vars='index', value_vars=features)
            targets = df[df['index']==target_name]
            df = df[(df['index'] != target_name) & (df['index'] != target_name) & (df['variable'] != target_name) & (df['variable'] != target_name)]

            # combine the correlated variables into ordered pairs
            # aggregate the max correlation and sort pairs
            # split out the variables from the pair
            # join the target variable correlations for each variable pair

            df['joined'] = df[['index', 'variable']].apply(lambda row: ':'.join(np.sort(row.values.astype(str))), axis=1)

            df = df.groupby('joined', as_index=False) \
                .agg({'value':'max'}) \
                .sort_values(by='value', ascending=False)

            df[['var_1','var_2']] = df['joined'].str.split(":",expand=True).astype(int)

            df = df.merge(targets, how='left', left_on='var_1', right_on='variable') \
                .merge(targets, how='left', left_on='var_2', right_on='variable')
            df.rename(columns = {'value_x':'var_pair_corr', 'value_y':'var_1_target_corr', 'value': 'var_2_target_corr'}, inplace = True)

            # Take only variable pairs with a correlation greater than threshold
            # determine which variable has a higher correlation with the target.
            # The higher of the two gets marked as a win
            # While the other gets marked as a loss
            # the wins and losses for each variable are then grouped and summed

            exceeds = df[df['var_pair_corr']>self.threshold]

            # break if none above threshold
            if len(exceeds['var_pair_corr'])==0:
                break

            # "correlation competition"
            exceeds[['var_1.win']] = exceeds.apply(lambda row: 1 if row['var_1_target_corr'] > row['var_2_target_corr'] else 0, axis=1)
            exceeds[['var_1.loss']] = exceeds.apply(lambda row: 1 if row['var_2_target_corr'] > row['var_1_target_corr'] else 0, axis=1)
            exceeds[['var_2.win']] = exceeds.apply(lambda row: 1 if row['var_1_target_corr'] < row['var_2_target_corr'] else 0, axis=1)
            exceeds[['var_2.loss']] = exceeds.apply(lambda row: 1 if row['var_2_target_corr'] < row['var_1_target_corr'] else 0, axis=1)

            # aggregate scores
            var1 = exceeds[['var_1', 'var_1.win', 'var_1.loss']].groupby('var_1', as_index=False) \
                .agg({'var_1.win':'sum', 'var_1.loss':'sum'})
            var1.rename(columns = {'var_1':'var', 'var_1.win':'win', 'var_1.loss':'loss'}, inplace=True)

            var2 = exceeds[['var_2', 'var_2.win', 'var_2.loss']].groupby('var_2', as_index=False) \
                .agg({'var_2.win':'sum', 'var_2.loss':'sum'})
            var2.rename(columns = {'var_2':'var', 'var_2.win':'win', 'var_2.loss':'loss'}, inplace=True)

            corrcomps = pd.concat([var1,var2], axis=0).groupby('var', as_index=False) \
                .agg({'win':'sum', 'loss':'sum'})

            # drop variables which had 0 wins
            # IE collinear variables which were always least related to the target
            dropvars = corrcomps[corrcomps['win']==0]['var']

            dataframe = dataframe.drop(dropvars, axis=1)

            i += 1

        X = [self.attribute_names[col] for col in dataframe.columns]

        return X

```

Feature Tracking

For the purpose of knowing feature origin or transformation, we have taken steps to rename the feature such that we can readily know everything about the feature.

Here is how renames columns will look like (Consider Previous application table.):

- All columns will be renamed to PA_0_XX: where PA_0_ is a prefix and XX is the actual column name.
- Revert renaming of ID columns. Because ID's are used across tables for comparison.
- If any transformation is done on features:
 - Log : PA_0_L_XX
 - SQRT: PA_0_R_XX
- Any New feature created will be named as : PA_N_XX.
- All New Features with transformations:
 - Log : PA_N_L_XX
 - SQRT: PA_N_R_XX
- Now if we have a attribute months balance which is present in both Bureau and POS Cash balance, they will be renamed to BR_0_MONTHS_BALANCE and PA_0_MONTHS_BALANCE respectively. And if we do sqrt transformation, we will add "_R after the 0 to show that they have root transformation.

Using string manipulation, we can readily create multiple lists to include or exclude such elements from the pipeline.

Benefits:

1. There are features which are common between tables, like monthly balance. This will help us know monthly balance from which table is useful for prediction.
2. Based on the column name, we know the origin, transformation, new features created just by looking at the column name. We can quickly create list each like an example given below:

All new cols ip_new_cols = [col for col in ip.columns if "IP_N" in col]

Cols on which transformation took place ip_unt_cols = [col for col in ip.columns if "IP_O_DAYS_ENTRY_PAYMENT" in col]

Transformed cols ip_r_cols = [col for col in ip.columns if "IP_N_R" in col] ip_l_cols = [col for col in ip.columns if "IP_N_L" in col]

OHE Discrepancy between Train and Test Sets

It was observed when trying to train the neural network model that there are some missing columns between the one-hot-encoded results of `application_train.csv` and `application_test.csv`. This indicated missing category variables in the categorical columns of `application_test`. Below you can see the features and their missing attributes.

```
app_test = datasets_test['application_test']
app_train = pd.read_csv(os.getcwd() + DATA_DIR +
    'application_train.csv')
app_train = reduce_mem_usage(app_train)

cat_cols = ['CODE_GENDER', 'NAME_INCOME_TYPE',
    'NAME_FAMILY_STATUS']
# cat_cols = feat_cat

print(app_train[cat_cols].apply(lambda col:
    col.unique()).to_markdown())
print(' ')
print(app_test[cat_cols].apply(lambda col:
    col.unique()).to_markdown())
```

[OUT]:

0	
CODE_GENDER	['M', 'F', 'XNA'] Categories (3, object): ['F', 'M', 'XNA']
NAME_INCOME_TYPE	['Working', 'State servant', 'Commercial associate', 'Pensioner', 'Unemployed', 'Student', 'Businessman', 'Maternity leave'] Categories (8, object): ['Businessman', 'Commercial associate', 'Maternity leave', 'Pensioner', 'State servant', 'Student', 'Unemployed', 'Working']
NAME_FAMILY_STATUS	['Single / not married', 'Married', 'Civil marriage', 'Widow', 'Separated', 'Unknown'] Categories (6, object): ['Civil marriage', 'Married', 'Separated', 'Single / not married', 'Unknown', 'Widow']
1	
CODE_GENDER	['F', 'M'] Categories (2, object): ['F', 'M']
NAME_INCOME_TYPE	['Working', 'State servant', 'Pensioner', 'Commercial associate', 'Businessman', 'Student', 'Unemployed'] Categories (7, object): ['Businessman', 'Commercial associate', 'Pensioner', 'State servant', 'Student', 'Unemployed', 'Working']
NAME_FAMILY_STATUS	['Married', 'Single / not married', 'Civil marriage', 'Widow', 'Separated'] Categories (5, object): ['Civil marriage', 'Married', 'Separated', 'Single / not married', 'Widow']

This was resolved by removing any rows with these missing values from the training data immediately after data denormalization. This is seen in the code below.

```
# denormalize and clean text
for ds_name in datasets_test:
    if ds_name == 'application_test':
        test_data =
datasets_test['application_test'].replace(to_replace='\s+', value='_', regex=True) \
.replace(to_replace='-', value='_', regex=True) \
.replace(to_replace='\\/', value='_', regex=True) \
.replace(to_replace='\\(', value=' ', regex=True) \
.replace(to_replace='\\)', value=' ', regex=True) \
.replace(to_replace='\\:', value=' ', regex=True) \
.replace(to_replace='\\,', value=' ', regex=True)
    else:
        test_data = test_data.merge(datasets_test[ds_name],
on='SK_ID_CURR', how='left')

test_data =
test_data.loc[:,~test_data.columns.str.startswith('Unnamed:')]
test_data =
test_data.loc[:,~test_data.columns.str.startswith('SK_ID_PREV')]

### HANDLE application_test MISSING COLUMN VARIABLES
test_data = test_data[
    (test_data.CODE_GENDER != 'XNA') & \
    (test_data.NAME_INCOME_TYPE != 'Maternity leave') & \
    (test_data.NAME_FAMILY_STATUS != 'Unknown')
]
```

Bureau Balance

1. This function is specifically for Bureau Balance table. Below are the intial pre-processing steps done before passing this table into the pipeline.

- Since this table has only 2 features, no features are dropped or created for this table.
- Take absolute of the months balance attribute. which was provided as negative values, as it is relative to application date.
- Any column or row with more than 70% of its data as null will be deleted from the dataframe, as the threshold is set to .7.
- Once processed, store the transformed data into a csv file. Benefit of this is that data can be passed ed csv file. directly to model for merging into application train/test

In [16]:

```
def bbal_eda(df):
    print("bureau_bal :: EDA and transformation")
    print('')
    bbal = df
    #bureau balance table contains all the data so no need to drop any column
    #Adding new features, take the abs for the monthly balance attribute.
    bbal['MONTHS_BALANCE'] = bbal['MONTHS_BALANCE'].abs()
    return (eda_transformation(bbal,0))
```

In [17]:

```
bbal = datasets['bureau_balance']
bbal = bbal_eda(bbal)
#datasets_transformed['bureau_balance'] = bbal
```

```
bureau_bal :: EDA and transformation

-----
# of ID's: 1
ID's:
['SK_ID_BUREAU']

-----
# All features: 2
All features:
['STATUS', 'MONTHS_BALANCE']

Missing data:
      Total  Percent
STATUS          0      0.0
MONTHS_BALANCE    0      0.0

-----
# of Numerical features: 2
Numerical features:
['SK_ID_BUREAU', 'MONTHS_BALANCE']

Numerical Statistical Summary:

      SK_ID_BUREAU  MONTHS_BALANCE
count  2.729992e+07  2.729992e+07
mean   6.036297e+06  3.074169e+01
std    4.923489e+05  2.386451e+01
min    5.001709e+06  0.000000e+00
25%   5.730933e+06  1.100000e+01
50%   6.070821e+06  2.500000e+01
75%   6.431951e+06  4.600000e+01
max   6.842888e+06  9.600000e+01

-----
# of Categorical features: 1
Categorical features:
['STATUS']

Categorical Statistical Summary:

Categories:

      STATUS
0        C
1        0
2        X
3        1
4        2
5        3
6        5
7        4

-----
# of OHE categorical features: 8
OHE Categorical features: ['STATUS_X', 'STATUS_3', 'STATUS_2', 'STATUS_4', 'STATUS_5', 'STATUS_C', 'STATUS_1', 'STATUS_0']
-----
```

```
df.shape: (27299925, 10)
```

```
Aggregated Features:
```

```
df[['STATUS_X', 'STATUS_3', 'STATUS_2', 'STATUS_4', 'MONTHS_BALANCE', 'STATUS_5', 'STATUS_C', 'STATUS_1', 'STATUS_0']][0:5]:
```

```
    STATUS_X  STATUS_3  STATUS_2  STATUS_4  MONTHS_BALANCE  STATUS_5  STATUS_C \
```

```
0          0          0          0          0          0          0          0
```

```
1          1          0          0          0          1          0          0
```

```
1          0          0          0          0          2          0          0
```

```
2          0          0          0          0          3          0          0
```

```
1          0          0          0          0          4          0          0
```

```
3          0          0          0          0          1          0          0
```

```
4          0          0          0          0          0          0          0
```

```
1          0          0          0          0          0          0          0
```

```
-----
```

```
Aggregated Features:
```

```
MONTHS_BALANCE_count
```

```
STATUS_0_mean
```

```
STATUS_0_median
```

```
STATUS_0_var
```

```
STATUS_1_mean
```

```
STATUS_1_median
```

```
STATUS_1_var
```

```
STATUS_2_mean
```

```
STATUS_2_median
```

```
STATUS_2_var
```

```
STATUS_3_mean
```

```
STATUS_3_median
```

```
STATUS_3_var
```

```
STATUS_4_mean
```

```
STATUS_4_median
```

```
STATUS_4_var
```

```
STATUS_5_mean
```

```
STATUS_5_median
```

```
STATUS_5_var
```

```
STATUS_C_mean
```

```
STATUS_C_median
```

```
STATUS_C_var
```

```
STATUS_X_mean
```

```
STATUS_X_median
```

```
STATUS_X_var
```

```
-----
```

```
Aggregated Data:
```

	count	mean	std	min	25%	\
STATUS_0_mean	817395.0	3.994569e-01	0.347193	0.0	0.086957	
STATUS_1_median	817395.0	1.404462e-03	0.035453	0.0	0.000000	
STATUS_C_var	811206.0	9.398683e-02	0.102171	0.0	0.000000	
STATUS_X_median	817395.0	1.815352e-01	0.380877	0.0	0.000000	

STATUS_C_mean	817395.0	3.710395e-01	0.379741	0.0	0.000000
STATUS_C_median	817395.0	4.211721e-01	0.490946	0.0	0.000000
STATUS_2_var	811206.0	6.811640e-04	0.007217	0.0	0.000000
STATUS_X_var	811206.0	6.244586e-02	0.096936	0.0	0.000000
MONTHS_BALANCE_count	817395.0	3.339869e+01	25.794666	1.0	13.000000
STATUS_5_var	811206.0	7.843847e-04	0.011837	0.0	0.000000
STATUS_4_mean	817395.0	1.608014e-04	0.002889	0.0	0.000000
STATUS_3_median	817395.0	0.000000e+00	0.000000	0.0	0.000000
STATUS_3_mean	817395.0	2.469056e-04	0.003800	0.0	0.000000
STATUS_5_mean	817395.0	1.492400e-03	0.027131	0.0	0.000000
STATUS_4_var	811206.0	1.601024e-04	0.002804	0.0	0.000000
STATUS_5_median	817395.0	1.025208e-03	0.031811	0.0	0.000000
STATUS_1_mean	817395.0	1.116493e-02	0.046621	0.0	0.000000
STATUS_X_mean	817395.0	2.157315e-01	0.337008	0.0	0.000000
STATUS_3_var	811206.0	2.446794e-04	0.003681	0.0	0.000000
STATUS_0_median	817395.0	3.616085e-01	0.474283	0.0	0.000000
STATUS_2_median	817395.0	8.563791e-06	0.002594	0.0	0.000000
STATUS_0_var	811206.0	1.298257e-01	0.103613	0.0	0.013514
STATUS_2_mean	817395.0	7.070824e-04	0.008025	0.0	0.000000
STATUS_1_var	811206.0	9.649919e-03	0.035212	0.0	0.000000
STATUS_4_median	817395.0	6.116994e-07	0.000553	0.0	0.000000

	50%	75%	max
STATUS_0_mean	0.307692	0.705882	1.000000
STATUS_1_median	0.000000	0.000000	1.000000
STATUS_C_var	0.058634	0.194444	0.500000
STATUS_X_median	0.000000	0.000000	1.000000
STATUS_C_mean	0.285714	0.764045	1.000000
STATUS_C_median	0.000000	1.000000	1.000000
STATUS_2_var	0.000000	0.000000	0.333333
STATUS_X_var	0.000000	0.100000	0.500000
MONTHS_BALANCE_count	26.000000	48.000000	97.000000
STATUS_5_var	0.000000	0.000000	0.333333
STATUS_4_mean	0.000000	0.000000	0.500000
STATUS_3_median	0.000000	0.000000	0.000000
STATUS_3_mean	0.000000	0.000000	0.400000
STATUS_5_mean	0.000000	0.000000	1.000000
STATUS_4_var	0.000000	0.000000	0.333333
STATUS_5_median	0.000000	0.000000	1.000000
STATUS_1_mean	0.000000	0.000000	1.000000
STATUS_X_mean	0.024390	0.304348	1.000000
STATUS_3_var	0.000000	0.000000	0.333333
STATUS_0_median	0.000000	1.000000	1.000000
STATUS_2_median	0.000000	0.000000	1.000000
STATUS_0_var	0.128571	0.226129	0.500000
STATUS_2_mean	0.000000	0.000000	0.714286
STATUS_1_var	0.000000	0.000000	0.500000
STATUS_4_median	0.000000	0.000000	0.500000

Bureau

1. This function is specifically for Bureau table. Below are the intial pre-processing steps done before passing this table into the pipeline.

- For columns with DAYS in name, there are two with negative values, we will take the absolute for those.
- Data from Buereau balance table is rolled up in Bureau table befor any EDA. This will enable using all the feautes from buereau balance table as well before rolling up all data to main table, i.e. application train.
- while doing left join, we updated OHE column names to more readale forms by removing any space or spcl charater to "_" which is widely used in column names.
- Any column or row with more than 70% of its data as null will be deleted from the dataframe, as the threshold is set to .7.
- Once processed, store the transformed data into a csv file. Benefit of this is that data can be passed ed csv file. directly to model for merging into application train/test table. We do not have to repeatedly perform expensive EDA/ETL/Transformation.
- **Features transformation:** Below are the features which are transformed. (N_R: New feature with root transformation. N_L: New feature with log transformation.

```
bureau['BR_N_R_MONTHS_BALANCE_count'] =  
np.sqrt(bureau['BR_0_MONTHS_BALANCE_count'])  
  
bureau['BR_N_R_DAYS_CREDIT'] =  
np.sqrt(bureau['BR_0_DAYS_CREDIT'].abs())  
  
bureau['BR_N_R_DAYS_ENDDATE_FACT'] =  
np.sqrt(bureau['BR_0_DAYS_ENDDATE_FACT'].abs())  
  
bureau['BR_N_L_CREDIT_DAY_OVERDUE'] =  
np.log(bureau['BR_0_CREDIT_DAY_OVERDUE'].abs() + 1)
```

In [27]:

```
def bureau_eda(df):
    bureau = df
    drop_list_bureau = []

    #Adding new features
    #bureau['MONTHS_BALANCE'] = bureau['MONTHS_BALANCE'].abs()
    for c in [co for co in bureau.columns if 'DAYS' in co]:
        bureau[c] = bureau[c].replace({365243.0: np.nan})
        bureau[c] = bureau[c].abs()
    # Drop elements in drop list
    threshold = 0.7

    #Dropping rows with missing value rate higher than threshold
    bureau = bureau.loc[bureau.isnull().mean(axis=1) < threshold]

    return (eda_transformation(bureau,1))
```

In [28]:

```
bureau = datasets['bureau']

# rollup bureau_bal
# gets rid of the unwanted characters in categorical columns entries - make them all lowercase
bureau = bureau.merge(bbal, on='SK_ID_BUREAU', how='left') \
    .replace(to_replace='\s+', value='_', regex=True) \
    .replace(to_replace='-', value='_', regex=True) \
    .replace(to_replace='\/', value='_', regex=True) \
    .replace(to_replace='\(', value=' ', regex=True) \
    .replace(to_replace='\)', value=' ', regex=True) \
    .replace(to_replace='\: ', value=' ', regex=True) \
    .replace(to_replace='\\', value=' ', regex=True) \
    .drop('SK_ID_BUREAU', axis=1)
```

Feature Transformation Viz

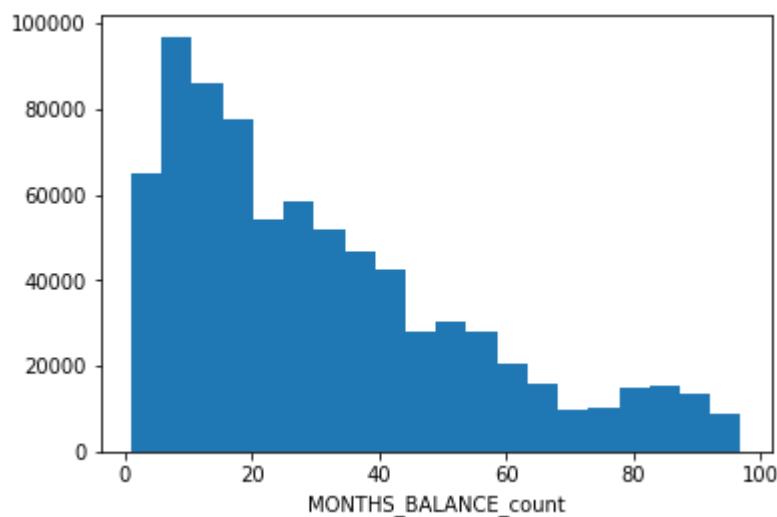
MONTHS_BALANCE, DAYS_CREDIT, CREDIT_DAY_OVERDUE

In [31]:

```
for i in ['MONTHS_BALANCE_count', 'DAYS_CREDIT', 'CREDIT_DAY_OVERDUE', 'DAY']:
    print('---')
    print(f'{i}')
    print(((bureau[i].abs())).skew())
    plt.hist(bureau[i], bins=20)
    plt.xlabel(i)
    plt.show()
    print('')
    print(f'{i} :: root-transformed')
    print((np.sqrt(bureau[i].abs())).skew())
    plt.hist(np.sqrt(bureau[i].abs()), bins=20)
    plt.xlabel(i)
    plt.show()
    print('')
    print(f'{i} :: log-transformed')
    print((np.log(bureau[i].abs() + 1)).skew())
    plt.hist(np.log(bureau[i].abs() + 1), bins=20)
    plt.xlabel(i)
    plt.show()
```

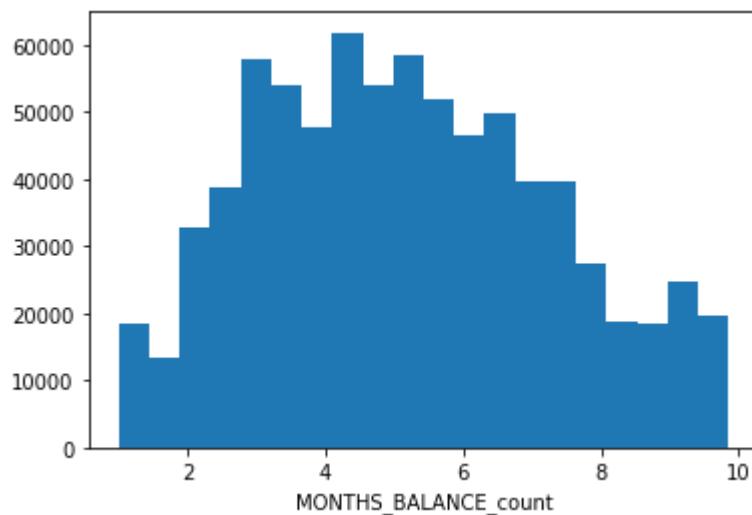
MONTHS_BALANCE_count

0.9154039138929194



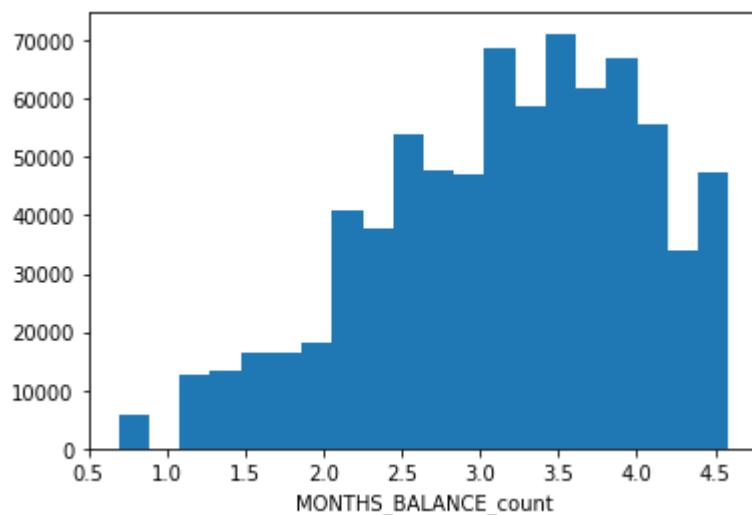
MONTHS_BALANCE_count :: root-transformed

0.24441137316189374

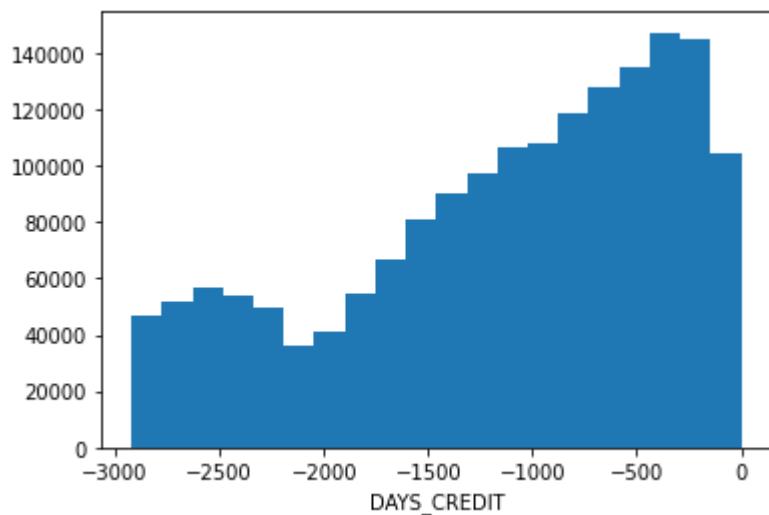


MONTHS_BALANCE_count :: log-transformed

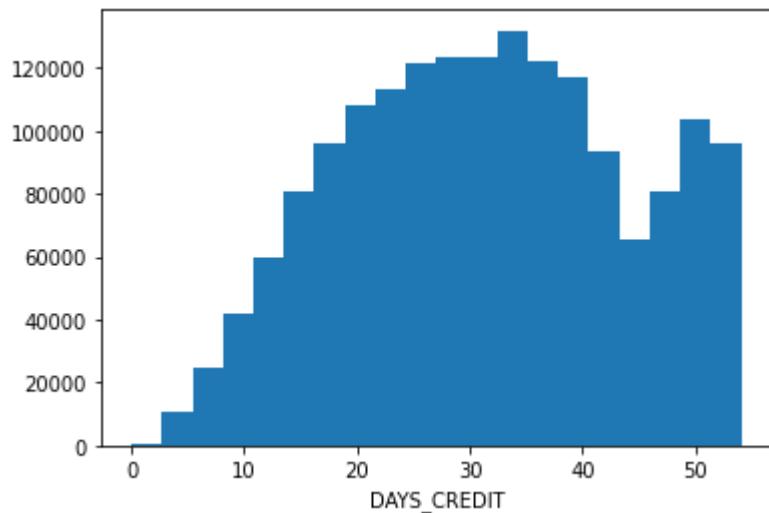
-0.5100152267601524



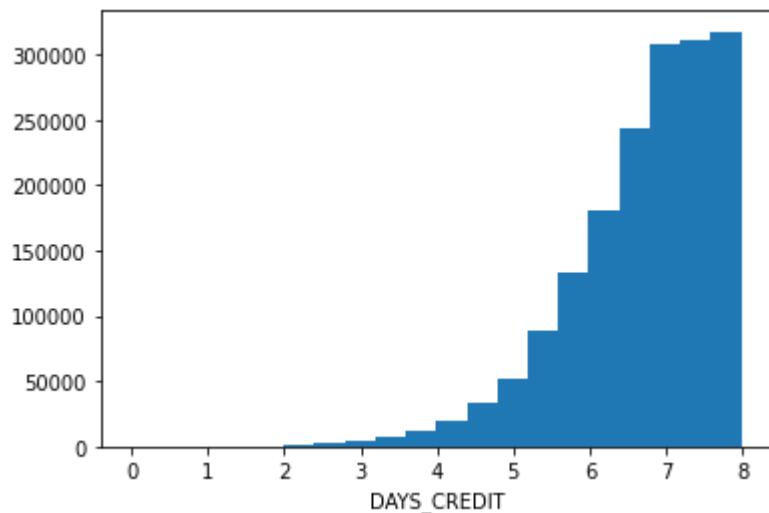
DAY_S_CREDIT
0.5823490530504081



DAY_S_CREDIT :: root-transformed
-0.024596608125142392

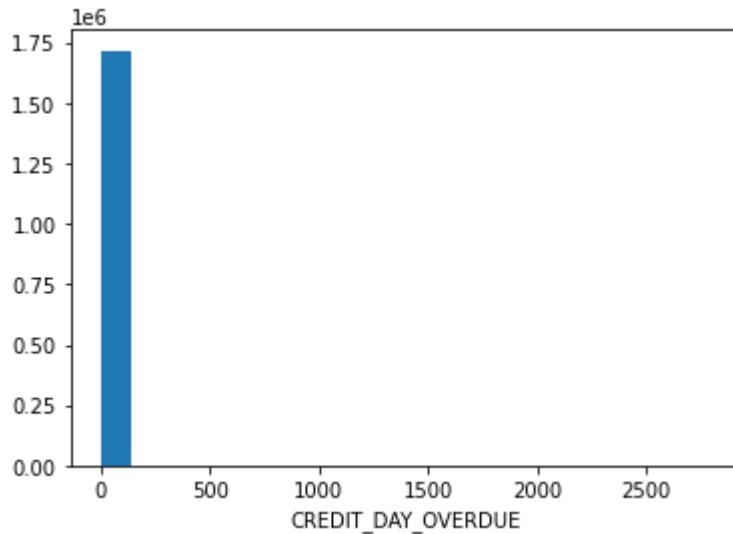


DAY_S_CREDIT :: log-transformed
-1.1083716870401357



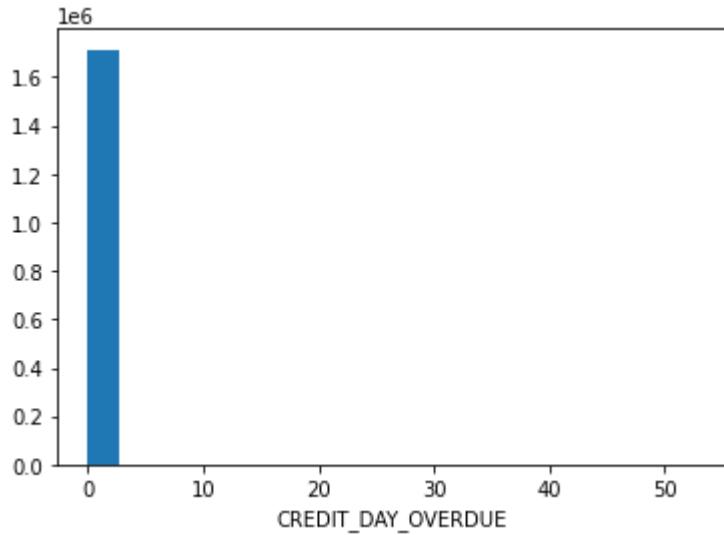
CREDIT_DAY_OVERDUE

55.93100542458554



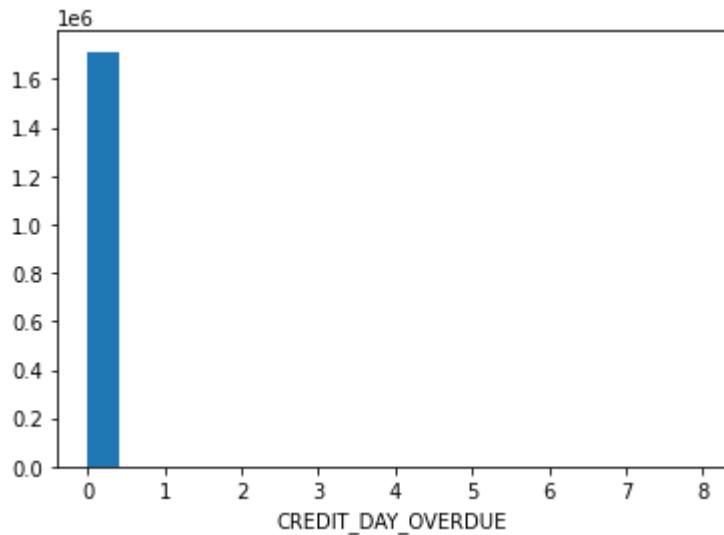
CREDIT_DAY_OVERDUE :: root-transformed

42.29872284059052

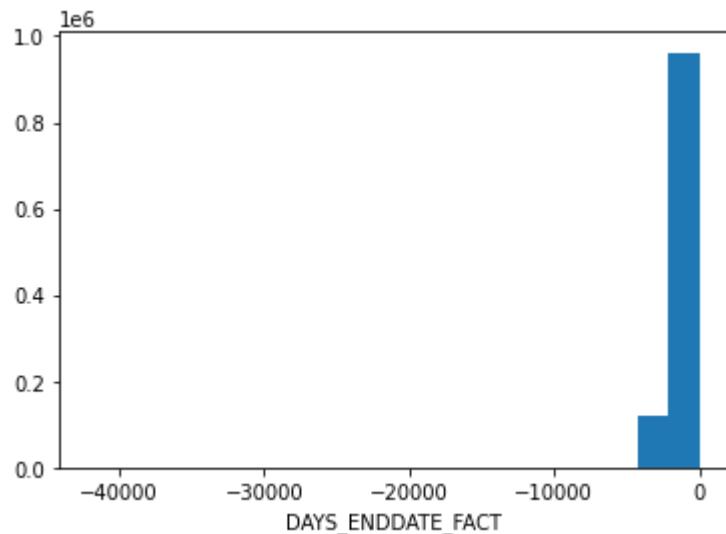


CREDIT_DAY_OVERDUE :: log-transformed

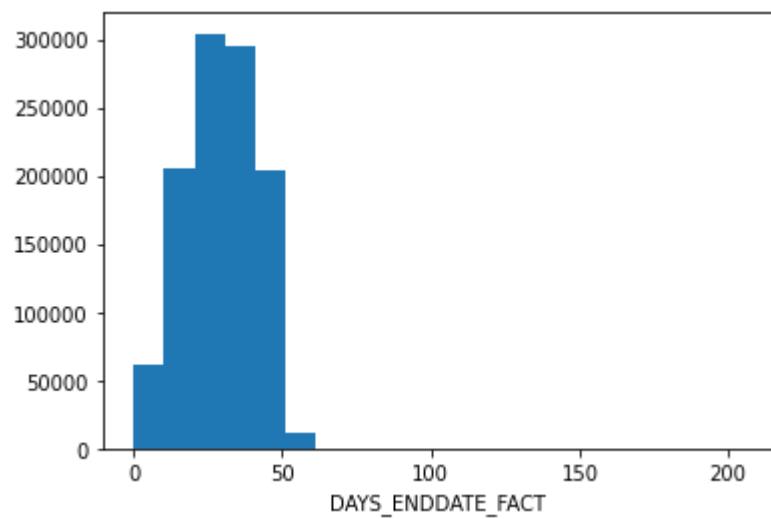
25.132997881098458



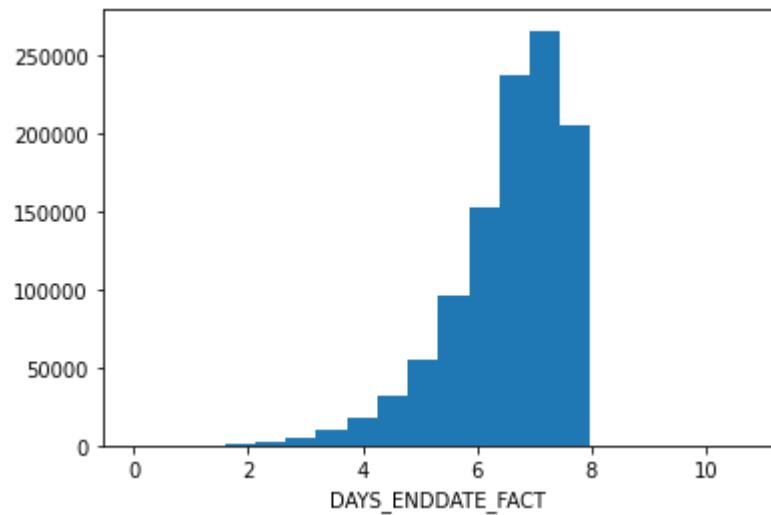
DAY_S_ENDDATE_FACT
0.7747538315821675



DAY_S_ENDDATE_FACT :: root-transformed
-0.06124530079580782



DAY_S_ENDDATE_FACT :: log-transformed
-1.2985975629755915



Features Engineering

Please refer to section 6.2.1 and 6.2.3 for details on feature renaming and transformation.

In [32]:

```
bureau.columns = ['BR_0_' + col for col in bureau.columns]
bureau['SK_ID_CURR'] = bureau['BR_0_SK_ID_CURR']
bureau.drop(['BR_0_SK_ID_CURR'],axis=1,inplace=True)

bureau['BR_N_R_MONTHS_BALANCE_count'] = np.sqrt(bureau['BR_0_MONTHS_BALANCE'])
bureau['BR_N_R_DAYS_CREDIT'] = np.sqrt(bureau['BR_0_DAYS_CREDIT'].abs())
bureau['BR_N_R_DAYS_ENDDATE_FACT'] = np.sqrt(bureau['BR_0_DAYS_ENDDATE_FACT'])
bureau['BR_N_L_CREDIT_DAY_OVERDUE'] = np.log(bureau['BR_0_CREDIT_DAY_OVERDUE'])

bureau['BR_O_R_MONTHS_BALANCE_count'] = bureau['BR_0_MONTHS_BALANCE_count']
bureau.drop(['BR_0_MONTHS_BALANCE_count'],axis=1,inplace=True)

bureau['BR_O_R_DAYS_CREDIT'] = bureau['BR_0_DAYS_CREDIT']
bureau.drop(['BR_0_DAYS_CREDIT'],axis=1,inplace=True)

bureau['BR_O_R_DAYS_ENDDATE_FACT'] = bureau['BR_0_DAYS_ENDDATE_FACT']
bureau.drop(['BR_0_DAYS_ENDDATE_FACT'],axis=1,inplace=True)

bureau['BR_O_L_CREDIT_DAY_OVERDUE'] = bureau['BR_0_CREDIT_DAY_OVERDUE']
bureau.drop(['BR_0_CREDIT_DAY_OVERDUE'],axis=1,inplace=True)
```

In [33]:

```
bureau_trans_untrans = bureau_edu(bureau).drop('index', axis=1)
```

```

-----
# of ID's: 1
ID's:
['SK_ID_CURR']

-----
# All features: 44
All features:
['BR_O_AMT_CREDIT_SUM_LIMIT', 'BR_O_R_DAYS_ENDDATE_FACT', 'BR_O_AMT_CREDIT_M
AX_OVERDUE', 'BR_O_STATUS_5_var', 'BR_N_R_DAYS_ENDDATE_FACT', 'BR_O_STATUS_X
_var', 'BR_O_STATUS_C_mean', 'BR_O_R_MONTHS_BALANCE_count', 'BR_O_STATUS_4_m
edian', 'BR_O_STATUS_X_mean', 'BR_O_STATUS_3_mean', 'BR_O_AMT_CREDIT_SUM_DEB
T', 'BR_N_R_MONTHS_BALANCE_count', 'BR_O_DAYS_CREDIT_ENDDATE', 'BR_O_CREDIT_
CURRENCY', 'BR_O_STATUS_1_var', 'BR_O_STATUS_0_median', 'BR_O_STATUS_C_var',
'BR_O_STATUS_5_median', 'BR_O_STATUS_5_mean', 'BR_O_AMT_CREDIT_SUM', 'BR_O_S
TATUS_2_mean', 'BR_O_CREDIT_ACTIVE', 'BR_O_STATUS_4_var', 'BR_O_CREDIT_TYPE
', 'BR_O_STATUS_0_mean', 'BR_O_L_CREDIT_DAY_OVERDUE', 'BR_O_STATUS_1_median
', 'BR_O_STATUS_4_mean', 'BR_O_STATUS_C_median', 'BR_O_STATUS_X_median', 'BR
_O_AMT_ANNUITY', 'BR_O_STATUS_0_var', 'BR_O_STATUS_1_mean', 'BR_O_DAYS_CREDI
T_UPDATE', 'BR_O_R_DAYS_CREDIT', 'BR_N_R_DAYS_CREDIT', 'BR_O_AMT_CREDIT_SUM_
OVERDUE', 'BR_O_STATUS_3_var', 'BR_O_CNT_CREDIT_PROLONG', 'BR_O_STATUS_2_var
', 'BR_N_L_CREDIT_DAY_OVERDUE', 'BR_O_STATUS_3_median', 'BR_O_STATUS_2_media
n']

```

Missing data:

	Total	Percent
BR_O_AMT_ANNUITY	1189119	70.833681
BR_O_AMT_CREDIT_MAX_OVERDUE	1086808	64.739198
BR_O_STATUS_1_var	910298	54.224815
BR_O_STATUS_5_var	910298	54.224815
BR_O_STATUS_X_var	910298	54.224815
BR_O_STATUS_3_var	910298	54.224815
BR_O_STATUS_C_var	910298	54.224815
BR_O_STATUS_0_var	910298	54.224815
BR_O_STATUS_4_var	910298	54.224815
BR_O_STATUS_2_var	910298	54.224815
BR_O_STATUS_1_mean	904394	53.873124
BR_O_STATUS_5_mean	904394	53.873124
BR_O_STATUS_0_mean	904394	53.873124
BR_O_STATUS_2_mean	904394	53.873124
BR_O_STATUS_1_median	904394	53.873124
BR_O_STATUS_4_mean	904394	53.873124
BR_O_STATUS_5_median	904394	53.873124
BR_O_STATUS_2_median	904394	53.873124
BR_O_STATUS_0_median	904394	53.873124
BR_O_STATUS_X_median	904394	53.873124
BR_N_R_MONTHS_BALANCE_count	904394	53.873124
BR_O_STATUS_3_median	904394	53.873124
BR_O_STATUS_3_mean	904394	53.873124
BR_O_STATUS_X_mean	904394	53.873124
BR_O_STATUS_4_median	904394	53.873124
BR_O_R_MONTHS_BALANCE_count	904394	53.873124
BR_O_STATUS_C_mean	904394	53.873124
BR_O_STATUS_C_median	904394	53.873124
BR_O_R_DAYS_ENDDATE_FACT	595973	35.501040
BR_N_R_DAYS_ENDDATE_FACT	595973	35.501040
BR_O_AMT_CREDIT_SUM_LIMIT	554100	33.006741
BR_O_AMT_CREDIT_SUM_DEBT	226702	13.504231
BR_O_DAYS_CREDIT_ENDDATE	90253	5.376209

BR_O_AMT_CREDIT_SUM	8	0.000477
BR_O_L_CREDIT_DAY_OVERDUE	0	0.000000
BR_O_CREDIT_TYPE	0	0.000000
BR_O_DAYS_CREDIT_UPDATE	0	0.000000
BR_O_R_DAYS_CREDIT	0	0.000000
BR_N_R_DAYS_CREDIT	0	0.000000
BR_O_AMT_CREDIT_SUM_OVERDUE	0	0.000000
BR_O_CREDIT_CURRENCY	0	0.000000
BR_O_CNT_CREDIT_PROLONG	0	0.000000
BR_N_L_CREDIT_DAY_OVERDUE	0	0.000000
BR_O_CREDIT_ACTIVE	0	0.000000

# of Numerical features:	42	
Numerical features:		
['BR_O_DAYS_CREDIT_ENDDATE', 'BR_O_AMT_CREDIT_MAX_OVERDUE', 'BR_O_CNT_CREDIT_PROLONG', 'BR_O_AMT_CREDIT_SUM', 'BR_O_AMT_CREDIT_SUM_DEBT', 'BR_O_AMT_CREDIT_SUM_LIMIT', 'BR_O_AMT_CREDIT_SUM_OVERDUE', 'BR_O_DAYS_CREDIT_UPDATE', 'BR_O_AMT_ANNUITY', 'BR_O_STATUS_X_median', 'BR_O_STATUS_X_mean', 'BR_O_STATUS_X_var', 'BR_O_STATUS_3_median', 'BR_O_STATUS_3_mean', 'BR_O_STATUS_3_var', 'BR_O_STATUS_2_median', 'BR_O_STATUS_2_mean', 'BR_O_STATUS_2_var', 'BR_O_STATUS_4_median', 'BR_O_STATUS_4_mean', 'BR_O_STATUS_4_var', 'BR_O_STATUS_5_median', 'BR_O_STATUS_5_mean', 'BR_O_STATUS_5_var', 'BR_O_STATUS_C_median', 'BR_O_STATUS_C_mean', 'BR_O_STATUS_C_var', 'BR_O_STATUS_1_median', 'BR_O_STATUS_1_mean', 'BR_O_STATUS_1_var', 'BR_O_STATUS_0_median', 'BR_O_STATUS_0_mean', 'BR_O_STATUS_0_var', 'SK_ID_CURR', 'BR_N_R_MONTHS_BALANCE_count', 'BR_N_R_DAYS_CREDIT', 'BR_N_R_DAYS_ENDDATE_FACT', 'BR_N_L_CREDIT_DAY_OVERDUE', 'BR_O_R_MONTHS_BALANCE_count', 'BR_O_R_DAYS_CREDIT', 'BR_O_R_DAYS_ENDDATE_FACT', 'BR_O_L_CREDIT_DAY_OVERDUE']		
Numerical Statistical Summary:		
	BR_O_DAYS_CREDIT_ENDDATE	BR_O_AMT_CREDIT_MAX_OVERDUE \
count	1.588495e+06	5.919400e+05
mean	1.825047e+03	3.825418e+03
std	4.705750e+03	2.060316e+05
min	0.000000e+00	0.000000e+00
25%	3.820000e+02	0.000000e+00
50%	8.620000e+02	0.000000e+00
75%	1.503000e+03	0.000000e+00
max	4.206000e+04	1.159872e+08
	BR_O_CNT_CREDIT_PROLONG	BR_O_AMT_CREDIT_SUM BR_O_AMT_CREDIT_SUM_DEB
T \ count	1.678748e+06	1.678740e+06 1.452046e+0
6		
mean	6.554289e-03	3.507372e+05 1.354165e+0
5		
std	9.729295e-02	1.139831e+06 6.703209e+0
5		
min	0.000000e+00	0.000000e+00 -4.705600e+0
6		
25%	0.000000e+00	5.044500e+04 0.000000e+0
0		
50%	0.000000e+00	1.234570e+05 0.000000e+0
0		
75%	0.000000e+00	3.059100e+05 3.924000e+0
4		
max	9.000000e+00	5.850000e+08 1.701000e+0

	BR_O_AMT_CREDIT_SUM_LIMIT	BR_O_AMT_CREDIT_SUM_OVERDUE	\
count	1.124648e+06	1.678748e+06	
mean	6.229515e+03	3.572584e+01	
std	4.503203e+04	5.891416e+03	
min	-5.864061e+05	0.000000e+00	
25%	0.000000e+00	0.000000e+00	
50%	0.000000e+00	0.000000e+00	
75%	0.000000e+00	0.000000e+00	
max	4.705600e+06	3.756681e+06	
	BR_O_DAYS_CREDIT_UPDATE	BR_O_AMT_ANNUITY	BR_O_STATUS_X_median
\			
count	1.678748e+06	4.896290e+05	774354.000000
mean	5.986786e+02	1.571244e+04	0.181671
std	7.212737e+02	3.258296e+05	0.380827
min	0.000000e+00	0.000000e+00	0.000000
25%	3.400000e+01	0.000000e+00	0.000000
50%	4.070000e+02	0.000000e+00	0.000000
75%	9.130000e+02	1.350000e+04	0.000000
max	4.194700e+04	1.184534e+08	1.000000
	BR_O_STATUS_0_var	SK_ID_CURR	BR_N_R_MONTHS_BALANCE_count
count	768450.000000	1.678748e+06	774354.000000
mean	0.133522	2.781912e+05	5.172397
std	0.103150	1.029396e+05	2.114712
min	0.000000	1.000010e+05	1.000000
25%	0.027027	1.888270e+05	3.464102
50%	0.135976	2.780110e+05	5.000000
75%	0.229167	3.673970e+05	6.708204
max	0.500000	4.562550e+05	9.848858
	BR_N_R_DAYS_CREDIT	BR_N_R_DAYS_ENDDATE_FACT	\
count	1.678748e+06	1.082775e+06	
mean	3.156889e+01	2.958773e+01	
std	1.240363e+01	1.191652e+01	
min	0.000000e+00	0.000000e+00	
25%	2.197726e+01	2.061553e+01	
50%	3.160696e+01	2.994996e+01	
75%	4.092676e+01	3.858756e+01	
max	5.405553e+01	2.049951e+02	
	BR_N_L_CREDIT_DAY_OVERDUE	BR_O_R_MONTHS_BALANCE_count	\
count	1.678748e+06	774354.000000	
mean	9.694972e-03	31.225694	
std	2.153840e-01	23.483837	
min	0.000000e+00	1.000000	
25%	0.000000e+00	12.000000	
50%	0.000000e+00	25.000000	
75%	0.000000e+00	45.000000	
max	7.934872e+00	97.000000	
	BR_O_R_DAYS_CREDIT	BR_O_R_DAYS_ENDDATE_FACT	BR_O_L_CREDIT_DAY_OVERD
UE			
count	1.678748e+06	1.082775e+06	1.678748e+
06			
mean	1.150445e+03	1.017437e+03	7.189822e-
01			

```
    std          7.945300e+02           7.140106e+02           3.369521e+
01
    min          0.000000e+00           0.000000e+00           0.000000e+
00
    25%         4.830000e+02           4.250000e+02           0.000000e+
00
    50%         9.990000e+02           8.970000e+02           0.000000e+
00
    75%        1.675000e+03           1.489000e+03           0.000000e+
00
    max        2.922000e+03           4.202300e+04           2.792000e+
03
```

[8 rows x 42 columns]

```
-----
# of Categorical features: 3
Categorical features:
['BR_O_CREDIT_ACTIVE', 'BR_O_CREDIT_CURRENCY', 'BR_O_CREDIT_TYPE']
```

Categorical Statistical Summary:

Categories:

```
BR_O_CREDIT_ACTIVE           [Closed, Active, Sold, Bad_debt]
BR_O_CREDIT_CURRENCY         [currency_1, currency_2, currency_4, currency_3]
BR_O_CREDIT_TYPE             [Consumer_credit, Credit_card, Mortgage, Car_1...
dtype: object
```

```
-----
# of OHE categorical features: 23
OHE Categorical features: ['BR_O_CREDIT_TYPE_Another_type_of_loan', 'BR_O_CREDIT_TYPE_Loan_for_purchase_of_shares_margin_lending', 'BR_O_CREDIT_TYPE_Unknown_type_of_loan', 'BR_O_CREDIT_CURRENCY_currency_2', 'BR_O_CREDIT_TYPE_Mortgage', 'BR_O_CREDIT_CURRENCY_currency_1', 'BR_O_CREDIT_TYPE_Loan_for_business_development', 'BR_O_CREDIT_TYPE_Loan_for_the_purchase_of_equipment', 'BR_O_CREDIT_TYPE_Car_loan', 'BR_O_CREDIT_ACTIVE_Closed', 'BR_O_CREDIT_ACTIVE_Active', 'BR_O_CREDIT_TYPE_Credit_card', 'BR_O_CREDIT_TYPE_Cash_loan_non_earmarked', 'BR_O_CREDIT_TYPE_Consumer_credit', 'BR_O_CREDIT_ACTIVE_Bad_debt', 'BR_O_CREDIT_ACTIVE_Sold', 'BR_O_CREDIT_TYPE_Mobile_operator_loan', 'BR_O_CREDIT_CURRENCY_currency_3', 'BR_O_CREDIT_TYPE_Microloan', 'BR_O_CREDIT_CURRENCY_currency_4', 'BR_O_CREDIT_TYPE_Loan_for_working_capital_replenishment', 'BR_O_CREDIT_TYPE_Real_estate_loan', 'BR_O_CREDIT_TYPE_Interbank_credit']
```

df.shape: (1678748, 65)

Aggregated Features:

```
df[['BR_O_CREDIT_TYPE_Another_type_of_loan', 'BR_O_CREDIT_TYPE_Unknown_type_
of_loan', 'BR_O_CREDIT_CURRENCY_currency_2', 'BR_O_CREDIT_TYPE_Mortgage', 'BR_O_STATUS_4_median', 'BR_O_AMT_CREDIT_SUM_DEBT', 'BR_N_R_MONTHS_BALANCE_count', 'BR_O_DAYS_CREDIT_ENDDATE', 'BR_O_CREDIT_CURRENCY_currency_1', 'BR_O_CREDIT_TYPE_Loan_for_business_development', 'BR_O_STATUS_1_var', 'BR_O_STATUS_0_median', 'BR_O_CREDIT_TYPE_Car_loan', 'BR_O_CREDIT_ACTIVE_Active', 'BR_O_STATUS_4_var', 'BR_O_STATUS_0_mean', 'BR_O_L_CREDIT_DAY_OVERDUE', 'BR_O_STATUS_1_median', 'BR_O_CREDIT_TYPE_Consumer_credit', 'BR_O_STATUS_C_median', 'BR_O_CREDIT_CURRENCY_currency_3', 'BR_O_STATUS_X_median', 'BR_O_AMT_ANNUITY', 'BR_O_STATUS_0_var', 'BR_O_AMT_CREDIT_SUM_OVERDUE', 'BR_N_L_CREDIT_DAY_OVERDUE', 'BR_O_STATUS_3_median', 'BR_O_CREDIT_TYPE_Loan_for_working_capital_repl
```

```

enishment', 'BR_O_CREDIT_TYPE_Real_estate_loan', 'BR_O_CREDIT_TYPE_Interbank
_credit', 'BR_O_AMT_CREDIT_SUM_LIMIT', 'BR_O_R_DAYS_ENDDATE_FACT', 'BR_O_AMT
_CREDIT_MAX_OVERDUE', 'BR_O_CREDIT_TYPE_Loan_for_purchase_of_shares_margin_1
ending', 'BR_O_STATUS_5_var', 'BR_N_R_DAYS_ENDDATE_FACT', 'BR_O_STATUS_X_var
', 'BR_O_STATUS_C_mean', 'BR_O_R_MONTHS_BALANCE_count', 'BR_O_STATUS_X_mean
', 'BR_O_STATUS_3_mean', 'BR_O_CREDIT_TYPE_Loan_for_the_purchase_of_equipmen
t', 'BR_O_STATUS_C_var', 'BR_O_STATUS_5_median', 'BR_O_STATUS_5_mean', 'BR_O
_AMT_CREDIT_SUM', 'BR_O_CREDIT_ACTIVE_Closed', 'BR_O_STATUS_2_mean', 'BR_O_C
REDIT_TYPE_Credit_card', 'BR_O_CREDIT_TYPE_Cash_loan_non_earmarked', 'BR_O_C
REDIT_ACTIVE_Bad_debt', 'BR_O_STATUS_4_mean', 'BR_O_CREDIT_ACTIVE_Sold', 'BR
_O_CREDIT_TYPE_Mobile_operator_loan', 'BR_O_STATUS_1_mean', 'BR_O_DAYS_CREDI
T_UPDATE', 'BR_O_R_DAYS_CREDIT', 'BR_O_CREDIT_TYPE_Microloan', 'BR_N_R_DAYS_
CREDIT', 'BR_O_STATUS_3_var', 'BR_O_CNT_CREDIT_PROLONG', 'BR_O_STATUS_2_var
', 'BR_O_CREDIT_CURRENCY_currency_4', 'BR_O_STATUS_2_median']] [0:5]:
    BR_O_CREDIT_TYPE_Another_type_of_loan \
0
1
4
5
6

    BR_O_CREDIT_TYPE_Unknown_type_of_loan  BR_O_CREDIT_CURRENCY_currency_2 \
0
1
4
5
6

    BR_O_CREDIT_TYPE_Mortgage  BR_O_STATUS_4_median  BR_O_AMT_CREDIT_SUM_DEBT
\
0
1
4
5
6

    BR_N_R_MONTHS_BALANCE_count  BR_O_DAYS_CREDIT_ENDDATE \
0
1
4
5
6

    BR_O_CREDIT_CURRENCY_currency_1 \
0
1
4
5
6

    BR_O_CREDIT_TYPE_Loan_for_business_development ...  BR_O_STATUS_1_mean
\
0
1
4
5
6

    BR_O_DAYS_CREDIT_UPDATE  BR_O_R_DAYS_CREDIT  BR_O_CREDIT_TYPE_Microloan

```

```

\

0 131 497 0
1 20 208 0
4 21 629 0
5 31 273 0
6 22 43 0

BR_N_R_DAYS_CREDIT BR_O_STATUS_3_var BR_O_CNT_CREDIT_PROLONG \
0 22.293497 NaN 0
1 14.422205 NaN 0
4 25.079872 NaN 0
5 16.522712 NaN 0
6 6.557439 NaN 0

BR_O_STATUS_2_var BR_O_CREDIT_CURRENCY_currency_4 BR_O_STATUS_2_median
0 NaN 0 NaN
1 NaN 0 NaN
4 NaN 0 NaN
5 NaN 0 NaN
6 NaN 0 NaN

[5 rows x 64 columns]
-----
Aggregated Features:
BR_N_L_CREDIT_DAY_OVERDUE_max
BR_N_L_CREDIT_DAY_OVERDUE_mean
BR_N_L_CREDIT_DAY_OVERDUE_median
BR_N_L_CREDIT_DAY_OVERDUE_min
BR_N_L_CREDIT_DAY_OVERDUE_sum
BR_N_L_CREDIT_DAY_OVERDUE_var
BR_N_R_DAYS_CREDIT_max
BR_N_R_DAYS_CREDIT_mean
BR_N_R_DAYS_CREDIT_median
BR_N_R_DAYS_CREDIT_min
BR_N_R_DAYS_CREDIT_sum
BR_N_R_DAYS_CREDIT_var
BR_N_R_DAYS_ENDDATE_FACT_max
BR_N_R_DAYS_ENDDATE_FACT_mean
BR_N_R_DAYS_ENDDATE_FACT_median
BR_N_R_DAYS_ENDDATE_FACT_min
BR_N_R_DAYS_ENDDATE_FACT_sum
BR_N_R_DAYS_ENDDATE_FACT_var
BR_N_R_MONTHS_BALANCE_count_max
BR_N_R_MONTHS_BALANCE_count_mean
BR_N_R_MONTHS_BALANCE_count_median
BR_N_R_MONTHS_BALANCE_count_min
BR_N_R_MONTHS_BALANCE_count_sum
BR_N_R_MONTHS_BALANCE_count_var
BR_O_AMT_ANNUITY_max
BR_O_AMT_ANNUITY_mean
BR_O_AMT_ANNUITY_median
BR_O_AMT_ANNUITY_min
BR_O_AMT_ANNUITY_sum
BR_O_AMT_ANNUITY_var
BR_O_AMT_CREDIT_MAX_OVERDUE_max
BR_O_AMT_CREDIT_MAX_OVERDUE_mean
BR_O_AMT_CREDIT_MAX_OVERDUE_median
BR_O_AMT_CREDIT_MAX_OVERDUE_min
BR_O_AMT_CREDIT_MAX_OVERDUE_sum

```

BR_O_AMT_CREDIT_MAX_OVERDUE_var
BR_O_AMT_CREDIT_SUM_DEBT_max
BR_O_AMT_CREDIT_SUM_DEBT_mean
BR_O_AMT_CREDIT_SUM_DEBT_median
BR_O_AMT_CREDIT_SUM_DEBT_min
BR_O_AMT_CREDIT_SUM_DEBT_sum
BR_O_AMT_CREDIT_SUM_DEBT_var
BR_O_AMT_CREDIT_SUM_LIMIT_max
BR_O_AMT_CREDIT_SUM_LIMIT_mean
BR_O_AMT_CREDIT_SUM_LIMIT_median
BR_O_AMT_CREDIT_SUM_LIMIT_min
BR_O_AMT_CREDIT_SUM_LIMIT_sum
BR_O_AMT_CREDIT_SUM_LIMIT_var
BR_O_AMT_CREDIT_SUM_OVERDUE_max
BR_O_AMT_CREDIT_SUM_OVERDUE_mean
BR_O_AMT_CREDIT_SUM_OVERDUE_median
BR_O_AMT_CREDIT_SUM_OVERDUE_min
BR_O_AMT_CREDIT_SUM_OVERDUE_sum
BR_O_AMT_CREDIT_SUM_OVERDUE_var
BR_O_AMT_CREDIT_SUM_max
BR_O_AMT_CREDIT_SUM_mean
BR_O_AMT_CREDIT_SUM_median
BR_O_AMT_CREDIT_SUM_min
BR_O_AMT_CREDIT_SUM_sum
BR_O_AMT_CREDIT_SUM_var
BR_O_CNT_CREDIT_PROLONG_max
BR_O_CNT_CREDIT_PROLONG_mean
BR_O_CNT_CREDIT_PROLONG_median
BR_O_CNT_CREDIT_PROLONG_min
BR_O_CNT_CREDIT_PROLONG_sum
BR_O_CNT_CREDIT_PROLONG_var
BR_O_CREDIT_ACTIVE_Active_mean
BR_O_CREDIT_ACTIVE_Active_median
BR_O_CREDIT_ACTIVE_Active_var
BR_O_CREDIT_ACTIVE_Bad_debt_mean
BR_O_CREDIT_ACTIVE_Bad_debt_median
BR_O_CREDIT_ACTIVE_Bad_debt_var
BR_O_CREDIT_ACTIVE_Closed_mean
BR_O_CREDIT_ACTIVE_Closed_median
BR_O_CREDIT_ACTIVE_Closed_var
BR_O_CREDIT_ACTIVE_Sold_mean
BR_O_CREDIT_ACTIVE_Sold_median
BR_O_CREDIT_ACTIVE_Sold_var
BR_O_CREDIT_CURRENCY_currency_1_mean
BR_O_CREDIT_CURRENCY_currency_1_median
BR_O_CREDIT_CURRENCY_currency_1_var
BR_O_CREDIT_CURRENCY_currency_2_mean
BR_O_CREDIT_CURRENCY_currency_2_median
BR_O_CREDIT_CURRENCY_currency_2_var
BR_O_CREDIT_CURRENCY_currency_3_mean
BR_O_CREDIT_CURRENCY_currency_3_median
BR_O_CREDIT_CURRENCY_currency_3_var
BR_O_CREDIT_CURRENCY_currency_4_mean
BR_O_CREDIT_CURRENCY_currency_4_median
BR_O_CREDIT_CURRENCY_currency_4_var
BR_O_CREDIT_TYPE_Another_type_of_loan_mean
BR_O_CREDIT_TYPE_Another_type_of_loan_median
BR_O_CREDIT_TYPE_Another_type_of_loan_var
BR_O_CREDIT_TYPE_Car_loan_mean

BR_0_CREDIT_TYPE_Car_loan_median
BR_0_CREDIT_TYPE_Car_loan_var
BR_0_CREDIT_TYPE_Cash_loan_non_earmarked_mean
BR_0_CREDIT_TYPE_Cash_loan_non_earmarked_median
BR_0_CREDIT_TYPE_Cash_loan_non_earmarked_var
BR_0_CREDIT_TYPE_Consumer_credit_mean
BR_0_CREDIT_TYPE_Consumer_credit_median
BR_0_CREDIT_TYPE_Consumer_credit_var
BR_0_CREDIT_TYPE_Credit_card_mean
BR_0_CREDIT_TYPE_Credit_card_median
BR_0_CREDIT_TYPE_Credit_card_var
BR_0_CREDIT_TYPE_Interbank_credit_mean
BR_0_CREDIT_TYPE_Interbank_credit_median
BR_0_CREDIT_TYPE_Interbank_credit_var
BR_0_CREDIT_TYPE_Loan_for_business_development_mean
BR_0_CREDIT_TYPE_Loan_for_business_development_median
BR_0_CREDIT_TYPE_Loan_for_business_development_var
BR_0_CREDIT_TYPE_Loan_for_purchase_of_shares_margin_lending_mean
BR_0_CREDIT_TYPE_Loan_for_purchase_of_shares_margin_lending_median
BR_0_CREDIT_TYPE_Loan_for_purchase_of_shares_margin_lending_var
BR_0_CREDIT_TYPE_Loan_for_the_purchase_of_equipment_mean
BR_0_CREDIT_TYPE_Loan_for_the_purchase_of_equipment_median
BR_0_CREDIT_TYPE_Loan_for_the_purchase_of_equipment_var
BR_0_CREDIT_TYPE_Loan_for_working_capital_replenishment_mean
BR_0_CREDIT_TYPE_Loan_for_working_capital_replenishment_median
BR_0_CREDIT_TYPE_Loan_for_working_capital_replenishment_var
BR_0_CREDIT_TYPE_Microloan_mean
BR_0_CREDIT_TYPE_Microloan_median
BR_0_CREDIT_TYPE_Microloan_var
BR_0_CREDIT_TYPE_Mobile_operator_loan_mean
BR_0_CREDIT_TYPE_Mobile_operator_loan_median
BR_0_CREDIT_TYPE_Mobile_operator_loan_var
BR_0_CREDIT_TYPE_Mortgage_mean
BR_0_CREDIT_TYPE_Mortgage_median
BR_0_CREDIT_TYPE_Mortgage_var
BR_0_CREDIT_TYPE_Real_estate_loan_mean
BR_0_CREDIT_TYPE_Real_estate_loan_median
BR_0_CREDIT_TYPE_Real_estate_loan_var
BR_0_CREDIT_TYPE_Unknown_type_of_loan_mean
BR_0_CREDIT_TYPE_Unknown_type_of_loan_median
BR_0_CREDIT_TYPE_Unknown_type_of_loan_var
BR_0_DAYS_CREDIT_ENDDATE_max
BR_0_DAYS_CREDIT_ENDDATE_mean
BR_0_DAYS_CREDIT_ENDDATE_median
BR_0_DAYS_CREDIT_ENDDATE_min
BR_0_DAYS_CREDIT_ENDDATE_sum
BR_0_DAYS_CREDIT_ENDDATE_var
BR_0_DAYS_CREDIT_UPDATE_max
BR_0_DAYS_CREDIT_UPDATE_mean
BR_0_DAYS_CREDIT_UPDATE_median
BR_0_DAYS_CREDIT_UPDATE_min
BR_0_DAYS_CREDIT_UPDATE_sum
BR_0_DAYS_CREDIT_UPDATE_var
BR_0_L_CREDIT_DAY_OVERDUE_max
BR_0_L_CREDIT_DAY_OVERDUE_mean
BR_0_L_CREDIT_DAY_OVERDUE_median
BR_0_L_CREDIT_DAY_OVERDUE_min
BR_0_L_CREDIT_DAY_OVERDUE_sum
BR_0_L_CREDIT_DAY_OVERDUE_var

BR_O_R_DAYS_CREDIT_max
BR_O_R_DAYS_CREDIT_mean
BR_O_R_DAYS_CREDIT_median
BR_O_R_DAYS_CREDIT_min
BR_O_R_DAYS_CREDIT_sum
BR_O_R_DAYS_CREDIT_var
BR_O_R_DAYS_ENDDATE_FACT_max
BR_O_R_DAYS_ENDDATE_FACT_mean
BR_O_R_DAYS_ENDDATE_FACT_median
BR_O_R_DAYS_ENDDATE_FACT_min
BR_O_R_DAYS_ENDDATE_FACT_sum
BR_O_R_DAYS_ENDDATE_FACT_var
BR_O_R_MONTHS_BALANCE_count_max
BR_O_R_MONTHS_BALANCE_count_mean
BR_O_R_MONTHS_BALANCE_count_median
BR_O_R_MONTHS_BALANCE_count_min
BR_O_R_MONTHS_BALANCE_count_sum
BR_O_R_MONTHS_BALANCE_count_var
BR_O_STATUS_0_mean_max
BR_O_STATUS_0_mean_mean
BR_O_STATUS_0_mean_median
BR_O_STATUS_0_mean_min
BR_O_STATUS_0_mean_sum
BR_O_STATUS_0_mean_var
BR_O_STATUS_0_median_max
BR_O_STATUS_0_median_mean
BR_O_STATUS_0_median_median
BR_O_STATUS_0_median_min
BR_O_STATUS_0_median_sum
BR_O_STATUS_0_median_var
BR_O_STATUS_0_var_max
BR_O_STATUS_0_var_mean
BR_O_STATUS_0_var_median
BR_O_STATUS_0_var_min
BR_O_STATUS_0_var_sum
BR_O_STATUS_0_var_var
BR_O_STATUS_1_mean_max
BR_O_STATUS_1_mean_mean
BR_O_STATUS_1_mean_median
BR_O_STATUS_1_mean_min
BR_O_STATUS_1_mean_sum
BR_O_STATUS_1_mean_var
BR_O_STATUS_1_median_max
BR_O_STATUS_1_median_mean
BR_O_STATUS_1_median_median
BR_O_STATUS_1_median_min
BR_O_STATUS_1_median_sum
BR_O_STATUS_1_median_var
BR_O_STATUS_1_var_max
BR_O_STATUS_1_var_mean
BR_O_STATUS_1_var_median
BR_O_STATUS_1_var_min
BR_O_STATUS_1_var_sum
BR_O_STATUS_1_var_var
BR_O_STATUS_2_mean_max
BR_O_STATUS_2_mean_mean
BR_O_STATUS_2_mean_median
BR_O_STATUS_2_mean_min
BR_O_STATUS_2_mean_sum

BR_0_STATUS_2_mean_var
BR_0_STATUS_2_median_max
BR_0_STATUS_2_median_mean
BR_0_STATUS_2_median_median
BR_0_STATUS_2_median_min
BR_0_STATUS_2_median_sum
BR_0_STATUS_2_median_var
BR_0_STATUS_2_var_max
BR_0_STATUS_2_var_mean
BR_0_STATUS_2_var_median
BR_0_STATUS_2_var_min
BR_0_STATUS_2_var_sum
BR_0_STATUS_2_var_var
BR_0_STATUS_3_mean_max
BR_0_STATUS_3_mean_mean
BR_0_STATUS_3_mean_median
BR_0_STATUS_3_mean_min
BR_0_STATUS_3_mean_sum
BR_0_STATUS_3_mean_var
BR_0_STATUS_3_median_max
BR_0_STATUS_3_median_mean
BR_0_STATUS_3_median_median
BR_0_STATUS_3_median_min
BR_0_STATUS_3_median_sum
BR_0_STATUS_3_median_var
BR_0_STATUS_3_var_max
BR_0_STATUS_3_var_mean
BR_0_STATUS_3_var_median
BR_0_STATUS_3_var_min
BR_0_STATUS_3_var_sum
BR_0_STATUS_3_var_var
BR_0_STATUS_4_mean_max
BR_0_STATUS_4_mean_mean
BR_0_STATUS_4_mean_median
BR_0_STATUS_4_mean_min
BR_0_STATUS_4_mean_sum
BR_0_STATUS_4_mean_var
BR_0_STATUS_4_median_max
BR_0_STATUS_4_median_mean
BR_0_STATUS_4_median_median
BR_0_STATUS_4_median_min
BR_0_STATUS_4_median_sum
BR_0_STATUS_4_median_var
BR_0_STATUS_4_var_max
BR_0_STATUS_4_var_mean
BR_0_STATUS_4_var_median
BR_0_STATUS_4_var_min
BR_0_STATUS_4_var_sum
BR_0_STATUS_4_var_var
BR_0_STATUS_5_mean_max
BR_0_STATUS_5_mean_mean
BR_0_STATUS_5_mean_median
BR_0_STATUS_5_mean_min
BR_0_STATUS_5_mean_sum
BR_0_STATUS_5_mean_var
BR_0_STATUS_5_median_max
BR_0_STATUS_5_median_mean
BR_0_STATUS_5_median_median
BR_0_STATUS_5_median_min

```

BR_0_STATUS_5_median_sum
BR_0_STATUS_5_median_var
BR_0_STATUS_5_var_max
BR_0_STATUS_5_var_mean
BR_0_STATUS_5_var_median
BR_0_STATUS_5_var_min
BR_0_STATUS_5_var_sum
BR_0_STATUS_5_var_var
BR_0_STATUS_C_mean_max
BR_0_STATUS_C_mean_mean
BR_0_STATUS_C_mean_median
BR_0_STATUS_C_mean_min
BR_0_STATUS_C_mean_sum
BR_0_STATUS_C_mean_var
BR_0_STATUS_C_median_max
BR_0_STATUS_C_median_mean
BR_0_STATUS_C_median_median
BR_0_STATUS_C_median_min
BR_0_STATUS_C_median_sum
BR_0_STATUS_C_median_var
BR_0_STATUS_C_var_max
BR_0_STATUS_C_var_mean
BR_0_STATUS_C_var_median
BR_0_STATUS_C_var_min
BR_0_STATUS_C_var_sum
BR_0_STATUS_C_var_var
BR_0_STATUS_X_mean_max
BR_0_STATUS_X_mean_mean
BR_0_STATUS_X_mean_median
BR_0_STATUS_X_mean_min
BR_0_STATUS_X_mean_sum
BR_0_STATUS_X_mean_var
BR_0_STATUS_X_median_max
BR_0_STATUS_X_median_mean
BR_0_STATUS_X_median_median
BR_0_STATUS_X_median_min
BR_0_STATUS_X_median_sum
BR_0_STATUS_X_median_var
BR_0_STATUS_X_var_max
BR_0_STATUS_X_var_mean
BR_0_STATUS_X_var_median
BR_0_STATUS_X_var_min
BR_0_STATUS_X_var_sum
BR_0_STATUS_X_var_var
index

```

Aggregated Data:

	count	mean
\		
BR_0_CREDIT_TYPE_Loan_for_the_purchase_of_equip...	304767.0	1.300471e-05
BR_0_CREDIT_CURRENCY_currency_3_mean	304767.0	7.747138e-05
BR_N_R_DAYS_ENDDATE_FACT_max	268155.0	3.644382e+01
BR_O_R_DAYS_ENDDATE_FACT_max	268155.0	1.465231e+03
BR_0_CREDIT_CURRENCY_currency_4_median	304767.0	1.640598e-06
...
BR_0_STATUS_0_mean_min	134542.0	2.024299e-01
BR_0_AMT_CREDIT_SUM_LIMIT_min	280503.0	1.117188e+03
BR_0_CREDIT_ACTIVE_Bad_debt_median	304767.0	4.921793e-06

BR_O_CREDIT_TYPE_Interbank_credit_mean	304767.0	5.468659e-07
BR_O_CREDIT_CURRENCY_currency_2_mean	304767.0	7.063657e-04
	std	min
\		
BR_O_CREDIT_TYPE_Loan_for_the_purchase_of_equip...	0.002777	0.000
BR_O_CREDIT_CURRENCY_currency_3_mean	0.004427	0.000
BR_N_R_DAYS_ENDDATE_FACT_max	11.708072	0.000
BR_O_R_DAYS_ENDDATE_FACT_max	787.620310	0.000
BR_O_CREDIT_CURRENCY_currency_4_median	0.000906	0.000
...
BR_O_STATUS_0_mean_min	0.274220	0.000
BR_O_AMT_CREDIT_SUM_LIMIT_min	20651.558290	-586406.115
BR_O_CREDIT_ACTIVE_Bad_debt_median	0.002025	0.000
BR_O_CREDIT_TYPE_Interbank_credit_mean	0.000302	0.000
BR_O_CREDIT_CURRENCY_currency_2_mean	0.016990	0.000
	25%	50%
\		
BR_O_CREDIT_TYPE_Loan_for_the_purchase_of_equip...	0.000000	0.000000
BR_O_CREDIT_CURRENCY_currency_3_mean	0.000000	0.000000
BR_N_R_DAYS_ENDDATE_FACT_max	28.513155	38.026307
BR_O_R_DAYS_ENDDATE_FACT_max	813.000000	1446.000000
BR_O_CREDIT_CURRENCY_currency_4_median	0.000000	0.000000
...
BR_O_STATUS_0_mean_min	0.000000	0.086207
BR_O_AMT_CREDIT_SUM_LIMIT_min	0.000000	0.000000
BR_O_CREDIT_ACTIVE_Bad_debt_median	0.000000	0.000000
BR_O_CREDIT_TYPE_Interbank_credit_mean	0.000000	0.000000
BR_O_CREDIT_CURRENCY_currency_2_mean	0.000000	0.000000
	75%	ma
x		
BR_O_CREDIT_TYPE_Loan_for_the_purchase_of_equip...	0.000000	1.000000e+0
0		
BR_O_CREDIT_CURRENCY_currency_3_mean	0.000000	1.000000e+0
0		
BR_N_R_DAYS_ENDDATE_FACT_max	46.925473	2.049951e+0
2		
BR_O_R_DAYS_ENDDATE_FACT_max	2202.000000	4.202300e+0
4		
BR_O_CREDIT_CURRENCY_currency_4_median	0.000000	5.000000e-0
1		
...
...
BR_O_STATUS_0_mean_min	0.285714	1.000000e+0
0		
BR_O_AMT_CREDIT_SUM_LIMIT_min	0.000000	4.500000e+0
6		
BR_O_CREDIT_ACTIVE_Bad_debt_median	0.000000	1.000000e+0
0		
BR_O_CREDIT_TYPE_Interbank_credit_mean	0.000000	1.666667e-0
1		
BR_O_CREDIT_CURRENCY_currency_2_mean	0.000000	1.000000e+0
0		

[316 rows x 8 columns]

```
In [34]: bureau_trans_untrans.to_csv(os.getcwd() + DATA_DIR + 'bureau_agg_data_trans_
```

```
In [43]: br = datasets['bureau']

# rollup bureau_bal
# gets rid of the unwanted characters in categorical columns entries
    .replace(to_replace='\s+', value='_', regex=True) \
    .replace(to_replace='\'-', value='_', regex=True) \
    .replace(to_replace='\\/', value='_', regex=True) \
    .replace(to_replace='\'(', value=''', regex=True) \
    .replace(to_replace=''\)', value=''', regex=True) \
    .replace(to_replace='\'::', value=''', regex=True) \
    .replace(to_replace='\' ,', value=''', regex=True) \
    .drop('SK_ID_BUREAU', axis=1)
```

```
In [44]: br['ROOT_MONTHS_BALANCE_count'] = np.sqrt(br['MONTHS_BALANCE_count'])
br['ROOT_DAYS_CREDIT'] = np.sqrt(br['DAYS_CREDIT'].abs())
br['ROOT_DAYS_ENDDATE_FACT'] = np.sqrt(br['DAYS_ENDDATE_FACT'].abs())
br['LOG_CREDIT_DAY_OVERDUE'] = np.log(br['CREDIT_DAY_OVERDUE'].abs() + 1)
```

```
In [ ]: br_transformed = br.drop(['MONTHS_BALANCE_count', 'DAYS_CREDIT', 'CREDIT_DA']
br_transformed = bureau_eda(br_transformed)
br_transformed.to_csv(os.getcwd() + DATA_DIR + 'bureau_agg_data_transformed
```

```

-----
# of ID's: 1
ID's:
['SK_ID_CURR']

-----
# All features: 40
All features:
['STATUS_C_var', 'LOG_CREDIT_DAY_OVERDUE', 'CREDIT_CURRENCY', 'STATUS_X_median', 'STATUS_C_median', 'STATUS_X_var', 'AMT_CREDIT_SUM_OVERDUE', 'AMT_CREDIT_SUM', 'STATUS_3_median', 'STATUS_3_mean', 'STATUS_5_mean', 'CNT_CREDIT_PROLONG', 'STATUS_5_median', 'STATUS_1_mean', 'AMT_CREDIT_SUM_LIMIT', 'AMT_ANNUITY', 'ROOT_DAYS_CREDIT', 'STATUS_3_var', 'STATUS_2_median', 'STATUS_4_mean', 'STATUS_0_mean', 'STATUS_1_median', 'DAYS_CREDIT_ENDDATE', 'STATUS_C_mean', 'STATUS_2_var', 'ROOT_MONTHS_BALANCE_count', 'AMT_CREDIT_SUM_DEBT', 'CREDIT_TYPE', 'STATUS_5_var', 'AMT_CREDIT_MAX_OVERDUE', 'STATUS_4_mean', 'DAYS_CREDIT_UPDATE', 'STATUS_4_var', 'STATUS_X_mean', 'STATUS_0_median', 'STATUS_0_var', 'ROOT_DAYS_ENDDATE_FACT', 'STATUS_2_mean', 'STATUS_1_var', 'CREDIT_ACTIVE']

```

Missing data:

	Total	Percent
AMT_ANNUITY	1064802	68.507601
AMT_CREDIT_MAX_OVERDUE	976510	62.827040
STATUS_C_var	785833	50.559197
STATUS_4_var	785833	50.559197
STATUS_2_var	785833	50.559197
STATUS_5_var	785833	50.559197
STATUS_3_var	785833	50.559197
STATUS_X_var	785833	50.559197
STATUS_0_var	785833	50.559197
STATUS_1_var	785833	50.559197
STATUS_X_mean	779929	50.179343
STATUS_0_median	779929	50.179343
STATUS_4_median	779929	50.179343
ROOT_MONTHS_BALANCE_count	779929	50.179343
STATUS_2_mean	779929	50.179343
STATUS_C_mean	779929	50.179343
STATUS_1_median	779929	50.179343
STATUS_4_mean	779929	50.179343
STATUS_0_mean	779929	50.179343
STATUS_2_median	779929	50.179343
STATUS_5_median	779929	50.179343
STATUS_X_median	779929	50.179343
STATUS_C_median	779929	50.179343
STATUS_3_median	779929	50.179343
STATUS_3_mean	779929	50.179343
STATUS_5_mean	779929	50.179343
STATUS_1_mean	779929	50.179343
ROOT_DAYS_ENDDATE_FACT	542773	34.921118
AMT_CREDIT_SUM_LIMIT	430737	27.712907
AMT_CREDIT_SUM_DEBT	144124	9.272700
DAYS_CREDIT_ENDDATE	81383	5.236048
AMT_CREDIT_SUM	5	0.000322
CREDIT_TYPE	0	0.000000
CNT_CREDIT_PROLONG	0	0.000000
LOG_CREDIT_DAY_OVERDUE	0	0.000000
DAYS_CREDIT_UPDATE	0	0.000000
AMT_CREDIT_SUM_OVERDUE	0	0.000000

```

ROOT_DAYS_CREDIT          0    0.000000
CREDIT_CURRENCY           0    0.000000
CREDIT_ACTIVE              0    0.000000

-----
# of Numerical features: 38
Numerical features:
['SK_ID_CURR', 'DAYS_CREDIT_ENDDATE', 'AMT_CREDIT_MAX_OVERDUE', 'CNT_CREDIT_
PROLONG', 'AMT_CREDIT_SUM', 'AMT_CREDIT_SUM_DEBT', 'AMT_CREDIT_SUM_LIMIT', 'AMT_CREDIT_SUM_OVERDUE', 'DAYS_CREDIT_UPDATE', 'AMT_ANNUITY', 'STATUS_X_medi
an', 'STATUS_X_mean', 'STATUS_X_var', 'STATUS_3_median', 'STATUS_3_mean', 'S
TATUS_3_var', 'STATUS_2_median', 'STATUS_2_mean', 'STATUS_2_var', 'STATUS_4_
median', 'STATUS_4_mean', 'STATUS_4_var', 'STATUS_5_median', 'STATUS_5_mean
', 'STATUS_5_var', 'STATUS_C_median', 'STATUS_C_mean', 'STATUS_C_var', 'STAT
US_1_median', 'STATUS_1_mean', 'STATUS_1_var', 'STATUS_0_median', 'STATUS_0_
mean', 'STATUS_0_var', 'ROOT_MONTHS_BALANCE_count', 'ROOT_DAYS_CREDIT', 'ROO
T_DAYS_ENDDATE_FACT', 'LOG_CREDIT_DAY_OVERDUE']

```

Numerical Statistical Summary:

	SK_ID_CURR	DAYS_CREDIT_ENDDATE	AMT_CREDIT_MAX_OVERDUE	\		
count	1.554283e+06	1.472900e+06	5.777730e+05			
mean	2.781766e+05	1.724138e+03	3.707606e+03			
std	1.029210e+05	4.470893e+03	2.084828e+05			
min	1.000010e+05	0.000000e+00	0.000000e+00			
25%	1.888350e+05	3.750000e+02	0.000000e+00			
50%	2.780340e+05	8.460000e+02	0.000000e+00			
75%	3.673420e+05	1.478000e+03	0.000000e+00			
max	4.562550e+05	4.206000e+04	1.159872e+08			
	CNT_CREDIT_PROLONG	AMT_CREDIT_SUM	AMT_CREDIT_SUM_DEBT	\		
count	1.554283e+06	1.554278e+06	1.410159e+06			
mean	6.313522e-03	3.438281e+05	1.296205e+05			
std	9.518150e-02	1.137067e+06	6.581787e+05			
min	0.000000e+00	0.000000e+00	-4.705600e+06			
25%	0.000000e+00	4.945500e+04	0.000000e+00			
50%	0.000000e+00	1.214910e+05	0.000000e+00			
75%	0.000000e+00	2.942775e+05	2.977650e+04			
max	9.000000e+00	5.850000e+08	1.701000e+08			
	AMT_CREDIT_SUM_LIMIT	AMT_CREDIT_SUM_OVERDUE	DAYS_CREDIT_UPDATE	\		
count	1.123546e+06	1.554283e+06	1.554283e+06			
mean	6.235625e+03	2.881801e+01	5.981201e+02			
std	4.505369e+04	4.957950e+03	7.242388e+02			
min	-5.864061e+05	0.000000e+00	0.000000e+00			
25%	0.000000e+00	0.000000e+00	3.500000e+01			
50%	0.000000e+00	0.000000e+00	4.100000e+02			
75%	0.000000e+00	0.000000e+00	9.030000e+02			
max	4.705600e+06	3.756681e+06	4.194700e+04			
	AMT_ANNUITY	...	STATUS_1_median	STATUS_1_mean	STATUS_1_var	\
count	4.894810e+05	...	774354.000000	774354.000000	768450.000000	
mean	1.571016e+04	...	0.001443	0.011471	0.009917	
std	3.258786e+05	...	0.035930	0.047255	0.035728	
min	0.000000e+00	...	0.000000	0.000000	0.000000	
25%	0.000000e+00	...	0.000000	0.000000	0.000000	
50%	0.000000e+00	...	0.000000	0.000000	0.000000	
75%	1.350000e+04	...	0.000000	0.000000	0.000000	
max	1.184534e+08	...	1.000000	1.000000	0.500000	

	STATUS_0_median	STATUS_0_mean	STATUS_0_var	\
count	774354.000000	774354.000000	768450.000000	
mean	0.375246	0.413294	0.133522	
std	0.477813	0.346153	0.103150	
min	0.000000	0.000000	0.000000	
25%	0.000000	0.103448	0.027027	
50%	0.000000	0.333333	0.135976	
75%	1.000000	0.727273	0.229167	
max	1.000000	1.000000	0.500000	
	ROOT_MONTHS_BALANCE_count	ROOT_DAYS_CREDIT	ROOT_DAYS_ENDDATE_FACT	
\				
count	774354.000000	1.554283e+06	1.011510e+06	
mean	5.172397	3.163231e+01	2.963710e+01	
std	2.114712	1.235788e+01	1.194345e+01	
min	1.000000	0.000000e+00	0.000000e+00	
25%	3.464102	2.213594e+01	2.061553e+01	
50%	5.000000	3.167018e+01	2.984962e+01	
75%	6.708204	4.090232e+01	3.874274e+01	
max	9.848858	5.405553e+01	2.049951e+02	
	LOG_CREDIT_DAY_OVERDUE			
count	1.554283e+06			
mean	9.155844e-03			
std	2.107809e-01			
min	0.000000e+00			
25%	0.000000e+00			
50%	0.000000e+00			
75%	0.000000e+00			
max	7.934872e+00			

[8 rows x 38 columns]

of Categorical features: 3
Categorical features:
['CREDIT_ACTIVE', 'CREDIT_CURRENCY', 'CREDIT_TYPE']

Categorical Statistical Summary:

Categories:

CREDIT_ACTIVE	[Closed, Active, Sold, Bad_debt]
CREDIT_CURRENCY	[currency_1, currency_2, currency_4, currency_3]
CREDIT_TYPE	[Consumer_credit, Credit_card, Car_loan, Mortg...]
dtype: object	

of OHE categorical features: 23
OHE Categorical features: ['CREDIT_TYPE_Another_type_of_loan', 'CREDIT_ACTIVE_Closed', 'CREDIT_TYPE_Credit_card', 'CREDIT_TYPE_Interbank_credit', 'CREDIT_TYPE_Loan_for_purchase_of_shares_margin_lending', 'CREDIT_TYPE_Mortgage', 'CREDIT_TYPE_Loan_for_the_purchase_of_equipment', 'CREDIT_TYPE_Consumer_credit', 'CREDIT_CURRENCY_currency_1', 'CREDIT_CURRENCY_currency_4', 'CREDIT_TYPE_Cash_loan_non_earmarked', 'CREDIT_CURRENCY_currency_3', 'CREDIT_ACTIVE_Sold', 'CREDIT_ACTIVE_Active', 'CREDIT_TYPE_Microloan', 'CREDIT_TYPE_Loan_for_working_capital_replenishment', 'CREDIT_TYPE_Mobile_operator_loan', 'CREDIT_T

```
YPE_Loan_for_business_development', 'CREDIT_CURRENCY_currency_2', 'CREDIT_TYPE  
PE_Real_estate_loan', 'CREDIT_TYPE_Unknown_type_of_loan', 'CREDIT_ACTIVE_Bad  
_debt', 'CREDIT_TYPE_Car_loan']  
-----  
df.shape: (1554283, 61)
```

Aggregated Features:

```
df[['CREDIT_TYPE_Another_type_of_loan', 'CREDIT_ACTIVE_Closed', 'STATUS_C_va  
r', 'LOG_CREDIT_DAY_OVERDUE', 'STATUS_X_median', 'CREDIT_TYPE_Credit_card',  
'CREDIT_TYPE_Interbank_credit', 'CREDIT_TYPE_Loan_for_purchase_of_shares_mar  
gin_lending', 'STATUS_C_median', 'CREDIT_TYPE_Mortgage', 'STATUS_X_var', 'AM  
T_CREDIT_SUM_OVERDUE', 'AMT_CREDIT_SUM', 'CREDIT_TYPE_Loan_for_the_purchase_  
of_equipment', 'STATUS_3_median', 'STATUS_3_mean', 'CREDIT_TYPE_Consumer_cre  
dit', 'STATUS_5_mean', 'CNT_CREDIT_PROLONG', 'STATUS_5_median', 'STATUS_1_me  
an', 'AMT_CREDIT_SUM_LIMIT', 'CREDIT_CURRENCY_currency_1', 'AMT_ANNUITY', 'R  
OOT_DAYS_CREDIT', 'STATUS_3_var', 'STATUS_2_median', 'CREDIT_CURRENCY_curren  
cy_4', 'STATUS_4_median', 'CREDIT_TYPE_Cash_loan_non_earmarked', 'STATUS_0_m  
ean', 'STATUS_1_median', 'CREDIT_CURRENCY_currency_3', 'DAYS_CREDIT_ENDDATE  
' , 'STATUS_C_mean', 'STATUS_2_var', 'CREDIT_ACTIVE_Sold', 'ROOT_MONTHS_BALAN  
CE_count', 'AMT_CREDIT_SUM_DEBT', 'CREDIT_ACTIVE_Active', 'CREDIT_TYPE_Loan_  
for_working_capital_replenishment', 'CREDIT_TYPE_Microloan', 'STATUS_5_var',  
'AMT_CREDIT_MAX_OVERDUE', 'CREDIT_TYPE_Mobile_operator_loan', 'STATUS_4_mean  
' , 'CREDIT_TYPE_Loan_for_business_development', 'DAYS_CREDIT_UPDATE', 'CREDI  
T_CURRENCY_currency_2', 'STATUS_4_var', 'CREDIT_TYPE_Real_estate_loan', 'CRE  
DIT_TYPE_Unknown_type_of_loan', 'STATUS_X_mean', 'STATUS_0_median', 'STATUS_  
0_var', 'ROOT_DAYS_ENDDATE_FACT', 'CREDIT_ACTIVE_Bad_debt', 'STATUS_2_mean',  
'STATUS_1_var', 'CREDIT_TYPE_Car_loan']] [0:5]:  
CREDIT_TYPE_Another_type_of_loan CREDIT_ACTIVE_Closed STATUS_C_var \\\n0 0 1 NaN  
5 0 0 NaN  
6 0 0 NaN  
7 0 1 NaN  
8 0 1 NaN  
  
LOG_CREDIT_DAY_OVERDUE STATUS_X_median CREDIT_TYPE_Credit_card \\\n0 0.0 NaN 0  
5 0.0 NaN 1  
6 0.0 NaN 0  
7 0.0 NaN 0  
8 0.0 NaN 0  
  
CREDIT_TYPE_Interbank_credit \\\n0 0  
5 0  
6 0  
7 0  
8 0  
  
CREDIT_TYPE_Loan_for_purchase_of_shares_margin_lending STATUS_C_median  
\\\n0 0 NaN  
5 0 NaN  
6 0 NaN  
7 0 NaN  
8 0 NaN  
  
CREDIT_TYPE_Mortgage ... CREDIT_TYPE_Real_estate_loan \\\n0 0 ... 0  
5 0 ... 0
```

6	0	...		0
7	0	...		0
8	0	...		0
	CREDIT_TYPE_Unknown_type_of_loan	STATUS_X_mean	STATUS_0_median	\
0	0	NaN	NaN	
5	0	NaN	NaN	
6	0	NaN	NaN	
7	0	NaN	NaN	
8	0	NaN	NaN	
	STATUS_0_var	ROOT_DAYS_ENDDATE_FACT	CREDIT_ACTIVE_Bad_debt	\
0	NaN	12.369317	0	
5	NaN	NaN	0	
6	NaN	NaN	0	
7	NaN	41.352146	0	
8	NaN	28.982753	0	
	STATUS_2_mean	STATUS_1_var	CREDIT_TYPE_Car_loan	
0	NaN	NaN	0	
5	NaN	NaN	0	
6	NaN	NaN	0	
7	NaN	NaN	0	
8	NaN	NaN	0	

[5 rows x 60 columns]

POS - POS_CASH_balance

1. This function is specifically for POS table. Below are the intial pre-processing steps done before passing this table into the pipeline.

- Create a drop list.
 - Attributes that will be dropped are added to this list and all columns will be deleted before passing the dataframe into the eda function.
- Create new features based on analysis. 3 new features were created:
 - Percentage of installments pending.
 - Number of installments pending.
 - Days with Tolerance.
- Take absolute of the months balance attribute. which was provided as negative values, as it is relative to application date.
- Replace " " with "_" for OHE columns.
- Any column or row with more than 70% of its data as null will be deleted from the dataframe, as the threshold is set to .7.
- Once processed, store the transformBenefit of this is that the data can be passed ed csv file. directly to model for merging into application train/test table. We do not have to repeatedly perform expensive EDA/ETL/Transformation.
- **Features Transformation:** Below are the features which are transformed.

(**PS_N_R_XX**: New feature with root transformation. **PS_N_L_XX**: New feature with log transformation. And **PS_N_XX** stands for new feature added.

#SQRT Transformation

- pos['PS_N_R_CNT_INSTALMENT_FUTURE'] =
np.sqrt(pos['PS_O_CNT_INSTALMENT_FUTURE'].abs())
- pos['PS_N_R_MONTHS_BALANCE'] =
np.sqrt(pos['PS_O_MONTHS_BALANCE'].abs())
- pos['PS_N_R_CNT_INSTAL_PNDNG'] =
np.sqrt(pos['PS_N_CNT_INSTAL_PNDNG'].abs())

#Log Transformation.

- pos['PS_N_L_CNT_INSTALMENT'] =
np.log(pos['PS_O_CNT_INSTALMENT'].abs()+1)

#New Features

- pos['PS_N_PERC_INSL_PNDNG']=pos['PS_O_CNT_INSTALMENT_FUTURE']/pos['PS_O_CNT_INSTALMENT']
- pos['PS_N_CNT_INSTAL_PNDNG']=pos['PS_O_CNT_INSTALMENT']-
pos['PS_O_CNT_INSTALMENT_FUTURE']
- pos['PS_N_DAYS_WTHT_TOLRNC']=pos['PS_O_SK_DPD']-

```
pos['PS_O_SK_DPD_DEF']
```

POS Feature Engineering

Addition and Transformation

Please refer to section 6.2.1 and 6.2.4 for details on feature renaming and transformation.

In [38]:

```
def pos_eda(df):
    pos = df
    drop_list_pos = []

    #Adding new features

    #new
    pos['PS_N_PERC_INSTL_PNDNG'] = pos['PS_O_CNT_INSTALMENT_FUTURE'] / pos['PS_O_CNT_INSTALMENT']
    pos['PS_N_CNT_INSTAL_PNDNG'] = pos['PS_O_CNT_INSTALMENT'] - pos['PS_O_CNT_INSTALMENT_FUTURE']
    pos['PS_N_DAYS_WHT_TOLRNC'] = pos['PS_O_SK_DPD'] - pos['PS_O_SK_DPD_DEF']
    pos['PS_O_MONTHS_BALANCE'] = pos['PS_O_MONTHS_BALANCE'].abs()

    #root
    pos['PS_N_R_CNT_INSTALMENT_FUTURE'] = np.sqrt(pos['PS_O_CNT_INSTALMENT_FUTURE'])
    pos['PS_N_R_MONTHS_BALANCE'] = np.sqrt(pos['PS_O_MONTHS_BALANCE'].abs())
    pos['PS_N_R_CNT_INSTAL_PNDNG'] = np.sqrt(pos['PS_N_CNT_INSTAL_PNDNG'])

    #log_transformed
    pos['PS_N_L_CNT_INSTALMENT'] = np.log(pos['PS_O_CNT_INSTALMENT'].abs() + 1)
    # Drop elements in drop list
    threshold = 0.7

    #Dropping rows with missing value rate higher than threshold
    pos = pos.loc[pos.isnull().mean(axis=1) < threshold]
    pos = pos.loc[pos.isnull().mean(axis=1) < threshold]

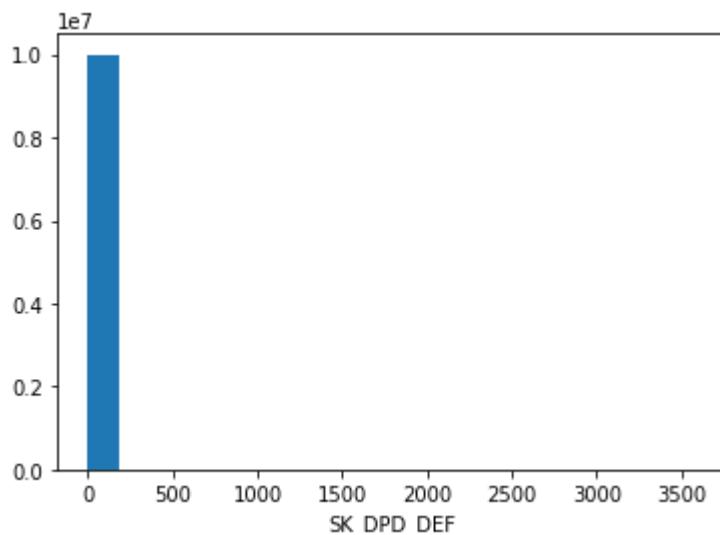
    return (eda_transformation(pos, 4))
```

Feature Transformation Viz

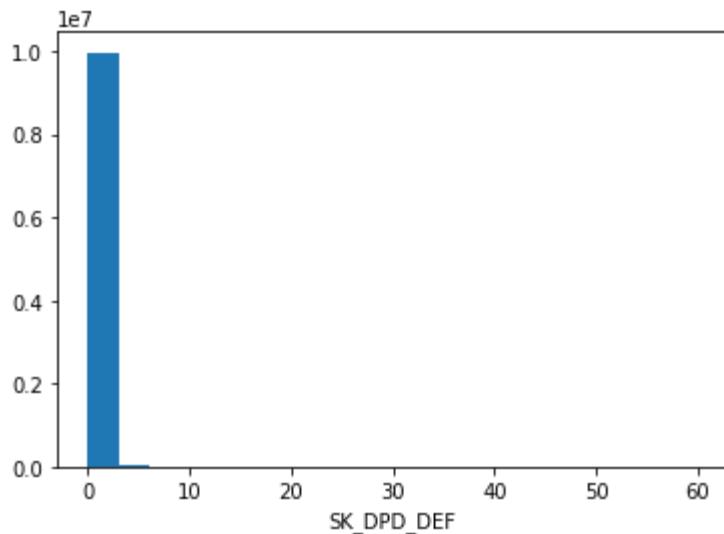
In [39]:

```
for i,ds_name in enumerate(datasets.keys()):  
  
    if ds_name.upper() in ("POS_CASH_BALANCE"):  
        ps = datasets[ds_name]  
        ps['POS_PERC_INSTL_PNDNG']=ps['CNT_INSTALMENT_FUTURE']/ps['CNT_INSTALMENT']  
        ps['POS_CNT_INSTAL_PNDNG']=ps['CNT_INSTALMENT']-ps['CNT_INSTALMENT_FUTURE']  
        ps['POS_DAYS_WHT_TOLRNC']=ps['SK_DPD']-ps['SK_DPD_DEF']  
        print("-----")  
        id_cols,num_cols,_,_ = id_num_cat_feature(ps, text = False)  
        id_cols.append('SK_ID_PREV')  
        new_cols = ['POS_PERC_INSTL_PNDNG','POS_CNT_INSTAL_PNDNG','POS_DAYS_WHT_TOLRNC']  
        cols= list(set(num_cols) - set(id_cols) )  
        #cols = ['PS_N_PERC_INSTL_PNDNG','PS_N_CNT_INSTAL_PNDNG','PS_N_DAYS_WHT_TOLRNC']  
        ps.replace([np.inf, -np.inf], 0, inplace=True)  
        for i in cols:  
            print('---')  
            print(f'{i}')  
            print(((ps[i].abs())).skew())  
            plt.hist(ps[i], bins=20)  
            plt.xlabel(i)  
            plt.show()  
            print('')  
            print(f'{i} :: root-transformed')  
            print((np.sqrt(ps[i].abs())).skew())  
            plt.hist(np.sqrt(ps[i].abs()), bins=20)  
            plt.xlabel(i)  
            plt.show()  
            print('')  
            print(f'{i} :: log-transformed')  
            print((np.log(ps[i].abs() + 1)).skew())  
            plt.hist(np.log(ps[i].abs() + 1), bins=20)  
            plt.xlabel(i)  
            plt.show()
```

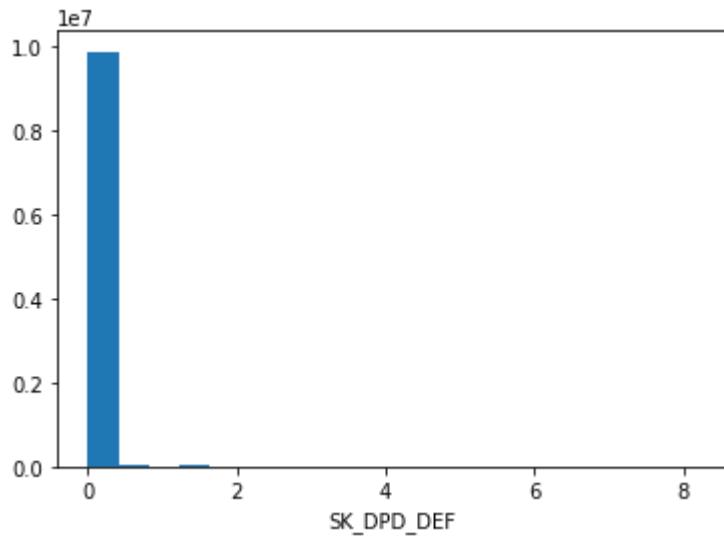
SK_DPD_DEF
66.3399058063118



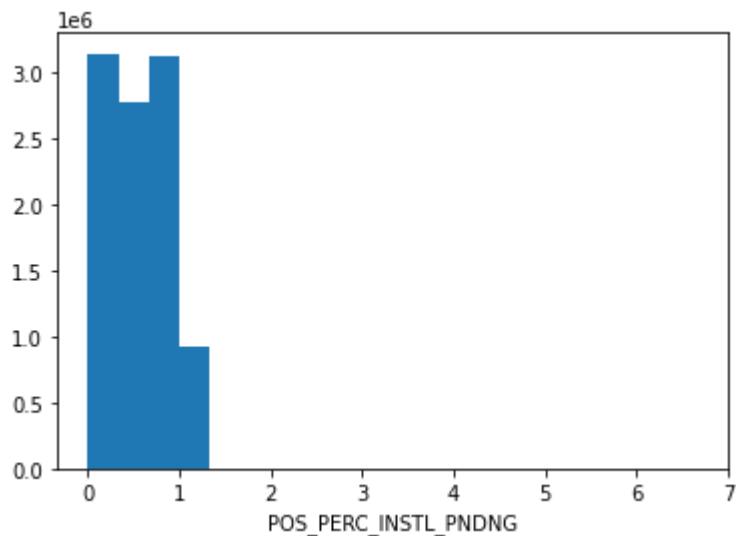
SK_DPD_DEF :: root-transformed
46.40958540356541



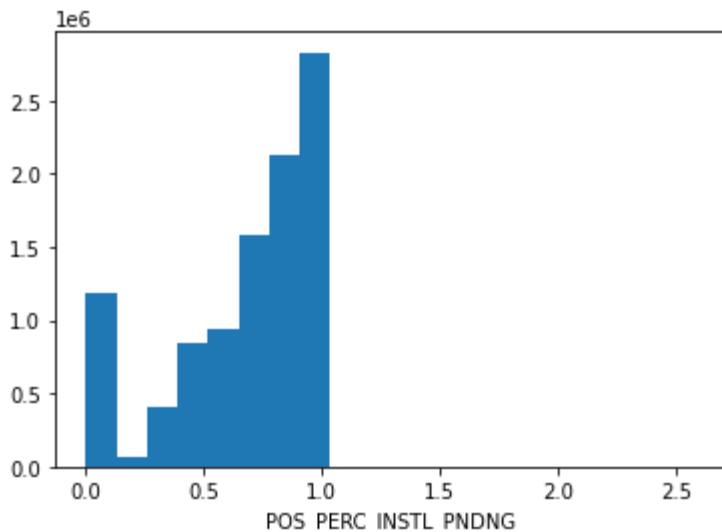
`SK_DPD_DEF` :: log-transformed
16.62957470160757



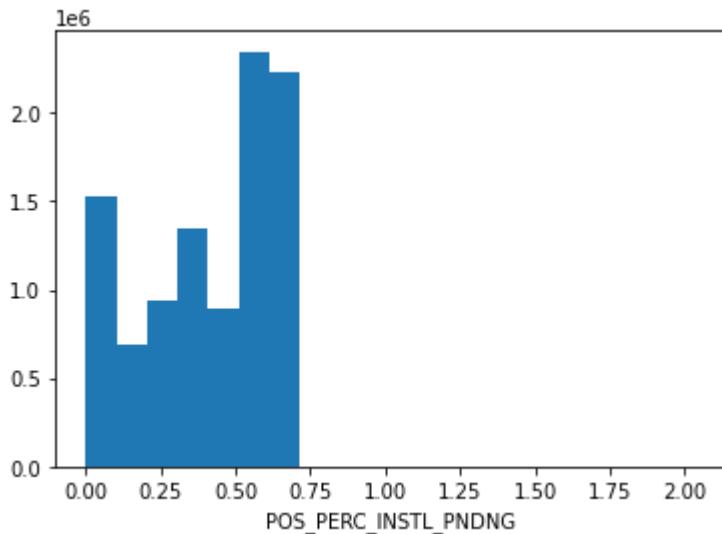
`POS_PERC_INSTL_PNDNG`
-0.2475198904955197



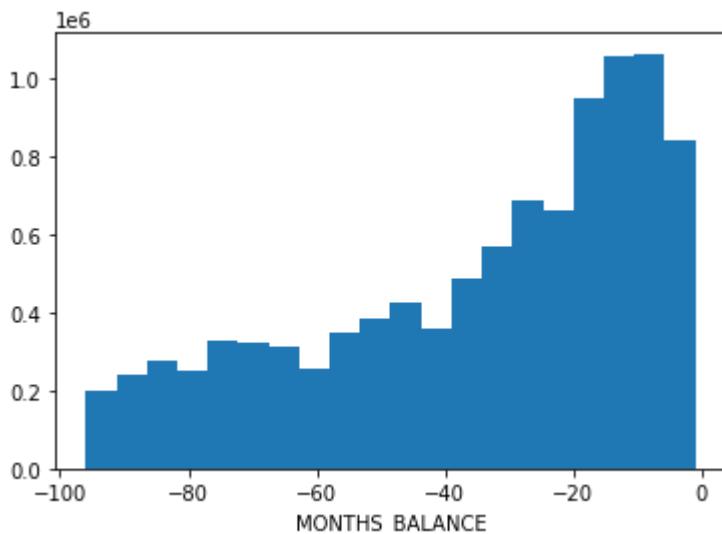
`POS_PERC_INSTL_PNDNG` :: root-transformed
-1.0355844205261202



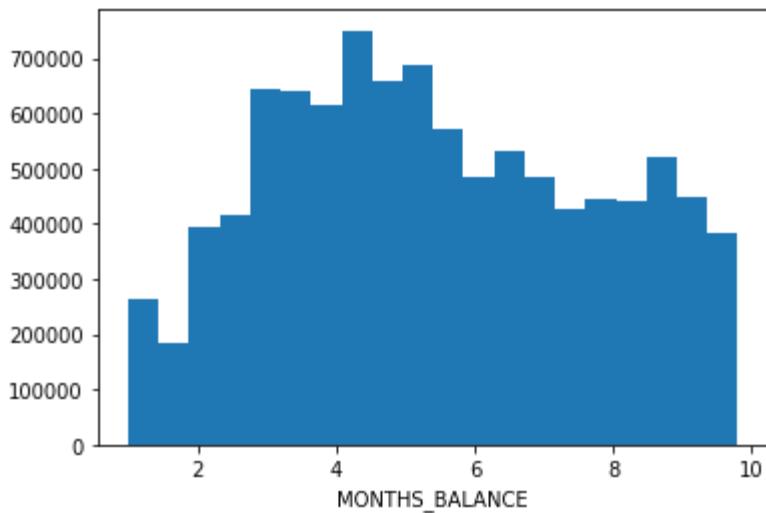
`POS_PERC_INSTL_PNDNG` :: log-transformed
-0.5256347973480694



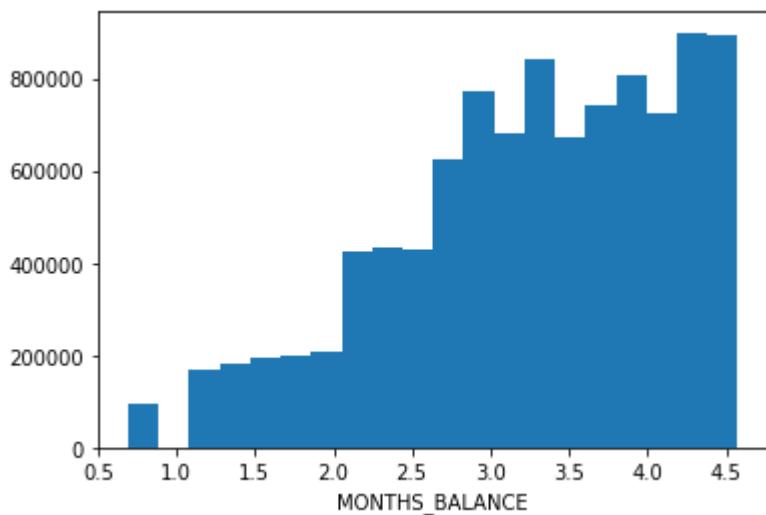
`MONTHS_BALANCE`
0.6727771482709451



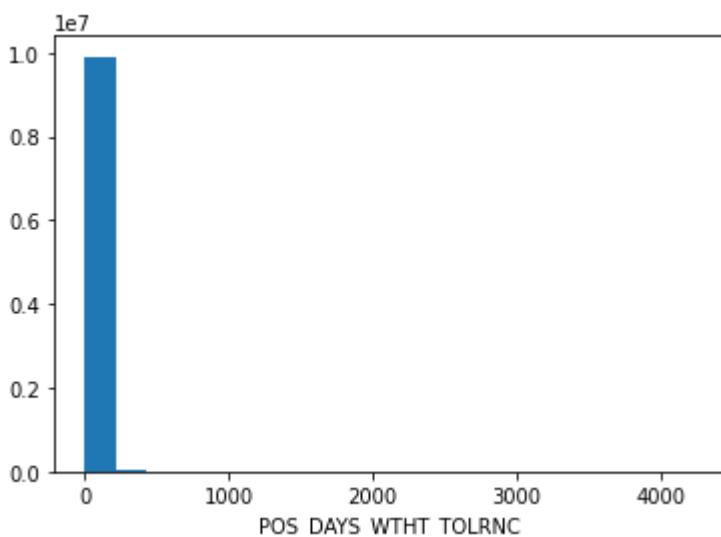
`MONTHS_BALANCE` :: root-transformed
0.12321186323061706



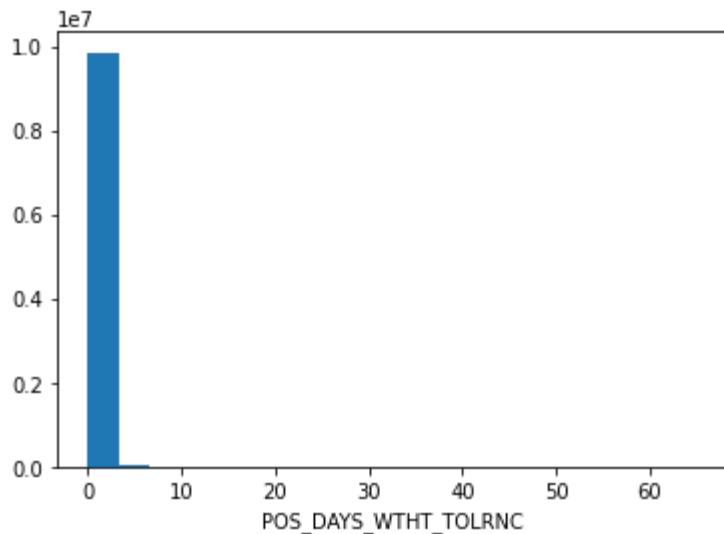
MONTHS_BALANCE :: log-transformed
-0.5836190959029857



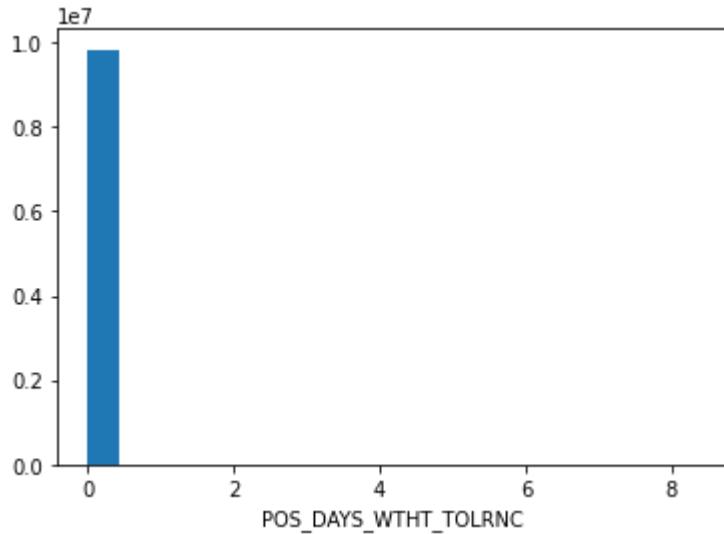
POS_DAYS_WTHT_TOLRNC
15.289244033858985



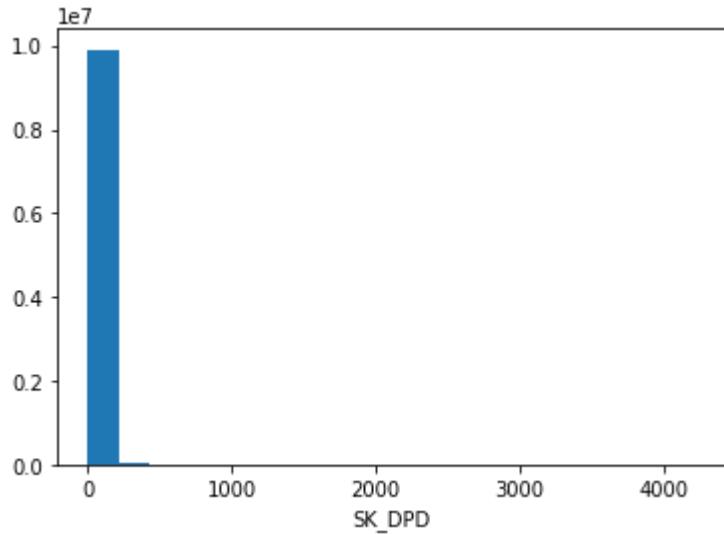
POS_DAYS_WTHT_TOLRNC :: root-transformed
11.11877660706051



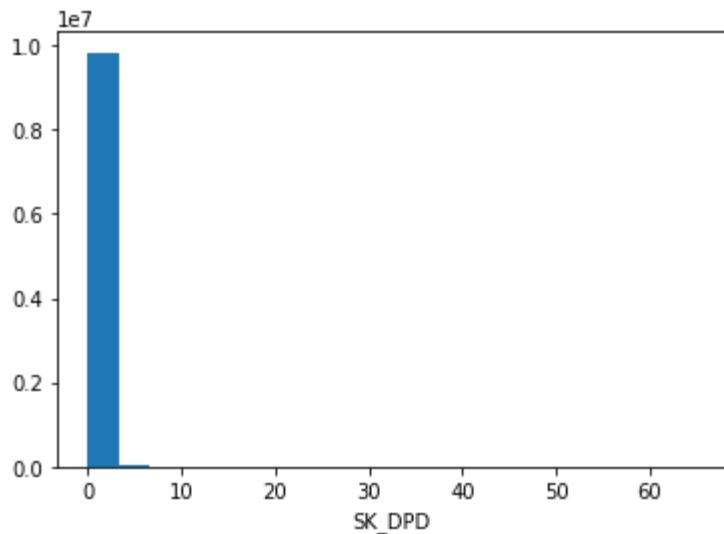
`POS_DAYS_WTHT_TOLRNC` :: log-transformed
8.448415251720528



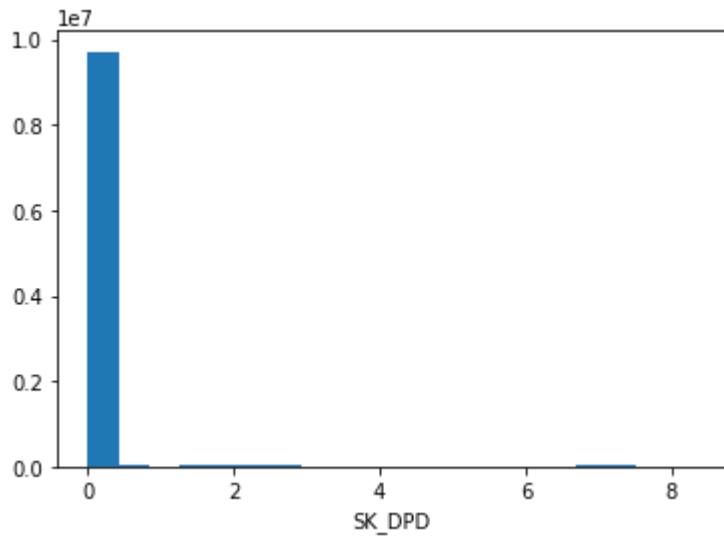
`SK_DPD`
14.8991260746636



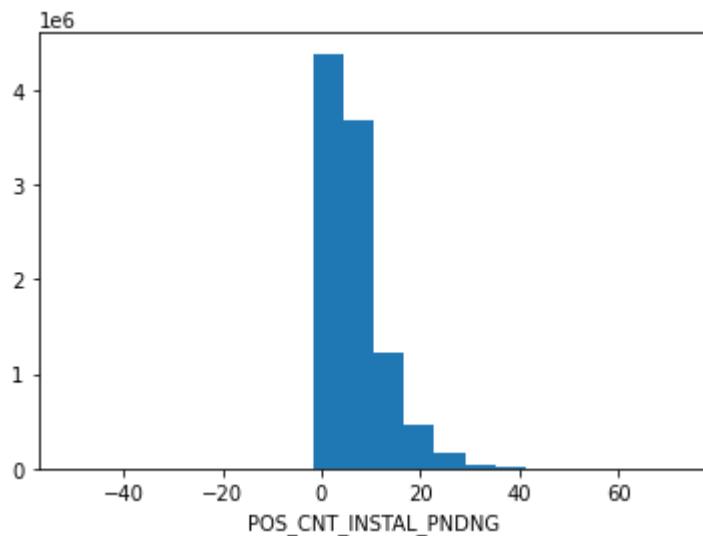
`SK_DPD` :: root-transformed
10.804740778190768



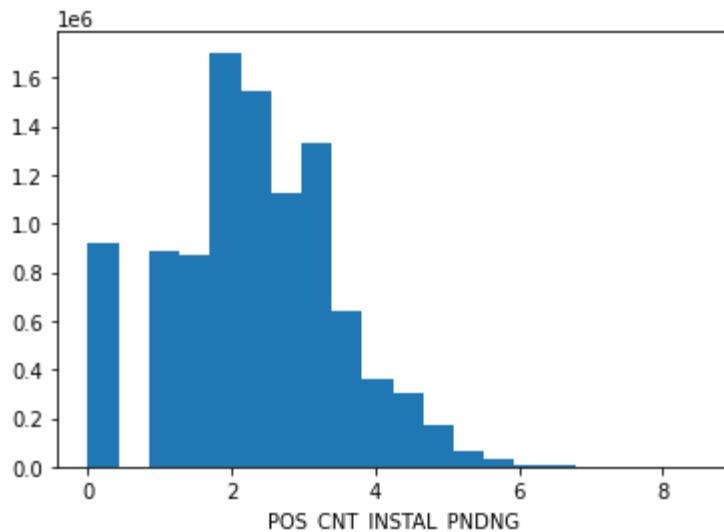
`SK_DPD` :: log-transformed
7.733558753736214



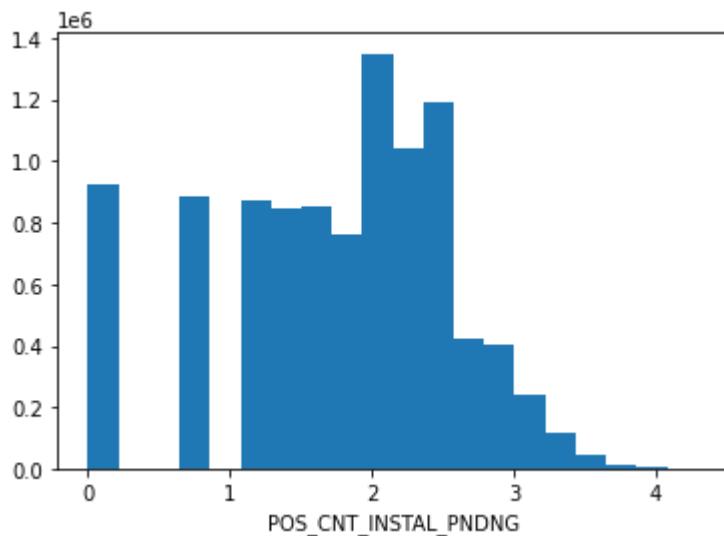
`POS_CNT_INSTAL_PNDNG`
1.6857600133645079



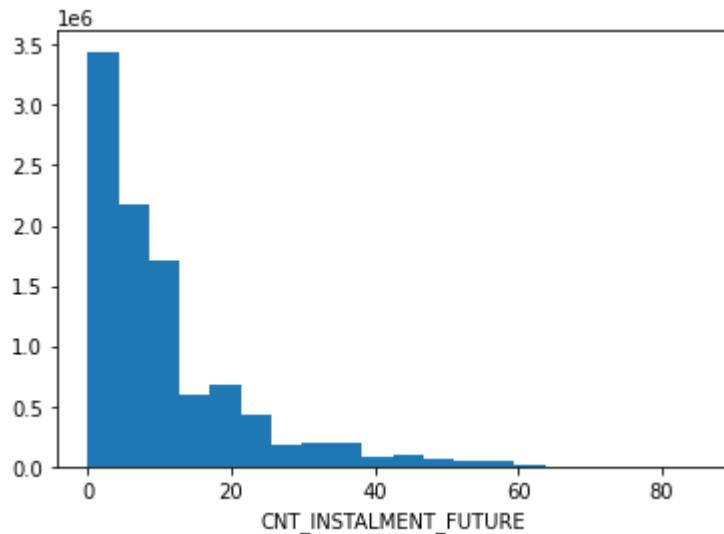
`POS_CNT_INSTAL_PNDNG` :: root-transformed
0.06054248421297969



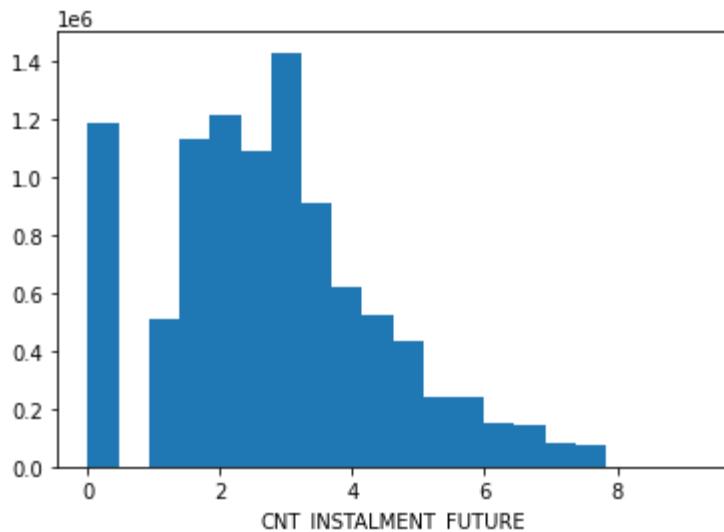
`POS_CNT_INSTAL_PNDNG` :: log-transformed
-0.4295519384783594



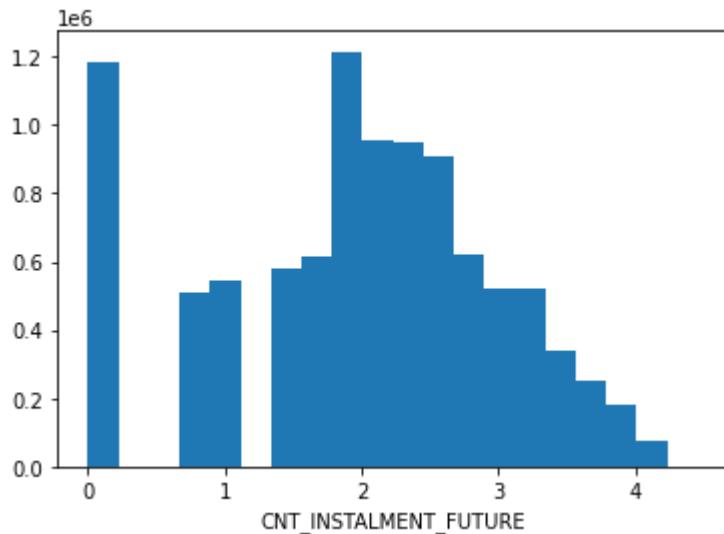
`CNT_INSTALMENT_FUTURE`
1.846745870196922



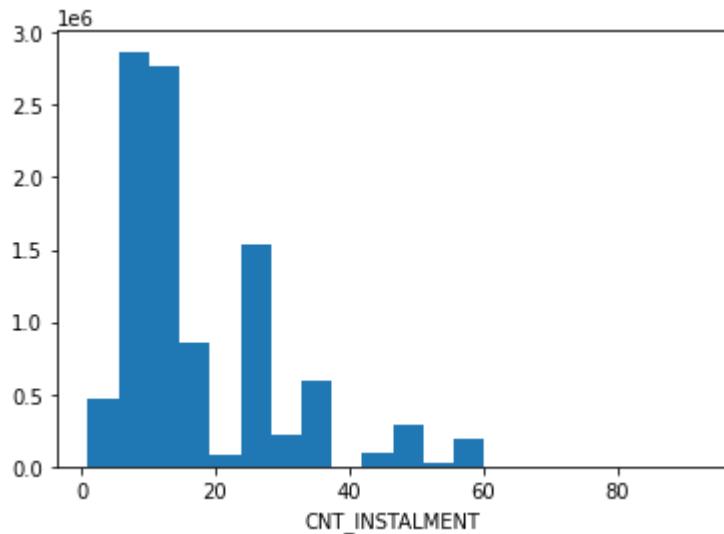
`CNT_INSTALMENT_FUTURE` :: root-transformed
0.37171265275398624



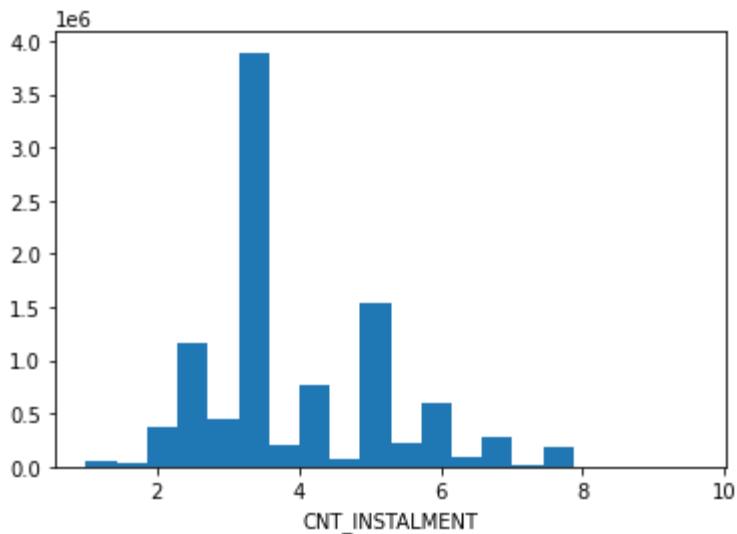
CNT_INSTALMENT_FUTURE :: log-transformed
-0.3699991913333291



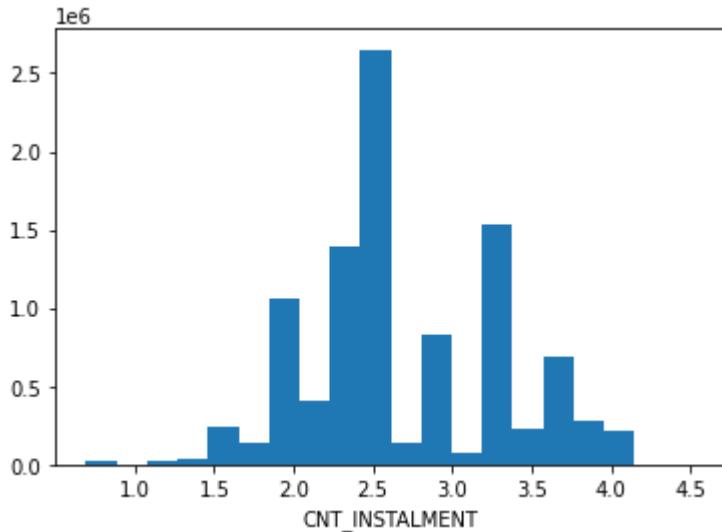
CNT_INSTALMENT
1.6017337881127245



CNT_INSTALMENT :: root-transformed
0.8911225677989296



CNT_INSTALMENT :: log-transformed
0.21052419234378852



```
In [16]:  
#pos = datasets['POS_CASH_balance']  
pos = datasets['POS_CASH_balance']  
pos.columns = ['PS_0_' + col for col in pos.columns]  
pos['SK_ID_CURR'] = pos['PS_0_SK_ID_CURR']  
pos['SK_ID_PREV'] = pos['PS_0_SK_ID_PREV']  
pos.drop(['PS_0_SK_ID_CURR'], axis=1, inplace=True)  
pos.drop(['PS_0_SK_ID_PREV'], axis=1, inplace=True)
```

```
In [17]:  
pos = pos_eda(pos)
```

```

-----
# of ID's: 1
ID's:
['SK_ID_CURR']

-----
# All features: 17
All features:
['PS_O_MONTHS_BALANCE', 'PS_O_NAME_CONTRACT_STATUS', 'PS_O_POS_PERC_INSTL_PNDNG', 'PS_N_PERC_INSTL_PNDNG', 'PS_O_SK_DPD_DEF', 'PS_O_CNT_INSTALMENT', 'PS_O_POS_CNT_INSTAL_PNDNG', 'SK_ID_PREV', 'PS_N_R_CNT_INSTAL_PNDNG', 'PS_O_CNT_INSTALMENT_FUTURE', 'PS_N_R_CNT_INSTALMENT_FUTURE', 'PS_N_DAYS_WTHT_TOLRNC', 'PS_N_CNT_INSTAL_PNDNG', 'PS_N_R_MONTHS_BALANCE', 'PS_O_SK_DPD', 'PS_O_POS_DAYS_WTHT_TOLRNC', 'PS_N_L_CNT_INSTALMENT']

```

Missing data:

	Total	Percent
PS_N_R_CNT_INSTAL_PNDNG	26184	0.261804
PS_O_POS_PERC_INSTL_PNDNG	26184	0.261804
PS_N_PERC_INSTL_PNDNG	26184	0.261804
PS_N_CNT_INSTAL_PNDNG	26184	0.261804
PS_O_POS_CNT_INSTAL_PNDNG	26184	0.261804
PS_O_CNT_INSTALMENT_FUTURE	26087	0.260835
PS_N_R_CNT_INSTALMENT_FUTURE	26087	0.260835
PS_N_L_CNT_INSTALMENT	26071	0.260675
PS_O_CNT_INSTALMENT	26071	0.260675
PS_O_NAME_CONTRACT_STATUS	0	0.000000
SK_ID_PREV	0	0.000000
PS_N_DAYS_WTHT_TOLRNC	0	0.000000
PS_O_SK_DPD_DEF	0	0.000000
PS_N_R_MONTHS_BALANCE	0	0.000000
PS_O_SK_DPD	0	0.000000
PS_O_POS_DAYS_WTHT_TOLRNC	0	0.000000
PS_O_MONTHS_BALANCE	0	0.000000

of Numerical features: 17

Numerical features:

	PS_O_MONTHS_BALANCE	PS_O_CNT_INSTALMENT	PS_O_CNT_INSTALMENT_FUTURE
\			
count	1.000136e+07	9.975287e+06	9.975271e+06
mean	3.501259e+01	1.708965e+01	1.048384e+01
std	2.606657e+01	1.199506e+01	1.110906e+01
min	1.000000e+00	1.000000e+00	0.000000e+00
25%	1.300000e+01	1.000000e+01	3.000000e+00
50%	2.800000e+01	1.200000e+01	7.000000e+00
75%	5.400000e+01	2.400000e+01	1.400000e+01
max	9.600000e+01	9.200000e+01	8.500000e+01

Numerical Statistical Summary:

	PS_O_SK_DPD	PS_O_SK_DPD_DEF	PS_O_POS_PERC_INSTL_PNDNG	\
\				
count	1.000136e+07	9.975287e+06	9.975271e+06	
mean	3.501259e+01	1.708965e+01	1.048384e+01	
std	2.606657e+01	1.199506e+01	1.110906e+01	
min	1.000000e+00	1.000000e+00	0.000000e+00	
25%	1.300000e+01	1.000000e+01	3.000000e+00	
50%	2.800000e+01	1.200000e+01	7.000000e+00	
75%	5.400000e+01	2.400000e+01	1.400000e+01	
max	9.600000e+01	9.200000e+01	8.500000e+01	

PS_O_SK_DPD PS_O_SK_DPD_DEF PS_O_POS_PERC_INSTL_PNDNG \

count	1.000136e+07	1.000136e+07	9.975174e+06
mean	1.160693e+01	6.544684e-01	5.469942e-01
std	1.327140e+02	3.276249e+01	3.303088e-01
min	0.000000e+00	0.000000e+00	0.000000e+00
25%	0.000000e+00	0.000000e+00	2.500000e-01
50%	0.000000e+00	0.000000e+00	5.833333e-01
75%	0.000000e+00	0.000000e+00	8.333333e-01
max	4.231000e+03	3.595000e+03	6.666667e+00
	PS_O_POS_CNT_INSTAL_PNDNG	PS_O_POS_DAYS_WTHT_TOLRNC	SK_ID_CURR \
count	9.975174e+06	1.000136e+07	1.000136e+07
mean	6.605944e+00	1.095246e+01	2.784039e+05
std	5.923767e+00	1.286431e+02	1.027637e+05
min	-5.100000e+01	0.000000e+00	1.000010e+05
25%	2.000000e+00	0.000000e+00	1.895500e+05
50%	5.000000e+00	0.000000e+00	2.786540e+05
75%	9.000000e+00	0.000000e+00	3.674290e+05
max	7.200000e+01	4.231000e+03	4.562550e+05
	SK_ID_PREV	PS_N_PERC_INSL_PNDNG	PS_N_CNT_INSTAL_PNDNG \
count	1.000136e+07	9.975174e+06	9.975174e+06
mean	1.903217e+06	5.469942e-01	6.605944e+00
std	5.358465e+05	3.303088e-01	5.923767e+00
min	1.000001e+06	0.000000e+00	-5.100000e+01
25%	1.434405e+06	2.500000e-01	2.000000e+00
50%	1.896565e+06	5.833333e-01	5.000000e+00
75%	2.368963e+06	8.333333e-01	9.000000e+00
max	2.843499e+06	6.666667e+00	7.200000e+01
	PS_N_DAYS_WTHT_TOLRNC	PS_N_R_CNT_INSTALMENT_FUTURE \	
count	1.000136e+07	9.975271e+06	
mean	1.095246e+01	2.762049e+00	
std	1.286431e+02	1.689652e+00	
min	0.000000e+00	0.000000e+00	
25%	0.000000e+00	1.732051e+00	
50%	0.000000e+00	2.645751e+00	
75%	0.000000e+00	3.741657e+00	
max	4.231000e+03	9.219544e+00	
	PS_N_R_MONTHS_BALANCE	PS_N_R_CNT_INSTAL_PNDNG	PS_N_L_CNT_INSTALMENT
count	1.000136e+07	9.975174e+06	9.975287e+06
mean	5.459486e+00	2.275142e+00	2.710852e+00
std	2.281797e+00	1.199347e+00	5.981731e-01
min	1.000000e+00	0.000000e+00	6.931472e-01
25%	3.605551e+00	1.414214e+00	2.397895e+00
50%	5.291503e+00	2.236068e+00	2.564949e+00
75%	7.348469e+00	3.000000e+00	3.218876e+00
max	9.797959e+00	8.485281e+00	4.532599e+00

of Categorical features: 1
Categorical features:
['PS_O_NAME_CONTRACT_STATUS']

Categorical Statistical Summary:

Categories:

```

PS_O_NAME_CONTRACT_STATUS
0           Active
1           Completed
2           Signed
3           Approved
4   Returned to the store
5           Demand
6           Canceled
7           XNA
8           Amortized debt

-----
# of OHE categorical features: 9
OHE Categorical features: ['PS_O_NAME_CONTRACT_STATUS_Returned to the store',
 'PS_O_NAME_CONTRACT_STATUS_Amortized debt', 'PS_O_NAME_CONTRACT_STATUS_Demand',
 'PS_O_NAME_CONTRACT_STATUS_XNA', 'PS_O_NAME_CONTRACT_STATUS_Approved',
 'PS_O_NAME_CONTRACT_STATUS_Canceled', 'PS_O_NAME_CONTRACT_STATUS_Complete',
 'PS_O_NAME_CONTRACT_STATUS_Active', 'PS_O_NAME_CONTRACT_STATUS_Signed']
-----
df.shape: (10001358, 26)

```

Aggregated Features:

```

df[['PS_O_NAME_CONTRACT_STATUS_Returned to the store', 'PS_O_MONTHS_BALANCE',
 'PS_O_POS_PERC_INSTL_PNDNG', 'PS_O_NAME_CONTRACT_STATUS_Demand', 'PS_N_PECNINSTL_PNDNG',
 'PS_O_NAME_CONTRACT_STATUS_Approved', 'PS_O_SK_DPD_DEF', 'PS_O_CNT_INSTALMENT',
 'PS_O_POS_CNT_INSTAL_PNDNG', 'PS_O_NAME_CONTRACT_STATUS_Canceled',
 'SK_ID_PREV', 'PS_N_R_CNT_INSTAL_PNDNG', 'PS_O_CNT_INSTALMENT_FUTURE',
 'PS_N_R_CNT_INSTALMENT_FUTURE', 'PS_O_NAME_CONTRACT_STATUS_Amortized_debt',
 'PS_N_DAYS_WHT_TOLRNC', 'PS_O_NAME_CONTRACT_STATUS_XNA', 'PS_N_CNT_INSTAL_PNDNG',
 'PS_N_R_MONTHS_BALANCE', 'PS_O_SK_DPD', 'PS_O_POS_DAYS_WHT_TOLRNC',
 'PS_O_NAME_CONTRACT_STATUS_Completed', 'PS_O_NAME_CONTRACT_STATUS_Active',
 'PS_N_L_CNT_INSTALMENT', 'PS_O_NAME_CONTRACT_STATUS_Signed']] [0:5]:
    PS_O_NAME_CONTRACT_STATUS_Returned to the store  PS_O_MONTHS_BALANCE \
0                           0                         31
1                           0                         33
2                           0                         32
3                           0                         35
4                           0                         35

```

```

    PS_O_POS_PERC_INSTL_PNDNG  PS_O_NAME_CONTRACT_STATUS_Demand \
0           0.937500                      0
1           0.972222                      0
2           0.750000                      0
3           0.875000                      0
4           0.972222                      0

```

```

    PS_N_PERC_INSTL_PNDNG  PS_O_NAME_CONTRACT_STATUS_Approved  PS_O_SK_DPD_DEF \
F \
0           0.937500                      0
0
1           0.972222                      0
0
2           0.750000                      0
0
3           0.875000                      0
0
4           0.972222                      0
0

```

	PS_O_CNT_INSTALMENT	PS_O_POS_CNT_INSTAL_PNDNG	\
0	48.0	3.0	
1	36.0	1.0	
2	12.0	3.0	
3	48.0	6.0	
4	36.0	1.0	

	PS_O_NAME_CONTRACT_STATUS_Canceled	...	PS_N_DAYS_WTHT_TOLRNC	\
0	0	...	0	
1	0	...	0	
2	0	...	0	
3	0	...	0	
4	0	...	0	

	PS_O_NAME_CONTRACT_STATUS_XNA	PS_N_CNT_INSTAL_PNDNG	\
0	0	3.0	
1	0	1.0	
2	0	3.0	
3	0	6.0	
4	0	1.0	

	PS_N_R_MONTHS_BALANCE	PS_O_SK_DPD	PS_O_POS_DAYS_WTHT_TOLRNC	\
0	5.567764	0	0	
1	5.744563	0	0	
2	5.656854	0	0	
3	5.916080	0	0	
4	5.916080	0	0	

	PS_O_NAME_CONTRACT_STATUS_Completed	PS_O_NAME_CONTRACT_STATUS_Active	\
0	0	1	
1	0	1	
2	0	1	
3	0	1	
4	0	1	

	PS_N_L_CNT_INSTALMENT	PS_O_NAME_CONTRACT_STATUS_Signed	
0	3.891820	0	
1	3.610918	0	
2	2.564949	0	
3	3.891820	0	
4	3.610918	0	

[5 rows x 25 columns]

Aggregated Features:

- PS_N_CNT_INSTAL_PNDNG_count
- PS_N_CNT_INSTAL_PNDNG_max
- PS_N_CNT_INSTAL_PNDNG_mean
- PS_N_CNT_INSTAL_PNDNG_median
- PS_N_CNT_INSTAL_PNDNG_min
- PS_N_CNT_INSTAL_PNDNG_sum
- PS_N_CNT_INSTAL_PNDNG_var
- PS_N_DAYS_WTHT_TOLRNC_count
- PS_N_DAYS_WTHT_TOLRNC_max
- PS_N_DAYS_WTHT_TOLRNC_mean
- PS_N_DAYS_WTHT_TOLRNC_median
- PS_N_DAYS_WTHT_TOLRNC_min
- PS_N_DAYS_WTHT_TOLRNC_sum
- PS_N_DAYS_WTHT_TOLRNC_var

PS_N_L_CNT_INSTALMENT_count
PS_N_L_CNT_INSTALMENT_max
PS_N_L_CNT_INSTALMENT_mean
PS_N_L_CNT_INSTALMENT_median
PS_N_L_CNT_INSTALMENT_min
PS_N_L_CNT_INSTALMENT_sum
PS_N_L_CNT_INSTALMENT_var
PS_N_PERC_INSTL_PNDNG_count
PS_N_PERC_INSTL_PNDNG_max
PS_N_PERC_INSTL_PNDNG_mean
PS_N_PERC_INSTL_PNDNG_median
PS_N_PERC_INSTL_PNDNG_min
PS_N_PERC_INSTL_PNDNG_sum
PS_N_PERC_INSTL_PNDNG_var
PS_N_R_CNT_INSTALMENT_FUTURE_count
PS_N_R_CNT_INSTALMENT_FUTURE_max
PS_N_R_CNT_INSTALMENT_FUTURE_mean
PS_N_R_CNT_INSTALMENT_FUTURE_median
PS_N_R_CNT_INSTALMENT_FUTURE_min
PS_N_R_CNT_INSTALMENT_FUTURE_sum
PS_N_R_CNT_INSTALMENT_FUTURE_var
PS_N_R_CNT_INSTAL_PNDNG_count
PS_N_R_CNT_INSTAL_PNDNG_max
PS_N_R_CNT_INSTAL_PNDNG_mean
PS_N_R_CNT_INSTAL_PNDNG_median
PS_N_R_CNT_INSTAL_PNDNG_min
PS_N_R_CNT_INSTAL_PNDNG_sum
PS_N_R_CNT_INSTAL_PNDNG_var
PS_N_R_MONTHS_BALANCE_count
PS_N_R_MONTHS_BALANCE_max
PS_N_R_MONTHS_BALANCE_mean
PS_N_R_MONTHS_BALANCE_median
PS_N_R_MONTHS_BALANCE_min
PS_N_R_MONTHS_BALANCE_sum
PS_N_R_MONTHS_BALANCE_var
PS_O_CNT_INSTALMENT_FUTURE_count
PS_O_CNT_INSTALMENT_FUTURE_max
PS_O_CNT_INSTALMENT_FUTURE_mean
PS_O_CNT_INSTALMENT_FUTURE_median
PS_O_CNT_INSTALMENT_FUTURE_min
PS_O_CNT_INSTALMENT_FUTURE_sum
PS_O_CNT_INSTALMENT_FUTURE_var
PS_O_CNT_INSTALMENT_count
PS_O_CNT_INSTALMENT_max
PS_O_CNT_INSTALMENT_mean
PS_O_CNT_INSTALMENT_median
PS_O_CNT_INSTALMENT_min
PS_O_CNT_INSTALMENT_sum
PS_O_CNT_INSTALMENT_var
PS_O_MONTHS_BALANCE_count
PS_O_MONTHS_BALANCE_max
PS_O_MONTHS_BALANCE_mean
PS_O_MONTHS_BALANCE_median
PS_O_MONTHS_BALANCE_min
PS_O_MONTHS_BALANCE_sum
PS_O_MONTHS_BALANCE_var
PS_O_NAME_CONTRACT_STATUS_Active_mean
PS_O_NAME_CONTRACT_STATUS_Active_median
PS_O_NAME_CONTRACT_STATUS_Active_var

PS_0_NAME_CONTRACT_STATUS_Amortized_debt_mean
PS_0_NAME_CONTRACT_STATUS_Amortized_debt_median
PS_0_NAME_CONTRACT_STATUS_Amortized_debt_var
PS_0_NAME_CONTRACT_STATUS_Approved_mean
PS_0_NAME_CONTRACT_STATUS_Approved_median
PS_0_NAME_CONTRACT_STATUS_Approved_var
PS_0_NAME_CONTRACT_STATUS_Canceled_mean
PS_0_NAME_CONTRACT_STATUS_Canceled_median
PS_0_NAME_CONTRACT_STATUS_Canceled_var
PS_0_NAME_CONTRACT_STATUS_Completed_mean
PS_0_NAME_CONTRACT_STATUS_Completed_median
PS_0_NAME_CONTRACT_STATUS_Completed_var
PS_0_NAME_CONTRACT_STATUS_Demand_mean
PS_0_NAME_CONTRACT_STATUS_Demand_median
PS_0_NAME_CONTRACT_STATUS_Demand_var
PS_0_NAME_CONTRACT_STATUS_Returned to the store_mean
PS_0_NAME_CONTRACT_STATUS_Returned to the store_median
PS_0_NAME_CONTRACT_STATUS_Returned to the store_var
PS_0_NAME_CONTRACT_STATUS_Signed_mean
PS_0_NAME_CONTRACT_STATUS_Signed_median
PS_0_NAME_CONTRACT_STATUS_Signed_var
PS_0_NAME_CONTRACT_STATUS_XNA_mean
PS_0_NAME_CONTRACT_STATUS_XNA_median
PS_0_NAME_CONTRACT_STATUS_XNA_var
PS_0_POS_CNT_INSTAL_PNDNG_count
PS_0_POS_CNT_INSTAL_PNDNG_max
PS_0_POS_CNT_INSTAL_PNDNG_mean
PS_0_POS_CNT_INSTAL_PNDNG_median
PS_0_POS_CNT_INSTAL_PNDNG_min
PS_0_POS_CNT_INSTAL_PNDNG_sum
PS_0_POS_CNT_INSTAL_PNDNG_var
PS_0_POS_DAYS_WHT_TOLRNC_count
PS_0_POS_DAYS_WHT_TOLRNC_max
PS_0_POS_DAYS_WHT_TOLRNC_mean
PS_0_POS_DAYS_WHT_TOLRNC_median
PS_0_POS_DAYS_WHT_TOLRNC_min
PS_0_POS_DAYS_WHT_TOLRNC_sum
PS_0_POS_DAYS_WHT_TOLRNC_var
PS_0_POS_PERC_INSL_PNDNG_count
PS_0_POS_PERC_INSL_PNDNG_max
PS_0_POS_PERC_INSL_PNDNG_mean
PS_0_POS_PERC_INSL_PNDNG_median
PS_0_POS_PERC_INSL_PNDNG_min
PS_0_POS_PERC_INSL_PNDNG_sum
PS_0_POS_PERC_INSL_PNDNG_var
PS_0_SK_DPD_DEF_count
PS_0_SK_DPD_DEF_max
PS_0_SK_DPD_DEF_mean
PS_0_SK_DPD_DEF_median
PS_0_SK_DPD_DEF_min
PS_0_SK_DPD_DEF_sum
PS_0_SK_DPD_DEF_var
PS_0_SK_DPD_count
PS_0_SK_DPD_max
PS_0_SK_DPD_mean
PS_0_SK_DPD_median
PS_0_SK_DPD_min
PS_0_SK_DPD_sum
PS_0_SK_DPD_var

SK_ID_PREV_count

Aggregated Data:

	count	mean	\
PS_O_POS_PERC_INSTL_PNDNG_var	336858.0	1.073225e-01	
PS_O_NAME_CONTRACT_STATUS_Returned to the store...	336880.0	8.026822e-04	
PS_O_NAME_CONTRACT_STATUS_XNA_var	336880.0	3.676374e-08	
PS_O_NAME_CONTRACT_STATUS_Approved_mean	337252.0	5.954276e-04	
PS_O_CNT_INSTALMENT_FUTURE_mean	337224.0	9.176876e+00	
...	
PS_O_POS_CNT_INSTAL_PNDNG_mean	337224.0	5.478194e+00	
PS_O_CNT_INSTALMENT_max	337224.0	2.042600e+01	
PS_N_DAYS_WTHT_TOLRNC_min	337252.0	4.597749e-02	
PS_N_R_MONTHS_BALANCE_count	337252.0	2.965544e+01	
PS_O_NAME_CONTRACT_STATUS_Canceled_median	337252.0	0.000000e+00	
	std	min	\
PS_O_POS_PERC_INSTL_PNDNG_var	0.040023	0.000000	
PS_O_NAME_CONTRACT_STATUS_Returned to the store...	0.008835	0.000000	
PS_O_NAME_CONTRACT_STATUS_XNA_var	0.000015	0.000000	
PS_O_NAME_CONTRACT_STATUS_Approved_mean	0.007529	0.000000	
PS_O_CNT_INSTALMENT_FUTURE_mean	6.501034	0.000000	
...	
PS_O_POS_CNT_INSTAL_PNDNG_mean	2.635582	-0.736842	
PS_O_CNT_INSTALMENT_max	13.909296	1.000000	
PS_N_DAYS_WTHT_TOLRNC_min	4.625809	0.000000	
PS_N_R_MONTHS_BALANCE_count	24.531971	1.000000	
PS_O_NAME_CONTRACT_STATUS_Canceled_median	0.000000	0.000000	
	25%	50%	\
PS_O_POS_PERC_INSTL_PNDNG_var	0.091667	0.105324	
PS_O_NAME_CONTRACT_STATUS_Returned to the store...	0.000000	0.000000	
PS_O_NAME_CONTRACT_STATUS_XNA_var	0.000000	0.000000	
PS_O_NAME_CONTRACT_STATUS_Approved_mean	0.000000	0.000000	
PS_O_CNT_INSTALMENT_FUTURE_mean	5.000000	6.989411	
...	
PS_O_POS_CNT_INSTAL_PNDNG_mean	3.685714	5.000000	
PS_O_CNT_INSTALMENT_max	12.000000	12.000000	
PS_N_DAYS_WTHT_TOLRNC_min	0.000000	0.000000	
PS_N_R_MONTHS_BALANCE_count	12.000000	22.000000	
PS_O_NAME_CONTRACT_STATUS_Canceled_median	0.000000	0.000000	
	75%	max	
PS_O_POS_PERC_INSTL_PNDNG_var	0.120833	1.95109	
PS_O_NAME_CONTRACT_STATUS_Returned to the store...	0.000000	0.50000	
PS_O_NAME_CONTRACT_STATUS_XNA_var	0.000000	0.00625	
PS_O_NAME_CONTRACT_STATUS_Approved_mean	0.000000	1.00000	
PS_O_CNT_INSTALMENT_FUTURE_mean	11.666667	60.00000	
...	
PS_O_POS_CNT_INSTAL_PNDNG_mean	6.625000	36.09589	
PS_O_CNT_INSTALMENT_max	24.000000	92.00000	
PS_N_DAYS_WTHT_TOLRNC_min	0.000000	882.00000	
PS_N_R_MONTHS_BALANCE_count	39.000000	295.00000	
PS_O_NAME_CONTRACT_STATUS_Canceled_median	0.000000	0.00000	

[133 rows x 8 columns]

In [18]:

```
pos['PS_0_SK_ID_PREV_count'] = pos['SK_ID_PREV_count']
pos.drop(['SK_ID_PREV_count'],axis=1,inplace=True)
datasets_transformed['POS_CASH_balance'] = pos
#pos.to_csv(os.getcwd() + DATA_DIR + 'pos_agg_data_tr.csv')
```

PREVAPP - Previous Application

1. This function is specifically for POS table. Below are the initial pre-processing steps done before passing this table into the pipeline.

- Create a drop list.
 - Attributes that will be dropped will be added to this list and all columns will be deleted before passing the dataframe into the eda function.
- Create new features. based on analysis, 6 new features were created:
 - Count of approved previous application.
 - Count of Rejected previous applications.
 - Difference: Amount requested in application - Actual credit amount.
 - Ratio - Ratio of application amount to amount credited.
 - Ratio - Ratio of amount credited to amount annuity
 - Ratio - Ratio of down payment to amount credited.
- There are number of attributes which are in days and amount. For that, we created list of columns which ends with 'DAYS' and 'AMT'
- Analysis on attributes with date shows that many are capped to 365243, which is 100 years. This looks to be added by the system and not user data. This will be replaced by nan and later imputed.
- Another observation was that days columns are marked as negative and so the absolute values were calculated and used.
- Added below attributes to droplist:
 - WEEKDAY_APPR_PROCESS_START
 - HOUR_APPR_PROCESS_START
- Any column or row with more than 70% of its data as null will be deleted from the dataframe as the threshold is set to .7.
- Once processed, store the transformed csv file. Benefit of this is that we can then pass it directly to model for merging into application train/test table. We do not have to perform expensive EDA/ETL/Transformation everytime we want to process the same data.
- **Features Transformation:** Below are the features which are transformed.
(PS_N_R_XX: New feature with root transformation. **PS_N_L_XX:** New feature with log transformation. And **PS_N_XX** stands for new feature added.

```
    #Adding new features
    prevapp['PA_N_APPR_CNT'] =
prevapp['PA_O_NAME_CONTRACT_STATUS'].map(lambda x: 1 if (x ==
'Approved') else 0)
    prevapp['PA_N_REJ_CNT'] =
-----[PA_O_NAME_CONTRACT_STATUS]----- 1 if ...
```

```

prevapp['PA_O_NAME_CONTRACT_STATUS'].map(lambda x: 1 if (x == 'Refused') else 0)
    prevapp['PA_N_APCTN_CRDT_DIFF'] =
prevapp['PA_O_AMT_APPLICATION'] - prevapp['PA_O_AMT_CREDIT']
    prevapp['PA_N_APCTN_CRDT_RATIO'] =
prevapp['PA_O_AMT_APPLICATION'] / prevapp['PA_O_AMT_CREDIT']
    prevapp['PA_N_CRDT_ANNUITY_RATIO'] =
prevapp['PA_O_AMT_CREDIT']/prevapp['PA_O_AMT_ANNUITY']
    prevapp['PA_N_DWN_PYMNT_CRDT_RATIO'] =
prevapp['PA_O_AMT_DOWN_PAYMENT'] / prevapp['PA_O_AMT_CREDIT']

```

```

__#Root__
    prevapp['PA_N_R_RATE_DOWN_PAYMENT'] =
np.sqrt(prevapp['PA_O_RATE_DOWN_PAYMENT'].abs())
    prevapp['PA_N_R_AMT_CREDIT'] =
np.sqrt(prevapp['PA_O_AMT_CREDIT'].abs())
    prevapp['PA_N_R_AMT_ANNUITY'] =
np.sqrt(prevapp['PA_O_AMT_ANNUITY'].abs())
    prevapp['PA_N_R_CNT_PAYMENT'] =
np.sqrt(prevapp['PA_O_CNT_PAYMENT'].abs())
    prevapp['PA_N_R_AMT_GOODS_PRICE'] =
np.sqrt(prevapp['PA_O_AMT_GOODS_PRICE'].abs())
    prevapp['PA_N_R_DWN_PYMNT_CRDT_RATIO'] =
np.sqrt(prevapp['PA_N_DWN_PYMNT_CRDT_RATIO'].abs())
    prevapp['PA_N_R_DAYS_DECISION'] =
np.sqrt(prevapp['PA_O_DAYS_DECISION'].abs())

```

```

__#Log__
    prevapp['PA_N_L_CRDT_ANNUITY_RATIO'] =
np.log(prevapp['PA_N_CRDT_ANNUITY_RATIO'].abs()+1)
    prevapp['PA_N_L_DAYS_TERMINATION'] =
np.log(prevapp['PA_O_DAYS_TERMINATION'].abs()+1)
    prevapp['PA_N_L_AMT_DOWN_PAYMENT'] =
np.log(prevapp['PA_O_AMT_DOWN_PAYMENT'].abs()+1)
    prevapp['PA_N_L_SELLERPLACE_AREA'] =
np.log(prevapp['PA_O_SELLERPLACE_AREA'].abs()+1)
    prevapp['PA_N_L_AMT_APPLICATION'] =
np.log(prevapp['PA_O_AMT_APPLICATION'].abs()+1)
    prevapp['PA_N_L_PREV_APCTN_CRDT_DIFF'] =
np.log(prevapp['PA_N_APCTN_CRDT_DIFF'].abs()+1)

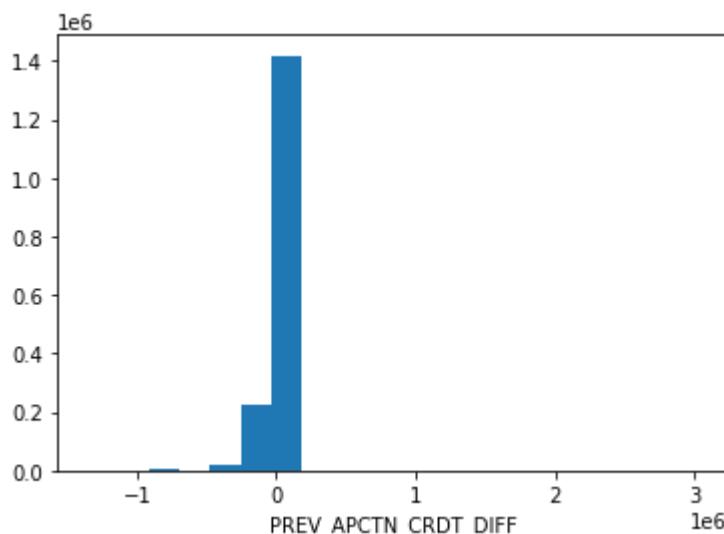
```

Feature transformation Viz

In [19]:

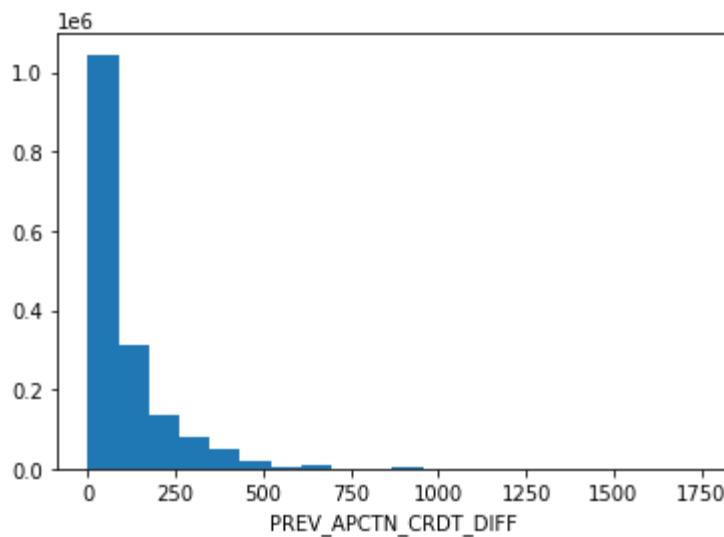
```
for i,ds_name in enumerate(datasets.keys()):  
  
    if ds_name.lower() in ("previous_application"):  
        ps = datasets[ds_name]  
  
        ps['PREV_APCTN_CRDT_DIFF'] = ps['AMT_APPLICATION'] - ps['AMT_CREDIT']  
        ps['PREV_APCTN_CRDT_RATIO'] = ps['AMT_APPLICATION'] / ps['AMT_CREDIT']  
        ps['PREV_CRDT_ANNUITY_RATIO'] = ps['AMT_CREDIT']/ps['AMT_ANNUITY']  
        ps['PREV_DWN_PYMNT_CRDT_RATIO'] = ps['AMT_DOWN_PAYMENT'] / ps['AMT_CREDIT']  
        print("-----")  
        id_cols,num_cols,_,_ = id_num_cat_feature(ps, text = False)  
        id_cols.append('SK_ID_PREV')  
        new_cols = ['PREV_APCTN_CRDT_DIFF','PREV_APCTN_CRDT_RATIO','PREV_CRDT_ANNUITY_RATIO']  
        cols= list(set(num_cols) - set(id_cols) )  
        #cols = ['PS_N_PERC_INSTL_PNDNG','PS_N_CNT_INSTAL_PNDNG','PS_N_DAYS_BRDING']  
        ps.replace([np.inf, -np.inf], 0, inplace=True)  
        for i in cols:  
            print('---')  
            print(f'{i}')  
            print(((ps[i].abs())).skew())  
            plt.hist(ps[i], bins=20)  
            plt.xlabel(i)  
            plt.show()  
            print('')  
            print(f'{i} :: root-transformed')  
            print((np.sqrt(ps[i].abs())).skew())  
            plt.hist(np.sqrt(ps[i].abs()), bins=20)  
            plt.xlabel(i)  
            plt.show()  
            print('')  
            print(f'{i} :: log-transformed')  
            print((np.log(ps[i].abs() + 1)).skew())  
            plt.hist(np.log(ps[i].abs() + 1), bins=20)  
            plt.xlabel(i)  
            plt.show()
```

PREV_APCTN_CRDT_DIFF
6.344083386658649



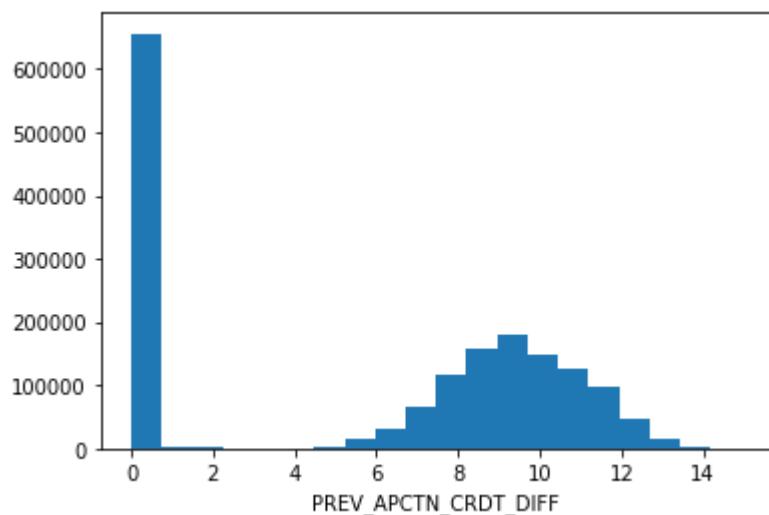
PREV_APCTN_CRDT_DIFF :: root-transformed

2.217741270233417



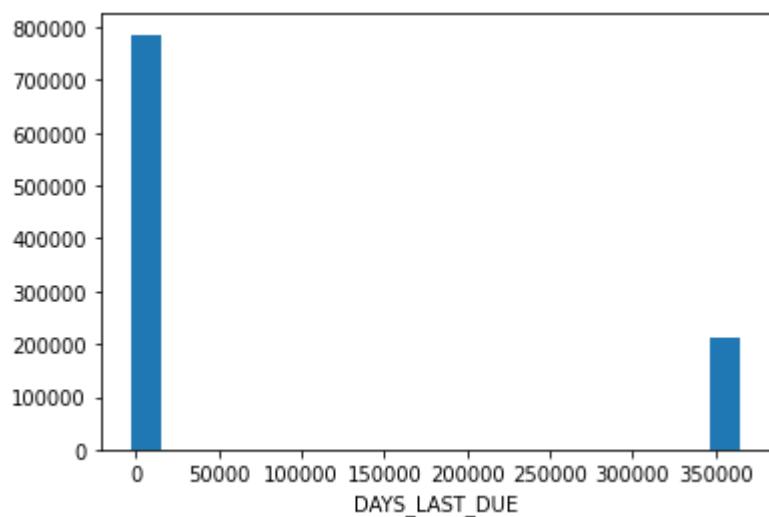
PREV_APCTN_CRDT_DIFF :: log-transformed

-0.2217185365132907

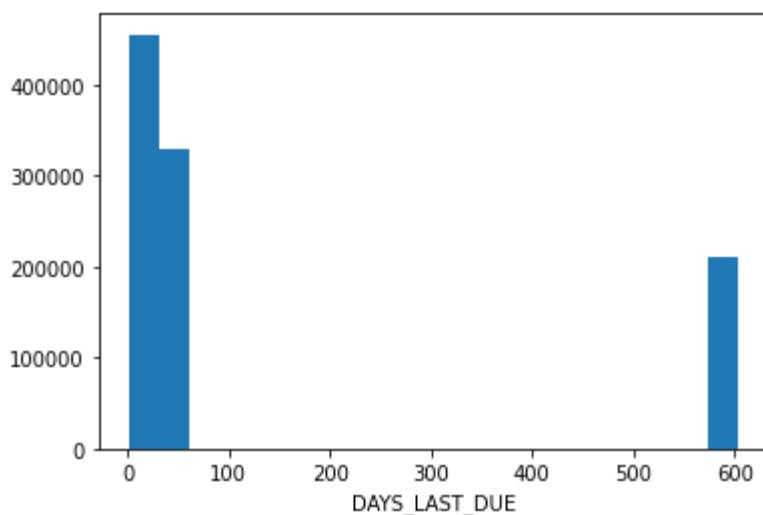


DAYS_LAST_DUE

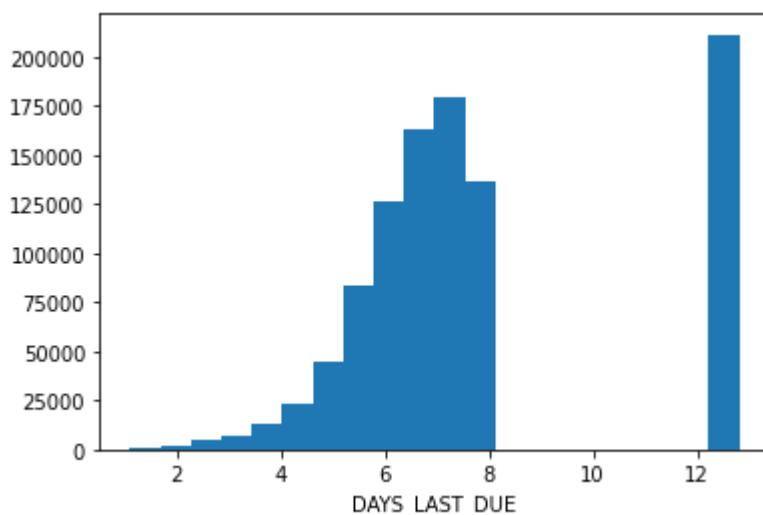
1.4104719224863964



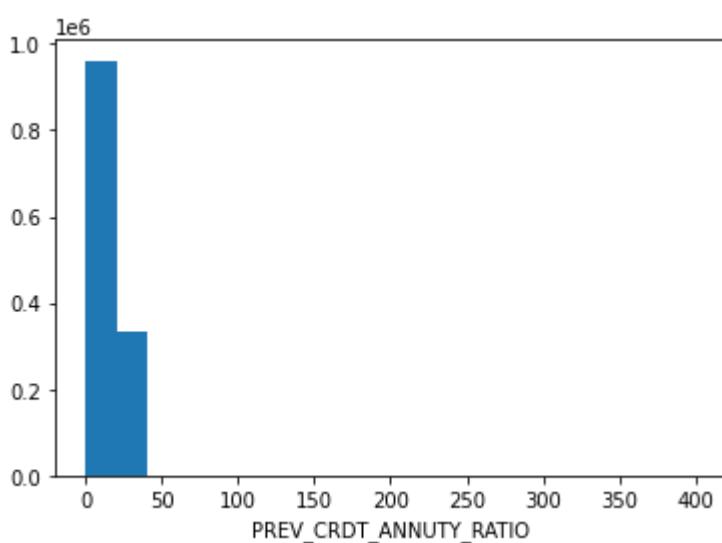
DAYS_LAST_DUE :: root-transformed
1.4021645628401813



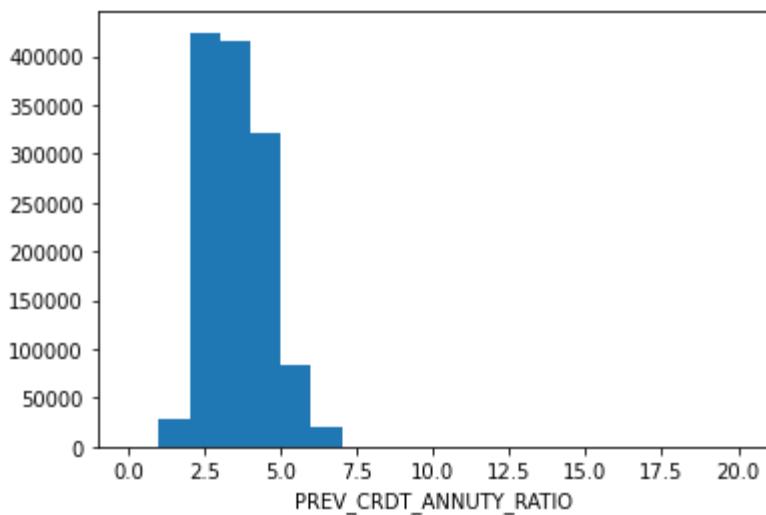
DAYS_LAST_DUE :: log-transformed
0.9101952485919501



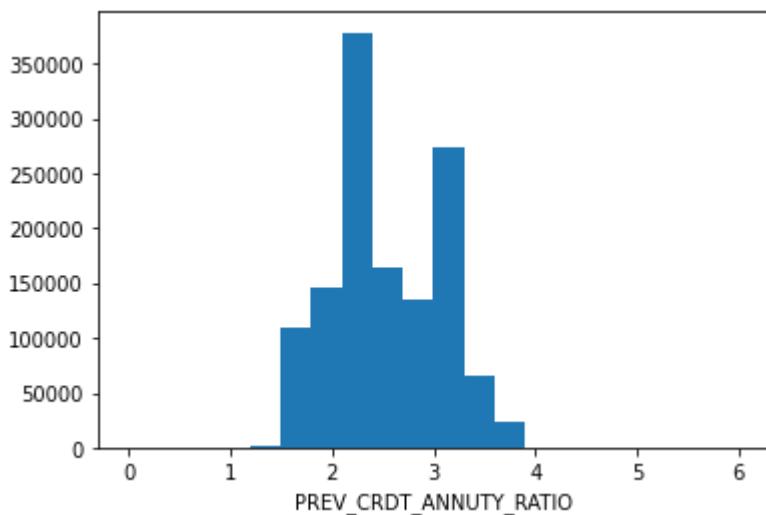
PREV_CRDT_ANNUITY_RATIO
2.876249430215522



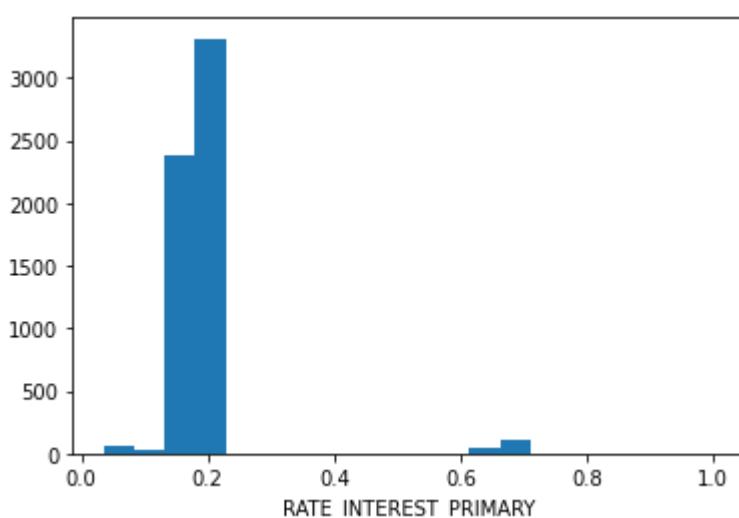
PREV_CRDT_ANNUITY_RATIO :: root-transformed
0.7432657759849333



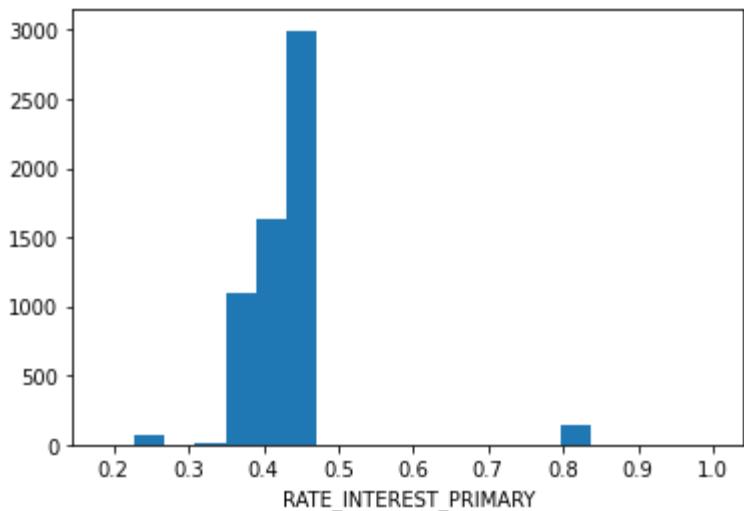
PREV_CRDT_ANNUITY_RATIO :: log-transformed
0.18850517321902646



RATE_INTEREST_PRIMARY
5.1982040214378245

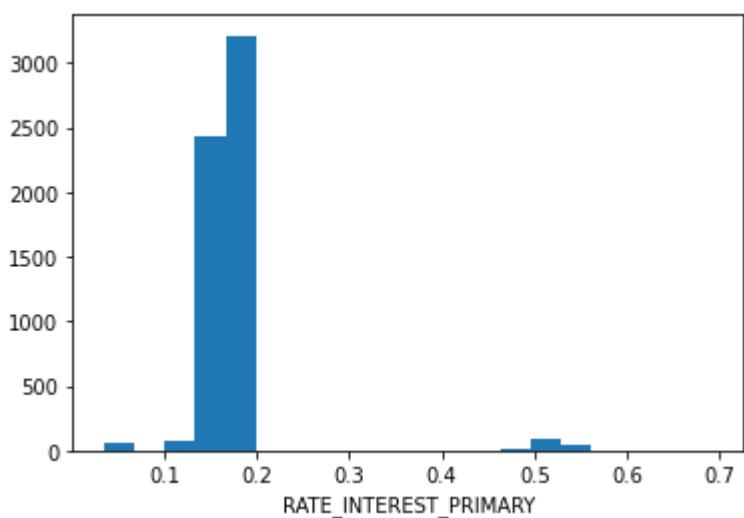


RATE_INTEREST_PRIMARY :: root-transformed
4.181220853960075



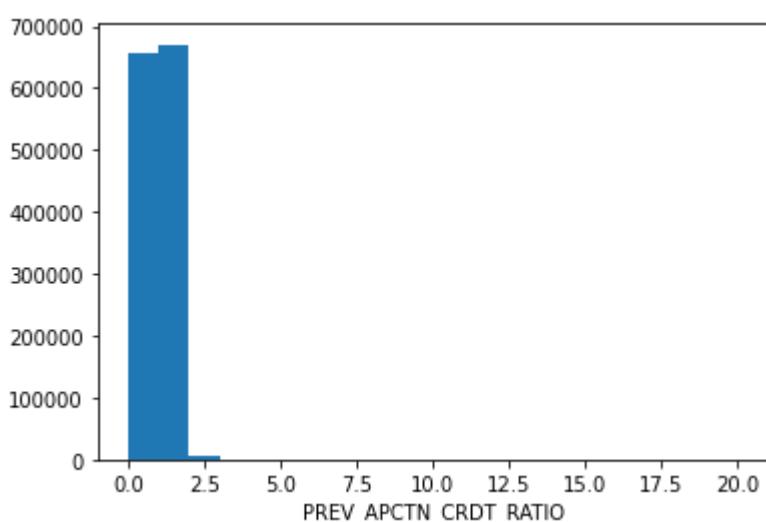
RATE_INTEREST_PRIMARY :: log-transformed

4.851854576983604



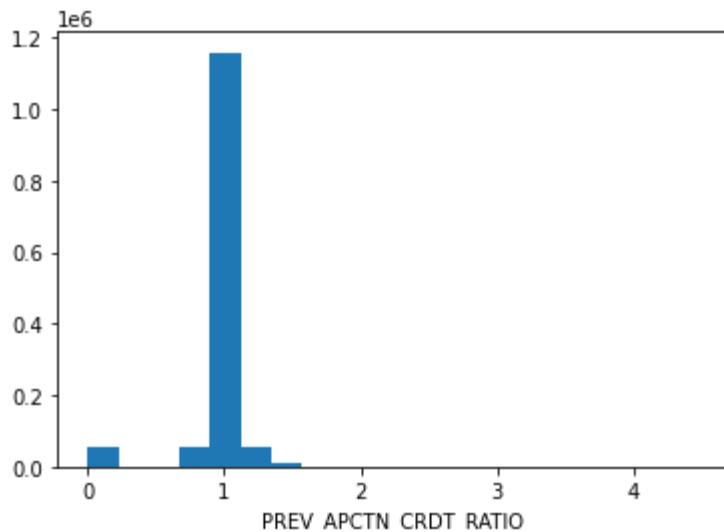
PREV_APCTN_CRDT_RATIO

3.5498941376692477

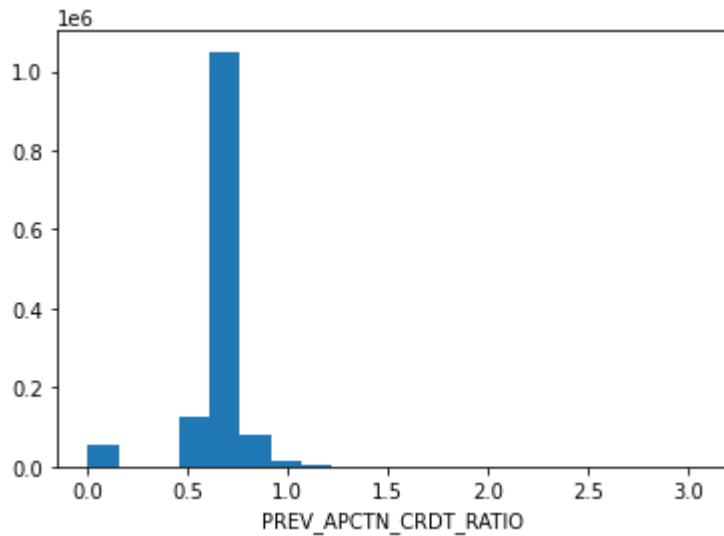


PREV_APCTN_CRDT_RATIO :: root-transformed

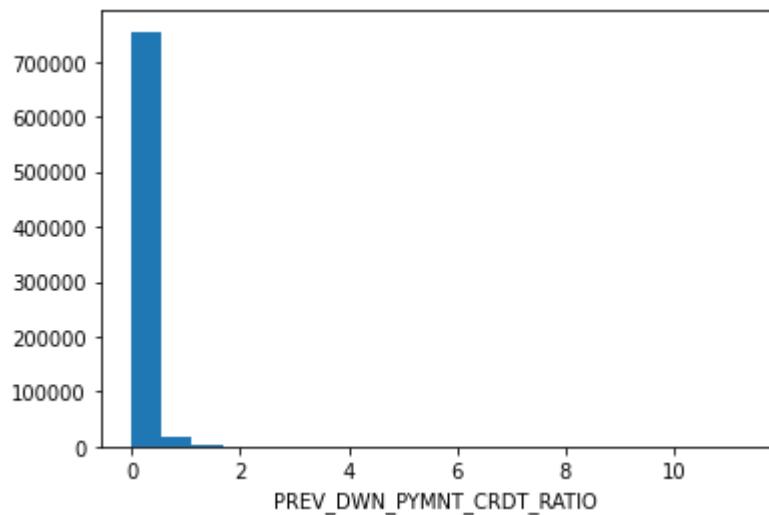
-3.3436718450485863



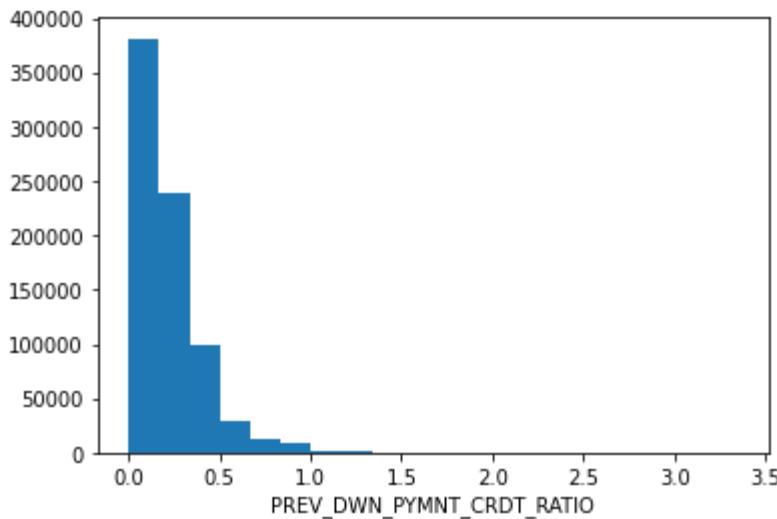
PREV_APCTN_CRDT_RATIO :: log-transformed
-2.465732653158542



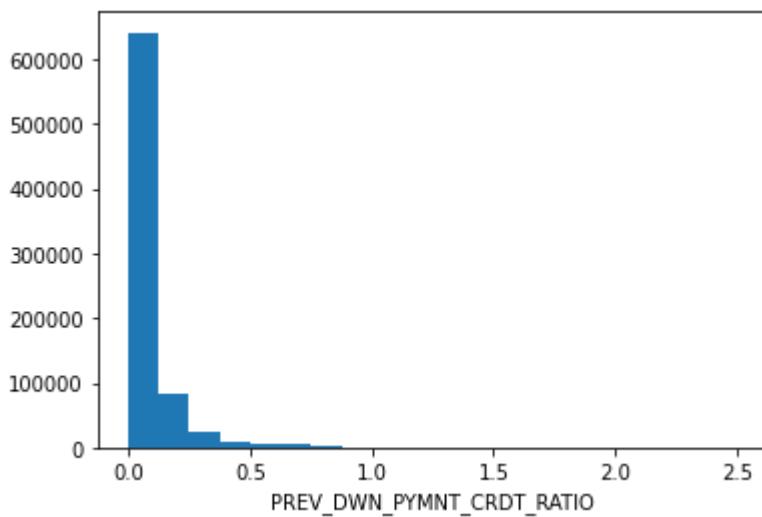
PREV_DWN_PYMNT_CRDT_RATIO
7.0792232487975095



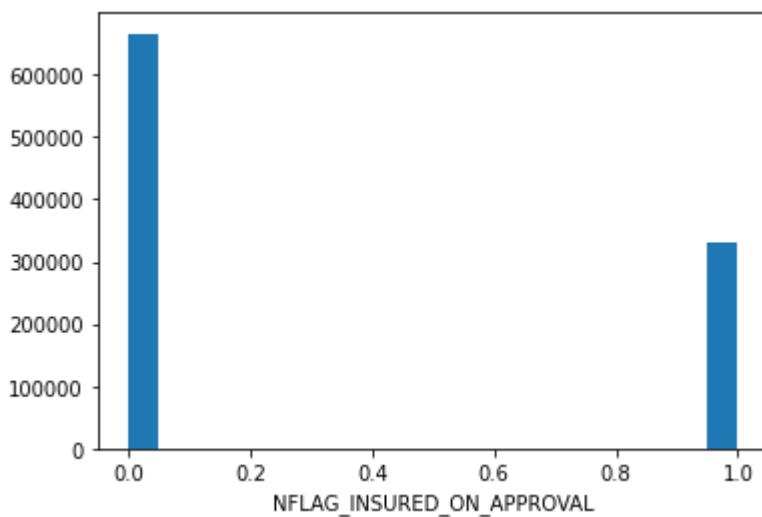
PREV_DWN_PYMNT_CRDT_RATIO :: root-transformed
1.0905092083461416



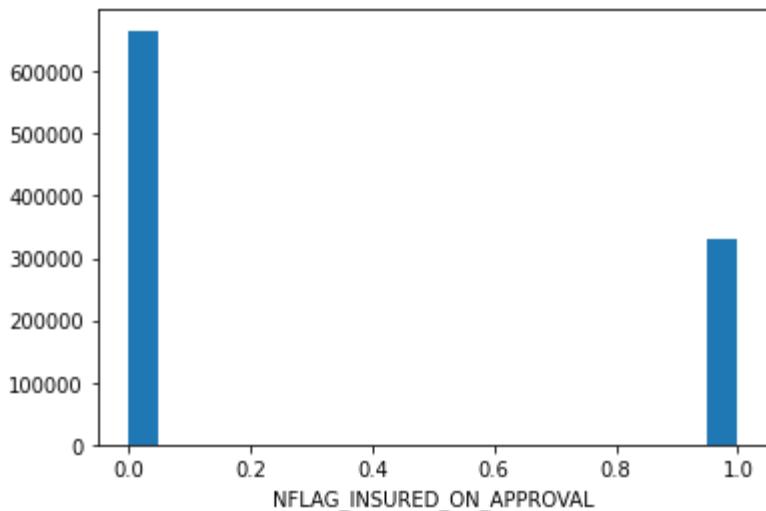
`PREV_DWN_PYMNT_CRDT_RATIO` :: log-transformed
3.220238091689389



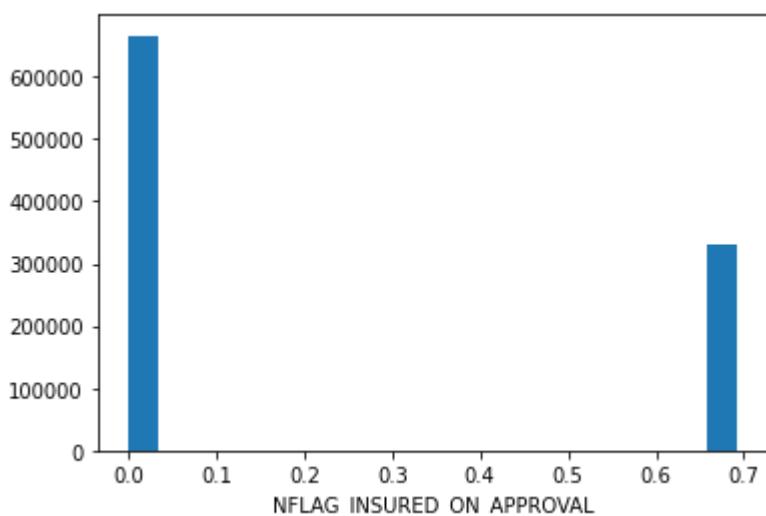
`NFLAG_INSURED_ON_APPROVAL`
0.7107536014428991



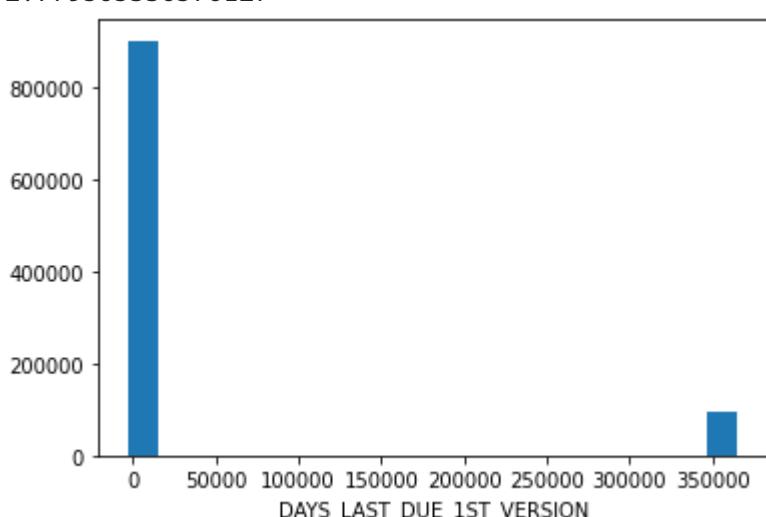
`NFLAG_INSURED_ON_APPROVAL` :: root-transformed
0.7107536014428991



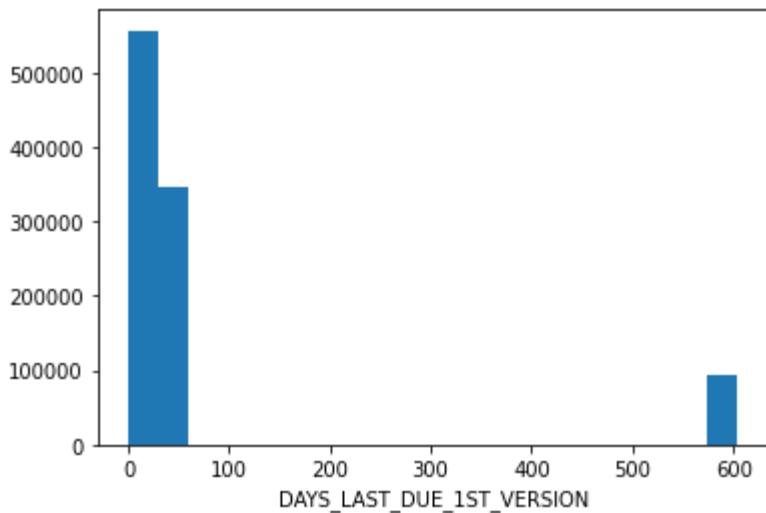
`NFLAG_INSURED_ON_APPROVAL` :: log-transformed
0.7107536014429007



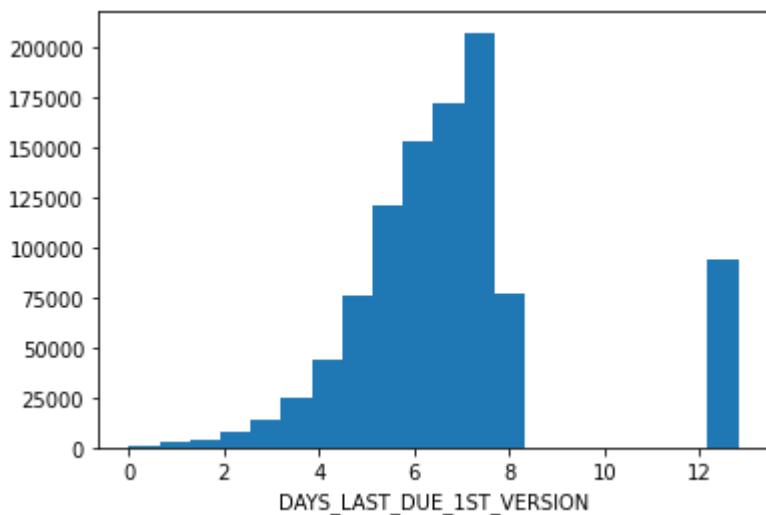
`DAYSLASTDUE1STVERSION`
2.779565336576127



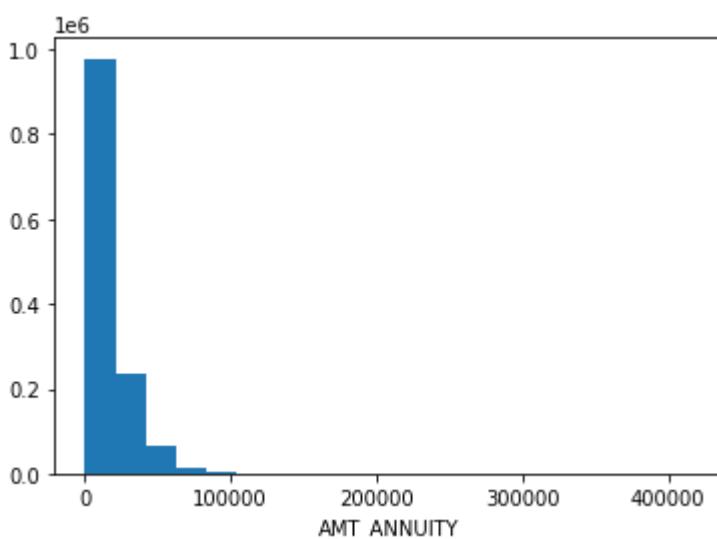
`DAYSLASTDUE1STVERSION` :: root-transformed
2.750722004740827



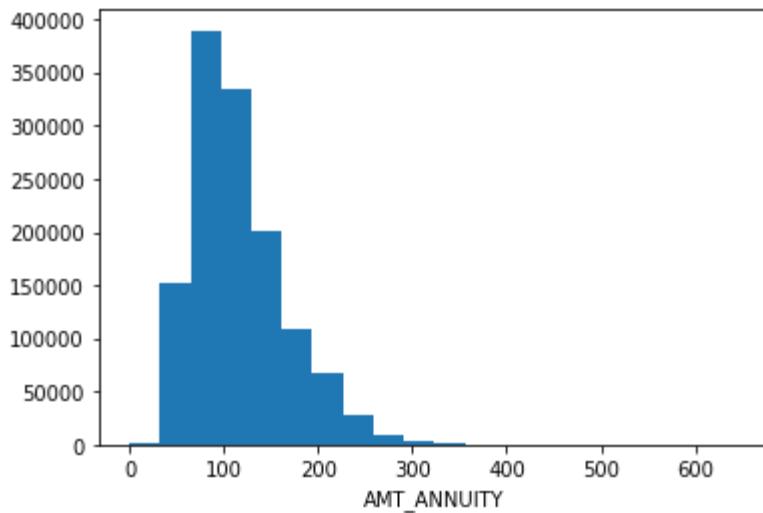
`DAYS_LAST_DUE_1ST_VERSION` :: log-transformed
1.2142534247824746



`AMT_ANNUITY`
2.6925715126729455

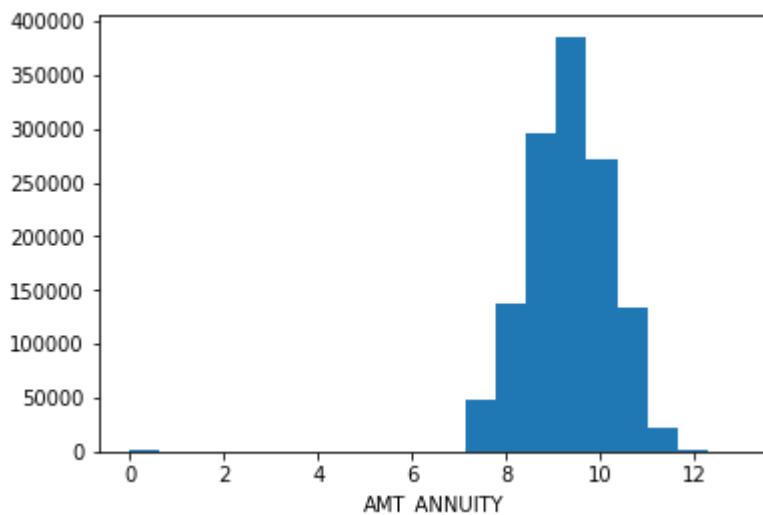


`AMT_ANNUITY` :: root-transformed
1.076652567884181



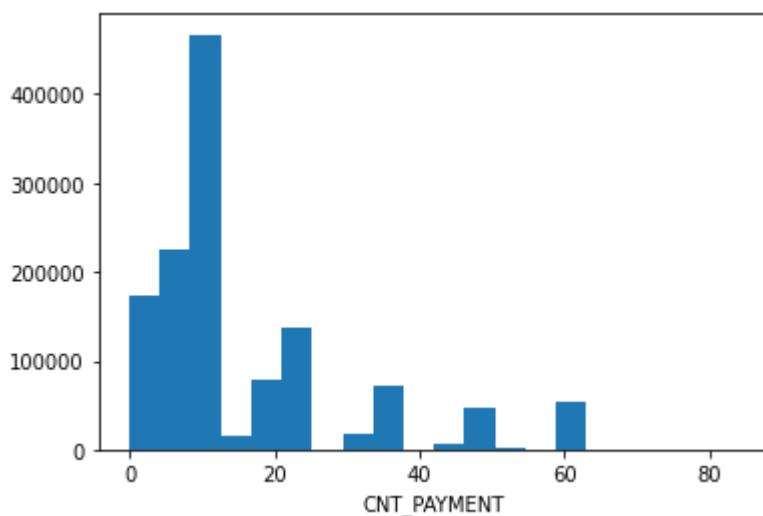
AMT_ANNUITY :: log-transformed

-1.3632213189255926



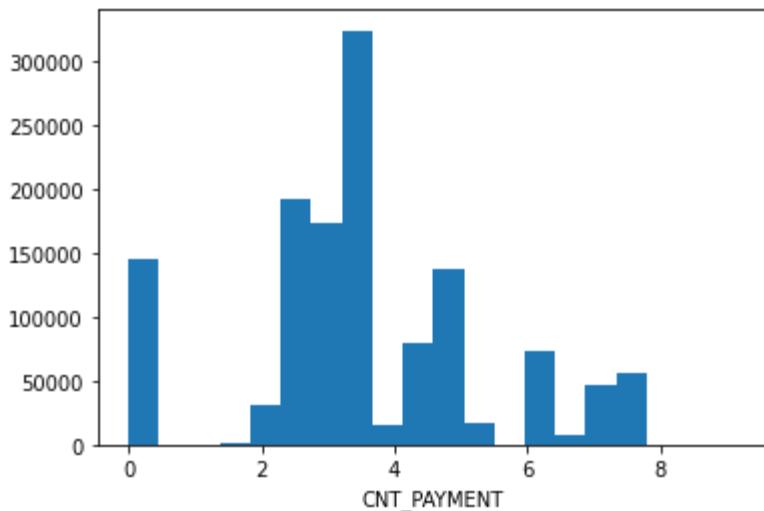
CNT_PAYMENT

1.5314029823509057

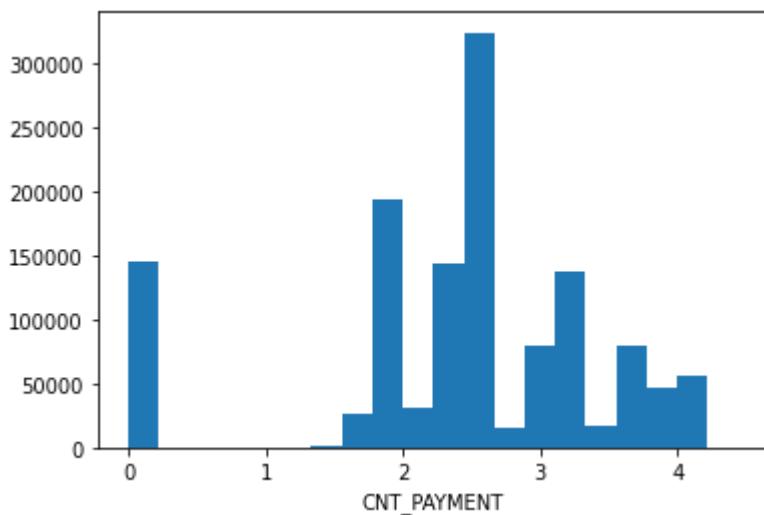


CNT_PAYMENT :: root-transformed

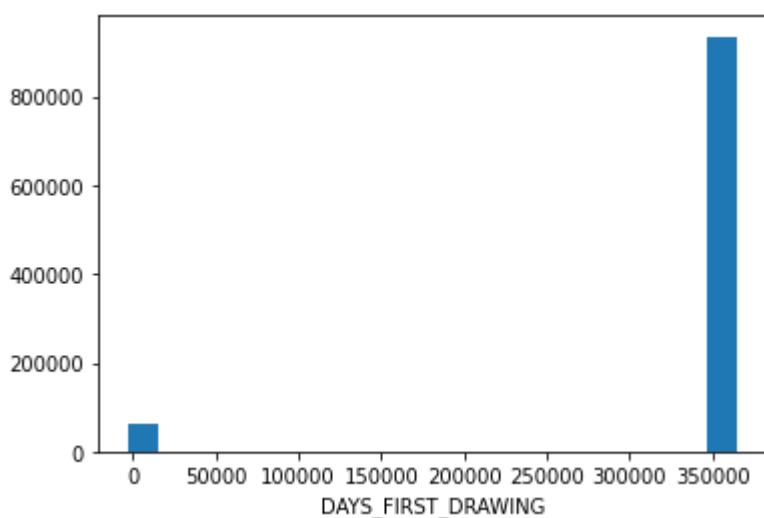
0.10327531988542854



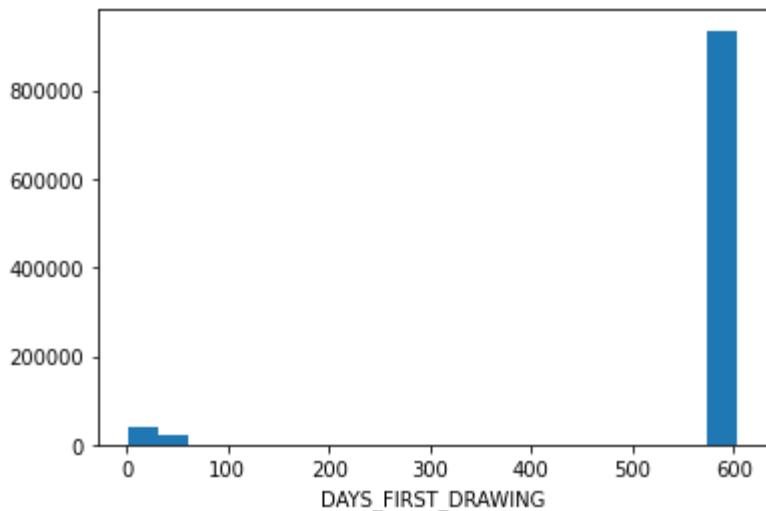
CNT_PAYMENT :: log-transformed
-0.9644992627507585



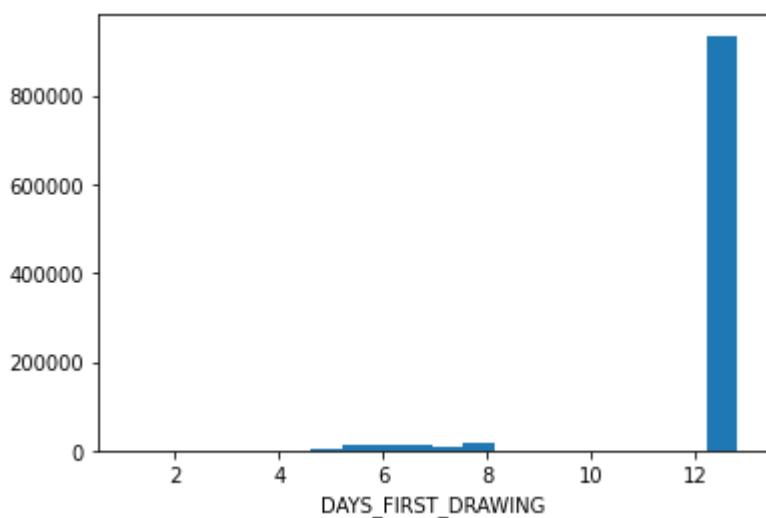
DAYS_FIRST_DRAWING
-3.6013431058889798



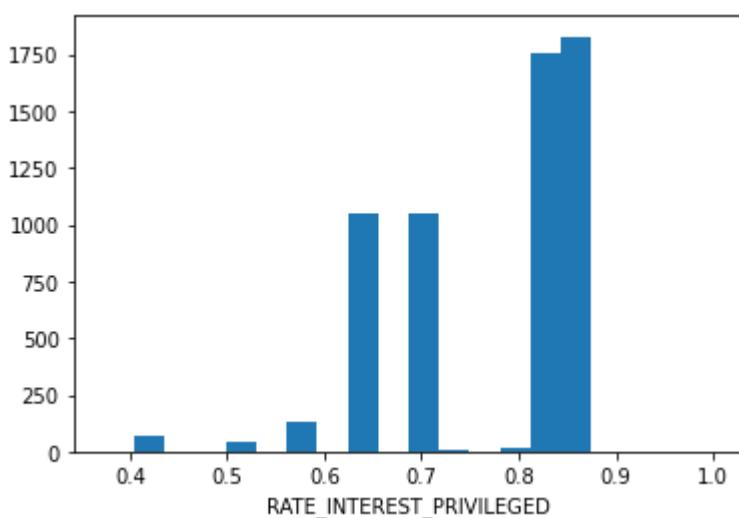
DAYS_FIRST_DRAWING :: root-transformed
-3.60515187706272



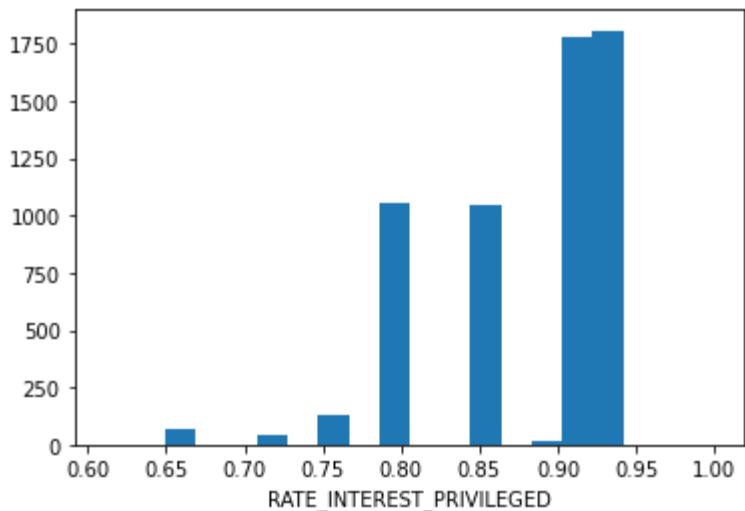
`DAYS_FIRST_DRAWING` :: log-transformed
-3.776679204812683



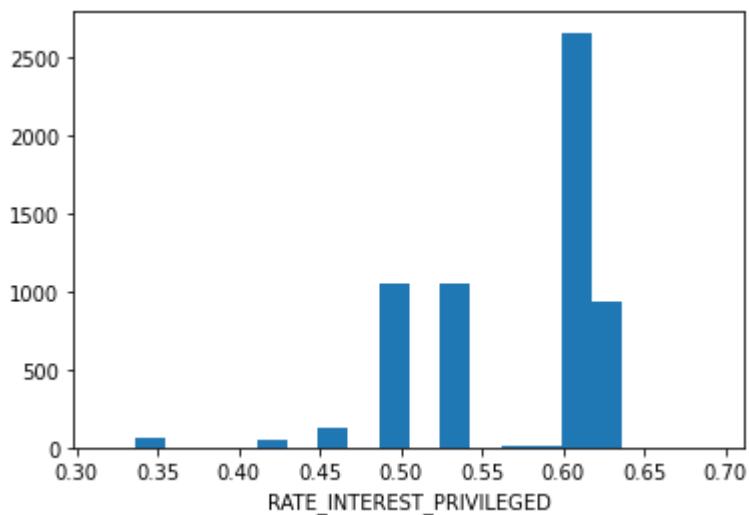
`RATE_INTEREST_PRIVILEGED`
-1.0076799803409757



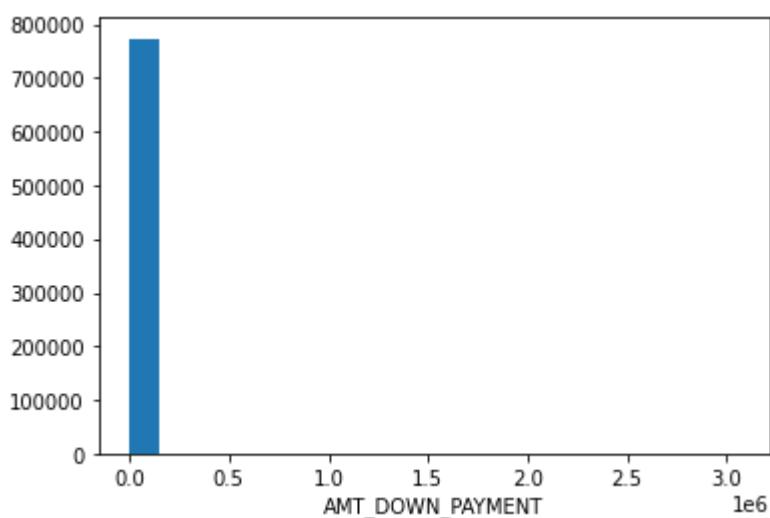
`RATE_INTEREST_PRIVILEGED` :: root-transformed
-1.1704844801940282



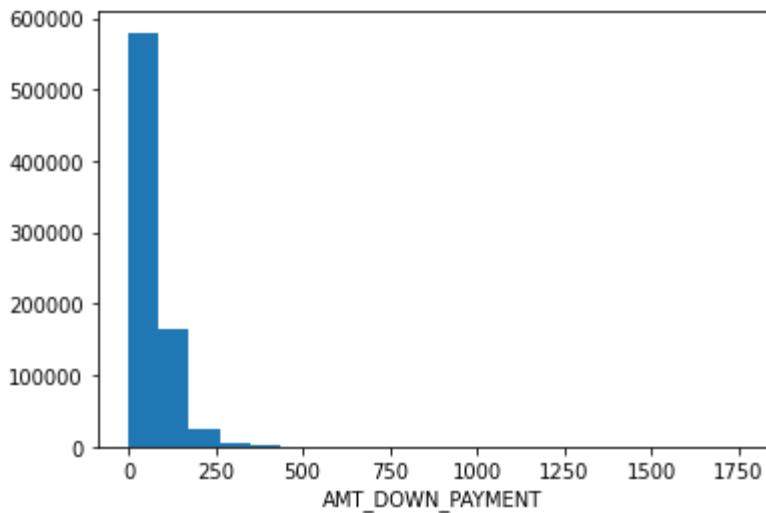
RATE_INTEREST_PRIVILEGED :: log-transformed
-1.1313534826383589



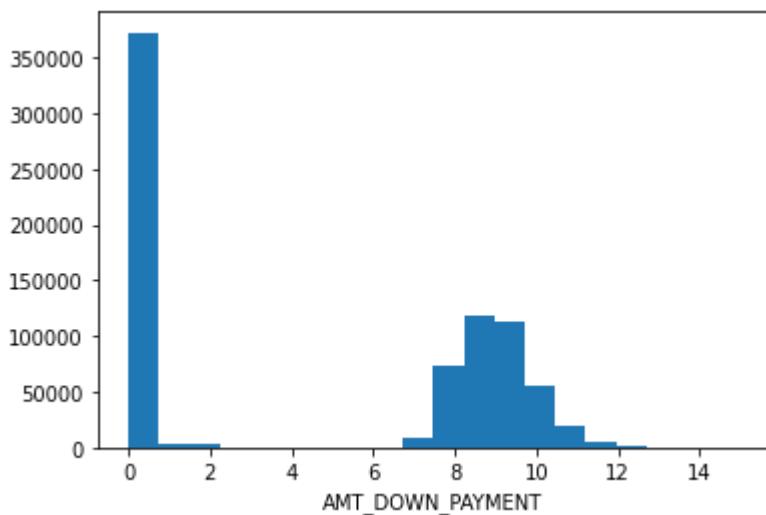
AMT_DOWN_PAYMENT
36.476575814458435



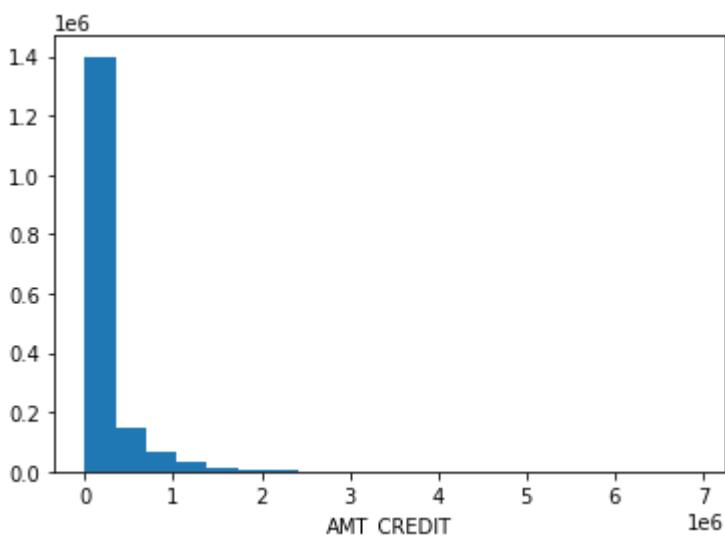
AMT_DOWN_PAYMENT :: root-transformed
2.1425097933880832



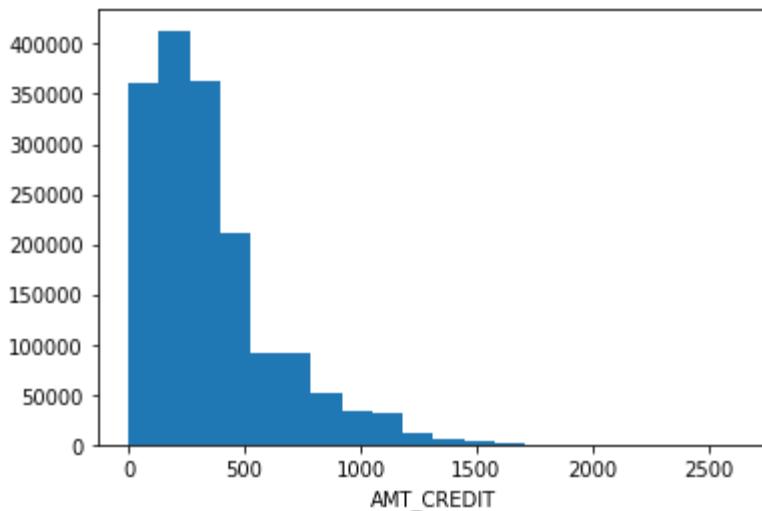
AMT_DOWN_PAYMENT :: log-transformed
0.02528934698923228



AMT_CREDIT
3.2458146495467357

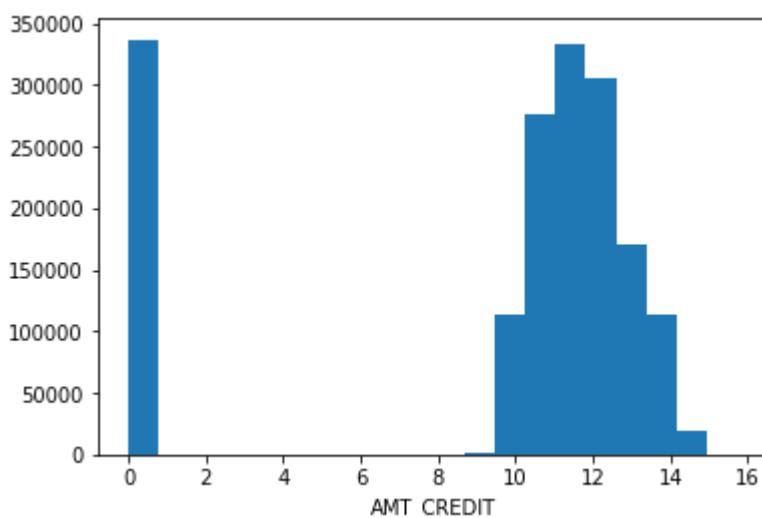


AMT_CREDIT :: root-transformed
1.1884171772637526



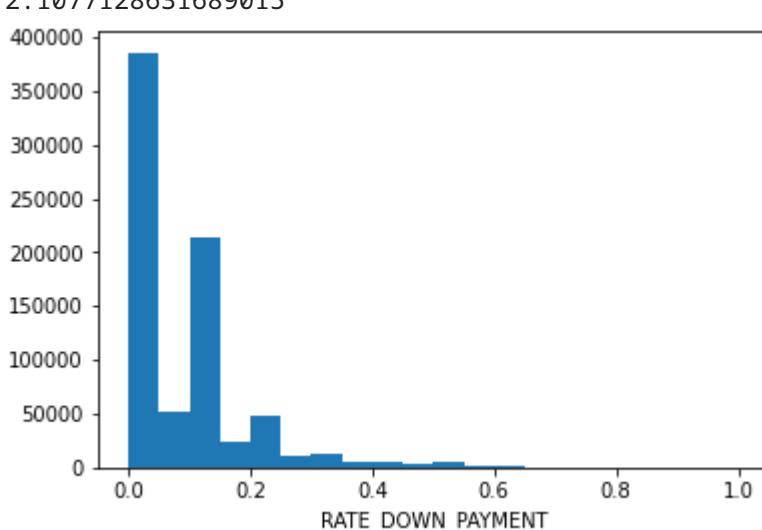
AMT_CREDIT :: log-transformed

-1.3225201292528215



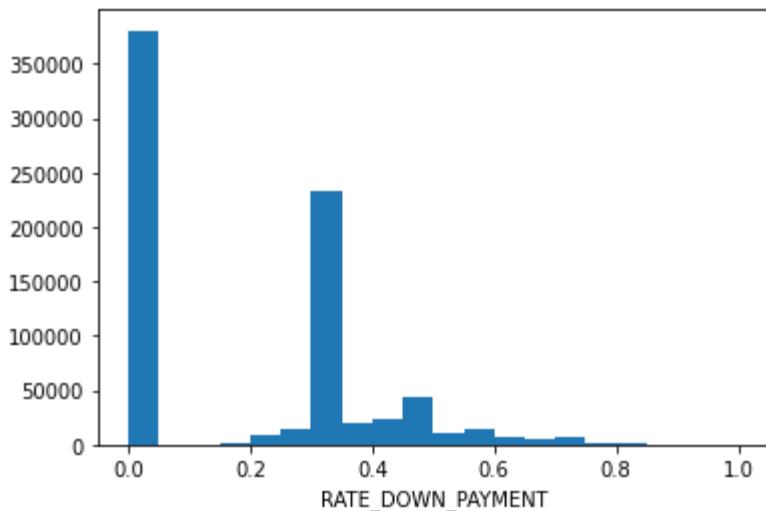
RATE_DOWN_PAYMENT

2.1077128631689015

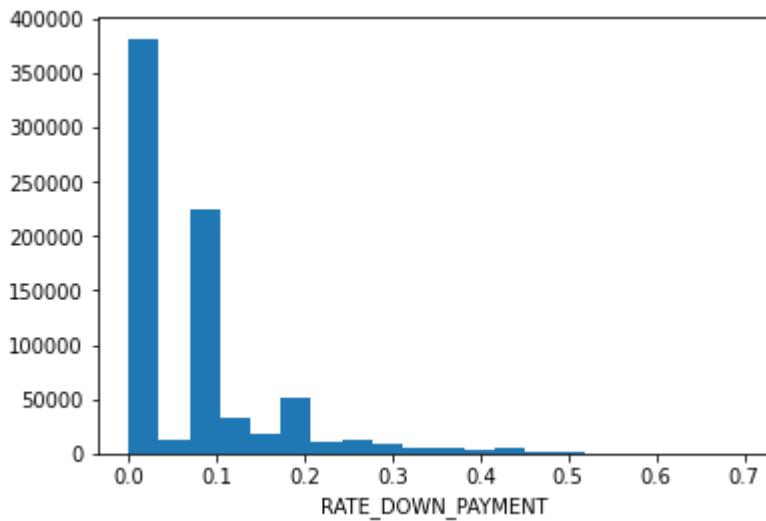


RATE_DOWN_PAYMENT :: root-transformed

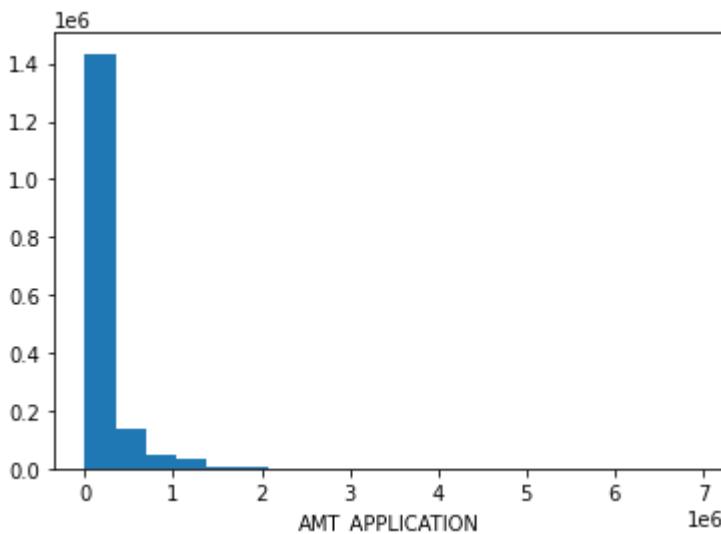
0.4887729447812732



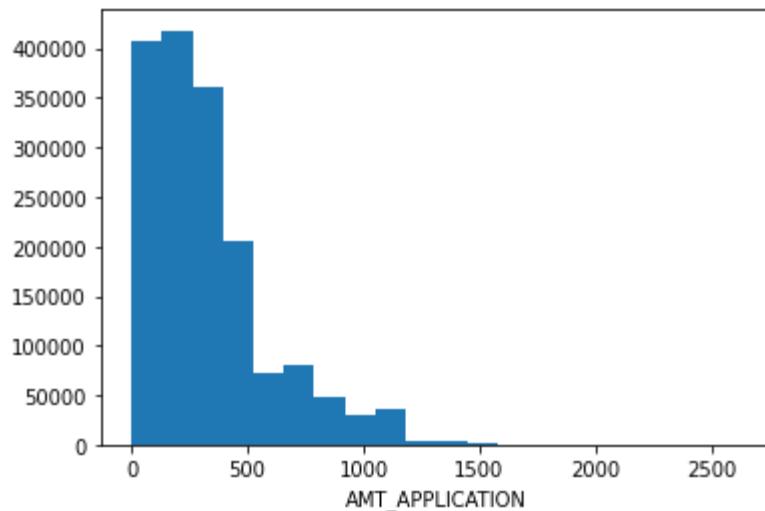
RATE_DOWN_PAYMENT :: log-transformed
1.6723763443292825



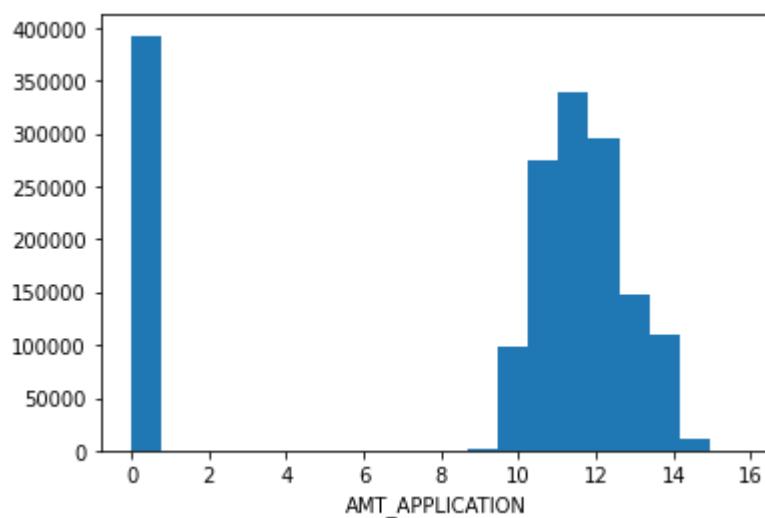
AMT_APPLICATION
3.391442176069141



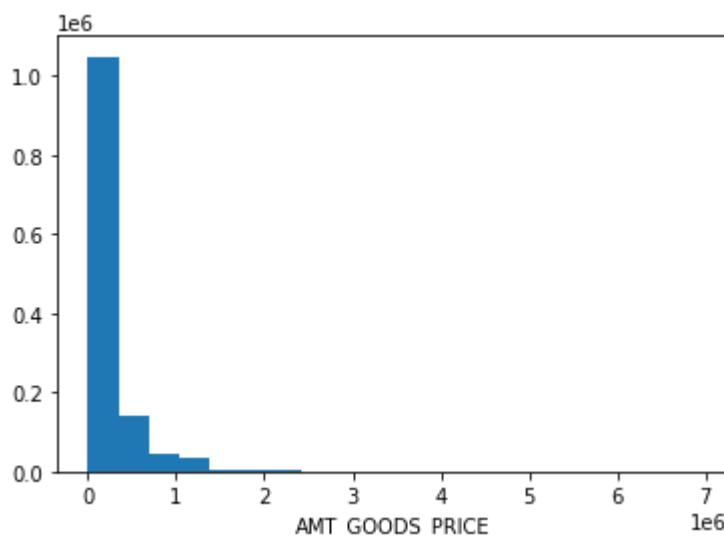
AMT_APPLICATION :: root-transformed
1.2059847917921396



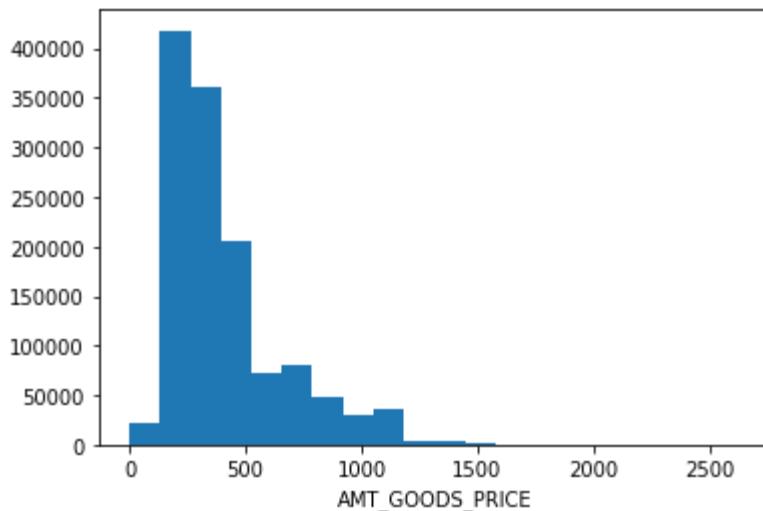
AMT_APPLICATION :: log-transformed
-1.1234245370618026



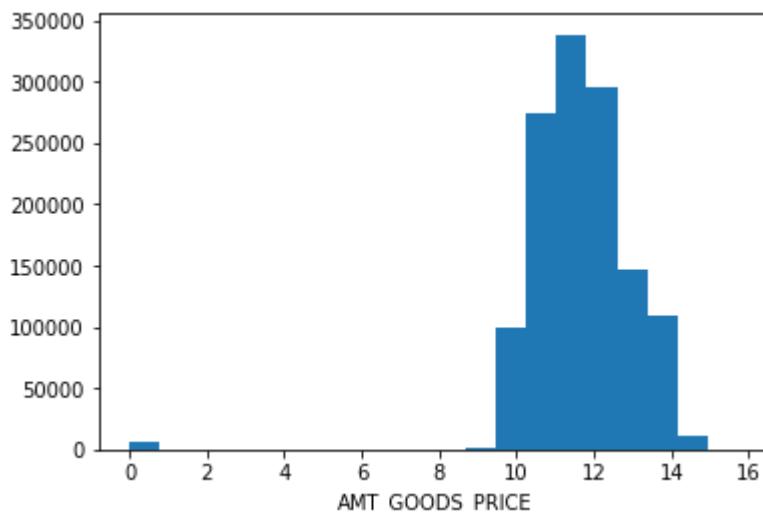
AMT_GOODS_PRICE
3.0736896781923937



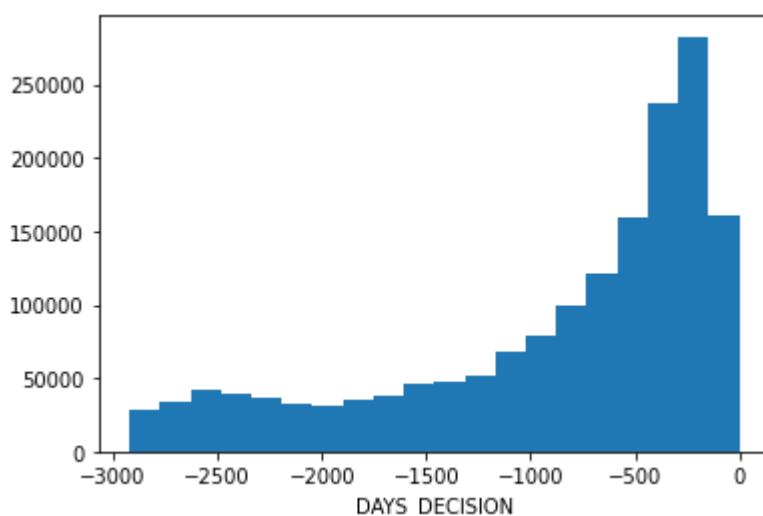
AMT_GOODS_PRICE :: root-transformed
1.5531368918166617



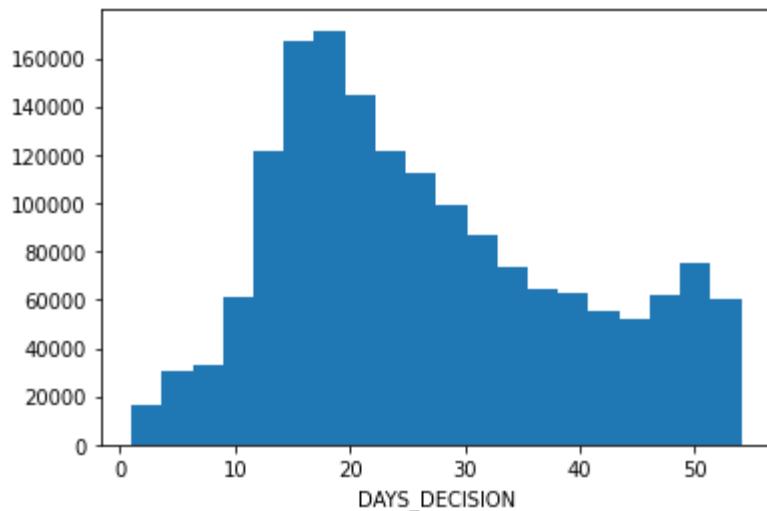
AMT_GOODS_PRICE :: log-transformed
-2.9346053356830573



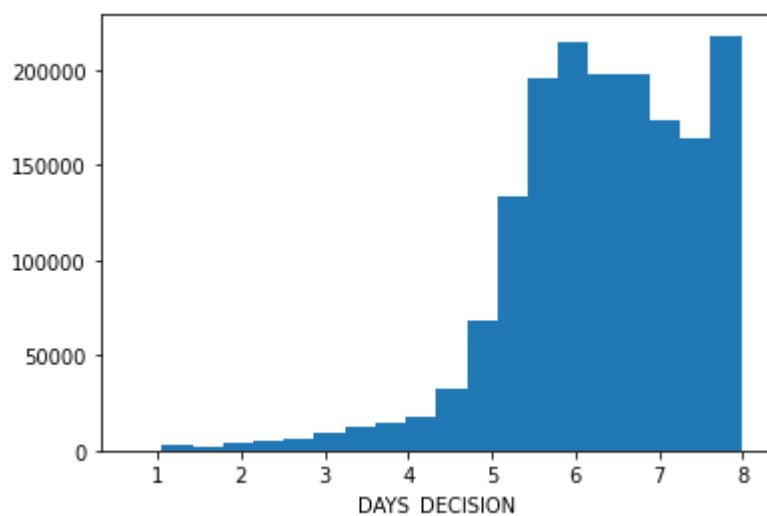
DAYS_DECISION
1.0530796747910551



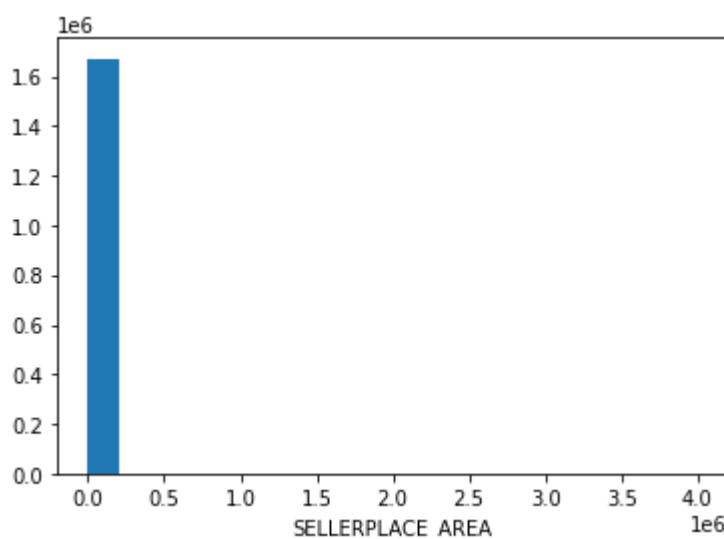
DAYS_DECISION :: root-transformed
0.4430118819867479



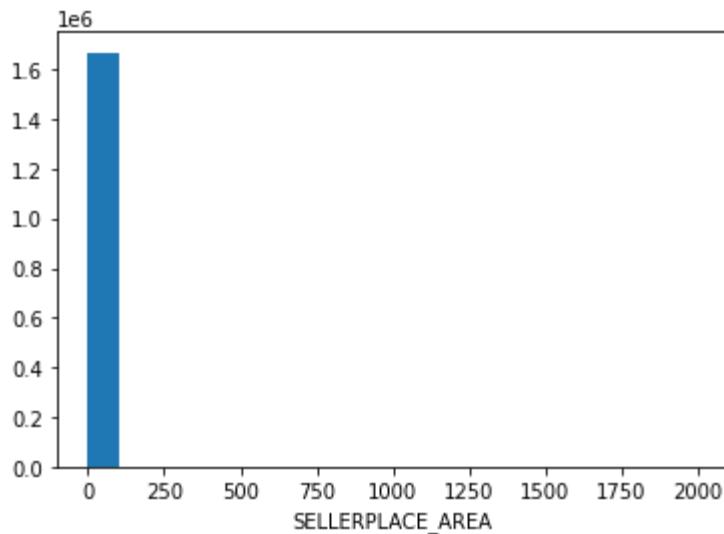
DAYS_DECISION :: log-transformed
-0.8954142045526644



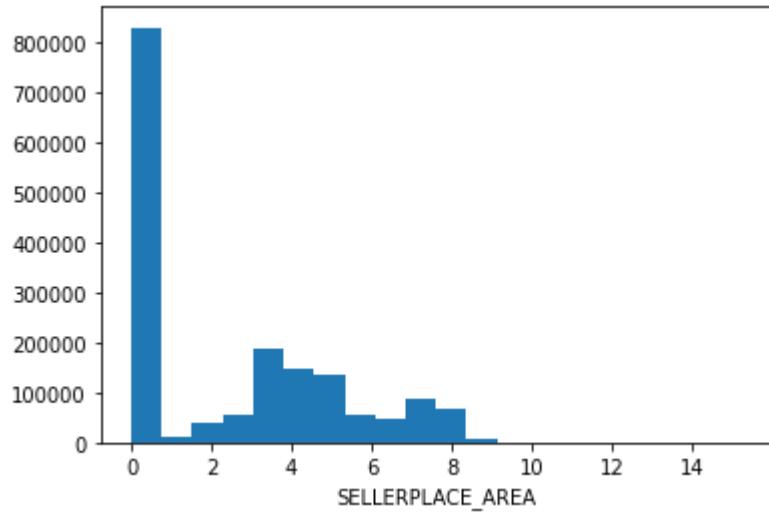
SELLERPLACE_AREA
529.628875957608



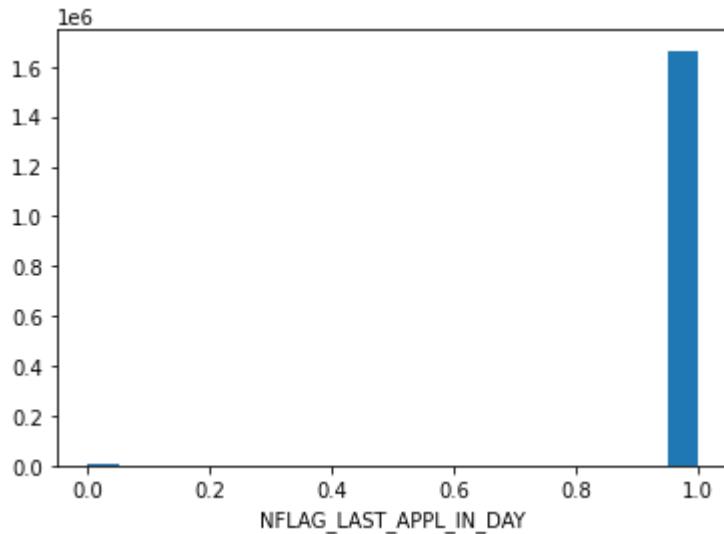
SELLERPLACE_AREA :: root-transformed
10.439917585336213



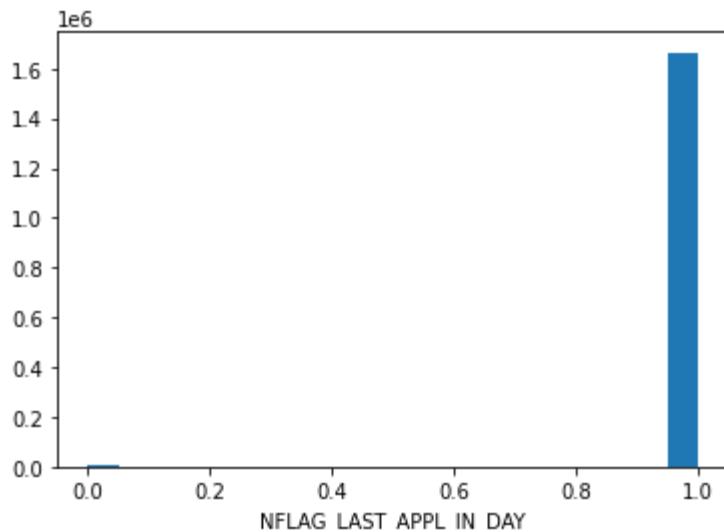
SELLERPLACE_AREA :: log-transformed
0.7615342627121997



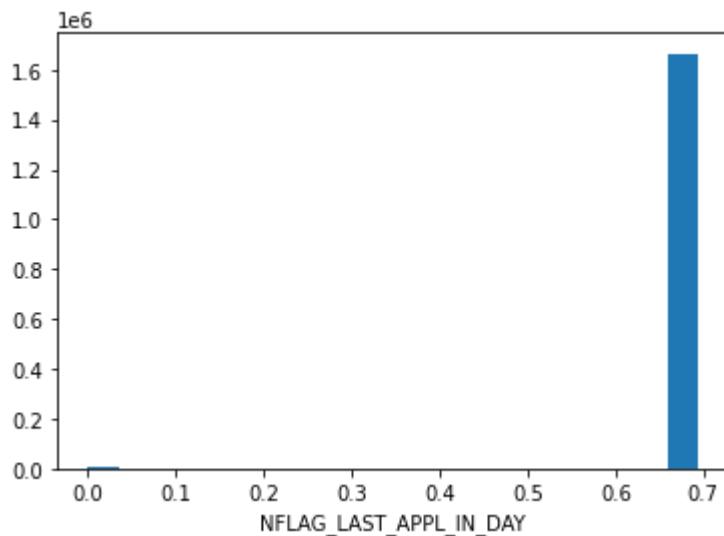
NFLAG_LAST_APPL_IN_DAY
-16.735924436415573



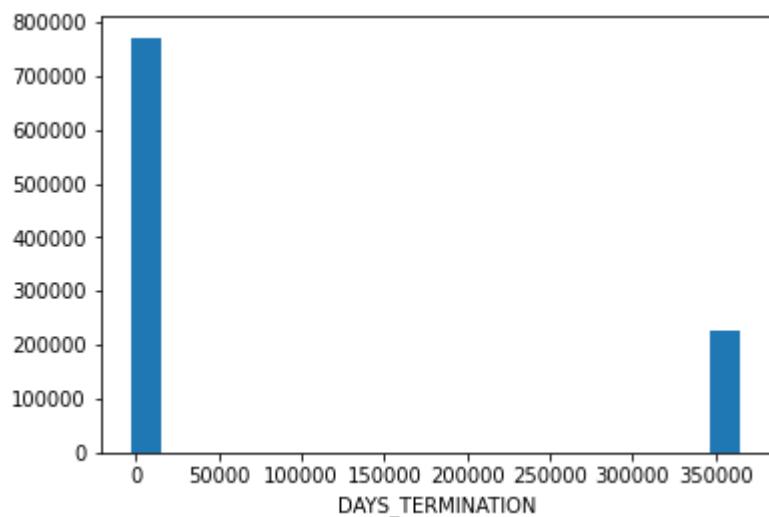
NFLAG_LAST_APPL_IN_DAY :: root-transformed
-16.735924436415573



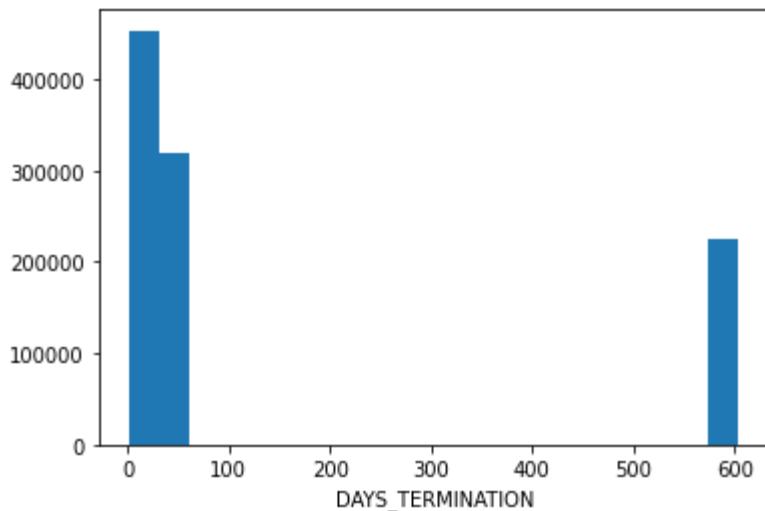
`NFLAG_LAST_APPL_IN_DAY` :: log-transformed
-16.735924436415598



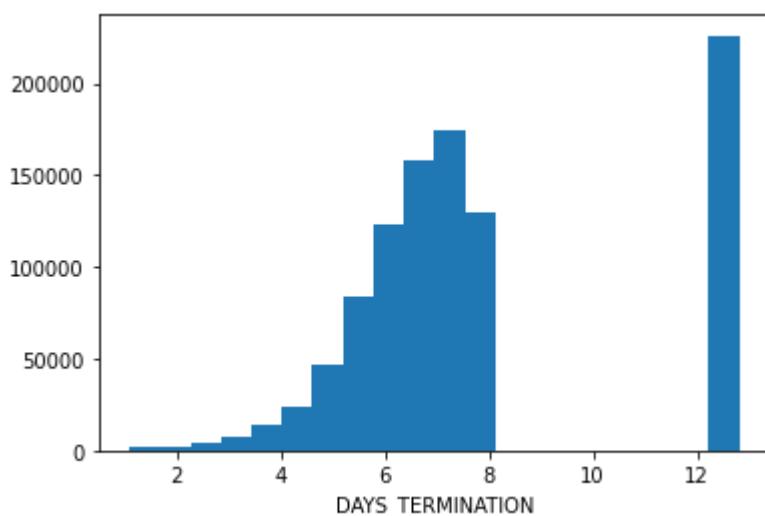
`DAY_S_TERMINATION`
1.3063753941692364



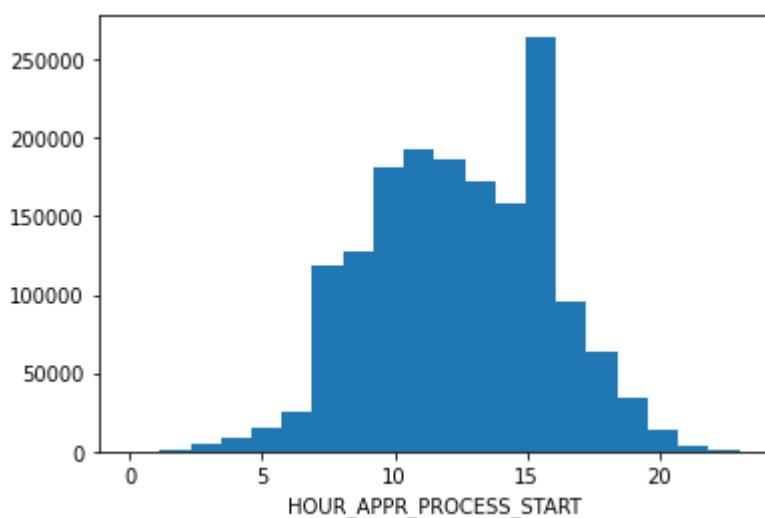
`DAY_S_TERMINATION` :: root-transformed
1.2987423017412347



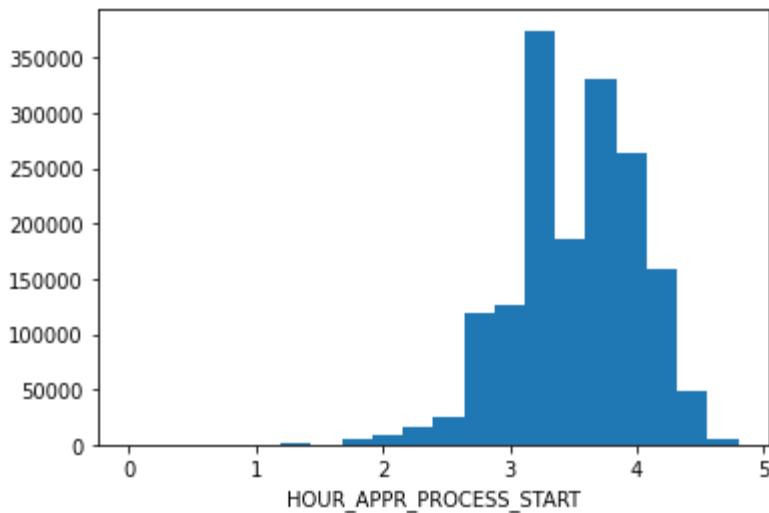
DAYS_TERMINATION :: log-transformed
0.8299535521677465



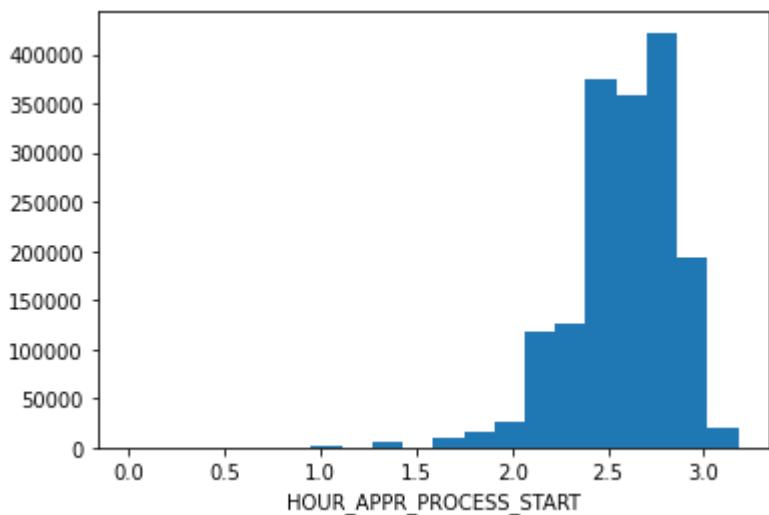
HOUR_APPR_PROCESS_START
-0.02562924914988886



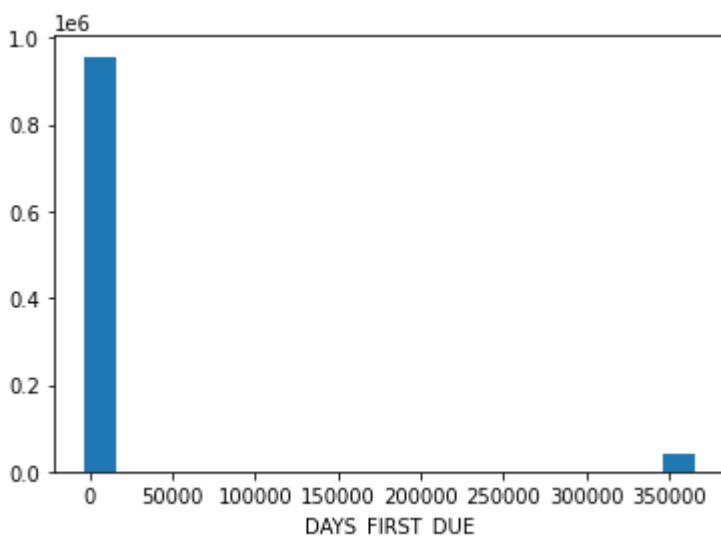
HOUR_APPR_PROCESS_START :: root-transformed
-0.4871718845515148



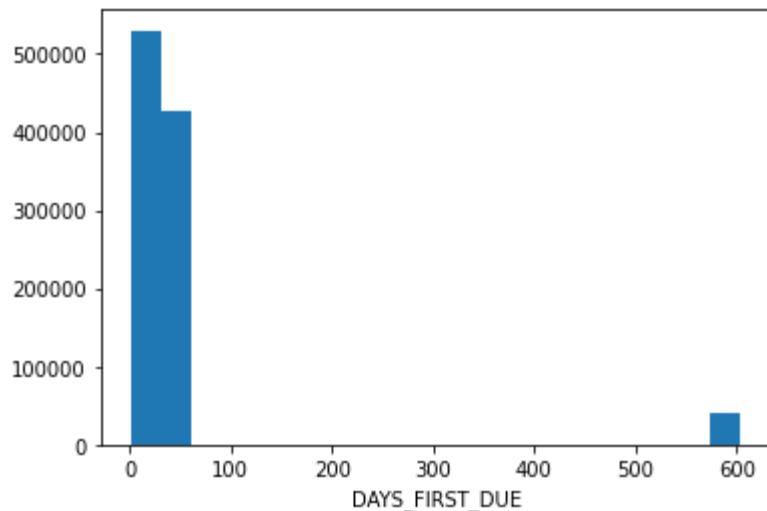
`HOUR_APPR_PROCESS_START` :: log-transformed
-0.9905313206137432



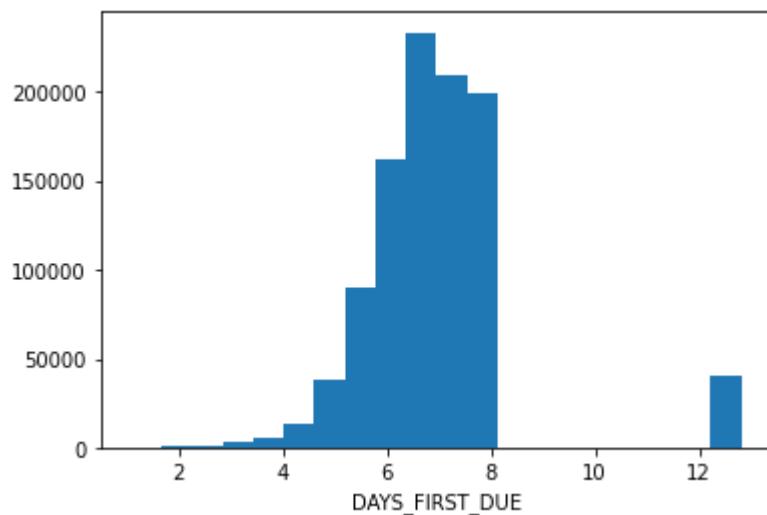
`DAYS_FIRST_DUE`
4.644087013439658



`DAYS_FIRST_DUE` :: root-transformed
4.561420954631297



DAYS_FIRST_DUE :: log-transformed
2.002201100706739



PREVAPP Feature Engineering

Addition and Transformation

Please refer to section 6.2.1 and 6.2.5 for details on feature renaming and transformation.

In [20]:

```
def prevapp_eda(df):
    prevapp = df
    drop_list_pa = []
    #drop_list_pa.append('PA_O_PREV_REJ_CNT')
    #Day and Amount columns
    day_cols = [col for col in prevapp.columns if 'DAY' in col]

    amt_cols = [col for col in prevapp.columns if 'AMT' in col]

    #Adding new features
    prevapp['PA_N_APRTN_CNT'] = prevapp['PA_O_NAME_CONTRACT_STATUS'].map(lambda x: 1 if x == 'APPROVED' else 0)
    prevapp['PA_N_REJ_CNT'] = prevapp['PA_O_NAME_CONTRACT_STATUS'].map(lambda x: 1 if x == 'REFUSED' else 0)
    prevapp['PA_N_APCTN_CRDT_DIFF'] = prevapp['PA_O_AMT_APPLICATION'] - prevapp['PA_O_AMT_CREDIT']
    prevapp['PA_N_APCTN_CRDT_RATIO'] = prevapp['PA_O_AMT_APPLICATION'] / prevapp['PA_O_AMT_CREDIT']
    prevapp['PA_N_CRDT_ANNUITY_RATIO'] = prevapp['PA_O_AMT_CREDIT'] / prevapp['PA_O_AMT_ANNUITY']
    prevapp['PA_N_DWN_PYMNT_CRDT_RATIO'] = prevapp['PA_O_AMT_DOWN_PAYMENT'] / prevapp['PA_O_AMT_CREDIT']
    prevapp["PA_O_CASH_LOAN_PURPOSE"] = np.where(~prevapp["PA_O_NAME_CASH_LOAN_PURPOSE"].isna(), 1, 0)

    #root
    prevapp['PA_N_R_RATE_DOWN_PAYMENT'] = np.sqrt(prevapp['PA_O_RATE_DOWN_PAYMENT'])
    prevapp['PA_N_R_AMT_CREDIT'] = np.sqrt(prevapp['PA_O_AMT_CREDIT'].abs())
    prevapp['PA_N_R_AMT_ANNUITY'] = np.sqrt(prevapp['PA_O_AMT_ANNUITY'].abs())
    prevapp['PA_N_R_CNT_PAYMENT'] = np.sqrt(prevapp['PA_O_CNT_PAYMENT'].abs())
    prevapp['PA_N_R_AMT_GOODS_PRICE'] = np.sqrt(prevapp['PA_O_AMT_GOODS_PRICE'])
    prevapp['PA_N_R_DWN_PYMNT_CRDT_RATIO'] = np.sqrt(prevapp['PA_N_DWN_PYMNT_CRDT_RATIO'])
    prevapp['PA_N_R_DAYS_DECISION'] = np.sqrt(prevapp['PA_O_DAYS_DECISION'])

    #log
    prevapp['PA_N_L_CRDT_ANNUITY_RATIO'] = np.log(prevapp['PA_N_CRDT_ANNUITY'])
    prevapp['PA_N_L_DAYS_TERMINATION'] = np.log(prevapp['PA_O_DAYS_TERMINATION'])
    prevapp['PA_N_L_AMT_DOWN_PAYMENT'] = np.log(prevapp['PA_O_AMT_DOWN_PAYMENT'])
    prevapp['PA_N_L_SELLERPLACE_AREA'] = np.log(prevapp['PA_O_SELLERPLACE_AREA'])
    prevapp['PA_N_L_AMT_APPLICATION'] = np.log(prevapp['PA_O_AMT_APPLICATION'])
    prevapp['PA_N_L_PREV_APCTN_CRDT_DIFF'] = np.log(prevapp['PA_N_APCTN_CRDT_DIFF'])

    for c in [co for co in prevapp.columns if 'DAYS' in co]:
        prevapp[c] = prevapp[c].replace({365243.0: np.nan})
        prevapp[c] = prevapp[c].abs()

    drop_list_pa.append('PA_O_WEEKDAY_APPR_PROCESS_START') ## weekday data
    drop_list_pa.append('PA_O_HOUR_APPR_PROCESS_START') ## Hour application

    # Drop elements in the drop list
    drop_list_pa.append('PA_O_WEEKDAY_APPR_PROCESS_START') ## weekday data
    drop_list_pa.append('PA_O_HOUR_APPR_PROCESS_START') ## Hour application

    threshold = 0.7
    drop_list_pa = list(prevapp.columns[prevapp.isnull().mean() > threshold])

    prevapp = prevapp.drop(columns=drop_list_pa, axis=1)

    #Dropping columns with missing value rate higher than threshold
    prevapp = prevapp[prevapp.columns[prevapp.isnull().mean() < threshold]]

    #Dropping rows with missing value rate higher than threshold
```

```
prevapp = prevapp.loc[prevapp.isnull().mean(axis=1) < threshold]

prevapp= eda_transformation(prevapp,3)
return prevapp
```

```
In [21]: prevapp = datasets['previous_application']
prevapp.columns = ['PA_0_' + col for col in prevapp.columns]
prevapp['SK_ID_CURR'] = prevapp['PA_0_SK_ID_CURR']
prevapp['SK_ID_PREV'] = prevapp['PA_0_SK_ID_PREV']
prevapp.drop(['PA_0_SK_ID_CURR'],axis=1, inplace=True)
prevapp.drop(['PA_0_SK_ID_PREV'],axis=1,inplace=True)
```

```
In [22]: prevapp = prevapp_eda(prevapp)
```

```

-----
# of ID's: 1
ID's:
['SK_ID_CURR']

-----
# All features: 57
All features:
['PA_N_L_SELLERPLACE_AREA', 'PA_O_DAYS_LAST_DUE', 'PA_N_DWN_PYMNT_CRDT_RATIO',
 'PA_O_AMT_ANNUITY', 'PA_O_NAME_GOODS_CATEGORY', 'PA_O_PREV_CRDT_ANNUTY_RA-
 TIO', 'PA_N_L_PREV_APCTN_CRDT_DIFF', 'PA_O_CODE_REJECT_REASON', 'PA_O_HOUR_A-
 PPR_PROCESS_START', 'PA_O_NAME_SELLER_INDUSTRY', 'PA_O_AMT_APPLICATION', 'PA-
 O_NFLAG_LAST_APPL_IN_DAY', 'PA_O_NAME_CLIENT_TYPE', 'PA_O_CNT_PAYMENT', 'PA-
 O_NAME_PAYMENT_TYPE', 'PA_O_NAME_YIELD_GROUP', 'PA_O_DAYS_DECISION', 'PA_N-
 R_DAYS_DECISION', 'PA_O_CHANNEL_TYPE', 'PA_O_DAYS_LAST_DUE_1ST_VERSION', 'PA-
 O_PREV_DWN_PYMNT_CRDT_RATIO', 'PA_O_NAME_CASH_LOAN_PURPOSE', 'PA_O_NAME_TYP-
 E_SUITE', 'PA_N_APRV_CNT', 'PA_N_APCTN_CRDT_RATIO', 'PA_N_R_AMT_ANNUITY', 'PA-
 O_SELLERPLACE_AREA', 'PA_O_DAYS_FIRST_DUE', 'PA_O_DAYS_TERMINATION', 'PA_N-
 R_AMT_CREDIT', 'PA_N_R_AMT_GOODS_PRICE', 'PA_N_R_DWN_PYMNT_CRDT_RATIO', 'PA-
 O_AMT_DOWN_PAYMENT', 'PA_O_FLAG_LAST_APPL_PER_CONTRACT', 'PA_O_CASH_LOAN_PU-
 RPOSE', 'PA_O_PREV_APCTN_CRDT_RATIO', 'PA_N_APCTN_CRDT_DIFF', 'PA_N_L_AMT_AP-
 PLICATION', 'PA_O_RATE_DOWN_PAYMENT', 'SK_ID_PREV', 'PA_N_R_CNT_PAYMENT', 'PA-
 O_AMT_CREDIT', 'PA_O_NFLAG_INSURED_ON_APPROVAL', 'PA_N_REJ_CNT', 'PA_N_CRD-
 T_ANNUTY_RATIO', 'PA_N_L_CRDT_ANNUTY_RATIO', 'PA_N_L_AMT_DOWN_PAYMENT', 'PA-
 O_NAME_CONTRACT_TYPE', 'PA_O_PREV_APCTN_CRDT_DIFF', 'PA_O_WEEKDAY_APPR_PROCE-
 SS_START', 'PA_O_NAME_CONTRACT_STATUS', 'PA_O_NAME_PRODUCT_TYPE', 'PA_N_R_RA-
 TE_DOWN_PAYMENT', 'PA_O_NAME_PORTFOLIO', 'PA_O_AMT_GOODS_PRICE', 'PA_O_PROD-
 UCT_COMBINATION', 'PA_N_L_DAYS_TERMINATION']

```

Missing data:

	Total	Percent
PA_O_DAYS_TERMINATION	898978	53.824121
PA_N_DWN_PYMNT_CRDT_RATIO	895844	53.636480
PA_N_R_RATE_DOWN_PAYMENT	895844	53.636480
PA_N_L_AMT_DOWN_PAYMENT	895844	53.636480
PA_O_RATE_DOWN_PAYMENT	895844	53.636480
PA_O_AMT_DOWN_PAYMENT	895844	53.636480
PA_N_R_DWN_PYMNT_CRDT_RATIO	895844	53.636480
PA_O_PREV_DWN_PYMNT_CRDT_RATIO	895844	53.636480
PA_O_DAYS_LAST_DUE	884286	52.944473
PA_O_NAME_TYPE_SUITE	820405	49.119754
PA_O_DAYS_LAST_DUE_1ST_VERSION	766929	45.918008
PA_O_DAYS_FIRST_DUE	713710	42.731650
PA_O_NFLAG_INSURED_ON_APPROVAL	673065	40.298129
PA_N_L_DAYS_TERMINATION	673065	40.298129
PA_O_AMT_GOODS_PRICE	385515	23.081773
PA_N_R_AMT_GOODS_PRICE	385515	23.081773
PA_O_PREV_CRDT_ANNUTY_RATIO	373859	22.383898
PA_N_CRD_T_ANNUTY_RATIO	373859	22.383898
PA_N_L_CRDT_ANNUTY_RATIO	373859	22.383898
PA_O_AMT_ANNUITY	372235	22.286665
PA_N_R_AMT_ANNUITY	372235	22.286665
PA_N_R_CNT_PAYMENT	372230	22.286366
PA_O_CNT_PAYMENT	372230	22.286366
PA_O_PREV_APCTN_CRDT_RATIO	336768	20.163165
PA_N_APCTN_CRDT_RATIO	336768	20.163165
PA_O_PRODUCT_COMBINATION	346	0.020716
PA_N_R_AMT_CREDIT	1	0.000060
PA_N_APCTN_CRDT_DIFF	1	0.000060

PA_O_AMT_CREDIT	1	0.000060
PA_O_PREV_APCTN_CRDT_DIFF	1	0.000060
PA_N_L_PREV_APCTN_CRDT_DIFF	1	0.000060
PA_O_NAME_GOODS_CATEGORY	0	0.000000
PA_O_NAME_CONTRACT_TYPE	0	0.000000
SK_ID_PREV	0	0.000000
PA_N_REJ_CNT	0	0.000000
PA_O_WEEKDAY_APPR_PROCESS_START	0	0.000000
PA_O_NAME_CONTRACT_STATUS	0	0.000000
PA_O_NAME_PRODUCT_TYPE	0	0.000000
PA_O_NAME_PORTFOLIO	0	0.000000
PA_O_NAME_PAYMENT_TYPE	0	0.000000
PA_N_L_AMT_APPLICATION	0	0.000000
PA_N_APRA_CNT	0	0.000000
PA_O_DAYS_DECISION	0	0.000000
PA_N_R_DAYS_DECISION	0	0.000000
PA_O_CHANNEL_TYPE	0	0.000000
PA_O_NAME_CLIENT_TYPE	0	0.000000
PA_O_NAME_CASH_LOAN_PURPOSE	0	0.000000
PA_O_NFLAG_LAST_APPL_IN_DAY	0	0.000000
PA_O_SELLERPLACE_AREA	0	0.000000
PA_O_NAME_YIELD_GROUP	0	0.000000
PA_O_AMT_APPLICATION	0	0.000000
PA_O_NAME_SELLER_INDUSTRY	0	0.000000
PA_O_HOUR_APPR_PROCESS_START	0	0.000000
PA_O_CODE_REJECT_REASON	0	0.000000
PA_O_FLAG_LAST_APPL_PER_CONTRACT	0	0.000000
PA_O_CASH_LOAN_PURPOSE	0	0.000000
PA_N_L_SELLERPLACE_AREA	0	0.000000

of Numerical features: 41

Numerical features:

```
['PA_O_AMT_ANNUITY', 'PA_O_AMT_APPLICATION', 'PA_O_AMT_CREDIT', 'PA_O_AMT_DOWN_PAYMENT', 'PA_O_AMT_GOODS_PRICE', 'PA_O_HOUR_APPR_PROCESS_START', 'PA_O_NFLAG_LAST_APPL_IN_DAY', 'PA_O_RATE_DOWN_PAYMENT', 'PA_O_DAYS_DECISION', 'PA_O_SELLERPLACE_AREA', 'PA_O_CNT_PAYMENT', 'PA_O_DAYS_FIRST_DUE', 'PA_O_DAYS_LAST_DUE_1ST_VERSION', 'PA_O_DAYS_LAST_DUE', 'PA_O_DAYS_TERMINATION', 'PA_O_NFLAG_INSURED_ON_APPROVAL', 'PA_O_PREV_APCTN_CRDT_DIFF', 'PA_O_PREV_APCTN_CRDT_RATIO', 'PA_O_PREV_CRDT_ANNUITY_RATIO', 'PA_O_PREV_DWN_PYMNT_CRDT_RATIO', 'SK_ID_CURR', 'SK_ID_PREV', 'PA_N_APRA_CNT', 'PA_N_REJ_CNT', 'PA_N_APCTN_CRDT_DIFF', 'PA_N_APCTN_CRDT_RATIO', 'PA_N_CRDT_ANNUITY_RATIO', 'PA_N_DWN_PYMNT_CRDT_RATIO', 'PA_N_R_RATE_DOWN_PAYMENT', 'PA_N_R_AMT_CREDIT', 'PA_N_R_AMT_ANNUITY', 'PA_N_R_CNT_PAYMENT', 'PA_N_R_AMT_GOODS_PRICE', 'PA_N_R_DWN_PYMNT_CRDT_RATIO', 'PA_N_R_DAYS_DECISION', 'PA_N_L_CRDT_ANNUITY_RATIO', 'PA_N_L_DAYS_TERMINATION', 'PA_N_L_AMT_DOWN_PAYMENT', 'PA_N_L_SELLERPLACE_AREA', 'PA_N_L_AMT_APPLICATION', 'PA_N_L_PREV_APCTN_CRDT_DIFF']
```

Numerical Statistical Summary:

	PA_O_AMT_ANNUITY	PA_O_AMT_APPLICATION	PA_O_AMT_CREDIT	\
count	1.297979e+06	1.670214e+06	1.670213e+06	
mean	1.595512e+04	1.752339e+05	1.961140e+05	
std	1.478214e+04	2.927798e+05	3.185746e+05	
min	0.000000e+00	0.000000e+00	0.000000e+00	
25%	6.321780e+03	1.872000e+04	2.416050e+04	
50%	1.125000e+04	7.104600e+04	8.054100e+04	
75%	2.065842e+04	1.803600e+05	2.164185e+05	
max	4.180581e+05	6.905160e+06	6.905160e+06	

	PA_O_AMT_DOWN_PAYMENT	PA_O_AMT_GOODS_PRICE	\
count	7.743700e+05	1.284699e+06	
mean	6.697402e+03	2.278473e+05	
std	2.092150e+04	3.153966e+05	
min	-9.000000e-01	0.000000e+00	
25%	0.000000e+00	5.084100e+04	
50%	1.638000e+03	1.123200e+05	
75%	7.740000e+03	2.340000e+05	
max	3.060045e+06	6.905160e+06	
	PA_O_HOUR_APPR_PROCESS_START	PA_O_NFLAG_LAST_APPL_IN_DAY	\
count	1.670214e+06	1.670214e+06	
mean	1.248418e+01	9.964675e-01	
std	3.334028e+00	5.932963e-02	
min	0.000000e+00	0.000000e+00	
25%	1.000000e+01	1.000000e+00	
50%	1.200000e+01	1.000000e+00	
75%	1.500000e+01	1.000000e+00	
max	2.300000e+01	1.000000e+00	
	PA_O_RATE_DOWN_PAYMENT	PA_O_DAYS_DECISION	PA_O_SELLERPLACE_AREA
...			\
count	774370.000000	1.670214e+06	1.670214e+06
...			
mean	0.079637	8.806797e+02	3.139511e+02
...			
std	0.107823	7.790997e+02	7.127443e+03
...			
min	-0.000015	1.000000e+00	-1.000000e+00
...			
25%	0.000000	2.800000e+02	-1.000000e+00
...			
50%	0.051605	5.810000e+02	3.000000e+00
...			
75%	0.108909	1.300000e+03	8.200000e+01
...			
max	1.000000	2.922000e+03	4.000000e+06
...			
	PA_N_R_CNT_PAYMENT	PA_N_R_AMT_GOODS_PRICE	\
count	1.297984e+06	1.284699e+06	
mean	3.543017e+00	4.059744e+02	
std	1.871126e+00	2.510619e+02	
min	0.000000e+00	0.000000e+00	
25%	2.449490e+00	2.254795e+02	
50%	3.464102e+00	3.351418e+02	
75%	4.898979e+00	4.837355e+02	
max	9.165151e+00	2.627767e+03	
	PA_N_R_DWN_PYMNT_CREDT_RATIO	PA_N_R_DAYS_DECISION	\
count	774370.000000	1.670214e+06	
mean	0.205906	2.674869e+01	
std	0.233113	1.285253e+01	
min	0.000000	1.000000e+00	
25%	0.000000	1.673320e+01	
50%	0.223026	2.410394e+01	
75%	0.333333	3.605551e+01	
max	3.350297	5.405553e+01	

	PA_N_L_CRDT_ANNUITY_RATIO	PA_N_L_DAYS_TERMINATION \
count	1.296355e+06	997149.000000
mean	inf	7.886538
std	NaN	2.843802
min	7.472144e-01	1.098612
25%	2.221847e+00	6.104793
50%	2.405454e+00	7.066467
75%	3.044522e+00	7.824846
max	inf	12.808321
	PA_N_L_AMT_DOWN_PAYMENT	PA_N_L_SELLERPLACE_AREA \
count	774370.000000	1.670214e+06
mean	4.588255	2.719648e+00
std	4.524383	2.425873e+00
min	0.000000	0.000000e+00
25%	0.000000	6.931472e-01
50%	7.401842	1.386294e+00
75%	8.954286	4.418841e+00
max	14.933941	1.520181e+01
	PA_N_L_AMT_APPLICATION	PA_N_L_PREV_APCTN_CRDT_DIFF
count	1.670214e+06	1.670213e+06
mean	8.959559e+00	5.726476e+00
std	5.055680e+00	4.792151e+00
min	0.000000e+00	0.000000e+00
25%	9.837401e+00	0.000000e+00
50%	1.117110e+01	7.903042e+00
75%	1.210272e+01	9.792668e+00
max	1.574778e+01	1.492092e+01

[8 rows x 41 columns]

of Categorical features: 17
Categorical features:
['PA_O_NAME_CONTRACT_TYPE', 'PA_O_WEEKDAY_APPR_PROCESS_START', 'PA_O_FLAG_LAST_APPL_PER_CONTRACT', 'PA_O_NAME_CASH_LOAN_PURPOSE', 'PA_O_NAME_CONTRACT_STATUS', 'PA_O_NAME_PAYMENT_TYPE', 'PA_O_CODE_REJECT_REASON', 'PA_O_NAME_TYPE_SUITE', 'PA_O_NAME_CLIENT_TYPE', 'PA_O_NAME_GOODS_CATEGORY', 'PA_O_NAME_PORTFOLIO', 'PA_O_NAME_PRODUCT_TYPE', 'PA_O_CHANNEL_TYPE', 'PA_O_NAME_SELLER_INDUSTRY', 'PA_O_NAME_YIELD_GROUP', 'PA_O_PRODUCT_COMBINATION', 'PA_O_CASH_LOAN_PURPOSE']

Categorical Statistical Summary:

Categories:

PA_O_NAME_CONTRACT_TYPE	[Consumer loans, Cash loans, Revolving loans, ...]
PA_O_WEEKDAY_APPR_PROCESS_START	[SATURDAY, THURSDAY, TUESDAY, MONDAY, FRIDAY, ...]
PA_O_FLAG_LAST_APPL_PER_CONTRACT	[Y, N]
PA_O_NAME_CASH_LOAN_PURPOSE	[XAP, XNA, Repairs, Everyday expenses, Car rep...]
PA_O_NAME_CONTRACT_STATUS	[Approved, Refused, Canceled, Unusual offer]

```

PA_O_NAME_PAYMENT_TYPE [Cash through the bank, XNA, Non-cash fr
om you...
PA_O_CODE_REJECT_REASON [XAP, HC, LIMIT, CLIENT, SCOFR, SCO, XN
A, VERI...
PA_O_NAME_TYPE_SUITE [nan, Unaccompanied, Spouse, partner, Fa
mily, ...
PA_O_NAME_CLIENT_TYPE [Repeater, New, Refres
hed, XNA]
PA_O_NAME_GOODS_CATEGORY [Mobile, XNA, Consumer Electronics, Cons
tructi...
PA_O_NAME_PORTFOLIO [POS, Cash, XNA, Car
ds, Cars]
PA_O_NAME_PRODUCT_TYPE [XNA, x-sell,
walk-in]
PA_O_CHANNEL_TYPE [Country-wide, Contact center, Credit an
d cash...
PA_O_NAME_SELLER_INDUSTRY [Connectivity, XNA, Consumer electronic
s, Indu...
PA_O_NAME_YIELD_GROUP [middle, low_action, high, low_nor
mal, XNA]
PA_O_PRODUCT_COMBINATION [POS mobile with interest, Cash X-Sell:
low, C...
PA_O_CASH_LOAN_PURPOSE [XAP, XN
A, Other]
dtype: object
-----
```

```

# of OHE categorical features: 146
OHE Categorical features: ['PA_O_NAME_CASH_LOAN_PURPOSE_Hobby', 'PA_O_NAME_C
ONTRACT_TYPE_Cash loans', 'PA_O_CASH_LOAN_PURPOSE_Other', 'PA_O_PRODUCT_COMB
INATION_POS other with interest', 'PA_O_NAME_GOODS_CATEGORY_Clothing and Acc
essories', 'PA_O_NAME_SELLER_INDUSTRY_Tourism', 'PA_O_PRODUCT_COMBINATION_Ca
sh Street: high', 'PA_O_NAME_SELLER_INDUSTRY_Industry', 'PA_O_NAME_CASH_LOAN
_PURPOSE_Wedding / gift / holiday', 'PA_O_NAME_GOODS_CATEGORY_Medicine', 'PA
_O_CHANNEL_TYPE_Country-wide', 'PA_O_NAME_PORTFOLIO_Cars', 'PA_O_NAME_CASH_L
OAN_PURPOSE_XNA', 'PA_O_NAME_GOODS_CATEGORY_Direct Sales', 'PA_O_NAME_CASH_L
OAN_PURPOSE_Other', 'PA_O_PRODUCT_COMBINATION_POS household with interest',
'PA_O_NAME_GOODS_CATEGORY_Medical Supplies', 'PA_O_NAME_CASH_LOAN_PURPOSE_Re
fusal to name the goal', 'PA_O_NAME_PORTFOLIO_Cards', 'PA_O_PRODUCT_COMBINAT
ION_Cash X-Sell: low', 'PA_O_PRODUCT_COMBINATION_POS industry without intere
st', 'PA_O_NAME_YIELD_GROUP_high', 'PA_O_NAME_TYPE_SUITE_Family', 'PA_O_NAME
_GOODS_CATEGORY_Gardening', 'PA_O_NAME_PORTFOLIO_XNA', 'PA_O_NAME_SELLER_IND
USTRY_Jewelry', 'PA_O_NAME_CASH_LOAN_PURPOSE_Buying a holiday home / land',
'PA_O_PRODUCT_COMBINATION_Card Street', 'PA_O_NAME_CONTRACT_STATUS_Unused of
fer', 'PA_O_NAME_GOODS_CATEGORY_Furniture', 'PA_O_NAME_TYPE_SUITE_Unaccompan
ied', 'PA_O_FLAG_LAST_APPL_PER_CONTRACT_Y', 'PA_O_NAME_PAYMENT_TYPE_Cashless
from the account of the employer', 'PA_O_NAME_SELLER_INDUSTRY_Clothing', 'PA
_O_NAME_TYPE_SUITE_Group of people', 'PA_O_WEEKDAY_APPR_PROCESS_START_TUESDA
Y', 'PA_O_WEEKDAY_APPR_PROCESS_START_WEDNESDAY', 'PA_O_NAME_GOODS_CATEGORY_A
udio/Video', 'PA_O_CHANNEL_TYPE_Car dealer', 'PA_O_WEEKDAY_APPR_PROCESS_STAR
T_MONDAY', 'PA_O_NAME_CLIENT_TYPE_Refresher', 'PA_O_FLAG_LAST_APPL_PER CONTR
ACT_N', 'PA_O_NAME_CASH_LOAN_PURPOSE_Business development', 'PA_O_CODE_REJEC
T_REASON_SCOFR', 'PA_O_CHANNEL_TYPE_Stone', 'PA_O_NAME_GOODS_CATEGORY_Mobile
', 'PA_O_WEEKDAY_APPR_PROCESS_START_FRIDAY', 'PA_O_NAME_PAYMENT_TYPE_XNA',
'PA_O_PRODUCT_COMBINATION_POS industry with interest', 'PA_O_NAME_CASH_LOAN_P
URPOSE_Buying a new car', 'PA_O_NAME_PORTFOLIO_POS', 'PA_O_NAME_PAYMENT_TYPE
_Cash through the bank', 'PA_O_NAME_CONTRACT_STATUS_Canceled', 'PA_O_WEEKDAY
_APPR_PROCESS_START_THURSDAY', 'PA_O_NAME_CASH_LOAN_PURPOSE_Buying a home',
'PA_O_NAME_CLIENT_TYPE_Repeater', 'PA_O_NAME_GOODS_CATEGORY_Photo / Cinema E
```

quipment', 'PA_O_NAME_GOODS_CATEGORY_Construction Materials', 'PA_O_NAME_GOODS_CATEGORY_Vehicles', 'PA_O_NAME_CASH_LOAN_PURPOSE_Medicine', 'PA_O_NAME_CLIENT_TYPE_New', 'PA_O_NAME_CONTRACT_STATUS_Refused', 'PA_O_NAME_CASH_LOAN_PURPOSE_Payments on other loans', 'PA_O_NAME_CASH_LOAN_PURPOSE_Repairs', 'PA_O_NAME_CONTRACT_STATUS_Approved', 'PA_O_NAME_GOODS_CATEGORY_Animals', 'PA_O_NAME_GOODS_CATEGORY_Sport and Leisure', 'PA_O_PRODUCT_COMBINATION_POS mobile without interest', 'PA_O_CASH_LOAN_PURPOSE_XAP', 'PA_O_NAME_GOODS_CATEGORY_Office Appliances', 'PA_O_CHANNEL_TYPE_Channel of corporate sales', 'PA_O_NAME_YIELD_GROUP_middle', 'PA_O_NAME_YIELD_GROUP_low_action', 'PA_O_PRODUCT_COMBINATION_POS mobile with interest', 'PA_O_NAME_GOODS_CATEGORY_Other', 'PA_O_NAME_GOODS_CATEGORY_Education', 'PA_O_NAME_GOODS_CATEGORY_Tourism', 'PA_O_NAME_TYPE_SUITE_Children', 'PA_O_NAME_GOODS_CATEGORY_Jewelry', 'PA_O_PRODUCT_COMBINATION_Cash Street: low', 'PA_O_CODE_REJECT_REASON_SYSTEM', 'PA_O_NAME_CONTRACT_TYPE_Consumer loans', 'PA_O_NAME_GOODS_CATEGORY_Weapon', 'PA_O_NAME_GOODS_CATEGORY_Homewares', 'PA_O_NAME_CASH_LOAN_PURPOSE_Purchase of electronic equipment', 'PA_O_CODE_REJECT_REASON_CLIENT', 'PA_O_NAME_CASH_LOAN_PURPOSE_Journey', 'PA_O_NAME_CONTRACT_TYPE_Revolving loans', 'PA_O_PRODUCT_COMBINATION_Cash X-Sell: middle', 'PA_O_CODE_REJECT_REASON_SCO', 'PA_O_CODE_REJECT_REASON_XAP', 'PA_O_NAME_CASH_LOAN_PURPOSE_Building a house or an annex', 'PA_O_NAME_YIELD_GROUP_low_normal', 'PA_O_NAME_GOODS_CATEGORY_Additional Service', 'PA_O_NAME_CASH_LOAN_PURPOSE_Urgent needs', 'PA_O_NAME_CASH_LOAN_PURPOSE_Furniture', 'PA_O_CODE_REJECT_REASON_LIMIT', 'PA_O_NAME_TYPE_SUITE_Other_A', 'PA_O_PRODUCT_COMBINATION_POS others without interest', 'PA_O_NAME_TYPE_SUITE_Other_B', 'PA_O_NAME_GOODS_CATEGORY_Auto Accessories', 'PA_O_NAME_PORTFOLIO_Cash', 'PA_O_NAME_CASH_LOAN_PURPOSE_Education', 'PA_O_NAME_GOODS_CATEGORY_XNA', 'PA_O_CHANNEL_TYPE_AP+ (Cash loan)', 'PA_O_PRODUCT_COMBINATION_Card X-Sell', 'PA_O_NAME_CASH_LOAN_PURPOSE_Everyday expenses', 'PA_O_NAME_CONTRACT_TYPE_XNA', 'PA_O_NAME_CASH_LOAN_PURPOSE_Buying a used car', 'PA_O_CODE_REJECT_REASON_VERIF', 'PA_O_NAME_GOODS_CATEGORY_Computers', 'PA_O_NAME_SELLER_INDUSTRY_Furniture', 'PA_O_WEEKDAY_APPR_PROCESS_START_SUNDAY', 'PA_O_NAME_CASH_LOAN_PURPOSE_Buying a garage', 'PA_O_NAME_PRODUCT_TYPE_XNA', 'PA_O_CHANNEL_TYPE_Contact center', 'PA_O_NAME_SELLER_INDUSTRY_XNA', 'PA_O_PRODUCT_COMBINATION_POS household without interest', 'PA_O_NAME_PAYMENT_TYPE_Non-cash from your account', 'PA_O_NAME_GOODS_CATEGORY_Consumer Electronics', 'PA_O_NAME_CASH_LOAN_PURPOSE_Money for a third person', 'PA_O_PRODUCT_COMBINATION_Cash', 'PA_O_CODE_REJECT_REASON_HC', 'PA_O_WEEKDAY_APPR_PROCESS_START_SATURDAY', 'PA_O_CHANNEL_TYPE_Credit and cash offices', 'PA_O_NAME_SELLER_INDUSTRY_Construction', 'PA_O_NAME_PRODUCT_TYPE_walk-in', 'PA_O_NAME_SELLER_INDUSTRY_Auto technology', 'PA_O_NAME_GOODS_CATEGORY_Insurance', 'PA_O_NAME_YIELD_GROUP_XNA', 'PA_O_CASH_LOAN_PURPOSE_XNA', 'PA_O_NAME_SELLER_INDUSTRY_Connectivity', 'PA_O_PRODUCT_COMBINATION_Cash Street: middle', 'PA_O_NAME_CASH_LOAN_PURPOSE_Gasification / water supply', 'PA_O_PRODUCT_COMBINATION_Cash X-Sell: high', 'PA_O_NAME_CASH_LOAN_PURPOSE_XAP', 'PA_O_NAME_TYPE_SUITE_Spouse, partner', 'PA_O_NAME_CLIENT_TYPE_XNA', 'PA_O_NAME_GOODS_CATEGORY_Fitness', 'PA_O_NAME_SELLER_INDUSTRY_Consumer electronics', 'PA_O_NAME_SELLER_INDUSTRY MLM partners', 'PA_O_NAME_CASH_LOAN_PURPOSE_Car repairs', 'PA_O_NAME_PRODUCT_TYPE_x-sell', 'PA_O_CODE_REJECT_REASON_XNA', 'PA_O_NAME_GOODS_CATEGORY_House Construction', 'PA_O_CHANNEL_TYPE_Regional / Local']

df.shape: (1670214, 187)

Aggregated Features:

df[['PA_N_L_SELLERPLACE_AREA', 'PA_O_NAME_CASH_LOAN_PURPOSE_Hobby', 'PA_O_NAME_CONTRACT_TYPE_Cash loans', 'PA_O_CASH_LOAN_PURPOSE_Other', 'PA_O_PRODUCT_COMBINATION_POS other with interest', 'PA_O_NFLAG_LAST_APPL_IN_DAY', 'PA_O_NAME_GOODS_CATEGORY_Clothing and Accessories', 'PA_O_NAME_SELLER_INDUSTRY_Tourism', 'PA_O_DAYS_LAST_DUE_1ST_VERSION', 'PA_O_PRODUCT_COMBINATION_Cash Street: high', 'PA_O_NAME_SELLER_INDUSTRY_Industry', 'PA_N_APCTN_CRDT_RATIO', 'PA_O_NAME_CASH_LOAN_PURPOSE_Wedding / gift / holiday', 'PA_O_NAME_GOODS_CATEG

ORY_Medicine', 'PA_O_CHANNEL_TYPE_Country-wide', 'PA_O_NAME_PORTFOLIO_Cars', 'PA_O_NAME_CASH_LOAN_PURPOSE_XNA', 'PA_O_NAME_GOODS_CATEGORY_Direct Sales', 'PA_O_NAME_CASH_LOAN_PURPOSE_Other', 'PA_O_PRODUCT_COMBINATION_POS household with interest', 'PA_O_NAME_GOODS_CATEGORY_Medical Supplies', 'PA_N_L_AMT_APP LICATION', 'SK_ID_PREV', 'PA_N_R_CNT_PAYMENT', 'PA_O_NAME_CASH_LOAN_PURPOSE_Refusal to name the goal', 'PA_O_AMT_CREDIT', 'PA_O_NAME_PORTFOLIO_Cards', 'PA_O_PRODUCT_COMBINATION_Cash X-Sell: low', 'PA_O_NFLAG_INSURED_ON_APPROVAL', 'PA_O_PRODUCT_COMBINATION_POS industry without interest', 'PA_O_NAME_YIELD_GROUP_high', 'PA_O_NAME_TYPE_SUITE_Family', 'PA_O_NAME_GOODS_CATEGORY_Gardening', 'PA_O_NAME_PORTFOLIO_XNA', 'PA_O_NAME_SELLER_INDUSTRY_Jewelry', 'PA_O_NAME_CASH_LOAN_PURPOSE_Buying a holiday home / land', 'PA_O_PRODUCT_COMBINATION_Card Street', 'PA_O_NAME_CONTRACT_STATUS_Used offer', 'PA_N_R_AMT_CREDIT', 'PA_N_APCTN_CRDT_DIFF', 'PA_O_NAME_GOODS_CATEGORY_Furniture', 'PA_O_NAME_TYPE_SUITE_Unaccompanied', 'PA_O_FLAG_LAST_APPL_PER_CONTRACT_Y', 'PA_O_NAME_PAYMENT_TYPE_Cashless from the account of the employer', 'PA_O_NAME_SELLER_INDUSTRY_Clothing', 'PA_O_NAME_TYPE_SUITE_Group of people', 'PA_O_WEEKDAY_APPR_PROCESS_START_TUESDAY', 'PA_O_WEEKDAY_APPR_PROCESS_START_WEDNESDAY', 'PA_O_NAME_GOODS_CATEGORY_Audio/Video', 'PA_O_CHANNEL_TYPE_Car dealer', 'PA_O_WEEKDAY_APPR_PROCESS_START_MONDAY', 'PA_O_HOUR_APPR_PROCESS_START', 'PA_O_NAME_CLIENT_TYPE_Refresher', 'PA_O_AMT_APPLICATION', 'PA_O_FLAG_LAST_APPL_PER_CONTRACT_N', 'PA_O_NAME_CASH_LOAN_PURPOSE_Business development', 'PA_O_CODE_REJECT_REASON_SCOFR', 'PA_O_CHANNEL_TYPE_Stone', 'PA_N_R_DAYS_DECISION', 'PA_O_NAME_GOODS_CATEGORY_Mobile', 'PA_O_WEEKDAY_APPR_PROCESS_START_FRIDAY', 'PA_O_NAME_PAYMENT_TYPE_XNA', 'PA_O_PRODUCT_COMBINATION_POS industry with interest', 'PA_O_DAYS_FIRST_DUE', 'PA_O_NAME_CASH_LOAN_PURPOSE_Buying a new car', 'PA_O_NAME_PORTFOLIO_POS', 'PA_O_DAYS_TERMINATION', 'PA_O_NAME_PAYMENT_TYPE_Cash through the bank', 'PA_O_AMT_DOWN_PAYMENT', 'PA_O_NAME_CONTRACT_STATUS_Canceled', 'PA_O_WEEKDAY_APPR_PROCESS_START_THURSDAY', 'PA_O_NAME_CASH_LOAN_PURPOSE_Buying a home', 'PA_O_NAME_CLIENT_TYPE_Repeater', 'PA_O_NAME_GOODS_CATEGORY_Photo / Cinema Equipment', 'PA_O_RATE_DOWN_PAYMENT', 'PA_N_REJ_CNT', 'PA_N_CRDT_ANNUITY_RATIO', 'PA_N_L_CRDT_ANNUITY_RATIO', 'PA_O_NAME_GOODS_CATEGORY_Construction Materials', 'PA_O_NAME_GOODS_CATEGORY_Vehicles', 'PA_O_NAME_CASH_LOAN_PURPOSE_Medicine', 'PA_O_NAME_CLIENT_TYPE_New', 'PA_O_NAME_CONTRACT_STATUS_Refused', 'PA_N_L_AMT_DOWN_PAYMENT', 'PA_O_AMT_GOODS_PRICE', 'PA_O_NAME_CASH_LOAN_PURPOSE_Payments on other loans', 'PA_O_NAME_CASH_LOAN_PURPOSE_Repairs', 'PA_O_NAME_CONTRACT_STATUS_Approved', 'PA_O_DAYS_LAST_DUE', 'PA_O_NAME_GOODS_CATEGORY_Animals', 'PA_O_NAME_GOODS_CATEGORY_Sport and Leisure', 'PA_O_PRODUCT_COMBINATION_POS mobile without interest', 'PA_N_DWN_PYMNT_CRDT_RATIO', 'PA_O_CASH_LOAN_PURPOSE_XAP', 'PA_O_AMT_ANNUITY', 'PA_O_NAME_GOODS_CATEGORY_Office Appliances', 'PA_O_CHANNEL_TYPE_Channel of corporate sales', 'PA_O_NAME_YIELD_GROUP_middle', 'PA_O_NAME_YIELD_GROUP_low_action', 'PA_O_PRODUCT_COMBINATION_POS mobile with interest', 'PA_O_NAME_GOODS_CATEGORY_Other', 'PA_O_NAME_GOODS_CATEGORY_Education', 'PA_O_NAME_GOODS_CATEGORY_Tourism', 'PA_O_NAME_TYPE_SUITE_Children', 'PA_O_NAME_GOODS_CATEGORY_Jewelry', 'PA_O_PRODUCT_COMBINATION_Cash Street: low', 'PA_O_PREV_DWN_PYMNT_CRDT_RATIO', 'PA_O_CODE_REJECT_REASON_SYSTEM', 'PA_O_NAME_CONTRACT_TYPE_Consumer loans', 'PA_O_NAME_GOODS_CATEGORY_Weapon', 'PA_O_NAME_GOODS_CATEGORY_Homewares', 'PA_O_NAME_CASH_LOAN_PURPOSE_Purchase of electronic equipment', 'PA_O_CODE_REJECT_REASON_CLIENT', 'PA_O_NAME_CASH_LOAN_PURPOSE_Journey', 'PA_O_NAME_CONTRACT_TYPE_Revolving loans', 'PA_O_PRODUCT_COMBINATION_Cash X-Sell: middle', 'PA_O_CODE_REJECT_REASON_SCO', 'PA_O_CODE_REJECT_REASON_XAP', 'PA_O_NAME_CASH_LOAN_PURPOSE_Building a house or an annex', 'PA_O_NAME_YIELD_GROUP_low_normal', 'PA_O_NAME_GOODS_CATEGORY_Additional Service', 'PA_O_PREV_APCTN_CRDT_RATIO', 'PA_O_NAME_CASH_LOAN_PURPOSE_Urgent needs', 'PA_O_NAME_CASH_LOAN_PURPOSE_Furniture', 'PA_O_CODE_REJECT_REASON_LIMIT', 'PA_O_NAME_TYPE_SUITE_Other_A', 'PA_O_PRODUCT_COMBINATION_POS others without interest', 'PA_O_NAME_TYPE_SUITE_Other_B', 'PA_O_NAME_GOODS_CATEGORY_Auto Accessories', 'PA_O_NAME_PORTFOLIO_Cash', 'PA_O_NAME_CASH_LOAN_PURPOSE_Education', 'PA_O_NAME_GOODS_CATEGORY_XNA', 'PA_O_CHANNEL_TYPE_AP+ (Cash loan)', 'PA_O_PRODUCT_COMBINATION_Ca

rd X-Sell', 'PA_O_NAME_CASH_LOAN_PURPOSE_Everyday expenses', 'PA_O_NAME_CONTACT_TYPE_XNA', 'PA_O_NAME_CASH_LOAN_PURPOSE_Buying a used car', 'PA_O_CODE_REJECT_REASON_VERIF', 'PA_O_NAME_GOODS_CATEGORY_Computers', 'PA_O_NAME_SELLER_INDUSTRY_Furniture', 'PA_O_WEEKDAY_APPR_PROCESS_START_SUNDAY', 'PA_O_NAME_CASH_LOAN_PURPOSE_Buying a garage', 'PA_O_PREV_CRDT_ANNUITY_RATIO', 'PA_O_NAME_PRODUCT_TYPE_XNA', 'PA_O_CHANNEL_TYPE_Contact center', 'PA_N_L_PREV_APCTN_CRDT_DIFF', 'PA_O_NAME_SELLER_INDUSTRY_XNA', 'PA_O_PRODUCT_COMBINATION_POS_household without interest', 'PA_O_NAME_PAYMENT_TYPE_Non-cash from your account', 'PA_O_NAME_GOODS_CATEGORY_Consumer Electronics', 'PA_O_NAME_CASH_LOAN_PURPOSE_Money for a third person', 'PA_O_CNT_PAYMENT', 'PA_O_PRODUCT_COMBINATION_Cash', 'PA_O_CODE_REJECT_REASON_HC', 'PA_O_WEEKDAY_APPR_PROCESS_START_SATURDAY', 'PA_O_DAYS_DECISION', 'PA_O_CHANNEL_TYPE_Credit and cash offices', 'PA_O_NAME_SELLER_INDUSTRY_Construction', 'PA_N_APRA_CNT', 'PA_O_NAME_PRODUCT_TYPE_walk-in', 'PA_O_NAME_SELLER_INDUSTRY_Auto technology', 'PA_O_NAME_GOODS_CATEGORY_House Construction', 'PA_O_NAME_GOODS_CATEGORY_Insurance', 'PA_O_NAME_YIELD_GROUP_XNA', 'PA_N_R_AMT_ANNUITY', 'PA_O_SELLERPLACE_AREA', 'PA_O_CASH_LOAN_PURPOSE_XNA', 'PA_N_R_AMT_GOODS_PRICE', 'PA_O_NAME_SELLER_INDUSTRY_Connectivity', 'PA_N_R_DWN_PYMNT_CRDT_RATIO', 'PA_O_PRODUCT_COMBINATION_Cash Street: middle', 'PA_O_NAME_CASH_LOAN_PURPOSE_Gasification / water supply', 'PA_O_PRODUCT_COMBINATION_Cash X-Sell: high', 'PA_O_NAME_CASH_LOAN_PURPOSE_XAP', 'PA_O_NAME_TYPE_SUITE_Spouse, partner', 'PA_O_NAME_CLIENT_TYPE_XNA', 'PA_O_PREV_APCTN_CRDT_DIFF', 'PA_O_NAME_GOODS_CATEGORY_Fitness', 'PA_O_NAME_SELLER_INDUSTRY_Consumer electronics', 'PA_O_NAME_SELLER_INDUSTRY_MLM_partners', 'PA_O_NAME_CASH_LOAN_PURPOSE_Car repairs', 'PA_O_NAME_PRODUCT_TYPE_x-sell', 'PA_O_CODE_REJECT_REASON_XNA', 'PA_N_R_RATE_DOWN_PAYMENT', 'PA_O_CHANNEL_TYPE_Regional / Local', 'PA_N_L_DAYS_TERMINATION']] [0:5]:
 PA_N_L_SELLERPLACE_AREA PA_O_NAME_CASH_LOAN_PURPOSE_Hobby \

0	3.583519	0
1	0.693147	0
2	0.693147	0
3	0.693147	0
4	0.693147	0

0	PA_O_NAME_CONTRACT_TYPE_Cash loans	PA_O_CASH_LOAN_PURPOSE_Other	\
1	0	0	
2	1	0	
3	1	0	
4	1	1	

0	PA_O_PRODUCT_COMBINATION_POS_other with interest	\
1	0	
2	0	
3	0	
4	0	

0	PA_O_NFLAG_LAST_APPL_IN_DAY	\
1	1	
2	1	
3	1	
4	1	

0	PA_O_NAME_GOODS_CATEGORY_Clothing and Accessories	\
1	0	
2	0	
3	0	

4		0
0	PA_O_NAME_SELLER_INDUSTRY_Tourism	PA_O_DAYS_LAST_DUE_1ST_VERSION \ 0
1		300.0
2		916.0
3		59.0
4		152.0
		NaN
F	PA_O_PRODUCT_COMBINATION_Cash Street: high ...	PA_O_PREV_APCTN_CRDT_DIF \ 0
0		0 ... 0.
0		
1		0 ... -72171.
0		
2		0 ... -23944.
5		
3		0 ... -20790.
0		
4		1 ... -66555.
0		
PA_O_NAME_GOODS_CATEGORY_Fitness	\ 0	
0		0
1		0
2		0
3		0
4		0
PA_O_NAME_SELLER_INDUSTRY_Consumer electronics	\ 0	
0		0
1		0
2		0
3		0
4		0
PA_O_NAME_SELLER_INDUSTRY_MLM partners	\ 0	
0		0
1		0
2		0
3		0
4		0
PA_O_NAME_CASH_LOAN_PURPOSE_Car repairs	PA_O_NAME_PRODUCT_TYPE_x-sell \ 0	0
1		1
2		1
3		1
4		0
PA_O_CODE_REJECT_REASON_XNA	PA_N_R_RATE_DOWN_PAYMENT \ 0	0.0
1		NaN
2		NaN
3		NaN
4		NaN
PA_O_CHANNEL_TYPE_Regional / Local	PA_N_L_DAYS_TERMINATION \ 0	3.637586

1	0	12.808321
2	0	12.808321
3	0	5.181784
4	0	NaN

[5 rows x 186 columns]

Aggregated Features:

PA_N_APCTN_CRDT_DIFF_count
PA_N_APCTN_CRDT_DIFF_max
PA_N_APCTN_CRDT_DIFF_mean
PA_N_APCTN_CRDT_DIFF_median
PA_N_APCTN_CRDT_DIFF_min
PA_N_APCTN_CRDT_DIFF_sum
PA_N_APCTN_CRDT_DIFF_var
PA_N_APCTN_CRDT_RATIO_count
PA_N_APCTN_CRDT_RATIO_max
PA_N_APCTN_CRDT_RATIO_mean
PA_N_APCTN_CRDT_RATIO_median
PA_N_APCTN_CRDT_RATIO_min
PA_N_APCTN_CRDT_RATIO_sum
PA_N_APCTN_CRDT_RATIO_var
PA_N_APRA_CNT_count
PA_N_APRA_CNT_max
PA_N_APRA_CNT_mean
PA_N_APRA_CNT_median
PA_N_APRA_CNT_min
PA_N_APRA_CNT_sum
PA_N_APRA_CNT_var
PA_N_CRDT_ANNUITY_RATIO_count
PA_N_CRDT_ANNUITY_RATIO_max
PA_N_CRDT_ANNUITY_RATIO_mean
PA_N_CRDT_ANNUITY_RATIO_median
PA_N_CRDT_ANNUITY_RATIO_min
PA_N_CRDT_ANNUITY_RATIO_sum
PA_N_CRDT_ANNUITY_RATIO_var
PA_N_DWN_PYMNT_CRDT_RATIO_count
PA_N_DWN_PYMNT_CRDT_RATIO_max
PA_N_DWN_PYMNT_CRDT_RATIO_mean
PA_N_DWN_PYMNT_CRDT_RATIO_median
PA_N_DWN_PYMNT_CRDT_RATIO_min
PA_N_DWN_PYMNT_CRDT_RATIO_sum
PA_N_DWN_PYMNT_CRDT_RATIO_var
PA_N_L_AMT_APPLICATION_count
PA_N_L_AMT_APPLICATION_max
PA_N_L_AMT_APPLICATION_mean
PA_N_L_AMT_APPLICATION_median
PA_N_L_AMT_APPLICATION_min
PA_N_L_AMT_APPLICATION_sum
PA_N_L_AMT_APPLICATION_var
PA_N_L_AMT_DOWN_PAYMENT_count
PA_N_L_AMT_DOWN_PAYMENT_max
PA_N_L_AMT_DOWN_PAYMENT_mean
PA_N_L_AMT_DOWN_PAYMENT_median
PA_N_L_AMT_DOWN_PAYMENT_min
PA_N_L_AMT_DOWN_PAYMENT_sum
PA_N_L_AMT_DOWN_PAYMENT_var
PA_N_L_CRDT_ANNUITY_RATIO_count
PA_N_L_CRDT_ANNUITY_RATIO_max

PA_N_L_CRDT_ANNUITY_RATIO_mean
PA_N_L_CRDT_ANNUITY_RATIO_median
PA_N_L_CRDT_ANNUITY_RATIO_min
PA_N_L_CRDT_ANNUITY_RATIO_sum
PA_N_L_CRDT_ANNUITY_RATIO_var
PA_N_L_DAYS_TERMINATION_count
PA_N_L_DAYS_TERMINATION_max
PA_N_L_DAYS_TERMINATION_mean
PA_N_L_DAYS_TERMINATION_median
PA_N_L_DAYS_TERMINATION_min
PA_N_L_DAYS_TERMINATION_sum
PA_N_L_DAYS_TERMINATION_var
PA_N_L_PREV_APCTN_CRDT_DIFF_count
PA_N_L_PREV_APCTN_CRDT_DIFF_max
PA_N_L_PREV_APCTN_CRDT_DIFF_mean
PA_N_L_PREV_APCTN_CRDT_DIFF_median
PA_N_L_PREV_APCTN_CRDT_DIFF_min
PA_N_L_PREV_APCTN_CRDT_DIFF_sum
PA_N_L_PREV_APCTN_CRDT_DIFF_var
PA_N_L_SELLERPLACE_AREA_count
PA_N_L_SELLERPLACE_AREA_max
PA_N_L_SELLERPLACE_AREA_mean
PA_N_L_SELLERPLACE_AREA_median
PA_N_L_SELLERPLACE_AREA_min
PA_N_L_SELLERPLACE_AREA_sum
PA_N_L_SELLERPLACE_AREA_var
PA_N_REJ_CNT_count
PA_N_REJ_CNT_max
PA_N_REJ_CNT_mean
PA_N_REJ_CNT_median
PA_N_REJ_CNT_min
PA_N_REJ_CNT_sum
PA_N_REJ_CNT_var
PA_N_R_AMT_ANNUITY_count
PA_N_R_AMT_ANNUITY_max
PA_N_R_AMT_ANNUITY_mean
PA_N_R_AMT_ANNUITY_median
PA_N_R_AMT_ANNUITY_min
PA_N_R_AMT_ANNUITY_sum
PA_N_R_AMT_ANNUITY_var
PA_N_R_AMT_CREDIT_count
PA_N_R_AMT_CREDIT_max
PA_N_R_AMT_CREDIT_mean
PA_N_R_AMT_CREDIT_median
PA_N_R_AMT_CREDIT_min
PA_N_R_AMT_CREDIT_sum
PA_N_R_AMT_CREDIT_var
PA_N_R_AMT_GOODS_PRICE_count
PA_N_R_AMT_GOODS_PRICE_max
PA_N_R_AMT_GOODS_PRICE_mean
PA_N_R_AMT_GOODS_PRICE_median
PA_N_R_AMT_GOODS_PRICE_min
PA_N_R_AMT_GOODS_PRICE_sum
PA_N_R_AMT_GOODS_PRICE_var
PA_N_R_CNT_PAYMENT_count
PA_N_R_CNT_PAYMENT_max
PA_N_R_CNT_PAYMENT_mean
PA_N_R_CNT_PAYMENT_median
PA_N_R_CNT_PAYMENT_min

PA_N_R_CNT_PAYMENT_sum
PA_N_R_CNT_PAYMENT_var
PA_N_R_DAYS_DECISION_count
PA_N_R_DAYS_DECISION_max
PA_N_R_DAYS_DECISION_mean
PA_N_R_DAYS_DECISION_median
PA_N_R_DAYS_DECISION_min
PA_N_R_DAYS_DECISION_sum
PA_N_R_DAYS_DECISION_var
PA_N_R_DWN_PYMNT_CREDT_RATIO_count
PA_N_R_DWN_PYMNT_CREDT_RATIO_max
PA_N_R_DWN_PYMNT_CREDT_RATIO_mean
PA_N_R_DWN_PYMNT_CREDT_RATIO_median
PA_N_R_DWN_PYMNT_CREDT_RATIO_min
PA_N_R_DWN_PYMNT_CREDT_RATIO_sum
PA_N_R_DWN_PYMNT_CREDT_RATIO_var
PA_N_R_RATE_DOWN_PAYMENT_count
PA_N_R_RATE_DOWN_PAYMENT_max
PA_N_R_RATE_DOWN_PAYMENT_mean
PA_N_R_RATE_DOWN_PAYMENT_median
PA_N_R_RATE_DOWN_PAYMENT_min
PA_N_R_RATE_DOWN_PAYMENT_sum
PA_N_R_RATE_DOWN_PAYMENT_var
PA_O_AMT_ANNUITY_count
PA_O_AMT_ANNUITY_max
PA_O_AMT_ANNUITY_mean
PA_O_AMT_ANNUITY_median
PA_O_AMT_ANNUITY_min
PA_O_AMT_ANNUITY_sum
PA_O_AMT_ANNUITY_var
PA_O_AMT_APPLICATION_count
PA_O_AMT_APPLICATION_max
PA_O_AMT_APPLICATION_mean
PA_O_AMT_APPLICATION_median
PA_O_AMT_APPLICATION_min
PA_O_AMT_APPLICATION_sum
PA_O_AMT_APPLICATION_var
PA_O_AMT_CREDIT_count
PA_O_AMT_CREDIT_max
PA_O_AMT_CREDIT_mean
PA_O_AMT_CREDIT_median
PA_O_AMT_CREDIT_min
PA_O_AMT_CREDIT_sum
PA_O_AMT_CREDIT_var
PA_O_AMT_DOWN_PAYMENT_count
PA_O_AMT_DOWN_PAYMENT_max
PA_O_AMT_DOWN_PAYMENT_mean
PA_O_AMT_DOWN_PAYMENT_median
PA_O_AMT_DOWN_PAYMENT_min
PA_O_AMT_DOWN_PAYMENT_sum
PA_O_AMT_DOWN_PAYMENT_var
PA_O_AMT_GOODS_PRICE_count
PA_O_AMT_GOODS_PRICE_max
PA_O_AMT_GOODS_PRICE_mean
PA_O_AMT_GOODS_PRICE_median
PA_O_AMT_GOODS_PRICE_min
PA_O_AMT_GOODS_PRICE_sum
PA_O_AMT_GOODS_PRICE_var
PA_O_CASH_LOAN_PURPOSE_Other_mean

PA_O_CASH_LOAN_PURPOSE_Other_median
PA_O_CASH_LOAN_PURPOSE_Other_var
PA_O_CASH_LOAN_PURPOSE_XAP_mean
PA_O_CASH_LOAN_PURPOSE_XAP_median
PA_O_CASH_LOAN_PURPOSE_XAP_var
PA_O_CASH_LOAN_PURPOSE_XNA_mean
PA_O_CASH_LOAN_PURPOSE_XNA_median
PA_O_CASH_LOAN_PURPOSE_XNA_var
PA_O_CHANNEL_TYPE_AP+ (Cash loan)_mean
PA_O_CHANNEL_TYPE_AP+ (Cash loan)_median
PA_O_CHANNEL_TYPE_AP+ (Cash loan)_var
PA_O_CHANNEL_TYPE_Car dealer_mean
PA_O_CHANNEL_TYPE_Car dealer_median
PA_O_CHANNEL_TYPE_Car dealer_var
PA_O_CHANNEL_TYPE_Channel of corporate sales_mean
PA_O_CHANNEL_TYPE_Channel of corporate sales_median
PA_O_CHANNEL_TYPE_Channel of corporate sales_var
PA_O_CHANNEL_TYPE_Contact center_mean
PA_O_CHANNEL_TYPE_Contact center_median
PA_O_CHANNEL_TYPE_Contact center_var
PA_O_CHANNEL_TYPE_Country-wide_mean
PA_O_CHANNEL_TYPE_Country-wide_median
PA_O_CHANNEL_TYPE_Country-wide_var
PA_O_CHANNEL_TYPE_Credit and cash offices_mean
PA_O_CHANNEL_TYPE_Credit and cash offices_median
PA_O_CHANNEL_TYPE_Credit and cash offices_var
PA_O_CHANNEL_TYPE_Regional / Local_mean
PA_O_CHANNEL_TYPE_Regional / Local_median
PA_O_CHANNEL_TYPE_Regional / Local_var
PA_O_CHANNEL_TYPE_Stone_mean
PA_O_CHANNEL_TYPE_Stone_median
PA_O_CHANNEL_TYPE_Stone_var
PA_O_CNT_PAYMENT_count
PA_O_CNT_PAYMENT_max
PA_O_CNT_PAYMENT_mean
PA_O_CNT_PAYMENT_median
PA_O_CNT_PAYMENT_min
PA_O_CNT_PAYMENT_sum
PA_O_CNT_PAYMENT_var
PA_O_CODE_REJECT_REASON_CLIENT_mean
PA_O_CODE_REJECT_REASON_CLIENT_median
PA_O_CODE_REJECT_REASON_CLIENT_var
PA_O_CODE_REJECT_REASON_HC_mean
PA_O_CODE_REJECT_REASON_HC_median
PA_O_CODE_REJECT_REASON_HC_var
PA_O_CODE_REJECT_REASON_LIMIT_mean
PA_O_CODE_REJECT_REASON_LIMIT_median
PA_O_CODE_REJECT_REASON_LIMIT_var
PA_O_CODE_REJECT_REASON_SCOFR_mean
PA_O_CODE_REJECT_REASON_SCOFR_median
PA_O_CODE_REJECT_REASON_SCOFR_var
PA_O_CODE_REJECT_REASON_SCO_mean
PA_O_CODE_REJECT_REASON_SCO_median
PA_O_CODE_REJECT_REASON_SCO_var
PA_O_CODE_REJECT_REASON_SYSTEM_mean
PA_O_CODE_REJECT_REASON_SYSTEM_median
PA_O_CODE_REJECT_REASON_SYSTEM_var
PA_O_CODE_REJECT_REASON_VERIF_mean
PA_O_CODE_REJECT_REASON_VERIF_median

PA_O_CODE_REJECT_REASON_VERIF_var
PA_O_CODE_REJECT_REASON_XAP_mean
PA_O_CODE_REJECT_REASON_XAP_median
PA_O_CODE_REJECT_REASON_XAP_var
PA_O_CODE_REJECT_REASON_XNA_mean
PA_O_CODE_REJECT_REASON_XNA_median
PA_O_CODE_REJECT_REASON_XNA_var
PA_O_DAYS_DECISION_count
PA_O_DAYS_DECISION_max
PA_O_DAYS_DECISION_mean
PA_O_DAYS_DECISION_median
PA_O_DAYS_DECISION_min
PA_O_DAYS_DECISION_sum
PA_O_DAYS_DECISION_var
PA_O_DAYS_FIRST_DUE_count
PA_O_DAYS_FIRST_DUE_max
PA_O_DAYS_FIRST_DUE_mean
PA_O_DAYS_FIRST_DUE_median
PA_O_DAYS_FIRST_DUE_min
PA_O_DAYS_FIRST_DUE_sum
PA_O_DAYS_FIRST_DUE_var
PA_O_DAYS_LAST_DUE_1ST_VERSION_count
PA_O_DAYS_LAST_DUE_1ST_VERSION_max
PA_O_DAYS_LAST_DUE_1ST_VERSION_mean
PA_O_DAYS_LAST_DUE_1ST_VERSION_median
PA_O_DAYS_LAST_DUE_1ST_VERSION_min
PA_O_DAYS_LAST_DUE_1ST_VERSION_sum
PA_O_DAYS_LAST_DUE_1ST_VERSION_var
PA_O_DAYS_LAST_DUE_count
PA_O_DAYS_LAST_DUE_max
PA_O_DAYS_LAST_DUE_mean
PA_O_DAYS_LAST_DUE_median
PA_O_DAYS_LAST_DUE_min
PA_O_DAYS_LAST_DUE_sum
PA_O_DAYS_LAST_DUE_var
PA_O_DAYS_TERMINATION_count
PA_O_DAYS_TERMINATION_max
PA_O_DAYS_TERMINATION_mean
PA_O_DAYS_TERMINATION_median
PA_O_DAYS_TERMINATION_min
PA_O_DAYS_TERMINATION_sum
PA_O_DAYS_TERMINATION_var
PA_O_FLAG_LAST_APPL_PER_CONTRACT_N_mean
PA_O_FLAG_LAST_APPL_PER_CONTRACT_N_median
PA_O_FLAG_LAST_APPL_PER_CONTRACT_N_var
PA_O_FLAG_LAST_APPL_PER_CONTRACT_Y_mean
PA_O_FLAG_LAST_APPL_PER_CONTRACT_Y_median
PA_O_FLAG_LAST_APPL_PER_CONTRACT_Y_var
PA_O_HOUR_APPR_PROCESS_START_count
PA_O_HOUR_APPR_PROCESS_START_max
PA_O_HOUR_APPR_PROCESS_START_mean
PA_O_HOUR_APPR_PROCESS_START_median
PA_O_HOUR_APPR_PROCESS_START_min
PA_O_HOUR_APPR_PROCESS_START_sum
PA_O_HOUR_APPR_PROCESS_START_var
PA_O_NAME_CASH_LOAN_PURPOSE_Building a house or an annex_mean
PA_O_NAME_CASH_LOAN_PURPOSE_Building a house or an annex_median
PA_O_NAME_CASH_LOAN_PURPOSE_Building a house or an annex_var
PA_O_NAME_CASH_LOAN_PURPOSE_Business development_mean

PA_O_NAME_CASH_LOAN_PURPOSE_Business development_median
PA_O_NAME_CASH_LOAN_PURPOSE_Business development_var
PA_O_NAME_CASH_LOAN_PURPOSE_Buying a garage_mean
PA_O_NAME_CASH_LOAN_PURPOSE_Buying a garage_median
PA_O_NAME_CASH_LOAN_PURPOSE_Buying a garage_var
PA_O_NAME_CASH_LOAN_PURPOSE_Buying a holiday home / land_mean
PA_O_NAME_CASH_LOAN_PURPOSE_Buying a holiday home / land_median
PA_O_NAME_CASH_LOAN_PURPOSE_Buying a holiday home / land_var
PA_O_NAME_CASH_LOAN_PURPOSE_Buying a home_mean
PA_O_NAME_CASH_LOAN_PURPOSE_Buying a home_median
PA_O_NAME_CASH_LOAN_PURPOSE_Buying a home_var
PA_O_NAME_CASH_LOAN_PURPOSE_Buying a new car_mean
PA_O_NAME_CASH_LOAN_PURPOSE_Buying a new car_median
PA_O_NAME_CASH_LOAN_PURPOSE_Buying a new car_var
PA_O_NAME_CASH_LOAN_PURPOSE_Buying a used car_mean
PA_O_NAME_CASH_LOAN_PURPOSE_Buying a used car_median
PA_O_NAME_CASH_LOAN_PURPOSE_Buying a used car_var
PA_O_NAME_CASH_LOAN_PURPOSE_Car repairs_mean
PA_O_NAME_CASH_LOAN_PURPOSE_Car repairs_median
PA_O_NAME_CASH_LOAN_PURPOSE_Car repairs_var
PA_O_NAME_CASH_LOAN_PURPOSE_Education_mean
PA_O_NAME_CASH_LOAN_PURPOSE_Education_median
PA_O_NAME_CASH_LOAN_PURPOSE_Education_var
PA_O_NAME_CASH_LOAN_PURPOSE_Everyday expenses_mean
PA_O_NAME_CASH_LOAN_PURPOSE_Everyday expenses_median
PA_O_NAME_CASH_LOAN_PURPOSE_Everyday expenses_var
PA_O_NAME_CASH_LOAN_PURPOSE_Furniture_mean
PA_O_NAME_CASH_LOAN_PURPOSE_Furniture_median
PA_O_NAME_CASH_LOAN_PURPOSE_Furniture_var
PA_O_NAME_CASH_LOAN_PURPOSE_Gasification / water supply_mean
PA_O_NAME_CASH_LOAN_PURPOSE_Gasification / water supply_median
PA_O_NAME_CASH_LOAN_PURPOSE_Gasification / water supply_var
PA_O_NAME_CASH_LOAN_PURPOSE_Hobby_mean
PA_O_NAME_CASH_LOAN_PURPOSE_Hobby_median
PA_O_NAME_CASH_LOAN_PURPOSE_Hobby_var
PA_O_NAME_CASH_LOAN_PURPOSE_Journey_mean
PA_O_NAME_CASH_LOAN_PURPOSE_Journey_median
PA_O_NAME_CASH_LOAN_PURPOSE_Journey_var
PA_O_NAME_CASH_LOAN_PURPOSE_Medicine_mean
PA_O_NAME_CASH_LOAN_PURPOSE_Medicine_median
PA_O_NAME_CASH_LOAN_PURPOSE_Medicine_var
PA_O_NAME_CASH_LOAN_PURPOSE_Money for a third person_mean
PA_O_NAME_CASH_LOAN_PURPOSE_Money for a third person_median
PA_O_NAME_CASH_LOAN_PURPOSE_Money for a third person_var
PA_O_NAME_CASH_LOAN_PURPOSE_Other_mean
PA_O_NAME_CASH_LOAN_PURPOSE_Other_median
PA_O_NAME_CASH_LOAN_PURPOSE_Other_var
PA_O_NAME_CASH_LOAN_PURPOSE_Payments on other loans_mean
PA_O_NAME_CASH_LOAN_PURPOSE_Payments on other loans_median
PA_O_NAME_CASH_LOAN_PURPOSE_Payments on other loans_var
PA_O_NAME_CASH_LOAN_PURPOSE_Purchase of electronic equipment_mean
PA_O_NAME_CASH_LOAN_PURPOSE_Purchase of electronic equipment_median
PA_O_NAME_CASH_LOAN_PURPOSE_Purchase of electronic equipment_var
PA_O_NAME_CASH_LOAN_PURPOSE_Refusal to name the goal_mean
PA_O_NAME_CASH_LOAN_PURPOSE_Refusal to name the goal_median
PA_O_NAME_CASH_LOAN_PURPOSE_Refusal to name the goal_var
PA_O_NAME_CASH_LOAN_PURPOSE_Repairs_mean
PA_O_NAME_CASH_LOAN_PURPOSE_Repairs_median
PA_O_NAME_CASH_LOAN_PURPOSE_Repairs_var

PA_O_NAME_CASH_LOAN_PURPOSE_Urgent needs_mean
PA_O_NAME_CASH_LOAN_PURPOSE_Urgent needs_median
PA_O_NAME_CASH_LOAN_PURPOSE_Urgent needs_var
PA_O_NAME_CASH_LOAN_PURPOSE_Wedding / gift / holiday_mean
PA_O_NAME_CASH_LOAN_PURPOSE_Wedding / gift / holiday_median
PA_O_NAME_CASH_LOAN_PURPOSE_Wedding / gift / holiday_var
PA_O_NAME_CASH_LOAN_PURPOSE_XAP_mean
PA_O_NAME_CASH_LOAN_PURPOSE_XAP_median
PA_O_NAME_CASH_LOAN_PURPOSE_XAP_var
PA_O_NAME_CASH_LOAN_PURPOSE_XNA_mean
PA_O_NAME_CASH_LOAN_PURPOSE_XNA_median
PA_O_NAME_CASH_LOAN_PURPOSE_XNA_var
PA_O_NAME_CLIENT_TYPE_New_mean
PA_O_NAME_CLIENT_TYPE_New_median
PA_O_NAME_CLIENT_TYPE_New_var
PA_O_NAME_CLIENT_TYPE_Refresher_mean
PA_O_NAME_CLIENT_TYPE_Refresher_median
PA_O_NAME_CLIENT_TYPE_Refresher_var
PA_O_NAME_CLIENT_TYPE_Repeater_mean
PA_O_NAME_CLIENT_TYPE_Repeater_median
PA_O_NAME_CLIENT_TYPE_Repeater_var
PA_O_NAME_CLIENT_TYPE_XNA_mean
PA_O_NAME_CLIENT_TYPE_XNA_median
PA_O_NAME_CLIENT_TYPE_XNA_var
PA_O_NAME_CONTRACT_STATUS_Approved_mean
PA_O_NAME_CONTRACT_STATUS_Approved_median
PA_O_NAME_CONTRACT_STATUS_Approved_var
PA_O_NAME_CONTRACT_STATUS_Canceled_mean
PA_O_NAME_CONTRACT_STATUS_Canceled_median
PA_O_NAME_CONTRACT_STATUS_Canceled_var
PA_O_NAME_CONTRACT_STATUS_Refused_mean
PA_O_NAME_CONTRACT_STATUS_Refused_median
PA_O_NAME_CONTRACT_STATUS_Refused_var
PA_O_NAME_CONTRACT_STATUS_Unused_offer_mean
PA_O_NAME_CONTRACT_STATUS_Unused_offer_median
PA_O_NAME_CONTRACT_STATUS_Unused_offer_var
PA_O_NAME_CONTRACT_TYPE_Cash_loans_mean
PA_O_NAME_CONTRACT_TYPE_Cash_loans_median
PA_O_NAME_CONTRACT_TYPE_Cash_loans_var
PA_O_NAME_CONTRACT_TYPE_Consumer_loans_mean
PA_O_NAME_CONTRACT_TYPE_Consumer_loans_median
PA_O_NAME_CONTRACT_TYPE_Consumer_loans_var
PA_O_NAME_CONTRACT_TYPE_Revolving_loans_mean
PA_O_NAME_CONTRACT_TYPE_Revolving_loans_median
PA_O_NAME_CONTRACT_TYPE_Revolving_loans_var
PA_O_NAME_CONTRACT_TYPE_XNA_mean
PA_O_NAME_CONTRACT_TYPE_XNA_median
PA_O_NAME_CONTRACT_TYPE_XNA_var
PA_O_NAME_GOODS_CATEGORY_Additional_Service_mean
PA_O_NAME_GOODS_CATEGORY_Additional_Service_median
PA_O_NAME_GOODS_CATEGORY_Additional_Service_var
PA_O_NAME_GOODS_CATEGORY_Animals_mean
PA_O_NAME_GOODS_CATEGORY_Animals_median
PA_O_NAME_GOODS_CATEGORY_Animals_var
PA_O_NAME_GOODS_CATEGORY_Audio/Video_mean
PA_O_NAME_GOODS_CATEGORY_Audio/Video_median
PA_O_NAME_GOODS_CATEGORY_Audio/Video_var
PA_O_NAME_GOODS_CATEGORY_Auto_Accessories_mean
PA_O_NAME_GOODS_CATEGORY_Auto_Accessories_median

PA_0_NAME_GOODS_CATEGORY_Auto Accessories_var
PA_0_NAME_GOODS_CATEGORY_Clothing and Accessories_mean
PA_0_NAME_GOODS_CATEGORY_Clothing and Accessories_median
PA_0_NAME_GOODS_CATEGORY_Clothing and Accessories_var
PA_0_NAME_GOODS_CATEGORY_Computers_mean
PA_0_NAME_GOODS_CATEGORY_Computers_median
PA_0_NAME_GOODS_CATEGORY_Computers_var
PA_0_NAME_GOODS_CATEGORY_Construction Materials_mean
PA_0_NAME_GOODS_CATEGORY_Construction Materials_median
PA_0_NAME_GOODS_CATEGORY_Construction Materials_var
PA_0_NAME_GOODS_CATEGORY_Consumer Electronics_mean
PA_0_NAME_GOODS_CATEGORY_Consumer Electronics_median
PA_0_NAME_GOODS_CATEGORY_Consumer Electronics_var
PA_0_NAME_GOODS_CATEGORY_Direct Sales_mean
PA_0_NAME_GOODS_CATEGORY_Direct Sales_median
PA_0_NAME_GOODS_CATEGORY_Direct Sales_var
PA_0_NAME_GOODS_CATEGORY_Education_mean
PA_0_NAME_GOODS_CATEGORY_Education_median
PA_0_NAME_GOODS_CATEGORY_Education_var
PA_0_NAME_GOODS_CATEGORY_Fitness_mean
PA_0_NAME_GOODS_CATEGORY_Fitness_median
PA_0_NAME_GOODS_CATEGORY_Fitness_var
PA_0_NAME_GOODS_CATEGORY_Furniture_mean
PA_0_NAME_GOODS_CATEGORY_Furniture_median
PA_0_NAME_GOODS_CATEGORY_Furniture_var
PA_0_NAME_GOODS_CATEGORY_Gardening_mean
PA_0_NAME_GOODS_CATEGORY_Gardening_median
PA_0_NAME_GOODS_CATEGORY_Gardening_var
PA_0_NAME_GOODS_CATEGORY_Homewares_mean
PA_0_NAME_GOODS_CATEGORY_Homewares_median
PA_0_NAME_GOODS_CATEGORY_Homewares_var
PA_0_NAME_GOODS_CATEGORY_House Construction_mean
PA_0_NAME_GOODS_CATEGORY_House Construction_median
PA_0_NAME_GOODS_CATEGORY_House Construction_var
PA_0_NAME_GOODS_CATEGORY_Insurance_mean
PA_0_NAME_GOODS_CATEGORY_Insurance_median
PA_0_NAME_GOODS_CATEGORY_Insurance_var
PA_0_NAME_GOODS_CATEGORY_Jewelry_mean
PA_0_NAME_GOODS_CATEGORY_Jewelry_median
PA_0_NAME_GOODS_CATEGORY_Jewelry_var
PA_0_NAME_GOODS_CATEGORY_Medical Supplies_mean
PA_0_NAME_GOODS_CATEGORY_Medical Supplies_median
PA_0_NAME_GOODS_CATEGORY_Medical Supplies_var
PA_0_NAME_GOODS_CATEGORY_Medicine_mean
PA_0_NAME_GOODS_CATEGORY_Medicine_median
PA_0_NAME_GOODS_CATEGORY_Medicine_var
PA_0_NAME_GOODS_CATEGORY_Mobile_mean
PA_0_NAME_GOODS_CATEGORY_Mobile_median
PA_0_NAME_GOODS_CATEGORY_Mobile_var
PA_0_NAME_GOODS_CATEGORY_Office Appliances_mean
PA_0_NAME_GOODS_CATEGORY_Office Appliances_median
PA_0_NAME_GOODS_CATEGORY_Office Appliances_var
PA_0_NAME_GOODS_CATEGORY_Other_mean
PA_0_NAME_GOODS_CATEGORY_Other_median
PA_0_NAME_GOODS_CATEGORY_Other_var
PA_0_NAME_GOODS_CATEGORY_Photo / Cinema Equipment_mean
PA_0_NAME_GOODS_CATEGORY_Photo / Cinema Equipment_median
PA_0_NAME_GOODS_CATEGORY_Photo / Cinema Equipment_var
PA_0_NAME_GOODS_CATEGORY_Sport and Leisure_mean

PA_O_NAME_GOODS_CATEGORY_Sport and Leisure_median
PA_O_NAME_GOODS_CATEGORY_Sport and Leisure_var
PA_O_NAME_GOODS_CATEGORY_Tourism_mean
PA_O_NAME_GOODS_CATEGORY_Tourism_median
PA_O_NAME_GOODS_CATEGORY_Tourism_var
PA_O_NAME_GOODS_CATEGORY_Vehicles_mean
PA_O_NAME_GOODS_CATEGORY_Vehicles_median
PA_O_NAME_GOODS_CATEGORY_Vehicles_var
PA_O_NAME_GOODS_CATEGORY_Weapon_mean
PA_O_NAME_GOODS_CATEGORY_Weapon_median
PA_O_NAME_GOODS_CATEGORY_Weapon_var
PA_O_NAME_GOODS_CATEGORY_XNA_mean
PA_O_NAME_GOODS_CATEGORY_XNA_median
PA_O_NAME_GOODS_CATEGORY_XNA_var
PA_O_NAME_PAYMENT_TYPE_Cash through the bank_mean
PA_O_NAME_PAYMENT_TYPE_Cash through the bank_median
PA_O_NAME_PAYMENT_TYPE_Cash through the bank_var
PA_O_NAME_PAYMENT_TYPE_Cashless from the account of the employer_mean
PA_O_NAME_PAYMENT_TYPE_Cashless from the account of the employer_median
PA_O_NAME_PAYMENT_TYPE_Cashless from the account of the employer_var
PA_O_NAME_PAYMENT_TYPE_Non-cash from your account_mean
PA_O_NAME_PAYMENT_TYPE_Non-cash from your account_median
PA_O_NAME_PAYMENT_TYPE_Non-cash from your account_var
PA_O_NAME_PAYMENT_TYPE_XNA_mean
PA_O_NAME_PAYMENT_TYPE_XNA_median
PA_O_NAME_PAYMENT_TYPE_XNA_var
PA_O_NAME_PORTFOLIO_Cards_mean
PA_O_NAME_PORTFOLIO_Cards_median
PA_O_NAME_PORTFOLIO_Cards_var
PA_O_NAME_PORTFOLIO_Cars_mean
PA_O_NAME_PORTFOLIO_Cars_median
PA_O_NAME_PORTFOLIO_Cars_var
PA_O_NAME_PORTFOLIO_Cash_mean
PA_O_NAME_PORTFOLIO_Cash_median
PA_O_NAME_PORTFOLIO_Cash_var
PA_O_NAME_PORTFOLIO_POS_mean
PA_O_NAME_PORTFOLIO_POS_median
PA_O_NAME_PORTFOLIO_POS_var
PA_O_NAME_PORTFOLIO_XNA_mean
PA_O_NAME_PORTFOLIO_XNA_median
PA_O_NAME_PORTFOLIO_XNA_var
PA_O_NAME_PRODUCT_TYPE_XNA_mean
PA_O_NAME_PRODUCT_TYPE_XNA_median
PA_O_NAME_PRODUCT_TYPE_XNA_var
PA_O_NAME_PRODUCT_TYPE_walk-in_mean
PA_O_NAME_PRODUCT_TYPE_walk-in_median
PA_O_NAME_PRODUCT_TYPE_walk-in_var
PA_O_NAME_PRODUCT_TYPE_x-sell_mean
PA_O_NAME_PRODUCT_TYPE_x-sell_median
PA_O_NAME_PRODUCT_TYPE_x-sell_var
PA_O_NAME_SELLER_INDUSTRY_Auto technology_mean
PA_O_NAME_SELLER_INDUSTRY_Auto technology_median
PA_O_NAME_SELLER_INDUSTRY_Auto technology_var
PA_O_NAME_SELLER_INDUSTRY_Clothing_mean
PA_O_NAME_SELLER_INDUSTRY_Clothing_median
PA_O_NAME_SELLER_INDUSTRY_Clothing_var
PA_O_NAME_SELLER_INDUSTRY_Connectivity_mean
PA_O_NAME_SELLER_INDUSTRY_Connectivity_median
PA_O_NAME_SELLER_INDUSTRY_Connectivity_var

PA_O_NAME_SELLER_INDUSTRY_Construction_mean
PA_O_NAME_SELLER_INDUSTRY_Construction_median
PA_O_NAME_SELLER_INDUSTRY_Construction_var
PA_O_NAME_SELLER_INDUSTRY_Consumer electronics_mean
PA_O_NAME_SELLER_INDUSTRY_Consumer electronics_median
PA_O_NAME_SELLER_INDUSTRY_Consumer electronics_var
PA_O_NAME_SELLER_INDUSTRY_Furniture_mean
PA_O_NAME_SELLER_INDUSTRY_Furniture_median
PA_O_NAME_SELLER_INDUSTRY_Furniture_var
PA_O_NAME_SELLER_INDUSTRY_Industry_mean
PA_O_NAME_SELLER_INDUSTRY_Industry_median
PA_O_NAME_SELLER_INDUSTRY_Industry_var
PA_O_NAME_SELLER_INDUSTRY_Jewelry_mean
PA_O_NAME_SELLER_INDUSTRY_Jewelry_median
PA_O_NAME_SELLER_INDUSTRY_Jewelry_var
PA_O_NAME_SELLER_INDUSTRY MLM partners_mean
PA_O_NAME_SELLER_INDUSTRY MLM partners_median
PA_O_NAME_SELLER_INDUSTRY MLM partners_var
PA_O_NAME_SELLER_INDUSTRY_Tourism_mean
PA_O_NAME_SELLER_INDUSTRY_Tourism_median
PA_O_NAME_SELLER_INDUSTRY_Tourism_var
PA_O_NAME_SELLER_INDUSTRY_XNA_mean
PA_O_NAME_SELLER_INDUSTRY_XNA_median
PA_O_NAME_SELLER_INDUSTRY_XNA_var
PA_O_NAME_TYPE_SUITE_Children_mean
PA_O_NAME_TYPE_SUITE_Children_median
PA_O_NAME_TYPE_SUITE_Children_var
PA_O_NAME_TYPE_SUITE_Family_mean
PA_O_NAME_TYPE_SUITE_Family_median
PA_O_NAME_TYPE_SUITE_Family_var
PA_O_NAME_TYPE_SUITE_Group of people_mean
PA_O_NAME_TYPE_SUITE_Group of people_median
PA_O_NAME_TYPE_SUITE_Group of people_var
PA_O_NAME_TYPE_SUITE_Other_A_mean
PA_O_NAME_TYPE_SUITE_Other_A_median
PA_O_NAME_TYPE_SUITE_Other_A_var
PA_O_NAME_TYPE_SUITE_Other_B_mean
PA_O_NAME_TYPE_SUITE_Other_B_median
PA_O_NAME_TYPE_SUITE_Other_B_var
PA_O_NAME_TYPE_SUITE_Spouse, partner_mean
PA_O_NAME_TYPE_SUITE_Spouse, partner_median
PA_O_NAME_TYPE_SUITE_Spouse, partner_var
PA_O_NAME_TYPE_SUITE_Unaccompanied_mean
PA_O_NAME_TYPE_SUITE_Unaccompanied_median
PA_O_NAME_TYPE_SUITE_Unaccompanied_var
PA_O_NAME_YIELD_GROUP_XNA_mean
PA_O_NAME_YIELD_GROUP_XNA_median
PA_O_NAME_YIELD_GROUP_XNA_var
PA_O_NAME_YIELD_GROUP_high_mean
PA_O_NAME_YIELD_GROUP_high_median
PA_O_NAME_YIELD_GROUP_high_var
PA_O_NAME_YIELD_GROUP_low_action_mean
PA_O_NAME_YIELD_GROUP_low_action_median
PA_O_NAME_YIELD_GROUP_low_action_var
PA_O_NAME_YIELD_GROUP_low_normal_mean
PA_O_NAME_YIELD_GROUP_low_normal_median
PA_O_NAME_YIELD_GROUP_low_normal_var
PA_O_NAME_YIELD_GROUP_middle_mean
PA_O_NAME_YIELD_GROUP_middle_median

PA_O_NAME_YIELD_GROUP_middle_var
PA_O_NFLAG_INSURED_ON_APPROVAL_count
PA_O_NFLAG_INSURED_ON_APPROVAL_max
PA_O_NFLAG_INSURED_ON_APPROVAL_mean
PA_O_NFLAG_INSURED_ON_APPROVAL_median
PA_O_NFLAG_INSURED_ON_APPROVAL_min
PA_O_NFLAG_INSURED_ON_APPROVAL_sum
PA_O_NFLAG_INSURED_ON_APPROVAL_var
PA_O_NFLAG_LAST_APPL_IN_DAY_count
PA_O_NFLAG_LAST_APPL_IN_DAY_max
PA_O_NFLAG_LAST_APPL_IN_DAY_mean
PA_O_NFLAG_LAST_APPL_IN_DAY_median
PA_O_NFLAG_LAST_APPL_IN_DAY_min
PA_O_NFLAG_LAST_APPL_IN_DAY_sum
PA_O_NFLAG_LAST_APPL_IN_DAY_var
PA_O_PREV_APCTN_CRDT_DIFF_count
PA_O_PREV_APCTN_CRDT_DIFF_max
PA_O_PREV_APCTN_CRDT_DIFF_mean
PA_O_PREV_APCTN_CRDT_DIFF_median
PA_O_PREV_APCTN_CRDT_DIFF_min
PA_O_PREV_APCTN_CRDT_DIFF_sum
PA_O_PREV_APCTN_CRDT_DIFF_var
PA_O_PREV_APCTN_CRDT_RATIO_count
PA_O_PREV_APCTN_CRDT_RATIO_max
PA_O_PREV_APCTN_CRDT_RATIO_mean
PA_O_PREV_APCTN_CRDT_RATIO_median
PA_O_PREV_APCTN_CRDT_RATIO_min
PA_O_PREV_APCTN_CRDT_RATIO_sum
PA_O_PREV_APCTN_CRDT_RATIO_var
PA_O_PREV_CRDT_ANNUITY_RATIO_count
PA_O_PREV_CRDT_ANNUITY_RATIO_max
PA_O_PREV_CRDT_ANNUITY_RATIO_mean
PA_O_PREV_CRDT_ANNUITY_RATIO_median
PA_O_PREV_CRDT_ANNUITY_RATIO_min
PA_O_PREV_CRDT_ANNUITY_RATIO_sum
PA_O_PREV_CRDT_ANNUITY_RATIO_var
PA_O_PREV_DWN_PYMNT_CRDT_RATIO_count
PA_O_PREV_DWN_PYMNT_CRDT_RATIO_max
PA_O_PREV_DWN_PYMNT_CRDT_RATIO_mean
PA_O_PREV_DWN_PYMNT_CRDT_RATIO_median
PA_O_PREV_DWN_PYMNT_CRDT_RATIO_min
PA_O_PREV_DWN_PYMNT_CRDT_RATIO_sum
PA_O_PREV_DWN_PYMNT_CRDT_RATIO_var
PA_O_PRODUCT_COMBINATION_Card_Street_mean
PA_O_PRODUCT_COMBINATION_Card_Street_median
PA_O_PRODUCT_COMBINATION_Card_Street_var
PA_O_PRODUCT_COMBINATION_Card_X-Sell_mean
PA_O_PRODUCT_COMBINATION_Card_X-Sell_median
PA_O_PRODUCT_COMBINATION_Card_X-Sell_var
PA_O_PRODUCT_COMBINATION_Cash_Street_high_mean
PA_O_PRODUCT_COMBINATION_Cash_Street_high_median
PA_O_PRODUCT_COMBINATION_Cash_Street_high_var
PA_O_PRODUCT_COMBINATION_Cash_Street_low_mean
PA_O_PRODUCT_COMBINATION_Cash_Street_low_median
PA_O_PRODUCT_COMBINATION_Cash_Street_low_var
PA_O_PRODUCT_COMBINATION_Cash_Street_middle_mean
PA_O_PRODUCT_COMBINATION_Cash_Street_middle_median
PA_O_PRODUCT_COMBINATION_Cash_Street_middle_var
PA_O_PRODUCT_COMBINATION_Cash_X-Sell_high_mean

PA_O_PRODUCT_COMBINATION_Cash X-Sell: high_median
PA_O_PRODUCT_COMBINATION_Cash X-Sell: high_var
PA_O_PRODUCT_COMBINATION_Cash X-Sell: low_mean
PA_O_PRODUCT_COMBINATION_Cash X-Sell: low_median
PA_O_PRODUCT_COMBINATION_Cash X-Sell: low_var
PA_O_PRODUCT_COMBINATION_Cash X-Sell: middle_mean
PA_O_PRODUCT_COMBINATION_Cash X-Sell: middle_median
PA_O_PRODUCT_COMBINATION_Cash X-Sell: middle_var
PA_O_PRODUCT_COMBINATION_Cash_mean
PA_O_PRODUCT_COMBINATION_Cash_median
PA_O_PRODUCT_COMBINATION_Cash_var
PA_O_PRODUCT_COMBINATION_POS household with interest_mean
PA_O_PRODUCT_COMBINATION_POS household with interest_median
PA_O_PRODUCT_COMBINATION_POS household with interest_var
PA_O_PRODUCT_COMBINATION_POS household without interest_mean
PA_O_PRODUCT_COMBINATION_POS household without interest_median
PA_O_PRODUCT_COMBINATION_POS household without interest_var
PA_O_PRODUCT_COMBINATION_POS industry with interest_mean
PA_O_PRODUCT_COMBINATION_POS industry with interest_median
PA_O_PRODUCT_COMBINATION_POS industry with interest_var
PA_O_PRODUCT_COMBINATION_POS industry without interest_mean
PA_O_PRODUCT_COMBINATION_POS industry without interest_median
PA_O_PRODUCT_COMBINATION_POS industry without interest_var
PA_O_PRODUCT_COMBINATION_POS mobile with interest_mean
PA_O_PRODUCT_COMBINATION_POS mobile with interest_median
PA_O_PRODUCT_COMBINATION_POS mobile with interest_var
PA_O_PRODUCT_COMBINATION_POS mobile without interest_mean
PA_O_PRODUCT_COMBINATION_POS mobile without interest_median
PA_O_PRODUCT_COMBINATION_POS mobile without interest_var
PA_O_PRODUCT_COMBINATION_POS other with interest_mean
PA_O_PRODUCT_COMBINATION_POS other with interest_median
PA_O_PRODUCT_COMBINATION_POS other with interest_var
PA_O_PRODUCT_COMBINATION_POS others without interest_mean
PA_O_PRODUCT_COMBINATION_POS others without interest_median
PA_O_PRODUCT_COMBINATION_POS others without interest_var
PA_O_RATE_DOWN_PAYMENT_count
PA_O_RATE_DOWN_PAYMENT_max
PA_O_RATE_DOWN_PAYMENT_mean
PA_O_RATE_DOWN_PAYMENT_median
PA_O_RATE_DOWN_PAYMENT_min
PA_O_RATE_DOWN_PAYMENT_sum
PA_O_RATE_DOWN_PAYMENT_var
PA_O_SELLERPLACE_AREA_count
PA_O_SELLERPLACE_AREA_max
PA_O_SELLERPLACE_AREA_mean
PA_O_SELLERPLACE_AREA_median
PA_O_SELLERPLACE_AREA_min
PA_O_SELLERPLACE_AREA_sum
PA_O_SELLERPLACE_AREA_var
PA_O_WEEKDAY_APPR_PROCESS_START_FRIDAY_mean
PA_O_WEEKDAY_APPR_PROCESS_START_FRIDAY_median
PA_O_WEEKDAY_APPR_PROCESS_START_FRIDAY_var
PA_O_WEEKDAY_APPR_PROCESS_START_MONDAY_mean
PA_O_WEEKDAY_APPR_PROCESS_START_MONDAY_median
PA_O_WEEKDAY_APPR_PROCESS_START_MONDAY_var
PA_O_WEEKDAY_APPR_PROCESS_START_SATURDAY_mean
PA_O_WEEKDAY_APPR_PROCESS_START_SATURDAY_median
PA_O_WEEKDAY_APPR_PROCESS_START_SATURDAY_var
PA_O_WEEKDAY_APPR_PROCESS_START_SUNDAY_mean

```

PA_O_WEEKDAY_APPR_PROCESS_START_SUNDAY_median
PA_O_WEEKDAY_APPR_PROCESS_START_SUNDAY_var
PA_O_WEEKDAY_APPR_PROCESS_START_THURSDAY_mean
PA_O_WEEKDAY_APPR_PROCESS_START_THURSDAY_median
PA_O_WEEKDAY_APPR_PROCESS_START_THURSDAY_var
PA_O_WEEKDAY_APPR_PROCESS_START_TUESDAY_mean
PA_O_WEEKDAY_APPR_PROCESS_START_TUESDAY_median
PA_O_WEEKDAY_APPR_PROCESS_START_TUESDAY_var
PA_O_WEEKDAY_APPR_PROCESS_START_WEDNESDAY_mean
PA_O_WEEKDAY_APPR_PROCESS_START_WEDNESDAY_median
PA_O_WEEKDAY_APPR_PROCESS_START_WEDNESDAY_var
SK_ID_PREV_count

```

Aggregated Data:

	count	mean	\
PA_O_DAYS_FIRST_DUE_sum	338857.0	3123.592229	
PA_O_NAME_CASH_LOAN_PURPOSE_Wedding / gift / ho...	338857.0	0.000193	
PA_N_R_AMT_CREDIT_mean	338857.0	328.016258	
PA_O_NAME_CASH_LOAN_PURPOSE_XNA_mean	338857.0	0.285948	
PA_O_NAME_SELLER_INDUSTRY_Connectivity_median	338857.0	0.158878	
...	
PA_O_NAME_CASH_LOAN_PURPOSE_Refusal to name the...	338857.0	0.000008	
PA_O_DAYS_FIRST_DUE_count	338857.0	2.822736	
PA_N_R_AMT_CREDIT_count	338857.0	4.928961	
PA_N_L_AMT_APPLICATION_max	338857.0	12.258319	
PA_O_CODE_REJECT_REASON_XAP_median	338857.0	0.932714	
	std	min	\
PA_O_DAYS_FIRST_DUE_sum	3000.942419	0.0	
PA_O_NAME_CASH_LOAN_PURPOSE_Wedding / gift / ho...	0.012475	0.0	
PA_N_R_AMT_CREDIT_mean	145.747144	0.0	
PA_O_NAME_CASH_LOAN_PURPOSE_XNA_mean	0.299414	0.0	
PA_O_NAME_SELLER_INDUSTRY_Connectivity_median	0.341318	0.0	
...	
PA_O_NAME_CASH_LOAN_PURPOSE_Refusal to name the...	0.001997	0.0	
PA_O_DAYS_FIRST_DUE_count	1.980678	0.0	
PA_N_R_AMT_CREDIT_count	4.220714	1.0	
PA_N_L_AMT_APPLICATION_max	1.280430	0.0	
PA_O_CODE_REJECT_REASON_XAP_median	0.226373	0.0	
	25%	50%	
\			
PA_O_DAYS_FIRST_DUE_sum	849.000000	2287.000000	
PA_O_NAME_CASH_LOAN_PURPOSE_Wedding / gift / ho...	0.000000	0.000000	
PA_N_R_AMT_CREDIT_mean	228.473193	301.033996	
PA_O_NAME_CASH_LOAN_PURPOSE_XNA_mean	0.000000	0.250000	
PA_O_NAME_SELLER_INDUSTRY_Connectivity_median	0.000000	0.000000	
...	
PA_O_NAME_CASH_LOAN_PURPOSE_Refusal to name the...	0.000000	0.000000	
PA_O_DAYS_FIRST_DUE_count	1.000000	2.000000	
PA_N_R_AMT_CREDIT_count	2.000000	4.000000	
PA_N_L_AMT_APPLICATION_max	11.508641	12.257293	
PA_O_CODE_REJECT_REASON_XAP_median	1.000000	1.000000	
	75%	max	
PA_O_DAYS_FIRST_DUE_sum	4425.000000	42423.00000	
PA_O_NAME_CASH_LOAN_PURPOSE_Wedding / gift / ho...	0.000000	1.000000	
PA_N_R_AMT_CREDIT_mean	396.862697	2012.46118	

```

PA_O_NAME_CASH_LOAN_PURPOSE_XNA_mean           0.500000   1.000000
PA_O_NAME_SELLER_INDUSTRY_Connectivity_median 0.000000   1.000000
...
PA_O_NAME_CASH_LOAN_PURPOSE_Refusal to name the... 0.000000   1.000000
PA_O_DAYS_FIRST_DUE_count                      4.000000  25.000000
PA_N_R_AMT_CREDIT_count                        7.000000  77.000000
PA_N_L_AMT_APPLICATION_max                     13.046564 15.74778
PA_O_CODE_REJECT_REASON_XAP_median            1.000000   1.000000

```

[712 rows x 8 columns]

In [23]:

```

prevapp['PS_O_SK_ID_PREV_count'] = prevapp['SK_ID_PREV_count']
prevapp.drop(['SK_ID_PREV_count'],axis=1,inplace=True)
datasets_transformed['previous_application'] = prevapp
#prevapp.to_csv(os.getcwd() + DATA_DIR + 'prevapp_agg_data_tr.csv')

```

CCB - Credit card Balance EDA

Feature transformation Viz

Using the visualizations below where ever the skew become more centrally distributed compared to norm was log transformed. Where there were a substantial amount of 0's it was left as is.

In [160...]

```
id_cols,num_cols,_,_ = id_num_cat_feature(ccb, text = False)
```

In []:

```

def log_comparision(df):
    id_cols,num_cols,_,_ = id_num_cat_feature(ccb, text = False)
    cols= list(set(num_cols) - set(id_cols))
    ccb.replace([np.inf, -np.inf], 0, inplace=True)
    for i in cols:
        print(i)
        plt.subplot(1, 2, 1)
        plt.xlabel("normal")
        plt.hist(ccb[i], bins=20)
        plt.subplot(1, 2, 2)
        plt.xlabel("log_transformed")
        plt.hist(np.log(ccb[i].abs() + 1), bins=20)
        #fig.tight_layout(pad=5.0)
        plt.show()
ccb = datasets['credit_card_balance']
log_comparision(ccb)

```

In [163...]

```
cols_trans_to_log_ccb = ["CNT_INSTALMENT_MATURE_CUM", "AMT_CREDIT_LIMIT_ACTI
```

EDA and Feature Engineering Function on Credit Card Balance

1. This function is specifically for Credit Card Balance table. Below are the pre-processing done before passing this table into the pipeline.

- Absolute value of the Months Balance is used as that allows only use of positive values without effecting the correlation, as it is relative to the application date.
- Any column or row with more than 70% of its data as null will be deleted from the dataframe as the threshold is set to .7.
- Similar to above, once processed, store the transformed csv file. Benefit of this is that we can then pass it directly to model for merging into application train/test table. We do not have to perform expensive EDA/ETL/Transformation everytime we want to process the same data.

2. Specific Feature Transformations

- The NAME_CONTRACT_STATUS contained the below values with the count and ratio.

	COUNT	RATIO
Active	3698436	0.96
Completed	128918	0.03
Signed	11058	0.00
Demand	1365	0.00
Sent proposal	513	0.00
Refused	17	0.00
Approved	5	0.00

- All values with Active and Complete were made into one category. If complete that is a positive finding or if active that is also a positive finding as that means it payments are being made on it or the status is healthy enough to stay active for the credit.

- The other created column is of a count of the number of credit lines the person has open. This was done by aggregated the SK_ID_CURR. This also can give us some glimpse into number of credit lines open vs the target. The higher the credit lines should demonstrate a positive finding as the newer ones may have been denied as they would have a bad credit score and

possibly not get approved for another line.

CCB Feature Engineering

Addition and Transformation

Please refer to section 6.2.1 and 6.2.6 for details on feature renaming and transformation.

```
In [178]:
```

```
def ccb_eda(df, col_to_drop, cols_trans_to_log_ccb):
    #dropping a highly correlated columns (.95 or .99)
    ccb = df.drop(columns_to_drop, axis = 1)

    #log tranformations
    for column in cols_trans_to_log_ccb:
        name = "AT_L_O_" + column
        df[name] = np.log(df[column].abs() + 1)

    ccb["NAME_CONTRACT_STATUS"] = np.where((ccb["NAME_CONTRACT_STATUS"].isin(
        "Active", "Completed"
    )), "GoodStatus", ccb["NAME_CONTRACT_STATUS"])

    #Adding new features
    ccb['MONTHS_BALANCE'] = ccb['MONTHS_BALANCE'].abs()

    #replacing " " with _ for OHE cols.
    ccb['NAME_CONTRACT_STATUS']=ccb['NAME_CONTRACT_STATUS'].apply(lambda x:
threshold = 0.7

    #Dropping columns with missing value rate higher than threshold
    ccb = ccb[ccb.columns[ccb.isnull().mean() < threshold]]

    #Dropping rows with missing value rate higher than threshold
    ccb = ccb.loc[ccb.isnull().mean(axis=1) < threshold]
    agg = eda_transformation(ccb,4)
    agg['CC_COUNT'] = ccb.groupby('SK_ID_CURR').size()

    return (agg)
```

Highest correlations on Credit Card Balance

Correlation is calculated with the highest correlated columns being dropped. The threshold used was when columns had a correlation of .95 or higher, only one of the columns was kept.

```
In [ ]: high_correlation(datasets['credit_card_balance'], remove=['SK_ID_CURR', 'SK_ID_PREV'], corr_coef="pearson", corr_value = 0.7)

In [166... #below are .99 correlations
columns_to_drop = ["AMT_RECEIVABLE_AT_1ST_DRAWING_DATE", "AMT_TOTAL_RECEIVABLE_AT_1ST_DRAWING_DATE", "AMT_RECEIVABLE_AT_LAST_DRAWING_DATE"]

In [208... #below are .95+ correlations
columns_to_drop = ["AMT_RECEIVABLE_AT_1ST_DRAWING_DATE", "AMT_TOTAL_RECEIVABLE_AT_1ST_DRAWING_DATE", "AMT_RECEIVABLE_AT_LAST_DRAWING_DATE"]
```

In [179...]

```
ccb = datasets['credit_card_balance']
ccb = ccb_eda(ccb, columns_to_drop, cols_trans_to_log_ccb)
datasets_transformed['credit_card_balance'] = ccb

#ccb.to_csv(os.getcwd() + DATA_DIR + 'ccb_agg_data.csv')
```

```

-----
# of ID's: 1
ID's:
['SK_ID_CURR']

-----
# All features: 21
All features:
['AT_L_O_AMT_CREDIT_LIMIT_ACTUAL', 'CNT_INSTALMENT_MATURE_CUM', 'AMT_CREDIT_
LIMIT_ACTUAL', 'AT_L_O_AMT_PAYMENT_CURRENT', 'AT_L_O_CNT_INSTALMENT_MATURE_C
UM', 'AMT_BALANCE', 'SK_ID_PREV', 'SK_DPD', 'CNT_DRAWINGS_OTHER_CURRENT', 'S
K_DPD_DEF', 'CNT_DRAWINGS_ATM_CURRENT', 'CNT_DRAWINGS_POS_CURRENT', 'AMT_DRA
WINGS_ATM_CURRENT', 'CNT_DRAWINGS_CURRENT', 'AMT_INST_MIN_REGULARITY', 'MONT
HS_BALANCE', 'AMT_DRAWINGS_CURRENT', 'AMT_DRAWINGS_OTHER_CURRENT', 'NAME_CON
TRACT_STATUS', 'AMT_PAYMENT_CURRENT', 'AMT_DRAWINGS_POS_CURRENT']

```

Missing data:

	Total	Percent
AMT_PAYMENT_CURRENT	767988	19.998063
AT_L_O_AMT_PAYMENT_CURRENT	767988	19.998063
CNT_DRAWINGS_ATM_CURRENT	749816	19.524872
CNT_DRAWINGS_OTHER_CURRENT	749816	19.524872
AMT_DRAWINGS_OTHER_CURRENT	749816	19.524872
AMT_DRAWINGS_ATM_CURRENT	749816	19.524872
CNT_DRAWINGS_POS_CURRENT	749816	19.524872
AMT_DRAWINGS_POS_CURRENT	749816	19.524872
CNT_INSTALMENT_MATURE_CUM	305236	7.948208
AMT_INST_MIN_REGULARITY	305236	7.948208
AT_L_O_CNT_INSTALMENT_MATURE_CUM	305236	7.948208
SK_DPD	0	0.000000
SK_DPD_DEF	0	0.000000
SK_ID_PREV	0	0.000000
AMT_BALANCE	0	0.000000
CNT_DRAWINGS_CURRENT	0	0.000000
MONTHS_BALANCE	0	0.000000
AMT_DRAWINGS_CURRENT	0	0.000000
NAME_CONTRACT_STATUS	0	0.000000
AMT_CREDIT_LIMIT_ACTUAL	0	0.000000
AT_L_O_AMT_CREDIT_LIMIT_ACTUAL	0	0.000000

```

-----
# of Numerical features: 21
Numerical features:

```

```

['SK_ID_PREV', 'SK_ID_CURR', 'MONTHS_BALANCE', 'AMT_BALANCE', 'AMT_CREDIT_LI
MIT_ACTUAL', 'AMT_DRAWINGS_ATM_CURRENT', 'AMT_DRAWINGS_CURRENT', 'AMT_DRAWIN
GS_OTHER_CURRENT', 'AMT_DRAWINGS_POS_CURRENT', 'AMT_INST_MIN_REGULARITY', 'A
MT_PAYMENT_CURRENT', 'CNT_DRAWINGS_ATM_CURRENT', 'CNT_DRAWINGS_CURRENT', 'CN
T_DRAWINGS_OTHER_CURRENT', 'CNT_DRAWINGS_POS_CURRENT', 'CNT_INSTALMENT_MATUR
E_CUM', 'SK_DPD', 'SK_DPD_DEF', 'AT_L_O_CNT_INSTALMENT_MATURE_CUM', 'AT_L_O_
AMT_CREDIT_LIMIT_ACTUAL', 'AT_L_O_AMT_PAYMENT_CURRENT']

```

Numerical Statistical Summary:

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	\
count	3.840312e+06	3.840312e+06	3.840312e+06	3.840312e+06	
mean	1.904504e+06	2.783242e+05	3.452192e+01	5.830016e+04	
std	5.364695e+05	1.027045e+05	2.666775e+01	1.063070e+05	
min	1.000018e+06	1.000060e+05	1.000000e+00	-4.202502e+05	
25%	1.434385e+06	1.895170e+05	1.100000e+01	0.000000e+00	

50%	1.897122e+06	2.783960e+05	2.800000e+01	0.000000e+00	
75%	2.369328e+06	3.675800e+05	5.500000e+01	8.904669e+04	
max	2.843496e+06	4.562500e+05	9.600000e+01	1.505902e+06	
count	3.840312e+06		3.090496e+06		
mean	1.538080e+05		5.961325e+03		
std	1.651457e+05		2.822569e+04		
min	0.000000e+00		-6.827310e+03		
25%	4.500000e+04		0.000000e+00		
50%	1.125000e+05		0.000000e+00		
75%	1.800000e+05		0.000000e+00		
max	1.350000e+06		2.115000e+06		
count	3.840312e+06	AMT_DRAWINGS_CURRENT	AMT_DRAWINGS_OTHER_CURRENT	AMT_INST_MIN_REGULARITY	\
mean	7.433388e+03		2.881696e+02		
std	3.384608e+04		8.201989e+03		
min	-6.211620e+03		0.000000e+00		
25%	0.000000e+00		0.000000e+00		
50%	0.000000e+00		0.000000e+00		
75%	0.000000e+00		0.000000e+00		
max	2.287098e+06		1.529847e+06		
count	3.090496e+06	AMT_DRAWINGS_POS_CURRENT	AMT_INST_MIN_REGULARITY	... \	
mean	2.968805e+03		3.540204e+03		
std	2.079689e+04		5.600154e+03		
min	0.000000e+00		0.000000e+00		
25%	0.000000e+00		0.000000e+00		
50%	0.000000e+00		0.000000e+00		
75%	0.000000e+00		6.633911e+03		
max	2.239274e+06		2.028820e+05		
count	3.090496e+06	CNT_DRAWINGS_ATM_CURRENT	CNT_DRAWINGS_CURRENT	CNT_DRAWINGS_POS_CURRENT	\
mean	3.094490e-01		7.031439e-01		
std	1.100401e+00		3.190347e+00		
min	0.000000e+00		0.000000e+00		
25%	0.000000e+00		0.000000e+00		
50%	0.000000e+00		0.000000e+00		
75%	0.000000e+00		0.000000e+00		
max	5.100000e+01		1.650000e+02		
count	3.090496e+06	CNT_DRAWINGS_OTHER_CURRENT	CNT_DRAWINGS_POS_CURRENT	CNT_INSTALMENT_MATURE_CUM	\
mean	4.812496e-03		3.090496e+06	3.535076e+06	
std	8.263861e-02		5.594791e-01	3.240649e+00	
min	0.000000e+00		0.000000e+00	0.000000e+00	
25%	0.000000e+00		0.000000e+00	0.000000e+00	
50%	0.000000e+00		0.000000e+00	0.000000e+00	
75%	0.000000e+00		0.000000e+00	0.000000e+00	
max	1.200000e+01		1.650000e+02	3.840312e+06	
count	3.535076e+06	CNT_INSTALMENT_MATURE_CUM	SK_DPD	SK_DPD_DEF	\
mean	2.082508e+01	3.840312e+06	3.840312e+06	3.316220e-01	
std	2.005149e+01	9.283667e+00	9.751570e+01	2.147923e+01	
min	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00	

25%	4.000000e+00	0.000000e+00	0.000000e+00
50%	1.500000e+01	0.000000e+00	0.000000e+00
75%	3.200000e+01	0.000000e+00	0.000000e+00
max	1.200000e+02	3.260000e+03	3.260000e+03
	AT_L_O_CNT_INSTALMENT_MATURE_CUM	AT_L_O_AMT_CREDIT_LIMIT_ACTUAL	\
count	3.535076e+06	3.840312e+06	
mean	2.447229e+00	9.548810e+00	
std	1.345021e+00	4.765206e+00	
min	0.000000e+00	0.000000e+00	
25%	1.609438e+00	1.071444e+01	
50%	2.772589e+00	1.163072e+01	
75%	3.496508e+00	1.210072e+01	
max	4.795791e+00	1.411562e+01	
	AT_L_O_AMT_PAYMENT_CURRENT		
count	3.072324e+06		
mean	6.627161e+00		
std	3.345278e+00		
min	0.000000e+00		
25%	5.032853e+00		
50%	7.902376e+00		
75%	9.105091e+00		
max	1.527161e+01		

[8 rows x 21 columns]

of Categorical features: 1

Categorical features:

['NAME_CONTRACT_STATUS']

Categorical Statistical Summary:

Categories:

	NAME_CONTRACT_STATUS
0	GoodStatus
1	Demand
2	Signed
3	Sent_proposal
4	Refused
5	Approved

of OHE categorical features: 6

OHE Categorical features: ['NAME_CONTRACT_STATUS_Demand', 'NAME_CONTRACT_STATUS_Signed', 'NAME_CONTRACT_STATUS_Refused', 'NAME_CONTRACT_STATUS_Approved', 'NAME_CONTRACT_STATUS_GoodStatus']

df.shape: (3840312, 27)

Aggregated Features:

```
df[['AT_L_O_AMT_CREDIT_LIMIT_ACTUAL', 'CNT_INSTALMENT_MATURE_CUM', 'NAME_CONTRACT_STATUS_Sent_proposal', 'AMT_CREDIT_LIMIT_ACTUAL', 'AT_L_O_AMT_PAYMENT_CURRENT', 'NAME_CONTRACT_STATUS_Signed', 'AT_L_O_CNT_INSTALMENT_MATURE_CUM', 'AMT_BALANCE', 'SK_ID_PREV', 'SK_DPD', 'CNT_DRAWINGS_OTHER_CURRENT', 'SK_DPD_DEF', 'CNT_DRAWINGS_ATM_CURRENT', 'CNT_DRAWINGS_POS_CURRENT', 'NAME_CONTRAC
```

T_STATUS_GoodStatus', 'NAME_CONTRACT_STATUS_Demand', 'AMT_DRAWINGS_ATM_CURRENT', 'CNT_DRAWINGS_CURRENT', 'AMT_INST_MIN_REGULARITY', 'MONTHS_BALANCE', 'AMT_DRAWINGS_CURRENT', 'AMT_DRAWINGS_OTHER_CURRENT', 'NAME_CONTRACT_STATUS_Refused', 'AMT_PAYMENT_CURRENT', 'AMT_DRAWINGS_POS_CURRENT', 'NAME_CONTRACT_STATUS_Approved']] [0:5]:

	AT_L_0_AMT_CREDIT_LIMIT_ACTUAL	CNT_INSTALMENT_MATURE_CUM	\
0	11.813037	35.0	
1	10.714440	69.0	
2	13.017005	30.0	
3	12.323860	10.0	
4	13.017005	101.0	

	NAME_CONTRACT_STATUS_Sent_proposal	AMT_CREDIT_LIMIT_ACTUAL	\
0	0	135000	
1	0	45000	
2	0	450000	
3	0	225000	
4	0	450000	

	AT_L_0_AMT_PAYMENT_CURRENT	NAME_CONTRACT_STATUS_Signed	\
0	7.496097	0	
1	7.719130	0	
2	7.719130	0	
3	9.386476	0	
4	10.203629	0	

	AT_L_0_CNT_INSTALMENT_MATURE_CUM	AMT_BALANCE	SK_ID_PREV	SK_DPD	...	\
0	3.583519	56.970	2562384	0	...	
1	4.248495	63975.555	2582071	0	...	
2	3.433987	31815.225	1740877	0	...	
3	2.397895	236572.110	1389973	0	...	
4	4.624973	453919.455	1891521	0	...	

	AMT_DRAWINGS_ATM_CURRENT	CNT_DRAWINGS_CURRENT	AMT_INST_MIN_REGULARITY	\
0	0.0	1	1700.325	
1	2250.0	1	2250.000	
2	0.0	0	2250.000	
3	2250.0	1	11795.760	
4	0.0	1	22924.890	

	MONTHS_BALANCE	AMT_DRAWINGS_CURRENT	AMT_DRAWINGS_OTHER_CURRENT	\
0	6	877.5	0.0	
1	1	2250.0	0.0	
2	7	0.0	0.0	
3	4	2250.0	0.0	
4	1	11547.0	0.0	

	NAME_CONTRACT_STATUS_Refused	AMT_PAYMENT_CURRENT	\
0	0	1800.0	
1	0	2250.0	
2	0	2250.0	
3	0	11925.0	
4	0	27000.0	

	AMT_DRAWINGS_POS_CURRENT	NAME_CONTRACT_STATUS_Approved
0	877.5	0
1	0.0	0
2	0.0	0

3	0.0	0
4	11547.0	0

[5 rows x 26 columns]

Aggregated Features:
AMT_BALANCE_count
AMT_BALANCE_max
AMT_BALANCE_mean
AMT_BALANCE_median
AMT_BALANCE_min
AMT_BALANCE_sum
AMT_BALANCE_var
AMT_CREDIT_LIMIT_ACTUAL_count
AMT_CREDIT_LIMIT_ACTUAL_max
AMT_CREDIT_LIMIT_ACTUAL_mean
AMT_CREDIT_LIMIT_ACTUAL_median
AMT_CREDIT_LIMIT_ACTUAL_min
AMT_CREDIT_LIMIT_ACTUAL_sum
AMT_CREDIT_LIMIT_ACTUAL_var
AMT_DRAWINGS_ATM_CURRENT_count
AMT_DRAWINGS_ATM_CURRENT_max
AMT_DRAWINGS_ATM_CURRENT_mean
AMT_DRAWINGS_ATM_CURRENT_median
AMT_DRAWINGS_ATM_CURRENT_min
AMT_DRAWINGS_ATM_CURRENT_sum
AMT_DRAWINGS_ATM_CURRENT_var
AMT_DRAWINGS_CURRENT_count
AMT_DRAWINGS_CURRENT_max
AMT_DRAWINGS_CURRENT_mean
AMT_DRAWINGS_CURRENT_median
AMT_DRAWINGS_CURRENT_min
AMT_DRAWINGS_CURRENT_sum
AMT_DRAWINGS_CURRENT_var
AMT_DRAWINGS_OTHER_CURRENT_count
AMT_DRAWINGS_OTHER_CURRENT_max
AMT_DRAWINGS_OTHER_CURRENT_mean
AMT_DRAWINGS_OTHER_CURRENT_median
AMT_DRAWINGS_OTHER_CURRENT_min
AMT_DRAWINGS_OTHER_CURRENT_sum
AMT_DRAWINGS_OTHER_CURRENT_var
AMT_DRAWINGS_POS_CURRENT_count
AMT_DRAWINGS_POS_CURRENT_max
AMT_DRAWINGS_POS_CURRENT_mean
AMT_DRAWINGS_POS_CURRENT_median
AMT_DRAWINGS_POS_CURRENT_min
AMT_DRAWINGS_POS_CURRENT_sum
AMT_DRAWINGS_POS_CURRENT_var
AMT_INST_MIN_REGULARITY_count
AMT_INST_MIN_REGULARITY_max
AMT_INST_MIN_REGULARITY_mean
AMT_INST_MIN_REGULARITY_median
AMT_INST_MIN_REGULARITY_min
AMT_INST_MIN_REGULARITY_sum
AMT_INST_MIN_REGULARITY_var
AMT_PAYMENT_CURRENT_count
AMT_PAYMENT_CURRENT_max
AMT_PAYMENT_CURRENT_mean
AMT_PAYMENT_CURRENT_median

AMT_PAYMENT_CURRENT_min
AMT_PAYMENT_CURRENT_sum
AMT_PAYMENT_CURRENT_var
AT_L_0_AMT_CREDIT_LIMIT_ACTUAL_count
AT_L_0_AMT_CREDIT_LIMIT_ACTUAL_max
AT_L_0_AMT_CREDIT_LIMIT_ACTUAL_mean
AT_L_0_AMT_CREDIT_LIMIT_ACTUAL_median
AT_L_0_AMT_CREDIT_LIMIT_ACTUAL_min
AT_L_0_AMT_CREDIT_LIMIT_ACTUAL_sum
AT_L_0_AMT_CREDIT_LIMIT_ACTUAL_var
AT_L_0_AMT_PAYMENT_CURRENT_count
AT_L_0_AMT_PAYMENT_CURRENT_max
AT_L_0_AMT_PAYMENT_CURRENT_mean
AT_L_0_AMT_PAYMENT_CURRENT_median
AT_L_0_AMT_PAYMENT_CURRENT_min
AT_L_0_AMT_PAYMENT_CURRENT_sum
AT_L_0_AMT_PAYMENT_CURRENT_var
AT_L_0_CNT_INSTALMENT_MATURE_CUM_count
AT_L_0_CNT_INSTALMENT_MATURE_CUM_max
AT_L_0_CNT_INSTALMENT_MATURE_CUM_mean
AT_L_0_CNT_INSTALMENT_MATURE_CUM_median
AT_L_0_CNT_INSTALMENT_MATURE_CUM_min
AT_L_0_CNT_INSTALMENT_MATURE_CUM_sum
AT_L_0_CNT_INSTALMENT_MATURE_CUM_var
CNT_DRAWINGS_ATM_CURRENT_count
CNT_DRAWINGS_ATM_CURRENT_max
CNT_DRAWINGS_ATM_CURRENT_mean
CNT_DRAWINGS_ATM_CURRENT_median
CNT_DRAWINGS_ATM_CURRENT_min
CNT_DRAWINGS_ATM_CURRENT_sum
CNT_DRAWINGS_ATM_CURRENT_var
CNT_DRAWINGS_CURRENT_count
CNT_DRAWINGS_CURRENT_max
CNT_DRAWINGS_CURRENT_mean
CNT_DRAWINGS_CURRENT_median
CNT_DRAWINGS_CURRENT_min
CNT_DRAWINGS_CURRENT_sum
CNT_DRAWINGS_CURRENT_var
CNT_DRAWINGS_OTHER_CURRENT_count
CNT_DRAWINGS_OTHER_CURRENT_max
CNT_DRAWINGS_OTHER_CURRENT_mean
CNT_DRAWINGS_OTHER_CURRENT_median
CNT_DRAWINGS_OTHER_CURRENT_min
CNT_DRAWINGS_OTHER_CURRENT_sum
CNT_DRAWINGS_OTHER_CURRENT_var
CNT_DRAWINGS_POS_CURRENT_count
CNT_DRAWINGS_POS_CURRENT_max
CNT_DRAWINGS_POS_CURRENT_mean
CNT_DRAWINGS_POS_CURRENT_median
CNT_DRAWINGS_POS_CURRENT_min
CNT_DRAWINGS_POS_CURRENT_sum
CNT_DRAWINGS_POS_CURRENT_var
CNT_INSTALMENT_MATURE_CUM_count
CNT_INSTALMENT_MATURE_CUM_max
CNT_INSTALMENT_MATURE_CUM_mean
CNT_INSTALMENT_MATURE_CUM_median
CNT_INSTALMENT_MATURE_CUM_min
CNT_INSTALMENT_MATURE_CUM_sum
CNT_INSTALMENT_MATURE_CUM_var

MONTHS_BALANCE_count
 MONTHS_BALANCE_max
 MONTHS_BALANCE_mean
 MONTHS_BALANCE_median
 MONTHS_BALANCE_min
 MONTHS_BALANCE_sum
 MONTHS_BALANCE_var
 NAME_CONTRACT_STATUS_Approved_mean
 NAME_CONTRACT_STATUS_Approved_median
 NAME_CONTRACT_STATUS_Approved_var
 NAME_CONTRACT_STATUS_Demand_mean
 NAME_CONTRACT_STATUS_Demand_median
 NAME_CONTRACT_STATUS_Demand_var
 NAME_CONTRACT_STATUS_GoodStatus_mean
 NAME_CONTRACT_STATUS_GoodStatus_median
 NAME_CONTRACT_STATUS_GoodStatus_var
 NAME_CONTRACT_STATUS_Refused_mean
 NAME_CONTRACT_STATUS_Refused_median
 NAME_CONTRACT_STATUS_Refused_var
 NAME_CONTRACT_STATUS_Sent_proposal_mean
 NAME_CONTRACT_STATUS_Sent_proposal_median
 NAME_CONTRACT_STATUS_Sent_proposal_var
 NAME_CONTRACT_STATUS_Signed_mean
 NAME_CONTRACT_STATUS_Signed_median
 NAME_CONTRACT_STATUS_Signed_var
 SK_DPD_DEF_count
 SK_DPD_DEF_max
 SK_DPD_DEF_mean
 SK_DPD_DEF_median
 SK_DPD_DEF_min
 SK_DPD_DEF_sum
 SK_DPD_DEF_var
 SK_DPD_count
 SK_DPD_max
 SK_DPD_mean
 SK_DPD_median
 SK_DPD_min
 SK_DPD_sum
 SK_DPD_var
 SK_ID_PREV_count

Aggregated Data:

	count	mean	std	\
AMT_BALANCE_mean	103558.0	6.997319e+04	1.075378e+05	
CNT_DRAWINGS_POS_CURRENT_count	103558.0	2.984314e+01	3.542705e+01	
AMT_DRAWINGS_CURRENT_var	102866.0	1.798590e+09	7.646675e+09	
SK_DPD_DEF_var	102866.0	6.627654e+01	6.110026e+03	
CNT_DRAWINGS_CURRENT_count	103558.0	3.708368e+01	3.348363e+01	
...	
CNT_DRAWINGS_CURRENT_var	102866.0	1.487690e+01	6.514073e+01	
AMT_DRAWINGS_ATM_CURRENT_min	72194.0	6.107286e+02	1.117638e+04	
MONTHS_BALANCE_min	103558.0	1.518965e+00	7.348172e-01	
CNT_DRAWINGS_ATM_CURRENT_max	72194.0	3.918234e+00	3.620548e+00	
AT_L_0_AMT_PAYMENT_CURRENT_sum	103558.0	1.966124e+02	2.381554e+02	

	min	25%	50%	7
5% \				
AMT_BALANCE_mean	-2930.232558	0.0	2.499760e+04	9.699775e+0

4					
CNT_DRAWINGS_POS_CURRENT_count	0.000000	0.0	1.200000e+01	5.600000e+0	
1					
AMT_DRAWINGS_CURRENT_var	0.000000	0.0	1.918232e+08	1.215640e+0	
9					
SK_DPD_DEF_var	0.000000	0.0	0.000000e+00	0.000000e+0	
0					
CNT_DRAWINGS_CURRENT_count	1.000000	10.0	2.200000e+01	7.500000e+0	
1					
...	
...	
CNT_DRAWINGS_CURRENT_var	0.000000	0.0	4.674003e-01	4.458620e+0	
0					
AMT_DRAWINGS_ATM_CURRENT_min	-6827.310000	0.0	0.000000e+00	0.000000e+0	
0					
MONTHS_BALANCE_min	1.000000	1.0	1.000000e+00	2.000000e+0	
0					
CNT_DRAWINGS_ATM_CURRENT_max	0.000000	1.0	3.000000e+00	5.000000e+0	
0					
AT_L_0_AMT_PAYMENT_CURRENT_sum	0.000000	0.0	8.631560e+01	3.258712e+0	
2					
		max			
AMT_BALANCE_mean	9.286863e+05				
CNT_DRAWINGS_POS_CURRENT_count	1.920000e+02				
AMT_DRAWINGS_CURRENT_var	6.075000e+11				
SK_DPD_DEF_var	8.456734e+05				
CNT_DRAWINGS_CURRENT_count	1.920000e+02				
...	...				
CNT_DRAWINGS_CURRENT_var	3.528143e+03				
AMT_DRAWINGS_ATM_CURRENT_min	9.045000e+05				
MONTHS_BALANCE_min	1.200000e+01				
CNT_DRAWINGS_ATM_CURRENT_max	5.100000e+01				
AT_L_0_AMT_PAYMENT_CURRENT_sum	1.320039e+03				

[152 rows x 8 columns]

Installment Payments

1. This function is specifically for Installment Payments table. Below are the pre-processing done before passing this table into the pipeline.

- Absolute value of the Days Installment and Days Entry Payment are used as that allows only use of positive values without effecting the correlation. As more negative means longer time from the intial date.
- A new column of IP_DIFF_PAYMNT_INSTMNT was also calculated from difference of AMT_INSTALMENT from AMT_PAYMENT as a part of the feature select/transformation process.
- Any column or row with more than 70% of its data as null will be deleted from the dataframe as the threshold is set to .7.
- Similar to above, once processed, store the transfored csv file. Benefit of this is that we can then pass it directly to model for merging into application train/test table. We do not have to perform expensive EDA/ETL/Transformation everytime we want to process the data.

- **Features Transformation:** Below are the features which are transformed.

IP_N_R_XX: New feature with root transformation. And **IP_N_XX** stands for new feature added.

#Adding new features

```
ip['IP_O_DAYS_INSTALMENT'] = ip['IP_O_DAYS_INSTALMENT'].abs()
```

```
ip['IP_O_DAYS_ENTRY_PAYMENT'] = ip['IP_O_DAYS_ENTRY_PAYMENT'].abs()
```

```
ip['IP_N_DIFF_PAYMNT_INSTMNT'] = ip['IP_O_AMT_PAYMENT'] -  
ip['IP_O_AMT_INSTALMENT']
```

#Root

```
ip['IP_N_R_DAYS_ENTRY_PAYMENT'] =  
np.sqrt(ip['IP_O_DAYS_ENTRY_PAYMENT'].abs())
```

Feature transformation Viz

```
In [ ]:
for i,ds_name in enumerate(datasets.keys()):

    if ds_name.lower() in ("installments_payments"):
        ps = datasets[ds_name]

        ps['IP_DIFF_PAYMNT_INSTLMNT'] = ps['AMT_PAYMENT'] - ps['AMT_INSTALMNT']
        print("-----")
        id_cols,num_cols,_,_ = id_num_cat_feature(ps, text = False)
        id_cols.append('SK_ID_PREV')
        new_cols = ['IP_DIFF_PAYMNT_INSTLMNT']
        cols= list(set(num_cols) - set(id_cols) )
        #cols = ['PS_N_PERC_INSTL_PNDNG', 'PS_N_CNT_INSTAL_PNDNG', 'PS_N_DAYS']
        ps.replace([np.inf, -np.inf], 0, inplace=True)
        for i in cols:
            print('---')
            print(f'{i}')
            print(((ps[i].abs())).skew())
            plt.hist(ps[i], bins=20)
            plt.xlabel(i)
            plt.show()
            print('')
            print(f'{i} :: root-transformed')
            print((np.sqrt(ps[i].abs())).skew())
            plt.hist(np.sqrt(ps[i].abs()), bins=20)
            plt.xlabel(i)
            plt.show()
            print('')
            print(f'{i} :: log-transformed')
            print((np.log(ps[i].abs() + 1)).skew())
            plt.hist(np.log(ps[i].abs() + 1), bins=20)
            plt.xlabel(i)
            plt.show()
```

IP Feature Engineering

Addition and Transformation

Please refer to section 6.2.1 and 6.2.7 for details on feature renaming and transformation.

In [25]:

```
def ip_eda(df):
    ip = df
    drop_list_ip = []

    #Adding new features
    ip['IP_O_DAYS_INSTALMENT'] = ip['IP_O_DAYS_INSTALMENT'].abs()
    ip['IP_O_DAYS_ENTRY_PAYMENT'] = ip['IP_O_DAYS_ENTRY_PAYMENT'].abs()
    ip['IP_N_DIFF_PAYMNT_INSTMNT'] = ip['IP_O_AMT_PAYMENT'] - ip['IP_O_AMT']

    #root
    ip['IP_N_R_DAYS_ENTRY_PAYMENT'] = np.sqrt(ip['IP_O_DAYS_ENTRY_PAYMENT'])

    threshold = 0.7

    #Dropping columns with missing value rate higher than threshold
    ip = ip[ip.columns[ip.isnull().mean() < threshold]]

    #Dropping rows with missing value rate higher than threshold
    ip = ip.loc[ip.isnull().mean(axis=1) < threshold]

    return (eda_transformation(ip,4))
```

In [26]:

```
ip = datasets['installments_payments']
ip.columns = ['IP_O_' + col for col in ip.columns]
ip['SK_ID_CURR'] = ip['IP_O_SK_ID_CURR']
ip['SK_ID_PREV'] = ip['IP_O_SK_ID_PREV']
ip.drop(['IP_O_SK_ID_CURR'],axis=1, inplace=True)
ip.drop(['IP_O_SK_ID_PREV'],axis=1,inplace=True)
```

In [27]:

```
ip = ip_eda(ip)
```

```

-----
# of ID's: 1
ID's:
['SK_ID_CURR']

-----
# All features: 10
All features:
['IP_O_NUM_INSTALMENT_VERSION', 'IP_O_NUM_INSTALMENT_NUMBER', 'IP_N_R_DAYS_E
NTRY_PAYMENT', 'IP_N_DIFF_PAYMNT_INSTLMNT', 'IP_O_AMT_INSTALMENT', 'IP_O_AMT
_PAYMENT', 'IP_O_DAYS_ENTRY_PAYMENT', 'IP_O_DAYS_INSTALMENT', 'SK_ID_PREV',
'IP_O_IP_DIFF_PAYMNT_INSTLMNT']

Missing data:
      Total    Percent
IP_N_R_DAYS_ENTRY_PAYMENT    2905  0.021352
IP_N_DIFF_PAYMNT_INSTLMNT    2905  0.021352
IP_O_AMT_PAYMENT             2905  0.021352
IP_O_DAYS_ENTRY_PAYMENT     2905  0.021352
IP_O_IP_DIFF_PAYMNT_INSTLMNT 2905  0.021352
IP_O_NUM_INSTALMENT_VERSION   0    0.000000
IP_O_NUM_INSTALMENT_NUMBER    0    0.000000
IP_O_AMT_INSTALMENT          0    0.000000
IP_O_DAYS_INSTALMENT         0    0.000000
SK_ID_PREV                   0    0.000000

-----
# of Numerical features: 11
Numerical features:
['IP_O_NUM_INSTALMENT_VERSION', 'IP_O_NUM_INSTALMENT_NUMBER', 'IP_O_DAYS_INS
TALMENT', 'IP_O_DAYS_ENTRY_PAYMENT', 'IP_O_AMT_INSTALMENT', 'IP_O_AMT_PAYMEN
T', 'IP_O_IP_DIFF_PAYMNT_INSTLMNT', 'SK_ID_CURR', 'SK_ID_PREV', 'IP_N_DIFF_P
AYMNT_INSTLMNT', 'IP_N_R_DAYS_ENTRY_PAYMENT']

Numerical Statistical Summary:

      IP_O_NUM_INSTALMENT_VERSION  IP_O_NUM_INSTALMENT_NUMBER \
count           1.360540e+07           1.360540e+07
mean            8.566373e-01           1.887090e+01
std              1.035216e+00           2.666407e+01
min              0.000000e+00           1.000000e+00
25%              0.000000e+00           4.000000e+00
50%              1.000000e+00           8.000000e+00
75%              1.000000e+00           1.900000e+01
max              1.780000e+02           2.770000e+02

      IP_O_DAYS_INSTALMENT  IP_O_DAYS_ENTRY_PAYMENT  IP_O_AMT_INSTALMENT \
count           1.360540e+07           1.360250e+07           1.360540e+07
mean            1.042270e+03           1.051114e+03           1.705091e+04
std              8.009463e+02           8.005859e+02           5.057025e+04
min              1.000000e+00           1.000000e+00           0.000000e+00
25%              3.610000e+02           3.700000e+02           4.226085e+03
50%              8.180000e+02           8.270000e+02           8.884080e+03
75%              1.654000e+03           1.662000e+03           1.671021e+04
max              2.922000e+03           4.921000e+03           3.771488e+06

      IP_O_AMT_PAYMENT  IP_O_IP_DIFF_PAYMNT_INSTLMNT  SK_ID_CURR \
count           1.360250e+07           1.360250e+07           1.360540e+07
mean            1.723822e+04           1.871538e+02           2.784449e+05

```

```

std      5.473578e+04      1.910673e+04  1.027183e+05
min     0.000000e+00     -2.424726e+06  1.000010e+05
25%    3.398265e+03      0.000000e+00  1.896390e+05
50%    8.125515e+03      0.000000e+00  2.786850e+05
75%   1.610842e+04      0.000000e+00  3.675300e+05
max    3.771488e+06      2.630909e+06  4.562550e+05

SK_ID_PREV  IP_N_DIFF_PAYMNT_INSTLMNT  IP_N_R_DAYS_ENTRY_PAYMENT
count  1.360540e+07      1.360250e+07  1.360250e+07
mean   1.903365e+06      1.871538e+02  2.969799e+01
std    5.362029e+05      1.910673e+04  1.300551e+01
min   1.000001e+06     -2.424726e+06  1.000000e+00
25%   1.434191e+06      0.000000e+00  1.923538e+01
50%   1.896520e+06      0.000000e+00  2.875761e+01
75%   2.369094e+06      0.000000e+00  4.076763e+01
max   2.843499e+06      2.630909e+06  7.014984e+01

-----
# of Categorical features: 0
Categorical features:
[]

Categorical Statistical Summary:

Categories:

Series([], dtype: float64)

-----
# of OHE categorical features: 0
OHE Categorical features: []
-----
df.shape: (13605401, 11)

Aggregated Features:
df[['IP_O_NUM_INSTALMENT_VERSION', 'IP_O_NUM_INSTALMENT_NUMBER', 'IP_N_R_DAYS_ENTRY_PAYMENT', 'IP_N_DIFF_PAYMNT_INSTLMNT', 'IP_O_AMT_INSTALMENT', 'IP_O_AMT_PAYMENT', 'IP_O_DAYS_ENTRY_PAYMENT', 'IP_O_DAYS_INSTALMENT', 'SK_ID_PREV', 'IP_O_IP_DIFF_PAYMNT_INSTLMNT']] [0:5]:
  IP_O_NUM_INSTALMENT_VERSION  IP_O_NUM_INSTALMENT_NUMBER \
0                      1.0                         6
1                      0.0                        34
2                      2.0                         1
3                      1.0                         3
4                      1.0                         2

  IP_N_R_DAYS_ENTRY_PAYMENT  IP_N_DIFF_PAYMNT_INSTLMNT  IP_O_AMT_INSTALMENT \
\
0                  34.452866                  0.000          6948.360
1                  46.432747                  0.000          1716.525
2                  7.937254                  0.000         25425.000
3                 49.254441                  0.000         24350.130
4                 36.959437                 -4.455         2165.040

  IP_O_AMT_PAYMENT  IP_O_DAYS_ENTRY_PAYMENT  IP_O_DAYS_INSTALMENT \
0           6948.360                  1187.0            1180.0
1           1716.525                  2156.0            2156.0
2          25425.000                  63.0              63.0

```

3	24350.130	2426.0	2418.0
4	2160.585	1366.0	1383.0

SK_ID_PREV	IP_O_IP_DIFF_PAYMNT_INSTLMNT	
0	1054186	0.000
1	1330831	0.000
2	2085231	0.000
3	2452527	0.000
4	2714724	-4.455

Aggregated Features:

IP_N_DIFF_PAYMNT_INSTLMNT_count
IP_N_DIFF_PAYMNT_INSTLMNT_max
IP_N_DIFF_PAYMNT_INSTLMNT_mean
IP_N_DIFF_PAYMNT_INSTLMNT_median
IP_N_DIFF_PAYMNT_INSTLMNT_min
IP_N_DIFF_PAYMNT_INSTLMNT_sum
IP_N_DIFF_PAYMNT_INSTLMNT_var
IP_N_R_DAYS_ENTRY_PAYMENT_count
IP_N_R_DAYS_ENTRY_PAYMENT_max
IP_N_R_DAYS_ENTRY_PAYMENT_mean
IP_N_R_DAYS_ENTRY_PAYMENT_median
IP_N_R_DAYS_ENTRY_PAYMENT_min
IP_N_R_DAYS_ENTRY_PAYMENT_sum
IP_N_R_DAYS_ENTRY_PAYMENT_var
IP_O_AMT_INSTALMENT_count
IP_O_AMT_INSTALMENT_max
IP_O_AMT_INSTALMENT_mean
IP_O_AMT_INSTALMENT_median
IP_O_AMT_INSTALMENT_min
IP_O_AMT_INSTALMENT_sum
IP_O_AMT_INSTALMENT_var
IP_O_AMT_PAYMENT_count
IP_O_AMT_PAYMENT_max
IP_O_AMT_PAYMENT_mean
IP_O_AMT_PAYMENT_median
IP_O_AMT_PAYMENT_min
IP_O_AMT_PAYMENT_sum
IP_O_AMT_PAYMENT_var
IP_O_DAYS_ENTRY_PAYMENT_count
IP_O_DAYS_ENTRY_PAYMENT_max
IP_O_DAYS_ENTRY_PAYMENT_mean
IP_O_DAYS_ENTRY_PAYMENT_median
IP_O_DAYS_ENTRY_PAYMENT_min
IP_O_DAYS_ENTRY_PAYMENT_sum
IP_O_DAYS_ENTRY_PAYMENT_var
IP_O_DAYS_INSTALMENT_count
IP_O_DAYS_INSTALMENT_max
IP_O_DAYS_INSTALMENT_mean
IP_O_DAYS_INSTALMENT_median
IP_O_DAYS_INSTALMENT_min
IP_O_DAYS_INSTALMENT_sum
IP_O_DAYS_INSTALMENT_var
IP_O_IP_DIFF_PAYMNT_INSTLMNT_count
IP_O_IP_DIFF_PAYMNT_INSTLMNT_max
IP_O_IP_DIFF_PAYMNT_INSTLMNT_mean
IP_O_IP_DIFF_PAYMNT_INSTLMNT_median
IP_O_IP_DIFF_PAYMNT_INSTLMNT_min
IP_O_IP_DIFF_PAYMNT_INSTLMNT_sum

IP_O_IP_DIFF_PAYMNT_INSTLMNT_var
 IP_O_NUM_INSTALMENT_NUMBER_count
 IP_O_NUM_INSTALMENT_NUMBER_max
 IP_O_NUM_INSTALMENT_NUMBER_mean
 IP_O_NUM_INSTALMENT_NUMBER_median
 IP_O_NUM_INSTALMENT_NUMBER_min
 IP_O_NUM_INSTALMENT_NUMBER_sum
 IP_O_NUM_INSTALMENT_NUMBER_var
 IP_O_NUM_INSTALMENT_VERSION_count
 IP_O_NUM_INSTALMENT_VERSION_max
 IP_O_NUM_INSTALMENT_VERSION_mean
 IP_O_NUM_INSTALMENT_VERSION_median
 IP_O_NUM_INSTALMENT_VERSION_min
 IP_O_NUM_INSTALMENT_VERSION_sum
 IP_O_NUM_INSTALMENT_VERSION_var
 SK_ID_PREV_count

Aggregated Data:

	count	mean	std	\
IP_O_NUM_INSTALMENT_VERSION_mean	339587.0	1.047498e+00	6.207509e-01	
IP_N_R_DAYS_ENTRY_PAYMENT_mean	339578.0	2.740665e+01	9.829955e+00	
IP_O_NUM_INSTALMENT_NUMBER_min	339587.0	1.043962e+00	7.479743e-01	
IP_N_R_DAYS_ENTRY_PAYMENT_var	338610.0	7.721861e+01	8.110098e+01	
IP_O_NUM_INSTALMENT_VERSION_sum	339587.0	3.432079e+01	3.825223e+01	
...	
IP_O_AMT_INSTALMENT_sum	339587.0	6.831369e+05	8.933805e+05	
IP_O_AMT_PAYMENT_mean	339578.0	1.900381e+04	2.523135e+04	
IP_N_DIFF_PAYMNT_INSTLMNT_var	338610.0	3.465416e+08	3.435981e+09	
IP_N_DIFF_PAYMNT_INSTLMNT_max	339578.0	2.385596e+04	1.094190e+05	
IP_O_NUM_INSTALMENT_NUMBER_count	339587.0	4.006455e+01	4.105334e+01	
		min	25%	50%
\				
IP_O_NUM_INSTALMENT_VERSION_mean	0.000000	1.000000	1.018519	
IP_N_R_DAYS_ENTRY_PAYMENT_mean	1.732051	19.850258	26.720736	
IP_O_NUM_INSTALMENT_NUMBER_min	1.000000	1.000000	1.000000	
IP_N_R_DAYS_ENTRY_PAYMENT_var	0.000000	11.098620	49.098573	
IP_O_NUM_INSTALMENT_VERSION_sum	0.000000	12.000000	23.000000	
...	
IP_O_AMT_INSTALMENT_sum	0.000000	136683.360000	334395.225000	
IP_O_AMT_PAYMENT_mean	0.189000	7582.365310	12403.981645	
IP_N_DIFF_PAYMNT_INSTLMNT_var	0.000000	0.000000	0.000000	
IP_N_DIFF_PAYMNT_INSTLMNT_max	-20512.350000	0.000000	0.000000	
IP_O_NUM_INSTALMENT_NUMBER_count	1.000000	12.000000	25.000000	
		75%	max	
IP_O_NUM_INSTALMENT_VERSION_mean	1.111111e+00	3.900000e+01		
IP_N_R_DAYS_ENTRY_PAYMENT_mean	3.431009e+01	5.541660e+01		
IP_O_NUM_INSTALMENT_NUMBER_min	1.000000e+00	6.700000e+01		
IP_N_R_DAYS_ENTRY_PAYMENT_var	1.259610e+02	1.180801e+03		
IP_O_NUM_INSTALMENT_VERSION_sum	4.400000e+01	1.826000e+03		
...	
IP_O_AMT_INSTALMENT_sum	8.577727e+05	3.247978e+07		
IP_O_AMT_PAYMENT_mean	2.164505e+04	2.504590e+06		
IP_N_DIFF_PAYMNT_INSTLMNT_var	1.161385e+07	4.459936e+11		
IP_N_DIFF_PAYMNT_INSTLMNT_max	0.000000e+00	2.630909e+06		
IP_O_NUM_INSTALMENT_NUMBER_count	5.100000e+01	3.720000e+02		

```
[64 rows x 8 columns]
```

```
In [31]:  
#ip['IP_0_SK_ID_PREV_count'] = ip['SK_ID_PREV_count']  
#ip.drop(['SK_ID_PREV_count'],axis=1,inplace=True)  
datasets_transformed['installments_payments'] = ip  
#ip.to_csv(os.getcwd() + DATA_DIR + 'ip_agg_data_tr.csv')
```

```
In [ ]:  
#NEW Cols
```

```
In [32]:  
#all new cols  
ip_new_cols = [col for col in ip.columns if "IP_N" in col]  
#cols on which transformation took place  
ip_nt_cols = [col for col in ip.columns if "IP_0_DAYS_ENTRY_PAYMENT" in col]  
#transformed cols  
# root transformed  
ip_r_cols = [col for col in ip.columns if "IP_N_R" in col] #log transformed  
ip_l_cols = [col for col in ip.columns if "IP_N_L" in col]
```

```
In [6]:  
#ip_new_cols
```

Application Train

Feature Engineering

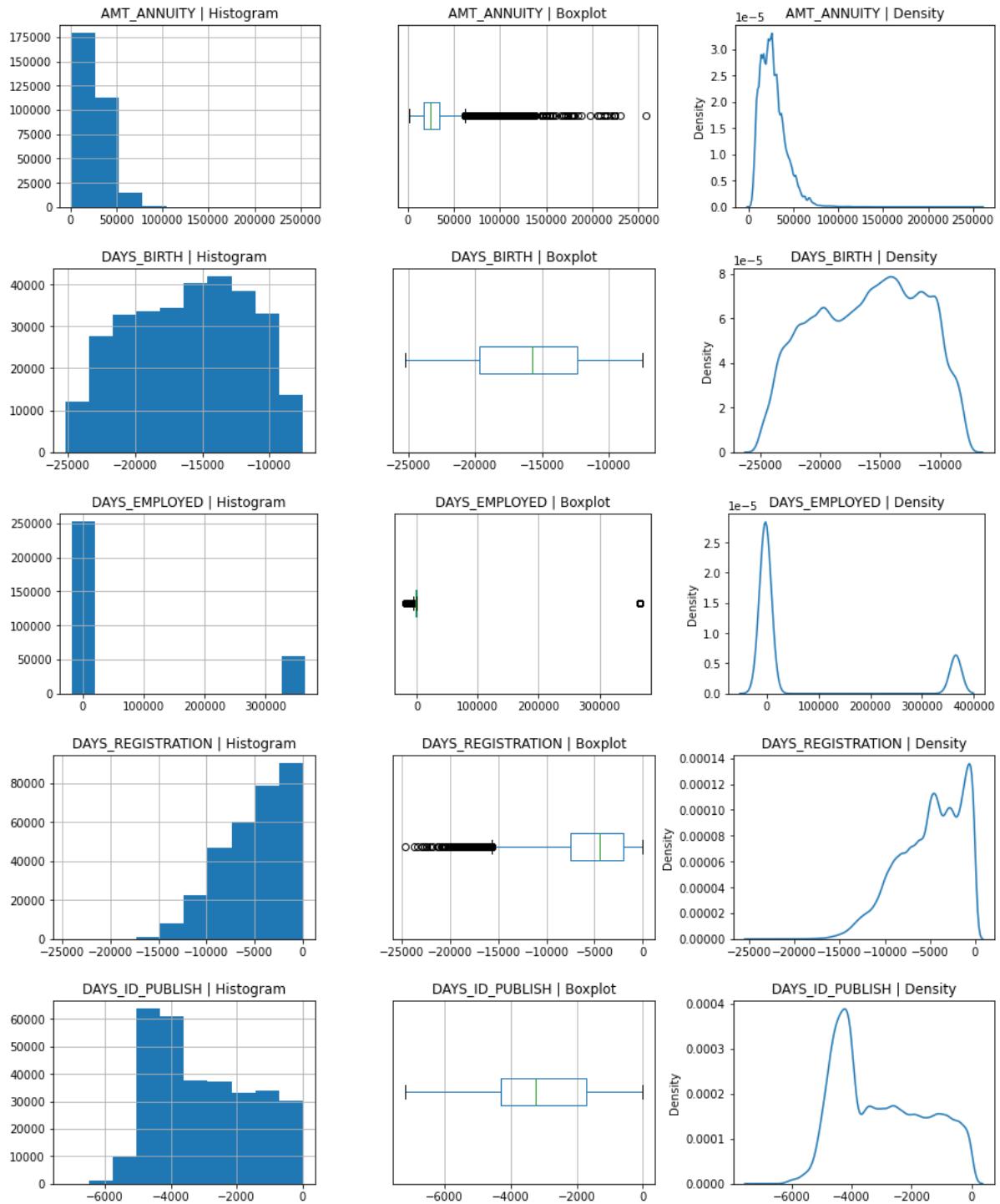
```
In [6]:  
train = datasets["application_train"]
```

Feature Transformation Viz

Visualization of columns within application train that were log transformed due to the scale of the data. DAYS_EMPLOYED has no central distribution and is distributed either to one extreme or another. For the other columns they are contained within a range of values eg. 'AMT_ANNUITY', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'DAYS_ID_PUBLISH'

The below are visualizations of the columns 'AMT_ANNUITY', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'DAYS_REGISTRATION', and 'DAYS_ID_PUBLISH' prior to log transformations.

```
In [16]:  
_,num_cols,_,_ = id_num_cat_feature(train, text = False)  
#datasets[ds_name].replace([np.inf, -np.inf], 0, inplace=True)  
  
cols_trans_to_log = ['AMT_ANNUITY', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'DAYS_ID_PUBLISH']  
  
num_plot(train, cols_trans_to_log, remove=['SK_ID_CURR', 'SK_ID_BUREAU', 'SK_ID_PREV'])
```



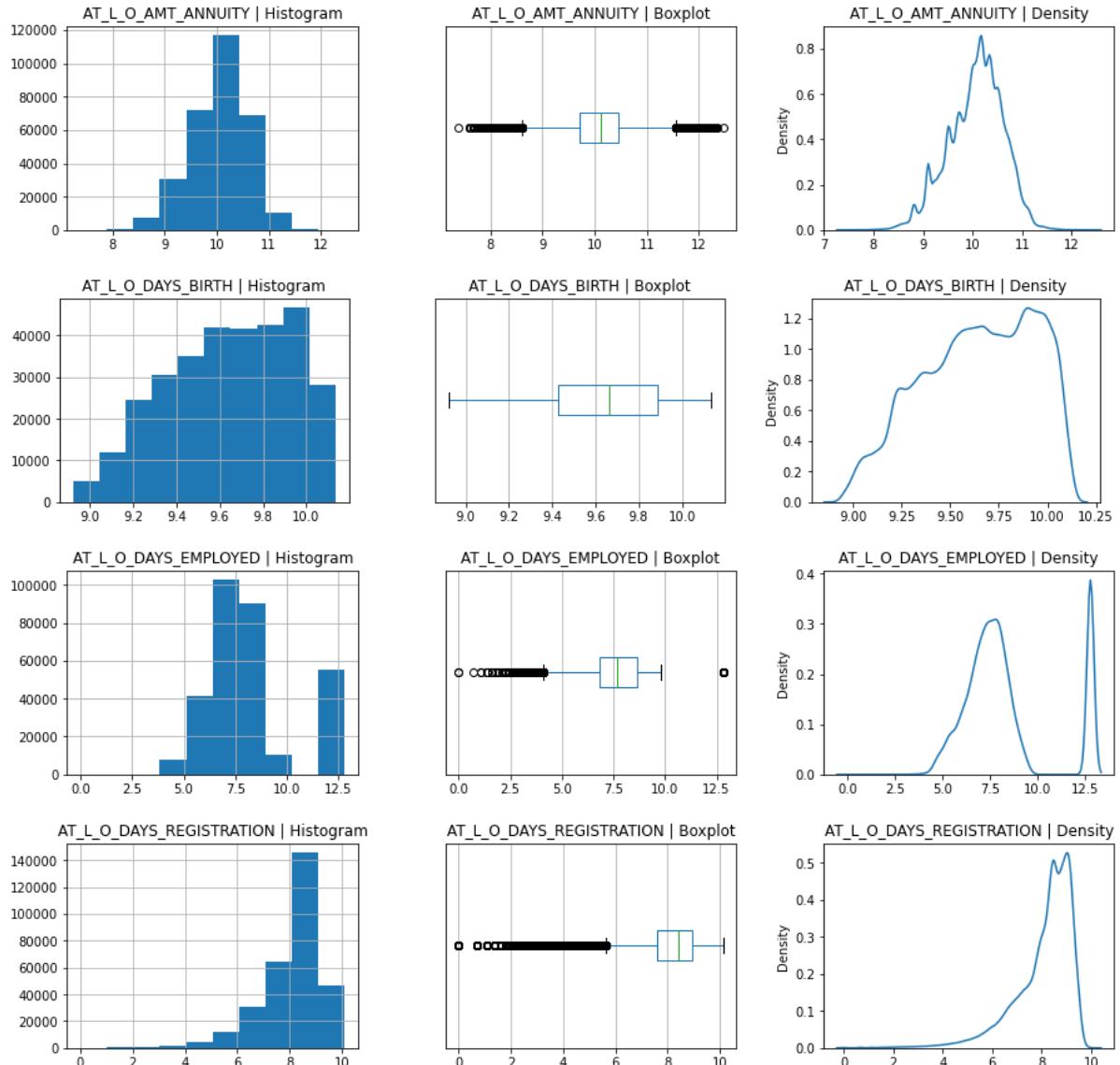
The below are columns 'AMT_ANNUITY', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'DAYS_ID_PUBLISH' log transformed and you can see for that the scale is smaller and AMT_ANNUITY and DAYS_EMPLOYED have become more centralized.

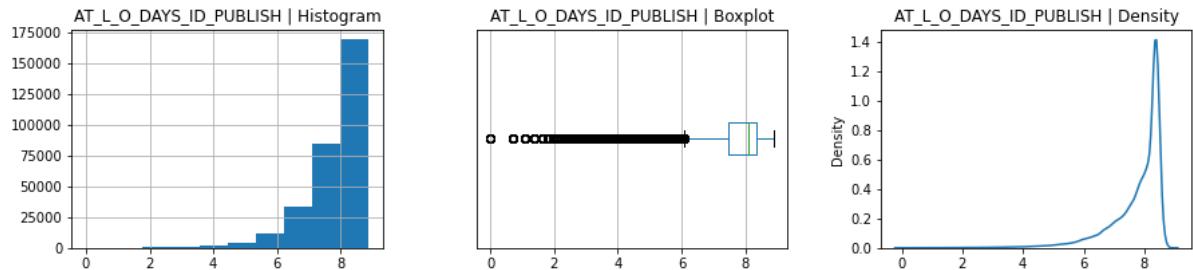
In [113]:

```
#Log Transformations
train_log_visual = train[['AMT_ANNUITY', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'DA'
for column in cols_trans_to_log:
    name = "AT_L_O_" + column
    train_visual[name] = np.log(train_visual[column].abs() + 1)

col_names_log = []
for column in cols_trans_to_log:
    name = "AT_L_O_" + column
    col_names_log.append(name)

num_plot(train_visual, col_names_log, remove=['SK_ID_CURR', 'SK_ID_BUREAU', '!
```





Highest Correlations on application train

```
In [23]: high_correlation(train, remove=['SK_ID_CURR', 'SK_ID_BUREAU'], corr_coef="pearson")
```

```
AMT_CREDIT
-----
AMT_GOODS_PRICE      0.986968
Name: AMT_CREDIT, dtype: float64

AMT_GOODS_PRICE
-----
AMT_CREDIT      0.986968
Name: AMT_GOODS_PRICE, dtype: float64

DAYS_EMPLOYED
-----
FLAG_EMP_PHONE    -0.999755
Name: DAYS_EMPLOYED, dtype: float64

FLAG_EMP_PHONE
-----
DAYS_EMPLOYED    -0.999755
Name: FLAG_EMP_PHONE, dtype: float64

REGION_RATING_CLIENT
-----
REGION_RATING_CLIENT_W_CITY      0.950842
Name: REGION_RATING_CLIENT, dtype: float64

REGION_RATING_CLIENT_W_CITY
-----
REGION_RATING_CLIENT      0.950842
Name: REGION_RATING_CLIENT_W_CITY, dtype: float64

APARTMENTS_AVG
-----
APARTMENTS_MEDI      0.995081
APARTMENTS_MODE      0.973259
Name: APARTMENTS_AVG, dtype: float64

BASEMENTAREA_AVG
-----
BASEMENTAREA_MEDI      0.994317
BASEMENTAREA_MODE      0.973496
Name: BASEMENTAREA_AVG, dtype: float64

YEARS_BEGINEXPLUATATION_AVG
-----
YEARS_BEGINEXPLUATATION_MEDI      0.993825
YEARS_BEGINEXPLUATATION_MODE      0.971893
Name: YEARS_BEGINEXPLUATATION_AVG, dtype: float64

YEARS_BUILD_AVG
-----
```

```
YEARS_BUILD_MEDI      0.998495
YEARS_BUILD_MODE      0.989444
Name: YEARS_BUILD_AVG, dtype: float64
```

```
COMMONAREA_AVG
-----
COMMONAREA_MEDI      0.995978
COMMONAREA_MODE      0.977147
Name: COMMONAREA_AVG, dtype: float64
```

```
ELEVATORS_AVG
-----
ELEVATORS_MEDI      0.996099
ELEVATORS_MODE      0.978837
Name: ELEVATORS_AVG, dtype: float64
```

```
ENTRANCES_AVG
-----
ENTRANCES_MEDI      0.996886
ENTRANCES_MODE      0.977743
Name: ENTRANCES_AVG, dtype: float64
```

```
FLOORSMAX_AVG
-----
FLOORSMAX_MEDI      0.997034
FLOORSMAX_MODE      0.985689
Name: FLOORSMAX_AVG, dtype: float64
```

```
FLOORSMIN_AVG
-----
FLOORSMIN_MEDI      0.997241
FLOORSMIN_MODE      0.985875
Name: FLOORSMIN_AVG, dtype: float64
```

```
LANDAREA_AVG
-----
LANDAREA_MEDI      0.991610
LANDAREA_MODE      0.973696
Name: LANDAREA_AVG, dtype: float64
```

```
LIVINGAPARTMENTS_AVG
-----
LIVINGAPARTMENTS_MEDI      0.993825
LIVINGAPARTMENTS_MODE      0.970117
Name: LIVINGAPARTMENTS_AVG, dtype: float64
```

```
LIVINGAREA_AVG
-----
LIVINGAREA_MEDI      0.995596
LIVINGAREA_MODE      0.972050
Name: LIVINGAREA_AVG, dtype: float64
```

```
NONLIVINGAPARTMENTS_AVG
-----
NONLIVINGAPARTMENTS_MEDI      0.990768
NONLIVINGAPARTMENTS_MODE      0.969370
Name: NONLIVINGAPARTMENTS_AVG, dtype: float64

NONLIVINGAREA_AVG
-----
NONLIVINGAREA_MEDI      0.990444
NONLIVINGAREA_MODE      0.966087
Name: NONLIVINGAREA_AVG, dtype: float64

APARTMENTS_MODE
-----
APARTMENTS_MEDI      0.977193
APARTMENTS_AVG       0.973259
Name: APARTMENTS_MODE, dtype: float64

BASEMENTAREA_MODE
-----
BASEMENTAREA_MEDI      0.977938
BASEMENTAREA_AVG       0.973496
Name: BASEMENTAREA_MODE, dtype: float64

YEARS_BEGINEXPLUATATION_MODE
-----
YEARS_BEGINEXPLUATATION_AVG      0.971893
YEARS_BEGINEXPLUATATION_MEDI      0.963539
Name: YEARS_BEGINEXPLUATATION_MODE, dtype: float64

YEARS_BUILD_MODE
-----
YEARS_BUILD_MEDI      0.989463
YEARS_BUILD_AVG       0.989444
Name: YEARS_BUILD_MODE, dtype: float64

COMMONAREA_MODE
-----
COMMONAREA_MEDI      0.979887
COMMONAREA_AVG       0.977147
Name: COMMONAREA_MODE, dtype: float64

ELEVATORS_MODE
-----
ELEVATORS_MEDI      0.982828
ELEVATORS_AVG       0.978837
Name: ELEVATORS_MODE, dtype: float64

ENTRANCES_MODE
```

```
-----  
ENTRANCES_MEDI      0.980677  
ENTRANCES_AVG       0.977743  
Name: ENTRANCES_MODE, dtype: float64
```

FLOORSMAX_MODE

```
-----  
FLOORSMAX_MEDI      0.988237  
FLOORSMAX_AVG        0.985689  
Name: FLOORSMAX_MODE, dtype: float64
```

FLOORSMIN_MODE

```
-----  
FLOORSMIN_MEDI      0.988406  
FLOORSMIN_AVG        0.985875  
Name: FLOORSMIN_MODE, dtype: float64
```

LANDAREA_MODE

```
-----  
LANDAREA_MEDI      0.980835  
LANDAREA_AVG        0.973696  
Name: LANDAREA_MODE, dtype: float64
```

LIVINGAPARTMENTS_MODE

```
-----  
LIVINGAPARTMENTS_MEDI      0.975605  
LIVINGAPARTMENTS_AVG        0.970117  
Name: LIVINGAPARTMENTS_MODE, dtype: float64
```

LIVINGAREA_MODE

```
-----  
LIVINGAREA_MEDI      0.974743  
LIVINGAREA_AVG        0.972050  
Name: LIVINGAREA_MODE, dtype: float64
```

NONLIVINGAPARTMENTS_MODE

```
-----  
NONLIVINGAPARTMENTS_MEDI      0.978575  
NONLIVINGAPARTMENTS_AVG        0.969370  
Name: NONLIVINGAPARTMENTS_MODE, dtype: float64
```

NONLIVINGAREA_MODE

```
-----  
NONLIVINGAREA_MEDI      0.975839  
NONLIVINGAREA_AVG        0.966087  
Name: NONLIVINGAREA_MODE, dtype: float64
```

APARTMENTS_MEDI

```
-----  
APARTMENTS_AVG        0.995081  
APARTMENTS_MODE        0.977193
```

Name: APARTMENTS_MEDI, dtype: float64

BASEMENTAREA_MEDI

BASEMENTAREA_AVG 0.994317
BASEMENTAREA_MODE 0.977938
Name: BASEMENTAREA_MEDI, dtype: float64

YEARS_BEGINEXPLUATATION_MEDI

YEARS_BEGINEXPLUATATION_AVG 0.993825
YEARS_BEGINEXPLUATATION_MODE 0.963539
Name: YEARS_BEGINEXPLUATATION_MEDI, dtype: float64

YEARS_BUILD_MEDI

YEARS_BUILD_AVG 0.998495
YEARS_BUILD_MODE 0.989463
Name: YEARS_BUILD_MEDI, dtype: float64

COMMONAREA_MEDI

COMMONAREA_AVG 0.995978
COMMONAREA_MODE 0.979887
Name: COMMONAREA_MEDI, dtype: float64

ELEVATORS_MEDI

ELEVATORS_AVG 0.996099
ELEVATORS_MODE 0.982828
Name: ELEVATORS_MEDI, dtype: float64

ENTRANCES_MEDI

ENTRANCES_AVG 0.996886
ENTRANCES_MODE 0.980677
Name: ENTRANCES_MEDI, dtype: float64

FLOORSMAX_MEDI

FLOORSMAX_AVG 0.997034
FLOORSMAX_MODE 0.988237
Name: FLOORSMAX_MEDI, dtype: float64

FLOORSMIN_MEDI

FLOORSMIN_AVG 0.997241
FLOORSMIN_MODE 0.988406
Name: FLOORSMIN_MEDI, dtype: float64

```
LANDAREA_MEDI
-----
LANDAREA_AVG      0.991610
LANDAREA_MODE     0.980835
Name: LANDAREA_MEDI, dtype: float64
```

```
LIVINGAPARTMENTS_MEDI
-----
LIVINGAPARTMENTS_AVG      0.993825
LIVINGAPARTMENTS_MODE     0.975605
Name: LIVINGAPARTMENTS_MEDI, dtype: float64
```

```
LIVINGAREA_MEDI
-----
LIVINGAREA_AVG      0.995596
LIVINGAREA_MODE     0.974743
Name: LIVINGAREA_MEDI, dtype: float64
```

```
NONLIVINGAPARTMENTS_MEDI
-----
NONLIVINGAPARTMENTS_AVG      0.990768
NONLIVINGAPARTMENTS_MODE     0.978575
Name: NONLIVINGAPARTMENTS_MEDI, dtype: float64
```

```
NONLIVINGAREA_MEDI
-----
NONLIVINGAREA_AVG      0.990444
NONLIVINGAREA_MODE     0.975839
Name: NONLIVINGAREA_MEDI, dtype: float64
```

```
OBS_30_CNT_SOCIAL_CIRCLE
-----
OBS_60_CNT_SOCIAL_CIRCLE      0.99849
Name: OBS_30_CNT_SOCIAL_CIRCLE, dtype: float64
```

```
OBS_60_CNT_SOCIAL_CIRCLE
-----
OBS_30_CNT_SOCIAL_CIRCLE      0.99849
Name: OBS_60_CNT_SOCIAL_CIRCLE, dtype: float64
```

Below are correlations in application train. The first is .99 and above while the second is .95 and above. The 99% correlations will be dropped as their correlation to each other is at 99%. With more time a better threshold could have been determined.

```
In [61]:
```

```
# columns to drop .99
train_cols_99 = ["FLAG_EMP_PHONE", "APARTMENTS_MEDI", "BASEMENTAREA_MEDI", "I
# columns to drop .95
train_cols_95 = ["FLAG_EMP_PHONE", "APARTMENTS_MEDI", "BASEMENTAREA_MEDI", "I
```

Categorical Analysis on Application Train

```
In [119...:
```

```
# Categorical Variables & Target
def cat_analyzer(dataframe, variable, target = None):
    print(variable)
    if target == None:
        print(pd.DataFrame({
            "COUNT": dataframe[variable].value_counts(),
            "RATIO": dataframe[variable].value_counts() / len(dataframe)}),
    else:
        temp = dataframe[dataframe[target].isnull() == False]
        print(pd.DataFrame({
            "COUNT":dataframe[variable].value_counts(),
            "RATIO":dataframe[variable].value_counts() / len(dataframe),
            "TARGET_COUNT":dataframe.groupby(variable)[target].count(),
            "TARGET_MEAN":temp.groupby(variable)[target].mean(),
            "TARGET_MEDIAN":temp.groupby(variable)[target].median(),
            "TARGET_STD":temp.groupby(variable)[target].std()}), end="\n\n")
```

Column analysis on WEEKDAY_APPR_PROCESS_START. All weekdays, Monday, Tuesday, Wednesday, Thursday, and Friday will be converted to weekday and all weekend days Saturday and Sunday to weekends

```
In [30]:
```

```
cat_analyzer(train, "WEEKDAY_APPR_PROCESS_START")
```

	WEEKDAY_APPR_PROCESS_START	COUNT	RATIO
TUESDAY	53901	0.175282	
WEDNESDAY	51934	0.168885	
MONDAY	50714	0.164918	
THURSDAY	50591	0.164518	
FRIDAY	50338	0.163695	
SATURDAY	33852	0.110084	
SUNDAY	16181	0.052619	

Column analysis on NAME_FAMILY_STATUS. If the person is single or not married then will be single, if the person is married or in a civil marriage then multiple. If the person is a widow or separated then Broken.

```
In [28]:
```

```
cat_analyzer(train, "NAME_FAMILY_STATUS")
```

NAME_FAMILY_STATUS	COUNT	RATIO
Married	196432	0.638780
Single / not married	45444	0.147780
Civil marriage	29775	0.096826
Separated	19770	0.064290
Widow	16088	0.052317
Unknown	2	0.000007

Column analysis for NAME_EDUCATION_TYPE. If the person has had higher education or academic degree then complete, if secondary or incomplete higher then partial, otherwise lower.

```
In [32]: cat_analyzer(train, "NAME_EDUCATION_TYPE")
```

NAME_EDUCATION_TYPE	COUNT	RATIO
Secondary / secondary special	218391	0.710189
Higher education	74863	0.243448
Incomplete higher	10277	0.033420
Lower secondary	3816	0.012409
Academic degree	164	0.000533

Column analysis for OCCUPATION_TYPE. Based on the skill level of the occupation it was consider to be a part of one of four groups Lowest Skill, Low Skill, Medium Skill, and Highest Skill.

```
In [120...]: cat_analyzer(train, "OCCUPATION_TYPE")
```

OCCUPATION_TYPE	COUNT	RATIO
Laborers	55186	0.179460
Sales staff	32102	0.104393
Core staff	27570	0.089655
Managers	21371	0.069497
Drivers	18603	0.060495
High skill tech staff	11380	0.037007
Accountants	9813	0.031911
Medicine staff	8537	0.027762
Security staff	6721	0.021856
Cooking staff	5946	0.019336
Cleaning staff	4653	0.015131
Private service staff	2652	0.008624
Low-skill Laborers	2093	0.006806
Waiters/barmen staff	1348	0.004384
Secretaries	1305	0.004244
Realty agents	751	0.002442
HR staff	563	0.001831
IT staff	526	0.001711

Feature Engineering Function for Application Train

Other feature engineering was also done using NAME_EDUCATION_TYPE, EXT_SOURCE, DAYS_EMPLOYED, DAYS_BIRTH, AMT_INCOME_TOTAL, AMT_CREDIT, INCOME_PER_PERSON, AMT_INCOME_TOTAL, CNT_FAM_MEMBERS, ANNUITY_INCOME_PERC, AMT_ANNUITY, AMT_INCOME_TOTAL, AMT_ANNUITY, AMT_CREDIT, AMT_GOODS_PRICE, DAYS_BIRTH, and AMT_GOODS_PRICE.

In [85]:

```

def feature_engineer_train(df, cols, cols_trans_to_log):
    df = df.drop(cols, axis = 1)
    df.replace([np.inf, -np.inf], 0, inplace=True)
    # WEEKDAY_APPR_PROCESS_START column feature engineering
    df["WEEKDAY_APPR_PROCESS_START"] = np.where(df.WEEKDAY_APPR_PROCESS_STAI
        ["MONDAY", "TUESDAY", "WEDNESDAY", "THRUSDAY", "FRIDAY"]), "weekday"

    df["WEEKDAY_APPR_PROCESS_START"] = np.where(df.WEEKDAY_APPR_PROCESS_STAI
        ["SATURDAY", "SUNDAY"]), "weekend", df.ORGANIZATION_TYPE)

    # NAME_FAMILY_STATUS column feature engineering
    df["NAME_FAMILY_STATUS"] = np.where(df.NAME_FAMILY_STATUS.isin(
        ["Married", "Civil marriage"]), "Mutiple", df.NAME_FAMILY_STATUS)

    df["NAME_FAMILY_STATUS"] = np.where(df.NAME_FAMILY_STATUS.isin(
        ["Single / not married"]), "Single", df.NAME_FAMILY_STATUS)

    df["NAME_FAMILY_STATUS"] = np.where(df.NAME_FAMILY_STATUS.isin(
        ["Separated", "Widow"]), "Broken", df.NAME_FAMILY_STATUS)

    # NAME_EDUCATION_TYPE column feature engineering
    df["NAME_EDUCATION_TYPE"] = np.where(df.NAME_EDUCATION_TYPE.isin(
        ["Higher education", "Academic degree"]), "Complete", df.NAME_EDUCATIO

    df["NAME_EDUCATION_TYPE"] = np.where(df.NAME_EDUCATION_TYPE.isin(
        ["Secondary / secondary special", "Incomplete higher"]), "Partial", d

    df["NAME_EDUCATION_TYPE"] = np.where(df.NAME_EDUCATION_TYPE.isin(
        ["Lower secondary"]), "Lower", df.NAME_EDUCATION_TYPE)

    # ORGANIZATION_TYPE column feature engineering
    df["ORGANIZATION_TYPE"] = np.where(df.ORGANIZATION_TYPE.str.contains("Bu
    df["ORGANIZATION_TYPE"] = np.where(df.ORGANIZATION_TYPE.str.contains("In
    df["ORGANIZATION_TYPE"] = np.where(df.ORGANIZATION_TYPE.str.contains("Tr
    df["ORGANIZATION_TYPE"] = np.where(df.ORGANIZATION_TYPE.str.contains("Te
    df["ORGANIZATION_TYPE"] = np.where(df.ORGANIZATION_TYPE.isin(["School",
    df["ORGANIZATION_TYPE"] = np.where(df.ORGANIZATION_TYPE.isin(["Emergenc
    df["ORGANIZATION_TYPE"] = np.where(df.ORGANIZATION_TYPE.isin(["Bank", "
    df["ORGANIZATION_TYPE"] = np.where(df.ORGANIZATION_TYPE.isin(["Realtor"
    df["ORGANIZATION_TYPE"] = np.where(df.ORGANIZATION_TYPE.isin(["Hotel",
    df["ORGANIZATION_TYPE"] = np.where(df.ORGANIZATION_TYPE.isin(["Cleaning

    # OCCUPATION_TYPE column feature engineering
    df["OCCUPATION_TYPE"] = np.where(df.OCCUPATION_TYPE.isin(["Low-skill La
    df["OCCUPATION_TYPE"] = np.where(df.OCCUPATION_TYPE.isin(["Laborers", "Se
    df["OCCUPATION_TYPE"] = np.where(df.OCCUPATION_TYPE.isin(["HR staff", "M
    df["OCCUPATION_TYPE"] = np.where(df.OCCUPATION_TYPE.isin(["Managers", "M

## EXT_SOURCE_MEAN FROM OTHER ASSOCIATIONS
df["EXT_MEAN"] = df[['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3']].mean()
df['EXT_SOURCES_SUM'] = df['EXT_SOURCE_1'] + df['EXT_SOURCE_2'] + df['EXT_SOURCE_3']

# NaN values for 365243 days of employment which is the max number
df['DAYS_EMPLOYED'].replace(365243, np.nan, inplace=True)

# Some simple new features (percentages)
df['AT_N_R_DAYS_EMPLOYED_PERC'] = df['DAYS_EMPLOYED'] / df['DAYS_BIRTH']
df['AT_N_R_INCOME_CREDIT_PERC'] = df['AMT_INCOME_TOTAL'] / df['AMT_CREDI

```

```

df['AT_N_R_INCOME_PER_PERSON'] = df['AMT_INCOME_TOTAL'] / df['CNT_FAM_MEMBERS']
df['AT_N_R_ANNUITY_INCOME_PERC'] = df['AMT_ANNUITY'] / df['AMT_INCOME_TOTAL']
df['AT_N_R_PAYMENT_RATE'] = df['AMT_ANNUITY'] / df['AMT_CREDIT']

#Others
df["AT_N_R_GOODS_TO_CREDIT"] = df["AMT_GOODS_PRICE"] / df["AMT_CREDIT"]
df['AT_N_R_ANNUITY_RATIO_INCOME'] = df['AMT_ANNUITY'] / df['AMT_INCOME_TOTAL']
df["AT_N_R_NEW_C_GP"] = (df["AMT_GOODS_PRICE"] - df["AMT_CREDIT"]) / df["AMT_GOODS_PRICE"]
df["AT_N_R_APP_AGE_YEARS"] = round(df["AMT_GOODS_PRICE"] * -1 / 365)
df['AT_N_R_INCOME_PER_PERSON_IN_FAMILY'] = df['AMT_INCOME_TOTAL'] / df['CNT_FAM_MEMBERS']
df['AT_N_R_PAYMENT_RATE'] = df['AMT_ANNUITY'] / df['AMT_CREDIT']
df['AT_N_R_LOAN_GOOD_RATIO'] = df['AMT_CREDIT'] / df['AMT_GOODS_PRICE']

#Log Transformations
for column in cols_trans_to_log:
    name = "AT_L_O_" + column
    df[name] = np.log(df[column].abs() + 1)

return df

```

Feature Engineering Application Train

All but one column with correlation of 95% with other columns were dropped.

```

In [181...]: train_eng = feature_engineer_train(train, train_cols_95, cols_trans_to_log)

In [182...]: train_eng.to_csv(os.getcwd() + DATA_DIR + 'application_train_eng.csv')

```

Feature Engineering Test Set

```

In [185...]: test = datasets["application_test"]

In [186...]: test_eng = feature_engineer_train(test, train_cols_95, cols_trans_to_log)

In [187...]: test_eng.to_csv(os.getcwd() + DATA_DIR + 'application_test_eng.csv')

```

Visual EDA

EDA Visualization functions

Following are the functions created to do visualization. These are rudimentary level plots which provide insights into a dataframe. The input to these functions will either be a dataframe or a combination of dataframe and type of attribute (numerical, categorical) depending upon the type of function.

- **Attr_Type:** This function plots a bar graph which shows the count of type of attribute, i.e. Number, Categorical, Date etc. We are using similar logic in summary statistics where depending upon the type of variable, we derive the numerical and categorical features to further do the processing.
- **Unique Values:** This function plots the bar chart for the unique value for each attribute. This process will give us some insights about the number of binary , ordinal and continuous (more than 10 unique values) features in the dataset. This info can also be read from info and summary python functions. Another thing to notice here is that y-axis is on log scale. This is because on such a large data set, few elements have more unique values(like CURR ID, SK ID PREV) which makes distribution highly skewed. By taking log, we will have fair idea about other features with lesser unique values.
- **Percent Missing:** This bar plot will show to percentage of nulls in the data. This plot will be helpful for us to see how much nulls a attribute has. In phase I, we created pipeline with attributes with more than 50% nulls. This information will also help while finding the correlation and cross-verify if some values are less or highly correlated. 1 thing to note here is that this plot also has y-axis log scaled for the same reason as provided above.
- **Categorical_count:** This function will take a dataframe and categorical features as input and will produce a bar plot for each category. This will enable us to understand if certain set of values are more correlated to the target. For example, we saw that less educated people are more likely to default on their loan, as per the data provided. Now that we have seen this info as a plot, we can analyze a little more. This will also help us group certain values which are negligible, For example, if 2 categories are related to 90 % of the data, we can group certain low occurring values to make them as 1 category. This can also be seen by using the F score to see how statistically important it can be.
- **Dendo:** This unique plot will groups together columns that have strong correlations in nullity. If a number of columns are grouped together at level zero, then the presence of nulls in one of those columns is directly related to the presence or absence of nulls in the others columns. The more separated the columns in the tree, the less likely the null values can be correlated between the columns.
- **Numerical_features:** This function will take only the numerical attributes from a dataframe and produce a dot plot. Each dot in this plot is a sample in our dataset and each subplot represents a different feature. The x axis shows the feature value, while the y axis shows the count of samples.

each subplot represents a different feature. The y-axis shows the feature value, while the x-axis is the sample index. These kind of plots provide a lot of ideas for data cleaning and EDA.

- **Num Hist:** This function takes dataframe and numerical attributes. This will plot histogram for all the features supplied. A very useful visualization to see the data distribution in 1 view. We can take note of attributes which needs transformation, as well as outliers. For a better input data, we should remove the outliers so that it does not influence the target outcome.
- **all_missing_values_plot:** This function will plot the missing in all the features of the dataframe supplied. For attributes which have no nulls will show as 1, if an attribute has half of the records null, then it will show 0.5.

In [77]:

```
def attr_type(df):  
    plt.figure(figsize=(20,20))  
    pd.value_counts(df.dtypes).sort_values().plot(kind="bar", figsize=(15, 8),  
                                                   title="Type of features- Numerical",  
                                                   ylabel="Number of features");  
    plt.show()  
  
def unique_values(df):  
    plt.figure(figsize=(20,20))  
    unique_values = df.select_dtypes(include="number").nunique().sort_values()  
    unique_values.plot.bar(logy=True, figsize=(15, 4), title="Unique values per feature");  
    plt.show()  
  
def percent_missing(df):  
    plt.figure(figsize=(20,20))  
    df.isna().mean().sort_values().plot(kind="bar", figsize=(15, 8), logy=True,  
                                         title="Percentage of missing values per feature",  
                                         ylabel="Ratio of missing values per feature");  
    plt.show()  
  
def categorical_count(df,categorical):  
    plt.figure(figsize=(20,20))  
    for i, col in enumerate(feat_cat):  
        ax = plt.subplot(5, 4, i+1)  
        sns.countplot(data=df[categorical], y=col, ax=ax,)  
    plt.suptitle('Category counts for all categorical variables')  
    plt.tight_layout()  
    plt.show()  
  
def dendo(df):  
    plt.figure(figsize=(20,20))  
    msno.dendrogram(df)  
    plt.show()  
  
def numerical_features(df,num_cat):  
    plt.figure(figsize=(20,20))  
    df[num_cat].plot(lw=0, marker=".", subplots=True, layout=(-1, 4),  
                     figsize=(15, 12), markersize=1);  
  
    plt.show()  
  
def num_hist(df):  
    plt.figure(figsize=(20, 20))
```

```

plt.figure(figsize=(20,20))
df[np.isfinite(df)].hist(bins=25, figsize=(15, 25), layout=(-1, 5), edgecolor='black')
plt.tight_layout()
plt.show()

[def continous_features(df,feat_num):
    plt.figure(figsize=(20,20))
    cols_continuous = df.select_dtypes(include="number").nunique() >= 25
    for i,ds_name in enumerate(datasets.keys()):
        #n = True
        #if n:
            if ds_name.lower() not in ("application_train",
                                         "application_test", "bureau_balance"):
                print("Table under consideration:",ds_name.upper())
                print("-----")
                id_cols, feat_num, feat_cat, features = id_num_cat_feature(dataset)
                only_num_cat = list(set(feat_num)-set(['SK_ID_CURR','SK_ID_PREV','SK_ID_BUREAU']))
                print("-----")
                print(ds_name.upper(),":-----Type of Features-----")
                attr_type(datasets[ds_name])
                print("-----")
                print(ds_name.upper(),":-----UNIQUE VALUES-----")
                unique_values(datasets[ds_name])
                print("-----")
                print(ds_name.upper(),":-----MISSING PERCENTAGE-----")
                percent_missing(datasets[ds_name])
                print("-----")
                print(ds_name.upper(),":-----CATEGORICAL COUNT-----")
                categorical_count(datasets[ds_name],feat_cat)
                print("-----")
                print(ds_name.upper(),":-----NUM FEATURES-DOT PRODUCT-----")
                numerical_features(datasets[ds_name],only_num_cat)
                print("-----")
                print(ds_name.upper(),":-----NUM FEATURES - HISTOGRAM-----")
                num_hist(datasets[ds_name][only_num_cat])
                print("-----")
                print(ds_name.upper(),":-----Continous Features-----")
                #continous_features(df_x,only_num_cat)
                print("-----")
                print(ds_name.upper(),":-----All Missing Features-----")
                all_missing_values_plot(datasets[ds_name])
                print("-----")
                print(ds_name.upper(),":-----DendoGram for Null-----")
                dendo(datasets[ds_name])
                print("-----")

```

Correlation Plots Maps

A correlation matrix is a table showing correlation coefficients between variables. Each cell in the table shows the correlation between two variables. This correlation is a value put on the relationship between two attributes. A correlation matrix is used to summarize data, then with that information do more advanced analysis.

```
In [80]: def corr_plot(data, remove=["Id"], corr_coef = "pearson", figsize=(20, 20))
    if len(remove) > 0:
        num_cols2 = [x for x in data.columns if (x not in remove)]

    sns.set(font_scale=1.1)
    c = data[num_cols2].corr(method = corr_coef)
    mask = np.triu(c.corr(method = corr_coef))
    plt.figure(figsize=figsize)
    sns.heatmap(c,
                annot=True,
                fmt='.1f',
                cmap='coolwarm',
                square=True,
                mask=mask,
                linewidths=1,
                cbar=False)
    plt.show()
```

```
In [ ]: for i,ds_name in enumerate(datasets.keys()):
    if(ds_name != "bureau_balance"):
        print("-----")
        print("Table under consideration FOR CORRELATION PLOT:",ds_name.upper())
        corr_plot(datasets[ds_name], remove=['SK_ID_CURR','SK_ID_BUREAU'], cbar=True)
        print("-----")
        print("-----")
```

As we have seen in Application train and application test dataframes, most of the **FLAG_DOCUMENT_XX** are null, we will remove these columns at later point of time

Numerical Plots and Visualizations

These visualizations consist of the paired plots for each column in the each table.

The first is a histogram, a diagram consisting of rectangles whose area is proportional to the frequency of a variable and whose width is equal to the class interval, which helps to visualize distribution.

The second is a boxplot, which is a standardized way of displaying the distribution of data based on a five number summary (“minimum”, first quartile Q1, median, third quartile Q3 and “maximum”). It can tell you about your outliers and what their values are.

<https://builtin.com/data-science/boxplot#:~:text=What%20is%20a%20Boxplot%3F, and%20what%20their%20values%20are.>

The third is a Kernel Density Plot which also helps to visualize distribution over a time period or some continuous value.

```
In [ ]: def grab_col_names(dataframe, cat_th=10, car_th=20, show_date=False):
    date_cols = [col for col in dataframe.columns if dataframe[col].dtypes == "datetime64[ns]"]
    cat_cols = dataframe.select_dtypes(["object", "category"]).columns.tolist()

    num_but_cat = [col for col in dataframe.select_dtypes(["float", "integer"]) if len(dataframe[col].unique()) < cat_th + 1]
    cat_but_car = [col for col in dataframe.select_dtypes(["object", "category"]) if len(dataframe[col].unique()) > car_th]

    cat_cols = cat_cols + num_but_cat
    cat_cols = [col for col in cat_cols if col not in cat_but_car]

    num_cols = dataframe.select_dtypes(["float", "integer"]).columns
    num_cols = [col for col in num_cols if col not in num_but_cat]

    if show_date == True:
        return date_cols, cat_cols, cat_but_car, num_cols, num_but_cat
    else:
        return cat_cols, cat_but_car, num_cols, num_but_cat
```

```
In [104...]: def num_plot(data, num_cols, remove=["Id"], hist_bins=10, figsize=(20, 4)):

    if len(remove) > 0:
        num_cols2 = [x for x in num_cols if (x not in remove)]

    for i in num_cols2:
        fig, axes = plt.subplots(1, 3, figsize=figsize)
        data.hist(str(i), bins=hist_bins, ax=axes[0])
        data.boxplot(str(i), ax=axes[1], vert=False);
        try:
            sns.kdeplot(np.array(data[str(i)]))
        except:
            ValueError

        axes[1].set_yticklabels([])
        axes[1].set_yticks([])
        axes[0].set_title(i + " | Histogram")
        axes[1].set_title(i + " | Boxplot")
        axes[2].set_title(i + " | Density")
    plt.show()
```

Determine more than 50% null

Determine attributes which have more than 50% NULLS. Once done, these will be used as part of feature engineering.

```
In [ ]: for i,ds_name in enumerate(datasets.keys()):
    if ds_name.lower() not in ("application_train",
                               "application_test", "bureau_balance"):
        print("-")
        #_,_, num_cols, _ = grab_col_names(datasets[ds_name], car_th=10)
        _,num_cols,_,_ = id_num_cat_feature(datasets[ds_name], text = False)
        datasets[ds_name].replace([np.inf, -np.inf], 0, inplace=True)
        print("Table under consideration FOR NUMERICAL PLOTS:",ds_name.upper())
        num_plot(datasets[ds_name], num_cols, remove=['SK_ID_CURR','SK_ID_BI'])
        print("-")
        print("-")
```

Correlation with the target column

```
In [97]: def correlation_against_target(df):
    df_joined = {}
    df_joined = datasets['application_train'][["SK_ID_CURR", "TARGET"]].merge(df,
    cols = df.columns
    keys = list(df.columns)
    keys.append("TARGET")
    return df_joined[keys].corr()['TARGET'].sort_values(ascending=False)
```

Correlations against the target on the Original Tables

```
In [98]: original_corr = {}
#barplot
for i,ds_name in enumerate(datasets.keys()):
    if(ds_name.upper() != "APPLICATION_TRAIN"):
        if (ds_name.upper() != "APPLICATION_TEST"):
            if (ds_name.upper() != "BUREAU_BALANCE"):
                if (ds_name.upper() == "BUREAU"):
                    if 'AMT_ANNUITY' in datasets["bureau"].columns:
                        datasets["bureau"] = datasets["bureau"].drop(["AMT_ANNUITY"])
    print("-")
    print("Correlation for Original Table:", ds_name.upper())
    print("-")
    ds = correlation_against_target(datasets[ds_name])
    print("-")
    original_corr[ds_name] = ds
    print(ds)
    print("-")
    print("-")
```

Correlation for Original Table: BUREAU

TARGET	1.000000
DAYS_CREDIT	0.061556
DAYS_CREDIT_UPDATE	0.041076
DAYS_ENDDATE_FACT	0.039057
DAYS_CREDIT_ENDDATE	0.026497
AMT_CREDIT_SUM_OVERDUE	0.006253
CREDIT_DAY_OVERDUE	0.002652
AMT_CREDIT_SUM_DEBT	0.002539
AMT_CREDIT_MAX_OVERDUE	0.001587
CNT_CREDIT_PROLONG	0.001523
SK_ID_CURR	-0.003024
AMT_CREDIT_SUM_LIMIT	-0.005990
SK_ID_BUREAU	-0.009018
AMT_CREDIT_SUM	-0.010606

Name: TARGET, dtype: float64

Correlation for Original Table: CREDIT_CARD_BALANCE

TARGET	1.000000
AMT_BALANCE	0.050098
AMT_TOTAL_RECEIVABLE	0.049839
AMT_RECVABLE	0.049803
AMT_RECEIVABLE_PRINCIPAL	0.049692
AMT_INST_MIN_REGULARITY	0.039798
CNT_DRAWINGS_ATM_CURRENT	0.038437
CNT_DRAWINGS_CURRENT	0.037793
CNT_DRAWINGS_POS_CURRENT	0.029536
AMT_DRAWINGS_ATM_CURRENT	0.024700
AMT_DRAWINGS_CURRENT	0.022378
AMT_CREDIT_LIMIT_ACTUAL	0.013823
AMT_PAYMENT_CURRENT	0.012929
AMT_PAYMENT_TOTAL_CURRENT	0.012302
SK_DPD_DEF	0.010538
AMT_DRAWINGS_POS_CURRENT	0.005084
AMT_DRAWINGS_OTHER_CURRENT	0.003843
CNT_DRAWINGS_OTHER_CURRENT	0.003044
SK_ID_PREV	0.002571
SK_DPD	0.001684
SK_ID_CURR	-0.004617
CNT_INSTALMENT_MATURE_CUM	-0.023684
MONTHS_BALANCE	-0.035695

Name: TARGET, dtype: float64

Correlation for Original Table: INSTALLMENTS_PAYMENTS

TARGET	1.000000
SK_ID_PREV	-0.000212
AMT_INSTALMENT	-0.001498
SK_ID_CURR	-0.002540

```
AMT_PAYMENT           -0.003623
IP_DIFF_PAYMNT_INSTLMNT -0.006376
NUM_INSTALMENT_VERSION -0.009896
NUM_INSTALMENT_NUMBER   -0.016190
DAYS_INSTALMENT        -0.034974
DAYS_ENTRY_PAYMENT     -0.035122
Name: TARGET, dtype: float64
```

```
Correlation for Original Table: PREVIOUS_APPLICATION
```

```
TARGET                1.000000
CNT_PAYMENT            0.030480
PREV_CREDT_ANNUITY_RATIO 0.029673
RATE_INTEREST_PRIVILEGED 0.028640
SK_ID_PREV              0.002009
NFLAG_INSURED_ON_APPROVAL 0.000653
AMT_GOODS_PRICE         0.000254
SK_ID_CURR              -0.001246
RATE_INTEREST_PRIMARY    -0.001470
AMT_CREDIT               -0.002350
SELLERPLACE_AREA         -0.002539
NFLAG_LAST_APPL_IN_DAY   -0.002887
AMT_APPLICATION          -0.005583
PREV_APCTN_CRDT_DIFF    -0.012277
AMT_ANNUITY              -0.014922
AMT_DOWN_PAYMENT          -0.016918
DAYS_LAST_DUE_1ST_VERSION -0.017128
DAYS_TERMINATION          -0.022160
PREV_DWN_PYMNT_CRDT_RATIO -0.022275
DAYS_LAST_DUE              -0.022598
PREV_APCTN_CRDT_RATIO    -0.022678
RATE_DOWN_PAYMENT          -0.026111
HOUR_APPR_PROCESS_START   -0.027809
DAYS_FIRST_DUE             -0.029025
DAYS_DECISION              -0.039901
PREV_APRV_CNT              -0.049161
DAYS_FIRST_DRAWING         -0.095723
PREV_REJ_CNT                NaN
Name: TARGET, dtype: float64
```

```
Correlation for Original Table: POS_CASH_BALANCE
```

```
TARGET                1.000000
CNT_INSTALMENT_FUTURE   0.021972
CNT_INSTALMENT            0.018506
SK_DPD                  0.009866
SK_DPD_DEF                0.008594
POS_PERC_INSTL_PNDNG     0.008377
POS_DAYS_WHT_TOLRNC      0.008126
SK_ID_PREV              -0.000056
SK_ID_CURR              -0.002245
POS_CNT_INSTAL_PNDNG     -0.003865
MONTHS_BALANCE           -0.020147
```

```
Name: TARGET, dtype: float64
```

Correlations on the Transformed Tables done with a right merge against the table

In [99]:

```
transformed_corr = {}
for i,ds_name in enumerate(datasets_transformed.keys()):
    datasets_transformed[ds_name].reset_index()
    #if(ds_name.upper() not in ( "APPLICATION_TRAIN", "APPLICATION_TEST", "BU
    if(ds_name.upper() != "APPLICATION_TRAIN"):
        if (ds_name.upper() != "APPLICATION_TEST"):
            #if (ds_name.upper() != "POS_CASH_BALANCE"):
            if (ds_name.upper() != "BUREAU_BALANCE"):
                print("-----")
                print("Correlation for Transformed Table:", ds_name.upper())
                print("-----")
                ds = correlation_against_target(datasets_transformed[ds_name])
                print("-----")
                transformed_corr[ds_name] = ds
                print(ds)
                print("-----")
                print("-----")
```

Correlation for Transformed Table: BUREAU	
TARGET	1.000000
CREDIT_ACTIVE_Active_mean	0.072625
CREDIT_ACTIVE_Active_median	0.070590
STATUS_1_var_max	0.070020
STATUS_1_var_mean	0.069149
...	
CREDIT_TYPE_Interbank_credit_median	NaN
CREDIT_TYPE_Cash_loan_non_earmarked_median	NaN
CREDIT_ACTIVE_Bad_debt_median	NaN
CREDIT_TYPE_Mobile_operator_loan_median	NaN
CREDIT_TYPE_Loan_for_purchase_of_shares_margin_lending_median	NaN
Name: TARGET, Length: 297, dtype: float64	
...	
...	
...	
Correlation for Transformed Table: POS_CASH_BALANCE	
TARGET	1.000000
POS_PERC_INSTL_PNDNG_mean	0.027904
CNT_INSTALMENT_FUTURE_mean	0.027827
POS_PERC_INSTL_PNDNG_median	0.027011
CNT_INSTALMENT_FUTURE_median	0.026968
POS_PERC_INSTL_PNDNG_min	0.021933
CNT_INSTALMENT_min	0.019840
CNT_INSTALMENT_FUTURE_min	0.019010
CNT_INSTALMENT_mean	0.018066
CNT_INSTALMENT_FUTURE_var	0.017262
CNT_INSTALMENT_FUTURE_max	0.013324
CNT_INSTALMENT_max	0.013296
NAME_CONTRACT_STATUS_Returned_to_the_store_mean	0.012036
CNT_INSTALMENT_var	0.011916
NAME_CONTRACT_STATUS_Returned_to_the_store_var	0.011821
CNT_INSTALMENT_median	0.011622
NAME_CONTRACT_STATUS_Signed_mean	0.010492
NAME_CONTRACT_STATUS_Signed_var	0.009890
NAME_CONTRACT_STATUS_Demand_var	0.008903
NAME_CONTRACT_STATUS_Active_var	0.008285
NAME_CONTRACT_STATUS_Amortized_debt_var	0.007904
NAME_CONTRACT_STATUS_Demand_mean	0.007201
NAME_CONTRACT_STATUS_Amortized_debt_mean	0.006389
POS_CNT_INSTL_PNDNG_min	0.006016
NAME_CONTRACT_STATUS_Signed_median	0.005164
NAME_CONTRACT_STATUS_Returned_to_the_store_median	0.005135
NAME_CONTRACT_STATUS_Demand_median	0.003160
NAME_CONTRACT_STATUS_Completed_var	0.002736
NAME_CONTRACT_STATUS_Approved_median	0.001096
NAME_CONTRACT_STATUS_Approved_var	0.000944
NAME_CONTRACT_STATUS_Completed_median	0.000757
NAME_CONTRACT_STATUS_Approved_mean	0.000699
NAME_CONTRACT_STATUS_Completed_mean	0.000434
NAME_CONTRACT_STATUS_Canceled_var	-0.000248
NAME_CONTRACT_STATUS_Canceled_mean	-0.000248
NAME_CONTRACT_STATUS_XNA_mean	-0.000784
NAME_CONTRACT_STATUS_Amortized_debt_median	-0.000784

NAME_CONTRACT_STATUS_XNA_var	-0.000784
SK_ID_CURR	-0.002137
POS_PERC_INSTL_PNDNG_var	-0.003301
POS_PERC_INSTL_PNDNG_max	-0.005398
NAME_CONTRACT_STATUS_Active_median	-0.005739
CNT_INSTALMENT_FUTURE_sum	-0.005881
NAME_CONTRACT_STATUS_Active_mean	-0.008031
POS_CNT_INSTAL_PNDNG_median	-0.008950
POS_CNT_INSTAL_PNDNG_mean	-0.014498
CNT_INSTALMENT_sum	-0.014670
POS_CNT_INSTAL_PNDNG_var	-0.017377
POS_CNT_INSTAL_PNDNG_max	-0.024394
POS_CNT_INSTAL_PNDNG_sum	-0.027543
POS_PERC_INSTL_PNDNG_sum	-0.030009
SK_ID_PREV_count	-0.035632
CNT_INSTALMENT_count	-0.035802
POS_PERC_INSTL_PNDNG_count	-0.035805
POS_CNT_INSTAL_PNDNG_count	-0.035805
CNT_INSTALMENT_FUTURE_count	-0.035807
NAME_CONTRACT_STATUS_Canceled_median	Nan
NAME_CONTRACT_STATUS_XNA_median	Nan

Name: TARGET, dtype: float64

Correlation for Transformed Table: PREVIOUS_APPLICATION

TARGET	1.000000
NAME_CONTRACT_STATUS_Refused_mean	0.077671
NAME_CONTRACT_STATUS_Refused_var	0.075867
CODE_REJECT_REASON_XAP_var	0.071560
NAME_CONTRACT_STATUS_Refused_median	0.064930
...	
PREV_REJ_CNT_var	NaN
NAME_GOODS_CATEGORY_Animals_median	NaN
NAME_GOODS_CATEGORY_House_Construction_median	NaN
NAME_GOODS_CATEGORY_House_Construction_mean	NaN
NAME_GOODS_CATEGORY_House_Construction_var	NaN

Name: TARGET, Length: 601, dtype: float64

Correlation for Transformed Table: CREDIT_CARD_BALANCE

TARGET	1.000000
CNT_DRAWINGS_ATM_CURRENT_mean	0.107692
CNT_DRAWINGS_CURRENT_max	0.101389
AMT_BALANCE_mean	0.087177
AMT_TOTAL_RECEIVABLE_mean	0.086490
...	
SK_DPD_DEF_min	NaN
SK_DPD_min	NaN
NAME_CONTRACT_STATUS_Refused_median	NaN
NAME_CONTRACT_STATUS_Sent_proposal_median	NaN
NAME_CONTRACT_STATUS_Approved_median	NaN

Name: TARGET, Length: 164, dtype: float64

Correlation for Transformed Table: INSTALLMENTS_PAYMENTS

TARGET	1.000000
NUM_INSTALMENT_NUMBER_max	0.006304
DAYSTINSTALMENT_min	0.003231
AMT_INSTALMENT_max	0.002324
DAYSENTRY_PAYMENT_min	0.002298
AMT_PAYMENT_max	0.001554
NUM_INSTALMENT_NUMBER_var	0.001040
AMT_INSTALMENT_var	-0.002151
NUM_INSTALMENT_NUMBER_min	-0.002334
SK_ID_CURR	-0.002363
AMT_PAYMENT_var	-0.003841
NUM_INSTALMENT_NUMBER_mean	-0.009537
IP_DIFF_PAYMNT_INSTLMNT_var	-0.009684
NUM_INSTALMENT_VERSION_var	-0.011427
IP_DIFF_PAYMNT_INSTLMNT_max	-0.014018
IP_DIFF_PAYMNT_INSTLMNT_median	-0.014302
NUM_INSTALMENT_NUMBER_median	-0.014897
NUM_INSTALMENT_NUMBER_sum	-0.017441
AMT_INSTALMENT_mean	-0.018409
NUM_INSTALMENT_VERSION_max	-0.018611
IP_DIFF_PAYMNT_INSTLMNT_min	-0.019144
AMT_INSTALMENT_sum	-0.019811
AMT_INSTALMENT_min	-0.020257
NUM_INSTALMENT_VERSION_median	-0.020722
DAYSTINSTALMENT_count	-0.021096
NUM_INSTALMENT_NUMBER_count	-0.021096
NUM_INSTALMENT_VERSION_count	-0.021096
SK_ID_PREV_count	-0.021096
AMT_INSTALMENT_count	-0.021096
IP_DIFF_PAYMNT_INSTLMNT_count	-0.021217
AMT_PAYMENT_count	-0.021217
DAYSENTRY_PAYMENT_count	-0.021217
AMT_PAYMENT_mean	-0.023169
AMT_PAYMENT_sum	-0.024375
AMT_INSTALMENT_median	-0.024873
AMT_PAYMENT_min	-0.025724
NUM_INSTALMENT_VERSION_mean	-0.027323
IP_DIFF_PAYMNT_INSTLMNT_sum	-0.027326
IP_DIFF_PAYMNT_INSTLMNT_mean	-0.029339
AMT_PAYMENT_median	-0.029548
NUM_INSTALMENT_VERSION_sum	-0.030063
NUM_INSTALMENT_VERSION_min	-0.032039
DAYSTINSTALMENT_sum	-0.035064
DAYSENTRY_PAYMENT_sum	-0.035227
DAYSTINSTALMENT_median	-0.039252
DAYSENTRY_PAYMENT_median	-0.039654
DAYSTINSTALMENT_mean	-0.043509
DAYSENTRY_PAYMENT_mean	-0.043992
DAYSENTRY_PAYMENT_var	-0.052071
DAYSTINSTALMENT_var	-0.052273
DAYSTINSTALMENT_max	-0.058648
DAYSENTRY_PAYMENT_max	-0.058794

Name: TARGET, dtype: float64

Highest Correlations on Original Tables

The below gives you the highest correlated attributes from each table with a threshold of .7. This is correlation to other attributes in the same table.

In [100]:

```
# Get high correlated variables
def high_correlation(data, remove=['SK_ID_CURR', 'SK_ID_BUREAU'], corr_coef=.7):
    if len(remove) > 0:
        cols = [x for x in data.columns if (x not in remove)]
        c = data[cols].corr(method=corr_coef)
    else:
        c = data.corr(method=corr_coef)

    for i in c.columns:
        cr = c.loc[i].loc[(c.loc[i] >= corr_value) | (c.loc[i] <= -corr_value)]
        if len(cr) > 0:
            print(i)
            print("-----")
            print(cr.sort_values(ascending=False))
            print("\n")
```

In [101]:

```
datasets["application_train"]
```

Out[101]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FI
0	100002	1	Cash loans	M	N	
1	100003	0	Cash loans	F	N	
2	100004	0	Revolving loans	M	Y	
3	100006	0	Cash loans	F	N	
4	100007	0	Cash loans	M	N	
...
307506	456251	0	Cash loans	M	N	
307507	456252	0	Cash loans	F	N	
307508	456253	0	Cash loans	F	N	
307509	456254	1	Cash loans	F	N	
307510	456255	0	Cash loans	F	N	

307511 rows × 122 columns

Below we are determining the highest correlations for each table.

```
In [ ]: for i,ds_name in enumerate(datasets.keys()):
    print("-----")
    print("Table under consideration FOR HIGHEST CORRELATIONS:",ds_name.upper())
    high_correlation(datasets[ds_name], remove=['SK_ID_CURR','SK_ID_BUREAU',
                                                'FLAG_DOCUMENT_12','FLAG_DOCU
                                                'FLAG_DOCUMENT_15','FLAG_DOCU
                                                'FLAG_DOCUMENT_18','FLAG_DOCU
                                                'FLAG_DOCUMENT_21','PREV_REJ
    print("-----")
    print("-----")
```

Now, we find the attributes which are highly correlated, either positive or negative. Looks like EXT_SOURCE columns are highly correlated.

```
In [103...]: correlations = datasets["application_train"].corr()['TARGET'].sort_values()
print('Most Positive Correlations:\n', correlations.tail(10))
print('\nMost Negative Correlations:\n', correlations.head(10))
```

Most Positive Correlations:

FLAG_DOCUMENT_3	0.044346
REG_CITY_NOT_LIVE_CITY	0.044395
FLAG_EMP_PHONE	0.045982
REG_CITY_NOT_WORK_CITY	0.050994
DAYS_ID_PUBLISH	0.051457
DAYS_LAST_PHONE_CHANGE	0.055218
REGION_RATING_CLIENT	0.058899
REGION_RATING_CLIENT_W_CITY	0.060893
DAYS_BIRTH	0.078239
TARGET	1.000000

Name: TARGET, dtype: float64

Most Negative Correlations:

EXT_SOURCE_3	-0.178919
EXT_SOURCE_2	-0.160472
EXT_SOURCE_1	-0.155317
DAYS_EMPLOYED	-0.044932
FLOORSMAX_AVG	-0.044003
FLOORSMAX_MEDI	-0.043768
FLOORSMAX_MODE	-0.043226
AMT_GOODS_PRICE	-0.039645
REGION_POPULATION_RELATIVE	-0.037227
ELEVATORS_AVG	-0.034199

Name: TARGET, dtype: float64

Top 10 highly correlated columns:

```
In [104...]: top_10 = correlations.abs().sort_values().tail(11)
top_10
```

```
Out[104]: FLAG_EMP_PHONE      0.045982
REG_CITY_NOT_WORK_CITY      0.050994
DAYS_ID_PUBLISH            0.051457
DAYS_LAST_PHONE_CHANGE     0.055218
REGION_RATING_CLIENT        0.058899
REGION_RATING_CLIENT_W_CITY 0.060893
DAYS_BIRTH                  0.078239
EXT_SOURCE_1                 0.155317
EXT_SOURCE_2                 0.160472
EXT_SOURCE_3                 0.178919
TARGET                       1.000000
Name: TARGET, dtype: float64
```

Data Preparation

Data Denormalization After EDA

This first section takes all of the aggregated tables and merges them into the single denormalized table, called `bureau_ip_ccb_prev_pos_merged`.

```
In [ ]: import pandas as pd
import os
from sklearn.model_selection import train_test_split

DATA_DIR = "/../Data/"

ds_names = ("application_train", "application_test", "ccb_agg_data", "ip_agg_data",
            "pos_agg_data", "bureau_agg_data")

datasets_agg = {}

for ds_name in ds_names:
    datasets_agg[ds_name] = pd.read_csv(os.getcwd() + DATA_DIR + f'{ds_name}.csv')

In [ ]: pos_merged = datasets_agg["application_train"].merge(datasets_agg["pos_agg_data"])
        .replace(to_replace='\s+', value='_', regex=True) \
        .replace(to_replace='-', value='_', regex=True) \
        .replace(to_replace='\(', value=''', regex=True) \
        .replace(to_replace='\)', value=''', regex=True)

In [ ]: pos_merged_test = datasets_agg["application_test"].merge(datasets_agg["pos_agg_data"])
        .replace(to_replace='\s+', value='_', regex=True) \
        .replace(to_replace='-', value='_', regex=True) \
        .replace(to_replace='\(', value=''', regex=True) \
        .replace(to_replace='\)', value=''', regex=True)
```

```
In [ ]: prev_pos_merged = pos_merged.merge(datasets_agg["prevapp_agg_data"], on='SK_ID_CURR')
        .replace(to_replace='\s+', value='_', regex=True) \
        .replace(to_replace='\-', value='_', regex=True) \
        .replace(to_replace='\(', value=''', regex=True) \
        .replace(to_replace='\)', value=''', regex=True)
```

```
In [ ]: prev_pos_merged_test = pos_merged_test.merge(datasets_agg["prevapp_agg_data"])
        .replace(to_replace='\s+', value='_', regex=True) \
        .replace(to_replace='\-', value='_', regex=True) \
        .replace(to_replace='\(', value=''', regex=True) \
        .replace(to_replace='\)', value=''', regex=True)
```

```
In [ ]: ccb_prev_pos_merged = prev_pos_merged.merge(datasets_agg["ccb_agg_data"], on='SK_ID_CURR')
        .replace(to_replace='\s+', value='_', regex=True) \
        .replace(to_replace='\-', value='_', regex=True) \
        .replace(to_replace='\(', value=''', regex=True) \
        .replace(to_replace='\)', value=''', regex=True)
```

```
In [ ]: ccb_prev_pos_merged_test = prev_pos_merged_test.merge(datasets_agg["ccb_agg_data"])
        .replace(to_replace='\s+', value='_', regex=True) \
        .replace(to_replace='\-', value='_', regex=True) \
        .replace(to_replace='\(', value=''', regex=True) \
        .replace(to_replace='\)', value=''', regex=True)
```

```
In [ ]: ip_ccb_prev_pos_merged = ccb_prev_pos_merged.merge(datasets_agg["ip_agg_data"], on='SK_ID_CURR')
        .replace(to_replace='\s+', value='_', regex=True) \
        .replace(to_replace='\-', value='_', regex=True) \
        .replace(to_replace='\(', value=''', regex=True) \
        .replace(to_replace='\)', value=''', regex=True)
```

```
In [ ]: ip_ccb_prev_pos_merged_test = ccb_prev_pos_merged_test.merge(datasets_agg["ip_agg_data"], on='SK_ID_CURR')
        .replace(to_replace='\s+', value='_', regex=True) \
        .replace(to_replace='\-', value='_', regex=True) \
        .replace(to_replace='\(', value=''', regex=True) \
        .replace(to_replace='\)', value=''', regex=True)
```

```
In [ ]: bureau_ip_ccb_prev_pos_merged = ip_ccb_prev_pos_merged.merge(datasets_agg["bureau_agg_data"], on='SK_ID_CURR')
        .replace(to_replace='\s+', value='_', regex=True) \
        .replace(to_replace='\-', value='_', regex=True) \
        .replace(to_replace='\(', value=''', regex=True) \
        .replace(to_replace='\)', value=''', regex=True)
```

```
In [ ]: bureau_ip_ccb_prev_pos_merged_test = ip_ccb_prev_pos_merged_test.merge(datasets_agg["bureau_agg_data"], on='SK_ID_CURR')
        .replace(to_replace='\s+', value='_', regex=True) \
        .replace(to_replace='\-', value='_', regex=True) \
        .replace(to_replace='\(', value=''', regex=True) \
        .replace(to_replace='\)', value=''', regex=True)
```

```
In [ ]: bureau_ip_ccb_prev_pos_merged.to_csv(os.getcwd() + DATA_DIR + 'bureau_ip_ccb_prev_pos_merged.csv')
```

```
In [ ]: bureau_ip_ccb_prev_pos_merged_test.to_csv(os.getcwd() + DATA_DIR + 'bureau_ip_ccb_prev_pos_merged_test.csv')
```

Data Denormalization after EDA and Feature Engineering

The below code is the denormalization process that occurred on the aggregated feature engineered dataset. This allowed joining of data for train and test and then was exported to a csv file. All of the individual tables were merged into one unnormalized table to train the model on.

```
In [82]: import pandas as pd
import os
from sklearn.model_selection import train_test_split

DATA_DIR = "./Data/"

ds_names = ("application_train_eng", "application_test_eng", "ccb_agg_data_tr",
            "pos_agg_data_tr", "bureau_agg_data_transformed")

datasets_agg = {}

for ds_name in ds_names:
    datasets_agg[ds_name] = pd.read_csv(os.getcwd() + DATA_DIR + f'{ds_name}.csv')
```

```
In [ ]: pos_merged = datasets_agg["application_train_eng"].merge(datasets_agg["pos_agg_data_tr"])
        .replace(to_replace='\s+', value='_', regex=True) \
        .replace(to_replace='-', value='_', regex=True) \
        .replace(to_replace='\(', value=''', regex=True) \
        .replace(to_replace='\)', value='''', regex=True)
```

```
In [ ]: pos_merged_test = datasets_agg["application_test_eng"].merge(datasets_agg["pos_agg_data_tr"])
        .replace(to_replace='\s+', value='_', regex=True) \
        .replace(to_replace='-', value='_', regex=True) \
        .replace(to_replace='\(', value=''', regex=True) \
        .replace(to_replace='\)', value='''', regex=True)
```

```
In [ ]: prev_pos_merged = pos_merged.merge(datasets_agg["prevapp_agg_data_tr"], on="SK_ID_CURR")
        .replace(to_replace='\s+', value='_', regex=True) \
        .replace(to_replace='-', value='_', regex=True) \
        .replace(to_replace='\(', value=''', regex=True) \
        .replace(to_replace='\)', value='''', regex=True)
```

```
In [ ]: prev_pos_merged_test = pos_merged_test.merge(datasets_agg["prevapp_agg_data"])
        .replace(to_replace='\s+', value='_', regex=True) \
        .replace(to_replace='\-', value='_', regex=True) \
        .replace(to_replace='\(', value=''', regex=True) \
        .replace(to_replace='\)', value=''', regex=True)
```

```
In [ ]: ccb_prev_pos_merged = prev_pos_merged.merge(datasets_agg["ccb_agg_data_tr"])
        .replace(to_replace='\s+', value='_', regex=True) \
        .replace(to_replace='\-', value='_', regex=True) \
        .replace(to_replace='\(', value=''', regex=True) \
        .replace(to_replace='\)', value=''', regex=True)
```

```
In [ ]: ccb_prev_pos_merged_test = prev_pos_merged_test.merge(datasets_agg["ccb_agg"])
        .replace(to_replace='\s+', value='_', regex=True) \
        .replace(to_replace='\-', value='_', regex=True) \
        .replace(to_replace='\(', value=''', regex=True) \
        .replace(to_replace='\)', value=''', regex=True)
```

```
In [ ]: ip_ccb_prev_pos_merged = ccb_prev_pos_merged.merge(datasets_agg["ip_agg_data"])
        .replace(to_replace='\s+', value='_', regex=True) \
        .replace(to_replace='\-', value='_', regex=True) \
        .replace(to_replace='\(', value=''', regex=True) \
        .replace(to_replace='\)', value=''', regex=True)
```

```
In [ ]: ip_ccb_prev_pos_merged_test = ccb_prev_pos_merged_test.merge(datasets_agg[""])
        .replace(to_replace='\s+', value='_', regex=True) \
        .replace(to_replace='\-', value='_', regex=True) \
        .replace(to_replace='\(', value=''', regex=True) \
        .replace(to_replace='\)', value=''', regex=True)
```

```
In [ ]: bureau_ip_ccb_prev_pos_merged = ip_ccb_prev_pos_merged.merge(datasets_agg[""])
        .replace(to_replace='\s+', value='_', regex=True) \
        .replace(to_replace='\-', value='_', regex=True) \
        .replace(to_replace='\(', value=''', regex=True) \
        .replace(to_replace='\)', value=''', regex=True)
```

```
In [ ]: bureau_ip_ccb_prev_pos_merged_test = ip_ccb_prev_pos_merged_test.merge(datasets_agg[""])
        .replace(to_replace='\s+', value='_', regex=True) \
        .replace(to_replace='\-', value='_', regex=True) \
        .replace(to_replace='\(', value=''', regex=True) \
        .replace(to_replace='\)', value=''', regex=True)
```

```
In [ ]: bureau_ip_ccb_prev_pos_merged.to_csv(os.getcwd() + DATA_DIR + 'bureau_ip_ccb_prev_pos_merged.csv')
```

```
In [ ]: bureau_ip_ccb_prev_pos_merged_test.to_csv(os.getcwd() + DATA_DIR + 'bureau_ip_ccb_prev_pos_merged_test.csv')
```

Hyperparameter Tuning and Analysis of Models

Many models were run and compared using different combinations of data pre-processing techniques, degrees of feature selection, and hyperparameter tunes. This section outlines the experiments and interprets their results. The main challenge of this section was developing models which could *actually* run and would not crash the python kernel. The computational cost of these algorithms is a common theme and will be revisited often in discussion.

Pre-Processing Techniques

Several techniques were employed to pre process data in various ways, and for different purposes. These are discussed in the sections below.

Feature Engineering

See section 6 for detailed explanations of the methods used to re-engineer and transform features in the various source tables. Untransformed and transformed datasets were one comparison the ML experiments addressed.

Data Type Optimization

When aggregated, the dataset is very large, approximately 2.5 gb of space. When reading these data in as-is, the large memory required to work with the dataset made it a cumbersome object on which to operate ML pipelines. The `reduce_mem_usage()` function is used immediately upon data read-in to counter this large memory requirement. Where possible, this function changes a given column's default datatype to a datatype with a smaller memory footprint. For example, an `int64` column composed of only 1s and 0s (IE OHE columns...) might be converted to an `int8` column with no data loss, and significant memory reduction.

This operation typically led to a ~70% memory size reduction of the imported dataset making it much easier to work with. The function is provided below with an example output. Credit to the publisher: <https://pythonsimplified.com/how-to-handle-large-datasets-in-python-with-pandas/>

```
def reduce_mem_usage(df):
    start_mem = df.memory_usage().sum() / 1024**3
    print('Memory usage of dataframe is {:.2f} MB'.format(start_mem))

    for col in df.columns:
        col_type = df[col].dtype

        if col_type != object:
            c_min = df[col].min()
            c_max = df[col].max()
            if str(col_type)[:3] == 'int':
                if c_min > np.iinfo(np.int8).min and c_max <
np.iinfo(np.int8).max:
                    df[col] = df[col].astype(np.int8)
```

```

        elif c_min > np.iinfo(np.int16).min and c_max <
np.iinfo(np.int16).max:
            df[col] = df[col].astype(np.int16)
        elif c_min > np.iinfo(np.int32).min and c_max <
np.iinfo(np.int32).max:
            df[col] = df[col].astype(np.int32)
        elif c_min > np.iinfo(np.int64).min and c_max <
np.iinfo(np.int64).max:
            df[col] = df[col].astype(np.int64)
    else:
        if c_min > np.finfo(np.float16).min and c_max <
np.finfo(np.float16).max:
            df[col] = df[col].astype(np.float16)
        elif c_min > np.finfo(np.float32).min and c_max <
np.finfo(np.float32).max:
            df[col] = df[col].astype(np.float32)
        else:
            df[col] = df[col].astype(np.float64)
    else:
        df[col] = df[col].astype('category')

end_mem = df.memory_usage().sum() / 1024**3
print('Memory usage after optimization is: {:.2f} MB'.format(end_mem))
print('Decreased by {:.1f}%'.format(100 * (start_mem - end_mem) / start_mem))

return df

```

[Example Output]:

```

application_train
Memory usage of dataframe is 0.28 MB
Memory usage after optimization is: 0.06 MB
Decreased by 79.2%
---
bureau_agg_data
Memory usage of dataframe is 0.66 MB
Memory usage after optimization is: 0.19 MB
Decreased by 71.0%
---
ccb_agg_data
Memory usage of dataframe is 0.13 MB
Memory usage after optimization is: 0.05 MB
Decreased by 63.8%
---
ip_agg_data
Memory usage of dataframe is 0.13 MB
Memory usage after optimization is: 0.05 MB
Decreased by 63.2%
---
pos_agg_data
Memory usage of dataframe is 0.22 MB
Memory usage after optimization is: 0.06 MB
Decreased by 73.0%
---
prevapp_agg_data
Memory usage of dataframe is 1.49 MB
Memory usage after optimization is: 0.40 MB

```

Collinearity Reduction

During Phase 2 exploratory data analysis, multicollinearity between input variables was shown to be prevalent in most of the data tables. Multicollinearity was further increased during the aggregation process as numeric input variables were proliferated into their mean, median, variance counterparts (amongst other aggregation types). This introduction of new (possibly) redundant variables to the ML algorithms significantly increases computation time and often reduces algorithm accuracy/effectiveness.

Used in conjunction with the `DataframeSelector()` transformer class, the `CollinearityReducer()` transformer class combats multicollinearity by removing the *most* (multi-) collinear columns which are *least* correlated to the target variable. The algorithm steps are described below:

1. Given input variable X and target variable y , calculate the correlation matrix
2. Pivot the correlation matrix into a long dataframe of correlation pairs and values (input variable 1, input variable 2, absolute correlation) and drop the following variable pairs:
 - any pair including the target variable
 - any pair with two of the same input variable
 - any pair with an absolute correlation value below a specified threshold value (default is 0.5)
3. For each variable pair, compare to see which variable is *more* correlated to the target variable. These input variables are given a 'win' while the input variable in the pair which is *less* correlated to the target variable is given a 'loss'.
4. Count the total 'wins' and 'losses' for each variable and drop from the original dataframe any variable with 0 wins.
5. Repeat steps 1 to 4 until there are no more input variable pairs with correlations above the threshold, no more input variables scoring 0 wins, or the specified maximum number of iterations has been reached.

The `CollinearityReducer()` thus applies a common multicollinearity solution - drop the variable which is least correlated to the target (Introduction to Statistical Learning Chapter 3). This is no simple task for human to perform on a high-dimensional dataset, but this class provides a mechanical solution so it does not have to be done manually. the `transform` method of this class creates a list of attribute names from the original dataframe which are *to be kept* - multicollinear classes culled by the `CollinearityReducer()` are *not* included in the output. This output list is then used as the input for the `DataframeSelector()` class in the pipeline. Thus, the `CollinearityReducer()` selects columns based only on the training data and there is no leakage from the validation or test data.

Notably, this algorithm is only applied to numerical variables as it is assumed there should naturally be some elevated degree of collinearity between one-hot-encoded categorical

variables. Furthermore, the algorithm should be applied to numerical data which has been subjected to the same scaling and imputation strategy as applied in the actual pipeline.

Both the `CollinearityReducer()` and `DataframeSelector()` classes are shown below along with a basic example of their usage.

```
# transformer reduces the list of columns by a subset
class DataframeSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attribute_names):
        self.attribute_names = attribute_names
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.attribute_names].values

# transformer produces a reduced column list by collinearity reduction
class CollinearityReducer(BaseEstimator, TransformerMixin):

    ...
    This class reduces features by measuring collinearity between the input variables and target.
    Works on numerical features based on the correlations between each variable pair.
    Of the variable pairs with absolute correlations above the threshold value the variables with the lowest target variable correlation are dropped from the input X.
    Repeat until no more collinear pairs with absolute correlations above the threshold or max_iter.

    Inputs:
        X (numpy array) - input variables
        y (numpy array) - target variable
        attribute_names (list) - column names of the input variables (from original dataframe order)
        threshold (int) - the absolute correlation threshold above which variable pairs are subject to the 'correlation competition'
        max_iter (int) - the maximum number of iterations to cut off the algorithm

    Output:
        list - attribute_names to be used by DataframeSelector class
    ...

    def __init__(self, attribute_names, threshold=0.5,
max_iter=None):
        self.attribute_names = attribute_names
        self.threshold = threshold
        self.max_iter = max_iter

    def fit(self, X, y):
        return self
```

```

def transform(self, X, y=None):

    dataframe = pd.concat([y, pd.DataFrame(X)], axis=1)

    i = 0
    while i <= self.max_iter:

        # read-in and assign columns
        # gets correlation matrix between variables and pivots
        to a longer df
        # identify target variable
        # drop same-name and target correlations pairs

        df = dataframe
        features = df.iloc[:,1:].columns
        target_name = df.iloc[:,0].name

        df = pd.melt(abs(df.corr())).reset_index(),
        id_vars='index', value_vars=features)
        targets = df[df['index']==target_name]
        df = df[(df['index'] != df['variable']) & (df['index']
        != target_name) & (df['variable'] != target_name)]

        # combine the correlated variables into ordered pairs
        # aggregate the max correlation and sort pairs
        # split out the variables from the pair
        # join the target variable correlations for each
        variable pair

        df['joined'] = df[['index', 'variable']].apply(lambda
        row: '::'.join(np.sort(row.values.astype(str))), axis=1)

        df = df.groupby('joined', as_index=False) \
            .agg({'value':'max'}) \
            .sort_values(by='value', ascending=False)

        df[['var_1','var_2']] =
        df['joined'].str.split("::",expand=True).astype(int)

        df = df.merge(targets, how='left', left_on='var_1',
        right_on='variable') \
            .merge(targets, how='left', left_on='var_2',
        right_on='variable')
        df.rename(columns = {'value_x':'var_pair_corr',
        'value_y':'var_1_target_corr', 'value':'var_2_target_corr'},
        inplace = True)

        # Take only variable pairs with a correlation greater
        than threshold
        # determine which variable has a higher correlation
        with the target.
        # The higher of the two gets marked as a win

```

```

    # the higher of the two gets marked as a win
    # While the other gets marked as a loss
    # the wins and losses for each variable are then
grouped and summed

    exceeds = df[df['var_pair_corr']>self.threshold]

    # break if none above threshold
    if len(exceeds['var_pair_corr'])==0:
        break

    # "correlation competition"
    exceeds['var_1_win'] = exceeds.apply(lambda row: 1 if
row["var_1_target_corr"] >= row["var_2_target_corr"] else 0,
axis=1)
    exceeds['var_1_loss'] = exceeds.apply(lambda row: 1 if
row["var_2_target_corr"] >= row["var_1_target_corr"] else 0,
axis=1)
    exceeds['var_2_win'] = exceeds.apply(lambda row: 1 if
row["var_1_target_corr"] < row["var_2_target_corr"] else 0, axis=1)
    exceeds['var_2_loss'] = exceeds.apply(lambda row: 1 if
row["var_2_target_corr"] < row["var_1_target_corr"] else 0, axis=1)

    # aggregate scores
    var1 = exceeds[['var_1', 'var_1_win',
'var_1_loss']].groupby('var_1', as_index=False) \
.agg({'var_1_win':'sum', 'var_1_loss':'sum'})
    var1.rename(columns = {'var_1':'var',
'var_1_win':'win', 'var_1_loss':'loss'}, inplace=True)

    var2 = exceeds[['var_2', 'var_2_win',
'var_2_loss']].groupby('var_2', as_index=False) \
.agg({'var_2_win':'sum', 'var_2_loss':'sum'})
    var2.rename(columns = {'var_2':'var',
'var_2_win':'win', 'var_2_loss':'loss'}, inplace=True)

    corrcomps = pd.concat([var1,var2],
axis=0).groupby('var', as_index=False) \
.agg({'win':'sum', 'loss':'sum'})

    # drop variables which had 0 wins
    # IE collinear variables which were always least
related to the target
    dropvars = corrcomps[corrcomps['win']==0]['var']

    dataframe = dataframe.drop(dropvars, axis=1)

    i += 1

X = [self.attribute_names[col] for col in

```

```

dataframe.columns]

    return X

### Example Usage ###

# determine feature types, reduce numerical features by
# collinearity reduction
id_col, feat_num, feat_cat, feature = id_num_cat_feature(X_train,
text = False)

cr = make_pipeline(
    SimpleImputer(strategy='median'),
    StandardScaler(),
    CollinearityReducer(attribute_names=feat_num, threshold = 0.7,
max_iter=2)
)

reduced_feat_num = cr.fit_transform(X_train[feat_num], y_train)

# Pipeline

num_pipeline = Pipeline([
    ('selector', DataFrameSelector(reduced_feat_num)),
    ('imputer', SimpleImputer(strategy="median")),
    ('std_scaler', StandardScaler())
])

```

Balancing Imbalanced Data

In the data provided through Kaggle, 92% were classified as people who can repay the loan and 8% were ones who could not. The data is highly skewed and caused our predictions to also be skewed. A class imbalance could be one of the major reason for lower scores. To tackle this problem and enhance our model predictions we implemented few techniques which would be helpful to balance the data.

- Random Under Sampling: This will help reduce the number of data points for class 1 (which has more data points) by sampling randomly and make it equal to that of class 0.
- Random Oversampling: We will bring the record count of class 0 equal to that of class 1. This method *was not* employed in the pipeline.
- Using multiple accuracy scores to see how many true and false predictions we are making. Some used scores were AUC-ROC, Recall, and balanced accuracy.
- We also used Ensemble Trees which perform better on imbalanced data.

Equally rebalancing the target classes in the training data was done while tuning the multilayer perceptron by adding the following code after the processing the data and before transforming them into the pytorch dataloader objects.

Credit to the author of this code: <https://www.analyticsvidhya.com/blog/2020/07/10-techniques-to-deal-with-classification-imbalance/>

```
In [12]: # https://www.analyticsvidhya.com/blog/2020/07/10-techniques-to-deal-with-classification-imbalance/
# https://www.analyticsvidhya.com/blog/2020/07/10-techniques-to-deal-with-c.

train_imb = pd.DataFrame(np.c_[y_train, X_train])

# class count
# class_count_0, class_count_1 = pd.DataFrame(y_train).value_counts()
class_count_0, class_count_1 = train_imb[0].value_counts()

# Separate class
class_0 = train_imb[train_imb[0] == 0]
class_1 = train_imb[train_imb[0] == 1] # print the shape of the class
print('class 0:', class_0.shape)
print('class 1:', class_1.shape)

class 0: (108548, 838)
class 1: (9532, 838)
```

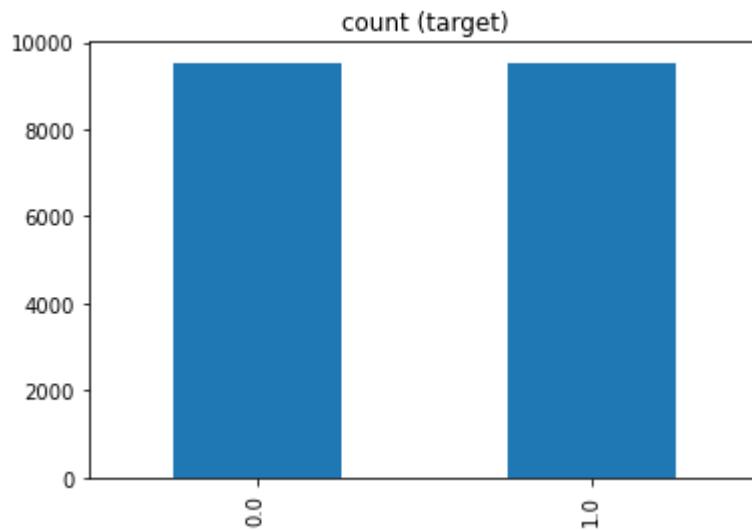
```
In [13]: # sample from class 0 (overrepresented class) the number of values in class
class_0_under = class_0.sample(class_count_1, random_state=42)

test_under = pd.concat([class_0_under, class_1], axis=0)

# this gives both classes equal representation
print("total class of 1 and 0:\n", test_under[0].value_counts()) # plot the count
test_under[0].value_counts().plot(kind='bar', title='count (target)')
plt.show()

# return arrays to be converted to dataloader tensors
X_train_bal = test_under.iloc[:,1:].to_numpy()
y_train_bal = test_under.iloc[:,0].to_numpy()
```

```
total class of 1 and 0:
 0.0    9532
 1.0    9532
Name: 0, dtype: int64
```



Experiment Summary

Hyperparameter tuning was conducted using `GridSearchCV` to methodically test different combinations of hyperparameters. Identical pipelines were run for various combinations of `CollinearityReducer()` hyperparameters to tune this transformer also. Oftentimes, only a subset of the data was used to train and tune models due to the immensity of the dataset - this alleviated computation time headaches. By the Law-of-Large-Numbers, optimal hyperparameters found on these 'micro pipes' are good proxies for optimal parameters on the pipelines using the full dataset.

An example of a "base" pipeline upon which various parameters and models were tuned is shown below. The Area-Under-the-Curve is the scoring metric used as it provides a better measure of fit than the Accuracy score. Where an AUC-ROC score of 0.5 indicates a model composed of randomly guessing, an AUC-ROC score of 1 represents a perfect model. For reference, the baseline logistic regression model scored an AUC-ROC score of ~0.76.

```
num_pipeline = Pipeline([
    ('selector', DataFrameSelector(reduced_feat_num)), # use only
    if CollinearityReducer() implemented
    ('imputer', SimpleImputer(strategy="median")),
    ('std_scaler', StandardScaler())
])

cat_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
])

data_pipeline = ColumnTransformer(
    transformers=[
        ("num_pipeline", num_pipeline, feat_num),
        ("cat_pipeline", cat_pipeline, feat_cat)
    ],
    remainder='drop',
    n_jobs=-1
)

full_pipeline_with_predictor = Pipeline([
    ("preparation", data_pipeline),
    ("classifier", Classifier()) # ML classifier
])

param_1 = [5, 10, 25, 50, 100]
param_2 = [0.1, 1, 10]

parameters = dict(
    classifier_param_1 = param_1,
    classifier_param_2 = param_2
)
```

```

grid = GridSearchCV(
    full_pipeline_with_predictor, param_grid = parameters,
    cv = 3, n_jobs = 4, scoring = 'roc_auc', verbose = 2
)

grid.fit(X_train, y_train)

```

Logistic Regression Model Tuning

The baseline pipeline is a Logistic Regression pipeline with numerical and categorical column transformers. A number of pipeline experiments were conducted and the results of these are shown below. For the sake of brevity, code is only included for the pipelines which gave the most impactful insights into hyperparameter tuning - otherwise only results of the experimental round are show along with bulleted explanations of the pipeline parameters and results.

Logistic Regression with Bureau and Application Data

- These pipelines were tested using only data aggregated from `bureau.csv` and `bureau_balance.csv`, and from `application_train.csv`
- Used 40% of available data
- Both pipelines used logistic regression with L1 regularization
- **Comparisons:**
 - untransformed vs transformed variables
 - L1 vs L2 regularization
- **Interpretations:**
 - The pipeline *without* the transformed data performs better
 - L1 was preferred regularization parameter for *both* pipelines

:	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC
0	LR_bureau_app_transformed	0.9202	0.9179	0.9169	0.7841	0.7255	0.7252
1	LR_bureau_app	0.9199	0.9193	0.9184	0.7683	0.7449	0.7460

Logistic Regression with All Untransformed Data - Regularization

- These pipelines were tested using data aggregated from all the child datasets, and from `application_train.csv`
- Used 40% of available data
- Both pipelines used untransformed data (not feature engineered) with collinearity reduction (`threshold=0.7, max_iter=2`)
- **Comparisons:**
 - L1 vs L2 regularization
- **Interpretations:**
 - Pipeline with L1 regularization parameter performed better

[22]:	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC
0	agg_1293_collinreduce_0.7_2	0.9225	0.9175	0.9182	0.7972	0.7507	0.7543
1	agg_1293_collinreduce:0.7-2-reg-L1	0.9223	0.9175	0.9184	0.7962	0.7532	0.7558

Logistic Regression with All Untransformed Data - Collinearity Reduction

- These pipelines were tested using data aggregated from all the child datasets, and from `application_train.csv`
 - Used 40% of available data
 - All pipelines used untransformed data (not feature engineered) with L1 regularization
 - **Comparisons:**
 - Collinearity Reduction
 - **Interpretations:**
 - Best performing Pipeline is the one with
`CollinearityReducer(threshold=0.5, max_iter=10)`

[11]:	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC
0	LR_agg1293_CR:0.7-2_{'logistic_Reg_penalty': ...}	0.9205	0.9192	0.9183	0.7893	0.7514	0.7605
1	LR_agg1293_CR:0.5-10_{'logistic_Reg_penalty':...}	0.9205	0.9191	0.9189	0.7851	0.7526	0.7639
2	LR_agg1293_{'logistic_Reg_penalty': 'l1'}	0.9215	0.9185	0.9182	0.8016	0.7516	0.7609

Final Hyperparameter Tuning for Logistic Regression

Hyperparameter tuning was done for Logistic Regression. The parameters it was tuned on was C, Penalty, and Solver. This was the algorithm of choice as it is very easy to realize and achieves very good performance with linearly separable classes and has achieved very good results so far and has continued to improve through the EDA and feature engineering.

```
In [ ]: y = bureau_ip_ccb_prev_pos_merged['TARGET']
X = bureau_ip_ccb_prev_pos_merged.drop(['SK_ID_CURR', 'TARGET', 'Unnamed: 0'])

# Split the provided training data into training and validation and test
_, X, _, y = train_test_split(X, y, test_size=0.2, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, rai
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, tes

X_train.replace([np.inf, -np.inf], np.nan, inplace=True)
X_valid.replace([np.inf, -np.inf], np.nan, inplace=True)
X_test.replace([np.inf, -np.inf], np.nan, inplace=True)
```

```
In [ ]: print(f"X train           shape: {X_train.shape}")  
        print(f"X validation     shape: {X_valid.shape}")  
        print(f"X test           shape: {X_test.shape}")
```

```
X train           shape: (39361, 1434)  
X validation     shape: (9841, 1434)  
X test           shape: (12301, 1434)
```

```
In [12]: id_cols, feat_num, feat_cat, features = id_num_cat_feature(X, text = False)
```

```
In [20]: num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy="median")),
    ('std_scaler', StandardScaler())
])

categorical_features = feat_cat
numerical_features = feat_num

selected_features = (numerical_features) + (categorical_features)

cat_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
])

data_pipeline = ColumnTransformer(transformers=[
    ("num_pipeline", num_pipeline, numerical_features),
    ("cat_pipeline", cat_pipeline, categorical_features)],
    remainder='drop',
    n_jobs=-1
)

full_pipeline_with_predictor = Pipeline([
    ("preparation", data_pipeline),
    ("logistic_Reg", LogisticRegression())
])

C = [100, 10, 1.0, 0.1, 0.01]
solvers = ['saga', 'liblinear']
penalty = [
    'l1'
]

parameters = dict(logistic_Reg__C=C,
                  logistic_Reg__solver=solvers,
                  logistic_Reg__penalty=penalty
)
```

```
In [21]: gd4 = GridSearchCV(full_pipeline_with_predictor, param_grid=parameters, cv=5)
```

```
In [22]: gd4.fit(X_train, y_train)
```



```
the coef_ did not converge
    warnings.warn("The max_iter was reached which means "
/Users/pragatwagle/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_
model/_sag.py:328: ConvergenceWarning: The max_iter was reached which means
the coef_ did not converge
    warnings.warn("The max_iter was reached which means "
/Users/pragatwagle/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_
model/_sag.py:328: ConvergenceWarning: The max_iter was reached which means
the coef_ did not converge
    warnings.warn("The max_iter was reached which means "
/Users/pragatwagle/opt/anaconda3/lib/python3.9/site-packages/sklearn/linear_
model/_sag.py:328: ConvergenceWarning: The max_iter was reached which means
the coef_ did not converge
    warnings.warn("The max_iter was reached which means "
```

```
Out[22]: GridSearchCV(cv=3,
                      estimator=Pipeline(steps=[('preparation',
                                                ColumnTransformer(n_jobs=-1,
                                                                  transformers=[('nu
m_pipeline',
                                                Pipeline(steps=[('imputer',
                                                                SimpleImputer(strategy='median'))),
                                                                ('std_scaler',
                                                                StandardScaler()))]),
                                                nnamed: [
                                                '0
_x',
                                                'C
NT_CHILDREN',
                                                'A
MT_INCOME_TOTAL',
                                                'A
MT_CREDIT',
                                                'A
MT_ANNUITY',
                                                'A
MT_GOODS_PRICE',
                                                'R
EGION_POPULATION_RELATIVE',
                                                'D
AYS_BIRTH',...
                                                'N
AME_HOUSING_TYPE',
                                                'O
CCUPATION_TYPE',
                                                'W
EEKDAY_APPR_PROCESS_START',
                                                'O
RGANIZATION_TYPE',
                                                'F
ONDKAPREMONT_MODE',
                                                'H
OUSETYPE_MODE',
                                                'W
ALLSMATERIAL_MODE',
                                                'E
MERGENCYSTATE_MODE'))])),
                      ('logistic_Reg', LogisticRegression
()]),
                      n_jobs=4,
                      param_grid={'logistic_Reg__C': [100, 10, 1.0, 0.1, 0.01],
                                  'logistic_Reg__penalty': ['l1'],
                                  'logistic_Reg__solver': ['saga', 'liblinear']},
                      scoring='roc_auc')
```

```
In [23]: gd4.best_params_
```

```
Out[23]: {'logistic_Reg__C': 0.01,  
          'logistic_Reg__penalty': 'l1',  
          'logistic_Reg__solver': 'liblinear'}
```

Nonparametric Model Tuning

Nonparametric classification models (as opposed to *parametric* classification models like logistic regression) do not rely on underlying assumptions on our dataset. This can be useful for high dimensional data. Decision Tree ("DT") models build trees of binary decisions by which to classify values in a feature space into its target classifications. Used in ensemble (many trees used together), these models can be very powerful! In this project, we explore two types of nonparametric models to compare and tune: *Random Forest* DT models and *Gradient Boosted* DT models.

- In **Random Forest Models**, the trees are grown independently on random samples of the observations. However, each split on each tree is performed using a random subset of the features, thereby decorrelating the trees, and leading to a more thorough exploration of model space relative to bagging. This algorithm combines the output of multiple (randomly created) Decision Trees to generate the final output. In this algorithm, each node in the decision tree is grown based on a random subset of features and subset of the input features.
- In **Gradient Boosted Models**, we only use the original data, and do not draw any random samples. The trees are grown successively, using a “slow” learning approach: each new tree is fit to the signal that is left over from the earlier trees, and shrunken down before it is used. These trees incrementally added to an ensemble by training each new instance to emphasize the training instances previously mis-modeled.

description excerpts from course notes

Tuning and testing of these models are explored below.

Random Forests and Gradient Boosting with Bureau and Application Data

- These pipelines were tested using data aggregated from `bureau.csv` and `bureau_balance.csv`, and from `application_train.csv`
- Used 40% of available data for the first 4 experiments, and 80% of data for the last two experiments
- **Comparisons:**
 - Random Forest and XGB (gradient boosted) classifiers
 - Collinearity Reduction (RF only)
 - Transformed vs Untransformed
 - training size
 - Random Forest Hyperparameters
- **Interpretations:**
 - Best performing Pipeline is the one with

- `CollinearityReducer(threshold=0.5, max_iter=10)`, Random Forest Algorithm on Transformed data with {'rfmax_depth': 25, 'rfmin_samples_leaf': 25}
- The CollinearityReducer() helped the Random Forest algorithm
 - Transformed data tended to do better than untransformed data for both Random Forest and XGB algorithms
 - XGB generally outperformed Random Forest
 - Larger training dataset improved the XGB scores

[35]:

	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC
0	RF_bureau_app_transformed	0.9999	0.9194	0.9193	1.0000	0.6739	0.6778
1	RF_bat_CR:0.5-10_{'rf_max_depth': 25, 'rf_mi...}	0.9193	0.9193	0.9193	0.9119	0.7283	0.7455
2	RF_ba_no-transform_CR:0.5-10_{'rf_max_depth': ...}	0.9193	0.9193	0.9193	0.9095	0.7243	0.7439
3	xgb_ba_no-transform_{'xgb_subsample': 0.8}	0.9567	0.9176	0.9165	0.9801	0.7117	0.7156
4	xgb_0.8_ba_no-transform_{'xgb_subsample': 0.8}	0.9301	0.9183	0.9186	0.8930	0.7396	0.7418
5	0.8_xgb_ba_trans_{'xgb_subsample': 0.8}	0.9302	0.9184	0.9183	0.8934	0.7409	0.7447

Random Forests and Gradient Boosting with All Data Except Credit Card

- These pipelines were tested using data aggregated from all the child datasets (except credit card), and from application_train.csv
- Used 10% of available data
- Used transformed data
- **Comparisons:**
 - Random Forest and XGB (gradient boosted) classifiers
 - Collinearity Reduction
 - Random Forest Hyperparameters
- **Interpretations:**
 - Best performing Pipeline is the XGB algorithm {'xgb_subsample':0.8} with `CollinearityReducer(threshold=0.5, max_iter=25)`
 - The CollinearityReducer() generally helped the XGB algorithm but generally hurt the Random Forest algorithm
 - XGB generally outperformed Random Forest

[27]:

	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC
0	0.1xgb_agg_trans_no-cr_{'xgb_subsample': 0.8}	0.9227	0.9193	0.9192	0.8590	0.7521	0.7472
1	0.1xgb_agg_trans_cr:0.5-10_{'xgb_subsample': ...}	0.9227	0.9195	0.9187	0.8522	0.7612	0.7438
2	0.1xgb_agg_trans_cr:0.5-25_{'xgb_subsample': ...}	0.9224	0.9187	0.9185	0.8459	0.7570	0.7492
3	0.1RF_agg_trans_{'rf_max_depth': 100, 'rf_mi...}	0.9193	0.9193	0.9192	0.9135	0.7133	0.7119
4	0.1RF_agg_trans_cr:0.5-25_{'rf_max_depth': 50...}	0.9193	0.9193	0.9192	0.9090	0.7188	0.7045
5	0.1RF_agg_trans_cr:0.35-25_{'rf_max_depth': 1...}	0.9193	0.9193	0.9192	0.9061	0.7073	0.7000

```
6 0.1RF_agg_trans_cr:0.5-50_{'rf__bootstrap': Fa... 0.9193 0.9193 0.9192 0.9953 0.6873 0.6828
```

Gradient Boosting with All Data Except Credit Card

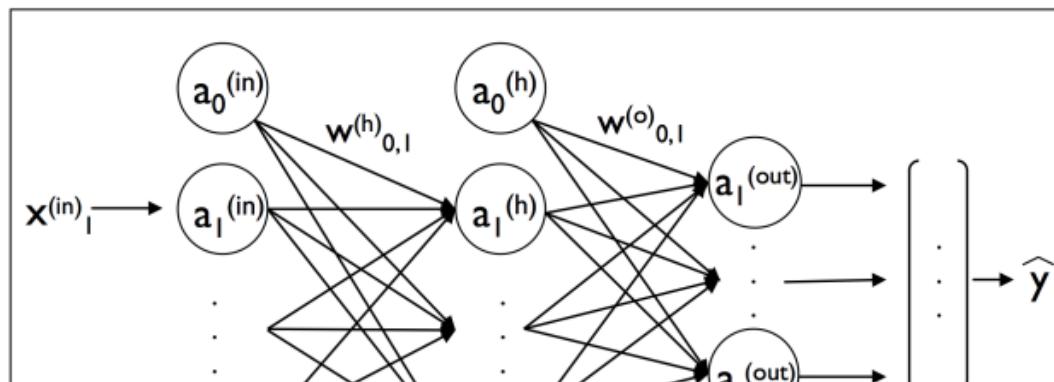
- These pipelines were tested using data aggregated from all the child datasets (except credit card), and from `application_train.csv`
- Used 10% of available data
- **Comparisons:**
 - Collinearity Reduction
 - Transformed vs untransformed data
- **Interpretations:**
 - Best performing Pipeline is the XGB algorithm `{'xgb__subsample':0.8}` with no Collinearity Reduction on untransformed data
 - The `CollinearityReducer()` generally improved performance with more iterations, but generally did not perform as well as models with a higher subsample parameter value *without* the `CollinearityReducer()`
 - Untransformed data outperformed transformed data

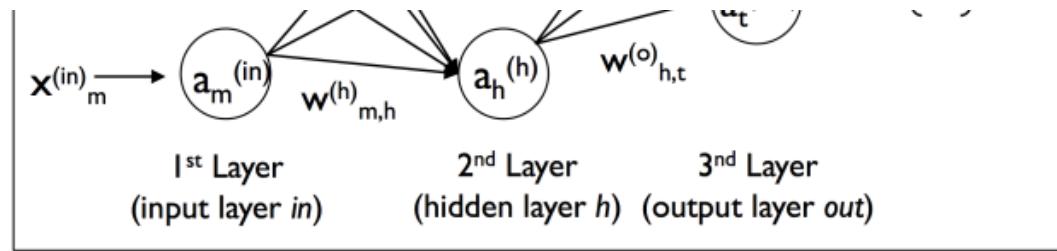
[12]:	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC
0	0.1xgb_agg_trans_no-cr_{'xgb__subsample': 0.8}	0.9227	0.9193	0.9192	0.8590	0.7521	0.7472
1	0.1xgb_agg_trans_cr:0.5-10_{'xgb__subsample': ...}	0.9227	0.9195	0.9187	0.8522	0.7612	0.7438
2	0.1xgb_agg_trans_cr:0.5-25_{'xgb__subsample': ...}	0.9224	0.9187	0.9185	0.8459	0.7570	0.7492

[9]:	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC
0	0.1xgb_agg_no-cr_{'xgb__subsample': 0.8}	0.9234	0.9185	0.9187	0.8580	0.7571	0.7534
1	0.1xgb_agg_cr:0.5-10_{'xgb__subsample': 0.5}	0.9239	0.9187	0.9174	0.8378	0.7550	0.7502
2	0.1xgb_agg_cr:0.5-25_{'xgb__subsample': 0.5}	0.9237	0.9177	0.9185	0.8383	0.7580	0.7530

Multilayer Perceptron Model Tuning

Multilayer Perceptron models were applied to the Home Credit data to predict whether an applicant would default on their loan. These models are described in the figure and text below.





Connecting multiple single neurons to a multilayer feedforward neural network is a special type of fully connected network called a *Multilayer Perceptron*. When such networks have more than one hidden layer, they are called *deep artificial neural network*.

The term *feedforward* refers to the fact that each layer serves as the input to the next layer without loops, in contrast to recurrent neural networks. The feedforward inputs are processed by an input layer of activation functions. These outputs are then processed as inputs for the hidden layer of activation functions. These outputs are then processed as inputs for the output layer of activation functions. This process is also referred to as **Forward Propagation**.

Forward Propagation Algorithm:

1. $Z^{(h)} = A^{(in)}W^{(h)}$ -net input of hidden layer
2. $A^{(h)} = \phi(Z^{(h)})$ -activation of hidden layer
3. $Z^{(out)} = A^{(h)}W^{(out)}$ -net input of output layer
4. $A^{(out)} = \phi(Z^{(out)})$ -activation of output layer

In the sections below, multilayer perceptron models are tuned by augmenting the optimization algorithms used, learning rate, neural network architecture, and data pre-processing techniques.

Multilayer Perceptron comparing Optimizers and Pre-Processing

- These pipelines were tested using aggregated data with the original and transformed features
- Used 60% of available data
- All pipelines used multilayer perceptron models with similar neuron architectures composed of linear objective functions and ReLU activation functions
- **Comparisons:**
 - Optimizers: SGD , Adam , Adagrad , Adadelta , Adamax
 - Collinearity Reduction + Target Class Rebalancing vs. Not
- **Interpretations:**
 - As measured by AUC-ROC, The pipeline with the stochastic gradient descent SGD optimizer performed significantly worse than the pipelines with other optimization algorithms. Pipelines with Adagrad performed the best, while Adadelta and

Adamax were also high performing. Adam was significantly better than SGD , but lagged slightly behind in performance compared to Adagrad .

- The models without the Collinearity reduction and target class rebalancing had a better AUC-ROC scores. However, because of the data imbalance, the models with the class rebalancing had better accuracy. This can be interpreted as the model is better able to predict the class based on input data, as opposed to guessing the overrepresented class 0 and being right 9/10 times.
 - Note: a model without the collinearity reduction but with the target class rebalancing was attempted but continuously failed on running with a kernel error.

	Dataset	Architecture string	Optimizer	Epochs	Learning Rate	Train accuracy	Valid accuracy	Test accuracy	Train auc	Valid auc	Test auc
0	0.6_agg_trans+orig_1585_noCR	1585-128-64-32-16-4-2	<class 'torch.optim.sgd.SGD'>	10	0.005	91.9%	91.9%	91.9%	65.1000000000001%	64.5%	64.9%
1	0.6_agg_trans+orig_1585_noCR	1585-128-64-32-16-4-2	<class 'torch.optim.adam.Adam'>	10	0.005	92.6000000000001%	91.4%	91.4%	84.7%	73.0%	73.8%

	Dataset	Architecture string	Optimizer	Epochs	Learning Rate	Train accuracy	Valid accuracy	Test accuracy	Train auc	Valid auc	Test auc
0	0.6_agg_trans+orig_1585_noCR	1585-128-64-32-16-4-2	<class 'torch.optim.adagrad.Adagrad'>	5	0.005	91.9%	91.9%	91.9%	79.0%	76.4%	76.6%
1	0.6_agg_trans+orig_1585_noCR	1585-64-32-16-4-2	<class 'torch.optim.adagrad.Adagrad'>	5	0.005	91.9%	91.9%	91.9%	80.1000000000001%	76.1%	76.2%
2	0.6_agg_trans+orig_1585_noCR	1585-256-128-64-32-16-4-2	<class 'torch.optim.adagrad.Adagrad'>	5	0.005	91.9%	91.9%	91.9%	80.1000000000001%	76.2%	76.7%
3	0.6_agg_trans+orig_1585_noCR	1585-256-128-64-32-16-4-2	<class 'torch.optim.adagrad.Adagrad'>	5	0.004	92.0%	91.9%	91.9%	79.9%	76.3%	76.7%
4	0.6_agg_trans+orig_1585_noCR	1585-256-128-64-32-16-4-2	<class 'torch.optim.adadelta.Adadelta'>	5	0.004	91.9%	91.9%	91.9%	56.8999999999999%	59.3%	58.6999999999999%
5	0.6_agg_trans+orig_1585_noCR	1585-128-64-32-16-4-2	<class 'torch.optim.adadelta.Adadelta'>	10	0.005	91.9%	91.9%	91.9%	74.3%	73.8%	73.4%
6	0.6_agg_trans+orig_1585_noCR	1585-64-32-16-4-2	<class 'torch.optim.adadelta.Adadelta'>	10	0.005	91.9%	91.9%	91.9%	73.6%	72.5%	72.3999999999999%
7	0.6_agg_trans+orig_1585_noCR	1585-64-32-16-4-2	<class 'torch.optim.adadelta.Adadelta'>	10	0.006	91.9%	91.9%	91.9%	73.9%	72.8999999999999%	72.8999999999999%

	Dataset	Architecture string	Optimizer	Epochs	Learning Rate	Train accuracy	Valid accuracy	Test accuracy	Train auc	Valid auc	Test auc	Recall	Balanced Accuracy
0	0.6_agg_orig+trans_bal_837_CR:0.5-10_ReLU	837-128-64-32-16-8-4-2	<class 'torch.optim.adagrad.Adagrad'>	5	0.005	0.726	0.709	0.71	0.797	0.754	0.758	0.71	0.691
1	0.6_agg_orig+trans_bal_837_CR:0.5-10_ReLU	837-128-64-32-16-4-2	<class 'torch.optim.adagrad.Adagrad'>	5	0.004	0.736	0.704	0.706	0.804	0.755	0.758	0.706	0.689
2	0.6_agg_orig+trans_bal_837_CR:0.5-10_ReLU	837-128-64-32-16-4-2	<class 'torch.optim.adam.Adam'>	5	0.004	0.727	0.676	0.68	0.803	0.748	0.751	0.68	0.688
3	0.6_agg_orig+trans_bal_837_CR:0.5-10_ReLU	837-128-64-32-16-4-2	<class 'torch.optim.adamax.Adamax'>	5	0.004	0.733	0.7	0.699	0.805	0.744	0.747	0.699	0.675
4	0.6_agg_orig+trans_bal_837_CR:0.5-10_ReLU	837-128-64-32-16-8-4-2	<class 'torch.optim.adagrad.Adagrad'>	5	0.001	0.703	0.686	0.686	0.771	0.749	0.753	0.686	0.687
5	0.6_agg_orig+trans_bal_837_CR:0.5-10_ReLU	837-128-64-32-16-8-4-2	<class 'torch.optim.adagrad.Adagrad'>	10	0.001	0.732	0.735	0.736	0.792	0.748	0.752	0.736	0.684
6	0.6_agg_orig+trans_bal_837_CR:0.5-10_ReLU	837-256-64-16-4-2	<class 'torch.optim.adagrad.Adagrad'>	5	0.001	0.714	0.703	0.704	0.786	0.752	0.756	0.704	0.688

Multilayer Perceptron comparing Activation Functions

- These pipelines were tested using aggregated data with the original and transformed features with collinearity reduction
- Used 60% of available data
- All pipelines used multilayer perceptron models with similar neuron layer architectures
- Comparisons:**
 - Activation Functions: Tanh vs. ReLU
- Interpretations:**
 - As measured by AUC-ROC, The pipeline using neurons composed of ReLU activation functions performed slightly better than those with the Tanh activation functions. For comparison see result tables in section 9.2.4.1.

	Dataset	Architecture string	Optimizer	Epochs	Learning Rate	Train accuracy	Valid accuracy	Test accuracy	Train auc	Valid auc	Test auc	Recall	Balanced Accuracy
0	0.6_agg_orig+trans_837_CR:0.5-10_Tanh	837-128-64-32-16-8-4-2	<class 'torch.optim.adagrad.Adagrad'>	5	0.005	0.919	0.919	0.919	0.496	0.5	0.5	0.919	0.5
1	0.6_agg_orig+trans_837_CR:0.5-10_Tanh	837-128-64-32-16-8-4-2	<class 'torch.optim.adagrad.Adagrad'>	5	0.005	0.92	0.918	0.918	0.77	0.752	0.759	0.919	0.511
2	0.6_agg_orig+trans_837_CR:0.5-10_Tanh	837-128-64-32-16-4-2	<class 'torch.optim.adagrad.Adagrad'>	5	0.006	0.922	0.918	0.918	0.788	0.755	0.759	0.918	0.518

Machine Learning Pipelines

Multilayer Perceptron comparing Hidden Layer Architectures and Batch Size Pre-Processing

Please [this blog](#) for more details of OHE when the validation/test have previously unseen unique values.

- These pipelines were tested using aggregated data with transformed features and no unique values.

OHE when previously unseen unique values in the test/validation set

Used 60% of available data

DataFrameSelector Class

- All pipelines used multilayer perceptron models with neuron architectures composed of

In [10]:

```
# Create a class to select numerical or categorical columns
# since Scikit-Learn doesn't handle DataFrames yet
class DataFrameSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attribute_names):
        self.attribute_names = attribute_names
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.attribute_names].values
```

- Also pipelines with an initial hidden layer in the 200s tended to perform well.
- When we try to transform the test set, after having fitted the encoder to the training set,
 - Pipelines with larger training batch sizes performed marginally better.
- we obtain a `ValueError`. This is because the there are new, previously unseen unique

values in the test set and the encoder doesn't know how to handle these values. In order to use both the transformed training and test sets in machine learning algorithms, we need them to have the same number of columns.

This last problem can be solved by using the option `handle_unknown='ignore'` of the `OneHotEncoder`, which, as the name suggests, will ignore previously unseen values when transforming the test set.

Here is an example that in action:

```
# Identify the categorical features we wish to consider.
cat_attribs = ['CODE_GENDER',
               'FLAG_OWN_REALTY', 'FLAG_OWN_CAR', 'NAME_CONTRACT_TYPE',
               'NAME_EDUCATION_TYPE', 'OCCUPATION_TYPE', 'NAME_INCOME_TYPE']

# Notice handle_unknown="ignore" in OHE which ignore values from
# the validation/test that
# do NOT occur in the training set
cat_pipeline = Pipeline([
    ('selector', DataFrameSelector(cat_attribs)),
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('ohe', OneHotEncoder(sparse=False,
handle_unknown="ignore"))])
```

Baseline Model Algorithms

The `sklearn.linear_model.LogisticRegression` implementation will be used for the baseline model with parameters `penalty` that can be any one of these `l1`, `l2`, `elasticnet`, a multi class of `ovr` to fit each label using a binary problem, and `C` which is the inverse of regularization strength. The Logistic Regression loss function will be calculated using cross entropy loss.

The objective function for learning a multinomial logistic regression model (log loss) can be stated as follows:

$$\text{CXE}(\theta) = \left[-\frac{1}{m} \sum_{i=1}^m (y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i)) \right]$$

Regularization helps reduce the risk of overfitting.

- Ridge Regularization (L2):
 - $\text{RidgeCXE}(\theta) = \text{CXE}(\theta) + \lambda \sum_{j=1}^n \theta_j^2$
- Lasso Regularization (L1):
 - $\text{LassoCXE}(\theta) = \text{CXE}(\theta) + \lambda \sum_{j=1}^n |\theta_j|$
- Elastic Net Regularization (Hybrid L1 + L2):
 - $\text{ElasticCXE}(\theta) = \text{CXE}(\theta) + r\lambda \sum_{j=1}^n |\theta_j| + \frac{1-r}{2}\lambda \sum_{j=1}^n \theta_j^2$

Metrics

Area Under the Curve-Receiver Operating Characteristics (AUC-ROC) score will be used to evaluate the accuracy of various logistic regression models (based on varying hyperparameters and thresholds). The AUC score represents the area under the ROC graph. The ROC graph represents a plot of the True Positive Rate (TPR) versus the False Positive Rate (FPR) – where each point on the graph represents a different logistic regression model. The higher the AUC-ROC score, the better. TPR and FPR are defined in equations 1 and 2, respectively.

Equation 1:

$$\text{True Positive Rate, TPR} = \text{True Positives} / (\text{True Positives} + \text{False Negatives})$$

Where:

True Positives = samples accurately classified as class 1
(class 1 = will default on loan)

False Negatives = samples incorrectly classified as class 0
(class 0 = will not default on loan)

Equation 2:

False Positive Rate, FPR = False Positives / (True Negatives + False Positives)

Where:

False Positives = samples incorrectly classified as class 1

True Negatives = samples accurately classified as class 0

Baseline Models - Before EDA

Here are some model runs using only the `application_train` data, with various feature selections.

Baseline with 14 features

In [13]:

```
# Identify the numeric features we wish to consider.
from sklearn.pipeline import Pipeline, FeatureUnion, make_pipeline
from sklearn.compose import ColumnTransformer
from sklearn.datasets import load_iris
import numpy as np
import pandas as pd
from time import time
from sklearn.model_selection import train_test_split # sklearn.cross_vali
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.impute import SimpleImputer
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score

import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

data = datasets["application_train"]
y = data['TARGET']
X = data.drop(['SK_ID_CURR', 'TARGET'], axis = 1) #drop some features with

# Split the provided training data into training and validation and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, r
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, te
print(f"X train          shape: {X_train.shape}")
print(f"X validation      shape: {X_valid.shape}")
print(f"X test           shape: {X_test.shape}")

num_attribs = [
    'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'DAYS_EMPLOYED', 'DAYS_BIRTH', 'EXT_SOURCE_1',
    'EXT_SOURCE_2', 'EXT_SOURCE_3']

num_pipeline = Pipeline([
    ('selector', DataFrameSelector(num_attribs)),
    ('imputer', SimpleImputer(strategy='mean')),
    ('std_scaler', StandardScaler()),
])
# Identify the categorical features we wish to consider.
cat_attribs = ['CODE_GENDER', 'FLAG_OWN_REALTY', 'FLAG_OWN_CAR', 'NAME_CONTRACT_TYPE',
               'NAME_EDUCATION_TYPE', 'OCCUPATION_TYPE', 'NAME_INCOME_TYPE']

selected_features = num_attribs + cat_attribs
# Notice handle_unknown="ignore" in OHE which ignore values from the validation set
# do NOT occur in the training set
cat_pipeline = Pipeline([
    ('selector', DataFrameSelector(cat_attribs)),
    #('imputer', SimpleImputer(strategy='most_frequent')),
    ('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
    ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
])

data_pipeline = ColumnTransformer(transformers=[
    ("num_pipeline", num_pipeline, num_attribs),
    ("cat_pipeline", cat_pipeline, cat_attribs)],
    remainder='drop',
    n_jobs=-1
)
```

```

full_pipeline_with_predictor = Pipeline([
    ("preparation", data_pipeline),
    ("linear", LogisticRegression())
])

param_grid = {'linear__penalty': [ #'l1', 'l2', 'elasticnet',
                                  'none']
              #, 'linear__C':[1.0#, 10.0, 100.0 ]
              }

gd2 = GridSearchCV(full_pipeline_with_predictor, param_grid= param_grid, cv=5)

model = gd2.fit(X_train, y_train)

try:
    expLog
except NameError:
    expLog = pd.DataFrame(columns=[ "exp_name",
                                     "Train Acc",
                                     "Valid Acc",
                                     "Test Acc",
                                     "Train AUC",
                                     "Valid AUC",
                                     "Test AUC"
                                     ])

```

exp_name = f"Baseline_{len(selected_features)}_features"

expLog.loc[len(expLog)] = [f"{exp_name}"] + list(np.round([accuracy_score(y_train, model.predict(X_train)),
 accuracy_score(y_valid, model.predict(X_valid)),
 accuracy_score(y_test, model.predict(X_test)),
 roc_auc_score(y_train, model.predict_proba(X_train)[:, 1]),
 roc_auc_score(y_valid, model.predict_proba(X_valid)[:, 1]),
 roc_auc_score(y_test, model.predict_proba(X_test)[:, 1])],
 4))
expLog

X train shape: (209107, 120)
X validation shape: (52277, 120)
X test shape: (46127, 120)

Out[13]:

	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC
0	Baseline_14_features	0.9199	0.9163	0.9193	0.7358	0.7357	0.7359

Split train, validation and test sets

In [14]:

```
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split

data = datasets["application_train"]
y = data['TARGET']
X = data.drop(['SK_ID_CURR', 'TARGET'], axis = 1) #drop some features with 0 variance

# Split the provided training data into training and validation and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state=42)
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.15, random_state=42)
print(f"X train           shape: {X_train.shape}")
print(f"X validation     shape: {X_valid.shape}")
print(f"X test            shape: {X_test.shape}")
```

```
X train           shape: (209107, 120)
X validation     shape: (52277, 120)
X test            shape: (46127, 120)
```

Baseline 2: All features

In [15]:

```
from sklearn.pipeline import Pipeline, FeatureUnion, make_pipeline
from sklearn.compose import ColumnTransformer
from sklearn.datasets import load_iris
import numpy as np
import pandas as pd
from time import time
from sklearn.model_selection import train_test_split # sklearn.cross_validation
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.impute import SimpleImputer
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score

data = datasets["application_train"]
y = data['TARGET']
X = data.drop(['SK_ID_CURR', 'TARGET'], axis = 1) #drop some features with 0 variance

# Split the provided training data into training and validation and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state=42)
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.15, random_state=42)
print(f"X train          shape: {X_train.shape}")
print(f"X validation    shape: {X_valid.shape}")
print(f"X test           shape: {X_test.shape}")

numerical_features = list(numerical_ix[2:])

num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy="median")),
    ('std_scaler', StandardScaler())
])
categorical_features = list(categorical_ix)

selected_features = (numerical_features) + (categorical_features)

cat_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
])

data_pipeline = ColumnTransformer(transformers=[
    ("num_pipeline", num_pipeline, numerical_features),
    ("cat_pipeline", cat_pipeline, categorical_features)],
    remainder='drop',
    n_jobs=-1
)

full_pipeline_with_predictor = Pipeline([
    ("preparation", data_pipeline),
    ("linear", LogisticRegression())
])

param_grid = {'linear__penalty':[#'l1', 'l2', 'elasticnet',
                                'none'],
              #'linear__C':[1.0#, 10.0, 100.0]
              }
```

```

gd1 = GridSearchCV(full_pipeline_with_predictor, param_grid= param_grid, cv=5)

model = gd1.fit(X_train, y_train)

try:
    expLog
except NameError:
    expLog = pd.DataFrame(columns=["exp_name",
                                    "Train Acc",
                                    "Valid Acc",
                                    "Test Acc",
                                    "Train AUC",
                                    "Valid AUC",
                                    "Test AUC"
                                    ])
exp_name = f"Baseline_{len(selected_features)}_features"
expLog.loc[len(expLog)] = [f"{exp_name}"] + list(np.round(
    [accuracy_score(y_train, model.predict(X_train)),
     accuracy_score(y_valid, model.predict(X_valid)),
     accuracy_score(y_test, model.predict(X_test)),
     roc_auc_score(y_train, model.predict_proba(X_train)[:, 1]),
     roc_auc_score(y_valid, model.predict_proba(X_valid)[:, 1]),
     roc_auc_score(y_test, model.predict_proba(X_test)[:, 1])],
    4))
expLog

```

X train shape: (209107, 120)
X validation shape: (52277, 120)
X test shape: (46127, 120)

Out[15]:

	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC
0	Baseline_14_features	0.9199	0.9163	0.9193	0.7358	0.7357	0.7359
1	Baseline_120_features	0.9200	0.9163	0.9193	0.7478	0.7472	0.7434

Baseline 3: 79 Features

Selected Features

Remove elements with more than 50% nulls

In [20]:

```
from sklearn.pipeline import Pipeline, FeatureUnion, make_pipeline
from sklearn.compose import ColumnTransformer
from sklearn.datasets import load_iris
import numpy as np
import pandas as pd
from time import time
from sklearn.model_selection import train_test_split # sklearn.cross_validation
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.linear_model import LogisticRegression
from sklearn.impute import SimpleImputer
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score

data = datasets["application_train"]
y = data['TARGET']
X = data.drop(['SK_ID_CURR', 'TARGET'], axis = 1) #drop some features with 0 variance

# Split the provided training data into training and validation and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state=42)
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.15, random_state=42)
print(f"X train          shape: {X_train.shape}")
print(f"X validation    shape: {X_valid.shape}")
print(f"X test           shape: {X_test.shape}")

numerical_features = list(numerical_ix[2:].drop(remove_num_nulls))

num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy="median")),
    ('std_scaler', StandardScaler())
])
categorical_features = list(categorical_ix.drop(remove_cat_nulls))

selected_features = (numerical_features) + (categorical_features)

cat_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
])

data_pipeline = ColumnTransformer(transformers=[
    ("num_pipeline", num_pipeline, numerical_features),
    ("cat_pipeline", cat_pipeline, categorical_features)],
    remainder='drop',
    n_jobs=-1
)

full_pipeline_with_predictor = Pipeline([
    ("preparation", data_pipeline),
    ("linear", LogisticRegression())
])

param_grid = {'linear__penalty': ['l1', 'l2', 'elasticnet'],
              'linear__C': [1.0, 10.0, 100.0]
            }
```

```

gd3 = GridSearchCV(full_pipeline_with_predictor, param_grid= param_grid, cv=5)

model = gd3.fit(X_train, y_train)

try:
    expLog
except NameError:
    expLog = pd.DataFrame(columns=["exp_name",
                                    "Train Acc",
                                    "Valid Acc",
                                    "Test Acc",
                                    "Train AUC",
                                    "Valid AUC",
                                    "Test AUC"])
])

exp_name = f"Baseline_{len(selected_features)}_features"
expLog.loc[len(expLog)] = [f"{exp_name}"] + list(np.round(
    [accuracy_score(y_train, model.predict(X_train)),
     accuracy_score(y_valid, model.predict(X_valid)),
     accuracy_score(y_test, model.predict(X_test)),
     roc_auc_score(y_train, model.predict_proba(X_train)[:, 1]),
     roc_auc_score(y_valid, model.predict_proba(X_valid)[:, 1]),
     roc_auc_score(y_test, model.predict_proba(X_test)[:, 1])],
    4))
expLog

```

```

X train          shape: (209107, 120)
X validation      shape: (52277, 120)
X test           shape: (46127, 120)

```

```

Out[20]:      exp_name  Train Acc  Valid Acc  Test Acc  Train AUC  Valid AUC  Test AUC
0  Baseline_14_features  0.9199  0.9163  0.9193  0.7358  0.7357  0.7359
1  Baseline_120_features  0.9200  0.9163  0.9193  0.7478  0.7472  0.7434
2  Baseline_79_features  0.9200  0.9164  0.9195  0.7441  0.7442  0.7406

```

Baseline 4 : 79 features; 2 log features

Remove elements with more than 50% nulls with log AMT_ANNUITY and AMT_CREDIT

In [21]:

```
data = datasets["application_train"]
y = data['TARGET']
X = data.drop(['SK_ID_CURR', 'TARGET'], axis = 1) #drop some features with 0 variance
X['LOG_AMT_ANNUITY'] = np.log(X['AMT_ANNUITY']) #add LOG_AMT_ANNUITY column
X = X.drop(['AMT_ANNUITY'], axis = 1) # drop AMT_ANNUITY column
X['LOG_AMT_CREDIT'] = np.log(X['AMT_CREDIT']) #add LOG_AMT_ANNUITY column
X = X.drop(['AMT_CREDIT'], axis = 1) # drop AMT_ANNUITY column

# Split the provided training data into training and validation and test sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_state=42)
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.15, random_state=42)
print(f"X train          shape: {X_train.shape}")
print(f"X validation      shape: {X_valid.shape}")
print(f"X test           shape: {X_test.shape}")

numerical_features = list(numerical_ix[2:].drop(remove_num_nulls))
numerical_features.append('LOG_AMT_ANNUITY')
numerical_features.append('LOG_AMT_CREDIT')
numerical_features.remove('AMT_CREDIT')
numerical_features.remove('AMT_ANNUITY')

num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy="median")),
    ('std_scaler', StandardScaler())
])

categorical_features = list(categorical_ix.drop(remove_cat_nulls))

selected_features = (numerical_features) + (categorical_features)

cat_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
])

data_pipeline = ColumnTransformer(transformers=[
    ("num_pipeline", num_pipeline, numerical_features),
    ("cat_pipeline", cat_pipeline, categorical_features)],
    remainder='drop',
    n_jobs=-1
)

full_pipeline_with_predictor = Pipeline([
    ("preparation", data_pipeline),
    ("linear", LogisticRegression())
])

param_grid = {'linear__penalty':[#'l1', 'l2', 'elasticnet',
                                'none'],
              #'linear__C':[1.0#, 10.0, 100.0]
              }

gd4 = GridSearchCV(full_pipeline_with_predictor, param_grid= param_grid, cv=5)

model = gd4.fit(X_train, y_train)
```

```

try:
    expLog
except NameError:
    expLog = pd.DataFrame(columns=[ "exp_name",
                                    "Train Acc",
                                    "Valid Acc",
                                    "Test Acc",
                                    "Train AUC",
                                    "Valid AUC",
                                    "Test AUC"
                                ])
exp_name = f"Baseline_{len(selected_features)}_features with log attributes"
expLog.loc[len(expLog)] = [f"{exp_name}"] + list(np.round(
    [accuracy_score(y_train, model.predict(X_train)),
     accuracy_score(y_valid, model.predict(X_valid)),
     accuracy_score(y_test, model.predict(X_test)),
     roc_auc_score(y_train, model.predict_proba(X_train)[:, 1]),
     roc_auc_score(y_valid, model.predict_proba(X_valid)[:, 1]),
     roc_auc_score(y_test, model.predict_proba(X_test)[:, 1])],
    4))
expLog

```

X train shape: (209107, 120)
X validation shape: (52277, 120)
X test shape: (46127, 120)

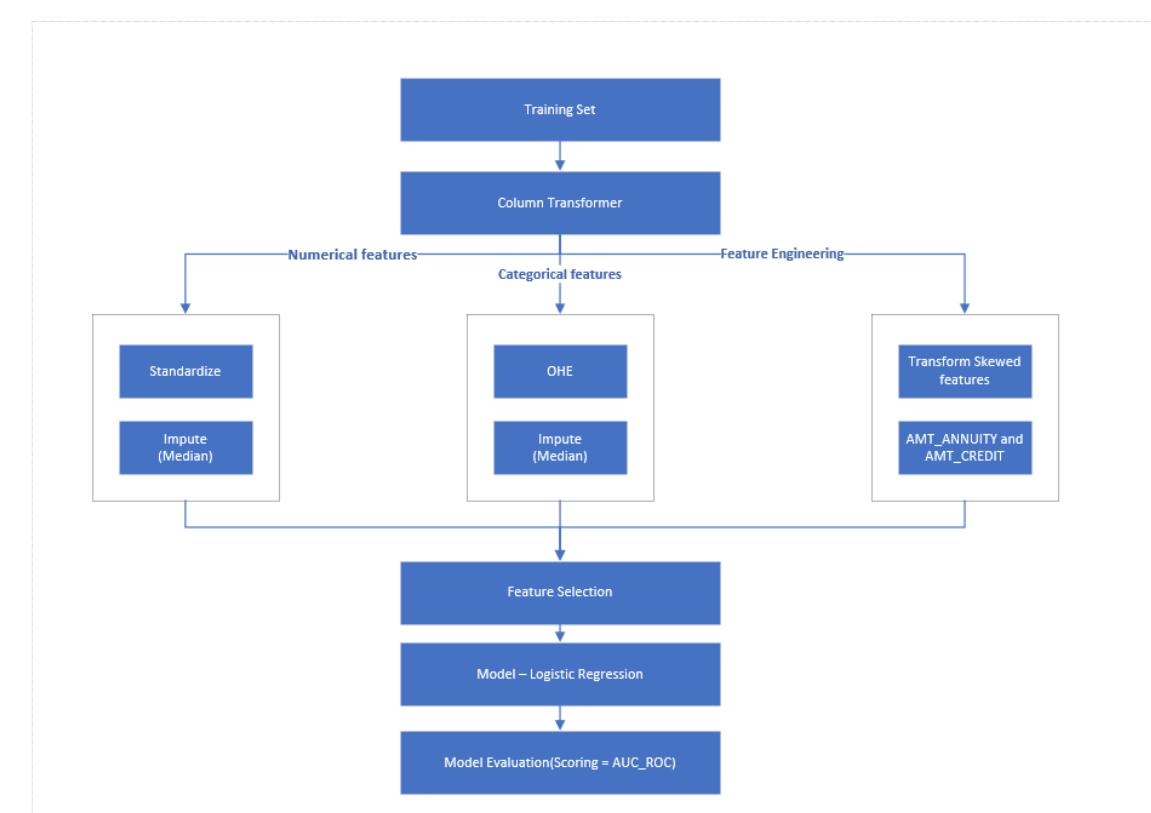
	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC
0	Baseline_14_features	0.9199	0.9163	0.9193	0.7358	0.7357	0.7359
1	Baseline_120_features	0.9200	0.9163	0.9193	0.7478	0.7472	0.7434
2	Baseline_79_features	0.9200	0.9164	0.9195	0.7441	0.7442	0.7406
3	Baseline_79_features with log attributes	0.9200	0.9164	0.9195	0.7443	0.7447	0.7405

Baseline Model on Aggregated Features - Post EDA

The model below was constructed after exploratory data analysis was performed on all the tables. The sections below outline the data denormalization process (aggregating the tables into a singular input variable 'X') and then incorporate them into a baseline logistic regression model without regularization.

Model Pipeline with All Features

This section runs the baseline ML pipeline using all the integrated data features as outlined by the block diagram below:



In [5]:

```
DATA_DIR = "/../Data/"

datasets_agg = {}
datasets_agg["bureau_ip_ccb_prev_pos_merged"] = pd.read_csv(os.getcwd() + D/
bureau_ip_ccb_prev_pos_merged = datasets_agg["bureau_ip_ccb_prev_pos_merged"]
```

In [7]:

```
y = bureau_ip_ccb_prev_pos_merged['TARGET']
X = bureau_ip_ccb_prev_pos_merged.drop(['SK_ID_CURR', 'TARGET', 'Unnamed: 0', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'DAYS_LAST_PHONE_CHANGE', 'DAYS_ID_PUBLISH'], axis=1)

#X['LOG_AMT_ANNUITY'] = np.log(X['AMT_ANNUITY']) #add LOG_AMT_ANNUITY column
#X = X.drop(['AMT_ANNUITY'], axis = 1) # drop AMT_ANNUITY column
#X['LOG_AMT_CREDIT'] = np.log(X['AMT_CREDIT']) #add LOG_AMT_ANNUITY column
#X = X.drop(['AMT_CREDIT'], axis = 1) # drop AMT_ANNUITY column

# Split the provided training data into training and validation and test
_, X, _, y = train_test_split(X, y, test_size=0.8, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.2, random_state=42)

X_train.replace([np.inf, -np.inf], np.nan, inplace=True)
X_valid.replace([np.inf, -np.inf], np.nan, inplace=True)
X_test.replace([np.inf, -np.inf], np.nan, inplace=True)
```

```
In [8]:  
print(f"X train      shape: {X_train.shape}")  
print(f"X validation  shape: {X_valid.shape}")  
print(f"X test        shape: {X_test.shape}")  
  
X train      shape: (157445, 1277)  
X validation  shape: (39362, 1277)  
X test        shape: (49202, 1277)  
  
In [58]:  
id_cols, feat_num, feat_cat, features = id_num_cat_feature(X, text = False)  
  
In [62]:  
num_pipeline = Pipeline([  
    ('imputer', SimpleImputer(strategy="median")),  
    ('std_scaler', StandardScaler())  
])  
  
categorical_features = feat_cat  
numerical_features = feat_num  
  
selected_features = (numerical_features) + (categorical_features)  
  
cat_pipeline = Pipeline([  
    ('imputer', SimpleImputer(strategy='most_frequent')),  
    ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))  
])  
  
data_pipeline = ColumnTransformer(transformers=[  
    ("num_pipeline", num_pipeline, numerical_features),  
    ("cat_pipeline", cat_pipeline, categorical_features)],  
    remainder='drop',  
    n_jobs=-1  
)  
  
full_pipeline_with_predictor = Pipeline([  
    ("preparation", data_pipeline),  
    # ('pca', decomposition.PCA()),  
    ("logistic_Reg", LogisticRegression(solver="liblinear"))  
])  
  
penalty = [  
    #'l1', 'l2'  
]  
  
parameters = dict()  
  
In [63]:  
gd4 = GridSearchCV(full_pipeline_with_predictor, param_grid= parameters, cv  
  
In [64]:  
gd4.fit(X_train, y_train)
```

```
Out[64]: GridSearchCV(cv=5,
                      estimator=Pipeline(steps=[('preparation',
                                                ColumnTransformer(n_jobs=-1,
                                                                  transformers=[('nu
m_pipeline',
                                                Pipeline(steps=[('imputer',
                                                                SimpleImputer(strategy='median'))),
                                                                ('std_scaler',
                                                                StandardScaler()))]),
                                                [ 'C
NT_CHILDREN',
                                                'A
MT_INCOME_TOTAL',
                                                'A
MT_CREDIT',
                                                'A
MT_ANNUITY',
                                                'A
MT_GOODS_PRICE',
                                                'R
EGION_POPULATION_RELATIVE',
                                                'D
AYS_BIRTH',
                                                'D
AYS_EMPLOYED', ...
                                                'N
AME_TYPE_SUITE',
                                                'N
AME_INCOME_TYPE',
                                                'N
AME_EDUCATION_TYPE',
                                                'N
AME_FAMILY_STATUS',
                                                'N
AME_HOUSING_TYPE',
                                                'O
CCUPATION_TYPE',
                                                'W
EEKDAY_APPR_PROCESS_START',
                                                'O
RGANIZATION_TYPE',
                                                'F
ONDKAPREMONT_MODE',
                                                'H
OUSETYPE_MODE',
                                                'W
ALLSMATERIAL_MODE',
                                                'E
MERGENCYSTATE_MODE'])])),
                      ('logistic_Reg',
                      LogisticRegression(solver='liblinear
'))),
                      n_jobs=-10, param_grid={}, scoring='roc_auc')
```

```
In [67]: from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score

X_valid.replace([np.inf, -np.inf], np.nan, inplace=True)
X_test.replace([np.inf, -np.inf], np.nan, inplace=True)

try:
    expLog
except NameError:
    expLog = pd.DataFrame(columns=["exp_name",
                                    "Train Acc",
                                    "Valid Acc",
                                    "Test Acc",
                                    "Train AUC",
                                    "Valid AUC",
                                    "Test AUC"
                                    ])
    )

exp_name = f"Baseline_{len(selected_features)}_features with log attributes"
expLog.loc[len(expLog)] = [f"{exp_name}"] + list(np.round(
    [accuracy_score(y_train, gd4.predict(X_train)),
     accuracy_score(y_valid, gd4.predict(X_valid)),
     accuracy_score(y_test, gd4.predict(X_test)),
     roc_auc_score(y_train, gd4.predict_proba(X_train)[:, 1]),
     roc_auc_score(y_valid, gd4.predict_proba(X_valid)[:, 1]),
     roc_auc_score(y_test, gd4.predict_proba(X_test)[:, 1])
    ],
    4))
expLog
```

	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC
0	Baseline_1277_features with log attributes	0.9198	0.9209	0.9186	0.7871	0.7663	0.7709

Pipelines After EDA and Feature Engineering

These are pipelines run during phase 3 after EDA and Feature Engineering

Read Aggregated and Feature Engineered Train data set

Below is the code to read the feature engineered aggregated train data set.

```
In [22]: DATA_DIR = "/../Data/"

datasets_agg = {}
datasets_agg["bureau_ip_ccb_prev_pos_merged_train_tr"] = pd.read_csv(os.geteuid())
bureau_ip_ccb_prev_pos_merged = datasets_agg["bureau_ip_ccb_prev_pos_merged"]
```

Pipeline 1 after EDA and Feature Engineering - All Aggregated and Feature Engineered Data

This pipeline model is Logistic Regression using the above tuned hyperparameters of C=0.01, penalty = l1, and solver = liblinear. The model was trained on the aggregated and feature engineered data. No collinearity reduce was used.

```
In [42]:  
y = bureau_ip_ccb_prev_pos_merged['TARGET']  
X = bureau_ip_ccb_prev_pos_merged.drop(['SK_ID_CURR', 'TARGET', 'Unnamed: 0',  
                                         'DAYS_BIRTH', 'DAYS_EMPLOYED', 'DAYS_REGISTRATION',  
                                         'DAYS_ID_PUBLISH', 'DAYS_LAST_PHONE_CHANGE',  
                                         'DAYS_LAST_PHONE_CHANGE'], axis=1)  
  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)  
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.25, random_state=42)  
  
X_train.replace([np.inf, -np.inf], np.nan, inplace=True)  
X_valid.replace([np.inf, -np.inf], np.nan, inplace=True)  
X_test.replace([np.inf, -np.inf], np.nan, inplace=True)
```

```
In [ ]:  
print(f"X train           shape: {X_train.shape}")  
print(f"X validation     shape: {X_valid.shape}")  
print(f"X test            shape: {X_test.shape}")
```

```
X train           shape: (157445, 1434)  
X validation     shape: (39362, 1434)  
X test            shape: (49202, 1434)
```

```
In [25]:  
id_cols, feat_num, feat_cat, features = id_num_cat_feature(X, text = False)
```

```
In [26]: num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy="median")),
    ('std_scaler', StandardScaler())
])

categorical_features = feat_cat
numerical_features = feat_num

selected_features = (numerical_features) + (categorical_features)

cat_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
])

data_pipeline = ColumnTransformer(transformers=[
    ("num_pipeline", num_pipeline, numerical_features),
    ("cat_pipeline", cat_pipeline, categorical_features)],
    remainder='drop',
    n_jobs=-1
)

full_pipeline_with_predictor = Pipeline([
    ("preparation", data_pipeline),
    ("logistic_Reg", LogisticRegression(solver="liblinear", penalty = 'l1'))
])

C = [0.1, 0.01]
solvers = ['saga', 'liblinear']
penalty = [
    'l1', 'l2'
]

parameters = dict()
```

```
In [27]: gd4 = GridSearchCV(full_pipeline_with_predictor, param_grid=parameters, cv=5)
```

```
In [28]: gd4.fit(X_train, y_train)
```

```
Out[28]: GridSearchCV(cv=3,
                      estimator=Pipeline(steps=[('preparation',
                                                ColumnTransformer(n_jobs=-1,
                                                                  transformers=[('num_pipeline',
                                                                 Pipeline(steps=[('imputer',
                                                                 SimpleImputer(strategy='median'))),
                                                                ('std_scaler',
                                                                 StandardScaler()))]),
                                                nnamed: [
                                              '0_x',
                                              'CNT_CHILDREN',
                                              'A_MT_INCOME_TOTAL',
                                              'A_MT_CREDIT',
                                              'A_MT_ANNUITY',
                                              'A_MT_GOODS_PRICE',
                                              'R_EGION_POPULATION_RELATIVE',
                                              'D_AYS_BIRTH',
                                              'N_AME_INCOME_TYPE',
                                              'N_AME_EDUCATION_TYPE',
                                              'N_AME_FAMILY_STATUS',
                                              'N_AME_HOUSING_TYPE',
                                              'O_CCUPATION_TYPE',
                                              'W_EEKDAY_APPR_PROCESS_START',
                                              'O_ORGANIZATION_TYPE',
                                              'F_ONDKAPREMONT_MODE',
                                              'H_OUSETYPE_MODE',
                                              'W_ALLSMATERIAL_MODE',
                                              'E_MERGENCYSTATE_MODE'])])),
                      ('logistic_Reg',
                       LogisticRegression(C=0.01, penalty='l1',
                                          solver='liblinear'))),
                      n_jobs=4, param_grid={}, scoring='roc_auc')
```

In [23]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import recall_score
from sklearn.metrics import balanced_accuracy_score

X_valid.replace([np.inf, -np.inf], np.nan, inplace=True)
X_test.replace([np.inf, -np.inf], np.nan, inplace=True)

try:
    expLogTr
except NameError:
    expLogTr = pd.DataFrame(columns=["exp_name",
                                      "hyperparamters",
                                      "Train Acc",
                                      "Valid Acc",
                                      "Test Acc",
                                      "Train AUC",
                                      "Valid AUC",
                                      "Test AUC",
                                      "Recall",
                                      "Balanced Accuracy"
                                     ])
exp_name = f"Baseline_{len(selected_features)}_features_and_{len(X_train)}_l"
expLogTr.loc[len(expLogTr)] = [f"{exp_name}" + [f"C=0.01, penalty=l1, and s",
                                                [accuracy_score(y_train, gd4.predict(X_train)),
                                                 accuracy_score(y_valid, gd4.predict(X_valid)),
                                                 accuracy_score(y_test, gd4.predict(X_test)),
                                                 roc_auc_score(y_train, gd4.predict_proba(X_train)[:, 1]),
                                                 roc_auc_score(y_valid, gd4.predict_proba(X_valid)[:, 1]),
                                                 roc_auc_score(y_test, gd4.predict_proba(X_test)[:, 1]),
                                                 recall_score(y_test, gd4.predict(X_test), average='weighted'),
                                                 balanced_accuracy_score(y_test, gd4.predict(X_test))
                                                ],
                                              4))

expLogTr
```

Out[23]:

	exp_name	hyperparamters	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC	Recall
0	Baseline_1434_features and 157445_number of sa...	C=0.01, penalty=l1, and solver=liblinear	0.9193	0.9214	0.9194	0.7750	0.7698	0.7734	0.9194
1	Baseline_1434_features and 177126_number of sa...	C=0.01, penalty=l1, and solver=liblinear	0.9204	0.9172	0.9187	0.7755	0.7741	0.7699	0.9187

Submission File Prep for EDA and Feature Engineering

For each SK_ID_CURR in the test set, you must predict a probability for the TARGET variable. The file should contain a header and have the following format:

```
SK_ID_CURR, TARGET  
100001, 0.1  
100005, 0.9  
100013, 0.2  
etc.
```

```
In [76]:  
DATA_DIR = "/../Data/"  
  
datasets_agg = {}  
datasets_agg["bureau_ip_ccb_prev_pos_merged_test_tr"] = pd.read_csv(os.getcwd() + DATA_DIR + "bureau_ip_ccb_prev_pos_merged_test_tr.csv")  
bureau_ip_ccb_prev_pos_merged_test = datasets_agg["bureau_ip_ccb_prev_pos_merged_test"]  
  
In [77]:  
data_test = bureau_ip_ccb_prev_pos_merged_test  
X_Kaggle_test = data_test.drop(['SK_ID_CURR', 'Unnamed: 0'], axis = 1)  
X_Kaggle_test.replace([np.inf, -np.inf], np.nan, inplace=True)  
  
print(X_Kaggle_test.shape)  
  
(48744, 1434)  
  
In [78]:  
X_Kaggle_test.replace([np.inf, -np.inf], np.nan, inplace=True)  
  
In [79]:  
test_class_scores = gd4.predict_proba(X_Kaggle_test)[:, 1]  
  
In [82]:  
submit_df = datasets_agg["application_test_eng"][['SK_ID_CURR']]  
submit_df['TARGET'] = test_class_scores  
submit_df.head()  
  
Out[82]:  
SK_ID_CURR  TARGET  
0 100001  0.074303  
1 100005  0.149398  
2 100013  0.025599  
3 100028  0.024277  
4 100038  0.143657  
  
In [ ]:  
submit_df.to_csv("submission6.csv", index=False)
```

Kaggle Submission for EDA and Feature Engineering --> Submission 5 and 6

Click on this [link](#)

	submission6.csv Complete (after deadline) · 1h ago · .8 split	0.76406	0.76898	<input type="checkbox"/>
	submission5.csv Complete (after deadline) · 1h ago · With 90% of data	0.76509	0.76808	<input type="checkbox"/>
	submission4.csv Complete (after deadline) · 4h ago · After Feature Engineering	0.76406	0.76898	<input type="checkbox"/>
	submission3.csv Complete (after deadline) · 7d ago · Second Submission for Phase 2	0.76011	0.7617	<input type="checkbox"/>
	submission2.csv Complete (after deadline) · 7d ago · The second submission	0.74958	0.75199	<input type="checkbox"/>
	submission.csv Complete (after deadline) · 8d ago · not 0 or 1	0.73653	0.73569	<input type="checkbox"/>

Collinearity Reducer Pipeline with EDA And Feature Engineering

This pipeline model is Logistic Regression using the above tuned hyperparameters of C=0.01, penalty = l1, and solver = liblinear. The aggregated and feature engineered data is passed through the below collinearity reducer and only the columns returned by that function is used in the model training process.

Refer to section 9.1.3 for details on the reducer

Import Data, Pipeline creation and train and test selection

```
In [18]: y = bureau_ip_ccb_prev_pos_merged['TARGET']
X = bureau_ip_ccb_prev_pos_merged.drop(['SK_ID_CURR', 'TARGET', 'Unnamed: 0',
_, X, _, y = train_test_split(X, y, test_size=0.8, random_state=42, stratify=y)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.5, random_state=42, stratify=y)

X_train.replace([np.inf, -np.inf], np.nan, inplace=True)
X_valid.replace([np.inf, -np.inf], np.nan, inplace=True)
X_test.replace([np.inf, -np.inf], np.nan, inplace=True)

print(f"X train           shape: {X_train.shape}")
print(f"X validation     shape: {X_valid.shape}")
print(f"X test            shape: {X_test.shape}")

X train           shape: (157445, 1434)
X validation     shape: (39362, 1434)
X test            shape: (49202, 1434)
```

```
In [ ]: id_col, feat_num, feat_cat, feature = id_num_cat_feature(X, text = False)
```

```
In [20]: ### Collinear Feature Reduction
cr = make_pipeline(
    SimpleImputer(strategy='median'),
    StandardScaler(),
    CollinearityReducer(attribute_names=feat_num, threshold = 0.7, max_iter=10)
)

reduced_feat_num = cr.fit_transform(X_train[feat_num], y_train)

print(f'Reduced numerical column count from {len(feat_num)}... ')
print(f'...to {len(reduced_feat_num)} by collinearity reduction.')
```

Reduced numerical column count from 1418...
...to 897 by collinearity reduction.

```
-----
NameError Traceback (most recent call last)
/var/folders/6f/zmcx1rs12vd7qxtgld2cqcf80000gn/T/ipykernel_979/913761432.py
in <module>
    41
    42     grid = GridSearchCV(
--> 43         full_pipeline_with_predictor, param_grid= parameters,
    44         cv = 3, n_jobs=-1, scoring='roc_auc', verbose=2
    45 )
```

NameError: name 'parameters' is not defined

```
In [ ]: X_train = X_train[reduced_feat_num]
```

```
In [ ]: X_train.shape
```

```
Out[ ]: (157445, 897)
```

```
In [69]: id_col, feat_num, feat_cat, feature = id_num_cat_feature(X_train, text = False)
```

```
In [ ]: num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy="median")),
    ('std_scaler', StandardScaler())
])

cat_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
])

data_pipeline = ColumnTransformer(transformers=[
    ("num_pipeline", num_pipeline, feat_num),
    ("cat_pipeline", cat_pipeline, feat_cat)],
    remainder='drop',
    n_jobs=-1
)

full_pipeline_with_predictor = Pipeline([
    ("preparation", data_pipeline),
    # ('pca', PCA()),
    ("logistic_Reg", LogisticRegression(solver="liblinear", penalty = 'l1',
    )
```

```
In [71]: parameters = dict()
```

```
In [72]: grid = GridSearchCV(
    full_pipeline_with_predictor, param_grid = dict(),
    cv = 3, n_jobs=4, scoring='roc_auc', verbose=2
)
```

```
In [73]: grid.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 1 candidates, totalling 3 fits
[CV] END ..... total time= 1.0s
[CV] END ..... total time= 1.0s
[CV] END ..... total time= 1.0s
[CV] END ..... total time= 1.2min
[CV] END ..... total time= 1.0s
[CV] END ..... total time= 1.2min
[CV] END ..... total time= 5.20s
```

```
Out[73]: GridSearchCV(cv=3,
                      estimator=Pipeline(steps=[('preparation',
                                                ColumnTransformer(n_jobs=-1,
                                                                  transformers=[('nu
m_pipeline',
                                                Pipeline(steps=[('imputer',
                                                                SimpleImputer(strategy='median'))),
                                                                ('std_scaler',
                                                                StandardScaler()))]),
                                                [ 'U
nnamed: '
                                                '0
_x',
                                                'A
MT_ANNUITY',
                                                'R
EGION_POPULATION_RELATIVE',
                                                '0
WN_CAR_AGE',
                                                'F
LAG_MOBIL',
                                                'F
LAG_WORK_PHONE',
                                                'F
LAG_CONT_MOBILE',
                                                'F
LAG_PHONE',
                                                '...
                                                'N
ONLIVINGAPARTMENTS_AVG',
                                                'N
ONLIVINGAREA_AVG',
                                                'T
OTALAREA_MODE', ...]), ('ca
t_pipeline',
                                                Pip
eline(steps=[('imputer',
                                                                SimpleImputer(strategy='most_frequent'))),
                                                                ('ohe',
                                                                OneHotEncoder(handle_unknown='ignore',
                                                                sparse=False))]),
                                                                []]]),
                                                                ('logistic_Reg',
                                                                LogisticRegression(C=0.01, penalty='
l1',
                                                                solver='liblinear
'))]),
                                                                n_jobs=4, param_grid={}, scoring='roc_auc', verbose=2)
```

In [74]:

```
from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import recall_score
from sklearn.metrics import balanced_accuracy_score

X_valid.replace([np.inf, -np.inf], np.nan, inplace=True)
X_test.replace([np.inf, -np.inf], np.nan, inplace=True)

try:
    expLogCR
except NameError:
    expLogCR = pd.DataFrame(columns=["exp_name",
                                      "hyperparamters",
                                      "Train Acc",
                                      "Valid Acc",
                                      "Test Acc",
                                      "Train AUC",
                                      "Valid AUC",
                                      "Test AUC",
                                      "Recall",
                                      "Balanced Accuracy"
                                     ])
exp_name = f"Baseline_{len(reduced_feat_num)}_features and {len(X_train)}_n"
expLogCR.loc[len(expLogCR)] = [f"{exp_name}" + [f"C=0.01, penalty=l1, and s",
                                                accuracy_score(y_train, grid.predict(X_train)),
                                                accuracy_score(y_valid, grid.predict(X_valid)),
                                                accuracy_score(y_test, grid.predict(X_test)),
                                                roc_auc_score(y_train, grid.predict_proba(X_train)[:, 1]),
                                                roc_auc_score(y_valid, grid.predict_proba(X_valid)[:, 1]),
                                                roc_auc_score(y_test, grid.predict_proba(X_test)[:, 1]),
                                                recall_score(y_test, grid.predict(X_test), average='weighted'),
                                                balanced_accuracy_score(y_test, grid.predict(X_test))
                                              ],
                                 4))

expLogCR
```

Out[74]:

	exp_name	hyperparamters	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC	Recall
0	Baseline_897_features and 157445_number of sam...	C=0.01, penalty=l1, and solver=liblinear	0.9197	0.9195	0.9198	0.7732	0.7681	0.7706	0.9198
1	Baseline_897_features and 157445_number of sam...	C=0.01, penalty=l1, and solver=liblinear	0.9196	0.9195	0.9199	0.7669	0.7623	0.7642	0.9199

Submission File Prep for Colinear Reducer

For each SK_ID_CURR in the test set, you must predict a probability for the TARGET variable. The file should contain a header and have the following format:

```
SK_ID_CURR, TARGET  
100001, 0.1  
100005, 0.9  
100013, 0.2  
etc.
```

```
In [75]:  
DATA_DIR = "/../Data/"  
  
datasets_agg = {}  
datasets_agg["bureau_ip_ccb_prev_pos_merged_test_tr"] = pd.read_csv(os.getcwd() + DATA_DIR + "bureau_ip_ccb_prev_pos_merged_test_tr.csv")  
bureau_ip_ccb_prev_pos_merged_test = datasets_agg["bureau_ip_ccb_prev_pos_merged_test"]  
  
In [76]:  
bureau_ip_ccb_prev_pos_merged_test = datasets_agg["bureau_ip_ccb_prev_pos_merged_test"]  
  
In [77]:  
bureau_ip_ccb_prev_pos_merged_test.shape  
  
Out[77]: (48744, 897)  
  
In [78]:  
data_test = bureau_ip_ccb_prev_pos_merged_test  
X_Kaggle_test = data_test  
X_Kaggle_test.replace([np.inf, -np.inf], np.nan, inplace=True)  
  
print(X_Kaggle_test.shape)  
  
(48744, 897)  
  
In [79]:  
test_class_scores = grid.predict_proba(X_Kaggle_test)[:, 1]  
  
In [83]:  
submit_df = datasets_agg["application_test_eng"][['SK_ID_CURR']]  
submit_df['TARGET'] = test_class_scores  
submit_df.head()  
  
Out[83]:  
SK_ID_CURR  TARGET  
0 100001  0.085754  
1 100005  0.114168  
2 100013  0.036689  
3 100028  0.028385  
4 100038  0.148769
```

```
In [84]: submit_df.to_csv("submissionWithReducer.csv",index=False)
```

Kaggle Submission for Collinearity Reducer Pipeline

Click on this [link](#)



Decision Tree Pipeline

This below model is trained using the aggregated and feature engineered data and trained using sklearn tree, the DecisionTreeClassifier. The hyperparameters used are max_depth=50,min_samples_split=5 and this was tuned in a different file using a smaller subset of data. The data is passed through the collinearity reducer prior to training.

```
In [92]: y = bureau_ip_ccb_prev_pos_merged['TARGET']
X = bureau_ip_ccb_prev_pos_merged.drop(['SK_ID_CURR', 'TARGET', 'Unnamed: 0',
_, X, _, y = train_test_split(X, y, test_size=0.8, random_state=42, stratify=y)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.5, random_state=42, stratify=y)

X_train.replace([np.inf, -np.inf], np.nan, inplace=True)
X_valid.replace([np.inf, -np.inf], np.nan, inplace=True)
X_test.replace([np.inf, -np.inf], np.nan, inplace=True)

print(f"X train           shape: {X_train.shape}")
print(f"X validation     shape: {X_valid.shape}")
print(f"X test            shape: {X_test.shape}")

X train           shape: (157445, 1434)
X validation     shape: (39362, 1434)
X test            shape: (49202, 1434)
```

```
In [93]: id_col, feat_num, feat_cat, feature = id_num_cat_feature(X, text = False)
```

```
In [94]: ### Collinear Feature Reduction
cr = make_pipeline(
    SimpleImputer(strategy='median'),
    StandardScaler(),
    CollinearityReducer(attribute_names=feat_num, threshold = 0.7, max_iter=100)
)

reduced_feat_num = cr.fit_transform(X_train[feat_num], y_train)

print(f'Reduced numerical column count from {len(feat_num)}... ')
print(f'...to {len(reduced_feat_num)} by collinearity reduction.')
```

Reduced numerical column count from 1418...
...to 897 by collinearity reduction.

```
Out[94]:
```

	Unnamed: 0_x	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_RE/
149542	149542	Revolving_loans	F	N	
243580	243580	Cash_loans	F	N	
69749	69749	Cash_loans	F	N	
199867	199867	Cash_loans	F	Y	
92996	92996	Cash_loans	F	Y	
...
127796	127796	Cash_loans	F	N	
151912	151912	Cash_loans	M	N	
28101	28101	Revolving_loans	F	Y	
305037	305037	Cash_loans	M	Y	
64492	64492	Cash_loans	F	Y	

157445 rows × 1434 columns

```
In [96]:
```

```
X_train = X_train[reduced_feat_num]
```

```
In [130...:
```

```
X_train.shape
```

```
Out[130]: (157445, 897)
```

```
In [98]:
```

```
id_col, feat_num, feat_cat, feature = id_num_cat_feature(X_train, text = Feature)
```

Hyperparameter tuning with Decision Tree led to a optimal max_depth of 50 and min_sample_split of 5. These values were used for the arguments of DecisionTreeClassifier.

```
In [129...]  
from sklearn import tree  
  
num_pipeline = Pipeline([  
    ('selector', DataFrameSelector(reduced_feat_num)),  
    ('imputer', SimpleImputer(strategy="median")),  
    ('std_scaler', StandardScaler())  
])  
  
cat_pipeline = Pipeline([  
    ('imputer', SimpleImputer(strategy='most_frequent')),  
    ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))  
])  
  
data_pipeline = ColumnTransformer(transformers=[  
    ("num_pipeline", num_pipeline, feat_num),  
    ("cat_pipeline", cat_pipeline, feat_cat)],  
    remainder='drop',  
    n_jobs=-1  
)  
  
full_pipeline_with_predictor = Pipeline([  
    ("preparation", data_pipeline),  
    ("DT", tree.DecisionTreeClassifier(max_depth=50,min_samples_split=5))  
])
```

```
In [131...]  
parameters = dict()
```

```
In [132...]  
grid = GridSearchCV(  
    full_pipeline_with_predictor, param_grid = dict(),  
    cv = 3, n_jobs=4, scoring='roc_auc', verbose=2  
)
```

```
In [133...]  
grid.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 1 candidates, totalling 3 fits  
[CV] END ..... total time= 2.  
6min  
[CV] END ..... total time= 2.  
6min  
[CV] END ..... total time= 2.  
6min
```

```
Out[133]: GridSearchCV(cv=3,
                       estimator=Pipeline(steps=[('preparation',
                                                 ColumnTransformer(n_jobs=-1,
                                                                     transformers=[('num_pipeline',
                                                                 Pipeline(steps=[('selector',
                                                                 DataFrameSelector(attribute_names=['Unnamed: '
                                                                 '0_x',
                                                                 'AMT_ANNUITY',
                                                                 'REGION_POPULATION_RELATIVE',
                                                                 'OWN_CAR_AGE',
                                                                 'FLAG_MOBIL',
                                                                 'FLAG_WORK_PHONE',
                                                                 'FLAG_CONT_MOBILE',
                                                                 'FLAG_PHONE',
                                                                 'FLAG_EMAIL',
                                                                 'CNT_FAM_MEMBERS',
                                                                 'RE...',
                                                                 'LANDAREA_AVG',
                                                                 'NONLIVINGAPARTMENTS_AVG',
                                                                 'NONLIVINGAREA_AVG',
                                                                 'TOTALAREA_MODE', ...]),
                                                                 ('cat_pipeline',
                                                                 Pipeline(steps=[('imputer',
                                                                 SimpleImputer(strategy='most_frequent')),
                                                                 ('ohe',
                                                                 OneHotEncoder(handle_unknown='ignore',
                                                                 sparse=False))]),
                                                                 []]]),
                                                 ('DT',
                                                 DecisionTreeClassifier(max_depth=5
0,
                                                                     min_samples_
split=5))]),
                       n_jobs=4, param_grid={}, scoring='roc_auc', verbose=2)
```

In [134]:

```

from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import recall_score
from sklearn.metrics import balanced_accuracy_score

X_valid.replace([np.inf, -np.inf], np.nan, inplace=True)
X_test.replace([np.inf, -np.inf], np.nan, inplace=True)

try:
    expLogTree
except NameError:
    expLogTree = pd.DataFrame(columns=["exp_name",
                                         "hyperparamters",
                                         "Train Acc",
                                         "Valid Acc",
                                         "Test Acc",
                                         "Train AUC",
                                         "Valid AUC",
                                         "Test AUC",
                                         "Recall",
                                         "Balanced Accuracy"
                                         ])
exp_name = f"Baseline_{len(reduced_feat_num)}_features_and_{len(X_train)}_n"
expLogTree.loc[len(expLogTree)] = [f"{exp_name}"] + [f"max_depth=50,min_samp"]
[accuracy_score(y_train, grid.predict(X_train)),
 accuracy_score(y_valid, grid.predict(X_valid)),
 accuracy_score(y_test, grid.predict(X_test)),
 roc_auc_score(y_train, grid.predict_proba(X_train)[:, 1]),
 roc_auc_score(y_valid, grid.predict_proba(X_valid)[:, 1]),
 roc_auc_score(y_test, grid.predict_proba(X_test)[:, 1]),
 recall_score(y_test, grid.predict(X_test), average='weighted'),
 balanced_accuracy_score(y_test, grid.predict(X_test))]
],
4))

expLogTree

```

Out[134]:

	exp_name	hyperparamters	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC
0	Baseline_897_features and 157445_number of sam...	max_depth=50,min_samples_split=5	0.9859	0.8624	0.8662	0.9894	0.463:

Xgboost Pipeline

This below model is trained using the aggregated and feature engineered data and trained using xgboost, the XGBClassifier with a subsample rate of 0.8. This was tuned with a smaller subset of data in a different file. The data is passed through the collinearity reducer prior to training.

```
In [92]:  
y = bureau_ip_ccb_prev_pos_merged['TARGET']  
X = bureau_ip_ccb_prev_pos_merged.drop(['SK_ID_CURR', 'TARGET', 'Unnamed: 0',  
_, X, _, y = train_test_split(X, y, test_size=0.8, random_state=42, stratify=y)  
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)  
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.5, random_state=42, stratify=y)  
  
X_train.replace([np.inf, -np.inf], np.nan, inplace=True)  
X_valid.replace([np.inf, -np.inf], np.nan, inplace=True)  
X_test.replace([np.inf, -np.inf], np.nan, inplace=True)  
  
print(f"X train shape: {X_train.shape}")  
print(f"X validation shape: {X_valid.shape}")  
print(f"X test shape: {X_test.shape}")  
  
X train shape: (157445, 1434)  
X validation shape: (39362, 1434)  
X test shape: (49202, 1434)
```

```
In [93]: id_col, feat_num, feat_cat, feature = id_num_cat_feature(X, text = False)
```

```
In [94]:  
### Collinear Feature Reduction  
cr = make_pipeline(  
    SimpleImputer(strategy='median'),  
    StandardScaler(),  
    CollinearityReducer(attribute_names=feat_num, threshold = 0.7, max_iter=100)  
)  
  
reduced_feat_num = cr.fit_transform(X_train[feat_num], y_train)  
  
print(f'Reduced numerical column count from {len(feat_num)}...')  
print(f'...to {len(reduced_feat_num)} by collinearity reduction.')  
  
Reduced numerical column count from 1418...  
...to 897 by collinearity reduction.
```

```
Out[94]:
```

	Unnamed: 0_x	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_RE/
149542	149542	Revolving_loans	F	N	
243580	243580	Cash_loans	F	N	
69749	69749	Cash_loans	F	N	
199867	199867	Cash_loans	F	Y	
92996	92996	Cash_loans	F	Y	
...
127796	127796	Cash_loans	F	N	
151912	151912	Cash_loans	M	N	
28101	28101	Revolving_loans	F	Y	
305037	305037	Cash_loans	M	Y	
64492	64492	Cash_loans	F	Y	

157445 rows × 1434 columns

```
In [96]:
```

```
X_train = X_train[reduced_feat_num]
```

```
In [ ]:
```

```
X_train.shape
```

```
Out[ ]:
```

```
(157445, 897)
```

```
In [98]:
```

```
id_col, feat_num, feat_cat, feature = id_num_cat_feature(X_train, text = Feature)
```

Hyperparameter tuning was done in a separate file and that led to a optimal subsample rate of 0.8 and this value was used for xgboost.

```
In [115...  
#!/usr/bin/env python  
#> pip install xgboost  
import xgboost as xgb  
  
num_pipeline = Pipeline([  
    ('selector', DataFrameSelector(reduced_feat_num)),  
    ('imputer', SimpleImputer(strategy="median")),  
    ('std_scaler', StandardScaler())  
)  
  
cat_pipeline = Pipeline([  
    ('imputer', SimpleImputer(strategy='most_frequent')),  
    ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))  
)  
  
data_pipeline = ColumnTransformer(transformers=[  
    ("num_pipeline", num_pipeline, feat_num),  
    ("cat_pipeline", cat_pipeline, feat_cat)],  
    remainder='drop',  
    n_jobs=-1  
)  
  
full_pipeline_with_predictor = Pipeline([  
    ("preparation", data_pipeline),  
    # ('pca', PCA()),  
    ("xgb", xgb.XGBClassifier(subsample = 0.8))  
)
```

```
In [117...  
parameters = dict()
```

```
In [118...  
grid = GridSearchCV(  
    full_pipeline_with_predictor, param_grid = dict(),  
    cv = 3, n_jobs=4, scoring='roc_auc', verbose=2  
)
```

```
In [119...  
grid.fit(X_train, y_train)
```

```
Fitting 3 folds for each of 1 candidates, totalling 3 fits  
[CV] END ..... total time= 6.  
2min  
[CV] END ..... total time= 6.  
2min  
[CV] END ..... total time= 6.  
2min
```

```
Out[119]: GridSearchCV(cv=3,
                       estimator=Pipeline(steps=[('preparation',
                                                 ColumnTransformer(n_jobs=-1,
                                                                     transformers=[('num_pipeline',
                                                                 Pipeline(steps=[('selector',
                                                                 DataFrameSelector(attribute_names=['Unnamed: '
                                                                 '0_x',
                                                                 'AMT_ANNUITY',
                                                                 'REGION_POPULATION_RELATIVE',
                                                                 'OWN_CAR_AGE',
                                                                 'FLAG_MOBIL',
                                                                 'FLAG_WORK_PHONE',
                                                                 'FLAG_CONT_MOBILE',
                                                                 'FLAG_PHONE',
                                                                 'FLAG_EMAIL',
                                                                 'CNT_FAM_MEMBERS',
                                                                 'RE...'),
                                                                interaction_constraint),
                                                                ts=None,
                                                                learning_rate=None,
                                                                max_bin=None,
                                                                max_cat_to_onehot=None,
                                                                max_delta_step=None,
                                                                max_depth=None,
                                                                max_leaves=None,
                                                                min_child_weight=None,
                                                                missing='nan',
                                                                monotone_constraints=None,
                                                                n_estimators=100,
                                                                n_jobs=None,
                                                                num_parallel_tree=None,
                                                                predictor=None,
                                                                random_state=None,
                                                                reg_alpha=None,
                                                                reg_lambda=None,
...))]),
                       n_jobs=4, param_grid={}, scoring='roc_auc', verbose=2)
```

In [128...]

```

from sklearn.metrics import accuracy_score
from sklearn.metrics import roc_auc_score
from sklearn.metrics import recall_score
from sklearn.metrics import balanced_accuracy_score

X_valid.replace([np.inf, -np.inf], np.nan, inplace=True)
X_test.replace([np.inf, -np.inf], np.nan, inplace=True)

try:
    expLogX
except NameError:
    expLogX = pd.DataFrame(columns=[
        "exp_name",
        "hyperparamters",
        "Train Acc",
        "Valid Acc",
        "Test Acc",
        "Train AUC",
        "Valid AUC",
        "Test AUC",
        "Recall",
        "Balanced Accuracy"
    ])

exp_name = f"Baseline_{len(reduced_feat_num)}_features and {len(X_train)}_number of samples"
expLogX.loc[len(expLogX)] = [f"{exp_name}"] + [f"max_depth=50,min_samples_split=5"]
[accuracy_score(y_train, grid.predict(X_train)),
 accuracy_score(y_valid, grid.predict(X_valid)),
 accuracy_score(y_test, grid.predict(X_test)),
 roc_auc_score(y_train, grid.predict_proba(X_train)[:, 1]),
 roc_auc_score(y_valid, grid.predict_proba(X_valid)[:, 1]),
 roc_auc_score(y_test, grid.predict_proba(X_test)[:, 1]),
 recall_score(y_test, grid.predict(X_test), average='weighted'),
 balanced_accuracy_score(y_test, grid.predict(X_test))]
],
4))

expLogX

```

Out[128]:

	exp_name	hyperparamters	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC
0	Baseline_897_features and 157445_number of samples	max_depth=50,min_samples_split=5	0.9355	0.9181	0.9182	0.9223	0.758

Neural Network Pipelines

The sections below discuss the neural network pipelines and final modelling changes implemented in Phase 4.

Multilayer Perceptron Model Function

In the previous sections, nonparametric and parametric classification algorithms were used in conjunction with the data preparation pre-processing pipeline to predict whether a loan candidate would default on their loan. Although the data preparation is largely the same, the neural network models discussed below implement multi-layer perceptrons to accomplish the prediction task instead of other machine-learning classification algorithms.

The function below is adapted from *Homework 11 Section 9* and integrated into the existing data preparation pipeline, developed in the previous phases, in the subsequent sections. It accomplishes the task of training and reporting outputs for a multi-layer perceptron model by the MLP Learning Procedure:

- Starting at the input layer, we forward propagate the patterns of the training data through the network to generate an output.
- Based on the network's output, we calculate the error that we want to minimize using a cost function that we will describe later.
- We backpropagate the error, find its derivative with respect to each weight in the network, and update the model.

The function is divided into subparts here and shown in the cell below.

Part 1 of the `run_hcdr_model()` function instantiates a neural network model parameterized by the inputs:

- `hidden_layer_neurons` specifies the size and number of hidden layers.
- `opt` specifies the optimization algorithm.
- `epochs` specifies the number of training epochs to run.
- `learning_rate` specifies the learning rate of the optimizer.

Loss functions are used to measure error between the prediction and the target value, indicating how far the algorithm model is from realizing the expected outcome. The neural network uses **cross entropy** as the loss function for classification, the formula for which is and the implementation materialized by `sklearn.nn.CrossEntropyLoss()`.

$$\text{CXE}(\theta) = \left[-\frac{1}{m} \sum_{i=1}^m (y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i)) \right]$$

By default, the model is constructed as a sequential forward-passing neural network composed of $M \times N$ neuron layers. The neurons are perceptrons composed of a linear prediction algorithm coupled with a ReLU (rectified linear unit) activation functions.

Part 2 of `run_hcdr_model()` defines two subfunctions to train and evaluate the instantiated model by iterating over tensor batches fed by the torch data-loader object and applying the MLP learning algorithm.

```
In [4]: #  
# Method to create, define and run a deep neural network model  
#  
def run_hcdr_model(  
    hidden_layer_neurons=[32, 16, 8],  
    opt=optim.SGD,  
    epochs=5,  
    learning_rate=4e-3,  
    return_model=False  
):  
    ### PART 1 ###  
    D_in = X_test.shape[1] # Input layer neurons depend on the input dataset  
    D_out = 2 # Output layer neurons - depend on what you're trying to predict  
  
    str_neurons = [str(h) for h in hidden_layer_neurons]  
    arch_string = f"{D_in}-{'-'.join(str_neurons)}-{D_out}"
```


In [6]:

```
# read-in
DATA_DIR = "/drive/MyDrive/ColabNotebooks/"

ds_names = (
    # "application_test",
    "application_train", "prevapp_agg_data_tr", "bureau_agg_data_trans_untrans",
    "ccb_agg_data_tr", "ip_agg_data_tr", "pos_agg_data_tr"
)

datasets_agg = {}

for ds_name in ds_names:
    print('---')
    print(ds_name)
    datasets_agg[ds_name] = pd.read_csv(os.getcwd() + DATA_DIR + f'{ds_name}.csv')
    datasets_agg[ds_name] = reduce_mem_usage(datasets_agg[ds_name])

    ---
    application_train
    Memory usage of dataframe is 0.28 MB
    Memory usage after optimization is: 0.06 MB
    Decreased by 79.2%
    ---
    prevapp_agg_data_tr
    Memory usage of dataframe is 1.73 MB
    Memory usage after optimization is: 0.46 MB
    Decreased by 73.2%
    ---
    bureau_agg_data_trans_untrans
    Memory usage of dataframe is 0.72 MB
    Memory usage after optimization is: 0.21 MB
    Decreased by 70.6%
    ---
    ccb_agg_data_tr
    Memory usage of dataframe is 0.11 MB
    Memory usage after optimization is: 0.04 MB
    Decreased by 66.6%
    ---
    ip_agg_data_tr
    Memory usage of dataframe is 0.17 MB
    Memory usage after optimization is: 0.06 MB
    Decreased by 63.4%
    ---
    pos_agg_data_tr
    Memory usage of dataframe is 0.34 MB
    Memory usage after optimization is: 0.09 MB
    Decreased by 73.3%
```


In [9]:

```
# deep learning model
torch.manual_seed(0)
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

# create train, validation, and test sets
y = agg_data['TARGET']
X = agg_data.drop(['SK_ID_CURR', 'TARGET'], axis = 1) #drop some features w.

_, X, _, y = train_test_split(X, y, test_size=0.6, random_state=42, stratify=y)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.2, random_state=42, stratify=y)

X_train.replace([np.inf, -np.inf], np.nan, inplace=True)
X_valid.replace([np.inf, -np.inf], np.nan, inplace=True)
X_test.replace([np.inf, -np.inf], np.nan, inplace=True)

print(f"X train           shape: {X_train.shape}")
print(f"X validation     shape: {X_valid.shape}")
print(f"X test            shape: {X_test.shape}")

# determine feature types
id_col, feat_num, feat_cat, feature = id_num_cat_feature(X, text = False)

X train           shape: (118084, 1461)
X validation     shape: (29521, 1461)
X test            shape: (36902, 1461)
```

In [10]:

```
### Collinear Feature Reduction

# reduce numerical features by collinearity reduction

cr = make_pipeline(
    SimpleImputer(strategy='median'),
    StandardScaler(),
    CollinearityReducer(attribute_names=feat_num, threshold = 0.5, max_iter=100))

tic = time.perf_counter()
reduced_feat_num = cr.fit_transform(X_train[feat_num], y_train)
toc = time.perf_counter()

print(f"Collinearity Reduction completed in {toc - tic:.4f} seconds.")
print(f'Reduced numerical column count by {len(reduced_feat_num)/len(feat_num)}')
print(f'From {len(feat_num)} columns to {len(reduced_feat_num)} columns.')
```

Collinearity Reduction completed in 3092.4290 seconds.
Reduced numerical column count by 0.4809688581314879% through collinearity reduction.
From 1445 columns to 695 columns.

In [11]:

```
### Main Pipeline

# Pipeline

num_pipeline = Pipeline([
    ('selector', DataFrameSelector(reduced_feat_num)),
    ('imputer', SimpleImputer(strategy="median")),
    ('std_scaler', StandardScaler())
])

cat_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
])

data_pipeline = ColumnTransformer(
    transformers=[
        ("num_pipeline", num_pipeline, feat_num),
        ("cat_pipeline", cat_pipeline, feat_cat)
    ],
    remainder='drop',
    n_jobs=-1
)

X_train = data_pipeline.fit_transform(X_train)
X_valid = data_pipeline.transform(X_valid) #Transform validation set with the same pipeline
X_test = data_pipeline.transform(X_test) #Transform test set with the same pipeline

y_train = y_train.to_numpy()
y_valid = y_valid.to_numpy()
y_test = y_test.to_numpy()

# convert numpy arrays to tensors
X_train_tensor = torch.from_numpy(X_train)
X_valid_tensor = torch.from_numpy(X_valid)
X_test_tensor = torch.from_numpy(X_test)
y_train_tensor = torch.from_numpy(y_train)
y_valid_tensor = torch.from_numpy(y_valid)
y_test_tensor = torch.from_numpy(y_test)

# create TensorDataset in PyTorch
hcdr_train = torch.utils.data.TensorDataset(X_train_tensor, y_train_tensor)
hcdr_valid = torch.utils.data.TensorDataset(X_valid_tensor, y_valid_tensor)
hcdr_test = torch.utils.data.TensorDataset(X_test_tensor, y_test_tensor)

print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)

# create dataloader
# DataLoader is implemented in PyTorch, which will return an iterator to it
train_batch_size = 96
valid_test_batch_size = 64
trainloader_hcdr = torch.utils.data.DataLoader(hcdr_train, batch_size=train_batch_size)
validloader_hcdr = torch.utils.data.DataLoader(hcdr_valid, batch_size=valid_test_batch_size)
testloader_hcdr = torch.utils.data.DataLoader(hcdr_test, batch_size=valid_test_batch_size)
```

(118084, 835) (118084,) (36902, 835) (36902,)

In [16]:

```
#=====
#   Modify START  #
#=====

(input_dataset) - description of input data: size_set_transformed_columns_C
(hidden_layers_neurons) - A list of the number of neurons in the hidden layer
(opt) - The optimizer function to use: SGD, Adam, etc., DEFAULT: optim.SGD
(epochs) - The total number of epochs to train your model for, DEFAULT: 5
(learning_rate) - The learning rate to take the gradient descent step with
''

input_dataset="0.6_agg_trans+orig_835_CR:0.5-10"
hidden_layer_neurons=[256, 128, 64, 32, 16, 4]
opt=optim.Adamax
epochs=5
learning_rate=4e-3

#=====
#   Modify END  #
#=====

arch_string, train_accuracy, valid_accuracy, test_accuracy, train_auc, valid
hidden_layer_neurons,
opt,
epochs,
learning_rate
)

try: hcdrLog
except : hcdrLog = pd.DataFrame(
    columns=[
        "Dataset",
        "Architecture string",
        "Optimizer",
        "Epochs",
        "Learning Rate",
        "Train accuracy",
        "Valid accuracy",
        "Test accuracy",
        "Train auc",
        "Valid auc",
        "Test auc"
    ]
)
hcdrLog.loc[len(hcdrLog)] = [
    input_dataset,
    arch_string,
    f"{opt}",
    f"{epochs}",
    f"{learning_rate}",
    f"{train_accuracy * 100}%",
    f"{valid_accuracy * 100}%",
    f"{test_accuracy * 100}%",
    f"{train_auc * 100}%",
    f"{valid_auc * 100}%",
    f"{test_auc * 100}%",
]
```

```

-----
Model:
Sequential(
    (0): Linear(in_features=835, out_features=256, bias=True)
    (1): ReLU()
    (2): Linear(in_features=256, out_features=128, bias=True)
    (3): ReLU()
    (4): Linear(in_features=128, out_features=64, bias=True)
    (5): ReLU()
    (6): Linear(in_features=64, out_features=32, bias=True)
    (7): ReLU()
    (8): Linear(in_features=32, out_features=16, bias=True)
    (9): ReLU()
    (10): Linear(in_features=16, out_features=4, bias=True)
    (11): ReLU()
    (12): Linear(in_features=4, out_features=2, bias=True)
)
-----
Epoch 1
----Train Accuracy: 0.918      Validation Accuracy: 0.919
----Train AUC-ROC: 0.725      Validation AUC-ROC: 0.758
Epoch 2
----Train Accuracy: 0.919      Validation Accuracy: 0.919
----Train AUC-ROC: 0.759      Validation AUC-ROC: 0.762
Epoch 3
----Train Accuracy: 0.919      Validation Accuracy: 0.919
----Train AUC-ROC: 0.768      Validation AUC-ROC: 0.755
Epoch 4
----Train Accuracy: 0.92      Validation Accuracy: 0.919
----Train AUC-ROC: 0.773      Validation AUC-ROC: 0.766
Epoch 5
----Train Accuracy: 0.921      Validation Accuracy: 0.919
----Train AUC-ROC: 0.781      Validation AUC-ROC: 0.765
-----
```

Out[16]:

	Dataset	Architecture string	Optimizer	Epochs
0	0.6_agg_trans+orig_835_CR:0.5-10	835-256-128-64-32-16-4-2	<class 'torch.optim.adamax.Adamax'>	5

Neural Network Final Pipeline

The Final Neural Network pipeline is shown below. The data for this pipeline is composed of both the original and transformed aggregated features, totalling 1585 feature columns. This pipeline feature selects the numeric variables using the Collinearity Reducer to reduce the total feature count to 835. The model also implements target class rebalancing. The neural network is composed of 6 hidden layers of perceptron neurons using the `Adagrad` optimizer and cross-entropy (CXE) loss function. Each neuron is composed of a linear objective function and a ReLU (rectified linear unit) activation function. The model received a lower AUC-ROC score than the baseline, only 0.758. However, the model achieved a balanced accuracy score of 0.691 - higher than the collinearity reduced and class rebalanced equivalent to the baseline MLP model (see section 9.2.4.1).

Train Model

In [5]:

```
# read-in
DATA_DIR = "/"

ds_names = (
    # "application_test",
    "application_train", "prevapp_agg_data_tr", "bureau_agg_data_trans_untr",
    "ccb_agg_data_tr", "ip_agg_data_tr", "pos_agg_data_tr"
)

datasets_agg = {}

for ds_name in ds_names:
    print('---')
    print(ds_name)
    datasets_agg[ds_name] = pd.read_csv(os.getcwd() + DATA_DIR + f'{ds_name}.csv')
    datasets_agg[ds_name] = reduce_mem_usage(datasets_agg[ds_name])
```

```

---
application_train
Memory usage of dataframe is 0.28 MB
Memory usage after optimization is: 0.06 MB
Decreased by 79.2%
---
prevapp_agg_data_tr
Memory usage of dataframe is 1.73 MB
Memory usage after optimization is: 0.46 MB
Decreased by 73.2%
---
bureau_agg_data_trans_untrans
Memory usage of dataframe is 0.72 MB
Memory usage after optimization is: 0.21 MB
Decreased by 70.6%
---
ccb_agg_data_tr
Memory usage of dataframe is 0.11 MB
Memory usage after optimization is: 0.04 MB
Decreased by 66.6%
---
ip_agg_data_tr
Memory usage of dataframe is 0.17 MB
Memory usage after optimization is: 0.06 MB
Decreased by 63.4%
---
pos_agg_data_tr
Memory usage of dataframe is 0.34 MB
Memory usage after optimization is: 0.09 MB
Decreased by 73.3%

```

In [6]:

```

# denormalize and clean text
for ds_name in datasets_agg:
    if ds_name == 'application_train':
        agg_data = datasets_agg['application_train'].replace(to_replace='\\s+',
                                                               replace(to_replace='\\-',
                                                               replace(to_replace='\\/',
                                                               replace(to_replace='\\(',
                                                               replace(to_replace='\\)'
                                                               replace(to_replace='\\:
                                                               replace(to_replace='\\,
    else:
        agg_data = agg_data.merge(datasets_agg[ds_name], on='SK_ID_CURR', how='left')

agg_data = agg_data.loc[:,~agg_data.columns.str.startswith('Unnamed: ')]
agg_data = agg_data.loc[:,~agg_data.columns.str.startswith('SK_ID_PREV')]
agg_data = agg_data[
    (agg_data.CODE_GENDER != 'XNA') & (agg_data.NAME_INCOME_TYPE != 'Maternity leave')
]

```

In [8]:

```
# deep learning model
torch.manual_seed(0)
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

# create train, validation, and test sets
y = agg_data['TARGET']
X = agg_data.drop(['SK_ID_CURR', 'TARGET'], axis = 1) #drop some features w.

_, X, _, y = train_test_split(X, y, test_size=0.6, random_state=42, stratify=y)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42, stratify=y)
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.2, random_state=42, stratify=y)

X_train.replace([np.inf, -np.inf], np.nan, inplace=True)
X_valid.replace([np.inf, -np.inf], np.nan, inplace=True)
X_test.replace([np.inf, -np.inf], np.nan, inplace=True)

print(f"X train           shape: {X_train.shape}")
print(f"X validation     shape: {X_valid.shape}")
print(f"X test            shape: {X_test.shape}")

# determine feature types
id_col, feat_num, feat_cat, feature = id_num_cat_feature(X, text = False)

X train           shape: (118080, 1461)
X validation     shape: (29520, 1461)
X test            shape: (36900, 1461)
```

In [9]:

```
### Collinear Feature Reduction

with open ('reduced_feat_num.ob', 'rb') as fp:
    reduced_feat_num = pickle.load(fp)

# reduce numerical features by collinearity reduction

# cr = make_pipeline(
#     SimpleImputer(strategy='median'),
#     StandardScaler(),
#     CollinearityReducer(attribute_names=feat_num, threshold = 0.5, max_iter=10),
# )

# tic = time.perf_counter()
# reduced_feat_num = cr.fit_transform(X_train[feat_num], y_train)
# toc = time.perf_counter()

# print(f"Collinearity Reduction completed in {toc - tic:0.4f} seconds.")
# print(f'Reduced numerical column count by {len(reduced_feat_num)/len(feat_num)}')
print(f'From {len(feat_num)} columns to {len(reduced_feat_num)} columns.')
```

From 1445 columns to 700 columns.

```
In [10]: ### Main Pipeline

# Pipeline

num_pipeline = Pipeline([
    ('selector', DataFrameSelector(reduced_feat_num)),
    ('imputer', SimpleImputer(strategy="median")),
    ('std_scaler', StandardScaler())
])

cat_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
])

data_pipeline = ColumnTransformer(
    transformers=[
        ("num_pipeline", num_pipeline, feat_num),
        ("cat_pipeline", cat_pipeline, feat_cat)
    ],
    remainder='drop',
    n_jobs=-1
)
```

```
In [11]: # prepare data as tensor -> dataloader

X_train = data_pipeline.fit_transform(X_train)
X_valid = data_pipeline.transform(X_valid) #Transform validation set with the same pipeline
X_test = data_pipeline.transform(X_test) #Transform test set with the same pipeline

y_train = y_train.to_numpy()
y_valid = y_valid.to_numpy()
y_test = y_test.to_numpy()
```

```
In [12]: # https://www.analyticsvidhya.com/blog/2020/07/10-techniques-to-deal-with-classification-imbalance

train_imb = pd.DataFrame(np.c_[y_train, X_train])

# class count
# class_count_0, class_count_1 = pd.DataFrame(y_train).value_counts()
class_count_0, class_count_1 = train_imb[0].value_counts()

# Separate class
class_0 = train_imb[train_imb[0] == 0]
class_1 = train_imb[train_imb[0] == 1] # print the shape of the class
print('class 0:', class_0.shape)
print('class 1:', class_1.shape)
```

```
class 0: (108548, 838)
class 1: (9532, 838)
```

In [13]:

```
# sample from class 0 (overrepresented class) the number of values in class
class_0_under = class_0.sample(class_count_1, random_state=42)

test_under = pd.concat([class_0_under, class_1], axis=0)

# this gives both classes equal representation
print("total class of 1 and 0:\n", test_under[0].value_counts()) # plot the
test_under[0].value_counts().plot(kind='bar', title='count (target)')
plt.show()

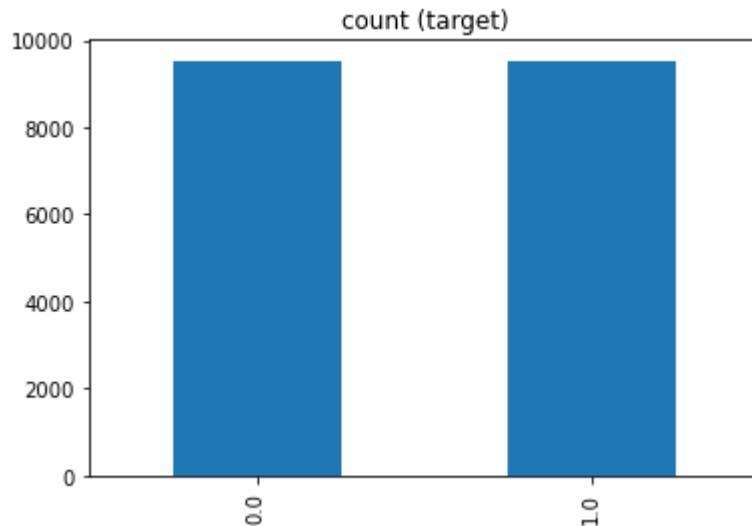
# return arrays to be converted to dataloader tensors
X_train_bal = test_under.iloc[:,1: ].to_numpy()
y_train_bal = test_under.iloc[:,0].to_numpy()
```

total class of 1 and 0:

0.0 9532

1.0 9532

Name: 0, dtype: int64



In [14]:

```
# convert numpy arrays to tensors
X_train_tensor = torch.from_numpy(X_train_bal)
X_valid_tensor = torch.from_numpy(X_valid)
X_test_tensor = torch.from_numpy(X_test)
y_train_tensor = torch.from_numpy(y_train_bal)
y_valid_tensor = torch.from_numpy(y_valid)
y_test_tensor = torch.from_numpy(y_test)

# create TensorDataset in PyTorch
hcdr_train = torch.utils.data.TensorDataset(X_train_tensor, y_train_tensor)
hcdr_valid = torch.utils.data.TensorDataset(X_valid_tensor, y_valid_tensor)
hcdr_test = torch.utils.data.TensorDataset(X_test_tensor, y_test_tensor)

print(X_train.shape, y_train.shape, X_test.shape, y_test.shape)

# create dataloader
# DataLoader is implemented in PyTorch, which will return an iterator to it.
train_batch_size = 96
valid_test_batch_size = 64
trainloader_hcdr = torch.utils.data.DataLoader(hcdr_train, batch_size=train_
validloader_hcdr = torch.utils.data.DataLoader(hcdr_valid, batch_size=valid_
testloader_hcdr = torch.utils.data.DataLoader(hcdr_test, batch_size=valid_t
```

(118080, 837) (118080,) (36900, 837) (36900,)

In [15]:

```
#=====
#   Modify START  #
#=====

(input_dataset) - description of input data: size_set_transformed_columns_C
(hidden_layers_neurons) - A list of the number of neurons in the hidden layer
(opt) - The optimizer function to use: SGD, Adam, etc., DEFAULT: optim.SGD
(epochs) - The total number of epochs to train your model for, DEFAULT: 5
(learning_rate) - The learning rate to take the gradient descent step with
''

input_dataset=f"0.6_agg_orig+trans_bal_{X_train.shape[1]}_CR:0.5-10_ReLU"
hidden_layer_neurons=[128, 64, 32, 16, 8, 4]
opt=optim.Adagrad
epochs=5
learning_rate=5e-3

#=====
#   Modify END  #
#=====

model, arch_string, train_accuracy, valid_accuracy, test_accuracy, train_auc,
hidden_layer_neurons,
opt,
epochs,
learning_rate,
return_model=True
)

try: hcdrLog
except : hcdrLog = pd.DataFrame(
    columns=[
        "Dataset",
        "Architecture string",
        "Optimizer",
        "Epochs",
        "Learning Rate",
        "Train accuracy",
        "Valid accuracy",
        "Test accuracy",
        "Train auc",
        "Valid auc",
        "Test auc",
        "Recall",
        "Balanced Accuracy"
    ]
)

hcdrLog.loc[len(hcdrLog)] = [
    input_dataset,
    arch_string,
    f"{opt}",
    f"{epochs}",
    f"{learning_rate}",
    f"{train_accuracy}",
    f"{valid_accuracy}",
    f"{test_accuracy}",
    f"{train_auc}",
```

```

-----
Model:
Sequential(
    (0): Linear(in_features=837, out_features=128, bias=True)
    (1): ReLU()
    (2): Linear(in_features=128, out_features=64, bias=True)
    (3): ReLU()
    (4): Linear(in_features=64, out_features=32, bias=True)
    (5): ReLU()
    (6): Linear(in_features=32, out_features=16, bias=True)
    (7): ReLU()
    (8): Linear(in_features=16, out_features=8, bias=True)
    (9): ReLU()
    (10): Linear(in_features=8, out_features=4, bias=True)
    (11): ReLU()
    (12): Linear(in_features=4, out_features=2, bias=True)
)
-----
Epoch 1
----Train Accuracy: 0.5 Validation Accuracy: 0.081
----Train AUC-ROC: 0.617 Validation AUC-ROC: 0.67
Epoch 2
----Train Accuracy: 0.637 Validation Accuracy: 0.739
----Train AUC-ROC: 0.715 Validation AUC-ROC: 0.745
Epoch 3
----Train Accuracy: 0.707 Validation Accuracy: 0.701
----Train AUC-ROC: 0.776 Validation AUC-ROC: 0.753
Epoch 4
----Train Accuracy: 0.718 Validation Accuracy: 0.705
----Train AUC-ROC: 0.788 Validation AUC-ROC: 0.751
Epoch 5
----Train Accuracy: 0.726 Validation Accuracy: 0.709
----Train AUC-ROC: 0.797 Validation AUC-ROC: 0.754
-----
```

Out[15]:

	Dataset	Architecture string	Optimizer
0	0.6_agg_orig+trans_bal_837_CR:0.5-10_ReLU	837-128-64-32-16-8-4-2	<class 'torch.optim.adagrad.Adagrad'>

In [18]:

```
# https://stackoverflow.com/questions/33686747/save-a-list-to-a-txt-file

import pickle
with open('reduced_feat_num.ob', 'wb') as fp:
    pickle.dump(reduced_feat_num, fp)

torch.save(model, f"{input_dataset}.pt")
```

Predict Test Data with Trained Model

In [5]:

```
# read-in
DATA_DIR = "/"

test_ds_names = (
    # "application_train",
    "application_test", "prevapp_agg_data_tr", "bureau_agg_data_trans_untrans",
    "ccb_agg_data_tr", "ip_agg_data_tr", "pos_agg_data_tr"
)

datasets_test = {}

for ds_name in test_ds_names:
    print('---')
    print(ds_name)
    datasets_test[ds_name] = pd.read_csv(os.getcwd() + DATA_DIR + f'{ds_name}.csv')
    datasets_test[ds_name] = reduce_mem_usage(datasets_test[ds_name])

    ---
    application_test
    Memory usage of dataframe is 0.04 MB
    Memory usage after optimization is: 0.01 MB
    Decreased by 79.1%
    ---
    prevapp_agg_data_tr
    Memory usage of dataframe is 1.73 MB
    Memory usage after optimization is: 0.46 MB
    Decreased by 73.2%
    ---
    bureau_agg_data_trans_untrans
    Memory usage of dataframe is 0.72 MB
    Memory usage after optimization is: 0.21 MB
    Decreased by 70.6%
    ---
    ccb_agg_data_tr
    Memory usage of dataframe is 0.11 MB
    Memory usage after optimization is: 0.04 MB
    Decreased by 66.6%
    ---
    ip_agg_data_tr
    Memory usage of dataframe is 0.17 MB
    Memory usage after optimization is: 0.06 MB
    Decreased by 63.4%
    ---
    pos_agg_data_tr
    Memory usage of dataframe is 0.34 MB
    Memory usage after optimization is: 0.09 MB
    Decreased by 73.3%
```

In [6]:

```
# denormalize and clean text
for ds_name in datasets_test:
    if ds_name == 'application_test':
        test_data = datasets_test['application_test'].replace(to_replace='\\'
                                                               .replace(to_replace='\\-'
                                                               .replace(to_replace='\\/'
                                                               .replace(to_replace='\\(
                                                               .replace(to_replace='\\)
                                                               .replace(to_replace='\\:
                                                               .replace(to_replace='\\,
    else:
        test_data = test_data.merge(datasets_test[ds_name], on='SK_ID_CURR')

test_data = test_data.loc[:,~test_data.columns.str.startswith('Unnamed:')]
test_data = test_data.loc[:,~test_data.columns.str.startswith('SK_ID_PREV')]
# test_data = test_data[
#     (test_data.CODE_GENDER != 'XNA') & (test_data.NAME_INCOME_TYPE != 'Maternit'
# ]
```

In [7]:

```
# create test sets
X_t = test_data.drop(['SK_ID_CURR'], axis = 1) #drop some features with que.

X_t.replace([np.inf, -np.inf], np.nan, inplace=True)

print(f"test set           shape: {X_t.shape}")

# determine feature types
id_col, feat_num, feat_cat, feature = id_num_cat_feature(X_t, text = False

with open ('reduced_feat_num.ob', 'rb') as fp:
    reduced_feat_num = pickle.load(fp)
```

test set shape: (48744, 1461)

```
In [8]: # Pipeline
```

```
num_pipeline = Pipeline([
    ('selector', DataFrameSelector(reduced_feat_num)),
    ('imputer', SimpleImputer(strategy="median")),
    ('std_scaler', StandardScaler())
])

cat_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
])

data_pipeline = ColumnTransformer(
    transformers=[
        ("num_pipeline", num_pipeline, feat_num),
        ("cat_pipeline", cat_pipeline, feat_cat)
    ],
    remainder='drop',
    n_jobs=-1
)
```

```
In [9]:
```

```
X_t = data_pipeline.fit_transform(X_t) #Transform test set with the same co
# convert numpy arrays to tensors
X_t_tensor = torch.from_numpy(X_t)

print(X_t_tensor.shape)
```

torch.Size([48744, 837])

```
In [10]:
```

```
model = torch.load("0.6_agg_orig+trans_bal_837_CR:0.5-10_ReLU.pt")
model.eval()

t_preds = model(X_t_tensor.float())
t_probs = F.softmax(t_preds, dim=1)[:,1].tolist()
```

```
In [11]:
```

```
submit_df = datasets_test["application_test"][['SK_ID_CURR']]
submit_df['TARGET'] = t_probs
submit_df.head()
```

```
Out[11]:
```

	SK_ID_CURR	TARGET
0	100001	0.275072
1	100005	0.877177
2	100013	0.208998
3	100028	0.347455
4	100038	0.890184

```
In [12]: submit_df.to_csv("submission.csv", index=False)
```

Leakage Analysis

Leakage:

Leakage is when some information in the training model which will not be available during test phase. This leads to overestimating the training result and then poor test predictions when actually predicting the class. There are 2 types of leakage:

1. Data leakage in training data.

2. Leakage in features.

- Leakage in training Data:

There are several places where leakage can happen in training data. It could be done during aggregation step, during standardization, rescaling, using the distribution of whole data set to do transformations, or during estimating the missing values.

Below steps show how we avoided such leakage blunders and hence prevented data leakage:

- Aggregation was done on sub tables which were subsequently rolled up to Train/Test tables. No aggregation was done on Train/Test table which was ultimately used for machine learning.
- No missing values were computed while doing aggregation. Nulls were treated as nulls only.
- Any data manipulation (aggregation, imputation, standardization) was done using pipelines which helps avoid leakage.
- Pipelines were created and all impute tasks, standardization, or any other data aggregation was done using the pipeline step under Gridsearch only.
- Using cross folds under Gridsearch and passing the parameters in the Gridsearch will only evaluate using the fold under consideration. Therefore, in case we took 5 folds, then each fold will have their own computed or imputed values.
- Validation set was used in cross fold for training data result evaluation.
- Test set was solely used for predictions. No test data was considered for anything else before predictions were needed.
- Once we prepared the aggregated data file, split was the 1st step that was done before any pipeline tasks. Then any transformation or other pipeline techniques were performed solely on the training data.
- Collinearity reducer utility was used to remove very low and highly correlated features to avoid overfitting.

- Leakage in features:

This is when a feature is highly correlated to the target that it influences the prediction. This was taken care by checking the correlation of all features with target feature to make sure we don't have a leaking feature.

By looking at the test accuracy, the results were not wildly different which suggested that we

Results and Discussion

Models before EDA

Below are the results from our Pre-EDA Experiment Log.

Experimental results

Please present the results of the various experiments that you conducted. The results should be shown in a table or image. Try to include the different details for each experiment.

Please include code sections when necessary as well as images or any relevant material

Out[87]:

	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC
0	Baseline_14_features	0.9198	0.9192	0.9158	0.7358	0.7358	0.7357
1	Baseline_120_features	0.9200	0.9163	0.9193	0.7478	0.7472	0.7434
2	Baseline_79_features	0.9200	0.9164	0.9195	0.7441	0.7442	0.7406
3	Baseline_79_features with log attributes	0.9200	0.9164	0.9195	0.7443	0.7447	0.7405

The results of our baseline models constructed before EDA and feature aggregation were a good starting point. With only a modicum of complexity, we were able to establish an ROC-AUC score of 74.34%. Below are the details on phase I results:

1. Initial run was considered with 14 features, 7 numerical and 7 categorical. We got training accuracy of 91.98%, test accuracy of 91.58% and AUC score as 73.57% for this very first model.
 - Num cols : 'AMT_INCOME_TOTAL',
'AMT_CREDIT','DAYS_EMPLOYED','DAYS_BIRTH','EXT_SOURCE_1',
'EXT_SOURCE_2','EXT_SOURCE_3',
 - Categorical Cols :
'CODE_GENDER','FLAG_OWN_REALTY','FLAG_OWN_CAR','NAME_CONTRACT_TYPE'
'OCCUPATION_TYPE', 'NAME_INCOME_TYPE'
2. Next we considered all 120 elements form Application train table. Here we got a marginal increase to 92% train accuracy, and 91.93% test accuracy. But there was improvement in AUC score, which was 74.34 % in this iteration.
3. Next, we used a function to remove features which were more than 50% NULL. This led to a list of 79 features. There was very minute change in the test and train score. AUC score reduced from 74.34% to 74.06%. Although reducing the number of features was definitely a boost but we expect same or higher AUC score.
4. Next, we log transformed 2 attributes which were highly skewed and used the features from run 3 from above step. This did not help either, we got same score as in last run up to first decimal point. 1 reason for this is because the attribute we log transformed had a high null percentage which didn't really help the score

Gap Analysis

Gap Analysis:

Until phase 3, we had the best AUC-ROC Kaggle submission score of 76.890%. Team 25 was second to us with a score of 76.47%. Our best model was Logistic Regression with tuned parameters and theirs LGBM. Their model was trained on 338 features whereas we used 1434 features.

With respect to phase 4, at the time of writing, 2 groups have posted the results for the current phase and achieved a Kaggle AUC-ROC score of 76.6 and 74.4. For us, the Kaggle submission score for phase 4 was 76.17%. The results for the held out test was a test AUC of 76.6%. This underperformed phase 3 in terms of Kaggle Public score by 0.00724 and Test AUC by .0068. 6 hidden layers were used to create the model with units sizes [256, 128, 64, 32, 16, 4]. A final layer consists of 2 linear neurons to predict the logits for each class.

We did not find balanced accuracy mentioned by anyone. Balancing data provided us with a good balanced accuracy score but our model suffered from lower test accuracy after balancing.

C	D	E	F	G	H	I
Group	Member names: First name and family name for each team member	member emails	Submission date and time	Phase number	Accuracy on your heldout test set	Use this column for sorting Kaggle submission AUC score (public)
5	Alex Dailey, Austin Slattery, Kai Tan, Xuemei Hu	aldail@iu.edu; ausslat@iu.edu; tankai@iu.edu; xh18@iu.edu	12/13/2022	4.0	0.919	0.766
10	Pragat Wagle, Mark Green, Sahil Dhingra, Kunal Singh	sahdhin@iu.edu, pwagle@iu.edu, singhku@iu.edu, margree@iu.edu	2022-12-13	4.0	0.919	0.7617
1	Jonathan Monser, Greyson Maddox, Miguel Grella, Bryce King	jmonser@iu.edu; gpmaddox@iu.edu; magrella@iu.edu; bk48@iu.edu;	12/12/2022	4.0	0.663	0.744

Conclusion

Supervised learning provides an ML model and evaluation metric to be a part of the decision making process of approving loans. We hypothesized that by improving data quality through feature engineering and experimenting with different machine learning algorithms, a better classification model could be trained compared to the baseline logistic regression model. Using the same data, the resulting AUC-ROC score for the Multilayer Perceptron was below our highest AUC-ROC of 0.7734 for Logistic Regression with a C of 0.01, lasso penalty, and lib-linear, with a test AUC of 76.8. The top feature was `External_Source_1`, `External_Source_2`, and `External_Source_3`. The results show that both Logistic Regression and Multilayer Perceptron models are performing well enough to continue focused experimentation. This model could be improved in the future through more feature engineering to address skewness, using better hardware to train and hyperparameter tune on the full dataset, and rebalancing the target classes in the logistic regression models.

Bibliography

1. <https://towardsdatascience.com/using-the-missingno-python-library-to-identify-and-visualise-missing-data-prior-to-machine-learning-34c8c5b5f009>
2. <https://www.analyticsvidhya.com/blog/2021/04/rapid-fire-eda-process-using-python-for-ml-implementation>
3. <https://www.kaggle.com/code/ekrembayar/homecredit-default-risk-step-by-step-1st-notebook/notebook#8.-Installments-Payments> -some code was directly used from this notebook, as the code was so well written, and was used as a reference for EDA, as explanations were thorough.
4. https://miykael.github.io/blog/2022/advanced_eda/
5. <https://www.kaggle.com/code/rahullalu/hcdr-feature-selection-for-pos-table/notebook>
6. <https://www.analyticsvidhya.com/blog/2020/07/10-techniques-to-deal-with-class-imbalance-in-machine-learning/>
7. Gareth James, Daniela Witten, Trevor Hastie, Robert Tibshirani. An Introduction to Statistical Learning : with Applications in R. Chapter 3. New York :Springer, 2013.

Data Dictionary

	Table	Row	Description	Special
1	application_{train test}.csv	SK_ID_CURR	ID of loan in our sample	
2	application_{train test}.csv	TARGET	"Target variable (1 - client with payment difficulties: he/she had late payment more than X days on at least one of the first Y installments of the loan in our sample, 0 - all other cases)"	
5	application_{train test}.csv	NAME_CONTRACT_TYPE	Identification if loan is cash or revolving	
6	application_{train test}.csv	CODE_GENDER	Gender of the client	
7	application_{train test}.csv	FLAG_OWN_CAR	Flag if the client owns a car	
8	application_{train test}.csv	FLAG_OWN_REALTY	Flag if client owns a house or flat	
9	application_{train test}.csv	CNT_CHILDREN	Number of children the client has	
10	application_{train test}.csv	AMT_INCOME_TOTAL	Income of the client	
11	application_{train test}.csv	AMT_CREDIT	Credit amount of the loan	
12	application_{train test}.csv	AMT_ANNUITY	Loan annuity	
13	application_{train test}.csv	AMT_GOODS_PRICE	For consumer loans it is the price of the goods for which the loan is given	
14	application_{train test}.csv	NAME_TYPE_SUITE	Who was accompanying client when he was applying for the loan	
15	application_{train test}.csv	NAME_INCOME_TYPE	"Clients income type (businessman, working, maternity leave,)"	
16	application_{train test}.csv	NAME_EDUCATION_TYPE	Level of highest education the client achieved	
17	application_{train test}.csv	NAME_FAMILY_STATUS	Family status of the client	
18	application_{train test}.csv	NAME_HOUSING_TYPE	"What is the housing situation of the client (renting, living with parents, ...)	

In []: