Phase 1: Project Proposal and Description

Home Credit Default Risk

Team FP_GroupN_10

Mark Green

margree@iu.edu (mailto:margree@iu.edu)

Pragat Wagle

pwagle@iupui.edu (mailto:pwagle@iupui.edu)

Sahil Dhingra

sahdhin@iu.edu (mailto:sahdhin@iu.edu)

Kunal Singh

singhku@iu.edu (mailto:singhku@iu.edu)

Table of Contents

Note: the sections below Attachment 1 (Section 10) all belong to the original baseline Jupyter notebook provided to the project.

Phase Leader Table

Phase	Leader	Phase Description
1	Mark	**Project Proposal**: Project Plan, Data description, algorithms and metrics, baseline models, pipelines
2	Kunal	**Baseline Report**: EDA, baseline pipeline, feature engineering, hyperparameter tuning, video presentation
3	Pragat	**Finetuning Model**: Additional feature engineering, feature selection, ensemble methods, video presentation
4	Sahil	**Final Report**: Implement neural network, advanced models and loss functions, video presentation

Credit Assignment Table

Phase	Person	Hours	Effort
1	Mark	16	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.
1	Sahil	14	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.
1	Pragat	12	Pragat's goal was to describes 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.
1	Kunal	10	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. Loan applicant data encompassing their demographics, social status, employment status, and previous credit history are re-engineered for machine learning model training. A variety of machine learnt classification models, such as Logistic Regression, are compared using performance metrics like Accuracy, F1-score, and area under the curve receiver operating characteristic ("AUC-ROC"). Hyperparameter tuning using an objective function on the given domain space and an optimization algorithm determines the best model to accomplish the classification task. Four initial baseline logistic regression models (using different input features derived from a single source: current loan applications) report a test accuracy near ~92%. This is expected to improve after integrating more data features and testing more advanced machine learning models - this will be accomplished by adhering to a strategy outlined by the project execution plan herein.

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. **FP_GroupN_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. This model will eventaully be used as a tool to evaluate future loan candidates.

This project proposal discusses the data associated with this task, the machine learning algorithms and pipelines to be developed, and a the results of a simple baseline model using the only `application` data. In addition, the following sections are provided to detail project planning and logistics:

- The Project Execution Plan provides an overview of project objectives, design, timeline, and expectations.
- The Phase Leader Table assigns the leader for each phase of the project; and,
- The Credit Assignment Table describes the phase tasks completed by each team member and the associated level of effort.

Project Execution Plan

From a machine learning perspective, this problem statement can be defined as:

> Determine the optimal binary classification model where the inputs are various features describing the applicant's personal behavioral and financial history to predict a target variable representing the risk of an applicant defaulting on the loan. Will the applicant repay the loan on time, or will the applicant default?

To accomplish this task, the project work has been divided up into four phases:

1. **Phase 1**
    - Consists of project planning and initially exploring the data and examining a baseline pipeline using a selected subset of data.
    - **Success**: Submit the Project Proposal Report and have a good gameplan.
2. **Phase 2**
    - Consists of exploratory data analysis ("EDA") on the full dataset and establishing a baseline pipeline using all of the features with appropriate evluation metrics.
    - **Success**: Submit a Baseline Models Report and update video, and successfully utilize the entire given dataset.
3. **Phase 3**
    - Consists of feature engineering, selection, and hyperparameter tuning of the models to determine the best.
    - **Success**: Submit a Model Tuning Report and update video, and tune our selected model to the best optimization we can achieve.
4. **Phase 4**
    - Consists of selecting an advanced neural network model and concluding the project findings in a final report.
    - **Success**: Submit the Final Report outlining our project accomplishments and implement a neural network model on the data.

Project Design

The tasks for each phase, task completion plans, team member assignments, and project due dates are summarized in Figure 1 below. The initial, middle, and final subdivisions of each phase indicate the "sprint" during which we will focus on a given task. This ensures that tasks are completed in an efficient order, and allows for team members flexibility to dedicate time to a given task based on their own schedule. This figure is will be updated weekly as more objectives are determined.

Entire Team
Pragat Wagle
Mark Green
Sahil Dhingra
Kunal Singh
Unassigned

Figure 1: Project Gantt Chart

Chart describes project tasks and responsibilities for each Phase. Within each phase is an approximate order of operations for completing the tasks, and role assignments where determined.

	Phase 1 *Project Description and Proposal* **Due:** 11/15/2022			**Phase 2** *EDA and Baseline Pipeline* **Due:** 11/29/2022			**Phase 3** *Feature Selection and Model Tuning* **Due:** 12/6/2022			**Phase 4** *Neural Network and Final Report* **Due:** 12/13/2022		
Tasks	***Initial***	***Middle***	***Final***	***Initial***	***Middle***	***Final***	***Initial***	***Middle***	***Final***	***Initial***	***Middle***	***Final***
Project Abstract												
Introduction												
Project Execution Plan												
Data Description												
Machine Learning Algorithms												
Machine Learning Pipelines												
Jupyter Notebook Baseline Modeling												
Conclusion												
Phase Leader Plan												
Credit Assignment Plan												
Overall proposal appearance												
Video and in class presentation												
Team and Plan updates												
Project Description (data and tasks)												
EDA												
Visual EDA												
Feature Engineering												
Hyperparameter Tuning												
Building Model												
Results and discussion of results												
Conclusion												
TBD												
TBD												
TBD												
TBD												
TBD												
TBD												

The Phase Leader for each phase is ultimately responsible for project delivery for that phase. These roles and responsibilities include but are not limited to organizing team members, delegating tasks appropriately ensuring they are completed, and editing, reviwing, and submitting the phase report.

Each team member is responsible for completing the task they are assigned. The Credit Assignment table summarizes the tasks completed by each team member each week. In addition, the group will meet often and work together at a time selected to accomodate teammembers in Eastern US Timezone, Pacific US Timezone, and India Standard Timezone. Group resources that have been established include:

- A Zoom room for video meetings, screen sharing, group collaboration
- A Signal groupchat for less formal communications
- A Github repository to share code and data
- A Sharepoint to share documents and miscellaneous files

Technical Workflow

Within the project phases outlined above, the team will be expected to develop the application. Technical steps to accomplish the challenge of developing this machine learning application are described below:

1. **Workspace Preparation** - Before modeling, we need to import the necessary libraries and the datasets. All need to be imported before we can look at the feature types and number of rows/columns in each file.
2. **Exploratory Data Analysis** - After data importing, we can investigate the data and answer questions like- How many features are present and how are they interlinked? What is the data quality, are there missing values? What are the different data types, are there many categorical features? Is the data imbalanced? And most importantly, are there any obvious patterns between the predictor and response features?

3. **Feature Engineering** - After exploring the data distributions, we can conduct feature engineering to prepare the data for model training. This includes operations like replacing outliers, imputing missing values, one-hot encodingcategorical variables, and rescaling the data. Since there are a number of relational databases, we can use extract, transform, load ("ETL") processes using automated feature Engineering with Feature Tools to connect the datasets. The additional features from these datasets will help improve the results over the base case (logistic regression).
4. **Classifier Models** - Training, Prediction and Comparison - After the dataset is split into training and testing sets, we can correct the data imbalances by undersampling the majority class. Then, we can train different classifier models (For Example: Logistic Regression, Random Forest, Decision Tree, Gaussian Naive Bayes, Support Vector Machines ("SVM"), or Aritifical Neural Networks ("ANN")) and compare their performance on the test data using metrics like Accuracy, F1-score and ROC AUC. After choosing the best classifier, we can use K-fold cross validation to select the best model. This will help us choose parameters that correspond to the best performance without creating a separate validation dataset.
5. **Hyperparameter Tuning** - After choosing the binary classifier, we can tune the hyperparameters for improving the model results through cross-validation, grid search, random search, and Bayesian optimization (Hypertext library). The hyperparameter tuning process will use an objective function on the given domain space, and an optimization algorithm to give the results. The ROC AUC validation scores from all three methods for different iterations can be compared to see trends.

Data Description

The data provided in the applications give insights into an applicant's current demographic and social status. Data from past accounts also give insight into an applicant's past financial behaviors. Notably, the primary key for each sample 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`.

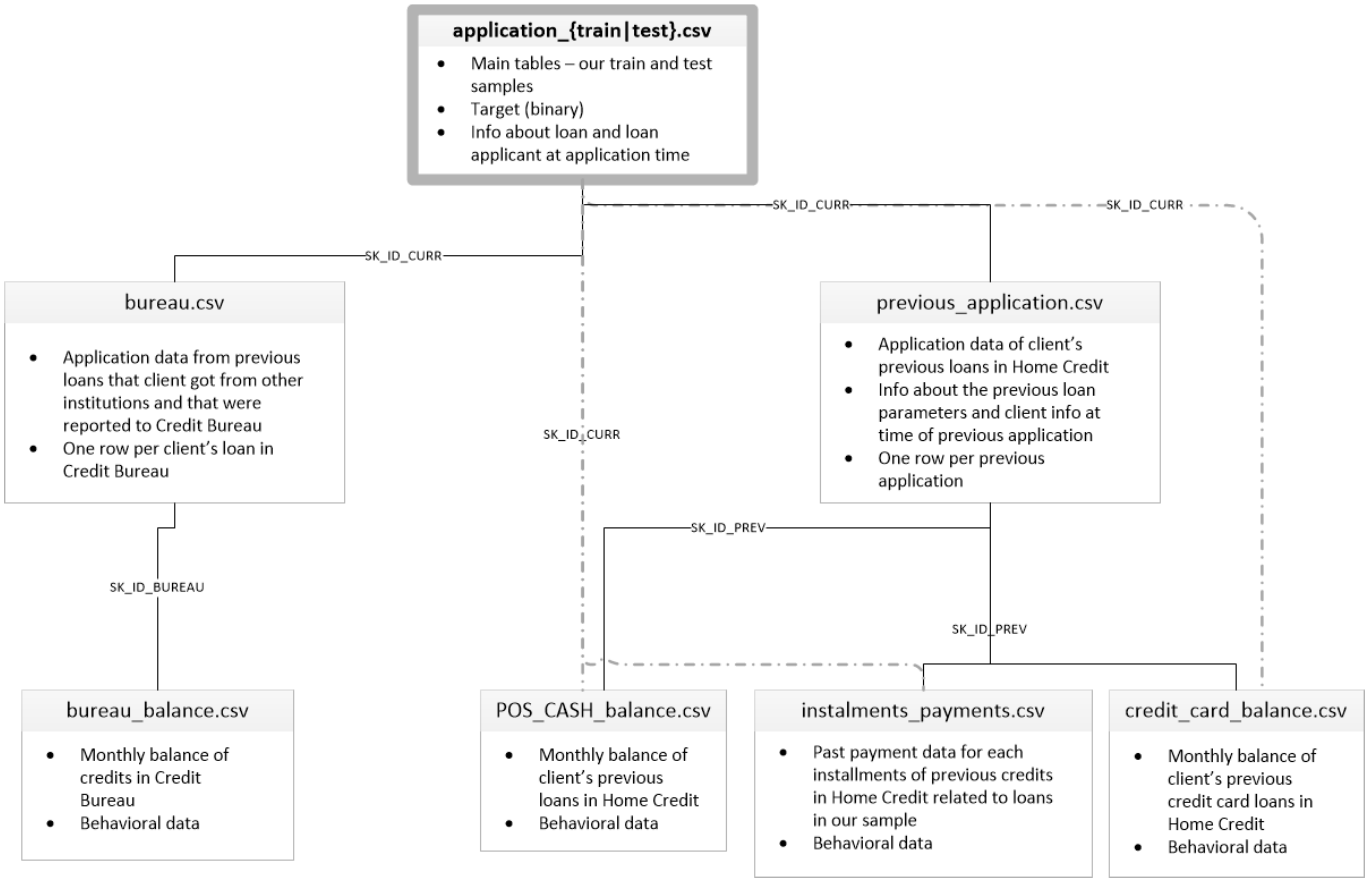


Figure 2: Entity Relation Diagram for Home Credit Applicant Datatables

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.

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.

Machine Learning Algorithms

Of main consideration are algorithms involving binary classification as the applicants will be classified into one of two groups: clients predicted to pay their loans or clients predicted to default. Support Vector Machines, Multilayer Perceptron, and Logistic Regression are the main algorithms being considered, with Logistic Regression as the algorithm used for the baseline model.

Logistic Regression

The `sklearn.linear_model.LogisticRegression` implementation will be used for the baseline model with parameters penalty that can be any one the these `l1`, `l2`, `elasticnet`, a mutli 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 with cross entropy loss.

The objective function for the learning a multinomial logistic regression model (log loss) can be stated as follows:

$$\mathrm{CXE}(\theta) = \left[-\frac{1}{m} \sum_{i=1}^{m} \left(y_i \cdot \log(p_i) + (1 - y_i) \cdot \log(1 - p_i) \right) \right]$$

Regularization helps reduce the risk of overfitting.

- Ridge Regularization (L2):
  - $\mathrm{RidgeCXE}(\theta) = \mathrm{CXE}(\theta) + \lambda \sum_{j=1}^{n} \theta_j^2$
- Lasso Regularization (L1):
  - $\mathrm{LassoCXE}(\theta) = \mathrm{CXE}(\theta) + \lambda \sum_{j=1}^{n} |\theta_j|$
- Elastic Net Regularization (Hybrid L1 + L2):
  - $\mathrm{ElasticCXE}(\theta) = \mathrm{CXE}(\theta) + r\lambda \sum_{j=1}^{n} |\theta_j| + \frac{1-r}{2} \lambda \sum_{j=1}^{n} \theta_j^2$

Support Vector Machines

Another algorithm in consideration is `sklearn.svm.SVC` with hinge loss using a soft margin and hard margin which is determined by the parameter `C`, the regularization parameter. The kernel is another consideration has the options ‘linear’, ‘poly’, ‘rbf’, ‘sigmoid’, ‘precomputed’.

The objective Hinge Loss function for SVMs can be written as:

$$J(w) = \frac{1}{M} \sum_{i=1}^{M} \mathrm{Max}[0, 1 - h_w(\mathbf{x_i})\mathbf{y_i}]^p$$

Multilayer Perceptron

Connecting multiple single neurons to a multilayer feedforward neural network is a special type of fully connected network called a *Multilayer Perceptron* ("MLP"). A multi-layered perceptron is the final algorithm which should be considered. The `PyTorch` library has an extensive deep learning functionality to build out

neural networks, including for MLP.

For this algorithm, the loss function to minimize for a given perceptron neuron is the sum of squared errors:

$$J(\mathbf{w}) = \frac{\mathbf{1}}{\mathbf{2}} \sum_{\mathbf{i}} (\mathbf{y^{(i)}} - \phi(\mathbf{z^{(i)}}))^{\mathbf{2}}$$

Where $y^{(i)}$ is the target class label of a sample $x^{(i)}$ and $\phi(z^{(i)})$ is the activation of the neuron, a linear activation function in the special case of Adaline such that:

$$\phi(z) = z = \sum_{j} w_j x_j = \mathbf{w^T x}$$

Evaluation Metrics

Some metrics that will be used include:

Evaluation Metric	Equation	Description
Accuracy	$\frac{TP + TN}{TP + TN + FP + FN}$	Number of correct predictions divided by the total number of predictions
Precision	$\frac{TP}{TP + FP}$	Number of true positives divided by the number of true & false positives
Recall	$\frac{TP}{TP + FN}$	Number of true positives divided by the number of true positives plus the number of false negatives
F1 Score	$\frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN}$	Harmonic mean of the precision and the recall
AUC ROC	$Sensitivity(\mathrm{TPR}) - (1 - Specificity(\mathrm{FPR}))$	Area under the curve of the receiver operator characteristic, a measure of the true and false positive rates

Machine Learning Pipelines

A generalized machine learning project pipeline is outlined in figure 3 below:

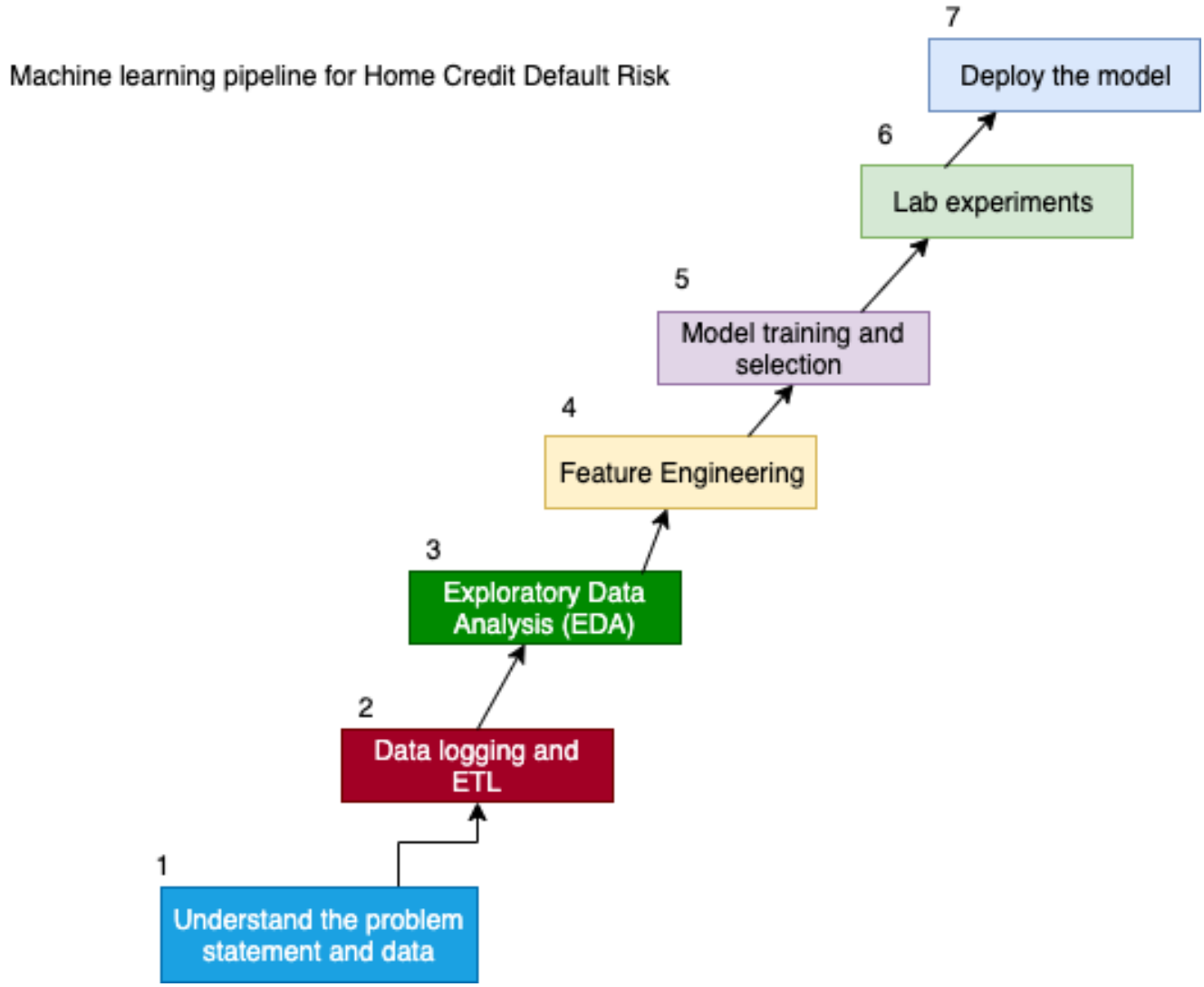


Figure 3: Visualization of key project steps to successfully build a machine learning model application.

These steps are contextualized for this project in the list below:

1. **Understanding the Problem**
    - The problem can be described as “A binary classification problem where the input are various features describing the financial and behavioural history of the loan applicants, in order to predict whether the loan will be repaid or defaulted“.
2. **Data Logging and ETL**
    - The data is segregated in multiple csv files related to eachother 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.

3. **Exploratory Data Analysis**

- After importing the data, we explore the data with the goal of answering questions like:
  - How many features are present?
  - How are they interlinked?
  - How is the data quality?
  - 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.

4. **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.

5. **Model Training and Selection**

- With the prepared data, machine learning algorithms can be trained and evaluated on validation data sets by their performance metrics. At this stage, hyperparameters can be tuned for each machine learning algorithm to select the optimal features and settings for each. For this binary classification task, models such as Logistic Regression, Support Vector Machines, and Neural Networks will be considered.

6. **Lab Experiments**

- This portion of the pipeline consists of making predictions on the held out test set to evaluate final performance of the tuned models. This step is where the final machine learning pipeline is ultimately selected. In the case of this project, it would predict if a client has the ability to pay back a loan and that prediction would be compared against the client's actual loan payback status. The ratio of that would ideally be the test accuracy.

7. **Model Deployment**

- Deploying the model is to deploy the final integrated tested model into a production environment. The model will be able to take some input data, similar to what it was trained on and provide an output or prediction. This can be deployed to some web-service, offline models such as a library, or some edge device or embedded model.

Baseline Model Exploration

The team developed a baseline machine learning pipeline using the starte Jupyter Notebook `HCDR_baseLine_submission_with_numerical_and_cat_features_to_kaggle.ipynb` .

1. First step was to load the data from Kaggle, store data and perform EDA.
2. During EDA, we did data visualization, correlation plots which led us to data pre-processing. We pre-processed the data, log transformed skewed attributes, found attributes with more than 50% null values, and plotted correlation between target and predictor variables.
3. Data was divided into test, validation and test sets.
4. Imputed missing values: for categorical - we imputed with most frequent and for numerical, we imputed with median value.
5. Standardization: Numerical data used standardized using Standard scaler and One Hot encoding was used for categorical data elements.
6. Feature Union: Once Imputed and standardized, we used ColumnTransformer to feed elements into data pipeline.
7. Four baseline models were created:
    - Baseline with all features.
    - Baseline with 14 features, 7 numerical and 7 categorical.
    - Baseline with features which has less than 50% NULLS.
    - Baseline with features which has less than 50% NULLS and log transformed columns.
8. Parameter grid was not used for initial phase but will be used once we move tenet phase.
9. Scoring used in grid search was ‘ROC-AUC’

This process is outlined in the block diagram Figure 4 below:

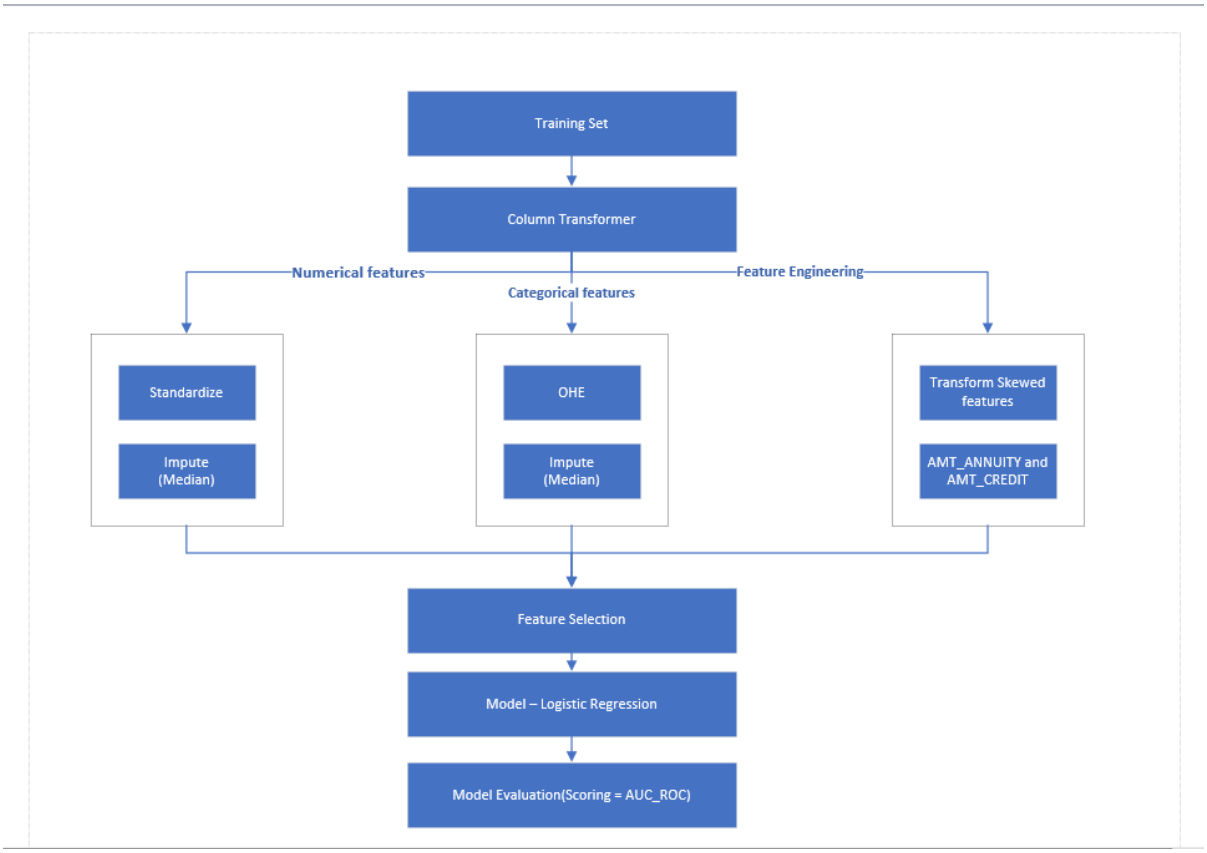


Figure 4: Block diagram outlining preliminary machine learning model pipeline steps.

Following above flow chart we created 4 baseline pipelines based on Logistic regression models:

- Baseline1_all features 120 raw inputs
- Baseline2_all features 14 inputs
- Baseline3_selected features 79 raw inputs
- Baseline4_selected features 79 raw inputs with log features (`log AMT_ANNUITY` , `log AMT_CREDIT`)

Our **All Feature** model has all the 120 features (except `SK_ID_CURR` , `TARGET`) as available in the training set. Then there are 2 sets of models where we are selecting the features that we want to use. 1st one, **Baseline with 14 attributes** has 14 attributes (using 7 numerical and 7 categorical attributes), and another one, **Baseline with 79 attributes** has 79 attributes. For the model with 79 attributes, features having more than 50% null values in the the training data were removed from consideration. The last model tried, **Baseline with 79 features with log features** has 2 attributes which were log transformed. Otherwise, this model uses the same elements from the 79 attributes model.

Numerical features were standardized using `StandardScaler` , and the median was used to fill in the missing values. **Categorical features** were one-hot-encoded to standardize, and missing values were filled using the most frequent values.

The **AUC_ROC** score is used to evaluate our models, and the results are summarized in the table below:

	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

Currently, only default parameters are used for the baseline models. Except for model with 14 features, all other models have training accuracy of 92%. Best validation accuracy was achieved for model 3 and 4. For the scoring, we used AUC score which was 74.34% for the model with all attributes. With more robust feature selection, OHE's, loss functions, and grid paramaters, the score could be improved substantially. A code example for model 2 (with the highest AUC score) is shown below:

```
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
dation in old versions
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
questionable value

# Split the provided training data into training and validationa and test
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, te
st_size=0.2, 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, c
v = 5, n_jobs=-1, scoring='roc_auc')

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"
```

Conclusion

In this phase we have understood the dataset by performing EDA where we have checked for the relationship between Target variable and numerical features,categorical features and also checked for the missing values for features.

Upon that, we build 4 pipelines using logistic regression and we have compared all the training and testing accuracies to see which pipeline works best. Our aim in this phase is to achieve best accuracy.

The baseline model with all features has the highest test AUC score in notebook(0.7434) and in Kaggle submission (0.733). We hope for a better score by doing more feature engineering in the next phase.

For the next phase, we will be performing feature engineering on all the data sets to find the important features, perform dimensionality reduction, SVC, and perform feature engineering.

Attachment 1 - Baseline Jupyter Notebook

Sections below belong to the original `HCDR_baseLine_submission_with_numerical_and_cat_features_to_kaggle.ipynb` document and have been edited by the Team appropriately to develop a baseline model using only the application data. The team made edits in sections 2.4 [12.4], 3.2 [13.2], 3.3 [13.3], 3.4 [13.4],3.5.1 [13.5.1],3.10 [3.10],3.11 [13.11],3.12 [13.12], 7 [17], 8 [18], and 9 [19]. These edits are marked by the following code comment style:

```
### TEAM 10 EDIT ###
### ...description/scope here... ###
### TEAM 10 EDIT ###
```

Home Credit Default Risk (HCDR)

The course project is based on the Home Credit Default Risk (HCDR) Kaggle Competition (https://www.kaggle.com/c/home-credit-default-risk/). The goal of this project is to predict whether or not a client will repay a loan. In order to make sure that people who struggle to get loans due to insufficient or non-existent credit histories have a positive loan experience, Home Credit makes use of a variety of alternative data--including telco and transactional information--to predict their clients' repayment abilities.

Some of the challenges

1. Dataset size
    - (688 meg uncompressed) with millions of rows of data
    - 2.71 Gig of data uncompressed

- Dealing with missing data
- Imbalanced datasets
- Summarizing transaction data

Dataset and how to download

Back ground Home Credit Group

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

Home Credit Group

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

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.

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, Kazahstan, 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 untrustworthly lenders.

Home Credit group has over 29 million customers, total assests of 21 billions Euro, over 160 millions loans, with the majority in Asia and 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.

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 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.

In [1]:
```
# ![alt](home_credit.png "Home credit")
```

Downloading the files via Kaggle API

Create a base directory:

```
DATA_DIR = "../../../Data/home-credit-default-risk"   #same level as cours
e repo in the data directory
```

Please download the project data files and data dictionary and unzip them using either of the following approaches:

1. Click on the `Download` button on the following Data Webpage (https://www.kaggle.com/c/home-credit-default-risk/data) and unzip the zip file to the `BASE_DIR`
2. If you plan to use the Kaggle API, please use the following steps.

In [2]:
```
### TEAM 10 EDIT ###
### MG - changed directory to work with project git repo directory #
##
### TEAM 10 EDIT ###
import os
DATA_DIR =  "../../../Data/home-credit-default-risk"
#DATA_DIR = os.path.join('./ddddd/')
#!mkdir $DATA_DIR
print(DATA_DIR)
```

```
../../../Data/home-credit-default-risk
```

Imports

In [3]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import os
import zipfile
from sklearn.base import BaseEstimator, TransformerMixin
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
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.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline, FeatureUnion
from pandas.plotting import scatter_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
import warnings
warnings.filterwarnings('ignore')
```

In [4]:
```
unzippingReq = False
if unzippingReq: #please modify this code
    zip_ref = zipfile.ZipFile('application_train.csv.zip', 'r')
    zip_ref.extractall('datasets')
    zip_ref.close()
    zip_ref = zipfile.ZipFile('application_test.csv.zip', 'r')
    zip_ref.extractall('datasets')
    zip_ref.close()
    zip_ref = zipfile.ZipFile('bureau_balance.csv.zip', 'r')
    zip_ref.extractall('datasets')
    zip_ref.close()
    zip_ref = zipfile.ZipFile('bureau.csv.zip', 'r')
    zip_ref.extractall('datasets')
    zip_ref.close()
    zip_ref = zipfile.ZipFile('credit_card_balance.csv.zip', 'r')
    zip_ref.extractall('datasets')
    zip_ref.close()
    zip_ref = zipfile.ZipFile('installments_payments.csv.zip', 'r')
    zip_ref.extractall('datasets')
    zip_ref.close()
    zip_ref = zipfile.ZipFile('POS_CASH_balance.csv.zip', 'r')
    zip_ref.extractall('datasets')
    zip_ref.close()
    zip_ref = zipfile.ZipFile('previous_application.csv.zip', 'r')
    zip_ref.extractall('datasets')
    zip_ref.close()
```

Data files overview

Data Dictionary

As part of the data download comes a Data Dictionary. It named `HomeCredit_columns_description.csv`

Application train

In [5]:

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import os
import zipfile
from sklearn.base import BaseEstimator, TransformerMixin
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
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.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline, FeatureUnion
from pandas.plotting import scatter_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
import warnings
warnings.filterwarnings('ignore')

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

datasets={}  # lets store the datasets in a dictionary so we can kee
p track of them easily
ds_name = 'application_train'
datasets[ds_name] = load_data(os.path.join(DATA_DIR, f'{ds_name}.csv
'), ds_name)

datasets['application_train'].shape
```

```
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
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

Out[5]: `(307511, 122)`

Application test

- **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 [6]:
```
ds_name = 'application_test'
datasets[ds_name] = load_data(os.path.join(DATA_DIR, f'{ds_name}.csv
'), 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_RE
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 ...

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 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 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.

In [7]:

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

for ds_name in ds_names:
    datasets[ds_name] = load_data(os.path.join(DATA_DIR, f'{ds_name}.csv'), 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
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_RE
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	CRED
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	

```
bureau_balance: shape is (27299925, 3)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27299925 entries, 0 to 27299924
Data columns (total 3 columns):
 #   Column          Dtype
---  ------          -----
 0   SK_ID_BUREAU    int64
 1   MONTHS_BALANCE  int64
 2   STATUS          object
dtypes: int64(2), object(1)
memory usage: 624.8+ MB
None
```

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

```
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_RECIVABLE                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_ACTU
0	2562384	378907	-6	56.970	135
1	2582071	363914	-1	63975.555	45
2	1740877	371185	-7	31815.225	450
3	1389973	337855	-4	236572.110	225
4	1891521	126868	-1	453919.455	450

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	DA
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	A
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_FUT
0	1803195	182943	-31	48.0	
1	1715348	367990	-33	36.0	
2	1784872	397406	-32	12.0	
3	1903291	269225	-35	48.0	
4	2341044	334279	-35	36.0	

```
CPU times: user 1min 11s, sys: 29.9 s, total: 1min 41s
Wall time: 2min 19s
```

In [8]:
```
for ds_name in datasets.keys():
    print(f'dataset {ds_name:24}: [ {datasets[ds_name].shape[0]:1
0,}, {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

Summary of Application train

In [9]:
```
datasets["application_train"].info()
```

```
<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
```

In [10]:
```
datasets["application_train"].describe() #numerical only features
```

Out[10]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	A
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	3
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	
std	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	
max	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	2

8 rows × 106 columns

In [11]:
```
datasets["application_test"].describe() #numerical only features
```

Out[11]:

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	A
count	48744.000000	48744.000000	4.874400e+04	4.874400e+04	48720.000000	
mean	277796.676350	0.397054	1.784318e+05	5.167404e+05	29426.240209	
std	103169.547296	0.709047	1.015226e+05	3.653970e+05	16016.368315	
min	100001.000000	0.000000	2.694150e+04	4.500000e+04	2295.000000	
25%	188557.750000	0.000000	1.125000e+05	2.606400e+05	17973.000000	
50%	277549.000000	0.000000	1.575000e+05	4.500000e+05	26199.000000	
75%	367555.500000	1.000000	2.250000e+05	6.750000e+05	37390.500000	
max	456250.000000	20.000000	4.410000e+06	2.245500e+06	180576.000000	

8 rows × 105 columns

In [12]:
```
datasets["application_train"].describe(include='all') #look at all c
ategorical and numerical
```

Out[12]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_
count	307511.000000	307511.000000	307511	307511	30
unique	NaN	NaN	2	3	
top	NaN	NaN	Cash loans	F	
freq	NaN	NaN	278232	202448	20
mean	278180.518577	0.080729	NaN	NaN	
std	102790.175348	0.272419	NaN	NaN	
min	100002.000000	0.000000	NaN	NaN	
25%	189145.500000	0.000000	NaN	NaN	
50%	278202.000000	0.000000	NaN	NaN	
75%	367142.500000	0.000000	NaN	NaN	
max	456255.000000	1.000000	NaN	NaN	

11 rows × 122 columns

Numerical and Categorial Data

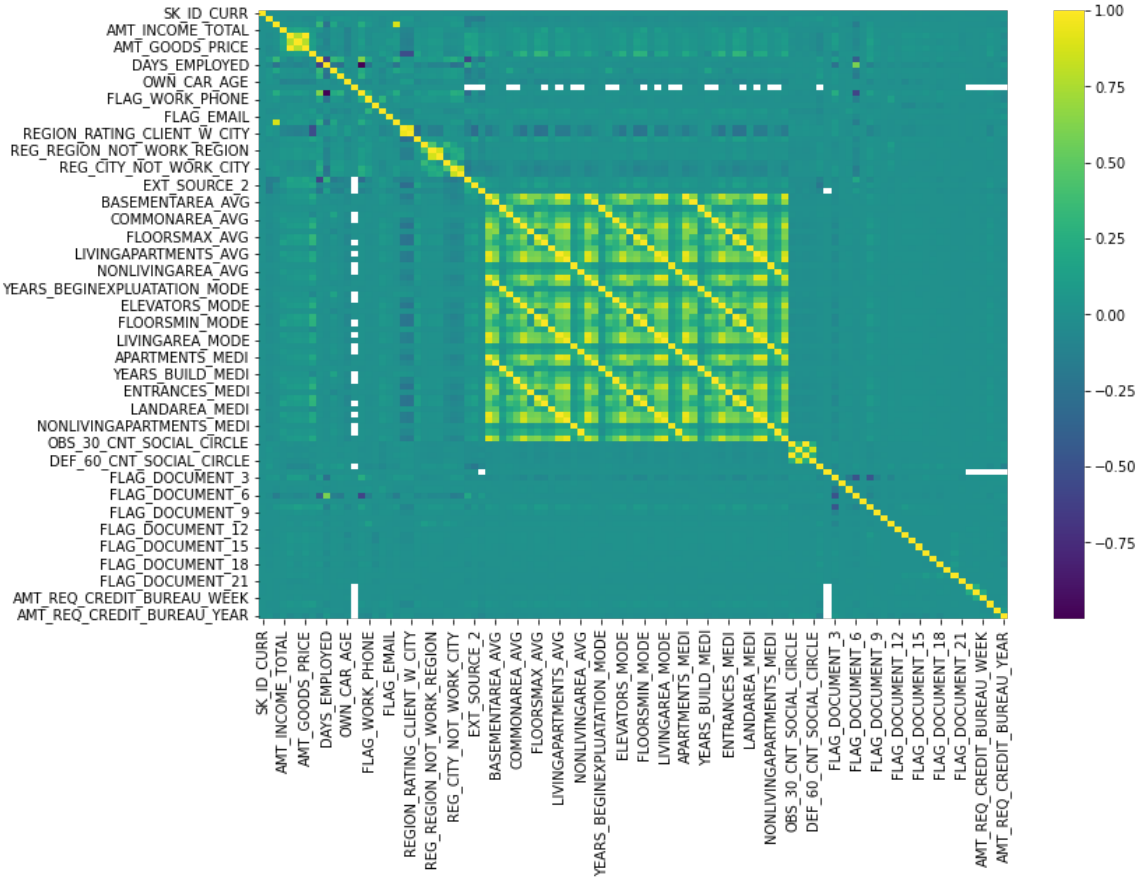
In [13]:
```
### TEAM 10 EDIT ###
### SD - all of section 3.2 [13.2] ###
### TEAM 10 EDIT ###

application_train = datasets["application_train"]
numerical_ix = application_train.select_dtypes(include=['int64', 'fl
oat64']).columns
categorical_ix = application_train.select_dtypes(include=['object',
'bool']).columns
num_features = list(numerical_ix)
cat_features = list(categorical_ix)
print(f"# of numerical   features: {len(numerical_ix)}")
print(f"Numerical   features: {numerical_ix}")
print('--------')
print(f"# of categorical features: {len(categorical_ix)}")
print(f"Categorical features: {categorical_ix}")
```

```
# of numerical   features: 106
Numerical   features: Index(['SK_ID_CURR', 'TARGET', 'CNT_CHILDREN',
'AMT_INCOME_TOTAL',
       'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE',
       'REGION_POPULATION_RELATIVE', 'DAYS_BIRTH', 'DAYS_EMPLOYED',
       ...
       'FLAG_DOCUMENT_18', 'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20',
       'FLAG_DOCUMENT_21', 'AMT_REQ_CREDIT_BUREAU_HOUR',
       'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK',
       'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QRT',
       'AMT_REQ_CREDIT_BUREAU_YEAR'],
      dtype='object', length=106)
--------
# of categorical features: 16
Categorical features: Index(['NAME_CONTRACT_TYPE', 'CODE_GENDER', 'F
LAG_OWN_CAR', 'FLAG_OWN_REALTY',
       'NAME_TYPE_SUITE', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE',
       'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'OCCUPATION_TYPE',
       'WEEKDAY_APPR_PROCESS_START', 'ORGANIZATION_TYPE', 'FONDKAPRE
MONT_MODE',
       'HOUSETYPE_MODE', 'WALLSMATERIAL_MODE', 'EMERGENCYSTATE_MODE
'],
      dtype='object')
```

Correlation- Training

In [14]:
```
plt.figure(figsize=(12,8))
sns.heatmap(datasets["application_train"][numerical_ix].corr(), cmap
="viridis")
```

Out[14]: <AxesSubplot:>



Check Skewness/Distribution for numerical data

Look for skewed column in numerical data but ignore dates, Days,Flags, status, ID's.

Skewness in : AMT_INCOME_TOTAL,AMT_CREDIT,AMT_ANNUITY

In [15]:
```
### TEAM 10 EDIT ###
### SD - histograms in section 3.3 [13.3] ###
### TEAM 10 EDIT ###

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

Out[15]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_A
0	100002	1	0	202500.0	406597.5	
1	100003	0	0	270000.0	1293502.5	
2	100004	0	0	67500.0	135000.0	
3	100006	0	0	135000.0	312682.5	
4	100007	0	0	121500.0	513000.0	
...	...	...	...	...	...	
307506	456251	0	0	157500.0	254700.0	
307507	456252	0	0	72000.0	269550.0	
307508	456253	0	0	153000.0	677664.0	
307509	456254	1	0	171000.0	370107.0	
307510	456255	0	0	157500.0	675000.0	

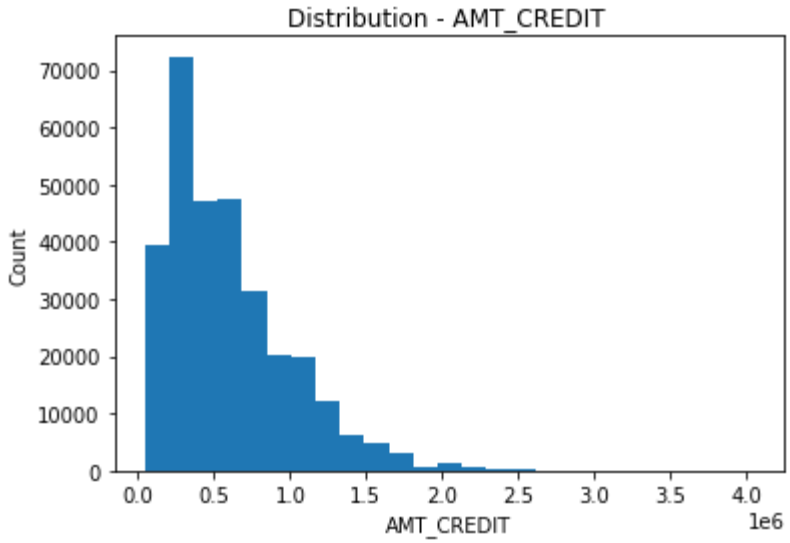
307511 rows × 106 columns

AMT_CREDIT and AMT_ANNUITY looks skewed. we wil do log transformation on these attributes and make them more mormalized.

AMT_CREDIT

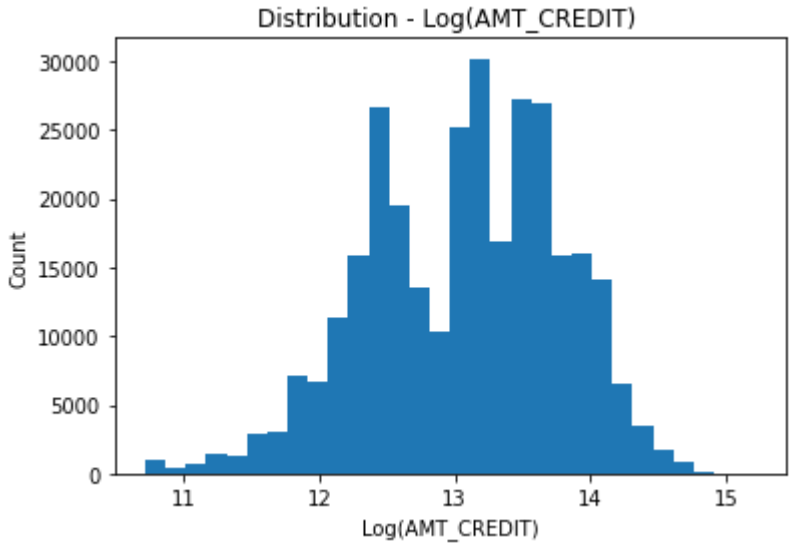
In [16]:
```
plt.hist(datasets["application_train"]['AMT_CREDIT'], bins=25);
plt.xlabel('AMT_CREDIT')
plt.ylabel('Count')
plt.title("Distribution - AMT_CREDIT ")
```

Out[16]: Text(0.5, 1.0, 'Distribution - AMT_CREDIT ')



In [17]:
```
plt.hist(np.log(datasets["application_train"]['AMT_CREDIT']), bins=30);
plt.xlabel('Log(AMT_CREDIT)')
plt.ylabel('Count')
plt.title("Distribution - Log(AMT_CREDIT) ")
```

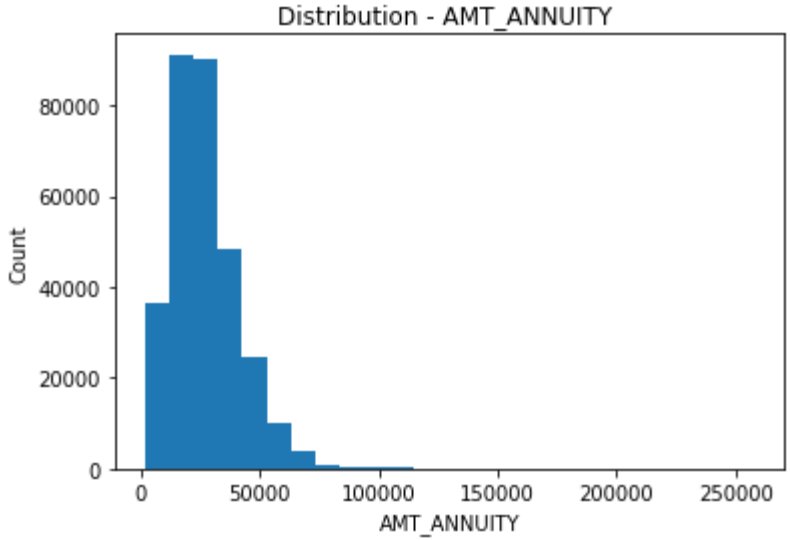
Out[17]: Text(0.5, 1.0, 'Distribution - Log(AMT_CREDIT) ')



AMT_ANNUITY

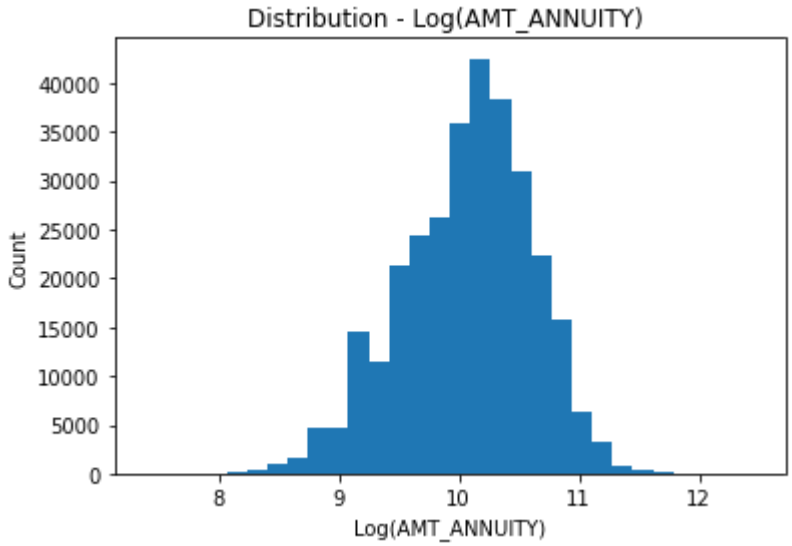
In [18]:
```
plt.hist(datasets["application_train"]['AMT_ANNUITY'], bins=25);
plt.xlabel('AMT_ANNUITY')
plt.ylabel('Count')
plt.title("Distribution - AMT_ANNUITY ")
```

Out[18]: Text(0.5, 1.0, 'Distribution - AMT_ANNUITY ')



In [19]:
```
plt.hist(np.log(datasets["application_train"]['AMT_ANNUITY']), bins=
30);
plt.xlabel('Log(AMT_ANNUITY)')
plt.ylabel('Count')
plt.title("Distribution - Log(AMT_ANNUITY) ")
```

Out[19]: Text(0.5, 1.0, 'Distribution - Log(AMT_ANNUITY) ')



Missing data for application train

In [20]:
```
### TEAM 10 EDIT ###
### SD - deriving features to use in pipelines section 3.4 [13.4] ##
#
### TEAM 10 EDIT ###

percent = (datasets["application_train"].isnull().sum()/datasets["ap
plication_train"].isnull().count()*100).sort_values(ascending = Fals
e).round(2)
sum_missing = datasets["application_train"].isna().sum().sort_values
(ascending = False)
missing_application_train_data  = pd.concat([percent, sum_missing],
axis=1, keys=['Percent', "Train Missing Count"])
missing_application_train_data.head(20)
```

Out[20]:

	Percent	Train Missing Count
COMMONAREA_MEDI	69.87	214865
COMMONAREA_AVG	69.87	214865
COMMONAREA_MODE	69.87	214865
NONLIVINGAPARTMENTS_MODE	69.43	213514
NONLIVINGAPARTMENTS_AVG	69.43	213514
NONLIVINGAPARTMENTS_MEDI	69.43	213514
FONDKAPREMONT_MODE	68.39	210295
LIVINGAPARTMENTS_MODE	68.35	210199
LIVINGAPARTMENTS_AVG	68.35	210199
LIVINGAPARTMENTS_MEDI	68.35	210199
FLOORSMIN_AVG	67.85	208642
FLOORSMIN_MODE	67.85	208642
FLOORSMIN_MEDI	67.85	208642
YEARS_BUILD_MEDI	66.50	204488
YEARS_BUILD_MODE	66.50	204488
YEARS_BUILD_AVG	66.50	204488
OWN_CAR_AGE	65.99	202929
LANDAREA_MEDI	59.38	182590
LANDAREA_MODE	59.38	182590
LANDAREA_AVG	59.38	182590

50% and more nulls

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

In [21]:

```
###TEAM 10 EDIT ###
### SD - evaluating features which are more than 50% NULL ###
### TEAM 10 EDIT ###

nulls_50 = missing_application_train_data[round(missing_application_
train_data['Percent']>50.0)==True]
#nulls_50.index

remove_num_nulls = list(set(nulls_50.index).intersection(set(numeric
al_ix)))
remove_cat_nulls = list(set(nulls_50.index).intersection(set(categor
ical_ix)))
```

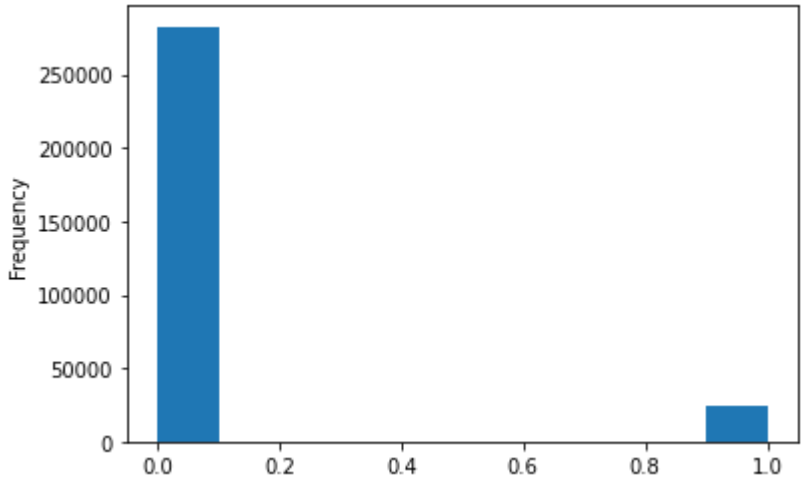
In [22]:
```
percent = (datasets["application_test"].isnull().sum()/datasets["app
lication_test"].isnull().count()*100).sort_values(ascending = Fals
e).round(2)
sum_missing = datasets["application_test"].isna().sum().sort_values
(ascending = False)
missing_application_train_data  = pd.concat([percent, sum_missing],
axis=1, keys=['Percent', "Test Missing Count"])
missing_application_train_data.head(20)
```

Out[22]:

	Percent	Test Missing Count
COMMONAREA_AVG	68.72	33495
COMMONAREA_MODE	68.72	33495
COMMONAREA_MEDI	68.72	33495
NONLIVINGAPARTMENTS_AVG	68.41	33347
NONLIVINGAPARTMENTS_MODE	68.41	33347
NONLIVINGAPARTMENTS_MEDI	68.41	33347
FONDKAPREMONT_MODE	67.28	32797
LIVINGAPARTMENTS_AVG	67.25	32780
LIVINGAPARTMENTS_MODE	67.25	32780
LIVINGAPARTMENTS_MEDI	67.25	32780
FLOORSMIN_MEDI	66.61	32466
FLOORSMIN_AVG	66.61	32466
FLOORSMIN_MODE	66.61	32466
OWN_CAR_AGE	66.29	32312
YEARS_BUILD_AVG	65.28	31818
YEARS_BUILD_MEDI	65.28	31818
YEARS_BUILD_MODE	65.28	31818
LANDAREA_MEDI	57.96	28254
LANDAREA_AVG	57.96	28254
LANDAREA_MODE	57.96	28254

Distribution of the target column

In [23]:
```
datasets["application_train"]['TARGET'].astype(int).plot.hist();
```



This shows that around 8% of people are not able to repay the loans back.

In [24]:
```
datasets["application_train"]['TARGET'].value_counts()/datasets["app
lication_train"]['TARGET'].shape[0]
```

Out[24]:
```
0    0.919271
1    0.080729
Name: TARGET, dtype: float64
```

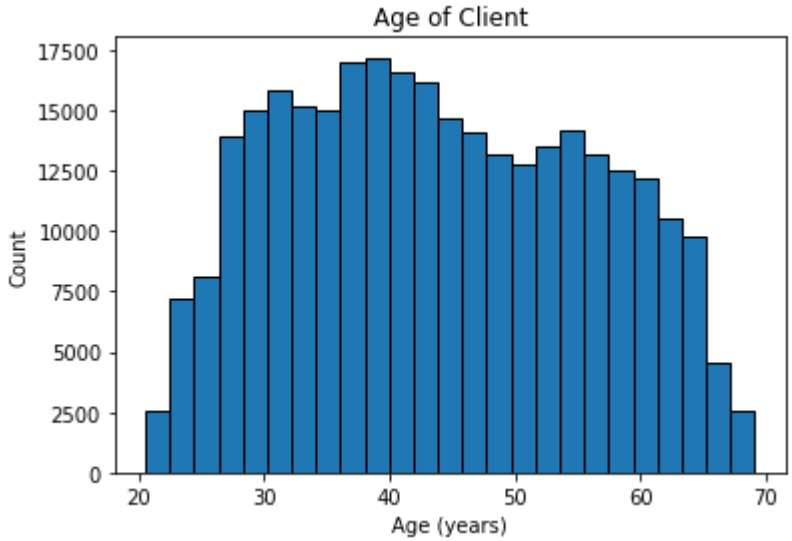
Correlation with the target column

In [25]:
```
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
```

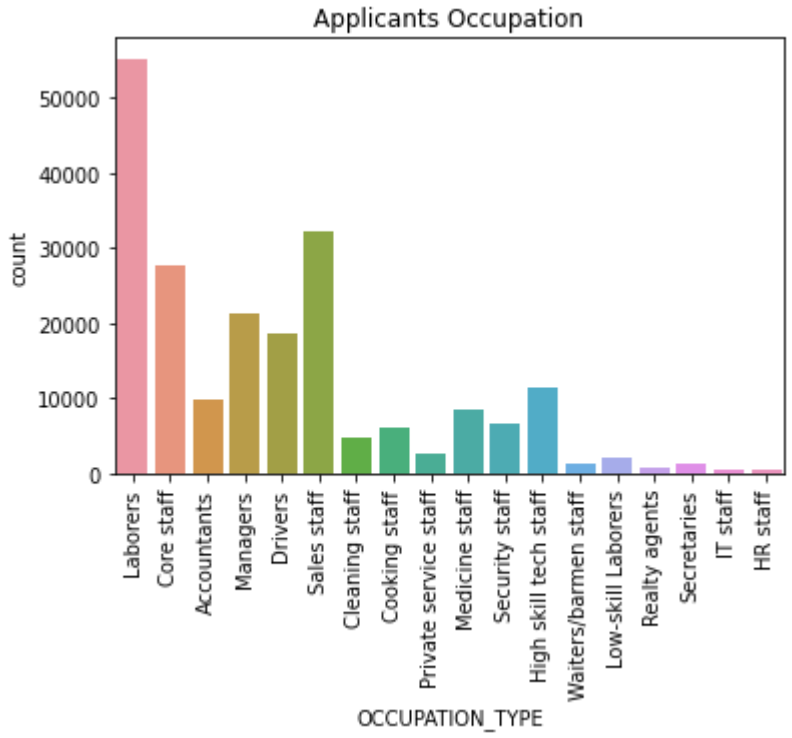
Applicants Age

In [26]:
```
plt.hist(datasets["application_train"]['DAYS_BIRTH'] / -365, edgecol
or = 'k', bins = 25)
plt.title('Age of Client'); plt.xlabel('Age (years)'); plt.ylabel('C
ount');
```



Applicants occupations

In [27]:
```
sns.countplot(x='OCCUPATION_TYPE', data=datasets["application_trai
n"]);
plt.title('Applicants Occupation');
plt.xticks(rotation=90);
```



17% of our applicants are labourers and around 10% are from the Sales. This seems like folks which are from the lower income range which apply for the loan.

In [28]:
```
100* datasets["application_train"]['OCCUPATION_TYPE'].value_counts
()/datasets["application_train"]['OCCUPATION_TYPE'].shape[0]
```

Out[28]:
```
Laborers                 17.946025
Sales staff              10.439301
Core staff                8.965533
Managers                  6.949670
Drivers                   6.049540
High skill tech staff     3.700681
Accountants               3.191105
Medicine staff            2.776161
Security staff            2.185613
Cooking staff             1.933589
Cleaning staff            1.513117
Private service staff     0.862408
Low-skill Laborers        0.680626
Waiters/barmen staff      0.438358
Secretaries               0.424375
Realty agents             0.244219
HR staff                  0.183083
IT staff                  0.171051
Name: OCCUPATION_TYPE, dtype: float64
```

Bureau

In [29]:
```
###TEAM 10 EDIT ###
### SD - EDA Bureau ###
### TEAM 10 EDIT ###


bureau = datasets['bureau']
```

In [30]:
```
percent = (bureau.isnull().sum()/bureau.isnull().count()*100).sort_values(ascending = False).round(2)
sum_missing = bureau.isna().sum().sort_values(ascending = False)
missing_application_train_data  = pd.concat([percent, sum_missing], axis=1, keys=['Percent', "Train Missing Count"])
missing_application_train_data.head(20)
```

Out[30]:

	Percent	Train Missing Count
AMT_ANNUITY	71.47	1226791
AMT_CREDIT_MAX_OVERDUE	65.51	1124488
DAYS_ENDDATE_FACT	36.92	633653
AMT_CREDIT_SUM_LIMIT	34.48	591780
AMT_CREDIT_SUM_DEBT	15.01	257669
DAYS_CREDIT_ENDDATE	6.15	105553
AMT_CREDIT_SUM	0.00	13
CREDIT_ACTIVE	0.00	0
CREDIT_CURRENCY	0.00	0
DAYS_CREDIT	0.00	0
CREDIT_DAY_OVERDUE	0.00	0
SK_ID_BUREAU	0.00	0
CNT_CREDIT_PROLONG	0.00	0
AMT_CREDIT_SUM_OVERDUE	0.00	0
CREDIT_TYPE	0.00	0
DAYS_CREDIT_UPDATE	0.00	0
SK_ID_CURR	0.00	0

In [31]:
```
target_train = datasets['application_train'][['SK_ID_CURR','TARGET']]
bureau_target = datasets['bureau'].merge(target_train,left_on='SK_ID_CURR',right_on='SK_ID_CURR')
```

In [32]:
```
bureau_target.head() ## roll up on days credit. OHE on credit activ
e.
```

Out[32]:

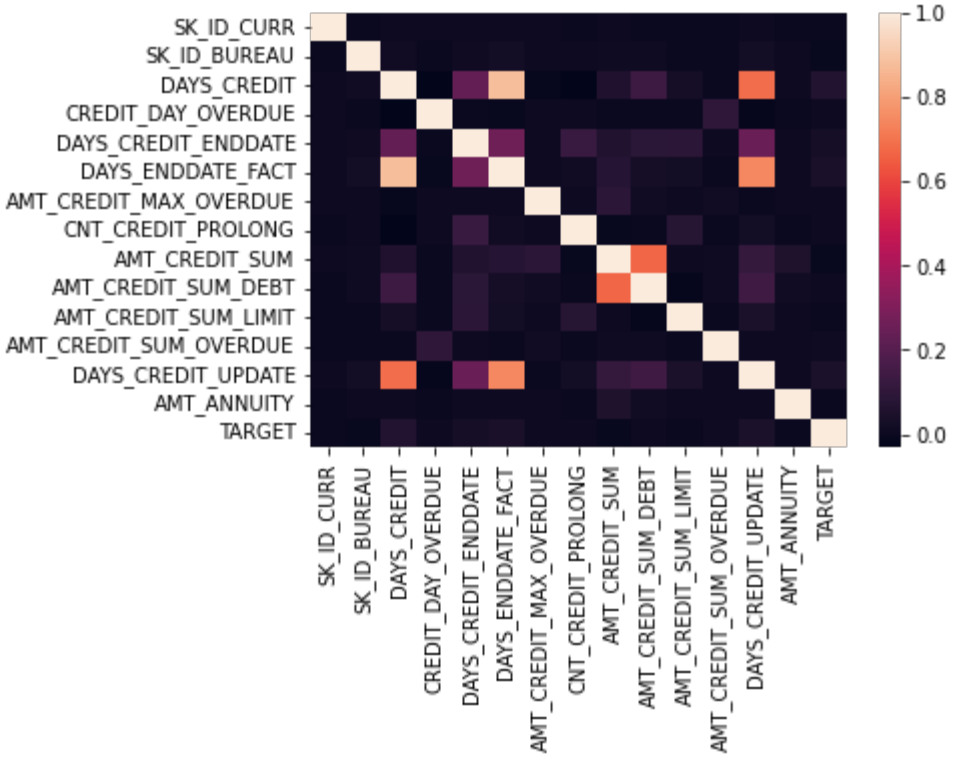
	SK_ID_CURR	SK_ID_BUREAU	CREDIT_ACTIVE	CREDIT_CURRENCY	DAYS_CREDIT	CRED
0	215354	5714462	Closed	currency 1	-497	
1	215354	5714463	Active	currency 1	-208	
2	215354	5714464	Active	currency 1	-203	
3	215354	5714465	Active	currency 1	-203	
4	215354	5714466	Active	currency 1	-629	

In [33]:
```
# This shows that few columns are highly co-related.
#We can decide later on case we want to drop or keep certain column
s.
bureau_target.corr().unstack().sort_values(ascending = False).drop_d
uplicates()
bureau_target.corr()['SK_ID_CURR']
```

Out[33]:
```
SK_ID_CURR                 1.000000
SK_ID_BUREAU               0.000007
DAYS_CREDIT                0.001230
CREDIT_DAY_OVERDUE         0.000293
DAYS_CREDIT_ENDDATE        0.000382
DAYS_ENDDATE_FACT          0.000443
AMT_CREDIT_MAX_OVERDUE     0.001518
CNT_CREDIT_PROLONG        -0.000606
AMT_CREDIT_SUM             0.001449
AMT_CREDIT_SUM_DEBT       -0.000844
AMT_CREDIT_SUM_LIMIT      -0.000496
AMT_CREDIT_SUM_OVERDUE    -0.000189
DAYS_CREDIT_UPDATE         0.000918
AMT_ANNUITY               -0.005983
TARGET                    -0.003024
Name: SK_ID_CURR, dtype: float64
```

In [34]:
```
sns.heatmap(bureau_target.corr())
```

Out[34]: <AxesSubplot:>



Bureau_balance

As per normalized view of tables provided above, this will be linked with bureau and then with application test/train table.

In [35]:
```
###TEAM 10 EDIT ###
### SD - EDA Bureau ###
### TEAM 10 EDIT ###

target_train = datasets['application_train'][['SK_ID_CURR','TARGET
']]
bureau_tgt = datasets['bureau'][['SK_ID_BUREAU','SK_ID_CURR']]
bureau_bal_target = datasets['bureau_balance'].merge(bureau_tgt,left
_on='SK_ID_BUREAU',right_on='SK_ID_BUREAU')
```

In [36]:
```
bureau_bal_target.head()
```

Out[36]:

	SK_ID_BUREAU	MONTHS_BALANCE	STATUS	SK_ID_CURR
0	5715448	0	C	380361
1	5715448	-1	C	380361
2	5715448	-2	C	380361
3	5715448	-3	C	380361
4	5715448	-4	C	380361

In [37]:
```
bureau_bal_target.corr().unstack().sort_values(ascending = False).drop_duplicates()
```

Out[37]:
```
SK_ID_BUREAU    SK_ID_BUREAU      1.000000
                MONTHS_BALANCE    0.010154
                SK_ID_CURR        0.000018
MONTHS_BALANCE  SK_ID_CURR       -0.000495
dtype: float64
```

As seen above, we are not getting much information out of Bureau balance table. We might drop this table all together but that could be done when we'll select the best elements based on KBest or attribute selection.

In [38]:
```
datasets["application_train"]
```

Out[38]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR
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

Credit card balance.

In [39]:
```
###TEAM 10 EDIT ###
### SD - EDA Credit card Balance ###
### TEAM 10 EDIT ###
target_train = datasets['application_train'][['SK_ID_CURR','TARGET
']]
cc_bal_target = datasets['credit_card_balance'].merge(target_train,l
eft_on='SK_ID_CURR',right_on='SK_ID_CURR')
```

In [40]:
```
cc_bal_target.head()
```

Out[40]:

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	AMT_CREDIT_LIMIT_ACTU
0	2582071	363914	-1	63975.555	45
1	2582071	363914	-82	16809.210	67
2	2582071	363914	-84	27577.890	67
3	2582071	363914	-7	65159.235	45
4	2582071	363914	-59	70475.850	67

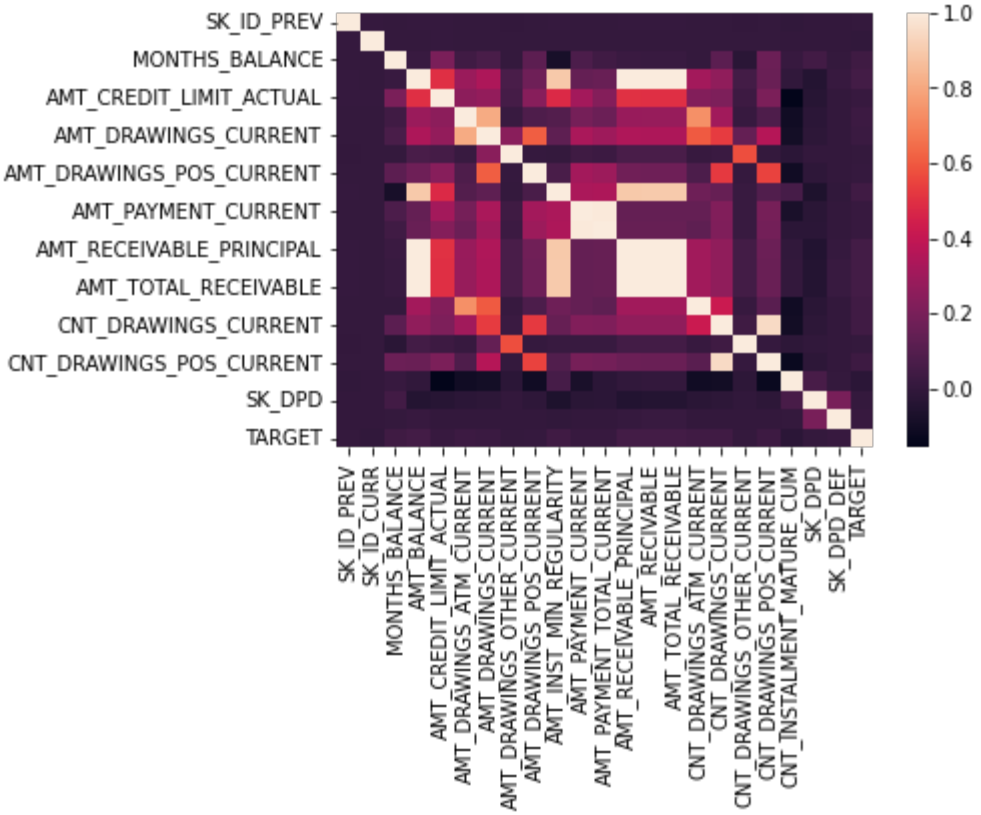
5 rows × 24 columns

In [41]:
```
 # SOme columns are highly co-related, we can make use of eleminatin
g those which are almost corelated.
cc_bal_target.corr().unstack().sort_values(ascending = False).drop_d
uplicates()[:17]
cc_bal_target.corr()['SK_ID_CURR']
```

Out[41]:
```
SK_ID_PREV                  0.005271
SK_ID_CURR                  1.000000
MONTHS_BALANCE              0.002675
AMT_BALANCE                 0.004613
AMT_CREDIT_LIMIT_ACTUAL     0.005961
AMT_DRAWINGS_ATM_CURRENT    0.000811
AMT_DRAWINGS_CURRENT        0.001595
AMT_DRAWINGS_OTHER_CURRENT  0.001514
AMT_DRAWINGS_POS_CURRENT    0.000690
AMT_INST_MIN_REGULARITY     0.004494
AMT_PAYMENT_CURRENT         0.001170
AMT_PAYMENT_TOTAL_CURRENT   0.001561
AMT_RECEIVABLE_PRINCIPAL    0.004714
AMT_RECIVABLE               0.004612
AMT_TOTAL_RECEIVABLE        0.004615
CNT_DRAWINGS_ATM_CURRENT    0.001887
CNT_DRAWINGS_CURRENT        0.003530
CNT_DRAWINGS_OTHER_CURRENT  0.000893
CNT_DRAWINGS_POS_CURRENT    0.003227
CNT_INSTALMENT_MATURE_CUM  -0.000314
SK_DPD                     -0.001321
SK_DPD_DEF                 -0.003515
TARGET                     -0.004617
Name: SK_ID_CURR, dtype: float64
```

In [42]:
```
sns.heatmap(cc_bal_target.corr())
```

Out[42]: <AxesSubplot:>



Dataset questions

Unique record for each SK_ID_CURR

In [43]:
```
datasets.keys()
```

Out[43]:
```
dict_keys(['application_train', 'application_test', 'bureau', 'bureau_balance', 'credit_card_balance', 'installments_payments', 'previous_application', 'POS_CASH_balance'])
```

In [44]:
```
len(datasets["application_train"]["SK_ID_CURR"].unique()) == datasets["application_train"].shape[0]
```

Out[44]:
```
True
```

In [45]:
```
np.intersect1d(datasets["application_train"]["SK_ID_CURR"], datasets["application_test"]["SK_ID_CURR"])
```

Out[45]:
```
array([], dtype=int64)
```

In [46]:
```
datasets["application_test"].shape
```

Out[46]:
```
(48744, 121)
```

In [47]:
```
datasets["application_train"].shape
```

Out[47]:
```
(307511, 122)
```

previous applications for the submission file

The persons in the kaggle submission file have had previous applications in the `previous_application.csv` . 47,800 out 48,744 people have had previous appications.

In [48]:
```
appsDF = datasets["previous_application"]
```

In [49]:
```
len(np.intersect1d(datasets["previous_application"]["SK_ID_CURR"], d
atasets["application_test"]["SK_ID_CURR"]))
```

Out[49]:
```
47800
```

In [50]:
```
print(f"There are  {appsDF.shape[0]:,} previous applications")
```

```
There are  1,670,214 previous applications
```

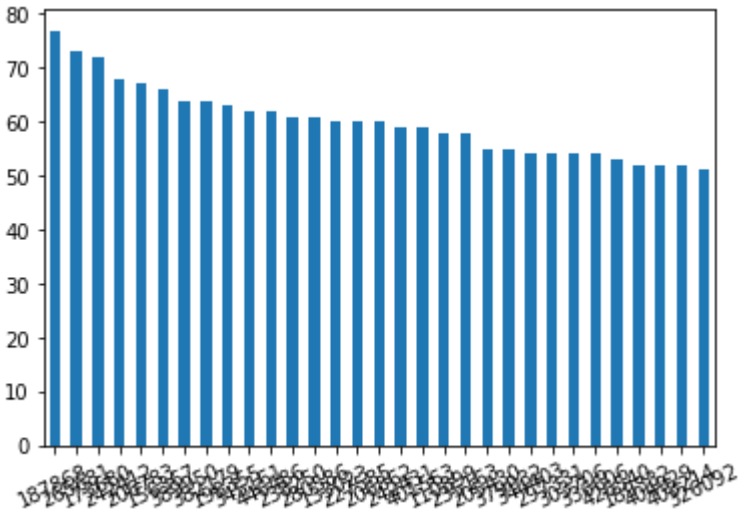
In [51]:
```
# How many entries are there for each month?
prevAppCounts = appsDF['SK_ID_CURR'].value_counts(dropna=False)
#appsDF
prevAppCounts
```

Out[51]:
```
187868    77
265681    73
173680    72
242412    68
206783    67
          ..
135285     1
311960     1
427136     1
241434     1
191629     1
Name: SK_ID_CURR, Length: 338857, dtype: int64
```

In [52]:
```
len(prevAppCounts[prevAppCounts >40])  #more that 40 previous applic
ations
```

Out[52]:
```
101
```

In [53]:
```
prevAppCounts[prevAppCounts >50].plot(kind='bar')
plt.xticks(rotation=25)
plt.show()
```



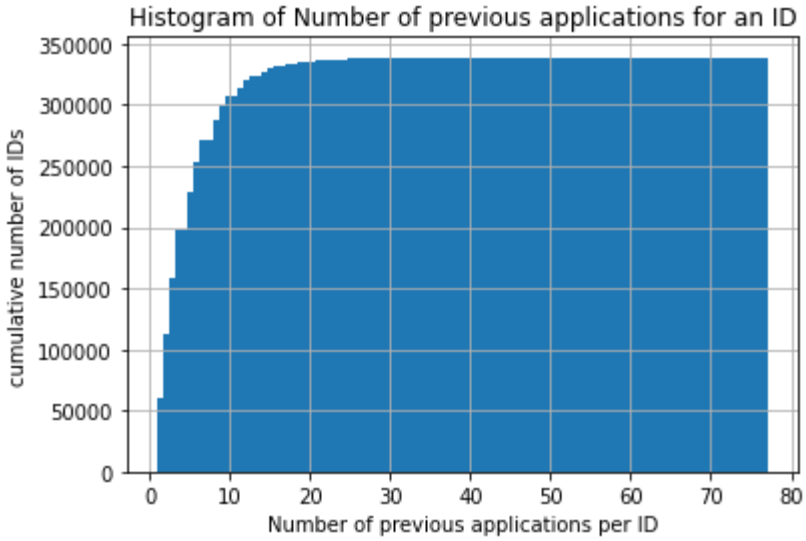
Histogram of Number of previous applications for an ID

In [54]:
```
sum(appsDF['SK_ID_CURR'].value_counts()==1)
```

Out[54]: 60458

In [55]:
```
plt.hist(appsDF['SK_ID_CURR'].value_counts(), cumulative =True, bins
= 100);
plt.grid()
plt.ylabel('cumulative number of IDs')
plt.xlabel('Number of previous applications per ID')
plt.title('Histogram of Number of previous applications for an ID')
```

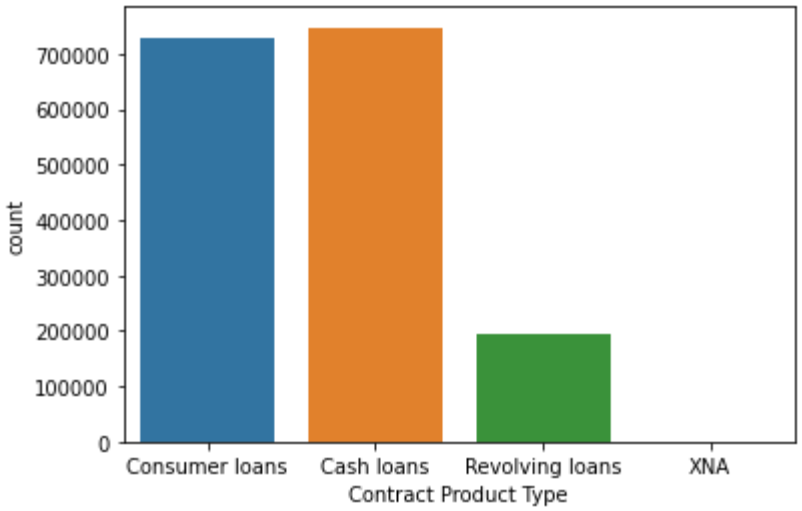
Out[55]:
```
Text(0.5, 1.0, 'Histogram of Number of previous applications for an
ID')
```



added new

In [56]:
```
sns.countplot(x='NAME_CONTRACT_TYPE', data=appsDF);
plt.xlabel('Contract Product Type')
appsDF['NAME_CONTRACT_TYPE'].value_counts()
```

Out[56]:
```
Cash loans         747553
Consumer loans     729151
Revolving loans    193164
XNA                   346
Name: NAME_CONTRACT_TYPE, dtype: int64
```



Can we differentiate applications by low, medium and high previous apps?

* Low = <5 claims (22%)
* Medium = 10 to 39 claims (58%)
* High = 40 or more claims (20%)

In [57]:
```
apps_all = appsDF['SK_ID_CURR'].nunique()
apps_5plus = appsDF['SK_ID_CURR'].value_counts()>=5
#print(apps_5plus)
apps_40plus = appsDF['SK_ID_CURR'].value_counts()>=40
apps_med_plus = 100 - apps_5plus- apps_40plus
print('Percentage with 10 or more previous apps:', np.round(100.*(su
m(apps_5plus)/apps_all),5))
print('Percentage with 11 to 39 no of apps:', np.round(100-(100.*(su
m(apps_5plus)/apps_all))-(100.*(sum(apps_40plus)/apps_all)),5))
print('Percentage with 40 or more previous apps:', np.round(100.*(su
m(apps_40plus)/apps_all),5))
```

```
Percentage with 10 or more previous apps: 41.76895
Percentage with 11 to 39 no of apps: 58.19653
Percentage with 40 or more previous apps: 0.03453
```

In [58]:
```
df1 = pd.DataFrame(prevAppCounts)
df1
```

Out[58]:

	SK_ID_CURR
187868	77
265681	73
173680	72
242412	68
206783	67
...	...
135285	1
311960	1
427136	1
241434	1
191629	1

338857 rows × 1 columns

Joining secondary tables with the primary table

In the case of the HCDR competition (and many other machine learning problems that involve multiple tables in 3NF or not) we need to join these datasets (denormalize) when using a machine learning pipeline. Joining the secondary tables with the primary table will lead to lots of new features about each loan application; these features will tend to be aggregate type features or meta data about the loan or its application. How can we do this when using Machine Learning Pipelines?

Joining `previous_application` with `application_x`

We refer to the `application_train` data (and also `application_test` data also) as the **primary table** and the other files as the **secondary tables** (e.g., `previous_application` dataset). All tables can be joined using the primary key `SK_ID_PREV`.

Let's assume we wish to generate a feature based on previous application attempts. In this case, possible features here could be:

- A simple feature could be the number of previous applications.
- Other summary features of original features such as `AMT_APPLICATION`, `AMT_CREDIT` could be based on average, min, max, median, etc.

To build such features, we need to join the `application_train` data (and also `application_test` data also) with the 'previous_application' dataset (and the other available datasets).

When joining this data in the context of pipelines, different strategies come to mind with various tradeoffs:

1. Preprocess each of the non-application data sets, thereby generating many new (derived) features, and then joining (aka merge) the results with the `application_train` data (the labeled dataset) and with the `application_test` data (the unlabeled submission dataset) prior to processing the data (in a train, valid, test partition) via your machine learning pipeline. [This approach is recommended for this HCDR competition. WHY?]

- Do the joins as part of the transformation steps. [Not recommended here. WHY?]. How can this be done? Will it work?
    - This would be necessary if we had dataset wide features such as IDF (inverse document frequency) which depend on the entire subset of data as opposed to a single loan application (e.g., a feature about the relative amount applied for such as the percentile of the loan amount being applied for).

I want you to think about this section and build on this.

In [59]:
```
pd.DataFrame(prevAppCounts)
```

Out[59]:

	SK_ID_CURR
187868	77
265681	73
173680	72
242412	68
206783	67
...	...
135285	1
311960	1
427136	1
241434	1
191629	1

338857 rows × 1 columns

Roadmap for secondary table processing

1. Transform all the secondary tables to features that can be joined into the main table the application table (labeled and unlabeled)
    - 'bureau', 'bureau_balance', 'credit_card_balance', 'installments_payments',
    - 'previous_application', 'POS_CASH_balance'

- Merge the transformed secondary tables with the primary tables (i.e., the `application_train` data (the labeled dataset) and with the `application_test` data (the unlabeled submission dataset)), thereby leading to X_train, y_train, X_valid, etc.
- Proceed with the learning pipeline using X_train, y_train, X_valid, etc.
- Generate a submission file using the learnt model

agg detour

Aggregate using one or more operations over the specified axis.

For more details see [agg (https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.agg.html)](https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFrame.agg.html)

```
DataFrame.agg(func, axis=0, *args, **kwargs**)
```

Aggregate using one or more operations over the specified axis.

In [60]:
```
appsDF.columns
```

Out[60]:
```
Index(['SK_ID_PREV', 'SK_ID_CURR', 'NAME_CONTRACT_TYPE', 'AMT_ANNUITY',
       'AMT_APPLICATION', 'AMT_CREDIT', 'AMT_DOWN_PAYMENT', 'AMT_GOODS_PRICE',
       'WEEKDAY_APPR_PROCESS_START', 'HOUR_APPR_PROCESS_START',
       'FLAG_LAST_APPL_PER_CONTRACT', 'NFLAG_LAST_APPL_IN_DAY',
       'RATE_DOWN_PAYMENT', 'RATE_INTEREST_PRIMARY',
       'RATE_INTEREST_PRIVILEGED', 'NAME_CASH_LOAN_PURPOSE',
       'NAME_CONTRACT_STATUS', 'DAYS_DECISION', 'NAME_PAYMENT_TYPE',
       'CODE_REJECT_REASON', 'NAME_TYPE_SUITE', 'NAME_CLIENT_TYPE',
       'NAME_GOODS_CATEGORY', 'NAME_PORTFOLIO', 'NAME_PRODUCT_TYPE',
       'CHANNEL_TYPE', 'SELLERPLACE_AREA', 'NAME_SELLER_INDUSTRY',
       'CNT_PAYMENT', 'NAME_YIELD_GROUP', 'PRODUCT_COMBINATION',
       'DAYS_FIRST_DRAWING', 'DAYS_FIRST_DUE', 'DAYS_LAST_DUE_1ST_VERSION',
       'DAYS_LAST_DUE', 'DAYS_TERMINATION', 'NFLAG_INSURED_ON_APPROVAL'],
      dtype='object')
```

Multiple condition expressions in Pandas

So far, both our boolean selections have involved a single condition. You can, of course, have as many conditions as you would like. To do so, you will need to combine your boolean expressions using the three logical operators and, or and not.

Use &, | , ~ Although Python uses the syntax and, or, and not, these will not work when testing multiple conditions with pandas. The details of why are explained here (https://medium.com/dunder-data/selecting-subsets-of-data-in-pandas-39e811c81a0c).

You must use the following operators with pandas:

- & for and
- | for or
- ~ for not

Missing values in prevApps

In [61]:
```
appsDF.isna().sum()
```

Out[61]:
```
SK_ID_PREV                         0
SK_ID_CURR                         0
NAME_CONTRACT_TYPE                 0
AMT_ANNUITY                   372235
AMT_APPLICATION                    0
AMT_CREDIT                         1
AMT_DOWN_PAYMENT              895844
AMT_GOODS_PRICE               385515
WEEKDAY_APPR_PROCESS_START         0
HOUR_APPR_PROCESS_START            0
FLAG_LAST_APPL_PER_CONTRACT        0
NFLAG_LAST_APPL_IN_DAY             0
RATE_DOWN_PAYMENT             895844
RATE_INTEREST_PRIMARY        1664263
RATE_INTEREST_PRIVILEGED     1664263
NAME_CASH_LOAN_PURPOSE             0
NAME_CONTRACT_STATUS               0
DAYS_DECISION                      0
NAME_PAYMENT_TYPE                  0
CODE_REJECT_REASON                 0
NAME_TYPE_SUITE               820405
NAME_CLIENT_TYPE                   0
NAME_GOODS_CATEGORY                0
NAME_PORTFOLIO                     0
NAME_PRODUCT_TYPE                  0
CHANNEL_TYPE                       0
SELLERPLACE_AREA                   0
NAME_SELLER_INDUSTRY               0
CNT_PAYMENT                   372230
NAME_YIELD_GROUP                   0
PRODUCT_COMBINATION              346
DAYS_FIRST_DRAWING            673065
DAYS_FIRST_DUE                673065
DAYS_LAST_DUE_1ST_VERSION     673065
DAYS_LAST_DUE                 673065
DAYS_TERMINATION              673065
NFLAG_INSURED_ON_APPROVAL     673065
dtype: int64
```

In [62]:
```
appsDF.columns
```

Out[62]:
```
Index(['SK_ID_PREV', 'SK_ID_CURR', 'NAME_CONTRACT_TYPE', 'AMT_ANNUITY',
       'AMT_APPLICATION', 'AMT_CREDIT', 'AMT_DOWN_PAYMENT', 'AMT_GOODS_PRICE',
       'WEEKDAY_APPR_PROCESS_START', 'HOUR_APPR_PROCESS_START',
       'FLAG_LAST_APPL_PER_CONTRACT', 'NFLAG_LAST_APPL_IN_DAY',
       'RATE_DOWN_PAYMENT', 'RATE_INTEREST_PRIMARY',
       'RATE_INTEREST_PRIVILEGED', 'NAME_CASH_LOAN_PURPOSE',
       'NAME_CONTRACT_STATUS', 'DAYS_DECISION', 'NAME_PAYMENT_TYPE',
       'CODE_REJECT_REASON', 'NAME_TYPE_SUITE', 'NAME_CLIENT_TYPE',
       'NAME_GOODS_CATEGORY', 'NAME_PORTFOLIO', 'NAME_PRODUCT_TYPE',
       'CHANNEL_TYPE', 'SELLERPLACE_AREA', 'NAME_SELLER_INDUSTRY',
       'CNT_PAYMENT', 'NAME_YIELD_GROUP', 'PRODUCT_COMBINATION',
       'DAYS_FIRST_DRAWING', 'DAYS_FIRST_DUE', 'DAYS_LAST_DUE_1ST_VERSION',
       'DAYS_LAST_DUE', 'DAYS_TERMINATION', 'NFLAG_INSURED_ON_APPROVAL'],
      dtype='object')
```

feature engineering for prevApp table

In [63]:
```
#agg_op_features
```

In [64]:
```
features = ['AMT_ANNUITY', 'AMT_APPLICATION']
print(f"{appsDF[features].describe()}")
agg_ops = ["min", "max", "mean"]
result = appsDF.groupby(["SK_ID_CURR"], as_index=False).agg("mean")
#group by ID
display(result.head())
print("-"*50)
result = appsDF.groupby(["SK_ID_CURR"], as_index=False).agg({'AMT_AN
NUITY' : agg_ops, 'AMT_APPLICATION' : agg_ops})
result.columns = result.columns.map('_'.join)
display(result)
result['range_AMT_APPLICATION'] = result['AMT_APPLICATION_max'] - re
sult['AMT_APPLICATION_min']
print(f"result.shape: {result.shape}")
result[0:10]
```

```
        AMT_ANNUITY  AMT_APPLICATION
count  1.297979e+06     1.670214e+06
mean   1.595512e+04     1.752339e+05
std    1.478214e+04     2.927798e+05
min    0.000000e+00     0.000000e+00
25%    6.321780e+03     1.872000e+04
50%    1.125000e+04     7.104600e+04
75%    2.065842e+04     1.803600e+05
max    4.180581e+05     6.905160e+06
```

	SK_ID_CURR	SK_ID_PREV	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN
0	100001	1.369693e+06	3951.000	24835.50	23787.00	
1	100002	1.038818e+06	9251.775	179055.00	179055.00	
2	100003	2.281150e+06	56553.990	435436.50	484191.00	
3	100004	1.564014e+06	5357.250	24282.00	20106.00	
4	100005	2.176837e+06	4813.200	22308.75	20076.75	

5 rows × 21 columns

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

	SK_ID_CURR_	AMT_ANNUITY_min	AMT_ANNUITY_max	AMT_ANNUITY_mean	AMT_A
0	100001	3951.000	3951.000	3951.000000	
1	100002	9251.775	9251.775	9251.775000	
2	100003	6737.310	98356.995	56553.990000	
3	100004	5357.250	5357.250	5357.250000	
4	100005	4813.200	4813.200	4813.200000	
...	...	...	...	...	
338852	456251	6605.910	6605.910	6605.910000	
338853	456252	10074.465	10074.465	10074.465000	
338854	456253	3973.095	5567.715	4770.405000	
338855	456254	2296.440	19065.825	10681.132500	
338856	456255	2250.000	54022.140	20775.391875	

338857 rows × 7 columns

```
result.shape: (338857, 8)
```

Out[64]:

	SK_ID_CURR_	AMT_ANNUITY_min	AMT_ANNUITY_max	AMT_ANNUITY_mean	AMT_APPLI(
0	100001	3951.000	3951.000	3951.000000	
1	100002	9251.775	9251.775	9251.775000	
2	100003	6737.310	98356.995	56553.990000	
3	100004	5357.250	5357.250	5357.250000	
4	100005	4813.200	4813.200	4813.200000	
5	100006	2482.920	39954.510	23651.175000	
6	100007	1834.290	22678.785	12278.805000	
7	100008	8019.090	25309.575	15839.696250	
8	100009	7435.845	17341.605	10051.412143	
9	100010	27463.410	27463.410	27463.410000	

In [65]:
```
result.isna().sum()
```

Out[65]:
```
SK_ID_CURR_                0
AMT_ANNUITY_min          480
AMT_ANNUITY_max          480
AMT_ANNUITY_mean         480
AMT_APPLICATION_min        0
AMT_APPLICATION_max        0
AMT_APPLICATION_mean       0
range_AMT_APPLICATION      0
dtype: int64
```

In [66]:
```
result
```

Out[66]:

	SK_ID_CURR_	AMT_ANNUITY_min	AMT_ANNUITY_max	AMT_ANNUITY_mean	AMT_A
0	100001	3951.000	3951.000	3951.000000	
1	100002	9251.775	9251.775	9251.775000	
2	100003	6737.310	98356.995	56553.990000	
3	100004	5357.250	5357.250	5357.250000	
4	100005	4813.200	4813.200	4813.200000	
...	...	...	...	...	
338852	456251	6605.910	6605.910	6605.910000	
338853	456252	10074.465	10074.465	10074.465000	
338854	456253	3973.095	5567.715	4770.405000	
338855	456254	2296.440	19065.825	10681.132500	
338856	456255	2250.000	54022.140	20775.391875	

338857 rows × 8 columns

feature transformer for prevApp table

In [67]:

```
# Create aggregate features (via pipeline)
class prevAppsFeaturesAggregater(BaseEstimator, TransformerMixin):
    def __init__(self, features=None): # no *args or **kargs

        self.features = features
        self.agg_ops = ["min", "max", "mean"]



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

    def transform(self, X, y=None):
        #from IPython.core.debugger import Pdb as pdb;    pdb().set_
trace() #breakpoint; dont forget to quit
        result = X.groupby(["SK_ID_CURR"], as_index=False).agg({ft:s
elf.agg_ops for ft in self.features})
        result.columns = result.columns.map(lambda ct: '_'.join([x f
or x in ct if x != '']))

        if 'AMT_APPLICATION' in features:
            result['range_AMT_APPLICATION'] = result['AMT_APPLICATIO
N_max'] - result['AMT_APPLICATION_min']
        return result



from sklearn.pipeline import make_pipeline
def test_driver_prevAppsFeaturesAggregater(df, features):
    print(f"df.shape: {df.shape}\n")
    print(f"df[{features}][0:5]: \n{df[features][0:5]}")
    test_pipeline = make_pipeline(prevAppsFeaturesAggregater(feature
s))
    return(test_pipeline.fit_transform(df))

## Sd - removed name payment type for now.
features = ['AMT_ANNUITY', 'AMT_APPLICATION']
features = ['AMT_ANNUITY',
       'AMT_APPLICATION', 'AMT_CREDIT', 'AMT_DOWN_PAYMENT', 'AMT_GOO
DS_PRICE',
       'RATE_DOWN_PAYMENT', 'RATE_INTEREST_PRIMARY',
       'RATE_INTEREST_PRIVILEGED', 'DAYS_DECISION', #'NAME_PAYMENT_T
YPE',
       'CNT_PAYMENT',
       'DAYS_FIRST_DRAWING', 'DAYS_FIRST_DUE', 'DAYS_LAST_DUE_1ST_VE
RSION',
       'DAYS_LAST_DUE', 'DAYS_TERMINATION']
#features = ['AMT_ANNUITY', 'AMT_APPLICATION']
res = test_driver_prevAppsFeaturesAggregater(appsDF, features)
print(f"HELLO")
print(f"Test driver: \n{res[0:10]}")
print(f"input[features][0:10]: \n{appsDF[0:10]}")

# QUESTION, should we lower case df['OCCUPATION_TYPE'] as Sales staf
```

```
f != 'Sales Staff'? (hint: YES)
```

```
df.shape: (1670214, 37)

df[['AMT_ANNUITY', 'AMT_APPLICATION', 'AMT_CREDIT', 'AMT_DOWN_PAYMEN
T', 'AMT_GOODS_PRICE', 'RATE_DOWN_PAYMENT', 'RATE_INTEREST_PRIMARY',
'RATE_INTEREST_PRIVILEGED', 'DAYS_DECISION', 'CNT_PAYMENT', 'DAYS_FI
RST_DRAWING', 'DAYS_FIRST_DUE', 'DAYS_LAST_DUE_1ST_VERSION', 'DAYS_L
AST_DUE', 'DAYS_TERMINATION']][0:5]:
   AMT_ANNUITY  AMT_APPLICATION  AMT_CREDIT  AMT_DOWN_PAYMENT  \
0     1730.430          17145.0     17145.0               0.0
1    25188.615         607500.0    679671.0               NaN
2    15060.735         112500.0    136444.5               NaN
3    47041.335         450000.0    470790.0               NaN
4    31924.395         337500.0    404055.0               NaN

   AMT_GOODS_PRICE  RATE_DOWN_PAYMENT  RATE_INTEREST_PRIMARY  \
0          17145.0                0.0               0.182832
1         607500.0                NaN                    NaN
2         112500.0                NaN                    NaN
3         450000.0                NaN                    NaN
4         337500.0                NaN                    NaN

   RATE_INTEREST_PRIVILEGED  DAYS_DECISION  CNT_PAYMENT  DAYS_FIRST_
DRAWING  \
0                  0.867336            -73         12.0            3
65243.0
1                       NaN           -164         36.0            3
65243.0
2                       NaN           -301         12.0            3
65243.0
3                       NaN           -512         12.0            3
65243.0
4                       NaN           -781         24.0
NaN

   DAYS_FIRST_DUE  DAYS_LAST_DUE_1ST_VERSION  DAYS_LAST_DUE  DAYS_TE
RMINATION
0           -42.0                      300.0          -42.0
-37.0
1          -134.0                      916.0       365243.0
365243.0
2          -271.0                       59.0       365243.0
365243.0
3          -482.0                     -152.0         -182.0
-177.0
4             NaN                        NaN            NaN
NaN
HELLO
Test driver:
   SK_ID_CURR  AMT_ANNUITY_min  AMT_ANNUITY_max  AMT_ANNUITY_mean  \
0      100001         3951.000         3951.000       3951.000000
1      100002         9251.775         9251.775       9251.775000
2      100003         6737.310        98356.995      56553.990000
3      100004         5357.250         5357.250       5357.250000
4      100005         4813.200         4813.200       4813.200000
```

```
5      100006         2482.920        39954.510      23651.175000
6      100007         1834.290        22678.785      12278.805000
7      100008         8019.090        25309.575      15839.696250
8      100009         7435.845        17341.605      10051.412143
9      100010        27463.410        27463.410      27463.410000

   AMT_APPLICATION_min  AMT_APPLICATION_max  AMT_APPLICATION_mean  \
0              24835.5              24835.5          24835.500000
1             179055.0             179055.0         179055.000000
2              68809.5             900000.0         435436.500000
3              24282.0              24282.0          24282.000000
4                  0.0              44617.5          22308.750000
5                  0.0             688500.0         272203.260000
6              17176.5             247500.0         150530.250000
7                  0.0             450000.0         155701.800000
8              40455.0             110160.0          76741.714286
9             247212.0             247212.0         247212.000000

   AMT_CREDIT_min  AMT_CREDIT_max  AMT_CREDIT_mean  ...  \
0         23787.0         23787.0     23787.000000  ...
1        179055.0        179055.0    179055.000000  ...
2         68053.5       1035882.0    484191.000000  ...
3         20106.0         20106.0     20106.000000  ...
4             0.0         40153.5     20076.750000  ...
5             0.0        906615.0    291695.500000  ...
6         14616.0        284400.0    166638.750000  ...
7             0.0        501975.0    162767.700000  ...
8         38574.0         98239.5     70137.642857  ...
9        260811.0        260811.0    260811.000000  ...

   DAYS_LAST_DUE_1ST_VERSION_min  DAYS_LAST_DUE_1ST_VERSION_max  \
0                         -1499.0                        -1499.0
1                           125.0                          125.0
2                         -1980.0                         -386.0
3                          -694.0                         -694.0
4                          -376.0                         -376.0
5                          -215.0                       365243.0
6                         -2056.0                          346.0
7                         -2341.0                          261.0
8                         -1330.0                          227.0
9                          -769.0                         -769.0

   DAYS_LAST_DUE_1ST_VERSION_mean  DAYS_LAST_DUE_min  DAYS_LAST_DUE_
max  \
0                   -1499.000000            -1619.0            -161
9.0
1                     125.000000              -25.0              -2
5.0
2                   -1004.333333            -1980.0             -53
6.0
3                    -694.000000             -724.0             -72
4.0
4                    -376.000000             -466.0             -46
6.0
```

```
5                   91584.000000            -425.0           36524
3.0
6                    -837.200000           -2056.0           36524
3.0
7                   -1044.500000           -2341.0              -6
9.0
8                    -478.285714           -1330.0           36524
3.0
9                    -769.000000            -769.0             -76
9.0

   DAYS_LAST_DUE_mean  DAYS_TERMINATION_min  DAYS_TERMINATION_max  \
0        -1619.000000                -1612.0                -1612.0
1          -25.000000                  -17.0                  -17.0
2        -1054.333333                -1976.0                 -527.0
3         -724.000000                 -714.0                 -714.0
4         -466.000000                 -460.0                 -460.0
5       182477.500000                 -416.0               365243.0
6        72136.200000                -2041.0               365243.0
7        -1209.500000                -2334.0                  -66.0
8        51666.857143                -1323.0               365243.0
9         -769.000000                 -762.0                 -762.0

   DAYS_TERMINATION_mean  range_AMT_APPLICATION
0           -1612.000000                    0.0
1             -17.000000                    0.0
2           -1047.333333                831190.5
3            -714.000000                    0.0
4            -460.000000                44617.5
5          182481.750000                688500.0
6           72143.800000                230323.5
7            -872.750000                450000.0
8           51672.857143                 69705.0
9            -762.000000                    0.0

[10 rows x 47 columns]
input[features][0:10]:
   SK_ID_PREV  SK_ID_CURR NAME_CONTRACT_TYPE  AMT_ANNUITY  AMT_APPLI
CATION  \
0     2030495      271877     Consumer loans     1730.430          1
7145.0
1     2802425      108129         Cash loans    25188.615         60
7500.0
2     2523466      122040         Cash loans    15060.735         11
2500.0
3     2819243      176158         Cash loans    47041.335         45
0000.0
4     1784265      202054         Cash loans    31924.395         33
7500.0
5     1383531      199383         Cash loans    23703.930         31
5000.0
6     2315218      175704         Cash loans          NaN
0.0
7     1656711      296299         Cash loans          NaN
```

```
0.0
8     2367563      342292         Cash loans          NaN
0.0
9     2579447      334349         Cash loans          NaN
0.0

   AMT_CREDIT  AMT_DOWN_PAYMENT  AMT_GOODS_PRICE WEEKDAY_APPR_PROCES
S_START  \
0     17145.0              0.0          17145.0                  S
ATURDAY
1    679671.0              NaN         607500.0                  T
HURSDAY
2    136444.5              NaN         112500.0
TUESDAY
3    470790.0              NaN         450000.0
MONDAY
4    404055.0              NaN         337500.0                  T
HURSDAY
5    340573.5              NaN         315000.0                  S
ATURDAY
6         0.0              NaN              NaN
TUESDAY
7         0.0              NaN              NaN
MONDAY
8         0.0              NaN              NaN
MONDAY
9         0.0              NaN              NaN                  S
ATURDAY

   HOUR_APPR_PROCESS_START  ... NAME_SELLER_INDUSTRY  CNT_PAYMENT  \
0                       15  ...         Connectivity         12.0
1                       11  ...                  XNA         36.0
2                       11  ...                  XNA         12.0
3                        7  ...                  XNA         12.0
4                        9  ...                  XNA         24.0
5                        8  ...                  XNA         18.0
6                       11  ...                  XNA          NaN
7                        7  ...                  XNA          NaN
8                       15  ...                  XNA          NaN
9                       15  ...                  XNA          NaN

   NAME_YIELD_GROUP       PRODUCT_COMBINATION  DAYS_FIRST_DRAWING  \
0            middle  POS mobile with interest            365243.0
1       low_action          Cash X-Sell: low            365243.0
2              high         Cash X-Sell: high            365243.0
3            middle       Cash X-Sell: middle            365243.0
4              high         Cash Street: high                 NaN
5        low_normal          Cash X-Sell: low            365243.0
6               XNA                     Cash                 NaN
7               XNA                     Cash                 NaN
8               XNA                     Cash                 NaN
9               XNA                     Cash                 NaN

  DAYS_FIRST_DUE DAYS_LAST_DUE_1ST_VERSION  DAYS_LAST_DUE DAYS_TERMI
```

```
NATION  \
0          -42.0                     300.0          -42.0
-37.0
1         -134.0                     916.0       365243.0         36
5243.0
2         -271.0                      59.0       365243.0         36
5243.0
3         -482.0                    -152.0         -182.0
-177.0
4            NaN                       NaN            NaN
NaN
5         -654.0                    -144.0         -144.0
-137.0
6            NaN                       NaN            NaN
NaN
7            NaN                       NaN            NaN
NaN
8            NaN                       NaN            NaN
NaN
9            NaN                       NaN            NaN
NaN

  NFLAG_INSURED_ON_APPROVAL
0                       0.0
1                       1.0
2                       1.0
3                       1.0
4                       NaN
5                       1.0
6                       NaN
7                       NaN
8                       NaN
```

Join the labeled dataset

In [68]:
```
prevApps_aggregated = prevAppsFeaturesAggregater(features).transform
(appsDF)
```

In [69]:
```
prevApps_aggregated['prev_app_cnt'] = pd.DataFrame(prevAppCounts)
```

In [70]:
```
prevApps_aggregated['prev_app_cnt']
```

Out[70]:
```
0          NaN
1          NaN
2          NaN
3          NaN
4          NaN
           ...
338852     3.0
338853    13.0
338854     6.0
338855    13.0
338856     4.0
Name: prev_app_cnt, Length: 338857, dtype: float64
```

In [71]:
```
appTrain = datasets['application_train']
appTest = datasets['application_test']
y_train = appTrain['TARGET']
X_train = appTrain
appTrain = appTrain.merge(prevApps_aggregated, how='left', on='SK_ID_CURR')
appTest = appTest.merge(prevApps_aggregated, how='left', on='SK_ID_CURR')
```

In [72]:
```
X_kaggle_test= appTest
```

In [73]:
```
~3==3
```

Out[73]:
```
False
```

In [74]:
```
datasets.keys()
```

Out[74]:
```
dict_keys(['application_train', 'application_test', 'bureau', 'bureau_balance', 'credit_card_balance', 'installments_payments', 'previous_application', 'POS_CASH_balance'])
```

Join the unlabeled dataset (i.e., the submission file)

In [75]:
```
# approval rate 'NFLAG_INSURED_ON_APPROVAL'
```

In [76]:
```
# Convert categorical features to numerical approximations (via pipe
line)
class ClaimAttributesAdder(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self
    def transform(self, X, y=None):
        charlson_idx_dt = {'0': 0, '1-2': 2, '3-4': 4, '5+': 6}
        los_dt = {'1 day': 1, '2 days': 2, '3 days': 3, '4 days': 4,
'5 days': 5, '6 days': 6,
          '1- 2 weeks': 11, '2- 4 weeks': 21, '4- 8 weeks': 42, '26+
weeks': 180}
        X['PayDelay'] = X['PayDelay'].apply(lambda x: int(x) if x !=
'162+' else int(162))
        X['DSFS'] = X['DSFS'].apply(lambda x: None if pd.isnull(x) e
lse int(x[0]) + 1)
        X['CharlsonIndex'] = X['CharlsonIndex'].apply(lambda x: char
lson_idx_dt[x])
        X['LengthOfStay'] = X['LengthOfStay'].apply(lambda x: None i
f pd.isnull(x) else los_dt[x])
        return X
```

Processing pipeline

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

Train, validation and Test sets (and the leakage problem we have mentioned previously):

Let's look at a small usecase to tell us how to deal with this:

- The OneHotEncoder is fitted to the training set, which means that for each unique value present in the training set, for each feature, a new column is created. Let's say we have 39 columns after the encoding up from 30 (before preprocessing).
- The output is a numpy array (when the option sparse=False is used), which has the disadvantage of losing all the information about the original column names and values.
- When we try to transform the test set, after having fitted the encoder to the training set, 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 a example that in action:

```
# Identify the categorical features we wish to consider.
cat_attribs = ['CODE_GENDER', 'FLAG_OWN_REALTY','FLAG_OWN_CAR','NAME_CONTR
ACT_TYPE',
               'NAME_EDUCATION_TYPE','OCCUPATION_TYPE','NAME_INCOME_TYPE']

# Notice handle_unknown="ignore" in OHE which ignore values from the valid
ation/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"))
    ])
```

OHE case study: The breast cancer wisconsin dataset (classification)

In [77]:
```
datasets["application_train"]['TARGET']
```

Out[77]:
```
0         1
1         0
2         0
3         0
4         0
         ..
307506    0
307507    0
307508    0
307509    1
307510    0
Name: TARGET, Length: 307511, dtype: int64
```

Please this blog (https://medium.com/hugo-ferreiras-blog/dealing-with-categorical-features-in-machine-learning-1bb70f07262d) for more details of OHE when the validation/test have previously unseen unique values.

HCDR preprocessing

In [78]:
```
# Split the provided training data into training and validationa and
test
# The kaggle evaluation test set has no labels
#
from sklearn.model_selection import train_test_split


appTrain = datasets['application_train']
appTest = datasets['application_test']
y_train = appTrain['TARGET']
X_train = appTrain
appTrain = appTrain.merge(prevApps_aggregated, how='left', on='SK_ID
_CURR')
appTest = appTest.merge(prevApps_aggregated, how='left', on='SK_ID_C
URR')

use_application_data_ONLY = False#use joined data
if use_application_data_ONLY:
    # just selected a few features for a baseline experiment
    selected_features = ['AMT_INCOME_TOTAL',  'AMT_CREDIT','DAYS_EMP
LOYED','DAYS_BIRTH','EXT_SOURCE_1',
        'EXT_SOURCE_2','EXT_SOURCE_3','CODE_GENDER', 'FLAG_OWN_REALT
Y','FLAG_OWN_CAR','NAME_CONTRACT_TYPE',
                    'NAME_EDUCATION_TYPE','OCCUPATION_TYPE','NAME_INC
OME_TYPE']
    X_train = datasets["application_train"][selected_features]
    y_train = datasets["application_train"]['TARGET']
    X_train, X_valid, y_train, y_valid = train_test_split(X_train, y
_train, test_size=0.15, random_state=42)
    X_train, X_test, y_train, y_test = train_test_split(X_train, y_t
rain, test_size=0.15, random_state=42)
    X_kaggle_test= datasets["application_test"][selected_features]
    # y_test = datasets["application_test"]['TARGET']   #why no  TAR
GET?!! (hint: kaggle competition)

selected_features = ['AMT_INCOME_TOTAL',  'AMT_CREDIT','DAYS_EMPLOYE
D','DAYS_BIRTH','EXT_SOURCE_1',
        'EXT_SOURCE_2','EXT_SOURCE_3','CODE_GENDER', 'FLAG_OWN_REALT
Y','FLAG_OWN_CAR','NAME_CONTRACT_TYPE',
                    'NAME_EDUCATION_TYPE','OCCUPATION_TYPE','NAME_INC
OME_TYPE']
y_train = X_train['TARGET']
X_train = X_train[selected_features]
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_tra
in, test_size=0.15, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X_train, y_trai
n, test_size=0.15, random_state=42)
X_kaggle_test= X_kaggle_test[selected_features]
# y_test = datasets["application_test"]['TARGET']   #why no  TARGE
T?!! (hint: kaggle competition)


print(f"X train           shape: {X_train.shape}")
print(f"X validation      shape: {X_valid.shape}")
```

```
print(f"X test            shape: {X_test.shape}")
print(f"X X_kaggle_test   shape: {X_kaggle_test.shape}")
```

```
X train           shape: (222176, 14)
X validation      shape: (46127, 14)
X test            shape: (39208, 14)
X X_kaggle_test   shape: (48744, 14)
```

In [79]:

```
from sklearn.base import BaseEstimator, TransformerMixin
import re

# Creates the following date features
# But could do so much more with these features
#    E.g.,
#      extract the domain address of the homepage and OneHotEncode i
t
#
# ['release_month','release_day','release_year', 'release_dayofweek
','release_quarter']
class prep_OCCUPATION_TYPE(BaseEstimator, TransformerMixin):
    def __init__(self, features="OCCUPATION_TYPE"): # no *args or **
kargs
        self.features = features
    def fit(self, X, y=None):
        return self  # nothing else to do
    def transform(self, X):
        df = pd.DataFrame(X, columns=self.features)
        #from IPython.core.debugger import Pdb as pdb;    pdb().set_
trace() #breakpoint; dont forget to quit
        df['OCCUPATION_TYPE'] = df['OCCUPATION_TYPE'].apply(lambda
x: 1. if x in ['Core Staff', 'Accountants', 'Managers', 'Sales Staff
', 'Medicine Staff', 'High Skill Tech Staff', 'Realty Agents', 'IT S
taff', 'HR Staff'] else 0.)
        #df.drop(self.features, axis=1, inplace=True)
        return np.array(df.values)  #return a Numpy Array to observe
the pipeline protocol


from sklearn.pipeline import make_pipeline
features = ["OCCUPATION_TYPE"]
def test_driver_prep_OCCUPATION_TYPE():
    print(f"X_train.shape: {X_train.shape}\n")
    print(f"X_train['name'][0:5]: \n{X_train[features][0:5]}")
    test_pipeline = make_pipeline(prep_OCCUPATION_TYPE(features))
    return(test_pipeline.fit_transform(X_train))

x = test_driver_prep_OCCUPATION_TYPE()
print(f"Test driver: \n{test_driver_prep_OCCUPATION_TYPE()[0:10,
:]}")
print(f"X_train['name'][0:10]: \n{X_train[features][0:10]}")


# QUESTION, should we lower case df['OCCUPATION_TYPE'] as Sales staf
f != 'Sales Staff'? (hint: YES)
```

```
X_train.shape: (222176, 14)

X_train['name'][0:5]:
       OCCUPATION_TYPE
21614      Sales staff
209797        Laborers
17976              NaN
282543  Security staff
52206              NaN
X_train.shape: (222176, 14)

X_train['name'][0:5]:
       OCCUPATION_TYPE
21614      Sales staff
209797        Laborers
17976              NaN
282543  Security staff
52206              NaN
Test driver:
[[0.]
 [0.]
 [0.]
 [0.]
 [0.]
 [1.]
 [0.]
 [0.]
 [0.]
 [0.]]
X_train['name'][0:10]:
       OCCUPATION_TYPE
21614      Sales staff
209797        Laborers
17976              NaN
282543  Security staff
52206              NaN
152195       Managers
70364       Core staff
11643              NaN
45591       Core staff
93535         Laborers
```

In [80]:

```python
# 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
```

Baseline Model

To get a baseline, we will use some of the features after being preprocessed through the pipeline. The baseline model is a logistic regression model

In [81]:
```
### TEAM 10 EDIT ###
### SD - all of section 7 [17] ###
### TEAM 10 EDIT ###
```

Baseline with 14 features

In [82]:
```
# 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.cros
s_validation in old versions
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

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/test that
# 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='m
issing')),
        ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignor
e"))
    ])

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_g
rid, cv = 5, n_jobs=-1, scoring='roc_auc')

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_trai
n)[:, 1]),
                 roc_auc_score(y_valid, model.predict_proba(X_vali
d)[:, 1]),
                 roc_auc_score(y_test, model.predict_proba(X_test)[:,
1])],
    4))
expLog
```

Out[82]:

	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

Split train, validation and test sets

In [83]:
```
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 questionable value

# Split the provided training data into training and validationa and test
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=0.2, 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 [84]:

```
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.cros
s_validation in old versions
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 feature
s with questionable value

# Split the provided training data into training and validationa and
test
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_tra
in, test_size=0.2, 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="ignor
e"))
    ])


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_g
rid, cv = 5, n_jobs=-1, scoring='roc_auc')

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_trai
n)[:, 1]),
                 roc_auc_score(y_valid, model.predict_proba(X_vali
d)[:, 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[84]:

	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

Baseline 3: 79 Features

Selected Features

Remove elements with more than 50% nulls

In [85]:

```
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.cros
s_validation in old versions
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 feature
s with questionable value

# Split the provided training data into training and validationa and
test
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_tra
in, test_size=0.2, 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="ignor
e"))
    ])


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]
                              }

gd3 = GridSearchCV(full_pipeline_with_predictor, param_grid= param_g
rid, cv = 5, n_jobs=-1, scoring='roc_auc')

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_trai
n)[:, 1]),
                 roc_auc_score(y_valid, model.predict_proba(X_vali
d)[:, 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[85]:

	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

Baseline 4 : 79 features; 2 log features

Remove elements with more than 50% nulls with log AMT_ANNUITY and AMT_CREDIT

In [86]:

```
data = datasets["application_train"]
y = data['TARGET']
X = data.drop(['SK_ID_CURR', 'TARGET'], axis = 1) #drop some feature
s with questionable value
X['LOG_AMT_ANNUITY'] = np.log(X['AMT_ANNUITY']) #add LOG_AMT_ANNUITY
colunm
X = X.drop(['AMT_ANNUITY'], axis = 1) # drop AMT_ANNUITY colunm
X['LOG_AMT_CREDIT'] = np.log(X['AMT_CREDIT']) #add LOG_AMT_ANNUITY c
olunm
X = X.drop(['AMT_CREDIT'], axis = 1) # drop AMT_ANNUITY colunm

# Split the provided training data into training and validationa and
test
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_tra
in, test_size=0.2, 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="ignor
e"))
    ])


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_g
rid, cv = 5, n_jobs=-1, scoring='roc_auc')

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 att
ributes"
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_trai
n)[:, 1]),
                 roc_auc_score(y_valid, model.predict_proba(X_vali
d)[:, 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[86]:

	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

Submission File Prep

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 [87]:
```
### TEAM 10 EDIT ###
### SD- all of section 8 [18] ###
### TEAM 10 EDIT ###
```

In [88]:
```
data_test = datasets["application_test"]
print(data_test.shape)

X_Kaggle_test = data_test.drop('SK_ID_CURR', axis = 1)
test_class_scores = gd3.predict_proba(X_Kaggle_test)[:, 1]

submit_df = datasets["application_test"][['SK_ID_CURR']]
submit_df['TARGET'] = test_class_scores

submit_df.head()
```

```
(48744, 121)
```

Out[88]:

	SK_ID_CURR	TARGET
0	100001	0.079261
1	100005	0.256575
2	100013	0.050367
3	100028	0.029005
4	100038	0.096225

In [89]:
```
submit_df.to_csv("submission.csv",index=False)
```

report submission

Click on this link (https://www.kaggle.com/c/home-credit-default-risk/submissions?sortBy=date&group=all&page=1)

In [1]:
```
### TEAM 10 EDIT ###
### SD - all of section 9 [19] ###
### TEAM 10 EDIT ###
```

Write-up

For this phase of the project, you will need to submit a write-up summarizing the work you did. The write-up form is available on Canvas (Modules-> Module 12.1 - Course Project - Home Credit Default Risk (HCDR)-> FP Phase 2 (HCDR) : write-up form). It has the following sections:

Abstract

Our goal was to determine how accurately we can predict a new applicant's ability to pay loans. Our area of focus during his week was EDA, feature engineering and building a baseline model for our overall goal. From the EDA, **write about eda** AFter seeing the coorelations and adding necessary features, creating a data pipeline for char as well as numerical values, we created a pipeline and fed it into logistic regression. We got the bestparameters for C=100 and penalty as 'l2' (Gridsearch, cv = 5). From this pipeline, we obtained an ROC score of 73.57% with an accuracy of 91.58%.
**write about submission

Introduction

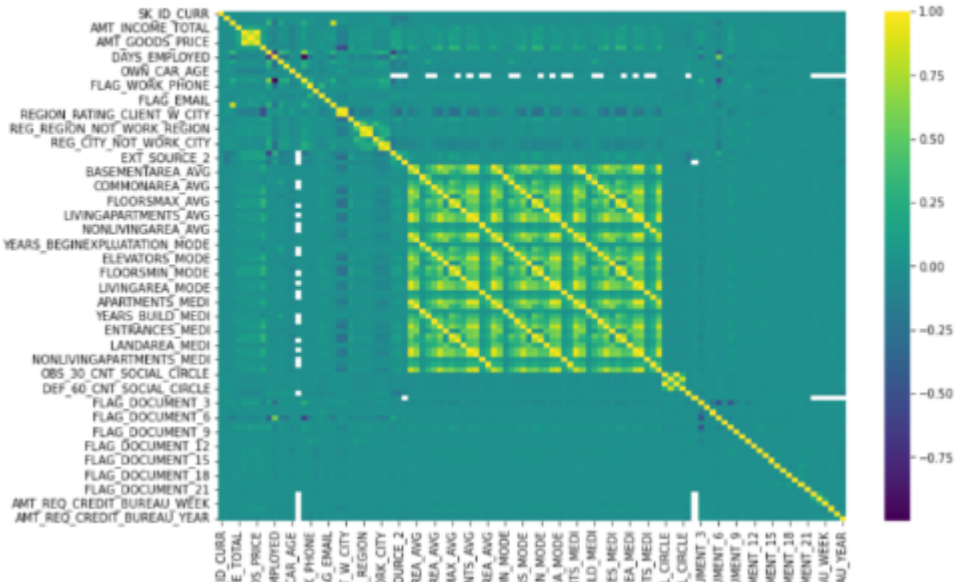
Feature Engineering and transformers

Please explain the work you conducted on feature engineering and transformers. Please include code sections when necessary as well as images or any relevant material.

For feature engineering, first we found 2 skewed attributes from the numerical features and did log transformation on the same. And, we also performed correlation between target variable and predictor variables. In additin, based on null value percentage for variables, we did feature engineering as well. Here are the screenshots for correlation and log transoformation.

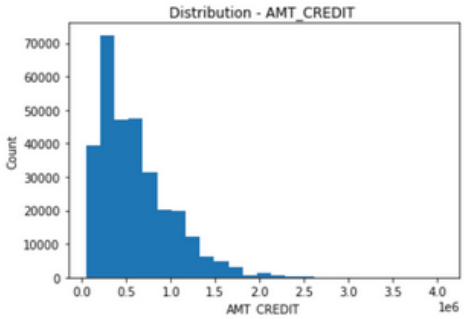
```
plt.figure(figsize=(12,8))
sns.heatmap(datasets["application_train"][numerical_ix].corr(), cmap="viridis")
```

```
<AxesSubplot:>
```



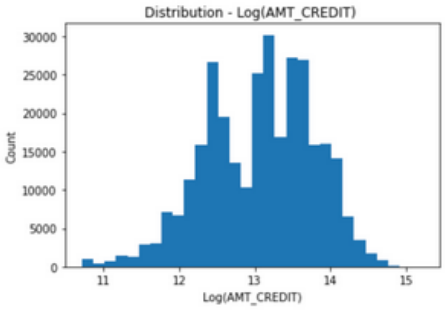
```
plt.hist(datasets["application_train"]['AMT_CREDIT'], bins=25);
plt.xlabel('AMT_CREDIT')
plt.ylabel('Count')
plt.title("Distribution - AMT_CREDIT ")
```

```
Text(0.5, 1.0, 'Distribution - AMT_CREDIT ')
```



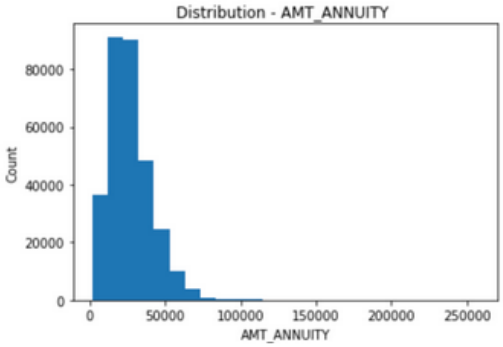

```
plt.hist(np.log(datasets["application_train"]['AMT_CREDIT']), bins=3
plt.xlabel('Log(AMT_CREDIT)')
plt.ylabel('Count')
plt.title("Distribution - Log(AMT_CREDIT) ")
```

```
Text(0.5, 1.0, 'Distribution - Log(AMT_CREDIT) ')
```



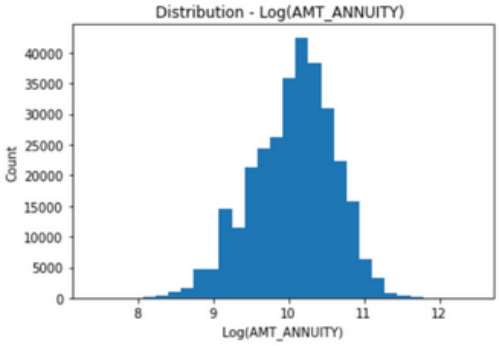

```
plt.hist(datasets["application_train"]['AMT_ANNUITY'],
plt.xlabel('AMT_ANNUITY')
plt.ylabel('Count')
plt.title("Distribution - AMT_ANNUITY ")
```

```
Text(0.5, 1.0, 'Distribution - AMT_ANNUITY ')
```




```
plt.hist(np.log(datasets["application_train"]['AMT_ANNUITY'])
plt.xlabel('Log(AMT_ANNUITY)')
plt.ylabel('Count')
plt.title("Distribution - Log(AMT_ANNUITY) ")
```

```
Text(0.5, 1.0, 'Distribution - Log(AMT_ANNUITY) ')
```



Pipelines

Please explain the pipelines you created for this project and how you used them Please include code sections when necessary as well as images or any relevant material

All pipelines are prepared under section 7.

Our **All Feature** model has all the 120 features (except 'SK_ID_CURR', 'TARGET')as available in the training set. Then there are 2 set of models where we are selecting the features that we want to use. 1st one, **Baseline with 14 attributes** has 14 attributes, and another one, **Baseline with 79 attributes** has 79 attributes. for model with 14 elements, we used 7 numerical and 7 categorical attributes. For model with 79 attributes, we removed the elements which have more than 50% null values in the the training data. Last model which we used, **Baseline with log features** it has 2 attributes which were log transformed. Apart from that, we used the 79 elements model with it.

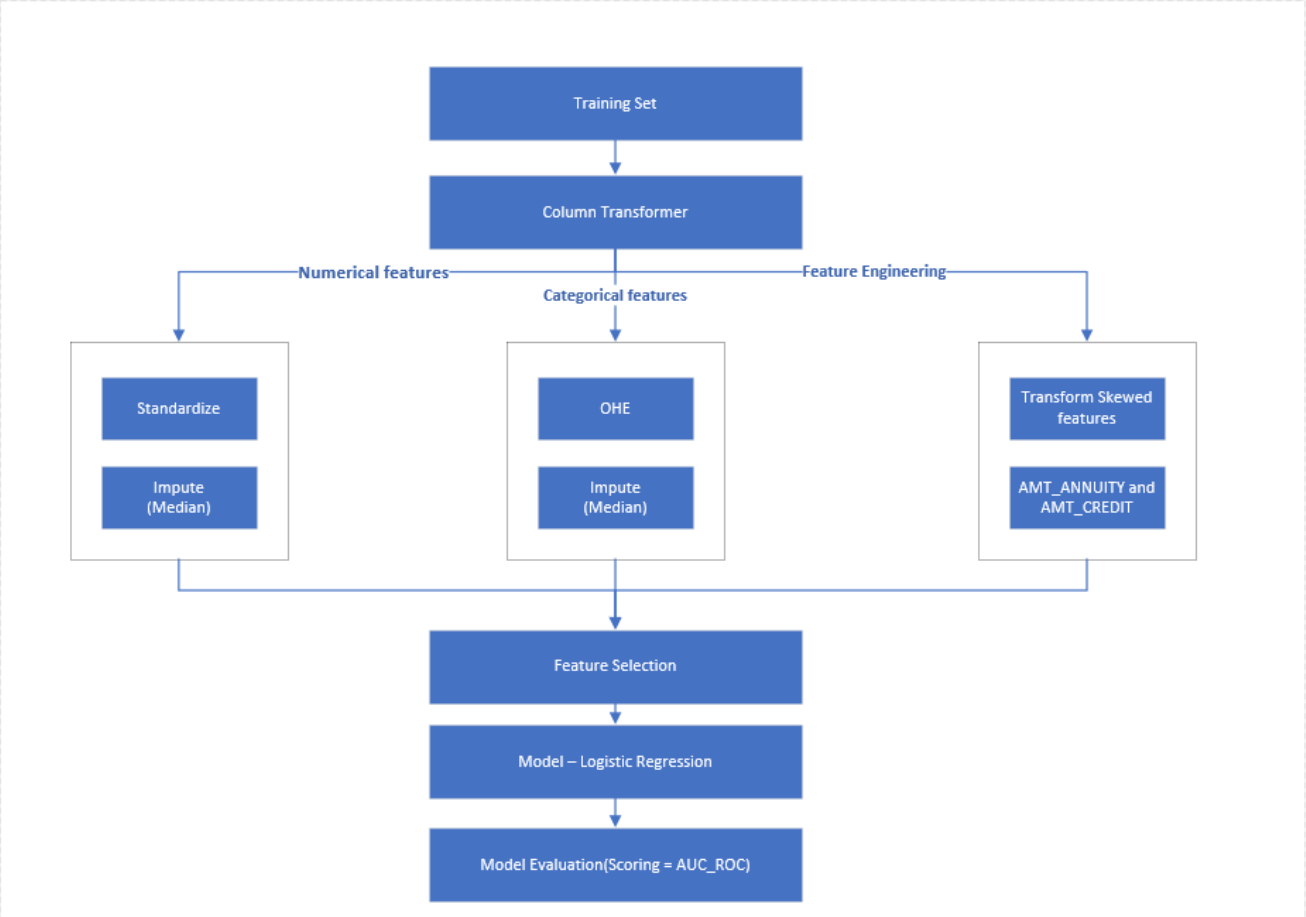
Numerical features were standardized using Standard scaler, and we used median to fill in the missing values. As far as categorical data is concerned, We did one hot encoding to standatdize and missing values were filled using the ost frequent values.

AUC_ROC score is used to evaluate our models.

We have created 3 Logistic regression models:

```
Baseline1_all features 120 raw inputs
Baseline2_all features 14 inputs
Baseline3_selected features 79 raw inputs
Baseline4_selected features 79 raw inputs with a log feature (log AMT_ANNU
ITY, log AMT_CREDIT)
```

	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



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

```
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)
```

:[86]:

	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

Discussion

We did 4 experimemts as stated in the picture attached. Except for model with 14 features, all other models have training accuracy of 92%. Best validation accuracy was schieved for model 3 and 4 and section 7.4 and 7.5. For the scoring, we used AUC score which was 74.34% for the model with all attributes.

With more robust feature selection, OHE's, loss functions, and grid paramaters, we think that the score could be improved substantially. We will use our current understanding and inculcalte the learnings and will try to improve our results.

Conclusion

In this phase we have understood the dataset by performing EDA where we have checked for the relationship between Target variable and numerical features,categorical features and also checked for the missing values for features.

Upon that, we build 4 pipelines using logistic regression and we have compared all the training and testing accuracies to see which pipeline works best. Our aim in this phase is to achieve best accuracy.

The baseline model with all features has the highest test AUC score in notebook (0.7434) and in Kaggle submission (0.733). We hope for a better score by doing more feature engineering in the next phase.

For the next phase, we will be performing feature engineering on all the data sets to find the important features, perform dimensionality reduction, SVC, and perform feature engineering.

Kaggle Submission