Project Proposal

Home Credit Default Risk
FP_GroupN_11

Kumud Sharma(kumsharm@iu.edu)
Kamna Chaudhary (kamchau@iu.edu)
Bhavya Mistry(brmistry@iu.edu)
Jaydeep Patel(jp157@iu.edu)

Abstract

In this project, we're aiming to forecast whether a particular client would pay back the loan they obtained. The "application train.csv" dataset will be the main one we use, although other information from related subsets of datasets will also be helpful. We want to perform extensive feature engineering and data preparation, and we might decide to lower the application dataset's current 121 feature count. We're going to release a couple various models so we can evaluate them. These models include logistic regression with lasso, ridge, and no regularization, support vector machines with a linear kernel and radial, decision trees or random forests, and neural networks with PyTorch with various network topologies. We may also employ K-nearest neighbors if we can significantly reduce the number of features. The F1 score measure will be applied to adjust the models' hyperparameters. To determine the model that performs the best, we will then compare the F1 scores of the tweaked models.

Data Description

The primary training segment of the data is organized in a CSV file named "application train.csv," and the corresponding test data are in "application test.csv." Only the target column for whether the client repaid the loan separates the training data set's columns from those in the test set. The application dataset has 121 feature columns that comprise both categorical variables, such gender, and numerical features, like income.

A number of additional datasets build on one another and correlate with the application dataset. Any credit history row for the customer prior to the application date that corresponds to a loan in the application dataset is present in the "bureau.csv" dataset. Data for the prior credits stated in the bureau dataset are included in the "bureau balance.csv" dataset for each month in history. These supporting datasets interact with one another and the application dataset in this manner. Categorical values as well as positive and negative numerical values are present in these auxiliary datasets. There are a total of six subsidiary datasets, but since the test set is dependent on the application set, that is the one we will be focusing on.

The graphic below illustrates the connections between the various datasets offered in the issue:

1. application_{train|test}.csv

This is the primary dataset, which consists of training and test samples. This set uses the SK ID CURR property to connect to other sets and contains information on the loan applicant's personal characteristics.

2. previous _application.csv

This dataset contains details regarding prior Home Credit loans made by the applicant. It includes characteristics including loan status, down payment, and loan type.

3. instalments_payments.csv

Information on loan repayment from the past is included in this dataset.

4. credit_card_balance.csv

Information on Home Credit credit card transactions is included in this dataset.

5. POS_CASH_balance.csv

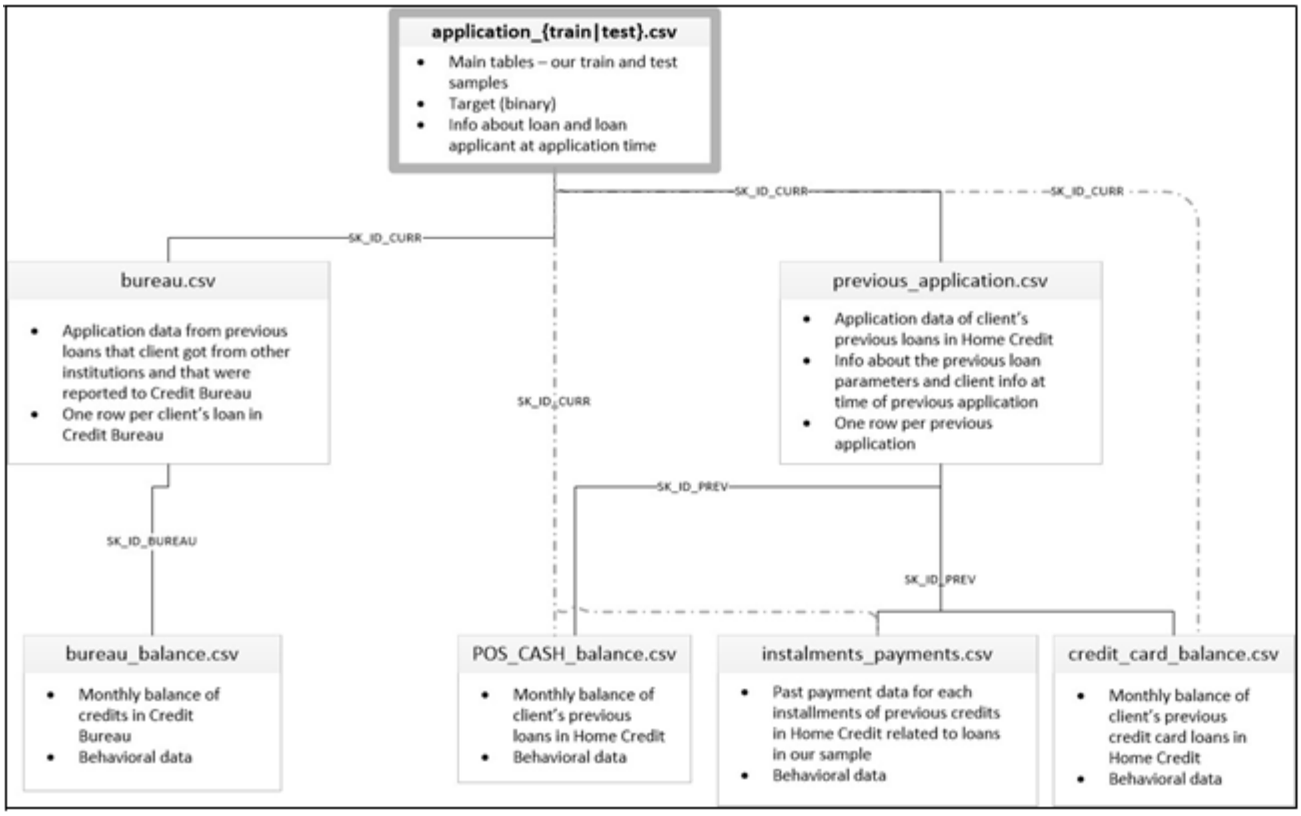
The items in this dataset describe the individual's past credit history at Home Credit, which includes personal loans and consumer credit.

6. bureau.csv

This dataset contains details about a person's prior credit history at other financial institutions that were reported to a credit agency.

7. Previous_application.csv

This dataset includes the credit bureau's monthly credit balance.



Machine Learning Algorithms and Metrics

Models:

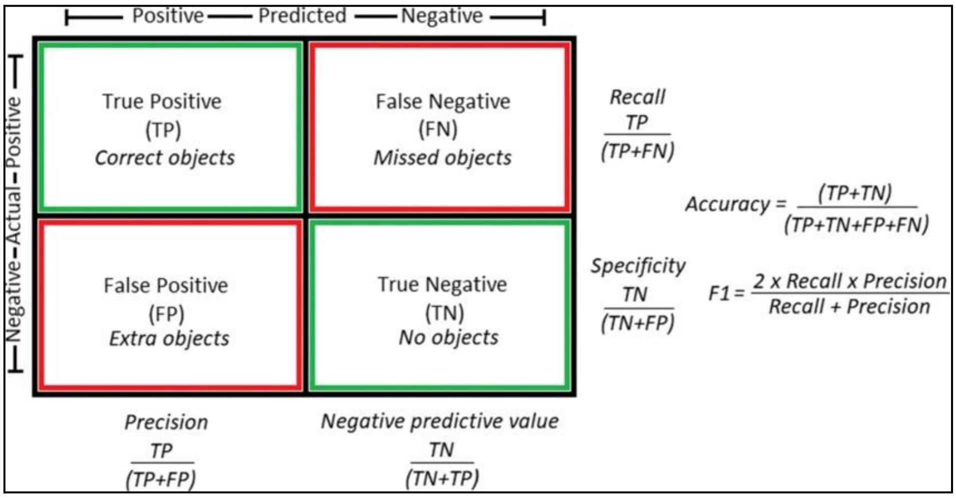
We are aiming to anticipate the client's ability to repay the loan, as explained in the abstract and data description module, and this ability is decided by a single binary output variable called TARGET (0|1). Under supervised learning, this is a binary classification issue because the goal has only two discrete possibilities that could occur. We will compare various categorization models using the chosen metrics to find the model that best fits the situation.

We will use Logistic Regression (with Lasso, Ridge, and No Regularization) as our basic model because this is a binary classification problem. Based on the Logistic Regression model, we will evaluate the effectiveness of different models. The likelihood of various classes or clusters occurring in the dataset will be predicted using a Naive Bayes classifier. We will use the Decision Tree Classifier model by developing rules based on the application train.csv dataset, notably on the income and credit columns such as amt income total, amt credit, etc. We will also investigate Support Vector Machines (SVM), Random Forest Classifier, K-Nearest Neighbors, and XGBoost Classifier. Later on, we'll concentrate on dimensionality reduction and hyperparameter tweaking, setting the stage for a thrilling algorithmic race.

Metrics:

1. Confusion Matrix:

A confusion matrix is used to evaluate the effectiveness of a classification model, and the resulting matrix reveals the degree of classification accuracy and potential prediction mistakes for each record. It is used to gauge AUC-ROC curve, recall, accuracy, and precision.



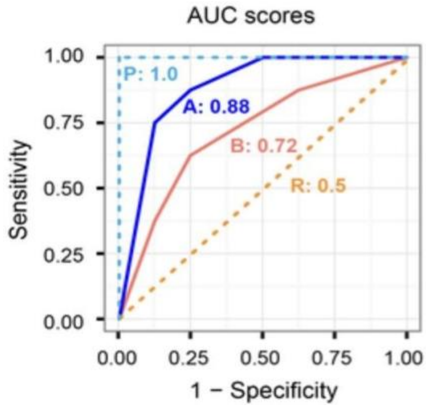
2. F-1 Score:

It will be used to compare the effectiveness of two classifiers based on their precision and recall values. It will be crucial in deciding which model fits the issue more effectively.

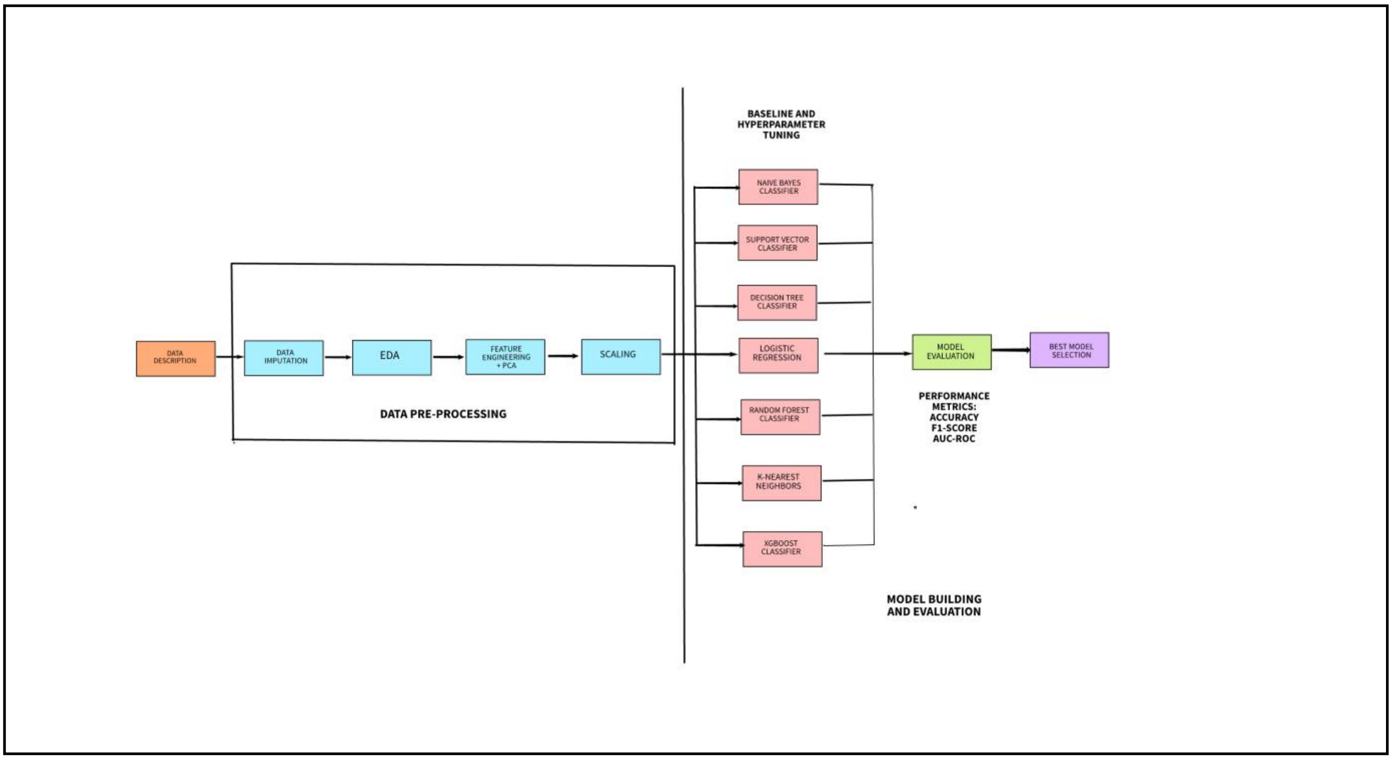
$$F_1 = 2 * \frac{precision * recall}{precision + recall}$$

3.AUC-ROC Curve:

When the actual outcome is positive, the ROC curve will assist us in calculating the likelihood of correctly predicting the positive class. Its performance is distilled into a single value using the AUC curve.



Machine Learning Pipeline

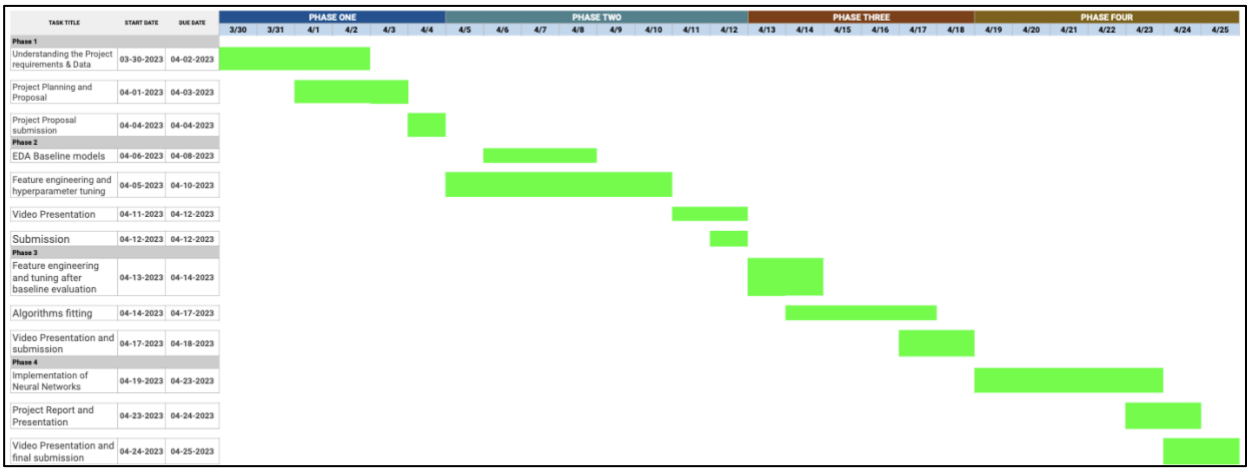


The dataset is briefly described at the outset of our project. It is crucial to gain an understanding of how the dataset is structured because we are working with numerous files here. We will identify the numerous problems with the current data, such as missing values, duplicate values, and inconsistent data types, and address each problem individually. We will use a straightforward imputer to substitute the missing values for the continuous variables' median values and the categorical features' modal values. An exploratory data analysis of the application train and application test data will then come after this. The features that are crucial for our goal feature can be identified based on the conclusions gained.

As each dataset contains more than 120 columns, we may apply principal component analysis to reduce the features that are less important based on their variance. All of the aforementioned machine learning models will then undergo baseline training and evaluation, which will allow us to assess how well each model performs in relation to the aforementioned performance indicators. Due to the dataset's extreme imbalance and the fact that accuracy cannot be trusted with such datasets, F1-score and ROC-AUC will be given higher priority. As ensemble models can choose a subset of features during training, they can perform better in this scenario than models like XGBoost and Random Forest. In order to prevent data loss, a sklearn pipeline will also be used. By running a successful race and evaluating which models perform best, we will choose the best ones and fine-tune them. A second model made out of a PyTorch deep neural network will also be put into practice and compared to our improved models. We will choose our best model after comparing the results.

Timeline

The proposed timeline for the Home Credit Default Risk Project:



Phase Leader Plan

Final Project Phase	**Phase Leader**	**Phase Plan**
Phase 0	Team	Team Formation
Phase 1: Project Proposal	Kumud Sharma	There will be a project proposal created outlining all of the project's components. Each team member is given a task to do. The dataset is investigated, and potential machine learning techniques that could be used in the project are described. A base pipeline for the project is chosen, and appropriate evaluation metrics are determined.
Phase 2: EDA + Baseline	Bhavya Mistry	With the chosen machine learning methods, this phase focuses on baseline model training and evaluation as well as exploratory data analysis, data imputation, and baseline model evaluation. The EDA and baseline performances of several models will be used to derive conclusions, and then judgments will be taken regarding hyperparameter tuning and feature selection.

Phase 3: Feature Engineering + Hyperparameter Tuning	Kamna Chaudhary	Here, we'll concentrate on the feature of choosing the most suitable features based on several strategies, including correlation, developing new features, and dimensionality reduction, as well as our comprehension of the data from earlier phases. To help determine the ideal set of parameters for each machine learning model, we will also begin experimenting with the different parameters of the models. This will pave the stage for an exciting race between the various algorithms.
Phase 4: Final Submission	Jaydeep Patel	Deep neural networks will be used in this step, and we will compare their performance to the existing fine-tuned models from our previous phase. We will choose the best model and submit it in our final submission based on our numerous performance indicators.

Credit Assignment Plan(Phase 1):

Team Member	**Task**
Kumud Sharma	EDA and Data Preprocessing
Kamna Chaudhary	Feature Engineering and Baseline Modeling
Bhavya Mistry	Feature Selection, Dimensionality Reduction, Model Evaluation
Jaydeep Patel	Hyperparameter Tuning, Best model selection

Home Credit Default Risk (HCDR)

The course project is based on the Home Credit Default Risk (HCDR) Kaggle Competition. 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 compressed) with millions of rows of data
    - 2.71 Gig of data uncompressed

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

Kaggle API setup

Kaggle is a Data Science Competition Platform which shares a lot of datasets. In the past, it was troublesome to submit your result as your have to go through the console in your browser and drag your files there. Now you can interact with Kaggle via the command line. E.g.,

```
! kaggle competitions files home-credit-default-risk
```

It is quite easy to setup, it takes me less than 15 minutes to finish a submission.

1. Install library

- Create a API Token (edit your profile on Kaggle.com); this produces `kaggle.json` file
- Put your JSON `kaggle.json` in the right place
- Access competition files; make submissions via the command (see examples below)
- Submit result

For more detailed information on setting the Kaggle API see here and here.

In [7]:
```
!pip install kaggle
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.7/site-packa
ges (1.5.12)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.7/site-pa
ckages (from kaggle) (1.15.0)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.7/site-pack
ages (from kaggle) (1.26.6)
Requirement already satisfied: requests in /usr/local/lib/python3.7/site-pac
kages (from kaggle) (2.25.1)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.7/s
ite-packages (from kaggle) (2.8.2)
Requirement already satisfied: certifi in /usr/local/lib/python3.7/site-pack
ages (from kaggle) (2021.5.30)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.7/si
te-packages (from kaggle) (5.0.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/site-package
s (from kaggle) (4.62.1)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3
.7/site-packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: chardet<5,>=3.0.2 in /usr/local/lib/python3.7
/site-packages (from requests->kaggle) (4.0.0)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/site
-packages (from requests->kaggle) (2.10)
WARNING: Running pip as the 'root' user can result in broken permissions and
conflicting behaviour with the system package manager. It is recommended to
use a virtual environment instead: https://pip.pypa.io/warnings/venv
WARNING: You are using pip version 21.2.4; however, version 21.3.1 is availa
ble.
You should consider upgrading via the '/usr/local/bin/python -m pip install
--upgrade pip' command.
```

In [8]:

```
!pwd
```

```
/root/shared/Dropbox/Projects/Courses/DataScienceAtScale/Src/I526_AML_Dev_CA
DG/Assignments/Unit-Project-Home-Credit-Default-Risk/HCDR_Phase_1_baseline_s
ubmission
```

In [15]:

```
!pwd
```

```
/root/shared/Dropbox/Projects/Courses/DataScienceAtScale/Src/I526_AML_Dev_CA
DG/Assignments/Unit-Project-Home-Credit-Default-Risk/HCDR_Phase_1_baseline_s
ubmission
```

In [17]:

```
!ls -l  ~/.kaggle/kaggle.json
```

```
-rw------- 1 root root 65 Nov  9 02:14 /root/.kaggle/kaggle.json
```

In [16]:

```
!mkdir ~/.kaggle
!cp kaggle.json ~/.kaggle
!chmod 600 ~/.kaggle/kaggle.json
```

```
mkdir: cannot create directory ‘/root/.kaggle’: File exists
```

In [18]:

```
! kaggle competitions files home-credit-default-risk
```

```
name                                  size  creationDate
----------------------------------  -----  -------------------
application_train.csv                158MB  2019-12-11 02:55:35
bureau_balance.csv                   358MB  2019-12-11 02:55:35
bureau.csv                           162MB  2019-12-11 02:55:35
POS_CASH_balance.csv                 375MB  2019-12-11 02:55:35
installments_payments.csv            690MB  2019-12-11 02:55:35
credit_card_balance.csv              405MB  2019-12-11 02:55:35
sample_submission.csv                524KB  2019-12-11 02:55:35
previous_application.csv             386MB  2019-12-11 02:55:35
application_test.csv                  25MB  2019-12-11 02:55:35
HomeCredit_columns_description.csv    37KB  2019-12-11 02:55:35
```

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

The `HomeCredit_columns_description.csv` acts as a data dictioanry.

There are 7 different sources of data:

- **application_train/application_test (307k rows, and 48k rows):** 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 (1.7 Million rows):** 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 (27 Million rows):** 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 (1.6 Million rows):** 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 (10 Million rows):** 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

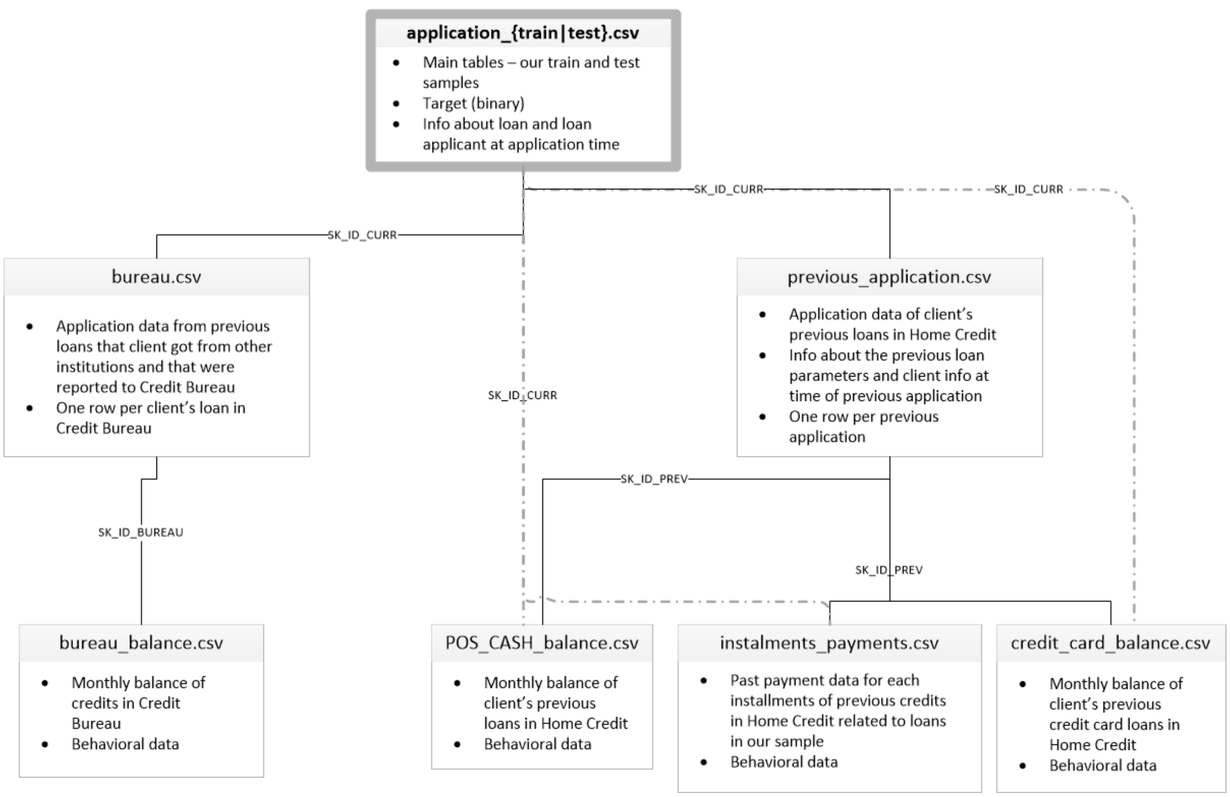
card can have many rows.

- **installments_payment (13.6 Million rows):** payment history for previous loans at Home Credit. There is one row for every made payment and one row for every missed payment.

Table sizes

```
name                       [  rows cols]     MegaBytes
-----------------------   ------------------  -------
application_train          : [  307,511, 122]:   158MB
application_test           : [   48,744, 121]:   25MB
bureau                     : [ 1,716,428, 17]    162MB
bureau_balance             : [ 27,299,925, 3]:   358MB
credit_card_balance        : [  3,840,312, 23]   405MB
installments_payments      : [ 13,605,401, 8]    690MB
previous_application       : [  1,670,214, 37]   386MB
POS_CASH_balance           : [ 10,001,358, 8]    375MB
```

In [ ]:



Downloading the files via Kaggle API

Create a base directory:

```
DATA_DIR = "../../../Data/home-credit-default-risk"   #same level as course 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 and unzip the zip file to the `BASE_DIR`
2. If you plan to use the Kaggle API, please use the following steps.

In [19]:
```
DATA_DIR = "../../../Data/home-credit-default-risk"   #same level as course
#DATA_DIR = os.path.join('./ddddd/')
!mkdir DATA_DIR
```

```
mkdir: cannot create directory ‘DATA_DIR’: File exists
```

In [20]:
```
!ls -l DATA_DIR
```

```
total 0
```

In [25]:
```
! kaggle competitions download home-credit-default-risk -p $DATA_DIR
```

```
Downloading home-credit-default-risk.zip to ../../../Data/home-credit-default-risk
100%|████████████████████████████████████████| 688M/688M [00:48<00:00, 16.4MB/s]
100%|████████████████████████████████████████| 688M/688M [00:48<00:00, 14.8MB/s]
```

In [29]:
```
!pwd
```

```
/root/shared/Dropbox/Projects/Courses/DataScienceAtScale/Src/I526_AML_Dev_CADG/Assignments/Unit-Project-Home-Credit-Default-Risk/HCDR_Phase_1_baseline_submission
```

In [28]:
```
!ls -l $DATA_DIR
```

```
total 705536
-rw-r--r-- 1 root root 721616255 Nov  9 02:19 home-credit-default-risk.zip
```

In [26]:
```
#!rm -r  DATA_DIR
```

Imports

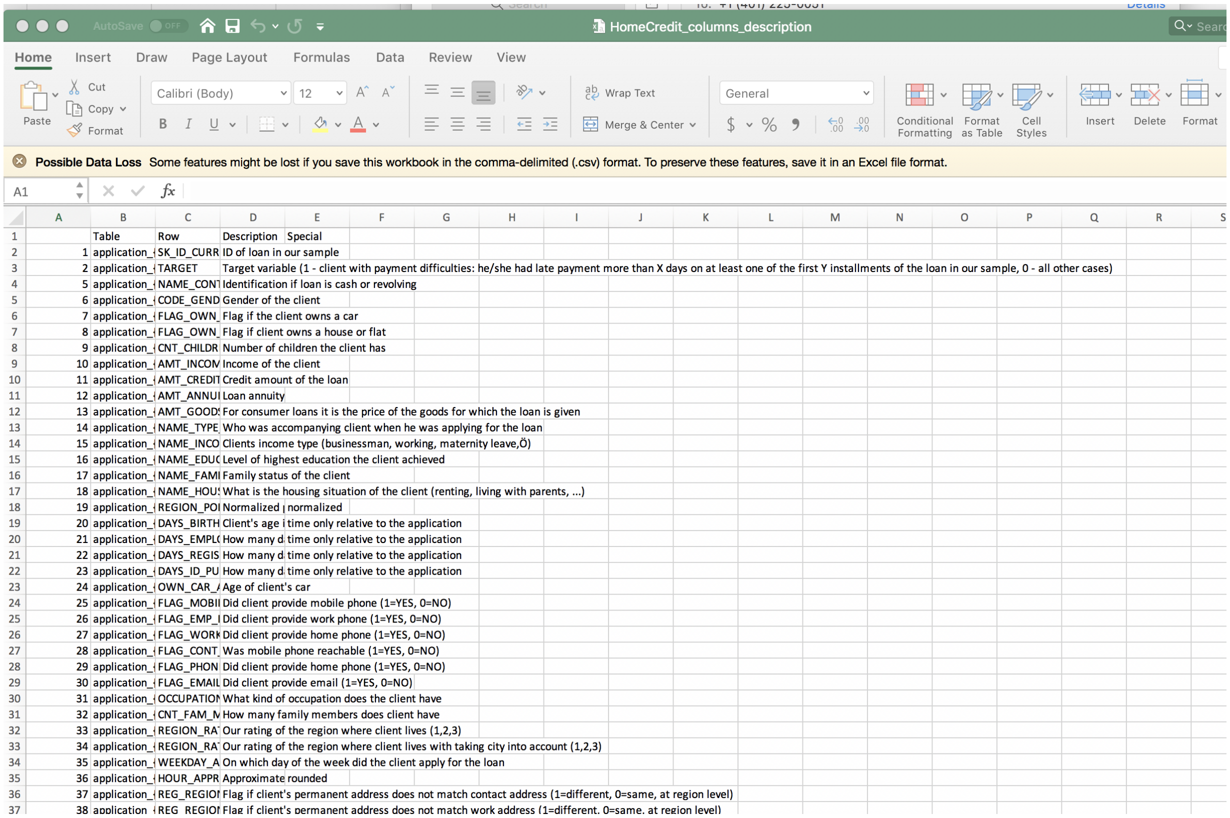
In [22]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import os
import zipfile
from sklearn.base import BaseEstimator, TransformerMixin
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection 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 [31]:
```
unzippingReq = True #True
if unzippingReq: #please modify this code
    zip_ref = zipfile.ZipFile(f'{DATA_DIR}/home-credit-default-risk.zip', 'r
    # extractall():  Extract all members from the archive to the current wor
    zip_ref.extractall('{DATA_DIR}')
    zip_ref.close()
```

Data files overview

Data Dictionary

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



	A	B	C	D	E
1		Table	Row	Description	Special
2	1	application_	SK_ID_CURR	ID of loan in our sample	
3	2	application_	TARGET	Target variable (1 - client with payment difficulties: he/she had late payment more than X days on at least one of the first Y installments of the loan in our sample, 0 - all other cases)	
4	5	application_	NAME_CONT	Identification if loan is cash or revolving	
5	6	application_	CODE_GEND	Gender of the client	
6	7	application_	FLAG_OWN_	Flag if the client owns a car	
7	8	application_	FLAG_OWN_	Flag if client owns a house or flat	
8	9	application_	CNT_CHILDR	Number of children the client has	
9	10	application_	AMT_INCOM	Income of the client	
10	11	application_	AMT_CREDIT	Credit amount of the loan	
11	12	application_	AMT_ANNUI	Loan annuity	
12	13	application_	AMT_GOODS	For consumer loans it is the price of the goods for which the loan is given	
13	14	application_	NAME_TYPE	Who was accompanying client when he was applying for the loan	
14	15	application_	NAME_INCO	Clients income type (businessman, working, maternity leave,Ö)	
15	16	application_	NAME_EDUC	Level of highest education the client achieved	
16	17	application_	NAME_FAMI	Family status of the client	
17	18	application_	NAME_HOUS	What is the housing situation of the client (renting, living with parents, ...)	
18	19	application_	REGION_POI	Normalized	normalized
19	20	application_	DAYS_BIRTH	Client's age i	time only relative to the application
20	21	application_	DAYS_EMPL(	How many d	time only relative to the application
21	22	application_	DAYS_REGIS	How many d	time only relative to the application
22	23	application_	DAYS_ID_PU	How many d	time only relative to the application
23	24	application_	OWN_CAR_/	Age of client's car	
24	25	application_	FLAG_MOBII	Did client provide mobile phone (1=YES, 0=NO)	
25	26	application_	FLAG_EMP_I	Did client provide work phone (1=YES, 0=NO)	
26	27	application_	FLAG_WORK	Did client provide home phone (1=YES, 0=NO)	
27	28	application_	FLAG_CONT_	Was mobile phone reachable (1=YES, 0=NO)	
28	29	application_	FLAG_PHON	Did client provide home phone (1=YES, 0=NO)	
29	30	application_	FLAG_EMAIL	Did client provide email (1=YES, 0=NO)	
30	31	application_	OCCUPATION	What kind of occupation does the client have	
31	32	application_	CNT_FAM_M	How many family members does client have	
32	33	application_	REGION_RAT	Our rating of the region where client lives (1,2,3)	
33	34	application_	REGION_RAT	Our rating of the region where client lives with taking city into account (1,2,3)	
34	35	application_	WEEKDAY_A	On which day of the week did the client apply for the loan	
35	36	application_	HOUR_APPR	Approximate	rounded
36	37	application_	REG_REGIOI	Flag if client's permanent address does not match contact address (1=different, 0=same, at region level)	
37	38	application_	REG_REGIOI	Flag if client's permanent address does not match work address (1=different, 0=same, at region level)	

Application train

In [34]:

```
ls -l ../../../Data/home-credit-default-risk/application_train.csv
```

```
total 705536
drwx------ 12 root root       384 Nov  9 02:26 home-credit-default-risk/
-rw-r--r--  1 root root 721616255 Nov  9 02:19 home-credit-default-risk.zip
```

In [35]:
```
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 keep track
ds_name = 'application_train'
DATA_DIR=f"{DATA_DIR}/home-credit-default-risk/"
datasets[ds_name] = load_data(os.path.join(DATA_DIR, f'{ds_name}.csv'), ds_n

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[35]:
```
(307511, 122)
```

In [63]:
```
DATA_DIR
```

Out[63]:
```
'../../../Data/home-credit-default-risk/home-credit-default-risk/'
```

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 [36]:
```
ds_name = 'application_test'
datasets[ds_name] = load_data(os.path.join(DATA_DIR, f'{ds_name}.csv'), ds_n
```

```
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_REA
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 [37]:
```
%%time
ds_names = ("application_train", "application_test", "bureau","bureau_balanc
            "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'),
```

```
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_REA
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	CREDI
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
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_FU
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 38.9 s, sys: 18.4 s, total: 57.3 s
Wall time: 1min 17s
```

In [38]:
```
for ds_name in datasets.keys():
    print(f'dataset {ds_name:24}: [ {datasets[ds_name].shape[0]:10,}, {datas
```

```
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 [39]:
```
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 [40]:
```
datasets["application_train"].describe() #numerical only features
```

Out[40]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05
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

8 rows × 106 columns

In [41]:
```
datasets["application_test"].describe() #numerical only features
```

Out [41]:

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY
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 [64]:

```
datasets["application_train"].describe(include='all') #look at all categoric
```

Out [64]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN
count	307511.000000	307511.000000	307511	307511	3
unique	NaN	NaN	2	3	
top	NaN	NaN	Cash loans	F	
freq	NaN	NaN	278232	202448	2(
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

Missing data for application train

In [65]:
```
percent = (datasets["application_train"].isnull().sum()/datasets["applicatic
sum_missing = datasets["application_train"].isna().sum().sort_values(ascendi
missing_application_train_data  = pd.concat([percent, sum_missing], axis=1,
missing_application_train_data.head(20)
```

Out[65]:

	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

In [66]:
```
percent = (datasets["application_test"].isnull().sum()/datasets["application
sum_missing = datasets["application_test"].isna().sum().sort_values(ascendin
missing_application_train_data  = pd.concat([percent, sum_missing], axis=1,
missing_application_train_data.head(20)
```

Out [66]:

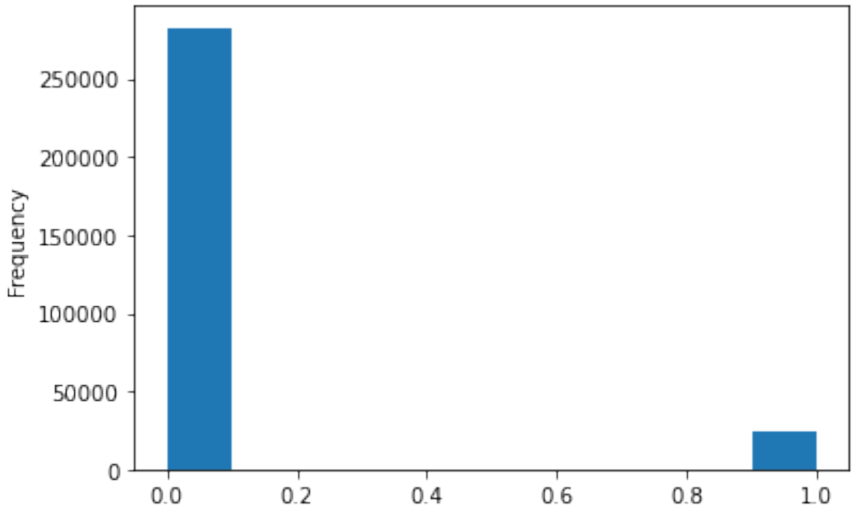
	Percent	Test Missing Count
COMMONAREA_AVG	68.72	33495
COMMONAREA_MODE	68.72	33495
COMMONAREA_MEDI	68.72	33495
NONLIVINGAPARTMENTS_AVG	68.41	33347
NONLIVINGAPARTMENTS_MODE	68.41	33347
NONLIVINGAPARTMENTS_MEDI	68.41	33347
FONDKAPREMONT_MODE	67.28	32797
LIVINGAPARTMENTS_AVG	67.25	32780
LIVINGAPARTMENTS_MODE	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 [68]:

```
import matplotlib.pyplot as plt
%matplotlib inline

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



Correlation with the target column

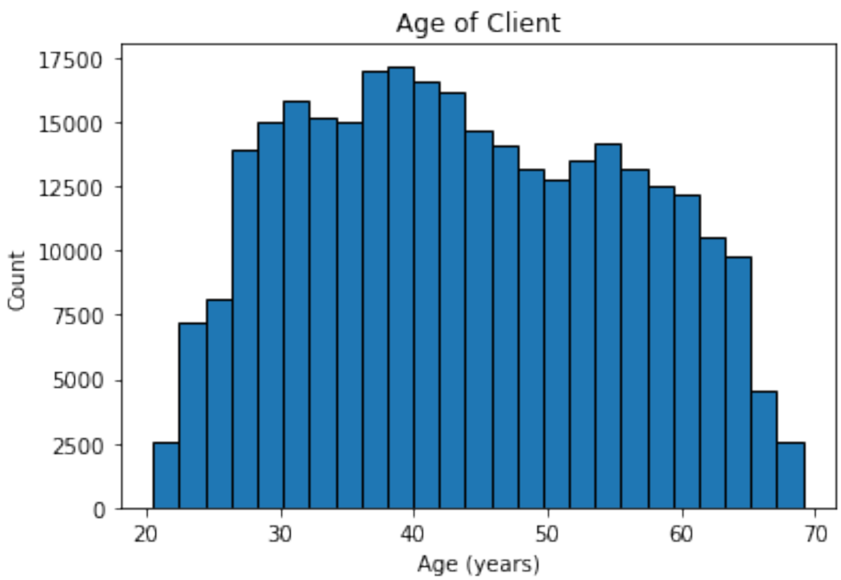
In [69]:
```
correlations = datasets["application_train"].corr()['TARGET'].sort_values()
print('Most Positive Correlations:\n', correlations.tail(10))
print('\nMost Negative Correlations:\n', correlations.head(10))
```

```
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
```

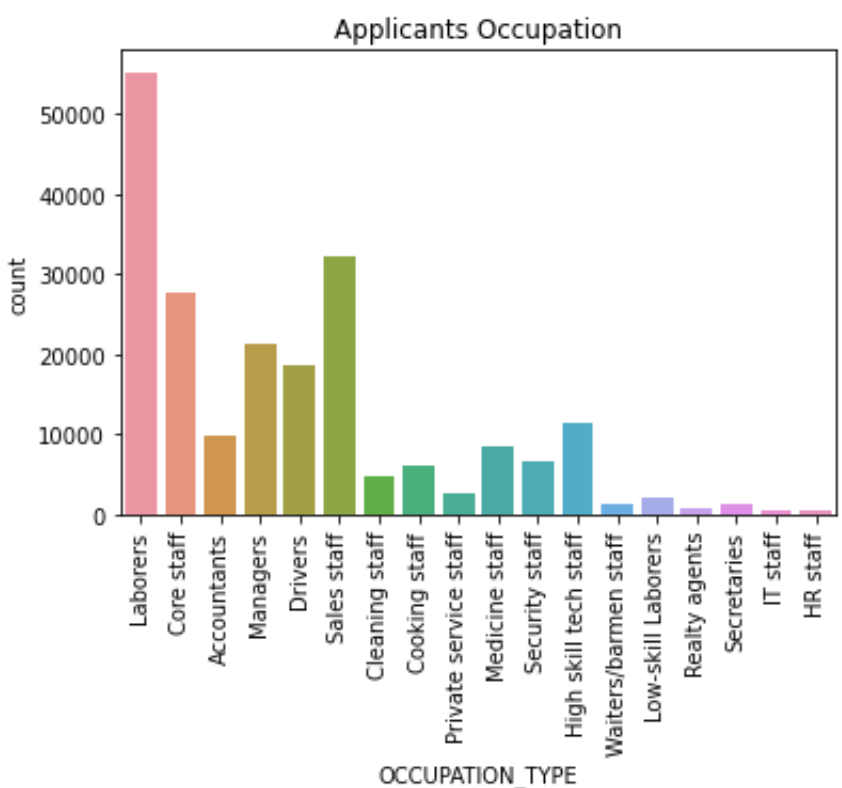
Applicants Age

In [70]:
```
plt.hist(datasets["application_train"]['DAYS_BIRTH'] / -365, edgecolor = 'k'
plt.title('Age of Client'); plt.xlabel('Age (years)'); plt.ylabel('Count');
```



Applicants occupations

In [71]:
```
sns.countplot(x='OCCUPATION_TYPE', data=datasets["application_train"]);
plt.title('Applicants Occupation');
plt.xticks(rotation=90);
```



In [ ]:

In [ ]:

Dataset questions

Unique record for each SK_ID_CURR

In [73]:
```
list(datasets.keys())
```

Out[73]:
```
['application_train',
 'application_test',
 'bureau',
 'bureau_balance',
 'credit_card_balance',
 'installments_payments',
 'previous_application',
 'POS_CASH_balance']
```

In [74]:
```
len(datasets["application_train"]["SK_ID_CURR"].unique()) == datasets["appli
```

Out[74]:
```
True
```

In [75]:
```
# is there an overlap between the test and train customers
np.intersect1d(datasets["application_train"]["SK_ID_CURR"], datasets["applic
```

Out[75]:
```
array([], dtype=int64)
```

In [76]:
```
#
datasets["application_test"].shape
```

Out[76]:
```
(48744, 121)
```

In [77]:
```
datasets["application_train"].shape
```

Out[77]:
```
(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 [81]:
```
appsDF = datasets["previous_application"]
display(appsDF.head())
print(f"{appsDF.shape[0]:,} rows, {appsDF.shape[1]:,} columns")
```

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION
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

```
1,670,214 rows, 37 columns
```

In [90]:
```
print(f"There are  {appsDF.shape[0]:,} previous applications")
```

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

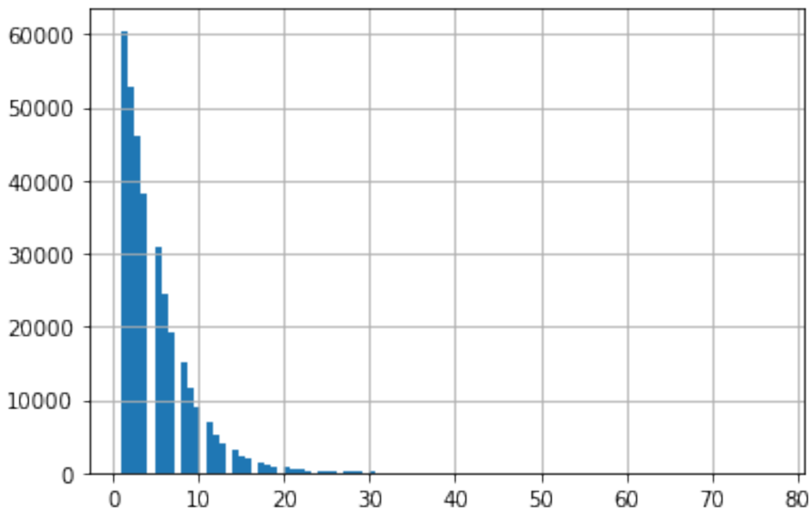
In [88]:
```
#Find the intersection of two arrays.
print(f'Number of train applicants with previous applications is {len(np.int
```

```
Number of train applicants with previous applications 291,057
```

In [89]:
```
#Find the intersection of two arrays.
print(f'Number of train applicants with previous applications is {len(np.int
```

```
Number of train applicants with previous applications is 47,800
```

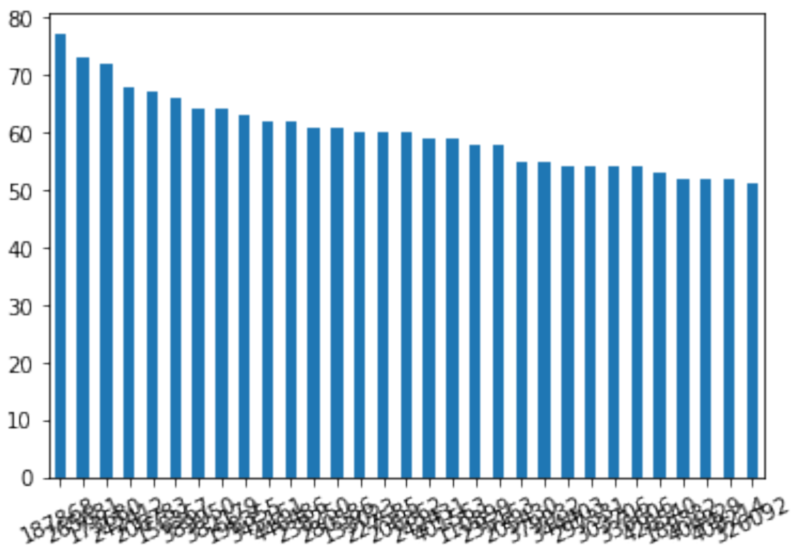
In [99]:
```
# How many previous applciations  per applicant in the previous_application
prevAppCounts = appsDF['SK_ID_CURR'].value_counts(dropna=False)
len(prevAppCounts[prevAppCounts >40])  #more that 40 previous applications
plt.hist(prevAppCounts[prevAppCounts>=0], bins=100)
plt.grid()
```



In [ ]:

In [100…

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



Histogram of Number of previous applications for an ID

In [101…

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

Out[101]:

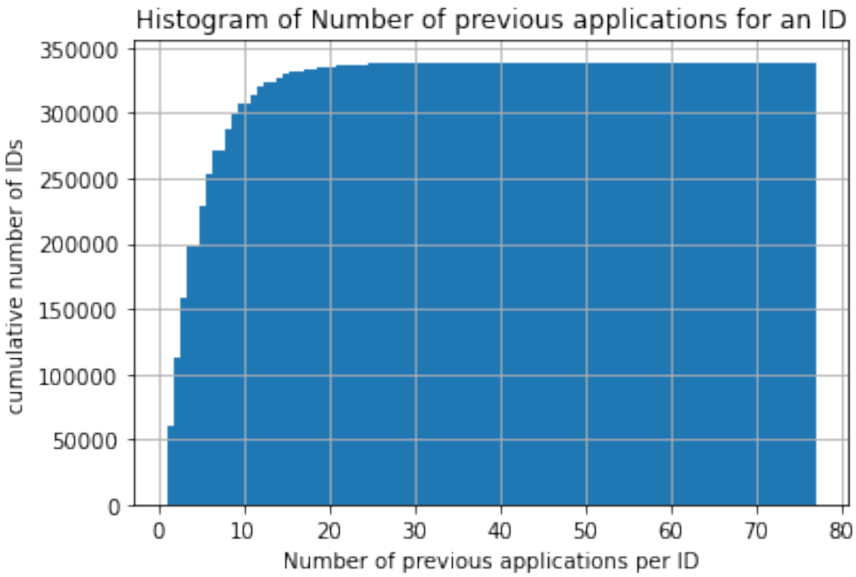
```
60458
```

In [102…

```
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[102]:

```
Text(0.5, 1.0, 'Histogram of Number of previous applications for an ID')
```



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 [103…

```
apps_all = appsDF['SK_ID_CURR'].nunique()
apps_5plus = appsDF['SK_ID_CURR'].value_counts()>=5
apps_40plus = appsDF['SK_ID_CURR'].value_counts()>=40
print('Percentage with 10 or more previous apps:', np.round(100.*(sum(apps_5
print('Percentage with 40 or more previous apps:', np.round(100.*(sum(apps_4
```

```
Percentage with 10 or more previous apps: 41.76895
Percentage with 40 or more previous apps: 0.03453
```

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.

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

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

Aggregate using one or more operations over the specified axis.

In [105…

```
df = pd.DataFrame([[1, 2, 3],
                    [4, 5, 6],
                   [7, 8, 9],
                   [np.nan, np.nan, np.nan]],
                   columns=['A', 'B', 'C'])
display(df)
```

	A	B	C
0	1.0	2.0	3.0
1	4.0	5.0	6.0
2	7.0	8.0	9.0
3	NaN	NaN	NaN

In [106…
```
df.agg({'A' : ['sum', 'min'], 'B' : ['min', 'max']})
#         A    B
#max    NaN  8.0
#min    1.0  2.0
#sum   12.0  NaN
```

Out[106]:

	A	B
sum	12.0	NaN
min	1.0	2.0
max	NaN	8.0

In [107…
```
df = pd.DataFrame({'A': [1, 1, 2, 2],
                   'B': [1, 2, 3, 4],
                   'C': np.random.randn(4)})
display(df)
```

	A	B	C
0	1	1	-1.029921
1	1	2	-2.058339
2	2	3	-0.683242
3	2	4	-1.904689

In [108…
```
# group by column A:
df.groupby('A').agg({'B': ['min', 'max'], 'C': 'sum'})
#    B             C
#  min max       sum
#A
#1    1    2  0.590716
#2    3    4  0.704907
```

Out[108]:

	B		C
	min	**max**	**sum**
A			
1	1	2	-3.08826
2	3	4	-2.58793

In [109…
```
appsDF.columns
```

Out[109]:
```
Index(['SK_ID_PREV', 'SK_ID_CURR', 'NAME_CONTRACT_TYPE', 'AMT_ANNUITY',
       'AMT_APPLICATION', 'AMT_CREDIT', 'AMT_DOWN_PAYMENT', 'AMT_GOODS_PRIC
E',
       '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')
```

In [ ]:

In [110…
```
funcs = ["a","b","c"]
{f:f"{f}_max" for f in funcs}
```

Out[110]:
```
{'a': 'a_max', 'b': 'b_max', 'c': 'c_max'}
```

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.

You must use the following operators with pandas:

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

In [111…
```
appsDF[0:50][(appsDF["SK_ID_CURR"]==175704)]
```

Out[111]:

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION
6	2315218	175704	Cash loans	NaN	0.0

1 rows × 37 columns

In [112…
```
appsDF[0:50][(appsDF["SK_ID_CURR"]==175704)]["AMT_CREDIT"]
```
Out[112]:
```
6    0.0
Name: AMT_CREDIT, dtype: float64
```

In [113…
```
appsDF[0:50][(appsDF["SK_ID_CURR"]==175704) & ~(appsDF["AMT_CREDIT"]==1.0)]
```
Out[113]:

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION
6	2315218	175704	Cash loans	NaN	0.0

1 rows × 37 columns

Missing values in prevApps

In [114…
```
appsDF.isna().sum()
```

Out[114]:

```
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 [115…

```
appsDF.columns
```

Out[115]:
```
Index(['SK_ID_PREV', 'SK_ID_CURR', 'NAME_CONTRACT_TYPE', 'AMT_ANNUITY',
       'AMT_APPLICATION', 'AMT_CREDIT', 'AMT_DOWN_PAYMENT', 'AMT_GOODS_PRIC
E',
       '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 [125…
```
appsDF[agg_op_features].head()
```

Out[125]:

	AMT_ANNUITY	AMT_APPLICATION
0	1730.430	17145.0
1	25188.615	607500.0
2	15060.735	112500.0
3	47041.335	450000.0
4	31924.395	337500.0

The groupby output will have an index or multi-index on rows corresponding to your chosen grouping variables. To avoid setting this index, pass “as_index=False” to the groupby operation.

```
import pandas as pd
import dateutil

# Load data from csv file
data = pd.DataFrame.from_csv('phone_data.csv')
# Convert date from string to date times
data['date'] = data['date'].apply(dateutil.parser.parse,
dayfirst=True)

data.groupby('month', as_index=False).agg({"duration": "sum"})
```

Pandas `reset_index()` to convert Multi-Index to Columns We can simplify the multi-index dataframe using reset_index() function in Pandas. By default, Pandas reset_index() converts the indices to columns.

Fixing Column names after Pandas agg() function to summarize grouped data

Since we have both the variable name and the operation performed in two rows in the Multi-Index dataframe, we can use that and name our new columns correctly.

For more details unstacking groupby results and examples please see here

For more details and examples please see here

In [165…

```
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 b
display(result.head())
print("-"*50)
result = appsDF.groupby(["SK_ID_CURR"], as_index=False).agg({'AMT_ANNUITY' :
result.columns = result.columns.map('_'.join)
display(result)
result['range_AMT_APPLICATION'] = result['AMT_APPLICATION_max'] - result['AM
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_DOW
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
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[165]:

	SK_ID_CURR_	AMT_ANNUITY_min	AMT_ANNUITY_max	AMT_ANNUITY_mean	AMT_APP
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 [166…

```
result.isna().sum()
```

Out[166]:

```
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
```

feature transformer for prevApp table

In [ ]:
```
# Create aggregate features (via pipeline)
class prevAppsFeaturesAggregater(BaseEstimator, TransformerMixin):
    def __init__(self, features=None): # no *args or **kargs
        self.features = features
        self.agg_op_features = {}
        for f in features:
            self.agg_op_features[f] = {f"{f}_{func}":func for func in ["min"

    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()
        result = X.groupby(["SK_ID_CURR"]).agg(self.agg_op_features)
        result.columns = result.columns.droplevel()
        result = result.reset_index(level=["SK_ID_CURR"])
        result['range_AMT_APPLICATION'] = result['AMT_APPLICATION_max'] - re
        return result # return dataframe with the join key "SK_ID_CURR"


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(features))
    return(test_pipeline.fit_transform(df))

features = ['AMT_ANNUITY', 'AMT_APPLICATION']
features = ['AMT_ANNUITY',
       'AMT_APPLICATION', 'AMT_CREDIT', 'AMT_DOWN_PAYMENT', 'AMT_GOODS_PRICE
       'RATE_DOWN_PAYMENT', 'RATE_INTEREST_PRIMARY',
       'RATE_INTEREST_PRIVILEGED', 'DAYS_DECISION', 'NAME_PAYMENT_TYPE',
       'CNT_PAYMENT',
       'DAYS_FIRST_DRAWING', 'DAYS_FIRST_DUE', 'DAYS_LAST_DUE_1ST_VERSION',
       '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 staff != 'Sa
```

Join the labeled dataset

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

In [136…
```
datasets.keys()
```

Out[136]:
```
dict_keys(['application_train', 'application_test', 'bureau', 'bureau_balan
ce', 'credit_card_balance', 'installments_payments', 'previous_application'
, 'POS_CASH_balance'])
```

In [ ]:
```
features = ['AMT_ANNUITY', 'AMT_APPLICATION']
prevApps_feature_pipeline = Pipeline([
        ('prevApps_add_features1', prevApps_add_features1()),  # add some ne
        ('prevApps_add_features2', prevApps_add_features2()),  # add some ne
        ('prevApps_aggregater', prevAppsFeaturesAggregater()), # Aggregate a
    ])


X_train= datasets["application_train"] #primary dataset
appsDF = datasets["previous_application"] #prev app


merge_all_data = False

# transform all the secondary tables
# 'bureau', 'bureau_balance', 'credit_card_balance', 'installments_payments'
# 'previous_application', 'POS_CASH_balance'

if merge_all_data:
    prevApps_aggregated = prevApps_feature_pipeline.transform(appsDF)

    #'bureau', 'bureau_balance', 'credit_card_balance', 'installments_paymen
    # 'previous_application', 'POS_CASH_balance'

# merge primary table and secondary tables using features based on meta data
if merge_all_data:
    # 1. Join/Merge in prevApps Data
    X_train = X_train.merge(prevApps_aggregated, how='left', on='SK_ID_CURR'

    # 2. Join/Merge in ...... Data
    #X_train = X_train.merge(...._aggregated, how='left', on="SK_ID_CURR")

    # 3. Join/Merge in .....Data
    #dX_train = X_train.merge(...._aggregated, how='left', on="SK_ID_CURR")

    # 4. Join/Merge in Aggregated ...... Data
    #X_train = X_train.merge(...._aggregated, how='left', on="SK_ID_CURR")

    # ......
```

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

In [ ]:
```
X_kaggle_test= datasets["application_test"]
if merge_all_data:
    # 1. Join/Merge in prevApps Data
    X_kaggle_test = X_kaggle_test.merge(prevApps_aggregated, how='left', on=

    # 2. Join/Merge in ...... Data
    #X_train = X_train.merge(...._aggregated, how='left', on="SK_ID_CURR")

    # 3. Join/Merge in .....Data
    #df_labeled = df_labeled.merge(...._aggregated, how='left', on="SK_ID_CU

    # 4. Join/Merge in Aggregated ...... Data
    #df_labeled = df_labeled.merge(...._aggregated, how='left', on="SK_ID_CU

    # ......
```

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

In [ ]:
```
# Convert categorical features to numerical approximations (via pipeline)
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
          '1- 2 weeks': 11, '2- 4 weeks': 21, '4- 8 weeks': 42, '26+ weeks':
        X['PayDelay'] = X['PayDelay'].apply(lambda x: int(x) if x != '162+'
        X['DSFS'] = X['DSFS'].apply(lambda x: None if pd.isnull(x) else int(
        X['CharlsonIndex'] = X['CharlsonIndex'].apply(lambda x: charlson_idx
        X['LengthOfStay'] = X['LengthOfStay'].apply(lambda x: None if pd.isn
        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_CONTRACT_TYPE',

'NAME_EDUCATION_TYPE','OCCUPATION_TYPE','NAME_INCOME_TYPE']

# Notice handle_unknown="ignore" in OHE which ignore values from
the validation/test that
# do NOT occur in the training set
cat_pipeline = Pipeline([
        ('selector', DataFrameSelector(cat_attribs)),
        ('imputer', SimpleImputer(strategy='most_frequent')),
        ('ohe', OneHotEncoder(sparse=False,
handle_unknown="ignore"))
    ])
```

In [208…
```
# load data
df = pd.read_csv('chronic_kidney_disease.csv', header="infer")
# names=['age', 'bp', 'sg', 'al', 'su', 'rbc', 'pc', 'pcc', 'ba', 'bgr', 'bu
# 'hemo', 'pcv', 'wc', 'rc', 'htn', 'dm', 'cad', 'appet', 'pe', 'ane', 'clas
# head of df
df.head(10)
```

Out[208]:

	age	bp	sg	al	su	rbc	pc	pcc	ba	bgr	...	pcv	wbcc
0	48	80	1.020	1	0	?	normal	notpresent	notpresent	121	...	44	7800
1	7	50	1.020	4	0	?	normal	notpresent	notpresent	?	...	38	6000
2	62	80	1.010	2	3	normal	normal	notpresent	notpresent	423	...	31	7500
3	48	70	1.005	4	0	normal	abnormal	present	notpresent	117	...	32	6700
4	51	80	1.010	2	0	normal	normal	notpresent	notpresent	106	...	35	7300
5	60	90	1.015	3	0	?	?	notpresent	notpresent	74	...	39	7800
6	68	70	1.010	0	0	?	normal	notpresent	notpresent	100	...	36	?
7	24	?	1.015	2	4	normal	abnormal	notpresent	notpresent	410	...	44	6900
8	52	100	1.015	3	0	normal	abnormal	present	notpresent	138	...	33	9600
9	53	90	1.020	2	0	abnormal	abnormal	present	notpresent	70	...	29	12100

10 rows × 25 columns

In [211…
```
# Categorical boolean mask
categorical_feature_mask = df.dtypes==object
categorical_feature_mask
```

Out[211]:

```
age      True
bp       True
sg       True
al       True
su       True
rbc      True
pc       True
pcc      True
ba       True
bgr      True
bu       True
sc       True
sod      True
pot      True
hemo     True
pcv      True
wbcc     True
rbcc     True
htn      True
dm       True
cad      True
appet    True
pe       True
ane      True
class    True
dtype: bool
```

In [209…

```
# filter categorical columns using mask and turn it into a list
categorical_cols = X.columns[categorical_feature_mask].tolist()
categorical_cols
```

```
---------------------------------------------------------------------------
AttributeError                            Traceback (most recent call last)
<ipython-input-209-4e95be920e1e> in <module>
      1 # Categorical boolean mask
----> 2 categorical_feature_mask = X.dtypes==object
      3 # filter categorical columns using mask and turn it into a list
      4 categorical_cols = X.columns[categorical_feature_mask].tolist()
      5 categorical_cols

AttributeError: 'list' object has no attribute 'dtypes'
```

In [203…

```
from sklearn.preprocessing import OneHotEncoder
import pandas as pd
categorical_feature_mask = [True, False]
# instantiate OneHotEncoder
enc = OneHotEncoder(categorical_features = categorical_feature_mask,sparse =
# categorical_features = boolean mask for categorical columns
# sparse = False output an array not sparse matrix
X_train = pd.DataFrame([['small', 1], ['small', 3], ['medium', 3], ['large',
X_test = [['small', 1.2],  ['medium', 4], ['EXTRA-large', 2]]
print(f"X_train:\n{X_train}")
print(f"enc.fit_transform(X_train):\n{enc.fit_transform(X_train)}")
print(f"enc.transform(X_test):\n{enc.transform(X_test)}")

print(f"enc.get_feature_names():\n{enc.get_feature_names()}")
```

```
X_train:
        0  1
0   small  1
1   small  3
2  medium  3
3   large  2
```

```
---------------------------------------------------------------------------
ValueError                                Traceback (most recent call last)
<ipython-input-203-7734cfc72489> in <module>
      9 X_test = [['small', 1.2],  ['medium', 4], ['EXTRA-large', 2]]
     10 print(f"X_train:\n{X_train}")
---> 11 print(f"enc.fit_transform(X_train):\n{enc.fit_transform(X_train)}")
     12 print(f"enc.transform(X_test):\n{enc.transform(X_test)}")
     13 

/usr/local/lib/python3.6/site-packages/sklearn/preprocessing/_encoders.py in
fit_transform(self, X, y)
    512             return _transform_selected(
    513                 X, self._legacy_fit_transform, self.dtype,
--> 514                 self._categorical_features, copy=True)
    515         else:
    516             return self.fit(X).transform(X)

/usr/local/lib/python3.6/site-packages/sklearn/preprocessing/base.py in _tra
nsform_selected(X, transform, dtype, selected, copy, retain_order)
     43     Xt : array or sparse matrix, shape=(n_samples, n_features_new)
     44     """
---> 45     X = check_array(X, accept_sparse='csc', copy=copy, dtype=FLOAT_D
TYPES)
     46 
     47     if sparse.issparse(X) and retain_order:

/usr/local/lib/python3.6/site-packages/sklearn/utils/validation.py in check_
array(array, accept_sparse, accept_large_sparse, dtype, order, copy, force_a
ll_finite, ensure_2d, allow_nd, ensure_min_samples, ensure_min_features, war
n_on_dtype, estimator)
    525             try:
    526                 warnings.simplefilter('error', ComplexWarning)
--> 527                 array = np.asarray(array, dtype=dtype, order=order)
    528             except ComplexWarning:
    529                 raise ValueError("Complex data not supported\n"

/usr/local/lib/python3.6/site-packages/numpy/core/numeric.py in asarray(a, d
type, order)
    499 
    500     """
--> 501     return array(a, dtype, copy=False, order=order)
    502 
    503 

ValueError: could not convert string to float: 'small'
```

In [ ]:
```
print(f"enc.categories_{enc.categories_}")

print(f"enc.categories_{enc.categories_}")
enc.transform([['Female', 1], ['Male', 4]]).toarray()


enc.inverse_transform([[0, 1, 1, 0, 0], [0, 0, 0, 1, 0]])


enc.get_feature_names()
```

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

In [184…
```
from sklearn.datasets import load_breast_cancer
data = load_breast_cancer(return_X_y=False)
X, y = load_breast_cancer(return_X_y=True)
print(y[[10, 50, 85]])
#([0, 1, 0])
list(data.target_names)
#['malignant', 'benign']
X.shape
```

```
[0 1 0]
```

Out[184]:
```
(569, 30)
```

In [186…
```
data.feature_names
```

Out[186]:
```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23')
```

Please this blog for more details of OHE when the validation/test have previously unseen unique values.

HCDR preprocessing

In [ ]:
```
# 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

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_EMPLOYED','
        'EXT_SOURCE_2','EXT_SOURCE_3','CODE_GENDER', 'FLAG_OWN_REALTY','FLAG
                   'NAME_EDUCATION_TYPE','OCCUPATION_TYPE','NAME_INCOME_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,
    X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, te
    X_kaggle_test= datasets["application_test"][selected_features]
    # y_test = datasets["application_test"]['TARGET']   #why no  TARGET?!! (

selected_features = ['AMT_INCOME_TOTAL',  'AMT_CREDIT','DAYS_EMPLOYED','DAYS
        'EXT_SOURCE_2','EXT_SOURCE_3','CODE_GENDER', 'FLAG_OWN_REALTY','FLAG
                   'NAME_EDUCATION_TYPE','OCCUPATION_TYPE','NAME_INCOME_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_train, test
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_s
X_kaggle_test= X_kaggle_test[selected_features]
# y_test = datasets["application_test"]['TARGET']   #why no  TARGET?!! (hint


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}")
```

In [ ]:
```
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 it
#
# ['release_month','release_day','release_year', 'release_dayofweek','releas
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()
        df['OCCUPATION_TYPE'] = df['OCCUPATION_TYPE'].apply(lambda x: 1. if
        #df.drop(self.features, axis=1, inplace=True)
        return np.array(df.values)  #return a Numpy Array to observe the pip


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 staff != 'Sa
```

In [ ]:
```
# 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
```

In [ ]:
```
# Identify the numeric features we wish to consider.
num_attribs = [
    'AMT_INCOME_TOTAL',  'AMT_CREDIT','DAYS_EMPLOYED','DAYS_BIRTH','EXT_SOUR
    '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_CONTRAC
               'NAME_EDUCATION_TYPE','OCCUPATION_TYPE','NAME_INCOME_TYPE']

# Notice handle_unknown="ignore" in OHE which ignore values from the validat
# do NOT occur in the training set
cat_pipeline = Pipeline([
        ('selector', DataFrameSelector(cat_attribs)),
        #('imputer', SimpleImputer(strategy='most_frequent')),
        ('imputer', SimpleImputer(strategy='constant', fill_value='missing'))
        ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
    ])

data_prep_pipeline = FeatureUnion(transformer_list=[
        ("num_pipeline", num_pipeline),
        ("cat_pipeline", cat_pipeline),
    ])
```

In [ ]:
```
list(datasets["application_train"].columns)
```

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 [ ]:
```
def pct(x):
    return round(100*x,3)
```

In [ ]:
```
try:
    expLog
except NameError:
    expLog = pd.DataFrame(columns=["exp_name",
                                   "Train Acc",
                                   "Valid Acc",
                                   "Test  Acc",
                                   "Train AUC",
                                   "Valid AUC",
                                   "Test  AUC"
                                  ])
```

In [ ]:
```
%%time
np.random.seed(42)
full_pipeline_with_predictor = Pipeline([
        ("preparation", data_prep_pipeline),
        ("linear", LogisticRegression())
    ])
model = full_pipeline_with_predictor.fit(X_train, y_train)
```

In [ ]:
```
from sklearn.metrics import accuracy_score

np.round(accuracy_score(y_train, model.predict(X_train)), 3)
```

Evaluation metrics

Submissions are evaluated on area under the ROC curve between the predicted probability and the observed target.

The SkLearn `roc_auc_score` function computes the area under the receiver operating characteristic (ROC) curve, which is also denoted by AUC or AUROC. By computing the area under the roc curve, the curve information is summarized in one number.

```
from sklearn.metrics import roc_auc_score
>>> y_true = np.array([0, 0, 1, 1])
>>> y_scores = np.array([0.1, 0.4, 0.35, 0.8])
>>> roc_auc_score(y_true, y_scores)
0.75
```

In [ ]:
```
from sklearn.metrics import roc_auc_score
roc_auc_score(y_train, model.predict_proba(X_train)[:, 1])
```

In [ ]:
```
exp_name = f"Baseline_{len(selected_features)}_features"
expLog.loc[len(expLog)] = [f"{exp_name}"] + list(np.round(
               [accuracy_score(y_train, model.predict(X_train)),
                accuracy_score(y_valid, model.predict(X_valid)),
                accuracy_score(y_test, model.predict(X_test)),
                roc_auc_score(y_train, model.predict_proba(X_train)[:, 1]),
                roc_auc_score(y_valid, model.predict_proba(X_valid)[:, 1]),
                roc_auc_score(y_test, model.predict_proba(X_test)[:, 1])],
    4))
expLog
```

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 [ ]:
```
test_class_scores = model.predict_proba(X_kaggle_test)[:, 1]
```

In [ ]:
```
test_class_scores[0:10]
```

In [ ]:
```
# Submission dataframe
submit_df = datasets["application_test"][['SK_ID_CURR']]
submit_df['TARGET'] = test_class_scores

submit_df.head()
```

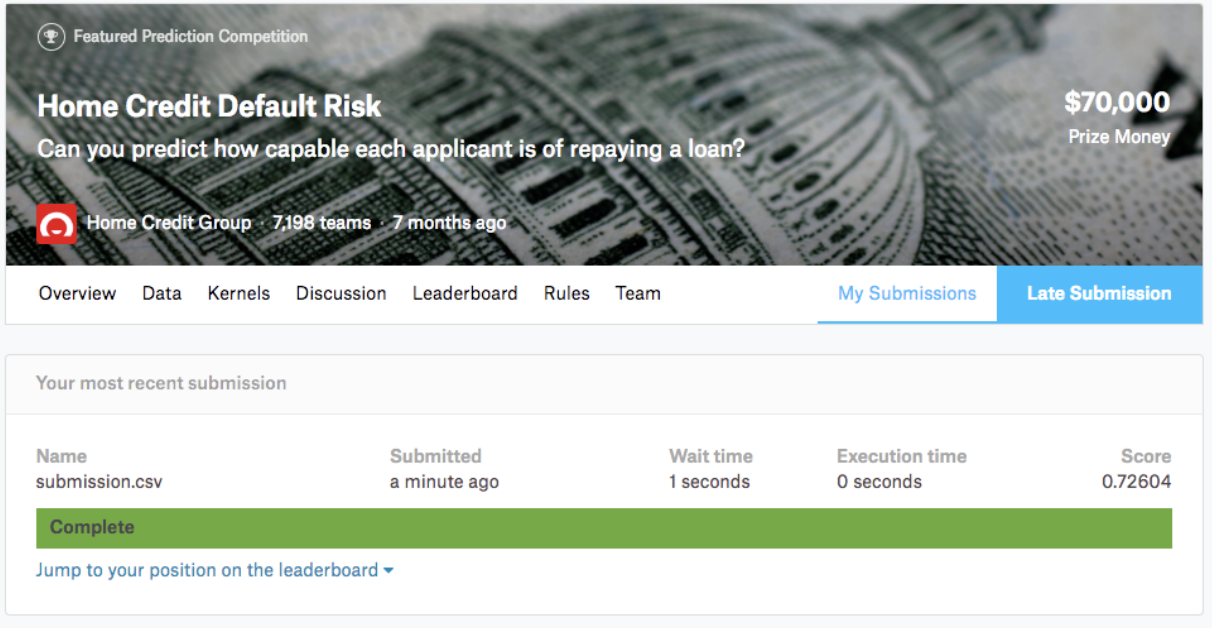
In [ ]:
```
submit_df.to_csv("submission.csv",index=False)
```

Kaggle submission via the command line API

In [ ]:
```
! kaggle competitions submit -c home-credit-default-risk -f submission.csv -
```

report submission

Click on this link



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

Please provide an abstract summarizing the work you did (150 words)

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

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

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

Discussion

Discuss & analyze your different experimental results

Please include code sections when necessary as well as images or any relevant material

Conclusion

Kaggle Submission

Please provide a screenshot of your best kaggle submission.
The screenshot should show the different details of the submission and not just the score.

References

Some of the material in this notebook has been adopted from here

In [2]:
```
nbconvert --to pdf --allow-chromium-download Phase2.ipynb
```

```
  File "/var/folders/41/xnpszdmx6k35m1m215kf2tmh0000gn/T/ipykernel_46743/332
4159810.py", line 1
    nbconvert --to pdf --allow-chromium-download Phase2.ipynb
                   ^
SyntaxError: invalid syntax
```

TODO: Predicting Loan Repayment with Automated Feature Engineering in Featuretools

Read the following:

- feature engineering via Featuretools library:
  - https://github.com/Featuretools/predict-loan-repayment/blob/master/Automated%20Loan%20Repayment.ipynb
- https://www.analyticsvidhya.com/blog/2018/08/guide-automated-feature-engineering-featuretools-python/
- feature engineering paper: https://dai.lids.mit.edu/wp-content/uploads/2017/10/DSAA_DSM_2015.pdf
- https://www.analyticsvidhya.com/blog/2017/08/catboost-automated-categorical-data/