A

Project Proposal

on

Home Credit Default Risk (HCDR)

Developed at

Luddy School of Informatics, Computing, and Engineering

Developed by

Aarushi Dua (aarudua@iu.edu)

Sai Teja Burla (saburla@iu.edu)

Lakshay Madaan (lmadaan@iu.edu)

Shyam Makwana (smakwana@iu.edu)

Guided By

James Shanahan (jshanah@indiana.edu)

700 N Woodlawn Ave, Bloomington, IN 47408

Spring - 2023

Group Name: FP_Group22

Team Members:

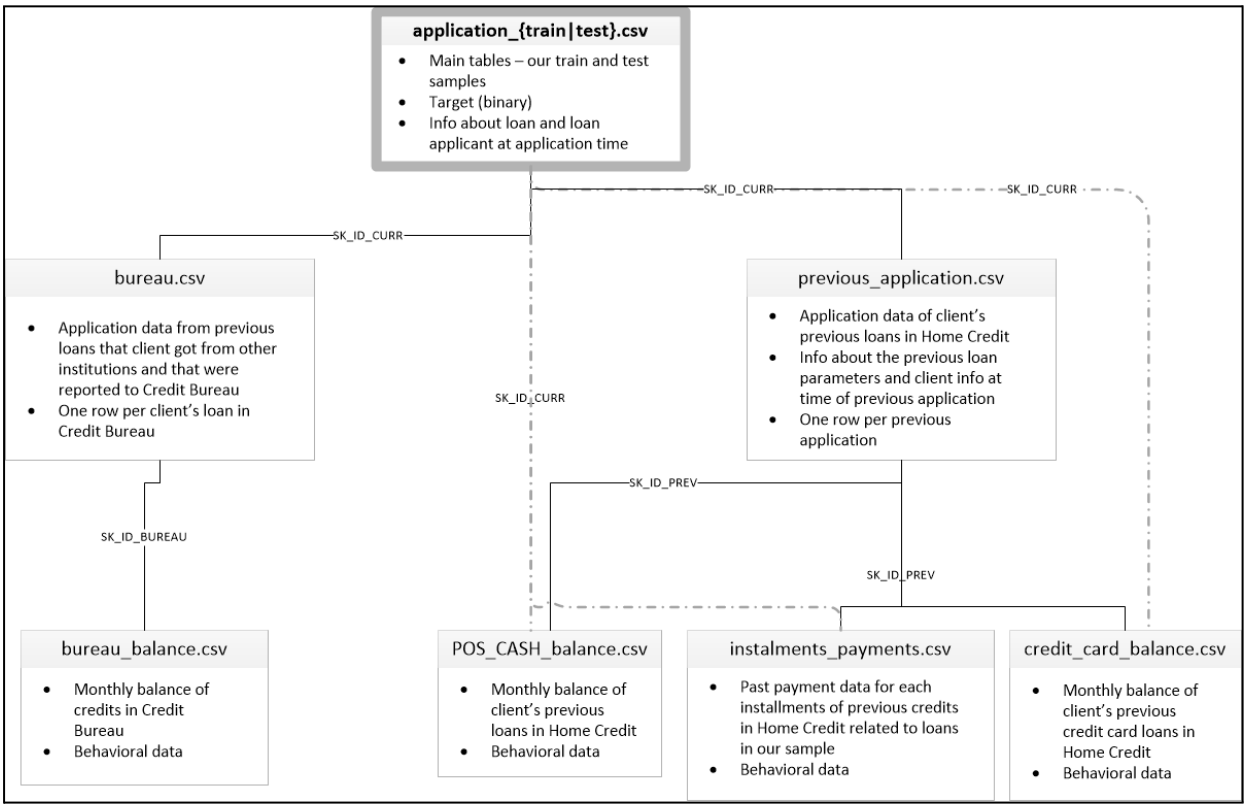
1. Aarushi Dua (aarudua@iu.edu)
2. Sai Teja Burla (saburla@iu.edu)
3. Lakshay Madaan (lmadaan@iu.edu)
4. Shyam Makwana (smakwana@iu.edu)

Team Photo:

Abstract

Home Credit is looking to develop a Machine Learning Pipeline that could aid in making accurate lending decisions with respect to clients who have insufficient or non-existent credit history and make their overall experience good. The provided dataset has a total of 7 tables which also includes telco and transactional information of various clients. The first step would be to combine these tables and convert them into one meaningful dataset. Next we will perform exploratory data analysis to understand the data. We then make pipelines; numeric and categorical; to deal with missing values and categorical data and to perform scaling, etc. and make the data ready for further use. After this we combine the two pipelines and as part of these pipelines we also implement various binary classifiers such as logistic regression, support vector classification, gaussian naive bayes, random forest classification, decision tree classification. Finally, to choose the best model among the various models that we implemented, we make use of various evaluation metrics such as confusion matrix, F1 score, precision, recall, accuracy score, and roc curve.

Data Description



1. application_{train|test}.csv

The primary table consists of the necessary information regarding home credit loan applications.

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

```
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_OWN_REALTY
0	100002	1	Cash loans	M	N	Y
1	100003	0	Cash loans	F	N	N
2	100004	0	Revolving loans	M	Y	Y
3	100006	0	Cash loans	F	N	Y
4	100007	0	Cash loans	M	N	Y

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_REALTY	CNT_CHILDREN
0	100001	Cash loans	F	N	Y	0
1	100005	Cash loans	M	N	Y	0
2	100013	Cash loans	M	Y	Y	0
3	100028	Cash loans	F	N	Y	2
4	100038	Cash loans	M	Y	N	1

5 rows × 121 columns

2. bureau.csv

This is a record of how well the client has paid back loans and borrowed money in the past, as reported by other banks or lenders.

All client's previous credits provided by other financial institutions were reported to the Credit Bureau (for clients who have a loan in our sample).

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

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

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

3. bureau_balance.csv

This has information about the amounts you owed and paid back each month in the past, which is recorded by a company that keeps track of people's credit histories.

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

```
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

4. POS_CASH_balance.csv

This table shows how much money the client owes to Home Credit every month from their previous purchases made using credit or cash loans.

Monthly balance snapshots of previous POS (point of sales) and cash loans that the applicant had with Home Credit.

This table has one row for each month of history of every previous credit in Home Credit (consumer credit and cash loans) related to loans in our sample – i.e. the table has (#loans in sample * # of relative previous credits * # of months in which we have some history observable for the previous credits) rows.

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

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

5. credit_card_balance.csv

This table shows how much money the applicant owed on their Home Credit credit cards in past months.

Monthly balance snapshots of previous credit cards that the applicant has with Home Credit.

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

```
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_ACTUAL
0	2562384	378907	-6	56.970	135000
1	2582071	363914	-1	63975.555	45000
2	1740877	371185	-7	31815.225	450000
3	1389973	337855	-4	236572.110	225000
4	1891521	126868	-1	453919.455	450000

5 rows × 23 columns

6. previous_application.csv

All previous applications for Home Credit loans of clients who have loans in our sample.

There is one row for each previous application related to loans in our data sample.

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

5 rows × 37 columns

7. installments_payments.csv

This shows whether or not you've paid back loans you got from Home Credit in the past.

Repayment history for the previously disbursed credits in Home Credit related to the loans in our sample.

There is a) one row for every payment that was made plus b) one row each for missed payment.

One row is equivalent to one payment of one installment OR one installment corresponding to one payment of one previous Home Credit credit related to loans in our sample.

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

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER	DAYS_INSTALMENT
0	1054186	161674	1.0	6	-1180.0
1	1330831	151639	0.0	34	-2156.0
2	2085231	193053	2.0	1	-63.0
3	2452527	199697	1.0	3	-2418.0
4	2714724	167756	1.0	2	-1383.0

Machine Learning Models and Metrics

Machine Learning algorithms:

The model that we are aiming to build should have the capability to classify the target variable into two classes, i.e. 1 or 0 which are the numeric representation for whether the person who took out a loan will repay it or not based on all the data provided to us in the dataset. For achieving this with accurate results we are planning on trying the following algorithms to see which one provides us with the best outcome:

1. **Logistic Regression** - This is a form of statistical model that is used to examine the correlation between a binary or categorical dependent variable and one or more independent variables. It employs a logistic function to simulate the likelihood that the dependent variable will fall into a certain category. The logistic function converts any input to a number between 0 and 1, which represents the likelihood of the binary result.

2. **Gaussian Naive Bayes** - It is a variant of the Naive Bayes algorithm for classification tasks in machine learning. It is predicated on the Bayes theorem and the supposition of data feature independence. The input characteristics in Gaussian Naive Bayes are thought to have a Gaussian or normal distribution. With the input data, the algorithm determines the likelihood of each class, and then it chooses the class with the highest probability as the anticipated output.

3. **Support Vector Classification** - This model is predicated on the notion of locating a hyperplane that separates the data points of several classes with the greatest possible margin. The support vectors, or data points that are most near the hyperplane are found, and then the model tries to maximize that distance. The output of the model is governed by the sign of the separation between the input data and the hyperplane, and the hyperplane is specified by a set of coefficients that are learnt during training.

4. **Decision Tree Classification** - In this model, based on the characteristics of the input data, a tree-like model of decisions and their potential outcomes is built. The method begins with the complete dataset and chooses the optimal feature to divide the data according to a certain criterion, such Gini Impurity or Information Gain. The data is then divided into subsets depending on the selected feature and the procedure is then performed iteratively for each subset until all the data in each subset belong to the same class, or a stopping requirement is satisfied.

5. **Random Forest Classification** - This model builds numerous decision trees using subsets of the input data and randomly choosing features for each split. Each decision tree only sees a subset of the data since they are trained on a bootstrapped sample of the data. The decision trees' majority vote determines the algorithm's final result.

Description of metrics and analysis:

The evaluation metrics that we want to use to check whether our model is performing well or not are as follows:

1. **Confusion Matrix** - A confusion matrix is a table that is frequently used to assess the effectiveness of a classification model. It is a matrix that compares the predicted labels of the model with the actual labels of the data. The confusion matrix consists of the following cells:
    a. True Positive(TP) - The number of rows that are correctly predicted as positive by the model.
    b. True Negative(TN) - The number of rows that are correctly predicted as negative by the model.
    c. False Positive(FP) - The number of rows that are incorrectly predicted as positive by the model.
    d. False Negative(FN) - The number of rows that are incorrectly predicted as negative by the model.

2. **Classification Report** - An evaluation of a classification model's performance is summarized in a classification report, which is commonly produced using metrics derived from a confusion matrix. A classification report generally includes the following metrics:
    a. **Precision** - The ratio of true positives among all the instances that the model classified as positive.
    Formula: TP / (TP + FP)

    b. **Recall** - The ratio of true positives among all the instances that actually belong to the positive class of the model.
    Formula: TP / (TP + FN)

    c. **F1-Score** - The harmonic mean of precision and recall.
    Formula: 2 * (Precision * Recall) / (Precision + Recall)

3. **Accuracy Score** - A typical performance metric in machine learning for assessing the accuracy of a classification model is the accuracy score. It is defined as the ratio of correctly classified instances out of the total number of instances in the dataset.
   Formula: (TP + TN) / (TP + TN + FP + FN)

4. **ROC Curve** - An ROC curve is a graphical representation of the performance of a binary classification model at different classification thresholds. In machine learning, it is frequently used to assess and contrast the effectiveness of various categorization models. The True Positive Rate and False Positive Rate are plotted on the y-axis and x-axis respectively.

Machine Learning Pipelines:

For solving this problem statement we plan on following the below explained pipeline:

1. **Problem Statement and Requirements** - The problem statement in a Machine Learning (ML) project specifies the particular aim or task that the ML model is meant to address which in this case would be to determine if a client will be able to repay a loan or not. The requirements of the project is to specify what is necessary to create a powerful ML model that solves the given problem. This contains any performance measures that will be monitored, the necessary data, the features to be used, the model architecture, and the method to be used. It should also outline the necessary hardware and software for developing and deploying the ML model.

2. **Data Acquisition** - The data needed for solving this problem statement is available on kaggle as Home Credit Default Risk. The data is divided into 7 csv files that we can use to come up with a solution.

3. **Data Visualization and EDA** - The graphical depiction of data and information is known as data visualization. In order to help illustrate patterns, trends, and correlations within the data, visualizations like charts, graphs, and maps are made. EDA on the other hand is the process of applying statistical and graphical tools to analyze and summarize a dataset's key features. EDA uses a number of approaches, including outlier detection, correlation analysis, and summary statistics. EDA would play a key role in ensuring our data's accuracy and quality and is a crucial stage in creating a machine learning model that works for our problem statement.

4. **Feature Engineering** - Implementing techniques for feature engineering that include combining existing features to create new ones, converting features using

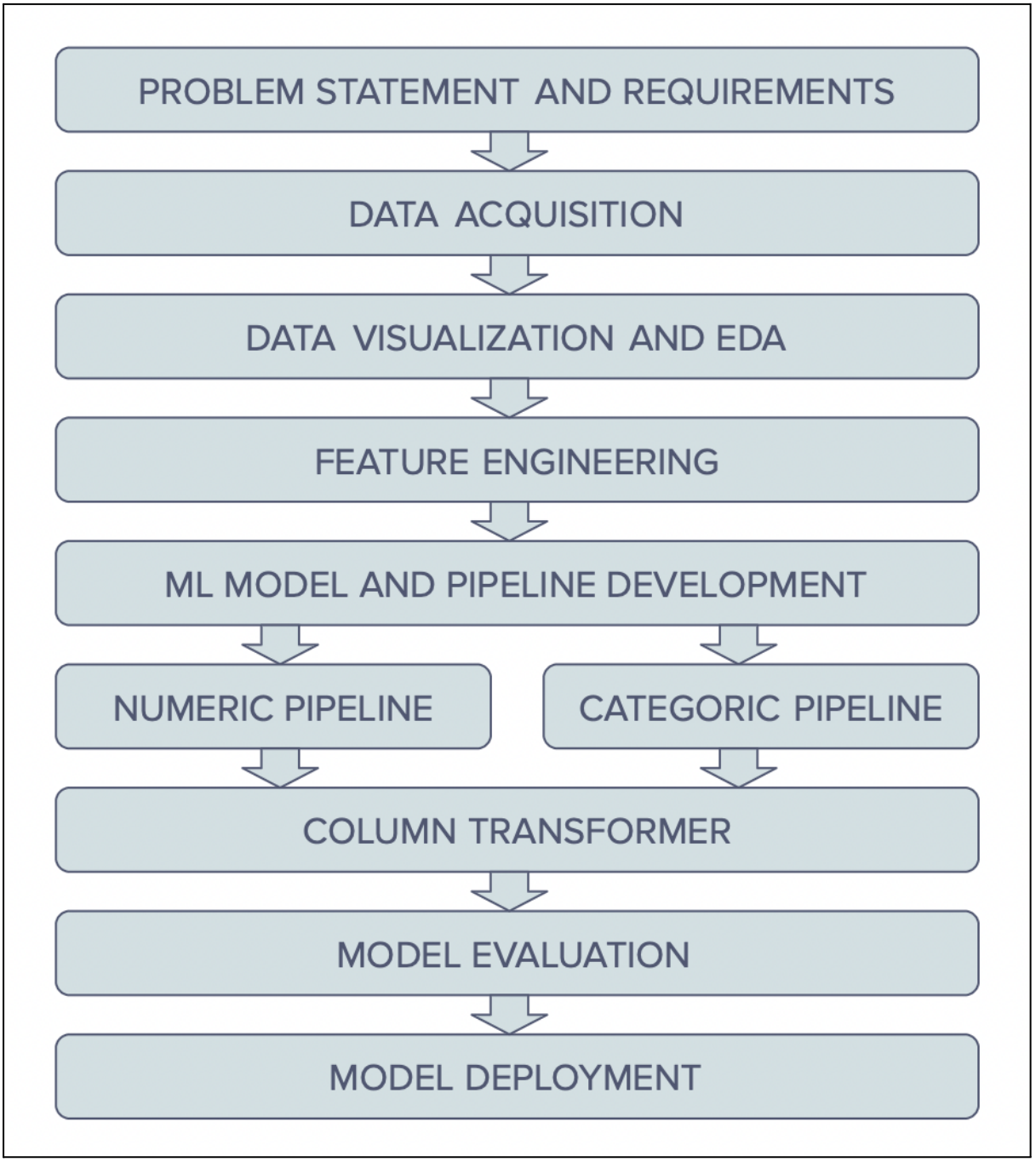
mathematical operations like exponential or logarithmic transformations, and normalizing or standardizing features to guarantee that they are of the same scale. Additional methods include handling missing data, encoding categorical variables, and principal component analysis for dimensionality reduction.

5. **ML Model and Pipeline Development** - The aim in this section is to build various ML models like; logistic regression, support vector classification, gaussian naive bayes, random forest classification, decision tree classification and to build the following three pipelines:
    a. **Numeric Pipeline** - This pipeline works with the numeric part of our data to process it. We plan on making use of SimpleImputer to take care of missing values and we also plan on including StandardScaler transformation for standardizing the data.
    b. **Categoric Pipeline** - This pipeline works with the categorical part of our data to help process it. In this pipeline we plan on making use of SimpleImputer for dealing with missing values by using the concept of mode. We make use of OneHotEncoder to convert categorical data into numeric values. We also make use of scaler to transform the present data.
    c. **Column Transformer** - This is the resultant pipeline that is a combination of the above two numeric and categorical pipelines.

6. **Model Evaluation** - In this step we fit the above made models and pipelines together and train the various models on the training data. We then use various evaluation metrics as previously mentioned like; confusion matrix, classification report, accuracy score and ROC curve to determine the best model among the ones made.

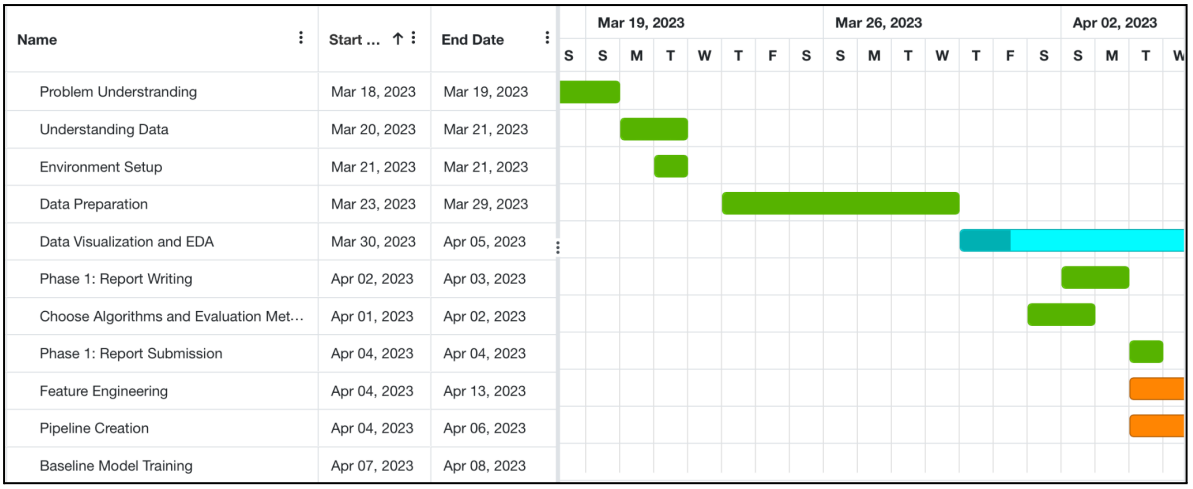
7. **Model Deployment** - The process of integrating a trained ML model into a production environment so that it may be used to generate predictions or categorize new data is known as model deployment.

Project Workflow Block Diagram

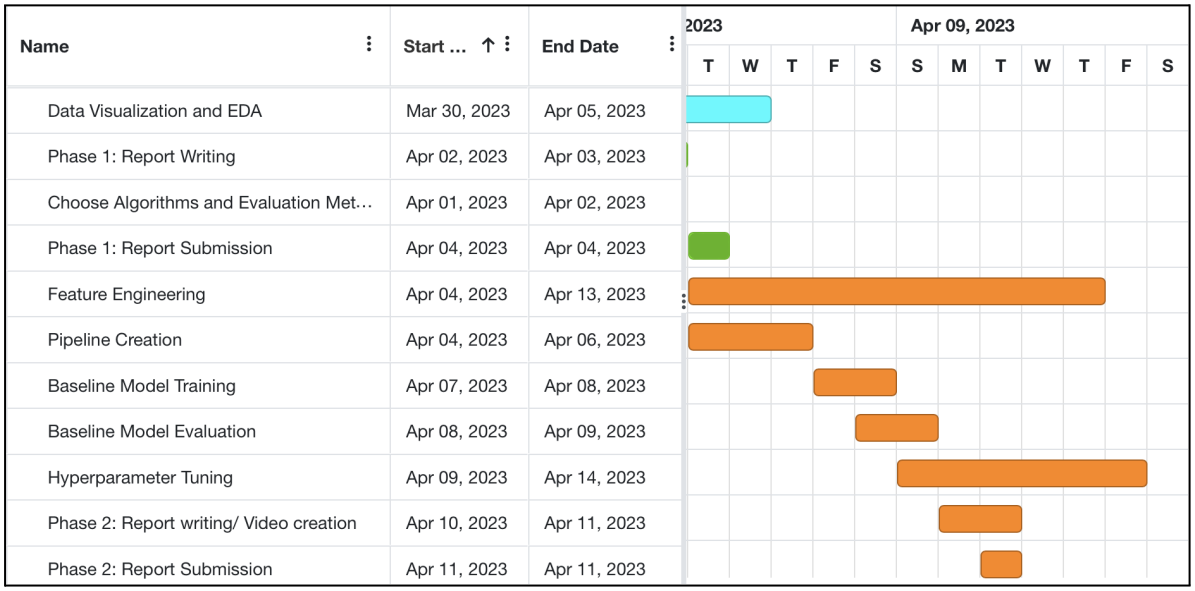


Gantt Chart

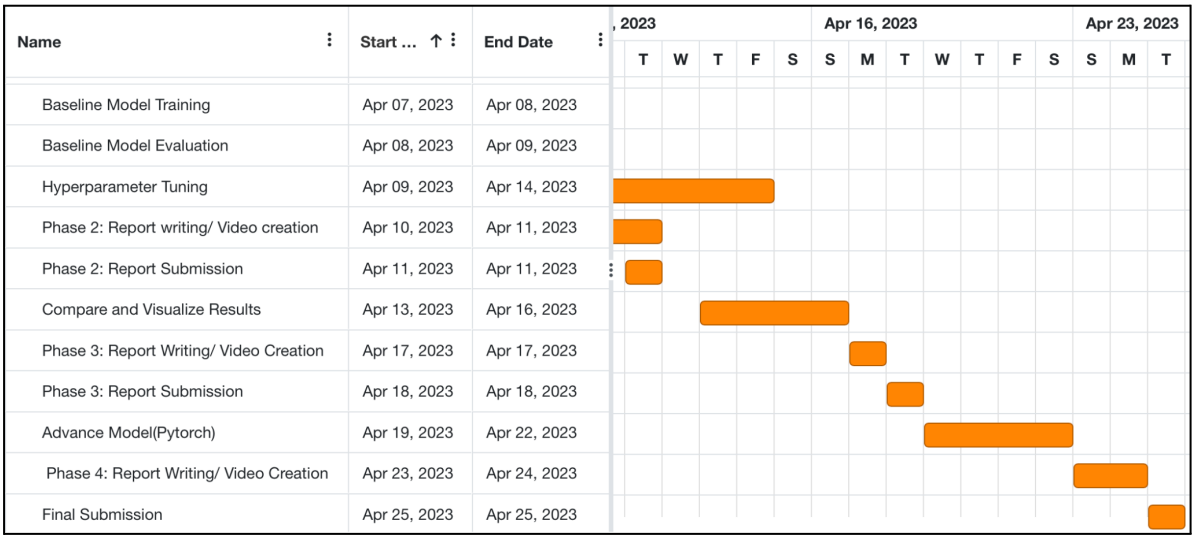
Here’s a brief timeline of the project shown below along with the status of each task as green represents its completed, blue on-going and orange yet to start.



Name	Start ...	End Date
Problem Understanding	Mar 18, 2023	Mar 19, 2023
Understanding Data	Mar 20, 2023	Mar 21, 2023
Environment Setup	Mar 21, 2023	Mar 21, 2023
Data Preparation	Mar 23, 2023	Mar 29, 2023
Data Visualization and EDA	Mar 30, 2023	Apr 05, 2023
Phase 1: Report Writing	Apr 02, 2023	Apr 03, 2023
Choose Algorithms and Evaluation Met…	Apr 01, 2023	Apr 02, 2023
Phase 1: Report Submission	Apr 04, 2023	Apr 04, 2023
Feature Engineering	Apr 04, 2023	Apr 13, 2023
Pipeline Creation	Apr 04, 2023	Apr 06, 2023
Baseline Model Training	Apr 07, 2023	Apr 08, 2023



Name	Start ...	End Date
Data Visualization and EDA	Mar 30, 2023	Apr 05, 2023
Phase 1: Report Writing	Apr 02, 2023	Apr 03, 2023
Choose Algorithms and Evaluation Met…	Apr 01, 2023	Apr 02, 2023
Phase 1: Report Submission	Apr 04, 2023	Apr 04, 2023
Feature Engineering	Apr 04, 2023	Apr 13, 2023
Pipeline Creation	Apr 04, 2023	Apr 06, 2023
Baseline Model Training	Apr 07, 2023	Apr 08, 2023
Baseline Model Evaluation	Apr 08, 2023	Apr 09, 2023
Hyperparameter Tuning	Apr 09, 2023	Apr 14, 2023
Phase 2: Report writing/ Video creation	Apr 10, 2023	Apr 11, 2023
Phase 2: Report Submission	Apr 11, 2023	Apr 11, 2023



Name	Start ...	End Date
Baseline Model Training	Apr 07, 2023	Apr 08, 2023
Baseline Model Evaluation	Apr 08, 2023	Apr 09, 2023
Hyperparameter Tuning	Apr 09, 2023	Apr 14, 2023
Phase 2: Report writing/ Video creation	Apr 10, 2023	Apr 11, 2023
Phase 2: Report Submission	Apr 11, 2023	Apr 11, 2023
Compare and Visualize Results	Apr 13, 2023	Apr 16, 2023
Phase 3: Report Writing/ Video Creation	Apr 17, 2023	Apr 17, 2023
Phase 3: Report Submission	Apr 18, 2023	Apr 18, 2023
Advance Model(Pytorch)	Apr 19, 2023	Apr 22, 2023
Phase 4: Report Writing/ Video Creation	Apr 23, 2023	Apr 24, 2023
Final Submission	Apr 25, 2023	Apr 25, 2023

Phase Leader Plan

Phase Number	Phase Leader	Tasks
Phase 1	Shyam Makwana	Project proposal, Layout, Notebook, Credit assignment plan, Gantt chart, Dataset
Phase 2	Aarushi Dua	Grab data, EDA, Metrics, Baseline models, Baseline pipeline, Brief report, Presentation and script
Phase 3	Sai Teja Burla	Feature engineering, Hyperparameter tuning, Additional feature selection, ensemble methods, Presentation and script
Phase 4	Lakshay Madaan	Neural Network, Advanced model architectures, Loss functions, Final project presentation and report

Credit Assignment Plan

Phase	Member Name	Task Name	Description
Phase 1	Aarushi Dua Sai Teja Burla Shyam Makwana Lakshay Madaan	Problem Understanding	Comprehending the problem and understanding the provided data
	Shyam Makwana	Assigning Tasks	Assigning tasks to each of the team members
		Phase Planning	Planning all the phases
		Data Description	Describing what information each of the given csv files have with a sample of their content
	Sai Teja Burla	Abstract	Brief description of the problem statement at hand
		Researched ML models and metric	Description of ML Models and Evaluation Metrics
		Block Diagram	Visual representation of the pipeline
	Aarushi Dua	Model Pipeline	Explained the steps to perform during the course of the project
		Gantt Chart	Summarize the timeline of the project using gantt charts
	Aarushi Dua Sai Teja Burla	Credit Assignment Plan	Design the four phases and describe the task required.
	Lakshay Madaan	Environmental Setup	Connected Data with docker
		Data Preparation	Checked Dataset are loading and performing basic data visualization.
Phase 2	Aarushi Dua Sai Teja Burla	Data Visualization and EDA	Visualize the dataset to gather insights and perform different data pre-processing techniques.
	Lakshay Madaan Shyam Makwana	Feature Engineering and Pipeline Creation	Data Transformation is required and will add/remove necessary features. Along with creating a pipeline for numeric and categorical features.
	Aarushi Dua	Model Training	Fit the training data on various ML algorithms using pipelines

	Sai Teja Burla	Model Evaluation	Apply different evaluation metrics to get the accuracy and loss
	Shyam Makwana	Research about different Hyperparameters	Gather hyperparameters of each model applied into the GridsearchCV
	Lakshay Madaan	Comparing Baseline Models	Compare the results of each baseline model.
Phase 3	Sai Teja Burla	Feature Importance	Figure out the importance of each features used
	Aarushi Dua	Tuning Hyperparameters	Hyperparameter tuning is performed again to improve the performance.
	Shyam Makhwana	Visualizing different algorithm's results	Plot bars and charts to display the results of each model
	Lakshay Madaan	Feature Engineering	Any further manipulation in features if required
Phase 4	Lakshay Madaan	Final Model Evaluation	Perform final model evaluation on the best model achieved.
	Shyam Makhwana	Report Writing and Video Editing	Write report for final submission and record the video presentation
	Sai Teja Burla Aarushi Dua	Adding Multilayer Perceptron	Introduce advance ML model; i.e Pytorch model for further improvement in model's accuracy

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 compressed) with millions of rows of data
   - 2.71 Gig of data uncompressed

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

Phase leader plan

Phase Number	Phase Leader	Tasks
Phase 1	Shyam Makwana	Project proposal, Layout, Notebook, Credit assignment plan, Gantt chart, Dataset
Phase 2	Aarushi Dua	Grab data, EDA, Metrics, Baseline models, Baseline pipeline, Brief report, Presentation and script
Phase 3	Sai Teja Burla	Feature engineering, Hyperparameter tuning, Additional feature selection, ensemble methods, Presentation and script
Phase 4	Lakshay Madaan	Neural Network, Advanced model architectures, Loss functions, Final project presentation and report

Credit Assignment Plan

Phase	Member Name	Task Name	Description
Phase 1	Aarushi Dua Sai Teja Burla Shyam Makwana Lakshay Madaan	Problem Understanding	Comprehending the problem and understanding the provided data
	Shyam Makwana	Assigning Tasks	Assigning tasks to each of the team members
		Phase Planning	Planning all the phases
		Data Description	Describing what information each of the given csv files have with a sample of their content
	Sai Teja Burla	Abstract	Brief description of the problem statement at hand
		Researched ML models and metric	Description of ML Models and Evaluation Metrics
		Block Diagram	Visual representation of the pipeline
	Aarushi Dua	Model Pipeline	Explained the steps to perform during the course of the project
		Gantt Chart	Summarize the timeline of the project using gantt charts
	Aarushi Dua Sai Teja Burla	Credit Assignment Plan	Design the four phases and describe the task required.
	Lakshay Madaan	Environmental Setup	Connected Data with docker
		Data Preparation	Checked Dataset are loading and performing basic data visualization.

Phase 2	Aarushi Dua Sai Teja Burla	Data Visualization and EDA	Visualize the dataset to gather insights and perform different data pre-processing techniques.
	Lakshay Madaan Shyam Makwana	Feature Engineering and Pipeline Creation	Data Transformation is required and will add/remove necessary features. Along with creating a pipeline for numeric and categorical features.
	Aarushi Dua	Model Training	Fit the training data on various ML algorithms using pipelines
	Sai Teja Burla	Model Evaluation	Apply different evaluation metrics to get the accuracy and loss
	Shyam Makwana	Research about different Hyperparameters	Gather hyperparameters of each model applied into the GridsearchCV
	Lakshay Madaan	Comparing Baseline Models	Compare the results of each baseline model.
Phase 3	Sai Teja Burla	Feature Importance	Figure out the importance of each features used
	Aarushi Dua	Tuning Hyperparameters	Hyperparameter tuning is performed again to improve the performance.
	Shyam Makhwana	Visualizing different algorithm's results	Plot bars and charts to display the results of each model
	Lakshay Madaan	Feature Engineering	Any further manipulation in features if required
Phase 4	Lakshay Madaan	Final Model Evaluation	Perform final model evaluation on the best model achieved.
	Shyam Makhwana	Report Writing and Video Editing	Write report for final submission and record the video presentation
	Sai Teja Burla Aarushi Dua	Adding Multilayer Perceptron	Introduce advance ML model; i.e Pytorch model for further improvement in model's accuracy

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 (https://www.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 (https://medium.com/@nokkk/make-your-kaggle-submissions-with-kaggle-official-api-f49093c04f8a) and here (https://github.com/Kaggle/kaggle-api).

In [1]:

```
!pip install kaggle
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.d
ev/colab-wheels/public/simple/
Requirement already satisfied: kaggle in /usr/local/lib/python3.9/di
st-packages (1.5.13)
Requirement already satisfied: requests in /usr/local/lib/python3.9/
dist-packages (from kaggle) (2.27.1)
Requirement already satisfied: python-dateutil in /usr/local/lib/pyt
hon3.9/dist-packages (from kaggle) (2.8.2)
Requirement already satisfied: python-slugify in /usr/local/lib/pyth
on3.9/dist-packages (from kaggle) (8.0.1)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.9
/dist-packages (from kaggle) (1.16.0)
Requirement already satisfied: certifi in /usr/local/lib/python3.9/d
ist-packages (from kaggle) (2022.12.7)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.9/d
ist-packages (from kaggle) (1.26.15)
Requirement already satisfied: tqdm in /usr/local/lib/python3.9/dist
-packages (from kaggle) (4.65.0)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib
/python3.9/dist-packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/loc
al/lib/python3.9/dist-packages (from requests->kaggle) (2.0.12)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python
3.9/dist-packages (from requests->kaggle) (3.4)
```

In [2]:

```
!pwd
```

```
/content
```

In [3]:

```
!pwd
```

```
/content
```

In [6]:

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

```
-rw-r--r-- 1 root root 66 Apr  4 19:56 kaggle.json
```

In [7]:

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

In [8]:

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

```
name                                  size  creationDate
-----------------------------------  -----  -------------------
previous_application.csv             386MB  2019-12-11 02:55:35
credit_card_balance.csv              405MB  2019-12-11 02:55:35
HomeCredit_columns_description.csv    37KB  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
bureau.csv                           162MB  2019-12-11 02:55:35
bureau_balance.csv                   358MB  2019-12-11 02:55:35
application_test.csv                  25MB  2019-12-11 02:55:35
sample_submission.csv                524KB  2019-12-11 02:55:35
application_train.csv                158MB  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 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 [9]:

```
DATA_DIR = "../data/home-credit-default-risk"   #same level as course repo in th
e data directory
#DATA_DIR = os.path.join('./ddddd/')
!mkdir ../data ../data/home-credit-default-risk
```

In [10]:

```
!ls -l {DATA_DIR}
```

```
total 0
```

In [11]:

```
! kaggle competitions download home-credit-default-risk -p $DATA_DIR
```

```
Downloading home-credit-default-risk.zip to ../data/home-credit-defa
ult-risk
 99% 681M/688M [00:06<00:00, 129MB/s]
100% 688M/688M [00:06<00:00, 118MB/s]
```

In [12]:

```
!pwd
```

```
/content
```

In [13]:

```
!ls -l $DATA_DIR
```

```
total 704708
-rw-r--r-- 1 root root 721616255 Apr  4 19:58 home-credit-default-ri
sk.zip
```

In [ ]:

```
#!rm -r  DATA_DIR
```

Imports

In [14]:

```
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 [15]:

```
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 working
directory. path specifies a different directory to extract to
    zip_ref.extractall('../data/home-credit-default-risk')
    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 [16]:

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

```
-rw-r--r-- 1 root root 166133370 Apr  4 19:58 ../data/home-credit-de
fault-risk/application_train.csv
```

In [17]:

```
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 of t
hem easily
ds_name = 'application_train'
DATA_DIR=f"../data/home-credit-default-risk/"
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[17]:

```
(307511, 122)
```

In [18]:

```
DATA_DIR
```

Out[18]:

```
'../data/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 [19]:

```
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_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 [20]:

```
%%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_n
ame)
```

```
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	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	1350
1	2582071	363914	-1	63975.555	450
2	1740877	371185	-7	31815.225	4500
3	1389973	337855	-4	236572.110	2250
4	1891521	126868	-1	453919.455	4500

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	DAY
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_FUTI
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 46.5 s, sys: 9.64 s, total: 56.2 s
Wall time: 58.4 s
```

In [21]:

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

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

Exploratory Data Analysis

Summary of Application train

In [22]:

```
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 [23]:

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

Out[23]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AI
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 [24]:

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

Out[24]:

	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 [25]:

```
datasets["application_train"].describe(include='all') #look at all categorical a
nd numerical
```

Out[25]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_
count	307511.000000	307511.000000	307511	307511	307
unique	NaN	NaN	2	3	
top	NaN	NaN	Cash loans	F	
freq	NaN	NaN	278232	202448	202
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 [26]:

```
percent = (datasets["application_train"].isnull().sum()/datasets["application_tr
ain"].isnull().count()*100).sort_values(ascending = False).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[26]:

	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 [27]:

```
percent = (datasets["application_test"].isnull().sum()/datasets["application_test"].isnull().count()*100).sort_values(ascending = False).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[27]:

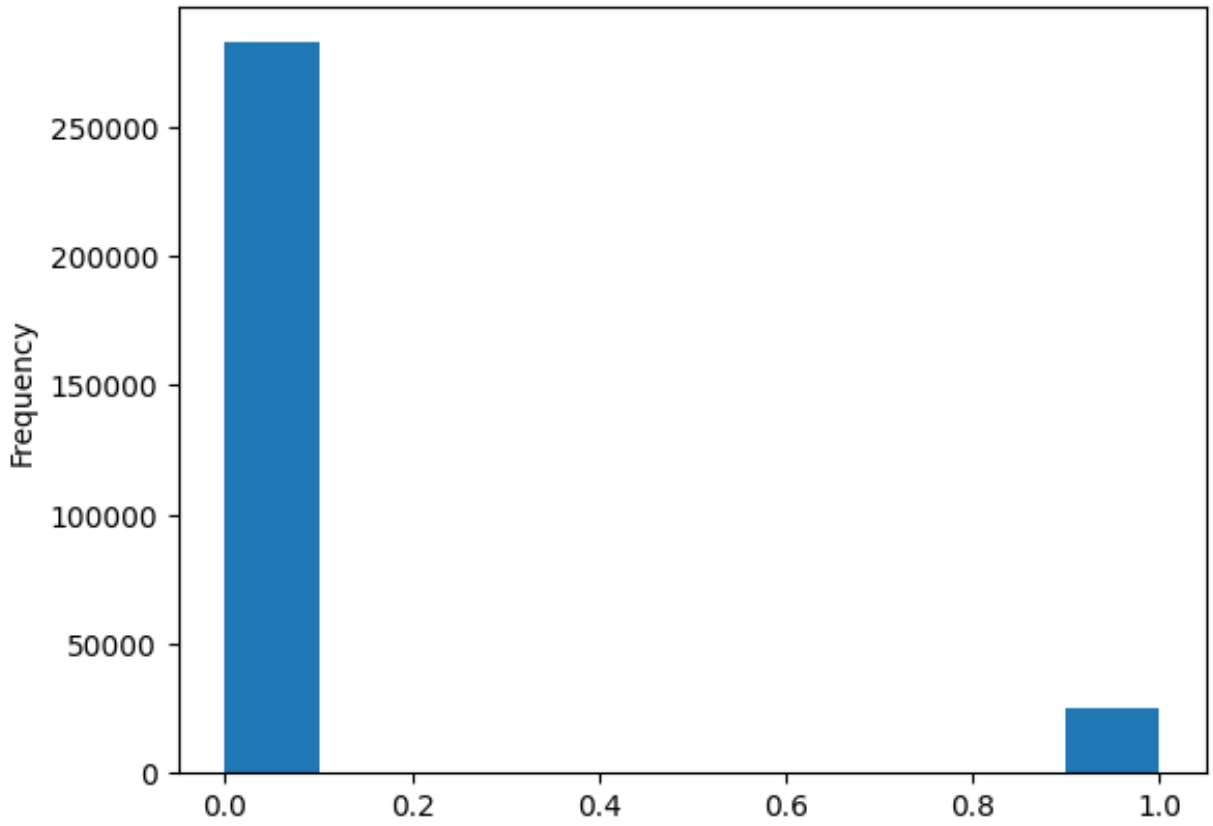
	Percent	Test Missing Count
COMMONAREA_AVG	68.72	33495
COMMONAREA_MODE	68.72	33495
COMMONAREA_MEDI	68.72	33495
NONLIVINGAPARTMENTS_AVG	68.41	33347
NONLIVINGAPARTMENTS_MODE	68.41	33347
NONLIVINGAPARTMENTS_MEDI	68.41	33347
FONDKAPREMONT_MODE	67.28	32797
LIVINGAPARTMENTS_AVG	67.25	32780
LIVINGAPARTMENTS_MODE	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 [28]:

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

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



Note: There's such a class imbalance between target 0 and 1 , we need to resolve this issue later before passing the input data to the model

Correlation with the target column

In [29]:

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

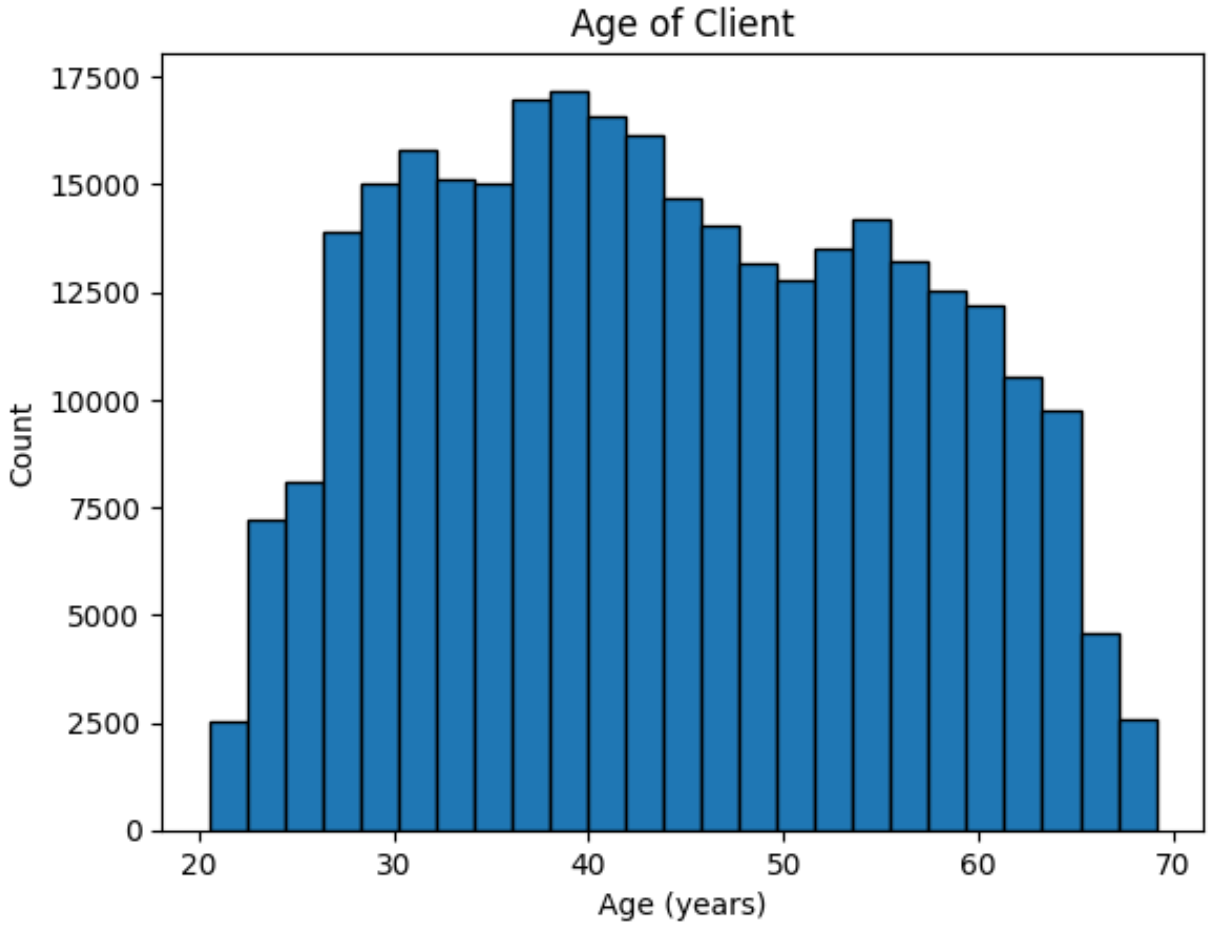
```
Most Positive Correlations:
 FLAG_DOCUMENT_3               0.044346
REG_CITY_NOT_LIVE_CITY         0.044395
FLAG_EMP_PHONE                 0.045982
REG_CITY_NOT_WORK_CITY         0.050994
DAYS_ID_PUBLISH                0.051457
DAYS_LAST_PHONE_CHANGE         0.055218
REGION_RATING_CLIENT           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 [30]:

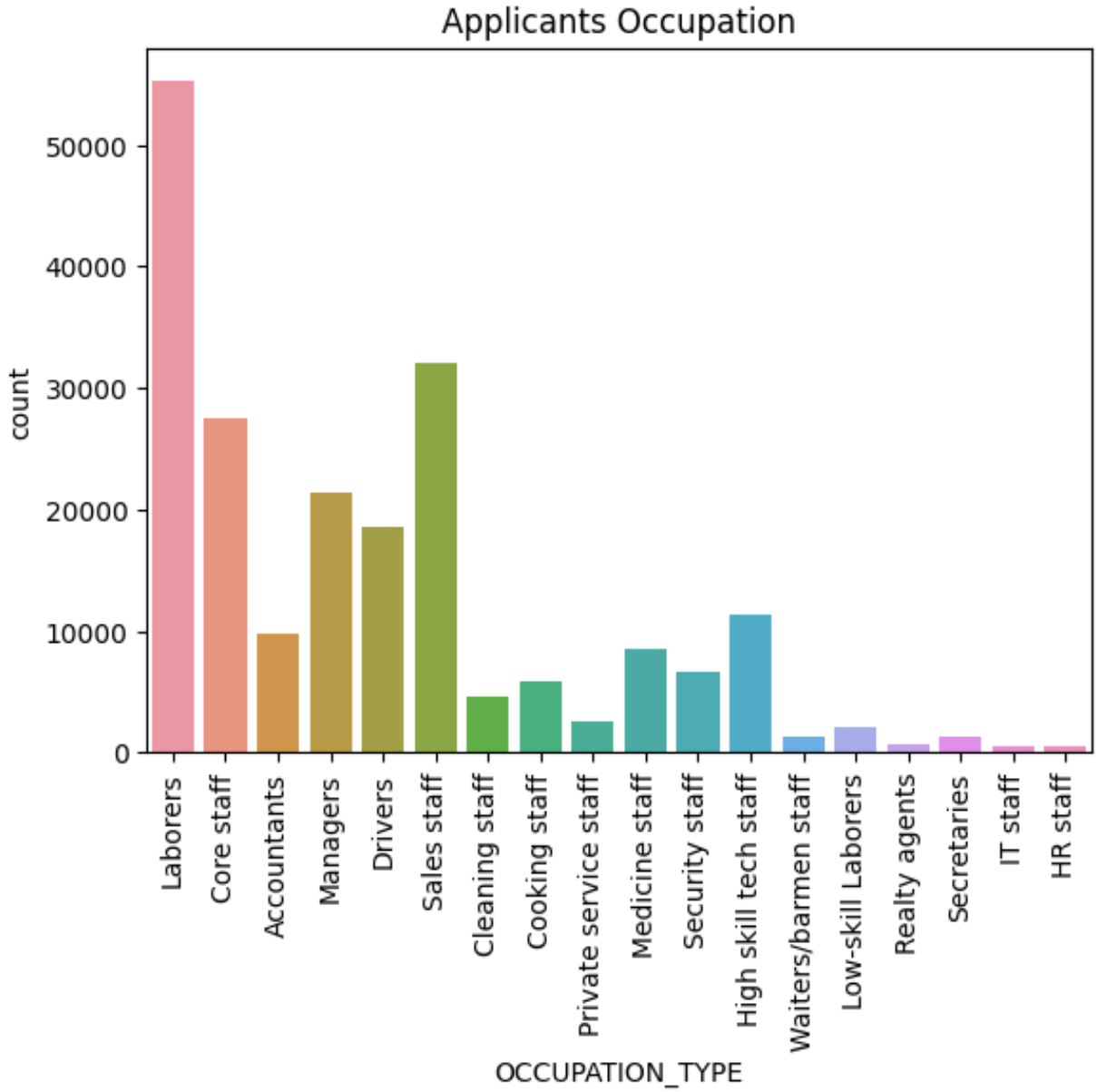
```
plt.hist(datasets["application_train"]['DAYS_BIRTH'] / -365, edgecolor = 'k', bi
ns = 25)
plt.title('Age of Client'); plt.xlabel('Age (years)'); plt.ylabel('Count');
```



Applicants occupations

In [31]:

```
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 [32]:

```
list(datasets.keys())
```

Out[32]:

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

In [33]:

```
len(datasets["application_train"]["SK_ID_CURR"].unique()) == datasets["applicati
on_train"].shape[0]
```

Out[33]:

```
True
```

In [34]:

```
# is there an overlap between the test and train customers
np.intersect1d(datasets["application_train"]["SK_ID_CURR"], datasets["applicatio
n_test"]["SK_ID_CURR"])
```

Out[34]:

```
array([], dtype=int64)
```

In [35]:

```
#
datasets["application_test"].shape
```

Out[35]:

```
(48744, 121)
```

In [36]:

```
datasets["application_train"].shape
```

Out[36]:

```
(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 [37]:

```
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	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

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

In [38]:

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

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

In [39]:

```
#Find the intersection of two arrays.
print(f'Number of train applicants with previous applications is {len(np.intersect1d(datasets["previous_application"]["SK_ID_CURR"], datasets["application_train"]["SK_ID_CURR"])):,}')
```

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

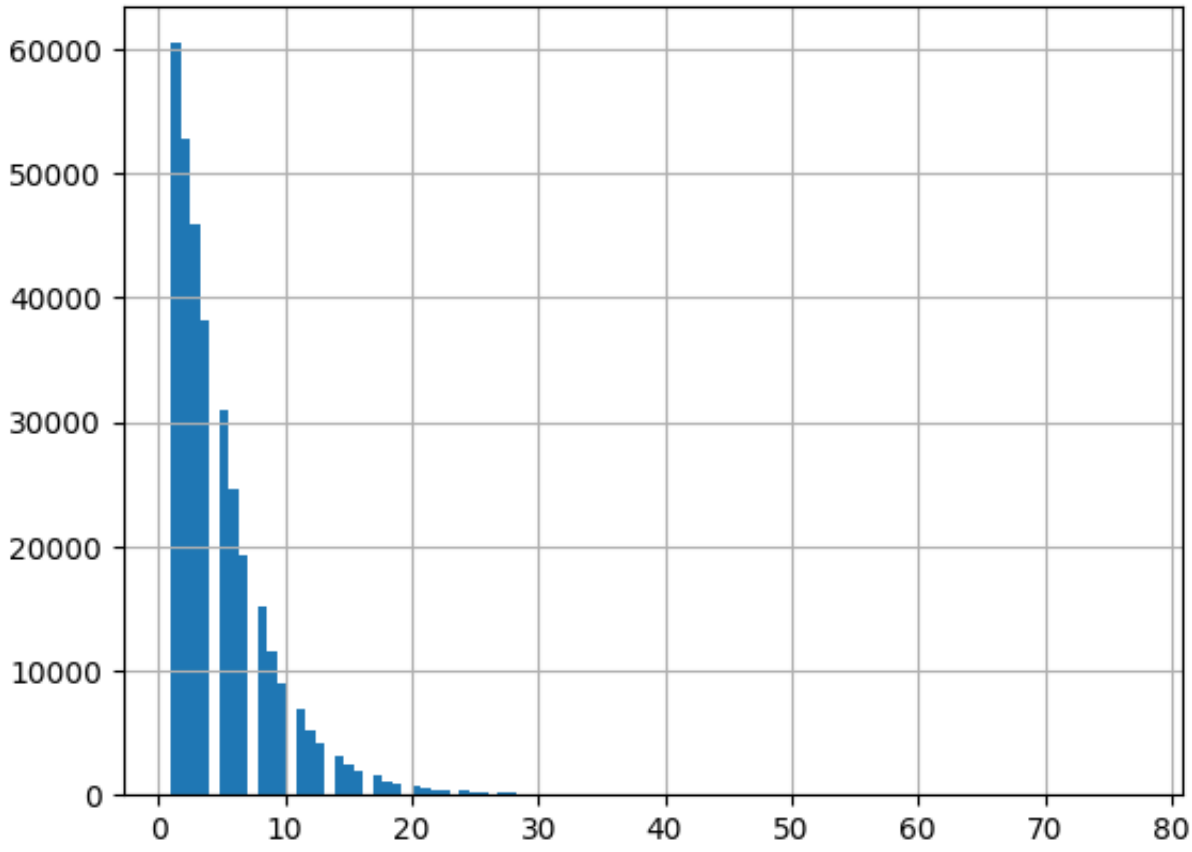
In [40]:

```
#Find the intersection of two arrays.
print(f'Number of train applicants with previous applications is {len(np.intersect1d(datasets["previous_application"]["SK_ID_CURR"], datasets["application_test"]["SK_ID_CURR"])):,}')
```

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

In [41]:

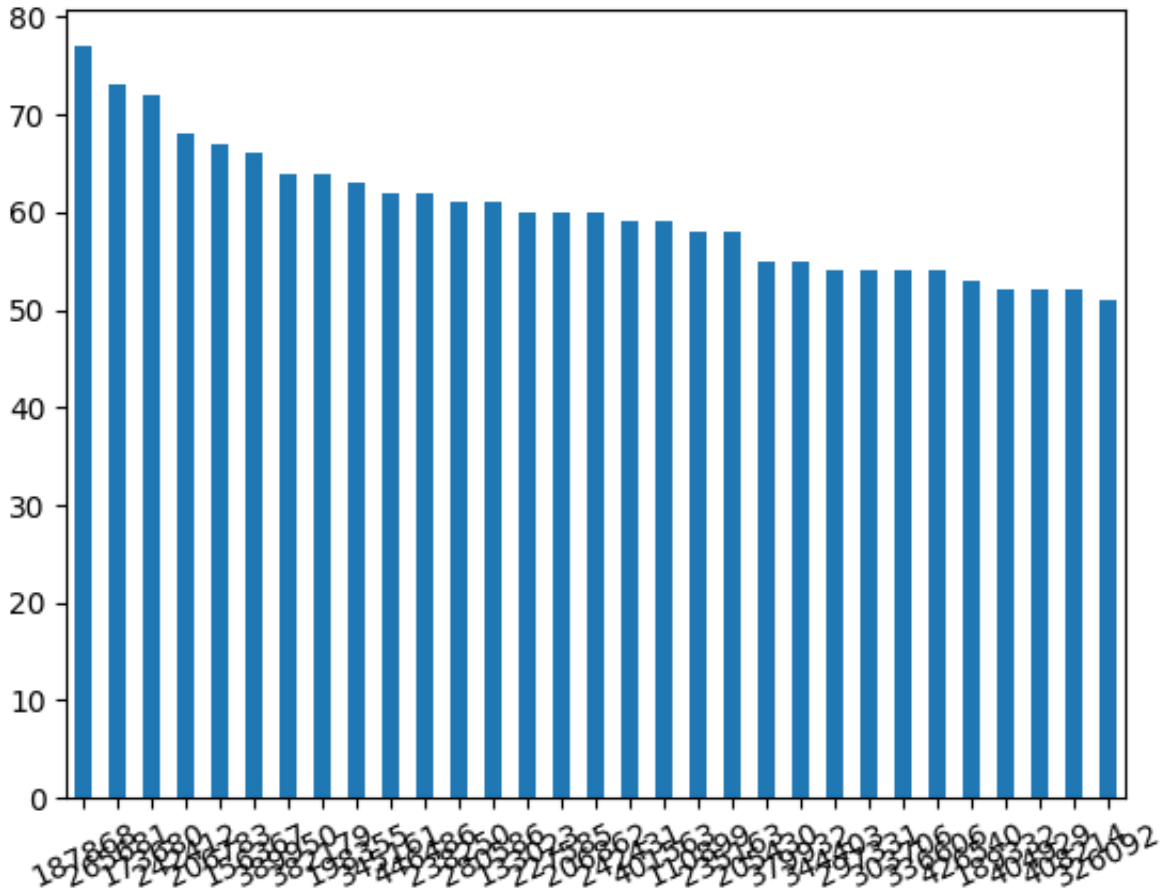
```
# How many previous applciations  per applicant in the previous_application
prevAppCounts = appsDF['SK_ID_CURR'].value_counts(dropna=False)
len(prevAppCounts[prevAppCounts >40])  #more that 40 previous applications
plt.hist(prevAppCounts[prevAppCounts>=0], bins=100)
plt.grid()
```



In [41]:

In [42]:

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



Histogram of Number of previous applications for an ID

In [43]:

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

Out[43]:

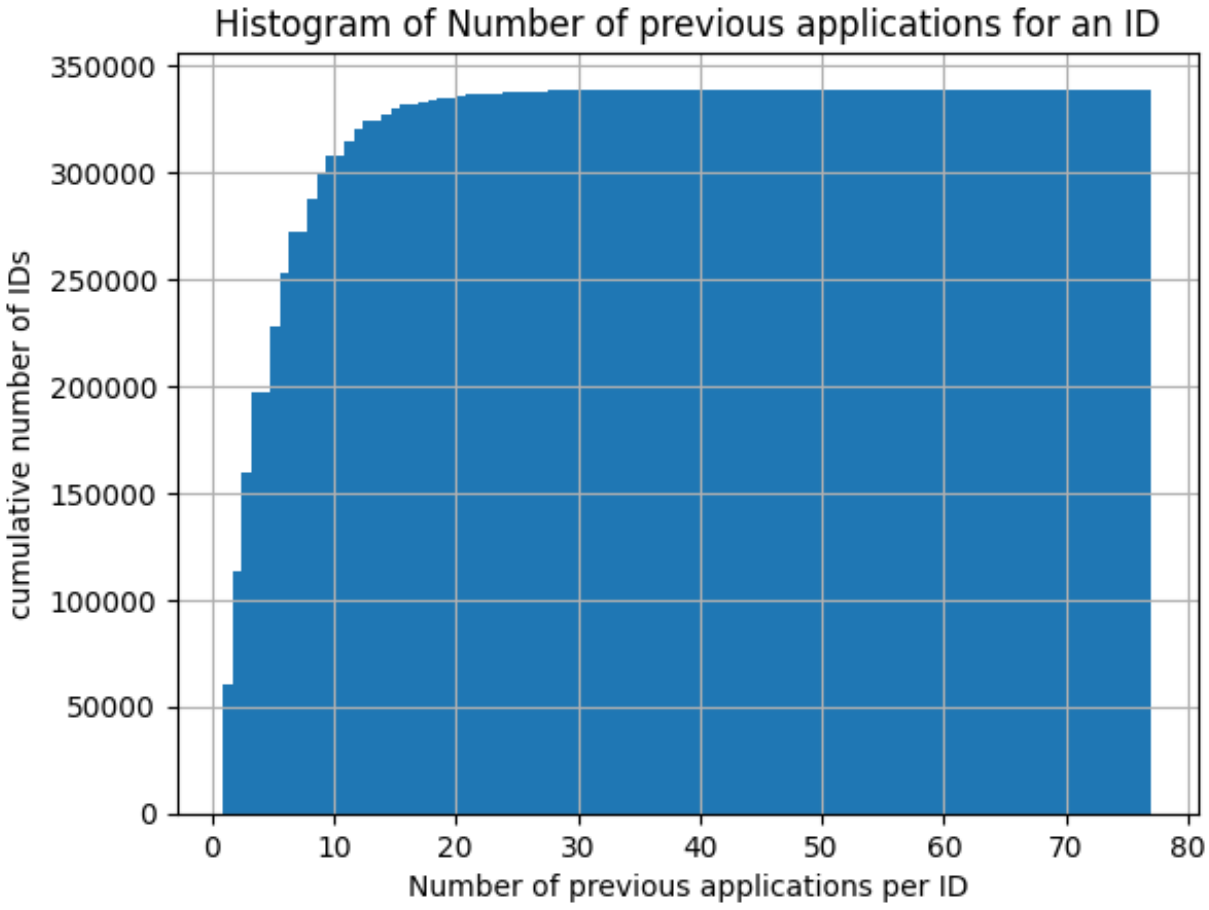
60458

In [44]:

```
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[44]:

```
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 [45]:

```
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_5plus
)/apps_all),5))
print('Percentage with 40 or more previous apps:', np.round(100.*(sum(apps_40plu
s)/apps_all),5))
```

```
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 (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 [46]:

```
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 [47]:

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

Out[47]:

	A	B
sum	12.0	NaN
min	1.0	2.0
max	NaN	8.0

In [48]:

```
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	-1.276933
1	1	2	-1.219068
2	2	3	-1.033182
3	2	4	-0.654260

In [49]:

```
# 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[49]:

	B		C
	min	max	sum
A			
1	1	2	-2.496002
2	3	4	-1.687442

In [50]:

```
appsDF.columns
```

Out[50]:

```
Index(['SK_ID_PREV', 'SK_ID_CURR', 'NAME_CONTRACT_TYPE', 'AMT_ANNUIT
Y',
       'AMT_APPLICATION', 'AMT_CREDIT', 'AMT_DOWN_PAYMENT', 'AMT_GOO
DS_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_VE
RSION',
       'DAYS_LAST_DUE', 'DAYS_TERMINATION', 'NFLAG_INSURED_ON_APPROV
AL'],
      dtype='object')
```

In [50]:

```

```

In [51]:

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

Out[51]:

```
{'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 (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

In [52]:

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

Out[52]:

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

1 rows × 37 columns

In [53]:

```
appsDF[0:50][(appsDF["SK_ID_CURR"]==175704)]["AMT_CREDIT"]
```

Out[53]:

```
6    0.0
Name: AMT_CREDIT, dtype: float64
```

In [54]:

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

Out[54]:

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

1 rows × 37 columns

Missing values in prevApps

In [55]:

```
appsDF.isna().sum()
```

Out[55]:

```
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 [56]:

```
appsDF.columns
```

Out[56]:

```
Index(['SK_ID_PREV', 'SK_ID_CURR', 'NAME_CONTRACT_TYPE', 'AMT_ANNUIT
Y',
       'AMT_APPLICATION', 'AMT_CREDIT', 'AMT_DOWN_PAYMENT', 'AMT_GOO
DS_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_VE
RSION',
       'DAYS_LAST_DUE', 'DAYS_TERMINATION', 'NFLAG_INSURED_ON_APPROV
AL'],
      dtype='object')
```

feature engineering for prevApp table

In [70]:

```
#appsDF['agg_op_features'].head()
```

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 (https://cmdlinetips.com/2020/05/fun-with-pandas-groupby-aggregate-multi-index-and-unstack/)

For more details and examples please see here (https://www.shanelynn.ie/summarising-aggregation-and-grouping-data-in-python-pandas/)

In [59]:

```
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_ANNUITY' : agg
_ops, 'AMT_APPLICATION' : agg_ops})
result.columns = result.columns.map('_'.join)
display(result)
result['range_AMT_APPLICATION'] = result['AMT_APPLICATION_max'] - result['AMT_AP
PLICATION_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_AI
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[59]:

	SK_ID_CURR_	AMT_ANNUITY_min	AMT_ANNUITY_max	AMT_ANNUITY_mean	AMT_APPLIC
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 [60]:

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

Out[60]:

```
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 [69]:

```
# 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", "
```

```
max", "mean"]}
            self.agg_op_features[f]=["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() #bre
akpoint; dont forget to quit
        result = X.groupby(["SK_ID_CURR"]).agg(self.agg_op_features)
        #result.columns = result.columns.droplevel()
        result.columns = ["_".join(x) for x in result.columns.ravel()]
        result = result.reset_index(level=["SK_ID_CURR"])
        result['range_AMT_APPLICATION'] = result['AMT_APPLICATION_max'] - result
['AMT_APPLICATION_min']
        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 != 'Sales
Staff'? (hint: YES)
```

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

df[['AMT_ANNUITY', 'AMT_APPLICATION']][0:5]:
   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
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

   range_AMT_APPLICATION
0                    0.0
1                    0.0
2               831190.5
3                    0.0
4                44617.5
5               688500.0
6               230323.5
7               450000.0
8                69705.0
9                    0.0
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
9                       NaN

[10 rows x 37 columns]
```

Join the labeled dataset

In [63]:

```
~3==3
```

Out[63]:

```
False
```

In [64]:

```
datasets.keys()
```

Out[64]:

```
dict_keys(['application_train', 'application_test', 'bureau', 'burea
u_balance', 'credit_card_balance', 'installments_payments', 'previou
s_application', 'POS_CASH_balance'])
```

In [65]:

```
features = ['AMT_ANNUITY', 'AMT_APPLICATION']
prevApps_feature_pipeline = Pipeline([
        ('prevApps_add_features1', prevApps_add_features1()),  # add some new fe
atures
        ('prevApps_add_features2', prevApps_add_features2()),  # add some new fe
atures
        ('prevApps_aggregater', prevAppsFeaturesAggregater()), # Aggregate acros
s old and new features
    ])


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_payments',
    # 'previous_application', 'POS_CASH_balance'

# merge primary table and secondary tables using features based on meta data and
aggregage stats
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")

    # ......
```

```
---------------------------------------------------------------------
-------
NameError                                 Traceback (most recent cal
l last)
<ipython-input-65-38e91d30f7a2> in <cell line: 2>()
      1 features = ['AMT_ANNUITY', 'AMT_APPLICATION']
      2 prevApps_feature_pipeline = Pipeline([
----> 3         ('prevApps_add_features1', prevApps_add_features1())
,  # add some new features
      4         ('prevApps_add_features2', prevApps_add_features2())
,  # add some new features
      5         ('prevApps_aggregater', prevAppsFeaturesAggregater()
), # Aggregate across old and new features

NameError: name 'prevApps_add_features1' is not defined
```

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='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
    #df_labeled = df_labeled.merge(...._aggregated, how='left', on="SK_ID_CURR")

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

    # .....
```

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': 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) else int(x[0]
) + 1)
        X['CharlsonIndex'] = X['CharlsonIndex'].apply(lambda x: charlson_idx_dt[
x])
        X['LengthOfStay'] = X['LengthOfStay'].apply(lambda x: None if 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"))
        ])
```

In [ ]:

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

In [ ]:

```
# Categorical boolean mask
categorical_feature_mask = df.dtypes==object
categorical_feature_mask
```

In [ ]:

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

In [ ]:

```
from sklearn.preprocessing import OneHotEncoder
import pandas as pd
categorical_feature_mask = [True, False]
# instantiate OneHotEncoder
enc = OneHotEncoder(categorical_features = categorical_feature_mask,sparse = Fal
se, handle_unknown='ignore')
# 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', 2]]
)
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()}")
```

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 [ ]:

```
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
```

In [ ]:

```
data.feature_names
```

Please [this blog (https://medium.com/hugo-ferreiras-blog/dealing-with-categorical-features-in-machine-learning-1bb70f07262d)](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 [ ]:

```
# 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','DAYS
_BIRTH','EXT_SOURCE_1',
        'EXT_SOURCE_2','EXT_SOURCE_3','CODE_GENDER', 'FLAG_OWN_REALTY','FLAG_OWN
_CAR','NAME_CONTRACT_TYPE',
                    '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, test
_size=0.15, random_state=42)
    X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_s
ize=0.15, random_state=42)
    X_kaggle_test= datasets["application_test"][selected_features]
    # y_test = datasets["application_test"]['TARGET']   #why no  TARGET?!! (hint
: kaggle competition)

selected_features = ['AMT_INCOME_TOTAL',  'AMT_CREDIT','DAYS_EMPLOYED','DAYS_BIR
TH','EXT_SOURCE_1',
        'EXT_SOURCE_2','EXT_SOURCE_3','CODE_GENDER', 'FLAG_OWN_REALTY','FLAG_OWN
_CAR','NAME_CONTRACT_TYPE',
                    '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_siz
e=0.15, random_state=42)
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_size=
0.15, random_state=42)
X_kaggle_test= X_kaggle_test[selected_features]
# y_test = datasets["application_test"]['TARGET']   #why no  TARGET?!! (hint: ka
ggle 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}")
```

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

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_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_TY
PE',
               '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')),
        ('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 (http://en.wikipedia.org/wiki/Receiver_operating_characteristic)](http://en.wikipedia.org/wiki/Receiver_operating_characteristic) 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 -m "b
aseline submission"
```

report submission

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

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 (https://www.kaggle.com/willkoehrsen/start-here-a-gentle-introduction/notebook)

In [ ]:

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://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/ (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://dai.lids.mit.edu/wp-content/uploads/2017/10/DSAA_DSM_2015.pdf)
- https://www.analyticsvidhya.com/blog/2017/08/catboost-automated-categorical-data/ (https://www.analyticsvidhya.com/blog/2017/08/catboost-automated-categorical-data/)