Home Credit Default Risk (HCDR)

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

Some of the challenges

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

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

Kaggle API setup

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

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

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

1. Install library

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

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

In [1]:
```
!pip install kaggle
```

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

In [2]:
```
!pwd
```

```
/root/shared/Documents/Indiana Coursework/INFO-I 526 Applied ML/AML-project
```

In [3]:
```
!mkdir ~/.kaggle
!cp /root/shared/Downloads/kaggle.json ~/.kaggle
!chmod 600 ~/.kaggle/kaggle.json
```

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

In [4]:
```
! kaggle competitions files home-credit-default-risk
```

```
name                                  size  creationDate
----------------------------------  -----  -------------------
application_test.csv                   25MB  2019-12-11 02:55:35
application_train.csv                 158MB  2019-12-11 02:55:35
HomeCredit_columns_description.csv     37KB  2019-12-11 02:55:35
bureau_balance.csv                    358MB  2019-12-11 02:55:35
installments_payments.csv             690MB  2019-12-11 02:55:35
previous_application.csv              386MB  2019-12-11 02:55:35
```

```
POS_CASH_balance.csv                375MB  2019-12-11 02:55:35
bureau.csv                          162MB  2019-12-11 02:55:35
sample_submission.csv               524KB  2019-12-11 02:55:35
credit_card_balance.csv             405MB  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

There are 7 different sources of data:

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

In [5]:

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

Downloading the files via Kaggle API

Create a base directory:

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

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

1. Click on the `Download` button on the following Data Webpage and unzip the zip file to the `BASE_DIR`
2. If you plan to use the Kaggle API, please use the following steps.

In [6]:
```
DATA_DIR = "../Data/home-credit-default-risk"   #same level as course repo in the data directory
#DATA_DIR = os.path.join('./ddddd/')
!mkdir $DATA_DIR
```

```
mkdir: cannot create directory '../Data/home-credit-default-risk': File exists
```

In [7]:
```
!ls -l $DATA_DIR
```

```
total 3332204
-rw-rw-r-- 1 root root     37383 Dec 11  2019 HomeCredit_columns_description.csv
-rw-rw-r-- 1 root root 392703158 Dec 11  2019 POS_CASH_balance.csv
-rw-rw-r-- 1 root root  26567651 Dec 11  2019 application_test.csv
-rw-rw-r-- 1 root root 166133370 Dec 11  2019 application_train.csv
-rw-rw-r-- 1 root root 170016717 Dec 11  2019 bureau.csv
-rw-rw-r-- 1 root root 375592889 Dec 11  2019 bureau_balance.csv
-rw-rw-r-- 1 root root 424582605 Dec 11  2019 credit_card_balance.csv
-rw-r--r-- 1 root root 721616255 Nov 10 07:15 home-credit-default-risk.zip
-rw-rw-r-- 1 root root 723118349 Dec 11  2019 installments_payments.csv
-rw-rw-r-- 1 root root 404973293 Dec 11  2019 previous_application.csv
-rw-rw-r-- 1 root root    536202 Dec 11  2019 sample_submission.csv
```

In [8]:
```
! kaggle competitions download home-credit-default-risk -p $DATA_DIR
```

```
home-credit-default-risk.zip: Skipping, found more recently modified local copy (use --force to force download)
```

Imports

In [9]:
```
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 [10]:

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

Data files overview

Data Dictionary

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

HomeCredit_columns_description

Possible Data Loss Some features might be lost if you save this workbook in the comma-delimited (.csv) format. To preserve these features, save it in an Excel file format.

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

Application train

In [11]:

```
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 them easily
ds_name = 'application_train'
datasets[ds_name] = load_data(os.path.join(DATA_DIR, f'{ds_name}.csv'), ds_name)

datasets['application_train'].shape
```

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

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

5 rows × 122 columns

Out[11]:
```
(307511, 122)
```

Application test

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

In [12]:
```
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_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL
0	100001	Cash loans	F	N	Y	0	135000.0
1	100005	Cash loans	M	N	Y	0	99000.0
2	100013	Cash loans	M	Y	Y	0	202500.0
3	100028	Cash loans	F	N	Y	2	315000.0
4	100038	Cash loans	M	Y	N	1	180000.0

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 [13]:

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

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

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

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

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	AMT_INCOME_TOTAL
0	100001	Cash loans	F	N	Y	0	135000.0
1	100005	Cash loans	M	N	Y	0	99000.0
2	100013	Cash loans	M	Y	Y	0	202500.0
3	100028	Cash loans	F	N	Y	2	315000.0
4	100038	Cash loans	M	Y	N	1	180000.0

5 rows × 121 columns

```
bureau: shape is (1716428, 17)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1716428 entries, 0 to 1716427
Data columns (total 17 columns):
 #   Column                  Dtype
---  ------                  -----
 0   SK_ID_CURR              int64
 1   SK_ID_BUREAU            int64
 2   CREDIT_ACTIVE           object
 3   CREDIT_CURRENCY         object
 4   DAYS_CREDIT             int64
 5   CREDIT_DAY_OVERDUE      int64
 6   DAYS_CREDIT_ENDDATE     float64
 7   DAYS_ENDDATE_FACT       float64
 8   AMT_CREDIT_MAX_OVERDUE  float64
 9   CNT_CREDIT_PROLONG      int64
 10  AMT_CREDIT_SUM          float64
 11  AMT_CREDIT_SUM_DEBT     float64
 12  AMT_CREDIT_SUM_LIMIT    float64
 13  AMT_CREDIT_SUM_OVERDUE  float64
 14  CREDIT_TYPE             object
 15  DAYS_CREDIT_UPDATE      int64
 16  AMT_ANNUITY             float64
dtypes: float64(8), int64(6), object(3)
```

```
memory usage: 222.6+ MB
None
```

	SK_ID_CURR	SK_ID_BUREAU	CREDIT_ACTIVE	CREDIT_CURRENCY	DAYS_CREDIT	CREDIT_DAY_OVERDUE	DAYS_CREDIT_ENDDATE
0	215354	5714462	Closed	currency 1	-497	0	-153.0
1	215354	5714463	Active	currency 1	-208	0	1075.0
2	215354	5714464	Active	currency 1	-203	0	528.0
3	215354	5714465	Active	currency 1	-203	0	NaN
4	215354	5714466	Active	currency 1	-629	0	1197.0

```
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_ACTUAL	AMT_DRAWINGS_ATM_CURRENT	AMT
0	2562384	378907	-6	56.970	135000	0.0	
1	2582071	363914	-1	63975.555	45000	2250.0	
2	1740877	371185	-7	31815.225	450000	0.0	
3	1389973	337855	-4	236572.110	225000	2250.0	
4	1891521	126868	-1	453919.455	450000	0.0	

5 rows × 23 columns

```
installments_payments: shape is (13605401, 8)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13605401 entries, 0 to 13605400
Data columns (total 8 columns):
 #   Column                  Dtype
---  ------                  -----
 0   SK_ID_PREV              int64
 1   SK_ID_CURR              int64
```

```
 2   NUM_INSTALMENT_VERSION  float64
 3   NUM_INSTALMENT_NUMBER   int64
 4   DAYS_INSTALMENT         float64
 5   DAYS_ENTRY_PAYMENT      float64
 6   AMT_INSTALMENT          float64
 7   AMT_PAYMENT             float64
dtypes: float64(5), int64(3)
memory usage: 830.4 MB
None
```

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER	DAYS_INSTALMENT	DAYS_ENTRY_PAYMENT
0	1054186	161674	1.0	6	-1180.0	-1187.0
1	1330831	151639	0.0	34	-2156.0	-2156.0
2	2085231	193053	2.0	1	-63.0	-63.0
3	2452527	199697	1.0	3	-2418.0	-2426.0
4	2714724	167756	1.0	2	-1383.0	-1366.0

```
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	AMT_DOWN_PAYMENT	AMT
0	2030495	271877	Consumer loans	1730.430	17145.0	17145.0	0.0	
1	2802425	108129	Cash loans	25188.615	607500.0	679671.0	NaN	
2	2523466	122040	Cash loans	15060.735	112500.0	136444.5	NaN	
3	2819243	176158	Cash loans	47041.335	450000.0	470790.0	NaN	
4	1784265	202054	Cash loans	31924.395	337500.0	404055.0	NaN	

5 rows × 37 columns

```
POS_CASH_balance: shape is (10001358, 8)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10001358 entries, 0 to 10001357
Data columns (total 8 columns):
 #   Column                 Dtype
---  ------                 -----
 0   SK_ID_PREV             int64
 1   SK_ID_CURR             int64
 2   MONTHS_BALANCE         int64
 3   CNT_INSTALMENT         float64
 4   CNT_INSTALMENT_FUTURE  float64
 5   NAME_CONTRACT_STATUS   object
 6   SK_DPD                 int64
```

```
 7   SK_DPD_DEF              int64
dtypes: float64(2), int64(5), object(1)
memory usage: 610.4+ MB
None
```

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

```
CPU times: user 40.8 s, sys: 43.3 s, total: 1min 24s
Wall time: 1min 31s
```

In [14]:

```python
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

Summary Statistics

In [15]:

```python
print('\033[1m' + "Size of each dataset : " + '\033[0m' , end = '\n' * 2)
for ds_name in datasets.keys():
    print(f'dataset {ds_name:24}: [ {datasets[ds_name].shape[0]:10,}, {datasets[ds_name].shape[1]:4}]')
```

```
Size of each dataset :
```

```
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]
```

In [16]:

```
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 [17]:

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

Out[17]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REG
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	307499.000000	3.072330e+05	
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	27108.573909	5.383962e+05	
std	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	14493.737315	3.694465e+05	
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	1615.500000	4.050000e+04	
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	16524.000000	2.385000e+05	
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	24903.000000	4.500000e+05	
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	34596.000000	6.795000e+05	
max	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	258025.500000	4.050000e+06	

8 rows × 106 columns

In [18]:

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

Out[18]:

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULATIO

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULATIO
count	48744.000000	48744.000000	4.874400e+04	4.874400e+04	48720.000000	4.874400e+04	48
mean	277796.676350	0.397054	1.784318e+05	5.167404e+05	29426.240209	4.626188e+05	
std	103169.547296	0.709047	1.015226e+05	3.653970e+05	16016.368315	3.367102e+05	
min	100001.000000	0.000000	2.694150e+04	4.500000e+04	2295.000000	4.500000e+04	
25%	188557.750000	0.000000	1.125000e+05	2.606400e+05	17973.000000	2.250000e+05	
50%	277549.000000	0.000000	1.575000e+05	4.500000e+05	26199.000000	3.960000e+05	
75%	367555.500000	1.000000	2.250000e+05	6.750000e+05	37390.500000	6.300000e+05	
max	456250.000000	20.000000	4.410000e+06	2.245500e+06	180576.000000	2.245500e+06	

8 rows × 105 columns

In [19]:

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

Out[19]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN
count	307511.000000	307511.000000	307511	307511	307511	307511	307511.00000
unique	NaN	NaN	2	3	2	2	Na
top	NaN	NaN	Cash loans	F	N	Y	Na
freq	NaN	NaN	278232	202448	202924	213312	Na
mean	278180.518577	0.080729	NaN	NaN	NaN	NaN	0.41705
std	102790.175348	0.272419	NaN	NaN	NaN	NaN	0.72212
min	100002.000000	0.000000	NaN	NaN	NaN	NaN	0.00000
25%	189145.500000	0.000000	NaN	NaN	NaN	NaN	0.00000
50%	278202.000000	0.000000	NaN	NaN	NaN	NaN	0.00000
75%	367142.500000	0.000000	NaN	NaN	NaN	NaN	1.00000
max	456255.000000	1.000000	NaN	NaN	NaN	NaN	19.00000

11 rows × 122 columns

In [20]:
```
from IPython.display import display, HTML
pd.set_option("display.max_rows", None, "display.max_columns", None)

# Full stats

def stats_summary1(df, df_name):
    print(datasets[df_name].info(verbose=True, null_counts=True ))
    print("-----"*15)
    print(f"Shape of the df {df_name} is {df.shape} \n")
    print("-----"*15)
    print(f"Statistical summary of {df_name} is :")
    print("-----"*15)
    print(f"Description of the df {df_name}:\n")
    print(display(HTML(np.round(datasets['application_train'].describe(),2).to_html())))
    #print(f"Description of the df {df_name}:\n",np.round(datasets['application_train'].describe(),2))

def stats_summary2(df, df_name):
    print(f"Description of the df continued for {df_name}:\n")
    print("-----"*15)
    print("Data type value counts: \n",df.dtypes.value_counts())
    print("\nReturn number of unique elements in the object. \n")
    print(df.select_dtypes('object').apply(pd.Series.nunique, axis = 0))


# List the categorical and Numerical features of a DF
def feature_datatypes_groups(df, df_name):
    df_dtypes = df.columns.to_series().groupby(df.dtypes).groups
    print("-----"*15)
    print(f"Categorical and Numerical(int + float) features  of {df_name}.")
    print("-----"*15)
    print()
    for k, v in df_dtypes.items():
        print({k.name: v})
        print("---"*10)
    print("\n \n")

# Null data list and plot.
def null_data_plot(df, df_name):
    percent = (df.isnull().sum()/df.isnull().count()*100).sort_values(ascending = False).round(2)
    sum_missing = df.isna().sum().sort_values(ascending = False)
    missing_data  = pd.concat([percent, sum_missing], axis=1, keys=['Percent', "Train Missing Count"])
    missing_data=missing_data[missing_data['Percent'] > 0]
    print("-----"*15)
    print("-----"*15)
    print('\n The Missing Data: \n')
#     display(missing_data)  # display few
```

```
    if len(missing_data)==0:
        print("No missing Data")
    else:
        display(HTML(missing_data.to_html()))  # display all the rows
        print("-----"*15)
        if len(df.columns)> 35:
            f,ax =plt.subplots(figsize=(8,15))
        else:
            f,ax =plt.subplots()
      #plt.xticks(rotation='90')
      #fig=sns.barplot(missing_data.index, missing_data["Percent"],alpha=0.8)
      #plt.xlabel('Features', fontsize=15)
      #plt.ylabel('Percent of missing values', fontsize=15)
        plt.title(f'Percent missing data for {df_name}.', fontsize=10)
        fig=sns.barplot(missing_data["Percent"],missing_data.index ,alpha=0.8)
        plt.xlabel('Percent of missing values', fontsize=10)
        plt.ylabel('Features', fontsize=10)
        plt.show()
        return missing_data


# Full consolidation of all the stats function.
def display_stats(df, df_name):
    print("--"*40)
    print(" "*20 + '\033[1m'+ df_name +  '\033[0m' +" "*20)
    print("--"*40)
    stats_summary1(df, df_name)

def display_feature_info(df, df_name):
    stats_summary2(df, df_name)
    feature_datatypes_groups(df, df_name)
    null_data_plot(df, df_name)
```

In [21]:

```
display_feature_info(datasets['application_train'], 'application_train')
```

```
Description of the df continued for application_train:

--------------------------------------------------------------------------------
Data type value counts:
 float64    65
int64      41
object     16
dtype: int64
```

```
Return number of unique elements in the object.

NAME_CONTRACT_TYPE             2
CODE_GENDER                    3
FLAG_OWN_CAR                   2
FLAG_OWN_REALTY                2
NAME_TYPE_SUITE                7
NAME_INCOME_TYPE               8
NAME_EDUCATION_TYPE            5
NAME_FAMILY_STATUS             6
NAME_HOUSING_TYPE              6
OCCUPATION_TYPE               18
WEEKDAY_APPR_PROCESS_START     7
ORGANIZATION_TYPE             58
FONDKAPREMONT_MODE             4
HOUSETYPE_MODE                 3
WALLSMATERIAL_MODE             7
EMERGENCYSTATE_MODE            2
dtype: int64
-----------------------------------------------------------------------------
Categorical and Numerical(int + float) features  of application_train.
-----------------------------------------------------------------------------

{'int64': Index(['SK_ID_CURR', 'TARGET', 'CNT_CHILDREN', 'DAYS_BIRTH', 'DAYS_EMPLOYED',
       'DAYS_ID_PUBLISH', 'FLAG_MOBIL', 'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE',
       'FLAG_CONT_MOBILE', 'FLAG_PHONE', 'FLAG_EMAIL', 'REGION_RATING_CLIENT',
       'REGION_RATING_CLIENT_W_CITY', 'HOUR_APPR_PROCESS_START',
       'REG_REGION_NOT_LIVE_REGION', 'REG_REGION_NOT_WORK_REGION',
       'LIVE_REGION_NOT_WORK_REGION', 'REG_CITY_NOT_LIVE_CITY',
       'REG_CITY_NOT_WORK_CITY', 'LIVE_CITY_NOT_WORK_CITY', 'FLAG_DOCUMENT_2',
       'FLAG_DOCUMENT_3', 'FLAG_DOCUMENT_4', 'FLAG_DOCUMENT_5',
       'FLAG_DOCUMENT_6', 'FLAG_DOCUMENT_7', 'FLAG_DOCUMENT_8',
       'FLAG_DOCUMENT_9', 'FLAG_DOCUMENT_10', 'FLAG_DOCUMENT_11',
       'FLAG_DOCUMENT_12', 'FLAG_DOCUMENT_13', 'FLAG_DOCUMENT_14',
       'FLAG_DOCUMENT_15', 'FLAG_DOCUMENT_16', 'FLAG_DOCUMENT_17',
       'FLAG_DOCUMENT_18', 'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20',
       'FLAG_DOCUMENT_21'],
      dtype='object')}
------------------------------
{'float64': Index(['AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE',
       'REGION_POPULATION_RELATIVE', 'DAYS_REGISTRATION', 'OWN_CAR_AGE',
       'CNT_FAM_MEMBERS', 'EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3',
       'APARTMENTS_AVG', 'BASEMENTAREA_AVG', 'YEARS_BEGINEXPLUATATION_AVG',
       'YEARS_BUILD_AVG', 'COMMONAREA_AVG', 'ELEVATORS_AVG', 'ENTRANCES_AVG',
       'FLOORSMAX_AVG', 'FLOORSMIN_AVG', 'LANDAREA_AVG',
       'LIVINGAPARTMENTS_AVG', 'LIVINGAREA_AVG', 'NONLIVINGAPARTMENTS_AVG',
```

```
       'NONLIVINGAREA_AVG', 'APARTMENTS_MODE', 'BASEMENTAREA_MODE',
       'YEARS_BEGINEXPLUATATION_MODE', 'YEARS_BUILD_MODE', 'COMMONAREA_MODE',
       'ELEVATORS_MODE', 'ENTRANCES_MODE', 'FLOORSMAX_MODE', 'FLOORSMIN_MODE',
       'LANDAREA_MODE', 'LIVINGAPARTMENTS_MODE', 'LIVINGAREA_MODE',
       'NONLIVINGAPARTMENTS_MODE', 'NONLIVINGAREA_MODE', 'APARTMENTS_MEDI',
       'BASEMENTAREA_MEDI', 'YEARS_BEGINEXPLUATATION_MEDI', 'YEARS_BUILD_MEDI',
       'COMMONAREA_MEDI', 'ELEVATORS_MEDI', 'ENTRANCES_MEDI', 'FLOORSMAX_MEDI',
       'FLOORSMIN_MEDI', 'LANDAREA_MEDI', 'LIVINGAPARTMENTS_MEDI',
       'LIVINGAREA_MEDI', 'NONLIVINGAPARTMENTS_MEDI', 'NONLIVINGAREA_MEDI',
       'TOTALAREA_MODE', 'OBS_30_CNT_SOCIAL_CIRCLE',
       'DEF_30_CNT_SOCIAL_CIRCLE', 'OBS_60_CNT_SOCIAL_CIRCLE',
       'DEF_60_CNT_SOCIAL_CIRCLE', 'DAYS_LAST_PHONE_CHANGE',
       'AMT_REQ_CREDIT_BUREAU_HOUR', 'AMT_REQ_CREDIT_BUREAU_DAY',
       'AMT_REQ_CREDIT_BUREAU_WEEK', 'AMT_REQ_CREDIT_BUREAU_MON',
       'AMT_REQ_CREDIT_BUREAU_QRT', 'AMT_REQ_CREDIT_BUREAU_YEAR'],
      dtype='object')}
------------------------------
{'object': Index(['NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY',
       'NAME_TYPE_SUITE', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE',
       'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'OCCUPATION_TYPE',
       'WEEKDAY_APPR_PROCESS_START', 'ORGANIZATION_TYPE', 'FONDKAPREMONT_MODE',
       'HOUSETYPE_MODE', 'WALLSMATERIAL_MODE', 'EMERGENCYSTATE_MODE'],
      dtype='object')}
------------------------------


------------------------------------------------------------------------------
------------------------------------------------------------------------------

 The Missing Data:
```

	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

	Percent	Train Missing Count
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
BASEMENTAREA_MEDI	58.52	179943
BASEMENTAREA_AVG	58.52	179943
BASEMENTAREA_MODE	58.52	179943
EXT_SOURCE_1	56.38	173378
NONLIVINGAREA_MODE	55.18	169682
NONLIVINGAREA_AVG	55.18	169682
NONLIVINGAREA_MEDI	55.18	169682
ELEVATORS_MEDI	53.30	163891
ELEVATORS_AVG	53.30	163891
ELEVATORS_MODE	53.30	163891
WALLSMATERIAL_MODE	50.84	156341
APARTMENTS_MEDI	50.75	156061
APARTMENTS_AVG	50.75	156061

	Percent	Train Missing Count
APARTMENTS_MODE	50.75	156061
ENTRANCES_MEDI	50.35	154828
ENTRANCES_AVG	50.35	154828
ENTRANCES_MODE	50.35	154828
LIVINGAREA_AVG	50.19	154350
LIVINGAREA_MODE	50.19	154350
LIVINGAREA_MEDI	50.19	154350
HOUSETYPE_MODE	50.18	154297
FLOORSMAX_MODE	49.76	153020
FLOORSMAX_MEDI	49.76	153020
FLOORSMAX_AVG	49.76	153020
YEARS_BEGINEXPLUATATION_MODE	48.78	150007
YEARS_BEGINEXPLUATATION_MEDI	48.78	150007
YEARS_BEGINEXPLUATATION_AVG	48.78	150007
TOTALAREA_MODE	48.27	148431
EMERGENCYSTATE_MODE	47.40	145755
OCCUPATION_TYPE	31.35	96391
EXT_SOURCE_3	19.83	60965
AMT_REQ_CREDIT_BUREAU_HOUR	13.50	41519
AMT_REQ_CREDIT_BUREAU_DAY	13.50	41519
AMT_REQ_CREDIT_BUREAU_WEEK	13.50	41519
AMT_REQ_CREDIT_BUREAU_MON	13.50	41519
AMT_REQ_CREDIT_BUREAU_QRT	13.50	41519
AMT_REQ_CREDIT_BUREAU_YEAR	13.50	41519
NAME_TYPE_SUITE	0.42	1292
OBS_30_CNT_SOCIAL_CIRCLE	0.33	1021

	Percent	Train Missing Count
DEF_30_CNT_SOCIAL_CIRCLE	0.33	1021
OBS_60_CNT_SOCIAL_CIRCLE	0.33	1021
DEF_60_CNT_SOCIAL_CIRCLE	0.33	1021
EXT_SOURCE_2	0.21	660
AMT_GOODS_PRICE	0.09	278

Commentary

We can see from the descriptive statistics for Days Birth, Days employed, Days registration, Days Id publish which is a negative value and is not expected.

- Own car age has a max of 91.
- Living space and realty has redundant features. This can be used for imputing the features and avaoid any issues.

Missing data for application train

In [22]:

```
percent = (datasets["application_train"].isnull().sum()/datasets["application_train"].isnull().count()*100).sor
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 Cou
missing_application_train_data.head(20)
```

Out[22]:

	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

	Percent	Train Missing Count
LANDAREA_MODE	59.38	182590
LANDAREA_AVG	59.38	182590

In [23]:

```
percent = (datasets["application_test"].isnull().sum()/datasets["application_test"].isnull().count()*100).sort_
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 Coun
missing_application_train_data.head(20)
```

Out[23]:

	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

	Percent	Test Missing Count
LANDAREA_MODE	57.96	28254

Distribution of the target column

Explore the distribution of values taken on by the target variable.

In [24]:
```
datasets["application_train"].groupby(['TARGET'])['SK_ID_CURR'].count()
```

Out[24]:
```
TARGET
0    282686
1     24825
Name: SK_ID_CURR, dtype: int64
```

In [25]:
```
datasets["application_train"]['TARGET'].plot.hist()
plt.show()
```

In [26]:

```
df_app_train=datasets["application_train"].copy()
df_app_train['DAYS_EMPLOYED_ANOM'] = df_app_train['DAYS_EMPLOYED'] == 365243
df_app_train['DAYS_EMPLOYED'].replace({365243:np.nan}, inplace=True)
plt.hist(df_app_train['DAYS_EMPLOYED'],edgecolor = 'k', bins = 25)
plt.title('DAYS_EMPLOYED'); plt.xlabel('No Of Days as per Dataset'); plt.ylabel('Count');
plt.show()
```

Number of Days employed is an important feature that can be used for predicting risk. However, the histogram shows that the data is not logical.

In [27]:

```
plt.hist(datasets["application_train"]['OWN_CAR_AGE'],edgecolor = 'k', bins = 25)
plt.title('OWN CAR AGE'); plt.xlabel('No Of Days as per Dataset'); plt.ylabel('Count');
plt.show()
```

There are number of applications that we can see from the histogram for those who have cars over 60 years old.

Correlation with the target column

In [28]:

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

pos = datasets["application_train"][['TARGET','DAYS_LAST_PHONE_CHANGE', 'REGION_RATING_CLIENT', 'REGION_RATING_
pos_corr = pos.corr()
sns.heatmap(pos_corr, annot = True, cmap='viridis')
plt.show()
```

```
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
```

In [29]:

```
print('\nMost Negative Correlations:\n', correlations.head(10))

neg = datasets["application_train"][['TARGET','EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3', 'DAYS_EMPLOYED']]
neg_corr = neg.corr()
```

```
sns.heatmap(neg_corr, annot = True, cmap='viridis')
plt.show()
```

```
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
```

The distribution of the top correlated features are plotted below.

In [30]:

```
var_neg_corr = correlations.head(10).index.values
numVar = var_neg_corr.shape[0]

plt.figure(figsize=(15,20))
for i,var in enumerate(var_neg_corr):
    dflt_var = datasets["application_train"].loc[datasets["application_train"]['TARGET']==1,var]
    dflt_non_var = datasets["application_train"].loc[datasets["application_train"]['TARGET']==0,var]

    plt.subplot(numVar,4,i+1)
    datasets["application_train"][var].hist()
```

```
    plt.title(var, fontsize = 10)
    plt.tight_layout()
plt.show()
```

Density plots of correlated features are plotted below

In [31]:

```
var_neg_corr = correlations.head(10).index.values
numVar = var_neg_corr.shape[0]

plt.figure(figsize=(10,40))
for i,var in enumerate(var_neg_corr):
    dflt_var = datasets["application_train"].loc[datasets["application_train"]['TARGET']==1,var]
    dflt_non_var = datasets["application_train"].loc[datasets["application_train"]['TARGET']==0,var]

    plt.subplot(numVar,3,i+1)
    plt.subplots_adjust(wspace=2)
    sns.kdeplot(dflt_var,label='Default')
    sns.kdeplot(dflt_non_var,label='No Default')
    #plt.xlabel(var)
    plt.ylabel('Density')
    plt.title(var, fontsize = 10)
```

```
    plt.tight_layout()
plt.show()
```

DAYS_EMPLOYED
FLOORSMAX_MODE
Density
FLOORSMAX_MODE

FLOORSMAX_AVG
1e−6
AMT_GOODS_PRICE
Density
AMT_GOODS_PRICE
1e6

FLOORSMAX_MEDI
REGION_POPULATION_RELATIVE
Density
REGION_POPULATION_RELATIVE

ELEVATORS_AVG
Density
ELEVATORS_AVG

Applicants Age

In [32]:
```
plt.hist(datasets["application_train"]['DAYS_BIRTH'] / -365, edgecolor = 'k', bins = 25)
plt.title('Age of Client'); plt.xlabel('Age (years)'); plt.ylabel('Count');
plt.show()
```

Applicants occupations

In [33]:
```
sns.countplot(x='OCCUPATION_TYPE', data=datasets["application_train"]);
plt.title('Applicants Occupation');
plt.xticks(rotation=90);
plt.show()
```

distribution of credit amounts

In [34]:

```
sns.histplot(x='AMT_CREDIT', data=datasets["application_train"], bins=50);
plt.title('Distribution of AMT_CREDIT');
plt.show()
#Credit amounts are right skewed and outlier exists
```

Visualize income vs loan amount, identified by default

In [35]:

```
fig, ax = plt.subplots(figsize=(10, 10))
f = sns.scatterplot(data = datasets["application_train"], x = 'AMT_INCOME_TOTAL', y = 'AMT_CREDIT', hue = 'TARG
f.set(xlim=(0, 1000000))
plt.show()
```

3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
AMT_CREDIT
0.0
0.2
0.4
0.6
0.8
1.0
1e6
AMT_INCOME_TOTAL

In [36]:

```
plt.subplots(figsize=(15, 15))
d = sns.boxplot(x = datasets["application_train"]['NAME_EDUCATION_TYPE'],
            y = datasets["application_train"]['AMT_CREDIT'],
            hue = datasets["application_train"]['NAME_FAMILY_STATUS'], palette="Set3")
d.set(ylim=(0, 2000000))
plt.xticks(rotation=90)
plt.show()
```

Dataset questions

Unique record for each SK_ID_CURR

In [37]:
```
datasets.keys()
```

Out[37]:
```
dict_keys(['application_train', 'application_test', 'bureau', 'bureau_balance', 'credit_card_balance', 'install
ments_payments', 'previous_application', 'POS_CASH_balance'])
```

In [38]:
```
len(datasets["application_train"]["SK_ID_CURR"].unique()) == datasets["application_train"].shape[0]
```

Out[38]:
```
True
```

In [39]:
```
np.intersect1d(datasets["application_train"]["SK_ID_CURR"], datasets["application_test"]["SK_ID_CURR"])
```

Out[39]:
```
array([], dtype=int64)
```

In [40]:
```
datasets["application_test"].shape
```

Out[40]:
```
(48744, 121)
```

In [41]:
```
datasets["application_train"].shape
```

Out[41]:
```
(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 [42]:
```
appsDF = datasets["previous_application"]
```

In [58]:
```
appsDF.shape
```

Out[58]:
```
(1670214, 37)
```

In [44]:
```
len(np.intersect1d(datasets["previous_application"]["SK_ID_CURR"], datasets["application_test"]["SK_ID_CURR"]))
```

Out[44]:
```
47800
```

In [45]:
```
print(f"There are  {appsDF.shape[0]:,} previous applications")
```

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

In [46]:
```
# How many entries are there for each month?
prevAppCounts = appsDF['SK_ID_CURR'].value_counts(dropna=False)
#prevAppCounts
```

In [47]:
```
len(prevAppCounts[prevAppCounts >40])  #more that 40 previous applications
```

Out[47]:
```
101
```

In [48]:
```
prevAppCounts[prevAppCounts >50].plot(kind='bar')
plt.xticks(rotation=25)
plt.show()
```

Histogram of Number of previous applications for an ID

In [49]:
```
sum(appsDF['SK_ID_CURR'].value_counts()==1)
```

Out[49]:
```
60458
```

In [50]:
```
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')
plt.show()
```

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 [51]:
```
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_40plus)/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

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

Aggregate using one or more operations over the specified axis.

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	AMT_CREDIT	AMT_DOWN_PAYMENT	AMT
6	2315218	175704	Cash loans	NaN	0.0	0.0	NaN	

In [53]:
```
appsDF[0:50][(appsDF["SK_ID_CURR"]==175704) & ~(appsDF["AMT_CREDIT"]==1.0)]
```

Out[53]:

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

Missing values in prevApps

In [54]:

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

Out[54]:

```
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 [55]:

```
appsDF.columns
```

Out[55]:

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

feature engineering for prevApp table

In [ ]:

```
# 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
# result.columns = result.columns.map('_'.join)
# display(result)
# result['range_AMT_APPLICATION'] = result['AMT_APPLICATION_max'] - result['AMT_APPLICATION_min']
# print(f"result.shape: {result.shape}")
# result[0:10]
```

```
        AMT_ANNUITY  AMT_APPLICATION
count  1.297979e+06     1.670214e+06
mean   1.595512e+04     1.752339e+05
std    1.478214e+04     2.927798e+05
min    0.000000e+00     0.000000e+00
25%    6.321780e+03     1.872000e+04
```

```
50%    1.125000e+04     7.104600e+04
75%    2.065842e+04     1.803600e+05
max    4.180581e+05     6.905160e+06
```

	SK_ID_CURR	SK_ID_PREV	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	AMT_GOODS_PRICE	HOUR_
0	100001	1.369693e+06	3951.000	24835.50	23787.00	2520.0	24835.5	
1	100002	1.038818e+06	9251.775	179055.00	179055.00	0.0	179055.0	
2	100003	2.281150e+06	56553.990	435436.50	484191.00	3442.5	435436.5	
3	100004	1.564014e+06	5357.250	24282.00	20106.00	4860.0	24282.0	
4	100005	2.176837e+06	4813.200	22308.75	20076.75	4464.0	44617.5	

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

In [ ]:

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

feature transformer for prevApp table

In [60]:

```
# # 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 = ["min", "max", "mean"]

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

#     def transform(self, X, y=None):
#         #from IPython.core.debugger import Pdb as pdb;    pdb().set_trace() #breakpoint; dont forget to quit
#         result = X.groupby(["SK_ID_CURR"], as_index=False).agg("mean") #group by ID
#         result = appsDF.groupby(["SK_ID_CURR"], as_index=False).agg({'AMT_ANNUITY' : self.agg_op_features, 'A
#         result.columns = result.columns.map('_'.join)
#         #display(result)
#         #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"
# #         result[0:10]

# 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_APPLICATION  \
0     2030495      271877     Consumer loans     1730.430          17145.0
1     2802425      108129         Cash loans    25188.615         607500.0
2     2523466      122040         Cash loans    15060.735         112500.0
3     2819243      176158         Cash loans    47041.335         450000.0
4     1784265      202054         Cash loans    31924.395         337500.0
5     1383531      199383         Cash loans    23703.930         315000.0
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_PROCESS_START  \
0     17145.0               0.0          17145.0                  SATURDAY
1    679671.0               NaN         607500.0                  THURSDAY
2    136444.5               NaN         112500.0                   TUESDAY
3    470790.0               NaN         450000.0                    MONDAY
4    404055.0               NaN         337500.0                  THURSDAY
5    340573.5               NaN         315000.0                  SATURDAY
6         0.0               NaN              NaN                   TUESDAY
7         0.0               NaN              NaN                    MONDAY
8         0.0               NaN              NaN                    MONDAY
9         0.0               NaN              NaN                  SATURDAY
```

```
   HOUR_APPR_PROCESS_START FLAG_LAST_APPL_PER_CONTRACT  \
0                       15                           Y
1                       11                           Y
2                       11                           Y
3                        7                           Y
4                        9                           Y
5                        8                           Y
6                       11                           Y
7                        7                           Y
8                       15                           Y
9                       15                           Y

   NFLAG_LAST_APPL_IN_DAY  RATE_DOWN_PAYMENT  RATE_INTEREST_PRIMARY  \
0                       1                0.0               0.182832
1                       1                NaN                    NaN
2                       1                NaN                    NaN
3                       1                NaN                    NaN
4                       1                NaN                    NaN
5                       1                NaN                    NaN
6                       1                NaN                    NaN
7                       1                NaN                    NaN
8                       1                NaN                    NaN
9                       1                NaN                    NaN

   RATE_INTEREST_PRIVILEGED NAME_CASH_LOAN_PURPOSE NAME_CONTRACT_STATUS  \
0                  0.867336                    XAP             Approved
1                       NaN                    XNA             Approved
2                       NaN                    XNA             Approved
3                       NaN                    XNA             Approved
4                       NaN                Repairs              Refused
5                       NaN      Everyday expenses             Approved
6                       NaN                    XNA             Canceled
7                       NaN                    XNA             Canceled
8                       NaN                    XNA             Canceled
9                       NaN                    XNA             Canceled

   DAYS_DECISION      NAME_PAYMENT_TYPE CODE_REJECT_REASON  NAME_TYPE_SUITE  \
0            -73  Cash through the bank                XAP              NaN
1           -164                    XNA                XAP    Unaccompanied
2           -301  Cash through the bank                XAP  Spouse, partner
3           -512  Cash through the bank                XAP              NaN
4           -781  Cash through the bank                 HC              NaN
5           -684  Cash through the bank                XAP           Family
6            -14                    XNA                XAP              NaN
7            -21                    XNA                XAP              NaN
8           -386                    XNA                XAP              NaN
```

```
9            -57                    XNA                XAP               NaN

  NAME_CLIENT_TYPE NAME_GOODS_CATEGORY NAME_PORTFOLIO NAME_PRODUCT_TYPE  \
0         Repeater              Mobile            POS               XNA
1         Repeater                 XNA           Cash            x-sell
2         Repeater                 XNA           Cash            x-sell
3         Repeater                 XNA           Cash            x-sell
4         Repeater                 XNA           Cash           walk-in
5         Repeater                 XNA           Cash            x-sell
6         Repeater                 XNA            XNA               XNA
7         Repeater                 XNA            XNA               XNA
8         Repeater                 XNA            XNA               XNA
9         Repeater                 XNA            XNA               XNA

             CHANNEL_TYPE  SELLERPLACE_AREA NAME_SELLER_INDUSTRY  \
0            Country-wide               35         Connectivity
1          Contact center               -1                  XNA
2  Credit and cash offices               -1                  XNA
3  Credit and cash offices               -1                  XNA
4  Credit and cash offices               -1                  XNA
5  Credit and cash offices               -1                  XNA
6  Credit and cash offices               -1                  XNA
7  Credit and cash offices               -1                  XNA
8  Credit and cash offices               -1                  XNA
9  Credit and cash offices               -1                  XNA

   CNT_PAYMENT NAME_YIELD_GROUP       PRODUCT_COMBINATION  DAYS_FIRST_DRAWING  \
0         12.0           middle  POS mobile with interest            365243.0
1         36.0       low_action          Cash X-Sell: low            365243.0
2         12.0             high         Cash X-Sell: high            365243.0
3         12.0           middle       Cash X-Sell: middle            365243.0
4         24.0             high         Cash Street: high                 NaN
5         18.0       low_normal          Cash X-Sell: low            365243.0
6          NaN              XNA                      Cash                 NaN
7          NaN              XNA                      Cash                 NaN
8          NaN              XNA                      Cash                 NaN
9          NaN              XNA                      Cash                 NaN

   DAYS_FIRST_DUE  DAYS_LAST_DUE_1ST_VERSION  DAYS_LAST_DUE  DAYS_TERMINATION  \
0           -42.0                      300.0          -42.0             -37.0
1          -134.0                      916.0       365243.0          365243.0
2          -271.0                       59.0       365243.0          365243.0
3          -482.0                     -152.0         -182.0            -177.0
4             NaN                        NaN            NaN               NaN
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
```

Feature Engineering on Primary & Secondary Datasets

Merge secondardy dataset with Primary dataset's (*application_train*) target variable to understand correlation between target variable and the secondary dataset's features.

In [60]:

```
def correlation_with_target(df):
    app_train = datasets["application_train"].copy()
    second_df = datasets[df].copy()
    corr_matrix = pd.concat([app_train.TARGET, second_df], axis=1).corr().filter(second_df.columns).filter(app_
    return corr_matrix
```

The following secondary datasets will be explored for correlation against the target variable.

In [61]:

```
for dataset in datasets.keys():
    print(dataset)
```

```
application_train
application_test
bureau
bureau_balance
credit_card_balance
installments_payments
previous_application
POS_CASH_balance
```

In [62]:

```
df_name = "bureau"
```

```
correlation_matrix = correlation_with_target(df_name)
print(f"Correlation of the {df_name} against the Target is :")
correlation_matrix.T.TARGET.sort_values(ascending= False)
```

Out[62]:

```
Correlation of the bureau against the Target is :
DAYS_CREDIT_UPDATE        0.002159
DAYS_CREDIT_ENDDATE       0.002048
SK_ID_BUREAU              0.001550
DAYS_CREDIT               0.001443
AMT_CREDIT_SUM            0.000218
DAYS_ENDDATE_FACT         0.000203
AMT_ANNUITY               0.000189
AMT_CREDIT_MAX_OVERDUE   -0.000389
CNT_CREDIT_PROLONG       -0.000495
AMT_CREDIT_SUM_LIMIT     -0.000558
AMT_CREDIT_SUM_DEBT      -0.000946
SK_ID_CURR               -0.001070
AMT_CREDIT_SUM_OVERDUE   -0.001464
CREDIT_DAY_OVERDUE       -0.001815
Name: TARGET, dtype: float64
```

In [63]:

```
df_name = "bureau_balance"
correlation_matrix = correlation_with_target(df_name)
print(f"Correlation of the {df_name} against the Target is :")
correlation_matrix.T.TARGET.sort_values(ascending= False)
```

Out[63]:

```
Correlation of the bureau_balance against the Target is :
SK_ID_BUREAU      0.001223
MONTHS_BALANCE   -0.005262
Name: TARGET, dtype: float64
```

In [64]:

```
df_name = "credit_card_balance"
correlation_matrix = correlation_with_target(df_name)
print(f"Correlation of the {df_name} against the Target is :")
correlation_matrix.T.TARGET.sort_values(ascending= False)
```

Out[64]:

```
Correlation of the credit_card_balance against the Target is :
CNT_DRAWINGS_ATM_CURRENT      0.001908
AMT_DRAWINGS_ATM_CURRENT      0.001520
AMT_INST_MIN_REGULARITY       0.001435
SK_ID_CURR                    0.001086
AMT_CREDIT_LIMIT_ACTUAL       0.000515
AMT_BALANCE                   0.000448
SK_ID_PREV                    0.000446
```

```
AMT_RECIVABLE                  0.000412
AMT_TOTAL_RECEIVABLE           0.000407
AMT_RECEIVABLE_PRINCIPAL       0.000383
SK_DPD                         0.000092
SK_DPD_DEF                    -0.000201
CNT_INSTALMENT_MATURE_CUM     -0.000342
MONTHS_BALANCE                -0.000768
AMT_PAYMENT_CURRENT           -0.001129
AMT_PAYMENT_TOTAL_CURRENT     -0.001395
AMT_DRAWINGS_CURRENT          -0.001419
CNT_DRAWINGS_CURRENT          -0.001764
CNT_DRAWINGS_OTHER_CURRENT    -0.001833
CNT_DRAWINGS_POS_CURRENT      -0.002387
AMT_DRAWINGS_OTHER_CURRENT    -0.002672
AMT_DRAWINGS_POS_CURRENT      -0.003518
Name: TARGET, dtype: float64
```

In [65]:

```
df_name = "installments_payments"
correlation_matrix = correlation_with_target(df_name)
print(f"Correlation of the {df_name} against the Target is :")
correlation_matrix.T.TARGET.sort_values(ascending= False)
```

Out[65]:

```
Correlation of the installments_payments against the Target is :
SK_ID_PREV                0.002891
NUM_INSTALMENT_VERSION    0.002511
NUM_INSTALMENT_NUMBER     0.000626
SK_ID_CURR               -0.000781
AMT_PAYMENT              -0.003512
DAYS_INSTALMENT          -0.003955
AMT_INSTALMENT           -0.003972
DAYS_ENTRY_PAYMENT       -0.004046
Name: TARGET, dtype: float64
```

In [66]:

```
df_name = "previous_application"
correlation_matrix = correlation_with_target(df_name)
print(f"Correlation of the {df_name} against the Target is :")
correlation_matrix.T.TARGET.sort_values(ascending= False)
```

Out[66]:

```
Correlation of the previous_application against the Target is :
AMT_DOWN_PAYMENT             0.002496
CNT_PAYMENT                  0.002341
DAYS_LAST_DUE_1ST_VERSION    0.001908
AMT_CREDIT                   0.001833
AMT_APPLICATION              0.001689
AMT_GOODS_PRICE              0.001676
```

```
SK_ID_CURR                   0.001107
NFLAG_INSURED_ON_APPROVAL    0.000879
RATE_DOWN_PAYMENT            0.000850
RATE_INTEREST_PRIMARY        0.000542
SK_ID_PREV                   0.000362
DAYS_DECISION               -0.000482
AMT_ANNUITY                 -0.000492
DAYS_FIRST_DUE              -0.000943
SELLERPLACE_AREA            -0.000954
DAYS_TERMINATION            -0.001072
NFLAG_LAST_APPL_IN_DAY      -0.001256
DAYS_FIRST_DRAWING          -0.001293
DAYS_LAST_DUE               -0.001940
HOUR_APPR_PROCESS_START     -0.002285
RATE_INTEREST_PRIVILEGED    -0.026427
Name: TARGET, dtype: float64
```

In [67]:

```
df_name = "POS_CASH_balance"
correlation_matrix = correlation_with_target(df_name)
print(f"Correlation of the {df_name} against the Target is :")
correlation_matrix.T.TARGET.sort_values(ascending= False)
```

Out[67]:

```
Correlation of the POS_CASH_balance against the Target is :
CNT_INSTALMENT_FUTURE    0.002811
MONTHS_BALANCE           0.002775
SK_ID_PREV               0.002164
CNT_INSTALMENT           0.001434
SK_DPD                   0.000050
SK_ID_CURR              -0.000136
SK_DPD_DEF              -0.001362
Name: TARGET, dtype: float64
```

Create Feature Aggregators

In [92]:

```
# Pipelines
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import LabelEncoder
from sklearn.base import BaseEstimator, TransformerMixin
from sklearn.pipeline import make_pipeline, Pipeline, FeatureUnion
from sklearn.preprocessing import MinMaxScaler, StandardScaler, OneHotEncoder


class FeaturesAggregator(BaseEstimator, TransformerMixin):
```

```
    def __init__(self, file_name=None, features=None, funcs=None): # no *args or **kargs
        self.file_name = file_name
        self.features = features
        self.funcs = funcs
        self.agg_op_features = {}
        for f in self.features:
            temp = {f"{file_name}_{f}_{func}":func for func in self.funcs}
            self.agg_op_features[f]=[(k, v) for k, v in temp.items()]
        print(self.agg_op_features)
    def fit(self, X, y=None):
        return self

    def transform(self, X, y=None):
        #from IPython.core.debugger import Pdb as pdb;    pdb().set_trace() #breakpoint; dont forget to quit
        result = X.groupby(["SK_ID_CURR"]).agg(self.agg_op_features)
        result.columns = result.columns.droplevel()
        result = result.reset_index(level=["SK_ID_CURR"])
        return result # return dataframe with the join key "SK_ID_CURR"
```

In [93]:

```
agg_funcs = ['min', 'max', 'mean']

# prevApps = datasets['previous_application']
prevApps_features = ['AMT_ANNUITY', 'AMT_APPLICATION','AMT_DOWN_PAYMENT','CNT_PAYMENT','RATE_INTEREST_PRIVILEGE

# bureau = datasets['bureau']
bureau_features = ['AMT_ANNUITY', 'AMT_CREDIT_SUM','DAYS_CREDIT','AMT_CREDIT_SUM_OVERDUE','CREDIT_DAY_OVERDUE']
# bureau_funcs = ['min', 'max', 'mean', 'count', 'sum']

# bureau_bal = datasets['bureau_balance']
bureau_bal_features = ['MONTHS_BALANCE']

# cc_bal = datasets['credit_card_balance']
cc_bal_features = ['MONTHS_BALANCE', 'AMT_BALANCE', 'CNT_INSTALMENT_MATURE_CUM','AMT_DRAWINGS_ATM_CURRENT'
                   ,'AMT_INST_MIN_REGULARITY','AMT_PAYMENT_TOTAL_CURRENT']

# installments_pmnts = datasets['installments_payments']
installments_pmnts_features = ['AMT_INSTALMENT', 'AMT_PAYMENT']

pos_cash_balance_features = ['CNT_INSTALMENT_FUTURE','MONTHS_BALANCE','SK_DPD_DEF']
```

Engineer New Features

Engineer new features capturing relationship between income and credit amount as well as annuity and income for Application

dataset

In [94]:
```
class engineer_features(BaseEstimator, TransformerMixin):
    def __init__(self, features=None):
        self

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

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

        # FROM APPLICATION
        # Income Credit Percentage
        X['ef_INCOME_CREDIT_PERCENT'] = (
            X.AMT_INCOME_TOTAL / X.AMT_CREDIT).replace(np.inf, 0)

        # Annuity as Percentage of Annual Income
        X['ef_ANN_INCOME_PERCENT'] = (
            X.AMT_ANNUITY / X.AMT_INCOME_TOTAL).replace(np.inf, 0)

        return X
```

Engineer new features capturing range of annuity, application, and downpayment amounts from the Previous Application dataset

In [95]:
```
class prevApp_engineer_features(BaseEstimator, TransformerMixin):
    def __init__(self, features=None):
        self

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

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

        # FROM PREVIOUS APPLICATION
        # Add Annuity, Application, and Downpaymet ranges
        X['ef_prevApps_AMT_ANNUITY_range'] = (X.prevApps_AMT_ANNUITY_max - X.prevApps_AMT_ANNUITY_min).replace(
        X['ef_prevApps_AMT_APPLICATION_range'] = (X.prevApps_AMT_APPLICATION_max - X.prevApps_AMT_APPLICATION_m
        X['ef_prevApps_AMT_DOWN_PAYMENT_range'] = (X.prevApps_AMT_DOWN_PAYMENT_max - X.prevApps_AMT_DOWN_PAYMEN

        return X
```

Build Pipeline for each Dataset

In [96]:

```
from sklearn.pipeline import make_pipeline, Pipeline

prevApps_features_pipeline = Pipeline([
#        ('prevApps_add_features1', prevApps_add_features1()),  # add some new features
#        ('prevApps_add_features2', prevApps_add_features2()),  # add some new features
    ('prevApps_aggregator', FeaturesAggregator('prevApps', prevApps_features, agg_funcs)), # Aggregate across c
    ('prevApp_engineer_features', prevApp_engineer_features())
    ])

bureau_features_pipeline = Pipeline([
    ('bureau_aggregator', FeaturesAggregator('bureau', bureau_features, agg_funcs)), # Aggregate across old anc
    ])

bureau_bal_features_pipeline = Pipeline([
    ('bureau_bal_aggregator', FeaturesAggregator('bureau_balance', bureau_bal_features , agg_funcs)), # Aggrega
    ])

cc_bal_features_pipeline = Pipeline([
    ('cc_bal_aggregator', FeaturesAggregator('credit_card_balance', cc_bal_features , agg_funcs)), # Aggregate
    ])

installments_pmnts_features_pipeline = Pipeline([
    ('installments_pmnts_features_aggregator', FeaturesAggregator('installments_pmnts', installments_pmnts_feat
    ])

pos_cash_balance_features_pipeline = Pipeline([
    ('pos_cash_balance_features_aggregator', FeaturesAggregator('pos_cash_balance', pos_cash_balance_features ,
    ])
```

```
{'AMT_ANNUITY': [('prevApps_AMT_ANNUITY_min', 'min'), ('prevApps_AMT_ANNUITY_max', 'max'), ('prevApps_AMT_ANNUI
TY_mean', 'mean')], 'AMT_APPLICATION': [('prevApps_AMT_APPLICATION_min', 'min'), ('prevApps_AMT_APPLICATION_ma
x', 'max'), ('prevApps_AMT_APPLICATION_mean', 'mean')], 'AMT_DOWN_PAYMENT': [('prevApps_AMT_DOWN_PAYMENT_min',
'min'), ('prevApps_AMT_DOWN_PAYMENT_max', 'max'), ('prevApps_AMT_DOWN_PAYMENT_mean', 'mean')], 'CNT_PAYMENT':
[('prevApps_CNT_PAYMENT_min', 'min'), ('prevApps_CNT_PAYMENT_max', 'max'), ('prevApps_CNT_PAYMENT_mean', 'mea
n')], 'RATE_INTEREST_PRIVILEGED': [('prevApps_RATE_INTEREST_PRIVILEGED_min', 'min'), ('prevApps_RATE_INTEREST_P
RIVILEGED_max', 'max'), ('prevApps_RATE_INTEREST_PRIVILEGED_mean', 'mean')]}
{'AMT_ANNUITY': [('bureau_AMT_ANNUITY_min', 'min'), ('bureau_AMT_ANNUITY_max', 'max'), ('bureau_AMT_ANNUITY_mea
n', 'mean')], 'AMT_CREDIT_SUM': [('bureau_AMT_CREDIT_SUM_min', 'min'), ('bureau_AMT_CREDIT_SUM_max', 'max'),
('bureau_AMT_CREDIT_SUM_mean', 'mean')], 'DAYS_CREDIT': [('bureau_DAYS_CREDIT_min', 'min'), ('bureau_DAYS_CREDI
T_max', 'max'), ('bureau_DAYS_CREDIT_mean', 'mean')], 'AMT_CREDIT_SUM_OVERDUE': [('bureau_AMT_CREDIT_SUM_OVERDU
E_min', 'min'), ('bureau_AMT_CREDIT_SUM_OVERDUE_max', 'max'), ('bureau_AMT_CREDIT_SUM_OVERDUE_mean', 'mean')],
'CREDIT_DAY_OVERDUE': [('bureau_CREDIT_DAY_OVERDUE_min', 'min'), ('bureau_CREDIT_DAY_OVERDUE_max', 'max'), ('bu
reau_CREDIT_DAY_OVERDUE_mean', 'mean')]}
{'MONTHS_BALANCE': [('bureau_balance_MONTHS_BALANCE_min', 'min'), ('bureau_balance_MONTHS_BALANCE_max', 'max'),
```

```
('bureau_balance_MONTHS_BALANCE_mean', 'mean')]}
{'MONTHS_BALANCE': [('credit_card_balance_MONTHS_BALANCE_min', 'min'), ('credit_card_balance_MONTHS_BALANCE_ma
x', 'max'), ('credit_card_balance_MONTHS_BALANCE_mean', 'mean')], 'AMT_BALANCE': [('credit_card_balance_AMT_BAL
ANCE_min', 'min'), ('credit_card_balance_AMT_BALANCE_max', 'max'), ('credit_card_balance_AMT_BALANCE_mean', 'me
an')], 'CNT_INSTALMENT_MATURE_CUM': [('credit_card_balance_CNT_INSTALMENT_MATURE_CUM_min', 'min'), ('credit_car
d_balance_CNT_INSTALMENT_MATURE_CUM_max', 'max'), ('credit_card_balance_CNT_INSTALMENT_MATURE_CUM_mean', 'mea
n')], 'AMT_DRAWINGS_ATM_CURRENT': [('credit_card_balance_AMT_DRAWINGS_ATM_CURRENT_min', 'min'), ('credit_card_b
alance_AMT_DRAWINGS_ATM_CURRENT_max', 'max'), ('credit_card_balance_AMT_DRAWINGS_ATM_CURRENT_mean', 'mean')],
'AMT_INST_MIN_REGULARITY': [('credit_card_balance_AMT_INST_MIN_REGULARITY_min', 'min'), ('credit_card_balance_A
MT_INST_MIN_REGULARITY_max', 'max'), ('credit_card_balance_AMT_INST_MIN_REGULARITY_mean', 'mean')], 'AMT_PAYMEN
T_TOTAL_CURRENT': [('credit_card_balance_AMT_PAYMENT_TOTAL_CURRENT_min', 'min'), ('credit_card_balance_AMT_PAYM
ENT_TOTAL_CURRENT_max', 'max'), ('credit_card_balance_AMT_PAYMENT_TOTAL_CURRENT_mean', 'mean')]}
{'AMT_INSTALMENT': [('installments_pmnts_AMT_INSTALMENT_min', 'min'), ('installments_pmnts_AMT_INSTALMENT_max',
'max'), ('installments_pmnts_AMT_INSTALMENT_mean', 'mean')], 'AMT_PAYMENT': [('installments_pmnts_AMT_PAYMENT_m
in', 'min'), ('installments_pmnts_AMT_PAYMENT_max', 'max'), ('installments_pmnts_AMT_PAYMENT_mean', 'mean')]}
{'CNT_INSTALMENT_FUTURE': [('pos_cash_balance_CNT_INSTALMENT_FUTURE_min', 'min'), ('pos_cash_balance_CNT_INSTAL
MENT_FUTURE_max', 'max'), ('pos_cash_balance_CNT_INSTALMENT_FUTURE_mean', 'mean')], 'MONTHS_BALANCE': [('pos_ca
sh_balance_MONTHS_BALANCE_min', 'min'), ('pos_cash_balance_MONTHS_BALANCE_max', 'max'), ('pos_cash_balance_MONT
HS_BALANCE_mean', 'mean')], 'SK_DPD_DEF': [('pos_cash_balance_SK_DPD_DEF_min', 'min'), ('pos_cash_balance_SK_DP
D_DEF_max', 'max'), ('pos_cash_balance_SK_DPD_DEF_mean', 'mean')]}
```

Engineer New features for Application Train Dataset

In [97]:

```
# Feature engineering pipeline for application_train
appln_new_features_pipeline = Pipeline([
    ('engineer_features', engineer_features()),  # add some new features
    ])
```

Prepare Datasets

In [98]:

```
# Primary Application Training Dataset
appsTrainDF = datasets['application_train']

# Secondary Datasets
prevAppsDF = datasets["previous_application"] #prev app
bureauDF = datasets["bureau"] #bureau app
bureaubalDF = datasets['bureau_balance']
ccbalDF = datasets["credit_card_balance"] #prev app
installmentspaymentsDF = datasets["installments_payments"] #bureau app
posbalDF = datasets['POS_CASH_balance']
```

Create Aggregate datasets after performing fit & transform

In [99]:
```
appsTrainDF_agg = appln_new_features_pipeline.fit_transform(appsTrainDF)

prevApps_agg = prevApps_features_pipeline.fit_transform(prevAppsDF)
# prevApps_agg = prevApp_new_features_pipeline.fit_transform(prevApps_agg)
bureau_agg = bureau_features_pipeline.fit_transform(bureauDF)
# bureaubal_agg = bureau_bal_features_pipeline.fit_transform(bureaubalDF)
ccblance_agg = cc_bal_features_pipeline.fit_transform(ccbalDF)
installments_pmnts_agg = installments_pmnts_features_pipeline.fit_transform(installmentspaymentsDF)
posbal_agg = pos_cash_balance_features_pipeline.fit_transform(posbalDF)
```

Join the labeled dataset

In [100…
```
~3==3
```

Out[100…
```
False
```

In [101…
```
datasets.keys()
```

Out[101…
```
dict_keys(['application_train', 'application_test', 'bureau', 'bureau_balance', 'credit_card_balance', 'install
ments_payments', 'previous_application', 'POS_CASH_balance'])
```

Perform data merging of primary application and secondary datasets.

In [102…
```
merge_all_data = True

# if merge_all_data:
#     prevApps_aggregated = prevApps_feature_pipeline.transform(appsDF)

# merge primary table and secondary tables using features based on meta data and  aggregage stats
if merge_all_data:
    appsTrainDF_agg = appsTrainDF_agg.merge(prevApps_agg, how='left', on='SK_ID_CURR')
    appsTrainDF_agg = appsTrainDF_agg.merge(bureau_agg, how='left', on="SK_ID_CURR")
    appsTrainDF_agg = appsTrainDF_agg.merge(ccblance_agg, how='left', on="SK_ID_CURR")
    appsTrainDF_agg = appsTrainDF_agg.merge(installments_pmnts_agg, how='left', on="SK_ID_CURR")
    appsTrainDF_agg = appsTrainDF_agg.merge(posbal_agg, how='left', on="SK_ID_CURR")
```

In [103…
```
appsTrainDF_agg.shape
```

Out[103…
```
(307511, 190)
```

In [104…

```
print(appsTrainDF.shape)
```

```
(307511, 124)
```

Check presence of newly engineered features

In [105…

```
appsTrainDF_agg[['ef_INCOME_CREDIT_PERCENT', 'ef_ANN_INCOME_PERCENT','ef_prevApps_AMT_ANNUITY_range'
                 , 'ef_prevApps_AMT_APPLICATION_range', 'ef_prevApps_AMT_DOWN_PAYMENT_range']].head()
```

Out[105…

	ef_INCOME_CREDIT_PERCENT	ef_ANN_INCOME_PERCENT	ef_prevApps_AMT_ANNUITY_range	ef_prevApps_AMT_APPLICATION_range
0	0.498036	0.121978	0.000	0.0
1	0.208736	0.132217	91619.685	831190.5
2	0.500000	0.100000	0.000	0.0
3	0.431748	0.219900	37471.590	688500.0
4	0.236842	0.179963	20844.495	230323.5

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

Perform data merging of primary application and secondary datasets.

In [107…

```
X_kaggle_test = datasets["application_test"]
X_kaggle_test = appln_new_features_pipeline.fit_transform(X_kaggle_test)

merge_all_data = True
if merge_all_data:
    X_kaggle_test = X_kaggle_test.merge(prevApps_agg, how='left', on='SK_ID_CURR')
    X_kaggle_test = X_kaggle_test.merge(bureau_agg, how='left', on="SK_ID_CURR")
    X_kaggle_test = X_kaggle_test.merge(ccblance_agg, how='left', on="SK_ID_CURR")
    X_kaggle_test = X_kaggle_test.merge(installments_pmnts_agg, how='left', on="SK_ID_CURR")
    X_kaggle_test = X_kaggle_test.merge(posbal_agg, how='left', on="SK_ID_CURR")
```

In [108…

```
print(X_kaggle_test.shape)

X_kaggle_test.head(3)
```

```
(48744, 189)
```

Out[108…

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL
0	100001	Cash loans	F	N	Y	0	135000.0
1	100005	Cash loans	M	N	Y	0	99000.0
2	100013	Cash loans	M	Y	Y	0	202500.0

In [109…

```
appsTrainDF_agg.columns.tolist()
```

Out[109…

```
['SK_ID_CURR',
 'TARGET',
 'NAME_CONTRACT_TYPE',
 'CODE_GENDER',
 'FLAG_OWN_CAR',
 'FLAG_OWN_REALTY',
 'CNT_CHILDREN',
 'AMT_INCOME_TOTAL',
 'AMT_CREDIT',
 'AMT_ANNUITY',
 'AMT_GOODS_PRICE',
 'NAME_TYPE_SUITE',
 'NAME_INCOME_TYPE',
 'NAME_EDUCATION_TYPE',
 'NAME_FAMILY_STATUS',
 'NAME_HOUSING_TYPE',
 'REGION_POPULATION_RELATIVE',
 'DAYS_BIRTH',
 'DAYS_EMPLOYED',
 'DAYS_REGISTRATION',
 'DAYS_ID_PUBLISH',
 'OWN_CAR_AGE',
 'FLAG_MOBIL',
 'FLAG_EMP_PHONE',
 'FLAG_WORK_PHONE',
 'FLAG_CONT_MOBILE',
 'FLAG_PHONE',
 'FLAG_EMAIL',
 'OCCUPATION_TYPE',
 'CNT_FAM_MEMBERS',
 'REGION_RATING_CLIENT',
 'REGION_RATING_CLIENT_W_CITY',
 'WEEKDAY_APPR_PROCESS_START',
 'HOUR_APPR_PROCESS_START',
```

```
'REG_REGION_NOT_LIVE_REGION',
'REG_REGION_NOT_WORK_REGION',
'LIVE_REGION_NOT_WORK_REGION',
'REG_CITY_NOT_LIVE_CITY',
'REG_CITY_NOT_WORK_CITY',
'LIVE_CITY_NOT_WORK_CITY',
'ORGANIZATION_TYPE',
'EXT_SOURCE_1',
'EXT_SOURCE_2',
'EXT_SOURCE_3',
'APARTMENTS_AVG',
'BASEMENTAREA_AVG',
'YEARS_BEGINEXPLUATATION_AVG',
'YEARS_BUILD_AVG',
'COMMONAREA_AVG',
'ELEVATORS_AVG',
'ENTRANCES_AVG',
'FLOORSMAX_AVG',
'FLOORSMIN_AVG',
'LANDAREA_AVG',
'LIVINGAPARTMENTS_AVG',
'LIVINGAREA_AVG',
'NONLIVINGAPARTMENTS_AVG',
'NONLIVINGAREA_AVG',
'APARTMENTS_MODE',
'BASEMENTAREA_MODE',
'YEARS_BEGINEXPLUATATION_MODE',
'YEARS_BUILD_MODE',
'COMMONAREA_MODE',
'ELEVATORS_MODE',
'ENTRANCES_MODE',
'FLOORSMAX_MODE',
'FLOORSMIN_MODE',
'LANDAREA_MODE',
'LIVINGAPARTMENTS_MODE',
'LIVINGAREA_MODE',
'NONLIVINGAPARTMENTS_MODE',
'NONLIVINGAREA_MODE',
'APARTMENTS_MEDI',
'BASEMENTAREA_MEDI',
'YEARS_BEGINEXPLUATATION_MEDI',
'YEARS_BUILD_MEDI',
'COMMONAREA_MEDI',
'ELEVATORS_MEDI',
'ENTRANCES_MEDI',
'FLOORSMAX_MEDI',
```

```
'FLOORSMIN_MEDI',
'LANDAREA_MEDI',
'LIVINGAPARTMENTS_MEDI',
'LIVINGAREA_MEDI',
'NONLIVINGAPARTMENTS_MEDI',
'NONLIVINGAREA_MEDI',
'FONDKAPREMONT_MODE',
'HOUSETYPE_MODE',
'TOTALAREA_MODE',
'WALLSMATERIAL_MODE',
'EMERGENCYSTATE_MODE',
'OBS_30_CNT_SOCIAL_CIRCLE',
'DEF_30_CNT_SOCIAL_CIRCLE',
'OBS_60_CNT_SOCIAL_CIRCLE',
'DEF_60_CNT_SOCIAL_CIRCLE',
'DAYS_LAST_PHONE_CHANGE',
'FLAG_DOCUMENT_2',
'FLAG_DOCUMENT_3',
'FLAG_DOCUMENT_4',
'FLAG_DOCUMENT_5',
'FLAG_DOCUMENT_6',
'FLAG_DOCUMENT_7',
'FLAG_DOCUMENT_8',
'FLAG_DOCUMENT_9',
'FLAG_DOCUMENT_10',
'FLAG_DOCUMENT_11',
'FLAG_DOCUMENT_12',
'FLAG_DOCUMENT_13',
'FLAG_DOCUMENT_14',
'FLAG_DOCUMENT_15',
'FLAG_DOCUMENT_16',
'FLAG_DOCUMENT_17',
'FLAG_DOCUMENT_18',
'FLAG_DOCUMENT_19',
'FLAG_DOCUMENT_20',
'FLAG_DOCUMENT_21',
'AMT_REQ_CREDIT_BUREAU_HOUR',
'AMT_REQ_CREDIT_BUREAU_DAY',
'AMT_REQ_CREDIT_BUREAU_WEEK',
'AMT_REQ_CREDIT_BUREAU_MON',
'AMT_REQ_CREDIT_BUREAU_QRT',
'AMT_REQ_CREDIT_BUREAU_YEAR',
'ef_INCOME_CREDIT_PERCENT',
'ef_ANN_INCOME_PERCENT',
'prevApps_AMT_ANNUITY_min',
'prevApps_AMT_ANNUITY_max',
```

```
'prevApps_AMT_ANNUITY_mean',
'prevApps_AMT_APPLICATION_min',
'prevApps_AMT_APPLICATION_max',
'prevApps_AMT_APPLICATION_mean',
'prevApps_AMT_DOWN_PAYMENT_min',
'prevApps_AMT_DOWN_PAYMENT_max',
'prevApps_AMT_DOWN_PAYMENT_mean',
'prevApps_CNT_PAYMENT_min',
'prevApps_CNT_PAYMENT_max',
'prevApps_CNT_PAYMENT_mean',
'prevApps_RATE_INTEREST_PRIVILEGED_min',
'prevApps_RATE_INTEREST_PRIVILEGED_max',
'prevApps_RATE_INTEREST_PRIVILEGED_mean',
'ef_prevApps_AMT_ANNUITY_range',
'ef_prevApps_AMT_APPLICATION_range',
'ef_prevApps_AMT_DOWN_PAYMENT_range',
'bureau_AMT_ANNUITY_min',
'bureau_AMT_ANNUITY_max',
'bureau_AMT_ANNUITY_mean',
'bureau_AMT_CREDIT_SUM_min',
'bureau_AMT_CREDIT_SUM_max',
'bureau_AMT_CREDIT_SUM_mean',
'bureau_DAYS_CREDIT_min',
'bureau_DAYS_CREDIT_max',
'bureau_DAYS_CREDIT_mean',
'bureau_AMT_CREDIT_SUM_OVERDUE_min',
'bureau_AMT_CREDIT_SUM_OVERDUE_max',
'bureau_AMT_CREDIT_SUM_OVERDUE_mean',
'bureau_CREDIT_DAY_OVERDUE_min',
'bureau_CREDIT_DAY_OVERDUE_max',
'bureau_CREDIT_DAY_OVERDUE_mean',
'credit_card_balance_MONTHS_BALANCE_min',
'credit_card_balance_MONTHS_BALANCE_max',
'credit_card_balance_MONTHS_BALANCE_mean',
'credit_card_balance_AMT_BALANCE_min',
'credit_card_balance_AMT_BALANCE_max',
'credit_card_balance_AMT_BALANCE_mean',
'credit_card_balance_CNT_INSTALMENT_MATURE_CUM_min',
'credit_card_balance_CNT_INSTALMENT_MATURE_CUM_max',
'credit_card_balance_CNT_INSTALMENT_MATURE_CUM_mean',
'credit_card_balance_AMT_DRAWINGS_ATM_CURRENT_min',
'credit_card_balance_AMT_DRAWINGS_ATM_CURRENT_max',
'credit_card_balance_AMT_DRAWINGS_ATM_CURRENT_mean',
'credit_card_balance_AMT_INST_MIN_REGULARITY_min',
'credit_card_balance_AMT_INST_MIN_REGULARITY_max',
'credit_card_balance_AMT_INST_MIN_REGULARITY_mean',
```

```
 'credit_card_balance_AMT_PAYMENT_TOTAL_CURRENT_min',
 'credit_card_balance_AMT_PAYMENT_TOTAL_CURRENT_max',
 'credit_card_balance_AMT_PAYMENT_TOTAL_CURRENT_mean',
 'installments_pmnts_AMT_INSTALMENT_min',
 'installments_pmnts_AMT_INSTALMENT_max',
 'installments_pmnts_AMT_INSTALMENT_mean',
 'installments_pmnts_AMT_PAYMENT_min',
 'installments_pmnts_AMT_PAYMENT_max',
 'installments_pmnts_AMT_PAYMENT_mean',
 'pos_cash_balance_CNT_INSTALMENT_FUTURE_min',
 'pos_cash_balance_CNT_INSTALMENT_FUTURE_max',
 'pos_cash_balance_CNT_INSTALMENT_FUTURE_mean',
 'pos_cash_balance_MONTHS_BALANCE_min',
 'pos_cash_balance_MONTHS_BALANCE_max',
 'pos_cash_balance_MONTHS_BALANCE_mean',
 'pos_cash_balance_SK_DPD_DEF_min',
 'pos_cash_balance_SK_DPD_DEF_max',
 'pos_cash_balance_SK_DPD_DEF_mean']
```

In [ ]:

```
# approval rate 'NFLAG_INSURED_ON_APPROVAL'
```

In [110…

```
# 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_CONTRACT_TYPE',
               'NAME_EDUCATION_TYPE','OCCUPATION_TYPE','NAME_INCOME_TYPE']

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

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

HCDR preprocessing

In [111…

```
#train_dataset = datasets["application_train"]
train_dataset=appsTrainDF
class_labels = ["No Default","Default"]
```

Dataframe Column Selector

In [112…

```
# 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
```

Numerical Pipeline Set-up

In [113…

```
# Identify the numeric features we wish to consider.
num_attribs = [
'AMT_INCOME_TOTAL',
'AMT_CREDIT',
'EXT_SOURCE_3',
'EXT_SOURCE_2',
'EXT_SOURCE_1',
'DAYS_EMPLOYED',
'DAYS_BIRTH',
'FLOORSMAX_AVG',
'FLOORSMAX_MEDI',
'FLOORSMAX_MODE',
'AMT_GOODS_PRICE',
'REGION_POPULATION_RELATIVE',
'REG_CITY_NOT_LIVE_CITY',
'FLAG_EMP_PHONE',
'REG_CITY_NOT_WORK_CITY',
'DAYS_ID_PUBLISH',
'DAYS_LAST_PHONE_CHANGE',
'REGION_RATING_CLIENT',
'REGION_RATING_CLIENT_W_CITY',
'ef_INCOME_CREDIT_PERCENT',
'ef_ANN_INCOME_PERCENT',
## Highly correlated previous applications
'prevApps_AMT_ANNUITY_mean',
'prevApps_AMT_DOWN_PAYMENT_min',
'prevApps_AMT_DOWN_PAYMENT_mean',
'prevApps_CNT_PAYMENT_max',
'prevApps_RATE_INTEREST_PRIVILEGED_mean',
```

```
'ef_prevApps_AMT_ANNUITY_range',
'ef_prevApps_AMT_APPLICATION_range',
'ef_prevApps_AMT_DOWN_PAYMENT_range',
## Highly correlated Bureau features
'bureau_AMT_ANNUITY_mean',
'bureau_AMT_CREDIT_SUM_mean',
'bureau_DAYS_CREDIT_mean',
'bureau_DAYS_CREDIT_max',
## Highly correlated Installment Payment features
'installments_pmnts_AMT_INSTALMENT_min',
'installments_pmnts_AMT_INSTALMENT_max',
'installments_pmnts_AMT_INSTALMENT_mean',
'installments_pmnts_AMT_PAYMENT_mean',
## Highly correlated Credit card balance features
'credit_card_balance_MONTHS_BALANCE_min',
'credit_card_balance_MONTHS_BALANCE_max',
'credit_card_balance_MONTHS_BALANCE_mean',
'credit_card_balance_AMT_BALANCE_min',
'credit_card_balance_AMT_BALANCE_max',
'credit_card_balance_AMT_BALANCE_mean',
'credit_card_balance_AMT_DRAWINGS_ATM_CURRENT_mean',
'credit_card_balance_AMT_INST_MIN_REGULARITY_mean',
'credit_card_balance_AMT_PAYMENT_TOTAL_CURRENT_mean',
## Highly correlated POS balance features
'pos_cash_balance_CNT_INSTALMENT_FUTURE_min',
'pos_cash_balance_CNT_INSTALMENT_FUTURE_max',
'pos_cash_balance_CNT_INSTALMENT_FUTURE_mean',
'pos_cash_balance_MONTHS_BALANCE_mean'
]
```

In [114…

```
print('Number of numerical features: ', len(num_attribs))
```

```
Number of numerical features:  50
```

In [115…

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

Categorical Attributes

In [116…

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

In [117…

```
print('Number of numerical features: ', len(cat_attribs))
```

```
Number of numerical features:  7
```

In [118…

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

Create Consolidated Data Pipeline

Use ColumnTransformer instead of FeatureUnion

In [119…

```
from sklearn.compose import ColumnTransformer

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

data_prep_pipeline = ColumnTransformer(transformers=[
      #( name, transformer,     columns)
       ("num_pipeline", num_pipeline, num_attribs),
       ("cat_pipeline", cat_pipeline, cat_attribs),
    ],
        n_jobs=-1
   )
```

Summarize Features Considered and Lengths

In [120…

```
print('Numerical Feature Family: ', num_attribs)
print('Numerical Feature Count:  ', len(num_attribs))
print('--------------------------')
```

```
print('Categorical Feature Family: ', cat_attribs)
print('Categorical Feature Count:  ', len(cat_attribs))
print('--------------------------')
print('Total Number of Input Features: ', len(num_attribs) + len(cat_attribs))
```

```
Numerical Feature Family:  ['AMT_INCOME_TOTAL', 'AMT_CREDIT', 'EXT_SOURCE_3', 'EXT_SOURCE_2', 'EXT_SOURCE_1',
'DAYS_EMPLOYED', 'DAYS_BIRTH', 'FLOORSMAX_AVG', 'FLOORSMAX_MEDI', 'FLOORSMAX_MODE', 'AMT_GOODS_PRICE', 'REGION_
POPULATION_RELATIVE', 'REG_CITY_NOT_LIVE_CITY', 'FLAG_EMP_PHONE', 'REG_CITY_NOT_WORK_CITY', 'DAYS_ID_PUBLISH',
'DAYS_LAST_PHONE_CHANGE', 'REGION_RATING_CLIENT', 'REGION_RATING_CLIENT_W_CITY', 'ef_INCOME_CREDIT_PERCENT', 'e
f_ANN_INCOME_PERCENT', 'prevApps_AMT_ANNUITY_mean', 'prevApps_AMT_DOWN_PAYMENT_min', 'prevApps_AMT_DOWN_PAYMENT
_mean', 'prevApps_CNT_PAYMENT_max', 'prevApps_RATE_INTEREST_PRIVILEGED_mean', 'ef_prevApps_AMT_ANNUITY_range',
'ef_prevApps_AMT_APPLICATION_range', 'ef_prevApps_AMT_DOWN_PAYMENT_range', 'bureau_AMT_ANNUITY_mean', 'bureau_A
MT_CREDIT_SUM_mean', 'bureau_DAYS_CREDIT_mean', 'bureau_DAYS_CREDIT_max', 'installments_pmnts_AMT_INSTALMENT_mi
n', 'installments_pmnts_AMT_INSTALMENT_max', 'installments_pmnts_AMT_INSTALMENT_mean', 'installments_pmnts_AMT_
PAYMENT_mean', 'credit_card_balance_MONTHS_BALANCE_min', 'credit_card_balance_MONTHS_BALANCE_max', 'credit_card
_balance_MONTHS_BALANCE_mean', 'credit_card_balance_AMT_BALANCE_min', 'credit_card_balance_AMT_BALANCE_max', 'c
redit_card_balance_AMT_BALANCE_mean', 'credit_card_balance_AMT_DRAWINGS_ATM_CURRENT_mean', 'credit_card_balance
_AMT_INST_MIN_REGULARITY_mean', 'credit_card_balance_AMT_PAYMENT_TOTAL_CURRENT_mean', 'pos_cash_balance_CNT_INS
TALMENT_FUTURE_min', 'pos_cash_balance_CNT_INSTALMENT_FUTURE_max', 'pos_cash_balance_CNT_INSTALMENT_FUTURE_mea
n', 'pos_cash_balance_MONTHS_BALANCE_mean']
Numerical Feature Count:   50
--------------------------
Categorical Feature Family:  ['CODE_GENDER', 'FLAG_OWN_REALTY', 'FLAG_OWN_CAR', 'NAME_CONTRACT_TYPE', 'NAME_EDU
CATION_TYPE', 'OCCUPATION_TYPE', 'NAME_INCOME_TYPE']
Categorical Feature Count:   7
--------------------------
Total Number of Input Features:  57
```

In [121…

```
selected_features = num_attribs + cat_attribs
len(selected_features)
```

Out[121…

```
57
```

Feature Engineering

In [105…

```
# 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', 'Accountant
#         #df['OCCUPATION_TYPE'] = df['OCCUPATION_TYPE'].apply(map(lambda x: 1. if x.lower() in ['Core Staff',
#         df['OCCUPATION_TYPE'] = df['OCCUPATION_TYPE'].apply(lambda x: 1. if x in list(map(lambda x: x.lower()
#         #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 [122…

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

Out[122…

```
['SK_ID_CURR',
 'TARGET',
 'NAME_CONTRACT_TYPE',
 'CODE_GENDER',
 'FLAG_OWN_CAR',
 'FLAG_OWN_REALTY',
 'CNT_CHILDREN',
 'AMT_INCOME_TOTAL',
 'AMT_CREDIT',
 'AMT_ANNUITY',
 'AMT_GOODS_PRICE',
```

```
'NAME_TYPE_SUITE',
'NAME_INCOME_TYPE',
'NAME_EDUCATION_TYPE',
'NAME_FAMILY_STATUS',
'NAME_HOUSING_TYPE',
'REGION_POPULATION_RELATIVE',
'DAYS_BIRTH',
'DAYS_EMPLOYED',
'DAYS_REGISTRATION',
'DAYS_ID_PUBLISH',
'OWN_CAR_AGE',
'FLAG_MOBIL',
'FLAG_EMP_PHONE',
'FLAG_WORK_PHONE',
'FLAG_CONT_MOBILE',
'FLAG_PHONE',
'FLAG_EMAIL',
'OCCUPATION_TYPE',
'CNT_FAM_MEMBERS',
'REGION_RATING_CLIENT',
'REGION_RATING_CLIENT_W_CITY',
'WEEKDAY_APPR_PROCESS_START',
'HOUR_APPR_PROCESS_START',
'REG_REGION_NOT_LIVE_REGION',
'REG_REGION_NOT_WORK_REGION',
'LIVE_REGION_NOT_WORK_REGION',
'REG_CITY_NOT_LIVE_CITY',
'REG_CITY_NOT_WORK_CITY',
'LIVE_CITY_NOT_WORK_CITY',
'ORGANIZATION_TYPE',
'EXT_SOURCE_1',
'EXT_SOURCE_2',
'EXT_SOURCE_3',
'APARTMENTS_AVG',
'BASEMENTAREA_AVG',
'YEARS_BEGINEXPLUATATION_AVG',
'YEARS_BUILD_AVG',
'COMMONAREA_AVG',
'ELEVATORS_AVG',
'ENTRANCES_AVG',
'FLOORSMAX_AVG',
'FLOORSMIN_AVG',
'LANDAREA_AVG',
'LIVINGAPARTMENTS_AVG',
'LIVINGAREA_AVG',
'NONLIVINGAPARTMENTS_AVG',
```

```
'NONLIVINGAREA_AVG',
'APARTMENTS_MODE',
'BASEMENTAREA_MODE',
'YEARS_BEGINEXPLUATATION_MODE',
'YEARS_BUILD_MODE',
'COMMONAREA_MODE',
'ELEVATORS_MODE',
'ENTRANCES_MODE',
'FLOORSMAX_MODE',
'FLOORSMIN_MODE',
'LANDAREA_MODE',
'LIVINGAPARTMENTS_MODE',
'LIVINGAREA_MODE',
'NONLIVINGAPARTMENTS_MODE',
'NONLIVINGAREA_MODE',
'APARTMENTS_MEDI',
'BASEMENTAREA_MEDI',
'YEARS_BEGINEXPLUATATION_MEDI',
'YEARS_BUILD_MEDI',
'COMMONAREA_MEDI',
'ELEVATORS_MEDI',
'ENTRANCES_MEDI',
'FLOORSMAX_MEDI',
'FLOORSMIN_MEDI',
'LANDAREA_MEDI',
'LIVINGAPARTMENTS_MEDI',
'LIVINGAREA_MEDI',
'NONLIVINGAPARTMENTS_MEDI',
'NONLIVINGAREA_MEDI',
'FONDKAPREMONT_MODE',
'HOUSETYPE_MODE',
'TOTALAREA_MODE',
'WALLSMATERIAL_MODE',
'EMERGENCYSTATE_MODE',
'OBS_30_CNT_SOCIAL_CIRCLE',
'DEF_30_CNT_SOCIAL_CIRCLE',
'OBS_60_CNT_SOCIAL_CIRCLE',
'DEF_60_CNT_SOCIAL_CIRCLE',
'DAYS_LAST_PHONE_CHANGE',
'FLAG_DOCUMENT_2',
'FLAG_DOCUMENT_3',
'FLAG_DOCUMENT_4',
'FLAG_DOCUMENT_5',
'FLAG_DOCUMENT_6',
'FLAG_DOCUMENT_7',
'FLAG_DOCUMENT_8',
```

```
 'FLAG_DOCUMENT_9',
 'FLAG_DOCUMENT_10',
 'FLAG_DOCUMENT_11',
 'FLAG_DOCUMENT_12',
 'FLAG_DOCUMENT_13',
 'FLAG_DOCUMENT_14',
 'FLAG_DOCUMENT_15',
 'FLAG_DOCUMENT_16',
 'FLAG_DOCUMENT_17',
 'FLAG_DOCUMENT_18',
 'FLAG_DOCUMENT_19',
 'FLAG_DOCUMENT_20',
 'FLAG_DOCUMENT_21',
 'AMT_REQ_CREDIT_BUREAU_HOUR',
 'AMT_REQ_CREDIT_BUREAU_DAY',
 'AMT_REQ_CREDIT_BUREAU_WEEK',
 'AMT_REQ_CREDIT_BUREAU_MON',
 'AMT_REQ_CREDIT_BUREAU_QRT',
 'AMT_REQ_CREDIT_BUREAU_YEAR',
 'ef_INCOME_CREDIT_PERCENT',
 'ef_ANN_INCOME_PERCENT']
```

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 [141…

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

In [144…

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

```
                                    "Train RMSE",
                                    "Valid RMSE",
                                    "Test  RMSE",
                                    "Train MAE",
                                    "Valid MAE",
                                    "Test  MAE",
                                    "Train Time",
                                    "Test Time",
                                    "Description"
                                   ])
```

Split Application into Train-Test split

In [145…

```
train_dataset = appsTrainDF_agg
```

In [146…

```
train_dataset.shape
```

Out[146…

```
(307511, 190)
```

In [147…

```
# Split Sample to feed the pipeline and it will result in a new dataset that is (1 / splits) the size
splits = 3

# Train Test split percentage
subsample_rate = 0.3

finaldf = np.array_split(train_dataset, splits)
X_train = finaldf[0][selected_features]
y_train = finaldf[0]['TARGET']
X_kaggle_test= X_kaggle_test[selected_features]

## split part of data
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, stratify=y_train,
                                                    test_size=subsample_rate, random_state=42)

X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train,stratify=y_train,test_size=0.15, random_

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

```
X train           shape: (60989, 57)
X validation      shape: (10763, 57)
X test            shape: (30752, 57)
X X_kaggle_test   shape: (48744, 57)
```

Define Pipeline

In [148…

```
%%time
np.random.seed(42)
full_pipeline_with_predictor = Pipeline([
        ("preparation", data_prep_pipeline),
        ("logistic", LogisticRegression())
    ])
```

```
CPU times: user 5.12 ms, sys: 0 ns, total: 5.12 ms
Wall time: 5.56 ms
```

Perform cross-fold validation and Train the model

Split the training data to 10 fold to perform Crossfold validation

In [149…

```
# Import Model selection libraries
from sklearn.model_selection import ShuffleSplit
from sklearn.model_selection import cross_val_score
```

In [150…

```
# Use ShuffleSplit() with 30 splits, 30% test_size and random state of 0
cvSplits = ShuffleSplit(n_splits=10, test_size=0.3, random_state=0)
```

In [151…

```
from time import time, ctime

start = time()
model = full_pipeline_with_predictor.fit(X_train, y_train)
np.random.seed(42)

# Set up cross validation scores
logit_scores = cross_val_score(full_pipeline_with_predictor, X_train, y_train, cv=cvSplits)
logit_score_train = pct(logit_scores.mean())
train_time = np.round(time() - start, 4)
```

```
# Time and score test predictions
start = time()
logit_score_test  = full_pipeline_with_predictor.score(X_test, y_test)
test_time = np.round(time() - start, 4)
```

Evaluation metrics

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

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

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

In [152…

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, accuracy_score, roc_auc_score
from scipy import stats
```

In [154…

```
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]),
                0,   # p-value not relevant for Baseline model
                np.round(np.sqrt(mean_squared_error(y_train, model.predict_proba(X_train)[:, 1])), 3),
                np.round(np.sqrt(mean_squared_error(y_valid, model.predict_proba(X_valid)[:, 1])), 3),
                np.round(np.sqrt(mean_squared_error(y_test, model.predict_proba(X_test)[:, 1])), 3),
                np.round(mean_absolute_error(y_train, model.predict_proba(X_train)[:, 1]), 3),
                np.round(mean_absolute_error(y_valid, model.predict_proba(X_valid)[:, 1]), 3),
                np.round(mean_absolute_error(y_test, model.predict_proba(X_test)[:, 1]), 3)
               ], 4)) \
                + [train_time,test_time] + [f"Baseline LR {len(selected_features)}"]

expLog
```

Out[154…

	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC	P Score	Train RMSE	Valid RMSE	Test RMSE	Train MAE	Valid MAE	Test MAE	Train Time	Test Time
0	Baseline_57_features	0.9189	0.9194	0.9184	0.7538	0.744	0.7478	0.0	0.262	0.262	0.263	0.137	0.138	0.137	21.7724	0.6446

Confusion Matrix

In [155…

```
from sklearn.metrics import confusion_matrix
```

In [156…

```
def confusion_matrix_def(model,X_train,y_train,X_test,y_test):
    #Prediction
    preds_test = model.predict(X_test)
    preds_train = model.predict(X_train)
    cm_train = confusion_matrix(y_train, preds_train).astype(np.float32)
    #print(cm_train)
    cm_train /= cm_train.sum(axis=1)[:, np.newaxis]

    cm_test = confusion_matrix(y_test, preds_test).astype(np.float32)
    #print(cm_test)
    cm_test /= cm_test.sum(axis=1)[:, np.newaxis]

    class_labels = ['No Default','Default']

    plt.figure(figsize=(20, 8))

    plt.subplot(121)
    g = sns.heatmap(cm_train, vmin=0, vmax=1, annot=True, cmap="Reds")
    plt.xlabel("Predicted", fontsize=14)
    plt.ylabel("True", fontsize=14)
    g.set(xticklabels=class_labels, yticklabels=class_labels)
    plt.title("Train", fontsize=14)

    plt.subplot(122)
    g = sns.heatmap(cm_test, vmin=0, vmax=1, annot=True, cmap="Reds")
    plt.xlabel("Predicted", fontsize=14)
    plt.ylabel("True", fontsize=14)
    g.set(xticklabels=class_labels, yticklabels=class_labels)
    plt.title("Test", fontsize=14) ;
```

In [157…

```
# Confusion matrix for baseline model
```

```
confusion_matrix_def(model,X_train,y_train,X_test,y_test)
plt.show()
```

Tune Baseline Model Parameters with GridSearch

The baseline Logistic Regression model was tuned across different parameters evaluated for the following metrics:

- Train Accuracy
- Cross fold Train Accuracy
- Test Accuracy
- Train AUC Score
- Cross fold AUC Score
- Test AUC Score
- p-value
- Train RMSE Score

- Cross fold RMSE Score
- Test RMSE Score
- Train MAE Score
- Cross fold MAE Score
- Test MAE Score

Perform Grid Search

In [158…

```
import json
```

In [159…

```
classifiers = [
        ('Logistic Regression', LogisticRegression(random_state=42))
   ]

params_grid = {
        'Logistic Regression': {
            'penalty': ('l1', 'l2'),
            'tol': (0.0001, 0.00001, 0.0000001),
            'C': (10, 1, 0.1, 0.01),
        }
}
```

In [164…

```
results = [logit_scores]
names = ['Baseline LR']
for (name, classifier) in classifiers:
        # Print classifier and parameters
        print('****** START', name,'*****')
        parameters = params_grid[name]
        print("Parameters:")
        for p in sorted(parameters.keys()):
            print("\t"+str(p)+": "+ str(parameters[p]))

        # generate the pipeline
        full_pipeline_with_predictor = Pipeline([
        ("preparation", data_prep_pipeline),
        ("predictor", classifier)
        ])

        # Execute the grid search
        params = {}
```

```
        for p in parameters.keys():
            pipe_key = 'predictor__'+str(p)
            params[pipe_key] = parameters[p]
        grid_search = GridSearchCV(full_pipeline_with_predictor, params, cv=5,
                                   n_jobs=-1,verbose=1)
        grid_search.fit(X_train, y_train)

        # Best estimator score
        best_train = pct(grid_search.best_score_)

        # Best estimator fitting time
        start = time()
        model = grid_search.best_estimator_.fit(X_train, y_train)
        train_time = round(time() - start, 4)

        # Best estimator prediction time
        start = time()
        best_test_accuracy = pct(grid_search.best_estimator_.score(X_test, y_test))
        test_time = round(time() - start, 4)

        # Best train scores
        best_train_scores = cross_val_score(grid_search.best_estimator_, X_train, y_train,cv=cvSplits)
        best_train_accuracy = pct(best_train_scores.mean())
        results.append(best_train_scores)
        names.append(name)

        # Conduct t-test with baseline logit (control) and best estimator (experiment)
        (t_stat, p_value) = stats.ttest_rel(logit_scores, best_train_scores)

        # Create confusion matrix for the best model
        confusion_matrix_def(model,X_train,y_train,X_test,y_test)
        plt.show()

        # Collect the best parameters found by the grid search
        print("Best Parameters:")
        best_parameters = grid_search.best_estimator_.get_params()
        param_dump = []
        for param_name in sorted(params.keys()):
            param_dump.append((param_name, best_parameters[param_name]))
            print("\t"+str(param_name)+": " + str(best_parameters[param_name]))
        print("****** FINISH",name," *****")
        print("")

        # Record the results
        exp_name = name
        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]),
                 p_value,
                 np.round(np.sqrt(mean_squared_error(y_train, model.predict_proba(X_train)[:, 1])), 3),
                 np.round(np.sqrt(mean_squared_error(y_valid, model.predict_proba(X_valid)[:, 1])), 3),
                 np.round(np.sqrt(mean_squared_error(y_test, model.predict_proba(X_test)[:, 1])), 3),
                 np.round(mean_absolute_error(y_train, model.predict_proba(X_train)[:, 1]), 3),
                 np.round(mean_absolute_error(y_valid, model.predict_proba(X_valid)[:, 1]), 3),
                 np.round(mean_absolute_error(y_test, model.predict_proba(X_test)[:, 1]), 3)
                ], 4)) \
                 + [train_time,test_time] \
                 + [json.dumps(param_dump)]
```

```
****** START Logistic Regression *****
Parameters:
        C: (10, 1, 0.1, 0.01)
        penalty: ('l1', 'l2')
        tol: (0.0001, 1e-05, 1e-07)
Fitting 5 folds for each of 24 candidates, totalling 120 fits
```

```
Best Parameters:
        predictor__C: 0.01
        predictor__penalty: l2
        predictor__tol: 0.0001
****** FINISH Logistic Regression  *****
```

Boxplot with CV results

In [168…

```
# boxplot algorithm comparison
fig = plt.figure()
fig.suptitle('Classification Algorithm Comparison')
ax = fig.add_subplot(111)
plt.boxplot(results)
ax.set_xticklabels(names)
plt.grid()
plt.show()
```

Final Results

In [171…

```
print('Final experiment results:')
expLog
```

```
Final experiment results:
```

Out[171…

	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC	P Score	Train RMSE	Valid RMSE	Test RMSE	Train MAE	Valid MAE	Test MAE	Train Time	Te Tim
0	Baseline_57_features	0.9189	0.9194	0.9184	0.7538	0.7440	0.7478	0.000	0.262	0.262	0.263	0.137	0.138	0.137	21.7724	0.644
1	Logistic Regression	0.9188	0.9195	0.9186	0.7509	0.7433	0.7464	0.051	0.262	0.262	0.263	0.138	0.138	0.138	1.6323	0.585

Kaggle submission via the command line API

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 [172…

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

In [173…

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

Out[173…

```
array([0.06296224, 0.22933675, 0.04993455, 0.02661781, 0.11735878,
       0.05061509, 0.02893913, 0.06891914, 0.01692273, 0.09845579])
```

In [174…

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

submit_df.head()
```

Out[174…

	SK_ID_CURR	TARGET
0	100001	0.062962
1	100005	0.229337
2	100013	0.049935
3	100028	0.026618
4	100038	0.117359

In [175…

```
submit_df.to_csv("submission.csv",index=False)
```

In [176…

```
! kaggle competitions submit -c home-credit-default-risk -f submission.csv -m "baseline submission"
```

```
100%|□□□
```

Abstract

The primary purpose of the HCDR project is to create a machine learning model which can accurately predict the customer behavior on repayment of the loan.

In the first phase of this project, we conducted basic exploratory data analysis on all the datasets, created a baseline pipeline, and selected key metrics. We then conducted a statistical analysis of the numerical and categorical features. By doing feature engineering for the highly correlated features, we were able to evaluate a better baseline.

The results we obtained in this phase indicate that there is no statistical significance between our baseline and best performing model, since the P-value of 0.051 indicates no statistical significance between the experiments. Both models have a 91.9% accuracy score and a 75% AUC across the training, validation, and test datasets.

Our ROC_AUC score for the Kaggle submission was 0.74306.

Project Description

Home Credit is an international non-bank financial institution that aims to lend people money regardless of their credit history. Home credit groups focus on providing a positive borrowing experience for customers who do not bank on traditional sources. Thus, Home Credit Group published a dataset on Kaggle with the goal of identifying and solving unfair loan rejection.

The purpose of this project is to create a machine learning model which can accurately predict the customer behavior on repayment of the loan. Our task is to form a pipeline to build a baseline machine learning model using logistic regression classification algorithms. The final model will be evaluated using a number of different performance metrics that we can use to create a better model. Businesses can use this model to identify if a loan is at risk to default. The new model that is built will ensure that the clients who are capable of repaying their loans are not rejected and that loans would be given with a principal, maturity, and repayment calendar that will allow their clients to be successful.

The results of the machine learning pipelines are measured by using these metrics: Mean Absolute Error (MAE), Root Mean Square Error (RMSE), Accuracy Score, Precision, Recall, Confusion Matrix, and Area Under ROC Curve (AUC).

The results of our pipelines will be analyzed and ranked. The most efficient pipeline will be submitted to the Kaggle competition for the Home Credit Default Risk (HCDR).

Workflow

We are implementing the following workflow outlined below. In Phase 0, we understood the project modelling requirements and outlined our plans. In Phase 1, we are performing the first among three iterations of the remainder of the workflow

Data Description

The dataset contains 1 primary table and 6 seconday tables. \ \ **Primary Tables**

1. application_train \ This Primary table includes the application information for each loan application at Home Credit in one row. This row includes the target variable of whether or not the loan was repaid. We use this field as the basis to determine the feature importance. The target variable is binary in nature based since this is a classification problem. \ \ The target variable takes on two different values:

    - '1' - client with payment difficulties: he/she had late payment more than N days on at least one of the first M installments of the loan in our sample
    - '0' - all other cases \ \ There are 122 variables and 307,511 data entries.
2. application_test \ This table includes the application information for each loan application at Home Credit in one row. The features are the same as the train data but exclude the target variable. \ \ There are 121 variables and 48,744 data entries.

Secondary Tables

1. Bureau \ This table includes all previous credits received by a customer from other financial institutions prior to their loan application. There is one row for each previous credit, meaning a many-to-one relationship with the primary table. We could join it with primary table by using current application ID, SK_ID_CURR. \ \ There are 17 variables and 1,716,428 data entries.

2. Bureau Balance \ This table includes the monthly balance for a previous credit at other financial institutions. There is one row for each monthly balance, meaning a many-to-one relationship with the Bureau table. We could join it with bureau table by using bureau's ID, SK_ID_BUREAU. \ \ There are 3 variables and 27,299,925 data entries.

3. Previous Application \ This table includes previous applications for loans made by the customer at Home Credit. There is one row for each previous application, meaning a many-to-one relationship with the primary table. We could join it with primary table by using current application ID, SK_ID_CURR. There are four types of contracts: a. Consumer loan(POS – Credit limit given to buy consumer goods) b. Cash loan(Client is given cash) c. Revolving loan(Credit) d. XNA (Contract type without values) \ \ There are 37 variables and 1,670,214 data entries.

4. POS CASH Balance \ This table includes a monthly balance snapshot of a previous point of sale or cash loan that the customer has at Home Credit. There is one row for each monthly balance, meaning a many-to-one relationship with the Previous Application table. We would join it with Previous Application table by using previous application ID, SK_ID_PREV, then join it with primary table by using current application ID, SK_ID_CURR. \ \ There are 8 variables and 10,001,358 data entries.

5. Credit Card Balance \ This table includes a monthly balance snapshot of previous credit cards the customer has with Home Credit. There is one row for each previous monthly balance, meaning a many-to-one relationship with the Previous Application table.We could join it with Previous Application table by using previous application ID, SK_ID_PREV, then join it with primary table by using current application ID, SK_ID_CURR. \ \ There are 23 variables and 3,840,312 data entries.

6. Installments Payments \ This table includes previous repayments made or not made by the customer on credits issued by Home Credit. There is one row for each payment or missed payment, meaning a many-to-one relationship with the Previous Application table. We would join it with Previous Application table by using previous application ID, SK_ID_PREV, then join it with primary table by using current application ID, SK_ID_CURR. \ \ There are 8 variables and 13,605,401 data entries.

Data Tasks

The following data preprocessing tasks need to be achieved to prepare the datasets after downloading and unzipping the main application and secondary datasets:

1. Analyze missing values from application_train table and feature correlations with target variable.
2. Examine correlations between primary dataset's target variable and features from each secondary dataset.
3. Create pipelines for primary and secondary datasets that generate minimum, maximum, and mean metrics using aggregate functions.
4. Transform for primary and secondary datasets using the pipelines.
5. Perform feature engineering to build new features for _previous*application* dataset.
6. Join the primary application dataset (labeled train and unlabeled test) with secondary tables on SK_ID_CURR. A left join is used so that any loan application record IDs that are missing secondary data are not dropped and will instead be imputed (strategy discussed in pipeline).

7. Engineer new features around claim duration attributes and Occupation Type

EDA

For the Exploratory Data Analysis component of this phase, we did a precursor analysis on the data to ensure that our results would be accurate.

We looked at summary statistics for each table in the model. We primarily focused on the data distribution, identifying statistics such as the count, mean, standard deviation, minimum, IQR, and maximum.

We also looked at specific numerical and categorical features and visualized them. We created a heatmap to identify the correlation between each feature and the target variable. We also visualized the age, occupation, and distribution of credit amounts.

Please see the Exploratory Data Analysis section for our complete EDA.

Feature Engineering and transformers

In our feature engineering process, we created two types of features to enhance our dataset. First, we created new aggregate features based on aggregate functions to capture the minimum, maximum, and mean of numerical attributes across the primary and secondary datasets that were highly correlated with the target variable.

```
agg_funcs = ['min', 'max', 'mean']

# prevApps = datasets['previous_application']
prevApps_features = ['AMT_ANNUITY', 'AMT_APPLICATION','AMT_DOWN_PAYMENT','CNT_PAYMENT','RATE_INTEREST_PRIVILEGED']

# bureau = datasets['bureau']
bureau_features = ['AMT_ANNUITY', 'AMT_CREDIT_SUM','DAYS_CREDIT','AMT_CREDIT_SUM_OVERDUE','CREDIT_DAY_OVERDUE']
# bureau_funcs = ['min', 'max', 'mean', 'count', 'sum']

# bureau_bal = datasets['bureau_balance']
bureau_bal_features = ['MONTHS_BALANCE']

# cc_bal = datasets['credit_card_balance']
cc_bal_features = ['MONTHS_BALANCE', 'AMT_BALANCE', 'CNT_INSTALMENT_MATURE_CUM','AMT_DRAWINGS_ATM_CURRENT'
                   ,'AMT_INST_MIN_REGULARITY','AMT_PAYMENT_TOTAL_CURRENT']

# installments_pmnts = datasets['installments_payments']
installments_pmnts_features = ['AMT_INSTALMENT', 'AMT_PAYMENT']

pos_cash_balance_features = ['CNT_INSTALMENT_FUTURE','MONTHS_BALANCE','SK_DPD_DEF']
```

We decided to engineer the following new features from the Application and Previous Application datasets:

- **Application_Train:**
  - ef_INCOME_CREDIT_PERCENT - client income as a percentage share of the credit amount of the loan
  - ef_ANN_INCOME_PERCENT - loan annuity as a percentage share of client income
- **Previous_Application:** (all the below values are range calculations that take the difference between the max and min aggregate values)
  - ef_prevApps_AMT_ANNUITY_range
  - ef_prevApps_AMT_APPLICATION_range
  - ef_prevApps_AMT_DOWN_PAYMENT_range

In order to identify the highly correlated features, we created a simple function that took a secondary dataframe name as an input variable and generated a correlation matrix between all the features in the inputted dataframe and the primary dataset's target variable.

All the aggregate values were calculated from the original dataframes and a new of dataframes (comprising of primary and secondary datasets) were generated. After the secondary datasets were merged with the primary "application_train" dataset, the new

consolidated application training dataframe had a total of 187 features (including the aggregate calculations for specific features).

Further, the top highly correlated features (positive and negative) were chosen from both the primary and secondary datasets. These features were then classified into numerical and categorical variables to form inputs for 2 individual pipelines. In total, our baseline model comprised of 53 features (46 numerical and 7 catgorical features).

(Please see Feature Engineering section and Feature Aggregator for more details)

Pipelines

Implementing Logistic Regression as a baseline model is a good starting point for classification tasks due to its easy implementation and low computational requirements. For the first experiment, we combined our data preparation pipeline and Logistic Regression with deafult parameters (penalty = 'l2', C = 1.0, solver = 'lbfgs', tol = 1e-4). We wanted to fine tune the regularization (l1 vs l2), tolerance, and C hyper parameters using Grid Search and compare the resulting best estimator with the baseline model. We used 5 fold cross-validation along with the hyperparameters to tune the model with GridSearchCV function in Scikit-learn.

Here is the high-level workflow for the model pipeline followed by detailed steps:

1. Download data and perform data pre-processing tasks (joining primary and secondary datasets, transformation)
2. Create data pipeline with highly correlated numerical and categorical features.
3. Impute missing numerical attribute with mean values and categorical values with most frequent values.
4. Apply ColumnTransformer to combine both Numerical and Categorical features.
5. Create model with data pipeline and baseline model to fit training dataset
6. Evaluate the model using accuracy score, AUC score, RMSE and MAE for train, validation and test datasets. Record the results in a dataframe.
7. Perform Grid Search to tune the Logistic Regression model with regularization('l1', 'l2'), tolerance(0.0001, 0.00001, 0.0000001), and C(10, 1, 0.1, 0.01) hyper parameters and 5-fold cross-validation.
8. Record the Grid Search results to the dataframe (expLog) and find the best estimator based on accuracy scores.

Two experiments were conducted in total to develop a baseline model.

Experimental results

Below is the resulting table for the two baseline models we developed on the given dataset.

Final experiment results:

	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC	P Score	Train RMSE	Valid RMSE	Test RMSE	Train MAE	Valid MAE	Test MAE	Train Time	Test Time	Description
0	Baseline_57_features	0.9189	0.9194	0.9184	0.7538	0.7440	0.7478	0.000	0.262	0.262	0.263	0.137	0.138	0.137	21.7724	0.6446	Baseline LR 57
1	Logistic Regression	0.9188	0.9195	0.9186	0.7509	0.7433	0.7464	0.051	0.262	0.262	0.263	0.138	0.138	0.138	1.6323	0.5853	[["predictor__C", 0.01], ["predictor__penalty"...

Since HCDR is a Classification task, we used the following metrics to measure the Model performance.

MAE

The mean absolute error is the average of the absolute values of individual prediction errors over all instances in the test set. Each prediction error is the difference between the true value and the predicted value for the instance.

$$\mathrm{MAE}(\mathbf{X}, h_\theta) = \frac{1}{m}\sum_{i=1}^{m} |\mathbf{x}^{(i)} \cdot \theta - y^{(i)}|$$

RMSE

This root mean square error is the normalized distance between the vector of predicted values and the vector of observed values. First, the squared difference between each observed value and predicted value is calculated. RMSE is the square root of the summation of these squared differences.

$$\mathrm{RMSE}(\mathbf{X}, h_\theta) = \sqrt{\frac{1}{m}\sum_{i=1}^{m} (\mathbf{x}^{(i)} \cdot \theta - y^{(i)})^2}$$

Accuracy Score

This metric describes the fraction of correctly classified samples. In SKLearn, it can be modified to return solely the number of correct samples.Accuracy is the default scoring method for both logistic regression and k-Nearest Neighbors in scikit-learn.

$$\textbf{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN}$$

Precision

The precision is the ratio of true positives over the total number of predicted positives.

$$P = \frac{T_p}{T_p + F_p}$$

Recall

The recall is the ratio of true positives over the true positives and false negatives. Recall is assessing the ability of the classifier to find all the positive samples. The best value is 1 and the worst value is 0

$$R = \frac{T_p}{T_p + F_n}$$

Confusion Matrix

The confusion matrix, in this case for a binary classification, is a 2x2 matrix that contains the count of the true positives, false positives, true negatives, and false negatives.

		Actual Values	
		Positive (1)	Negative (0)
Predicted Values	Positive (1)	TP	FP
	Negative (0)	FN	TN

AUC (Area under ROC curve)

An ROC curve (receiver operating characteristic curve) is a graph showing the performance of a classification model at all classification thresholds. This curve plots two parameters: ▪ True Positive Rate ▪ False Positive Rate

AUC stands for "Area under the ROC Curve." That is, AUC measures the entire two-dimensional area underneath the entire ROC curve from (0,0) to (1,1).

AUC is desirable for the following two reasons:

1. AUC is scale-invariant. It measures how well predictions are ranked, rather than their absolute values.
2. AUC is classification-threshold-invariant. It measures the quality of the model's predictions irrespective of what classification threshold is chosen.

p-value

p-value is the probability of obtaining test results at least as extreme as the results actually observed, under the assumption that the null hypothesis is correct. A very small p-value means that such an extreme observed outcome would be very unlikely under the null hypothesis.

We will compare the classifiers with the baseline untuned model by conducting two-tailed hypothesis test.

Null Hypothesis, H0: There is no significant difference between the two machine learning pipelines. Alternate Hypothesis, HA: The two machine learning pipelines are different. A p-value less than or equal to the significance level is considered statistically significant.

Discussion

In our first experiment for our baseline model, we received fairly high accuracy scores for our training and validation datasets at around 91.9%. Our test accuracy score was also similarly placed at 91.8%. While this suggests a robust model, the AUC values for training, test, and validation datasets are only between 74% to 75%. This means that there is a significant enough probabilty for false positives or false negatives. Our second experiment, in which we applied Grid Search to identify the best hyperparameters for Logistic Regression, also exhibited similar accuracy and AUC metrics. In both experiments, the RMSE and MAE values were fairly low. In addition, the tuned algorithm from Grid Search did not achieve statistical significance but came very close as the p-value was 0.051. Therefore, we fail to reject our null hypothesis as there is evidence to suggest that there is no significant difference between the two pipelines.

We believe that a high accuracy score and somewhat lower AUC curves might be partly the result of an imbalanced dataset with respect to the target variable values. From our EDA, we found that nearly 92% of our application training data had a target value of 0 (no default) with the remaining 8% having a value of 1. With such a low sample of data for loans that defaulted, it is likely that the accuracy score will end up being high as there is less scope for false positives or false negatives.

For our Kaggle submission, we used the baseline model with best parameters (second experiment) since the test accuracy was slightly better.

We opted out of using Mean Absolute Percentage Error (MAPE) as an evaluation metric (as originally proposed in Phase 0) because we were getting an undefined output ('Inf') due to division by zero in the denominator.

Conclusion

In the Home Credit Default Risk (HCDR) project, we are using Home Credit’s data to better predict repayment of a loan by a customer with little to no credit history.

In Phase 1 of our project, we designed and developed a process to ingest Home Credit's application and secondary client data, analyze the dataset features, transform and engineer the best parameters, and evaluate machine learning algorithms. Our workflow led us to a tuned algorithm that was not statistically significant compared to untuned logistic regression but came close with a p-value of 0.051

We developed and evaluated a baseline model using tuned logistic regression. Though our model exhibited high accuracy scores, there is still some room for improvement with respect to the AUC scores. In Phase 2, we plan to refine our model training process by adding additional features from the bureau datasets. While evaluating other algorithms listed in our project proposal, we also plan to capture log loss as part of our model evaluation metrics in order to get a complete picture of model performance.

Challenges

One of the major challenges faced was identifying a method to select the most relevant features. It was initially difficult to achieve a balance between too few or too many features for our baseline model. Also, as mentioned in the Conclusion, our team maintains a healthy skepticism of the results as we believe we need to resample our dataset in the next phase. We also had to focus a lot of time on getting our code to work smoothly and ensuring the basic data transformations, feature engineering, and set-up is correct in addition to analyzing the results of our baseline model.

Along the way, we faced several technical issues in developing this notebook:

- **Memory:** - all four of us had to increase our memory sizes (a member had to increase from 4GB to 11GB) and our CPUs in order load the data and prevent our kernel from crashing midway. As we added visualizations, our notebook size also increased from 4 MB to 13 MB, requiring more space to operate.
- **Outliers:** - large amounts of missing data and outliers meant that we had to perform feature engineering only with highly correlated features in order to achieve the baseline model.
- **Version Control:** - our team used Github to collaborate and push our changes. We still ran into issues with some members not being able to push their changes and having to manually upload their notebook, thereby creating some downstream issues for others trying to pull the latest notebook.

Kaggle Submission

Below is the screenshot of our best kaggle submission.

References

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

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://www.analyticsvidhya.com/blog/2018/08/guide-automated-feature-engineering-featuretools-python/
- feature engineering paper: https://dai.lids.mit.edu/wp-content/uploads/2017/10/DSAA_DSM_2015.pdf
- https://www.analyticsvidhya.com/blog/2017/08/catboost-automated-categorical-data/