Team Group 25 details

Team Members

Dharani Doppalapudi ddoppala@iu.edu	Manognya Katapally makata@iu.edu	Krishna Sannidhi ssannidh@iu.edu	Godha Priyanka Gummula ggummula@iu.edu
			

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

Project Abstract

Credit rating is an essential indicator that is commonly required for loan purposes. . In order to get any kind of loan approved, A good credit score will be a valuable addition along with the steady income. But it must not be the only reason for loan approval. Our project calculates every factor which can help a person repay the loan they took. Analysis of monthly income from all the sources, existing loans, and history of repayment will be considered before approving a loan.

In Phase 1, we have completed EDA for application_train.csv, credit_card_balance.csv, previous_appliation.csv and installments_payments.csv. The correlation between the merged features and the target variable has also been calculated. On this premise, we established the Pipeline, which consists of the most positively and negatively correlated features. Numerical pipeline and categorical pipeline make up the Pipeline. A Standard Scaler and Imputer are used in the numerical pipeline, while a Dataframe Selector and One Hot Encoder are used in the categorical pipeline. The baseline Logistic Regression Model is created using this pipeline, and it has a test accuracy of 92 percent and a train accuracy of 91.7 percent.

In Phase 2, our goal is to establish pipeline with feature engineering setup and test on different models. We have completed Feature Engineering, hyper parameter tuning, feature selection, analysis of feature importance, ensemble methods and we have constructed optimal joins for the dataset. We have checked the RFM metrics and it's matching with our data. The correlation between the merged features and the target variable has also been calculated. On this premise, we established the Pipeline, which consists of the most positively and negatively correlated features. Numerical pipeline and categorical pipeline make up the Pipeline. A Standard Scaler and Imputer are used in the numerical pipeline, while a Data frame Selector and One Hot Encoder are used in the categorical pipeline. We tested with Support Vector Machine, Random Forest, Logistic Regression and K-Nearest Neighbors models. Out of all, Random Forest model provided better accuracies and so we considered it as the best suited model for our data. The Random Forest Model has a test accuracy of 68 percent, AUC score of 97 percent, Error rate of 2.6407. Kaggle score of 0.72559 and private score of 0.72778.

Data Description

We are working on Home Credit Default Risk dataset which is hosted on Kaggle. Our project will calculate every factor which can help a person repay the loan they took. Analysis of monthly income from all the sources, existing loans, and history of repayment will be considered before approving a loan.

We are currently working on the files application_train.csv, previous_appliation.csv, installments_payments.csv and credit_card_balance.csv.

Completed Tasks

- EDA for application_train.csv
- EDA for installments_payments.csv
- EDA for credit_card_balance.csv
- EDA for previous_appliation.csv
- Feature Engineering
- Understood the data and built pipeline for submission

Tasks to be Done

- Constructing optimal joins in the dataset.
- Testing with different models and Hyper parameter tuning
- Build a multi-layer perception (MLP) model

Modling pipelines

We tested on Logistic Regression, Support Vector Machine, Decision Trees, Random Forest and K-Nearest Neighbors models.

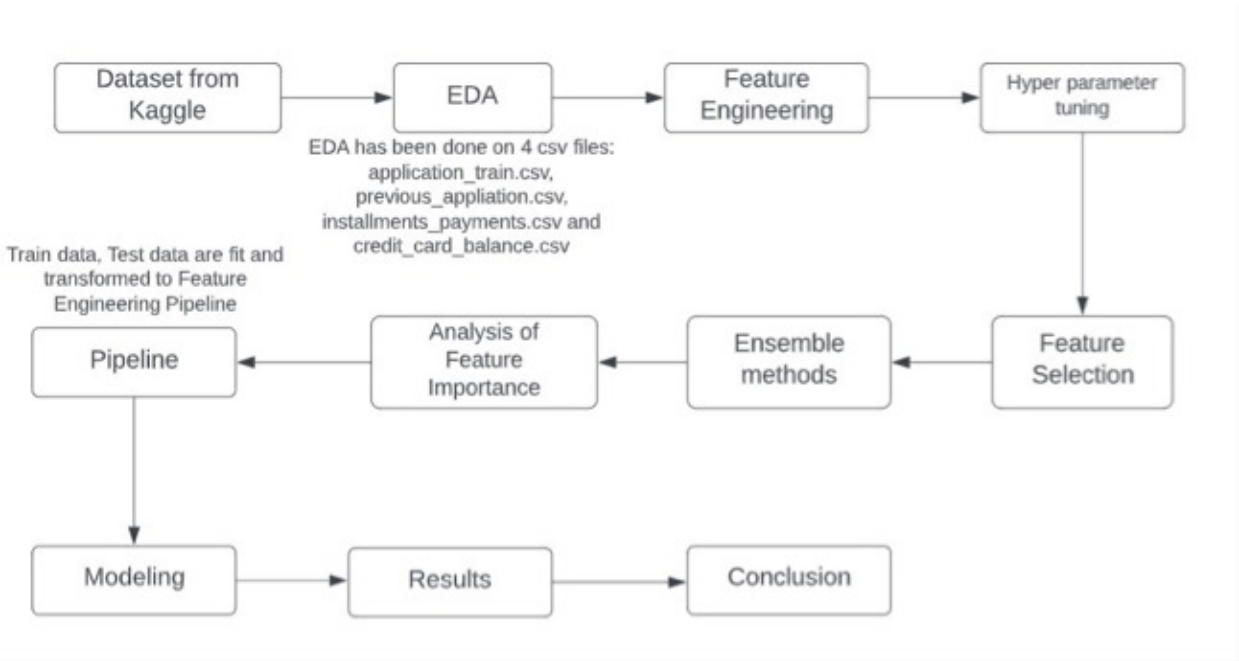
LOGISTIC REGRESSION: From the dataset we can see Target values either 0 or 1 which comes under the Binomial Logistic Regression. This model is trained to predict the probability of loan repayment dependent variable based on one or more independent variables that can be either continuous or categorical. As we are finding the probability the most suitable cost function would be log loss.

SUPPORT VECTOR MACHINE: Support Vector Machine algorithm finds the hyperplane in an N-dimensional space which distinctly classifies the target value. We are planning to use linear and non-linear kernels.

DECISION TREES: Decision tree builds classification model in the form of a tree structure. It breaks down a dataset into smaller and smaller subsets based on the conditions. The result is a tree with decision nodes and leaf nodes. We are planning to use Information Gain and Gini Index techniques to select the best attribute using Attribute Selection Measure (ASM).

RANDOM FOREST: It is an ensemble tree-based learning algorithm. The Random Forest Classifier is a set of decision trees from randomly selected subset of training set. We are planning to use the Bagging and Boosting ensemble strategies.

K-NEAREST NEIGHBORS: It is a data categorization approach that is used to estimate the chance that a data point will belong to one of two groups based on the data points closest to it.



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 /opt/conda/lib/python3.9/site-packages (1.5.12)
Requirement already satisfied: python-dateutil in /opt/conda/lib/python3.9/site-packages
(from kaggle) (2.8.2)
Requirement already satisfied: urllib3 in /opt/conda/lib/python3.9/site-packages (from k
aggle) (1.26.8)
Requirement already satisfied: python-slugify in /opt/conda/lib/python3.9/site-packages
(from kaggle) (5.0.2)
Requirement already satisfied: six>=1.10 in /opt/conda/lib/python3.9/site-packages (from
kaggle) (1.16.0)
Requirement already satisfied: tqdm in /opt/conda/lib/python3.9/site-packages (from kagg
le) (4.62.3)
Requirement already satisfied: certifi in /opt/conda/lib/python3.9/site-packages (from k
aggle) (2021.10.8)
Requirement already satisfied: requests in /opt/conda/lib/python3.9/site-packages (from
kaggle) (2.27.1)
Requirement already satisfied: text-unidecode>=1.3 in /opt/conda/lib/python3.9/site-pack
ages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: idna<4,>=2.5 in /opt/conda/lib/python3.9/site-packages (f
rom requests->kaggle) (3.1)
Requirement already satisfied: charset-normalizer~=2.0.0 in /opt/conda/lib/python3.9/sit
e-packages (from requests->kaggle) (2.0.10)
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting
behaviour with the system package manager. It is recommended to use a virtual environmen
t instead: https://pip.pypa.io/warnings/venv
```

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

```
/root/shared/AML/I526_AML_Student/Assignments/Unit-Project-Home-Credit-Default-Risk/HCDR
_Phase_1_baseline_submission
```

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

```
mkdir: cannot create directory ‘/root/.kaggle’: File exists
cp: cannot stat '/root/shared/Downloads/kaggle.json': No such file or directory
chmod: cannot access '/root/.kaggle/kaggle.json': No such file or directory
```

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

```
Traceback (most recent call last):
  File "/opt/conda/bin/kaggle", line 5, in <module>
    from kaggle.cli import main
  File "/opt/conda/lib/python3.9/site-packages/kaggle/__init__.py", line 23, in <module>
    api.authenticate()
  File "/opt/conda/lib/python3.9/site-packages/kaggle/api/kaggle_api_extended.py", line
164, in authenticate
    raise IOError('Could not find {}. Make sure it\'s located in'
OSError: Could not find kaggle.json. Make sure it's located in /root/.kaggle. Or use the
environment method.
```

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.

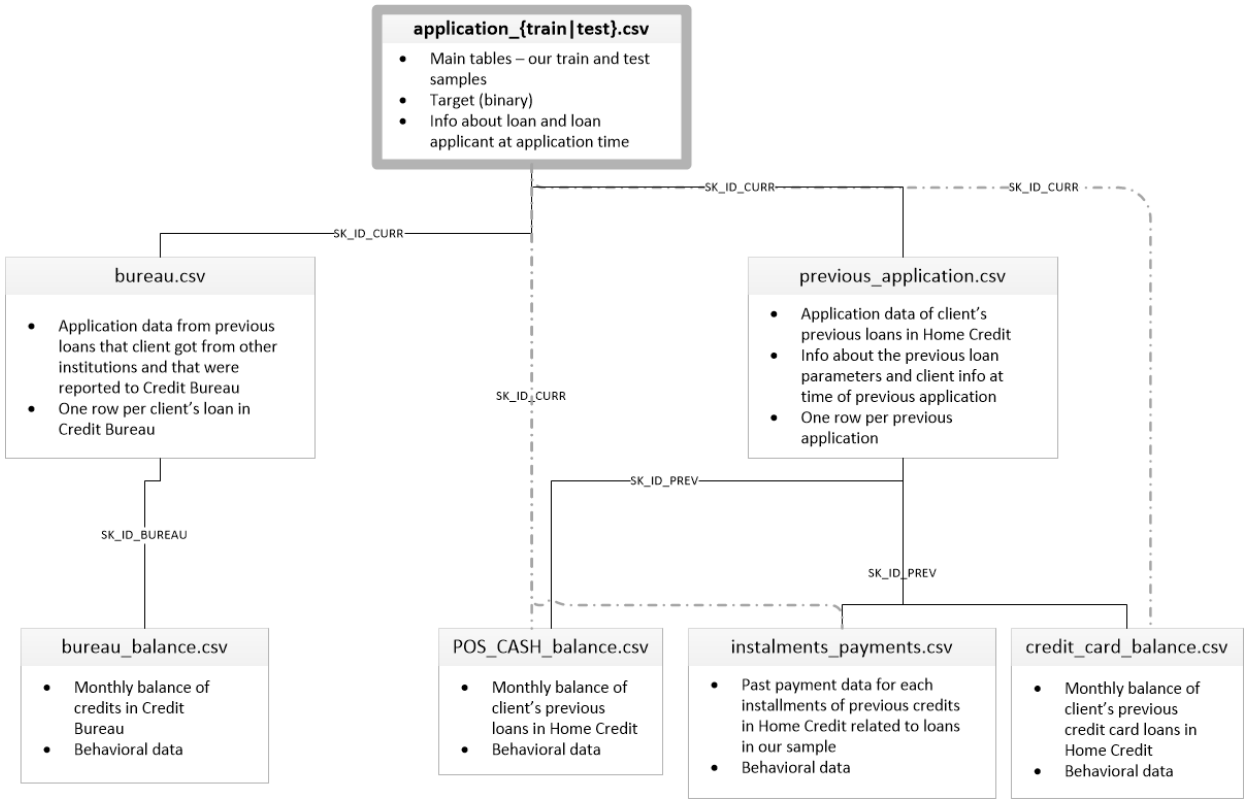
Doing so will ensure that clients capable of repayment are not rejected and that loans are given with a principal, maturity, and repayment calendar that will empower their clients to be successful.

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 [1]:
```
DATA_DIR = "../../../Data/home-credit-default-risk"   #same level as course repo in the
#DATA_DIR = os.path.join('./ddddd/')
!mkdir $DATA_DIR
```

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

In [2]:
```
!ls -l $DATA_DIR
```

```
total 2621364
-rw-r--r-- 1 root root     37383 Apr 13 02:34 HomeCredit_columns_description.csv
-rw-r--r-- 1 root root 392703158 Apr 13 02:39 POS_CASH_balance.csv
-rw-r--r-- 1 root root  26567651 Apr 13 02:35 application_test.csv
-rw-r--r-- 1 root root 166133370 Apr 13 02:37 application_train.csv
-rw-r--r-- 1 root root 170016717 Apr 13 02:37 bureau.csv
-rw-r--r-- 1 root root 375592889 Apr 13 02:39 bureau_balance.csv
-rw-r--r-- 1 root root 424582605 Apr 13 02:39 credit_card_balance.csv
-rw-r--r-- 1 root root 723118349 Apr 13 02:39 installments_payments.csv
```

```
-rw-r--r-- 1 root root 404973293 Apr 13 02:39 previous_application.csv
-rw-r--r-- 1 root root    536202 Apr 13 02:34 sample_submission.csv
```

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

```
Traceback (most recent call last):
  File "/opt/conda/bin/kaggle", line 5, in <module>
    from kaggle.cli import main
  File "/opt/conda/lib/python3.9/site-packages/kaggle/__init__.py", line 23, in <module>
    api.authenticate()
  File "/opt/conda/lib/python3.9/site-packages/kaggle/api/kaggle_api_extended.py", line
164, in authenticate
    raise IOError('Could not find {}. Make sure it\'s located in'
OSError: Could not find kaggle.json. Make sure it's located in /root/.kaggle. Or use the
environment method.
```

Imports

In [2]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import os
import json
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.pipeline import make_pipeline
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
from scipy import stats
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.linear_model import SGDClassifier
from sklearn.ensemble import RandomForestClassifier
import warnings
warnings.filterwarnings('ignore')
```

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

```
zip_ref.extractall('datasets')
zip_ref.close()
zip_ref = zipfile.ZipFile('bureau.csv.zip', 'r')
zip_ref.extractall('datasets')
zip_ref.close()
zip_ref = zipfile.ZipFile('credit_card_balance.csv.zip', 'r')
zip_ref.extractall('datasets')
zip_ref.close()
zip_ref = zipfile.ZipFile('installments_payments.csv.zip', 'r')
zip_ref.extractall('datasets')
zip_ref.close()
zip_ref = zipfile.ZipFile('POS_CASH_balance.csv.zip', 'r')
zip_ref.extractall('datasets')
zip_ref.close()
zip_ref = zipfile.ZipFile('previous_application.csv.zip', 'r')
zip_ref.extractall('datasets')
zip_ref.close()
```

Data files overview

Data Dictionary

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

`HomeCredit_columns_description.csv`

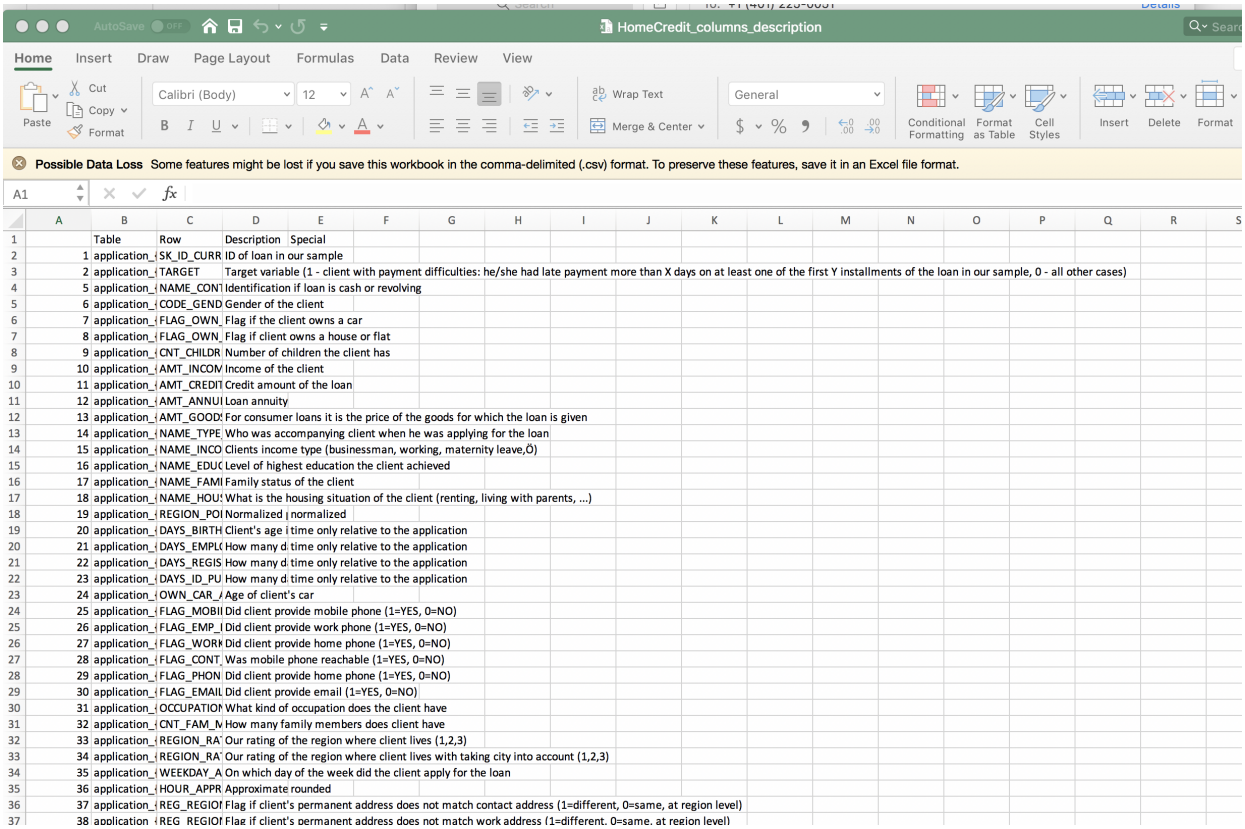


	Table	Row	Description	Special
2	1	application_	SK_ID_CURR	ID of loan in our sample
3	2	application_	TARGET	Target variable (1 - client with payment difficulties: he/she had late payment more than X days on at least one of the first Y installments of the loan in our sample, 0 - all other cases)
4	5	application_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

Application train

In [3]:

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
```

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

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

5 rows × 122 columns

Out[3]: (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 [4]:
```
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_CHI
0	100001	Cash loans	F	N	Y	
1	100005	Cash loans	M	N	Y	
2	100013	Cash loans	M	Y	Y	
3	100028	Cash loans	F	N	Y	
4	100038	Cash loans	M	Y	N	

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.

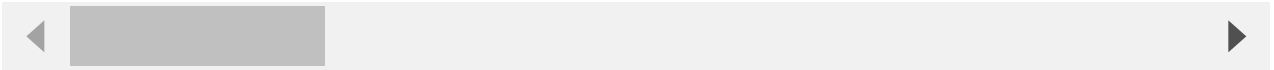
- credit_card_balance: monthly data about previous credit cards clients have had with Home Credit. Each row is one month of a credit card balance, and a single credit card can have many rows.
- **installments_payment:** payment history for previous loans at Home Credit. There is one row for every made payment and one row for every missed payment.

In [5]:
```
%%time
ds_names = ("application_train", "application_test", "bureau","bureau_balance","credit_
            "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
0	100002	1	Cash loans	M	N	Y
1	100003	0	Cash loans	F	N	N
2	100004	0	Revolving loans	M	Y	Y
3	100006	0	Cash loans	F	N	Y
4	100007	0	Cash loans	M	N	Y

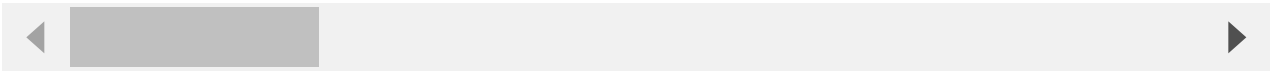
5 rows × 122 columns



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

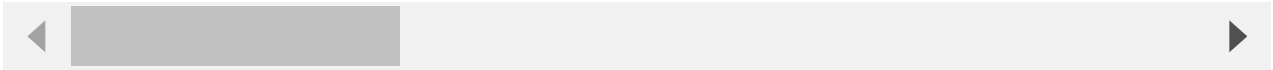
	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHI
0	100001	Cash loans	F	N	Y	
1	100005	Cash loans	M	N	Y	
2	100013	Cash loans	M	Y	Y	
3	100028	Cash loans	F	N	Y	
4	100038	Cash loans	M	Y	N	

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_OVERDU
0	215354	5714462	Closed	currency 1	-497	
1	215354	5714463	Active	currency 1	-208	
2	215354	5714464	Active	currency 1	-203	
3	215354	5714465	Active	currency 1	-203	
4	215354	5714466	Active	currency 1	-629	



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

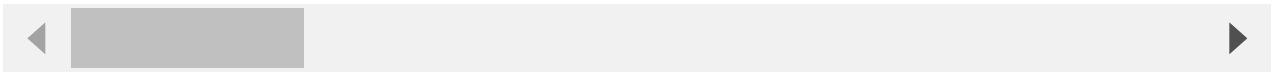
	SK_ID_BUREAU	MONTHS_BALANCE	STATUS
0	5715448	0	C
1	5715448	-1	C
2	5715448	-2	C
3	5715448	-3	C

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

5 rows × 23 columns



```
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_INSTALMI
0	1054186	161674	1.0	6	-11
1	1330831	151639	0.0	34	-21
2	2085231	193053	2.0	1	-
3	2452527	199697	1.0	3	-24
4	2714724	167756	1.0	2	-13

```
previous_application: shape is (1670214, 37)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1670214 entries, 0 to 1670213
Data columns (total 37 columns):
 #   Column                       Non-Null Count    Dtype
---  ------                       --------------    -----
 0   SK_ID_PREV                   1670214 non-null  int64
 1   SK_ID_CURR                   1670214 non-null  int64
 2   NAME_CONTRACT_TYPE           1670214 non-null  object
 3   AMT_ANNUITY                  1297979 non-null  float64
 4   AMT_APPLICATION              1670214 non-null  float64
 5   AMT_CREDIT                   1670213 non-null  float64
 6   AMT_DOWN_PAYMENT             774370 non-null   float64
 7   AMT_GOODS_PRICE              1284699 non-null  float64
 8   WEEKDAY_APPR_PROCESS_START   1670214 non-null  object
 9   HOUR_APPR_PROCESS_START      1670214 non-null  int64
 10  FLAG_LAST_APPL_PER_CONTRACT  1670214 non-null  object
 11  NFLAG_LAST_APPL_IN_DAY       1670214 non-null  int64
 12  RATE_DOWN_PAYMENT            774370 non-null   float64
 13  RATE_INTEREST_PRIMARY        5951 non-null     float64
 14  RATE_INTEREST_PRIVILEGED     5951 non-null     float64
 15  NAME_CASH_LOAN_PURPOSE       1670214 non-null  object
 16  NAME_CONTRACT_STATUS         1670214 non-null  object
 17  DAYS_DECISION                1670214 non-null  int64
 18  NAME_PAYMENT_TYPE            1670214 non-null  object
 19  CODE_REJECT_REASON           1670214 non-null  object
 20  NAME_TYPE_SUITE              849809 non-null   object
 21  NAME_CLIENT_TYPE             1670214 non-null  object
 22  NAME_GOODS_CATEGORY          1670214 non-null  object
 23  NAME_PORTFOLIO               1670214 non-null  object
 24  NAME_PRODUCT_TYPE            1670214 non-null  object
 25  CHANNEL_TYPE                 1670214 non-null  object
 26  SELLERPLACE_AREA             1670214 non-null  int64
 27  NAME_SELLER_INDUSTRY         1670214 non-null  object
 28  CNT_PAYMENT                  1297984 non-null  float64
 29  NAME_YIELD_GROUP             1670214 non-null  object
 30  PRODUCT_COMBINATION          1669868 non-null  object
 31  DAYS_FIRST_DRAWING           997149 non-null   float64
 32  DAYS_FIRST_DUE               997149 non-null   float64
 33  DAYS_LAST_DUE_1ST_VERSION    997149 non-null   float64
 34  DAYS_LAST_DUE                997149 non-null   float64
 35  DAYS_TERMINATION             997149 non-null   float64
 36  NFLAG_INSURED_ON_APPROVAL    997149 non-null   float64
dtypes: float64(15), int64(6), object(16)
memory usage: 471.5+ MB
None
```

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

5 rows × 37 columns

```
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_
0	1803195	182943	-31	48.0	45.0	
1	1715348	367990	-33	36.0	35.0	
2	1784872	397406	-32	12.0	9.0	
3	1903291	269225	-35	48.0	42.0	
4	2341044	334279	-35	36.0	35.0	

```
CPU times: user 20.2 s, sys: 5.9 s, total: 26.1 s
Wall time: 1min 18s
```

In [6]:

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

```
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

Application Train Dataset

In [14]:
```
datasets["application_test"].info()
```

```
<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
```

In [15]:
```
datasets["application_test"].isnull().sum()[datasets["application_test"].isnull().sum()
```

Out[15]:

OWN_CAR_AGE	32312
OCCUPATION_TYPE	15605
EXT_SOURCE_1	20532
EXT_SOURCE_3	8668
APARTMENTS_AVG	23887
BASEMENTAREA_AVG	27641
YEARS_BEGINEXPLUATATION_AVG	22856
YEARS_BUILD_AVG	31818
COMMONAREA_AVG	33495
ELEVATORS_AVG	25189
ENTRANCES_AVG	23579
FLOORSMAX_AVG	23321
FLOORSMIN_AVG	32466
LANDAREA_AVG	28254
LIVINGAPARTMENTS_AVG	32780
LIVINGAREA_AVG	23552
NONLIVINGAPARTMENTS_AVG	33347
NONLIVINGAREA_AVG	26084
APARTMENTS_MODE	23887
BASEMENTAREA_MODE	27641
YEARS_BEGINEXPLUATATION_MODE	22856
YEARS_BUILD_MODE	31818
COMMONAREA_MODE	33495
ELEVATORS_MODE	25189
ENTRANCES_MODE	23579
FLOORSMAX_MODE	23321
FLOORSMIN_MODE	32466
LANDAREA_MODE	28254
LIVINGAPARTMENTS_MODE	32780
LIVINGAREA_MODE	23552
NONLIVINGAPARTMENTS_MODE	33347
NONLIVINGAREA_MODE	26084
APARTMENTS_MEDI	23887
BASEMENTAREA_MEDI	27641
YEARS_BEGINEXPLUATATION_MEDI	22856
YEARS_BUILD_MEDI	31818
COMMONAREA_MEDI	33495
ELEVATORS_MEDI	25189
ENTRANCES_MEDI	23579
FLOORSMAX_MEDI	23321
FLOORSMIN_MEDI	32466
LANDAREA_MEDI	28254
LIVINGAPARTMENTS_MEDI	32780

```
LIVINGAREA_MEDI                23552
NONLIVINGAPARTMENTS_MEDI       33347
NONLIVINGAREA_MEDI             26084
FONDKAPREMONT_MODE             32797
HOUSETYPE_MODE                 23619
TOTALAREA_MODE                 22624
WALLSMATERIAL_MODE             23893
EMERGENCYSTATE_MODE            22209
AMT_REQ_CREDIT_BUREAU_HOUR      6049
AMT_REQ_CREDIT_BUREAU_DAY       6049
AMT_REQ_CREDIT_BUREAU_WEEK      6049
AMT_REQ_CREDIT_BUREAU_MON       6049
AMT_REQ_CREDIT_BUREAU_QRT       6049
AMT_REQ_CREDIT_BUREAU_YEAR      6049
dtype: int64
```

In [16]:

```
datasets["application_test"].describe()
```

Out[16]:

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOOD
count	48744.000000	48744.000000	4.874400e+04	4.874400e+04	48720.000000	4.874
mean	277796.676350	0.397054	1.784318e+05	5.167404e+05	29426.240209	4.626
std	103169.547296	0.709047	1.015226e+05	3.653970e+05	16016.368315	3.367
min	100001.000000	0.000000	2.694150e+04	4.500000e+04	2295.000000	4.500
25%	188557.750000	0.000000	1.125000e+05	2.606400e+05	17973.000000	2.250
50%	277549.000000	0.000000	1.575000e+05	4.500000e+05	26199.000000	3.960
75%	367555.500000	1.000000	2.250000e+05	6.750000e+05	37390.500000	6.300
max	456250.000000	20.000000	4.410000e+06	2.245500e+06	180576.000000	2.245

8 rows × 105 columns

In [17]:

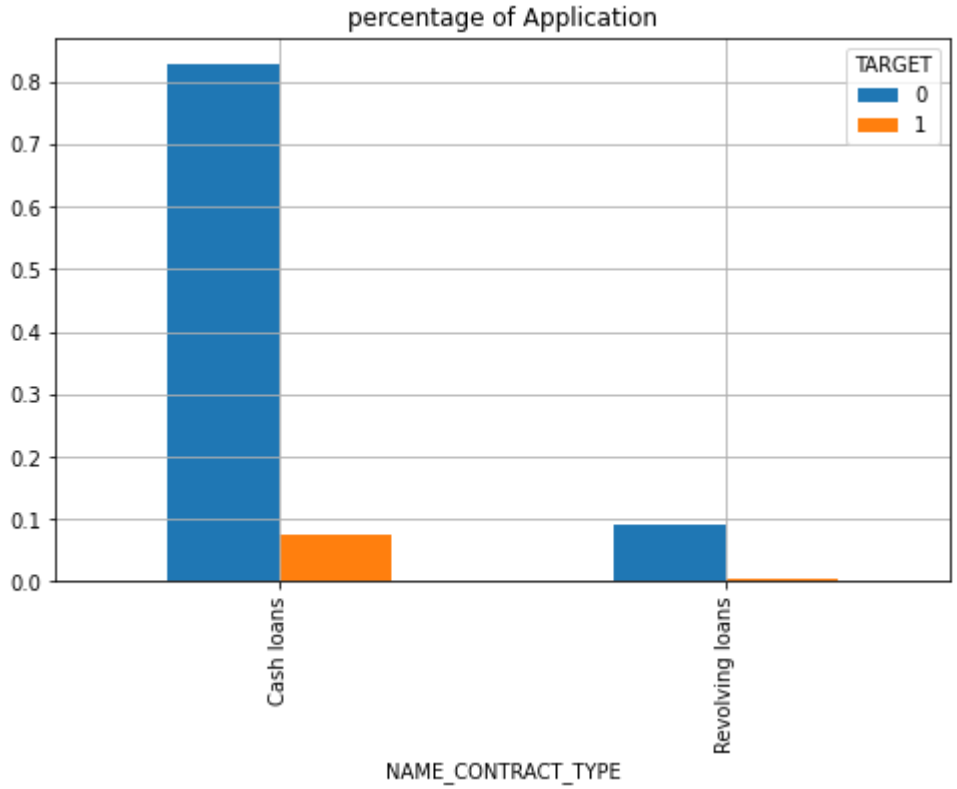
```
print("Number of applicaiton is the training dataset:", datasets["application_train"].s
print("Number of applicaiton turns to be defaulted:",datasets["application_train"][data
print("Overall, Bad Rate:{}%".format(datasets["application_train"][datasets["applicatio
summary = datasets["application_train"].groupby(['NAME_CONTRACT_TYPE', 'TARGET'])['SK_I
summary['perc_app'] = summary['Num_app']/summary['Num_app'].sum()
pd.pivot_table(summary.reset_index(), index = 'NAME_CONTRACT_TYPE', columns = 'TARGET',

summary.drop(summary.columns, axis=1)
```

```
Number of applicaiton is the training dataset: 307511
Number of applicaiton turns to be defaulted: 24825
Overall, Bad Rate:8.072881945686495%
```

Out[17]:

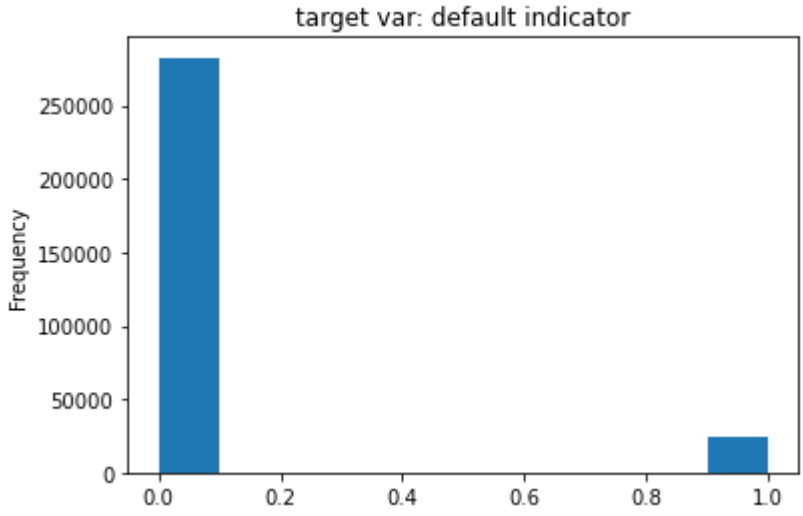
NAME_CONTRACT_TYPE	TARGET
Cash loans	**0**
	1
Revolving loans	**0**

NAME_CONTRACT_TYPE	TARGET
	1



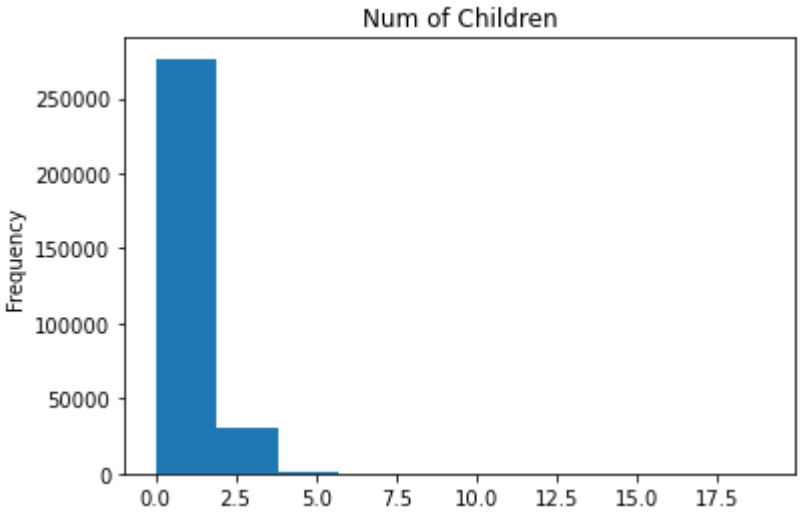
In [18]:
```
datasets["application_train"]['TARGET'].plot.hist(title = 'target var: default indicato
```

Out[18]: <AxesSubplot:title={'center':'target var: default indicator'}, ylabel='Frequency'>



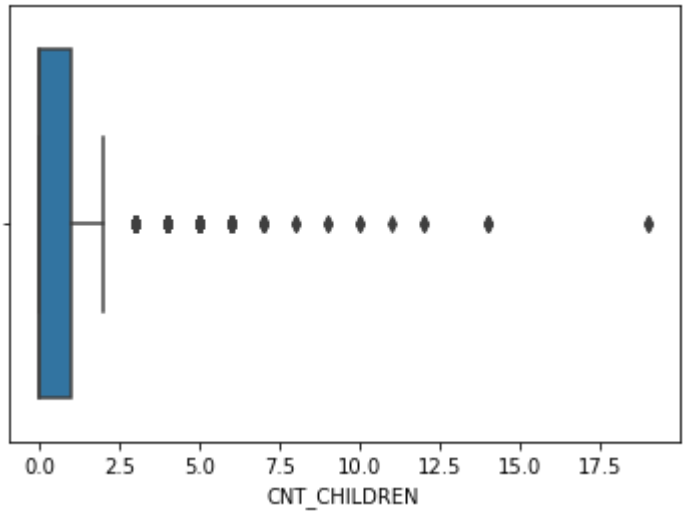
In [19]:
```
datasets["application_train"]['CNT_CHILDREN'].plot.hist(title = 'Num of Children')
```

Out[19]: <AxesSubplot:title={'center':'Num of Children'}, ylabel='Frequency'>



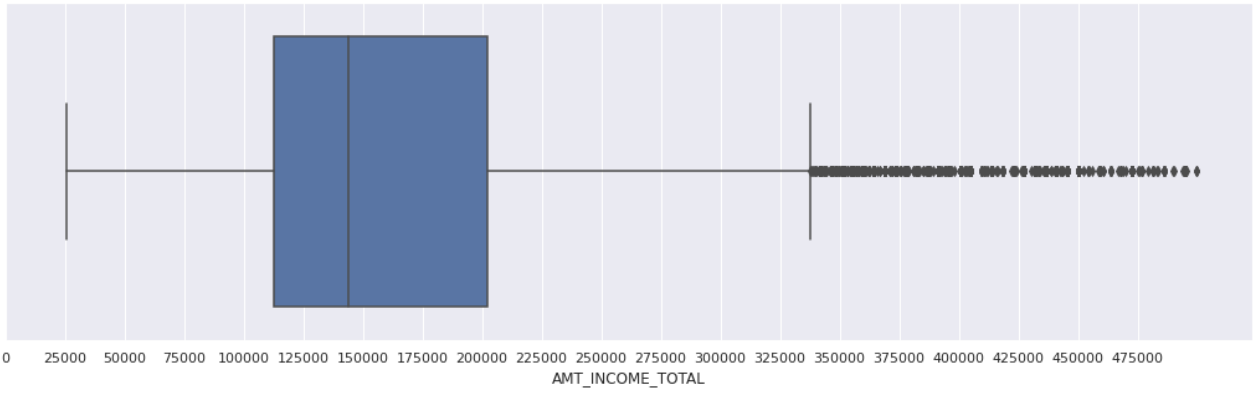
In [20]:
```
sns.boxplot(x=datasets["application_train"]['CNT_CHILDREN'])
```

Out[20]: <AxesSubplot:xlabel='CNT_CHILDREN'>



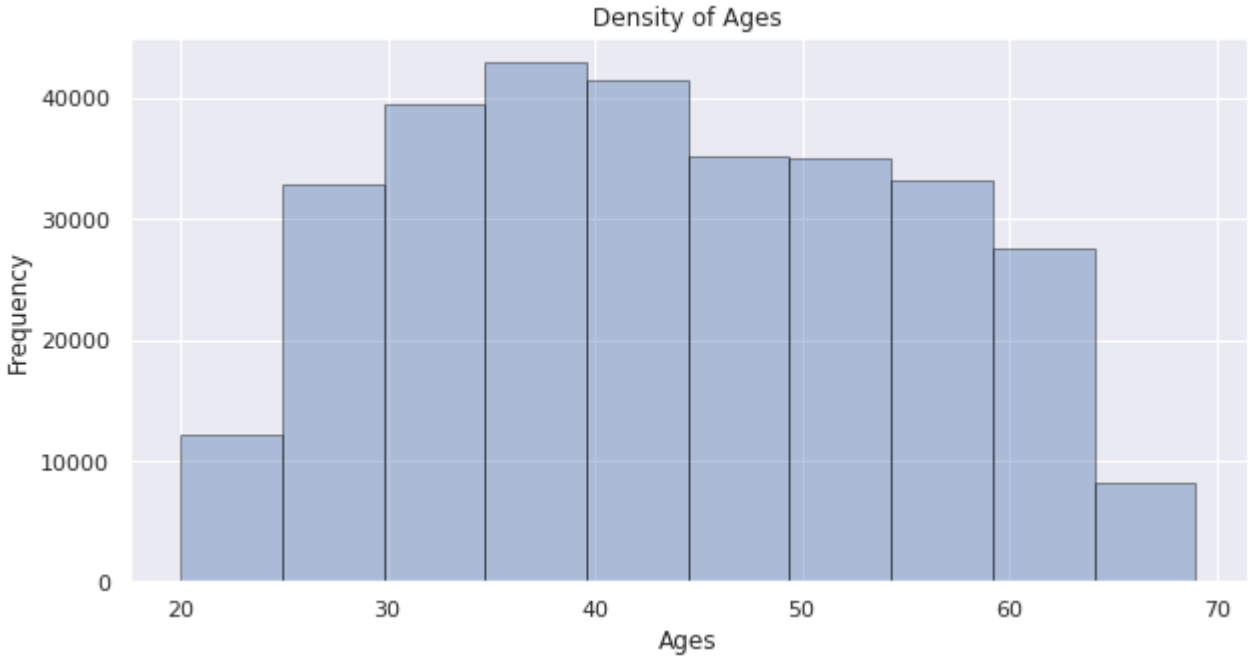
In [21]:
```
sns.set(rc={'figure.figsize':(18,5)})
ax = sns.boxplot(x=datasets["application_train"][datasets["application_train"].AMT_INCO
ax.set_xticks(range(0,500000,25000))
ax
```

Out[21]: <AxesSubplot:xlabel='AMT_INCOME_TOTAL'>



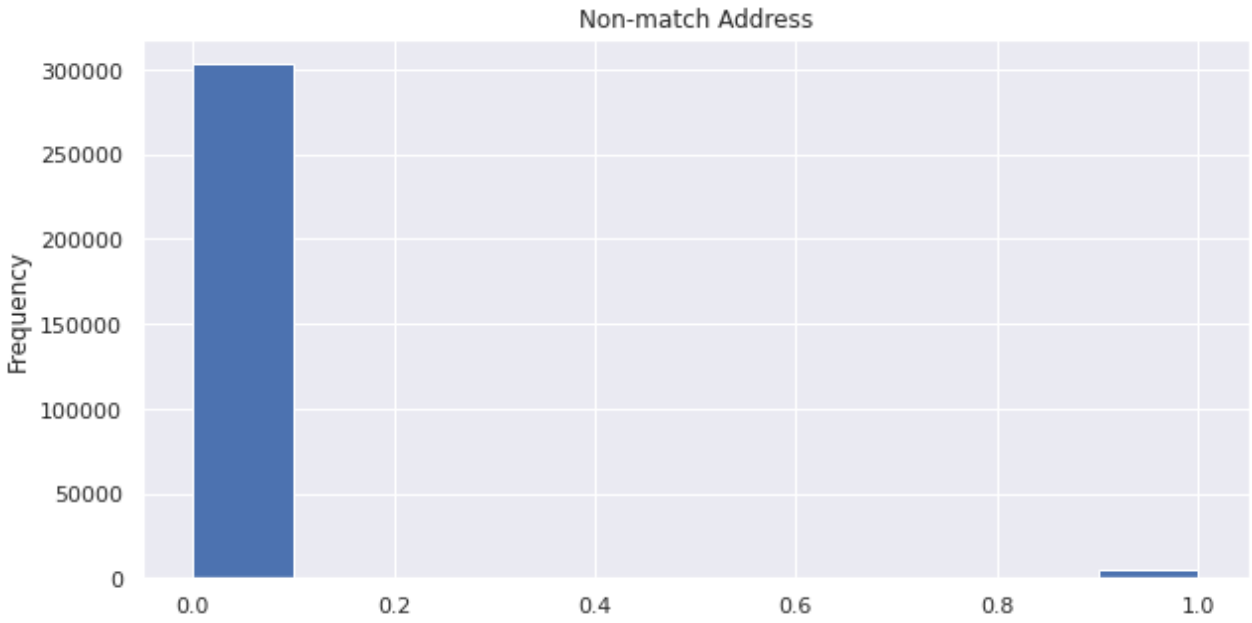
In [22]:
```
ages = [int(-x/365) for x in datasets["application_train"].DAYS_BIRTH]
sns.set(rc={'figure.figsize':(10,5)})
sns.distplot(ages, hist=True, kde=False,
             bins=10,hist_kws={'edgecolor':'black'})
plt.title('Density of Ages')
plt.xlabel('Ages')
plt.ylabel('Frequency')
```

Out[22]:
```
Text(0, 0.5, 'Frequency')
```



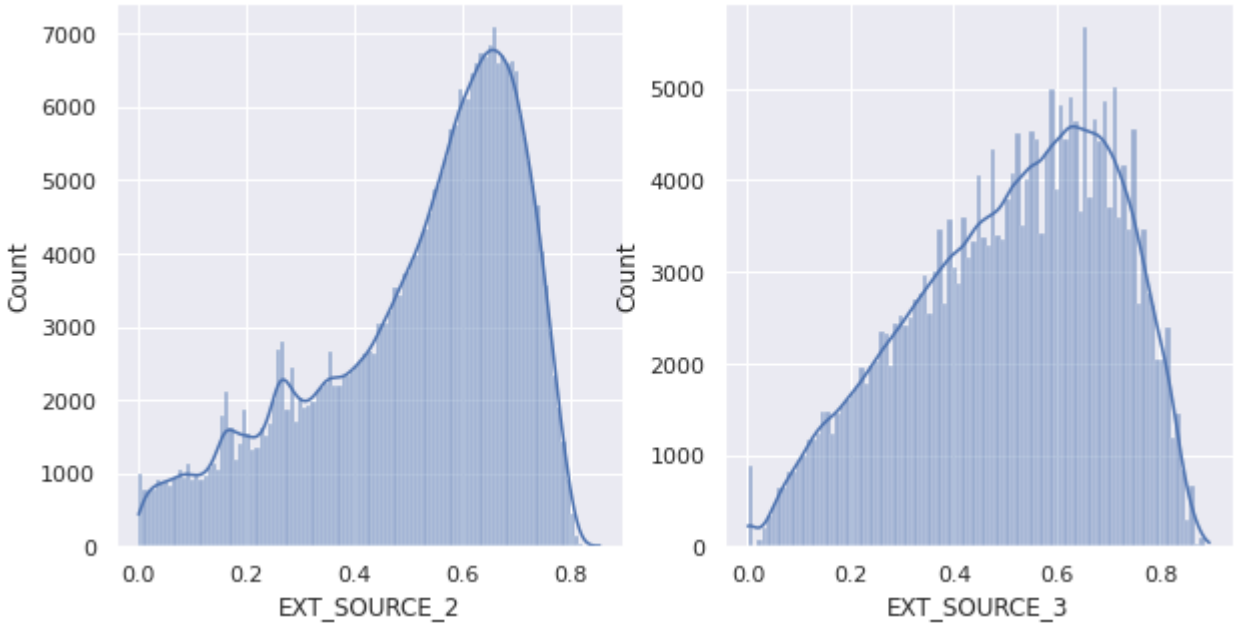
In [23]:
```
datasets["application_train"]['REG_REGION_NOT_LIVE_REGION'].plot.hist(title = 'Non-matc
```

Out[23]:
```
<AxesSubplot:title={'center':'Non-match Address'}, ylabel='Frequency'>
```



In [24]:
```
fig, ax = plt.subplots(1,2)
sns.histplot(data=datasets["application_train"], x="EXT_SOURCE_2",kde=True,ax=ax[0])
```

```
sns.histplot(data=datasets["application_train"], x="EXT_SOURCE_3",kde=True,ax=ax[1])
fig.show()
```

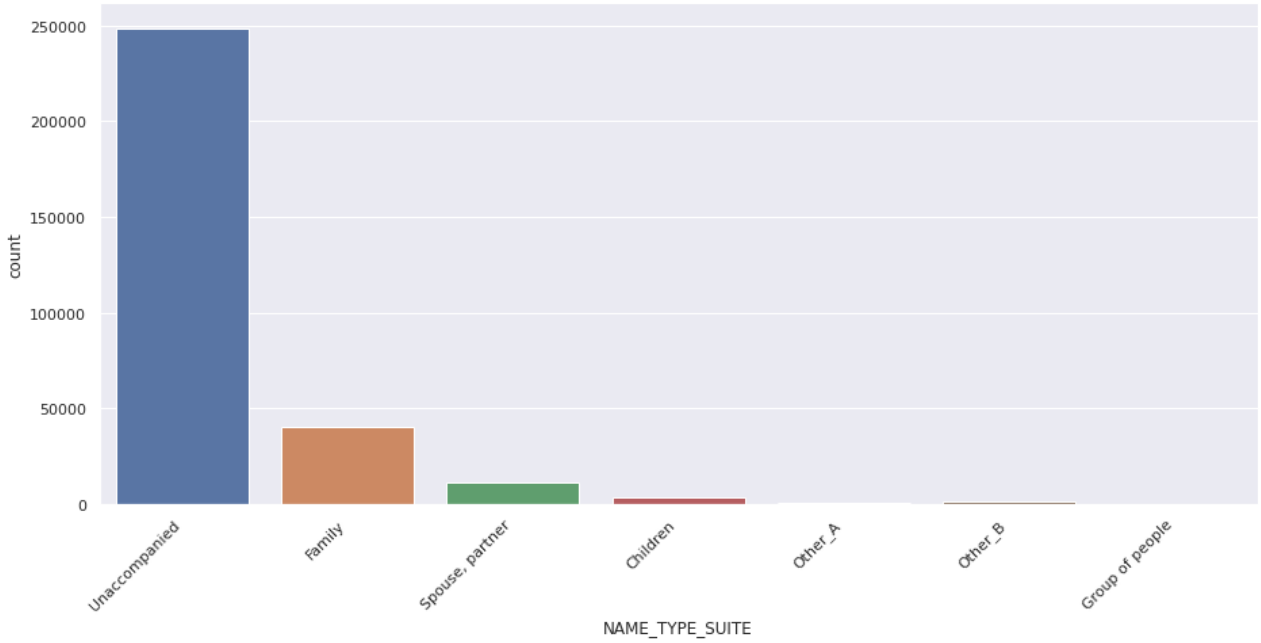


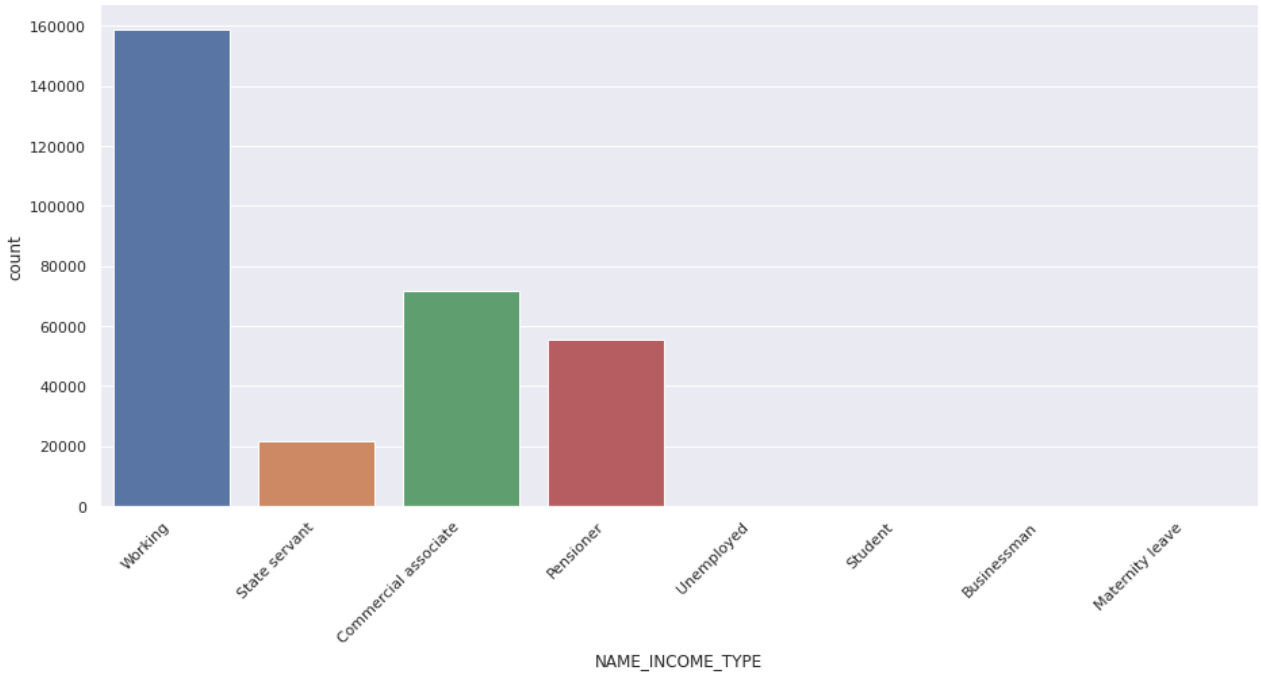
In [25]:

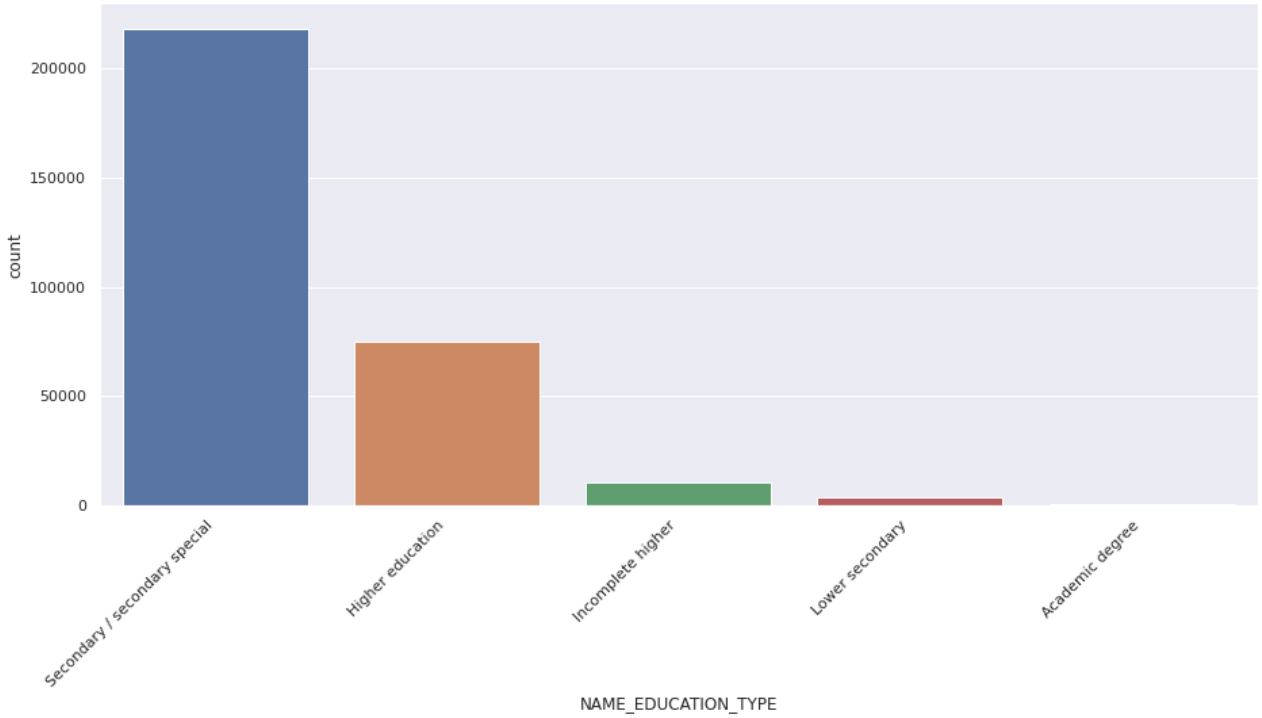
```
for col in datasets["application_train"].columns:
    if datasets["application_train"][col].dtype == 'object':
        print("object column %s have %s unique values"%(str(col),datasets["application_
        if datasets["application_train"][col].nunique() <= 3:
            print(datasets["application_train"][col].value_counts())
            print('-------------')
        else:
            ax = sns.catplot(x=col, kind="count", data=datasets["application_train"],he
            for axes in ax.axes.flat:
                axes.set_xticklabels(axes.get_xticklabels(),rotation = 45,horizontalali
```

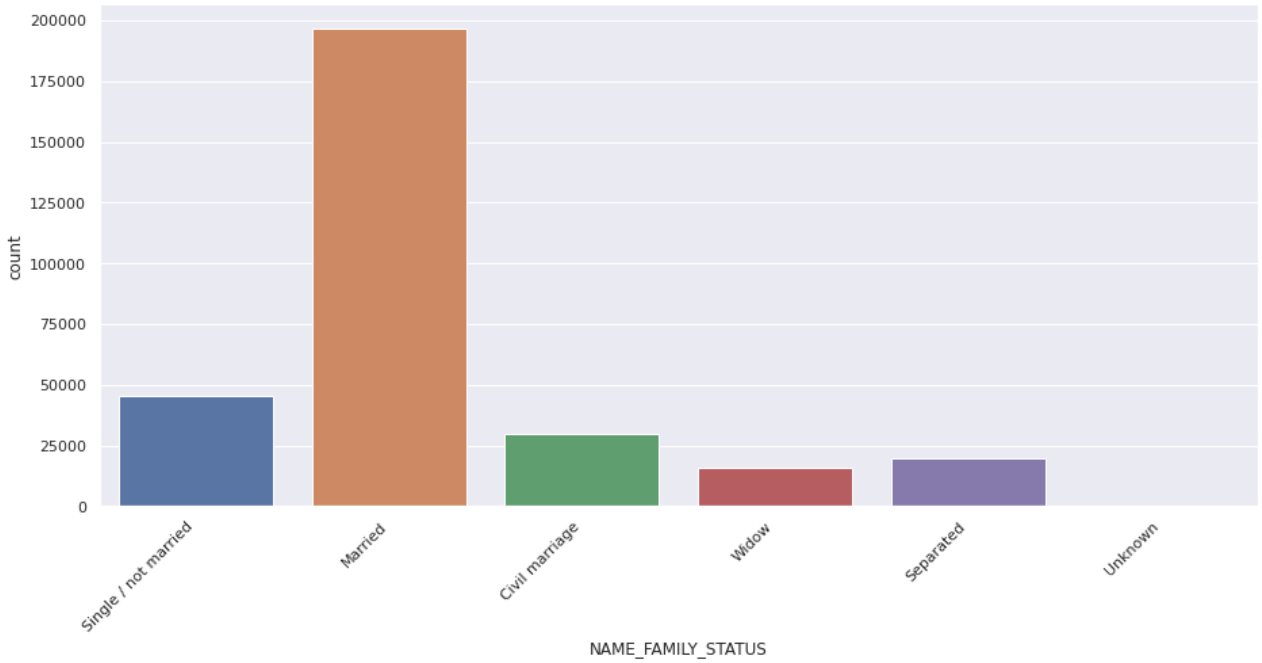
```
object column NAME_CONTRACT_TYPE have 2 unique values
Cash loans         278232
Revolving loans     29279
Name: NAME_CONTRACT_TYPE, dtype: int64
-------------
object column CODE_GENDER have 3 unique values
F      202448
M      105059
XNA         4
Name: CODE_GENDER, dtype: int64
-------------
object column FLAG_OWN_CAR have 2 unique values
N    202924
Y    104587
Name: FLAG_OWN_CAR, dtype: int64
-------------
object column FLAG_OWN_REALTY have 2 unique values
Y    213312
N     94199
Name: FLAG_OWN_REALTY, dtype: int64
-------------
object column NAME_TYPE_SUITE have 7 unique values
object column NAME_INCOME_TYPE have 8 unique values
object column NAME_EDUCATION_TYPE have 5 unique values
object column NAME_FAMILY_STATUS have 6 unique values
object column NAME_HOUSING_TYPE have 6 unique values
```

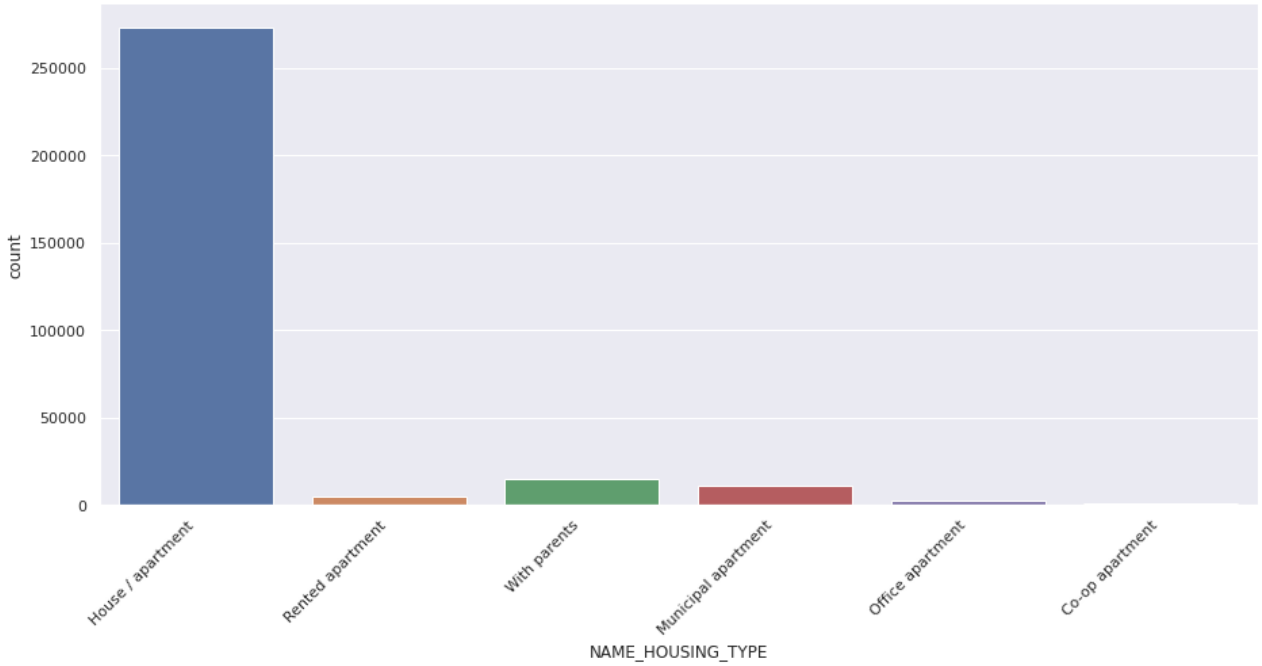
```
object column OCCUPATION_TYPE have 18 unique values
object column WEEKDAY_APPR_PROCESS_START have 7 unique values
object column ORGANIZATION_TYPE have 58 unique values
object column FONDKAPREMONT_MODE have 4 unique values
object column HOUSETYPE_MODE have 3 unique values
block of flats      150503
specific housing      1499
terraced house        1212
Name: HOUSETYPE_MODE, dtype: int64
-------------
object column WALLSMATERIAL_MODE have 7 unique values
object column EMERGENCYSTATE_MODE have 2 unique values
No     159428
Yes      2328
Name: EMERGENCYSTATE_MODE, dtype: int64
-------------
```

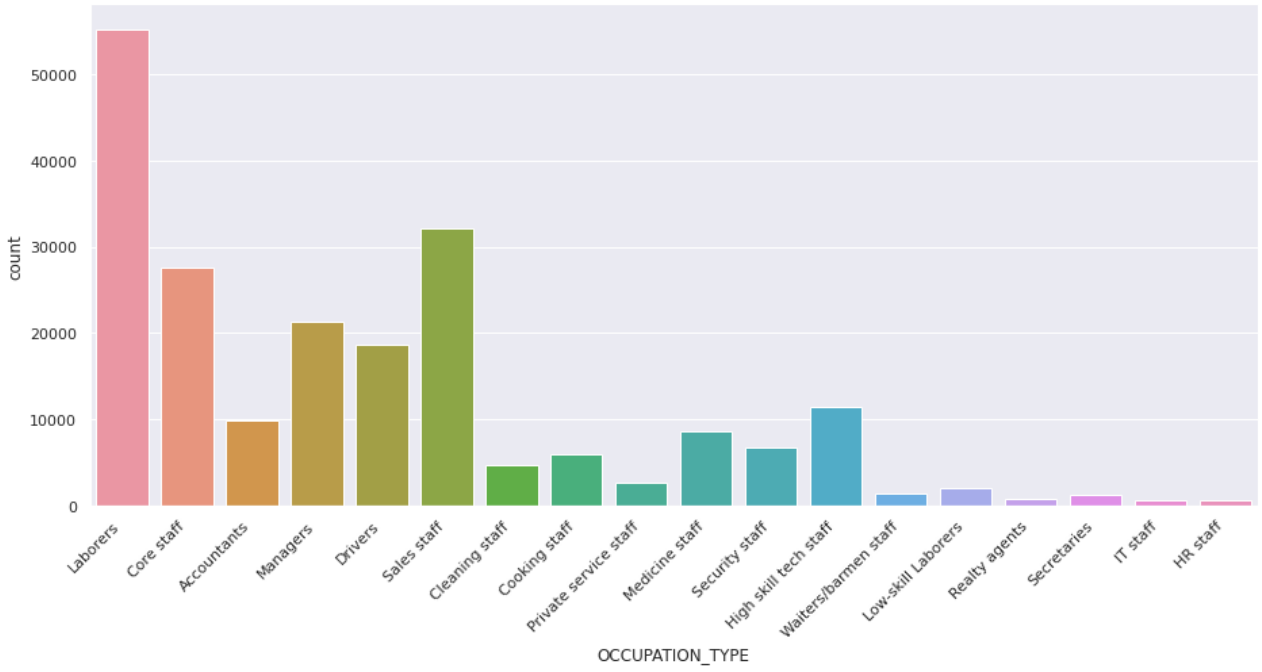


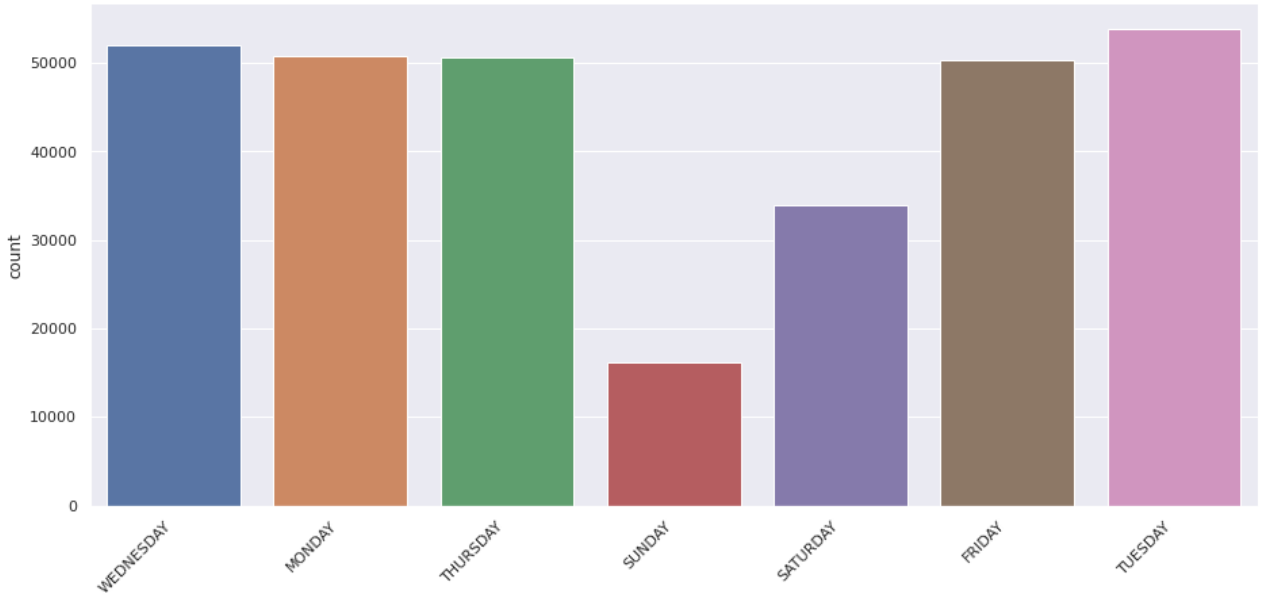


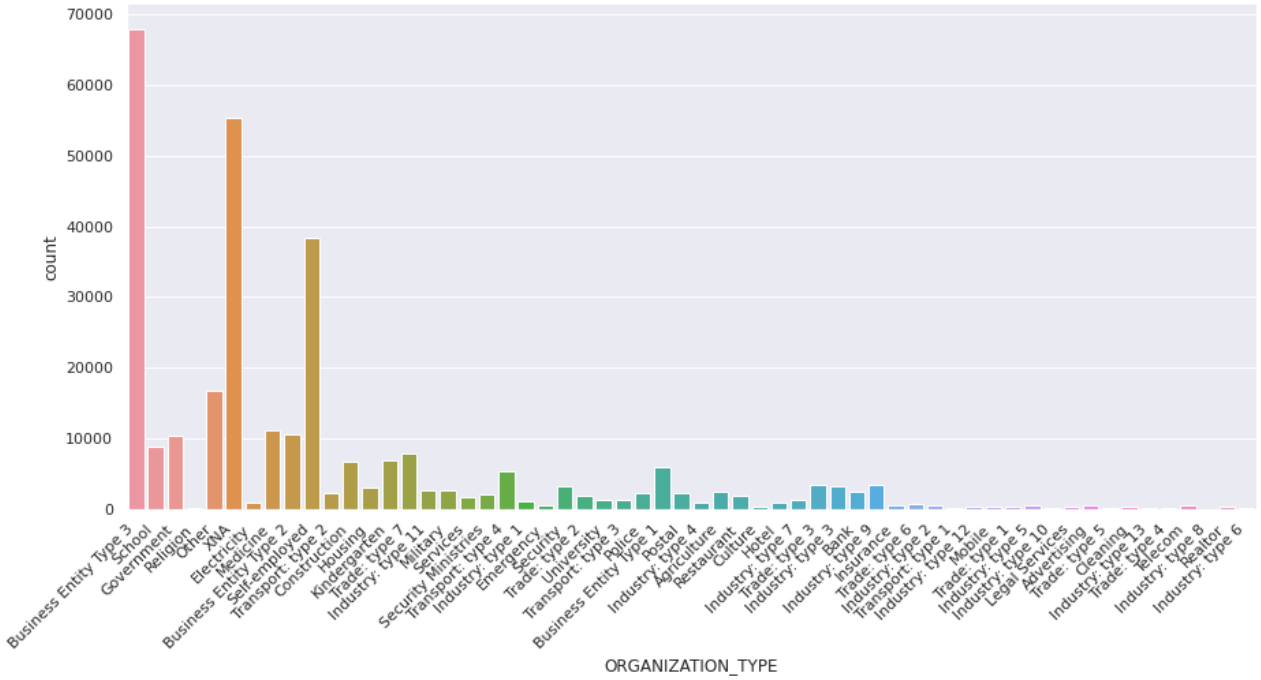

200000
150000
100000
50000
0
count
Secondary / secondary special
Higher education
Incomplete higher
Lower secondary
Academic degree
NAME_EDUCATION_TYPE

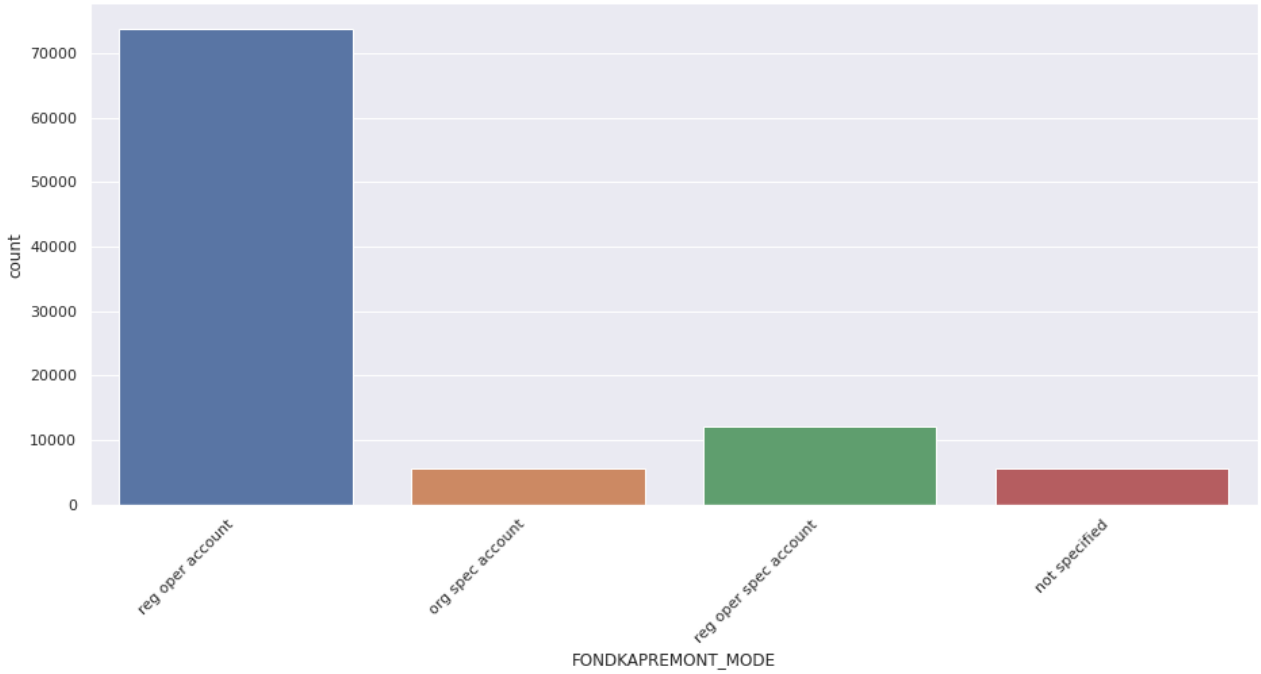

200000
175000
150000
125000
100000
75000
50000
25000
0
count
Single / not married
Married
Civil marriage
Widow
Separated
Unknown
NAME_FAMILY_STATUS

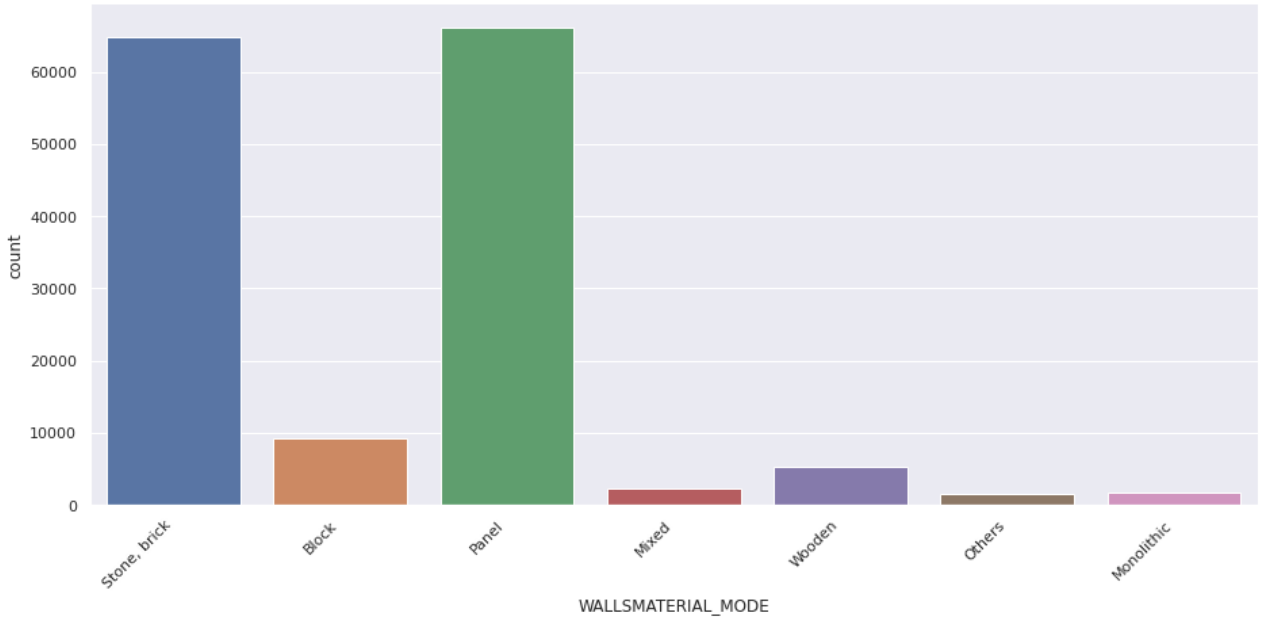

250000
200000
150000
100000
50000
0
count
House / apartment
Rented apartment
With parents
Municipal apartment
Office apartment
Co-op apartment
NAME_HOUSING_TYPE


50000
40000
30000
20000
10000
0
count
Laborers
Core staff
Accountants
Managers
Drivers
Sales staff
Cleaning staff
Cooking staff
Private service staff
Medicine staff
Security staff
High skill tech staff
Waiters/barmen staff
Low-skill Laborers
Realty agents
Secretaries
IT staff
HR staff
OCCUPATION_TYPE


count
50000
40000
30000
20000
10000
0
WEDNESDAY
MONDAY
THURSDAY
SUNDAY
SATURDAY
FRIDAY
TUESDAY
WEEKDAY_APPR_PROCESS_START


count
70000
60000
50000
40000
30000
20000
10000
0
Business Entity Type 3
School
Government
Religion
Other
XNA
Electricity
Medicine
Business Entity Type 2
Self-employed
Transport: type 2
Construction
Housing
Kindergarten
Trade: type 7
Industry: type 11
Military
Services
Security Ministries
Transport: type 4
Industry: type 1
Emergency
Security
Trade: type 2
University
Transport: type 3
Police
Business Entity Type 1
Industry: type 4
Postal
Agriculture
Restaurant
Culture
Hotel
Industry: type 7
Trade: type 3
Industry: type 3
Bank
Industry: type 9
Insurance
Trade: type 6
Industry: type 2
Transport: type 1
Industry: type 12
Mobile
Trade: type 1
Industry: type 5
Industry: type 10
Legal Services
Advertising
Trade: type 5
Cleaning
Industry: type 13
Trade: type 4
Telecom
Industry: type 8
Realtor
Industry: type 6
ORGANIZATION_TYPE





Analysis of the above graph

Occupation type

- The most common Occupation among the applicants is the Laborers, followed by Sales Staff and Core Staff.
- If we look at the proportion of Defaulters, we observe that the people with low-level Occupations such as
  1. Low-skill Laborers,
  2. Drivers,
  3. Waiters, etc.

  tend to have a higher Percentage of Default Rate than high-level Occupations. ### **Target variable**

- From distribution of the Target Variable in the Training Dataset it can be seen that there are only 8% (24.8k) Defaulters, and 91.9% (282.6k) Non-Defaulters in the train dataset.
- This shows that the Positive class is a minority class in our dataset.
- It also implies that it is an Imbalanced Dataset, and we need to come up with adequate ways to handle it. ### **Gender**
- It can be seen from the plot that more female clients have applied for loan when compared to the male applicants.

Education Type

- Majority of the applicants are from Secondary/secondary special followed by higher education ### **Region of House**
- Majority of the people have a region rating of 2.

In [26]:
```
datasets["application_train"].describe(include='all') #Look at all categorical and nume
```

Out[26]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_
count	307511.000000	307511.000000	307511	307511	307511	
unique	NaN	NaN	2	3	2	
top	NaN	NaN	Cash loans	F	N	
freq	NaN	NaN	278232	202448	202924	
mean	278180.518577	0.080729	NaN	NaN	NaN	
std	102790.175348	0.272419	NaN	NaN	NaN	
min	100002.000000	0.000000	NaN	NaN	NaN	
25%	189145.500000	0.000000	NaN	NaN	NaN	
50%	278202.000000	0.000000	NaN	NaN	NaN	
75%	367142.500000	0.000000	NaN	NaN	NaN	
max	456255.000000	1.000000	NaN	NaN	NaN	

11 rows × 122 columns

Application Test Dataset

In [27]:
```
datasets["application_test"].describe() #numerical only features
```

Out[27]:

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOOD
count	48744.000000	48744.000000	4.874400e+04	4.874400e+04	48720.000000	4.874
mean	277796.676350	0.397054	1.784318e+05	5.167404e+05	29426.240209	4.626
std	103169.547296	0.709047	1.015226e+05	3.653970e+05	16016.368315	3.367

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOOD
min	100001.000000	0.000000	2.694150e+04	4.500000e+04	2295.000000	4.500(
25%	188557.750000	0.000000	1.125000e+05	2.606400e+05	17973.000000	2.250(
50%	277549.000000	0.000000	1.575000e+05	4.500000e+05	26199.000000	3.960(
75%	367555.500000	1.000000	2.250000e+05	6.750000e+05	37390.500000	6.300(
max	456250.000000	20.000000	4.410000e+06	2.245500e+06	180576.000000	2.245

8 rows × 105 columns

Bureau Dataset

In [28]:
```
datasets['bureau'].info()
datasets['bureau'].describe()
```

```
<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
```

Out[28]:

	SK_ID_CURR	SK_ID_BUREAU	DAYS_CREDIT	CREDIT_DAY_OVERDUE	DAYS_CREDIT_ENDDATE	DA
count	1.716428e+06	1.716428e+06	1.716428e+06	1.716428e+06	1.610875e+06	
mean	2.782149e+05	5.924434e+06	-1.142108e+03	8.181666e-01	5.105174e+02	
std	1.029386e+05	5.322657e+05	7.951649e+02	3.654443e+01	4.994220e+03	
min	1.000010e+05	5.000000e+06	-2.922000e+03	0.000000e+00	-4.206000e+04	
25%	1.888668e+05	5.463954e+06	-1.666000e+03	0.000000e+00	-1.138000e+03	
50%	2.780550e+05	5.926304e+06	-9.870000e+02	0.000000e+00	-3.300000e+02	
75%	3.674260e+05	6.385681e+06	-4.740000e+02	0.000000e+00	4.740000e+02	
max	4.562550e+05	6.843457e+06	0.000000e+00	2.792000e+03	3.119900e+04	

Bureau balance

In [29]:
```
datasets['bureau_balance'].info()
datasets['bureau_balance'].describe()
```

```
<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
```

Out[29]:

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

Credit card balance

In [30]:
```
datasets['credit_card_balance'].info()
datasets['credit_card_balance'].describe()
```

```
<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
```

Out[30]:

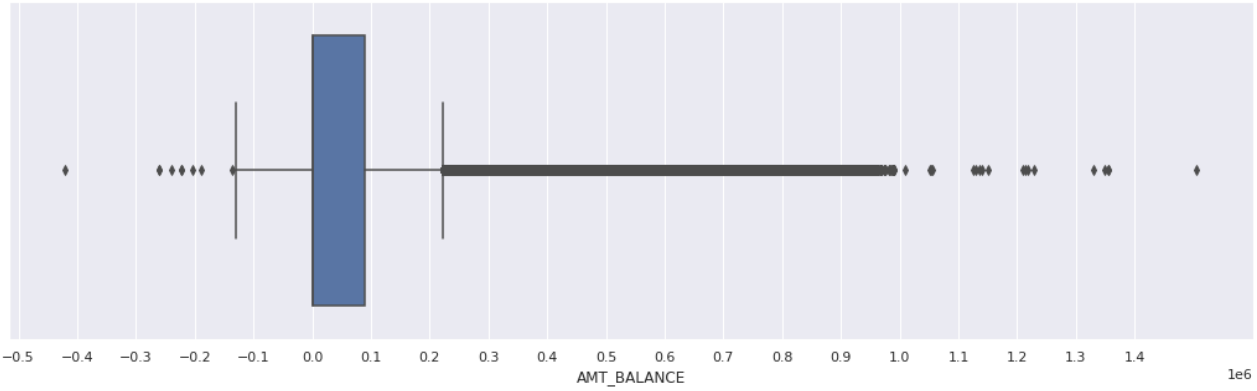
	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	AMT_CREDIT_LIMIT_ACTUAL	A
count	3.840312e+06	3.840312e+06	3.840312e+06	3.840312e+06	3.840312e+06	
mean	1.904504e+06	2.783242e+05	-3.452192e+01	5.830016e+04	1.538080e+05	
std	5.364695e+05	1.027045e+05	2.666775e+01	1.063070e+05	1.651457e+05	
min	1.000018e+06	1.000060e+05	-9.600000e+01	-4.202502e+05	0.000000e+00	
25%	1.434385e+06	1.895170e+05	-5.500000e+01	0.000000e+00	4.500000e+04	
50%	1.897122e+06	2.783960e+05	-2.800000e+01	0.000000e+00	1.125000e+05	
75%	2.369328e+06	3.675800e+05	-1.100000e+01	8.904669e+04	1.800000e+05	
max	2.843496e+06	4.562500e+05	-1.000000e+00	1.505902e+06	1.350000e+06	

8 rows × 22 columns

In [31]:

```
sns.set(rc={'figure.figsize':(18,5)})
ax = sns.boxplot(x=datasets['credit_card_balance']['AMT_BALANCE'])
ax.set_xticks([x/10*1e6 for x in range(-5,15,1)])
ax
```

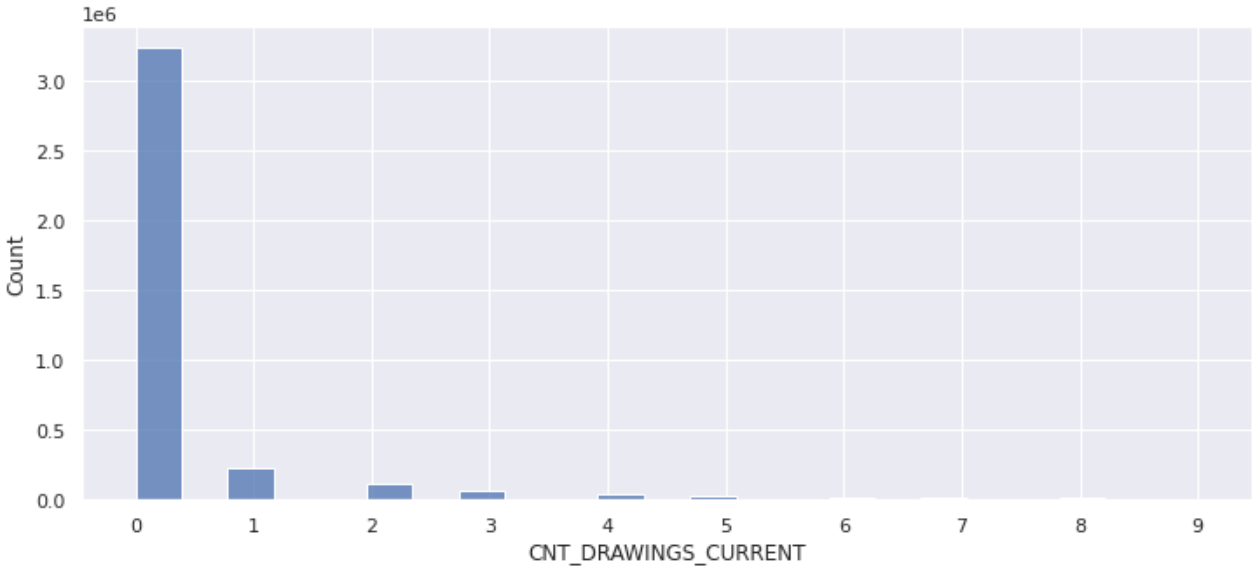
Out[31]: <AxesSubplot:xlabel='AMT_BALANCE'>



In [32]:

```
sns.set(rc={'figure.figsize':(12,5)})
ax = sns.histplot(data=datasets['credit_card_balance'][datasets['credit_card_balance'].
ax.set_xticks(range(0,10,1))
ax
```

Out[32]: <AxesSubplot:xlabel='CNT_DRAWINGS_CURRENT', ylabel='Count'>



installment payments

In [33]:
```
datasets['installments_payments'].info()
datasets['installments_payments'].describe()
```

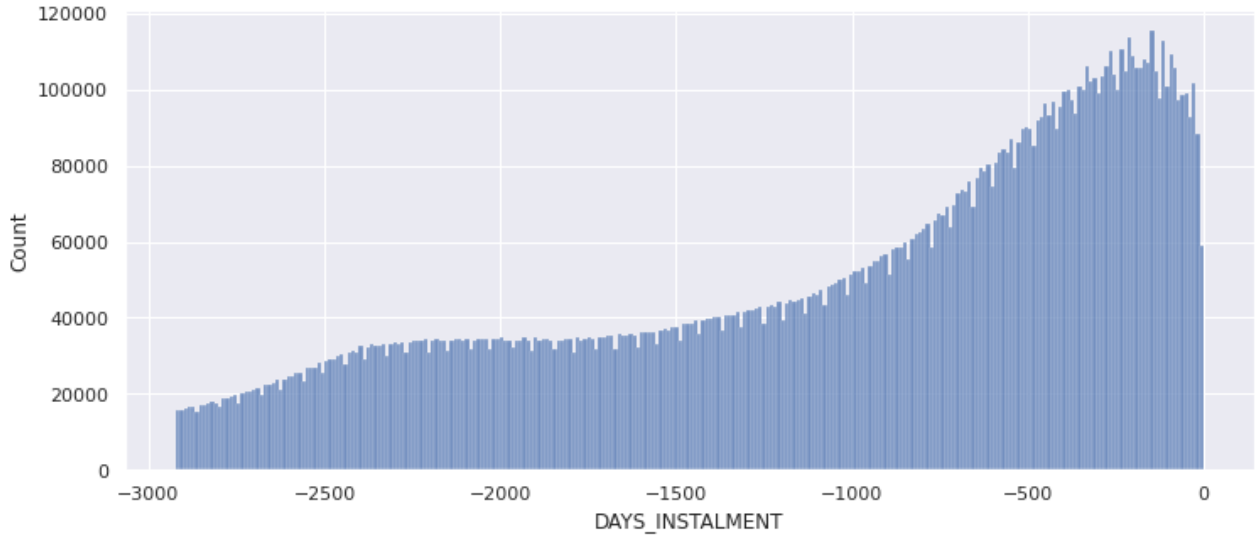
```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13605401 entries, 0 to 13605400
Data columns (total 8 columns):
 #   Column                  Dtype
---  ------                  -----
 0   SK_ID_PREV              int64
 1   SK_ID_CURR              int64
 2   NUM_INSTALMENT_VERSION  float64
 3   NUM_INSTALMENT_NUMBER   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
```

Out[33]:

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER	DAYS_IN
count	1.360540e+07	1.360540e+07	1.360540e+07	1.360540e+07	1.3
mean	1.903365e+06	2.784449e+05	8.566373e-01	1.887090e+01	-1.0
std	5.362029e+05	1.027183e+05	1.035216e+00	2.666407e+01	8.0
min	1.000001e+06	1.000010e+05	0.000000e+00	1.000000e+00	-2.9
25%	1.434191e+06	1.896390e+05	0.000000e+00	4.000000e+00	-1.6
50%	1.896520e+06	2.786850e+05	1.000000e+00	8.000000e+00	-8.1
75%	2.369094e+06	3.675300e+05	1.000000e+00	1.900000e+01	-3.6
max	2.843499e+06	4.562550e+05	1.780000e+02	2.770000e+02	-1.0

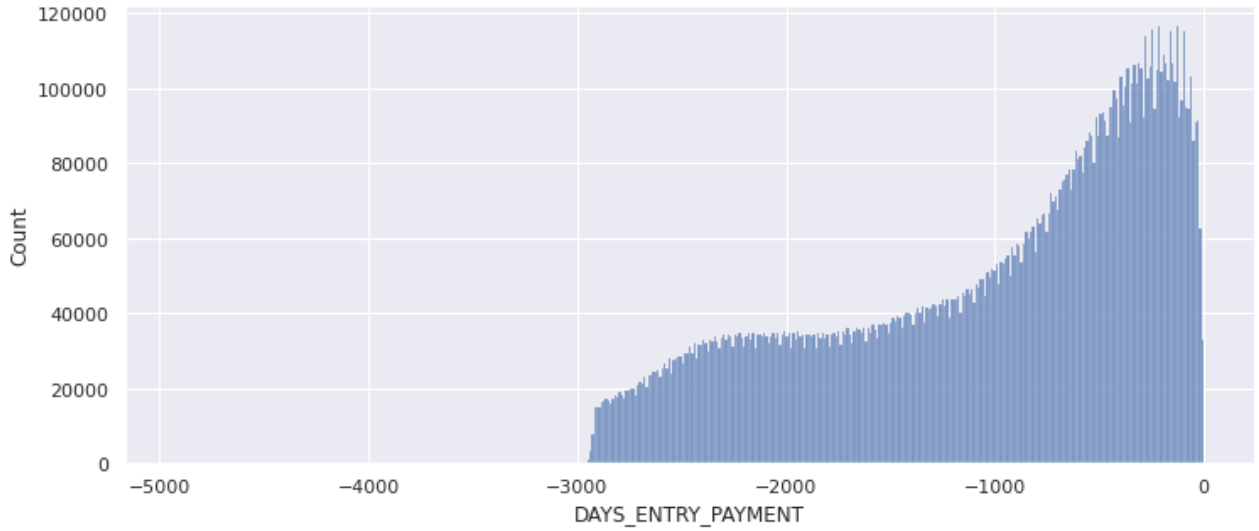
In [34]:
```
sns.histplot(data=datasets['installments_payments'], x="DAYS_INSTALMENT")
```

Out[34]: <AxesSubplot:xlabel='DAYS_INSTALMENT', ylabel='Count'>



In [35]:
```
sns.histplot(data=datasets['installments_payments'], x="DAYS_ENTRY_PAYMENT")
```

Out[35]: <AxesSubplot:xlabel='DAYS_ENTRY_PAYMENT', ylabel='Count'>



previous applications

In [36]:
```
datasets['previous_application'].info()
datasets['previous_application'].describe()
```

```
<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
```

Out[36]:

	SK_ID_PREV	SK_ID_CURR	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAY
count	1.670214e+06	1.670214e+06	1.297979e+06	1.670214e+06	1.670213e+06	7.74370
mean	1.923089e+06	2.783572e+05	1.595512e+04	1.752339e+05	1.961140e+05	6.69740
std	5.325980e+05	1.028148e+05	1.478214e+04	2.927798e+05	3.185746e+05	2.09215
min	1.000001e+06	1.000010e+05	0.000000e+00	0.000000e+00	0.000000e+00	-9.0000(
25%	1.461857e+06	1.893290e+05	6.321780e+03	1.872000e+04	2.416050e+04	0.00000
50%	1.923110e+06	2.787145e+05	1.125000e+04	7.104600e+04	8.054100e+04	1.63800
75%	2.384280e+06	3.675140e+05	2.065842e+04	1.803600e+05	2.164185e+05	7.74000
max	2.845382e+06	4.562550e+05	4.180581e+05	6.905160e+06	6.905160e+06	3.06004

8 rows × 21 columns

Pos cash balance

In [37]:

```
datasets['POS_CASH_balance'].info()
datasets['POS_CASH_balance'].describe()
```

```
<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
```

Out[37]:

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	CNT_INSTALMENT	CNT_INSTALMENT_FUTURE
count	1.000136e+07	1.000136e+07	1.000136e+07	9.975287e+06	9.975271e+06
mean	1.903217e+06	2.784039e+05	-3.501259e+01	1.708965e+01	1.048384e+01
std	5.358465e+05	1.027637e+05	2.606657e+01	1.199506e+01	1.110906e+01
min	1.000001e+06	1.000010e+05	-9.600000e+01	1.000000e+00	0.000000e+00
25%	1.434405e+06	1.895500e+05	-5.400000e+01	1.000000e+01	3.000000e+00
50%	1.896565e+06	2.786540e+05	-2.800000e+01	1.200000e+01	7.000000e+00
75%	2.368963e+06	3.674290e+05	-1.300000e+01	2.400000e+01	1.400000e+01
max	2.843499e+06	4.562550e+05	-1.000000e+00	9.200000e+01	8.500000e+01

Missing Data

In [38]:

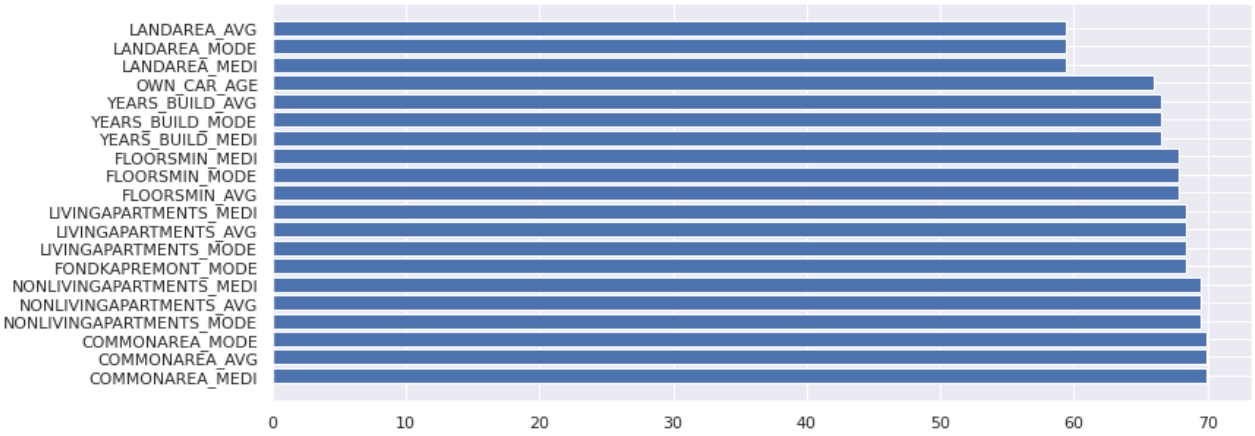
```
# returns the count and percentage of missing data from the dataset
def get_missing_data(data_set):
    percent = (datasets[data_set].isnull().sum()/datasets[data_set].isnull().count()*10
    _missing_count = datasets[data_set].isna().sum().sort_values(ascending = False)


    missing_application_train_data  = pd.concat(
        [percent, _missing_count],
        axis=1,
        keys=['Percent', "Missing Count"])

    return missing_application_train_data.head(20)

def missing_data_info(data_set):
    data = get_missing_data(data_set)
    print(data)
    _data = data[data['Percent'] != 0.0]
#     plt.figure(figsize = (5,10))
    plt.barh(y = _data.index, width = _data['Percent'])
    plt.show()
```

Application train

In [39]:

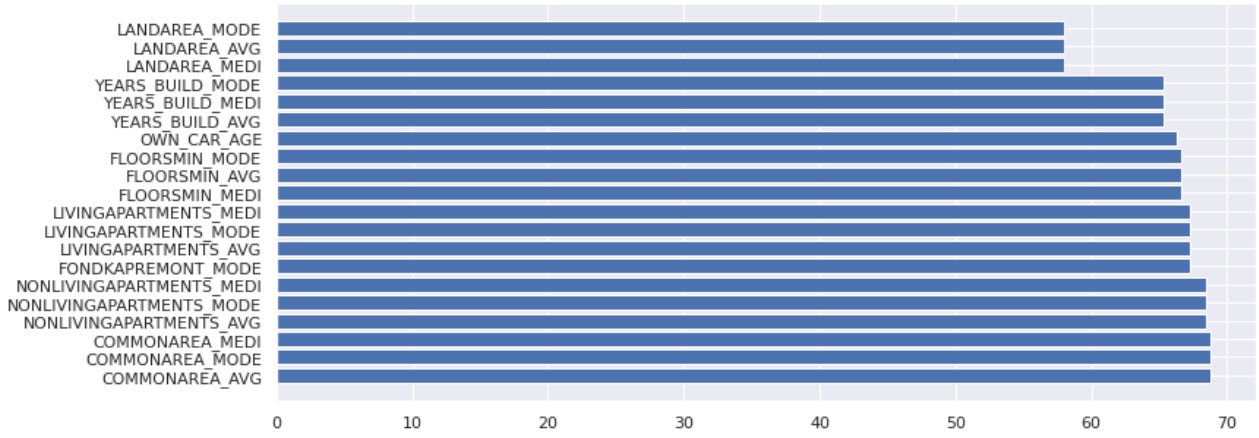
```
missing_data_info('application_train')
```

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



In [40]:
```
missing_data_info('application_test')
```

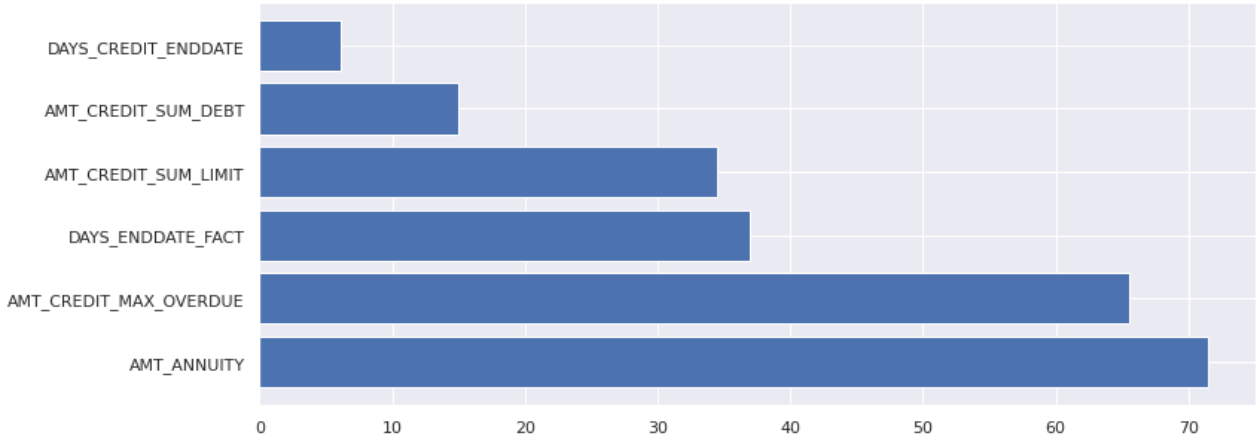
	Percent	Missing Count
COMMONAREA_AVG	68.72	33495
COMMONAREA_MODE	68.72	33495
COMMONAREA_MEDI	68.72	33495
NONLIVINGAPARTMENTS_AVG	68.41	33347
NONLIVINGAPARTMENTS_MODE	68.41	33347
NONLIVINGAPARTMENTS_MEDI	68.41	33347
FONDKAPREMONT_MODE	67.28	32797
LIVINGAPARTMENTS_AVG	67.25	32780
LIVINGAPARTMENTS_MODE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



In [41]:

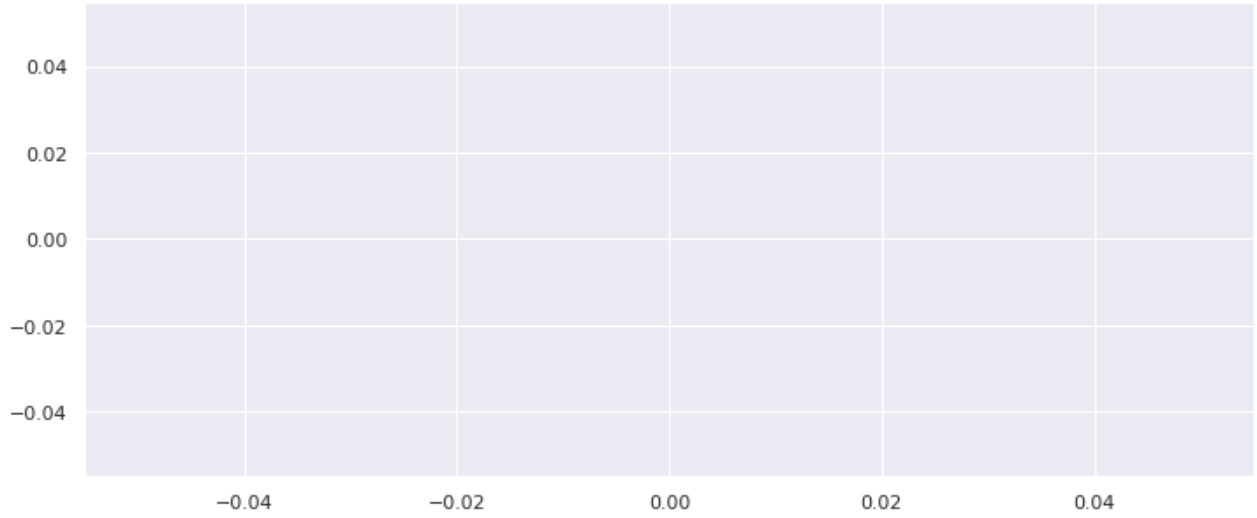
```
### other datasets

for ds_name in ds_names[2:]:
    print(f'DATASET - {ds_name}')
    missing_data_info(ds_name)
    print('\n\n')
```

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



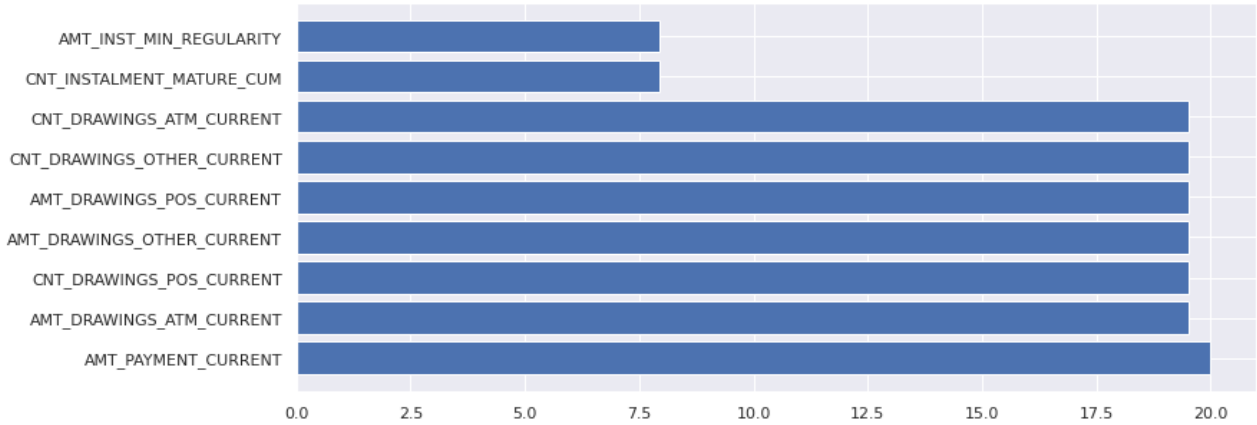

```
DATASET - bureau_balance
                Percent  Missing Count
SK_ID_BUREAU        0.0              0
```

```
MONTHS_BALANCE      0.0             0
STATUS              0.0             0
```



DATASET - credit_card_balance

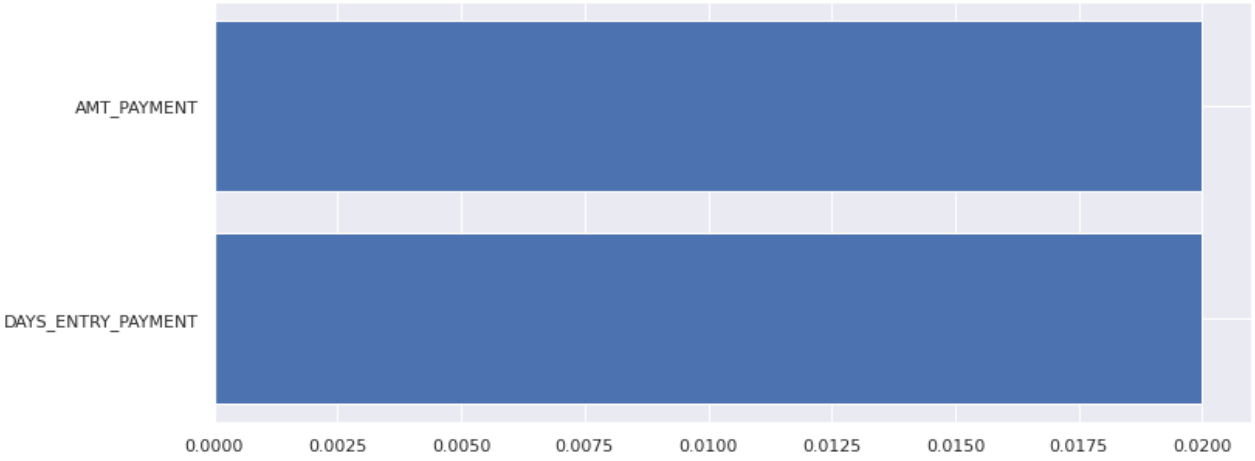
	Percent	Missing Count
AMT_PAYMENT_CURRENT	20.00	767988
AMT_DRAWINGS_ATM_CURRENT	19.52	749816
CNT_DRAWINGS_POS_CURRENT	19.52	749816
AMT_DRAWINGS_OTHER_CURRENT	19.52	749816
AMT_DRAWINGS_POS_CURRENT	19.52	749816
CNT_DRAWINGS_OTHER_CURRENT	19.52	749816
CNT_DRAWINGS_ATM_CURRENT	19.52	749816
CNT_INSTALMENT_MATURE_CUM	7.95	305236
AMT_INST_MIN_REGULARITY	7.95	305236
SK_ID_PREV	0.00	0
AMT_TOTAL_RECEIVABLE	0.00	0
SK_DPD	0.00	0
NAME_CONTRACT_STATUS	0.00	0
CNT_DRAWINGS_CURRENT	0.00	0
AMT_PAYMENT_TOTAL_CURRENT	0.00	0
AMT_RECIVABLE	0.00	0
AMT_RECEIVABLE_PRINCIPAL	0.00	0
SK_ID_CURR	0.00	0
AMT_DRAWINGS_CURRENT	0.00	0
AMT_CREDIT_LIMIT_ACTUAL	0.00	0



DATASET - installments_payments

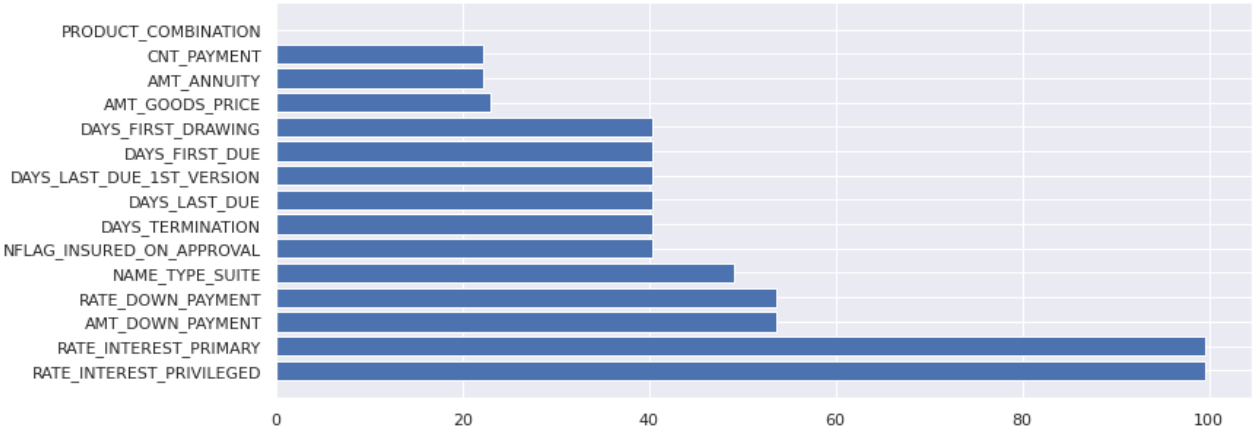
	Percent	Missing Count
DAYS_ENTRY_PAYMENT	0.02	2905
AMT_PAYMENT	0.02	2905

```
SK_ID_PREV                  0.00            0
SK_ID_CURR                  0.00            0
NUM_INSTALMENT_VERSION      0.00            0
NUM_INSTALMENT_NUMBER       0.00            0
DAYS_INSTALMENT             0.00            0
AMT_INSTALMENT              0.00            0
```



DATASET - previous_application

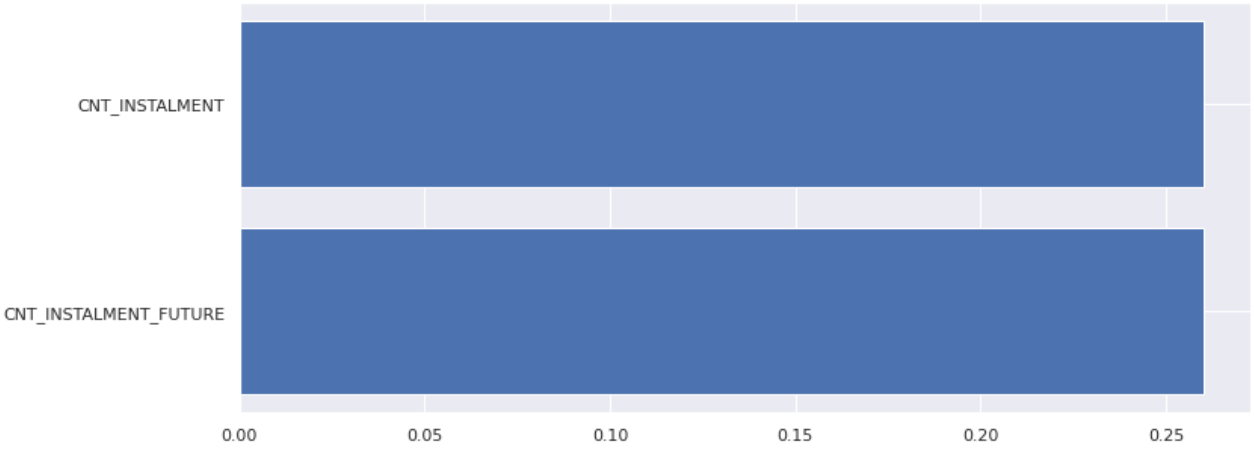
	Percent	Missing Count
RATE_INTEREST_PRIVILEGED	99.64	1664263
RATE_INTEREST_PRIMARY	99.64	1664263
AMT_DOWN_PAYMENT	53.64	895844
RATE_DOWN_PAYMENT	53.64	895844
NAME_TYPE_SUITE	49.12	820405
NFLAG_INSURED_ON_APPROVAL	40.30	673065
DAYS_TERMINATION	40.30	673065
DAYS_LAST_DUE	40.30	673065
DAYS_LAST_DUE_1ST_VERSION	40.30	673065
DAYS_FIRST_DUE	40.30	673065
DAYS_FIRST_DRAWING	40.30	673065
AMT_GOODS_PRICE	23.08	385515
AMT_ANNUITY	22.29	372235
CNT_PAYMENT	22.29	372230
PRODUCT_COMBINATION	0.02	346
AMT_CREDIT	0.00	1
NAME_YIELD_GROUP	0.00	0
NAME_PORTFOLIO	0.00	0
NAME_SELLER_INDUSTRY	0.00	0
SELLERPLACE_AREA	0.00	0



DATASET - POS_CASH_balance

	Percent	Missing Count

```
CNT_INSTALMENT_FUTURE     0.26          26087
CNT_INSTALMENT            0.26          26071
SK_ID_PREV                0.00              0
SK_ID_CURR                0.00              0
MONTHS_BALANCE            0.00              0
NAME_CONTRACT_STATUS      0.00              0
SK_DPD                    0.00              0
SK_DPD_DEF                0.00              0
```



Undersampling

As the data is imbalanced and highly biased, the defaulted population is much smaller than the good population. In here, we simply conducted a random undersampling process to the original dataset, keep all defaulted accounts

In [12]:

```
datasets["application_train_w"]  = datasets["application_train"][datasets["application_
datasets["application_train_w"]['weight'] = 1

num_default_cashloans = datasets["application_train_w"][(datasets["application_train_w"
num_default_revolvingloans = datasets["application_train_w"][(datasets["application_tra

# Undersampling Cash Loans
df_sample = datasets["application_train"][(datasets["application_train"].NAME_CONTRACT_
                    (datasets["application_train"].TARGET == 0)].sample(n = int(1.3 *

df_sample['weight'] = datasets["application_train"][(datasets["application_train"].NAME
datasets["application_train_w"] = pd.concat([datasets["application_train_w"], df_sample


# Undersampling Revovling Loans
df_sample = datasets["application_train"][(datasets["application_train"].NAME_CONTRACT_
                    (datasets["application_train"].TARGET == 0)].sample(n = int(1.3 *

df_sample['weight'] = datasets["application_train"][(datasets["application_train"].NAME
                                (datasets["application_train"].TARGET == 0)].shape[0]/in
datasets["application_train_w"] = pd.concat([datasets["application_train_w"], df_sample
```

Correlation Analysis

The correlation coefficient is not the best way to identify the relevancs between the features. But it

gives an idea about possible relationship between the data.

Correlation with the target column

In [14]:
```
correlations = datasets["application_train_w"].corr()['TARGET'].sort_values(ascending=F
print('MOST POSITIVE CORRELATIONS\n', correlations.head(10))
print('\n\nMOST NEGATIVE CORRELATIONS:\n', correlations.tail(10))
```

```
MOST POSITIVE CORRELATIONS
 TARGET                         1.000000
DAYS_BIRTH                      0.150671
DAYS_LAST_PHONE_CHANGE          0.114039
DAYS_ID_PUBLISH                 0.109171
REGION_RATING_CLIENT_W_CITY     0.107468
REGION_RATING_CLIENT            0.104992
FLAG_EMP_PHONE                  0.095970
DAYS_REGISTRATION               0.086068
REG_CITY_NOT_WORK_CITY          0.079283
REG_CITY_NOT_LIVE_CITY          0.069133
Name: TARGET, dtype: float64


MOST NEGATIVE CORRELATIONS:
 FLOORSMAX_MEDI     -0.086627
FLOORSMAX_AVG      -0.086911
DAYS_EMPLOYED      -0.094139
EXT_SOURCE_2       -0.270352
EXT_SOURCE_1       -0.287023
EXT_SOURCE_3       -0.306722
weight             -0.949066
FLAG_MOBIL               NaN
FLAG_DOCUMENT_10         NaN
FLAG_DOCUMENT_12         NaN
Name: TARGET, dtype: float64
```

Analysis on Applicant's age

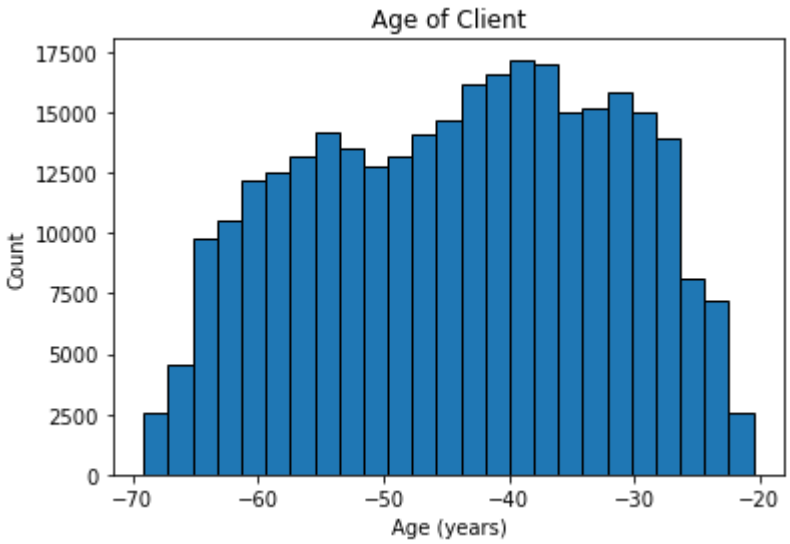
- From the top 10 most correlated features we can see that Days of birth is the most prominent feature.
- The DAYS_BIRTH indicates the age of the client in negative value at the time of the application
- The correlation between the feature and the TARGET variable is positive but the value of the feature is negative.
- This indicates that the client is less likely to default a loan as they get older.
- The absolute value of this feature is used to eliminate this uncertainity

In [15]:
```
# Find the correlation of the positive days since birth and target
datasets["application_train"]['DAYS_BIRTH'] = abs(datasets["application_train"]['DAYS_B
_corr = datasets["application_train"].corr()['TARGET']

_corr['DAYS_BIRTH']
```

Out[15]:
```
-0.07823930830982459
```

In [16]:
```
plt.hist(datasets["application_train"]['DAYS_BIRTH'] / -365, edgecolor = 'k', bins = 25
plt.title('Age of Client');
```

```
plt.xlabel('Age (years)');
plt.ylabel('Count');
```



Average failure to repay loans by age bracket.

In [17]:
```
age_data = datasets['application_train'][['TARGET', 'DAYS_BIRTH']]
age_data['YEARS_BIRTH'] = age_data['DAYS_BIRTH'] / 365

# Binning the age data
age_data['YEARS_BINNED'] = pd.cut(age_data['YEARS_BIRTH'], bins = np.linspace(20, 70, n
age_data.head(10)
```

Out[17]:

	TARGET	DAYS_BIRTH	YEARS_BIRTH	YEARS_BINNED
0	1	9461	25.920548	(25.0, 30.0]
1	0	16765	45.931507	(45.0, 50.0]
2	0	19046	52.180822	(50.0, 55.0]
3	0	19005	52.068493	(50.0, 55.0]
4	0	19932	54.608219	(50.0, 55.0]
5	0	16941	46.413699	(45.0, 50.0]
6	0	13778	37.747945	(35.0, 40.0]
7	0	18850	51.643836	(50.0, 55.0]
8	0	20099	55.065753	(55.0, 60.0]
9	0	14469	39.641096	(35.0, 40.0]

In [18]:
```
# Group by the bin and calculate averages
age_groups  = age_data.groupby('YEARS_BINNED').mean()
age_groups
```

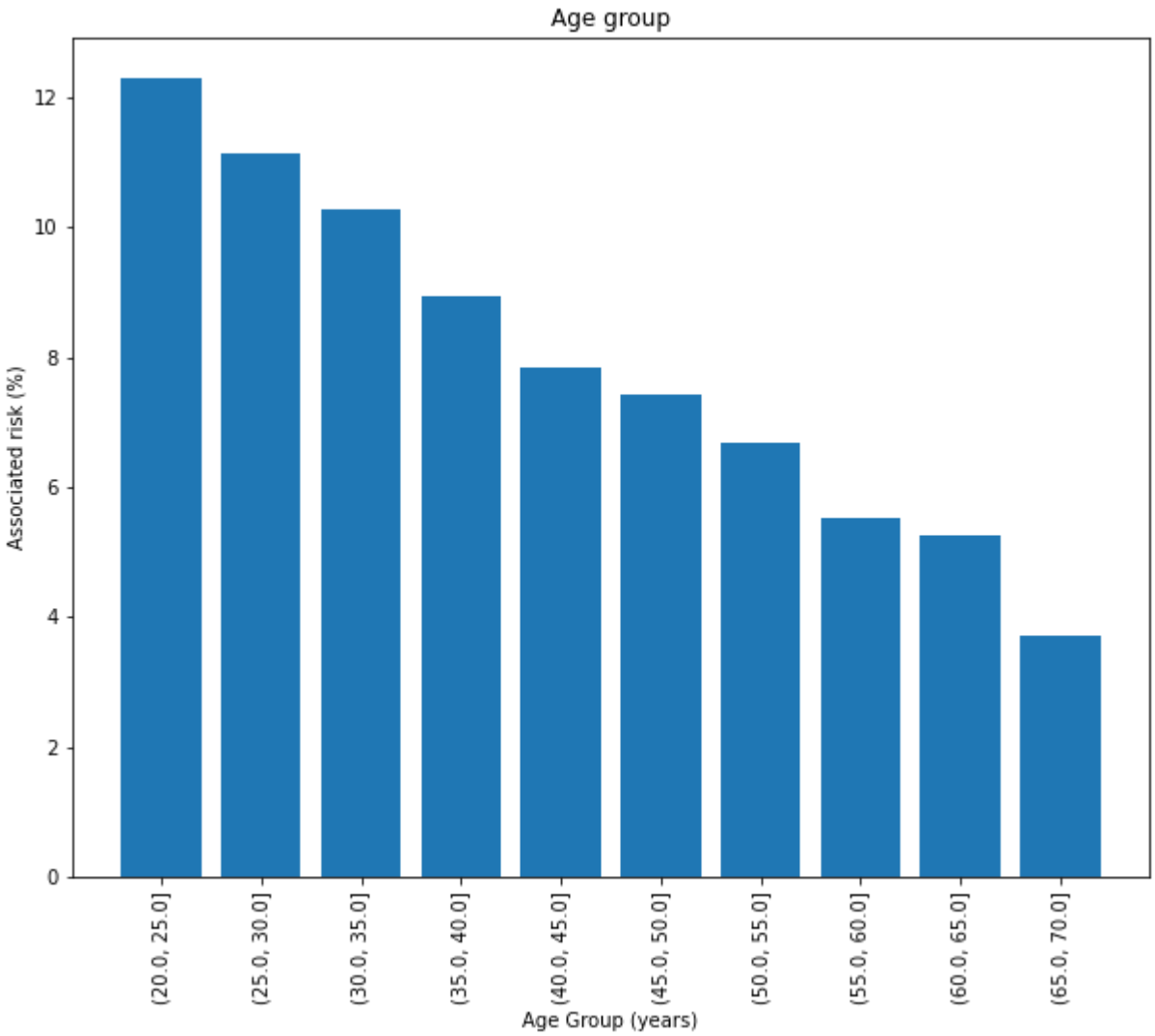
Out[18]:

	TARGET	DAYS_BIRTH	YEARS_BIRTH

YEARS_BINNED	TARGET	DAYS_BIRTH	YEARS_BIRTH
YEARS_BINNED			
(20.0, 25.0]	0.123036	8532.795625	23.377522
(25.0, 30.0]	0.111436	10155.219250	27.822518
(30.0, 35.0]	0.102814	11854.848377	32.479037
(35.0, 40.0]	0.089414	13707.908253	37.555913
(40.0, 45.0]	0.078491	15497.661233	42.459346
(45.0, 50.0]	0.074171	17323.900441	47.462741
(50.0, 55.0]	0.066968	19196.494791	52.593136
(55.0, 60.0]	0.055314	20984.262742	57.491131
(60.0, 65.0]	0.052737	22780.547460	62.412459
(65.0, 70.0]	0.037270	24292.614340	66.555108

In [19]:

```
plt.figure(figsize = (10, 8))

# Plot the age bins and the average of the TARGET as a bar plot
plt.bar(age_groups.index.astype(str), 100 * age_groups['TARGET'])

# Plot labeling
plt.xticks(rotation = 90);
plt.xlabel('Age Group (years)');
plt.ylabel('Associated risk (%)')
plt.title('Age group');
```



- It can be seen from the given graph that the young age people tend to not repay the loan
- Whereas as the age increases the client is more likely to repay the loan.

Negative correlations with the target variable

- It can be seen from the top 10 features with negative correlations EXT_SOURCE_n has the highest negative correlation with the target variable
- They are the normalized score for each applicant accumulated from the external source

In [20]:
```
_ext_source = datasets['application_train'][['TARGET', 'EXT_SOURCE_1', 'EXT_SOURCE_2',
_ext_source_corrs = _ext_source.corr()
_ext_source_corrs
```

Out[20]:

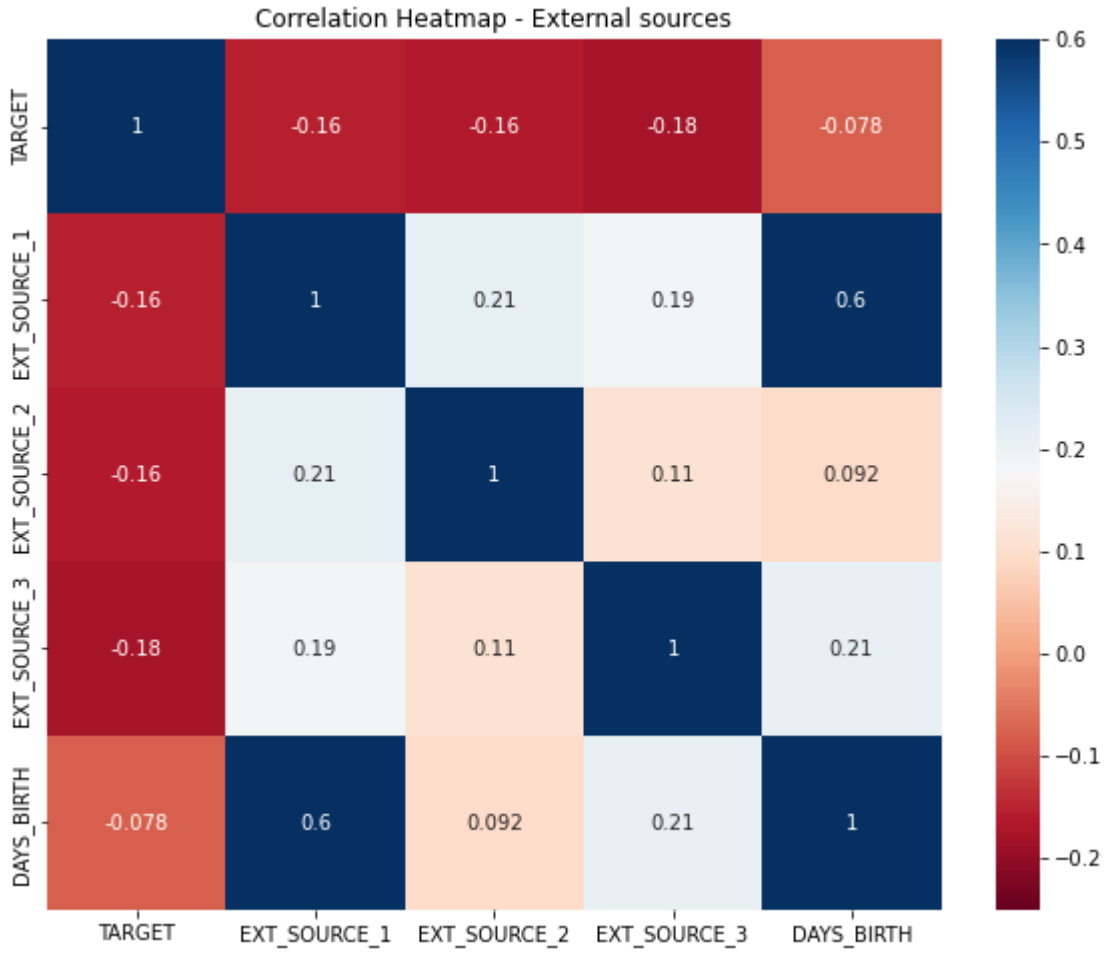
	TARGET	EXT_SOURCE_1	EXT_SOURCE_2	EXT_SOURCE_3	DAYS_BIRTH
TARGET	1.000000	-0.155317	-0.160472	-0.178919	-0.078239
EXT_SOURCE_1	-0.155317	1.000000	0.213982	0.186846	0.600610
EXT_SOURCE_2	-0.160472	0.213982	1.000000	0.109167	0.091996
EXT_SOURCE_3	-0.178919	0.186846	0.109167	1.000000	0.205478

	TARGET	EXT_SOURCE_1	EXT_SOURCE_2	EXT_SOURCE_3	DAYS_BIRTH
DAYS_BIRTH	-0.078239	0.600610	0.091996	0.205478	1.000000

Heat map

- The heat map for the highly correlated features is implemented as follows

In [21]:
```
plt.figure(figsize = (10, 8))

# Correlation Heatmap of the Exterior Sources
sns.heatmap(_ext_source_corrs, vmin = -0.25, annot = True, vmax = 0.6, cmap='RdBu')
plt.title('Correlation Heatmap - External sources');
```



- From the heat map it is clear that as the value provided by the external source increases the default risk associated with the client decreases
- There exists a positive correlation between the client age and external source 1. This indicates that client age could be one of the factor being involved in calculating this score.

Duplicate data

In [22]:
```
columns_without_id = [col for col in datasets['application_train'].columns if col!='SK_
```

```
#Checking for duplicates in the data.
datasets['application_train'][datasets['application_train'].duplicated(subset = columns

print('The no of duplicates in the data:',datasets['application_train'][datasets['appli
      .shape[0])
```

```
The no of duplicates in the data: 0
```

- It can be seen that there is no duplicate entries in the given application train dataset

In [23]:
```
len(datasets["application_train"]["SK_ID_CURR"].unique()) == datasets["application_trai
```

Out[23]:
```
True
```

In [24]:
```
np.intersect1d(datasets["application_train"]["SK_ID_CURR"], datasets["application_test"
```

Out[24]:
```
array([], dtype=int64)
```

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 [25]:
```
appsDF = datasets["previous_application"]
```

In [26]:
```
len(np.intersect1d(datasets["previous_application"]["SK_ID_CURR"], datasets["applicatio
```

Out[26]:
```
47800
```

In [27]:
```
print(f"There are  {appsDF.shape[0]:,} previous applications")
```

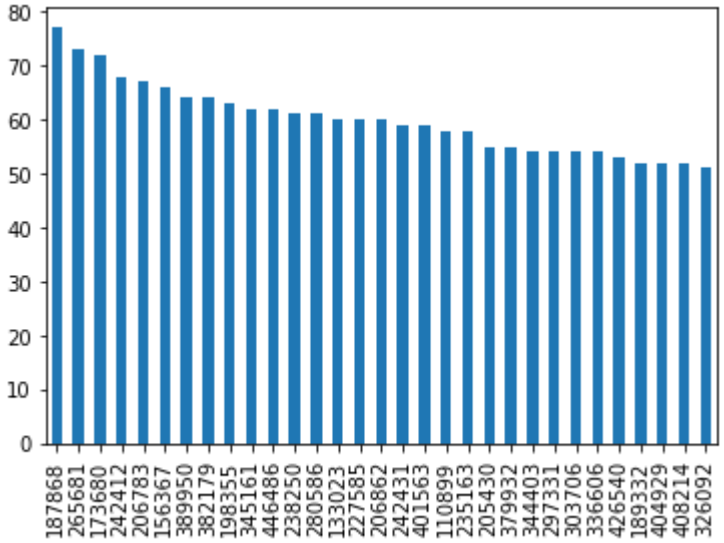
```
There are  1,670,214 previous applications
```

In [28]:
```
# How many entries are there for each month?
prevAppCounts = appsDF['SK_ID_CURR'].value_counts(dropna=False)
```

In [29]:
```
len(prevAppCounts[prevAppCounts >40])  #more that 40 previous applications
```

Out[29]:
```
101
```

In [30]:
```
prevAppCounts[prevAppCounts >50].plot(kind='bar')
plt.xticks(rotation=90)
plt.show()
```



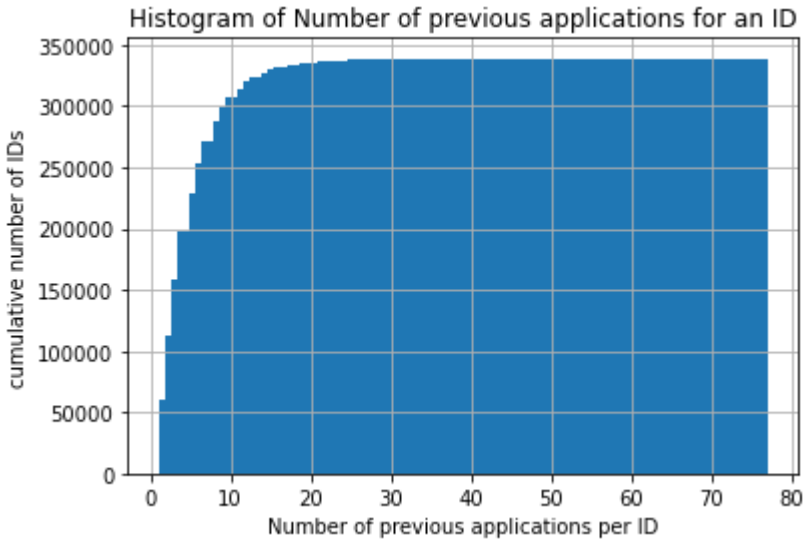
Histogram of Number of previous applications for an ID

In [31]:
```
sum(appsDF['SK_ID_CURR'].value_counts()==1)
```

Out[31]: 60458

In [32]:
```
plt.hist(appsDF['SK_ID_CURR'].value_counts(), cumulative =True, bins = 100);
plt.grid()
plt.ylabel('cumulative number of IDs')
plt.xlabel('Number of previous applications per ID')
plt.title('Histogram of Number of previous applications for an ID')
```

Out[32]: Text(0.5, 1.0, 'Histogram of Number of previous applications for an ID')



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

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

In [33]:
```
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_
print('Percentage with 40 or more previous apps:', np.round(100.*(sum(apps_40plus)/apps
```

```
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

In [34]:
```
features = ['AMT_ANNUITY', 'AMT_APPLICATION']
agg_op_features = {}
for f in features: #build agg dictionary
    agg_op_features[f] = ["min", "max", "mean"]
print(f"{appsDF[features].describe()}")
print(agg_op_features)
result = appsDF.groupby(["SK_ID_CURR"]).agg(agg_op_features)
# print(result.columns)
# result.columns = result.columns.droplevel() #drop 1 of the header row but keep the fe
df_previous_use = pd.DataFrame()
for x1, x2 in result.columns:
    new_col = x1 + "_" + x2
    df_previous_use[new_col] = result[x1][x2]
# df_previous_use = df_previous_use.reset_index(level=["SK_ID_CURR"])
df_previous_use['range_AMT_APPLICATION'] = df_previous_use['AMT_APPLICATION_max'] - df_
print(f"df_previous_use.shape: {df_previous_use.shape}")
df_previous_use[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
{'AMT_ANNUITY': ['min', 'max', 'mean'], 'AMT_APPLICATION': ['min', 'max', 'mean']}
df_previous_use.shape: (338857, 7)
```

Out[34]:

SK_ID_CURR	AMT_ANNUITY_min	AMT_ANNUITY_max	AMT_ANNUITY_mean	AMT_APPLICATION_min	A
100001	3951.000	3951.000	3951.000000	24835.5	

SK_ID_CURR	AMT_ANNUITY_min	AMT_ANNUITY_max	AMT_ANNUITY_mean	AMT_APPLICATION_min	A
100002	9251.775	9251.775	9251.775000	179055.0	
100003	6737.310	98356.995	56553.990000	68809.5	
100004	5357.250	5357.250	5357.250000	24282.0	
100005	4813.200	4813.200	4813.200000	0.0	
100006	2482.920	39954.510	23651.175000	0.0	
100007	1834.290	22678.785	12278.805000	17176.5	
100008	8019.090	25309.575	15839.696250	0.0	
100009	7435.845	17341.605	10051.412143	40455.0	
100010	27463.410	27463.410	27463.410000	247212.0	

In [35]:
```
result.isna().sum()
```

Out[35]:
```
AMT_ANNUITY      min     480
                 max     480
                 mean    480
AMT_APPLICATION  min       0
                 max       0
                 mean      0
dtype: int64
```

Feature Engineering

Here, we are using the another data sources in order to get more in-depth details about the credit holder to predict the home loan repayable or not.

In [36]:
```
# Create aggregate features (via pipeline)
class FeaturesAggregater(BaseEstimator, TransformerMixin):
    def __init__(self, features=None, agg_needed=["mean"]): # no *args or **kargs
        self.features = features
        self.agg_needed = agg_needed
        self.agg_op_features = {}
        for f in features:
            self.agg_op_features[f] = self.agg_needed[:]

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

    def transform(self, X, y=None):
        result = X.groupby(["SK_ID_CURR"]).agg(self.agg_op_features)
        df_result = pd.DataFrame()
        for x1, x2 in result.columns:
            new_col = x1 + "_" + x2
            df_result[new_col] = result[x1][x2]
        df_result = df_result.reset_index(level=["SK_ID_CURR"])
        return df_result
```

Feature engineering for Previous Applications

Missing values

In [37]:
```
datasets["previous_application"].isna().sum()
```

Out[37]:
```
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 [38]:
```
previous_feature = ["AMT_APPLICATION", "AMT_CREDIT", "AMT_ANNUITY", "approved_credit_ra
                    "AMT_ANNUITY_credit_ratio", "Interest_ratio", "LTV_ratio", "SK_ID_P

agg_needed = ["min", "max", "mean", "count", "sum"]

def previous_feature_aggregation(df, feature, agg_needed):
    df['approved_credit_ratio'] = (df['AMT_APPLICATION']/df['AMT_CREDIT']).replace(np.i
    # installment over credit approved ratio
    df['AMT_ANNUITY_credit_ratio'] = (df['AMT_ANNUITY']/df['AMT_CREDIT']).replace(np.in
    # total interest payment over credit ratio
    df['Interest_ratio'] = (df['AMT_ANNUITY']/df['AMT_CREDIT']).replace(np.inf, 0)
    #  loan cover ratio
    df['LTV_ratio'] = (df['AMT_CREDIT']/df['AMT_GOODS_PRICE']).replace(np.inf, 0)
    df['approved'] = np.where(df.AMT_CREDIT >0 ,1, 0)
```

```
    test_pipeline = make_pipeline(FeaturesAggregater(feature, agg_needed))
    return(test_pipeline.fit_transform(df))

datasets['previous_application_agg'] = previous_feature_aggregation(datasets["previous_
```

Missing value after the feature engineering

In [39]:
```
datasets["previous_application_agg"].isna().sum()
```

Out[39]:
```
SK_ID_CURR                          0
AMT_APPLICATION_min                 0
AMT_APPLICATION_max                 0
AMT_APPLICATION_mean                0
AMT_APPLICATION_count               0
AMT_APPLICATION_sum                 0
AMT_CREDIT_min                      0
AMT_CREDIT_max                      0
AMT_CREDIT_mean                     0
AMT_CREDIT_count                    0
AMT_CREDIT_sum                      0
AMT_ANNUITY_min                   480
AMT_ANNUITY_max                   480
AMT_ANNUITY_mean                  480
AMT_ANNUITY_count                   0
AMT_ANNUITY_sum                     0
approved_credit_ratio_min         252
approved_credit_ratio_max         252
approved_credit_ratio_mean        252
approved_credit_ratio_count         0
approved_credit_ratio_sum           0
AMT_ANNUITY_credit_ratio_min      481
AMT_ANNUITY_credit_ratio_max      481
AMT_ANNUITY_credit_ratio_mean     481
AMT_ANNUITY_credit_ratio_count      0
AMT_ANNUITY_credit_ratio_sum        0
Interest_ratio_min                481
Interest_ratio_max                481
Interest_ratio_mean               481
Interest_ratio_count                0
Interest_ratio_sum                  0
LTV_ratio_min                    1073
LTV_ratio_max                    1073
LTV_ratio_mean                   1073
LTV_ratio_count                     0
LTV_ratio_sum                       0
SK_ID_PREV_min                      0
SK_ID_PREV_max                      0
SK_ID_PREV_mean                     0
SK_ID_PREV_count                    0
SK_ID_PREV_sum                      0
approved_min                        0
approved_max                        0
approved_mean                       0
approved_count                      0
approved_sum                        0
dtype: int64
```

Feature engineering for Installment payments

Missing values

In [40]:
```
datasets["installments_payments"].isna().sum()
```

Out[40]:
```
SK_ID_PREV                   0
SK_ID_CURR                   0
NUM_INSTALMENT_VERSION       0
NUM_INSTALMENT_NUMBER        0
DAYS_INSTALMENT              0
DAYS_ENTRY_PAYMENT        2905
AMT_INSTALMENT               0
AMT_PAYMENT               2905
dtype: int64
```

In [41]:
```
payments_features = ["DAYS_INSTALMENT_DIFF", "AMT_PATMENT_PCT"]

agg_needed = ["mean"]

def payments_feature_aggregation(df, feature, agg_needed):
    df['DAYS_INSTALMENT_DIFF'] = df['DAYS_INSTALMENT'] - df['DAYS_ENTRY_PAYMENT']
    df['AMT_PATMENT_PCT'] = [x/y if (y != 0) & pd.notnull(y) else np.nan for x,y in zip

    test_pipeline = make_pipeline(FeaturesAggregater(feature, agg_needed))
    return(test_pipeline.fit_transform(df))

datasets['installments_payments_agg'] = payments_feature_aggregation(datasets["installm
```

Missing value after the feature engineering

In [42]:
```
datasets["installments_payments_agg"].isna().sum()
```

Out[42]:
```
SK_ID_CURR                   0
DAYS_INSTALMENT_DIFF_mean    9
AMT_PATMENT_PCT_mean        12
dtype: int64
```

Feature engineering Credit card balance

Missing value

In [43]:
```
datasets["credit_card_balance"].isna().sum()
```

Out[43]:
```
SK_ID_PREV                      0
SK_ID_CURR                      0
MONTHS_BALANCE                  0
AMT_BALANCE                     0
AMT_CREDIT_LIMIT_ACTUAL         0
AMT_DRAWINGS_ATM_CURRENT   749816
AMT_DRAWINGS_CURRENT            0
AMT_DRAWINGS_OTHER_CURRENT 749816
AMT_DRAWINGS_POS_CURRENT   749816
AMT_INST_MIN_REGULARITY    305236
AMT_PAYMENT_CURRENT        767988
AMT_PAYMENT_TOTAL_CURRENT       0
AMT_RECEIVABLE_PRINCIPAL        0
AMT_RECIVABLE                   0
AMT_TOTAL_RECEIVABLE            0
CNT_DRAWINGS_ATM_CURRENT   749816
CNT_DRAWINGS_CURRENT            0
```

```
CNT_DRAWINGS_OTHER_CURRENT    749816
CNT_DRAWINGS_POS_CURRENT      749816
CNT_INSTALMENT_MATURE_CUM     305236
NAME_CONTRACT_STATUS               0
SK_DPD                             0
SK_DPD_DEF                         0
dtype: int64
```

In [44]:
```
credit_features = ["AMT_BALANCE", "AMT_DRAWINGS_PCT", "AMT_DRAWINGS_ATM_PCT", "AMT_DRAW
                   "AMT_PRINCIPAL_RECEIVABLE_PCT", "CNT_DRAWINGS_ATM_CURRENT", "CNT_DRA
                   "CNT_DRAWINGS_POS_CURRENT", "SK_DPD", "SK_DPD_DEF"]

agg_needed = ["mean"]

def credit_feature_aggregation(df, feature, agg_needed):
    df['AMT_DRAWINGS_PCT'] = [x/y if (y != 0) & pd.notnull(y) else np.nan for x,y in zi
    df['AMT_DRAWINGS_ATM_PCT'] = [x/y if (y != 0) & pd.notnull(y) else np.nan for x,y i
    df['AMT_DRAWINGS_OTHER_PCT'] = [x/y if (y != 0) & pd.notnull(y) else np.nan for x,y
    df['AMT_DRAWINGS_POS_PCT'] = [x/y if (y != 0) & pd.notnull(y) else np.nan for x,y i
    df['AMT_PRINCIPAL_RECEIVABLE_PCT'] = [x/y if (y != 0) & pd.notnull(y) else np.nan f

    test_pipeline = make_pipeline(FeaturesAggregater(feature, agg_needed))
    return(test_pipeline.fit_transform(df))


datasets['credit_card_balance_agg'] = credit_feature_aggregation(datasets["credit_card_
```

Missing values after feature engineering

In [45]:
```
datasets["credit_card_balance_agg"].isna().sum()
```

Out[45]:
```
SK_ID_CURR                          0
AMT_BALANCE_mean                    0
AMT_DRAWINGS_PCT_mean            1113
AMT_DRAWINGS_ATM_PCT_mean       32278
AMT_DRAWINGS_OTHER_PCT_mean     32278
AMT_DRAWINGS_POS_PCT_mean       32278
AMT_PRINCIPAL_RECEIVABLE_PCT_mean    33033
CNT_DRAWINGS_ATM_CURRENT_mean   31364
CNT_DRAWINGS_CURRENT_mean           0
CNT_DRAWINGS_OTHER_CURRENT_mean 31364
CNT_DRAWINGS_POS_CURRENT_mean   31364
SK_DPD_mean                         0
SK_DPD_DEF_mean                     0
dtype: int64
```

In [46]:
```
datasets.keys()
```

Out[46]:
```
dict_keys(['application_train', 'application_test', 'bureau', 'bureau_balance', 'credit_
card_balance', 'installments_payments', 'previous_application', 'POS_CASH_balance', 'app
lication_train_w', 'previous_application_agg', 'installments_payments_agg', 'credit_card
_balance_agg'])
```

In [47]:
```
X_train= datasets["application_train_w"] #primary dataset

merge_all_data = True
```

```
# merge primary table and secondary tables using features based on meta data and  aggre
if merge_all_data:
    # 1. Join/Merge in prevApps Data
    X_train = X_train.merge(datasets["previous_application_agg"], how='left', on='SK_ID

    # 2. Join/Merge in Installments Payments  Data
    X_train = X_train.merge(datasets["installments_payments_agg"], how='left', on="SK_I

    # 3. Join/Merge in Credit Card Balance Data
    X_train = X_train.merge(datasets["credit_card_balance_agg"], how='left', on="SK_ID_

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

    # ......
```

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

In [48]:

```
X_kaggle_test= datasets["application_test"]

# merge primary table and secondary tables using features based on meta data and  aggre
if merge_all_data:
    # 1. Join/Merge in prevApps Data
    X_kaggle_test = X_kaggle_test.merge(datasets["previous_application_agg"], how='left

    # 2. Join/Merge in Installments Payments  Data
    X_kaggle_test = X_kaggle_test.merge(datasets["installments_payments_agg"], how='lef

    # 3. Join/Merge in Credit Card Balance Data
    X_kaggle_test = X_kaggle_test.merge(datasets["credit_card_balance_agg"], how='left'

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

    # ......
```

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"))
    ])
```

HCDR preprocessing

Creating Datasets

In [49]:
```
#train_dataset = datasets["application_train"]
train_dataset=X_train
class_labels = ["No Default","Default"]
```

In [50]:
```
# Create a class to select numerical or categorical columns since Scikit-Learn doesn't
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
```

Finding the top positive and negtive Correlation features with respective to target variable.

In [51]:
```
correlations = train_dataset.corr()['TARGET'].sort_values(ascending=False)
print('MOST POSITIVE CORRELATIONS\n', correlations.head(10))
print('\n\nMOST NEGATIVE CORRELATIONS:\n', correlations.tail(10))
```

```
MOST POSITIVE CORRELATIONS
```

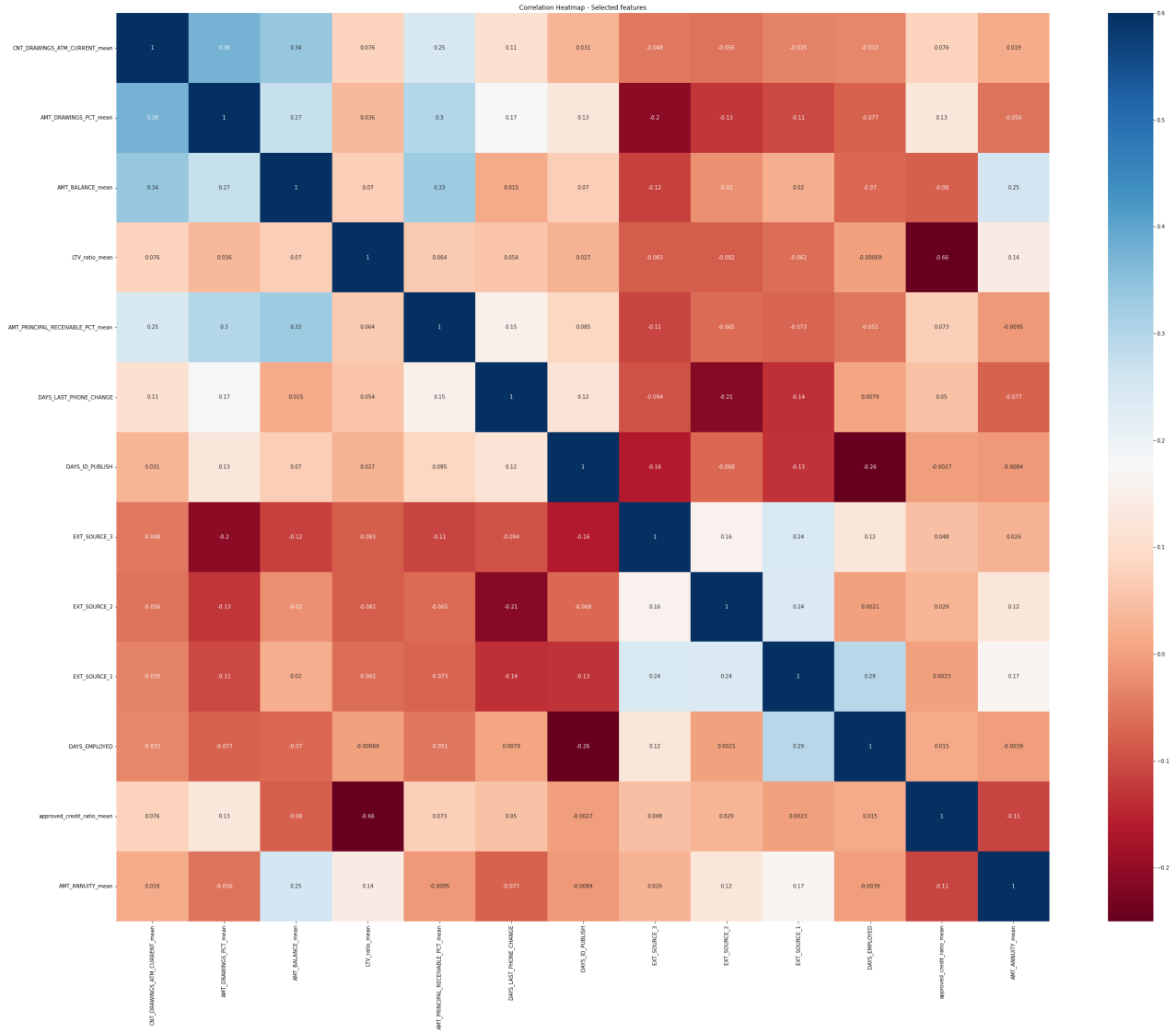
```
 TARGET                         1.000000
AMT_DRAWINGS_PCT_mean           0.174101
DAYS_BIRTH                      0.150671
CNT_DRAWINGS_CURRENT_mean       0.132638
CNT_DRAWINGS_ATM_CURRENT_mean   0.130541
AMT_BALANCE_mean                0.130352
LTV_ratio_mean                  0.124084
AMT_DRAWINGS_ATM_PCT_mean       0.120099
DAYS_LAST_PHONE_CHANGE          0.114039
DAYS_ID_PUBLISH                 0.109171
Name: TARGET, dtype: float64


MOST NEGATIVE CORRELATIONS:
 FLOORSMAX_MEDI     -0.086627
FLOORSMAX_AVG      -0.086911
DAYS_EMPLOYED      -0.094139
EXT_SOURCE_2       -0.270352
EXT_SOURCE_1       -0.287023
EXT_SOURCE_3       -0.306722
weight             -0.949066
FLAG_MOBIL               NaN
FLAG_DOCUMENT_10         NaN
FLAG_DOCUMENT_12         NaN
Name: TARGET, dtype: float64
```

Selecting the numerical features

Initially, considered the top positive and negtive corelated features with respective to target variable and further refined by finding the correlationg between the above selected features.

In [52]:

```
if merge_all_data:
    num_attribs = ["CNT_DRAWINGS_ATM_CURRENT_mean", "AMT_DRAWINGS_PCT_mean", "AMT_BALAN
                   "AMT_PRINCIPAL_RECEIVABLE_PCT_mean", "DAYS_LAST_PHONE_CHANGE", "DAYS_
                   "EXT_SOURCE_3", "EXT_SOURCE_2","EXT_SOURCE_1", "DAYS_EMPLOYED", "app
                   "AMT_ANNUITY_mean"]
else:
    num_attribs = ['AMT_INCOME_TOTAL', 'AMT_CREDIT','DAYS_EMPLOYED','DAYS_BIRTH','EXT_S

_ext_source = X_train[num_attribs]
_ext_source_corrs = _ext_source.corr()
_ext_source_corrs

plt.figure(figsize = (40, 32))

# Correlation Heatmap of the Exterior Sources
sns.heatmap(_ext_source_corrs, vmin = -0.25, annot = True, vmax = 0.6, cmap='RdBu')
plt.title('Correlation Heatmap - Selected features');
```



Numerical Pipeline Definition

In [53]:
```
# Numerical Pipeline definition
num_pipeline = Pipeline([
        ('selector', DataFrameSelector(num_attribs)),
        ('imputer', SimpleImputer(strategy='mean')),
        ('std_scaler', StandardScaler()),
    ])
```

Selecting the categorical features

In [54]:
```
# 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']
```

Categorical Pipeline Definition

Notice handle_unknown="ignore" in OHE which ignore values from the validation/test that do NOT occur in the training set

In [55]:
```
cat_pipeline = Pipeline([
```

```
        ('selector', DataFrameSelector(cat_attribs)),
        ('imputer', SimpleImputer(strategy='most_frequent')),
        #('imputer', SimpleImputer(strategy='constant', fill_value='missing')),
        ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
    ])
```

Create Data Preparation Pipeline

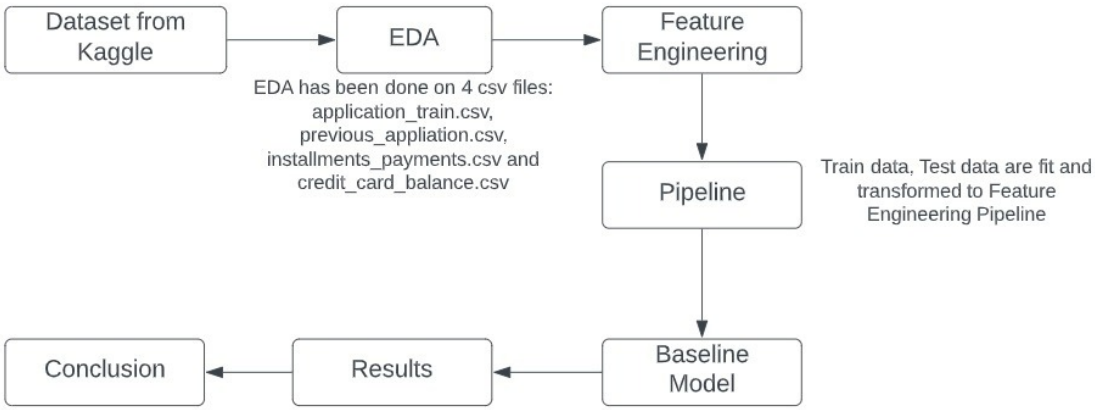
With Feature union, combine numerical and categorical Pipeline together to prepare for Data pipeline

In [56]:
```
data_prep_pipeline = FeatureUnion(transformer_list=[
        ("num_pipeline", num_pipeline),
        ("cat_pipeline", cat_pipeline),
    ])
```

In [57]:
```
# Selected Features
selected_features = num_attribs + cat_attribs + ["SK_ID_CURR"]
tot_features = f"{len(selected_features)}:   Num:{len(num_attribs)},    Cat:{len(cat_at
#Total Feature selected for processing
tot_features
```

Out[57]: '21:   Num:13,   Cat:7'

Baseline model with Imbalanced Dataset



Create Train and Test Datasets

In [58]:
```
# Split Sample to feed the pipeline and it will result in a new dataset that is (1 / sp
splits = 3

# Train Test split percentage
subsample_rate = 0.3

# finaldf = np.array_split(train_dataset, splits)
X_train = train_dataset[selected_features]
y_train = train_dataset['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_st

X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train,stratify=y_train

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}")
print(f"X y_train         shape: {y_train.shape}")
# print(y_train.shape)
```

```
X train           shape: (10191, 21)
X validation      shape: (1799, 21)
X test            shape: (5139, 21)
X X_kaggle_test   shape: (48744, 21)
X y_train         shape: (10191,)
```

In [59]:

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

In [143…

```
try:
    expLog
except NameError:
    expLog = pd.DataFrame(columns=["exp_name",
                                   "Train Acc",
                                   "Valid Acc",
                                   "Test  Acc",
                                   "Train AUC",
                                   "Valid AUC",
                                   "Test  AUC",
                                   "Train F1 Score",
                                   "Test F1 Score",
                                   "Train Log Loss",
                                   "Test Log Loss",
                                   "P Score",
                                   "Train Time",
                                   "Test Time",
                                   "Description"
                                  ])
```

Modeling

Now that we have explored the data, cleaned it, preprocessed it and added a new feature to it, we can start the modeling part of the project by applying Machine Learning algorithms. In this section, you will have a baseline logistic regression model and grid searches on different models. In the end, you will find out which parameters are the best for each algorithm and you will be able to compare the performance of the models with the baseline model.

Building baseline logistic 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 [61]:
```
# Define Pipeline
%time
np.random.seed(42)
full_pipeline_with_predictor = Pipeline([
        ("preparation", data_prep_pipeline),
        ("linear", LogisticRegression())
    ])
```

```
CPU times: user 5 µs, sys: 1 µs, total: 6 µs
Wall time: 11.9 µs
```

Perform cross-fold validation and Train the model

In [62]:
```
# Split the training data to 15 fold to perform Crossfold validation

from sklearn.model_selection import ShuffleSplit

cvSplits = ShuffleSplit(n_splits=5, test_size=0.2, random_state=42)
```

In [63]:
```
import time

start = time.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=cvSpli
logit_score_train = pct(logit_scores.mean())
train_time = np.round(time.time() - start, 4)

# Time and score test predictions
start = time.time()
logit_score_test  = full_pipeline_with_predictor.score(X_test, y_test)
test_time = np.round(time.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 [64]:
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import accuracy_score, confusion_matrix, f1_score, log_loss, class


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

Out[64]: 0.684

In [65]:
```
from sklearn.metrics import roc_auc_score
roc_auc_score(y_train, model.predict_proba(X_train)[:, 1])
```

Out[65]: 0.7466210622634238

In [66]:
```
# Calculate metrics
exp_name = f"Baseline_{len(selected_features)}_features"
expLog.loc[len(expLog)] = [f"{exp_name}"] + list(np.round(
               [logit_score_train,
                pct(accuracy_score(y_valid, model.predict(X_valid))),
                pct(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]),
                f1_score(y_train, model.predict(X_train)),
                f1_score(y_test, model.predict(X_test)),
                log_loss(y_train, model.predict(X_train)),
                log_loss(y_test, model.predict(X_test)),0 ],4)) \
                + [train_time,test_time] + [f" Logistic reg features {tot_features} wit
expLog
```

Out[66]:

	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC	Train F1 Score	Test F1 Score	Train Log Loss	Test Log Loss	S
0	Baseline_21_features	68.112	68.482	68.574	0.7466	0.7406	0.7399	0.6108	0.6128	10.9165	10.8544	

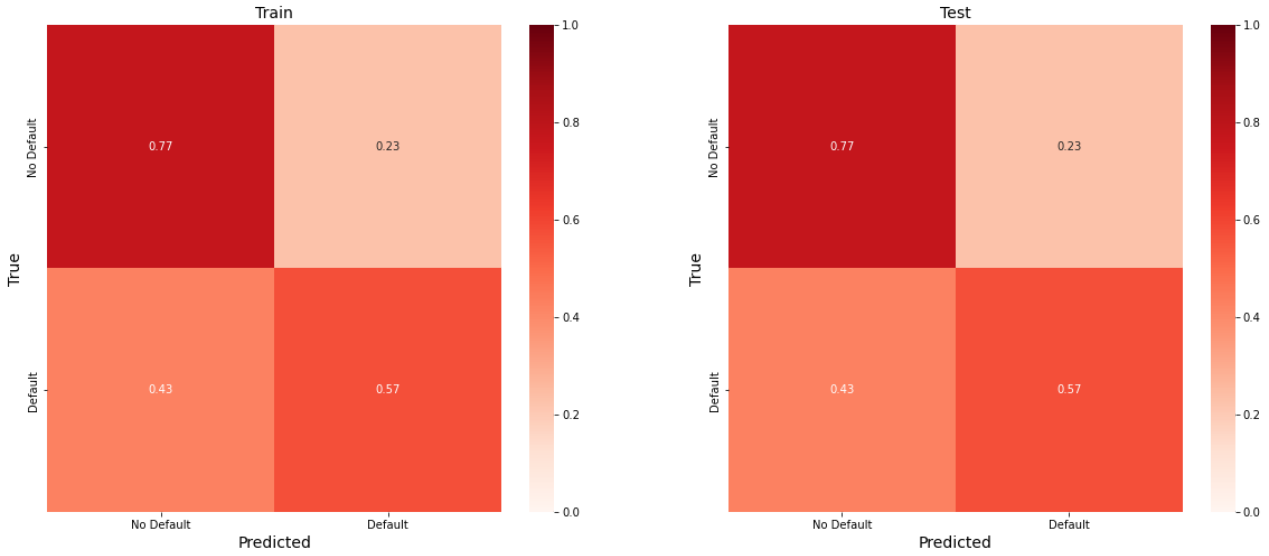
Confusion Matrix

In [67]:
```
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]
```

```
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);
```

Create confusion matrix for baseline model

In [68]:
```
confusion_matrix_def(model,X_train,y_train,X_test,y_test)
```



In [69]:
```
print(X_train.shape)
```

```
(10191, 21)
```

Decision Trees

Now that we have explored the data, cleaned it, preprocessed it, and added a new feature to it, we can start the modeling part of the project by applying Machine Learning algorithms. In this section we apply decision tree and random forests.

Decision Tree is one of the popular and most widely used Machine Learning Algorithms because of its robustness to noise, tolerance against missing information, handling of irrelevant, redundant predictive attribute values, low computational cost, interpretability, fast run time and robust predictors.

Cost functions used for classification and regression.

In both cases the cost functions try to find most homogeneous branches, or branches having groups with similar responses.

Regression : sum(y — prediction)^2

Classification : G = sum(pk * (1 — pk))

A Gini score gives an idea of how good a split is by how mixed the response classes are in the groups created by the split. Here, pk is proportion of same class inputs present in a particular group.

DecisionTreeClassifier

The function to measure the quality of a split. Supported criteria are “gini” for the Gini impurity and “entropy” for the information gain. Information gain uses the entropy measure as the impurity measure and splits a node such that it gives the most amount of information gain. Whereas Gini Impurity measures the divergences between the probability distributions of the target attribute’s values and splits a node such that it gives the least amount of impurity.

Gini : $1 - \sum_{i=1}^{m}(P_j^2)$

Entropy : $\sum_{i=1}^{m}\left(P_j \cdot \log\left(P_j\right)\right)$

DecisionTreeRegressor

The decision of making strategic splits heavily affects a tree’s accuracy. The decision criteria is different for classification and regression trees.Decision trees regression normally use mean squared error (MSE) to decide to split a node in two or more sub-nodes.

Suppose we are doing a binary tree the algorithm first will pick a value, and split the data into two subset. For each subset, it will calculate the MSE separately. The tree chooses the value with results in smallest MSE value.

$MSE = \frac{1}{m}\sum_{i=1}^{m}(x_i - y_i)^2$

Feature importance formula

To calculate the importance of each feature, we will mention the decision point itself and its child nodes as well. The following formula covers the calculation of feature importance.

For each decision tree, Scikit-learn calculates a nodes importance using Gini Importance, assuming only two child nodes (binary tree):

$$ni_j = w_j C_j - w_{left(j)} C_{left(j)} - w_{right(j)} C_{right(j)}$$

Where

ni_j= the importance of node j

w_j = weighted number of samples reaching node j

C_j= the impurity value of node j

left(j) = child node from left split on node j

right(j) = child node from right split on node j

The importance for each feature on a decision tree is then calculated as:

$$fi_i = \frac{\sum_{j:node\ j\ splits\ on\ feature\ i} ni_j}{\sum_{k\ \epsilon\ all\ nodes} ni_k}$$

These can then be normalized to a value between 0 and 1 by dividing by the sum of all feature importance values:

$$norm fi_i = \frac{fi_i}{\sum_{j\ \epsilon\ all\ features} fi_j}$$

Decision Trees Parameters for classification

criterion{"gini", "entropy"}, default="gini" :The function to measure the quality of a split. Supported criteria are "gini" for the Gini impurity and "entropy" for the information gain.

max_depth, default=None : The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

min_samples_leaf int or float, default=1 : The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.

Decision Trees Parameters for Regression

criterion{"mse", "friedman_mse", "mae", "poisson"}, default="mse" : The function to measure the quality of a split.

max_depth, default=None : The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

min_samples_leaf int or float, default=1 : The minimum number of samples required to be at a leaf node. A split point at any depth will only be considered if it leaves at least min_samples_leaf training samples in each of the left and right branches. This may have the effect of smoothing the model, especially in regression.

Random Forest Parameters

A random forest is a meta estimator that fits a number of decision tree classifiers on various sub-samples of the dataset and uses averaging to improve the predictive accuracy and control over-

fitting.

n_estimatorsint, default=100 : The number of trees in the forest.

max_depthint, default=None : The maximum depth of the tree. If None, then nodes are expanded until all leaves are pure or until all leaves contain less than min_samples_split samples.

max_features : The number of features to consider when looking for the best split.

min_impurity_decreasefloat, default=0.0 : Threshold for early stopping in tree growth. A node will be split if this split induces a decrease of the impurity greater than or equal to this value.

bootstrapbool, default=True : Whether bootstrap samples are used when building trees. If False, the whole dataset is used to build each tree.

Classification code

In [70]:

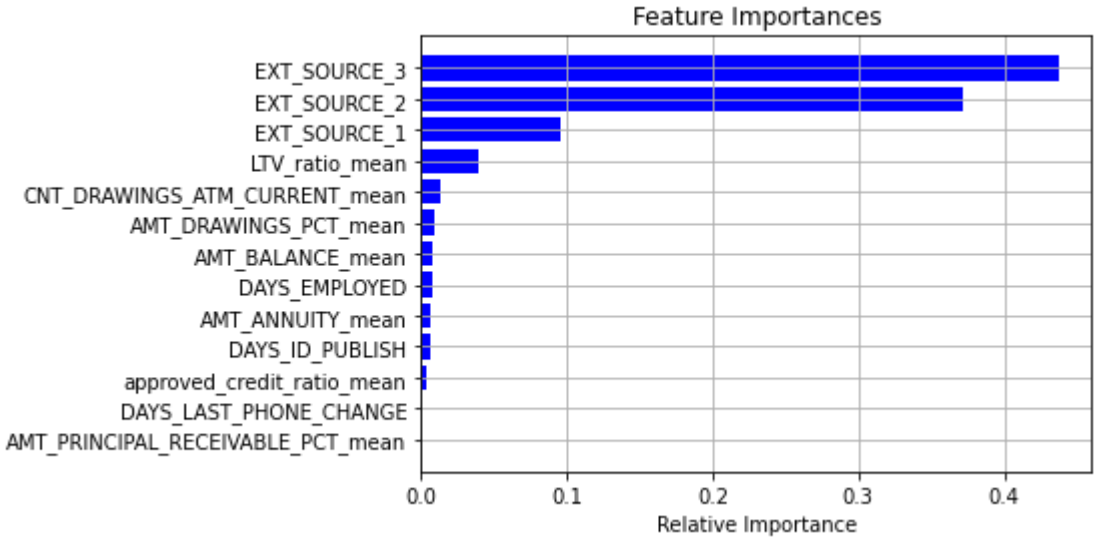
```
results = pd.DataFrame(columns=["ExpID", "Cross fold train accuracy", "Test Accuracy",
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
# A Function to execute the grid search and record the results.
def ConductGridSearch(X_train, y_train, X_test, y_test, i=0, prefix='', n_jobs=-1,verbo
    # Create a list of classifiers for our grid search experiment
    classifiers = [
        ('DecisionTrees', DecisionTreeClassifier(random_state=42)),
        ('RandomForest', RandomForestClassifier(random_state=42))

    ]

    # Arrange grid search parameters for each classifier
    params_grid = {
        'DecisionTrees' : {
            'criterion':['gini','entropy'],
            'max_depth':range(1,10),
            'min_samples_leaf':range(1,5)
        },

       'RandomForest':  {
            'max_depth': [15, 22],
            'max_features': [5, 7],
            'min_samples_leaf': [5, 10],
            'min_impurity_decrease':[0,1e-3,1e-4],
            'bootstrap': [True],
            'n_estimators':[150, 200]}
    }

    for (name, classifier) in classifiers:
        i += 1
        # Print classifier and parameters
        print('****** START',prefix, 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", FeatureUnion(transformer_list=[("num_pipeline", num_pipelin
    ("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, scoring='accur
                           n_jobs=n_jobs, verbose=verbose)
grid_search.fit(X_train, y_train)

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

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

features = num_attribs[:]
importances = grid_search.best_estimator_.named_steps["predictor"].feature_impo
# print(importances)
indices = np.argsort(importances)

plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='b', align='center')
plt.yticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.grid()
plt.show();
# Best estimator prediction time
start = time.time()
best_test_accuracy = pct(grid_search.best_estimator_.score(X_test, y_test))
test_time = round(time.time() - start, 4)

best_train_scores = cross_val_score(full_pipeline_with_predictor,X_train , y_tr
best_train_accuracy = pct(best_train_scores.mean())

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

# 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",prefix,name," *****")
print("")

# Record the results
results.loc[i] = [prefix+name, best_train_accuracy, best_test_accuracy, round(p
```

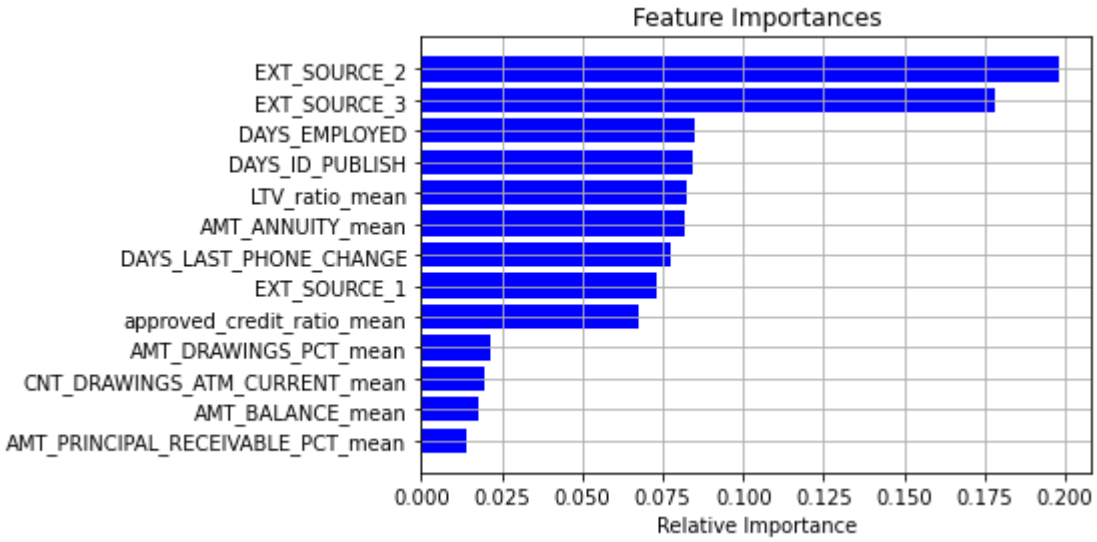
In [71]:

```
ConductGridSearch(X_train[num_attribs], y_train, X_test[num_attribs], y_test, 0, "Best
```

```
****** START Best Model: DecisionTrees *****
Parameters:
        criterion: ['gini', 'entropy']
        max_depth: range(1, 10)
        min_samples_leaf: range(1, 5)
Fitting 5 folds for each of 72 candidates, totalling 360 fits
```




```
Best Parameters:
        predictor__criterion: entropy
        predictor__max_depth: 5
        predictor__min_samples_leaf: 3
****** FINISH Best Model: DecisionTrees  *****

****** START Best Model: RandomForest *****
Parameters:
        bootstrap: [True]
        max_depth: [15, 22]
        max_features: [5, 7]
        min_impurity_decrease: [0, 0.001, 0.0001]
        min_samples_leaf: [5, 10]
        n_estimators: [150, 200]
Fitting 5 folds for each of 48 candidates, totalling 240 fits
```




```
Best Parameters:
        predictor__bootstrap: True
        predictor__max_depth: 15
        predictor__max_features: 5
        predictor__min_impurity_decrease: 0
        predictor__min_samples_leaf: 5
```

```
        predictor__n_estimators: 200
****** FINISH Best Model: RandomForest  *****
```

In [72]:
```
results
```

Out[72]:

	ExpID	Cross fold train accuracy	Test Accuracy	p-value	Train Time(s)	Test Time(s)	Experiment description
1	Best Model:DecisionTrees	58.450	66.063	0.00002	0.0384	0.0025	[["predictor__criterion", "entropy"], ["predic...
2	Best Model:RandomForest	67.347	67.601	0.15381	3.3169	0.1008	[["predictor__bootstrap", true], ["predictor__...

Conduct Grid Search using a variety of Classification Algorithms

In this section, we're going to find the best model by caomparing and change the hyperparameters in all models using GridSearchCV Note: These questions should be answered in Canvas once you have completed this section.

Best train accuracy

```
Please submit the code snippet you added to calculate the best train
accuracy in the section below of the notebook
```

Best parameters for Logistic Regression

```
Based on the results obtained after conducting Grid Search in this
section , choose the best parameters for Logistic Regression
```

Best parameters for k-nearest neighbors

```
Based on the results obtained after conducting Grid Search in this
section, what is  the best parameter for n_neighbors for k-nearest
neighbors?
```

SVM Test Accuracy

Please enter the calculated value for test Accuracy of Support vector model in the section below of the notebook. (Report your number to 1 decimal point of precision. For example: 2.5)

Statistical significance

```
Which one of the models listed below is the most statistically
significant based on the results of this section?
* Stochastic GD
```

* RandomForest
* Logistic Regression

Choosing the best model

Given the results that you obtained for the different models in this section, based on what information would you choose the best model to deploy?
* Cross fold Train Accuracy
* Test Accuracy
* p-value
* Train Time
* Test Time

In [73]:

```
results = pd.DataFrame(columns=["ExpID", "Cross fold train accuracy", "Test Accuracy",
# A Function to execute the grid search and record the results.
def ConductGridSearch(X_train, y_train, X_test, y_test, i=0, prefix='', n_jobs=-1,verbo
    # Create a list of classifiers for our grid search experiment
    classifiers = [
        ('Logistic Regression', LogisticRegression(random_state=42)),
        ('K-Nearest Neighbors', KNeighborsClassifier()),
        ('Support Vector', SVC(random_state=42)),
        ('Stochastic GD', SGDClassifier(random_state=42)),
        ('RandomForest', RandomForestClassifier(random_state=42)),
    ]

    # Arrange grid search parameters for each classifier
    params_grid = {
        'Logistic Regression': {
            'penalty': ('l1', 'l2'),
            'tol': (0.0001, 0.00001, 0.0000001),
            'C': (10, 1, 0.1, 0.01),
        },
        'K-Nearest Neighbors': {
            'n_neighbors': (3, 5, 7, 8, 11),
            'p': (1,2),
        },
        'Support Vector' : {
            'kernel': ('rbf', 'poly'),
            'degree': (1, 2, 3, 4, 5),
            'C': (10, 1, 0.1, 0.01),
        },
        'Stochastic GD': {
            'loss': ('hinge', 'perceptron', 'log'),
            'penalty': ('l1', 'l2', 'elasticnet'),
            'tol': (0.0001, 0.00001, 0.0000001),
            'alpha': (0.1, 0.01, 0.001, 0.0001),
        },
        'RandomForest':  {
            'max_depth': [9, 15, 22, 26, 30],
            'max_features': [1, 3, 5],
            'min_samples_split': [5, 10, 15],
            'min_samples_leaf': [3, 5, 10],
            'bootstrap': [False],
            'n_estimators':[20, 80, 150, 200, 300]},
    }
```

```
    for (name, classifier) in classifiers:
        i += 1
        # Print classifier and parameters
        print('****** START',prefix, 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, scoring='accur
                                   n_jobs=n_jobs, verbose=verbose)
        grid_search.fit(X_train, y_train)

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

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

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


        best_train_scores = cross_val_score(full_pipeline_with_predictor,X_train , y_tr
        best_train_accuracy = pct(best_train_scores.mean())


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

        # 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",prefix,name," *****")
        print("")

        # Record the results
        results.loc[i] = [prefix+name, best_train_accuracy, best_test_accuracy, round(p
```

In [74]:

```
ConductGridSearch(X_train, y_train, X_test, y_test, 0, "Best Model:",  n_jobs=-1,verbos
```

```
****** START Best Model: 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
```

```
/opt/conda/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: Convergenc
eWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: Convergenc
eWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: Convergenc
eWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: Convergenc
eWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: Convergenc
eWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: Convergenc
eWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: Convergenc
eWarning: lbfgs failed to converge (status=1):
```

```
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: Convergenc
eWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: Convergenc
eWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: Convergenc
eWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: Convergenc
eWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: Convergenc
eWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: Convergenc
eWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: Convergenc
eWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.
```

```
Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: Convergenc
eWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: Convergenc
eWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: Convergenc
eWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: Convergenc
eWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: Convergenc
eWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: Convergenc
eWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: Convergenc
eWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
```

```
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: Convergenc
eWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: Convergenc
eWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: Convergenc
eWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: Convergenc
eWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: Convergenc
eWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: Convergenc
eWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: Convergenc
eWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
```

```
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: Convergenc
eWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
/opt/conda/lib/python3.9/site-packages/sklearn/linear_model/_logistic.py:814: Convergenc
eWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regression
  n_iter_i = _check_optimize_result(
Best Parameters:
        predictor__C: 0.1
        predictor__penalty: l2
        predictor__tol: 0.0001
****** FINISH Best Model: Logistic Regression  *****

****** START Best Model: K-Nearest Neighbors *****
Parameters:
        n_neighbors: (3, 5, 7, 8, 11)
        p: (1, 2)
Fitting 5 folds for each of 10 candidates, totalling 50 fits
Best Parameters:
        predictor__n_neighbors: 11
        predictor__p: 2
****** FINISH Best Model: K-Nearest Neighbors  *****

****** START Best Model: Support Vector *****
Parameters:
        C: (10, 1, 0.1, 0.01)
        degree: (1, 2, 3, 4, 5)
        kernel: ('rbf', 'poly')
Fitting 5 folds for each of 40 candidates, totalling 200 fits
Best Parameters:
        predictor__C: 0.1
        predictor__degree: 1
        predictor__kernel: poly
****** FINISH Best Model: Support Vector  *****

****** START Best Model: Stochastic GD *****
Parameters:
        alpha: (0.1, 0.01, 0.001, 0.0001)
        loss: ('hinge', 'perceptron', 'log')
        penalty: ('l1', 'l2', 'elasticnet')
        tol: (0.0001, 1e-05, 1e-07)
Fitting 5 folds for each of 108 candidates, totalling 540 fits
Best Parameters:
        predictor__alpha: 0.01
        predictor__loss: hinge
        predictor__penalty: l2
        predictor__tol: 0.0001
****** FINISH Best Model: Stochastic GD  *****

****** START Best Model: RandomForest *****
Parameters:
        bootstrap: [False]
        max_depth: [9, 15, 22, 26, 30]
```

```
        max_features: [1, 3, 5]
        min_samples_leaf: [3, 5, 10]
        min_samples_split: [5, 10, 15]
        n_estimators: [20, 80, 150, 200, 300]
Fitting 5 folds for each of 675 candidates, totalling 3375 fits
Best Parameters:
        predictor__bootstrap: False
        predictor__max_depth: 15
        predictor__max_features: 5
        predictor__min_samples_leaf: 3
        predictor__min_samples_split: 15
        predictor__n_estimators: 300
****** FINISH Best Model: RandomForest  *****
```

In [75]:
```
results
```

Out[75]:

	ExpID	Cross fold train accuracy	Test Accuracy	p-value	Train Time(s)	Test Time(s)	Experiment description
1	Best Model:Logistic Regression	68.112	68.496	NaN	0.0753	0.0098	[["predictor__C", 0.1], ["predictor__penalty",...
2	Best Model:K-Nearest Neighbors	62.599	65.110	0.00000	0.0277	0.7243	[["predictor__n_neighbors", 11], ["predictor__...
3	Best Model:Support Vector	67.641	68.749	0.15379	2.2785	0.7592	[["predictor__C", 0.1], ["predictor__degree", ...
4	Best Model:Stochastic GD	66.533	68.749	0.03284	0.0469	0.0087	[["predictor__alpha", 0.01], ["predictor__loss...
5	Best Model:RandomForest	67.719	68.009	0.31118	3.1621	0.1551	[["predictor__bootstrap", false], ["predictor_...

Baseline Model - With sampled data

Create Train and Test Datasets

In [76]:
```
#X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train,stratify=y_trai

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: (10191, 21)
X validation      shape: (1799, 21)
X test            shape: (5139, 21)
X X_kaggle_test   shape: (48744, 21)
```

In [77]:
```
np.bincount(y_train)
```

Out[77]:
```
array([5768, 4423])
```

Concatenate our training data back together

In [78]:
```
train_data = pd.concat([X_train, y_train], axis=1)
train_data.head()
```

Out[78]:

	CNT_DRAWINGS_ATM_CURRENT_mean	AMT_DRAWINGS_PCT_mean	AMT_BALANCE_mean	LTV_rat
12451	1.636364	0.223560	93534.446250	
7770	0.285714	NaN	0.000000	
1414	0.696970	0.025971	173067.220125	
258	NaN	NaN	NaN	
921	NaN	0.000000	0.000000	

5 rows × 22 columns

In [79]:
```
from sklearn.utils import resample

# separate minority and majority classes
no_default_data = train_data[train_data.TARGET==0]
default_data = train_data[train_data.TARGET==1]

# sample minority
default_sampled_data = resample(default_data,
                           replace=True, # sample with replacement
                           n_samples=len(no_default_data), # match number in majority cl
                           random_state=42) # reproducible

# combine majority and upsampled minority
train_data = pd.concat([no_default_data, default_sampled_data])

train_data.TARGET.value_counts()
```

Out[79]:
```
0    5768
1    5768
Name: TARGET, dtype: int64
```

In [80]:
```
y_train = train_data['TARGET']
X_train = train_data[selected_features]
```

Create a Pipeline with Baseline Model

In [146…
```
%time
np.random.seed(42)
logistic_full_pipeline_with_predictor = Pipeline([
        ("preparation", data_prep_pipeline),
        ("linear", LogisticRegression(random_state=42, C = 0.1, penalty="l2", tol=0.000
    ])
random_forest_full_pipeline_with_predictor = Pipeline([
```

```
    ("preparation", data_prep_pipeline),
    ("random forest", RandomForestClassifier(random_state=42, bootstrap=False, max_dept
                                             min_samples_leaf=3, min_samples_split=15,
])

knn_full_pipeline_with_predictor = Pipeline([
    ("preparation", data_prep_pipeline),
    ("knn", KNeighborsClassifier(n_neighbors=11, p=2))
])

svm_full_pipeline_with_predictor = Pipeline([
    ("preparation", data_prep_pipeline),
    ("svm", SVC(random_state=42, C=0.1, degree=1, kernel="poly", probability=True))
])

sgd_full_pipeline_with_predictor = Pipeline([
    ("preparation", data_prep_pipeline),
    ("sgd", SGDClassifier(random_state=42, alpha=0.01, penalty="l2", tol=0.0001))
])
```

```
CPU times: user 2 µs, sys: 0 ns, total: 2 µs
Wall time: 4.53 µs
```

In [147…

```
# Create crossfold validation splits

cvSplits = ShuffleSplit(n_splits=15, test_size=0.3, random_state=0)
```

In [130…

```
# Baseline logistic regression Prediction
import time

start = time.time()
logit_model= logistic_full_pipeline_with_predictor.fit(X_train, y_train)
np.random.seed(42)

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

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

In [131…

```
# best knn Prediction
import time

start = time.time()
knn_model= knn_full_pipeline_with_predictor.fit(X_train, y_train)
np.random.seed(42)

# Set up cross validation scores
knn_scores = cross_val_score(knn_full_pipeline_with_predictor,X_train , y_train,cv=cvSp
knn_score_train = pct(knn_scores.mean())
train_time = np.round(time.time() - start, 4)
```

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

In [132…

```
# Best svm Prediction
import time

start = time.time()
svm_model= svm_full_pipeline_with_predictor.fit(X_train, y_train)
np.random.seed(42)

# Set up cross validation scores
svm_scores = cross_val_score(svm_full_pipeline_with_predictor,X_train , y_train,cv=cvSp
svm_score_train = pct(svm_scores.mean())
train_time = np.round(time.time() - start, 4)

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

In [148…

```
# Best Stochastic GD Prediction
import time

start = time.time()
sgd_model= sgd_full_pipeline_with_predictor.fit(X_train, y_train)
np.random.seed(42)

# Set up cross validation scores
sgd_scores = cross_val_score(sgd_full_pipeline_with_predictor,X_train , y_train,cv=cvSp
sgd_score_train = pct(sgd_scores.mean())
train_time = np.round(time.time() - start, 4)

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

In [134…

```
# Random Forest Prediction
import time

start = time.time()
rf_model= random_forest_full_pipeline_with_predictor.fit(X_train, y_train)
np.random.seed(42)

# Set up cross validation scores
rf_scores = cross_val_score(random_forest_full_pipeline_with_predictor,X_train , y_trai
rf_score_train = pct(rf_scores.mean())
train_time = np.round(time.time() - start, 4)

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

In [144…

```
# Baseline metrics
# Accuracy, AUC score, F1 Score and Log loss used for measuring the baseline model

exp_name = f"Logistic_{len(selected_features)}_features"
expLog.loc[len(expLog)] = [f"{exp_name}"] + list(np.round(
               [logit_score_train,
                pct(accuracy_score(y_valid, logit_model.predict(X_valid))),
                pct(accuracy_score(y_test, logit_model.predict(X_test))),
                roc_auc_score(y_train, logit_model.predict_proba(X_train)[:, 1]),
                roc_auc_score(y_valid, logit_model.predict_proba(X_valid)[:, 1]),
                roc_auc_score(y_test, logit_model.predict_proba(X_test)[:, 1]),
                f1_score(y_train, logit_model.predict(X_train)),
                f1_score(y_test, logit_model.predict(X_test)),
                log_loss(y_train, logit_model.predict(X_train)),
                log_loss(y_test, logit_model.predict(X_test)),0 ],4)) \
                + [train_time,test_time] + [f"Untuned Balanced Logistic reg features {t

exp_name = f"KNN_{len(selected_features)}_features"
expLog.loc[len(expLog)] = [f"{exp_name}"] + list(np.round(
               [logit_score_train,
                pct(accuracy_score(y_valid, knn_model.predict(X_valid))),
                pct(accuracy_score(y_test, knn_model.predict(X_test))),
                roc_auc_score(y_train, knn_model.predict_proba(X_train)[:, 1]),
                roc_auc_score(y_valid, knn_model.predict_proba(X_valid)[:, 1]),
                roc_auc_score(y_test, knn_model.predict_proba(X_test)[:, 1]),
                f1_score(y_train, knn_model.predict(X_train)),
                f1_score(y_test, knn_model.predict(X_test)),
                log_loss(y_train, knn_model.predict(X_train)),
                log_loss(y_test, knn_model.predict(X_test)),0 ],4)) \
                + [train_time,test_time] + [f"Untuned Balanced KNN reg features {tot_fe

exp_name = f"SVM_{len(selected_features)}_features"
expLog.loc[len(expLog)] = [f"{exp_name}"] + list(np.round(
               [logit_score_train,
                pct(accuracy_score(y_valid, svm_model.predict(X_valid))),
                pct(accuracy_score(y_test, svm_model.predict(X_test))),
                roc_auc_score(y_train, svm_model.predict_proba(X_train)[:, 1]),
                roc_auc_score(y_valid, svm_model.predict_proba(X_valid)[:, 1]),
                roc_auc_score(y_test, svm_model.predict_proba(X_test)[:, 1]),
                f1_score(y_train, svm_model.predict(X_train)),
                f1_score(y_test, svm_model.predict(X_test)),
                log_loss(y_train, svm_model.predict(X_train)),
                log_loss(y_test, svm_model.predict(X_test)),0 ],4)) \
                + [train_time,test_time] + [f"Untuned Balanced SVM reg features {tot_fe

# exp_name = f"SGD_{len(selected_features)}_features"
# expLog.loc[len(expLog)] = [f"{exp_name}"] + list(np.round(
#                [logit_score_train,
#                 accuracy_score(y_valid, sgd_model.predict(X_valid)),
#                 accuracy_score(y_test, sgd_model.predict(X_test)), "NaN", "NaN", "NaN
#                 f1_score(y_train, sgd_model.predict(X_train)),
#                 f1_score(y_test, sgd_model.predict(X_test)),
#                 log_loss(y_train, sgd_model.predict(X_train)),
#                 log_loss(y_test, sgd_model.predict(X_test)),0 ],4)) \
#                 + [train_time,test_time] + [f"Untuned Balanced SGD reg features {tot_

exp_name = f"Random_forest_{len(selected_features)}_features"
expLog.loc[len(expLog)] = [f"{exp_name}"] + list(np.round(
               [rf_score_train,
```

```
                pct(accuracy_score(y_valid, rf_model.predict(X_valid))),
                pct(accuracy_score(y_test, rf_model.predict(X_test))),
                roc_auc_score(y_train, rf_model.predict_proba(X_train)[:, 1]),
                roc_auc_score(y_valid, rf_model.predict_proba(X_valid)[:, 1]),
                roc_auc_score(y_test, rf_model.predict_proba(X_test)[:, 1]),
                f1_score(y_train, rf_model.predict(X_train)),
                f1_score(y_test, rf_model.predict(X_test)),
                log_loss(y_train, rf_model.predict(X_train)),
                log_loss(y_test, rf_model.predict(X_test)),0 ],4)) \
                + [train_time,test_time] + [f"Untuned Balanced Random Forest features {
expLog
```

Out[144...

	exp_name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC	Train F1 Score	Test F1 Score	Train Log Loss	
0	Logistic_21_features	68.082	68.538	66.842	0.7468	0.7393	0.7388	0.6822	0.6376	10.9372	11.4
1	KNN_21_features	68.082	63.257	63.047	0.8123	0.6776	0.6715	0.7406	0.5922	9.2576	12.7
2	SVM_21_features	68.082	68.760	66.900	0.7455	0.7393	0.7397	0.6833	0.6357	10.8893	11.4
3	Random_forest_21_features	76.596	68.205	67.990	0.9796	0.7360	0.7384	0.9240	0.6075	2.6407	11.0

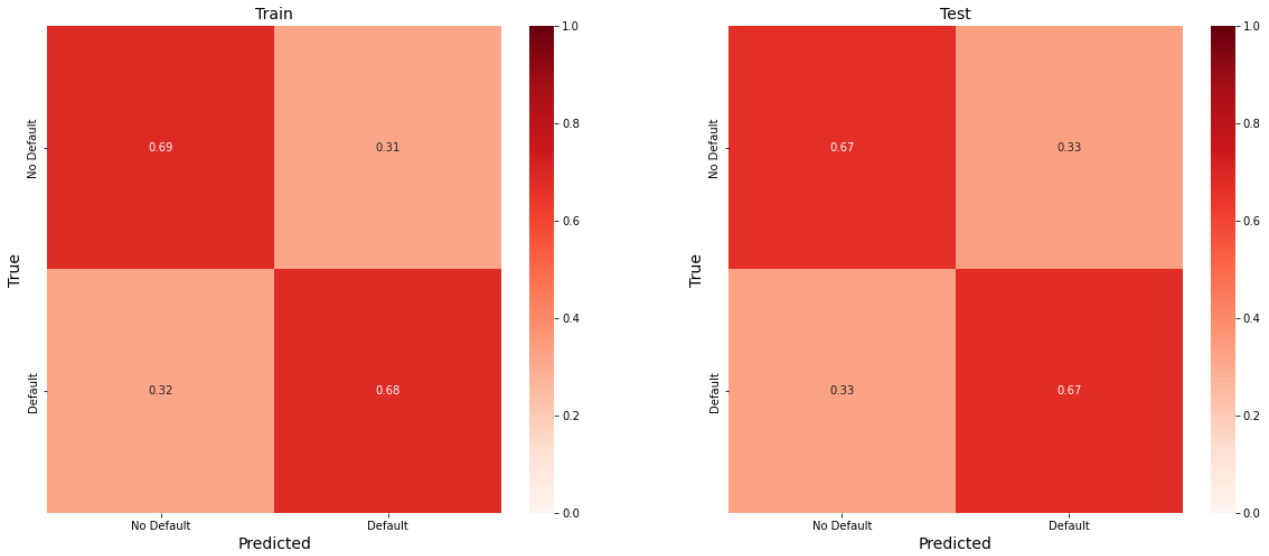
In [ ]:

```
# Baseline metrics
# Accuracy, AUC score, F1 Score and Log loss used for measuring the baseline model

exp_name = f"Random_forest_{len(selected_features)}_features"
expLog.loc[len(expLog)] = [f"{exp_name}"] + list(np.round(
               [rf_score_train,
                pct(accuracy_score(y_valid, rf_model.predict(X_valid))),
                pct(accuracy_score(y_test, rf_model.predict(X_test))),
                roc_auc_score(y_train, rf_model.predict_proba(X_train)[:, 1]),
                roc_auc_score(y_valid, rf_model.predict_proba(X_valid)[:, 1]),
                roc_auc_score(y_test, rf_model.predict_proba(X_test)[:, 1]),
                f1_score(y_train, rf_model.predict(X_train)),
                f1_score(y_test, rf_model.predict(X_test)),
                log_loss(y_train, rf_model.predict(X_train)),
                log_loss(y_test, rf_model.predict(X_test)),0 ],4)) \
```

```
                + [train_time,test_time] + [f"Untuned Balanced Logistic reg features {t
expLog
```

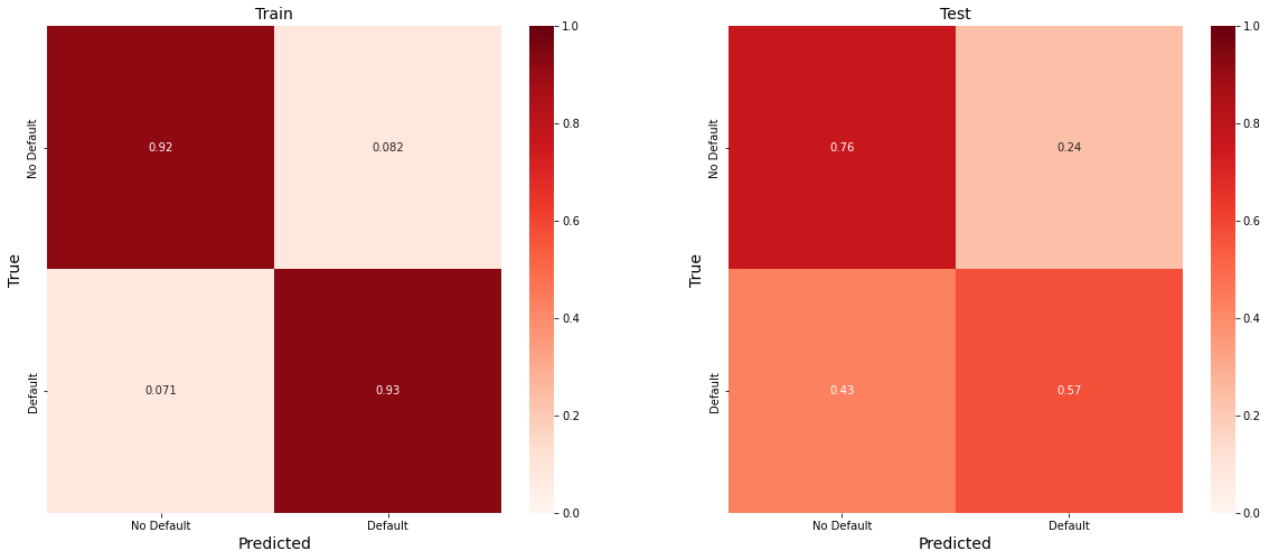
In [108…

```
# Confusion matrix

# Create confusion matrix for baseline model
confusion_matrix_def(logit_model,X_train,y_train,X_test,y_test)
```



In [109…

```
# Confusion matrix

# Create confusion matrix for Random forest model
confusion_matrix_def(rf_model,X_train,y_train,X_test,y_test)
```



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 [151…

```
rf_test_class_scores = rf_model.predict_proba(X_kaggle_test)[:, 1]
logit_test_class_scores = logit_model.predict_proba(X_kaggle_test)[:, 1]
svm_test_class_scores = svm_model.predict_proba(X_kaggle_test)[:, 1]
knn_test_class_scores = knn_model.predict_proba(X_kaggle_test)[:, 1]
```

In [152…

```
print(X_kaggle_test.shape)
print(len(rf_test_class_scores))
print(len(logit_test_class_scores))
print(len(svm_test_class_scores))
print(len(knn_test_class_scores))
print(X_kaggle_test.head())
```

```
(48744, 21)
48744
48744
48744
48744
   CNT_DRAWINGS_ATM_CURRENT_mean  AMT_DRAWINGS_PCT_mean  AMT_BALANCE_mean  \
0                            NaN                    NaN               NaN
1                            NaN                    NaN               NaN
2                       0.255556               0.037798      18159.919219
3                       0.045455               0.027362       8085.058163
4                            NaN                    NaN               NaN

   LTV_ratio_mean  AMT_PRINCIPAL_RECEIVABLE_PCT_mean  DAYS_LAST_PHONE_CHANGE  \
0        0.957782                                NaN                 -1740.0
1        0.899950                                NaN                     0.0
2        1.052363                           0.632973                 -856.0
3        0.967479                           0.911662                -1805.0
4        1.131358                                NaN                 -821.0

   DAYS_ID_PUBLISH  EXT_SOURCE_3  EXT_SOURCE_2  EXT_SOURCE_1  ...  \
0             -812      0.159520      0.789654      0.752614  ...
1            -1623      0.432962      0.291656      0.564990  ...
2            -3503      0.610991      0.699787           NaN  ...
3            -4208      0.612704      0.509677      0.525734  ...
4            -4262           NaN      0.425687      0.202145  ...

   approved_credit_ratio_mean  AMT_ANNUITY_mean  CODE_GENDER FLAG_OWN_REALTY  \
0                    1.044079          3951.000            F               Y
1                    1.111173          4813.200            M               Y
2                    0.956503         11478.195            M               Y
3                    0.777028          8091.585            F               Y
4                    0.884003         17782.155            M               N

  FLAG_OWN_CAR NAME_CONTRACT_TYPE            NAME_EDUCATION_TYPE  \
0            N         Cash loans               Higher education
1            N         Cash loans  Secondary / secondary special
2            Y         Cash loans               Higher education
3            N         Cash loans  Secondary / secondary special
4            Y         Cash loans  Secondary / secondary special

      OCCUPATION_TYPE NAME_INCOME_TYPE SK_ID_CURR
0                 NaN          Working     100001
```

```
1  Low-skill Laborers          Working     100005
2             Drivers          Working     100013
3         Sales staff          Working     100028
4                 NaN          Working     100038

[5 rows x 21 columns]
```

In [112…
```
test_class_scores[0:10]
```

Out[112…
```
array([0.27916713, 0.54058272, 0.23359758, 0.19467377, 0.64605331,
       0.40543823, 0.20686865, 0.25578426, 0.10672724, 0.40684109])
```

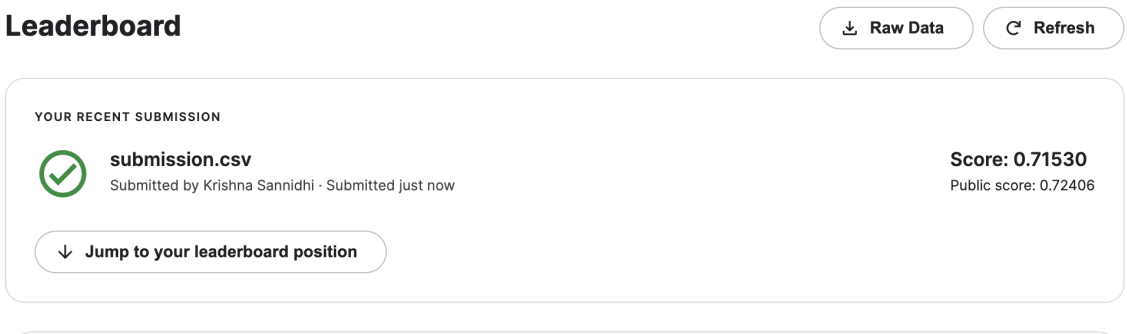
In [153…
```
# Submission dataframe
rf_submit_df = X_kaggle_test[['SK_ID_CURR']]
rf_submit_df['TARGET'] = rf_test_class_scores

logit_submit_df = X_kaggle_test[['SK_ID_CURR']]
logit_submit_df['TARGET'] = logit_test_class_scores

svm_submit_df = X_kaggle_test[['SK_ID_CURR']]
svm_submit_df['TARGET'] = svm_test_class_scores

knn_submit_df = X_kaggle_test[['SK_ID_CURR']]
knn_submit_df['TARGET'] = knn_test_class_scores
```

In [154…
```
rf_submit_df.to_csv("submission_rf.csv",index=False)
logit_submit_df.to_csv("submission_logit.csv",index=False)
svm_submit_df.to_csv("submission_svm.csv",index=False)
knn_submit_df.to_csv("submission_knn.csv",index=False)
```

Results/Discussions

For Logistic regression,

- We have achieved an accuracy 0.919 and an AUC score of 0.735
- Kaggle score: 0.71530
- Screenshot:



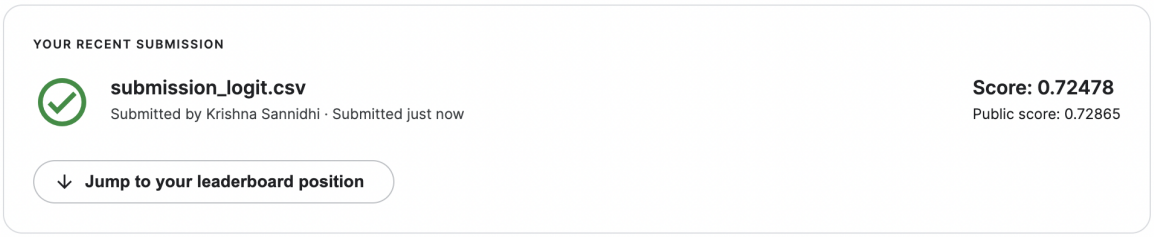
Initially, we have tried building our base line model with Dummy Classifier which results 83.4% accurancy to improve accurancy based on the datasets EDA we built Logistic Regression Model which resulted in around 92% accurancy score after the kaggle submission we obtained the score of 0.71530. So we consider this model as better fit for baseline model. And we also observed that data

is more towards target 0 value than 1. To get better accurancy we have to try by undersampling the data.

After Feature engineering and feature importance

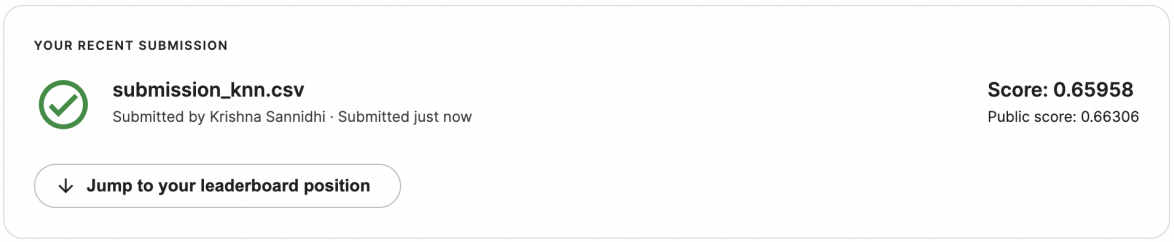
For Logistic regression,

- Test ACU: 0.7388
- Kaggle score: 0.72865
- Screenshot:



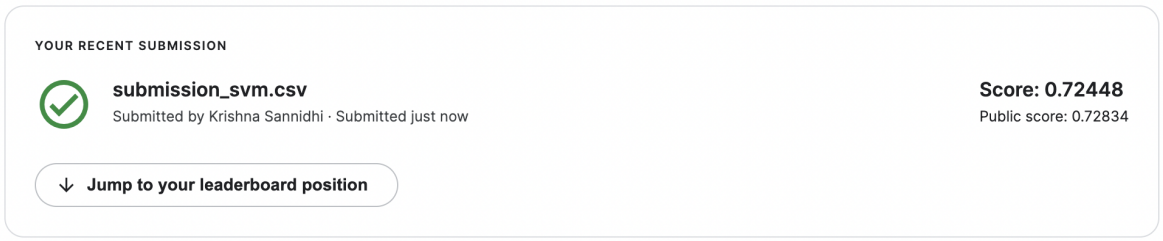
For K-nearest neighbour,

- Test ACU: 0.6715
- Kaggle score: 0.65958
- Screenshot:



For Support Vector Machine,

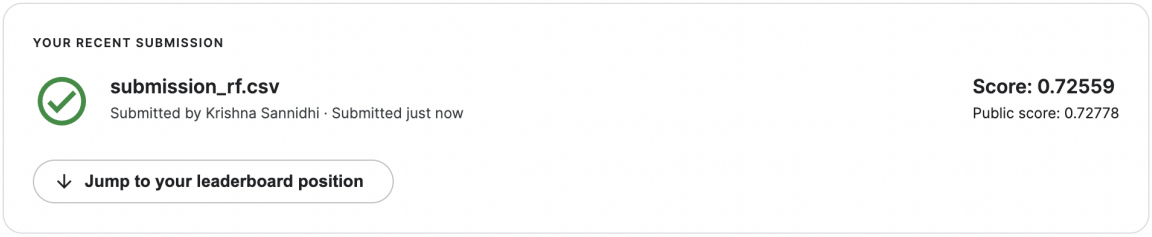
- Test ACU: 0.7397
- Kaggle scoe: 0.72448
- Screenshoot:



For Random Forest,

- Test ACU: 0.7384
- kaggle score: 0.72559

- Screenshot:



Conclusion

The primary goal of this project is to build a Machine Learning model that can predict a person will be able to repay the loan that is granted. Our project is to predict if a person can repay their loan based on many factors like their previous application history, monthly balances of previous credits, credit card balance etc. This model aids in choosing the applicants if they can repay the loan based of variety of factors considered.

The logistic regression model has been chosen as baseline model to predict whether or not a person can repay the loan. For logistic regression, As we found concrete data variance , we have performed up sampling and down sampling. The logistic regression achieved an accuracy of 92%. We have also submitted this on kaggle which got us a score of 0.71530

Additionally, we have also done feature engineering. To find the best fit, We will experiments other machine learning models and chose the one which gives the most accurate results.

Our Project focuses on different ways to check if a person is eligible for a loan and if can repay it. This is very important so that no person will be rejected from getting a loan based on just credit score. It is fair to consider other factors to assess one's repayment capability. Custom features like each person's bill payment frequency, Days employed, mean of balance maintained,Loan taken in terms of price of goods purchased etc when employed with different machine learning models like Support Vector Machine, K-Nearest Neighbors, Decision Trees, Random Forest and Logistic Regression help us reach the goal we aimed for.

In phase 2, We performed feature engineering, hyper parameter tuning, feature importance analysis, feature selection, ensemble methods, Formed a pipeline model and tested it on Support Vector Machine, K-Nearest Neighbors, Random Forest and Logistic Regression. Tested all the completed phases on the above five machine learning models.

Accuracies and AUC scores for .

1. Support Vector Machine:
    - Accuracy: 66.900
    - Error Rate:11.4324
    - AUC Score: 0. 7397
2. K-Nearest Neighbors:

    - Accuracy: 63.047

- Error Rate: 12.7632
- AUC Score: 0.6715

3. Random Forest:

- Accuracy: 67.990
- Error Rate: 2.6407
- AUC Score: 0.9796

4. Logistic Regression:

- Accuracy:66.842
- Error Rate: 10.9372
- AUC Score: 0.7393

Among the above models, Random Forest fit our data well and achieved a less error rate of 2.6407

In future, We will build a multilayer perceptron model a multi-headed load default system using the OOP API in PyTorch

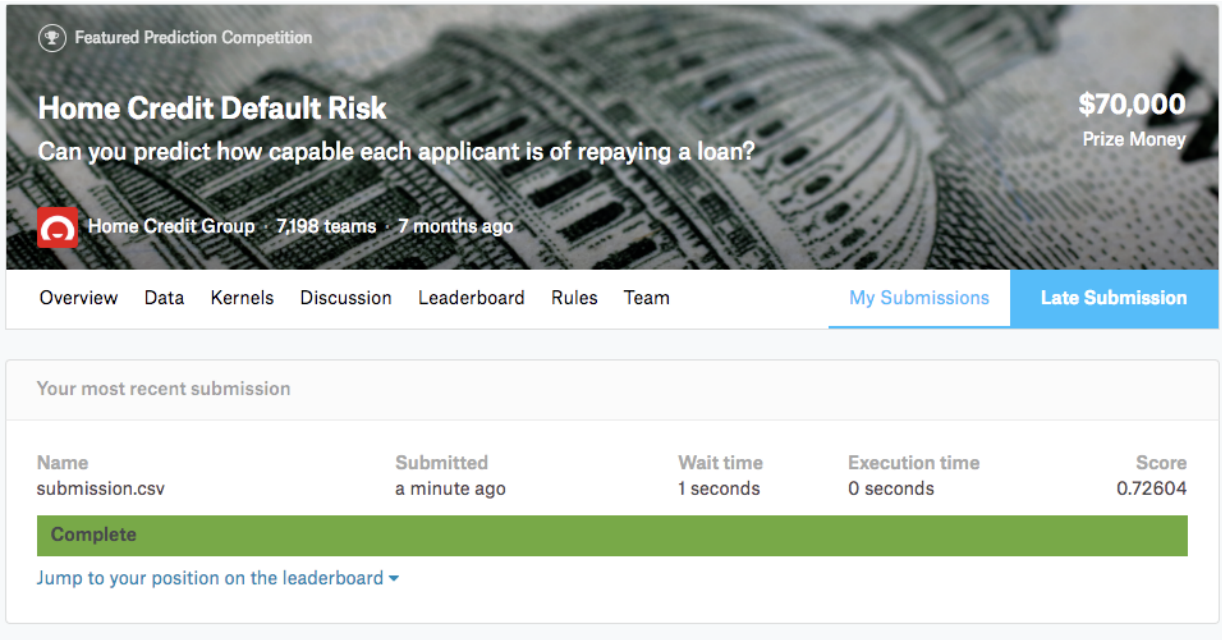
Kaggle submission via the command line API

In [86]:
```
! kaggle competitions submit -c home-credit-default-risk -f submission.csv -m "baseline
```

```
Traceback (most recent call last):
  File "/Users/ankitapriya/opt/anaconda3/bin/kaggle", line 5, in <module>
    from kaggle.cli import main
  File "/Users/ankitapriya/opt/anaconda3/lib/python3.8/site-packages/kaggle/__init__.p
y", line 23, in <module>
    api.authenticate()
  File "/Users/ankitapriya/opt/anaconda3/lib/python3.8/site-packages/kaggle/api/kaggle_a
pi_extended.py", line 164, in authenticate
    raise IOError('Could not find {}. Make sure it\'s located in'
OSError: Could not find kaggle.json. Make sure it's located in /Users/ankitapriya/.kaggl
e. Or use the environment method.
```

report submission

Click on this link



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/

In [ ]: