

Team Information

Project Title - 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.

Group Number - 21

Names -

Kumar Saurabh (ksaurabh@iu.edu)

Shubham Thakur (sbmthakur@iu.edu)

Ameya Dalvi (abdalvi@iu.edu)

Vishwa Shrirame (vshriram@iu.edu)

Team Photos



Kumar Saurabh



Shubham Thakur



Ameya Dalvi



Vishwa Shrirame

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 you 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
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: kaggle in /usr/local/lib/python3.7/site-packages (1.5.12)
Requirement already satisfied: requests in /usr/local/lib/python3.7/site-packages (from kaggle) (2.25.1)
Requirement already satisfied: certifi in /usr/local/lib/python3.7/site-packages (from kaggle) (2021.5.30)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.7/site-packages (from kaggle) (2.8.2)
Requirement already satisfied: tqdm in /usr/local/lib/python3.7/site-packages (from kaggle) (4.62.1)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.7/site-packages (from kaggle) (1.15.0)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.7/site-packages (from kaggle) (1.26.6)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.7/site-packages (from kaggle) (5.0.2)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.7/site-packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/site-packages (from requests->kaggle) (2.10)
Requirement already satisfied: chardet<5,>=3.0.2 in /usr/local/lib/python3.7/site-packages (from requests->kaggle) (4.0.0)
WARNING: You are using pip version 21.2.4; however, version 21.3.1 is available.
You should consider upgrading via the '/usr/local/bin/python -m pip install --upgrade pip' command.
```

```
In [6]: !pwd
/N/home/u100/vshriram/Carbonate/I526_AML_Student/Assignments/Unit-Project-Home-Credit-Default-Risk

In [10]: !mkdir ~/.kaggle
#!cp /N/u/vshriram/Carbonate/Downloads/kaggle.json ~/.kaggle
!chmod 600 ~/.kaggle/kaggle.json

mkdir: cannot create directory '/N/u/vshriram/Carbonate/.kaggle': File exists

In [11]: ! kaggle competitions files home-credit-default-risk
```

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

Dataset and how to download

Back ground Home Credit Group

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

Home Credit Group

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

While Home Credit is currently using various statistical and machine learning methods to make these predictions, they're challenging Kagglers to help them unlock the full potential of their data. Doing so will ensure that clients capable of repayment are not rejected and that loans are given with a principal, maturity, and repayment calendar that will empower their clients to be successful.

Background on the dataset

Home Credit is a non-banking financial institution, founded in 1997 in the Czech Republic.

The company operates in 14 countries (including United States, Russia, Kazakhstan, 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 untrustworthy lenders.

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

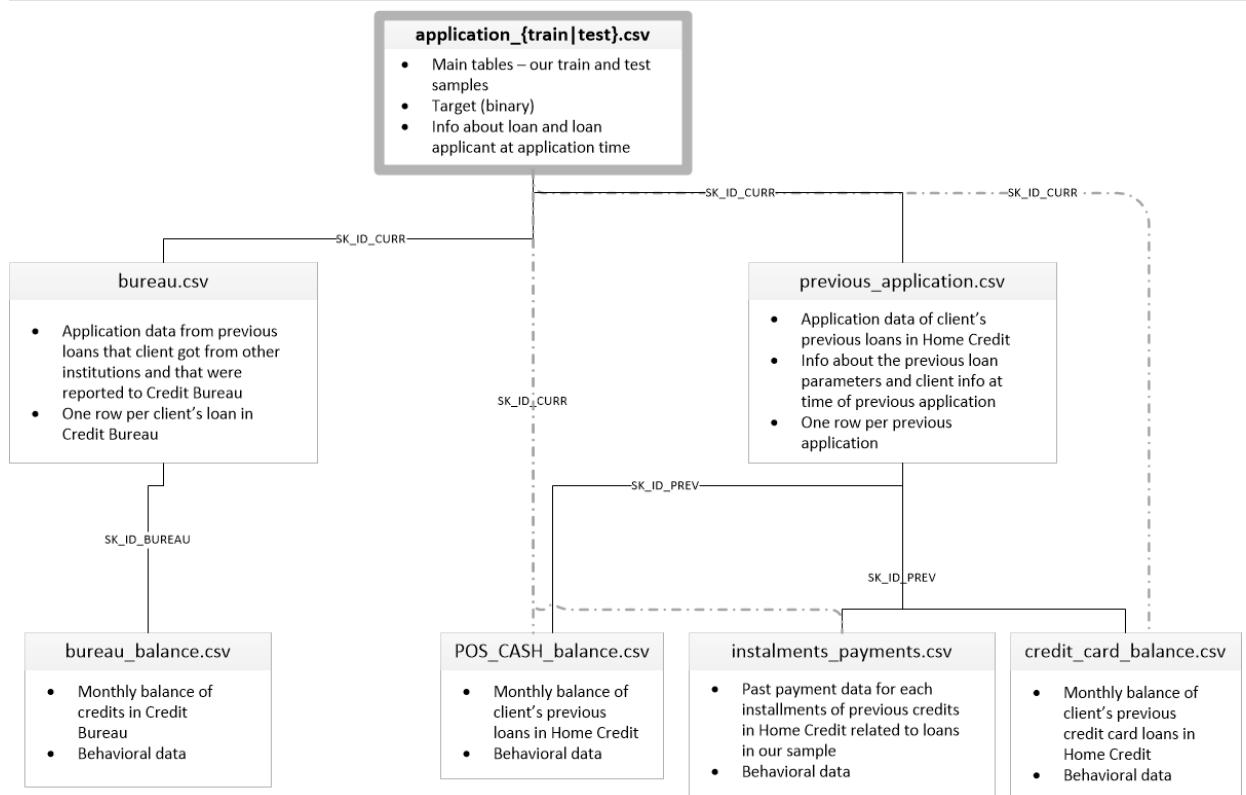
There are 7 different sources of data:

- **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 [ ]: # ! [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 = "/N/u/vshriram/Carbonate/I526_AML_Student/Assignments/Unit-Project  
#DATA_DIR = os.path.join('./ddddd/')  
!mkdir $DATA_DIR
```

```
mkdir: cannot create directory '/N/u/vshriram/Carbonate/I526_AML_Student/As  
signments/Unit-Project-Home-Credit-Default-Risk': File exists
```

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

```
total 3338400  
-rw-rw-r-- 1 vshriram vshriram 11666135 Dec  5 17:31 Group21_Phase1.ipynb  
drwxrwxr-x 3 vshriram vshriram 32768 Nov 29 20:47 HCDR_Phase_1_baseline  
_submission  
-rw-rw-r-- 1 vshriram vshriram 37383 Dec 11 2019 HomeCredit_columns_de  
scription.csv  
-rw-rw-r-- 1 vshriram vshriram 392703158 Dec 11 2019 POS_CASH_balance.csv  
-rw-rw-r-- 1 vshriram vshriram 26567651 Dec 11 2019 application_test.csv  
-rw-rw-r-- 1 vshriram vshriram 166133370 Dec 11 2019 application_train.csv  
-rw-rw-r-- 1 vshriram vshriram 170016717 Dec 11 2019 bureau.csv  
-rw-rw-r-- 1 vshriram vshriram 375592889 Dec 11 2019 bureau_balance.csv  
-rw-rw-r-- 1 vshriram vshriram 424582605 Dec 11 2019 credit_card_balance.c  
sv  
-rw-r--r-- 1 vshriram vshriram 721616255 Dec  4 11:40 home-credit-default-r  
isk.zip  
-rw-rw-r-- 1 vshriram vshriram 723118349 Dec 11 2019 installments_payments  
.csv  
-rw-rw-r-- 1 vshriram vshriram 404973293 Dec 11 2019 previous_application.  
csv  
-rw-rw-r-- 1 vshriram vshriram 536202 Dec 11 2019 sample_submission.csv
```

```
In [14]: ! kaggle competitions download home-credit-default-risk -p $DATA_DIR
```

```
Downloading home-credit-default-risk.zip to /N/u/vshriram/Carbonate/I526_AM  
L_Student/Assignments/Unit-Project-Home-Credit-Default-Risk  
99%|██████████| 682M/688M [00:06<00:00, 140  
MB/s]  
100%|██████████| 688M/688M [00:06<00:00, 115  
MB/s]
```

```
In [1]: pip install missingno
```

```
Defaulting to user installation because normal site-packages is not writeable
Requirement already satisfied: missingno in /N/home/u100/vshriram/Carbonate/.local/lib/python3.7/site-packages (0.5.0)
Requirement already satisfied: scipy in /usr/local/lib/python3.7/site-packages (from missingno) (1.6.2)
Requirement already satisfied: numpy in /usr/local/lib/python3.7/site-packages (from missingno) (1.19.5)
Requirement already satisfied: matplotlib in /usr/local/lib/python3.7/site-packages (from missingno) (3.4.2)
Requirement already satisfied: seaborn in /usr/local/lib/python3.7/site-packages (from missingno) (0.11.2)
Requirement already satisfied: pillow>=6.2.0 in /usr/local/lib/python3.7/site-packages (from matplotlib->missingno) (8.3.1)
Requirement already satisfied: kiwisolver>=1.0.1 in /usr/local/lib/python3.7/site-packages (from matplotlib->missingno) (1.3.1)
Requirement already satisfied: pyparsing>=2.2.1 in /usr/local/lib/python3.7/site-packages (from matplotlib->missingno) (2.4.7)
Requirement already satisfied: cycler>=0.10 in /usr/local/lib/python3.7/site-packages (from matplotlib->missingno) (0.10.0)
Requirement already satisfied: python-dateutil>=2.7 in /usr/local/lib/python3.7/site-packages (from matplotlib->missingno) (2.8.2)
Requirement already satisfied: six in /usr/local/lib/python3.7/site-packages (from cycler>=0.10->matplotlib->missingno) (1.15.0)
Requirement already satisfied: pandas>=0.23 in /usr/local/lib/python3.7/site-packages (from seaborn->missingno) (1.3.2)
Requirement already satisfied: pytz>=2017.3 in /usr/local/lib/python3.7/site-packages (from pandas>=0.23->seaborn->missingno) (2021.1)
WARNING: You are using pip version 21.2.4; however, version 21.3.1 is available.
You should consider upgrading via the '/usr/local/bin/python -m pip install --upgrade pip' command.
Note: you may need to restart the kernel to use updated packages.
```

Imports

In [2]:

```
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, make_pipeline
from pandas.plotting import scatter_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
import missingno as msno
from sklearn.metrics import mean_squared_error
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.compose import ColumnTransformer, make_column_transformer
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score
import warnings
warnings.filterwarnings('ignore')
```

In [7]:

```
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`

The screenshot shows an Excel spreadsheet titled "HomeCredit_columns_description". The table has columns labeled "Table", "Row", "Description", and "Special". Rows 1 through 38 contain various application-related fields and their meanings. Row 1 is a header row. Row 2: "1 application_SK_ID_CURR ID of loan in our sample". Row 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)". Row 4: "5 application_NAME_CONT Identification if loan is cash or revolving". Row 5: "6 application_GEND Gender of the client". Row 6: "7 application_FLAG_OWN Flag if the client owns a car". Row 7: "8 application_FLAG_OWN Flag if client owns a house or flat". Row 8: "9 application_CNT_CHILD Number of children the client has". Row 9: "10 application_AMT_INCOM Income of the client". Row 10: "11 application_AMT_CREDIT Credit amount of the loan". Row 11: "12 application_AMT_ANNUI Loan annuity". Row 12: "13 application_AMT_GOODI For consumer loans it is the price of the goods for which the loan is given". Row 13: "14 application_NAME_TYPE Who was accompanying client when he was applying for the loan". Row 14: "15 application_NAME_INCO Clients income type (businessman, working, maternity leave,0)". Row 15: "16 application_NAME_EDUC Level of highest education the client achieved". Row 16: "17 application_NAME_FAMI Family status of the client". Row 17: "18 application_NAME_HOU What is the housing situation of the client (renting, living with parents, ...)" . Row 18: "19 application_REGION_POI Normalized | normalized". Row 19: "20 application_DAYS_BIRTH Client's age | time only relative to the application". Row 20: "21 application_DAYS_EMPL How many d time only relative to the application". Row 21: "22 application_DAYS_REGIS How many d time only relative to the application". Row 22: "23 application_DAYS_ID_PU How many d time only relative to the application". Row 23: "24 application_OWNER_CAR_Age of client's car". Row 24: "25 application_FLAG_MOBIL Did client provide mobile phone (1=YES, 0=NO)". Row 25: "26 application_FLAG_EMP_Did client provide work phone (1=YES, 0=NO)". Row 26: "27 application_FLAG_WORL Did client provide home phone (1=YES, 0=NO)". Row 27: "28 application_FLAG_CONT Was mobile phone reachable (1=YES, 0=NO)". Row 28: "29 application_FLAG_PHON Did client provide home phone (1=YES, 0=NO)". Row 29: "30 application_FLAG_EMAIL Did client provide email (1=YES, 0=NO)". Row 30: "31 application_OCCUPATIO! What kind of occupation does the client have". Row 31: "32 application_CNT_FAM_N How many family members does client have". Row 32: "33 application_REGION_RA Our rating of the region where client lives (1,2,3)". Row 33: "34 application_REGION_RA Our rating of the region where client lives with taking city into account (1,2,3)". Row 34: "35 application_WEEKDAY_A On which day of the week did the client apply for the loan". Row 35: "36 application_HOUR_APPR Approximate rounded". Row 36: "37 application_REG_REGION Flag if client's permanent address does not match contact address (1=different, 0=same, at region level)". Row 37: "38 application_REG_REGIO! Flag if client's permanent address does not match work address (1=different, 0=same, at region level)". Row 38: "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
ds_name = 'application_train'
datasets[ds_name] = load_data(os.path.join(DATA_DIR, f'{ds_name}.csv'), ds_name)

datasets['application_train'].shape
```

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

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

5 rows × 122 columns

Out[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_
```

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

5 rows × 121 columns

The application dataset has the most information about the client: Gender, income, family status, education ...

The Other datasets

- **bureau:** data concerning client's previous credits from other financial institutions. Each previous credit has its own row in bureau, but one loan in the application data can have multiple previous credits.
- **bureau_balance:** monthly data about the previous credits in bureau. Each row is one month of a previous credit, and a single previous credit can have multiple rows, one for each month of the credit length.
- **previous_application:** previous applications for loans at Home Credit of clients who have loans in the application data. Each current loan in the application data can have multiple previous loans. Each previous application has one row and is identified by the feature SK_ID_PREV.
- **POS_CASH_BALANCE:** monthly data about previous point of sale or cash loans clients have had with Home Credit. Each row is one month of a previous point of sale or cash loan, and a single previous loan can have many rows.
- **credit_card_balance:** monthly data about previous credit cards clients have had with Home Credit. Each row is one month of a credit card balance, and a single credit card can have many rows.
- **installments_payment:** payment history for previous loans at Home Credit. There is one row for every made payment and one row for every missed payment.

In [5]:

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

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

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

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

5 rows × 122 columns

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

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

5 rows × 121 columns

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

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

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

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

```
credit_card_balance: shape is (3840312, 23)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3840312 entries, 0 to 3840311
Data columns (total 23 columns):
 #   Column           Dtype  
 --- 
 0   SK_ID_PREV      int64  
 1   SK_ID_CURR      int64  
 2   MONTHS_BALANCE int64  
 3   AMT_BALANCE     float64 
 4   AMT_CREDIT_LIMIT_ACTUAL int64  
 5   AMT_DRAWINGS_ATM_CURRENT float64 
 6   AMT_DRAWINGS_CURRENT  float64 
 7   AMT_DRAWINGS_OTHER_CURRENT float64 
 8   AMT_DRAWINGS_POS_CURRENT float64 
 9   AMT_INST_MIN_REGULARITY float64 
 10  AMT_PAYMENT_CURRENT float64 
 11  AMT_PAYMENT_TOTAL_CURRENT float64 
 12  AMT_RECEIVABLE_PRINCIPAL float64 
 13  AMT_RECVABLE      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_ACT
0	2562384	378907		-6	56.970
1	2582071	363914		-1	63975.555
2	1740877	371185		-7	31815.225
3	1389973	337855		-4	236572.110
4	1891521	126868		-1	453919.455

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	D.
0	1054186	161674		1.0	6
1	1330831	151639		0.0	34
2	2085231	193053		2.0	1
3	2452527	199697		1.0	3
4	2714724	167756		1.0	2

```

previous_application: shape is (1670214, 37)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1670214 entries, 0 to 1670213
Data columns (total 37 columns):
 #   Column           Non-Null Count   Dtype  
--- 
 0   SK_ID_PREV      1670214 non-null   int64  
 1   SK_ID_CURR      1670214 non-null   int64  
 2   NAME_CONTRACT_TYPE 1670214 non-null   object  
 3   AMT_ANNUITY     1297979 non-null   float64 
 4   AMT_APPLICATION 1670214 non-null   float64 
 5   AMT_CREDIT       1670213 non-null   float64 
 6   AMT_DOWN_PAYMENT 774370 non-null   float64 
 7   AMT_GOODS_PRICE  1284699 non-null   float64 
 8   WEEKDAY_APPR_PROCESS_START 1670214 non-null   object  
 9   HOUR_APPR_PROCESS_START  1670214 non-null   int64  
 10  FLAG_LAST_APPL_PER_CONTRACT 1670214 non-null   object  
 11  NFLAG_LAST_APPL_IN_DAY    1670214 non-null   int64  
 12  RATE_DOWN_PAYMENT     774370 non-null   float64 
 13  RATE_INTEREST_PRIMARY 5951 non-null   float64 
 14  RATE_INTEREST_PRIVILEGED 5951 non-null   float64 
 15  NAME_CASH_LOAN_PURPOSE 1670214 non-null   object  
 16  NAME_CONTRACT_STATUS  1670214 non-null   object  
 17  DAYS_DECISION      1670214 non-null   int64  
 18  NAME_PAYMENT_TYPE   1670214 non-null   object  
 19  CODE_REJECT_REASON  1670214 non-null   object  
 20  NAME_TYPE_SUITE     849809 non-null   object  
 21  NAME_CLIENT_TYPE   1670214 non-null   object  
 22  NAME_GOODS_CATEGORY 1670214 non-null   object  
 23  NAME_PORTFOLIO      1670214 non-null   object  
 24  NAME_PRODUCT_TYPE   1670214 non-null   object  
 25  CHANNEL_TYPE        1670214 non-null   object  
 26  SELLERPLACE_AREA    1670214 non-null   int64  
 27  NAME_SELLER_INDUSTRY 1670214 non-null   object  
 28  CNT_PAYMENT        1297984 non-null   float64 
 29  NAME_YIELD_GROUP   1670214 non-null   object  
 30  PRODUCT_COMBINATION 1669868 non-null   object  
 31  DAYS_FIRST_DRAWING 997149 non-null   float64 
 32  DAYS_FIRST_DUE     997149 non-null   float64 
 33  DAYS_LAST_DUE_1ST_VERSION 997149 non-null   float64 
 34  DAYS_LAST_DUE      997149 non-null   float64 
 35  DAYS_TERMINATION   997149 non-null   float64 
 36  NFLAG_INSURED_ON_APPROVAL 997149 non-null   float64 

dtypes: float64(15), int64(6), object(16)
memory usage: 471.5+ MB
None

```

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

5 rows × 37 columns

```

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

```

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

CPU times: user 36.3 s, sys: 7.7 s, total: 44 s
Wall time: 44 s

In [14]:

```

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

```

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

Exploratory Data Analysis

Summary of Application train

In [8]:

```
datasets["application_train"].info()
```

```

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Columns: 122 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(41), object(16)
memory usage: 286.2+ MB

```

```
In [9]: datasets["application_train"].describe() #numerical only features
```

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

8 rows × 106 columns

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

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY
count	48744.000000	48744.000000	4.874400e+04	4.874400e+04	48720.000000
mean	277796.676350	0.397054	1.784318e+05	5.167404e+05	29426.240205
std	103169.547296	0.709047	1.015226e+05	3.653970e+05	16016.368315
min	100001.000000	0.000000	2.694150e+04	4.500000e+04	2295.000000
25%	188557.750000	0.000000	1.125000e+05	2.606400e+05	17973.000000
50%	277549.000000	0.000000	1.575000e+05	4.500000e+05	26199.000000
75%	367555.500000	1.000000	2.250000e+05	6.750000e+05	37390.500000
max	456250.000000	20.000000	4.410000e+06	2.245500e+06	180576.000000

8 rows × 105 columns

```
In [17]: datasets["application_train"].describe(include='all') #look at all categories
```

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

11 rows × 122 columns

In [18]: `datasets["application_train"].size`

Out[18]: 37516342

In [19]: `datasets["application_train"].shape`

Out[19]: (307511, 122)

Dividing the train dataset into train and validation sets

```
In [20]: x = datasets["application_train"].drop(['TARGET'], axis = 1)
y = datasets["application_train"]["TARGET"]

# Split the provided training data into training and validation and test
X_train, X_test, y_train, y_test = train_test_split(x, y, test_size=0.20, random_state=42)
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.25, random_state=42)
print(f"X train           shape: {X_train.shape}")
print(f"X validation     shape: {X_valid.shape}")
print(f"X test            shape: {X_test.shape}")
```

X train shape: (196806, 121)
X validation shape: (49202, 121)
X test shape: (61503, 121)

Summary Statistics -

In [21]:

```
x_train.info()
x_train.describe()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 196806 entries, 9717 to 255
Columns: 121 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(40), object(16)
memory usage: 183.2+ MB
```

Out[21]:

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY
count	196806.000000	196806.000000	1.968060e+05	1.968060e+05	196798.000000
mean	278195.549368	0.416786	1.683316e+05	5.993323e+05	27109.915949
std	102732.472419	0.719989	1.055828e+05	4.029388e+05	14475.618420
min	100003.000000	0.000000	2.565000e+04	4.500000e+04	1993.500000
25%	189184.250000	0.000000	1.125000e+05	2.700000e+05	16524.000000
50%	278235.500000	0.000000	1.440000e+05	5.147775e+05	24907.500000
75%	367111.750000	1.000000	2.025000e+05	8.086500e+05	34609.500000
max	456254.000000	19.000000	1.350000e+07	4.050000e+06	258025.500000

8 rows × 105 columns

In [22]:

```
# Concatenating X_train and y_train
Xy_train = pd.concat([X_train, y_train], axis=1)
Xy_train.head()
```

Out[22]:

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OW...
9717	111307	Cash loans	F	N	
203356	335752	Cash loans	F	Y	
81757	194805	Cash loans	F	Y	
84860	198457	Cash loans	F	N	
234668	371838	Cash loans	F	N	

5 rows × 122 columns

Determine the categorical and numerical features

In [23]:

```
numerical_features = X_train.select_dtypes(include = ['int64', 'float64'])
categorical_features = X_train.select_dtypes(include = ['object', 'bool'])
print(f"\nNumerical features : {list(numerical_features)}")
print(f"\nCategorical features : {list(categorical_features)}")
```

Numerical features : ['SK_ID_CURR', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE', 'REGION_POPULATION_RELATIVE', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'DAYS_ID_PUBLISH', 'OWN_CAR_AGE', 'FLAG_MOBIL', 'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE', 'FLAG_CONT_MOBILE', 'FLAG_PHONE', 'FLAG_EMAIL', 'CNT_FAM_MEMBERS', 'REGION_RATING_CLIENT', 'REGION_RATING_CLIENT_W_CITY', 'HOUR_APPR_PROCESS_START', 'REG_REGION_NOT_LIVE_REGION', 'REG_REGION_NOT_WORK_REGION', 'LIVE_REGION_NOT_WORK_REGION', 'REG_CITY_NOT_LIVE_CITY', 'REG_CITY_NOT_WORK_CITY', 'LIVE_CITY_NOT_WORK_CITY', 'EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3', 'APARTMENTS_AVG', 'BASEMENTAREA_AVG', 'YEARS_BEGINEXPLUATATION_AVG', 'YEARS_BUILD_AVG', 'COMMONAREA_AVG', 'ELEVATORS_AVG', 'ENTRANCES_AVG', 'FLOORSMAX_AVG', 'FLOORSMIN_AVG', 'LANDAREA_AVG', 'LIVINGAPARTMENTS_AVG', 'LIVINGAREA_AVG', 'NONLIVINGAPARTMENTS_AVG', 'NONLIVINGAREA_AVG', 'APARTMENTS_MODE', 'BASEMENTAREA_MODE', 'YEARS_BEGINEXPLUATATION_MODE', 'YEARS_BUILD_MODE', 'COMMONAREA_MODE', 'ELEVATORS_MODE', 'ENTRANCES_MODE', 'FLOORSMAX_MODE', 'FLOORSMIN_MODE', 'LANDAREA_MODE', 'LIVINGAPARTMENTS_MODE', 'LIVINGAREA_MODE', 'NONLIVINGAPARTMENTS_MODE', 'NONLIVINGAREA_MODE', 'APARTMENTS_MEDI', 'BASEMENTAREA_MEDI', 'YEARS_BEGINEXPLUATATION_MEDI', 'YEARS_BUILD_MEDI', 'COMMONAREA_MEDI', 'ELEVATORS_MEDI', 'ENTRANCES_MEDI', 'FLOORSMAX_MEDI', 'FLOORSMIN_MEDI', 'LANDAREA_MEDI', 'LIVINGAPARTMENTS_MEDI', 'LIVINGAREA_MEDI', 'NONLIVINGAPARTMENTS_MEDI', 'NONLIVINGAREA_MEDI', 'TOTALAREA_MODE', 'OBS_30_CNT_SOCIAL_CIRCLE', 'DEF_30_CNT_SOCIAL_CIRCLE', 'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIRCLE', 'DAYS_LAST_PHONE_CHANGE', 'FLAG_DOCUMENT_2', 'FLAG_DOCUMENT_3', 'FLAG_DOCUMENT_4', 'FLAG_DOCUMENT_5', 'FLAG_DOCUMENT_6', 'FLAG_DOCUMENT_7', 'FLAG_DOCUMENT_8', 'FLAG_DOCUMENT_9', 'FLAG_DOCUMENT_10', 'FLAG_DOCUMENT_11', 'FLAG_DOCUMENT_12', 'FLAG_DOCUMENT_13', 'FLAG_DOCUMENT_14', 'FLAG_DOCUMENT_15', 'FLAG_DOCUMENT_16', 'FLAG_DOCUMENT_17', 'FLAG_DOCUMENT_18', 'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20', 'FLAG_DOCUMENT_21', 'AMT_REQ_CREDIT_BUREAU_HOUR', 'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK', 'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QRT', 'AMT_REQ_CREDIT_BUREAU_YEAR']

Categorical features : ['NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'NAME_TYPE_SUITE', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'OCCUPATION_TYPE', 'WEEKDAY_APPR_PROCESS_START', 'ORGANIZATION_TYPE', 'FONDKAPREMONT_MODE', 'HOUSEYPE_MODE', 'WALLSMATERIAL_MODE', 'EMERGENCYSTATE_MODE']

Missing data Analysis

Missing data for application train

In [24]:

```
percent = (datasets["application_train"].isnull().sum()/datasets["application_train"].shape[0]).sort_values(ascending=True)
sum_missing = datasets["application_train"].isna().sum().sort_values(ascending=True)
missing_application_train_data = pd.concat([percent, sum_missing], axis=1)
missing_application_train_data.head(20)
```

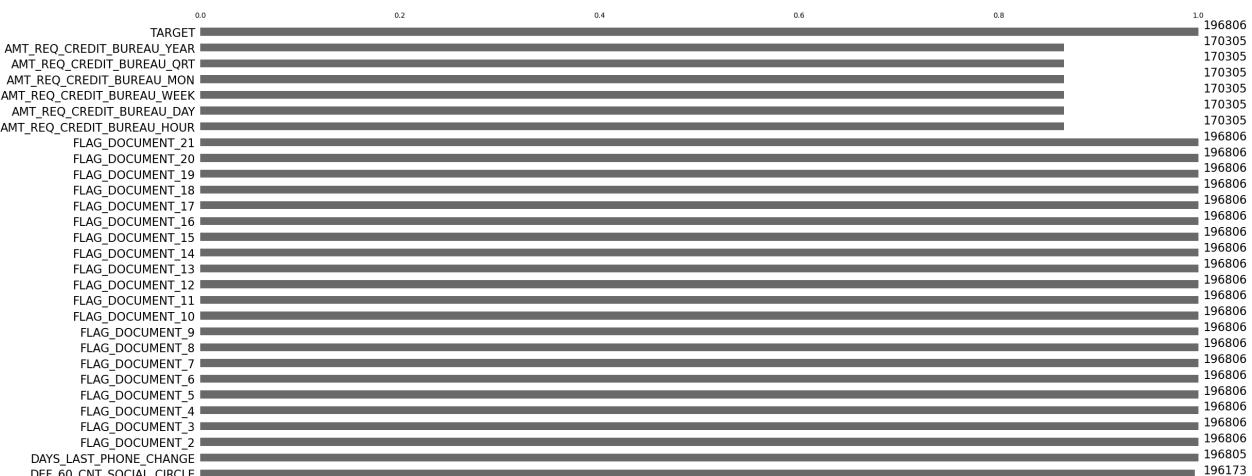
Out[24]:

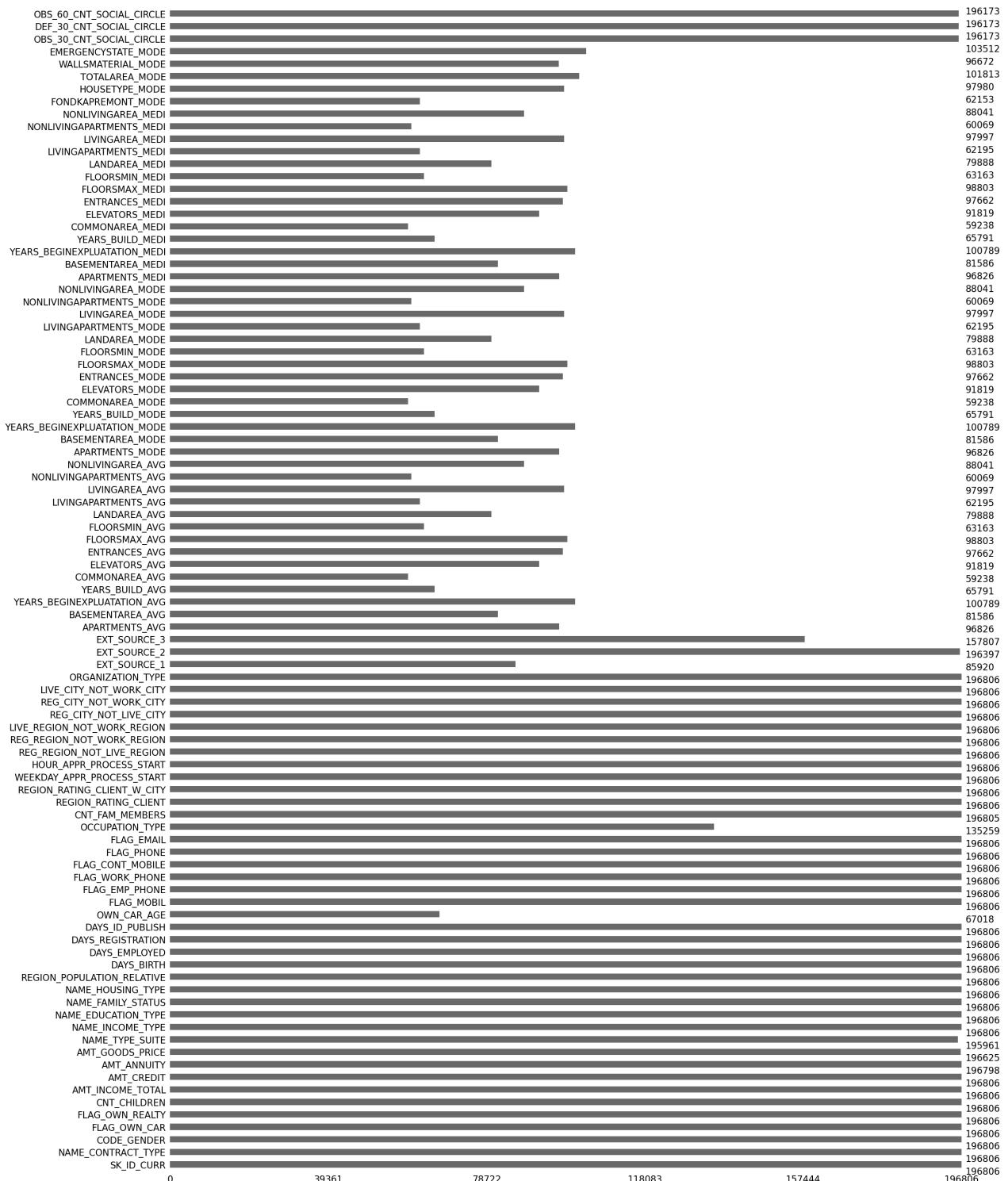
	Percent	Train Missing Count
COMMONAREA_MEDI	69.87	214865
COMMONAREA_AVG	69.87	214865
COMMONAREA_MODE	69.87	214865
NONLIVINGAPARTMENTS_MODE	69.43	213514
NONLIVINGAPARTMENTS_AVG	69.43	213514
NONLIVINGAPARTMENTS_MEDI	69.43	213514
FONDKAPREMONT_MODE	68.39	210295
LIVINGAPARTMENTS_MODE	68.35	210199
LIVINGAPARTMENTS_AVG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

Bar plot of missing values in each column

In [25]:

```
msno.bar(Xy_train)
plt.show()
```





In [26]:

`Xy_train.isnull().sum()`

```
Out[26]: SK_ID_CURR          0
          NAME_CONTRACT_TYPE    0
          CODE_GENDER           0
          FLAG_OWN_CAR          0
          FLAG_OWN_REALTY        0
                           ...
          AMT_REQ_CREDIT_BUREAU_WEEK 26501
          AMT_REQ_CREDIT_BUREAU_MON   26501
          AMT_REQ_CREDIT_BUREAU_QRT   26501
          AMT_REQ_CREDIT_BUREAU_YEAR  26501
          TARGET                  0
Length: 122, dtype: int64
```

We notice there are a lot of missing values in the dataset

```
In [27]: missing_application_train_data
```

	Percent	Train Missing Count
COMMONAREA_MEDI	69.87	214865
COMMONAREA_AVG	69.87	214865
COMMONAREA_MODE	69.87	214865
NONLIVINGAPARTMENTS_MODE	69.43	213514
NONLIVINGAPARTMENTS_AVG	69.43	213514
...
NAME_HOUSING_TYPE	0.00	0
NAME_FAMILY_STATUS	0.00	0
NAME_EDUCATION_TYPE	0.00	0
NAME_INCOME_TYPE	0.00	0
SK_ID_CURR	0.00	0

122 rows × 2 columns

```
In [28]: percent_data = missing_application_train_data.iloc[:,0]
missing_application_train_data['Percent']
percent_data_df = pd.DataFrame(missing_application_train_data['Percent'],
```

```
In [29]: percent_data_df = percent_data_df[percent_data_df['Percent'] != 0.0]
percent_data_df
```

Out[29]:

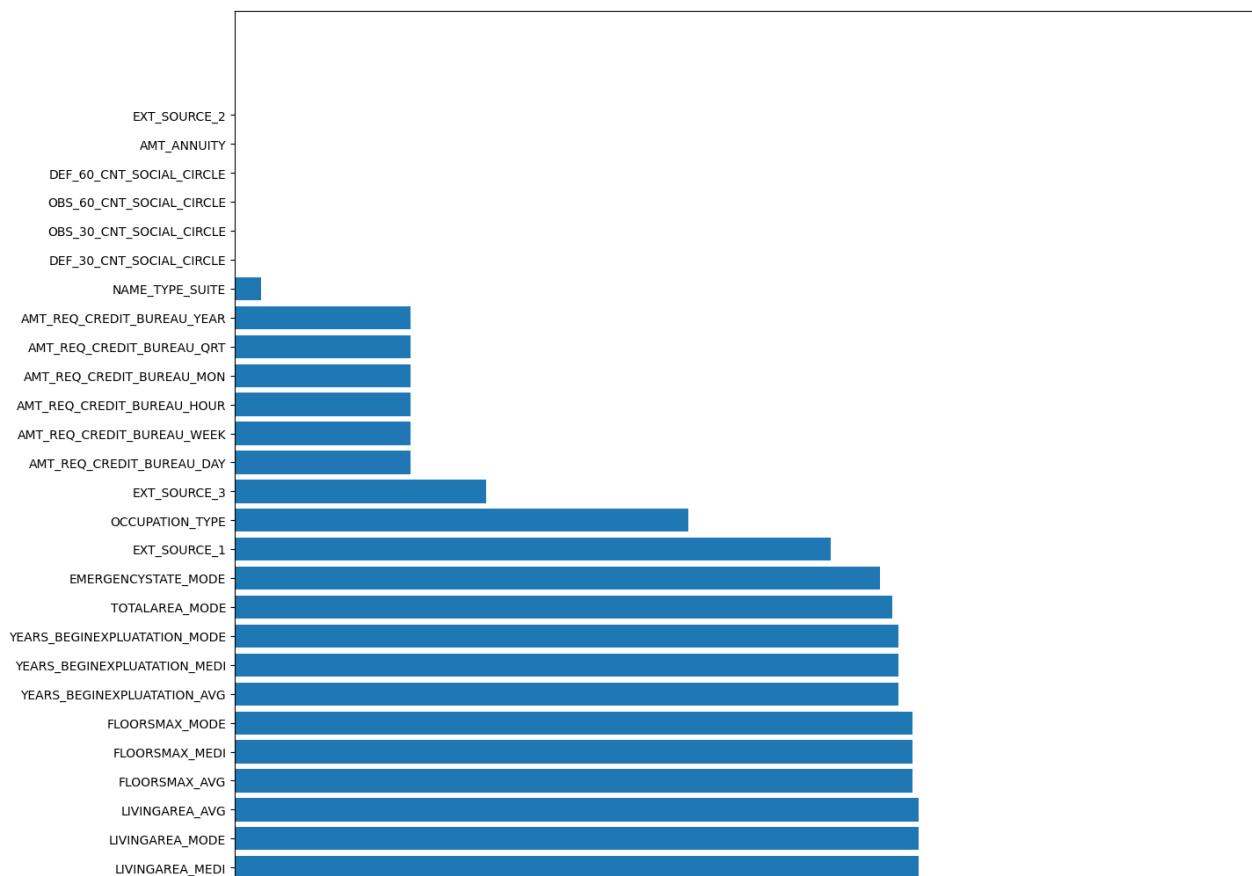
	Percent
COMMONAREA_MEDI	69.87
COMMONAREA_AVG	69.87
COMMONAREA_MODE	69.87
NONLIVINGAPARTMENTS_MODE	69.43
NONLIVINGAPARTMENTS_AVG	69.43
...	...
DEF_30_CNT_SOCIAL_CIRCLE	0.33
OBS_60_CNT_SOCIAL_CIRCLE	0.33
DEF_60_CNT_SOCIAL_CIRCLE	0.33
EXT_SOURCE_2	0.21
AMT_GOODS_PRICE	0.09

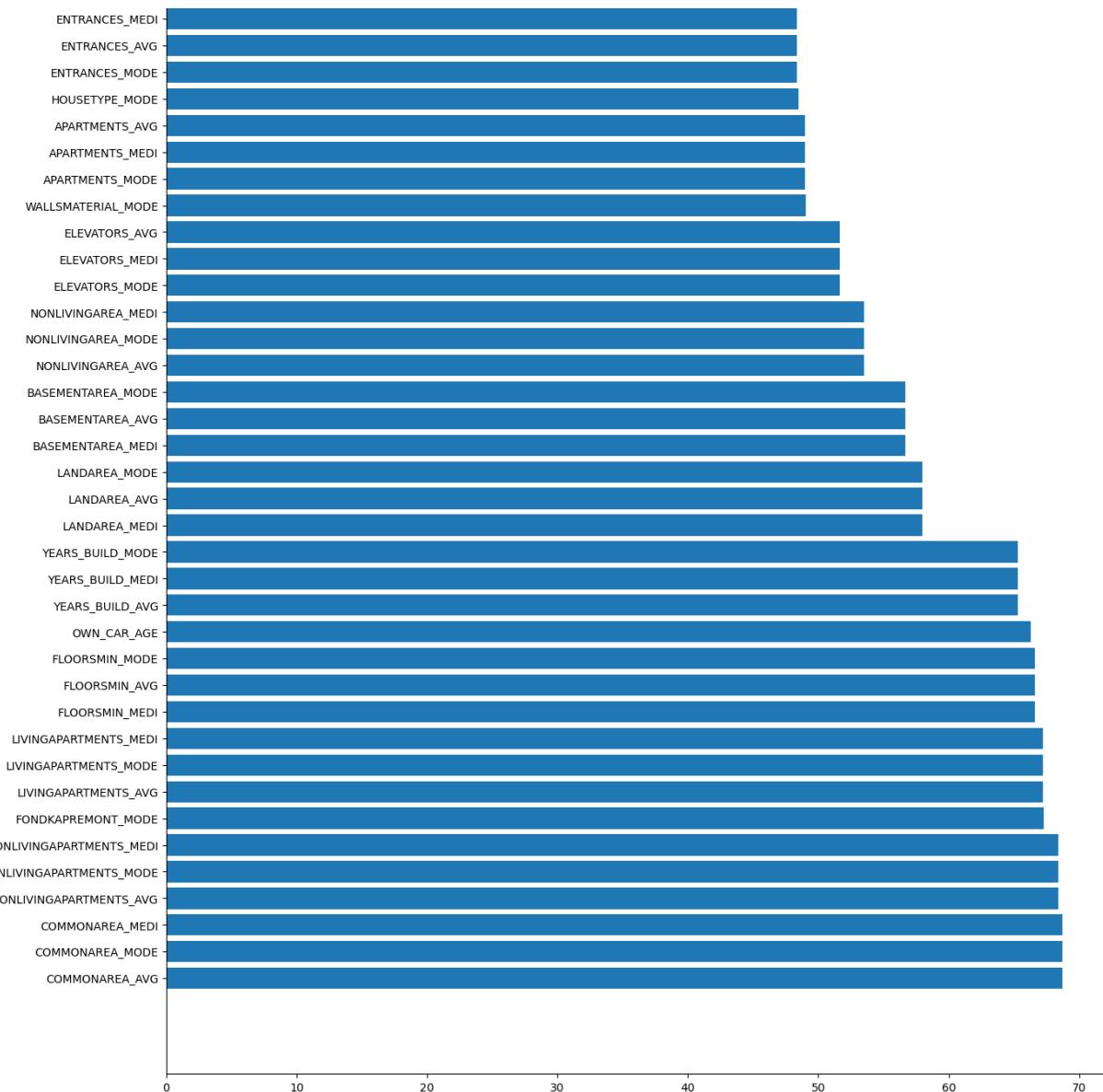
64 rows × 1 columns

Bar plot of the percentage of missing values in each column

In [23]:

```
plt.figure(figsize = (15,30))
plt.barh(y = percent_data_df.index, width = percent_data_df['Percent'])
plt.show()
```





Imputing Missing data

We will construct the numerical pipeline and categorical pipeline

In [28]:

```
num_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('imputer', SimpleImputer(strategy = 'median'))
])

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

In [29]:

```
data_pipeline = ColumnTransformer([
    ("num_pipeline", num_pipeline, numerical_features),
    ("cat_pipeline", cat_pipeline, categorical_features)], remainder = 'drop')

X_train_transformed = data_pipeline.fit_transform(X_train)

column_names = list(numerical_features) + \
    list(data_pipeline.transformers_[1][1].named_steps["ohe"].get_feature_names_out())

display(pd.DataFrame(X_train_transformed, columns=column_names).head())
```

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT...
0	-1.624501	-0.578880	-0.528796	1.188441	0.175508	
1	0.560257	0.810033	0.749824	1.304588	1.218472	
2	-0.811727	0.810033	0.110514	-0.452448	-0.310068	
3	-0.776179	-0.578880	0.749824	1.188441	0.529898	
4	0.911520	0.810033	-0.315693	0.559617	-0.197845	

5 rows × 245 columns

In [30]:

```
X_train_transformed_df = pd.DataFrame(X_train_transformed, columns=column_names)
X_train_transformed_df
```

Out[30]:

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT...
0	-1.624501	-0.578880	-0.528796	1.188441	0.175508	
1	0.560257	0.810033	0.749824	1.304588	1.218472	
2	-0.811727	0.810033	0.110514	-0.452448	-0.310068	
3	-0.776179	-0.578880	0.749824	1.188441	0.529898	
4	0.911520	0.810033	-0.315693	0.559617	-0.197845	
...
196801	0.105580	2.198946	0.962928	1.748359	0.769578	
196802	-0.647728	-0.578880	0.195756	2.349464	1.072985	
196803	0.569894	0.810033	0.110514	-0.705648	-0.784764	
196804	0.743462	2.198946	-0.315693	-0.640427	-1.242051	
196805	-1.731692	0.810033	0.536721	1.042029	0.270945	

196806 rows × 245 columns

```
In [31]: x_train_transformed_df.isnull().sum()
```

```
Out[31]: SK_ID_CURR          0
CNT_CHILDREN          0
AMT_INCOME_TOTAL       0
AMT_CREDIT             0
AMT_ANNUITY            0
...
WALLSMATERIAL_MODE_Panel 0
WALLSMATERIAL_MODE_Stone, brick 0
WALLSMATERIAL_MODE_Wooden 0
EMERGENCYSTATE_MODE_No   0
EMERGENCYSTATE_MODE_Yes  0
Length: 245, dtype: int64
```

Correlation Analysis

Correlation with the target column

```
In [32]: correlations = datasets["application_train"].corr()['TARGET'].sort_values()
print('Most Positive Correlations:\n', correlations.tail(10))
print('\nMost Negative Correlations:\n', correlations.head(10))
```

Most Positive Correlations:

FLAG_DOCUMENT_3	0.044346
REG_CITY_NOT_LIVE_CITY	0.044395
FLAG_EMP_PHONE	0.045982
REG_CITY_NOT_WORK_CITY	0.050994
DAYS_ID_PUBLISH	0.051457
DAYS_LAST_PHONE_CHANGE	0.055218
REGION_RATING_CLIENT	0.058899
REGION_RATING_CLIENT_W_CITY	0.060893
DAYS_BIRTH	0.078239
TARGET	1.000000

Name: TARGET, dtype: float64

Most Negative Correlations:

EXT_SOURCE_3	-0.178919
EXT_SOURCE_2	-0.160472
EXT_SOURCE_1	-0.155317
DAYS_EMPLOYED	-0.044932
FLOORSMAX_AVG	-0.044003
FLOORSMAX_MEDI	-0.043768
FLOORSMAX_MODE	-0.043226
AMT_GOODS_PRICE	-0.039645
REGION_POPULATION_RELATIVE	-0.037227
ELEVATORS_AVG	-0.034199

Name: TARGET, dtype: float64

```
In [33]: correlations = pd.DataFrame(correlations, columns = ['TARGET'])
correlations
```

Out[33]:

	TARGET
EXT_SOURCE_3	-0.178919
EXT_SOURCE_2	-0.160472
EXT_SOURCE_1	-0.155317
 DAYS_EMPLOYED	-0.044932
FLOORSMAX_AVG	-0.044003
...	...
DAYS_LAST_PHONE_CHANGE	0.055218
REGION_RATING_CLIENT	0.058899
REGION_RATING_CLIENT_W_CITY	0.060893
DAYS_BIRTH	0.078239
TARGET	1.000000

106 rows × 1 columns

In [34]:

```
# To get top 4 correlated attributes
correlations["abs_Target"] = np.abs(correlations["TARGET"])
display(correlations)
correlations.sort_values("abs_Target", ascending = False, inplace = True)
display(correlations)
correlations
```

	TARGET	abs_Target
EXT_SOURCE_3	-0.178919	0.178919
EXT_SOURCE_2	-0.160472	0.160472
EXT_SOURCE_1	-0.155317	0.155317
 DAYS_EMPLOYED	-0.044932	0.044932
FLOORSMAX_AVG	-0.044003	0.044003
...
DAYS_LAST_PHONE_CHANGE	0.055218	0.055218
REGION_RATING_CLIENT	0.058899	0.058899
REGION_RATING_CLIENT_W_CITY	0.060893	0.060893
DAYS_BIRTH	0.078239	0.078239
TARGET	1.000000	1.000000

106 rows × 2 columns

	TARGET	abs_Target
TARGET	1.000000	1.000000
EXT_SOURCE_3	-0.178919	0.178919
EXT_SOURCE_2	-0.160472	0.160472
EXT_SOURCE_1	-0.155317	0.155317
DAYS_BIRTH	0.078239	0.078239
...
FLAG_DOCUMENT_12	-0.000756	0.000756
FLAG_MOBIL	0.000534	0.000534
FLAG_CONT_MOBILE	0.000370	0.000370
FLAG_DOCUMENT_5	-0.000316	0.000316
FLAG_DOCUMENT_20	0.000215	0.000215

106 rows × 2 columns

Out[34]:

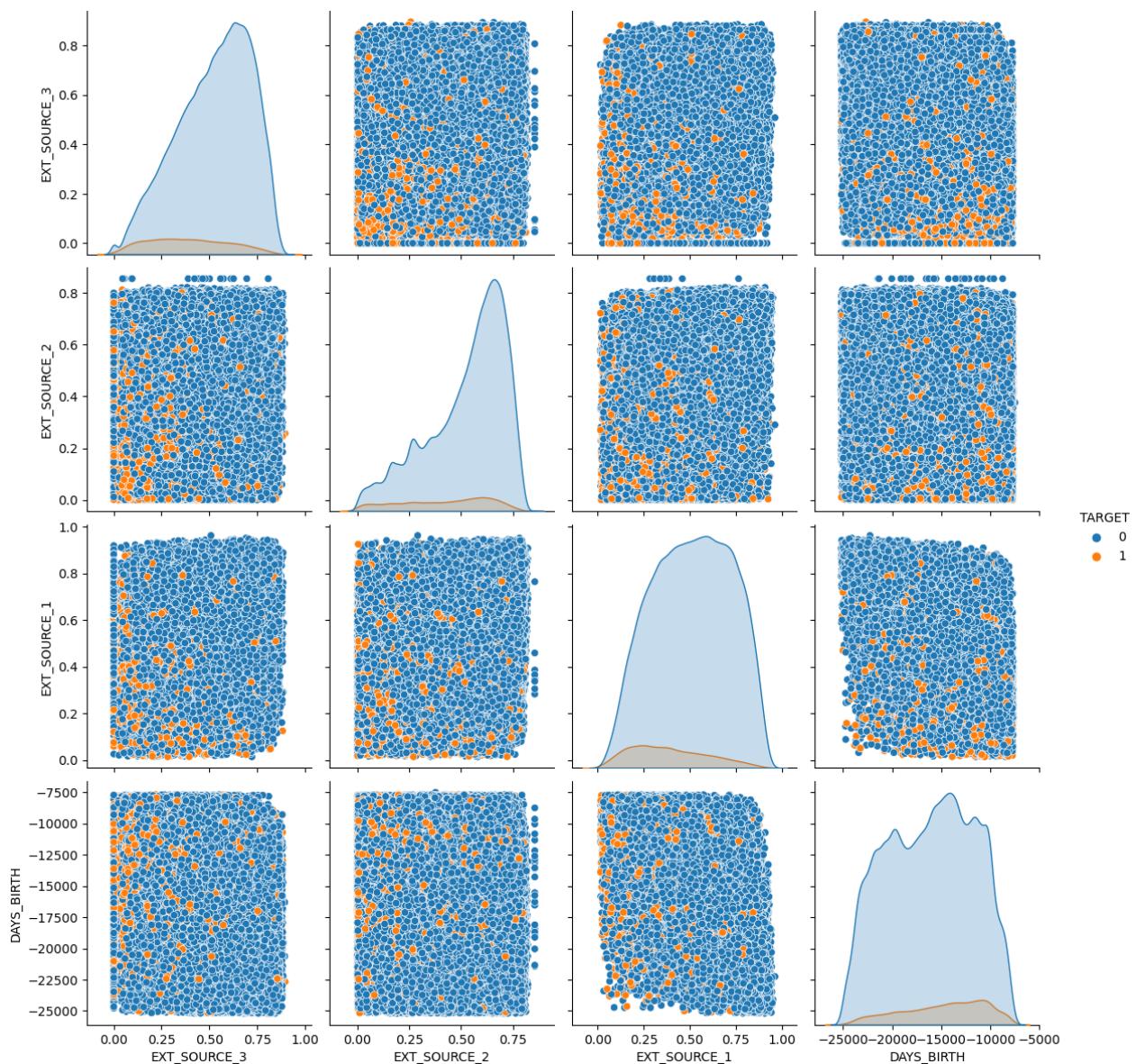
	TARGET	abs_Target
TARGET	1.000000	1.000000
EXT_SOURCE_3	-0.178919	0.178919
EXT_SOURCE_2	-0.160472	0.160472
EXT_SOURCE_1	-0.155317	0.155317
DAYS_BIRTH	0.078239	0.078239
...
FLAG_DOCUMENT_12	-0.000756	0.000756
FLAG_MOBIL	0.000534	0.000534
FLAG_CONT_MOBILE	0.000370	0.000370
FLAG_DOCUMENT_5	-0.000316	0.000316
FLAG_DOCUMENT_20	0.000215	0.000215

106 rows × 2 columns

Pair plot of the 4 top correlated features

In [31]:

```
attributes = ["EXT_SOURCE_3", "EXT_SOURCE_2", "EXT_SOURCE_1", "DAYS_BIRTH"]
sns.pairplot(data = datasets["application_train"], hue="TARGET", vars = attributes)
plt.show()
```



We can see the plot of the top 4 correlated attributes with the Target column.

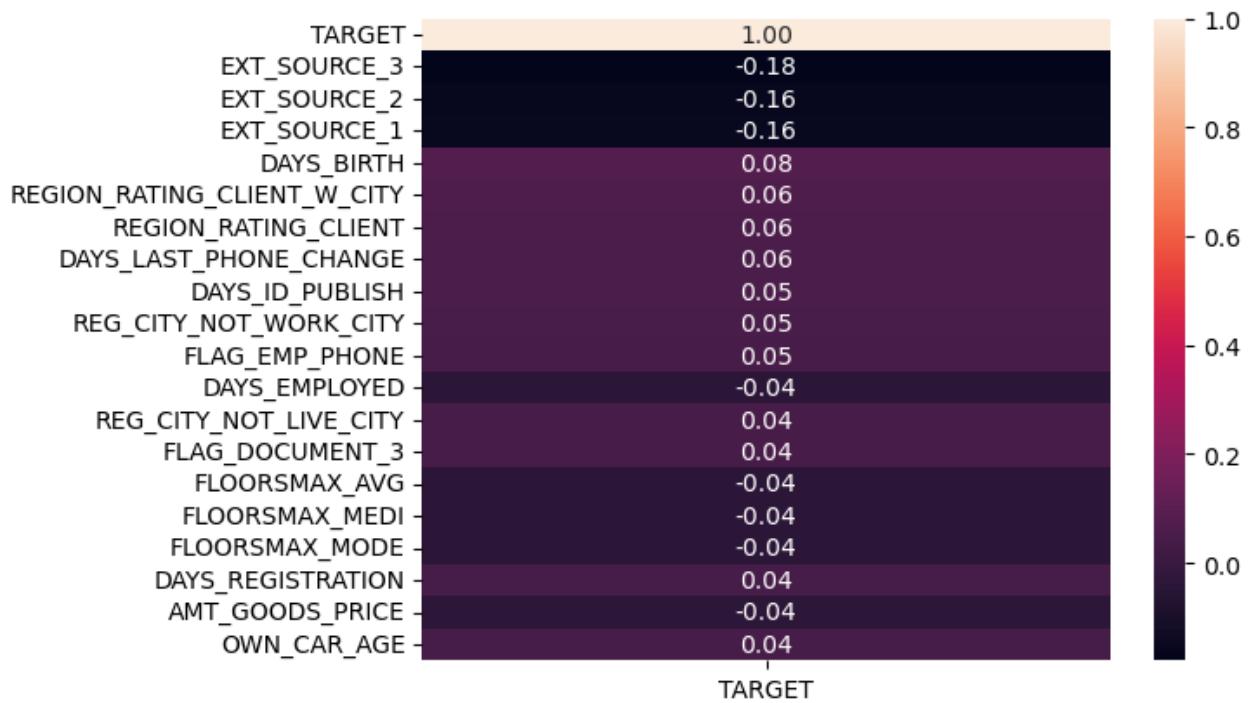
EXT_SOURCE_1 seems to be normally distributed while others are skewed but can be approximated to normal distribution.

```
In [35]: correlations.isnull().count()
```

```
Out[35]: TARGET      106
abs_Target    106
dtype: int64
```

Heatmap of correlated attributes

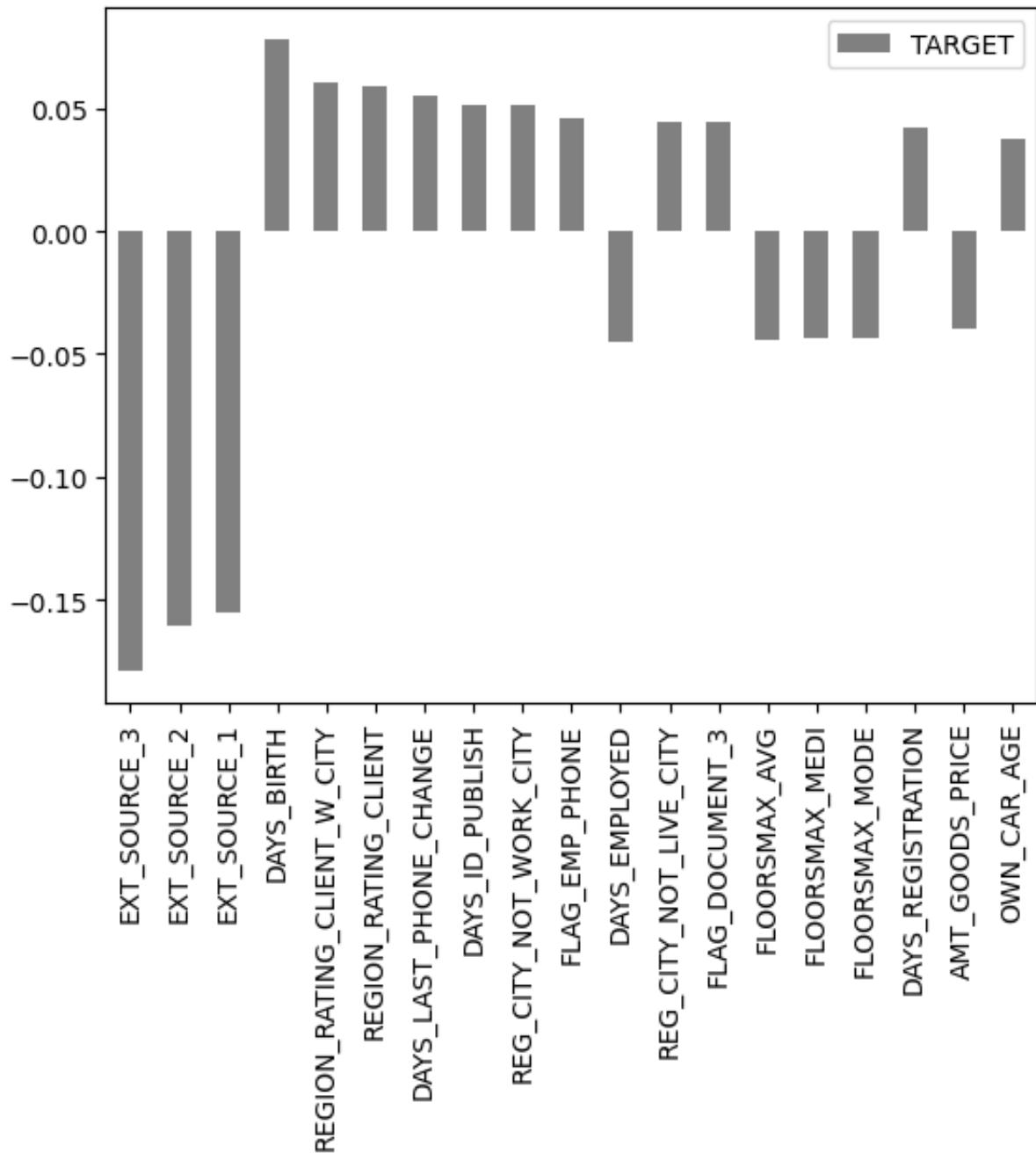
```
In [50]: corr_target = correlations.drop(['abs_Target'], axis = 1)
corr_target= corr_target[:20].dropna()
sns.heatmap(corr_target, annot=True, fmt=' .2f')
plt.show()
```



Bar plot of correlated attributes

```
In [51]:  
plt.figure(figsize=(15,7))  
plot = corr_target[1:].plot(kind = 'bar', color = 'grey')  
plt.setp(plot.get_xticklabels(), rotation=90)  
plt.show()
```

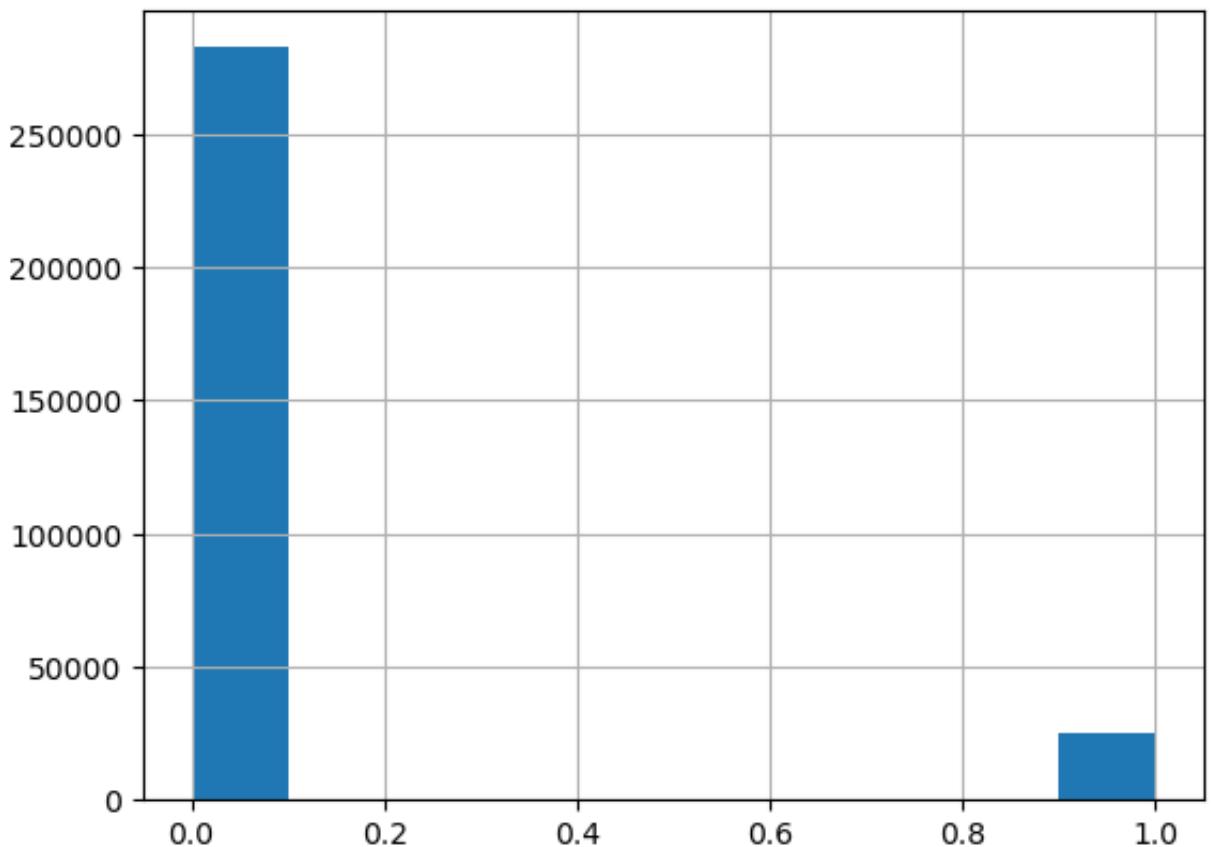
<Figure size 1500x700 with 0 Axes>



Visual EDA

Distribution of the target column

```
In [52]:  
datasets["application_train"]['TARGET'].astype(int).hist()  
plt.show()
```

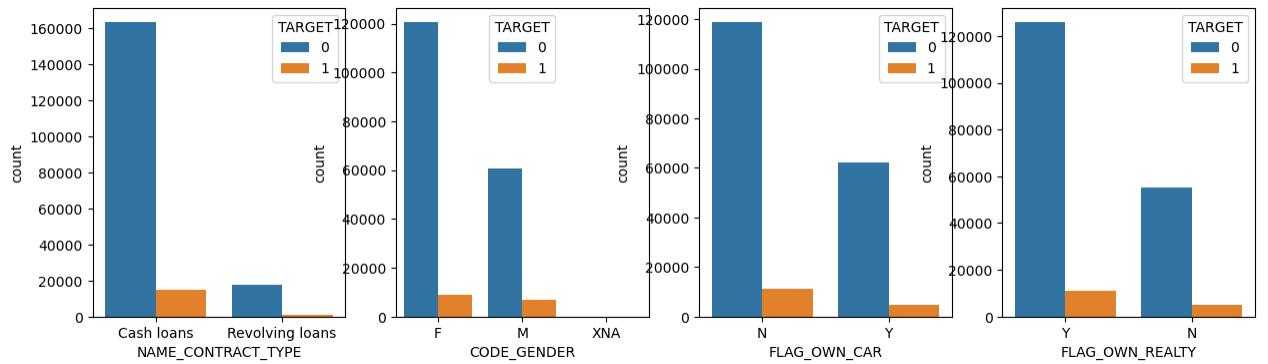


Evaluating categorical features with respect to TARGET

TARGET - 0: LOAN WAS REPAYED 1: LOAN WAS NOT REPAYED

In [55]:

```
cat_vars = list(categorical_features)[:4]
plt.figure(figsize=(15,4))
for idx, cat in enumerate(cat_vars):
    plt.subplot(1, len(cat_vars), idx+1)
    sns.countplot(Xy_train[cat], hue=Xy_train['TARGET'])
plt.show()
```



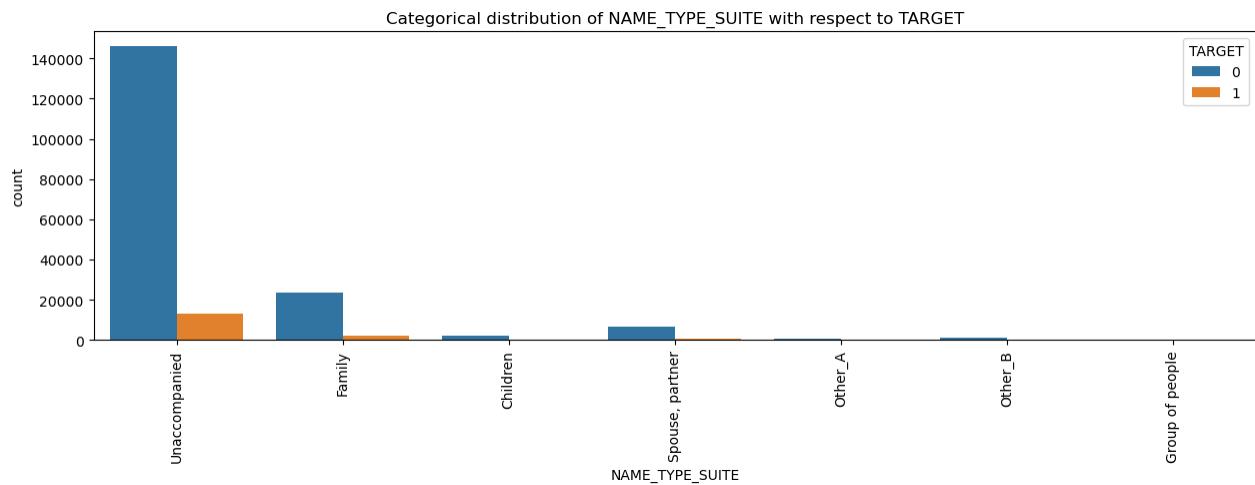
- We notice that Cash loans are being repaid more than Revolving loans
- More number of female borrowers have repaid loans than male borrowers
- We notice that a client who does not own a car repays loan easily than a client who owns a car
- We notice the same trend for a client who owns a house or flat. Owner of a real estate will repay loans less easily than a person who does not own any real estate

In [144]:

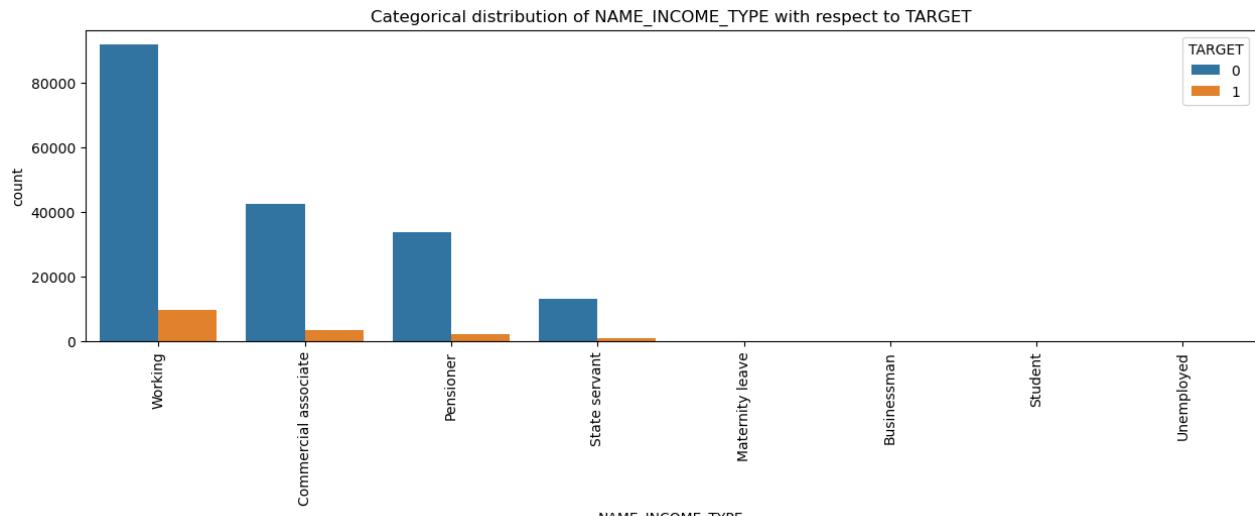
```
explanations = ['We can see that people who are not accompanied by anyone
```

In [145]:

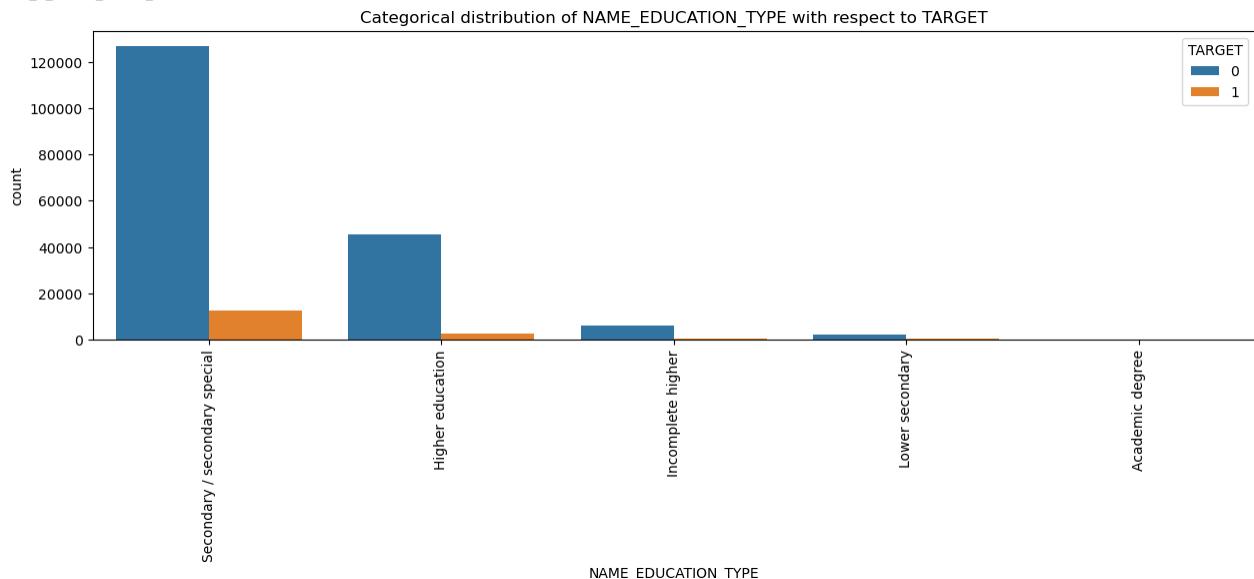
```
i = 0
for cat in list(categorical_features[4:14]):
    plt.figure(figsize=(15,4))
    plot = sns.countplot(x=cat, data=Xy_train, hue = Xy_train['TARGET'])
    plt.setp(plot.get_xticklabels(), rotation=90)
    plt.title(f'Categorical distribution of {cat} with respect to TARGET')
    plt.show()
    print(explanations[i])
    i += 1
```



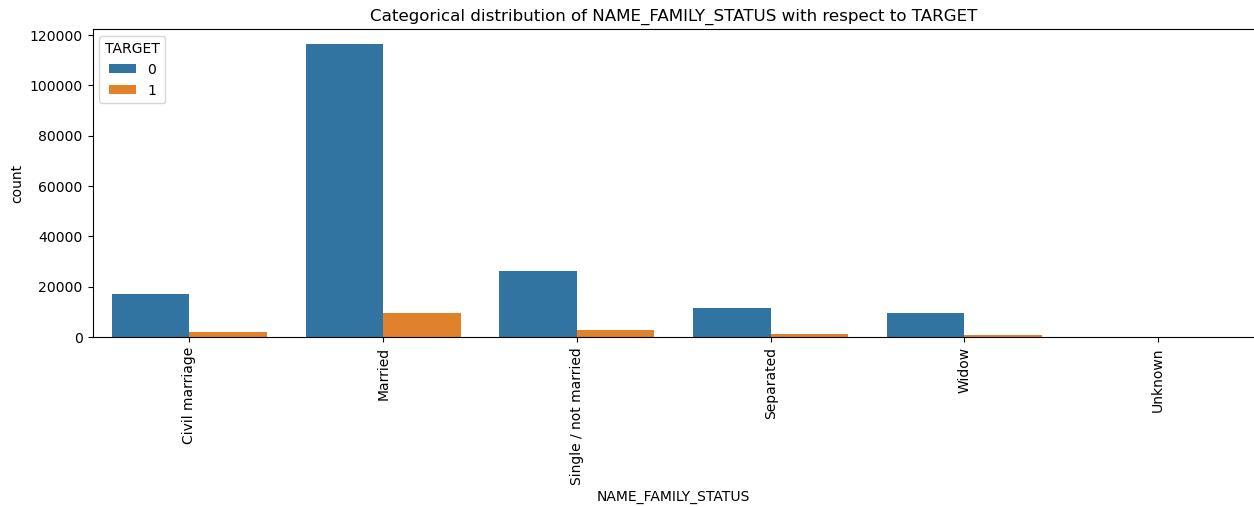
We can see that people who are not accompanied by anyone that is people having no dependents are able to repay loans easier.



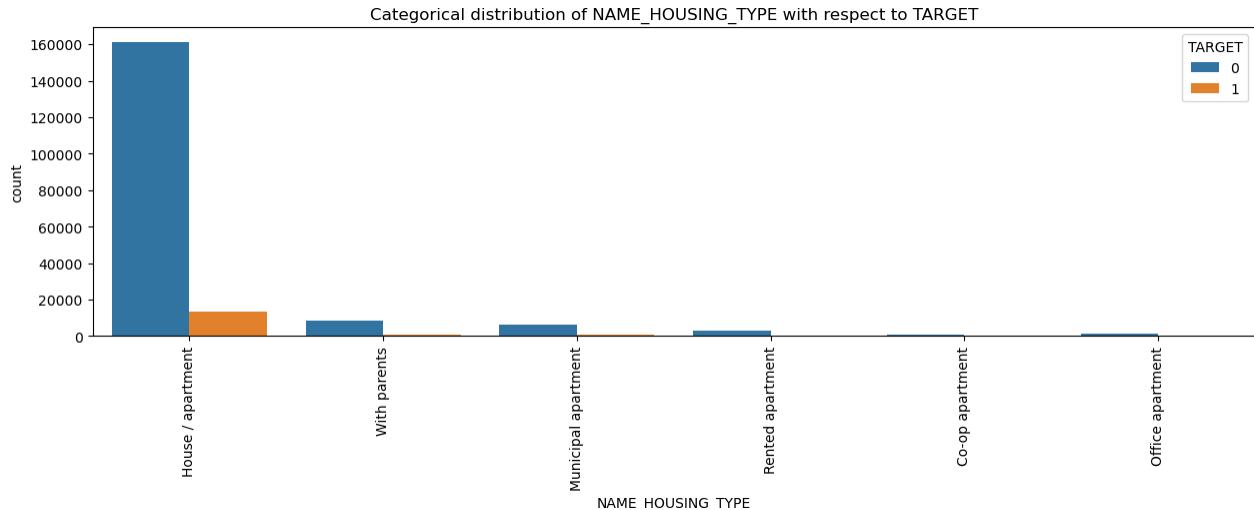
Working class people usually require more loans as compared to other income type people.



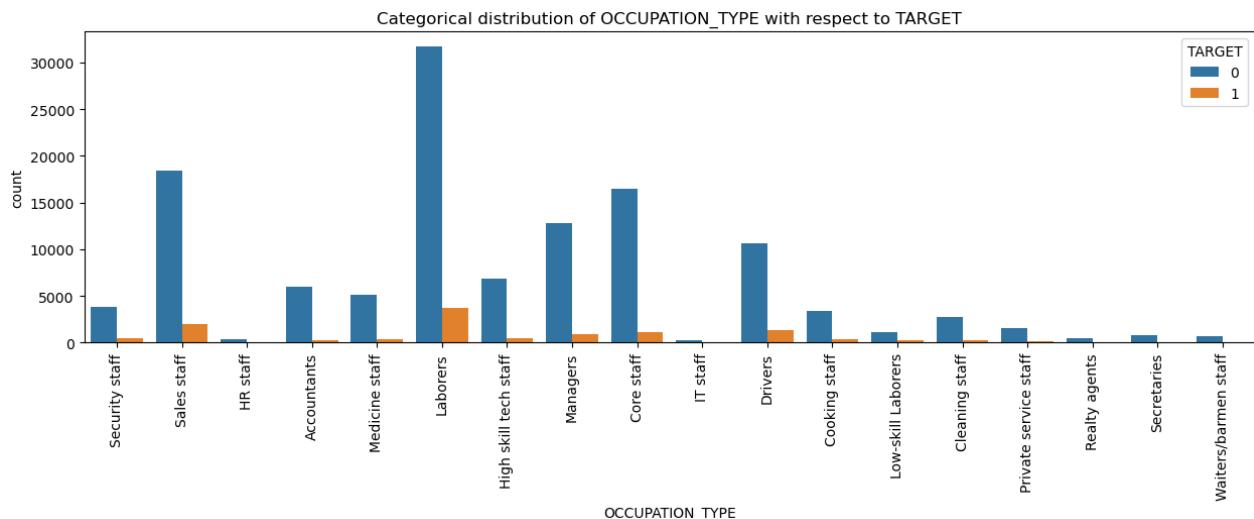
We see that people who have education as Secondary/Secondary special require more loans than people of other education backgrounds.



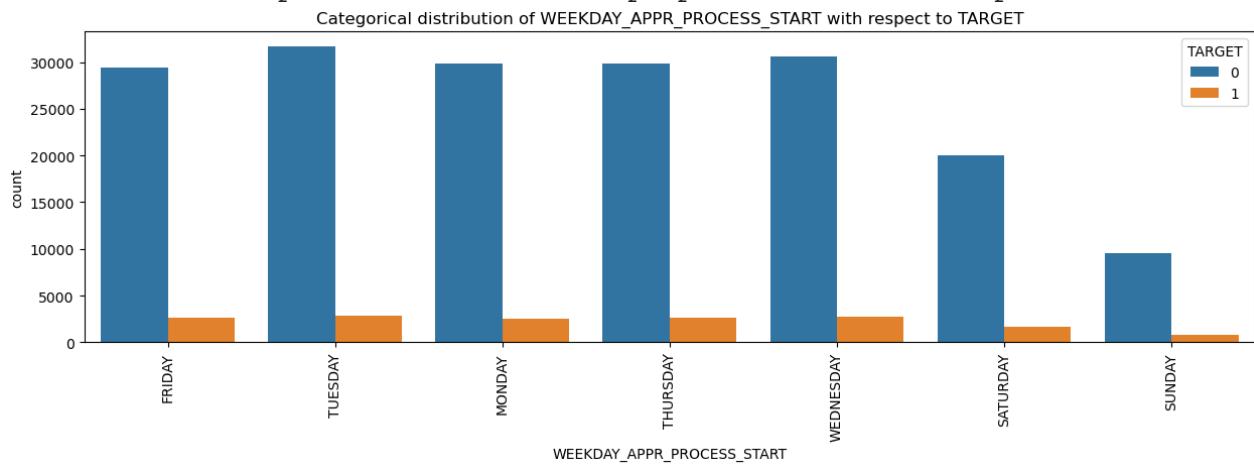
People who are married have taken more loans and repaid them as compared to people having other than marriage family status



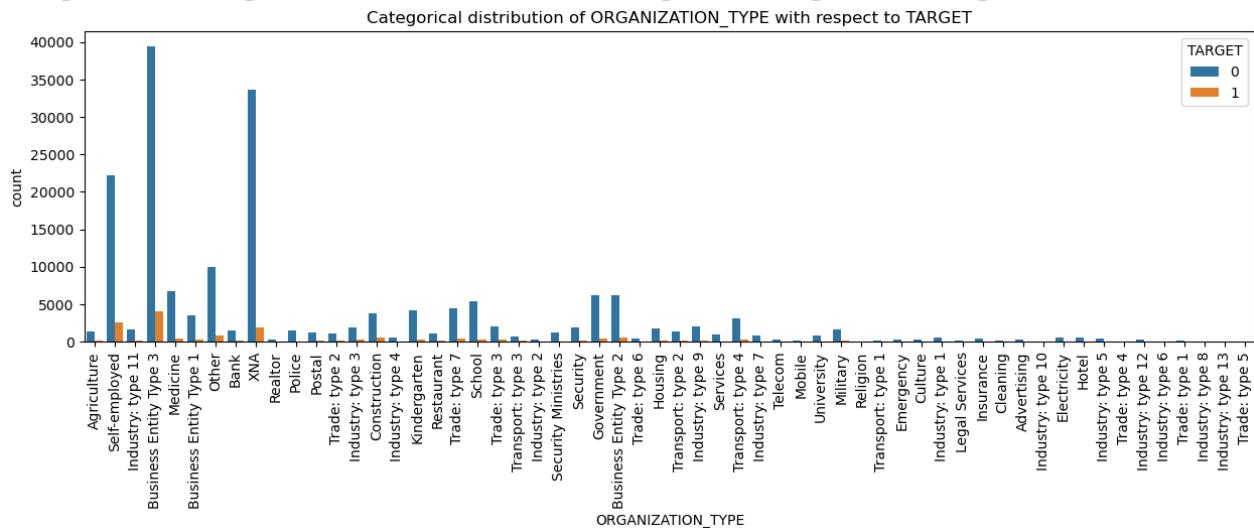
We see that people who live alone and in apartments require more loans than other people



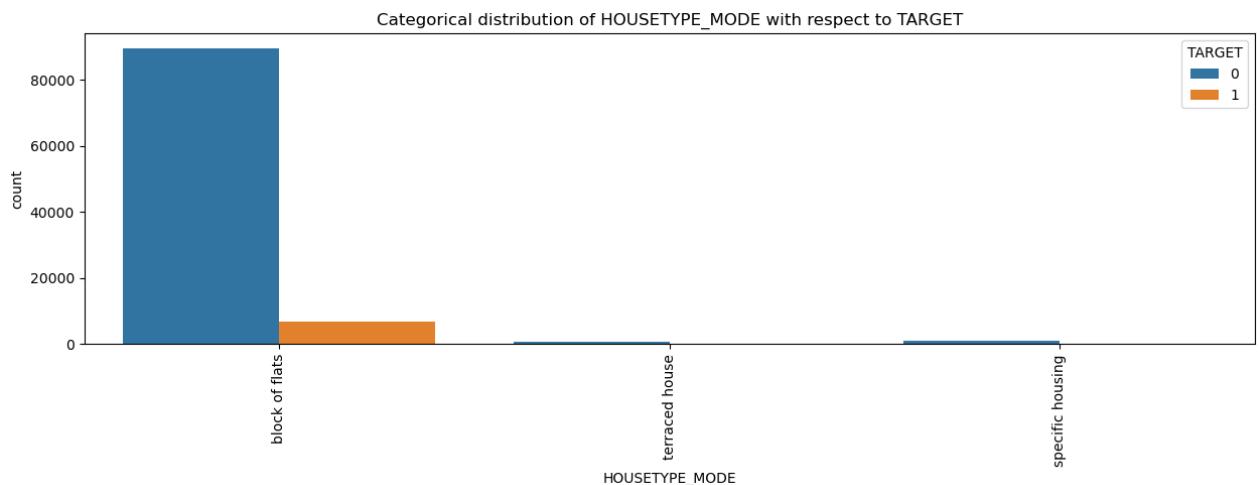
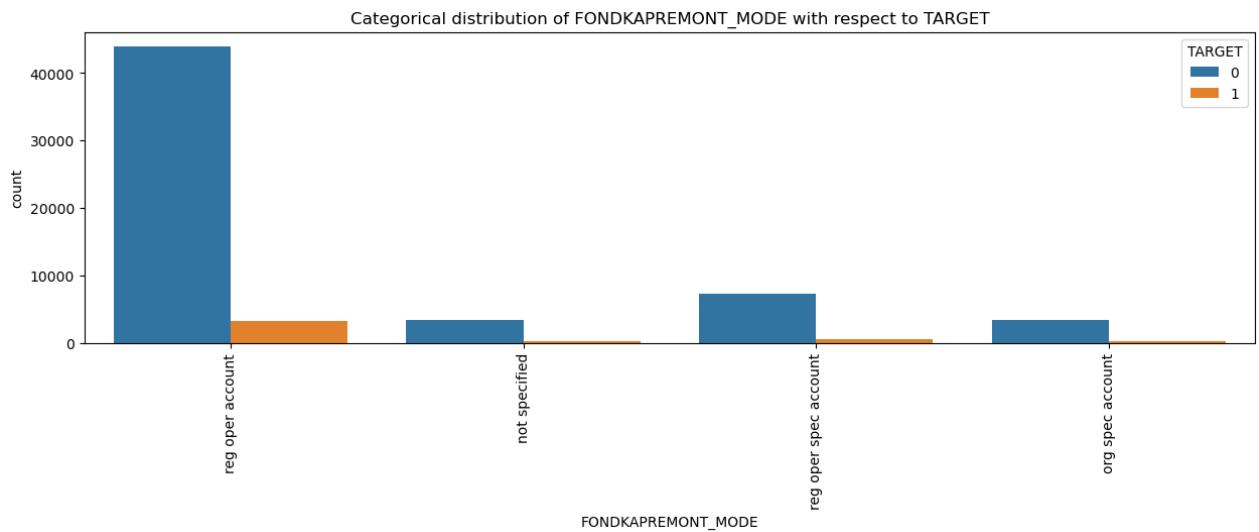
Laborers have repaid more loans than people with other occupations



People have repaid more loans on Tuesday than any other day of the week



People belonging to Business organization have repaid more loans than other organization types

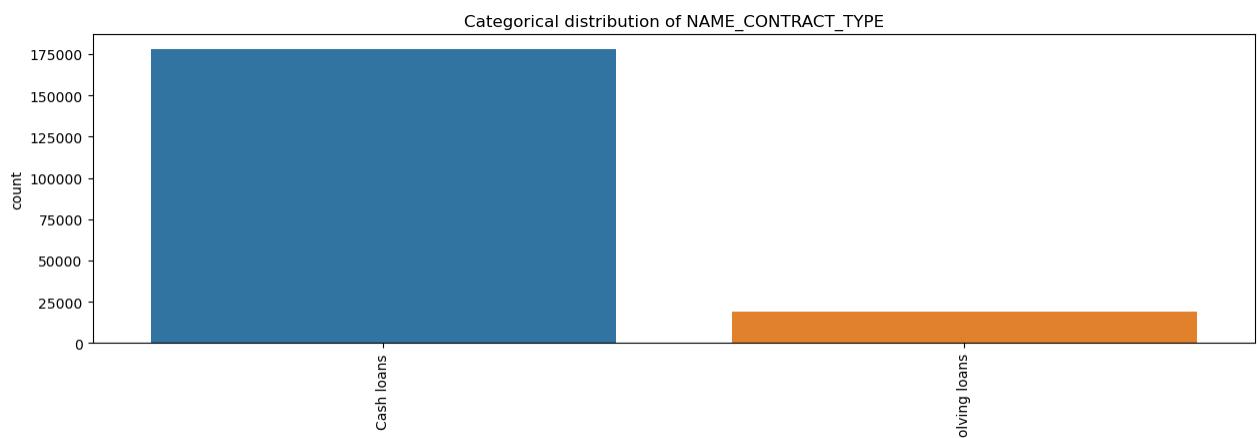


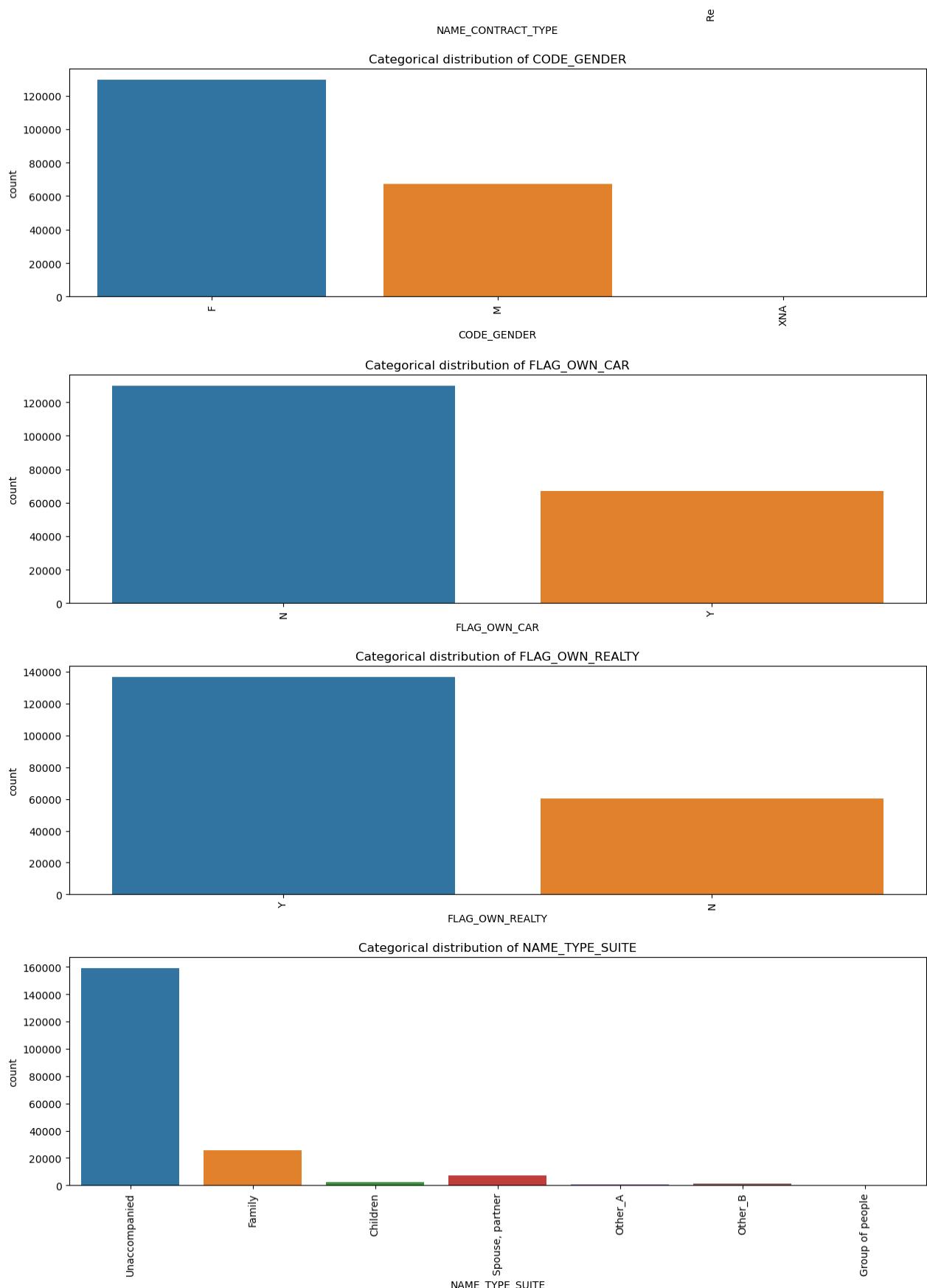
People who live in flats have repaid more loans than people who live in other types of houses

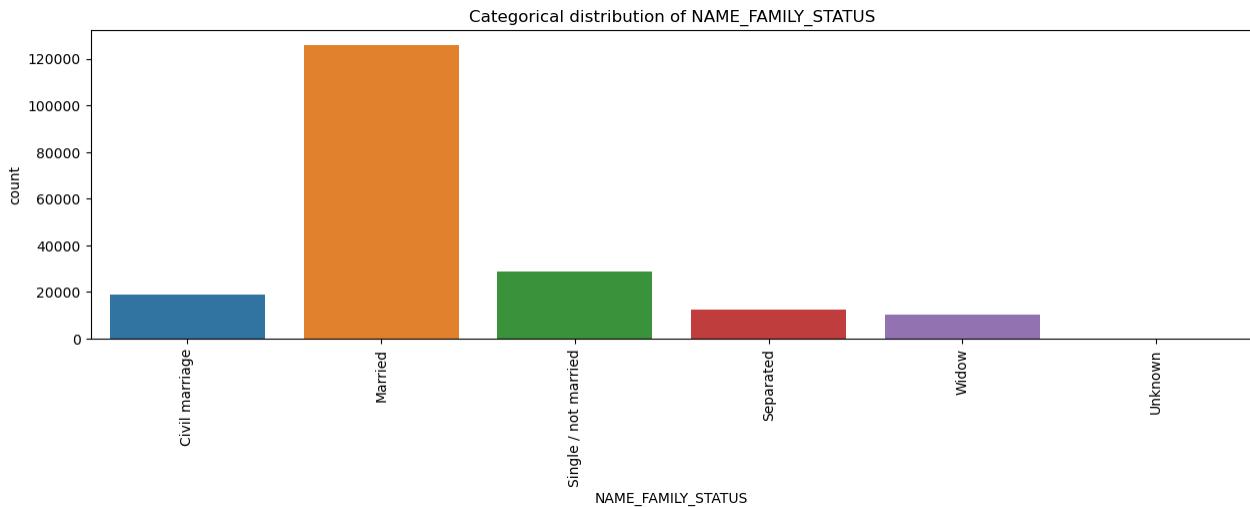
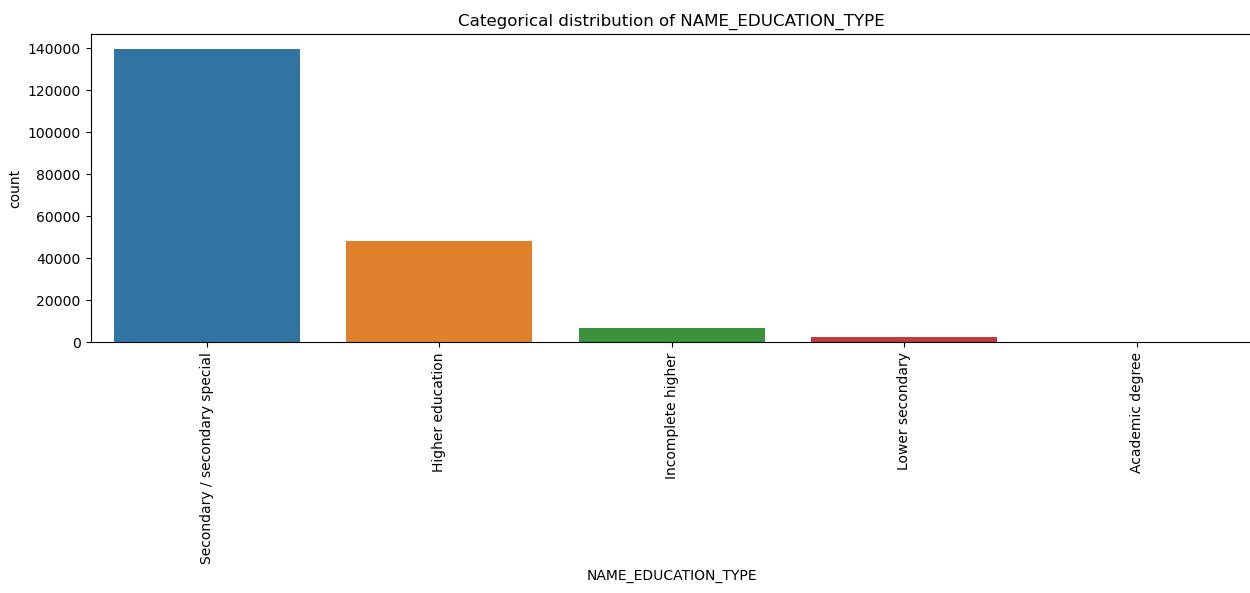
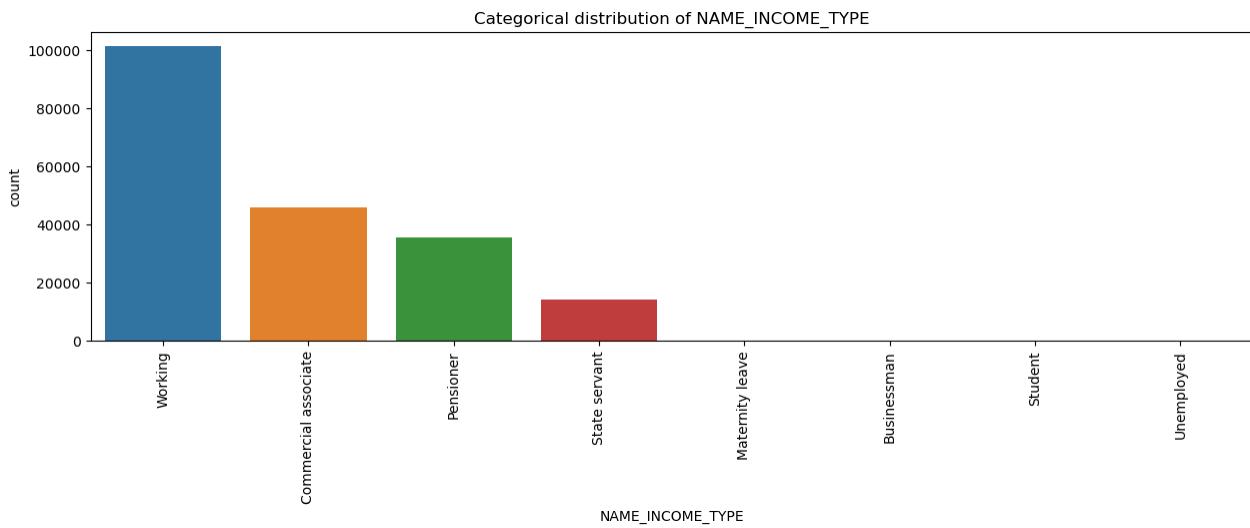
Categorical distribution

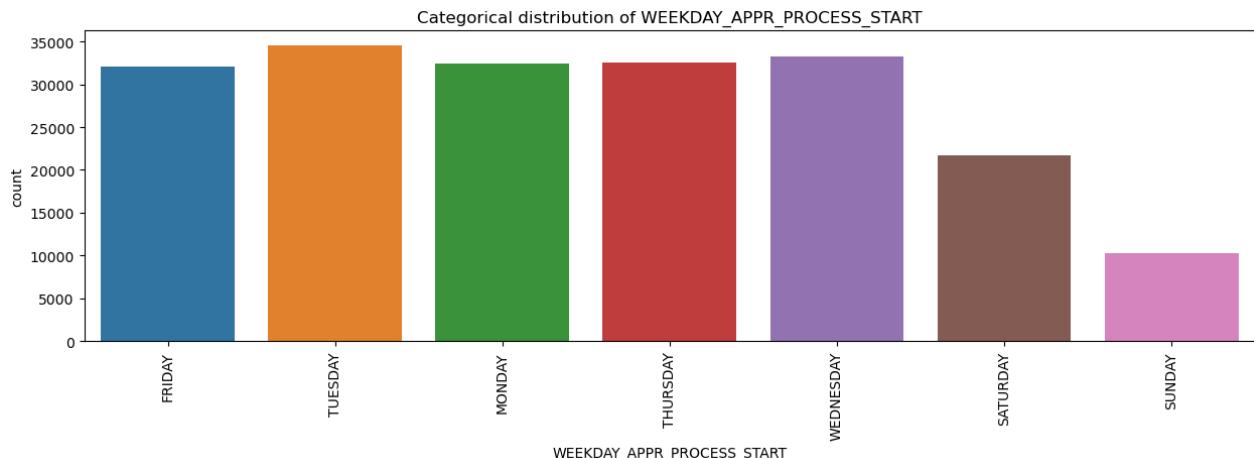
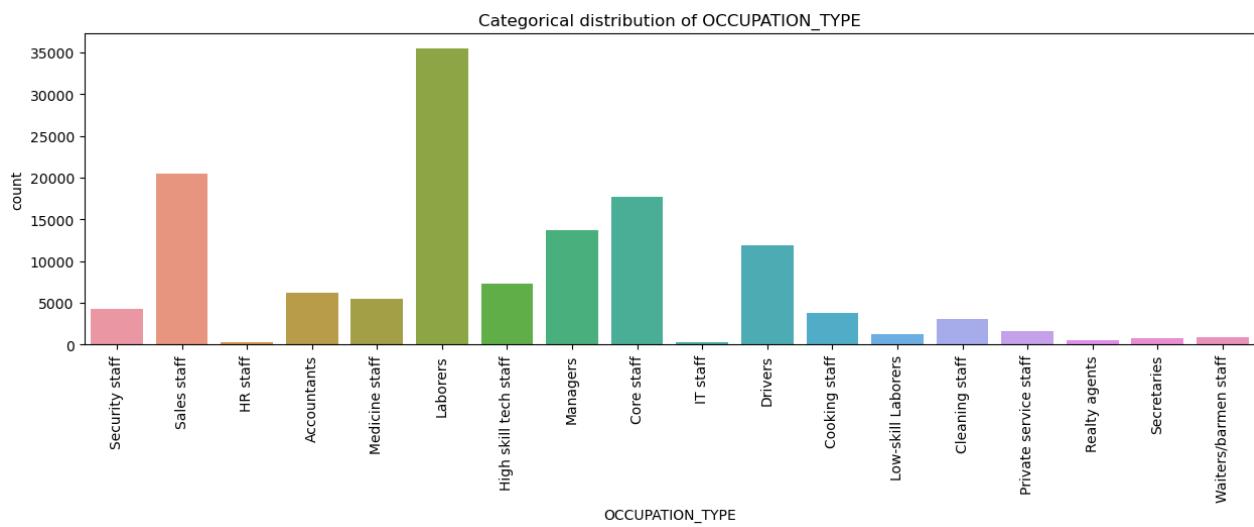
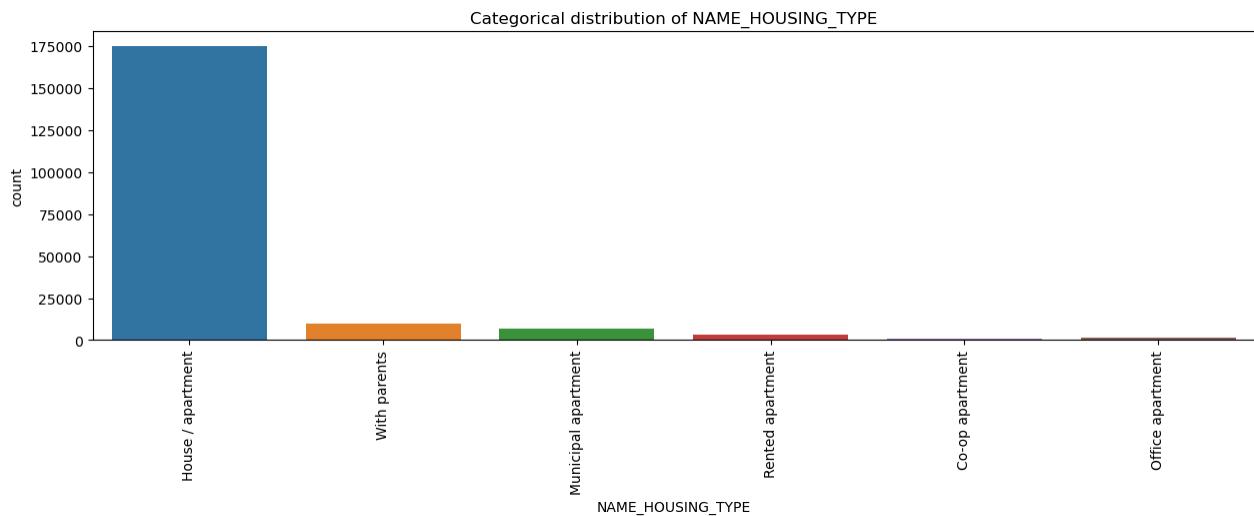
In [149...]

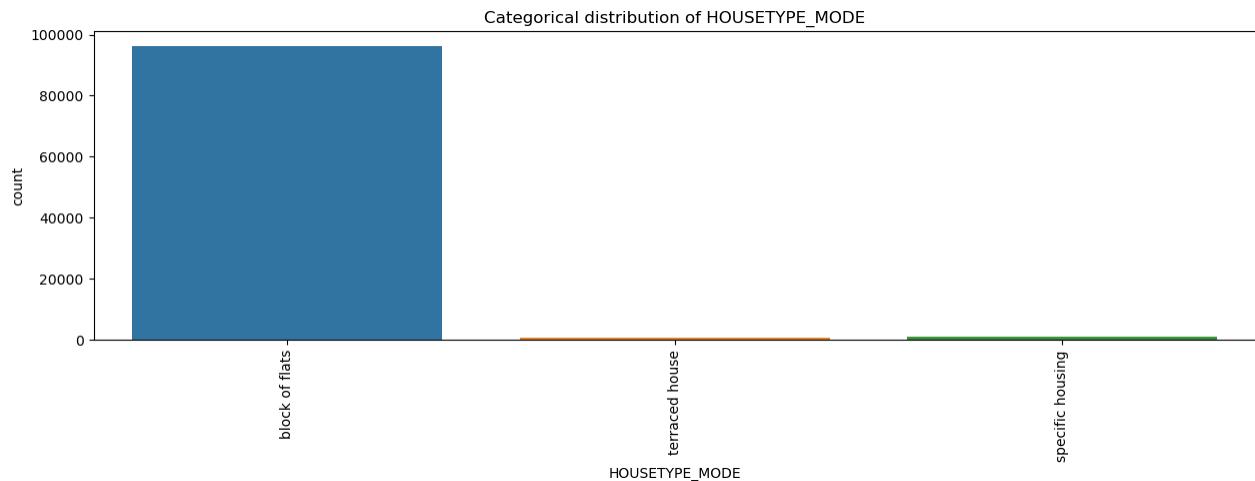
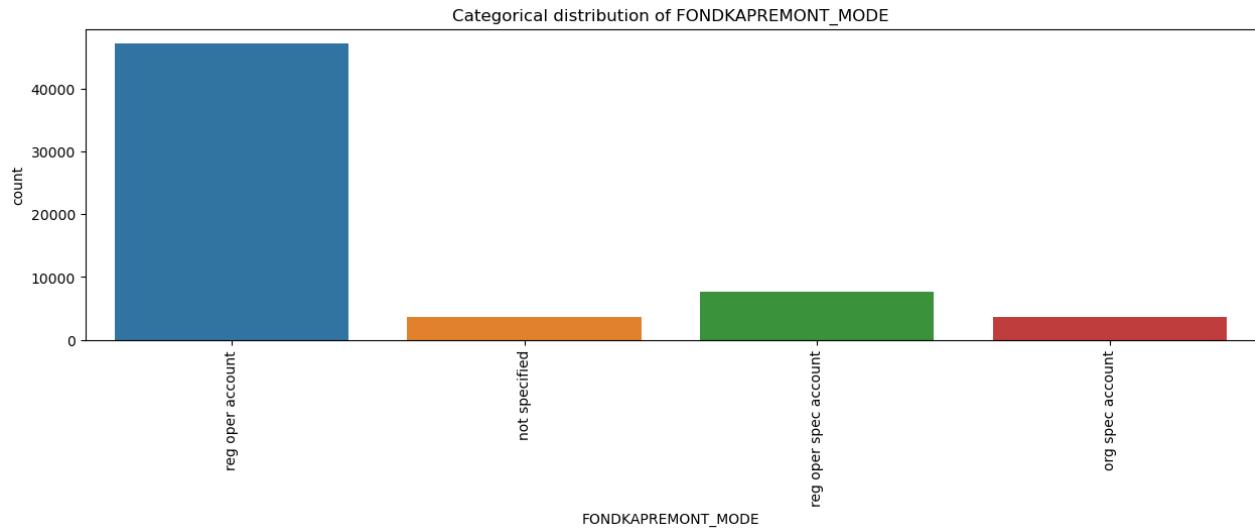
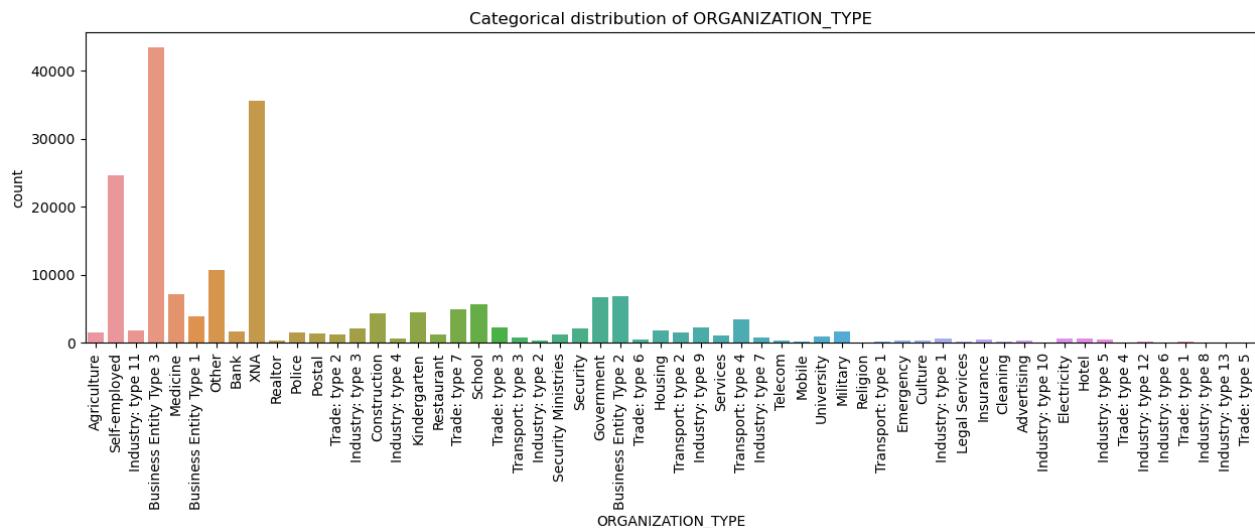
```
for cat in list(categorical_features[:14]):
    plt.figure(figsize=(15,4))
    plot = sns.countplot(x=cat, data=Xy_train)
    plt.setp(plot.get_xticklabels(), rotation=90)
    plt.title(f'Categorical distribution of {cat}')
    plt.show()
```







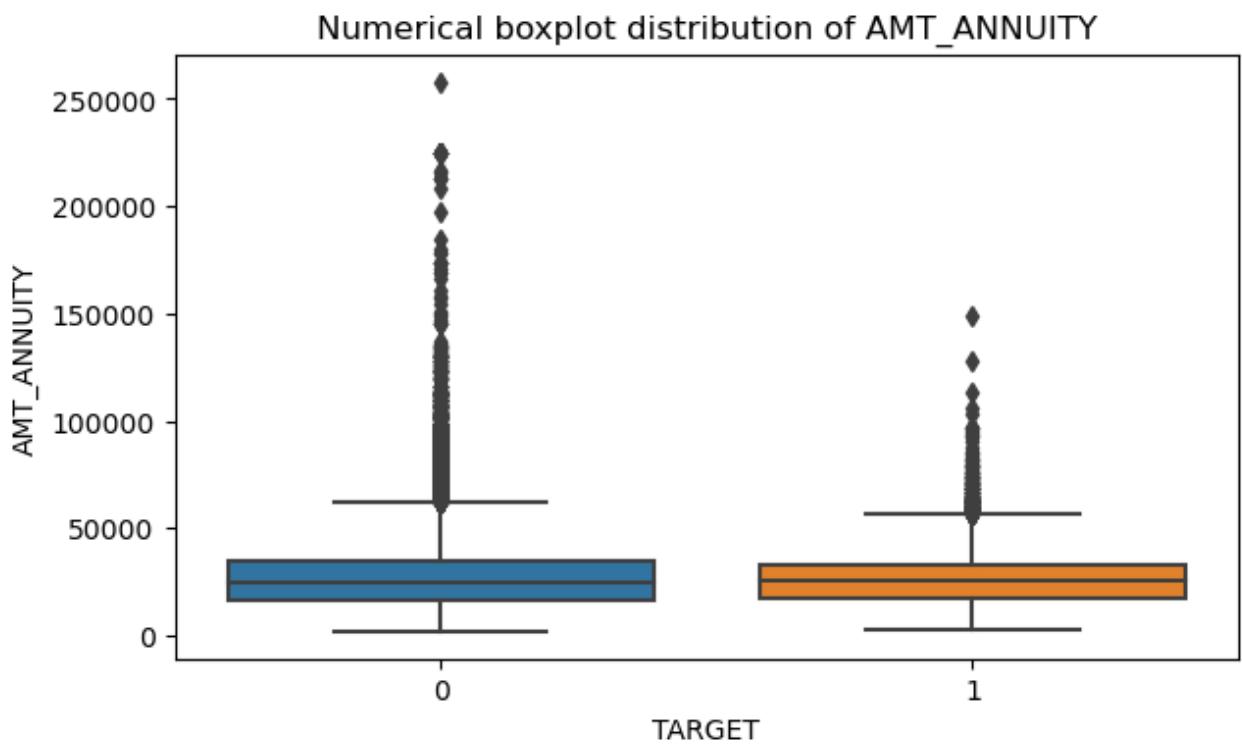
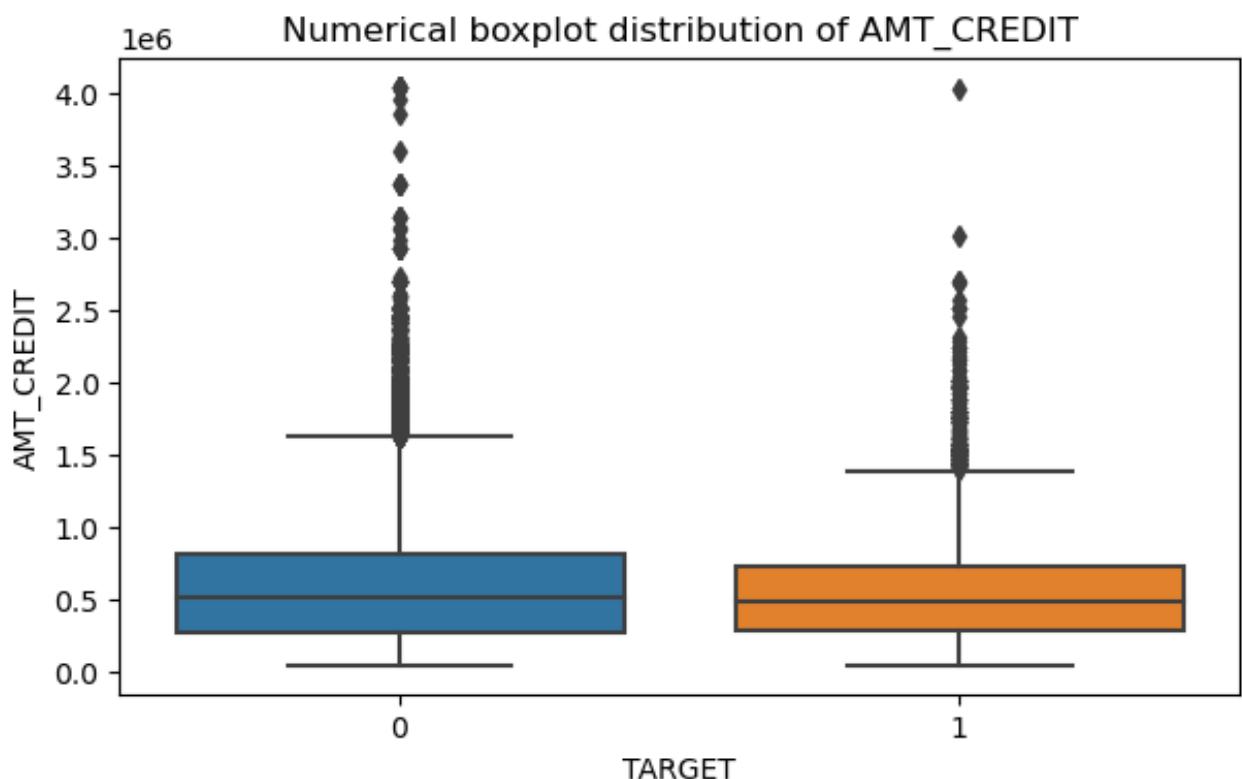


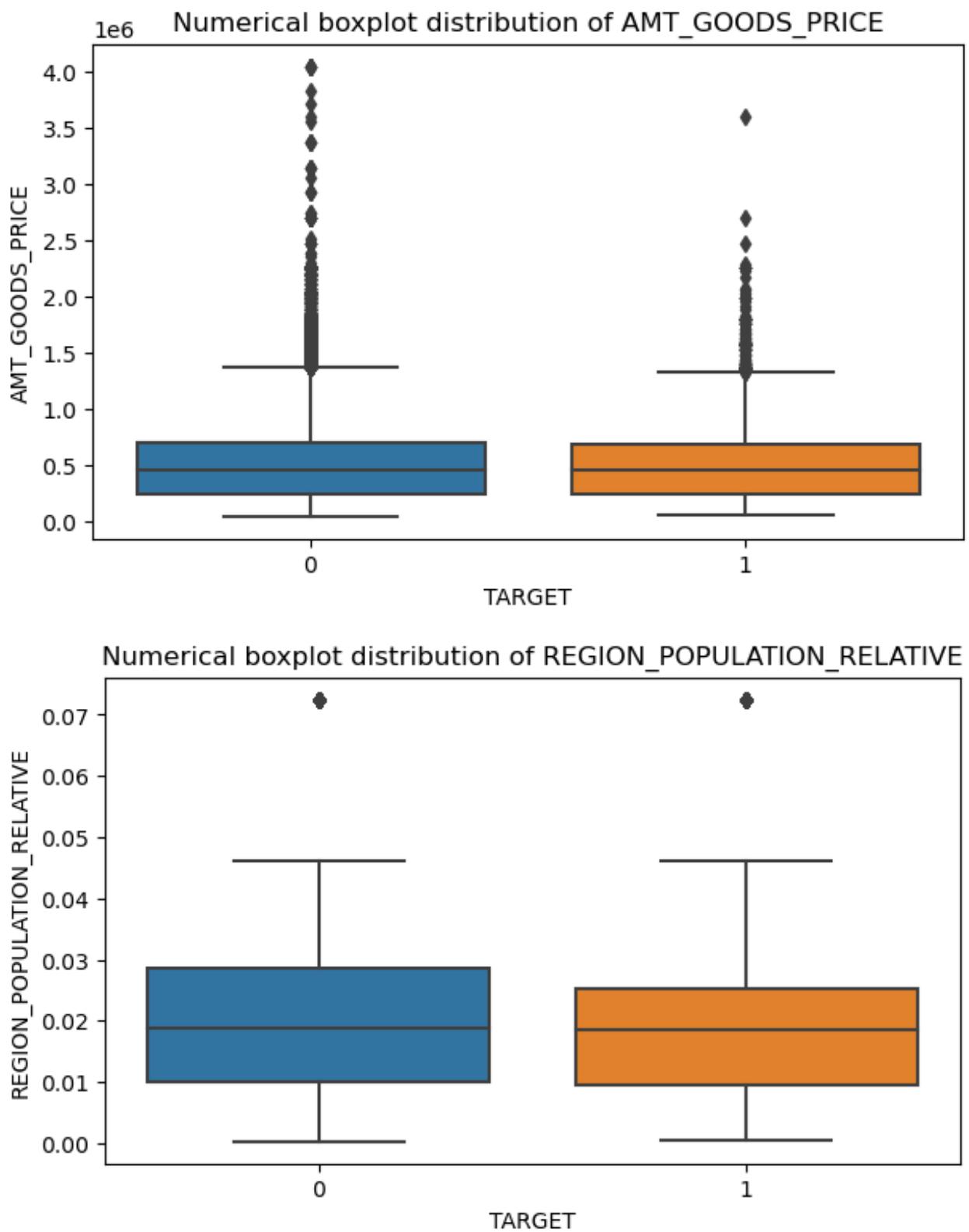


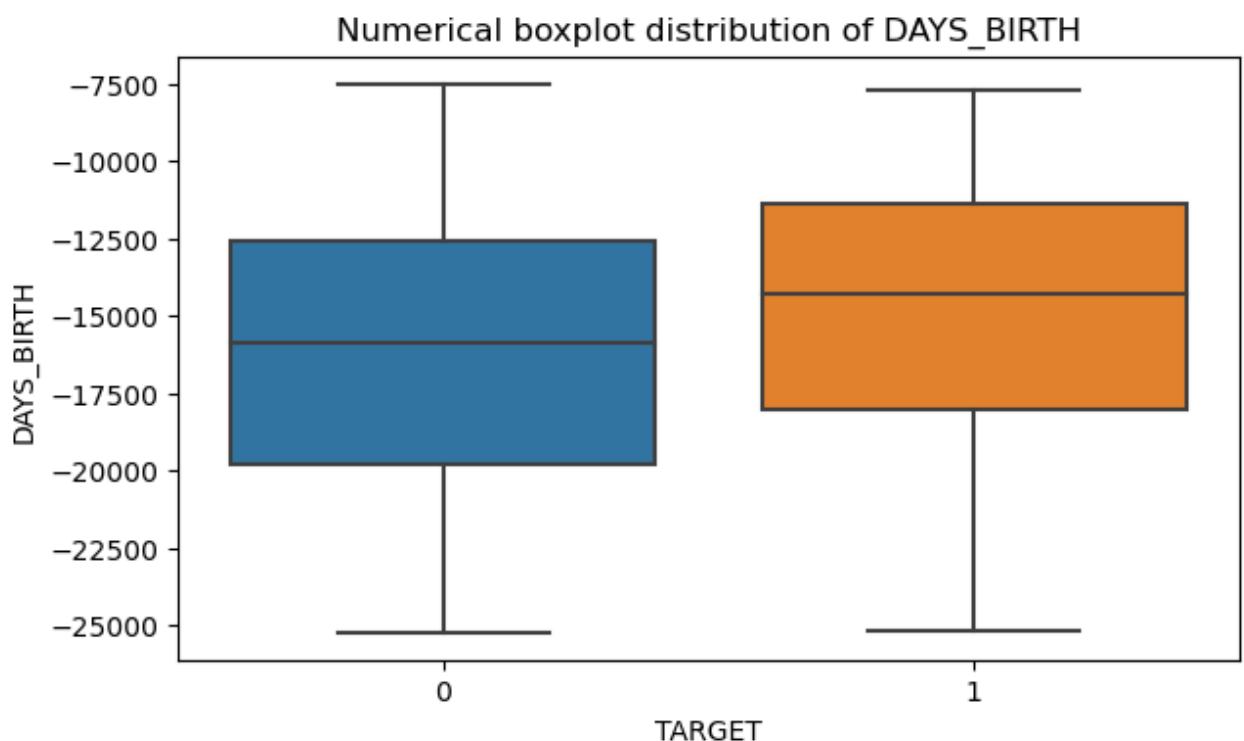
Numerical distribution

In [150...]

```
for num in list(numerical_features[3:8]):  
    plt.figure(figsize=(7, 4))  
    sns.boxplot(x = 'TARGET', y = num, data = Xy_train)  
    plt.title(f'Numerical boxplot distribution of {num}')  
    #plt.setp(plot.get_xticklabels(), rotation=90)  
    plt.show()
```



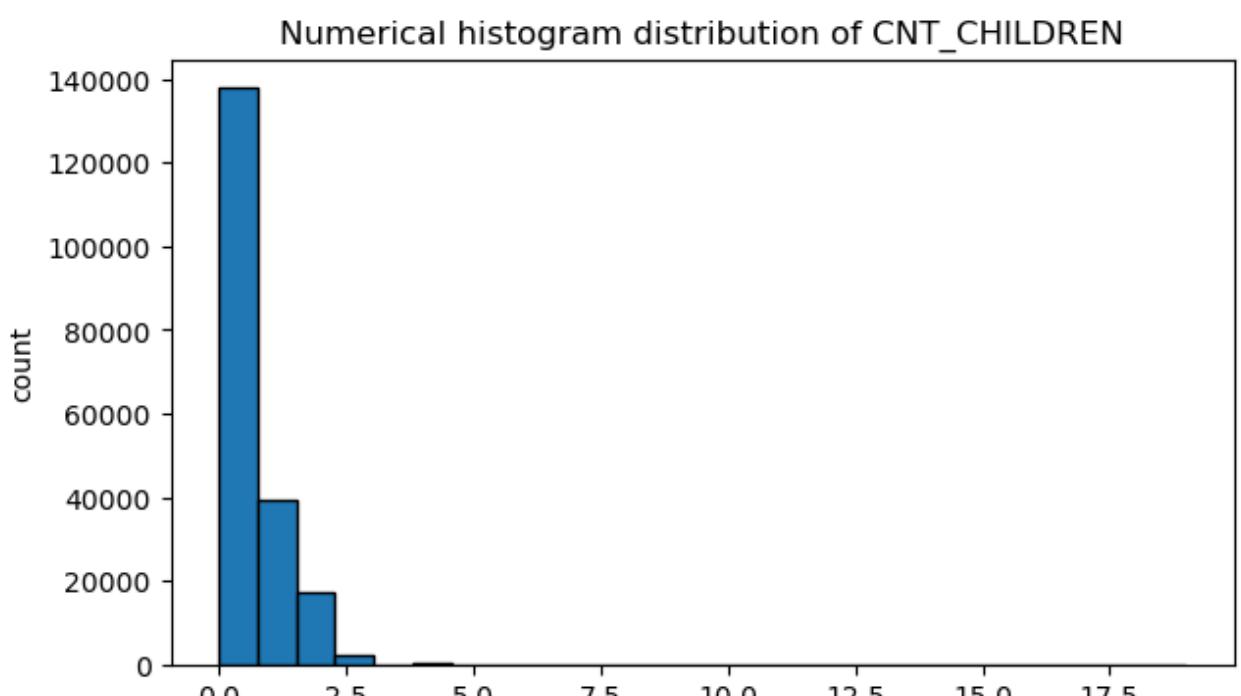




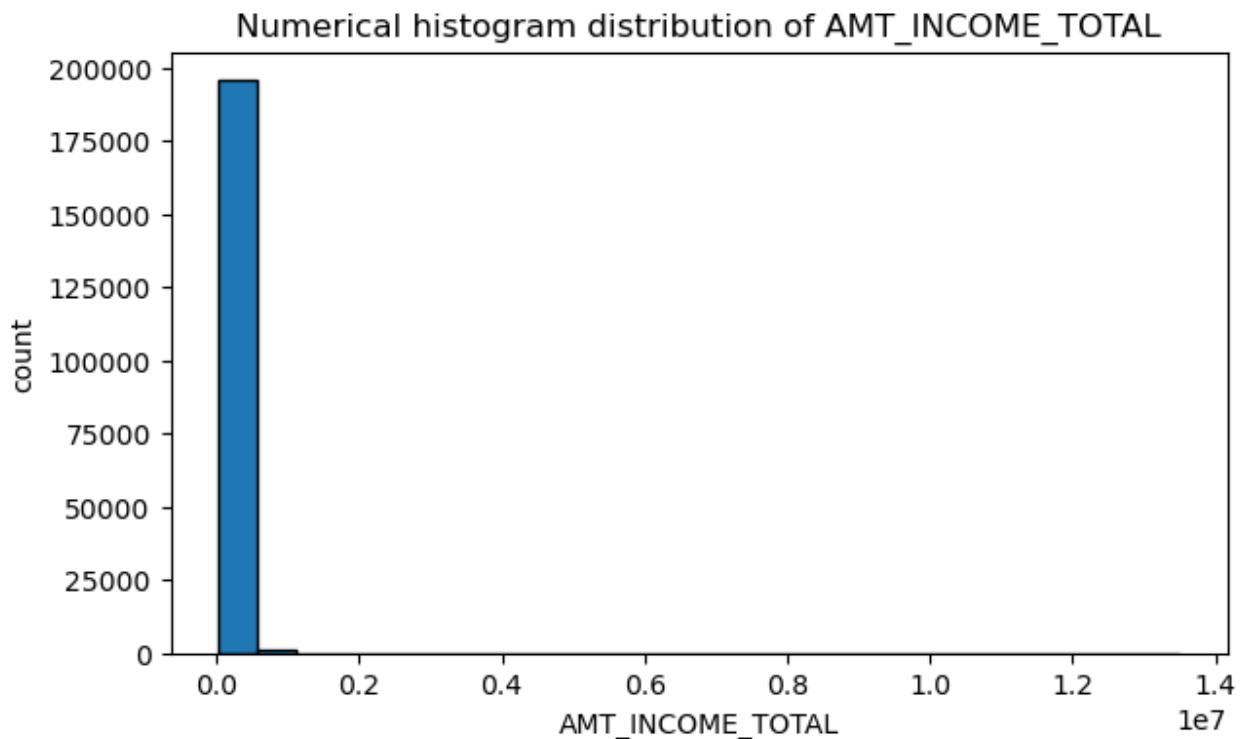
We notice that there are many outliers in the data as seen in the box plot. We can visualize the median and the quantiles of each column data by these box plots.

In [151...]

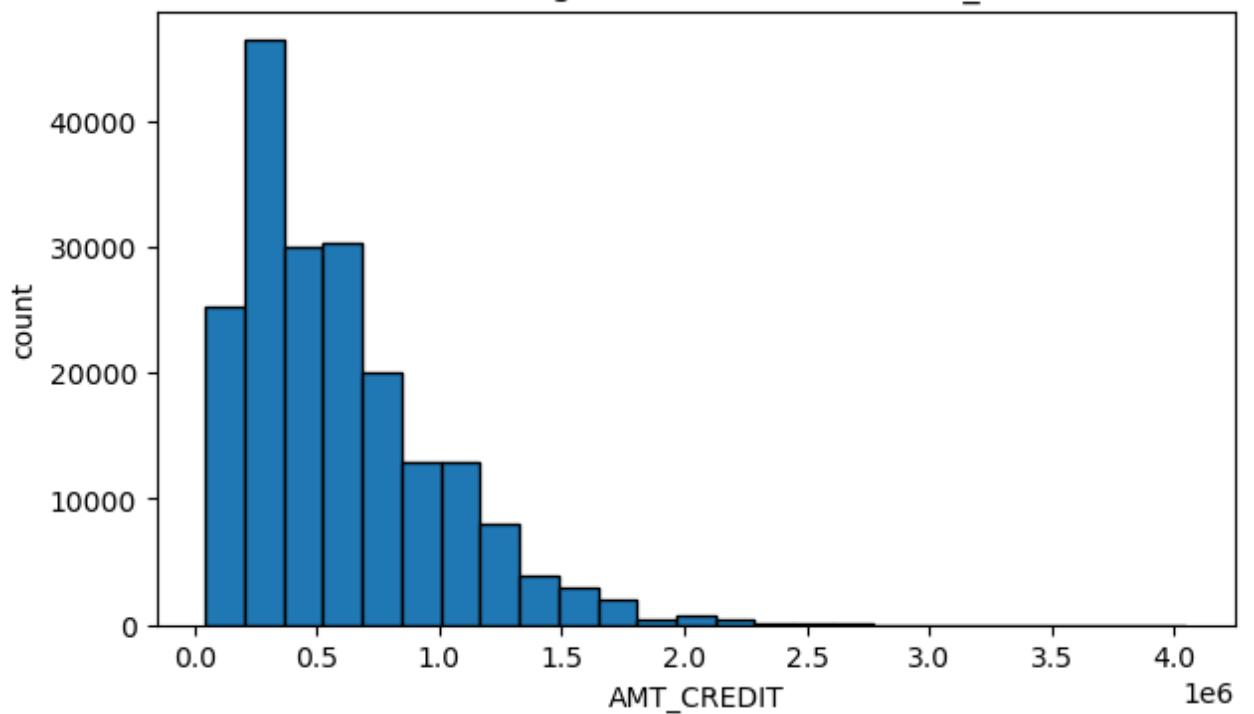
```
for num in list(numerical_features[1:7]):  
    plt.figure(figsize=(7, 4))  
    plt.hist(Xy_train[num], edgecolor = 'k', bins = 25)  
    plt.xlabel(num)  
    plt.ylabel('count')  
    plt.title(f'Numerical histogram distribution of {num}')  
    #plt.setp(plot.get_xticklabels(), rotation=90)  
    plt.show()
```

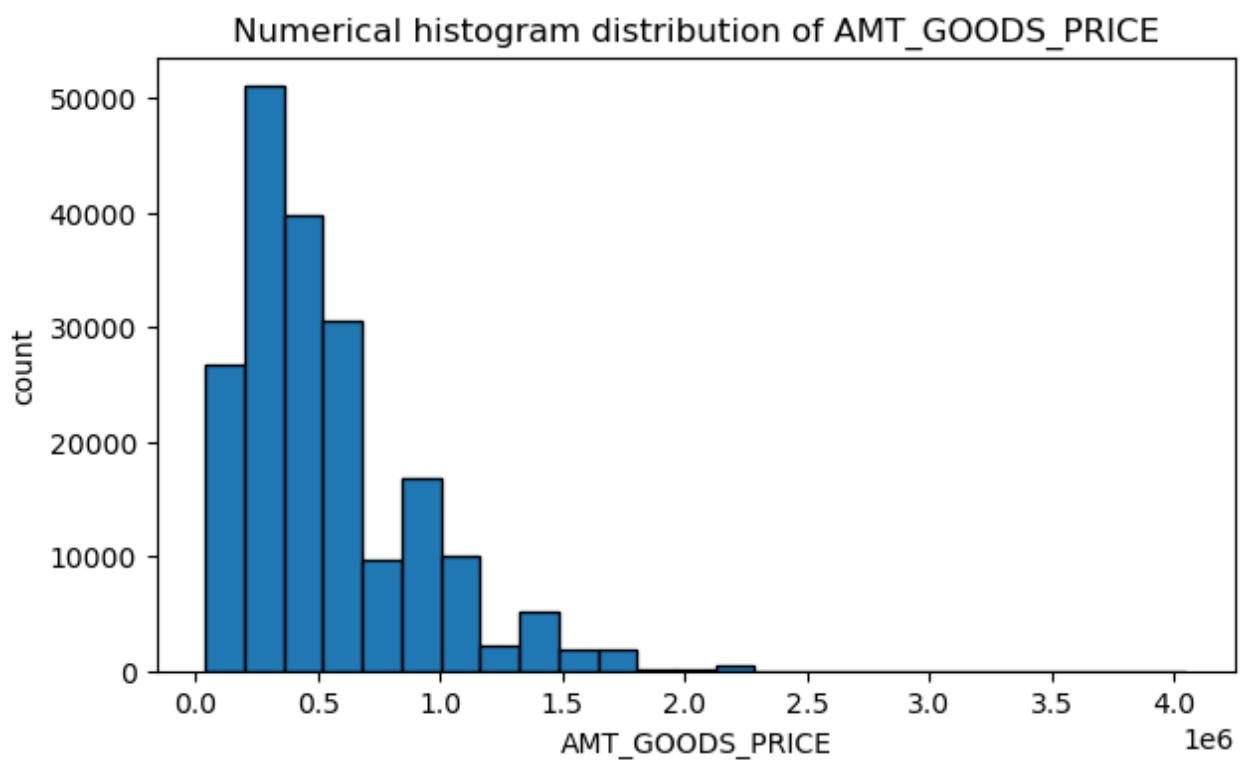
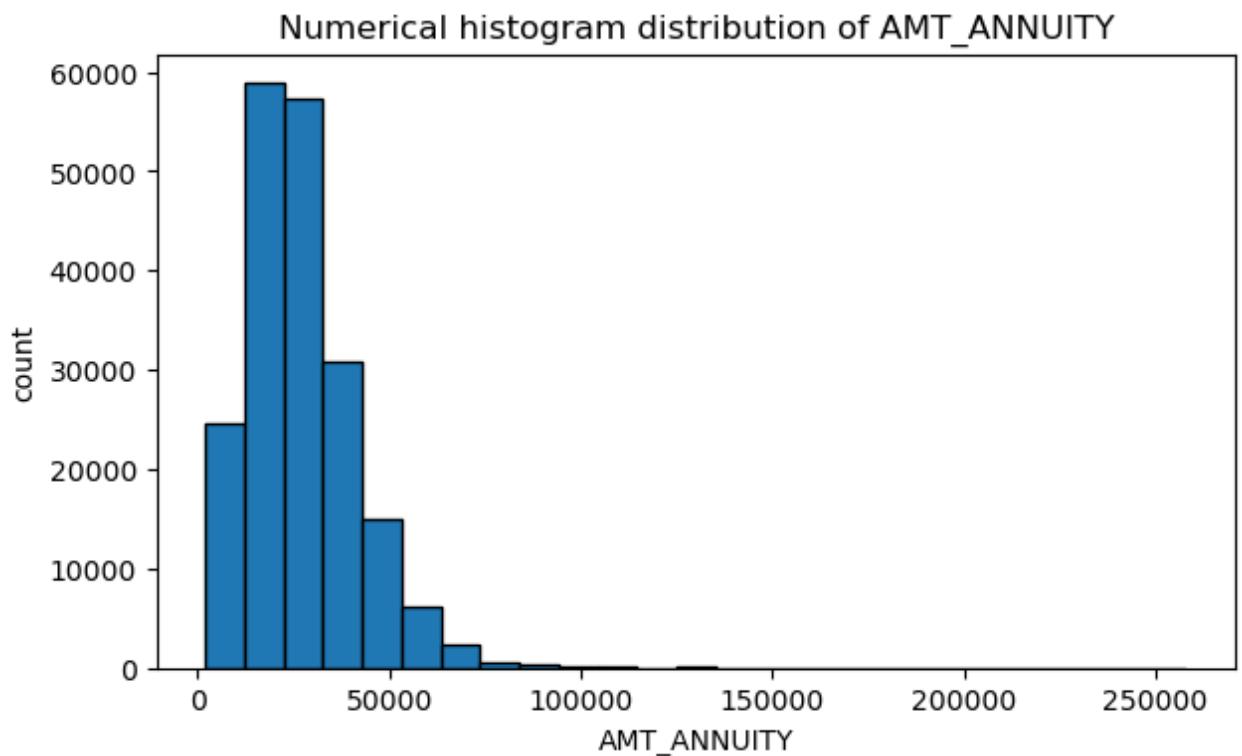


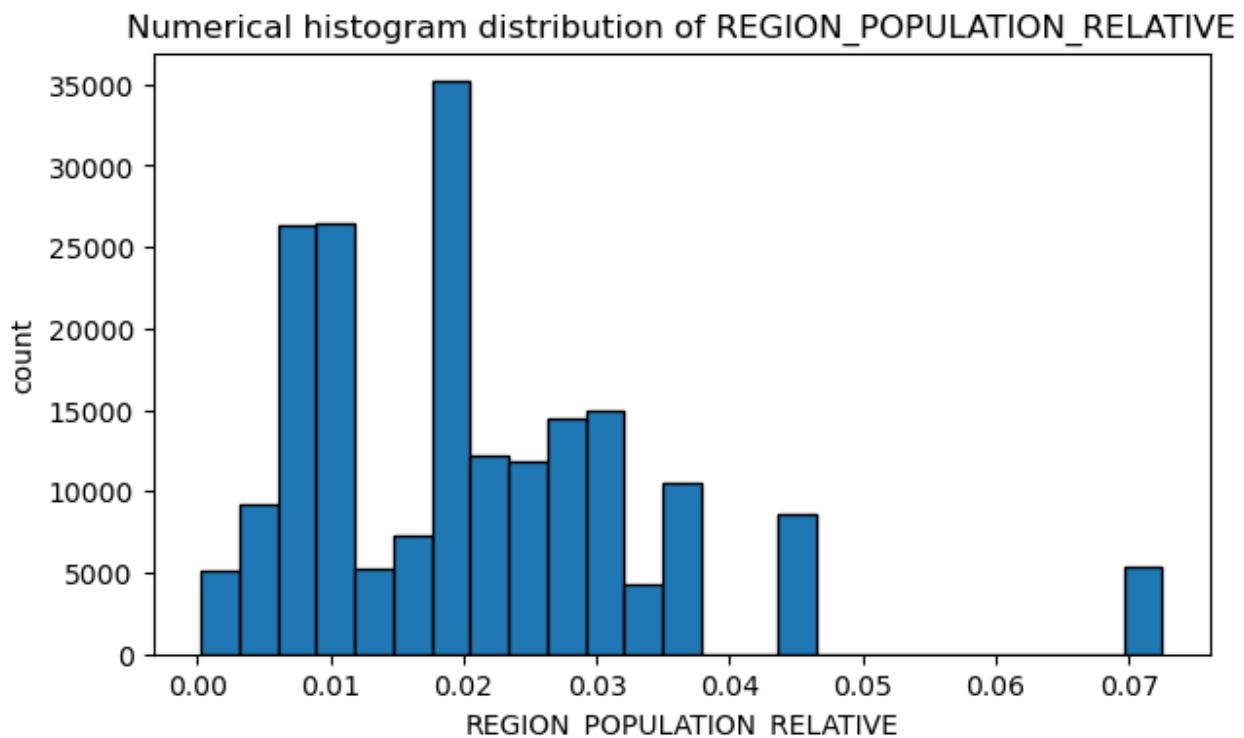
CNT_CHILDREN



Numerical histogram distribution of AMT_CREDIT





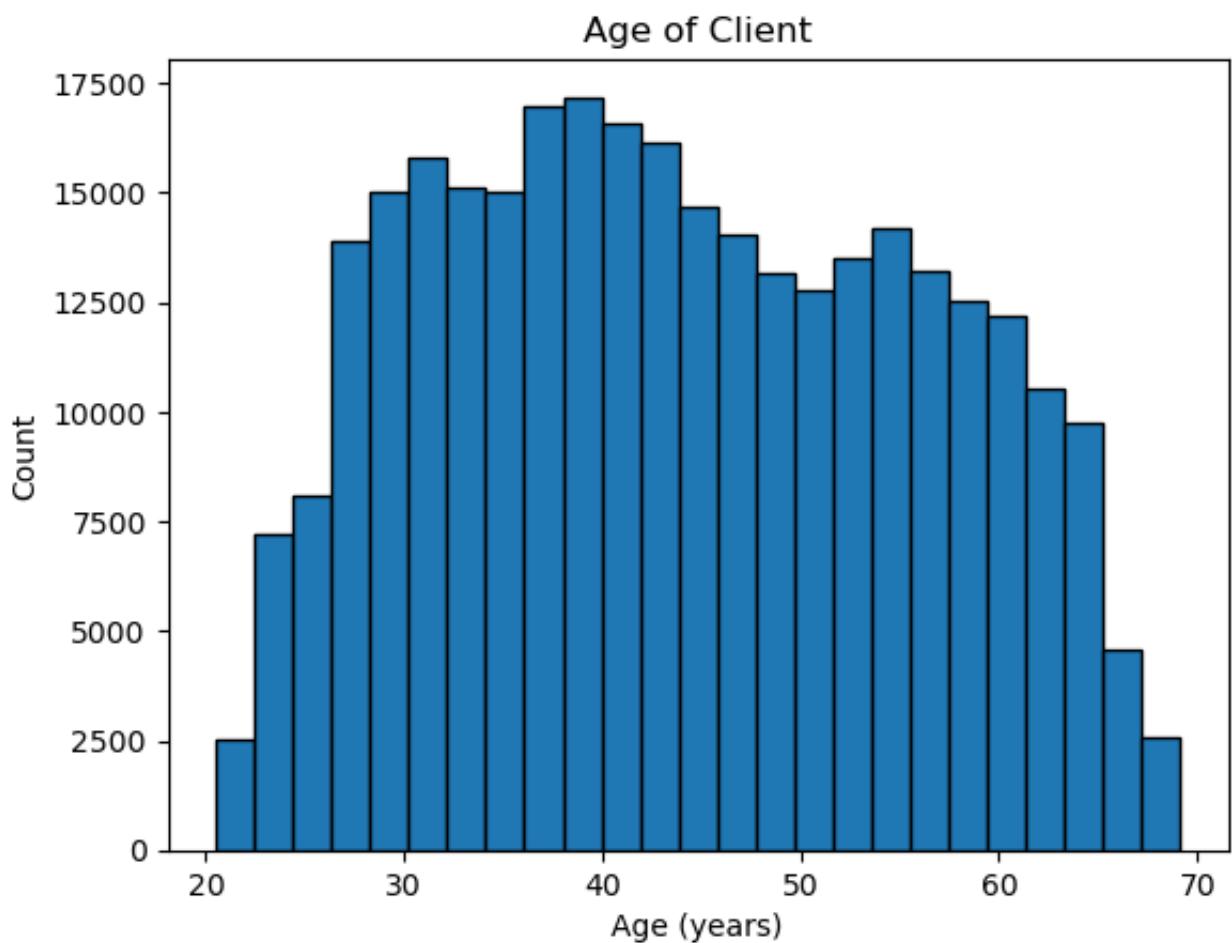


Histogram plot shows the distribution of data over a range. We have visualized each numerical data column's data distribution.

Applicants Age

In [32]:

```
plt.hist(datasets["application_train"]['DAYS_BIRTH'] / -365, edgecolor = 'black')
plt.title('Age of Client')
plt.xlabel('Age (years)')
plt.ylabel('Count')
plt.show()
```

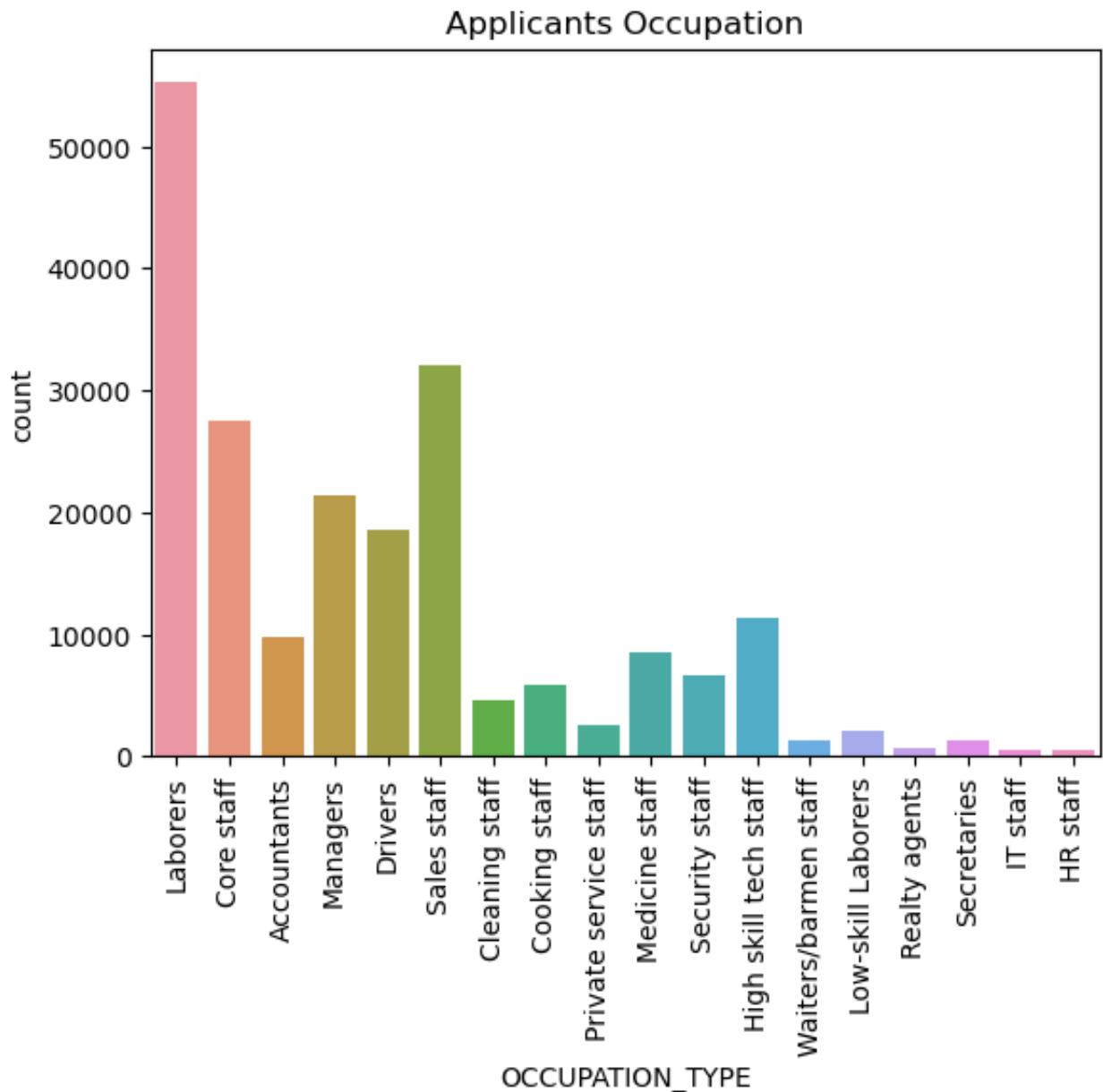


Here we can conclude that people of age 30-50 take more loan applications

Applicants occupations

In [197...]

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



Laborers require more loans as compared to other occupation type people

Dataset questions

Unique record for each SK_ID_CURR

```
In [209...]: datasets.keys()
```

```
Out[209...]: dict_keys(['application_train', 'application_test', 'bureau', 'bureau_balance', 'credit_card_balance', 'installments_payments', 'previous_application', 'POS_CASH_balance'])
```

```
In [210...]: len(datasets["application_train"]["SK_ID_CURR"].unique()) == datasets["app..."]
```

```
Out[210... True
```

```
In [211... np.intersect1d(datasets["application_train"]["SK_ID_CURR"], datasets["appli...
```

```
Out[211... array([], dtype=int64)
```

```
In [212... datasets["application_test"].shape
```

```
Out[212... (48744, 121)
```

```
In [213... datasets["application_train"].shape
```

```
Out[213... (307511, 122)
```

Input Features -

```
In [38]: numerical_features = X_train.select_dtypes(include = ['int64', 'float64'])
categorical_features = X_train.select_dtypes(include = ['object', 'bool'])
print(f"\nNumerical features : {list(numerical_features)}")
print(f"\nCategorical features : {list(categorical_features)}")
```

Numerical features : ['SK_ID_CURR', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE', 'REGION_POPULATION_RELATIVE', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'DAYS_ID_PUBLISH', 'OWN_CAR_AGE', 'FLAG_MOBIL', 'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE', 'FLAG_CONT_MOBILE', 'FLAG_PHONE', 'FLAG_EMAIL', 'CNT_FAM_MEMBERS', 'REGION_RATING_CLIENT', 'REGION_RATING_CLIENT_W_CITY', 'HOUR_APPR_PROCESS_START', 'REG_REGION_NOT_LIVE_REGION', 'REG_REGION_NOT_WORK_REGION', 'LIVE_REGION_NOT_WORK_REGION', 'REG_CITY_NOT_LIVE_CITY', 'REG_CITY_NOT_WORK_CITY', 'LIVE_CITY_NOT_WORK_CITY', 'EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3', 'APARTMENTS_AVG', 'BASEMENTAREA_AVG', 'YEARS_BEGINEXPLUATATION_AVG', 'YEARS_BUILD_AVG', 'COMMONAREA_AVG', 'ELEVATORS_AVG', 'ENTRANCES_AVG', 'FLOORSMAX_AVG', 'FLOORSMIN_AVG', 'LANDAREA_AVG', 'LIVINGAPARTMENTS_AVG', 'LIVINGAREA_AVG', 'NONLIVINGAPARTMENTS_AVG', 'NONLIVINGAREA_AVG', 'APARTMENTS_MODE', 'BASEMENTAREA_MODE', 'YEARS_BEGINEXPLUATATION_MODE', 'YEARS_BUILD_MODE', 'COMMONAREA_MODE', 'ELEVATORS_MODE', 'ENTRANCES_MODE', 'FLOORSMAX_MODE', 'FLOORSMIN_MODE', 'LANDAREA_MODE', 'LIVINGAPARTMENTS_MODE', 'LIVINGAREA_MODE', 'NONLIVINGAPARTMENTS_MODE', 'NONLIVINGAREA_MODE', 'APARTMENTS_MEDI', 'BASEMENTAREA_MEDI', 'YEARS_BEGINEXPLUATATION_MEDI', 'YEARS_BUILD_MEDI', 'COMMONAREA_MEDI', 'ELEVATORS_MEDI', 'ENTRANCES_MEDI', 'FLOORSMAX_MEDI', 'FLOORSMIN_MEDI', 'LANDAREA_MEDI', 'LIVINGAPARTMENTS_MEDI', 'LIVINGAREA_MEDI', 'NONLIVINGAPARTMENTS_MEDI', 'NONLIVINGAREA_MEDI', 'TOTALAREA_MODE', 'OBS_30_CNT_SOCIAL_CIRCLE', 'DEF_30_CNT_SOCIAL_CIRCLE', 'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIRCLE', 'DAYS_LAST_PHONE_CHANGE', 'FLAG_DOCUMENT_2', 'FLAG_DOCUMENT_3', 'FLAG_DOCUMENT_4', 'FLAG_DOCUMENT_5', 'FLAG_DOCUMENT_6', 'FLAG_DOCUMENT_7', 'FLAG_DOCUMENT_8', 'FLAG_DOCUMENT_9', 'FLAG_DOCUMENT_10', 'FLAG_DOCUMENT_11', 'FLAG_DOCUMENT_12', 'FLAG_DOCUMENT_13', 'FLAG_DOCUMENT_14', 'FLAG_DOCUMENT_15', 'FLAG_DOCUMENT_16', 'FLAG_DOCUMENT_17', 'FLAG_DOCUMENT_18', 'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20', 'FLAG_DOCUMENT_21', 'AMT_REQ_CREDIT_BUREAU_HOUR', 'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK', 'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QRT', 'AMT_REQ_CREDIT_BUREAU_YEAR']

Categorical features : ['NAME_CONTRACT_TYPE', 'CODE_GENDER', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'NAME_TYPE_SUITE', 'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'OCCUPATION_TYPE', 'WEEKDAY_APPR_PROCESS_START', 'ORGANIZATION_TYPE', 'FONDKAPREMONT_MODE', 'HOUSEYPE_MODE', 'WALLSMATERIAL_MODE', 'EMERGENCYSTATE_MODE']

```
In [61]: len(list(numerical_features))
```

Out[61]: 105

```
In [62]: len(list(categorical_features))
```

Out[62]: 16

```
In [158...]: total_input_features = len(list(numerical_features)) + len(list(categorical_features))
```

Out[158...]: 121

Modeling

Baseline Logistic Regression -

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

$$\operatorname{argmin}_{\theta} [\text{CXE}] = \operatorname{argmin}_{\theta} \left[-\frac{1}{m} \sum_{i=1}^m \left[y^{(i)} \log(\hat{p}^{(i)}) + (1 - y^{(i)}) \log(1 - \hat{p}^{(i)}) \right] \right]$$

The corresponding gradient function of partial derivatives is as follows (after a little bit of math):

$$\nabla_{\text{CXE}}(\theta) = \begin{pmatrix} \frac{\partial}{\partial \theta_0} \text{CXE}(\theta) \\ \frac{\partial}{\partial \theta_1} \text{CXE}(\theta) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \text{CXE}(\theta) \end{pmatrix} = \frac{2}{m} \mathbf{X}^T \cdot (\hat{\mathbf{p}}_y - \mathbf{y})$$

For completeness learning a binomial logistic regression model via gradient descent would use the following step iteratively:

$$\theta^{(\text{next step})} = \theta - \eta \nabla_{\text{CXE}}(\theta)$$

```
In [39]: num_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('imputer', SimpleImputer(strategy = 'median'))
])

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



```
In [197... results = pd.DataFrame(columns = ["Pipeline", "Dataset", "TrainAcc", "Valid
```



```
In [198... def model_logreg(X_train):

    data_pipeline = ColumnTransformer([
        ("num_pipeline", num_pipeline, numerical_features),
        ("cat_pipeline", cat_pipeline, categorical_features)], remainder =
```

$$X_{\text{train}}_{\text{transformed}} = \text{data_pipeline}.fit_transform(X_{\text{train}})$$

$$\text{column_names} = \text{list}(\text{numerical_features}) + \backslash$$

```

        list(data_pipeline.transformers_[1][1].named_steps["ohe"]

X_train_transformed_df = pd.DataFrame(X_train_transformed, columns=co

display(X_train_transformed_df.head())

clf_pipe = make_pipeline(data_pipeline, LogisticRegression())
clf_pipe.fit(X_train, y_train)

train_acc = clf_pipe.score(X_train, y_train)
validAcc = clf_pipe.score(X_valid, y_valid)
testAcc = clf_pipe.score(X_test, y_test)

## Plotting AUC / ROC
ns_probs = [0 for _ in range(len(y_test))]
lr_probs = clf_pipe.predict_proba(X_test)
# keep probabilities for the positive outcome only
lr_probs = lr_probs[:, 1]
# calculate scores
ns_auc = roc_auc_score(y_test, ns_probs)
lr_auc = roc_auc_score(y_test, lr_probs)

print('No Skill: ROC AUC=% .3f' % (ns_auc))
print('Logistic: ROC AUC=% .3f' % (lr_auc))
# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(y_test, lr_probs)
# plot the roc curve for the model
plt.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
plt.plot(lr_fpr, lr_tpr, marker='.', label='Logistic')
# axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# show the legend
plt.legend()
# show the plot
plt.show()

train_roc = roc_auc_score(y_train, clf_pipe.predict_proba(X_train)[:, 1])
test_roc = roc_auc_score(y_test, clf_pipe.predict_proba(X_test)[:, 1])
valid_roc = roc_auc_score(y_valid, clf_pipe.predict_proba(X_valid)[:, 1])

results.loc[len(results)] = ["Baseline Logistic Regression", "HCDR", f"{validAcc*100:.2f}%", f"{testAcc*100:.2f}%"]

display(results)
return clf_pipe

```

In [199...]

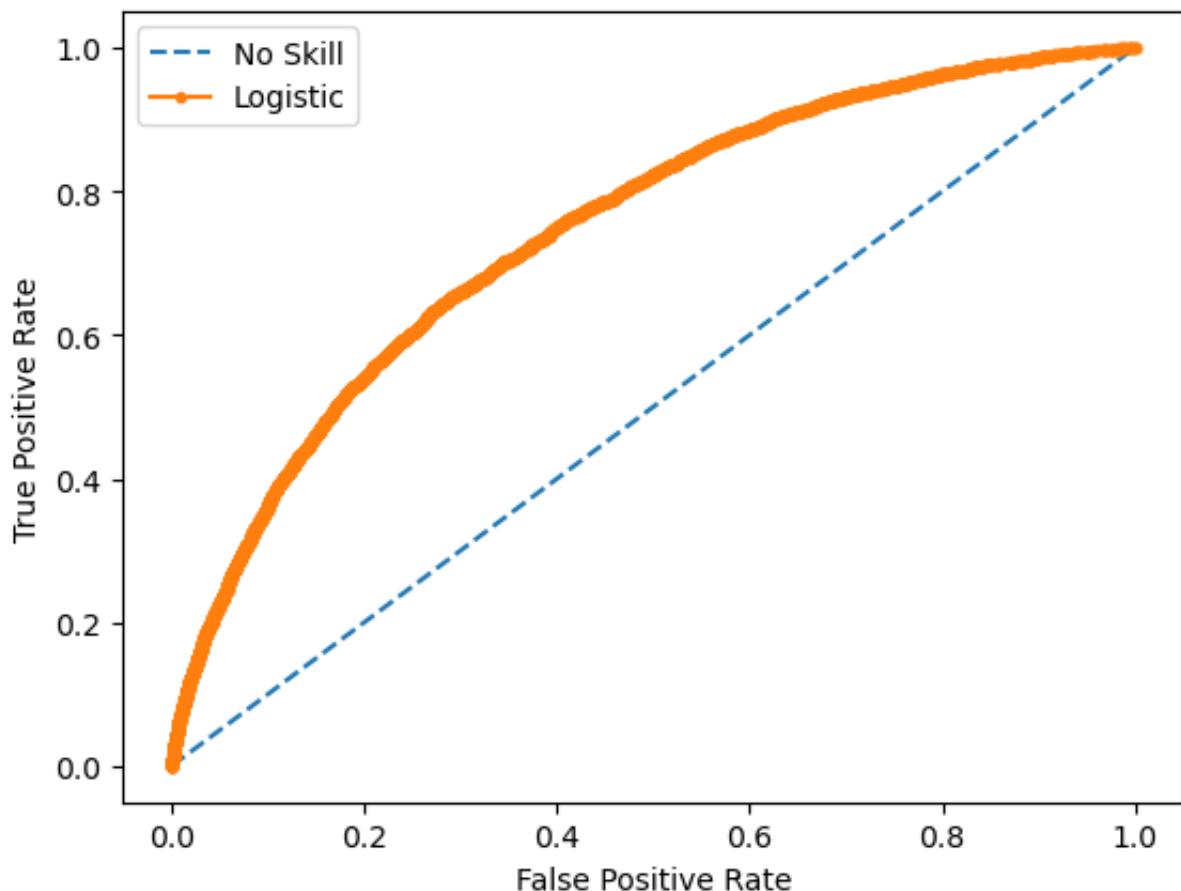
logreg = model_logreg(X_train)

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_
0	-1.624501	-0.578880	-0.528796	1.188441	0.175508	
1	0.560257	0.810033	0.749824	1.304588	1.218472	
2	-0.811727	0.810033	0.110514	-0.452448	-0.310068	
3	-0.776179	-0.578880	0.749824	1.188441	0.529898	
4	0.911520	0.810033	-0.315693	0.559617	-0.197845	

5 rows × 245 columns

No Skill: ROC AUC=0.500

Logistic: ROC AUC=0.745



	Pipeline	Dataset	TrainAcc	ValidAcc	TestAcc	TrainROC	TestROC	ValidROC
0	Baseline Logistic Regression	HCDR	91.95%	91.77%	91.93%	0.7491	0.7451	0.7407

Baseline Decision Tree Classifier -

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 : $\text{sum}(y - \text{prediction})^2$

Classification : $G = \text{sum}(\text{pk} * (1 - \text{pk}))$

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.

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

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

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:

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

fi_i is feature importance for i^{th} feature

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

$$normfi_i = \frac{fi_i}{\sum_{j \in \text{all features}} fi_j}$$

Reference: <https://towardsdatascience.com/the-mathematics-of-decision-trees-random-forest-and-feature-importance-in-scikit-learn-and-spark-f2861df67e3#:~:text=Feature%20importance%20is%20calculated%20as,the%20more%20it>

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.

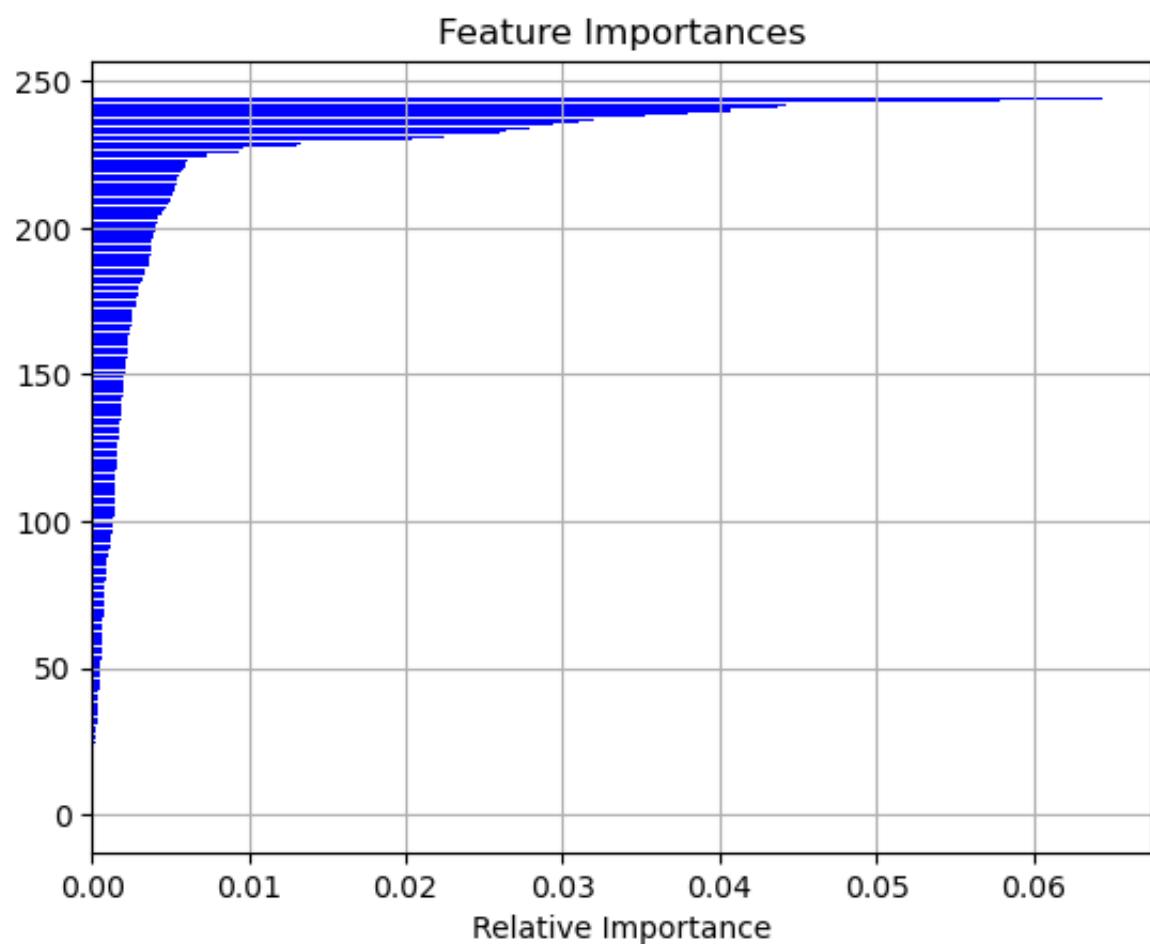
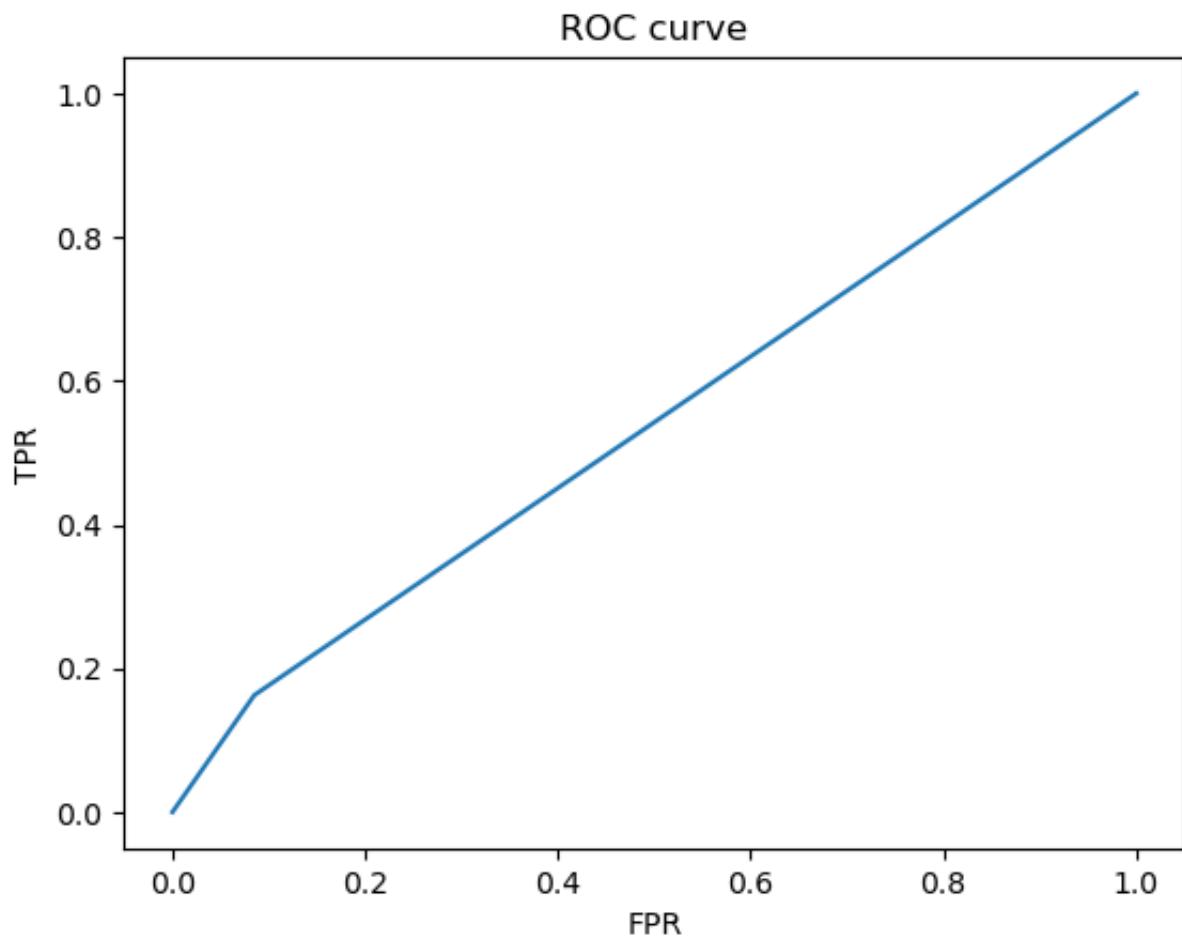
In [200...]

```
def model_dt(X_train):  
  
    tree = DecisionTreeClassifier(criterion='gini', random_state = 42)  
  
    data_pipeline_dt = make_pipeline(data_pipeline, tree)  
  
    data_pipeline_dt.fit(X_train, y_train)  
  
    train_acc = data_pipeline_dt.score(X_train, y_train)  
    validAcc = data_pipeline_dt.score(X_valid, y_valid)  
    testAcc = data_pipeline_dt.score(X_test, y_test)  
  
    predictions = data_pipeline_dt.predict_proba(X_test)  
    print ("Score",roc_auc_score(y_test, predictions[:,1]))  
  
    fpr, tpr, _ = roc_curve(y_test, predictions[:,1])  
  
    plt.clf()  
    plt.plot(fpr, tpr)  
    plt.xlabel('FPR')  
    plt.ylabel('TPR')  
    plt.title('ROC curve')  
    plt.show()  
  
    features = X_train.columns  
    importances = tree.feature_importances_  
    indices = np.argsort(importances)  
    top_feature = indices[0]  
  
    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();  
  
    print(f"The feature having highest importance is {features[top_feature]}")  
  
    train_roc = roc_auc_score(y_train, data_pipeline_dt.predict_proba(X_train))  
    test_roc = roc_auc_score(y_test, data_pipeline_dt.predict_proba(X_test))  
    valid_roc = roc_auc_score(y_valid, data_pipeline_dt.predict_proba(X_valid))  
  
    results.loc[len(results)] = ["Baseline Decision Tree", "HCDR", f"{train_acc:.8f}", f"{validAcc*100:.2f}%", f"{testAcc*100:.2f}%"]  
  
    display(results)
```

In [201...]

```
model_dt(X_train)
```

```
Score 0.5388834038729503
```



The feature having highest importance is NAME_EDUCATION_TYPE

	Pipeline	Dataset	TrainAcc	ValidAcc	TestAcc	TrainROC	TestROC	ValidROC
0	Baseline Logistic Regression	HCDR	91.95%	91.77%	91.93%	0.7491	0.7451	0.7407
1	Baseline Decision Tree	HCDR	100.00%	85.08%	85.46%	1.0	0.5389	0.5333

Baseline Random Forest Classifier -

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_estimators, 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.

Feature Importance

Yet another great quality of Random Forests is that they make it easy to measure the relative importance of each feature. Scikit-Learn measures a feature's importance by looking at how much the tree nodes that use that feature reduce impurity on average (across all trees in the forest). More precisely, it is a weighted average, where each node's weight is equal to the number of training samples that are associated with it.

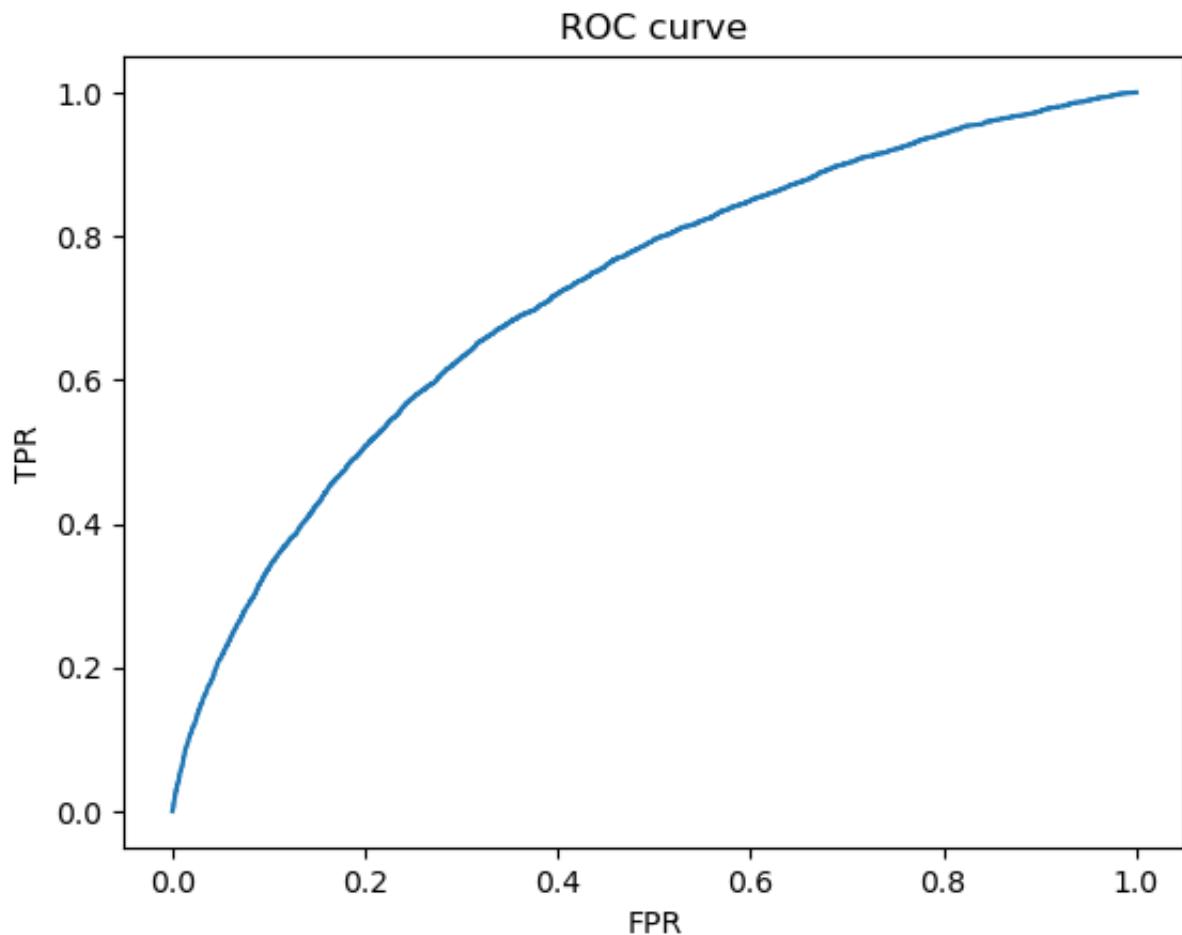
In [202...]

```
def model_rf(X_train):  
  
    RF = RandomForestClassifier(random_state = 42, n_estimators=20, criterion='gini')  
  
    data_pipeline_rf = make_pipeline(data_pipeline, RF)  
  
    data_pipeline_rf.fit(X_train, y_train)  
  
    train_acc = data_pipeline_rf.score(X_train, y_train)  
    validAcc = data_pipeline_rf.score(X_valid, y_valid)  
    testAcc = data_pipeline_rf.score(X_test, y_test)  
  
    predictions = data_pipeline_rf.predict_proba(X_test)  
    print ("Score",roc_auc_score(y_test, predictions[:,1]))  
  
    fpr, tpr, _ = roc_curve(y_test, predictions[:,1])  
  
    plt.clf()  
    plt.plot(fpr, tpr)  
    plt.xlabel('FPR')  
    plt.ylabel('TPR')  
    plt.title('ROC curve')  
    plt.show()  
  
    train_roc = roc_auc_score(y_train, data_pipeline_rf.predict_proba(X_train))  
    test_roc = roc_auc_score(y_test, data_pipeline_rf.predict_proba(X_test))  
    valid_roc = roc_auc_score(y_valid, data_pipeline_rf.predict_proba(X_valid))  
  
    results.loc[len(results)] = ["Baseline Random Forest", "HCDR", f"{train_acc:.2f}", f"{validAcc*100:.2f}%", f"{testAcc*100:.2f}"]  
  
    display(results)  
  
    return data_pipeline_rf,RF
```

In [203...]

```
rf_pipe, RF = model_rf(X_train)
```

Score 0.7202735379743133



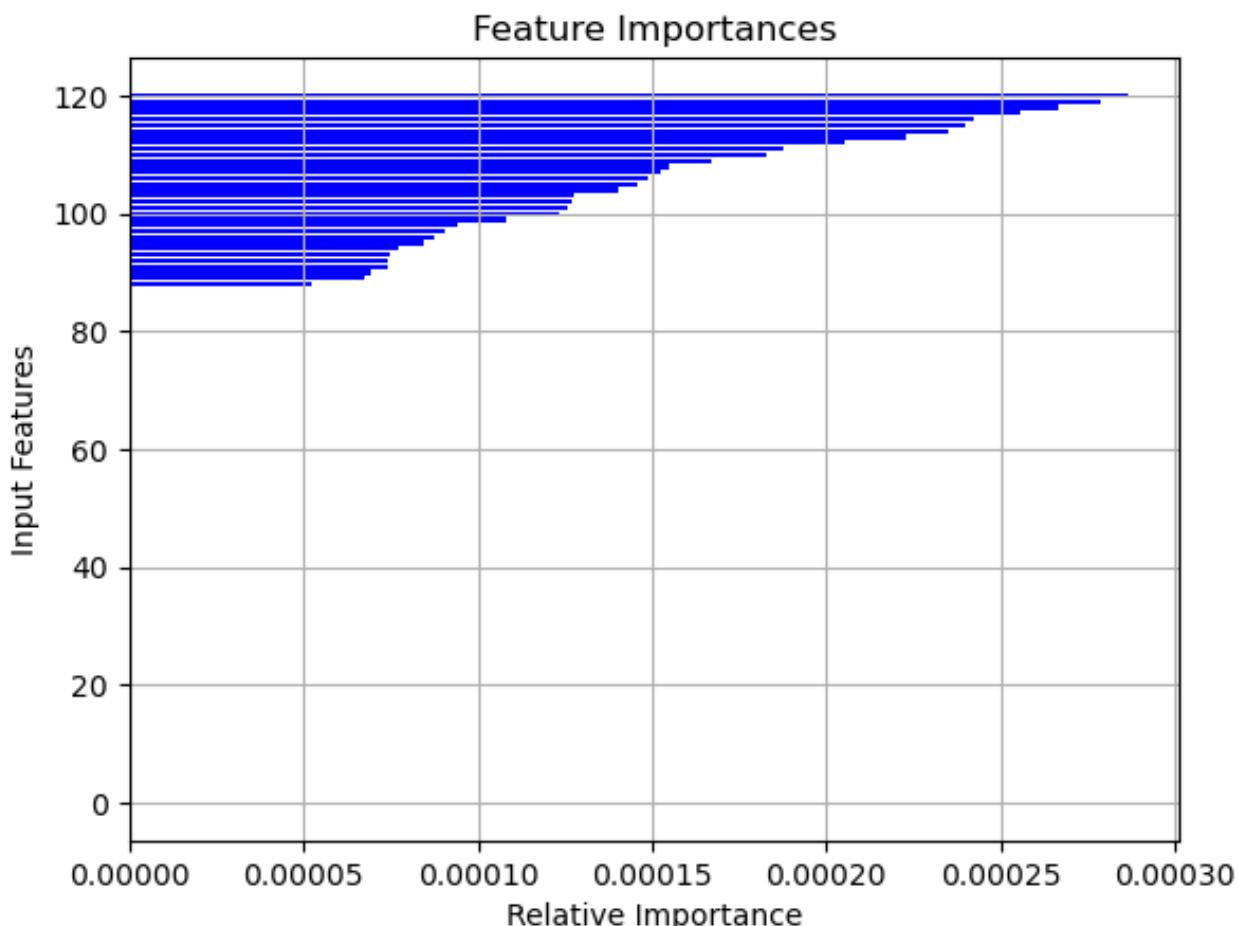
	Pipeline	Dataset	TrainAcc	ValidAcc	TestAcc	TrainROC	TestROC	ValidROC	
0	Baseline Logistic Regression	HCDR	91.95%	91.77%	91.93%	0.7491	0.7451	0.7407	
1	Baseline Decision Tree	HCDR	100.00%	85.08%	85.46%		1.0	0.5389	0.5333
2	Baseline Random Forest	HCDR	91.95%	91.78%	91.95%	0.7355	0.7203	0.7146	

Feature Importances of Random Forest Classifier

In [205...]

```
features = X_train.columns
#print(len(features))
importances = RF.feature_importances_
indices = np.argsort(importances)[:len(features)]
#print(len(indices))

plt.title('Feature Importances')
plt.barh(range(len(indices)), importances[indices], color='b', align='center')
# plt.yticks(range(len(indices)), [i for i in features])
plt.xlabel('Relative Importance')
plt.ylabel('Input Features')
plt.grid()
plt.show();
```



Feature Engineering

Bureau table

In [116...]

```
bureau_data = datasets['bureau']
```

In [117...]

```
bureau_data
```

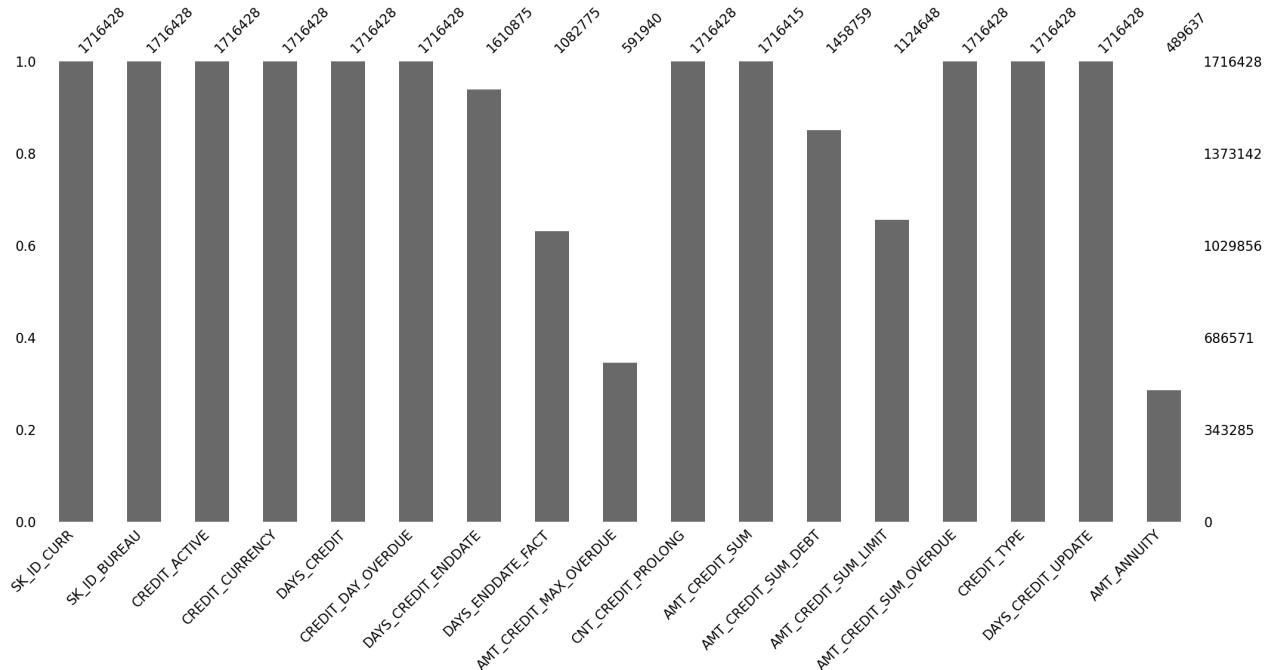
Out[117...]	SK_ID_CURR	SK_ID_BUREAU	CREDIT_ACTIVE	CREDIT_CURRENCY	DAY_S_CREDIT
0	215354	5714462	Closed	currency 1	-497
1	215354	5714463	Active	currency 1	-208
2	215354	5714464	Active	currency 1	-203
3	215354	5714465	Active	currency 1	-203
4	215354	5714466	Active	currency 1	-629
...
1716423	259355	5057750	Active	currency 1	-44
1716424	100044	5057754	Closed	currency 1	-2648
1716425	100044	5057762	Closed	currency 1	-1809
1716426	246829	5057770	Closed	currency 1	-1878
1716427	246829	5057778	Closed	currency 1	-463

1716428 rows × 17 columns

In [118...]: `bureau_data['DAY_S_CREDIT'].value_counts()`

```
Out[118...]:
-364    1330
-336    1248
-273    1238
-357    1218
-343    1203
...
-4      113
-3      74
-2      42
0       25
-1      17
Name: DAY_S_CREDIT, Length: 2923, dtype: int64
```

In [119...]: `msno.bar(bureau_data)`
`plt.show()`



```
In [120...]: corr = bureau_data.corr()['DAYS_CREDIT'].sort_values(ascending = False)
print('Most Positive Correlations:\n', corr.head(10))
print('\nMost Negative Correlations:\n', corr.tail(10))
```

Most Positive Correlations:

```
    DAYS_CREDIT           1.000000  
    DAYS_ENDDATE_FACT   0.875359  
    DAYS_CREDIT_UPDATE   0.688771  
    DAYS_CREDIT_ENDDATE 0.225682  
    AMT_CREDIT_SUM_DEBT 0.135397  
    AMT_CREDIT_SUM        0.050883  
    AMT_CREDIT_SUM_LIMIT 0.025140  
    SK_ID_BUREAU          0.013015  
    AMT_ANNUITY            0.005676  
    SK_ID_CURR             0.000266  
Name: DAYS_CREDIT, dtype: float64
```

Most Negative Correlations:

```
AMT_CREDIT_SUM_DEBT           0.135397  
AMT_CREDIT_SUM                 0.050883  
AMT_CREDIT_SUM_LIMIT           0.025140  
SK_ID_BUREAU                   0.013015  
AMT_ANNUITY                     0.005676  
SK_ID_CURR                      0.000266  
AMT_CREDIT_SUM_OVERDUE         -0.000383  
AMT_CREDIT_MAX_OVERDUE          -0.014724  
CREDIT_DAY_OVERDUE              -0.027266  
CNT_CREDIT_PROLONG                -0.030460  
Name: DAYS_CREDIT, dtype: float64
```

```
In [121]: bureau_corr = pd.DataFrame(corr, columns = [ 'DAYS_CREDIT' ])
bureau_corr
```

Out[121...]

	DAYS_CREDIT
 DAYS_CREDIT	1.000000
 DAYS_ENDDATE_FACT	0.875359
 DAYS_CREDIT_UPDATE	0.688771
 DAYS_CREDIT_ENDDATE	0.225682
 AMT_CREDIT_SUM_DEBT	0.135397
 AMT_CREDIT_SUM	0.050883
 AMT_CREDIT_SUM_LIMIT	0.025140
 SK_ID_BUREAU	0.013015
 AMT_ANNUITY	0.005676
 SK_ID_CURR	0.000266
 AMT_CREDIT_SUM_OVERDUE	-0.000383
 AMT_CREDIT_MAX_OVERDUE	-0.014724
 CREDIT_DAY_OVERDUE	-0.027266
 CNT_CREDIT_PROLONG	-0.030460

In [122...]

```
bureau_corr = bureau_corr[:20].dropna()
sns.heatmap(bureau_corr, annot = True ,fmt=' .2f ')
plt.show()
```



```
In [21]: bureau_data['CREDIT_TYPE'].value_counts()
```

```
Out[21]: Consumer credit           1251615
          Credit card              402195
          Car loan                 27690
          Mortgage                18391
          Microloan               12413
          Loan for business development   1975
          Another type of loan        1017
          Unknown type of loan       555
          Loan for working capital replenishment 469
          Cash loan (non-earmarked)    56
          Real estate loan           27
          Loan for the purchase of equipment 19
          Loan for purchase of shares (margin lending) 4
          Mobile operator loan        1
          Interbank credit            1
          Name: CREDIT_TYPE, dtype: int64
```

```
In [22]: bureau_data['CREDIT_ACTIVE'].value_counts()
```

```
Out[22]: Closed      1079273
          Active      630607
          Sold        6527
          Bad debt     21
          Name: CREDIT_ACTIVE, dtype: int64
```

```
In [123... bureau_data['CREDIT_ACTIVE_CLASSIFY'] = bureau_data['CREDIT_ACTIVE']

def classify(x):
    if x == 'Closed':
        y = 0
    else:
        y = 1
    return y

bureau_data['CREDIT_ACTIVE_CLASSIFY'] = bureau_data.apply(lambda x: classify(x))
```

```
In [146... bureau_data['CREDIT_ACTIVE_CLASSIFY'].value_counts()
```

```
Out[146]: 0      1079273
           1      637155
           Name: CREDIT_ACTIVE_CLASSIFY, dtype: int64
```

```
In [147... active_loan = bureau_data.groupby(by = ['SK_ID_CURR'])['CREDIT_ACTIVE_CLASSIFY'].mean()
active_loan
```

Out[147...]

	SK_ID_CURR	ACTIVE_LOANS_PERCENTAGE
0	100001	0.428571
1	100002	0.250000
2	100003	0.250000
3	100004	0.000000
4	100005	0.666667
...
305806	456249	0.153846
305807	456250	0.666667
305808	456253	0.500000
305809	456254	0.000000
305810	456255	0.454545

305811 rows × 2 columns

In [148...]

```
bureau_data = bureau_data.merge(active_loan, on = ['SK_ID_CURR'], how = 'left')
bureau_data
```

Out[148...]

	SK_ID_CURR	SK_ID_BUREAU	CREDIT_ACTIVE	CREDIT_CURRENCY	DAYS_CREDIT
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
...
1716423	259355	5057750	Active	currency 1	-44
1716424	100044	5057754	Closed	currency 1	-2648
1716425	100044	5057762	Closed	currency 1	-1809
1716426	246829	5057770	Closed	currency 1	-1878
1716427	246829	5057778	Closed	currency 1	-463

1716428 rows × 19 columns

```
In [149...]: bureau_data.drop(['AMT_ANNUITY', 'DAYS_ENDDATE_FACT', 'CREDIT_CURRENCY', 'CREDI
```

```
In [150...]: bureau_data
```

```
Out[150...]:
```

	SK_ID_CURR	SK_ID_BUREAU	CREDIT_ACTIVE	DAYS_CREDIT	CREDIT_DAY_OVERI
0	215354	5714462	Closed	-497	
1	215354	5714463	Active	-208	
2	215354	5714464	Active	-203	
3	215354	5714465	Active	-203	
4	215354	5714466	Active	-629	
...
1716423	259355	5057750	Active	-44	
1716424	100044	5057754	Closed	-2648	
1716425	100044	5057762	Closed	-1809	
1716426	246829	5057770	Closed	-1878	
1716427	246829	5057778	Closed	-463	

1716428 rows × 15 columns

```
In [151...]: bureau_data['CNT_CREDIT_PROLONG'].value_counts()
```

```
Out[151...]:
```

0	1707314
1	7620
2	1222
3	191
4	54
5	21
9	2
6	2
8	1
7	1

Name: CNT_CREDIT_PROLONG, dtype: int64

```
In [152...]: bureau_data['AMT_CREDIT_MAX_OVERDUE'].fillna(0, inplace = True)
```

```
In [153...]: bureau_data['AMT_CREDIT_SUM_LIMIT'].isnull().value_counts()
```

```
Out[153]: False    1124648
      True     591780
      Name: AMT_CREDIT_SUM_LIMIT, dtype: int64
```

```
In [154]: bureau_data['AMT_CREDIT_SUM'].fillna(0, inplace = True)
bureau_data['AMT_CREDIT_SUM_LIMIT'].fillna(0, inplace = True)
```

```
In [155]: bureau_data[['AMT_CREDIT_SUM', 'AMT_CREDIT_SUM_DEBT', 'AMT_CREDIT_SUM_LIMIT']]
```

	AMT_CREDIT_SUM	AMT_CREDIT_SUM_DEBT	AMT_CREDIT_SUM_LIMIT
0	91323.00	0.0	0.0
1	225000.00	171342.0	0.0
2	464323.50	NaN	0.0
3	90000.00	NaN	0.0
4	2700000.00	NaN	0.0
...
1716423	11250.00	11250.0	0.0
1716424	38130.84	0.0	0.0
1716425	15570.00	NaN	0.0
1716426	36000.00	0.0	0.0
1716427	22500.00	0.0	0.0

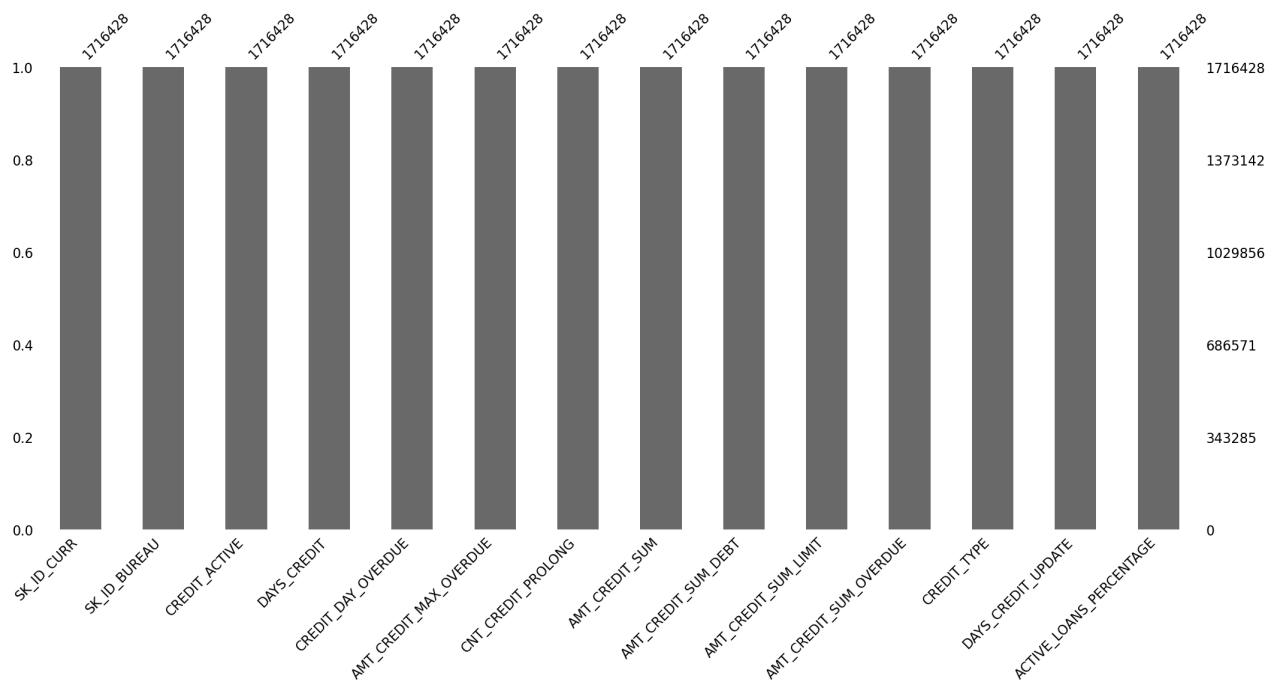
1716428 rows × 3 columns

```
In [156]: bureau_data['AMT_CREDIT_SUM_DEBT'].fillna(bureau_data['AMT_CREDIT_SUM'], inplace = True)
```

```
In [157]: difference = bureau_data['AMT_CREDIT_SUM'] - bureau_data['AMT_CREDIT_SUM_DEBT']
bureau_data['AMT_CREDIT_SUM_LIMIT'].fillna(difference, inplace = True)
```

```
In [158]: bureau_data.drop('DAYS_CREDIT_ENDDATE', axis = 1, inplace = True)
```

```
In [159]: msno.bar(bureau_data)
plt.show()
```



Our Model Pipeline

In [127...]

```
results = pd.DataFrame(columns = ["Pipeline",
                                    "Dataset",
                                    "TrainAcc",
                                    "ValidAcc",
                                    "TestAcc",
                                    "TrainROC",
                                    "TestROC",
                                    "ValidROC",
                                    "Feature Added"])
```

In [128...]

```
def model_logreg(train,feature):

    X = train.drop(['TARGET'], axis = 1)
    y = train["TARGET"]

    # Split the provided training data into training and validation and test sets
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
    X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.2)
    print(f"X train           shape: {X_train.shape}")
    print(f"X validation     shape: {X_valid.shape}")
    print(f"X test            shape: {X_test.shape}")

    numerical_features = X_train.select_dtypes(include = ['int64', 'float64'])
    categorical_features = X_train.select_dtypes(include = ['object', 'bool'])
    #print(f"\nNumerical features : {list(numerical_features)}")
    #print(f"\nCategorical features : {list(categorical_features)}")

    num_pipeline = Pipeline([
        ('scaler', StandardScaler()),
        ('imputer', SimpleImputer(strategy = 'median'))
```

```
])

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

data_pipeline = ColumnTransformer([
    ("num_pipeline", num_pipeline, numerical_features),
    ("cat_pipeline", cat_pipeline, categorical_features)], remainder = "drop")

X_train_transformed = data_pipeline.fit_transform(X_train)

column_names = list(numerical_features) + \
    list(data_pipeline.transformers_[1][1].named_steps["ohe"])

X_train_transformed_df = pd.DataFrame(X_train_transformed, columns=column_names)

display(X_train_transformed_df.head())

clf_pipe = make_pipeline(data_pipeline, LogisticRegression())
clf_pipe.fit(X_train, y_train)

train_acc = clf_pipe.score(X_train, y_train)
validAcc = clf_pipe.score(X_valid, y_valid)
testAcc = clf_pipe.score(X_test, y_test)

## Plotting AUC / ROC
ns_probs = [0 for _ in range(len(y_test))]
lr_probs = clf_pipe.predict_proba(X_test)
# keep probabilities for the positive outcome only
lr_probs = lr_probs[:, 1]
# calculate scores
ns_auc = roc_auc_score(y_test, ns_probs)
lr_auc = roc_auc_score(y_test, lr_probs)

print('No Skill: ROC AUC=% .3f' % (ns_auc))
print('Logistic: ROC AUC=% .3f' % (lr_auc))
# calculate roc curves
ns_fpr, ns_tpr, _ = roc_curve(y_test, ns_probs)
lr_fpr, lr_tpr, _ = roc_curve(y_test, lr_probs)
# plot the roc curve for the model
plt.plot(ns_fpr, ns_tpr, linestyle='--', label='No Skill')
plt.plot(lr_fpr, lr_tpr, marker='.', label='Logistic')
# axis labels
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
# show the legend
plt.legend()
# show the plot
plt.show()

train_roc = roc_auc_score(y_train, clf_pipe.predict_proba(X_train)[:, 1])
test_roc = roc_auc_score(y_test, clf_pipe.predict_proba(X_test)[:, 1])
valid_roc = roc_auc_score(y_valid, clf_pipe.predict_proba(X_valid)[:, 1])
```

```
results.loc[len(results)] = ["Logistic Regression", "HCDR", f"{train_acc:  
f}{validAcc*100:8.2f}%", f"{testAcc*100:8  
display(results)  
  
return clf_pipe
```

In []:

Feature 1 : Percentage of Active Loans

```
In [160... bureau_active_loans = pd.DataFrame(bureau_data[['SK_ID_CURR', 'ACTIVE_LOANS_
```

```
In [161... bureau_active_loans
```

```
Out[161...  
SK_ID_CURR ACTIVE_LOANS_PERCENTAGE  
0 215354 0.545455  
1 215354 0.545455  
2 215354 0.545455  
3 215354 0.545455  
4 215354 0.545455  
... ... ...  
1716423 259355 1.000000  
1716424 100044 0.545455  
1716425 100044 0.545455  
1716426 246829 0.258065  
1716427 246829 0.258065
```

1716428 rows × 2 columns

```
In [162... train = datasets['application_train']  
train
```

Out[162...]	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR
0	100002	1	Cash loans	M	N
1	100003	0	Cash loans	F	N
2	100004	0	Revolving loans	M	Y
3	100006	0	Cash loans	F	N
4	100007	0	Cash loans	M	N
...
307506	456251	0	Cash loans	M	N
307507	456252	0	Cash loans	F	N
307508	456253	0	Cash loans	F	N
307509	456254	1	Cash loans	F	N
307510	456255	0	Cash loans	F	N

307511 rows × 122 columns

```
In [167...]: train.drop(['DAYS_LAST_PHONE_CHANGE', 'OBS_30_CNT_SOCIAL_CIRCLE', 'OBS_60_CNT_SOCIAL_CIRCLE'], axis=1)
```

```
In [163...]: train = train.merge(bureau_active_loans, on = 'SK_ID_CURR', how = 'left')
```

```
In [164...]: train
```

Out[164...]	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR
0	100002	1	Cash loans	M	N
1	100002	1	Cash loans	M	N
2	100002	1	Cash loans	M	N
3	100002	1	Cash loans	M	N
4	100002	1	Cash loans	M	N
...
1509340	456255	0	Cash loans	F	N
1509341	456255	0	Cash loans	F	N
1509342	456255	0	Cash loans	F	N
1509343	456255	0	Cash loans	F	N
1509344	456255	0	Cash loans	F	N

1509345 rows × 123 columns

```
In [165]: train = train.loc[:, ~train.columns.str.startswith("FLAG_DOCUMENT_")]
```

```
In [168]: train
```

```
Out[168]:
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR
0	100002	1	Cash loans	M	N
1	100002	1	Cash loans	M	N
2	100002	1	Cash loans	M	N
3	100002	1	Cash loans	M	N
4	100002	1	Cash loans	M	N
...
1509340	456255	0	Cash loans	F	N
1509341	456255	0	Cash loans	F	N
1509342	456255	0	Cash loans	F	N
1509343	456255	0	Cash loans	F	N
1509344	456255	0	Cash loans	F	N

1509345 rows × 98 columns

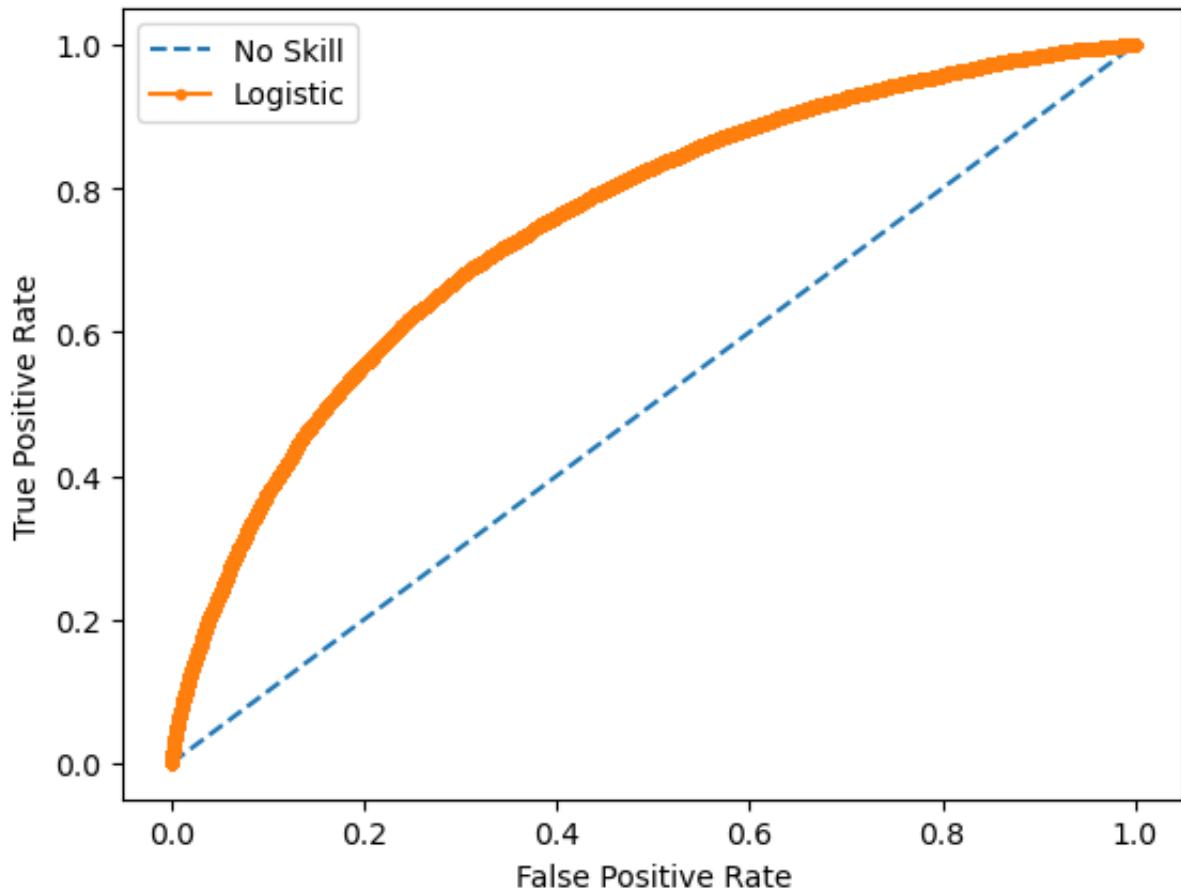
```
In [41]: active_loan_model = model_logreg(train, 'Active loan Percentage feature')
```

```
X train          shape: (965980, 97)
X validation    shape: (241496, 97)
X test          shape: (301869, 97)
```

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_
0	-0.714681	-0.585812	0.000407	-0.970848	-0.215580	
1	-1.062862	-0.585812	-0.191312	2.262115	1.247094	
2	1.282842	-0.585812	0.057922	-0.187606	-0.483416	
3	-0.163686	-0.585812	-0.287171	1.231743	0.630882	
4	-0.411456	-0.585812	1.150719	3.407022	2.134300	

5 rows × 220 columns

No Skill: ROC AUC=0.500
Logistic: ROC AUC=0.751



	Pipeline	Dataset	TrainAcc	ValidAcc	TestAcc	TrainROC	TestROC	ValidROC	Fea Ad
0	Logistic Regression	HCDR	92.13%	92.15%	92.13%	0.752	0.7505	0.7496	Active Percent fea

Feature 2: Credit income percentage feature

In [169...]

```
amt_income = train['AMT_INCOME_TOTAL']
amt_credit = train['AMT_CREDIT']
credit_income_perc = amt_income/amt_credit
train['CREDIT_INCOME_RATIO'] = credit_income_perc
train['CREDIT_INCOME_RATIO']
```

Out[169...]

0	0.498036
1	0.498036
2	0.498036
3	0.498036
4	0.498036
	...
1509340	0.233333
1509341	0.233333
1509342	0.233333
1509343	0.233333
1509344	0.233333

Name: CREDIT_INCOME_RATIO, Length: 1509345, dtype: float64

In [170...]

train

Out[170...]

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR
0	100002	1	Cash loans	M	N
1	100002	1	Cash loans	M	N
2	100002	1	Cash loans	M	N
3	100002	1	Cash loans	M	N
4	100002	1	Cash loans	M	N

1509340	456255	0	Cash loans	F	N
1509341	456255	0	Cash loans	F	N
1509342	456255	0	Cash loans	F	N
1509343	456255	0	Cash loans	F	N
1509344	456255	0	Cash loans	F	N

1509345 rows × 99 columns

In [44]:

credit_income_model = model_logreg(train, 'Credit Income Ratio feature')

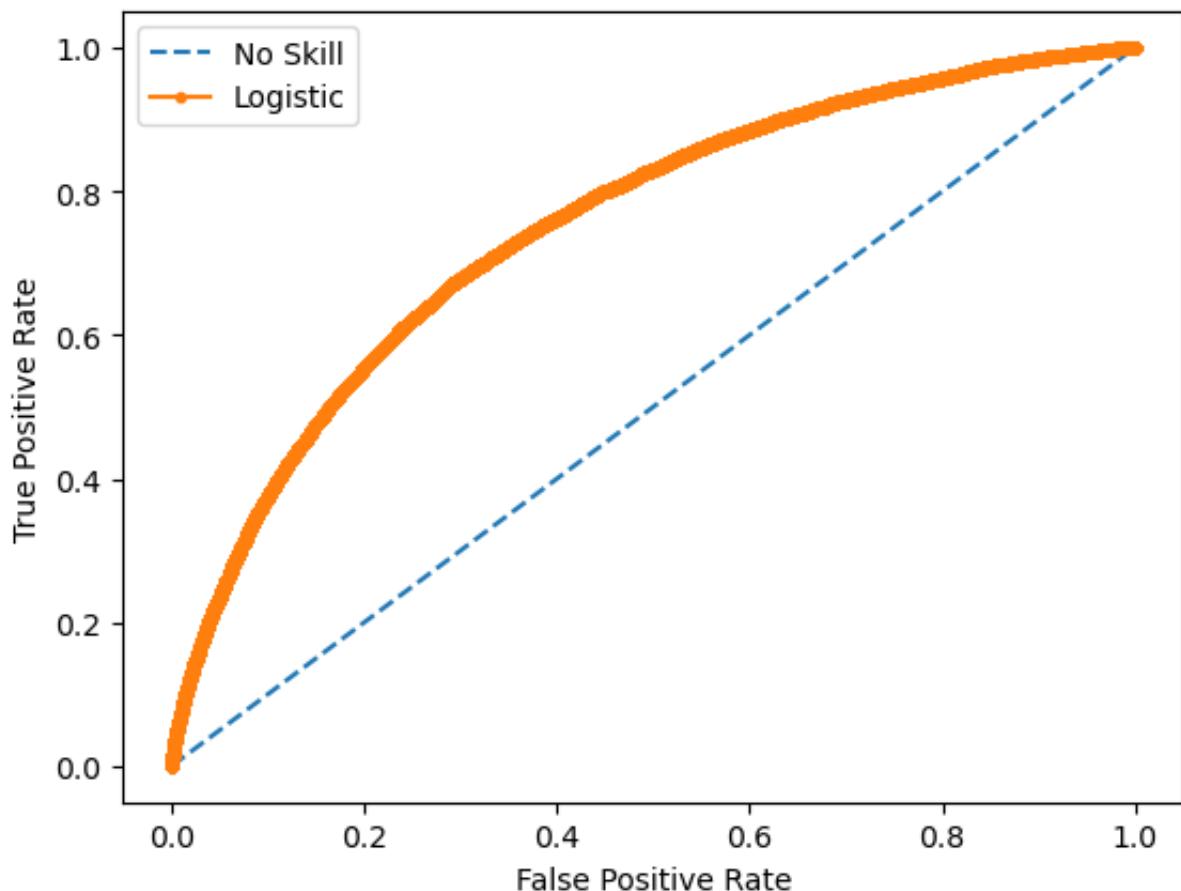
X train	shape: (965980, 98)
X validation	shape: (241496, 98)
X test	shape: (301869, 98)

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_
0	-0.714681	-0.585812	0.000407	-0.970848	-0.215580	
1	-1.062862	-0.585812	-0.191312	2.262115	1.247094	
2	1.282842	-0.585812	0.057922	-0.187606	-0.483416	
3	-0.163686	-0.585812	-0.287171	1.231743	0.630882	
4	-0.411456	-0.585812	1.150719	3.407022	2.134300	

5 rows × 221 columns

No Skill: ROC AUC=0.500

Logistic: ROC AUC=0.751



	Pipeline	Dataset	TrainAcc	ValidAcc	TestAcc	TrainROC	TestROC	ValidROC	Fea Ad
0	Logistic Regression	HCDR	92.13%	92.15%	92.13%	0.752	0.7505	0.7496	Active Percent fea
1	Logistic Regression	HCDR	92.13%	92.15%	92.13%	0.7526	0.7511	0.7502	Cr Inc F fea

Feature 3: Credit Annuity Ratio of the Application

How many years does it take for the borrower to repay the amount he asked for the application -

```
In [171...]: train['YEARS_TO_PAY'] = (train['AMT_CREDIT']/train['AMT_ANNUITY']).round()
```

```
In [172...]: train[['YEARS_TO_PAY']]
```

```
Out[172...]:
```

	YEARS_TO_PAY
0	16.0
1	16.0
2	16.0
3	16.0
4	16.0
...	...
1509340	14.0
1509341	14.0
1509342	14.0
1509343	14.0
1509344	14.0

1509345 rows × 1 columns

```
In [173...]: np.isinf(train[['YEARS_TO_PAY']]).values.sum()
```

```
Out[173...]: 0
```

```
In [174...]: train
```

Out[174...]	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR
0	100002	1	Cash loans	M	N
1	100002	1	Cash loans	M	N
2	100002	1	Cash loans	M	N
3	100002	1	Cash loans	M	N
4	100002	1	Cash loans	M	N
...
1509340	456255	0	Cash loans	F	N
1509341	456255	0	Cash loans	F	N
1509342	456255	0	Cash loans	F	N
1509343	456255	0	Cash loans	F	N
1509344	456255	0	Cash loans	F	N

1509345 rows × 100 columns

In [49]:

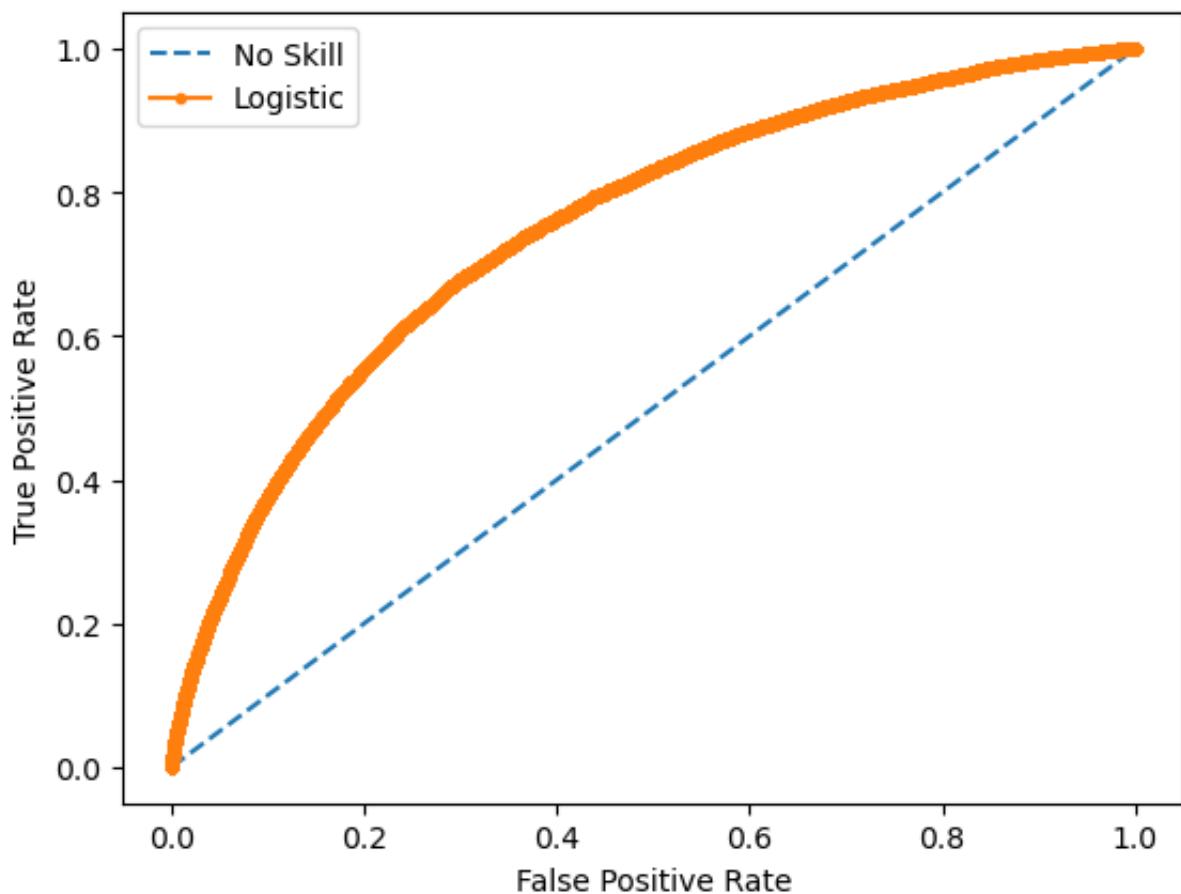
```
years_to_pay_model = model_logreg(train, 'Credit-Annuity Ratio of Current i
```

```
X train          shape: (965980, 99)
X validation    shape: (241496, 99)
X test          shape: (301869, 99)
```

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_
0	-0.714681	-0.585812	0.000407	-0.970848	-0.215580	
1	-1.062862	-0.585812	-0.191312	2.262115	1.247094	
2	1.282842	-0.585812	0.057922	-0.187606	-0.483416	
3	-0.163686	-0.585812	-0.287171	1.231743	0.630882	
4	-0.411456	-0.585812	1.150719	3.407022	2.134300	

5 rows × 222 columns

No Skill: ROC AUC=0.500
Logistic: ROC AUC=0.751



	Pipeline	Dataset	TrainAcc	ValidAcc	TestAcc	TrainROC	TestROC	ValidROC	Feat Ad
0	Logistic Regression	HCDR	92.13%	92.15%	92.13%	0.752	0.7505	0.7496	Active Percent fea
1	Logistic Regression	HCDR	92.13%	92.15%	92.13%	0.7526	0.7511	0.7502	C Inc F fea
2	Logistic Regression	HCDR	92.13%	92.15%	92.13%	0.7525	0.7511	0.7501	Cre Anr Rati Cur Applica

Feature 4: Income Annuity Ratio of the Application

```
In [175...]: train['INCOME_ANNUITY'] = (train['AMT_INCOME_TOTAL']/train['AMT_ANNUITY'])

In [176...]: train[['INCOME_ANNUITY']]
```

Out[176...]	INCOME_ANNUITY
0	8.0
1	8.0
2	8.0
3	8.0
4	8.0
...	...
1509340	3.0
1509341	3.0
1509342	3.0
1509343	3.0
1509344	3.0

1509345 rows × 1 columns

In [177...]: train

Out[177...]	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR
0	100002	1	Cash loans	M	N
1	100002	1	Cash loans	M	N
2	100002	1	Cash loans	M	N
3	100002	1	Cash loans	M	N
4	100002	1	Cash loans	M	N
...
1509340	456255	0	Cash loans	F	N
1509341	456255	0	Cash loans	F	N
1509342	456255	0	Cash loans	F	N
1509343	456255	0	Cash loans	F	N
1509344	456255	0	Cash loans	F	N

1509345 rows × 101 columns

In [53]: income_annuity_model = model_logreg(train, 'Income-Annuity Ratio of Current')

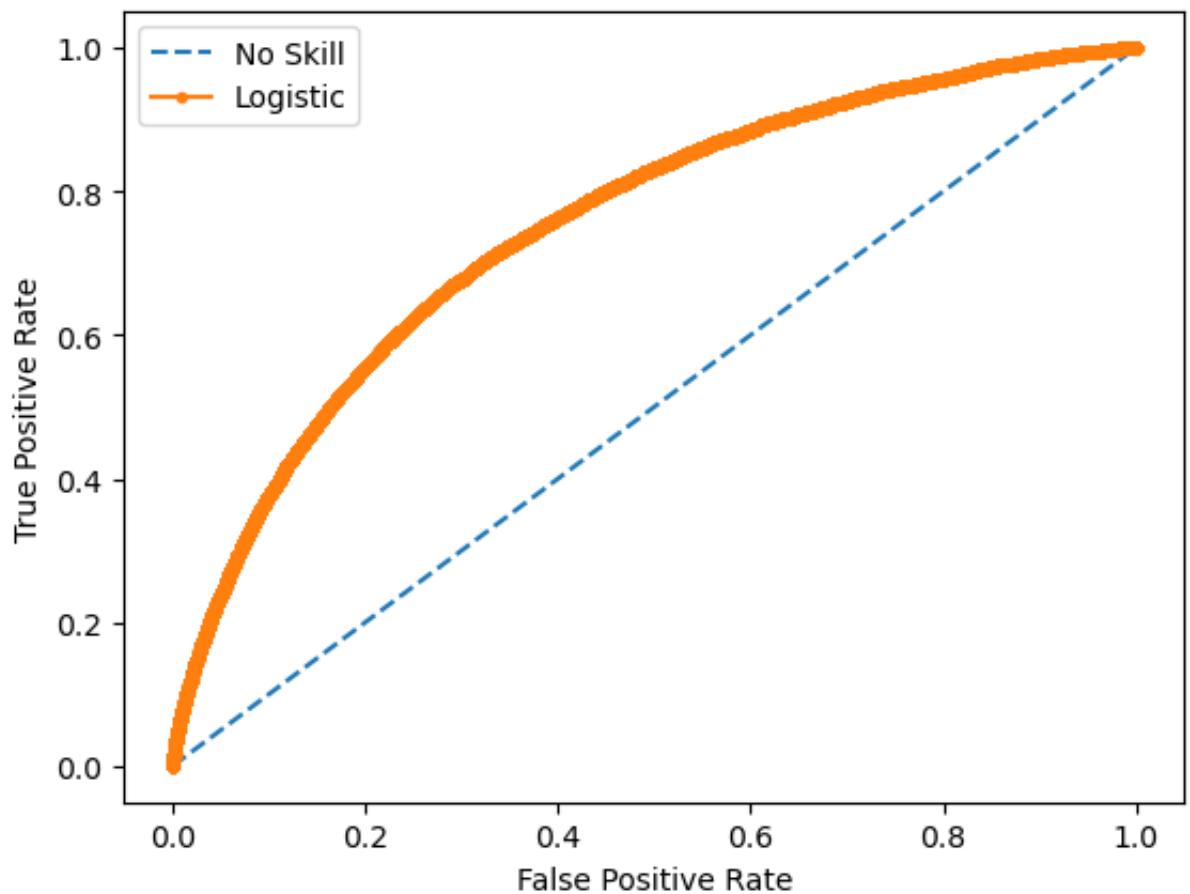
```
X train          shape: (965980, 100)
X validation    shape: (241496, 100)
X test          shape: (301869, 100)
```

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_
0	-0.714681	-0.585812	0.000407	-0.970848	-0.215580	
1	-1.062862	-0.585812	-0.191312	2.262115	1.247094	
2	1.282842	-0.585812	0.057922	-0.187606	-0.483416	
3	-0.163686	-0.585812	-0.287171	1.231743	0.630882	
4	-0.411456	-0.585812	1.150719	3.407022	2.134300	

5 rows × 223 columns

No Skill: ROC AUC=0.500

Logistic: ROC AUC=0.751



	Pipeline	Dataset	TrainAcc	ValidAcc	TestAcc	TrainROC	TestROC	ValidROC	Feat Ad
0	Logistic Regression	HCDR	92.13%	92.15%	92.13%	0.752	0.7505	0.7496	Active Percent fea
1	Logistic Regression	HCDR	92.13%	92.15%	92.13%	0.7526	0.7511	0.7502	C Inc F fea
2	Logistic Regression	HCDR	92.13%	92.15%	92.13%	0.7525	0.7511	0.7501	Cre Anr Rati Cur Applica
3	Logistic Regression	HCDR	92.14%	92.15%	92.13%	0.7527	0.7513	0.7504	Inco Anr Rati Cur Applica

Feature 5: Average of "Days past due" per customer

Credit card balance data

```
In [178]: credit_card = datasets['credit_card_balance']
```

```
In [179]: credit_card[['SK_ID_CURR', 'SK_ID_PREV', 'SK_DPD']]
```

	SK_ID_CURR	SK_ID_PREV	SK_DPD
0	378907	2562384	0
1	363914	2582071	0
2	371185	1740877	0
3	337855	1389973	0
4	126868	1891521	0
...
3840307	328243	1036507	0
3840308	347207	1714892	0
3840309	215757	1302323	0
3840310	430337	1624872	0
3840311	236760	2411345	0

3840312 rows × 3 columns

```
In [180...]: credit_card['SK_DPD'].value_counts()
```

```
Out[180...]: 0      3686957
1      90369
8      2772
32     2340
7      1797
...
1520    1
1712    1
195     1
1306    1
446     1
Name: SK_DPD, Length: 917, dtype: int64
```

```
In [181...]: credit_avg_grp = credit_card.groupby(by = ['SK_ID_CURR'])['SK_DPD'].mean()
credit_card = credit_card.merge(credit_avg_grp, on = ['SK_ID_CURR'], how =
```

```
In [182...]: credit_card
```

Out[182...]	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	AMT_CREDIT_LII
0	2562384	378907	-6	56.970	
1	2582071	363914	-1	63975.555	
2	1740877	371185	-7	31815.225	
3	1389973	337855	-4	236572.110	
4	1891521	126868	-1	453919.455	
...
3840307	1036507	328243	-9	0.000	
3840308	1714892	347207	-9	0.000	
3840309	1302323	215757	-9	275784.975	
3840310	1624872	430337	-10	0.000	
3840311	2411345	236760	-10	0.000	

3840312 rows × 24 columns

In [183...]: credit_card['AVG_DPD'].value_counts()

Out[183...]:

0.000000	2264271
0.010417	103968
0.020833	64224
0.031250	43648
0.010526	39615
...	...
28.750000	4
11.000000	4
5.750000	4
6.000000	3
9.000000	2

Name: AVG_DPD, Length: 3945, dtype: int64

In [184...]: credit_card['AVG_DPD'].isnull().sum()

Out[184...]: 0

In [185...]: credit_avg_dpd = credit_card[['SK_ID_CURR', 'AVG_DPD']]
credit_avg_dpd

Out[185...]

	SK_ID_CURR	AVG_DPD
0	378907	0.127660
1	363914	0.010417
2	371185	0.000000
3	337855	0.000000
4	126868	0.010417
...
3840307	328243	0.000000
3840308	347207	0.000000
3840309	215757	0.000000
3840310	430337	0.000000
3840311	236760	0.432432

3840312 rows × 2 columns

In [186...]

credit_avg_dpd = credit_avg_dpd[:1509345]

In [187...]

train

Out[187...]

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR
0	100002	1	Cash loans	M	N
1	100002	1	Cash loans	M	N
2	100002	1	Cash loans	M	N
3	100002	1	Cash loans	M	N
4	100002	1	Cash loans	M	N
...
1509340	456255	0	Cash loans	F	N
1509341	456255	0	Cash loans	F	N
1509342	456255	0	Cash loans	F	N
1509343	456255	0	Cash loans	F	N
1509344	456255	0	Cash loans	F	N

1509345 rows × 101 columns

In [188...]

```
import gc  
gc.collect()
```

Out[188...]
20

In [189...]

```
train = train.merge(credit_avg_dpd, on = 'SK_ID_CURR', how = 'left')  
train
```

Out[189...]

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR
0	100002	1	Cash loans	M	N
1	100002	1	Cash loans	M	N
2	100002	1	Cash loans	M	N
3	100002	1	Cash loans	M	N
4	100002	1	Cash loans	M	N
...
8472905	456255	0	Cash loans	F	N
8472906	456255	0	Cash loans	F	N
8472907	456255	0	Cash loans	F	N
8472908	456255	0	Cash loans	F	N
8472909	456255	0	Cash loans	F	N

8472910 rows × 102 columns

In [190...]

```
train['AVG_DPD'].isnull().sum()
```

Out[190...]

1054730

In [191...]

```
train['AVG_DPD'].value_counts()
```

Out[191...]

0.000000	4209361
0.010417	210565
0.020833	122392
0.010526	87922
0.031250	83724
...	
1.625000	3
2.923077	2
39.500000	2
9.200000	2
23.200000	1

Name: AVG_DPD, Length: 3603, dtype: int64

```
In [192...]: train = train[:1700000]
```

```
In [193...]: train
```

```
Out[193...]:
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR
0	100002	1	Cash loans	M	N
1	100002	1	Cash loans	M	N
2	100002	1	Cash loans	M	N
3	100002	1	Cash loans	M	N
4	100002	1	Cash loans	M	N
...
1699995	172394	1	Cash loans	M	Y
1699996	172394	1	Cash loans	M	Y
1699997	172394	1	Cash loans	M	Y
1699998	172394	1	Cash loans	M	Y
1699999	172394	1	Cash loans	M	Y

1700000 rows × 102 columns

```
In [194...]: train['AVG_DPD'].isnull().sum()
```

```
Out[194...]: 214613
```

```
In [195...]: train['AVG_DPD'].fillna(0, inplace = True)
```

```
In [196...]: train['AVG_DPD'].value_counts()
```

```
Out[196...]:
```

0.000000	1060899
0.010417	45943
0.020833	24935
0.031250	17953
0.010526	14949
...	
1.636364	3
3.111111	3
1.818182	3
3.166667	2
1.666667	2

Name: AVG_DPD, Length: 1216, dtype: int64

```
In [197]: train['AVG_DPD'].isnull().sum()
```

Out[197]: 0

```
In [198]: train
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR
0	100002	1	Cash loans	M	N
1	100002	1	Cash loans	M	N
2	100002	1	Cash loans	M	N
3	100002	1	Cash loans	M	N
4	100002	1	Cash loans	M	N
...
1699995	172394	1	Cash loans	M	Y
1699996	172394	1	Cash loans	M	Y
1699997	172394	1	Cash loans	M	Y
1699998	172394	1	Cash loans	M	Y
1699999	172394	1	Cash loans	M	Y

1700000 rows × 102 columns

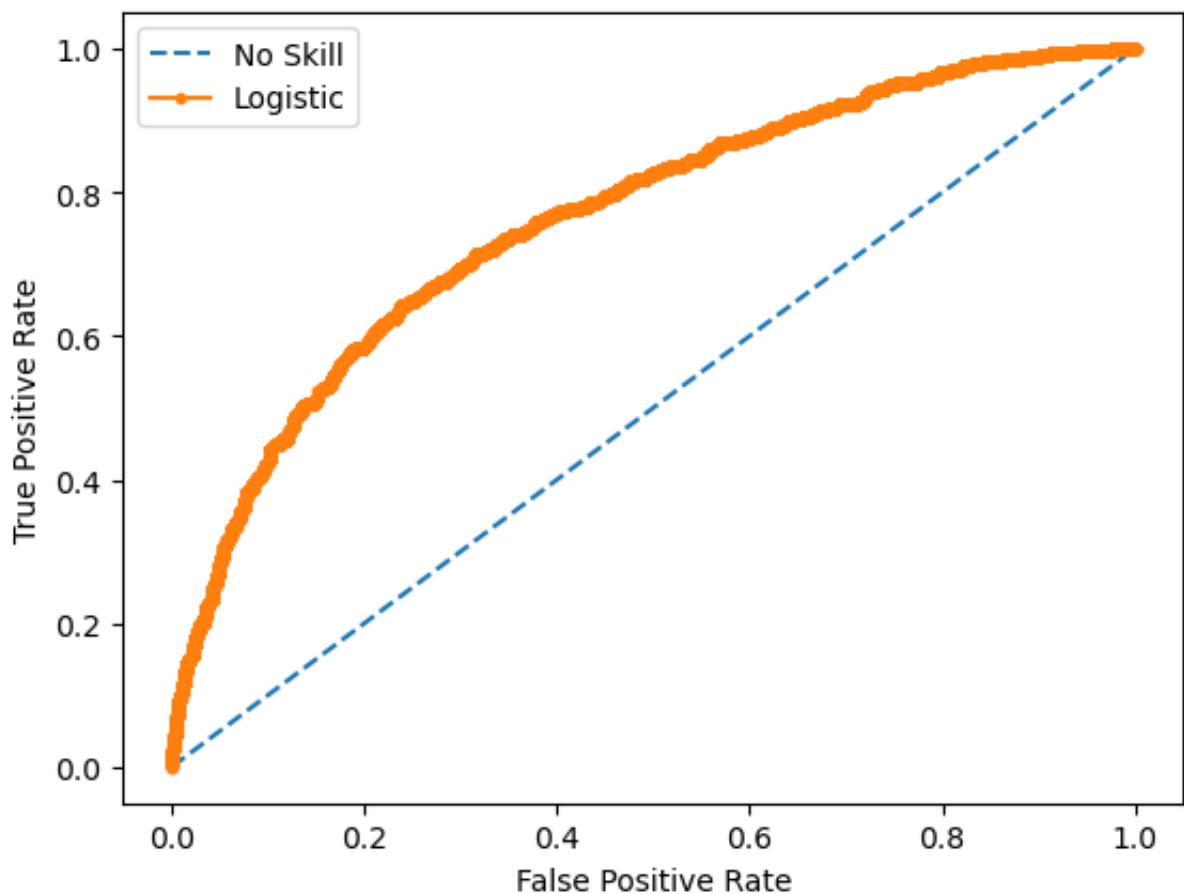
```
In [75]: avg_dpd_model = model_logreg(train, 'Average DPD Feature')
```

X train	shape: (1088000, 101)
X validation	shape: (272000, 101)
X test	shape: (340000, 101)

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_
0	1.630563	3.585626	-0.407585	-0.095343	0.075151	
1	1.312255	-0.555794	-0.166599	0.406231	0.618199	
2	0.219138	0.824679	0.194881	-0.552245	0.477013	
3	-1.725651	0.824679	-0.287092	-0.357542	0.265558	
4	-1.193248	-0.555794	-0.166599	0.120300	0.776871	

5 rows × 225 columns

No Skill: ROC AUC=0.500
 Logistic: ROC AUC=0.763



	Pipeline	Dataset	TrainAcc	ValidAcc	TestAcc	TrainROC	TestROC	ValidROC	Feat Ad
0	Logistic Regression	HCDR	92.13%	92.15%	92.13%	0.752	0.7505	0.7496	Active Percent fea
1	Logistic Regression	HCDR	92.13%	92.15%	92.13%	0.7526	0.7511	0.7502	Ci Inc F fea
2	Logistic Regression	HCDR	92.13%	92.15%	92.13%	0.7525	0.7511	0.7501	Cre Anr Rati Cur Applica
3	Logistic Regression	HCDR	92.14%	92.15%	92.13%	0.7527	0.7513	0.7504	Inco Anr Rati Cur Applica
4	Logistic Regression	HCDR	92.10%	92.02%	92.08%	0.7599	0.7628	0.7625	Ave Fea

Feature 6: Debt Overdue per Customer

In [239...]

bureau_data

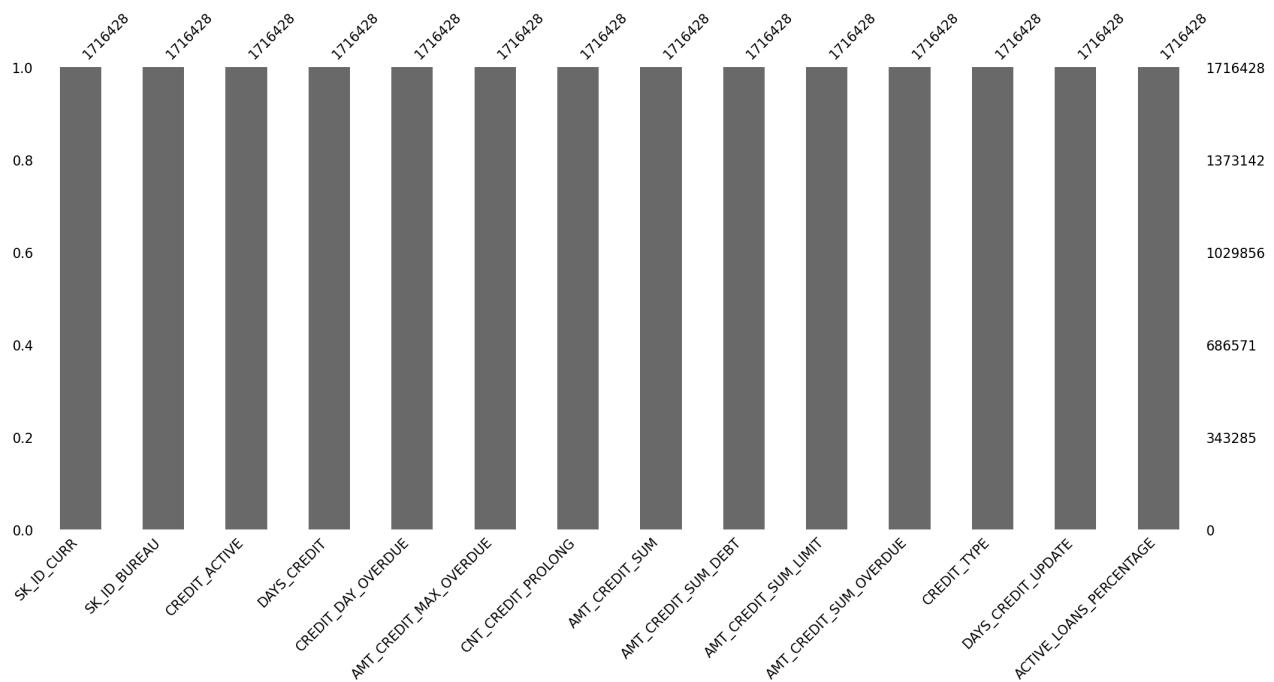
Out[239...]

	SK_ID_CURR	SK_ID_BUREAU	CREDIT_ACTIVE	DAY_S_CREDIT	CREDIT_DAY_OVERI
0	215354	5714462	Closed	-497	
1	215354	5714463	Active	-208	
2	215354	5714464	Active	-203	
3	215354	5714465	Active	-203	
4	215354	5714466	Active	-629	
...
1716423	259355	5057750	Active	-44	
1716424	100044	5057754	Closed	-2648	
1716425	100044	5057762	Closed	-1809	
1716426	246829	5057770	Closed	-1878	
1716427	246829	5057778	Closed	-463	

1716428 rows × 14 columns

In [82]:

```
msno.bar(bureau_data)
plt.show()
```



```
In [240...]: total_debt_grpby = bureau_data[['SK_ID_CURR', 'AMT_CREDIT_SUM_DEBT']].groupby('SK_ID_CURR').sum()
total_overdue_grpby = bureau_data[['SK_ID_CURR', 'AMT_CREDIT_SUM_OVERDUE']].groupby('SK_ID_CURR').sum()
```

```
In [241...]: bureau_data = bureau_data.merge(total_debt_grpby, on = ['SK_ID_CURR'], how = 'left')
bureau_data = bureau_data.merge(total_overdue_grpby, on = ['SK_ID_CURR'], how = 'left')
```

```
In [242...]: del total_debt_grpby
del total_overdue_grpby
```

```
In [243...]: bureau_data[['SK_ID_CURR', 'TOTAL_CUSTOMER_DEBT', 'TOTAL_CUSTOMER_OVERDUE']]
```

	SK_ID_CURR	TOTAL_CUSTOMER_DEBT	TOTAL_CUSTOMER_OVERDUE
0	215354	4141399.680	0.0
1	215354	4141399.680	0.0
2	215354	4141399.680	0.0
3	215354	4141399.680	0.0
4	215354	4141399.680	0.0
...
1716423	259355	22500.000	0.0
1716424	100044	1523334.600	0.0
1716425	100044	1523334.600	0.0
1716426	246829	237417.525	0.0
1716427	246829	237417.525	0.0

1716428 rows × 3 columns

In [244...]: `bureau_data['TOTAL_CUSTOMER_OVERDUE'].value_counts()`

```
Out[244...]: 0.000      1686808
4.500       2152
9.000       755
13.500      642
22.500      498
...
402.165     1
1361214.000 1
1617403.500 1
18920.250   1
20263.500   1
Name: TOTAL_CUSTOMER_OVERDUE, Length: 1369, dtype: int64
```

In [245...]: `bureau_data['TOTAL_CUSTOMER_DEBT'].value_counts()`

```
Out[245...]: 0.00      133334
225000.00    2392
450000.00    2009
90000.00     1245
675000.00    1182
...
11614.50     1
87561.00     1
187132.50    1
1871731.62   1
175054.50    1
Name: TOTAL_CUSTOMER_DEBT, Length: 201075, dtype: int64
```

In [246...]: `bureau_data['OVERDUE_DEBT_RATIO'] = bureau_data['TOTAL_CUSTOMER_OVERDUE'] /`

```
In [247...]: np.isinf(bureau_data[['OVERDUE_DEBT_RATIO']]).values.sum()
```

```
Out[247...]: 418
```

```
In [248...]: index_to_drop = bureau_data.loc[bureau_data['OVERDUE_DEBT_RATIO'] == np.inf].index
```

```
In [249...]: len(index_to_drop)
```

```
Out[249...]: 418
```

```
In [250...]: bureau_data.drop(index_to_drop.index, inplace = True)
```

```
In [251...]: np.isinf(bureau_data[['OVERDUE_DEBT_RATIO']]).values.sum()
```

```
Out[251...]: 0
```

```
In [253...]: bureau_data['OVERDUE_DEBT_RATIO'].value_counts()
```

```
Out[253...]: 0.000000    1553892  
1.000000      160  
0.000030       61  
0.014851       51  
0.000059       47  
...  
0.000233         1  
0.344647         1  
0.021647         1  
0.080689         1  
0.001196         1  
Name: OVERDUE_DEBT_RATIO, Length: 3630, dtype: int64
```

```
In [254...]: debt_ovd = bureau_data[['SK_ID_CURR', 'OVERDUE_DEBT_RATIO']]  
debt_ovd
```

Out[254...]

	SK_ID_CURR	OVERDUE_DEBT_RATIO
0	215354	0.0
1	215354	0.0
2	215354	0.0
3	215354	0.0
4	215354	0.0
...
1716423	259355	0.0
1716424	100044	0.0
1716425	100044	0.0
1716426	246829	0.0
1716427	246829	0.0

1716010 rows × 2 columns

In [257...]

train

Out[257...]

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR
0	100002	1	Cash loans	M	N
1	100002	1	Cash loans	M	N
2	100002	1	Cash loans	M	N
3	100002	1	Cash loans	M	N
4	100002	1	Cash loans	M	N
...
1699995	172394	1	Cash loans	M	Y
1699996	172394	1	Cash loans	M	Y
1699997	172394	1	Cash loans	M	Y
1699998	172394	1	Cash loans	M	Y
1699999	172394	1	Cash loans	M	Y

1700000 rows × 102 columns

In [258...]

debt_ovd = debt_ovd[:1500000]

In [259...]

train = train.merge(debt_ovd, on = 'SK_ID_CURR', how = 'left')

In [260...]

```
train = train[:1700000]
train
```

Out[260...]

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR
0	100002	1	Cash loans	M	N
1	100002	1	Cash loans	M	N
2	100002	1	Cash loans	M	N
3	100002	1	Cash loans	M	N
4	100002	1	Cash loans	M	N
...
1699995	108558	0	Cash loans	F	N
1699996	108558	0	Cash loans	F	N
1699997	108558	0	Cash loans	F	N
1699998	108558	0	Cash loans	F	N
1699999	108558	0	Cash loans	F	N

1700000 rows × 103 columns

In [102...]

```
train['OVERDUE_DEBT_RATIO'].value_counts()
```

Out[102...]

0.000000	1579821
0.000188	8064
0.000342	4860
0.000290	3584
0.000137	2400
...	
0.000027	1
0.000286	1
91.000000	1
0.013688	1
0.000699	1

Name: OVERDUE_DEBT_RATIO, Length: 79, dtype: int64

In [103...]

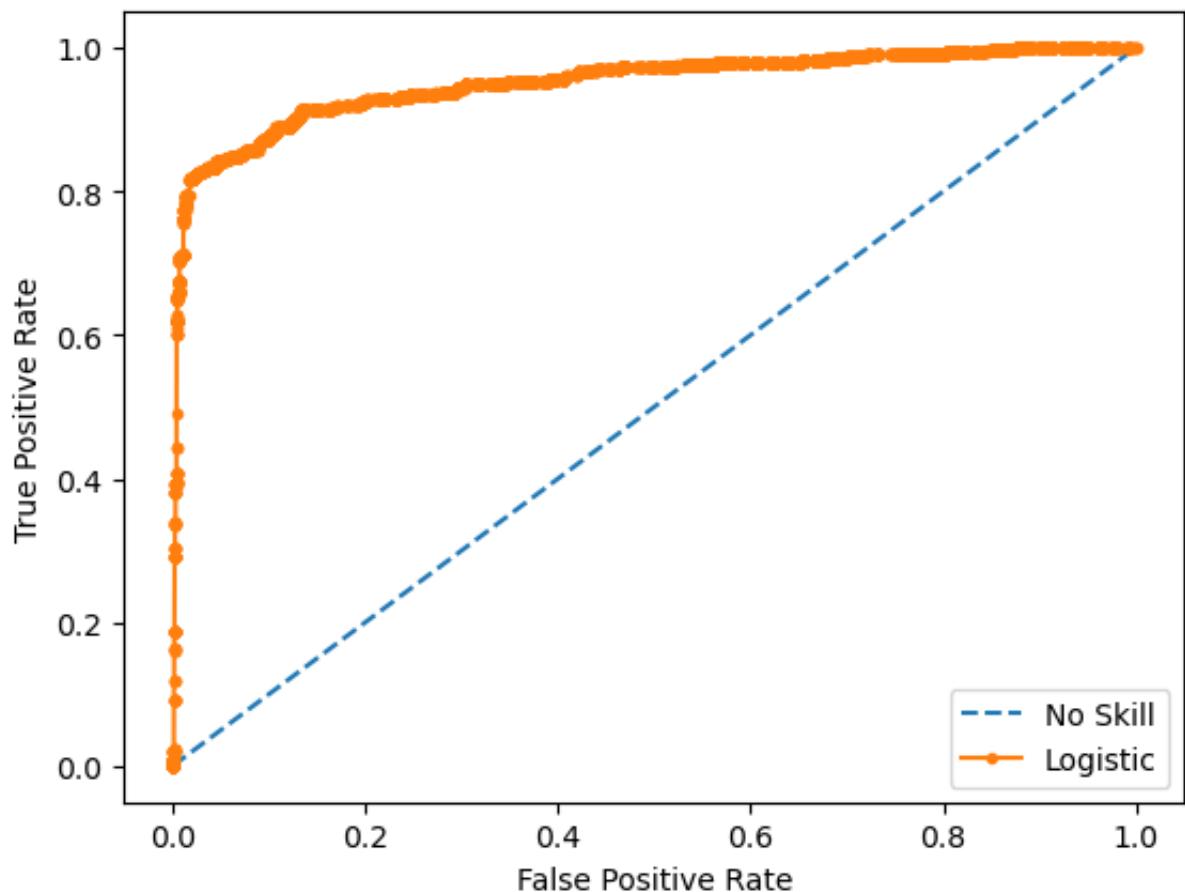
```
debt_ovd_model = model_logreg(train, 'Debt Overdue of each Customer Feature')
```

```
X train          shape: (1088000, 102)
X validation    shape: (272000, 102)
X test          shape: (340000, 102)
```

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_
0	1.634258	0.799427	-0.223311	-0.568658	-0.872830	
1	1.311380	-0.568021	-1.053014	-0.661751	-0.951324	
2	0.228227	0.799427	-0.223311	0.838082	0.035543	
3	-1.667291	2.166875	-0.223311	-0.568658	-0.600944	
4	-1.208422	-0.568021	1.021244	2.088601	0.864284	

5 rows × 221 columns

No Skill: ROC AUC=0.500
Logistic: ROC AUC=0.951



	Pipeline	Dataset	TrainAcc	ValidAcc	TestAcc	TrainROC	TestROC	ValidROC	Feature Add.
0	Logistic Regression	HCDR	97.04%	97.03%	97.00%	0.9521	0.9505	0.9515	Def Overd of ea Custom Featu

We believe that this sudden spike of accuracy is due to overfitting and thus we will keep the last test accuracy i.e 92.13% as our result from feature engineering

Random Forest Classifier (Ensemble method)

Feature Importances on ensemble method

In [116...]

```
def model_rf(train):

    X = train.drop(['TARGET'], axis = 1)
    y = train["TARGET"]

    # Split the provided training data into training and validation and test
    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2)
    X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.2)
    print(f"X train           shape: {X_train.shape}")
    print(f"X validation     shape: {X_valid.shape}")
    print(f"X test            shape: {X_test.shape}")

    numerical_features = X_train.select_dtypes(include = ['int64', 'float64'])
    categorical_features = X_train.select_dtypes(include = ['object', 'bool'])
    #print(f"\nNumerical features : {list(numerical_features)}")
    #print(f"\nCategorical features : {list(categorical_features)}")

    num_pipeline = Pipeline([
        ('scaler', StandardScaler()),
        ('imputer', SimpleImputer(strategy = 'median'))
    ])

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

    data_pipeline = ColumnTransformer([
        ("num_pipeline", num_pipeline, numerical_features),
        ("cat_pipeline", cat_pipeline, categorical_features)], remainder = "drop")

    RF = RandomForestClassifier(random_state = 42, n_estimators=20, criterion='gini')

    data_pipeline_rf = make_pipeline(data_pipeline, RF)

    data_pipeline_rf.fit(X_train, y_train)

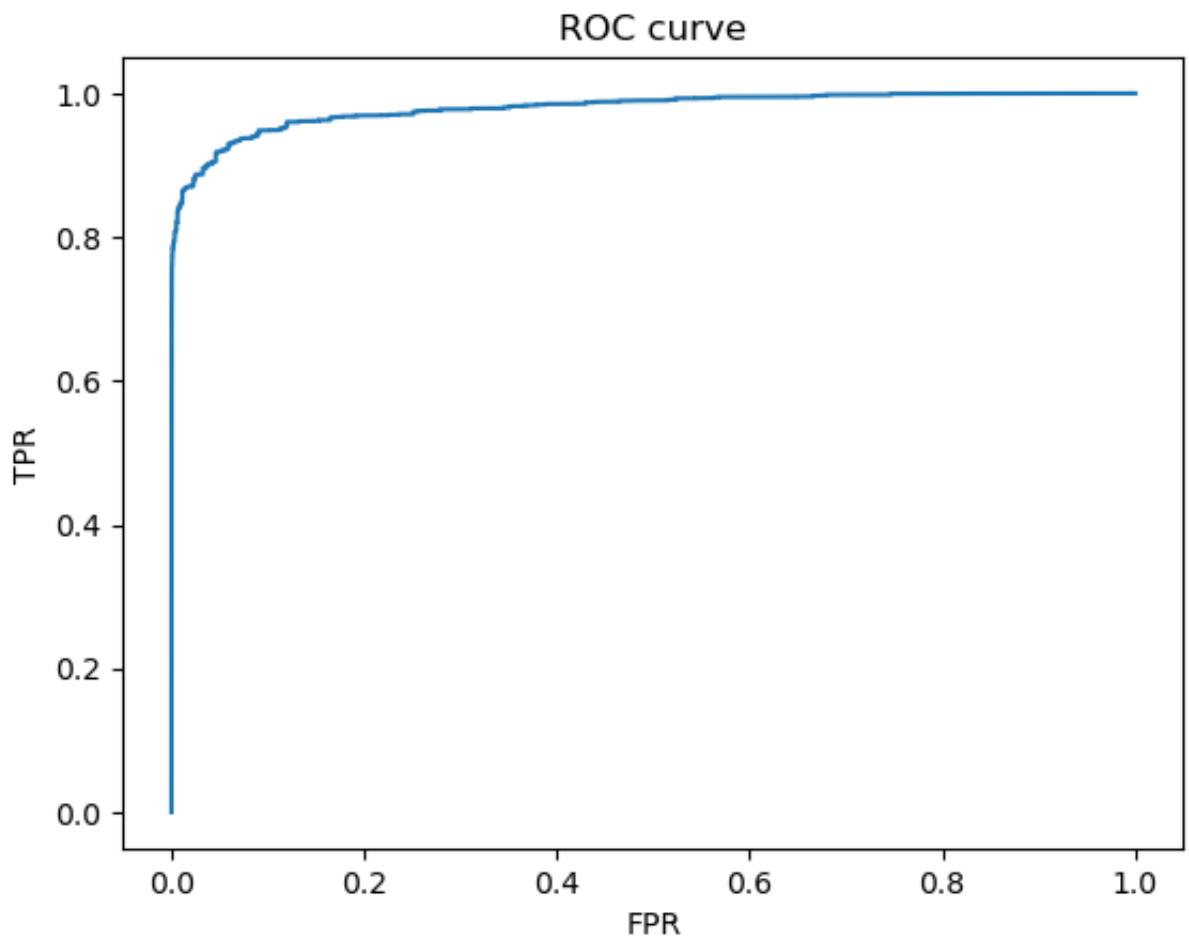
    train_acc = data_pipeline_rf.score(X_train, y_train)
    validAcc = data_pipeline_rf.score(X_valid, y_valid)
    testAcc = data_pipeline_rf.score(X_test, y_test)
```

```
predictions = data_pipeline_rf.predict_proba(X_test)
print ("Score",roc_auc_score(y_test, predictions[:,1]))\n\nfpr, tpr, _ = roc_curve(y_test, predictions[:,1])\n\nplt.clf()
plt.plot(fpr, tpr)
plt.xlabel('FPR')
plt.ylabel('TPR')
plt.title('ROC curve')
plt.show()\n\ntrain_roc = roc_auc_score(y_train, data_pipeline_rf.predict_proba(X_train))
test_roc = roc_auc_score(y_test, data_pipeline_rf.predict_proba(X_test))
valid_roc = roc_auc_score(y_valid, data_pipeline_rf.predict_proba(X_valid))\n\nresults.loc[len(results)] = [ "Random Forest", "HCDR", f'{train_acc*100:.8f}', f'{validAcc*100:.2f}%', f'{testAcc*100:.8f}' ]
display(results)\n\nreturn data_pipeline_rf,RF
```

In [117...]

```
debt_ovd_model_pipe, rf = model_rf(train)
```

```
X train          shape: (1088000, 102)
X validation    shape: (272000, 102)
X test          shape: (340000, 102)
Score 0.9796172080754798
```



	Pipeline	Dataset	TrainAcc	ValidAcc	TestAcc	TrainROC	TestROC	ValidROC	Feature Add
0	Logistic Regression	HCDR	97.04%	97.03%	97.00%	0.9521	0.9505	0.9515	Def Overd of ea Custom Featu
1	Random Forest	HCDR	96.77%	96.78%	96.76%	0.9803	0.9796	0.9786	Ensemle meth

In [162]:

train

Out[162...]	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR
0	100002	1	Cash loans	M	N
1	100002	1	Cash loans	M	N
2	100002	1	Cash loans	M	N
3	100002	1	Cash loans	M	N
4	100002	1	Cash loans	M	N
...
1699995	108558	0	Cash loans	F	N
1699996	108558	0	Cash loans	F	N
1699997	108558	0	Cash loans	F	N
1699998	108558	0	Cash loans	F	N
1699999	108558	0	Cash loans	F	N

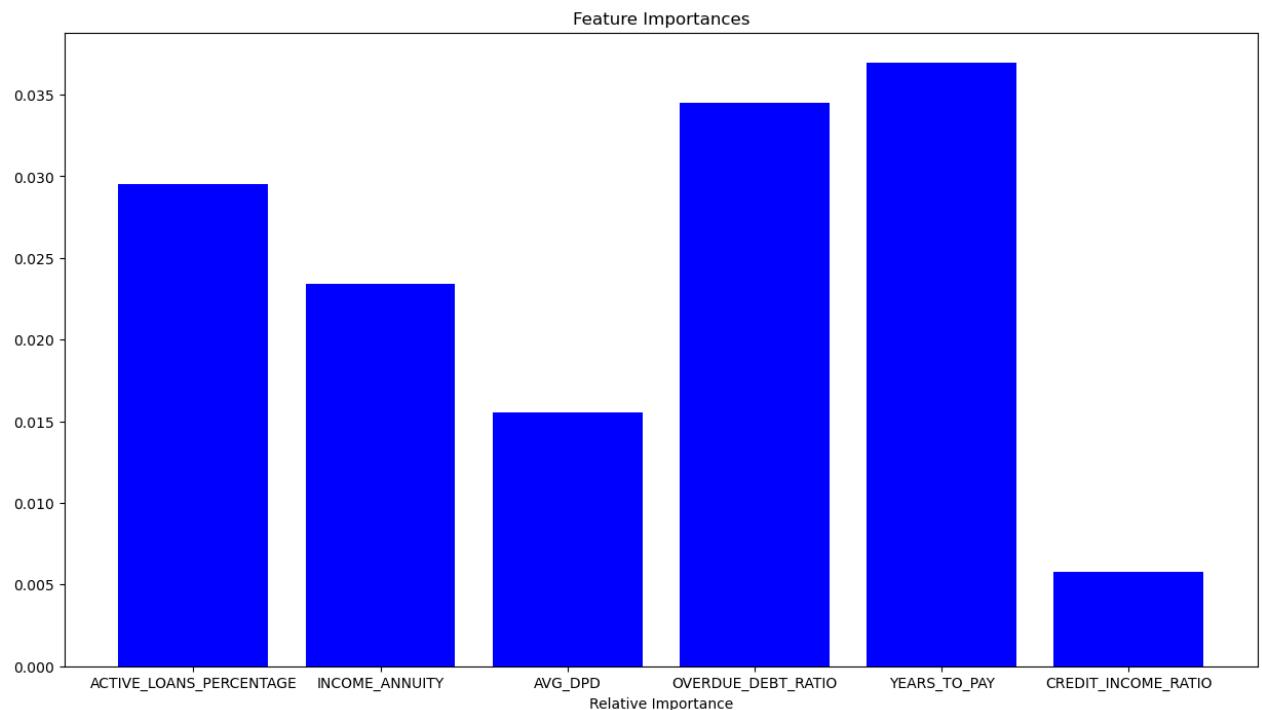
1700000 rows × 103 columns

```
In [104...]: train_columns = ['ACTIVE_LOANS_PERCENTAGE', 'CREDIT_INCOME_RATIO', 'YEARS_TO...
```

```
In [165...]: features = train_columns
importances = rf.feature_importances_
indices = np.argsort(importances[-6:])
importances[-6:]
```

```
Out[163...]: array([0.0000000e+00, 2.55603290e-04, 6.82355186e-05, 0.0000000e+00,
       0.0000000e+00, 0.0000000e+00])
```

```
In [176...]: plt.figure(figsize = (15, 8))
plt.title('Feature Importances')
plt.bar(range(len(indices)), importances[indices], color='b', align='center')
plt.xticks(range(len(indices)), [features[i] for i in indices])
plt.xlabel('Relative Importance')
plt.show();
```



As per the Feature Importances plot we conclude that Credit Annuity Ratio of the Application Feature(Feature no 3) is the most important feature

Hyperparameter Tuning

```
In [299]: train = pd.read_pickle('train2.pkl')
```

```
In [300]: train
```

Out[300...]	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR
0	100002	1	Cash loans	M	N
1	100002	1	Cash loans	M	N
2	100002	1	Cash loans	M	N
3	100002	1	Cash loans	M	N
4	100002	1	Cash loans	M	N
...
1699995	172394	1	Cash loans	M	Y
1699996	172394	1	Cash loans	M	Y
1699997	172394	1	Cash loans	M	Y
1699998	172394	1	Cash loans	M	Y
1699999	172394	1	Cash loans	M	Y

1700000 rows × 102 columns

```
In [262...]: train.to_pickle('train3.pkl')
```

```
In [301...]: X = train.drop(['TARGET'], axis = 1)
y = train["TARGET"]

# Split the provided training data into training and validation and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, random_state=42)
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=0.25, random_state=42)

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

X train           shape: (1088000, 101)
X validation     shape: (272000, 101)
X test            shape: (340000, 101)
```

In [302...]

```

numerical_features = X_train.select_dtypes(include = ['int64', 'float64'])
categorical_features = X_train.select_dtypes(include = ['object', 'bool'])

num_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('imputer', SimpleImputer(strategy = 'median'))
])

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

data_pipeline = ColumnTransformer([
    ("num_pipeline", num_pipeline, numerical_features),
    ("cat_pipeline", cat_pipeline, categorical_features)], remainder = 'drop')

```

In [303...]

```

X_train_transformed = data_pipeline.fit_transform(X_train)

column_names = list(numerical_features) + \
               list(data_pipeline.transformers_[1][1].named_steps["ohe"])

X_train_transformed_df = pd.DataFrame(X_train_transformed, columns=column_

```

In [304...]

```
x_train_transformed_df
```

Out[304...]

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY
0	1.630563	3.585626	-0.407585	-0.095343	0.07515`
1	1.312255	-0.555794	-0.166599	0.406231	0.61819`
2	0.219138	0.824679	0.194881	-0.552245	0.47701`
3	-1.725651	0.824679	-0.287092	-0.357542	0.26555`
4	-1.193248	-0.555794	-0.166599	0.120300	0.77687`
...
1087995	-0.306728	2.205153	-0.166599	-0.552245	-0.46239`
1087996	1.043074	-0.555794	0.797347	0.096633	0.32837`
1087997	1.186413	-0.555794	0.194881	-0.011405	-0.19005`
1087998	-1.327695	-0.555794	-0.431684	-0.882114	-1.12816`
1087999	1.666854	0.824679	-0.648572	0.096763	-0.56666`

1088000 rows × 225 columns

```
In [305...]  
x_test_transformed = data_pipeline.fit_transform(X_test)  
column_names = list(numerical_features) + \  
    list(data_pipeline.transformers_[1][1].named_steps["ohe"]  
  
X_test_transformed_df = pd.DataFrame(X_test_transformed, columns=column_na
```

```
In [306...]  
X_test_transformed_df
```

```
Out[306...]  
SK_ID_CURR CNT_CHILDREN AMT_INCOME_TOTAL AMT_CREDIT AMT_ANNUITY  
0 -0.447151 -0.556897 0.869128 1.011823 1.281185  
1 -0.391754 -0.556897 -0.030261 -0.676659 0.929630  
2 0.329466 -0.556897 0.434424 1.949191 1.178944  
3 -1.330138 -0.556897 0.239556 -0.012484 0.404555  
4 1.535746 -0.556897 -0.255108 -1.264610 -0.998764  
... ... ... ... ... ...  
339995 0.036900 -0.556897 -0.180159 0.801808 0.337469  
339996 0.655356 -0.556897 0.569332 0.865105 0.106539  
339997 1.709535 -0.556897 -0.434986 -0.558115 -0.428212  
339998 1.169414 -0.556897 -0.434986 -0.210165 0.238130  
339999 1.079105 0.824931 0.569332 1.606087 0.754175
```

340000 rows × 222 columns

```
In [307...]  
set(X_train_transformed_df.columns).difference(set(X_test_transformed_df.co
```

```
Out[307...]  
{'NAME_FAMILY_STATUS_Unknown',  
'NAME_INCOME_TYPE_Businessman',  
'NAME_INCOME_TYPE_Maternity leave'}
```

```
In [308...]  
X_test_transformed_df['NAME_FAMILY_STATUS_Unknown'] = 0  
X_test_transformed_df['NAME_INCOME_TYPE_Businessman'] = 0  
X_test_transformed_df['NAME_INCOME_TYPE_Maternity leave'] = 0
```

```
In [309...]  
X_train_transformed_df.shape
```

```
Out[309...]  
(1088000, 225)
```

```
In [310...]  
X_test_transformed_df.shape
```

```
Out[310... (340000, 225)
```

```
In [311... X_test_transformed = X_test_transformed_df.to_numpy()
```

```
In [312... X_test_transformed.shape
```

```
Out[312... (340000, 225)
```

Hyperparameter tuning for logistic regression

```
In [313... param_grid_lr = {  
    'C': [10**x for x in range(-3,4)],  
    'penalty': ['l1','l2']  
}
```

```
In [317... grid_search_lr = GridSearchCV(LogisticRegression(), param_grid = param_grid_lr)
```

```
In [318... grid_search_model1 = grid_search_lr.fit(X_train_transformed, y_train)
```

Best parameters for Logistic Regression

```
In [319... grid_search_lr.best_params_
```

```
Out[319... {'C': 100, 'penalty': 'l2'}
```

```
In [235... grid_search_lr.best_estimator_.fit(X_train_transformed, y_train)
```

```
Out[235... LogisticRegression(C=100)
```

```
In [279... best_train_accuracy = grid_search_lr.best_estimator_.score(X_train_transformed)  
best_train_accuracy*100
```

```
Out[279... 97.09981617647058
```

```
In [280... best_test_accuracy = grid_search_lr.best_estimator_.score(X_test_transformed)  
best_test_accuracy*100
```

```
Out[280... 92.4614705882353
```

After performing Hyperparameter tuning for logistic regression we are getting an improved test accuracy of 92.46%.

Hyperparameter tuning for random forest classifier

```
In [49]: param_grid = {  
    'n_estimators': [80, 100, 120],  
    'max_depth' : [3, 4],  
    'criterion' :['gini','entropy']  
}  
  
In [50]: grid_search = GridSearchCV(RandomForestClassifier(), param_grid = param_gr:  
  
In [52]: grid_search.fit(X_train_transformed, y_train)  
  
Out[52]: GridSearchCV(cv=2, estimator=RandomForestClassifier(),  
                      param_grid={'criterion': ['gini', 'entropy'], 'max_depth': [3,  
4],  
                      'n_estimators': [80, 100, 120]},  
                      scoring='accuracy')
```

Best parameters for Random Forest Classifier

```
In [53]: grid_search.best_params_  
  
Out[53]: {'criterion': 'gini', 'max_depth': 4, 'n_estimators': 80}  
  
In [55]: grid_search.best_estimator_.fit(X_train_transformed, y_train)  
  
Out[55]: RandomForestClassifier(max_depth=4, n_estimators=80)  
  
In [89]: best_train_accuracy = grid_search.best_estimator_.score(X_train_transformed)  
  
In [90]: best_train_accuracy*100  
  
Out[90]: 94.196875  
  
In [100...]: best_test_accuracy = grid_search.best_estimator_.score(X_test_transformed,  
  
In [101...]: best_test_accuracy*100  
  
Out[101...]: 92.975
```

After performing Hyperparameter tuning for random forest classifier we are getting an improved test accuracy of 92.975%.

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 [201...]: x_kaggle_test = datasets["application_test"]
```

```
In [202...]: x_kaggle_test
```

```
Out[202...]:
```

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALPROP
0	100001	Cash loans	F	N	
1	100005	Cash loans	M	N	
2	100013	Cash loans	M	Y	
3	100028	Cash loans	F	N	
4	100038	Cash loans	M	Y	
...
48739	456221	Cash loans	F	N	
48740	456222	Cash loans	F	N	
48741	456223	Cash loans	F	Y	
48742	456224	Cash loans	M	N	
48743	456250	Cash loans	F	Y	

48744 rows × 116 columns

```
In [204...]: x_kaggle_test.shape
```

```
Out[204...]: (48744, 116)
```

```
In [205...]: x_kaggle_test = x_kaggle_test.loc[:, ~x_kaggle_test.columns.str.startswith("D_")]
```

```
In [206...]: x_kaggle_test.shape
```

Out[206... (48744, 96)

```
In [207... X_kaggle_test['ACTIVE_LOANS_PERCENTAGE'] = train['ACTIVE_LOANS_PERCENTAGE']
X_kaggle_test['CREDIT_INCOME_RATIO'] = train['CREDIT_INCOME_RATIO']
X_kaggle_test['YEARS_TO_PAY'] = train['YEARS_TO_PAY']
X_kaggle_test['INCOME_ANNUITY'] = train['INCOME_ANNUITY']
X_kaggle_test['AVG_DPD'] = train['AVG_DPD']
```

In [208... X_kaggle_test

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALPROP
0	100001	Cash loans	F	N	
1	100005	Cash loans	M	N	
2	100013	Cash loans	M	Y	
3	100028	Cash loans	F	N	
4	100038	Cash loans	M	Y	
...
48739	456221	Cash loans	F	N	
48740	456222	Cash loans	F	N	
48741	456223	Cash loans	F	Y	
48742	456224	Cash loans	M	N	
48743	456250	Cash loans	F	Y	

48744 rows × 101 columns

```
In [282... numerical_features = X_kaggle_test.select_dtypes(include = ['int64', 'float64'])
categorical_features = X_kaggle_test.select_dtypes(include = ['object', 'bool'])

num_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('imputer', SimpleImputer(strategy = 'median'))
])

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

data_pipeline = ColumnTransformer([
    ("num_pipeline", num_pipeline, numerical_features),
    ("cat_pipeline", cat_pipeline, categorical_features)], remainder = 'drop')
```

```
In [283...]: X_kaggle_test_transformed = data_pipeline.fit_transform(X_kaggle_test)
column_names = list(numerical_features) + \
    list(data_pipeline.transformers_[1][1].named_steps["ohe"])

In [285...]: X_kaggle_test_transformed_df = pd.DataFrame(X_kaggle_test_transformed, columns=column_names)

In [329...]: X_kaggle_test_transformed_df.shape

Out[329...]: (48744, 225)

In [330...]: X_train_transformed_df.shape

Out[330...]: (1088000, 225)

In [324...]: set(X_train_transformed_df.columns).difference(set(X_kaggle_test_transformed))

Out[324...]: {'CODE_GENDER_XNA',
 'NAME_FAMILY_STATUS_Unknown',
 'NAME_INCOME_TYPE_Maternity leave'}

In [326...]: X_kaggle_test_transformed_df['CODE_GENDER_XNA'] = X_train_transformed_df['CODE_GENDER_XNA']
X_kaggle_test_transformed_df['NAME_FAMILY_STATUS_Unknown'] = X_train_transformed_df['NAME_FAMILY_STATUS_Unknown']
X_kaggle_test_transformed_df['NAME_INCOME_TYPE_Maternity leave'] = X_train_transformed_df['NAME_INCOME_TYPE_Maternity leave']

In [331...]: X_kaggle_test_transformed_4 = X_kaggle_test_transformed_df.to_numpy()

In [332...]: test_class_scores = grid_search_model1.predict_proba(X_kaggle_test_transformed_4)

In [333...]: test_class_scores[0:10]

Out[333...]: array([0.01205682, 0.23306839, 0.11699465, 0.10186553, 0.05286666,
       0.20762748, 0.05490155, 0.07793292, 0.00758586, 0.12317721])

In [334...]: # Submission dataframe
submit_df = datasets["application_test"][['SK_ID_CURR']]
submit_df['TARGET'] = test_class_scores

submit_df.head()
```

```
Out[334...]:
```

	SK_ID_CURR	TARGET
0	100001	0.012057
1	100005	0.233068
2	100013	0.116995
3	100028	0.101866
4	100038	0.052867

```
In [335...]: submit_df.to_csv("submission.csv", index=False)
```

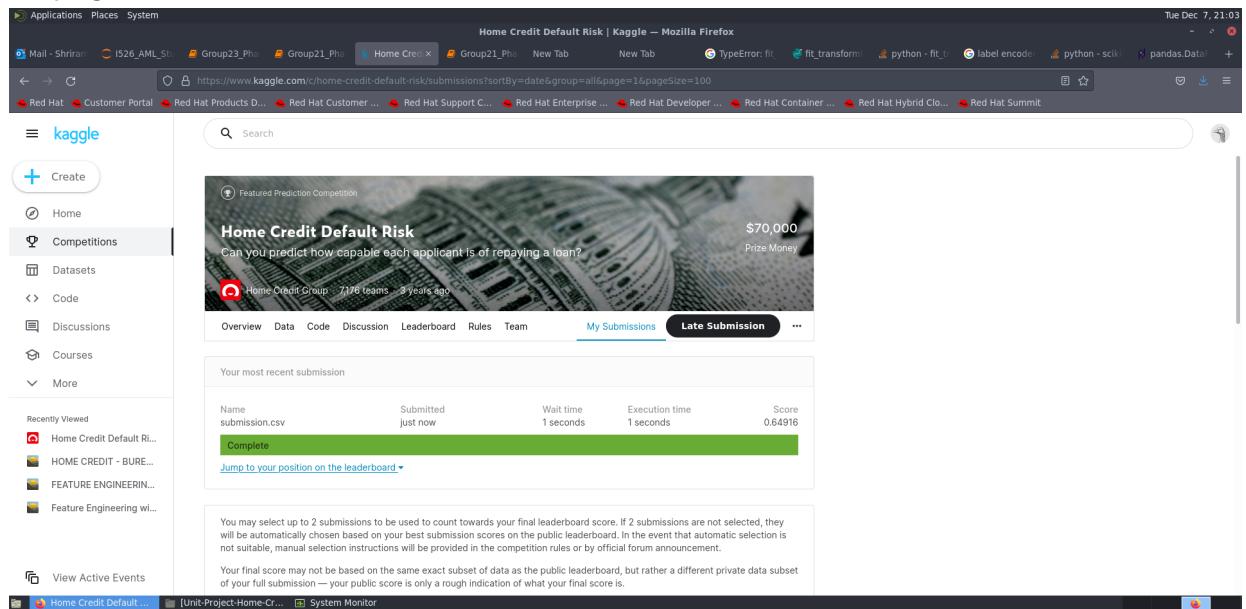
Kaggle submission via the command line API

```
In [336...]: ! kaggle competitions submit -c home-credit-default-risk -f submission.csv
```

100% | 1.26M/1.26M [00:01<00:00, 995 kB/s]
Successfully submitted to Home Credit Default Risk

Our Submission

file:///N/u/vshriram/Carbonate/Desktop/Screenshot%20from%202021-12-07%202021-03-55.png



report submission

Click on this [link](#)

Abstract

The main goal of this project is to build an optimal ML model which will predict if a loan applicant will be able to repay his/her loan. In Phase 2, we extended our work from Phase 1 and implemented Feature Engineering where we consider potential features from other tables, did feature selection from the derived features, analysis of feature importances and implemented Hyper Parameter Tuning. In feature engineering, we derived 6 additional features and we achieved an improvement over our baseline model however some feature models are leading to over-fitting and thus in our future scope we aim to select the most important features having balanced data to avoid over-fitting. The most important feature relevant to our goal was the Credit Annuity ratio of the current application. We identified this by implementing feature importances on our model. After performing Hyper-parameter tuning on logistic regression(our best pipeline) we achieved the test accuracy of 92.46%.

Introduction

Data Description:

- application_{train|test}.csv

The main table is divided into two files: Train (with TARGET) and Test (without TARGET) (without TARGET).

- bureau.csv

All past credit issued to the client by other financial institutions and reported to the Credit Bureau.

- bureau_balance.csv

Monthly balances of previous credits in Credit Bureau.

- POS_CASH_balance.csv

Monthly balance snapshots of the applicant's prior POS (point of sale) and cash loans with Home Credit.

- credit_card_balance.csv

Monthly balance snapshots of the applicant's prior credit cards with Home Credit.

- previous_application.csv

All prior Home Credit loan applications of clients with loans in our sample.

- installments_payments.csv

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

- HomeCredit_columns_description.csv

The columns in the various data files are described in this file.

The tasks to be tackled are:

- Feature Engineering
- Hyperparameter Tuning
- Analysis of Feature Importance
- Ensemble Methods

In Phase 2, we implemented Feature engineering to identify potential features that could help us get better results. We mainly derived 6 features:

- **Active loan Percentage feature:**

We take into consideration the number of active-loans feature from the bureau data so as to indicate whether a candidate will be able to handle another loan with his existing set of loans.

- **Credit Income Ratio feature:** This is the ratio of the candidate's total income to the loan amount requested. A higher value indicates a higher chance of repayment in ideal conditions.

- **Credit Annuity Ratio of the Application:**

This is the ratio of the loan amount requested to the loan annuity. The number of years does it take for the borrower to repay the amount he requested for the application.

- **Income Annuity Ratio of the Application:**

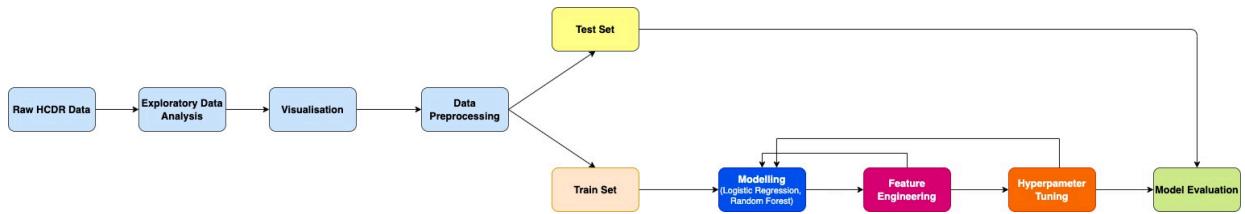
This is the ratio of the candidate's total income to the loan annuity. The number of years does it take for the borrower to repay the amount he requested for the application.

- **Average of "Days past due" per customer:**

The average of all the previous days past due of all previous loan applications of a candidate.

- **Debt Overdue per Customer:**

The ratio of Amount Overdue on the Loan amount to the total loan amount. Higher the Overdue, will it be considered as against



Pipelines

```
In [ ]:
num_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('imputer', SimpleImputer(strategy = 'median'))
])

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

data_pipeline = ColumnTransformer([
    ("num_pipeline", num_pipeline, numerical_features),
    ("cat_pipeline", cat_pipeline, categorical_features)], remainder = 'drop')
```

Here we created two different pipelines for numerical and categorical features respectively. Performed standardization and imputation on the numerical features and performed imputations and one-hot encoding on the categorical features. We combined the two pipelines using Column Transformer and passed for modeling.

```
In [ ]:
clf_pipe = make_pipeline(data_pipeline, LogisticRegression())
```

We are passing the combined data pipeline to Logistic Regression model in this pipeline

```
In [ ]:
RF = RandomForestClassifier(random_state = 42, n_estimators=20, criterion='gini')

data_pipeline_rf = make_pipeline(data_pipeline, RF)
```

Here we are passing the data pipeline to the Random Forest classifier

Feature Engineering

We derived 6 features in total out of which we selected 5 for our modeling.

1. Active loan Percentage feature
2. Credit Income Ratio feature
3. Credit Annuity Ratio of the Application
4. Income Annuity Ratio of the Application
5. Average of "Days past due" per customer
6. Debt Overdue per Customer

Impact of these features to the model -

1. Active loan Percentage feature
 - We achieved an improved result using this feature over our baseline model.
2. Credit Income Ratio feature
 - We achieved a slight improvement in the testROC using this feature.
3. Credit Annuity Ratio of the Application
 - This feature was identified as the most important using Feature Importances on the model.
4. Income Annuity Ratio of the Application
 - We achieved a slight improvement in the testROC using this feature.
5. Average of "Days past due" per customer
 - We achieved a significant improvement in the testROC using this feature.
6. Debt Overdue per Customer
 - We are receiving a spike in the result using this feature. We think this is because of overfitting.

Experimental results

Before Hyper-Parameter Tuning

	Pipeline	Dataset	TrainAcc	ValidAcc	TestAcc	TrainROC	TestROC	ValidROC	Feature Added
0	Logistic Regression	HCDR	92.13%	92.15%	92.13%	0.752	0.7505	0.7496	Active loan Percentage feature
1	Logistic Regression	HCDR	92.13%	92.15%	92.13%	0.7526	0.7511	0.7502	Credit Income Ratio feature
2	Logistic Regression	HCDR	92.13%	92.15%	92.13%	0.7525	0.7511	0.7501	Credit-Annuity Ratio of Current Application Fe...
3	Logistic Regression	HCDR	92.14%	92.15%	92.13%	0.7527	0.7513	0.7504	Income-Annuity Ratio of Current Application Fe...
4	Logistic Regression	HCDR	92.10%	92.02%	92.08%	0.7599	0.7628	0.7625	Average DPD Feature

After Hyper-Parameter Tuning

```
best_test_accuracy = grid_search_lr.best_estimator_.score(X_test_transformed, y_test)
best_test_accuracy*100
```

92.4614705882353

After performing Hyperparameter tuning for logistic regression we are getting an improved test accuracy of 92.46%.

Best parameters for Logistic Regression

```
grid_search_lr.best_params_
{'C': 100, 'penalty': 'l2'}
```

Best parameters for Random Forest Classifier

```
grid_search.best_params_
{'criterion': 'gini', 'max_depth': 4, 'n_estimators': 80}
```

Discussion Experimental Results

In this phase, we mainly considered Logistic Regression and Random Forest Classifier as our candidate models based on last phases' result.

Additionally, as Feature Engineering was to be done in this particular phase, we derived 6 new candidate features from the given set of datasets out of which we selected **5** for our model training.

Including these Features helped increase the Test accuracy of Logistic Regression from **91.93%** to **92.08%** as it can be seen from the results.

This accuracy was before Hyperparameter Tuning. We then performed Hyperparameter Tuning to further improve our insights on the model.

HyperParameter tuning helped improve the Test Accuracy for Logistic Regression from **92.08%** to **92.46%**

Important Callout: The last feature that we **did not** select for our model training was **Debt Overdue per Customer**. Selecting this feature caused a sudden spike in our test accuracy from **92.08% to 97.00%**. We believe that this sudden spike of accuracy is due to overfitting and thus we will keep the last test accuracy i.e 92.13% as our result from feature engineering.

Similarly for Random Forest Classifier, we were able to achieve test accuracy of **92.975%**

Conclusion

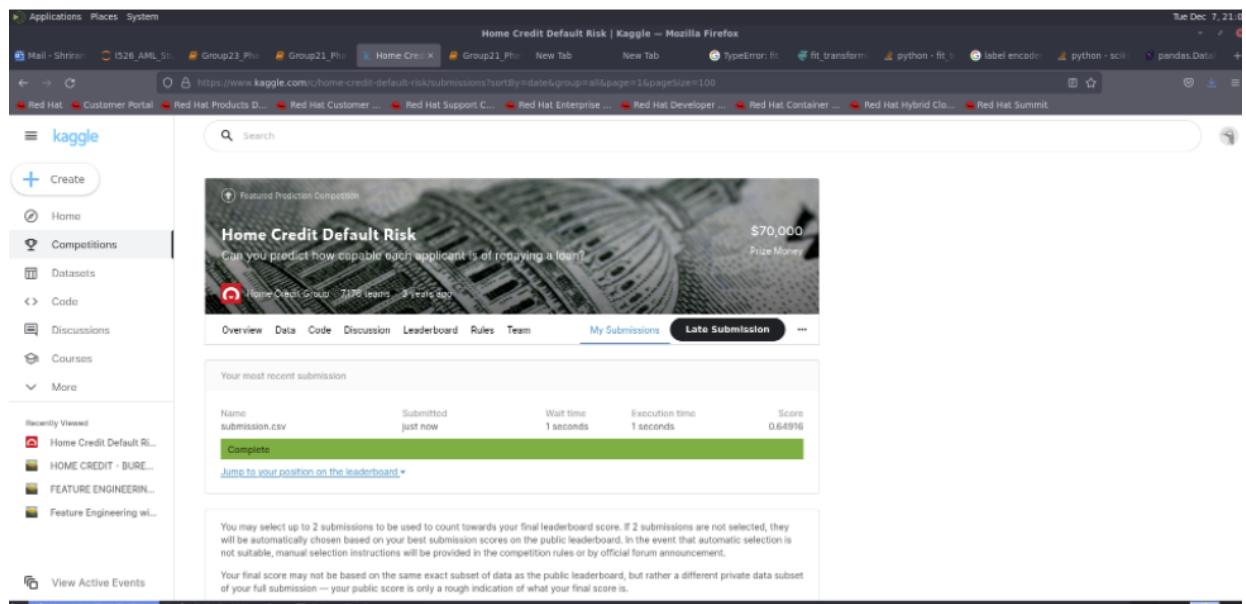
The main purpose of this project is to create a Machine Learning model that can predict whether or not a loan applicant will be able to repay the loan. Many worthy applicants with no credit history or default history are getting without any statistical analysis. The ML model in our work is trained using the HCDR dataset. It will be able to predict whether an applicant will be able to repay his loan or not based on the history of similar applicants in the past. This would help in filtering applicants with a good statistical backing derived from various factors that are taken into consideration. This would help both, a worthy applicant in securing a loan and the bank to grow their business further. We performed feature engineering, feature selection and hyperparameter tuning to improve our classification model to accurately predict whether the loan applicant is able to repay his loan or not. We identified that the Credit Annuity Ratio of the Application feature as the most important feature in our implementation. This feature is basically the ratio between the amount of loan credited to the annual annuity of the loan applicant. The result which we got after modeling and fine tuning provides confidence that it will be able to successfully predict applicants' credit worthiness. Their might be some inaccurate feature selections in our work as we got a decreased score on our kaggle submission. We will be analyzing what features are causing this and add or remove some more features to improve our score.

This is the second iteration of our model and subsequent iterations of the same are to be followed in the next phase. Moving on, we will be performing a deep learning algorithm to further aid our implementation.

Kaggle Submission

Please provide a screenshot of your best kaggle submission.

The screenshot should show the different details of the submission and not just the score.



References

Some of the material in this notebook has been adopted from [here](#)

1. [Understanding AUC - ROC Curve](#)
2. [Better Heatmaps and Correlation Matrix Plots in Python](#)
3. [Bar Plots and Modern Alternatives](#)
4. [Data Visualization using Matplotlib](#)
5. [sklearn.ensemble.RandomForestClassifier](#)
6. [sklearn.ensemble.RandomForestClassifier](#)
7. [Feature Engineering](#)

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/>