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


Kaggle API setup

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

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

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

1. Install library

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

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

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

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.7/site-packag
es (1.5.12)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.7/si
te-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: urllib3 in /usr/local/lib/python3.7/site-packa
ges (from kaggle) (1.26.6)
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.7/site-pac
kages (from kaggle) (1.15.0)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.7/sit
e-packages (from kaggle) (5.0.2)
Requirement already satisfied: certifi in /usr/local/lib/python3.7/site-packa
ges (from kaggle) (2021.5.30)
```

```
Requirement already satisfied: requests in /usr/local/lib/python3.7/site-pack
ages (from kaggle) (2.25.1)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.
7/site-packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: chardet<5,>=3.0.2 in /usr/local/lib/python3.7/
site-packages (from requests->kaggle) (4.0.0)
Requirement already satisfied: idna<3,>=2.5 in /usr/local/lib/python3.7/site-
packages (from requests->kaggle) (2.10)
WARNING: Running pip as the 'root' user can result in broken permissions and
conflicting behaviour with the system package manager. It is recommended to u
se a virtual environment instead: https://pip.pypa.io/warnings/venv
WARNING: You are using pip version 21.2.4; however, version 21.3.1 is availab
le.
You should consider upgrading via the '/usr/local/bin/python -m pip install
```

In [1]:
```
!pwd
```

```
/root/shared/I526_AML_Student/Assignments/Unit-Project-Home-Credit-Default-Ri
sk/HCDR_Phase_1_baseline_submission
```

In [2]:
```
!mkdir ~/.kaggle
!cp /root/shared/Downloads/kaggle.json ~/.kaggle
!chmod 600 ~/.kaggle/kaggle.json
```

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

In [3]:
```
! kaggle competitions files home-credit-default-risk
```

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

Dataset and how to download

Back ground Home Credit Group

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

Home Credit Group

Home Credit strives to broaden financial inclusion for the unbanked population by providing a positive and safe borrowing experience. In order to make sure this underserved

population has a positive loan experience, Home Credit makes use of a variety of alternative data--including telco and transactional information--to predict their clients' repayment abilities.

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

Background on the dataset

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

The company operates in 14 countries (including United States, Russia, Kazahstan, Belarus, China, India) and focuses on lending primarily to people with little or no credit history which will either not obtain loans or became victims of untrustworthly lenders.

Home Credit group has over 29 million customers, total assests of 21 billions Euro, over 160 millions loans, with the majority in Asia and and almost half of them in China (as of 19-05-2018).

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

Data files overview

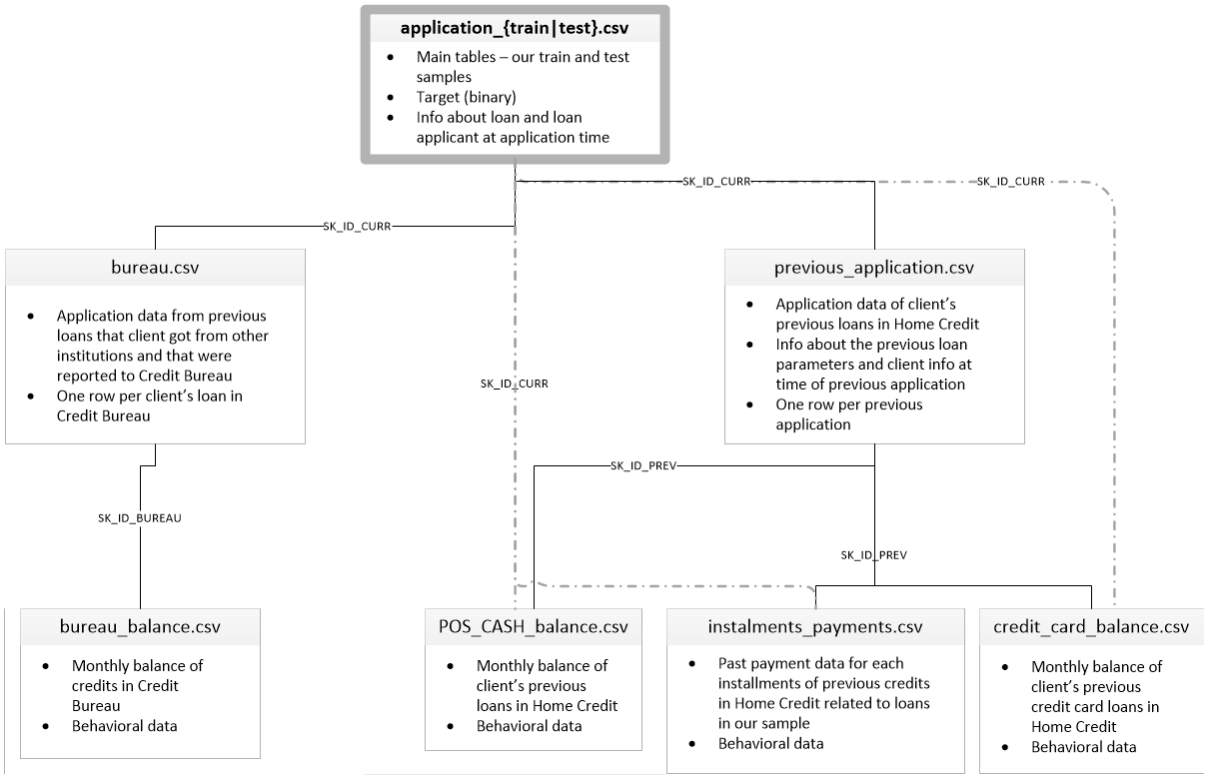
There are 7 different sources of data:

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

each month of the credit length.

- **previous_application:** previous applications for loans at Home Credit of clients who have loans in the application data. Each current loan in the application data can have multiple previous loans. Each previous application has one row and is identified by the feature SK_ID_PREV.
- **POS_CASH_BALANCE:** monthly data about previous point of sale or cash loans clients have had with Home Credit. Each row is one month of a previous point of sale or cash loan, and a single previous loan can have many rows.
- credit_card_balance: monthly data about previous credit cards clients have had with Home Credit. Each row is one month of a credit card balance, and a single credit card can have many rows.
- **installments_payment:** payment history for previous loans at Home Credit. There is one row for every made payment and one row for every missed payment.

In [ ]:
```
# ![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 [4]:
```
DATA_DIR = "../../../Data/home-credit-default-risk"   #same level as course r
#DATA_DIR = os.path.join('./ddddd/')
!mkdir $DATA_DIR
```

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

In [5]:
```
!ls -l $DATA_DIR
```

```
total 2641844
-rwxrwxrwx 1 root root     37383 Dec 11  2019 HomeCredit_columns_description.
csv
-rwxrwxrwx 1 root root 392703158 Dec 11  2019 POS_CASH_balance.csv
-rwxrwxrwx 1 root root  26567651 Dec 11  2019 application_test.csv
-rwxrwxrwx 1 root root 166133370 Dec 11  2019 application_train.csv
-rwxrwxrwx 1 root root 170016717 Dec 11  2019 bureau.csv
-rwxrwxrwx 1 root root 375592889 Dec 11  2019 bureau_balance.csv
-rwxrwxrwx 1 root root 424582605 Dec 11  2019 credit_card_balance.csv
-rw-r--r-- 1 root root  20971520 Nov 15 21:18 home-credit-default-risk.zip
-rwxrwxrwx 1 root root 723118349 Dec 11  2019 installments_payments.csv
-rwxrwxrwx 1 root root 404973293 Dec 11  2019 previous_application.csv
-rwxrwxrwx 1 root root    536202 Dec 11  2019 sample_submission.csv
```

In [41]:
```
! kaggle competitions download home-credit-default-risk -p $DATA_DIR
```

```
Downloading home-credit-default-risk.zip to ../../../Data/home-credit-default
-risk
  3%|▉                                      | 20.0M/688M [00:05<02:53, 4.04MB
/s]^C
  3%|▉                                      | 20.0M/688M [00:05<03:19, 3.51MB
/s]
User cancelled operation
```

Imports

In [77]:

```
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')
```

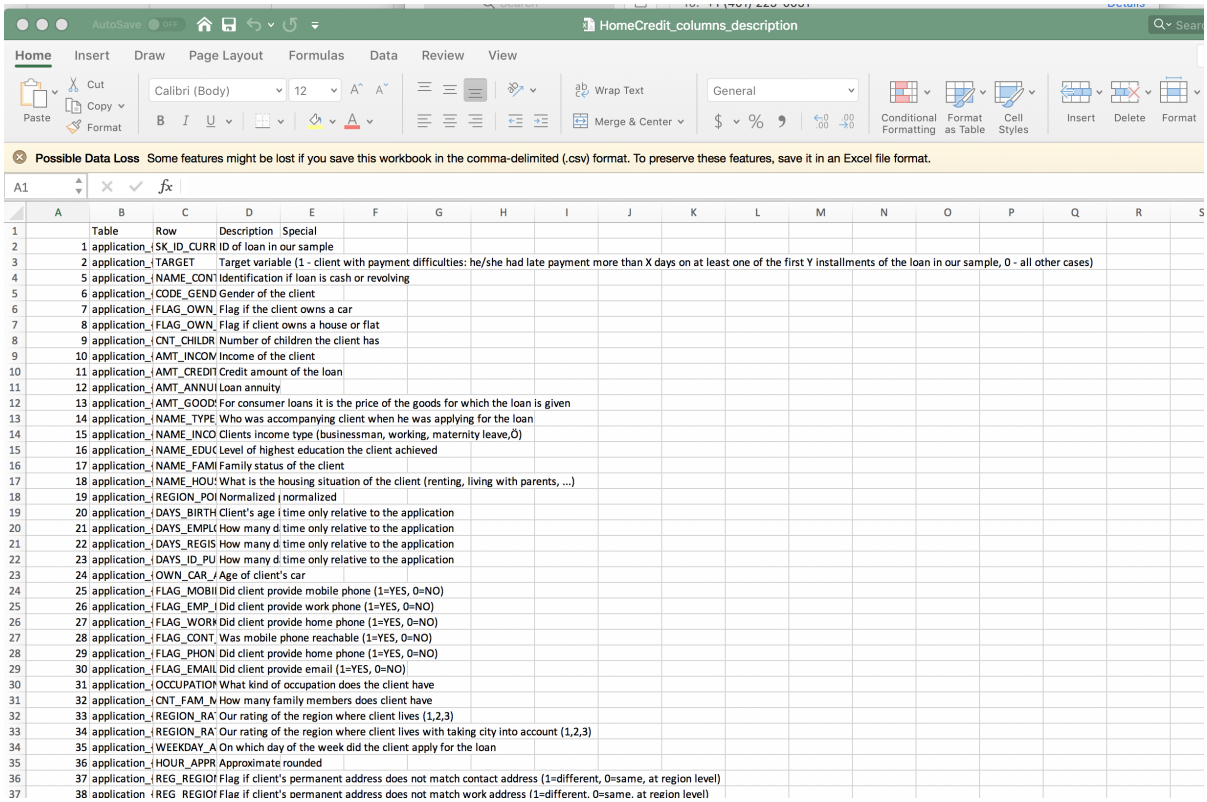
In [12]:

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

Data files overview

Data Dictionary

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



HomeCredit_columns_description

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

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

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

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_OW
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[7]: (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 [8]:
```
ds_name = 'application_test'
datasets[ds_name] = load_data(os.path.join(DATA_DIR, f'{ds_name}.csv'), ds_na
```

```
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_REALT
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 [9]:
```
%%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'), d
```

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

5 rows × 121 columns

```
bureau: shape is (1716428, 17)
```

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

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

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

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

```
   SK_ID_BUREAU  MONTHS_BALANCE  STATUS
credit_card_balance: shape is (3840312, 23)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3840312 entries, 0 to 3840311
Data columns (total 23 columns):
 #   Column                       Dtype
---  ------                       -----
 0   SK_ID_PREV                   int64
 1   SK_ID_CURR                   int64
 2   MONTHS_BALANCE               int64
 3   AMT_BALANCE                  float64
 4   AMT_CREDIT_LIMIT_ACTUAL      int64
 5   AMT_DRAWINGS_ATM_CURRENT     float64
 6   AMT_DRAWINGS_CURRENT         float64
 7   AMT_DRAWINGS_OTHER_CURRENT   float64
 8   AMT_DRAWINGS_POS_CURRENT     float64
 9   AMT_INST_MIN_REGULARITY      float64
 10  AMT_PAYMENT_CURRENT          float64
 11  AMT_PAYMENT_TOTAL_CURRENT    float64
 12  AMT_RECEIVABLE_PRINCIPAL     float64
 13  AMT_RECIVABLE                float64
 14  AMT_TOTAL_RECEIVABLE         float64
 15  CNT_DRAWINGS_ATM_CURRENT     float64
 16  CNT_DRAWINGS_CURRENT         int64
 17  CNT_DRAWINGS_OTHER_CURRENT   float64
 18  CNT_DRAWINGS_POS_CURRENT     float64
 19  CNT_INSTALMENT_MATURE_CUM    float64
 20  NAME_CONTRACT_STATUS         object
 21  SK_DPD                       int64
 22  SK_DPD_DEF                   int64
dtypes: float64(15), int64(7), object(1)
memory usage: 673.9+ MB
None
```

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	AMT_CREDIT_LIMIT_ACTUAI
0	2562384	378907	-6	56.970	13500(
1	2582071	363914	-1	63975.555	4500(
2	1740877	371185	-7	31815.225	45000(
3	1389973	337855	-4	236572.110	22500(
4	1891521	126868	-1	453919.455	45000(

5 rows × 23 columns

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

```
 6   AMT_INSTALMENT          float64
 7   AMT_PAYMENT             float64
dtypes: float64(5), int64(3)
memory usage: 830.4 MB
None
```

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER	DAYS
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
```

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AM
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_FUTU
0	1803195	182943	-31	48.0	4
1	1715348	367990	-33	36.0	3
2	1784872	397406	-32	12.0	
3	1903291	269225	-35	48.0	4
4	2341044	334279	-35	36.0	3

```
CPU times: user 40.8 s, sys: 4.79 s, total: 45.6 s
Wall time: 1min 21s
```

In [10]:

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

```
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
```

Out[9]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	A
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	3
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	
std	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	1
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	1
max	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	25

8 rows × 106 columns

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

Out[10]:

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

8 rows × 105 columns

In [11]:
```
datasets["application_train"].describe(include='all') #look at all categorica
```

Out[11]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_C
count	307511.000000	307511.000000	307511	307511	307
unique	NaN	NaN	2	3	
top	NaN	NaN	Cash loans	F	
freq	NaN	NaN	278232	202448	202
mean	278180.518577	0.080729	NaN	NaN	N
std	102790.175348	0.272419	NaN	NaN	N
min	100002.000000	0.000000	NaN	NaN	N
25%	189145.500000	0.000000	NaN	NaN	N
50%	278202.000000	0.000000	NaN	NaN	N
75%	367142.500000	0.000000	NaN	NaN	N
max	456255.000000	1.000000	NaN	NaN	N

11 rows × 122 columns

In [15]:
```
datasets["application_train"].size
```

Out[15]:
```
37516342
```

In [16]:
```
datasets["application_train"].shape
```

Out[16]:
```
(307511, 122)
```

Dividing the train dataset into train and validation sets

In [17]:
```
X = datasets["application_train"].drop(['TARGET'], axis = 1)
y = datasets["application_train"]["TARGET"]

# Split the provided training data into training and validationa and test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.20, ran
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_
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 [18]:
```
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[18]:

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	A
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	14475.618426	
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 [20]:
```
# Concatenating X_train and y_train
Xy_train = pd.concat([X_train, y_train], axis=1)
Xy_train.head()
```

Out[20]:

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_
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 [19]:
```
numerical_features = X_train.select_dtypes(include = ['int64', 'float64']).co
categorical_features = X_train.select_dtypes(include = ['object', 'bool']).co
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', 'DAY
S_BIRTH', 'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'DAYS_ID_PUBLISH', 'OWN_CAR_A
GE', '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_REGIO
N', 'REG_REGION_NOT_WORK_REGION', 'LIVE_REGION_NOT_WORK_REGION', 'REG_CITY_NO
T_LIVE_CITY', 'REG_CITY_NOT_WORK_CITY', 'LIVE_CITY_NOT_WORK_CITY', 'EXT_SOURC
E_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', 'LI
VINGAPARTMENTS_AVG', 'LIVINGAREA_AVG', 'NONLIVINGAPARTMENTS_AVG', 'NONLIVINGA
REA_AVG', 'APARTMENTS_MODE', 'BASEMENTAREA_MODE', 'YEARS_BEGINEXPLUATATION_MO
DE', 'YEARS_BUILD_MODE', 'COMMONAREA_MODE', 'ELEVATORS_MODE', 'ENTRANCES_MODE
', 'FLOORSMAX_MODE', 'FLOORSMIN_MODE', 'LANDAREA_MODE', 'LIVINGAPARTMENTS_MOD
E', 'LIVINGAREA_MODE', 'NONLIVINGAPARTMENTS_MODE', 'NONLIVINGAREA_MODE', 'APA
RTMENTS_MEDI', 'BASEMENTAREA_MEDI', 'YEARS_BEGINEXPLUATATION_MEDI', 'YEARS_BU
ILD_MEDI', 'COMMONAREA_MEDI', 'ELEVATORS_MEDI', 'ENTRANCES_MEDI', 'FLOORSMAX_
MEDI', 'FLOORSMIN_MEDI', 'LANDAREA_MEDI', 'LIVINGAPARTMENTS_MEDI', 'LIVINGARE
A_MEDI', 'NONLIVINGAPARTMENTS_MEDI', 'NONLIVINGAREA_MEDI', 'TOTALAREA_MODE',
'OBS_30_CNT_SOCIAL_CIRCLE', 'DEF_30_CNT_SOCIAL_CIRCLE', 'OBS_60_CNT_SOCIAL_CI
RCLE', '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_DOCUM
ENT_18', 'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20', 'FLAG_DOCUMENT_21', 'AMT_REQ
_CREDIT_BUREAU_HOUR', 'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEE
K', '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_TYP
E', 'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'OCCUPATION_TYPE', 'WEEKDAY_AP
PR_PROCESS_START', 'ORGANIZATION_TYPE', 'FONDKAPREMONT_MODE', 'HOUSETYPE_MODE
', 'WALLSMATERIAL_MODE', 'EMERGENCYSTATE_MODE']
```

Missing data Analysis

Missing data for application train

In [24]:

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

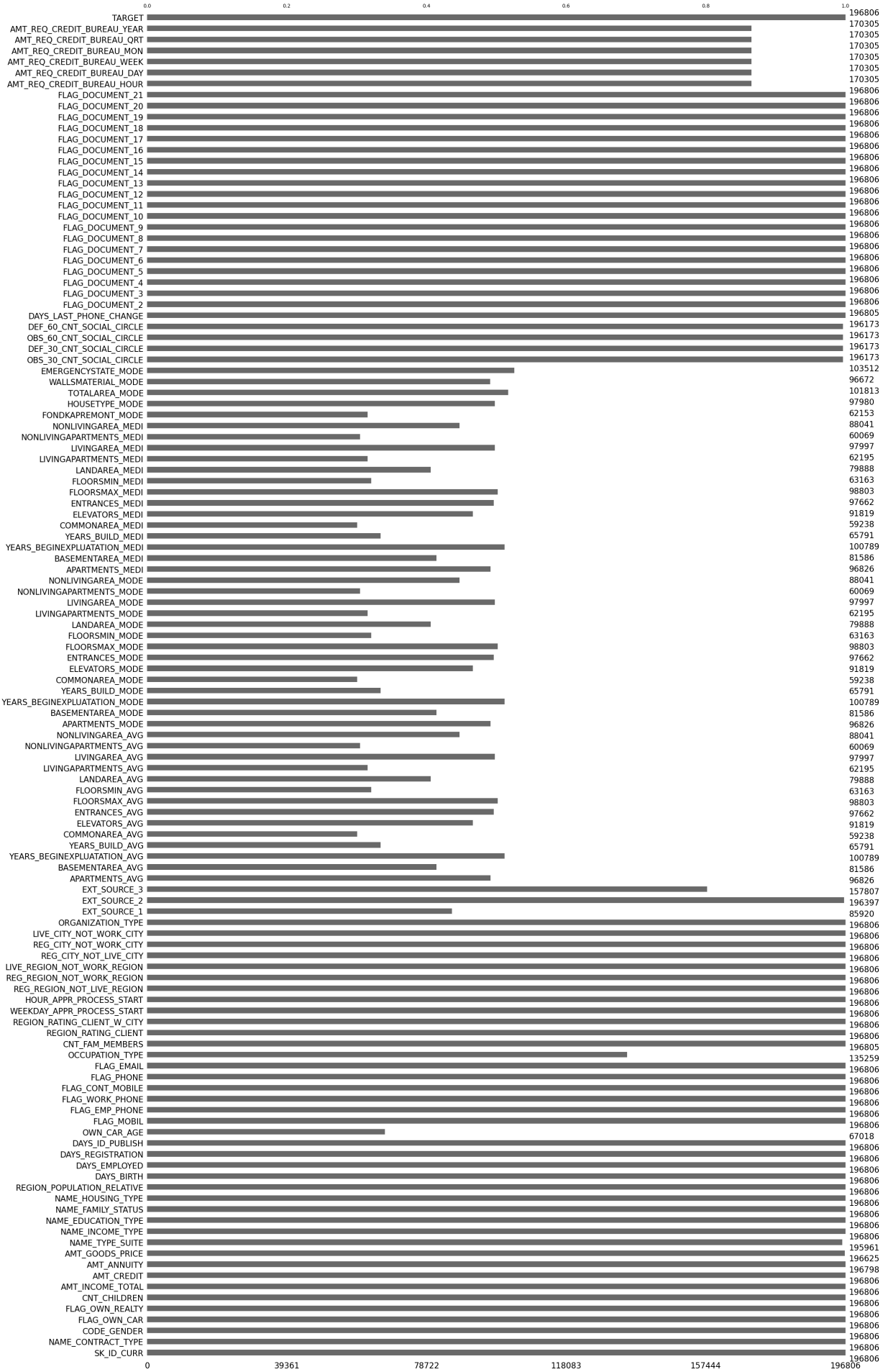
Out[24]:

	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

	Percent	Train Missing Count
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

Bar plot of missing values in each column

In [19]:
```
msno.bar(Xy_train)
plt.show()
```


0.0
0.2
0.4
0.6
0.8
1.0
TARGET 196806
AMT_REQ_CREDIT_BUREAU_YEAR 170305
AMT_REQ_CREDIT_BUREAU_QRT 170305
AMT_REQ_CREDIT_BUREAU_MON 170305
AMT_REQ_CREDIT_BUREAU_WEEK 170305
AMT_REQ_CREDIT_BUREAU_DAY 170305
AMT_REQ_CREDIT_BUREAU_HOUR 170305
FLAG_DOCUMENT_21 196806
FLAG_DOCUMENT_20 196806
FLAG_DOCUMENT_19 196806
FLAG_DOCUMENT_18 196806
FLAG_DOCUMENT_17 196806
FLAG_DOCUMENT_16 196806
FLAG_DOCUMENT_15 196806
FLAG_DOCUMENT_14 196806
FLAG_DOCUMENT_13 196806
FLAG_DOCUMENT_12 196806
FLAG_DOCUMENT_11 196806
FLAG_DOCUMENT_10 196806
FLAG_DOCUMENT_9 196806
FLAG_DOCUMENT_8 196806
FLAG_DOCUMENT_7 196806
FLAG_DOCUMENT_6 196806
FLAG_DOCUMENT_5 196806
FLAG_DOCUMENT_4 196806
FLAG_DOCUMENT_3 196806
FLAG_DOCUMENT_2 196806
DAYS_LAST_PHONE_CHANGE 196805
DEF_60_CNT_SOCIAL_CIRCLE 196173
OBS_60_CNT_SOCIAL_CIRCLE 196173
DEF_30_CNT_SOCIAL_CIRCLE 196173
OBS_30_CNT_SOCIAL_CIRCLE 196173
EMERGENCYSTATE_MODE 103512
WALLSMATERIAL_MODE 96672
TOTALAREA_MODE 101813
HOUSETYPE_MODE 97980
FONDKAPREMONT_MODE 62153
NONLIVINGAREA_MEDI 88041
NONLIVINGAPARTMENTS_MEDI 60069
LIVINGAREA_MEDI 97997
LIVINGAPARTMENTS_MEDI 62195
LANDAREA_MEDI 79888
FLOORSMIN_MEDI 63163
FLOORSMAX_MEDI 98803
ENTRANCES_MEDI 97662
ELEVATORS_MEDI 91819
COMMONAREA_MEDI 59238
YEARS_BUILD_MEDI 65791
YEARS_BEGINEXPLUATATION_MEDI 100789
BASEMENTAREA_MEDI 81586
APARTMENTS_MEDI 96826
NONLIVINGAREA_MODE 88041
NONLIVINGAPARTMENTS_MODE 60069
LIVINGAREA_MODE 97997
LIVINGAPARTMENTS_MODE 62195
LANDAREA_MODE 79888
FLOORSMIN_MODE 63163
FLOORSMAX_MODE 98803
ENTRANCES_MODE 97662
ELEVATORS_MODE 91819
COMMONAREA_MODE 59238
YEARS_BUILD_MODE 65791
YEARS_BEGINEXPLUATATION_MODE 100789
BASEMENTAREA_MODE 81586
APARTMENTS_MODE 96826
NONLIVINGAREA_AVG 88041
NONLIVINGAPARTMENTS_AVG 60069
LIVINGAREA_AVG 97997
LIVINGAPARTMENTS_AVG 62195
LANDAREA_AVG 79888
FLOORSMIN_AVG 63163
FLOORSMAX_AVG 98803
ENTRANCES_AVG 97662
ELEVATORS_AVG 91819
COMMONAREA_AVG 59238
YEARS_BUILD_AVG 65791
YEARS_BEGINEXPLUATATION_AVG 100789
BASEMENTAREA_AVG 81586
APARTMENTS_AVG 96826
EXT_SOURCE_3 157807
EXT_SOURCE_2 196397
EXT_SOURCE_1 85920
ORGANIZATION_TYPE 196806
LIVE_CITY_NOT_WORK_CITY 196806
REG_CITY_NOT_WORK_CITY 196806
REG_CITY_NOT_LIVE_CITY 196806
LIVE_REGION_NOT_WORK_REGION 196806
REG_REGION_NOT_WORK_REGION 196806
REG_REGION_NOT_LIVE_REGION 196806
HOUR_APPR_PROCESS_START 196806
WEEKDAY_APPR_PROCESS_START 196806
REGION_RATING_CLIENT_W_CITY 196806
REGION_RATING_CLIENT 196806
CNT_FAM_MEMBERS 196805
OCCUPATION_TYPE 135259
FLAG_EMAIL 196806
FLAG_PHONE 196806
FLAG_CONT_MOBILE 196806
FLAG_WORK_PHONE 196806
FLAG_EMP_PHONE 196806
FLAG_MOBIL 196806
OWN_CAR_AGE 67018
DAYS_ID_PUBLISH 196806
DAYS_REGISTRATION 196806
DAYS_EMPLOYED 196806
DAYS_BIRTH 196806
REGION_POPULATION_RELATIVE 196806
NAME_HOUSING_TYPE 196806
NAME_FAMILY_STATUS 196806
NAME_EDUCATION_TYPE 196806
NAME_INCOME_TYPE 196806
NAME_TYPE_SUITE 195961
AMT_GOODS_PRICE 196625
AMT_ANNUITY 196798
AMT_CREDIT 196806
AMT_INCOME_TOTAL 196806
CNT_CHILDREN 196806
FLAG_OWN_REALTY 196806
FLAG_OWN_CAR 196806
CODE_GENDER 196806
NAME_CONTRACT_TYPE 196806
SK_ID_CURR 196806
0
39361
78722
118083
157444
196806

In [22]:
```
Xy_train.isnull().sum()
```

Out[22]:
```
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 [25]:
```
missing_application_train_data
```

Out[25]:

	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 [26]:
```
percent_data = missing_application_train_data.iloc[:,0]
missing_application_train_data['Percent']
percent_data_df = pd.DataFrame(missing_application_train_data['Percent'], col
```

In [27]:
```
percent_data_df = percent_data_df[percent_data_df['Percent'] != 0.0]
percent_data_df
```

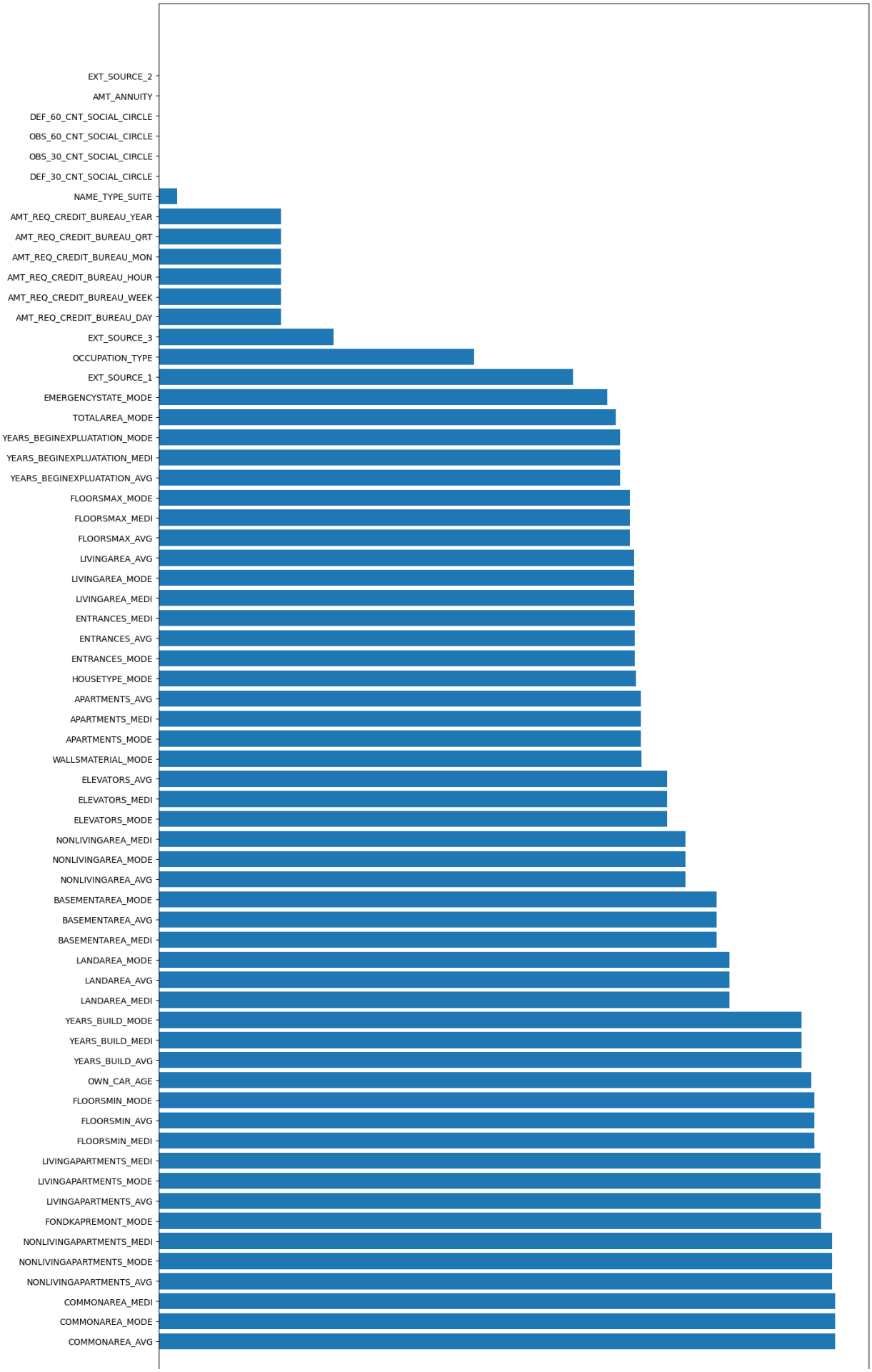
Out[27]:

	Percent
COMMONAREA_MEDI	69.87
COMMONAREA_AVG	69.87

	Percent
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

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()
```


EXT_SOURCE_2
AMT_ANNUITY
DEF_60_CNT_SOCIAL_CIRCLE
OBS_60_CNT_SOCIAL_CIRCLE
OBS_30_CNT_SOCIAL_CIRCLE
DEF_30_CNT_SOCIAL_CIRCLE
NAME_TYPE_SUITE
AMT_REQ_CREDIT_BUREAU_YEAR
AMT_REQ_CREDIT_BUREAU_QRT
AMT_REQ_CREDIT_BUREAU_MON
AMT_REQ_CREDIT_BUREAU_HOUR
AMT_REQ_CREDIT_BUREAU_WEEK
AMT_REQ_CREDIT_BUREAU_DAY
EXT_SOURCE_3
OCCUPATION_TYPE
EXT_SOURCE_1
EMERGENCYSTATE_MODE
TOTALAREA_MODE
YEARS_BEGINEXPLUATATION_MODE
YEARS_BEGINEXPLUATATION_MEDI
YEARS_BEGINEXPLUATATION_AVG
FLOORSMAX_MODE
FLOORSMAX_MEDI
FLOORSMAX_AVG
LIVINGAREA_AVG
LIVINGAREA_MODE
LIVINGAREA_MEDI
ENTRANCES_MEDI
ENTRANCES_AVG
ENTRANCES_MODE
HOUSETYPE_MODE
APARTMENTS_AVG
APARTMENTS_MEDI
APARTMENTS_MODE
WALLSMATERIAL_MODE
ELEVATORS_AVG
ELEVATORS_MEDI
ELEVATORS_MODE
NONLIVINGAREA_MEDI
NONLIVINGAREA_MODE
NONLIVINGAREA_AVG
BASEMENTAREA_MODE
BASEMENTAREA_AVG
BASEMENTAREA_MEDI
LANDAREA_MODE
LANDAREA_AVG
LANDAREA_MEDI
YEARS_BUILD_MODE
YEARS_BUILD_MEDI
YEARS_BUILD_AVG
OWN_CAR_AGE
FLOORSMIN_MODE
FLOORSMIN_AVG
FLOORSMIN_MEDI
LIVINGAPARTMENTS_MEDI
LIVINGAPARTMENTS_MODE
LIVINGAPARTMENTS_AVG
FONDKAPREMONT_MODE
NONLIVINGAPARTMENTS_MEDI
NONLIVINGAPARTMENTS_MODE
NONLIVINGAPARTMENTS_AVG
COMMONAREA_MEDI
COMMONAREA_MODE
COMMONAREA_AVG

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_

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

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GO
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_nam
X_train_transformed_df
```

Out[30]:

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AM
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	

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AM
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	

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

	TARGET	abs_Target
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

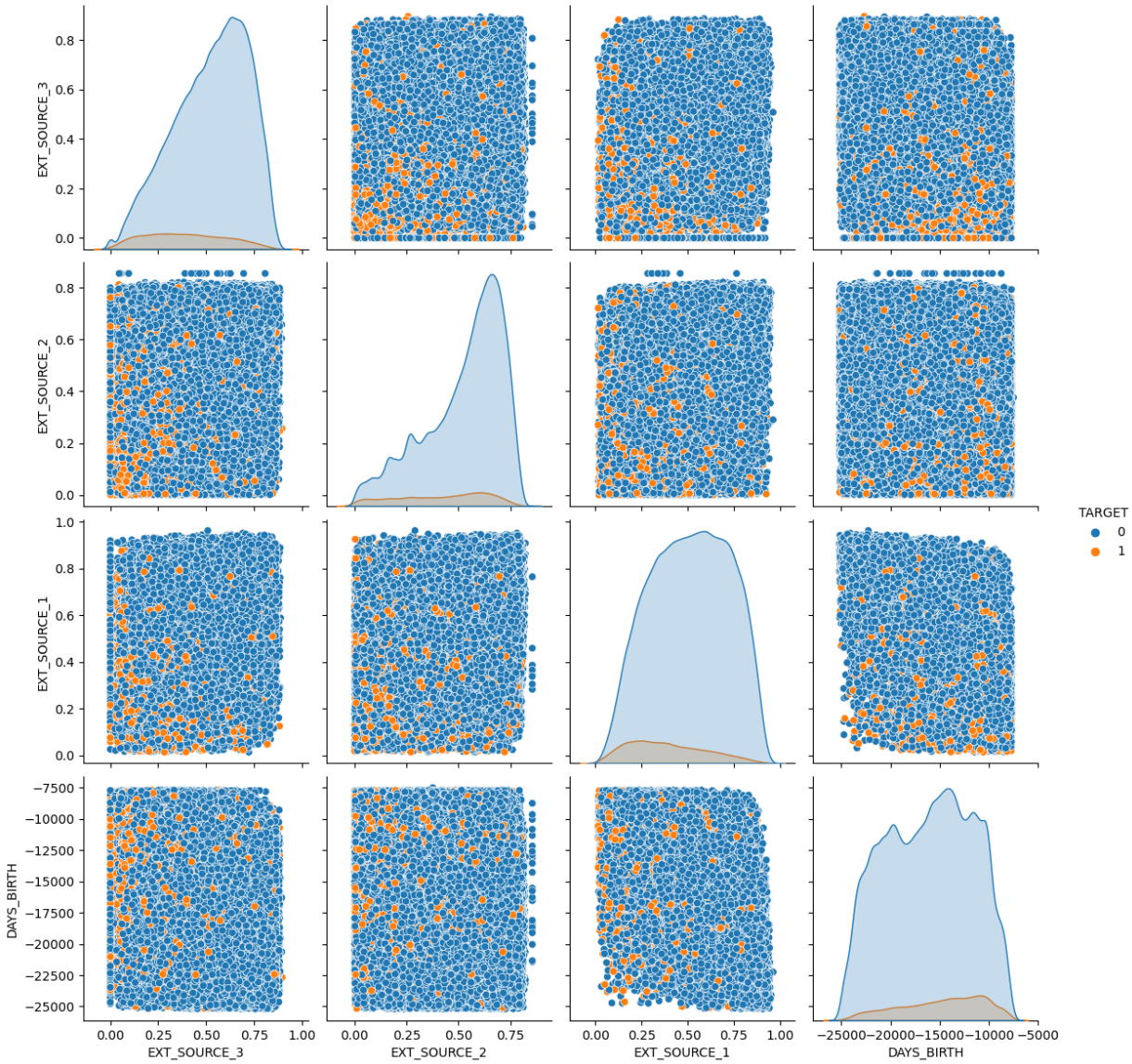
Out[34]:

	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 = attri
plt.show()
```



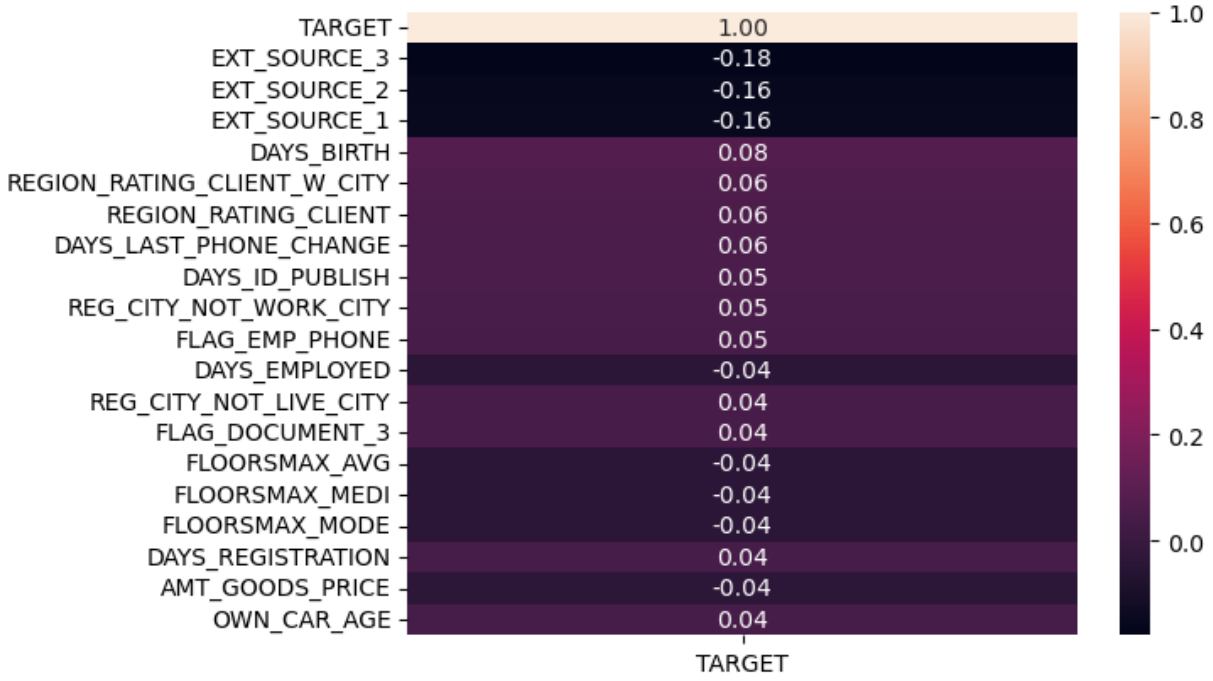
We can see the plot of the top 4 correlated attributes with the Target column. EXT_SOURCE_1 seems to be normally distributed while others are skewed but can be approximated to normal distribution.

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

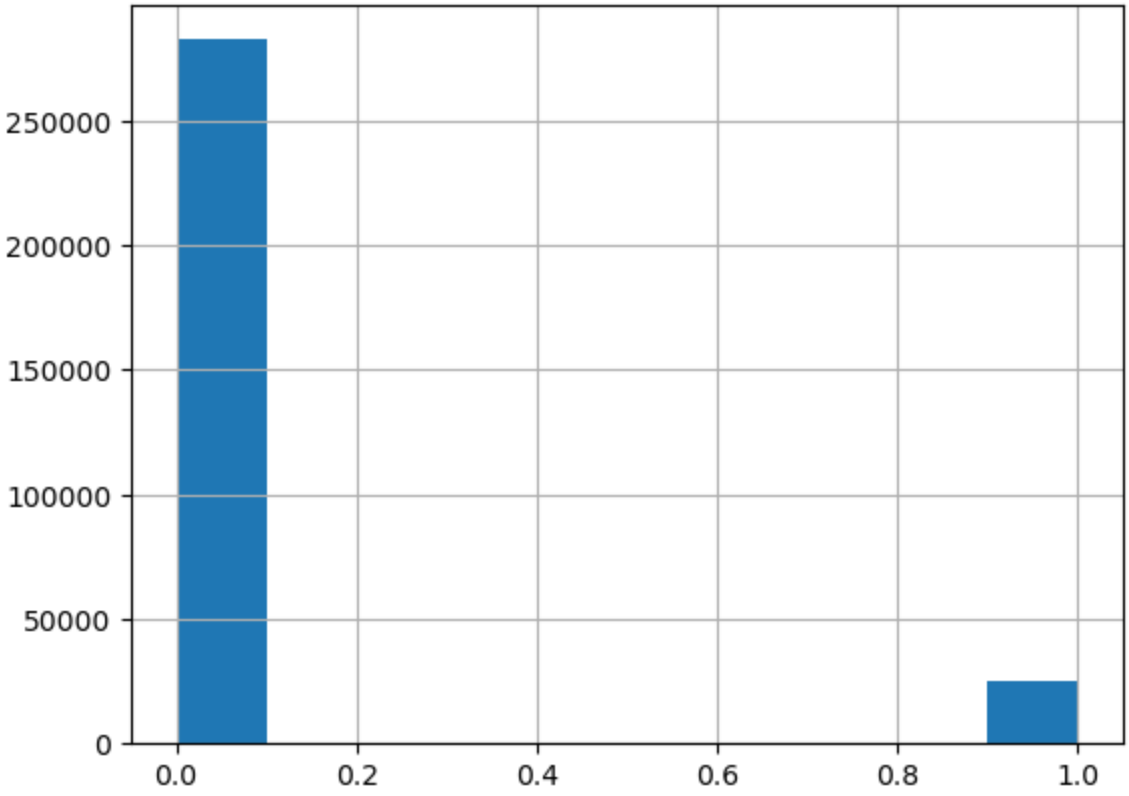
In [51]:
```
plt.figure(figsize=(15,7))
plot = corr_target[1:].plot(kind = 'bar', color = 'grey')
plt.setp(plot.get_xticklabels(), rotation=90)
plt.show()
```

```
<Figure size 1500x700 with 0 Axes>
```

Visual EDA

Distribution of the target column

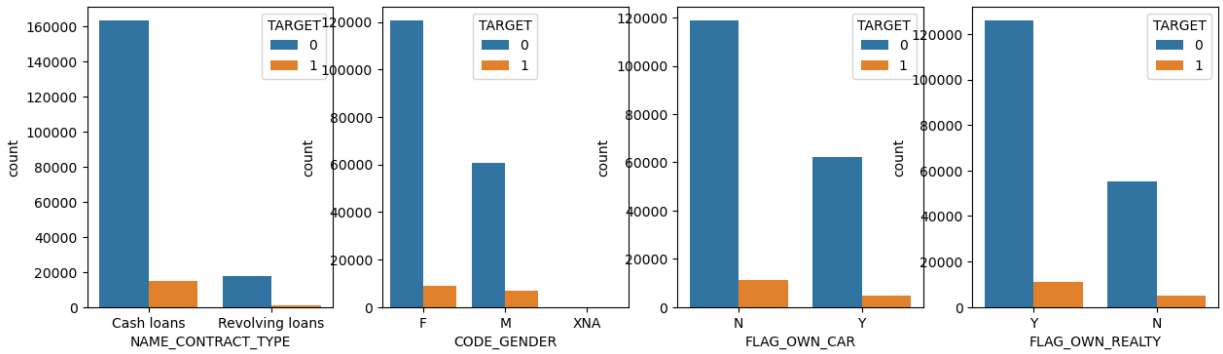
In [52]:
```
datasets["application_train"]['TARGET'].astype(int).hist()
plt.show()
```



Evaluating categorical features with respect to TARGET

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

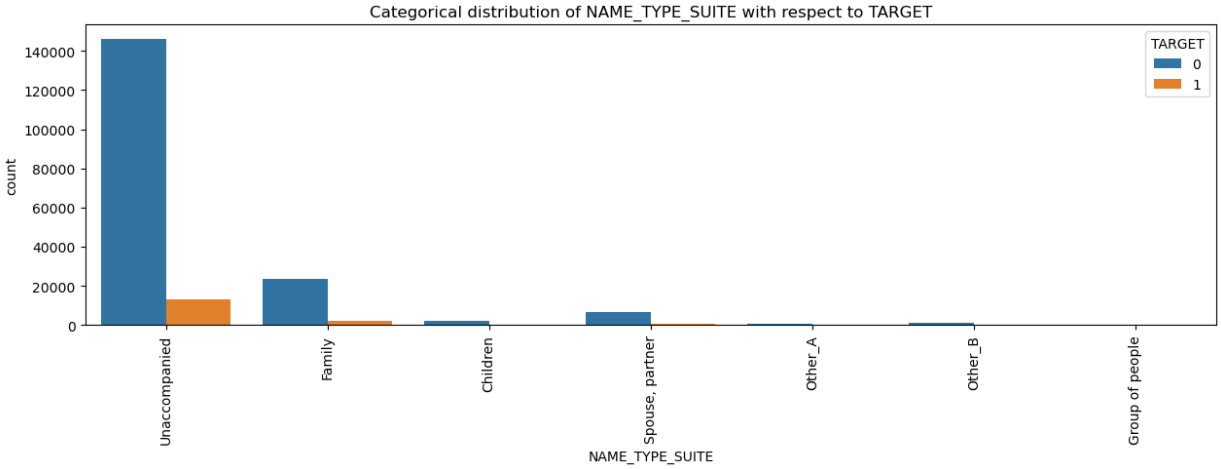
In [55]:
```
cat_vars = list(categorical_features)[:4]
plt.figure(figsize=(15,4))
for idx, cat in enumerate(cat_vars):
    plt.subplot(1, len(cat_vars), idx+1)
    sns.countplot(Xy_train[cat], hue=Xy_train['TARGET'])
plt.show()
```



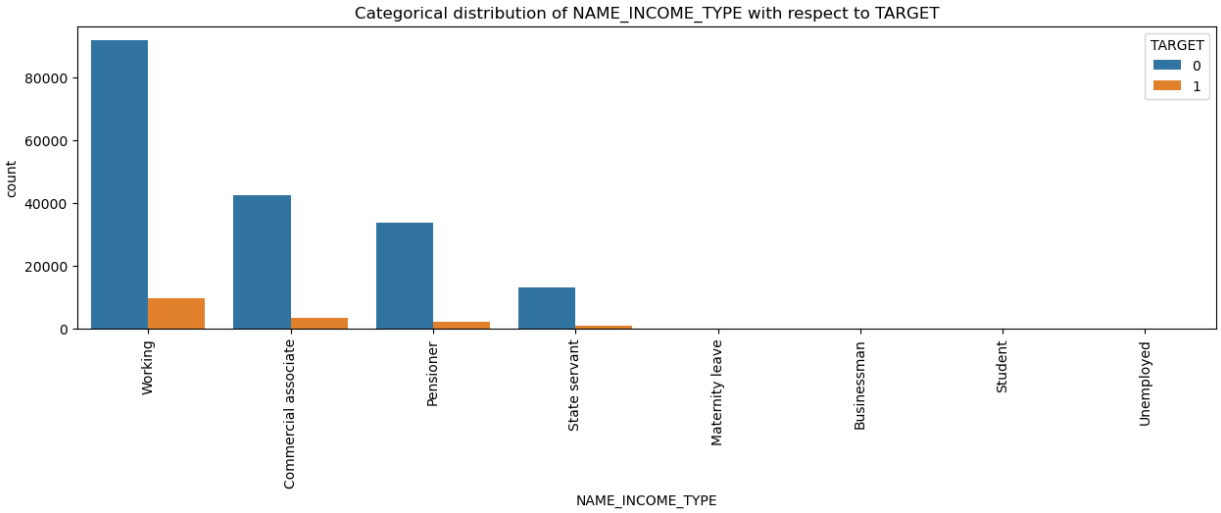
- We notice that Cash loans are beind repaid more than Revolving loans
- More number of female borrowers have repaid loans than male borrowers
- We notice that a client who do not own a car repays loan easily than a client who owns a car
- We notice the same trend for a client who owns a house or flat.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

In [144…
```
explainations = ['We can see that people who are not accompanied by anyone th
```

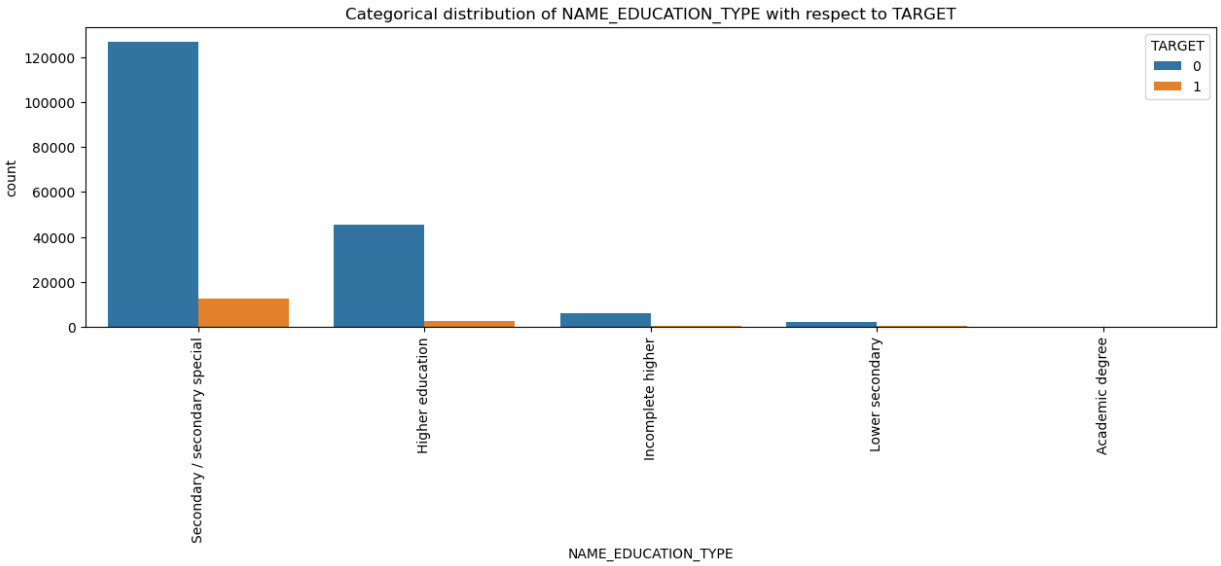
In [145…
```
i = 0
for cat in list(categorical_features[4:14]):
    plt.figure(figsize=(15,4))
    plot = sns.countplot(x=cat, data=Xy_train, hue = Xy_train['TARGET'])
    plt.setp(plot.get_xticklabels(), rotation=90)
    plt.title(f'Categorical distribution of {cat} with respect to TARGET')
    plt.show()
    print(explainations[i])
    i += 1
```



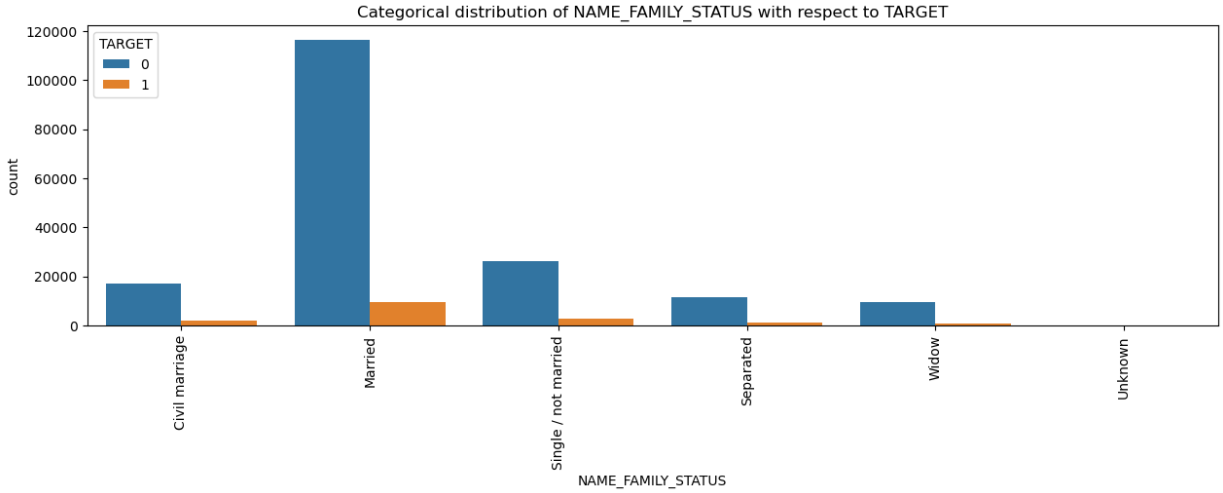
We can see that people who are not accompanied by anyone that is people havin g no dependents are able to repay loans easier.



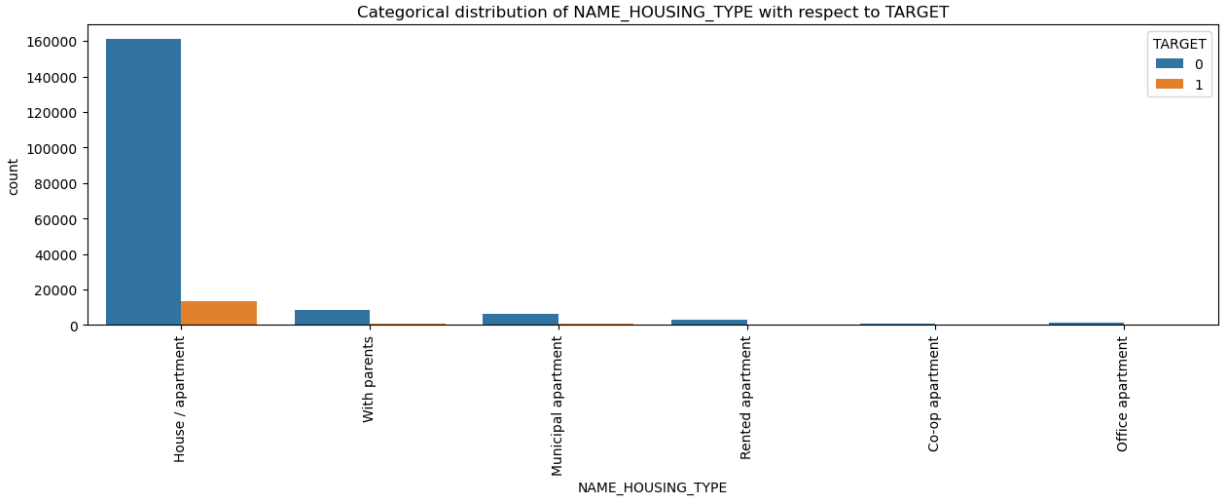
Working class people usually require more loans as compared to other income t ype people.



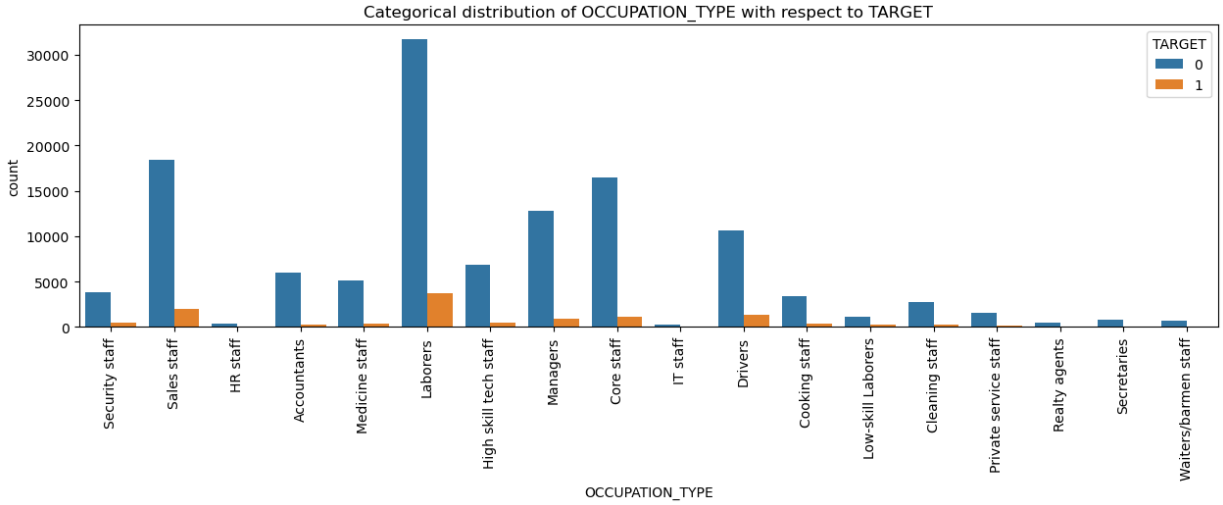
We see that people who have education as Secondary/Secondary special require more loans than people of other education backgrounds.



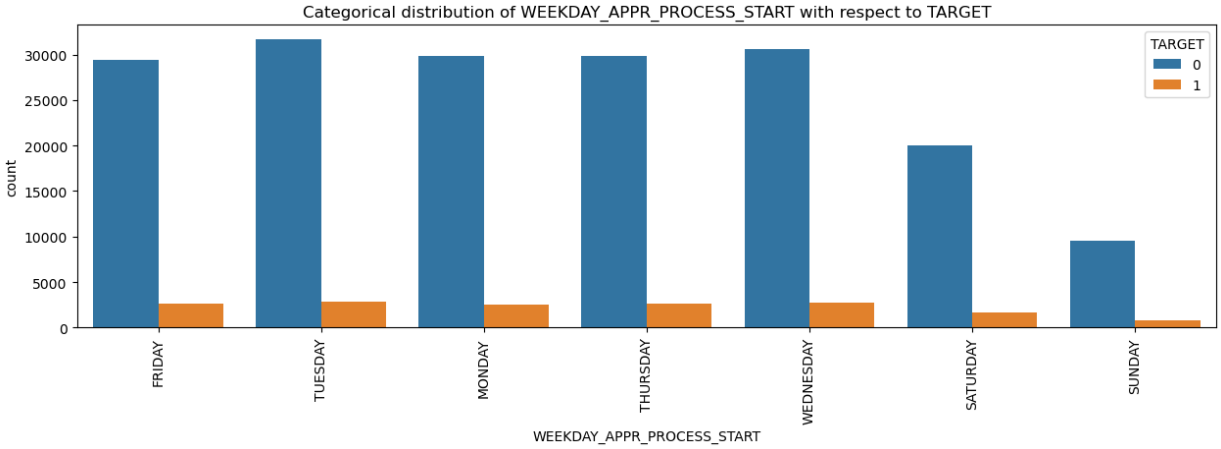
People who are married have taken more loans and repaid them as compared to p eople having other than marriage family status



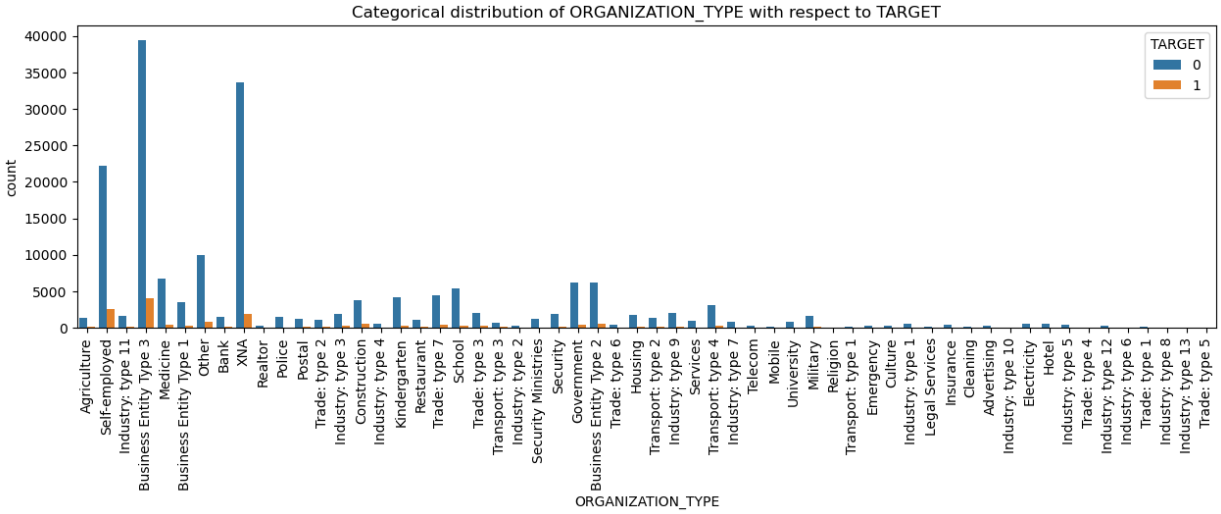
We see that people who live alone and in apartments require more loans than o ther people



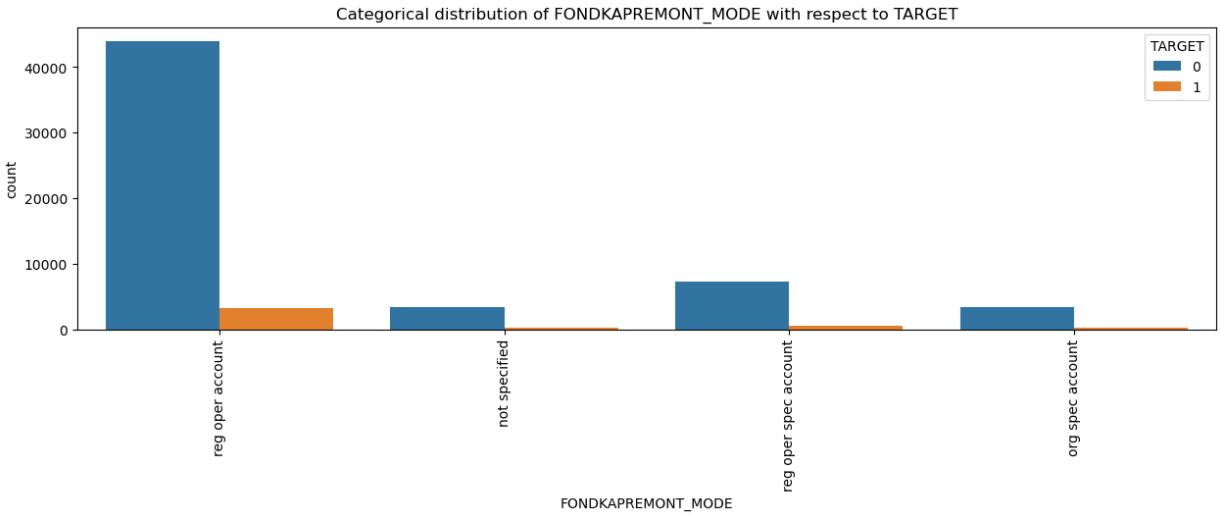
Laborers have repaid more loans than people with other occupations

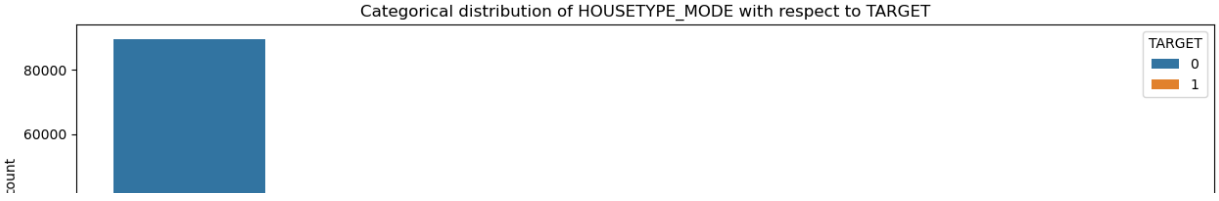


People have repaid more loans on Tuesday than any other day of the week



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

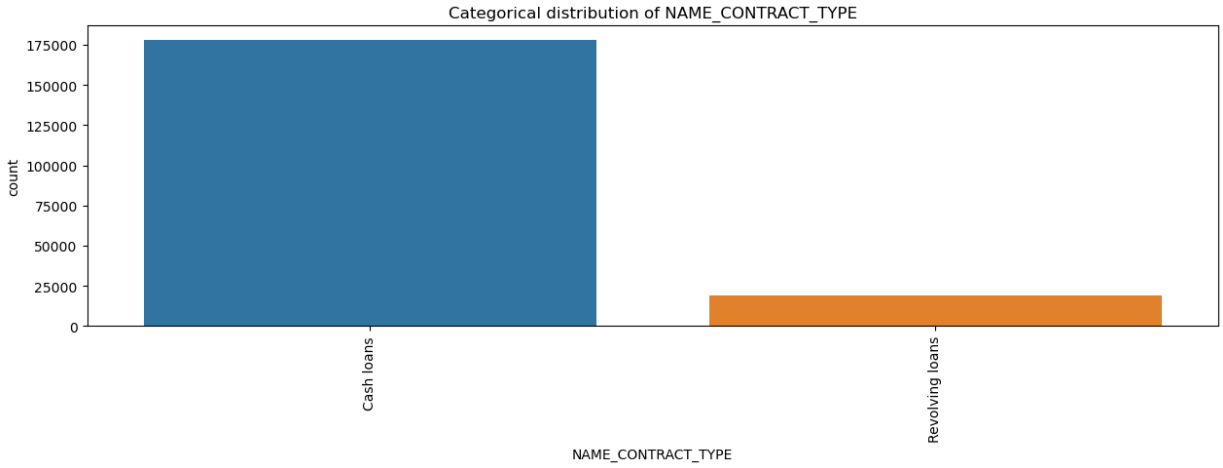


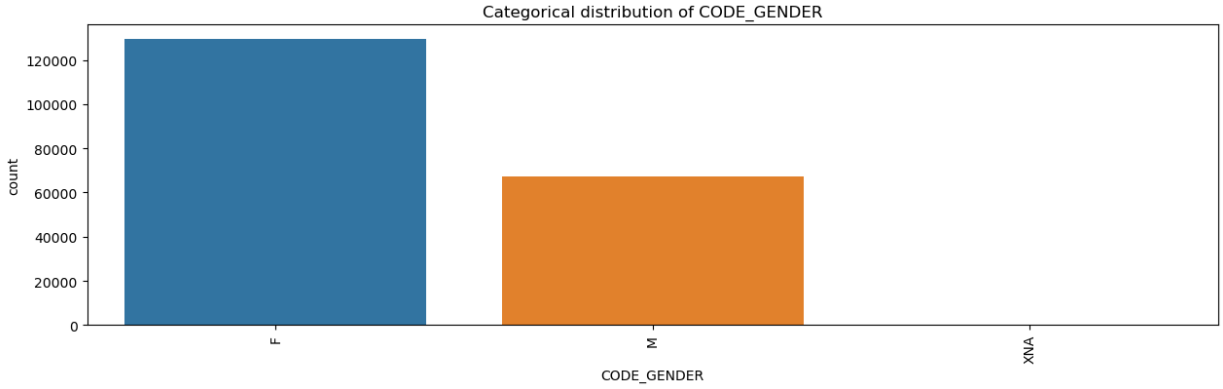


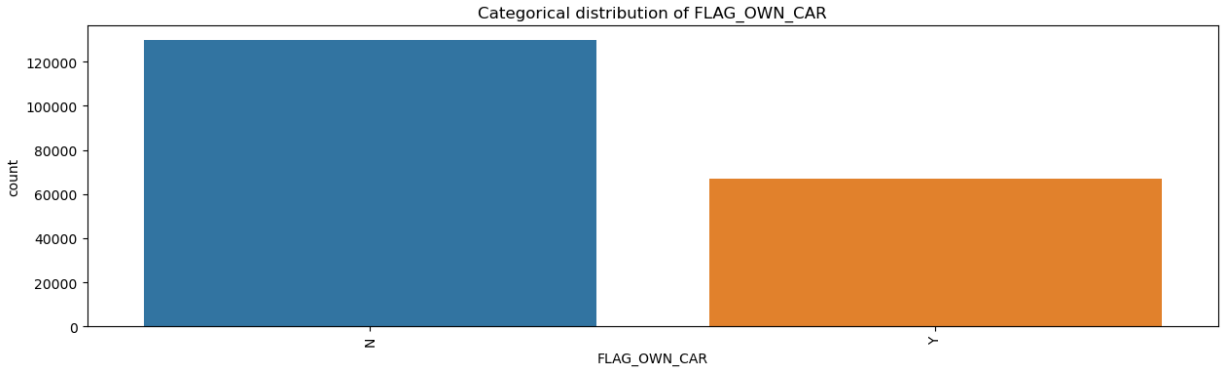
Categorical distribution

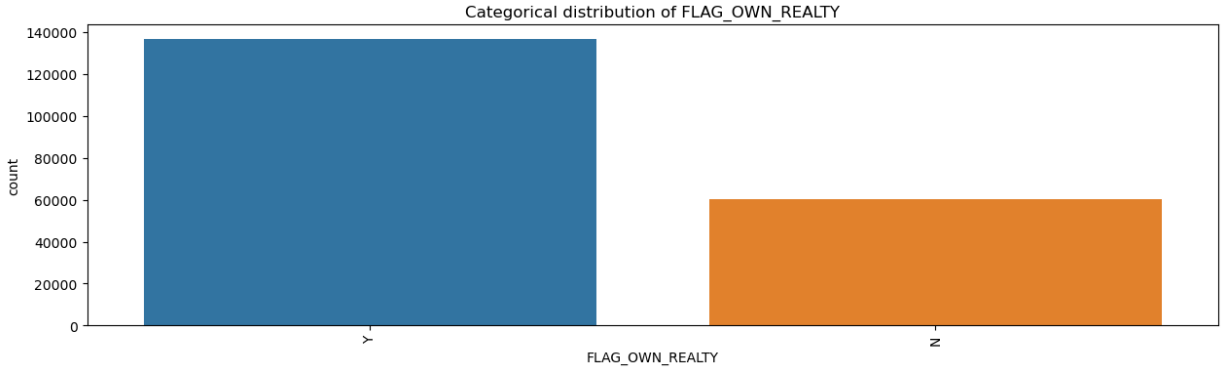
In [149…

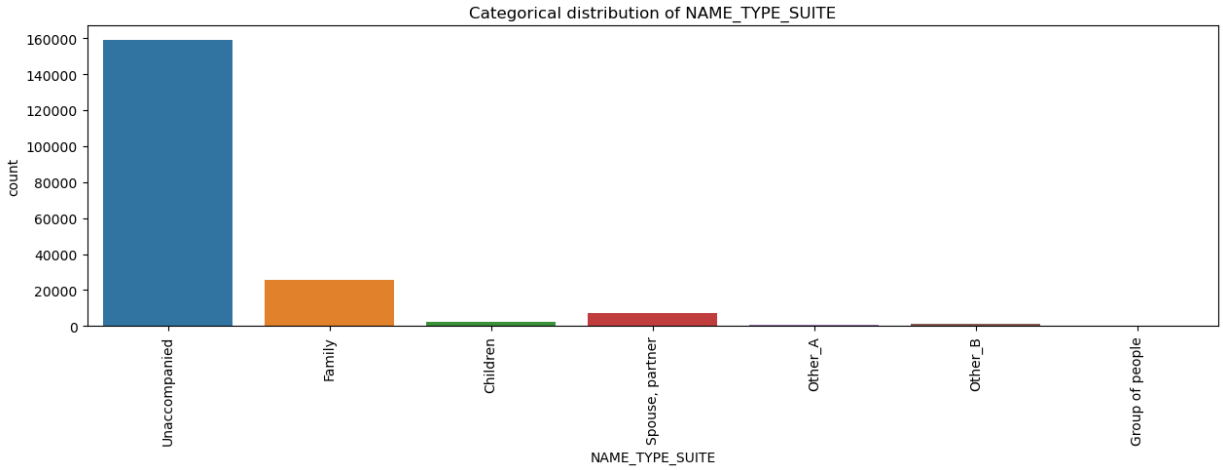
```
for cat in list(categorical_features[:14]):
    plt.figure(figsize=(15,4))
    plot = sns.countplot(x=cat, data=Xy_train)
    plt.setp(plot.get_xticklabels(), rotation=90)
    plt.title(f'Categorical distribution of {cat}')
    plt.show()
```

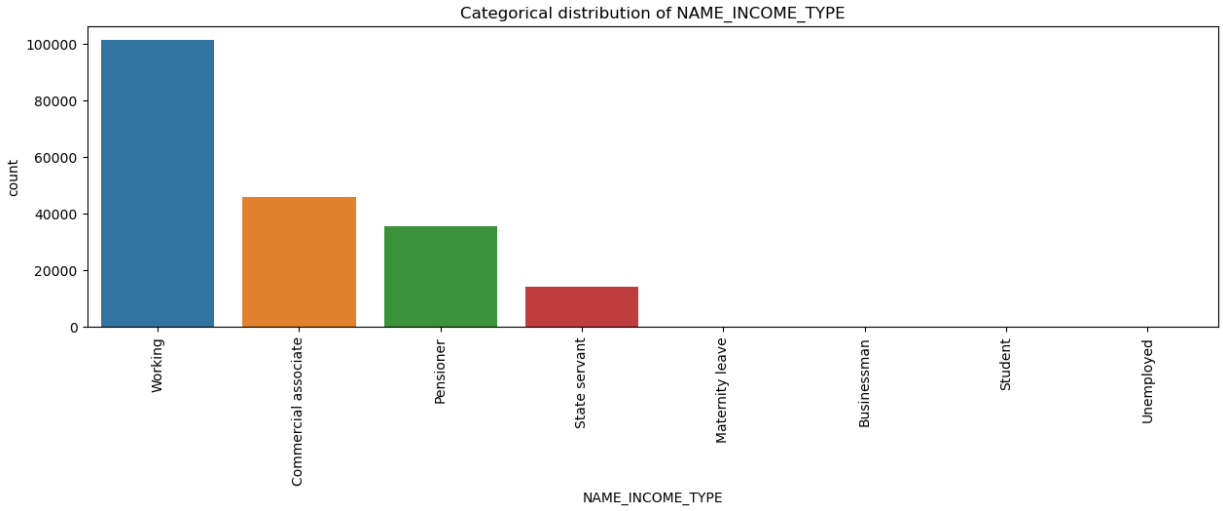


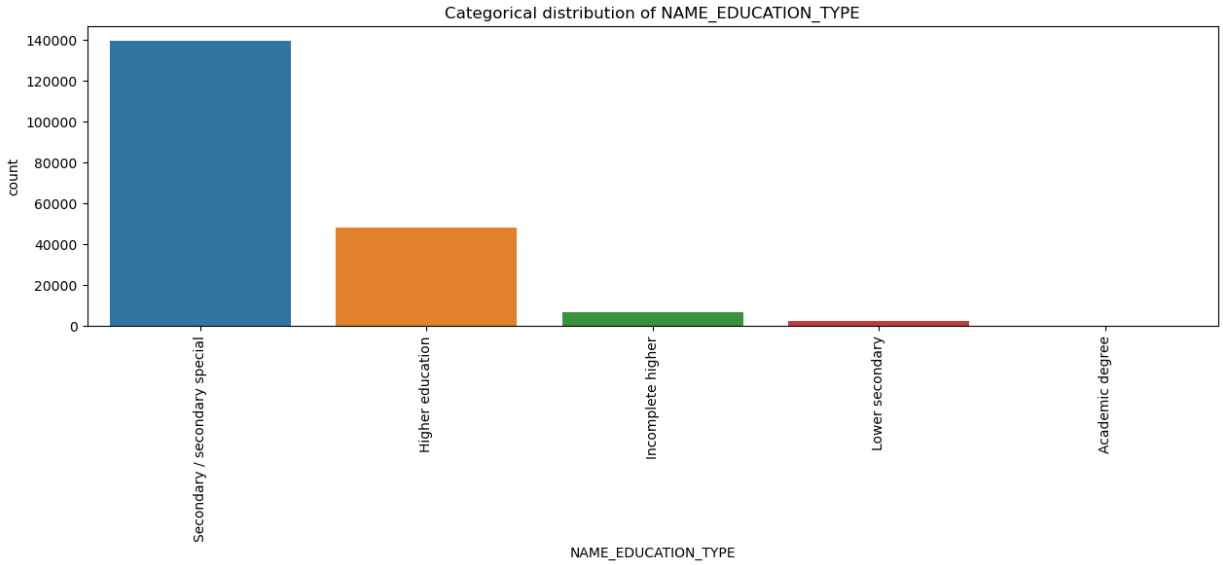


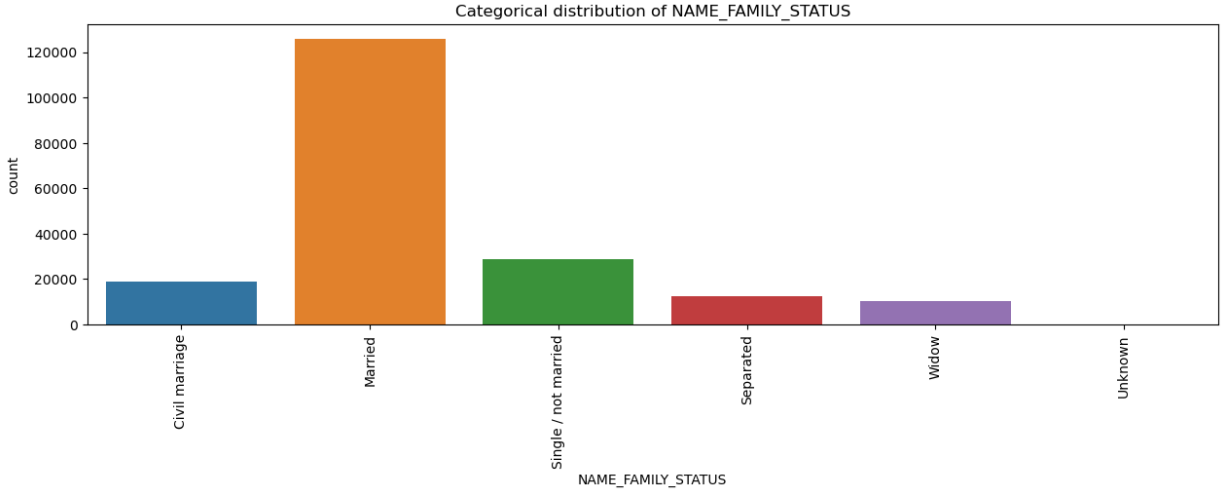


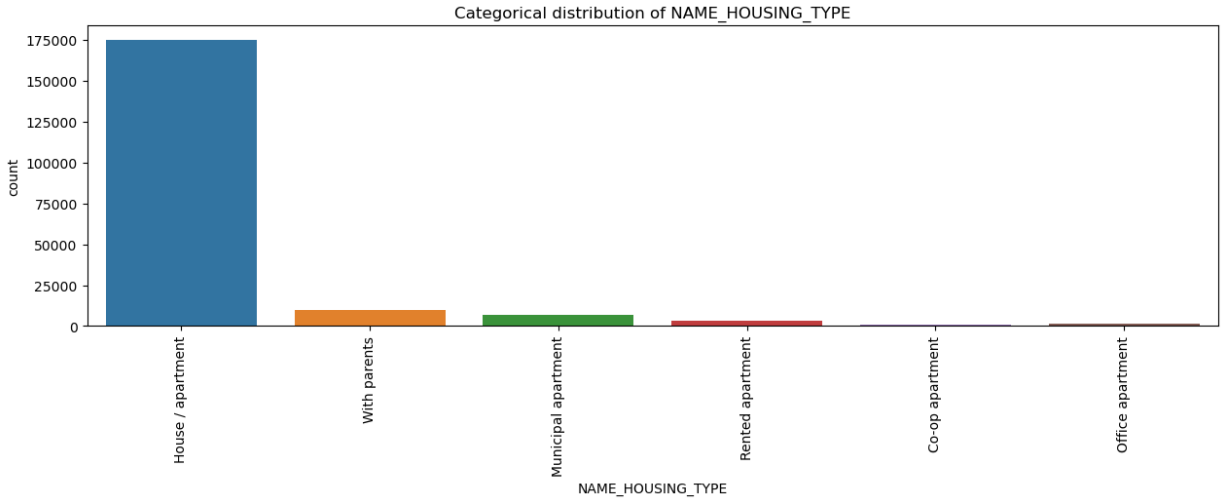

Categorical distribution of FLAG_OWN_REALTY
count
140000
120000
100000
80000
60000
40000
20000
0
Y
N
FLAG_OWN_REALTY

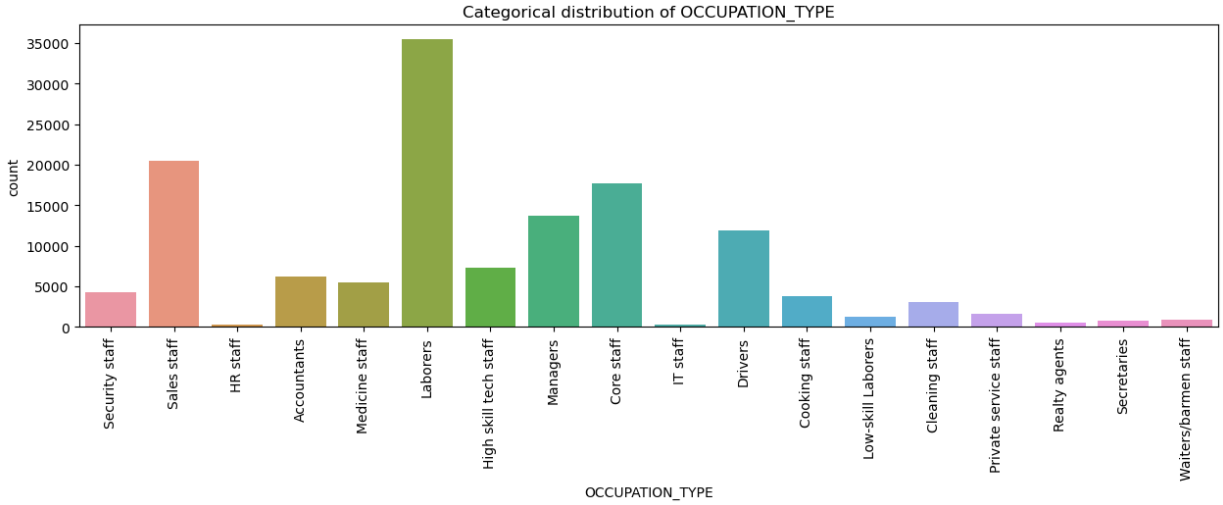

Categorical distribution of NAME_TYPE_SUITE
count
160000
140000
120000
100000
80000
60000
40000
20000
0
Unaccompanied
Family
Children
Spouse, partner
Other_A
Other_B
Group of people
NAME_TYPE_SUITE

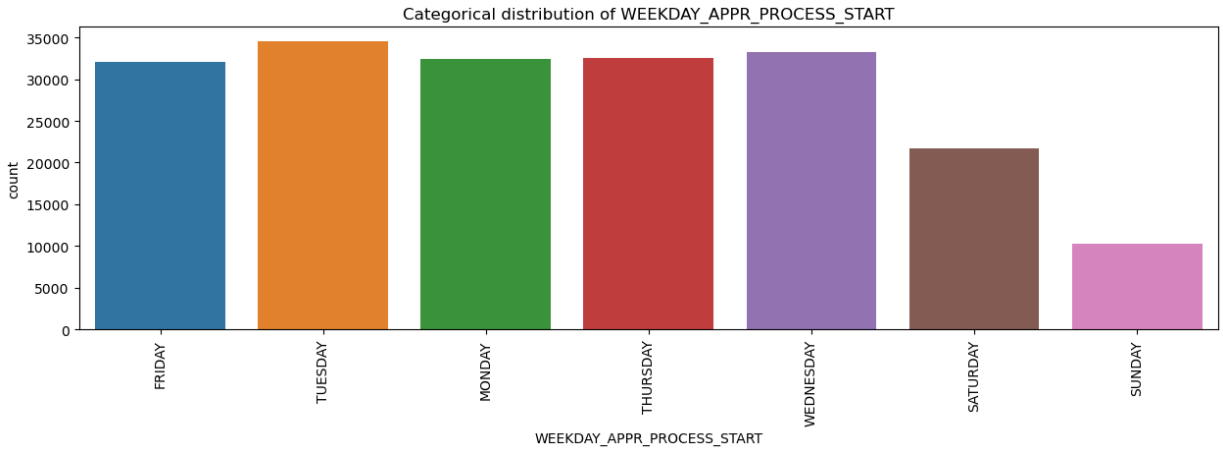

Categorical distribution of NAME_INCOME_TYPE
count
100000
80000
60000
40000
20000
0
Working
Commercial associate
Pensioner
State servant
Maternity leave
Businessman
Student
Unemployed
NAME_INCOME_TYPE

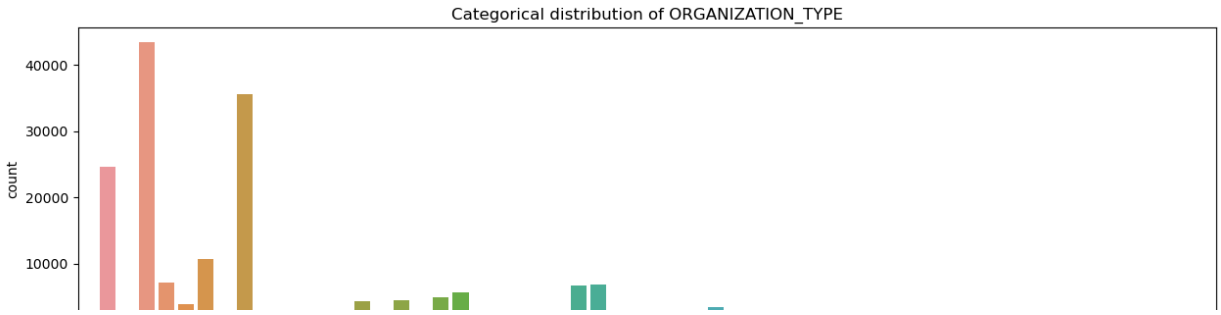

Categorical distribution of NAME_EDUCATION_TYPE
count
140000
120000
100000
80000
60000
40000
20000
0
Secondary / secondary special
Higher education
Incomplete higher
Lower secondary
Academic degree
NAME_EDUCATION_TYPE


Categorical distribution of NAME_FAMILY_STATUS
count
120000
100000
80000
60000
40000
20000
0
Civil marriage
Married
Single / not married
Separated
Widow
Unknown
NAME_FAMILY_STATUS


Categorical distribution of NAME_HOUSING_TYPE
count
175000
150000
125000
100000
75000
50000
25000
0
House / apartment
With parents
Municipal apartment
Rented apartment
Co-op apartment
Office apartment
NAME_HOUSING_TYPE



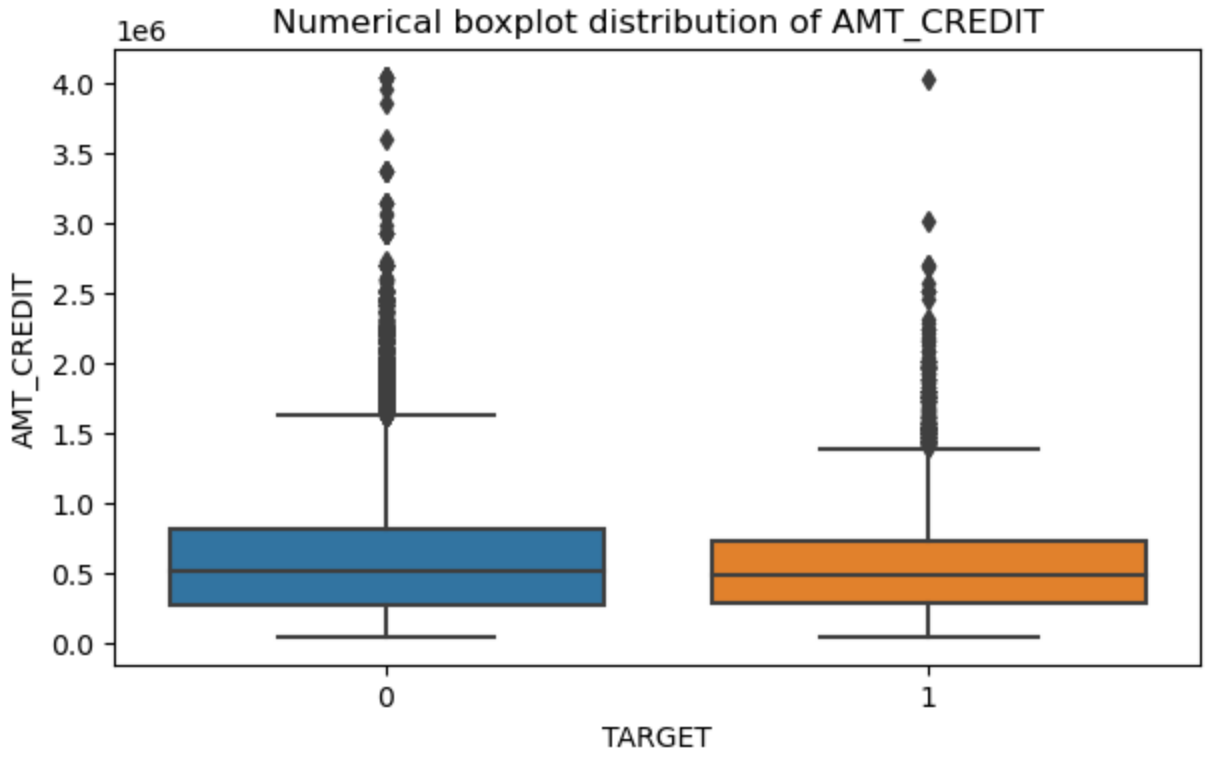


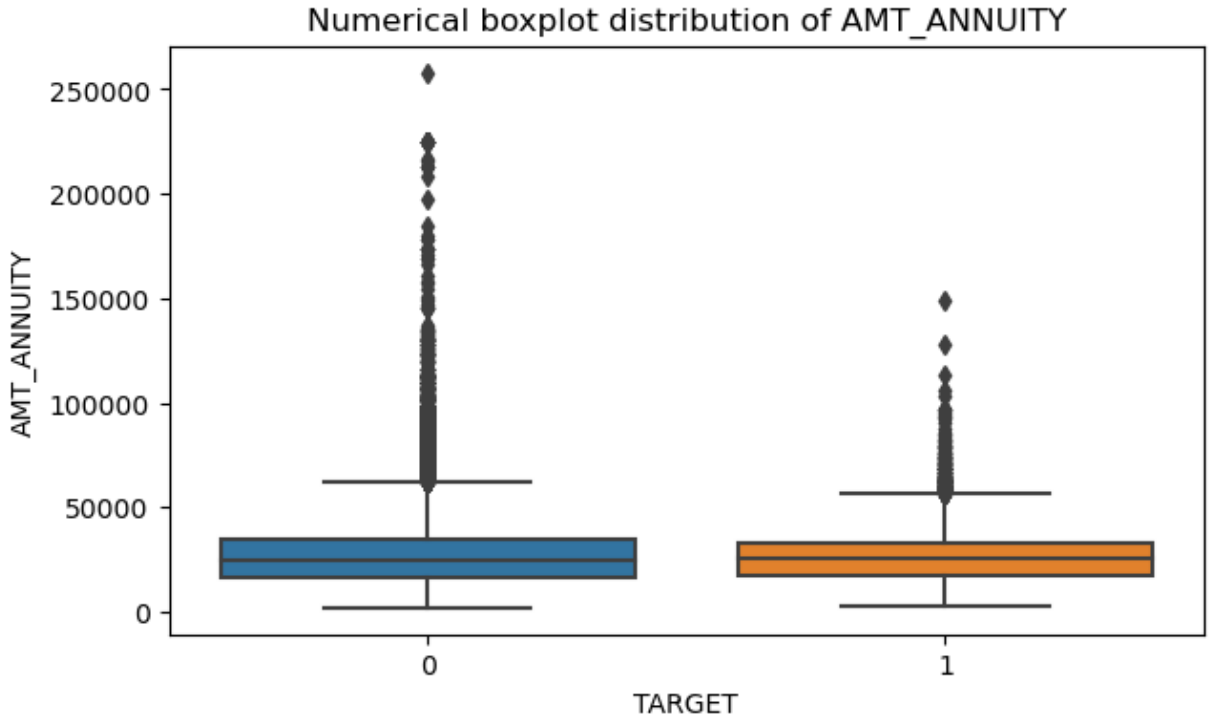


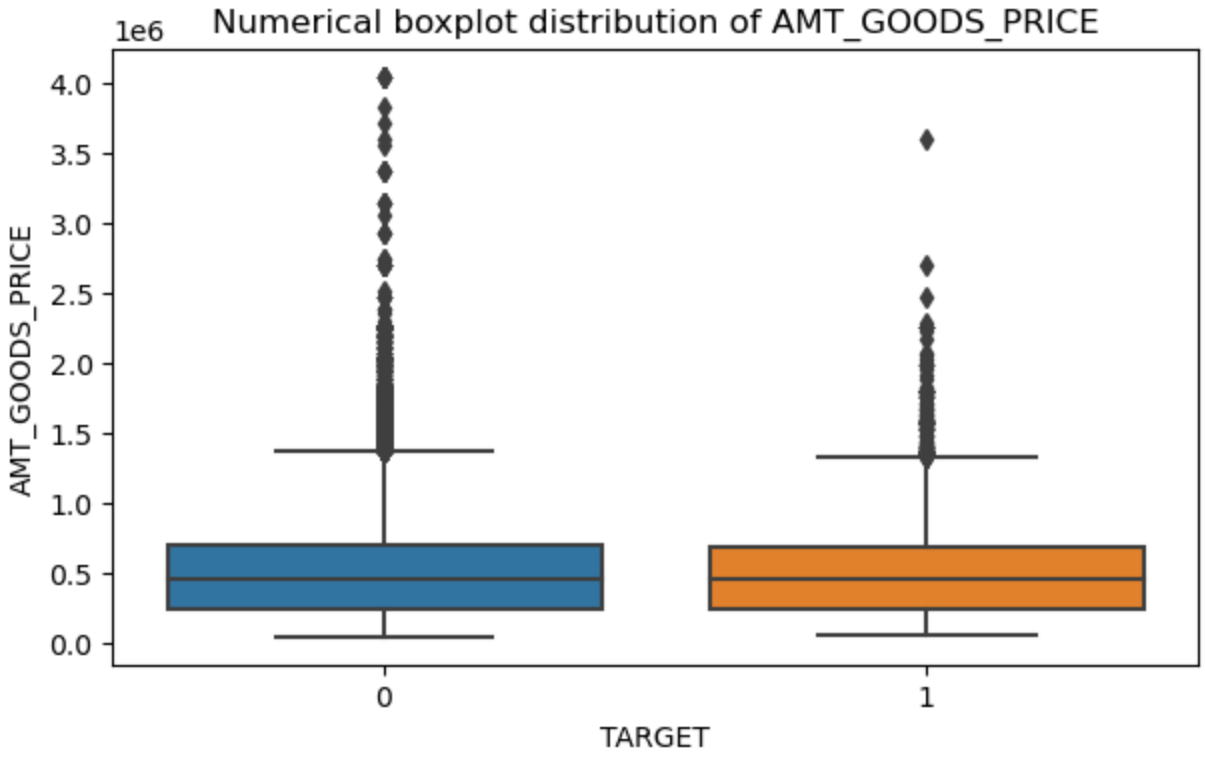
Numerical distribution

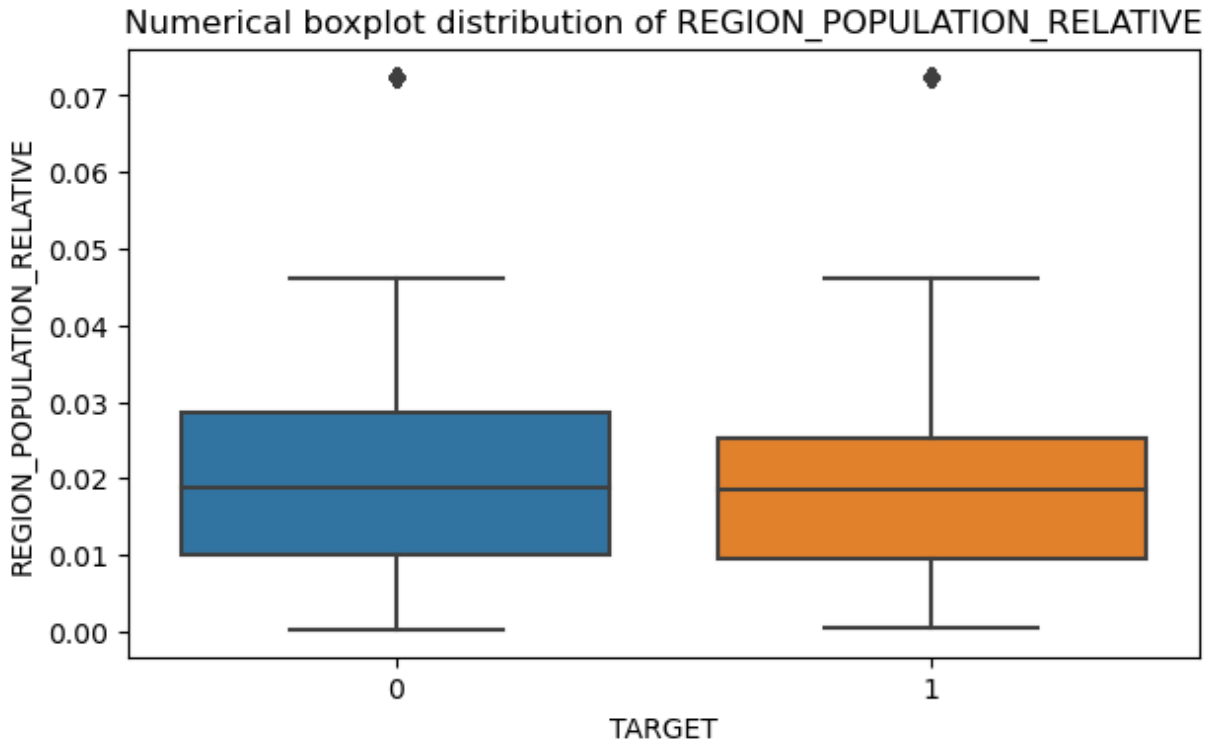
In [150…

```
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()
```


Numerical boxplot distribution of AMT_CREDIT
1e6
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
AMT_CREDIT
0
1
TARGET


Numerical boxplot distribution of AMT_ANNUITY
250000
200000
150000
100000
50000
0
AMT_ANNUITY
0
1
TARGET

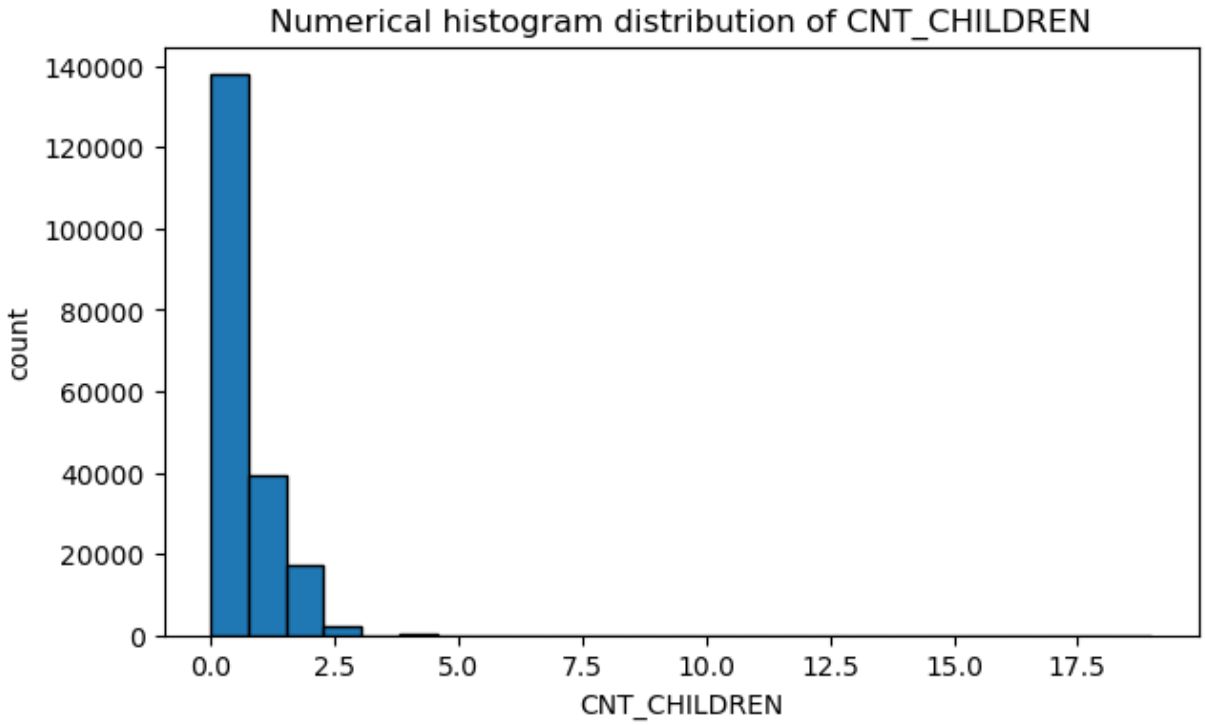

Numerical boxplot distribution of AMT_GOODS_PRICE
1e6
4.0
3.5
3.0
2.5
2.0
1.5
1.0
0.5
0.0
AMT_GOODS_PRICE
0
1
TARGET

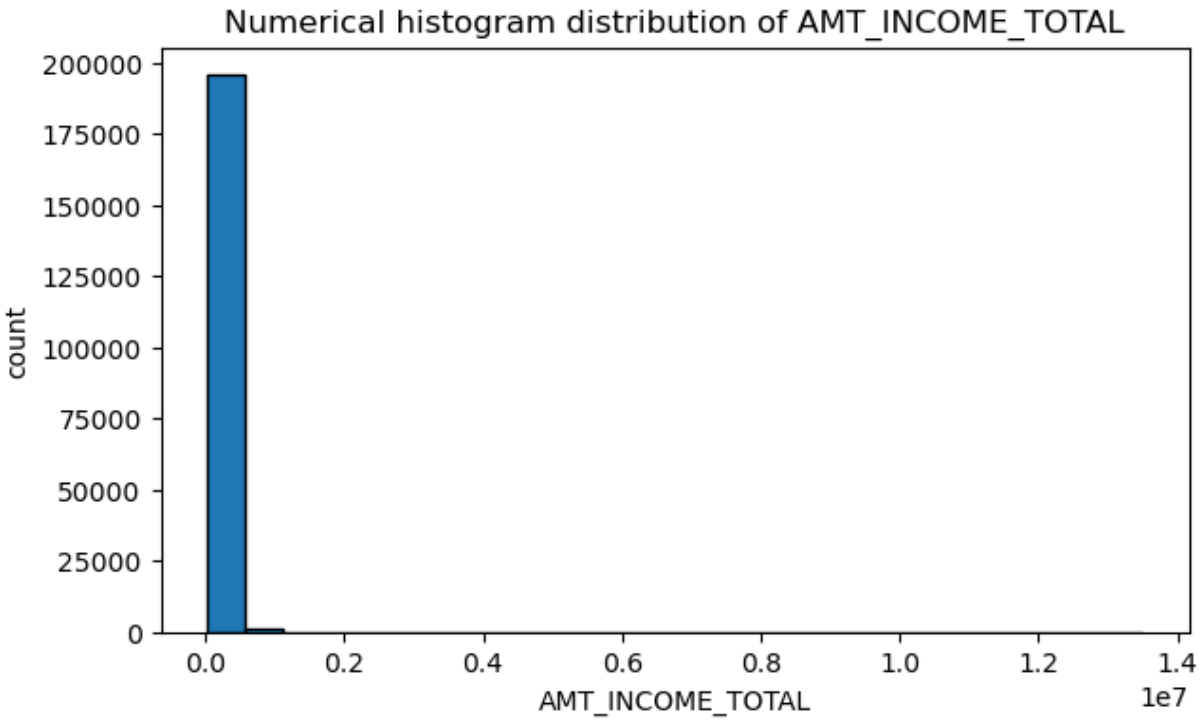

Numerical boxplot distribution of REGION_POPULATION_RELATIVE
0.07
0.06
0.05
0.04
0.03
0.02
0.01
0.00
REGION_POPULATION_RELATIVE
0
1
TARGET

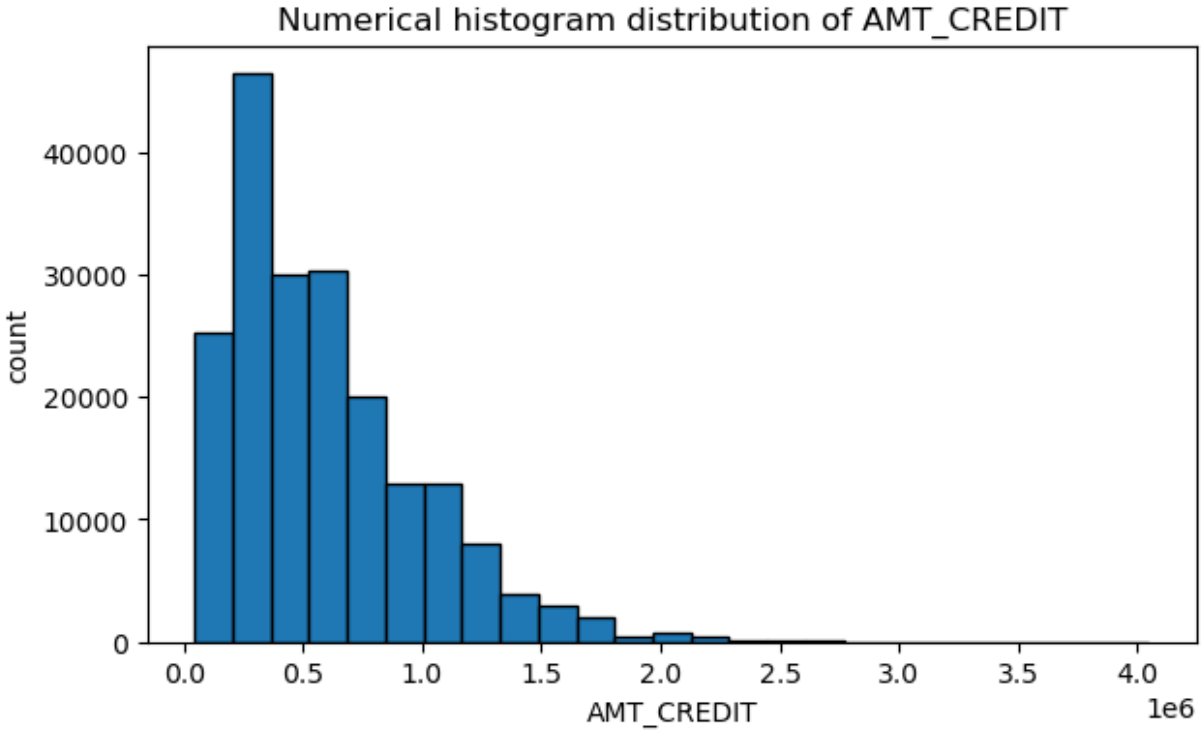
We notice that there are many outliers in the data as seen in the box plot. We can visualize the median and the quantiles of each column data by these box plots.

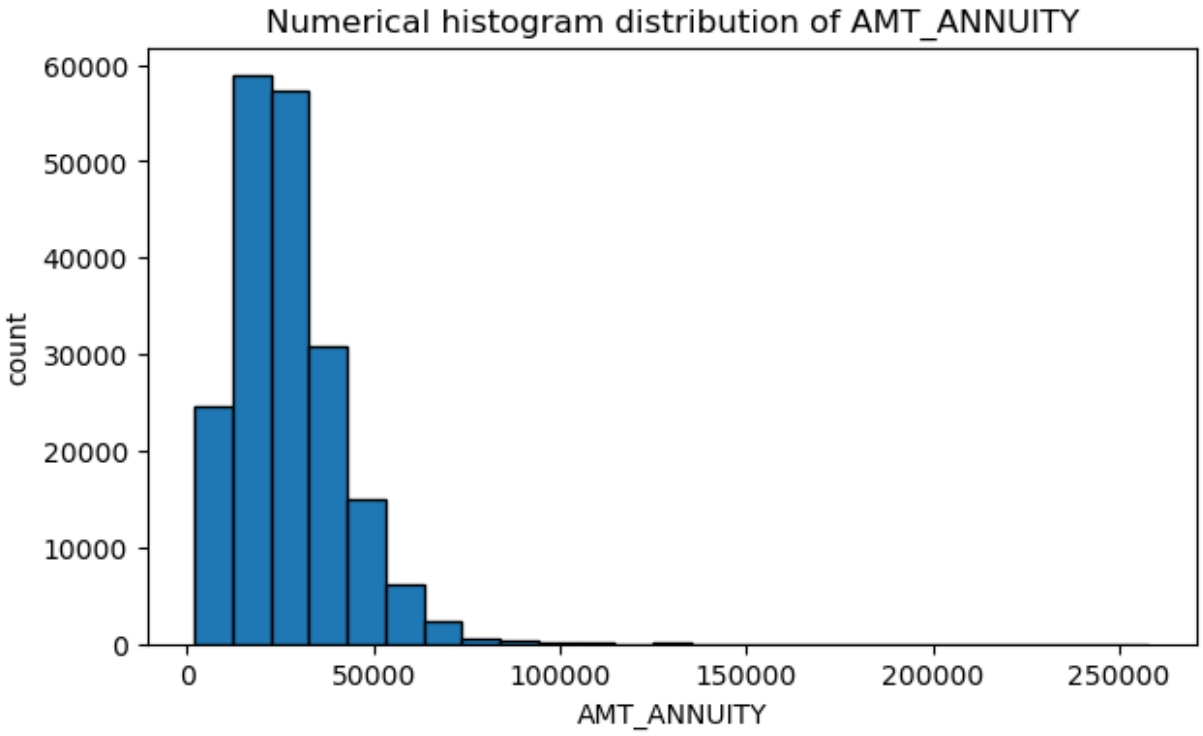
In [151…

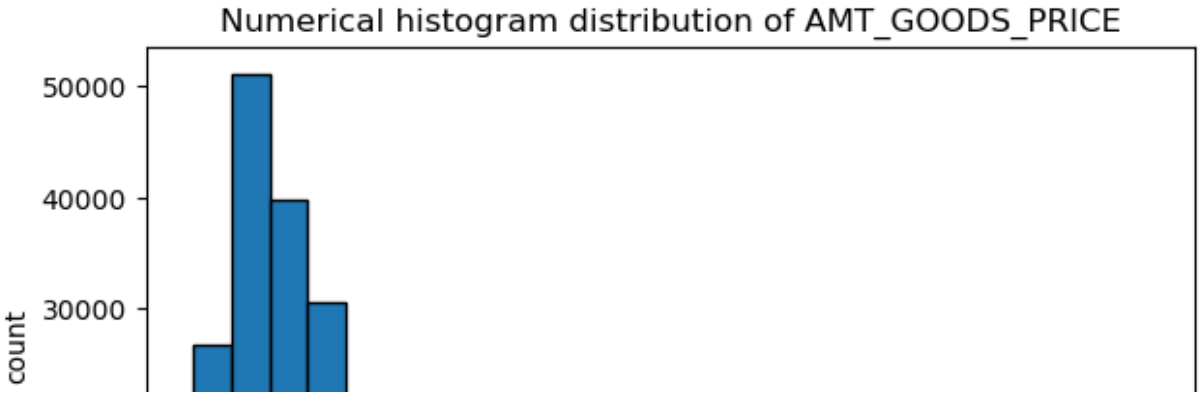
```
for num in list(numerical_features[1:7]):
    plt.figure(figsize=(7, 4))
    plt.hist(Xy_train[num], edgecolor = 'k', bins = 25)
    plt.xlabel(num)
    plt.ylabel('count')
    plt.title(f'Numerical histogram distribution of {num}')
    #plt.setp(plot.get_xticklabels(), rotation=90)
    plt.show()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




Numerical histogram distribution of AMT_INCOME_TOTAL
count
200000
175000
150000
125000
100000
75000
50000
25000
0
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4
AMT_INCOME_TOTAL
1e7


Numerical histogram distribution of AMT_CREDIT
count
40000
30000
20000
10000
0
0.0
0.5
1.0
1.5
2.0
2.5
3.0
3.5
4.0
AMT_CREDIT
1e6



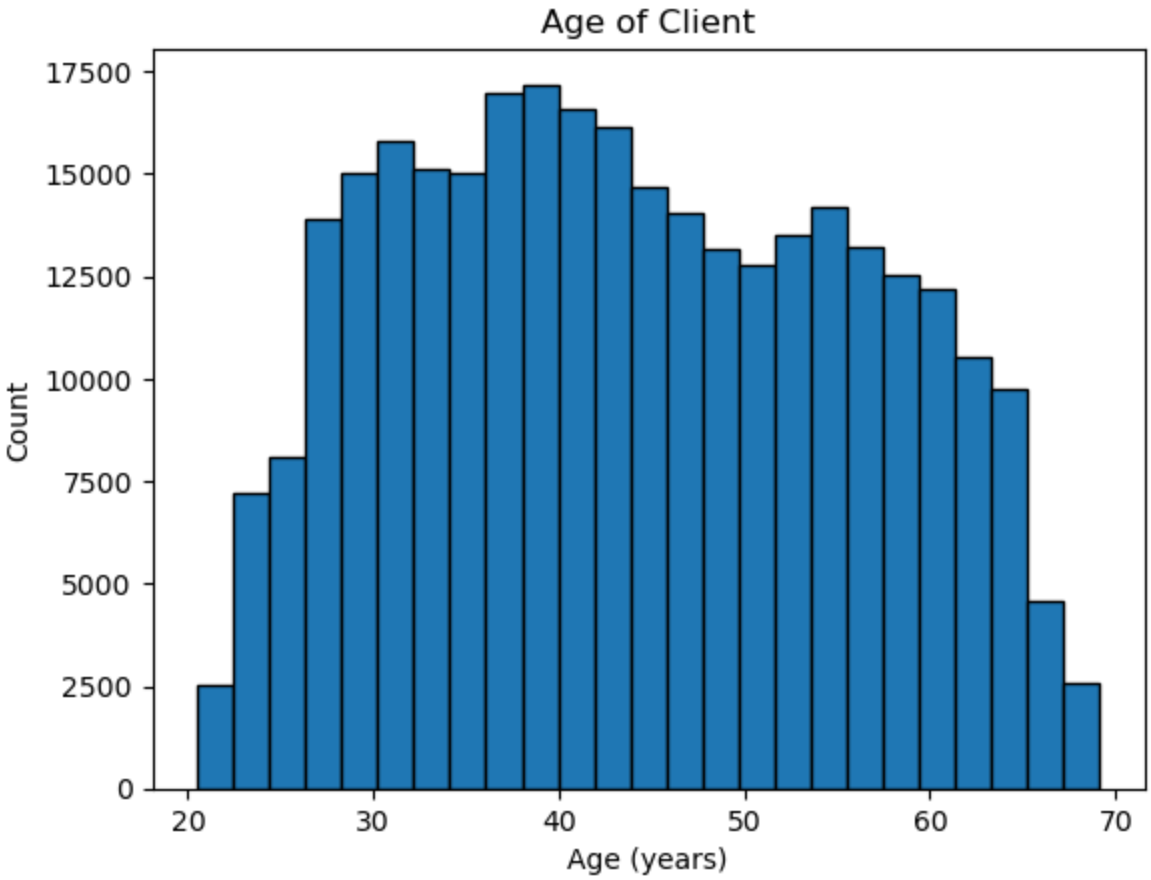


Histogram plot shows the distribution of data over a range. We have visualized each numerical data column's data distribution.

Applicants Age

In [32]:

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

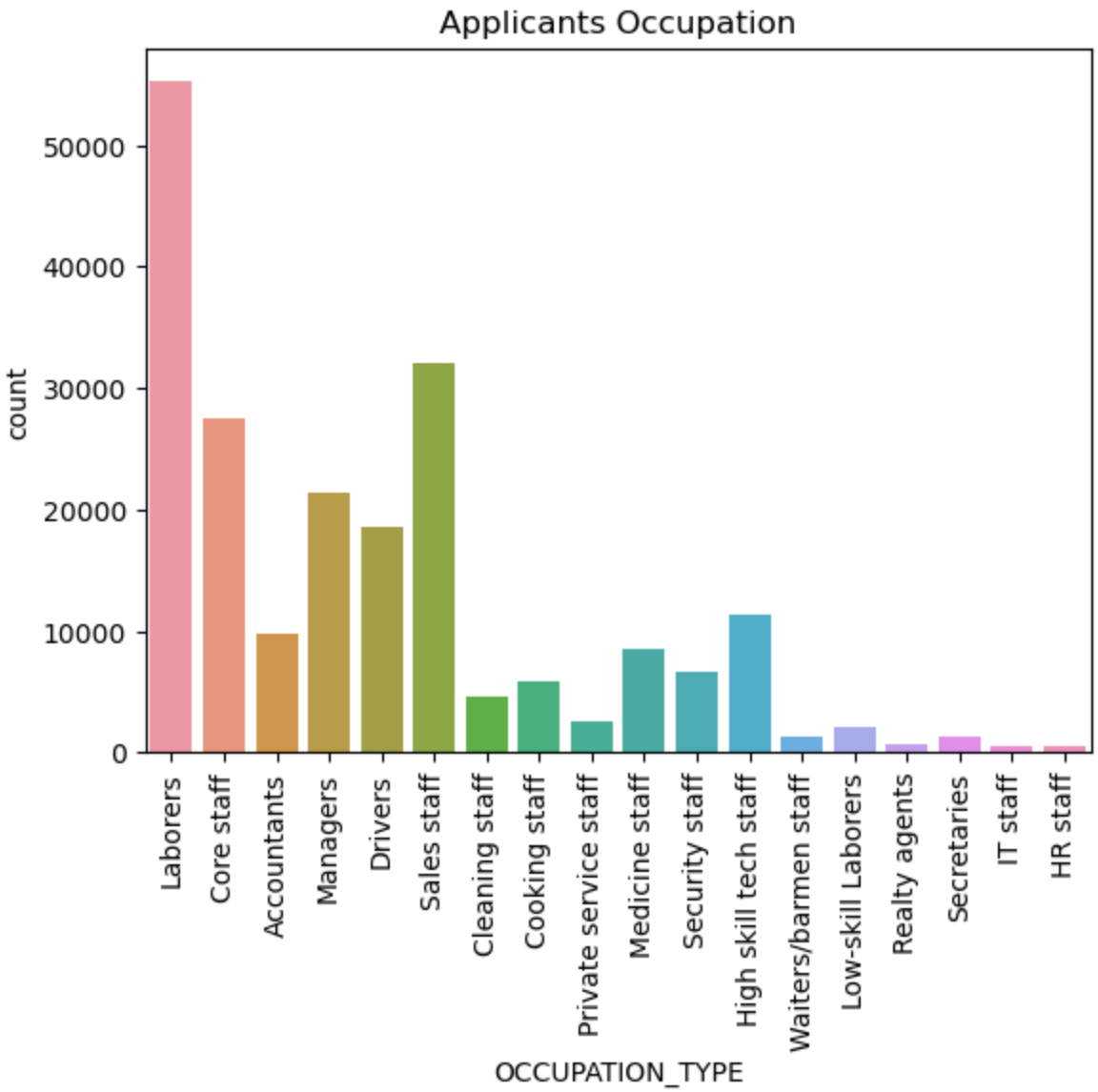


Here we can conclude that people of age 30-50 take more loan applications

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', 'P
OS_CASH_balance'])
```

In [210…
```
len(datasets["application_train"]["SK_ID_CURR"].unique()) == datasets["applic
```

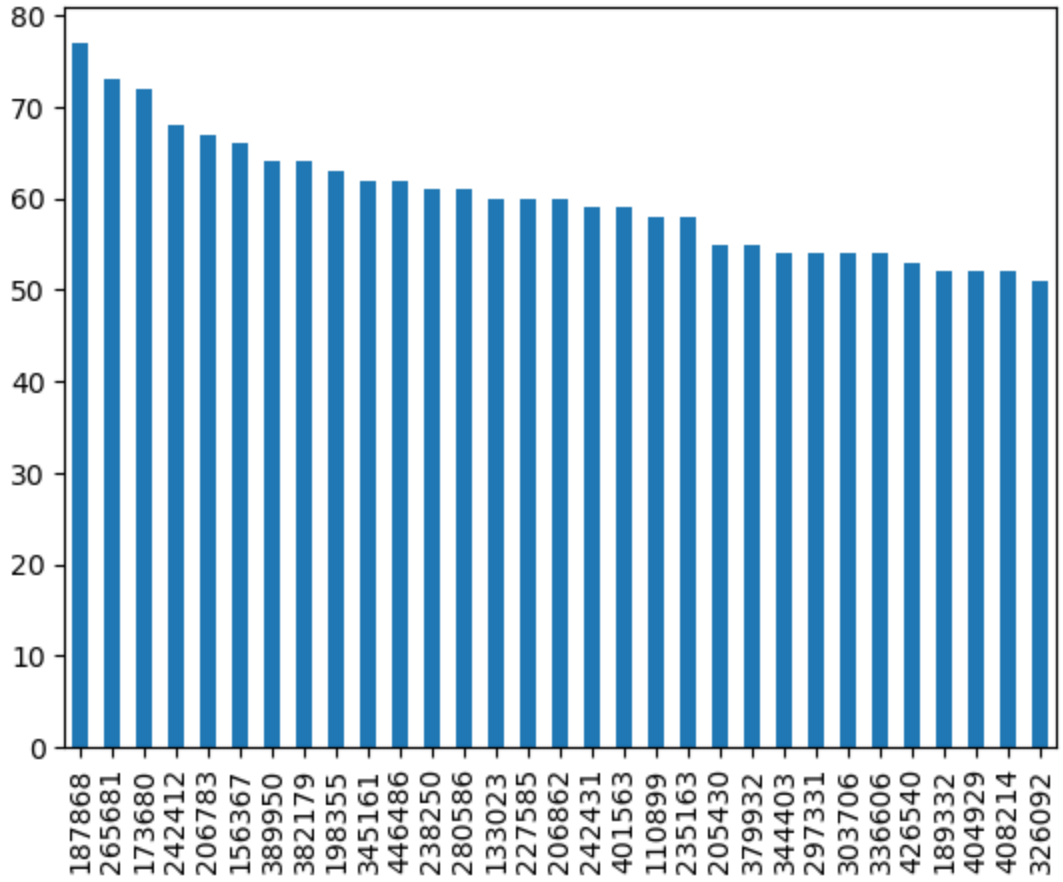
Out[210…
```
True
```

In [211…
```
np.intersect1d(datasets["application_train"]["SK_ID_CURR"], datasets["applica
```
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)
```

previous applications for the submission file

The persons in the kaggle submission file have had previous applications in the `previous_application.csv` . 47,800 out 48,744 people have had previous appications.

In [7]:
```
appsDF = datasets["previous_application"]
```
In [294…
```
len(np.intersect1d(datasets["previous_application"]["SK_ID_CURR"], datasets["
```
Out[294…
```
47800
```
In [295…
```
print(f"There are  {appsDF.shape[0]:,} previous applications")
```
```
There are  1,670,214 previous applications
```
In [8]:
```
# How many entries are there for each month?
prevAppCounts = appsDF['SK_ID_CURR'].value_counts(dropna=False)
```
In [297…
```
len(prevAppCounts[prevAppCounts >40])  #more that 40 previous applications
```
Out[297…
```
101
```
In [299…
```
prevAppCounts[prevAppCounts >50].plot(kind='bar')
plt.xticks(rotation=90)
plt.show()
```



Histogram of Number of previous applications for an ID

In [300…

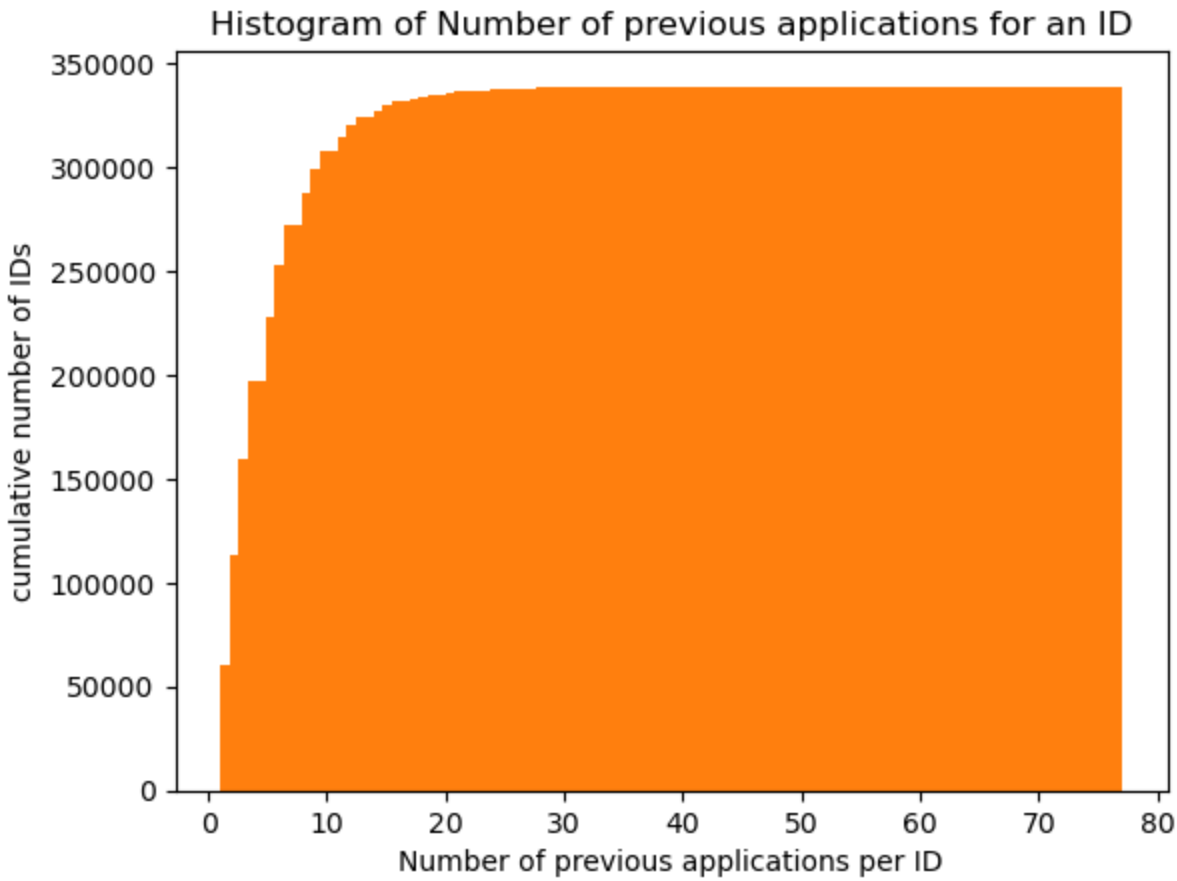
```
sum(appsDF['SK_ID_CURR'].value_counts()==1)
```

Out[300…

```
60458
```

In [302…

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



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

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

In [303…

```
apps_all = appsDF['SK_ID_CURR'].nunique()
apps_5plus = appsDF['SK_ID_CURR'].value_counts()>=5
apps_40plus = appsDF['SK_ID_CURR'].value_counts()>=40
print('Percentage with 10 or more previous apps:', np.round(100.*(sum(apps_5p
print('Percentage with 40 or more previous apps:', np.round(100.*(sum(apps_40
```

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

Input Features -

In [38]:

```
numerical_features = X_train.select_dtypes(include = ['int64', 'float64']).co
categorical_features = X_train.select_dtypes(include = ['object', 'bool']).co
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', 'DAY
S_BIRTH', 'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'DAYS_ID_PUBLISH', 'OWN_CAR_A
GE', '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_REGIO
N', 'REG_REGION_NOT_WORK_REGION', 'LIVE_REGION_NOT_WORK_REGION', 'REG_CITY_NO
T_LIVE_CITY', 'REG_CITY_NOT_WORK_CITY', 'LIVE_CITY_NOT_WORK_CITY', 'EXT_SOURC
E_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', 'LI
VINGAPARTMENTS_AVG', 'LIVINGAREA_AVG', 'NONLIVINGAPARTMENTS_AVG', 'NONLIVINGA
REA_AVG', 'APARTMENTS_MODE', 'BASEMENTAREA_MODE', 'YEARS_BEGINEXPLUATATION_MO
DE', 'YEARS_BUILD_MODE', 'COMMONAREA_MODE', 'ELEVATORS_MODE', 'ENTRANCES_MODE
', 'FLOORSMAX_MODE', 'FLOORSMIN_MODE', 'LANDAREA_MODE', 'LIVINGAPARTMENTS_MOD
E', 'LIVINGAREA_MODE', 'NONLIVINGAPARTMENTS_MODE', 'NONLIVINGAREA_MODE', 'APA
RTMENTS_MEDI', 'BASEMENTAREA_MEDI', 'YEARS_BEGINEXPLUATATION_MEDI', 'YEARS_BU
ILD_MEDI', 'COMMONAREA_MEDI', 'ELEVATORS_MEDI', 'ENTRANCES_MEDI', 'FLOORSMAX_
MEDI', 'FLOORSMIN_MEDI', 'LANDAREA_MEDI', 'LIVINGAPARTMENTS_MEDI', 'LIVINGARE
A_MEDI', 'NONLIVINGAPARTMENTS_MEDI', 'NONLIVINGAREA_MEDI', 'TOTALAREA_MODE',
'OBS_30_CNT_SOCIAL_CIRCLE', 'DEF_30_CNT_SOCIAL_CIRCLE', 'OBS_60_CNT_SOCIAL_CI
RCLE', '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_DOCUM
ENT_18', 'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20', 'FLAG_DOCUMENT_21', 'AMT_REQ
_CREDIT_BUREAU_HOUR', 'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEE
K', '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_TYP
E', 'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'OCCUPATION_TYPE', 'WEEKDAY_AP
PR_PROCESS_START', 'ORGANIZATION_TYPE', 'FONDKAPREMONT_MODE', 'HOUSETYPE_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_f
total_input_features
```

Out[158…
```
121
```

Modeling

Baseline Logistic Regression -

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

$$ \underset{\mathbf{\theta}}{\operatorname{argmin}}\left[\text{CXE}\right] = \underset{\mathbf{\theta}}{\operatorname{argmin}} \left[-\dfrac{1}{m} \sum\limits_{i=1}^{m}{\left[y^{(i)} log\left(\hat{p}^{(i)}\right) + (1 - y^{(i)}) log\left(1 - \hat{p}^{(i)}\right)\right]} \right] $$
The corresponding gradient function of partial derivatives is as follows (after a little bit of math):

$$ \begin{aligned} \nabla_\text{CXE}(\mathbf{\theta}) &= \begin{pmatrix} \frac{\partial} {\partial \theta_0} \text{CXE}(\mathbf{\theta}) \\ \frac{\partial}{\partial \theta_1} \text{CXE} (\mathbf{\theta}) \\ \vdots \\ \frac{\partial}{\partial \theta_n} \text{CXE}(\mathbf{\theta}) \end{pmatrix}\\ &= \dfrac{2}{m} \mathbf{X}^T \cdot (\hat{p}_y - \mathbf{y}) \end{aligned} $$
For completeness learning a binomial logistic regression model via gradient descent would use the following step iteratively:

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

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", "ValidAc
```

In [198…

```
def model_logreg(X_train):

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

    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=colum

    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"{tra
                                 f"{validAcc*100:8.2f}%", f"{testAcc*100:8.2f

    display(results)
    return clf_pipe
```

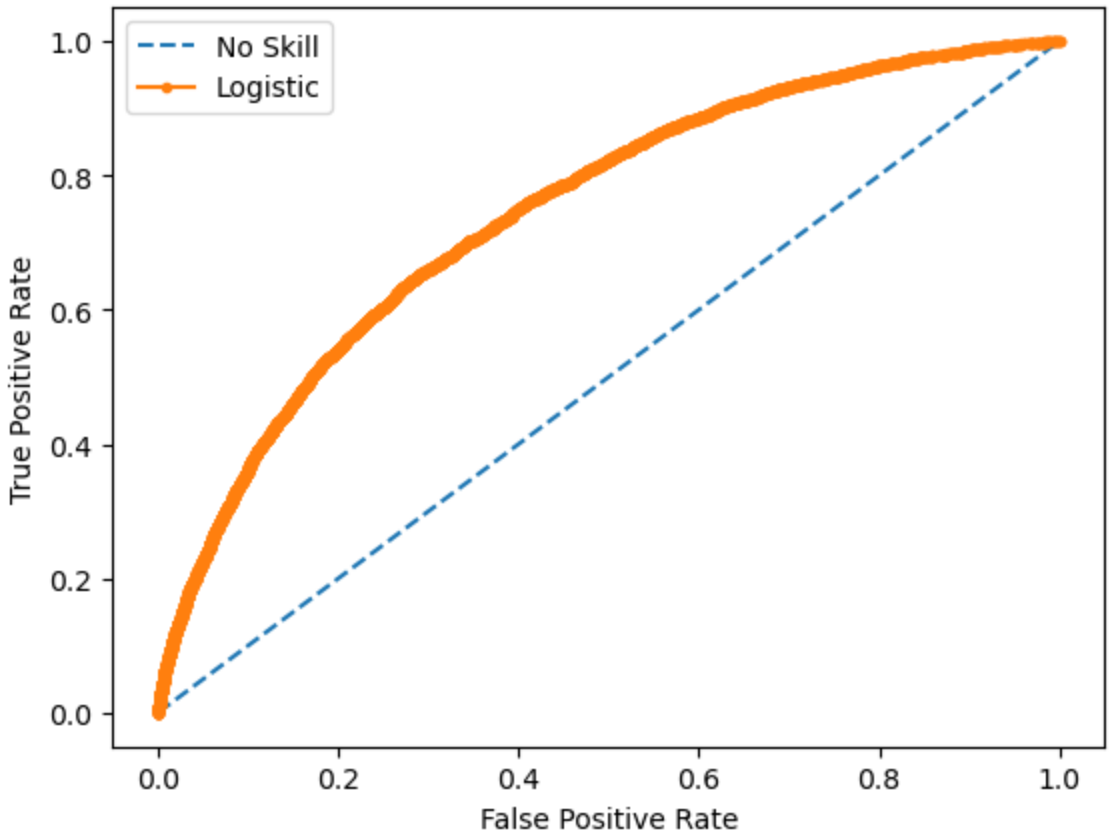
In [199…

```
logreg = model_logreg(X_train)
```

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GO
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 : sum(y — prediction)²

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

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

DecisionTreeClassifier

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

Gini : $\Large 1 - \sum^m_{i=1}(P_j^2)$

Entropy : $\Large \sum^m_{i=1}\left(P_j\cdot\:\log\:\left(P_j\right)\:\right)$

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):

$\Large ni_j = w_jC_j - w_{left(j)}C_{left(j)} - w_{right(j)}C_{right(j)}$

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:

$\Large fi_i = \frac{\sum_{j:node \hspace{0.1cm} j \hspace{0.1cm} splits \hspace{0.1cm} on \hspace{0.1cm} feature \hspace{0.1cm} i}ni_j}{\sum_{k \hspace{0.1cm} \epsilon

\hspace{0.1cm} all \hspace{0.1cm} 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:

$\Large normfi_i = \frac{fi_i}{\sum_{j \hspace{0.1cm} \epsilon \hspace{0.1cm} all \hspace{0.1cm} features}fi_j}$

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%20im

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

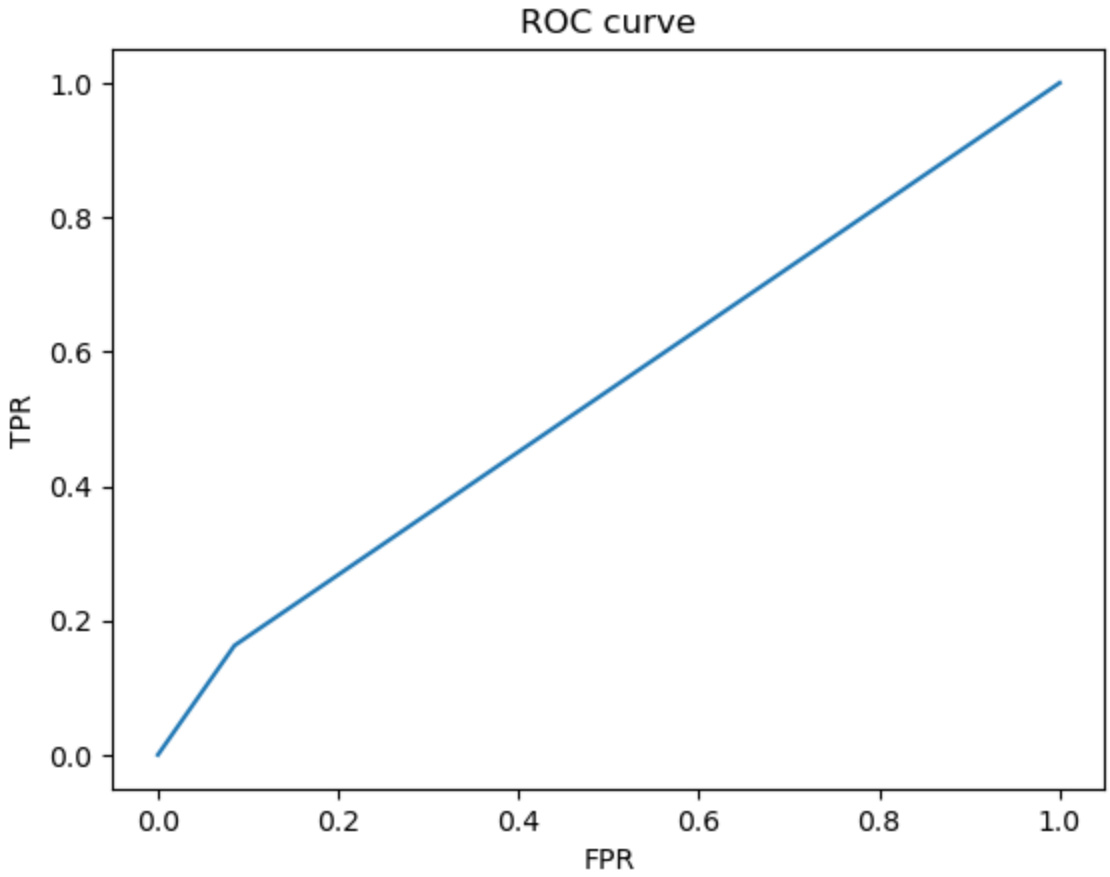
In [200…

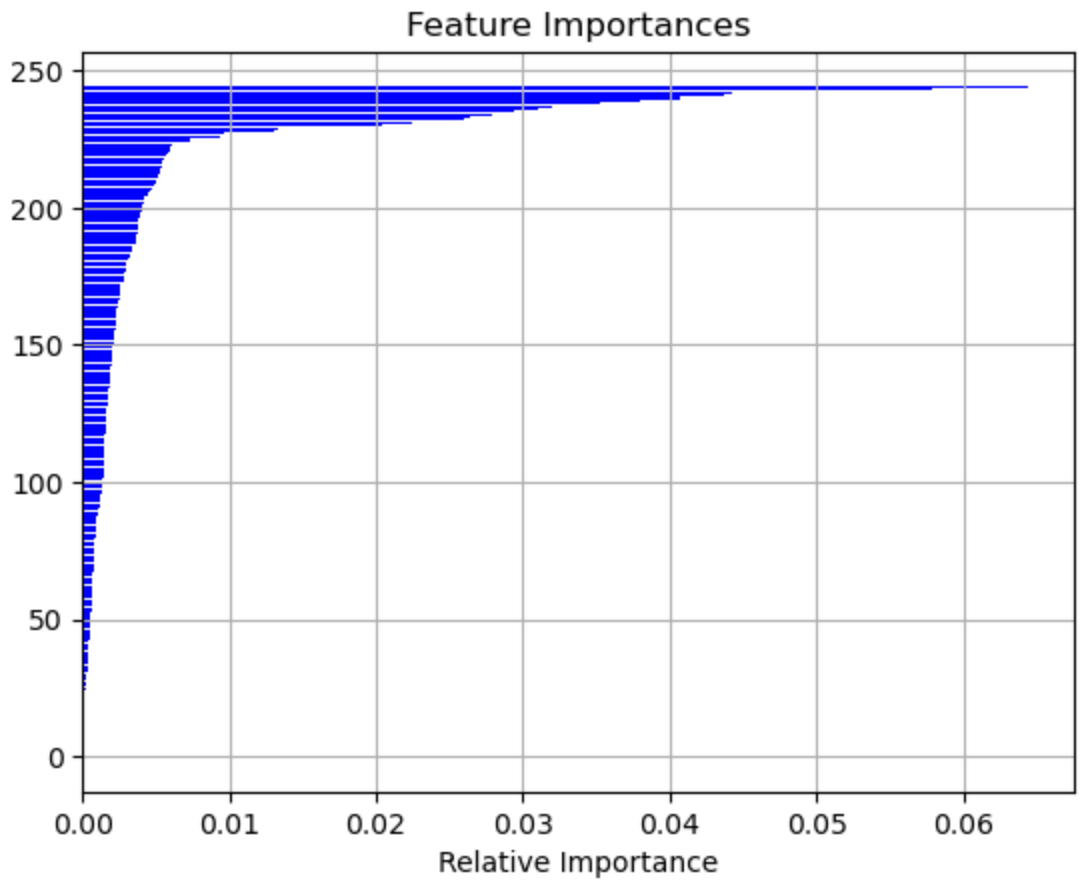
```
def model_dt(X_train):

    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='cen
    #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
                                 f"{validAcc*100:8.2f}%", f"{testAcc*100:8.2f

    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

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.

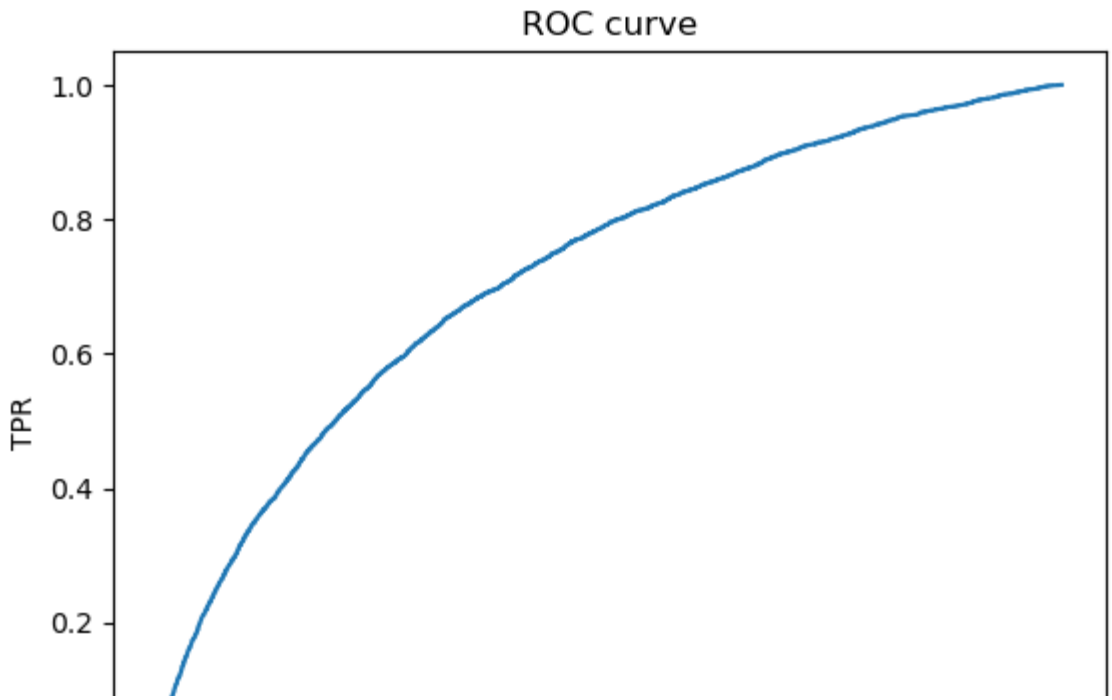
In [202…

```
def model_rf(X_train):

    RF = RandomForestClassifier(random_state = 42,n_estimators=20, criterion=

    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
                                 f"{validAcc*100:8.2f}%", f"{testAcc*100:8.2f

    display(results)

    return data_pipeline_rf,RF
```

In [203…

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

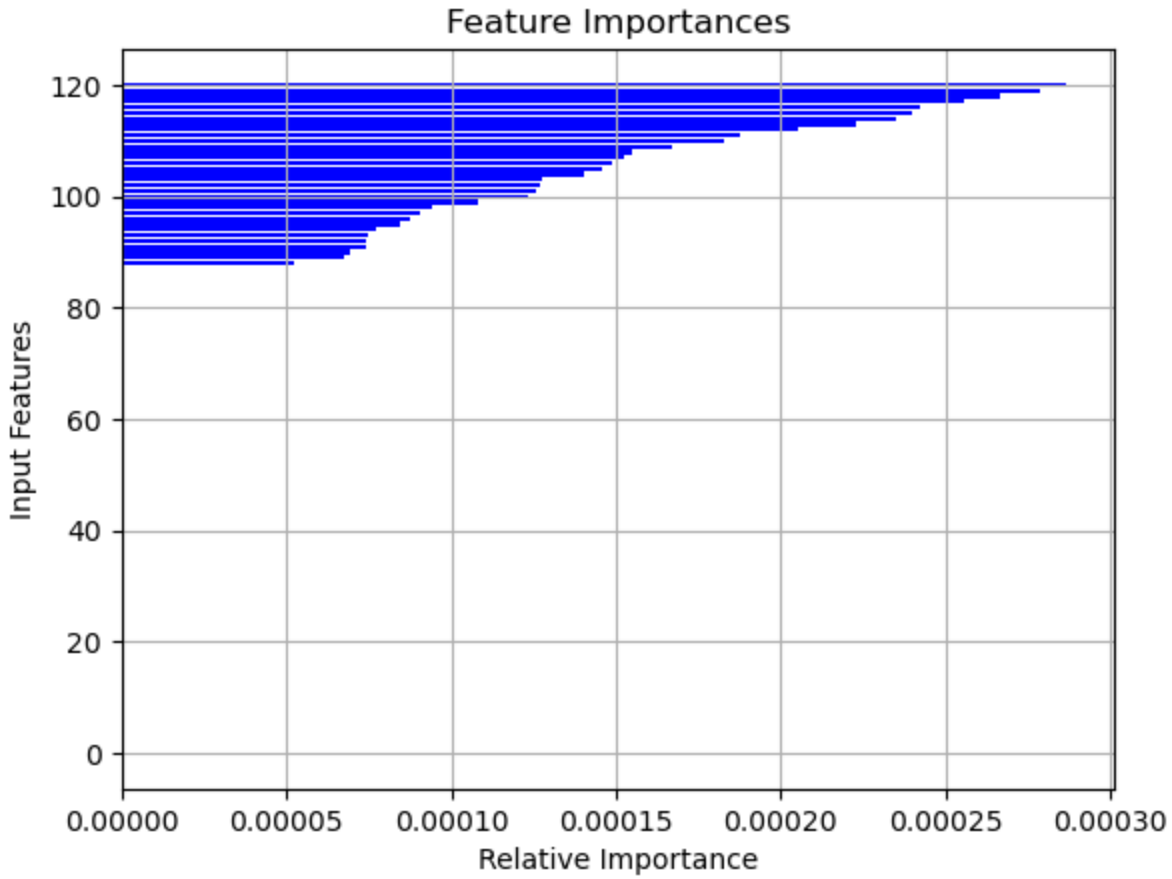
```
Score 0.7202735379743133
```



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 for prevApp table

In [38]:

```
features = ['AMT_ANNUITY', 'AMT_APPLICATION']
agg_op_features = {}
agg_op_features = {
    'AMT_ANNUITY': ['min', 'max', 'mean'],
    'AMT_APPLICATION': ['min', 'max', 'mean']}
print(agg_op_features)

print(f"{appsDF[features].describe()}")
prev = appsDF.groupby(["SK_ID_CURR"]).agg(agg_op_features)
prev
prev.columns = prev.columns.droplevel() #drop 1 of the header row but keep th
result = prev.reset_index(level=["SK_ID_CURR"])
result
# result['range_AMT_APPLICATION'] = result['AMT_APPLICATION_max'] - result['A
# print(f"result.shape: {result.shape}")
# result[0:10]
```

```
{'AMT_ANNUITY': ['min', 'max', 'mean'], 'AMT_APPLICATION': ['min', 'max', 'me
an']}
        AMT_ANNUITY  AMT_APPLICATION
count  1.297979e+06     1.670214e+06
mean   1.595512e+04     1.752339e+05
std    1.478214e+04     2.927798e+05
min    0.000000e+00     0.000000e+00
25%    6.321780e+03     1.872000e+04
50%    1.125000e+04     7.104600e+04
```

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

Out[38]:

	SK_ID_CURR	min	max	mean	min	max	mean
0	100001	3951.000	3951.000	3951.000000	24835.5	24835.5	24835.500
1	100002	9251.775	9251.775	9251.775000	179055.0	179055.0	179055.000
2	100003	6737.310	98356.995	56553.990000	68809.5	900000.0	435436.500
3	100004	5357.250	5357.250	5357.250000	24282.0	24282.0	24282.000
4	100005	4813.200	4813.200	4813.200000	0.0	44617.5	22308.750
...	...	...	...	...	...	...	...
338852	456251	6605.910	6605.910	6605.910000	40455.0	40455.0	40455.000
338853	456252	10074.465	10074.465	10074.465000	57595.5	57595.5	57595.500
338854	456253	3973.095	5567.715	4770.405000	19413.0	28912.5	24162.750
338855	456254	2296.440	19065.825	10681.132500	18846.0	223789.5	121317.750
338856	456255	2250.000	54022.140	20775.391875	45000.0	1170000.0	362770.875

338857 rows × 7 columns

In [39]:
```
agg_op_features
```

Out[39]:
```
{'AMT_ANNUITY': ['min', 'max', 'mean'],
 'AMT_APPLICATION': ['min', 'max', 'mean']}
```

In [40]:
```
result.isna().sum()
```

Out[40]:
```
SK_ID_CURR      0
min           480
max           480
mean          480
min             0
max             0
mean            0
dtype: int64
```

feature transformer for prevApp table

In [55]:

```
# Create aggregate features (via pipeline)
class prevAppsFeaturesAggregater(BaseEstimator, TransformerMixin):
    def __init__(self, features=None): # no *args or **kargs
        self.features = features
        self.agg_op_features = {}

        self.agg_op_features = {
    'AMT_ANNUITY': ['min', 'max', 'mean'],
    'AMT_APPLICATION': ['min', 'max', 'mean']}

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

    def transform(self, X, y=None):
        #from IPython.core.debugger import Pdb as pdb;    pdb().set_trace() #
        result = X.groupby(["SK_ID_CURR"]).agg(self.agg_op_features)
        result.columns = result.columns.droplevel()
        result = result.reset_index(level=["SK_ID_CURR"])
        #result['range_AMT_APPLICATION'] = result['AMT_APPLICATION_max'] - re
        return result # return dataframe with the join key "SK_ID_CURR"


from sklearn.pipeline import make_pipeline
def test_driver_prevAppsFeaturesAggregater(df, features):
    print(f"df.shape: {df.shape}\n")
    print(f"df[{features}][0:5]: \n{df[features][0:5]}")
    test_pipeline = make_pipeline(prevAppsFeaturesAggregater(features))
    return(test_pipeline.fit_transform(df))

features = ['AMT_ANNUITY', 'AMT_APPLICATION']
features = ['AMT_ANNUITY',
       'AMT_APPLICATION', 'AMT_CREDIT', 'AMT_DOWN_PAYMENT', 'AMT_GOODS_PRICE'
       'RATE_DOWN_PAYMENT', 'RATE_INTEREST_PRIMARY',
       'RATE_INTEREST_PRIVILEGED', 'DAYS_DECISION', 'NAME_PAYMENT_TYPE',
       'CNT_PAYMENT',
       'DAYS_FIRST_DRAWING', 'DAYS_FIRST_DUE', 'DAYS_LAST_DUE_1ST_VERSION',
       'DAYS_LAST_DUE', 'DAYS_TERMINATION']
features = ['AMT_ANNUITY', 'AMT_APPLICATION']
res = test_driver_prevAppsFeaturesAggregater(appsDF, features)
print(f"HELLO")
print(f"Test driver: \n{res[0:10]}")
print(f"input[features][0:10]: \n{appsDF[0:10]}")


# QUESTION, should we lower case df['OCCUPATION_TYPE'] as Sales staff != 'Sal
```

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

df[['AMT_ANNUITY', 'AMT_APPLICATION']][0:5]:
   AMT_ANNUITY  AMT_APPLICATION
0     1730.430          17145.0
1    25188.615         607500.0
2    15060.735         112500.0
3    47041.335         450000.0
4    31924.395         337500.0
HELLO
Test driver:
```

```
   SK_ID_CURR        min        max          mean       min       max  \
0      100001   3951.000   3951.000   3951.000000   24835.5   24835.5
1      100002   9251.775   9251.775   9251.775000  179055.0  179055.0
2      100003   6737.310  98356.995  56553.990000   68809.5  900000.0
3      100004   5357.250   5357.250   5357.250000   24282.0   24282.0
4      100005   4813.200   4813.200   4813.200000       0.0   44617.5
5      100006   2482.920  39954.510  23651.175000       0.0  688500.0
6      100007   1834.290  22678.785  12278.805000   17176.5  247500.0
7      100008   8019.090  25309.575  15839.696250       0.0  450000.0
8      100009   7435.845  17341.605  10051.412143   40455.0  110160.0
9      100010  27463.410  27463.410  27463.410000  247212.0  247212.0

           mean
0   24835.500000
1  179055.000000
2  435436.500000
3   24282.000000
4   22308.750000
5  272203.260000
6  150530.250000
7  155701.800000
8   76741.714286
9  247212.000000
input[features][0:10]:
   SK_ID_PREV  SK_ID_CURR NAME_CONTRACT_TYPE  AMT_ANNUITY  AMT_APPLICATION  \
0     2030495      271877     Consumer loans     1730.430          17145.0
1     2802425      108129         Cash loans    25188.615         607500.0
2     2523466      122040         Cash loans    15060.735         112500.0
3     2819243      176158         Cash loans    47041.335         450000.0
4     1784265      202054         Cash loans    31924.395         337500.0
5     1383531      199383         Cash loans    23703.930         315000.0
6     2315218      175704         Cash loans          NaN              0.0
7     1656711      296299         Cash loans          NaN              0.0
8     2367563      342292         Cash loans          NaN              0.0
9     2579447      334349         Cash loans          NaN              0.0

   AMT_CREDIT  AMT_DOWN_PAYMENT  AMT_GOODS_PRICE WEEKDAY_APPR_PROCESS_START
\
0     17145.0               0.0          17145.0                  SATURDAY
1    679671.0               NaN         607500.0                  THURSDAY
2    136444.5               NaN         112500.0                   TUESDAY
3    470790.0               NaN         450000.0                    MONDAY
4    404055.0               NaN         337500.0                  THURSDAY
5    340573.5               NaN         315000.0                  SATURDAY
6         0.0               NaN              NaN                   TUESDAY
7         0.0               NaN              NaN                    MONDAY
8         0.0               NaN              NaN                    MONDAY
9         0.0               NaN              NaN                  SATURDAY

   HOUR_APPR_PROCESS_START  ... NAME_SELLER_INDUSTRY  CNT_PAYMENT  \
0                       15  ...         Connectivity         12.0
1                       11  ...                  XNA         36.0
2                       11  ...                  XNA         12.0
3                        7  ...                  XNA         12.0
4                        9  ...                  XNA         24.0
5                        8  ...                  XNA         18.0
6                       11  ...                  XNA          NaN
7                        7  ...                  XNA          NaN
8                       15  ...                  XNA          NaN
```

```
9                      15  ...                  XNA          NaN

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

  DAYS_FIRST_DUE DAYS_LAST_DUE_1ST_VERSION  DAYS_LAST_DUE DAYS_TERMINATION  \
0          -42.0                    300.0          -42.0            -37.0
1         -134.0                    916.0       365243.0         365243.0
2         -271.0                     59.0       365243.0         365243.0
3         -482.0                   -152.0         -182.0           -177.0
4            NaN                      NaN            NaN              NaN
5         -654.0                   -144.0         -144.0           -137.0
6            NaN                      NaN            NaN              NaN
7            NaN                      NaN            NaN              NaN
8            NaN                      NaN            NaN              NaN
9            NaN                      NaN            NaN              NaN

  NFLAG_INSURED_ON_APPROVAL
0                       0.0
1                       1.0
2                       1.0
3                       1.0
4                       NaN
5                       1.0
6                       NaN
7                       NaN
8                       NaN
9                       NaN
```

In [56]:
```
appsDF
```

Out[56]:

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATI
0	2030495	271877	Consumer loans	1730.430	1714
1	2802425	108129	Cash loans	25188.615	60750
2	2523466	122040	Cash loans	15060.735	11250
3	2819243	176158	Cash loans	47041.335	45000
4	1784265	202054	Cash loans	31924.395	33750
...	...	...	...	...	
1670209	2300464	352015	Consumer loans	14704.290	26729
1670210	2357031	334635	Consumer loans	6622.020	8775

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATI
1670211	2659632	249544	Consumer loans	11520.855	10523
1670212	2785582	400317	Cash loans	18821.520	18000

In [59]:
```
res
```

Out[59]:

	SK_ID_CURR	min	max	mean	min	max	mean
0	100001	3951.000	3951.000	3951.000000	24835.5	24835.5	24835.500
1	100002	9251.775	9251.775	9251.775000	179055.0	179055.0	179055.000
2	100003	6737.310	98356.995	56553.990000	68809.5	900000.0	435436.500
3	100004	5357.250	5357.250	5357.250000	24282.0	24282.0	24282.000
4	100005	4813.200	4813.200	4813.200000	0.0	44617.5	22308.750
...	...	...	...	...	...	...	...
338852	456251	6605.910	6605.910	6605.910000	40455.0	40455.0	40455.000
338853	456252	10074.465	10074.465	10074.465000	57595.5	57595.5	57595.500
338854	456253	3973.095	5567.715	4770.405000	19413.0	28912.5	24162.750
338855	456254	2296.440	19065.825	10681.132500	18846.0	223789.5	121317.750
338856	456255	2250.000	54022.140	20775.391875	45000.0	1170000.0	362770.875

338857 rows × 7 columns

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 [115…
```
X_kaggle_test = datasets["application_test"]
```

In [124…
```
test_class_scores = logreg.predict_proba(X_kaggle_test)[:, 1]
```

In [125…
```
test_class_scores[0:10]
```

Out[125…
```
array([0.06300814, 0.24854787, 0.05847271, 0.03112963, 0.12385396,
       0.02810151, 0.02408003, 0.10026075, 0.01751935, 0.10972745])
```

In [126…

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

submit_df.head()
```

Out[126…

	SK_ID_CURR	TARGET
0	100001	0.063008
1	100005	0.248548
2	100013	0.058473
3	100028	0.031130
4	100038	0.123854

In [127…

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

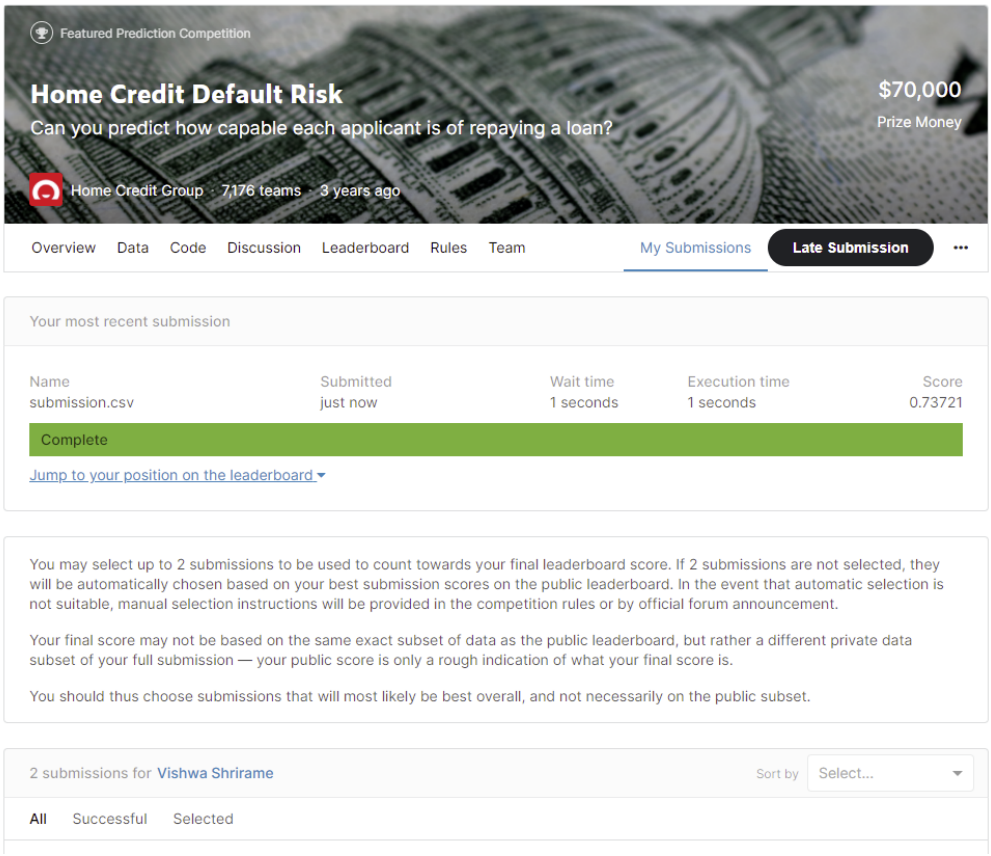
Kaggle submission via the command line API

In [128…

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

```
100%|████████████████████████████████████████| 1.26M/1.26M [00:02<00:00, 634kB
/s]
Successfully submitted to Home Credit Default Risk
```

Our Submission



report submission

Click on this link

Abstract

The main goal of this project is to build a ML model which will predict if a loan applicant will be able to repay his/her loan. We have used existing HCDR data to train our model. We took raw data from Kaggle and started EDA to understand and identify important features available which will be used to train our model. We did visual EDA of the tables to understand the meaning of each column, whether there is a numerical or categorical variable in each column and whether data is the independent or dependent variable.We started data preprocessing in which we imputed all the missing values using median for numerical and most frequent for categorical attributes. Now the data is split into 3 parts Train, Test, and Validation. In modeling, we implemented pipelines to train our model using the following estimators: Logistic Regression, Decision Tree, and Random Forest. The test accuracy score for the estimator is 91.93, 85.46, 91.95 respectively.On Kaggle's submission, we got a beat accuracy score for Logistic Regression which is 0.7372 hence we will be this as an estimator for predicting whether an applicant will be able to repay a loan or not.

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

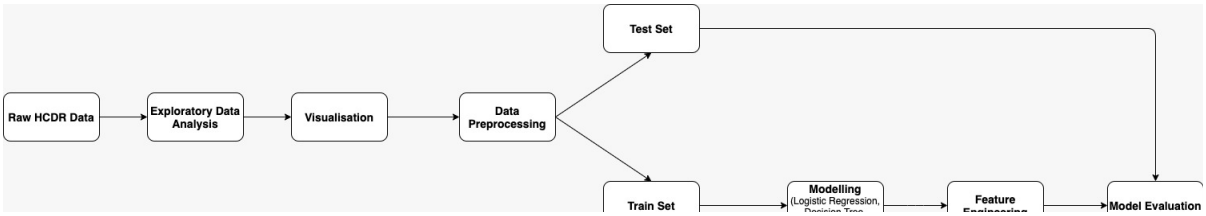
The columns in the various data files are described in this file.

Analyzing a borrower's background is crucial for banks because they have to make sure the borrower returns the loaned amount inside the allocated time interval. How can we solve this problem of identifying the borrower's background quickly and effectively? We plan to tackle this by developing a machine learning pipeline where we use the data from the past borrowers and predict whether the borrower will be able to repay the loan or not and thus we will make use of classification and regression machine learning algorithms to aid our implementation.

The tasks to be tackled are:

- Analyzing Raw HCDR Data

- Implementing Exploratory Data Analysis and Data Visualization
- Performing Data Preprocessing
- Splitting Data set into Train Test and Validation
- Modelling using Logistic Regression, Decision Tree and Random Forest
- Performing Feature Engineering
- Modelling Evaluation



Pipelines

Please explain the pipelines you created for this project and how you used them Please include code sections when necessary as well as images or any relevant material

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 [ ]:
```
tree = DecisionTreeClassifier(criterion='gini', random_state = 42)

data_pipeline_dt = make_pipeline(data_pipeline, tree)
```

Here we are passing the data pipeline to the Decision Tree classifier

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

data_pipeline_rf = make_pipeline(data_pipeline, RF)
```

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

Feature Engineering and transformers

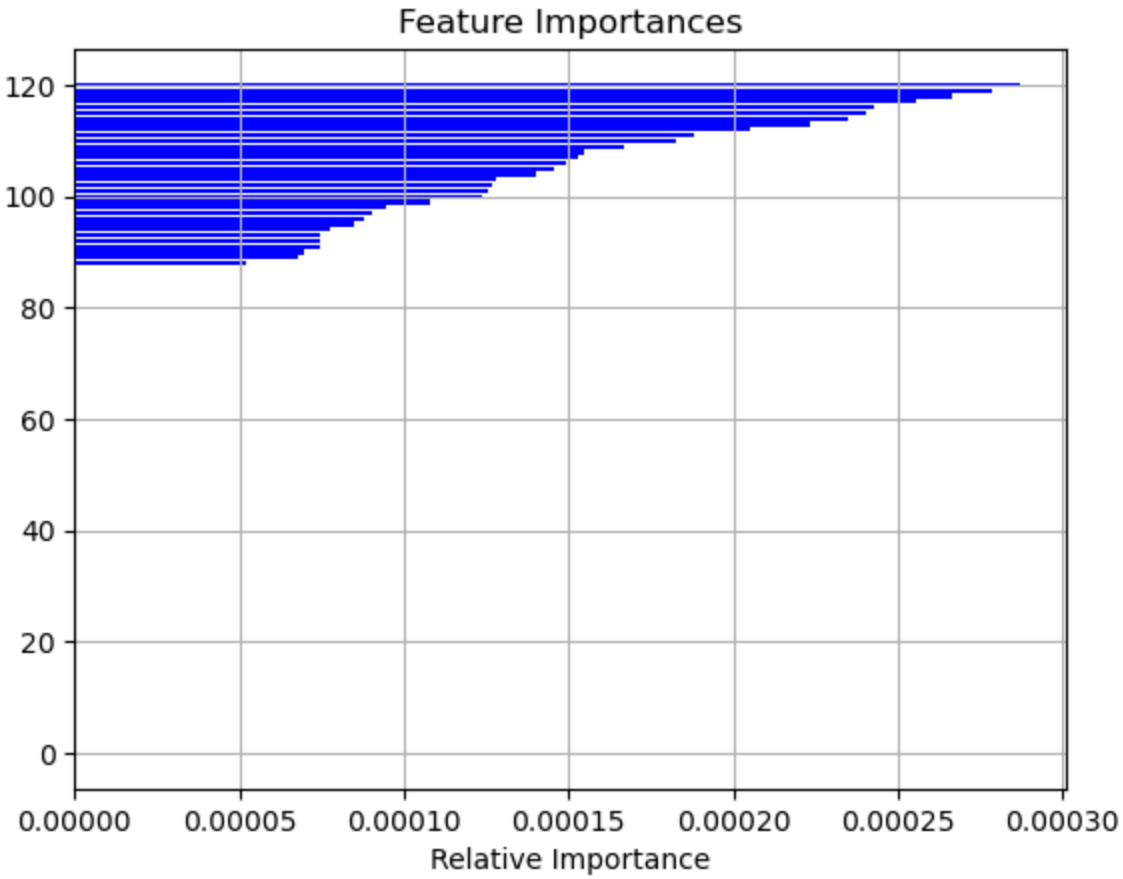
Please explain the work you conducted on feature engineering and transformers. Please include code sections when necessary as well as images or any relevant material

We used Column Transformer to combine the numerical and categorical pipeline

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

We calculated the feature importances from the Decision Tree and Random Forest model

In [213…
```
features = X_train.columns
importances = RF.feature_importances_
indices = np.argsort(importances)[:len(features)]
top_feature = indices[0]

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.grid()
plt.show();
```



In [217…

```
#Features
for idx,f in enumerate(features):
    print(f'{idx} {f}')
```

```
0 SK_ID_CURR
1 NAME_CONTRACT_TYPE
2 CODE_GENDER
3 FLAG_OWN_CAR
4 FLAG_OWN_REALTY
5 CNT_CHILDREN
6 AMT_INCOME_TOTAL
7 AMT_CREDIT
8 AMT_ANNUITY
9 AMT_GOODS_PRICE
10 NAME_TYPE_SUITE
11 NAME_INCOME_TYPE
12 NAME_EDUCATION_TYPE
13 NAME_FAMILY_STATUS
14 NAME_HOUSING_TYPE
15 REGION_POPULATION_RELATIVE
16 DAYS_BIRTH
17 DAYS_EMPLOYED
18 DAYS_REGISTRATION
19 DAYS_ID_PUBLISH
20 OWN_CAR_AGE
21 FLAG_MOBIL
22 FLAG_EMP_PHONE
23 FLAG_WORK_PHONE
24 FLAG_CONT_MOBILE
```

```
25 FLAG_PHONE
26 FLAG_EMAIL
27 OCCUPATION_TYPE
28 CNT_FAM_MEMBERS
29 REGION_RATING_CLIENT
30 REGION_RATING_CLIENT_W_CITY
31 WEEKDAY_APPR_PROCESS_START
32 HOUR_APPR_PROCESS_START
33 REG_REGION_NOT_LIVE_REGION
34 REG_REGION_NOT_WORK_REGION
35 LIVE_REGION_NOT_WORK_REGION
36 REG_CITY_NOT_LIVE_CITY
37 REG_CITY_NOT_WORK_CITY
38 LIVE_CITY_NOT_WORK_CITY
39 ORGANIZATION_TYPE
40 EXT_SOURCE_1
41 EXT_SOURCE_2
42 EXT_SOURCE_3
43 APARTMENTS_AVG
44 BASEMENTAREA_AVG
45 YEARS_BEGINEXPLUATATION_AVG
46 YEARS_BUILD_AVG
47 COMMONAREA_AVG
48 ELEVATORS_AVG
49 ENTRANCES_AVG
50 FLOORSMAX_AVG
51 FLOORSMIN_AVG
52 LANDAREA_AVG
53 LIVINGAPARTMENTS_AVG
54 LIVINGAREA_AVG
55 NONLIVINGAPARTMENTS_AVG
56 NONLIVINGAREA_AVG
57 APARTMENTS_MODE
58 BASEMENTAREA_MODE
59 YEARS_BEGINEXPLUATATION_MODE
60 YEARS_BUILD_MODE
61 COMMONAREA_MODE
62 ELEVATORS_MODE
63 ENTRANCES_MODE
64 FLOORSMAX_MODE
65 FLOORSMIN_MODE
66 LANDAREA_MODE
67 LIVINGAPARTMENTS_MODE
68 LIVINGAREA_MODE
69 NONLIVINGAPARTMENTS_MODE
70 NONLIVINGAREA_MODE
71 APARTMENTS_MEDI
72 BASEMENTAREA_MEDI
73 YEARS_BEGINEXPLUATATION_MEDI
74 YEARS_BUILD_MEDI
75 COMMONAREA_MEDI
76 ELEVATORS_MEDI
77 ENTRANCES_MEDI
78 FLOORSMAX_MEDI
79 FLOORSMIN_MEDI
80 LANDAREA_MEDI
81 LIVINGAPARTMENTS_MEDI
82 LIVINGAREA_MEDI
83 NONLIVINGAPARTMENTS_MEDI
```

```
84 NONLIVINGAREA_MEDI
85 FONDKAPREMONT_MODE
86 HOUSETYPE_MODE
87 TOTALAREA_MODE
88 WALLSMATERIAL_MODE
89 EMERGENCYSTATE_MODE
90 OBS_30_CNT_SOCIAL_CIRCLE
91 DEF_30_CNT_SOCIAL_CIRCLE
92 OBS_60_CNT_SOCIAL_CIRCLE
93 DEF_60_CNT_SOCIAL_CIRCLE
94 DAYS_LAST_PHONE_CHANGE
95 FLAG_DOCUMENT_2
96 FLAG_DOCUMENT_3
97 FLAG_DOCUMENT_4
98 FLAG_DOCUMENT_5
99 FLAG_DOCUMENT_6
100 FLAG_DOCUMENT_7
101 FLAG_DOCUMENT_8
102 FLAG_DOCUMENT_9
103 FLAG_DOCUMENT_10
104 FLAG_DOCUMENT_11
105 FLAG_DOCUMENT_12
106 FLAG_DOCUMENT_13
107 FLAG_DOCUMENT_14
108 FLAG_DOCUMENT_15
109 FLAG_DOCUMENT_16
110 FLAG_DOCUMENT_17
111 FLAG_DOCUMENT_18
112 FLAG_DOCUMENT_19
113 FLAG_DOCUMENT_20
114 FLAG_DOCUMENT_21
115 AMT_REQ_CREDIT_BUREAU_HOUR
116 AMT_REQ_CREDIT_BUREAU_DAY
117 AMT_REQ_CREDIT_BUREAU_WEEK
118 AMT_REQ_CREDIT_BUREAU_MON
119 AMT_REQ_CREDIT_BUREAU_QRT
120 AMT_REQ_CREDIT_BUREAU_YEAR
```

Experimental results

In [206…

```
results
```

Out[206…

	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

Discussion Experimental Results

After obtaining the results of different machine learning algorithms, we can state that

logistic regression and random forest have a higher accuracy as compared to decision tree model.

Logistic regression gave the Testing accuracy of 91.93%

Decision Tree gave the Testing accuracy of 85.46%

Random Forest gave the Testing accuracy of 91.95%

The test ROC value for Logistic regression and Random Forest is 0.745 and 0.721 respectively.

After analyzing the ROC curve, we came to the conclusion, that Logistic regression and random forest performed similarly where as Decision Tree performed poorly both in terms of testing accuracy and ROC value. The ROC value for Decision Tree is 0.5389 which is as good as any random guessing. After making our kaggle submission we found that the logistic regression gives a better score and thus we can conclude that our best model would be Logistic Regression

Conclusion

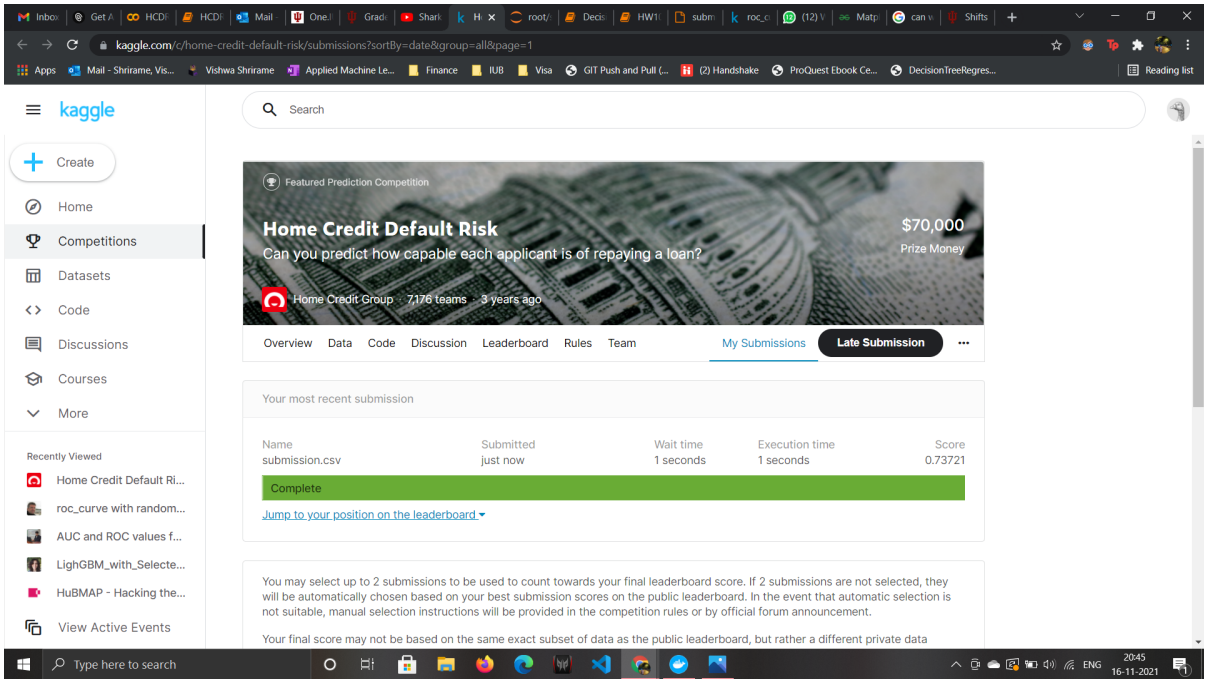
The main purpose of this project is to create a Machine Learning model that can predict whether or not a loan applicant will be able to repay the loan. Many worthy applicants with no credit history or default history are getting without any statistical analysis. The ML model which we are creating is trained with 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.

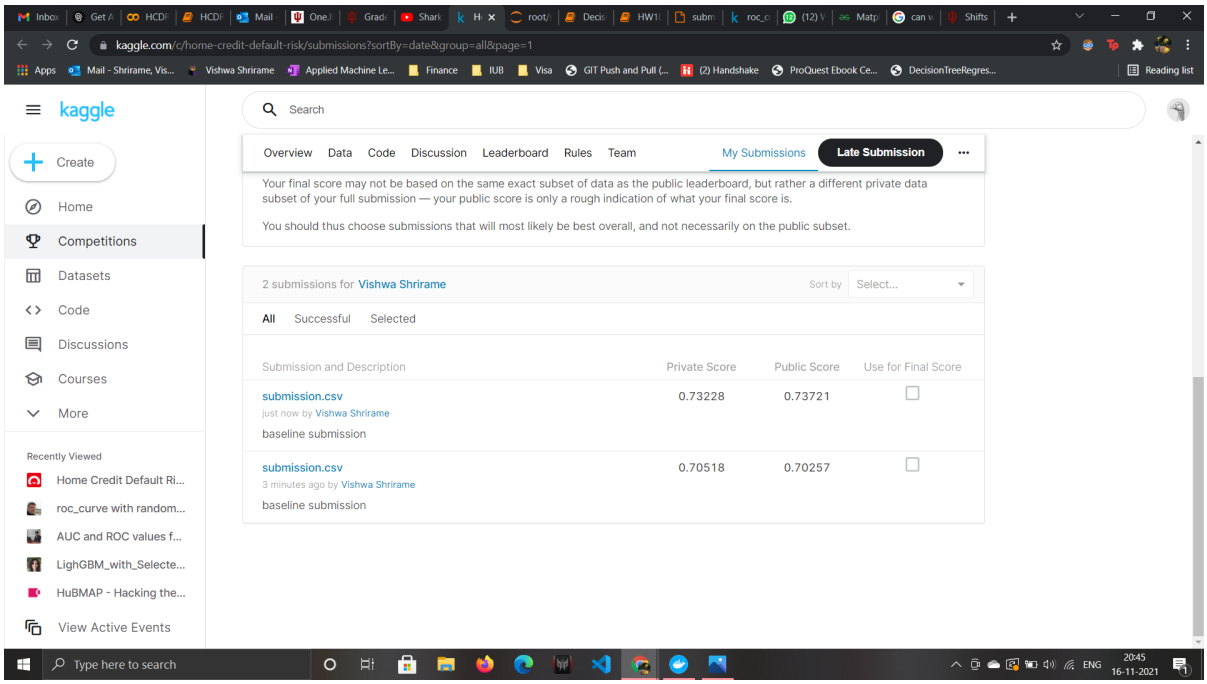
While training we made use of multiple estimators and concluded that Logistic Regression has best test accuracy **91.93%** and on Kaggle submission we have an accuracy score of **0.7372**. The result which we got after modeling provides confidence that it will be able to successfully predict applicants' credit worthiness.

This is the first iteration of our model and subsequent iterations of the same are to be followed in the next phase. Moving on, we will be performing several iterations of Feature Engineering and Hyper-Parameter Tuning.

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

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/