Home Credit Default Risk (HCDR)

Project Description

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

Some of the challenges

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

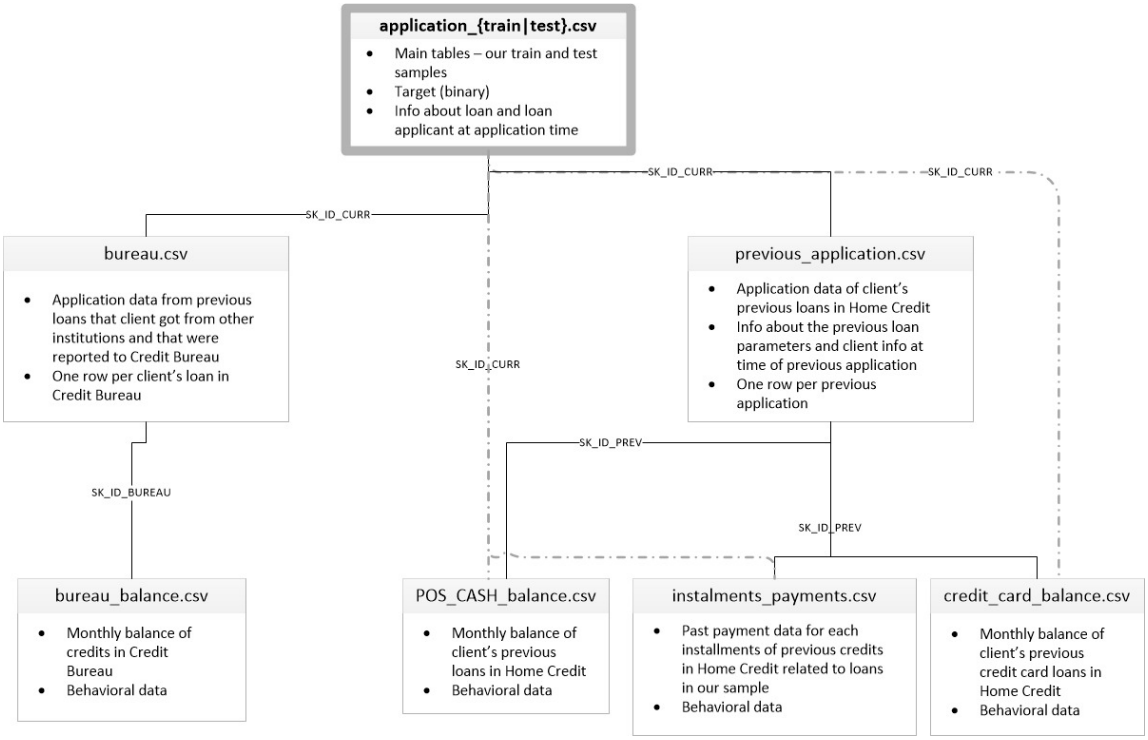
- Dealing with missing data
- Imbalanced datasets
- Summarizing transaction data

Dataset and Project Description

The main training section of the data is compiled in a CSV labeled “application_train.csv” and the counterpart test data in “application_test.csv”. The training data set’s columns differ from the test set only in that it includes the target column for whether the client repaid the loan. There are 121 feature columns in the application dataset which include both categorical, such as gender, and numerical features, such as income.

There are several other datasets that correlate to the application dataset, and they build off each other. The “bureau.csv” dataset contains a row for each previous credit the client had before the application date corresponding to the loan in the application dataset. The “bureau_balance.csv” dataset contains each month of history there is data for the previous credits mentioned in the bureau dataset. In this way, these subsidiary datasets build off the application dataset and each other. These subsidiary datasets contain categorical values and positive and negative numerical values. There are 6 subsidiary datasets in total, but the main dataset we will be looking at is the application set, as this is what the test set is based on.

The following diagram shows the relation between various datasets provided in the problem:



application_{train|test}.csv This is the main dataset consisting of test and training samples. It consists of personal details about the loan applicant and connects with other sets using SK_ID_CURR attribute. previous_application.csv This dataset consists of information about applicant's previous loans in Home credit. It contains attributes such as type of loan, status of loan, down-payment etc. instalments_payments.csv This dataset contains information about repayment of previous loans. credit_card_balance.csv This dataset consists of transaction information related to Home Credit credit cards. POS_CASH_balance.csv This dataset consists of entries that define the status of previous credit of the individual at Home Credit which includes consumer credit and cash loans. bureau.csv This dataset consists of information of the individual's previous credit history at other financial institutions that were reported to the credit bureau.

```
  7.  Previous_application.csv
This dataset contains the monthly balance of credits in the credit
bureau.
```

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 [ ]:
```
# conda install -y torchtext -c pytorch
```

```
Collecting package metadata (current_repodata.json): done
Solving environment: |
```

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

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.7/site-packages
(1.5.12)

Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.7/site-packages
(from kaggle) (1.15.0)

Requirement already satisfied: urllib3 in /usr/local/lib/python3.7/site-packages
(from kaggle) (1.26.6)

Requirement already satisfied: requests in /usr/local/lib/python3.7/site-packages
(from kaggle) (2.25.1)

Requirement already satisfied: python-dateutil in /usr/local/lib/python3.7/site-pa
ckages (from kaggle) (2.8.2)

Requirement already satisfied: certifi in /usr/local/lib/python3.7/site-packages
(from kaggle) (2021.5.30)

Requirement already satisfied: python-slugify in /usr/local/lib/python3.7/site-pac
kages (from kaggle) (5.0.2)

Requirement already satisfied: tqdm in /usr/local/lib/python3.7/site-packages (fro
m kaggle) (4.62.1)

Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.7/sit
e-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-packa
ges (from requests->kaggle) (2.10)

WARNING: Running pip as the 'root' user can result in broken permissions and confl
icting behaviour with the system package manager. It is recommended to use a virtu
al environment instead: https://pip.pypa.io/warnings/venv

WARNING: You are using pip version 21.2.4; however, version 21.3.1 is available.
```

```
You should consider upgrading via the '/usr/local/bin/python -m pip install --upgr
ade pip' command.
```

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

```
/root/shared/AML/I526_AML_Student/Assignments/Unit-Project-Home-Credit-Default-Ris
k/Phase2
```

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

```
/root/shared/AML/I526_AML_Student/Assignments/Unit-Project-Home-Credit-Default-Ris
k/Phase2
```

In [ ]:
```
!ls -l  ~/.kaggle/kaggle.json
```

```
-rw------- 1 root root 62 Nov 22 17:40 /root/.kaggle/kaggle.json
```

In [ ]:
```
# !mkdir ~/.kaggle
# !cp kaggle.json ~/.kaggle
!chmod 600 ~/.kaggle/kaggle.json
```

In [ ]:
```
! 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

previous_application.csv            386MB  2019-12-11 02:55:35

application_train.csv               158MB  2019-12-11 02:55:35

HomeCredit_columns_description.csv   37KB  2019-12-11 02:55:35

credit_card_balance.csv             405MB  2019-12-11 02:55:35

sample_submission.csv               524KB  2019-12-11 02:55:35

bureau.csv                          162MB  2019-12-11 02:55:35

bureau_balance.csv                  358MB  2019-12-11 02:55:35

application_test.csv                 25MB  2019-12-11 02:55:35
```

Dataset and how to download

Back ground Home Credit Group

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

Home Credit Group

Home Credit strives to broaden financial inclusion for the unbanked population by providing a positive and safe borrowing experience. In order to make sure this underserved population has a positive loan experience, Home Credit makes use of a variety of alternative

data--including telco and transactional information--to predict their clients' repayment abilities.

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

Background on the dataset

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

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

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

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

Data files overview

The `HomeCredit_columns_description.csv` acts as a data dictioanry.

There are 7 different sources of data:

- **application_train/application_test (307k rows, and 48k rows):** the main training and testing data with information about each loan application at Home Credit. Every loan has its own row and is identified by the feature SK_ID_CURR. The training application data comes with the TARGET indicating **0: the loan was repaid** or **1: the loan was not repaid**. The target variable defines if the client had payment difficulties meaning he/she had late payment more than X days on at least one of the first Y installments of the loan. Such case is marked as 1 while other all other cases as 0.
- **bureau (1.7 Million rows):** data concerning client's previous credits from other financial institutions. Each previous credit has its own row in bureau, but one loan in the application data can have multiple previous credits.
- **bureau_balance (27 Million rows):** monthly data about the previous credits in bureau. Each row is one month of a previous credit, and a single previous credit can have multiple rows, one for each month of the credit length.
- **previous_application (1.6 Million rows):** previous applications for loans at Home Credit of clients who have loans in the application data. Each current loan in the

application data can have multiple previous loans. Each previous application has one row and is identified by the feature SK_ID_PREV.

- **POS_CASH_BALANCE (10 Million rows):** monthly data about previous point of sale or cash loans clients have had with Home Credit. Each row is one month of a previous point of sale or cash loan, and a single previous loan can have many rows.
- credit_card_balance: monthly data about previous credit cards clients have had with Home Credit. Each row is one month of a credit card balance, and a single credit card can have many rows.
- **installments_payment (13.6 Million rows):** payment history for previous loans at Home Credit. There is one row for every made payment and one row for every missed payment.

Table sizes

```
name                         [  rows cols]     MegaBytes
-----------------------  ------------------  -------
application_train        : [  307,511, 122]:   158MB
application_test         : [   48,744, 121]:   25MB
bureau                   : [ 1,716,428, 17]    162MB
bureau_balance           : [ 27,299,925, 3]:   358MB
credit_card_balance      : [  3,840,312, 23]   405MB
installments_payments    : [ 13,605,401, 8]    690MB
previous_application     : [  1,670,214, 37]   386MB
POS_CASH_balance         : [ 10,001,358, 8]    375MB
```

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 [ ]:
```
# DATA_DIR = "/root/shared/AML/I526_AML_Student/Assignments/Unit-Project-Home-Credi
#DATA_DIR = os.path.join('./ddddd/')
# !mkdir DATA_DIR
DATA_DIR = '../input/home-credit-default-risk'
```

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

```
ls: cannot access 'DATA_DIR': No such file or directory
```

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

```
Downloading home-credit-default-risk.zip to /root/shared/AML/I526_AML_Student/Assi
gnments/Unit-Project-Home-Credit-Default-Risk/Phase2


 13%|████▊                                  | 87.0M/688M [00:04<00:33, 18.6MB/s]^
C


 13%|████▊                                  | 87.0M/688M [00:04<00:34, 18.4MB/s]


User cancelled operation
```

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

```
/root/shared/AML/I526_AML_Student/Assignments/Unit-Project-Home-Credit-Default-Ris
k/Phase2
```

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

```
total 831304


drwxr-xr-x 1 root root      4096 Nov 25 21:45 DATA_DIR


-rwxrwxrwx 1 root root   4988449 Nov 29 20:34 HCDR_baseLine_submission_with_numeri
cal_and_cat_features_to_kaggle_v1.ipynb


-rwxrwxrwx 1 root root   2215207 Nov 29 20:47 HCDR_baseLine_submission_with_numeri
cal_and_cat_features_to_kaggle_v2.ipynb


-rwxrwxrwx 1 root root        10 Nov 25 21:34 Phase2.md


-rw-r--r-- 1 root root  91226112 Nov 29 20:47 home-credit-default-risk.zip


-rwxrwxrwx 1 root root     66899 Nov 25 21:34 home_credit.png


-rw-r--r-- 1 root root 375168662 Nov 29 00:41 merged_data_test.csv


-rw-r--r-- 1 root root 375168662 Nov 29 00:41 merged_data_train.csv


-rwxrwxrwx 1 root root   1320236 Nov 25 21:34 submission.csv


-rwxrwxrwx 1 root root   1091396 Nov 25 21:35 submission.png
```

In [ ]:
```
#!rm -r  DATA_DIR
```

Imports

In [ ]:
```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import os
import zipfile
from sklearn.base import BaseEstimator, TransformerMixin
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.impute import SimpleImputer
```

```
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline, FeatureUnion
from pandas.plotting import scatter_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:
```
unzippingReq = True #True
if unzippingReq: #please modify this code
    zip_ref = zipfile.ZipFile(f'{DATA_DIR}/home-credit-default-risk.zip', 'r')
    # extractall():  Extract all members from the archive to the current working d
    zip_ref.extractall('DATA_DIR')
    zip_ref.close()
```

Data files overview

Data Dictionary

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

Application train

In [ ]:
```
ls -l /root/shared/AML/I526_AML_Student/Assignments/Unit-Project-Home-Credit-Defau
```

```
-rw-r--r-- 1 root root 166133370 Nov 25 21:45 /root/shared/AML/I526_AML_Student/As
signments/Unit-Project-Home-Credit-Default-Risk/Phase2/DATA_DIR/application_train.
csv
```

In [ ]:
```
```

In [ ]:
```
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')
%matplotlib inline


def load_data(in_path, name):
    df = pd.read_csv(in_path)
    print(f"{name}: shape is {df.shape}")
```

```
    print(df.info())
    display(df.head(5))
    return df

datasets={}  # lets store the datasets in a dictionary so we can keep track of ther
ds_name = 'application_train'
# DATA_DIR='/root/shared/AML/I526_AML_Student/Assignments/Unit-Project-Home-Credit-
datasets[ds_name] = load_data(os.path.join(DATA_DIR, f'{ds_name}.csv'), ds_name)

datasets['application_train'].shape
```

```
application_train: shape is (307511, 122)

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 307511 entries, 0 to 307510

Columns: 122 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR

dtypes: float64(65), int64(41), object(16)

memory usage: 286.2+ MB

None
```

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_RE
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[ ]:
```
(307511, 122)
```

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

Out[ ]:
```
dict_keys(['application_train'])
```

In [ ]:
```
DATA_DIR
```

Out[ ]:
```
'/root/shared/AML/I526_AML_Student/Assignments/Unit-Project-Home-Credit-Default-Risk/Phase2/DATA_DIR/'
```

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

```
application_test: shape is (48744, 121)


<class 'pandas.core.frame.DataFrame'>


RangeIndex: 48744 entries, 0 to 48743


Columns: 121 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR


dtypes: float64(65), int64(40), object(16)


memory usage: 45.0+ MB


None
```

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

5 rows × 121 columns

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

The Other datasets

- **bureau:** data concerning client's previous credits from other financial institutions. Each previous credit has its own row in bureau, but one loan in the application data can have multiple previous credits.
- **bureau_balance:** monthly data about the previous credits in bureau. Each row is one month of a previous credit, and a single previous credit can have multiple rows, one for each month of the credit length.
- **previous_application:** previous applications for loans at Home Credit of clients who have loans in the application data. Each current loan in the application data can have

multiple previous loans. Each previous application has one row and is identified by the feature SK_ID_PREV.

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

In [ ]:
```
ds_names = ("application_train", "application_test", "bureau","bureau_balance","cr
            "previous_application","POS_CASH_balance")
```

In [ ]:
```
%%time
ds_names = ("application_train", "application_test", "bureau","bureau_balance","cr
            "previous_application","POS_CASH_balance")

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

```
application_train: shape is (307511, 122)

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 307511 entries, 0 to 307510

Columns: 122 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR

dtypes: float64(65), int64(41), object(16)

memory usage: 286.2+ MB

None
```

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

5 rows × 121 columns

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

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 1716428 entries, 0 to 1716427

Data columns (total 17 columns):

 #   Column                  Dtype

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

 0   SK_ID_CURR              int64

 1   SK_ID_BUREAU            int64

 2   CREDIT_ACTIVE           object

 3   CREDIT_CURRENCY         object

 4   DAYS_CREDIT             int64

 5   CREDIT_DAY_OVERDUE      int64

 6   DAYS_CREDIT_ENDDATE     float64

 7   DAYS_ENDDATE_FACT       float64

 8   AMT_CREDIT_MAX_OVERDUE  float64

 9   CNT_CREDIT_PROLONG      int64

 10  AMT_CREDIT_SUM          float64

 11  AMT_CREDIT_SUM_DEBT     float64

 12  AMT_CREDIT_SUM_LIMIT    float64

 13  AMT_CREDIT_SUM_OVERDUE  float64

 14  CREDIT_TYPE             object

 15  DAYS_CREDIT_UPDATE      int64

 16  AMT_ANNUITY             float64

dtypes: float64(8), int64(6), object(3)

memory usage: 222.6+ MB

None
```

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

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

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

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

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	AMT_CREDIT_LIMIT_ACTUAL	AM
0	2562384	378907	-6	56.970	135000	
1	2582071	363914	-1	63975.555	45000	
2	1740877	371185	-7	31815.225	450000	
3	1389973	337855	-4	236572.110	225000	
4	1891521	126868	-1	453919.455	450000	

5 rows × 23 columns

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

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER	DAYS_INS
0	1054186	161674	1.0	6	
1	1330831	151639	0.0	34	
2	2085231	193053	2.0	1	
3	2452527	199697	1.0	3	
4	2714724	167756	1.0	2	

```
previous_application: shape is (1670214, 37)

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 1670214 entries, 0 to 1670213

Data columns (total 37 columns):

 #   Column                       Non-Null Count    Dtype

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

 0   SK_ID_PREV                   1670214 non-null  int64

 1   SK_ID_CURR                   1670214 non-null  int64

 2   NAME_CONTRACT_TYPE           1670214 non-null  object

 3   AMT_ANNUITY                  1297979 non-null  float64

 4   AMT_APPLICATION              1670214 non-null  float64

 5   AMT_CREDIT                   1670213 non-null  float64

 6   AMT_DOWN_PAYMENT             774370 non-null   float64

 7   AMT_GOODS_PRICE              1284699 non-null  float64

 8   WEEKDAY_APPR_PROCESS_START   1670214 non-null  object

 9   HOUR_APPR_PROCESS_START      1670214 non-null  int64

 10  FLAG_LAST_APPL_PER_CONTRACT  1670214 non-null  object

 11  NFLAG_LAST_APPL_IN_DAY       1670214 non-null  int64

 12  RATE_DOWN_PAYMENT            774370 non-null   float64

 13  RATE_INTEREST_PRIMARY        5951 non-null     float64

 14  RATE_INTEREST_PRIVILEGED     5951 non-null     float64

 15  NAME_CASH_LOAN_PURPOSE       1670214 non-null  object

 16  NAME_CONTRACT_STATUS         1670214 non-null  object

 17  DAYS_DECISION                1670214 non-null  int64

 18  NAME_PAYMENT_TYPE            1670214 non-null  object

 19  CODE_REJECT_REASON           1670214 non-null  object

 20  NAME_TYPE_SUITE              849809 non-null   object

 21  NAME_CLIENT_TYPE             1670214 non-null  object

 22  NAME_GOODS_CATEGORY          1670214 non-null  object

 23  NAME_PORTFOLIO               1670214 non-null  object

 24  NAME_PRODUCT_TYPE            1670214 non-null  object

 25  CHANNEL_TYPE                 1670214 non-null  object
```

```
 26  SELLERPLACE_AREA             1670214 non-null  int64
 27  NAME_SELLER_INDUSTRY         1670214 non-null  object
 28  CNT_PAYMENT                  1297984 non-null  float64
 29  NAME_YIELD_GROUP             1670214 non-null  object
 30  PRODUCT_COMBINATION          1669868 non-null  object
 31  DAYS_FIRST_DRAWING           997149 non-null   float64
 32  DAYS_FIRST_DUE               997149 non-null   float64
 33  DAYS_LAST_DUE_1ST_VERSION    997149 non-null   float64
 34  DAYS_LAST_DUE                997149 non-null   float64
 35  DAYS_TERMINATION             997149 non-null   float64
 36  NFLAG_INSURED_ON_APPROVAL    997149 non-null   float64
dtypes: float64(15), int64(6), object(16)
memory usage: 471.5+ MB
None
```

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CI
0	2030495	271877	Consumer loans	1730.430	17145.0	17
1	2802425	108129	Cash loans	25188.615	607500.0	679
2	2523466	122040	Cash loans	15060.735	112500.0	136
3	2819243	176158	Cash loans	47041.335	450000.0	470
4	1784265	202054	Cash loans	31924.395	337500.0	404

5 rows × 37 columns

```
POS_CASH_balance: shape is (10001358, 8)

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 10001358 entries, 0 to 10001357

Data columns (total 8 columns):

 #   Column                 Dtype

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

 0   SK_ID_PREV             int64

 1   SK_ID_CURR             int64

 2   MONTHS_BALANCE         int64

 3   CNT_INSTALMENT         float64

 4   CNT_INSTALMENT_FUTURE  float64

 5   NAME_CONTRACT_STATUS   object

 6   SK_DPD                 int64

 7   SK_DPD_DEF             int64

dtypes: float64(2), int64(5), object(1)

memory usage: 610.4+ MB

None
```

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

```
CPU times: user 52.5 s, sys: 4.99 s, total: 57.5 s
```

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

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

Out[ ]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_A
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	30749
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	2710
std	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	1449
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	161
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	1652
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	2490
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	3459
max	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	25802

8 rows × 106 columns

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

Out[ ]:

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_
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 [ ]:
```
datasets["application_train"].describe(include='all') #look at all categorical and
```

Out[ ]:

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

11 rows × 122 columns

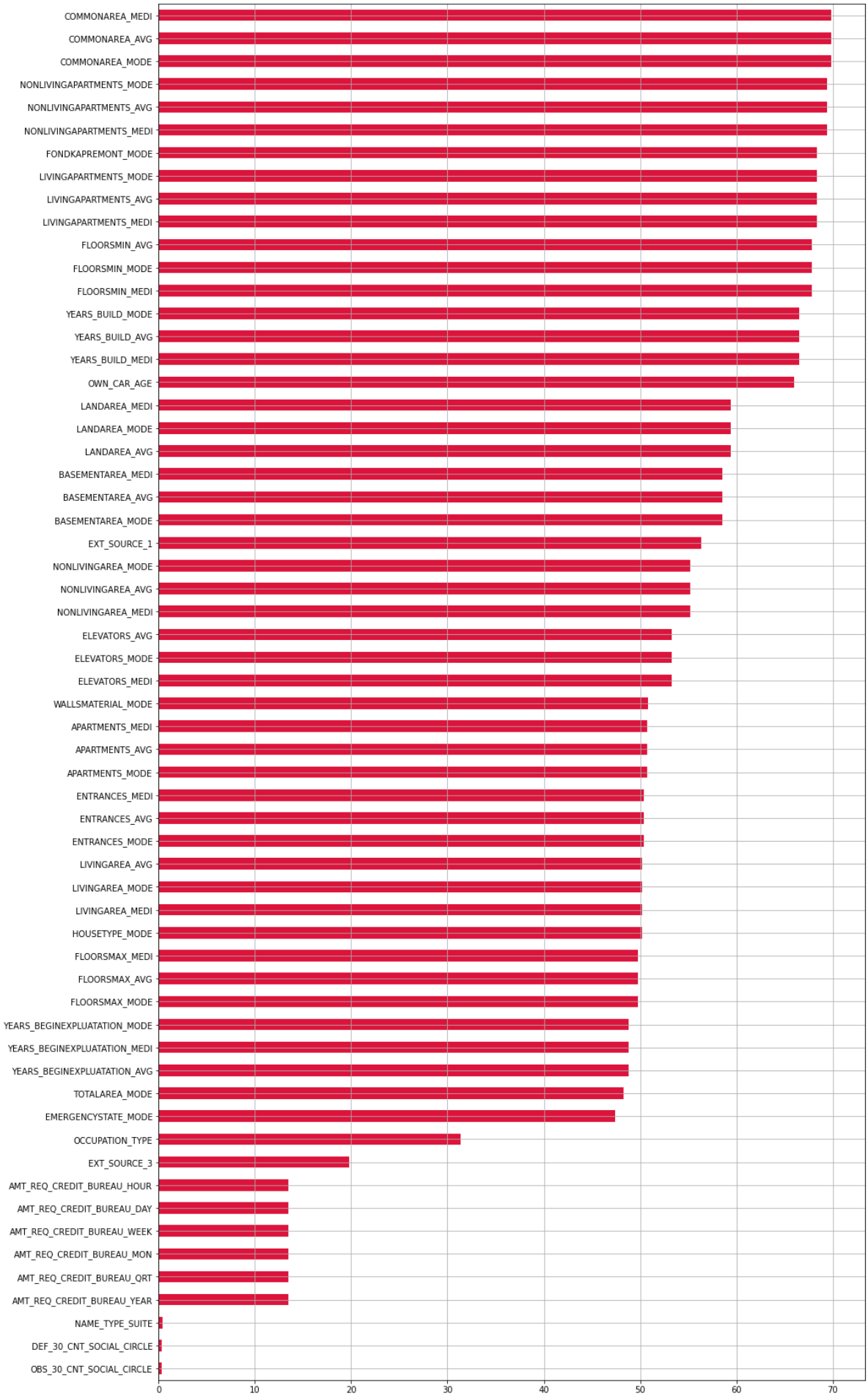
Missing data for application train

In [ ]:
```
percent = (datasets["application_train"].isnull().sum()/datasets["application_trai
sum_missing = datasets["application_train"].isna().sum().sort_values(ascending = Fa
missing_application_train_data  = pd.concat([percent, sum_missing], axis=1, keys=[
missing_application_train_data.head(20)
```

Out[ ]:

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

In [ ]:
```
plt.figure(figsize=(15, 7))
missing_application_train_data['Percent'].sort_values().tail(60).plot.barh(figsize
plt.grid(b=True)
plt.show();
```



In [ ]:
```
percent = (datasets["application_test"].isnull().sum()/datasets["application_test"
sum_missing = datasets["application_test"].isna().sum().sort_values(ascending = Fa
```

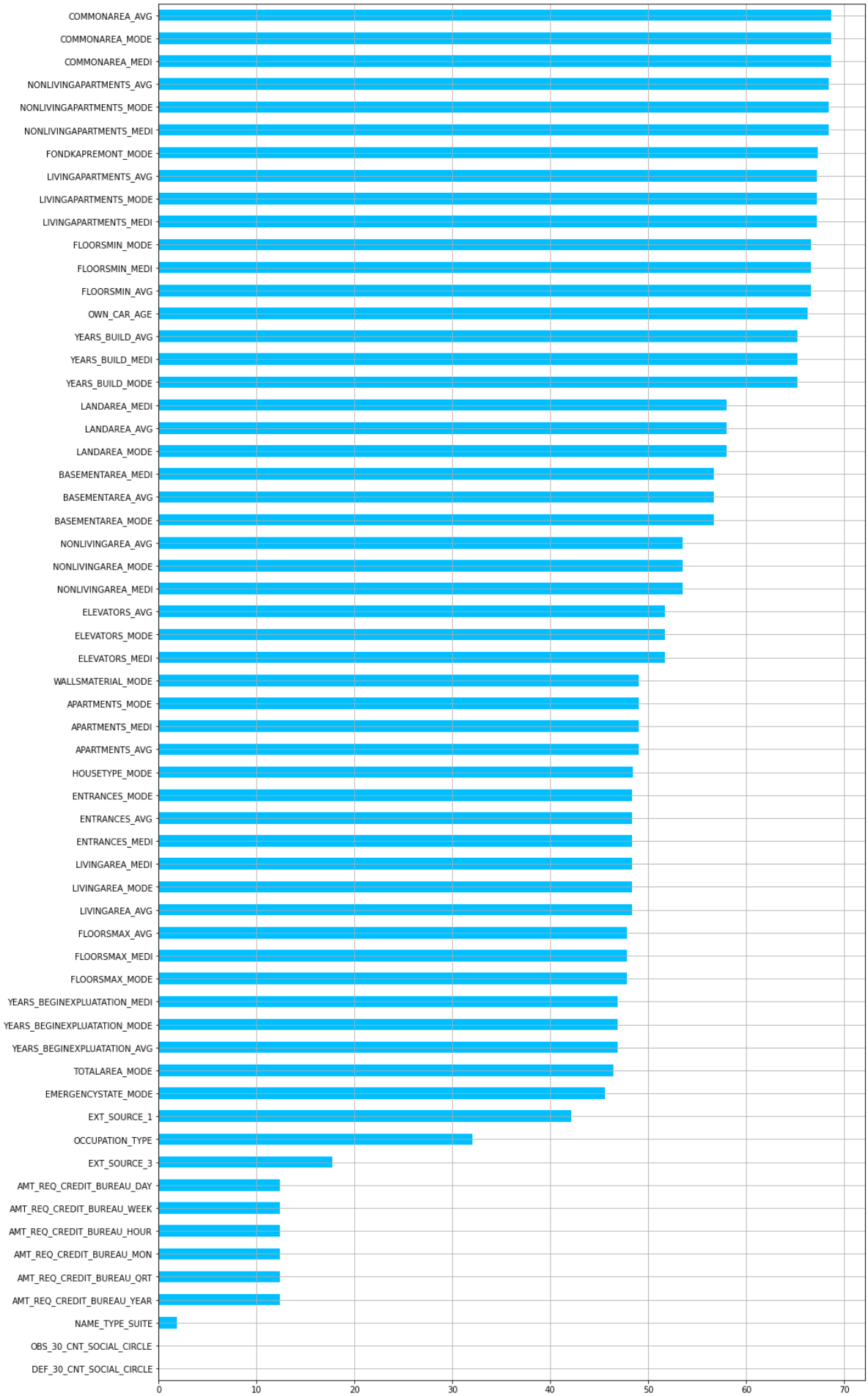
```
missing_application_test_data  = pd.concat([percent, sum_missing], axis=1, keys=['
missing_application_test_data.head(20)
```

Out[ ]:

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

In [ ]:

```
plt.figure(figsize=(15, 7))
missing_application_test_data['Percent'].sort_values().tail(60).plot.barh(figsize=
plt.grid(b=True)
plt.show();
```



Observation:

- We can see that a large portion of the data is missing from train and test sets

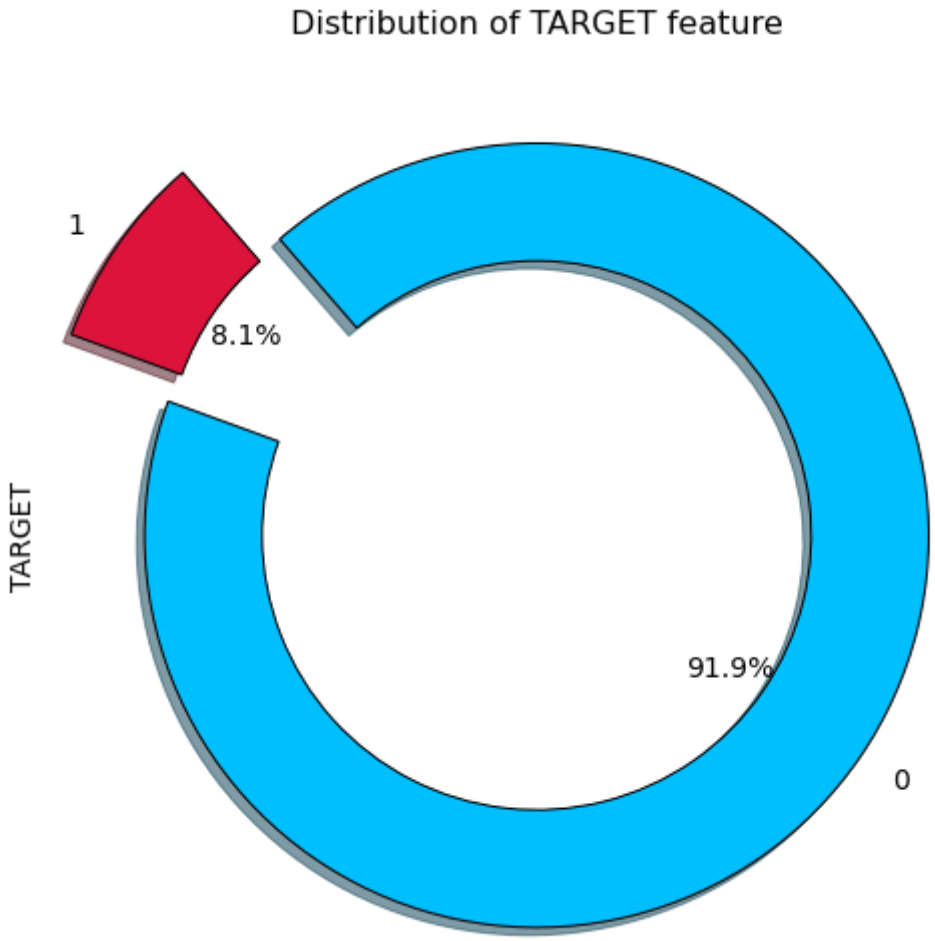
In [ ]:
```
# Setting up the train and test datasets

app_train = datasets["application_train"]
app_test = datasets["application_test"]
```

Distribution of the target column

In [ ]:
```
app_train['TARGET'].value_counts()
```

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

In [ ]:
```
plt.figure(figsize=(9, 9))
plt.pie(x=app_train['TARGET'].value_counts(),
        radius=1.3-0.3,
        labels=app_train['TARGET'].value_counts().index,
        autopct='%1.1f%%',
        colors=['deepskyblue', 'crimson'],
        explode=[0,0.3],
        wedgeprops={"edgecolor":"0", "width":0.3},
        startangle=160,
        shadow=True,
        textprops={'fontsize': 14})
plt.ylabel('TARGET', fontsize=14)
plt.title('Distribution of TARGET feature', fontsize=16)
plt.show()
```



Observation:

- We can observe a high amount of imbalance in the TARGET feature.
- This will cause issues when measuring the accuracy performance metric.

Correlation with the target column

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

```
Most Positive Correlations:

 ELEVATORS_AVG                -0.034199

REGION_POPULATION_RELATIVE   -0.037227

AMT_GOODS_PRICE              -0.039645

FLOORSMAX_MODE               -0.043226

FLOORSMAX_MEDI               -0.043768

FLOORSMAX_AVG                -0.044003

DAYS_EMPLOYED                -0.044932

EXT_SOURCE_1                 -0.155317

EXT_SOURCE_2                 -0.160472

EXT_SOURCE_3                 -0.178919

Name: TARGET, dtype: float64




Most Negative Correlations:

 TARGET                        1.000000

DAYS_BIRTH                    0.078239
```

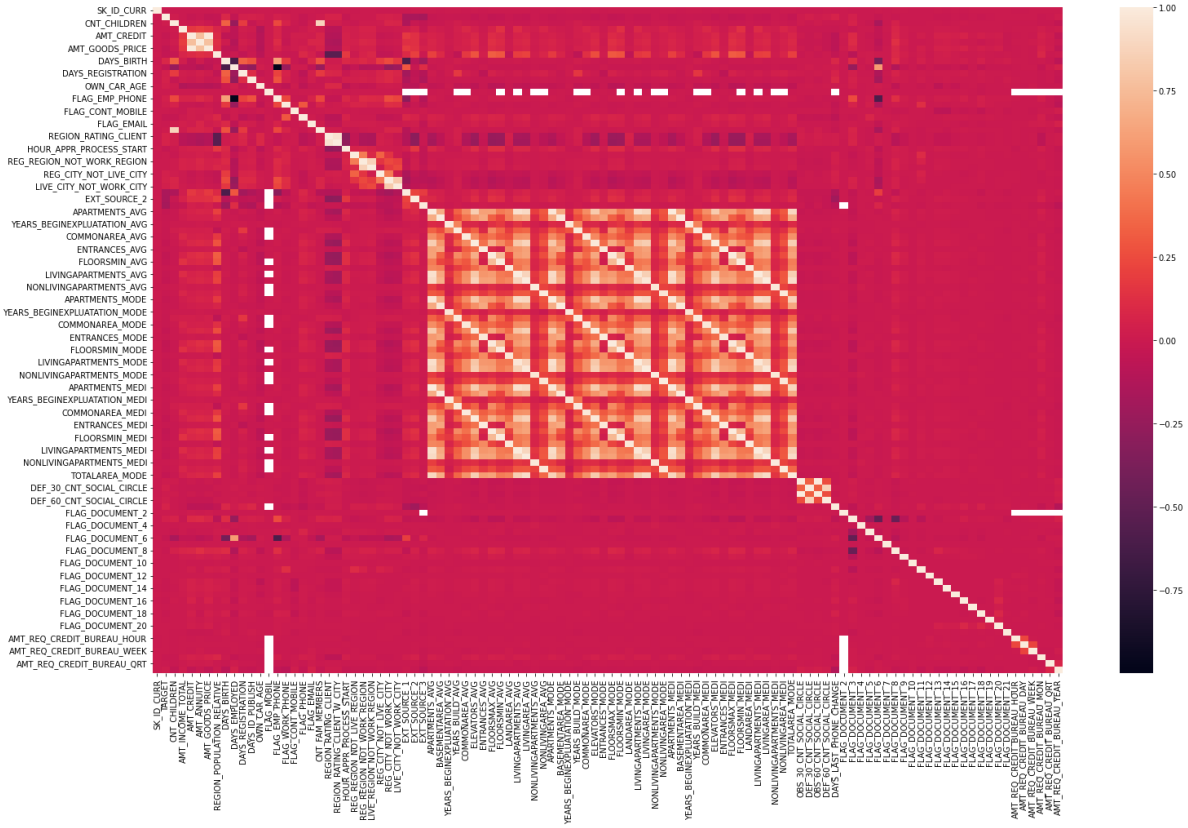
```
REGION_RATING_CLIENT_W_CITY    0.060893

REGION_RATING_CLIENT           0.058899

DAYS_LAST_PHONE_CHANGE         0.055218

DAYS_ID_PUBLISH                0.051457

REG_CITY_NOT_WORK_CITY         0.050994

FLAG_EMP_PHONE                 0.045982

REG_CITY_NOT_LIVE_CITY         0.044395

FLAG_DOCUMENT_3                0.044346

Name: TARGET, dtype: float64
```

Observation:

- The maximum positive correlation with TARGET feature is observed as 0.0782 with DAYS_BIRTH feature. We will obeserve that in the coming sections.
- This is followed by REGION_RATING, DAYS_LAST_PHONE_CHANGE, DAYS_ID_PUBLISH, and REG_CITY_NOT_WORK_CITY features.
- High indirect correlation is observed between TARGET and FLOORS features, External Sources, AMT_GOODS_PRICE, and relative population features.

In [ ]:
```
train_corr = datasets["application_train"].corr()
```

In [ ]:
```
plt.figure(figsize=(25, 15))
sns.heatmap(train_corr, cmap='rocket')
plt.plot();
```



Observation:

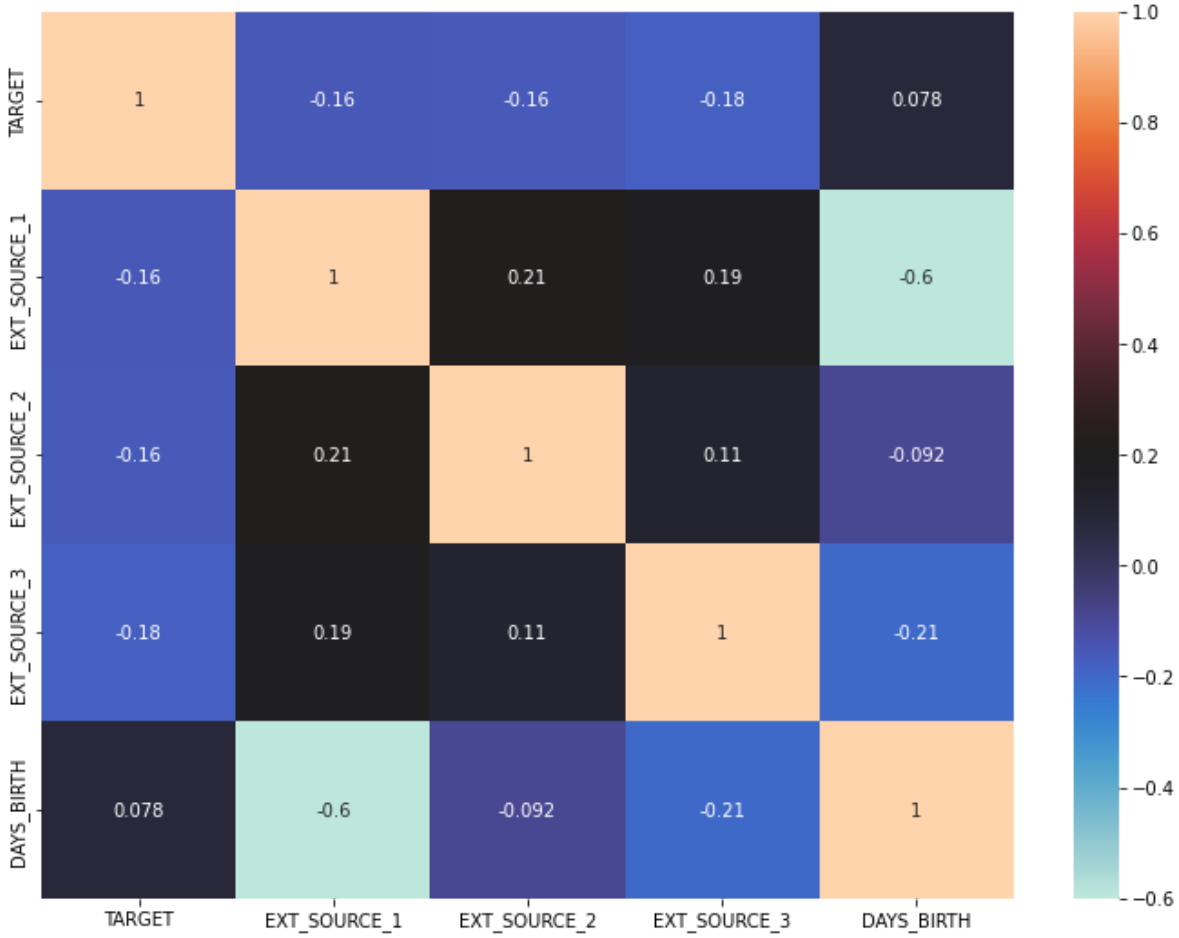
- The heatmap can't be made sense of as of now since we have 120+ columns to compare from.

In [ ]:
```
# Extract the EXT_SOURCE variables and show correlations
ext_source_data = app_train[['TARGET', 'EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_
ext_source_data_corrs = ext_source_data.corr()
ext_source_data_corrs
```

Out[ ]:

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

In [ ]:
```
plt.figure(figsize=(12, 9))
sns.heatmap(ext_source_data_corrs, annot=True, cmap='icefire')
plt.plot();
```



Observation:

- The heatmap shows us that external sources indirectly affect the TARGET feature.
- But we can also see that they are correlated to each other as well i.e. multicollinearity is present.

In [ ]:
```
app_train.describe()
```

Out[ ]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_A
count	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05	30749
mean	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05	2710
std	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05	1449
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	161
25%	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05	1652
50%	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05	2490
75%	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05	3459
max	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	25802

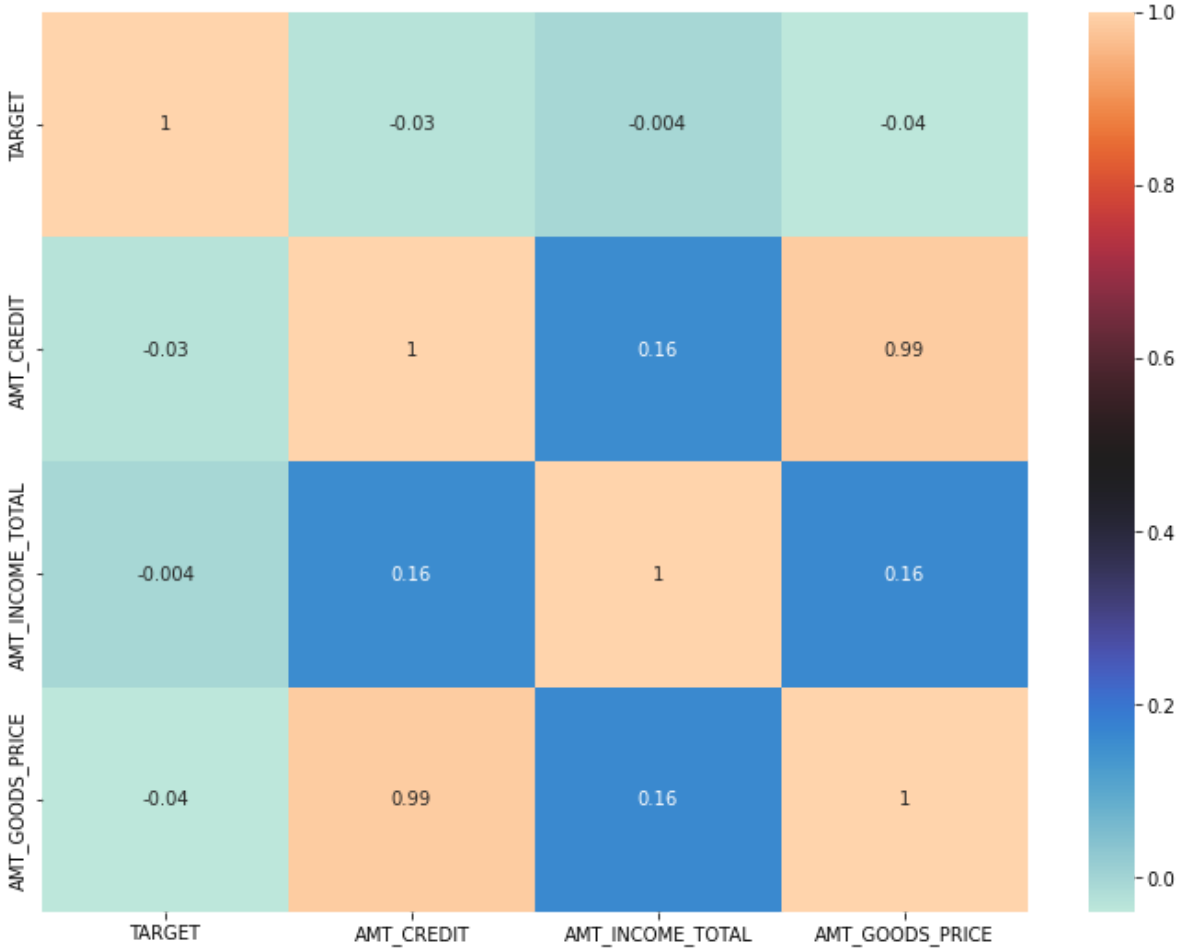
8 rows × 106 columns

In [ ]:
```
# Extract the AMOUNT variables and show correlations
amount_data = app_train[['TARGET', 'AMT_CREDIT', 'AMT_INCOME_TOTAL', 'AMT_GOODS_PR
amount_data_corrs = amount_data.corr()
amount_data_corrs
```

Out[ ]:

	TARGET	AMT_CREDIT	AMT_INCOME_TOTAL	AMT_GOODS_PRICE
TARGET	1.000000	-0.030369	-0.003982	-0.039645
AMT_CREDIT	-0.030369	1.000000	0.156870	0.986968
AMT_INCOME_TOTAL	-0.003982	0.156870	1.000000	0.159610
AMT_GOODS_PRICE	-0.039645	0.986968	0.159610	1.000000

In [ ]:
```
plt.figure(figsize=(12, 9))
sns.heatmap(amount_data_corrs, annot=True, cmap='icefire')
plt.plot();
```



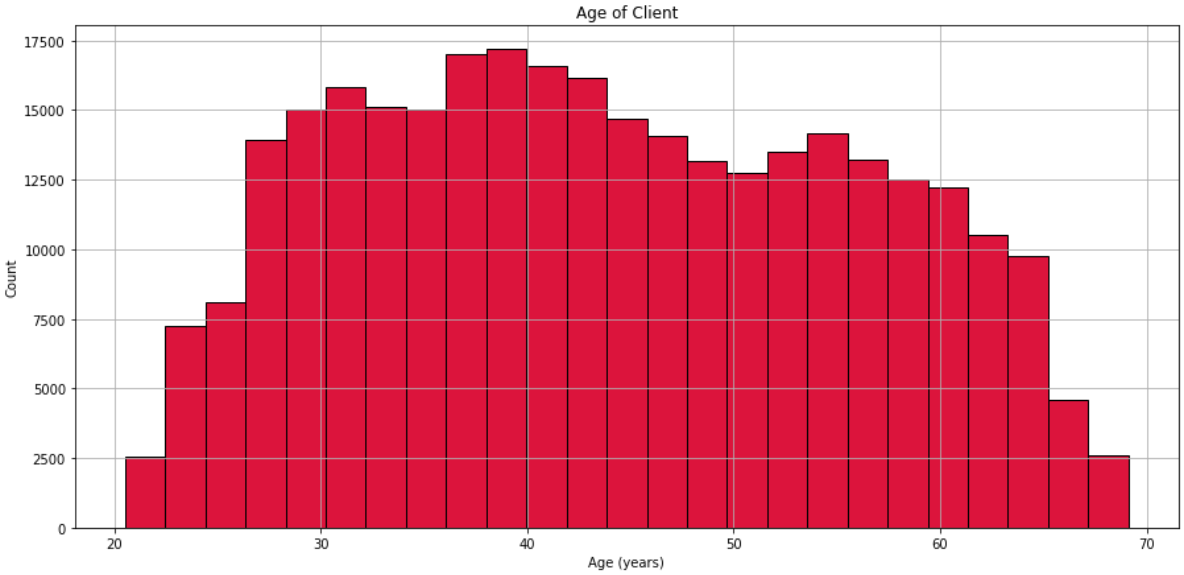
Observation:

- The heatmap shows us yet again a case of multicollinearity among the AMT featurs.
- We may have to deal with these features while proceeding with the modelling.

Exploratory Data Analysis on Categorical Features

Applicants Age

In [ ]:
```
plt.figure(figsize=(15, 7))
plt.hist(datasets["application_train"]['DAYS_BIRTH'] / -365, edgecolor = 'k', bins
plt.title('Age of Client')
plt.xlabel('Age (years)')
plt.ylabel('Count')
plt.grid(b=True)
plt.show()
```



Observation:

- Age is obtained by the DAYS_BIRTH feature which has negative values. This is inconsistent and should be taken care of.
- On plotting the age as number of years, we see a fairly standard distribution which is a good sign in such a complicated dataset as we have DAYS_BIRTH highly correlated with TARGET feature.

In [ ]:
```
# Age information into a separate dataframe
age_data = app_train[['TARGET', 'DAYS_BIRTH']]
age_data['YEARS_BIRTH'] = age_data['DAYS_BIRTH'] / -365

# Bin the age data
age_data['GROUPED_YEARS_BIRTH'] = pd.cut(age_data['YEARS_BIRTH'], bins = np.linspa
age_data.head(10)
```

Out[ ]:

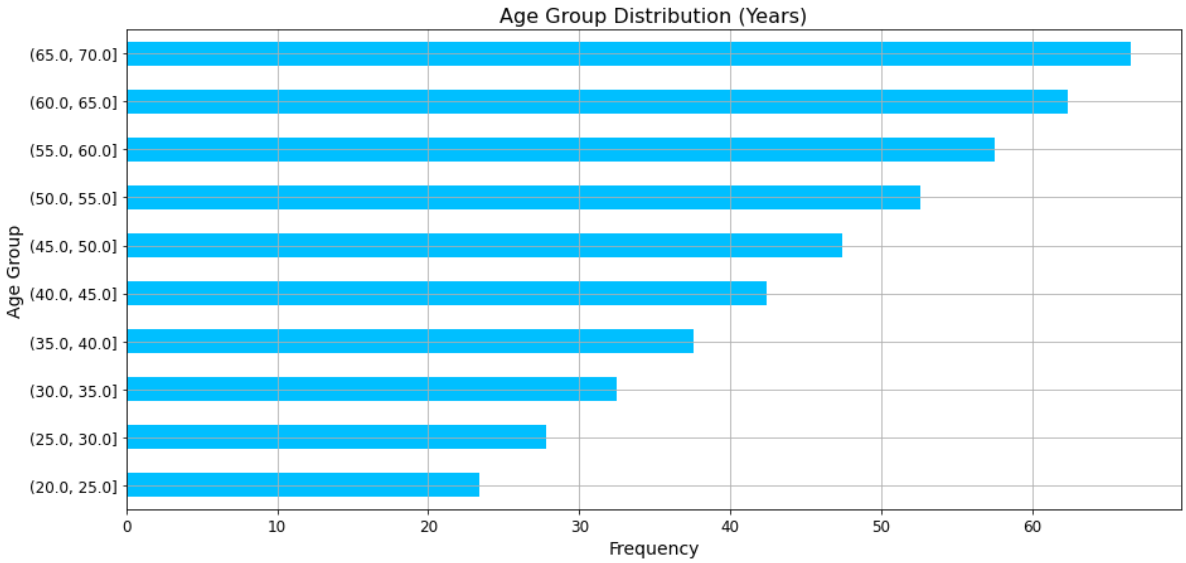
	TARGET	DAYS_BIRTH	YEARS_BIRTH	GROUPED_YEARS_BIRTH
0	1	-9461	25.920548	(25.0, 30.0]
1	0	-16765	45.931507	(45.0, 50.0]
2	0	-19046	52.180822	(50.0, 55.0]
3	0	-19005	52.068493	(50.0, 55.0]
4	0	-19932	54.608219	(50.0, 55.0]
5	0	-16941	46.413699	(45.0, 50.0]
6	0	-13778	37.747945	(35.0, 40.0]
7	0	-18850	51.643836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

In [ ]:
```
age_groups  = age_data.groupby('GROUPED_YEARS_BIRTH').mean()
age_groups
```

Out[ ]:

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

In [ ]:
```
age_groups['YEARS_BIRTH'].plot.barh(figsize=(15, 7), color='deepskyblue')
plt.title('Age Group Distribution (Years)', fontsize=16)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.xlabel('Frequency', fontsize=14)
plt.ylabel('Age Group', fontsize=14)
plt.grid(b=True)
plt.show()
```

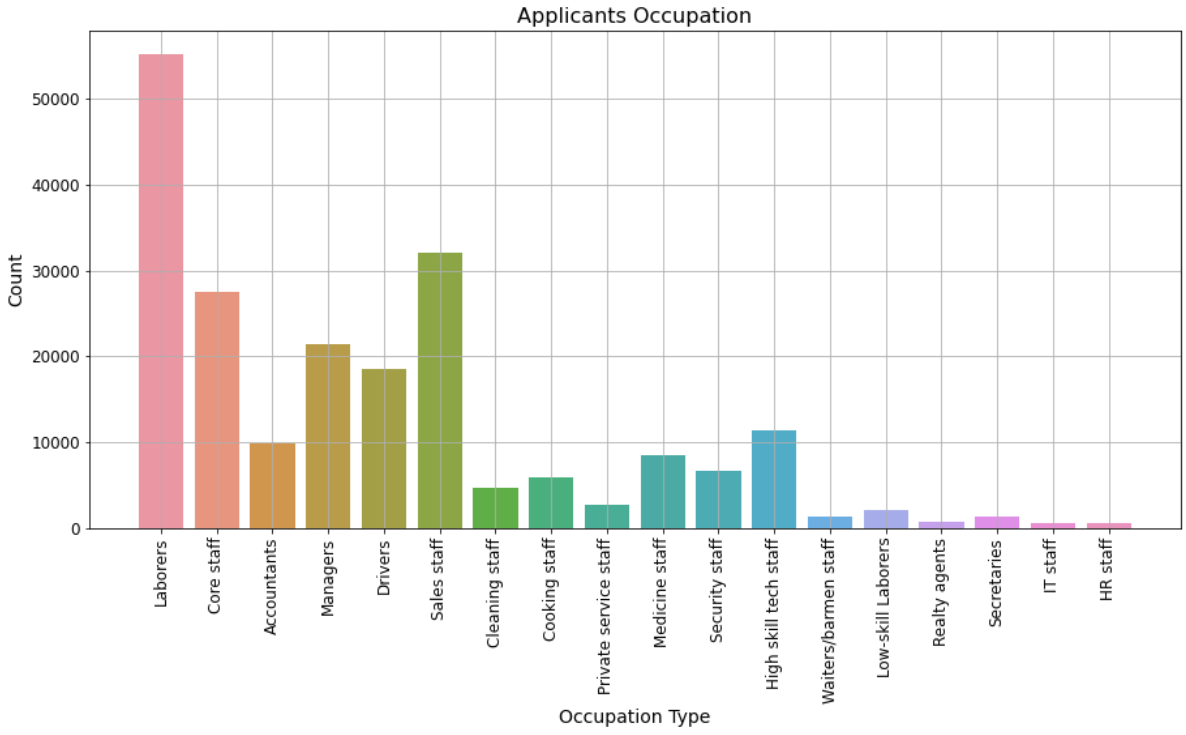


Observation:

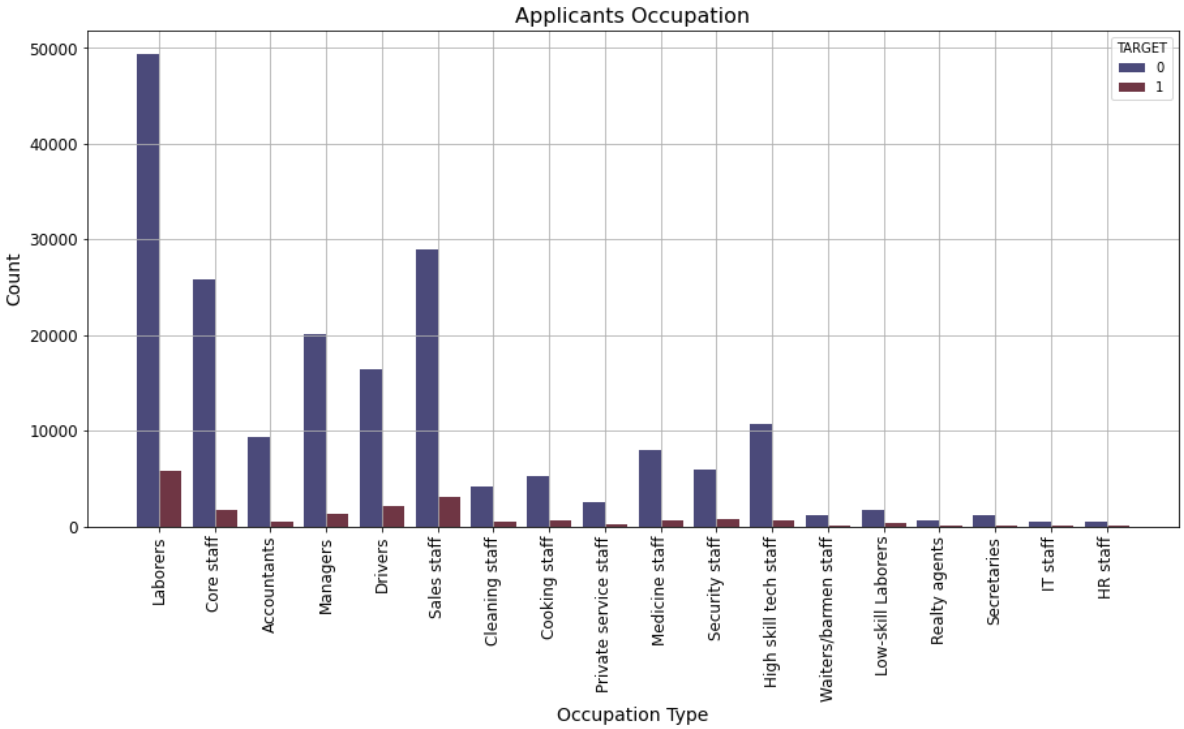
- After performing binning, we can observe that older people tend to take more loans than younger people.

Applicants occupations

In [ ]:
```
plt.figure(figsize=(15, 7))
sns.countplot(x='OCCUPATION_TYPE', data=datasets["application_train"])
plt.title('Applicants Occupation', fontsize=16)
plt.xlabel('Occupation Type', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.xticks(rotation=90, fontsize=12)
plt.yticks(fontsize=12)
plt.grid(b=True)
plt.plot();
```



In [ ]:
```
plt.figure(figsize=(15, 7))
sns.countplot(x='OCCUPATION_TYPE', data=datasets["application_train"], hue='TARGET
plt.title('Applicants Occupation', fontsize=16)
plt.xlabel('Occupation Type', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.xticks(rotation=90, fontsize=12)
plt.yticks(fontsize=12)
plt.grid(b=True)
plt.plot();
```



Observation:

- We see 18 different occupations among the borrowers, led by Laborers, Sales staff, Core staff, Managers and Drivers.
- We can't observe any specific trend as to which occupation class successfully pays back their loan.

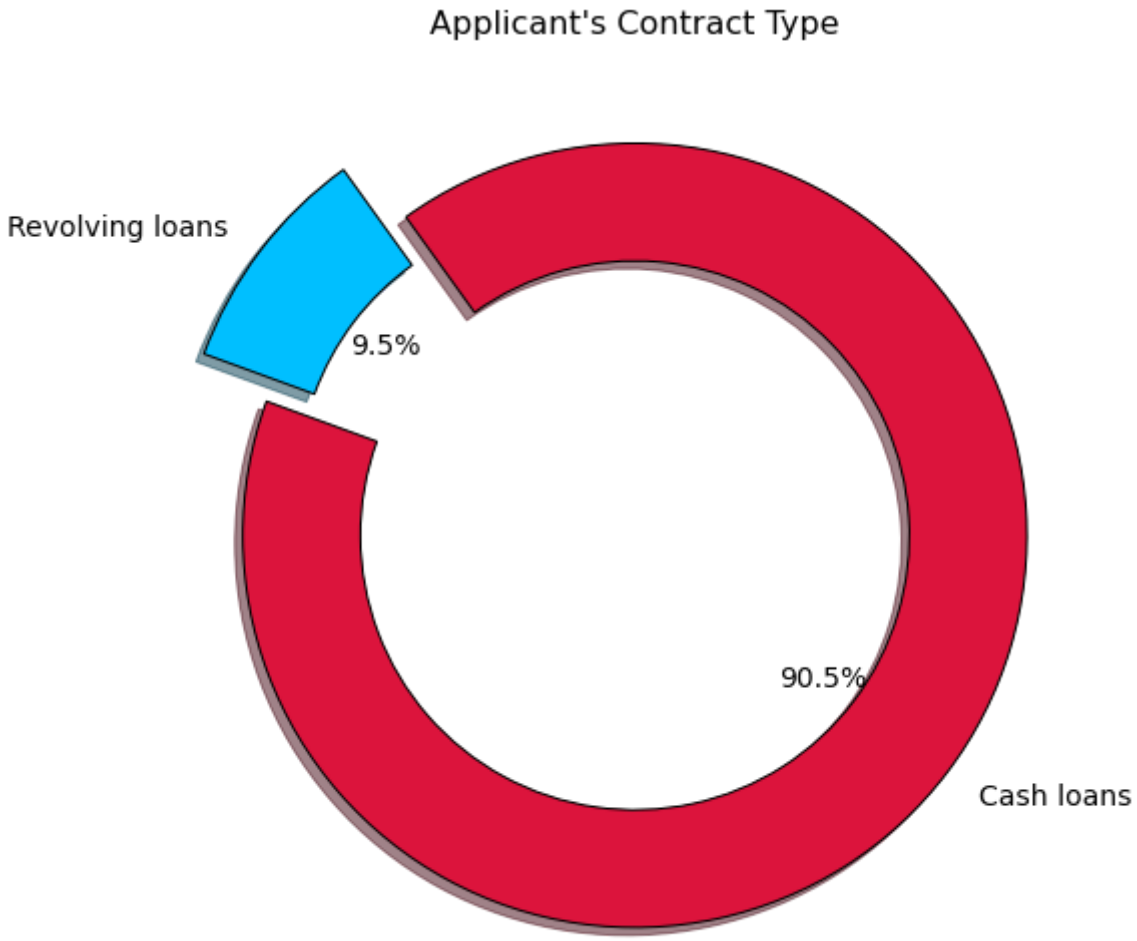
Applicant Contract Type

In [ ]:
```
app_train['NAME_CONTRACT_TYPE'].value_counts()
```

Out[ ]:
```
Cash loans         278232
Revolving loans     29279
Name: NAME_CONTRACT_TYPE, dtype: int64
```

In [ ]:
```
plt.figure(figsize=(9, 9))
plt.pie(x=app_train['NAME_CONTRACT_TYPE'].value_counts(),
        radius=1.3-0.3,
        labels=app_train['NAME_CONTRACT_TYPE'].value_counts().index,
        autopct='%1.1f%%',
        colors=['crimson', 'deepskyblue'],
        explode=[0,0.2],
        wedgeprops={"edgecolor":"0", "width":0.3},
        startangle=160,
```

```
        shadow=True,
        textprops={'fontsize': 14})
plt.ylabel('', fontsize=14)
plt.title("Applicant's Contract Type", fontsize=16)
plt.show()
```



Observation:

- There are two type of loan contracts - Cash loans (90.5%) and Revolving (re-pay and re-borrow again and again) loans (9.5%)

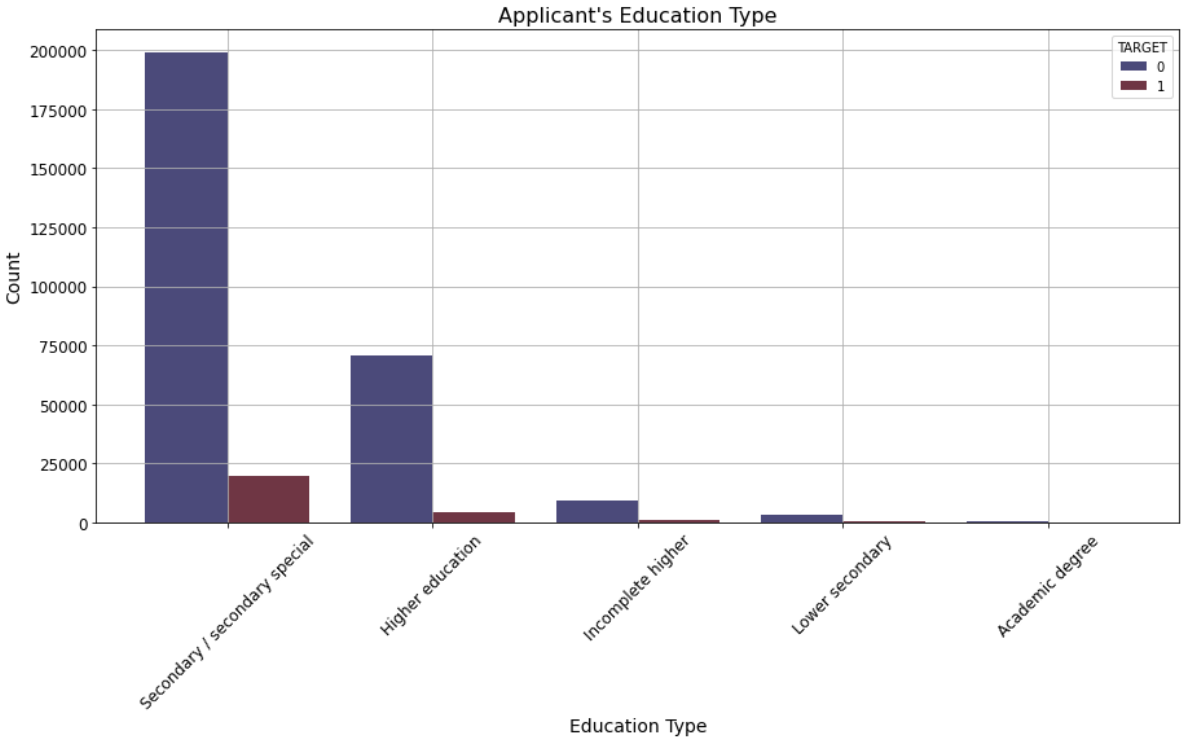
Applicant Education Type v/s TARGET

In [ ]:
```
app_train['NAME_EDUCATION_TYPE'].value_counts()
```

Out[ ]:
```
Secondary / secondary special    218391
Higher education                  74863
Incomplete higher                 10277
Lower secondary                    3816
Academic degree                     164
Name: NAME_EDUCATION_TYPE, dtype: int64
```

In [ ]:
```
plt.figure(figsize=(15, 7))
sns.countplot(x='NAME_EDUCATION_TYPE', data=app_train, palette='icefire', hue='TAR
plt.title("Applicant's Education Type", fontsize=16)
plt.xlabel('Education Type', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.xticks(rotation=45, fontsize=12)
```

```
plt.yticks(fontsize=12)
plt.grid(b=True)
plt.plot();
```



Observation:

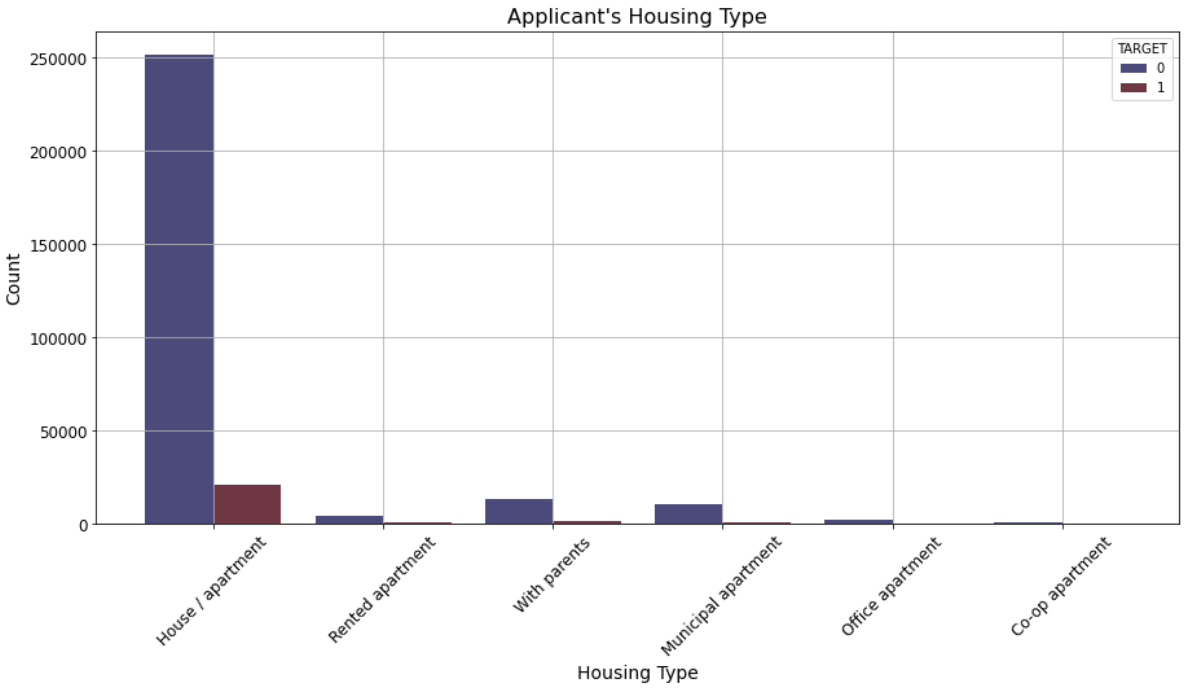
- We see most of the applicants have highest education of secondary and higher education with the lowest being Academic degree.

Applicant Housing Type v/s TARGET

In [ ]:
```
app_train['NAME_HOUSING_TYPE'].value_counts()
```

Out[ ]:
```
House / apartment      272868
With parents            14840
Municipal apartment     11183
Rented apartment         4881
Office apartment         2617
Co-op apartment          1122
Name: NAME_HOUSING_TYPE, dtype: int64
```

In [ ]:
```
plt.figure(figsize=(15, 7))
sns.countplot(x='NAME_HOUSING_TYPE', data=app_train, palette='icefire', hue='TARGE
plt.title("Applicant's Housing Type", fontsize=16)
plt.xlabel('Housing Type', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.xticks(rotation=45, fontsize=12)
plt.yticks(fontsize=12)
plt.grid(b=True)
plt.plot();
```

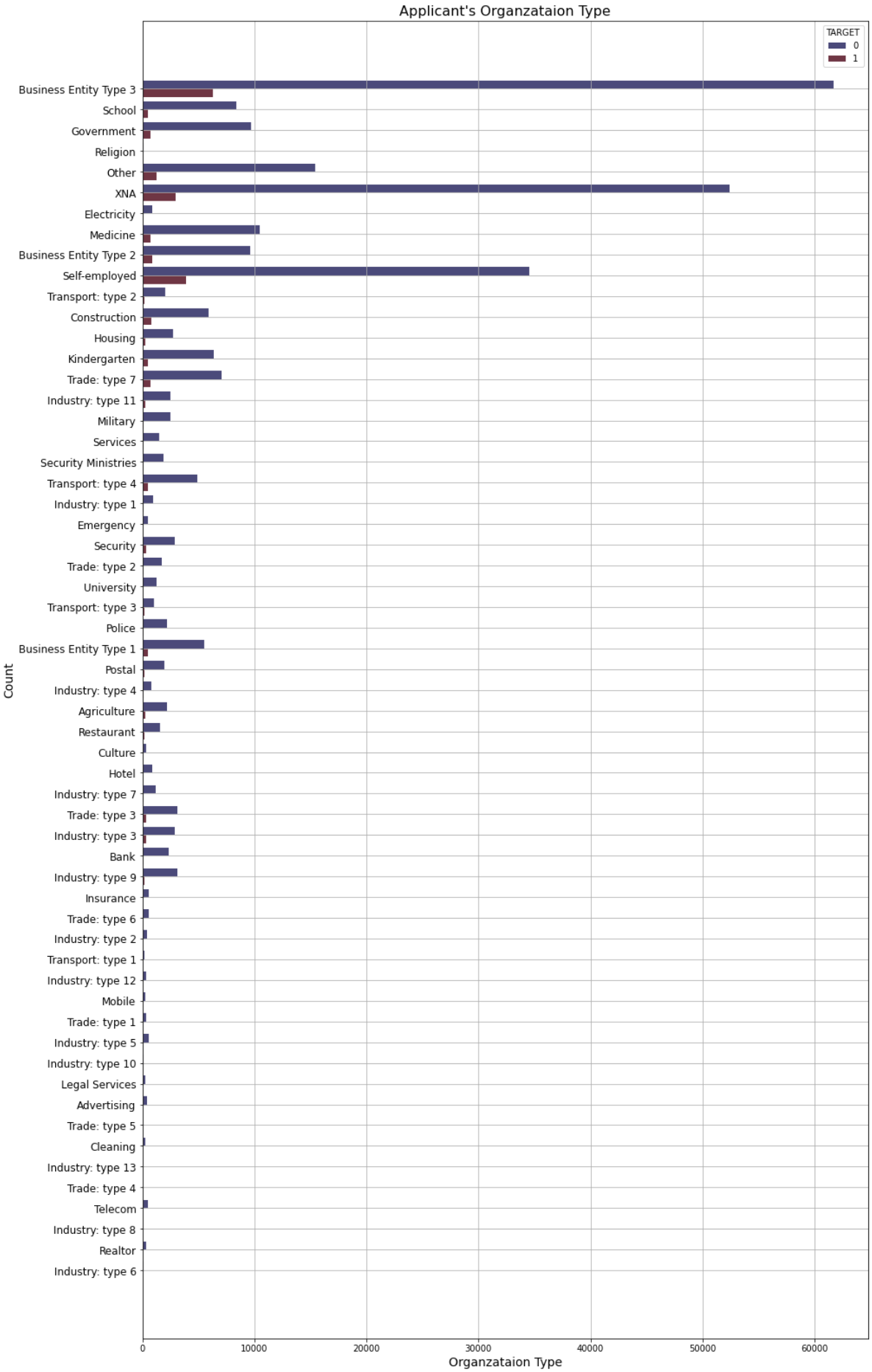


Observation:

- Most of the applicant's current house type is a house or an apartment followed by living with parents (searching for a new home for themselves) and municipal apartment.

Applicant Organization Type

In [ ]:
```
plt.figure(figsize=(15, 28))
sns.countplot(y='ORGANIZATION_TYPE', data=app_train, palette='icefire', hue='TARGE
plt.title("Applicant's Organzataion Type", fontsize=16)
plt.xlabel('Organzataion Type', fontsize=14)
plt.ylabel('Count', fontsize=14)
# plt.xticks(rotation=45, fontsize=12)
plt.yticks(fontsize=12)
plt.grid(b=True)
plt.plot();
```



Observation:

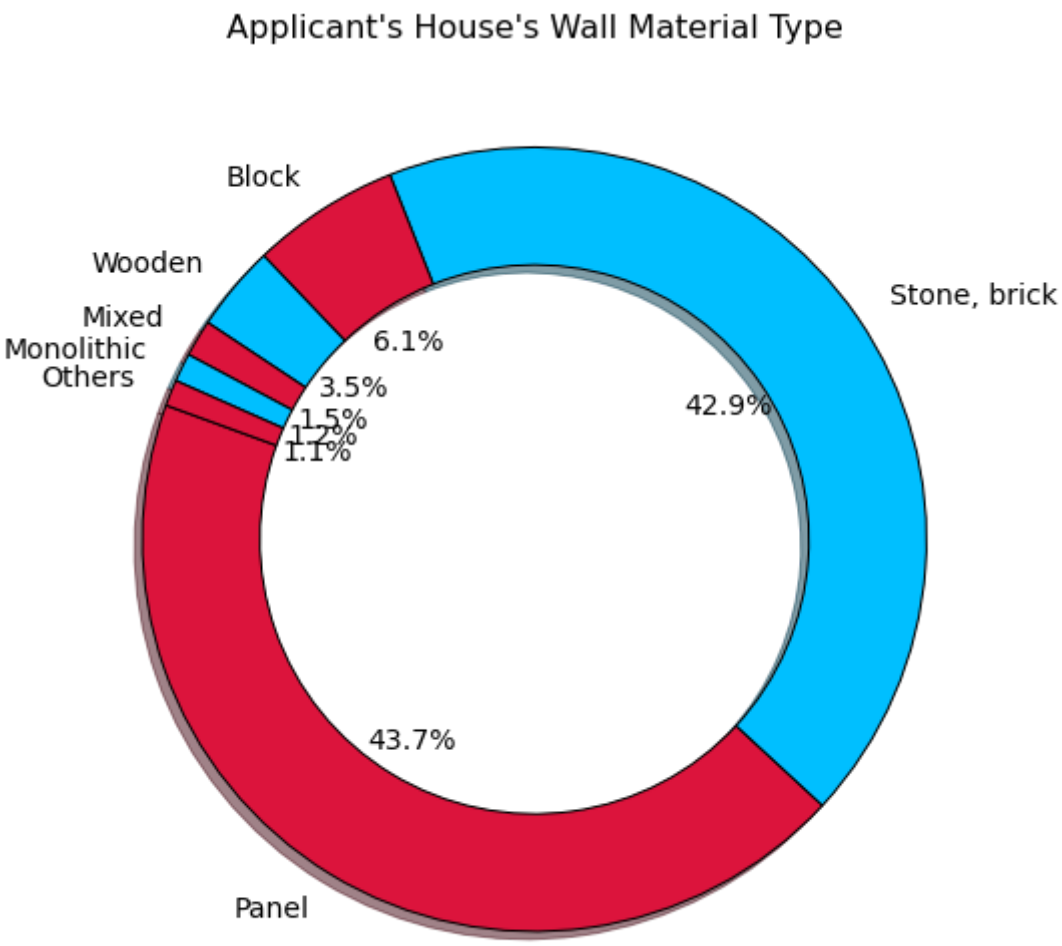
- There are several organization types across various applicants predominantly from Type 3 business entities and self-employement.

Applicant's House Wall Material Type

In [ ]:
```
app_train['WALLSMATERIAL_MODE'].value_counts()
```

Out[ ]:
```
Panel           66040
Stone, brick    64815
Block            9253
Wooden           5362
Mixed            2296
Monolithic       1779
Others           1625
Name: WALLSMATERIAL_MODE, dtype: int64
```

In [ ]:
```
plt.figure(figsize=(9, 9))
plt.pie(x=app_train['WALLSMATERIAL_MODE'].value_counts(),
        radius=1.3-0.3,
        labels=app_train['WALLSMATERIAL_MODE'].value_counts().index,
        autopct='%1.1f%%',
        colors=['crimson', 'deepskyblue'],
        # explode=[0.2,0.2,0,0.2,0,0.2,0],
        wedgeprops={"edgecolor":"0", "width":0.3},
        startangle=160,
        shadow=True,
        textprops={'fontsize': 14})
plt.ylabel('', fontsize=14)
plt.title("Applicant's House's Wall Material Type", fontsize=16)
plt.show()
```



Observation:

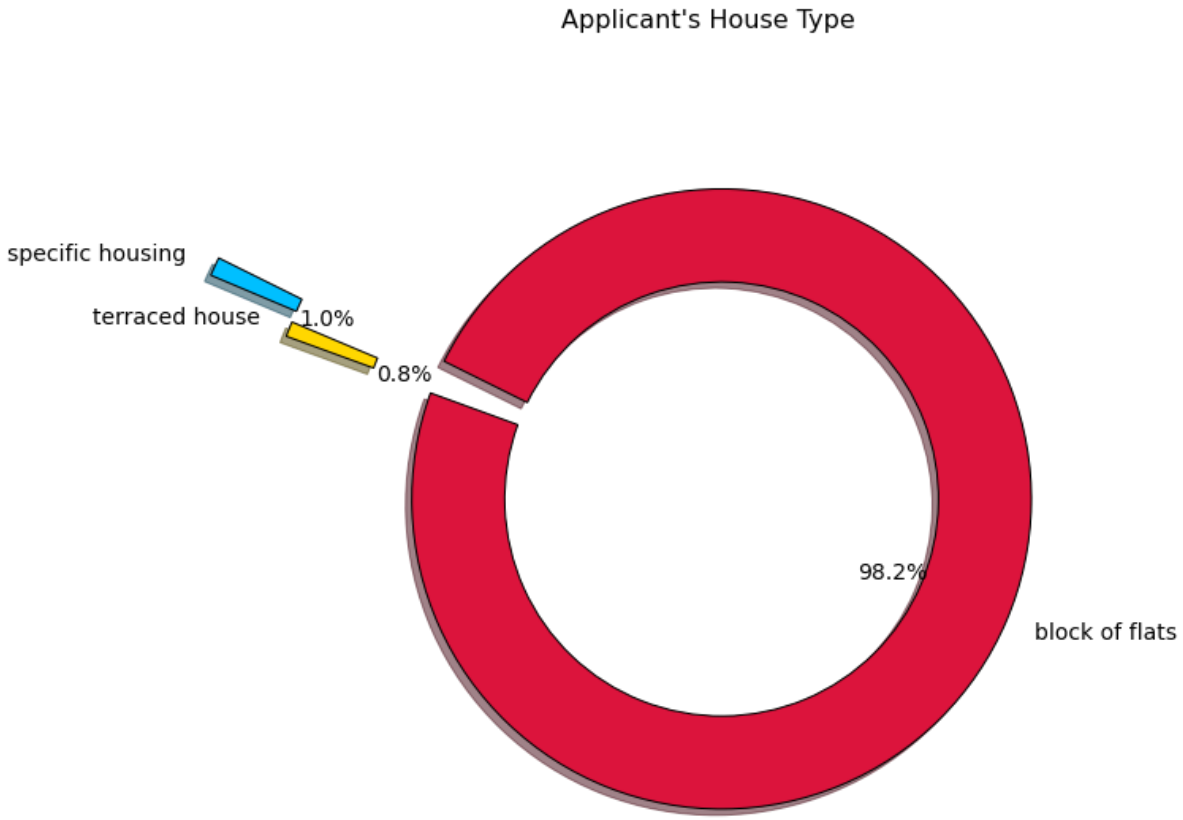
- Most Applicant's house's wall material are made of panels or stones and bricks, followed by cement blocks, wood or a mix of the earlier mentioned.

Applicant's House Type Part 2

In [ ]:
```
app_train['HOUSETYPE_MODE'].value_counts()
```

Out[ ]:
```
block of flats      150503
specific housing      1499
terraced house        1212
Name: HOUSETYPE_MODE, dtype: int64
```

In [ ]:
```
plt.figure(figsize=(9, 9))
plt.pie(x=app_train['HOUSETYPE_MODE'].value_counts(),
        radius=1.3-0.3,
        labels=app_train['HOUSETYPE_MODE'].value_counts().index,
        autopct='%1.1f%%',
        colors=['crimson', 'deepskyblue', 'gold'],
        explode=[0,0.8,0.5],
        wedgeprops={"edgecolor":"0", "width":0.3},
        startangle=160,
        shadow=True,
        textprops={'fontsize': 14})
plt.ylabel('', fontsize=14)
plt.suptitle("Applicant's House Type", fontsize=16)
plt.show()
```



Observation:

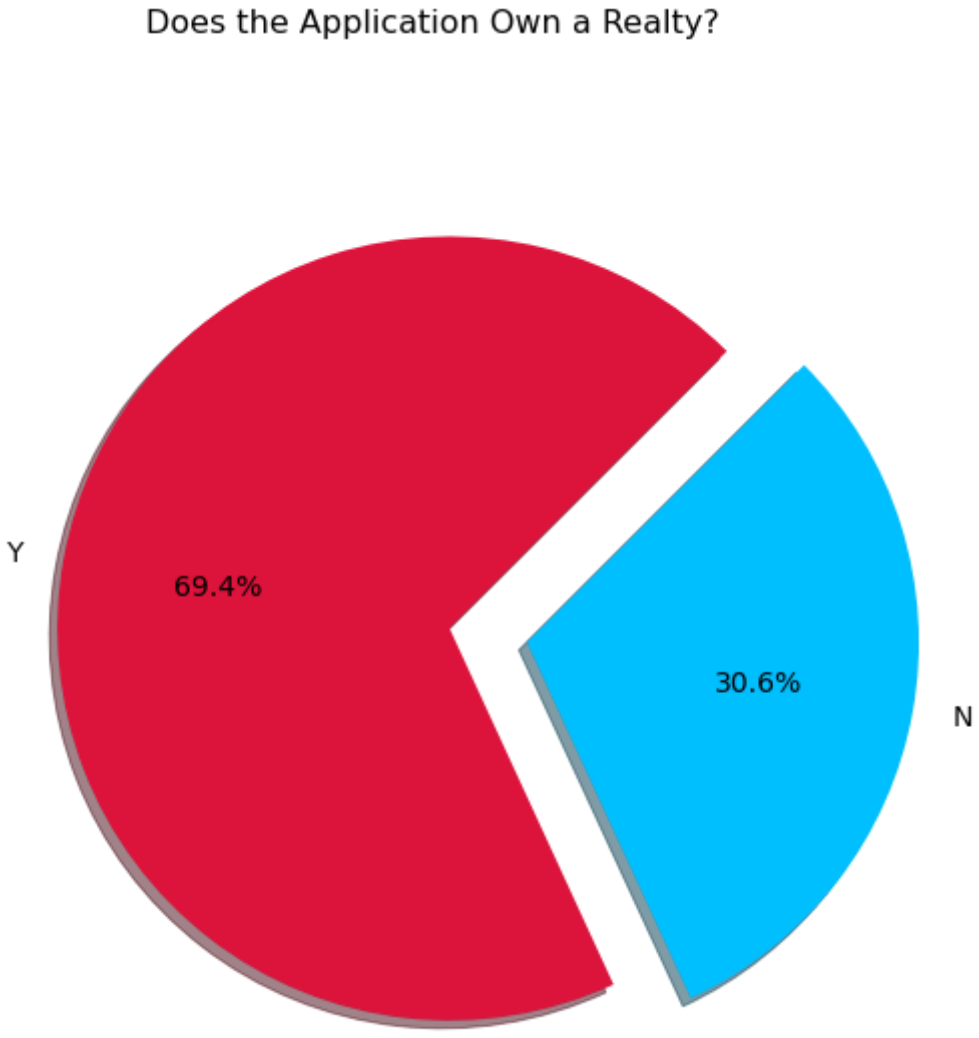
- Applicants mostly reside in flats (more than 98%) while the remaining either live in terraced or other specific house types.

Does an Applicant already own Realty?

In [ ]:
```
app_train['FLAG_OWN_REALTY'].value_counts()
```

Out[ ]:
```
Y    213312
N     94199
Name: FLAG_OWN_REALTY, dtype: int64
```

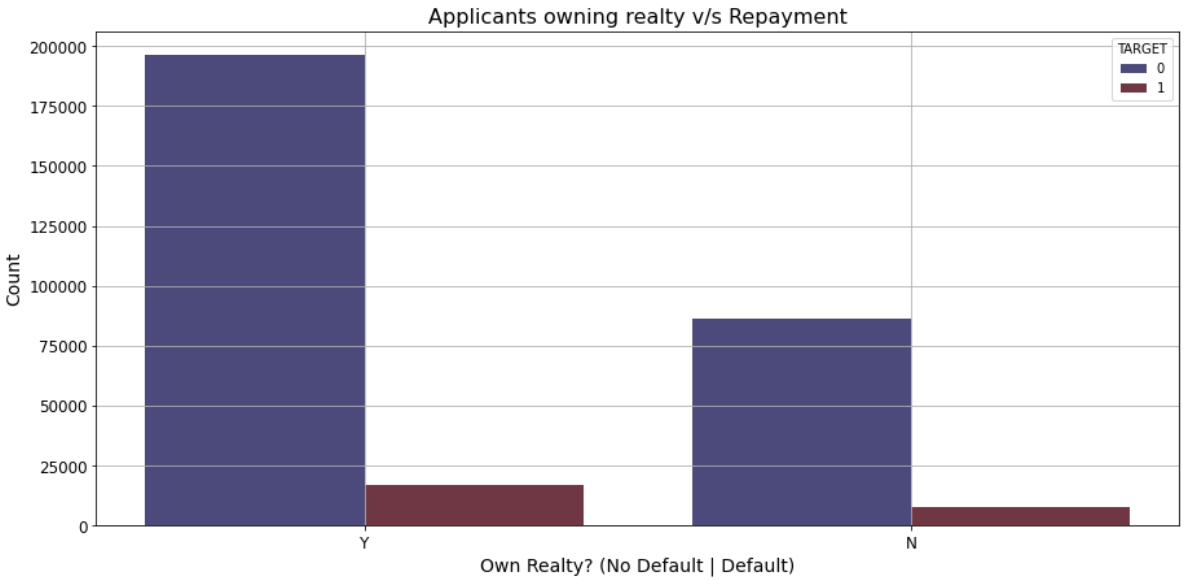
In [ ]:
```
plt.figure(figsize=(9, 9))
plt.pie(x=app_train['FLAG_OWN_REALTY'].value_counts(),
        radius=1.3-0.3,
        labels=app_train['FLAG_OWN_REALTY'].value_counts().index,
        autopct='%1.1f%%',
        colors=['crimson', 'deepskyblue'],
        explode=[0,0.2],
        # wedgeprops={"edgecolor":"0", "width":0.3},
        startangle=45,
        shadow=True,
        textprops={'fontsize': 14})
plt.ylabel('', fontsize=14)
plt.suptitle("Does the Application Own a Realty?", fontsize=16)
plt.show()
```



Observation:

- 69.4% Applicants already own a realty. Let's see the distribution of the repayment.

In [ ]:
```
plt.figure(figsize=(15, 7))
sns.countplot(x='FLAG_OWN_REALTY', data=app_train, palette='icefire', hue='TARGET'
plt.title("Applicants owning realty v/s Repayment", fontsize=16)
plt.xlabel('Own Realty? (No Default | Default)', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.grid(b=True)
plt.plot();
```



Observation:

- Most of the applicants in either class are not in default. Less than 25000 applicants own a realty and are in default for their repayment.

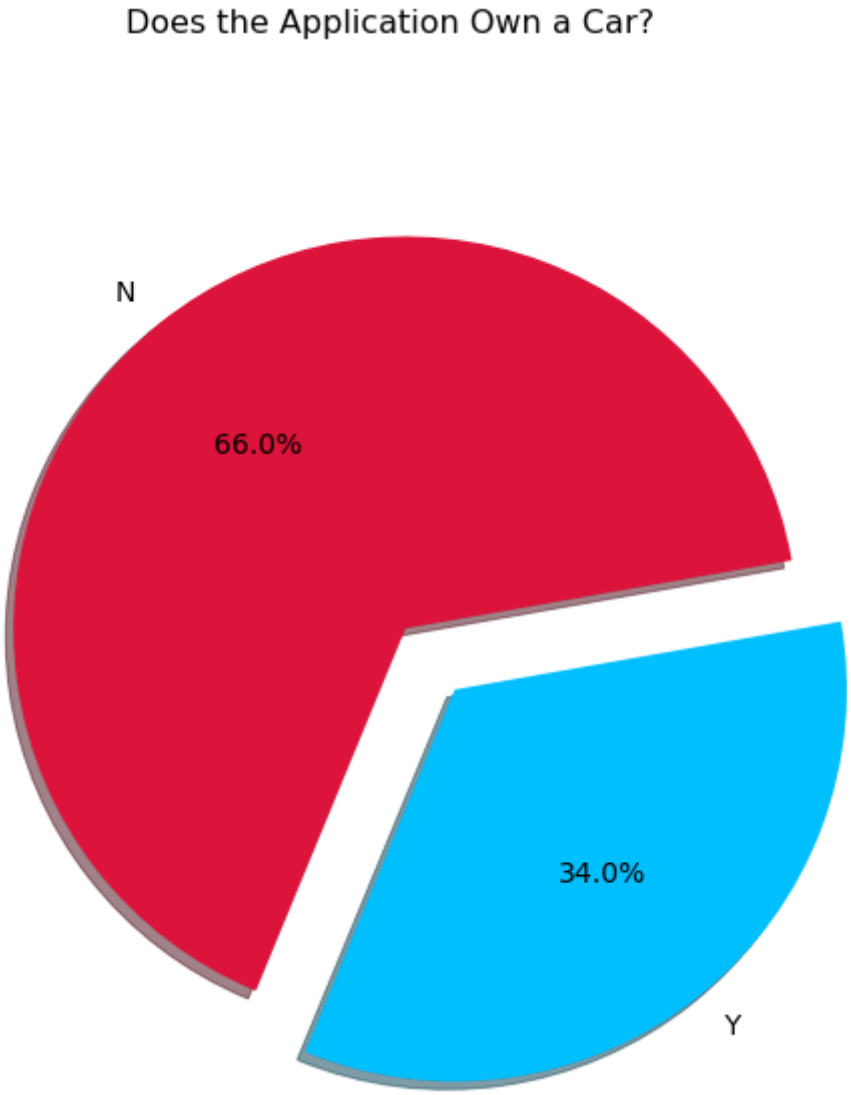
Does an Applicant own cars?

In [ ]:
```
app_train['FLAG_OWN_CAR'].value_counts()
```

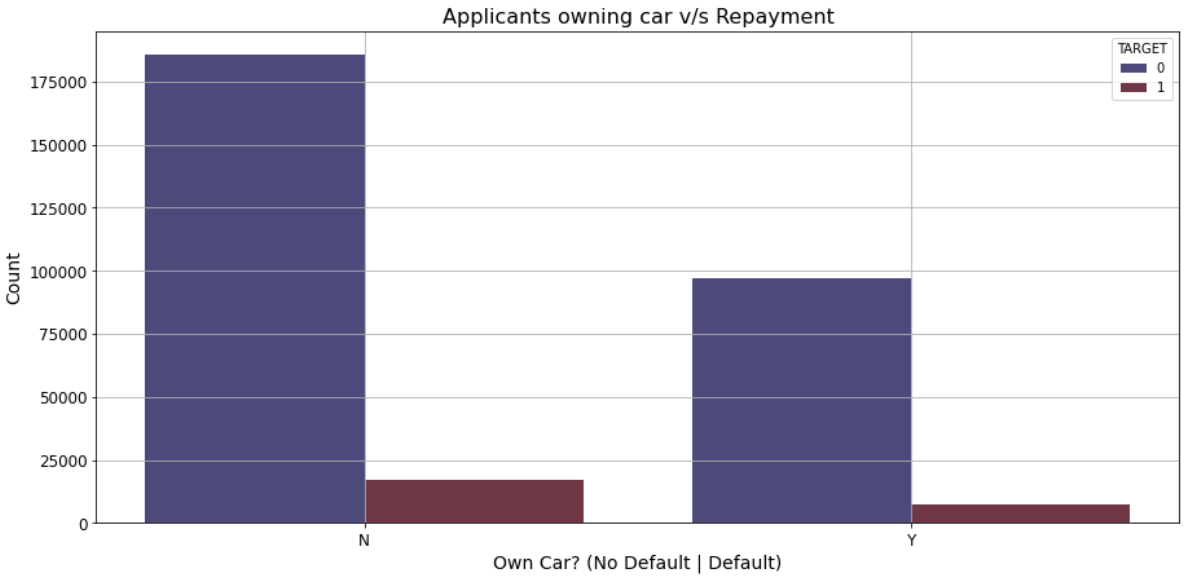
Out[ ]:
```
N    202924
Y    104587
Name: FLAG_OWN_CAR, dtype: int64
```

In [ ]:
```
plt.figure(figsize=(9, 9))
plt.pie(x=app_train['FLAG_OWN_CAR'].value_counts(),
        radius=1.3-0.3,
        labels=app_train['FLAG_OWN_CAR'].value_counts().index,
        autopct='%1.1f%%',
        colors=['crimson', 'deepskyblue'],
        explode=[0,0.2],
        # wedgeprops={"edgecolor":"0", "width":0.3},
        startangle=10,
        shadow=True,
        textprops={'fontsize': 14})
plt.ylabel('', fontsize=14)
```

```
plt.suptitle("Does the Application Own a Car?", fontsize=16)
plt.show()
```



In [ ]:
```
plt.figure(figsize=(15, 7))
sns.countplot(x='FLAG_OWN_CAR', data=app_train, palette='icefire', hue='TARGET')
plt.title("Applicants owning car v/s Repayment", fontsize=16)
plt.xlabel('Own Car? (No Default | Default)', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.grid(b=True)
plt.plot();
```

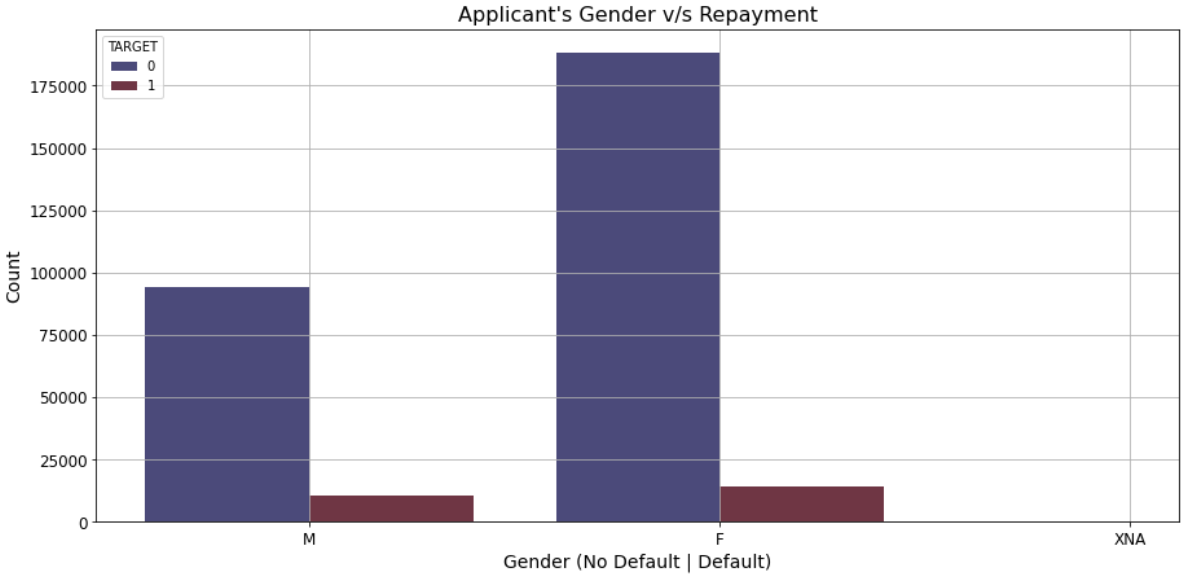


Observation:

- 34% applicants own atleast one car.
- No specific relation to be noted here when it comes to finding a relation between car ownership and loan repayment.

Which Gender seems more likely to take and re-pay loans?

In [ ]:
```
plt.figure(figsize=(15, 7))
sns.countplot(x='CODE_GENDER', data=app_train, palette='icefire', hue='TARGET')
plt.title("Applicant's Gender v/s Repayment", fontsize=16)
plt.xlabel('Gender (No Default | Default)', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.grid(b=True)
plt.plot();
```



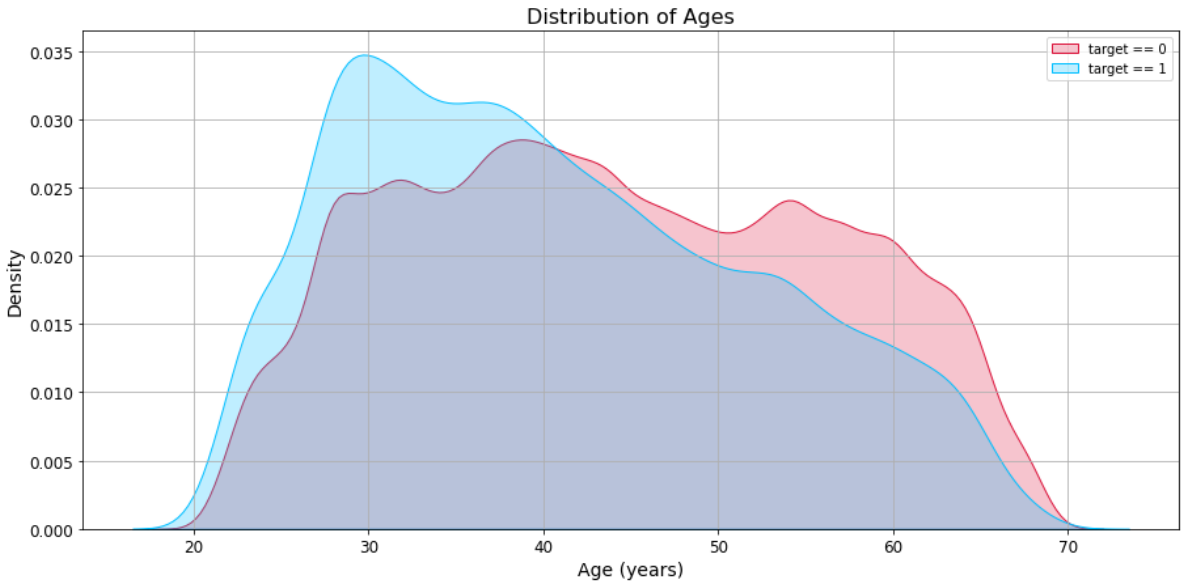
Observation:

- Most of the applicants are females and most of them have no default in their history.
- In case of male applicants, we can see that relatively higher number of applicants are in default.

Exploratory Data Analysis on Numeric/Continuous Features

Target v/s Age (in years)

In [ ]:
```
plt.figure(figsize = (15, 7))

# KDE plot of loans that were repaid on time
sns.kdeplot(app_train.loc[app_train['TARGET'] == 0, 'DAYS_BIRTH'] / -365, label =

# KDE plot of loans which were not repaid on time
sns.kdeplot(app_train.loc[app_train['TARGET'] == 1, 'DAYS_BIRTH'] / -365, label =

# Labeling of plot
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)
plt.xlabel('Age (years)', fontsize=14)
plt.ylabel('Density', fontsize=14)
plt.title('Distribution of Ages', fontsize=16)

plt.legend()
plt.grid(b=True)
plt.show()
```

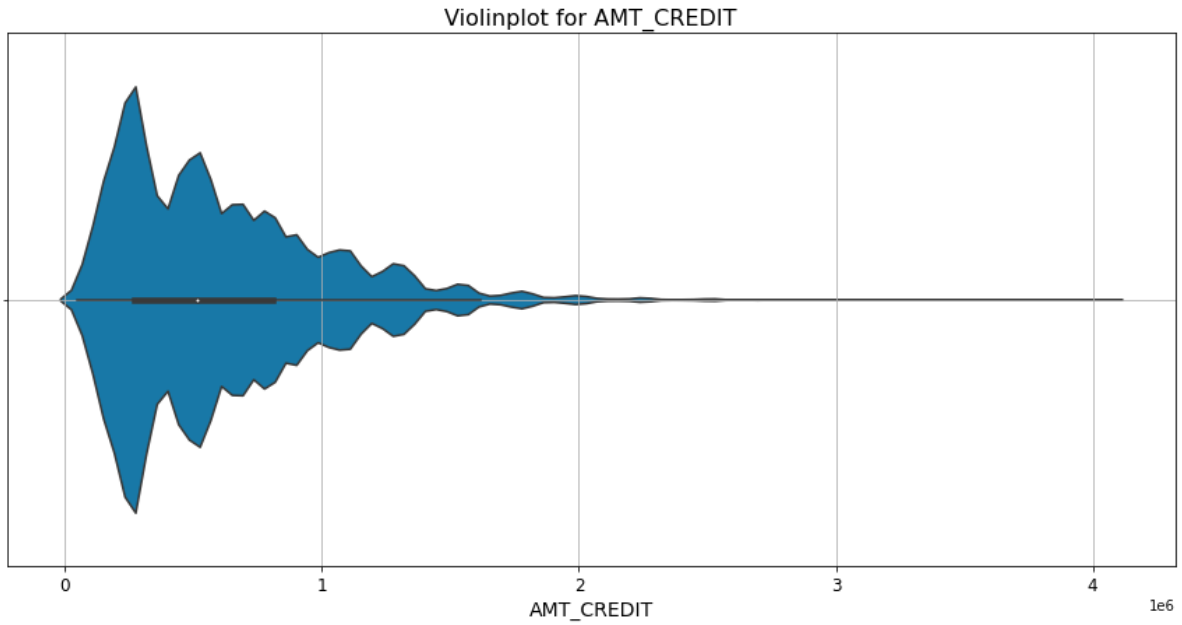


Observation

- We can observe a skew of defaults towards the younger applicants.
- This indicates that older applicants repaid their loans in a more timely/efficient manner.

Checking the distribution of AMT_CREDIT feature

In [ ]:
```
plt.figure(figsize=(15, 7))
sns.violinplot(x=app_train['AMT_CREDIT'], palette='winter')
plt.xticks(size=12)
plt.yticks(size=12)
plt.xlabel('AMT_CREDIT', size=14)
plt.title('Violinplot for AMT_CREDIT', size=16)
plt.grid(b=True)
```

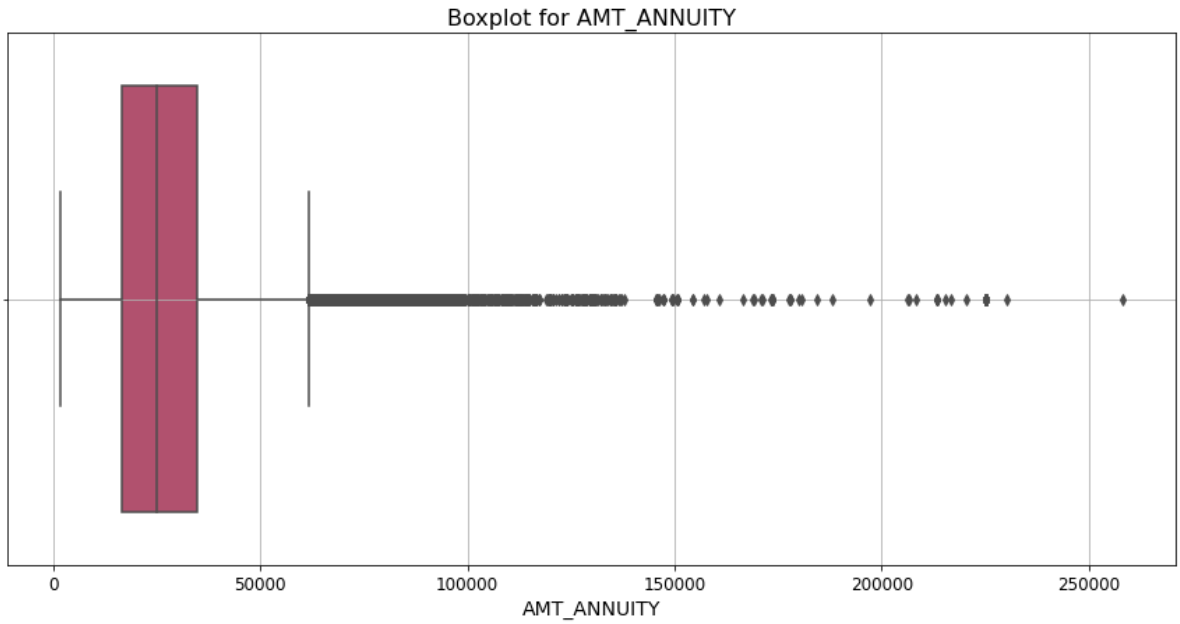


Observation

- We can observe that the feature is right skewed.
- Scaling might help us use this feature appropriately

Checking the distribution of AMT_ANNUITY feature

In [ ]:
```
plt.figure(figsize=(15, 7))
sns.boxplot(x=app_train['AMT_ANNUITY'], palette='flare')
plt.xticks(size=12)
plt.yticks(size=12)
plt.xlabel('AMT_ANNUITY', size=14)
plt.title('Boxplot for AMT_ANNUITY', size=16)
plt.grid(b=True)
```

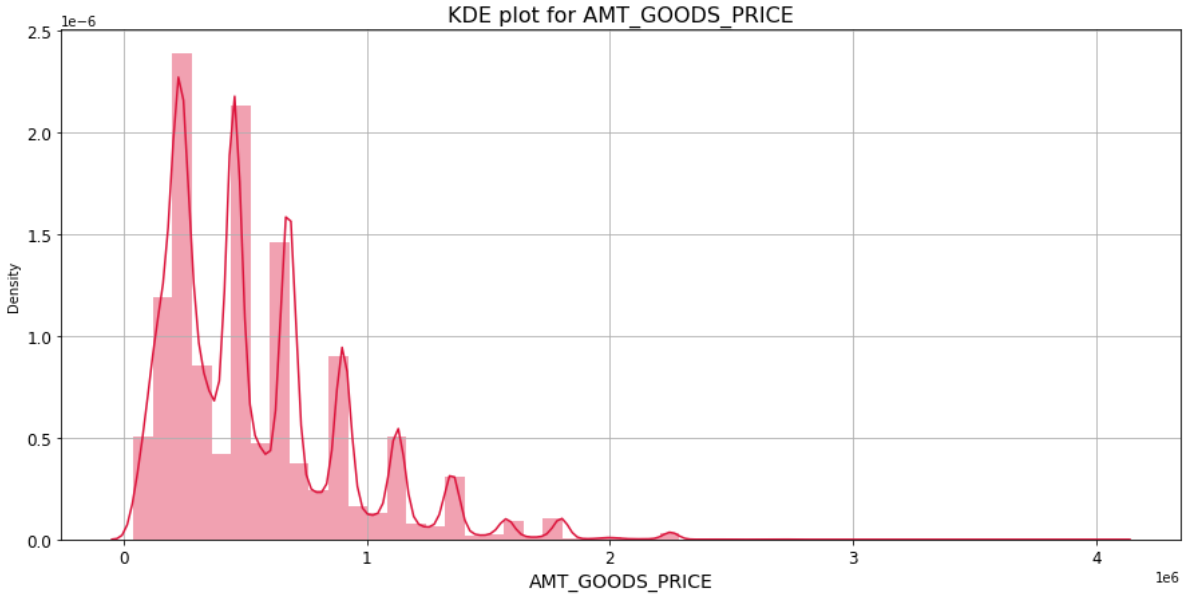


Observation

- We observe yet again a right skewed feature with a lot of outliers.
- We can't remove these outliers since we might lose important information.

Checking the distribution of AMT_GOODS_PRICE feature

In [ ]:
```
plt.figure(figsize=(15, 7))
sns.distplot(x=app_train['AMT_GOODS_PRICE'], color='crimson')
plt.xticks(size=12)
plt.yticks(size=12)
plt.xlabel('AMT_GOODS_PRICE', size=14)
plt.title('KDE plot for AMT_GOODS_PRICE', size=16)
plt.grid(b=True)
```



Observation

- We see yet another skewed distribution which is multi-modal in nature.

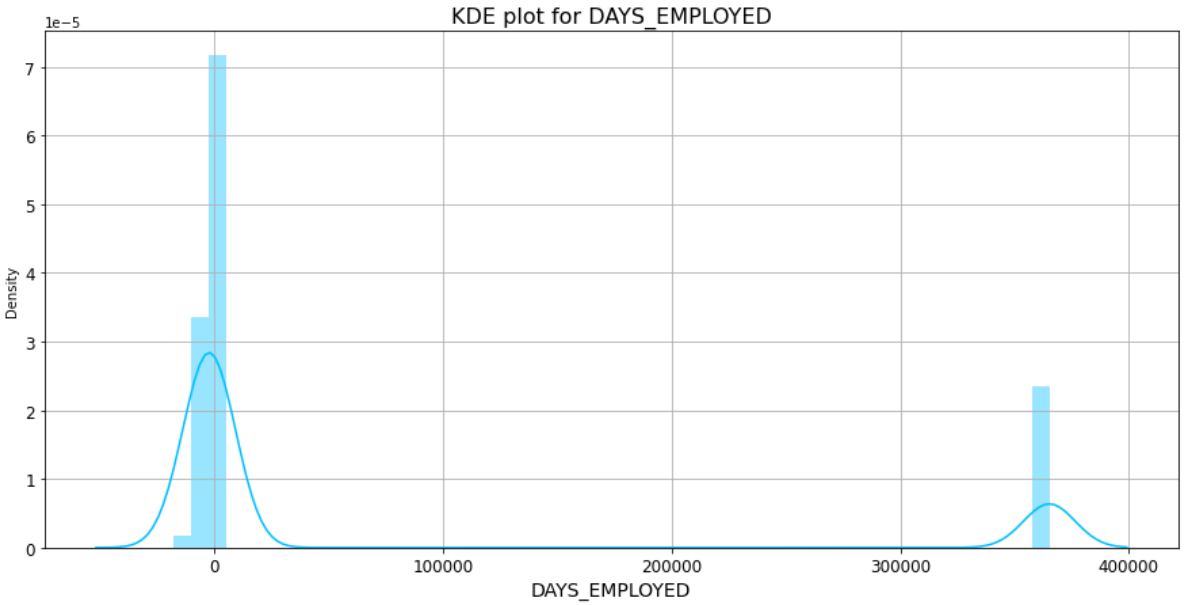
- Binning might help to make an efficient use of this feature.

Checking the distribution of DAYS_EMPLOYED feature

In [ ]:
```
app_train['DAYS_EMPLOYED'].describe()
```

Out[ ]:
```
count    307511.000000
mean      63815.045904
std      141275.766519
min      -17912.000000
25%       -2760.000000
50%       -1213.000000
75%        -289.000000
max      365243.000000
Name: DAYS_EMPLOYED, dtype: float64
```

In [ ]:
```
plt.figure(figsize=(15, 7))
sns.distplot(x=app_train['DAYS_EMPLOYED'], color='deepskyblue')
plt.xticks(size=12)
plt.yticks(size=12)
plt.xlabel('DAYS_EMPLOYED', size=14)
plt.title('KDE plot for DAYS_EMPLOYED', size=16)
plt.grid(b=True)
```

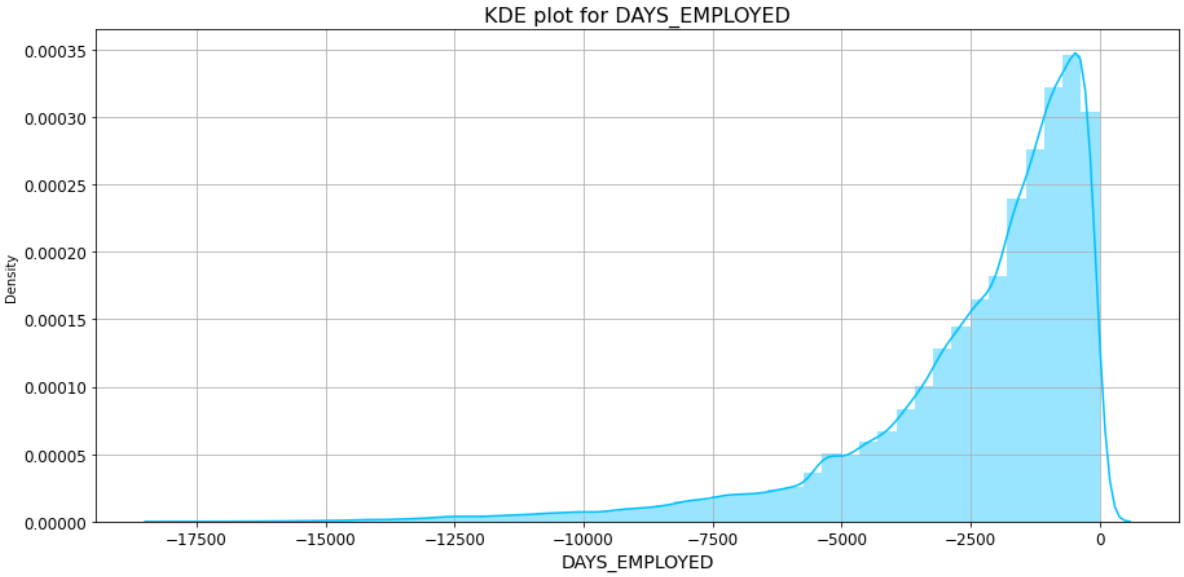


Observation

- Just like DAYS_BIRTH, this feature has negative days values.
- But we observe a weird anomaly here - max days employed is 365243 days which is a thousand years.
- We will simply ignore this anomaly (replace with appropriate values) and check the distribution of the feature again.

Checking the distribution of DAYS_EMPLOYED feature after removing the inconsistent value

In [ ]:
```
days_employed = app_train['DAYS_EMPLOYED']
days_employed = days_employed[days_employed<365243]
plt.figure(figsize=(15, 7))
sns.distplot(x=days_employed, color='deepskyblue')
plt.xticks(size=12)
plt.yticks(size=12)
plt.xlabel('DAYS_EMPLOYED', size=14)
plt.title('KDE plot for DAYS_EMPLOYED', size=16)
plt.grid(b=True)
```



Observation

- We observe a left skewed data in this plot which in turn would form a right skewed distribution if we flip the days to the positive side.

Fixing DAYS_EMPLOYES and DAYS_BIRTH features

In [ ]:
```
app_train['DAYS_BIRTH'] = app_train['DAYS_BIRTH'] / -1
app_test['DAYS_BIRTH'] = app_test['DAYS_BIRTH'] / -1

app_train['DAYS_EMPLOYED'] = app_train['DAYS_EMPLOYED'][app_train['DAYS_EMPLOYED']
app_test['DAYS_EMPLOYED'] = app_test['DAYS_EMPLOYED'][app_test['DAYS_EMPLOYED']<36
app_train['DAYS_EMPLOYED'] = app_train['DAYS_EMPLOYED']/-1
app_test['DAYS_EMPLOYED'] = app_test['DAYS_EMPLOYED']/-1

app_train['DAYS_BIRTH'].head()
```

Out[ ]:
```
0    -9461.0
1   -16765.0
2   -19046.0
3   -19005.0
4   -19932.0
Name: DAYS_BIRTH, dtype: float64
```

In [ ]:
```
app_test['DAYS_BIRTH'].head()
```

Out[ ]:
```
0   -19241.0
1   -18064.0
2   -20038.0
3   -13976.0
4   -13040.0
Name: DAYS_BIRTH, dtype: float64
```

In [ ]:
```
app_train['DAYS_EMPLOYED'].head()
```

Out[ ]:
```
0    -637.0
1   -1188.0
2    -225.0
3   -3039.0
4   -3038.0
Name: DAYS_EMPLOYED, dtype: float64
```

In [ ]:
```
app_test['DAYS_EMPLOYED'].head()
```

Out[ ]:
```
0   -2329.0
1   -4469.0
2   -4458.0
3   -1866.0
4   -2191.0
Name: DAYS_EMPLOYED, dtype: float64
```

Dataset questions

Unique record for each SK_ID_CURR

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

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

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

Out[ ]:
```
True
```

In [ ]:
```
# is there an overlap between the test and train customers
np.intersect1d(datasets["application_train"]["SK_ID_CURR"], datasets["application_
```

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

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

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

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

Out[ ]:
```
(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 [ ]:
```
appsDF = pd.read_csv('/root/shared/AML/I526_AML_Student/Assignments/Unit-Project-Ho
display(appsDF.head())
print(f"{appsDF.shape[0]:,} rows, {appsDF.shape[1]:,} columns")
```

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CI
0	2030495	271877	Consumer loans	1730.430	17145.0	17
1	2802425	108129	Cash loans	25188.615	607500.0	679
2	2523466	122040	Cash loans	15060.735	112500.0	136
3	2819243	176158	Cash loans	47041.335	450000.0	470
4	1784265	202054	Cash loans	31924.395	337500.0	404

5 rows × 37 columns

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

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

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

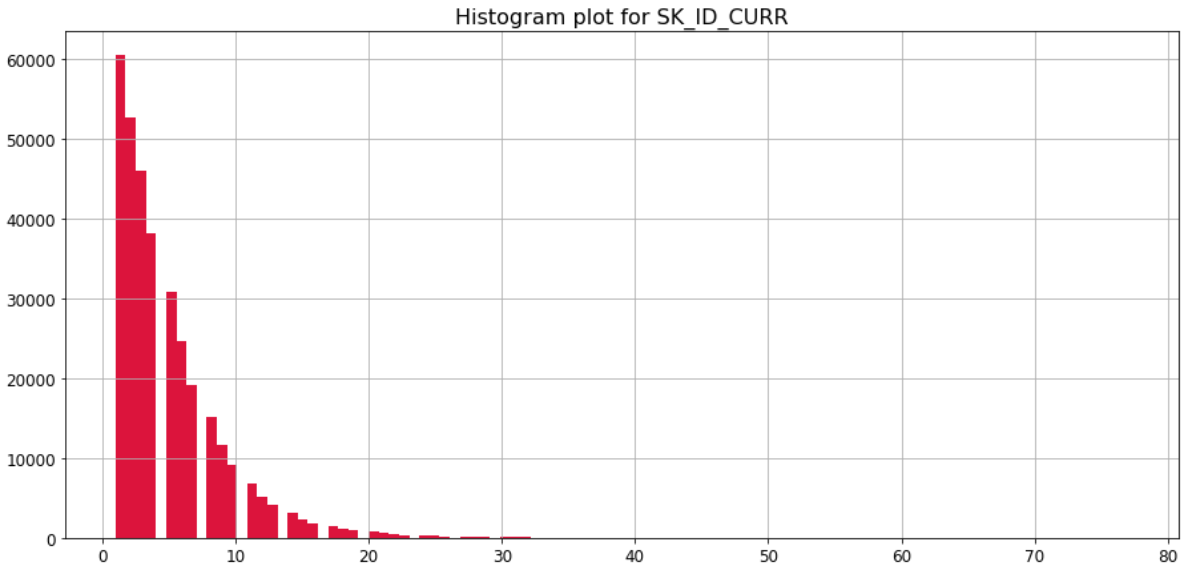
In [ ]:
```
#Find the intersection of two arrays.
print(f'Number of train applicants with previous applications is {len(np.intersect1
```

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

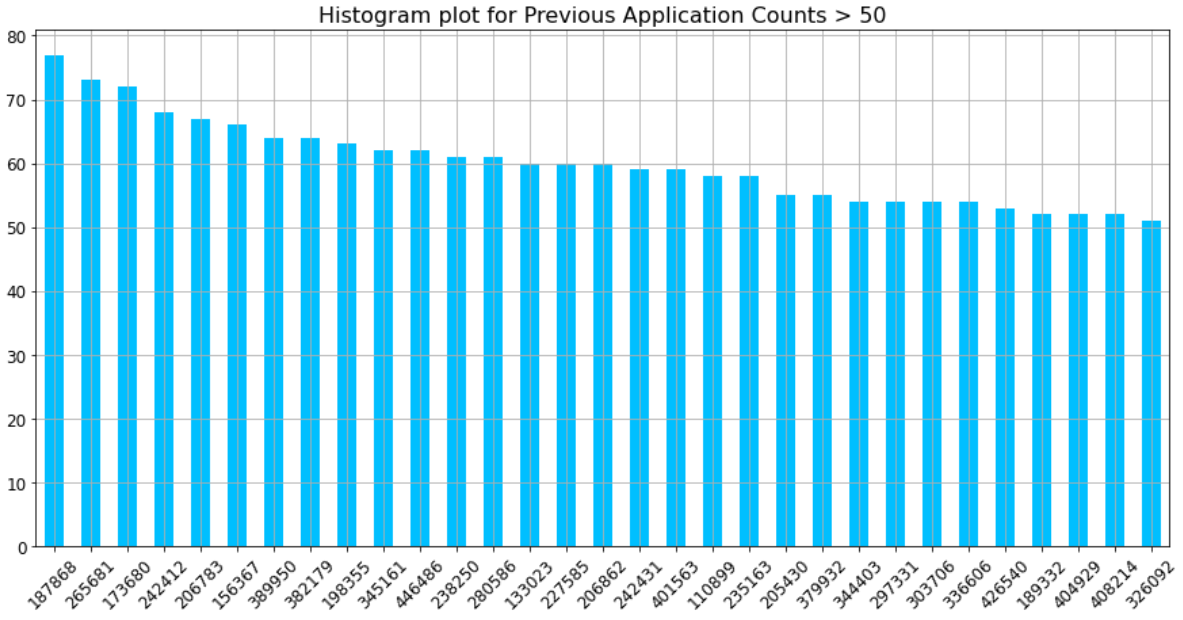
In [ ]:
```
#Find the intersection of two arrays.
print(f'Number of test applicants with previous applications is {len(np.intersect1d
```

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

In [ ]:
```
# How many previous applciations  per applicant in the previous_application
plt.figure(figsize=(15,7))
prevAppCounts = appsDF['SK_ID_CURR'].value_counts(dropna=False)
len(prevAppCounts[prevAppCounts >40])  #more that 40 previous applications
plt.hist(prevAppCounts[prevAppCounts>=0], bins=100, color='crimson')
plt.xticks(size=12)
plt.yticks(size=12)
plt.xlabel('', size=14)
plt.ylabel('', size=14)
plt.title('Histogram plot for SK_ID_CURR', size=16)
plt.grid(b=True)
```



In [ ]:
```
plt.figure(figsize=(15,7))
prevAppCounts[prevAppCounts > 50].plot(kind='bar', color='deepskyblue')
plt.xticks(size=12, rotation=45)
plt.yticks(size=12)
plt.xlabel('', size=14)
plt.ylabel('', size=14)
plt.title('Histogram plot for Previous Application Counts > 50', size=16)
plt.grid(b=True)
```



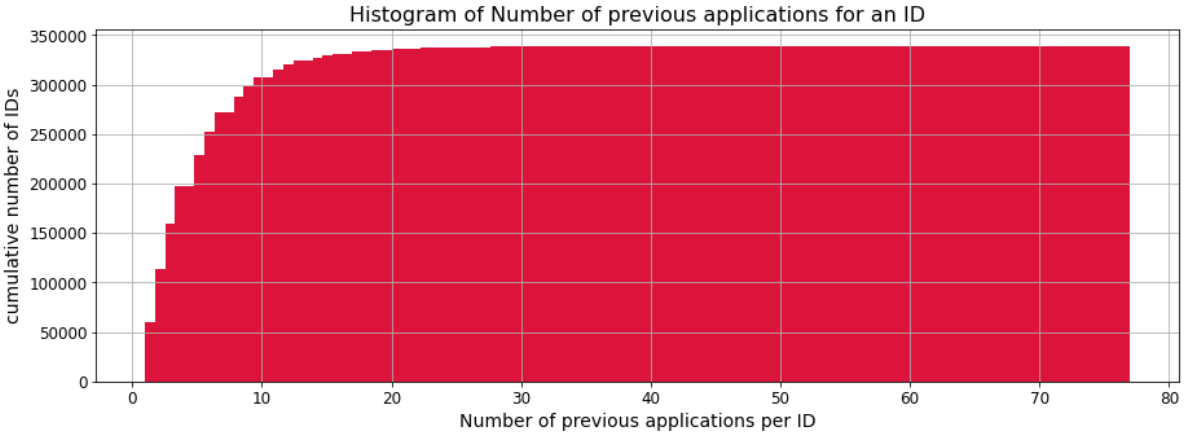
Histogram of Number of previous applications for an ID

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

Out[ ]:
```
60458
```

In [ ]:
```
plt.figure(figsize=(15,5))
plt.hist(appsDF['SK_ID_CURR'].value_counts(), cumulative =True, bins = 100, color=
plt.xticks(size=12)
plt.yticks(size=12)
plt.ylabel('cumulative number of IDs', size=14)
plt.xlabel('Number of previous applications per ID', size=14)
plt.title('Histogram of Number of previous applications for an ID', size=16)
```

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



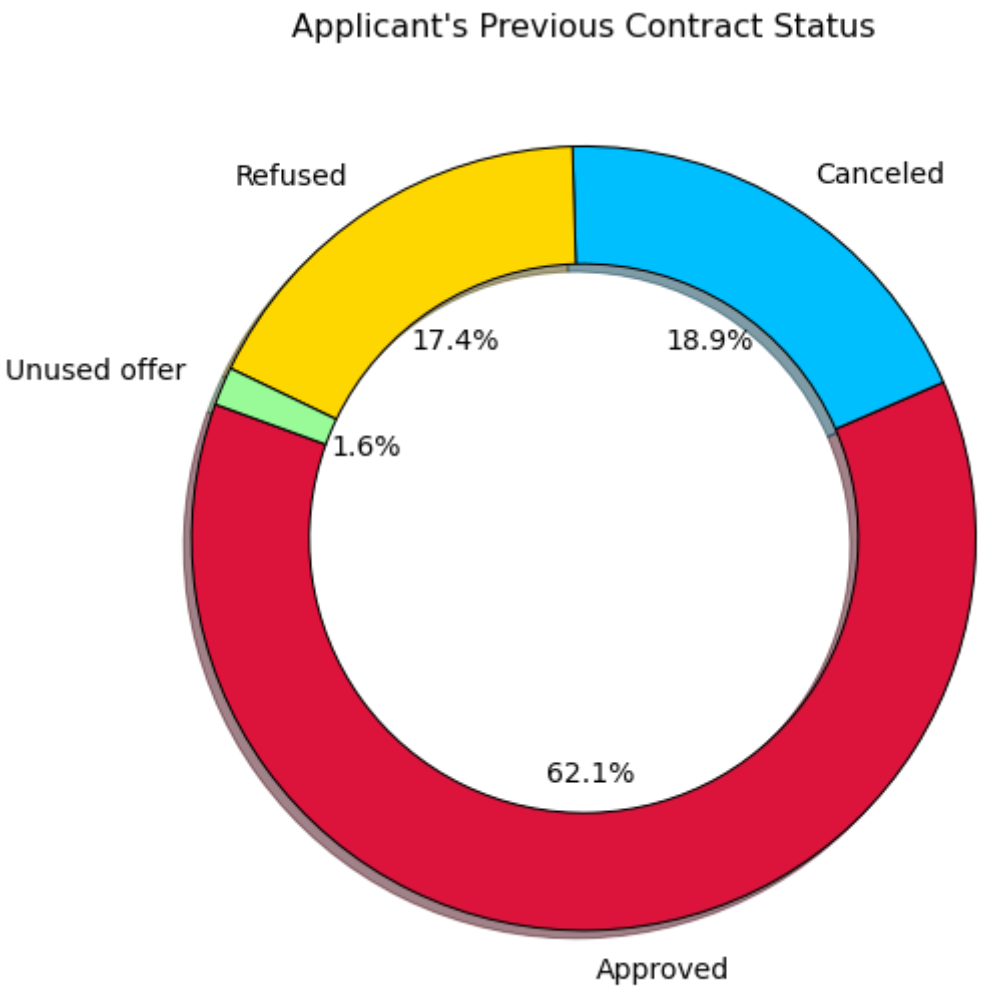
Can we differentiate applications by low, medium and high previous apps?

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

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

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


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

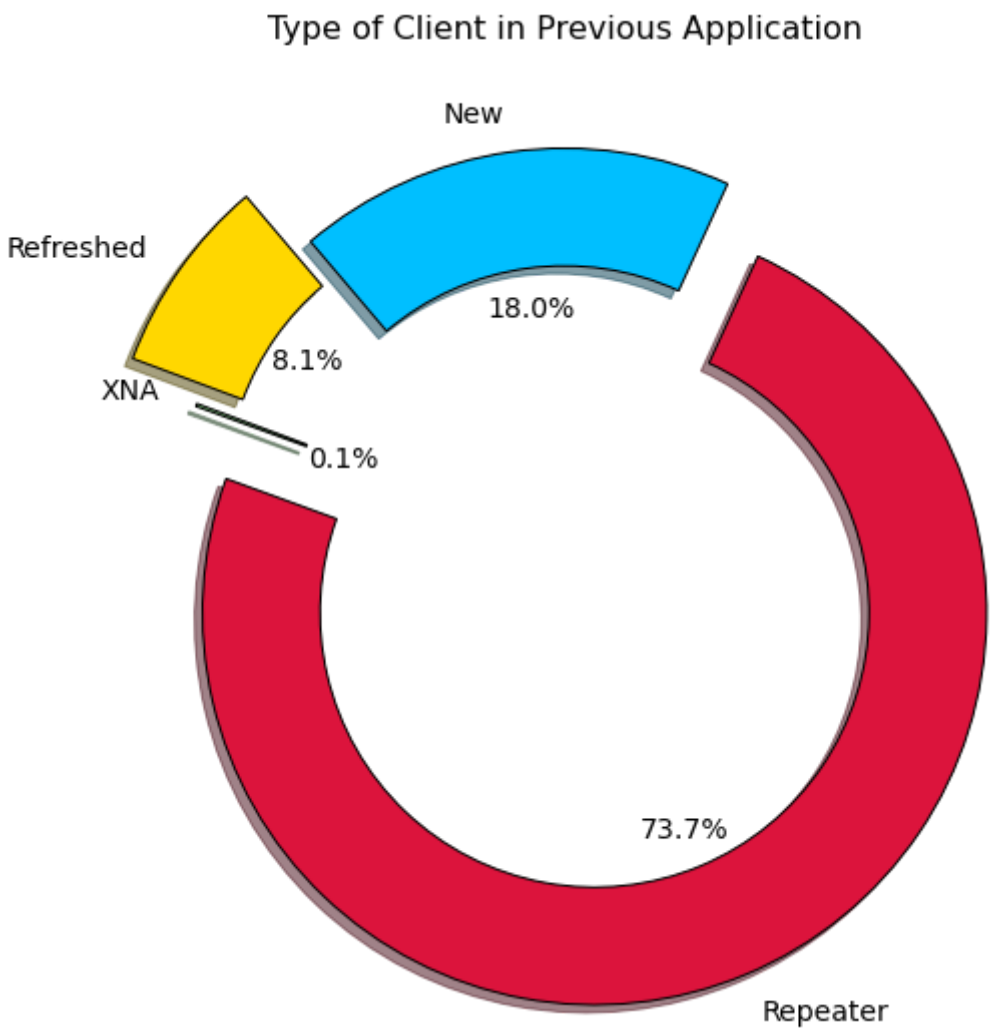
In [ ]:
```
plt.figure(figsize=(9, 9))
plt.pie(x=appsDF['NAME_CONTRACT_STATUS'].value_counts(),
        radius=1.3-0.3,
        labels=appsDF['NAME_CONTRACT_STATUS'].value_counts().index,
        autopct='%1.1f%%',
        colors=['crimson', 'deepskyblue', 'gold', 'palegreen'],
        wedgeprops={"edgecolor":"0", "width":0.3},
        startangle=160,
        shadow=True,
        textprops={'fontsize': 14})
plt.ylabel('', fontsize=14)
plt.title("Applicant's Previous Contract Status", fontsize=16)
plt.show()
```



Observation

- In previous applications, most of the applicants had their contracts approved.
- 36% of applicants had their contracts either rejected or cancelled and the rest 1.6% didn't use their contracts at all.

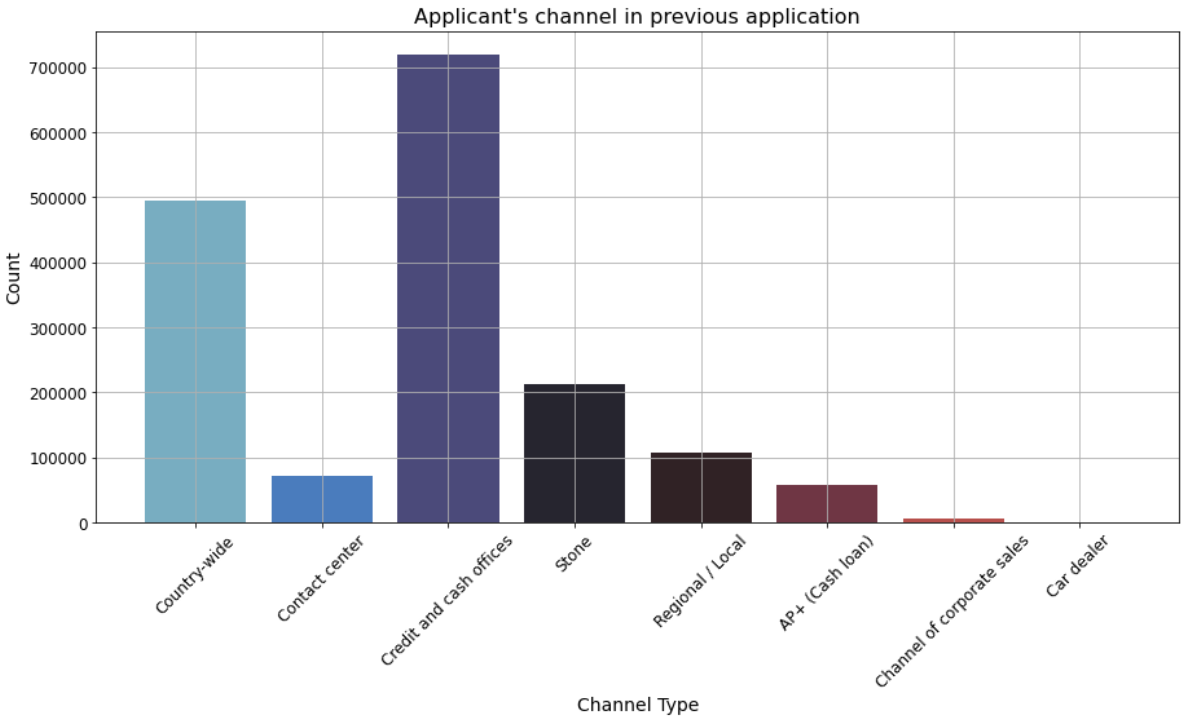
In [ ]:
```
plt.figure(figsize=(9, 9))
plt.pie(x=appsDF['NAME_CLIENT_TYPE'].value_counts(),
        radius=1.3-0.3,
        labels=appsDF['NAME_CLIENT_TYPE'].value_counts().index,
        autopct='%1.1f%%',
        colors=['crimson', 'deepskyblue', 'gold', 'palegreen'],
        explode=[0.2,0,0.2,0],
        wedgeprops={"edgecolor":"0", "width":0.3},
        startangle=160,
        shadow=True,
        textprops={'fontsize': 14})
plt.ylabel('', fontsize=14)
plt.title("Type of Client in Previous Application", fontsize=16)
plt.show()
```



Observation

- Most of the applicants are repeaters, followed by new applicants and refreshed applicants.

In [ ]:
```
plt.figure(figsize=(15, 7))
sns.countplot(x='CHANNEL_TYPE', data=appsDF, palette='icefire')
plt.title("Applicant's channel in previous application", fontsize=16)
plt.xlabel('Channel Type', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.xticks(fontsize=12, rotation=45)
plt.yticks(fontsize=12)
plt.grid(b=True)
plt.plot();
```



Observation

- Previous applicants primarily came through credit and cash offices and least via car dealers.

Feature Engineering

Performing Encoding on the Categorical Features of application_train and application_test

In [ ]:
```
# Label Encoding
# Create a label encoder object
le = LabelEncoder()
le_count = 0
cat_cols = []

# Iterate through the columns
for col in app_train:
    if app_train[col].dtype == 'object':
        # If 2 or fewer unique categories
        if len(list(app_train[col].unique())) <= 2:
            cat_cols.append(col)
            # Train on the training data
            le.fit(app_train[col])
            # Transform both training and testing data
            app_train[col] = le.transform(app_train[col])
            app_test[col] = le.transform(app_test[col])

            # Keep track of how many columns were label encoded
            le_count += 1

print('%d columns were label encoded.' % le_count)
```

```
0 columns were label encoded.
```

In [ ]:
```
# one-hot encoding of features
app_train = pd.get_dummies(app_train)
app_test = pd.get_dummies(app_test)

print('Training Features shape: ', app_train.shape)
print('Testing Features shape: ', app_test.shape)
```

```
Training Features shape:  (307511, 240)
```

```
Testing Features shape:  (48744, 239)
```

In [ ]:
```
train_labels = app_train['TARGET']

# Align the training and testing data, keep only columns present in both dataframes
app_train, app_test = app_train.align(app_test, join = 'inner', axis = 1)

# Add the target back in
app_train['TARGET'] = train_labels

print('Training Features shape: ', app_train.shape)
print('Testing Features shape: ', app_test.shape)
```

```
Training Features shape:  (307511, 240)
```

```
Testing Features shape:  (48744, 239)
```

Saving the cleaned train and test file for easy future access

In [ ]:
```
# app_train.to_csv('app_train.csv', index=False)
```

In [ ]:
```
# app_test.to_csv('app_test.csv', index=False)
```

Joining secondary tables with the primary table

In the case of the HCDR competition (and many other machine learning problems that involve multiple tables in 3NF or not) we need to join these datasets (denormalize) when using a machine learning pipeline. Joining the secondary tables with the primary table will lead to lots of new features about each loan application; these features will tend to be aggregate type features or meta data about the loan or its application. How can we do this when using Machine Learning Pipelines?

Joining `previous_application` with `application_x`

We refer to the `application_train` data (and also `application_test` data also) as the **primary table** and the other files as the **secondary tables** (e.g.,

`previous_application` dataset). All tables can be joined using the primary key `SK_ID_PREV`.

Let's assume we wish to generate a feature based on previous application attempts. In this case, possible features here could be:

- A simple feature could be the number of previous applications.
- Other summary features of original features such as `AMT_APPLICATION`, `AMT_CREDIT` could be based on average, min, max, median, etc.

To build such features, we need to join the `application_train` data (and also `application_test` data also) with the 'previous_application' dataset (and the other available datasets).

When joining this data in the context of pipelines, different strategies come to mind with various tradeoffs:

1. Preprocess each of the non-application data sets, thereby generating many new (derived) features, and then joining (aka merge) the results with the `application_train` data (the labeled dataset) and with the `application_test` data (the unlabeled submission dataset) prior to processing the data (in a train, valid, test partition) via your machine learning pipeline. [This approach is recommended for this HCDR competition. WHY?]

- Do the joins as part of the transformation steps. [Not recommended here. WHY?]. How can this be done? Will it work?
  - This would be necessary if we had dataset wide features such as IDF (inverse document frequency) which depend on the entire subset of data as opposed to a single loan application (e.g., a feature about the relative amount applied for such as the percentile of the loan amount being applied for).

I want you to think about this section and build on this.

Roadmap for secondary table processing

1. Transform all the secondary tables to features that can be joined into the main table the application table (labeled and unlabeled)
   - 'bureau', 'bureau_balance', 'credit_card_balance', 'installments_payments',
   - 'previous_application', 'POS_CASH_balance'

- Merge the transformed secondary tables with the primary tables (i.e., the `application_train` data (the labeled dataset) and with the `application_test` data (the unlabeled submission dataset)), thereby leading to X_train, y_train, X_valid, etc.
- Proceed with the learning pipeline using X_train, y_train, X_valid, etc.
- Generate a submission file using the learnt model

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

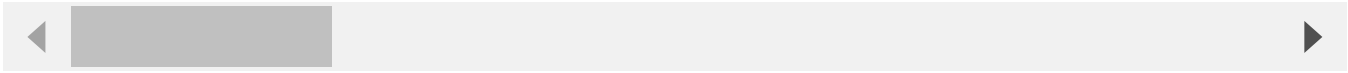
```
/root/shared/AML/I526_AML_Student/Assignments/Unit-Project-Home-Credit-Default-Risk/Phase2
```

In [ ]:
```
app_train = pd.read_csv('/kaggle/input/processed-data-hcdr/app_train.csv')
app_train.head()
```

Out[ ]:

	SK_ID_CURR	NAME_CONTRACT_TYPE	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	A
0	100002	0	0	1	0	
1	100003	0	0	0	0	
2	100004	1	1	1	0	
3	100006	0	0	1	0	
4	100007	0	0	1	0	

5 rows × 240 columns

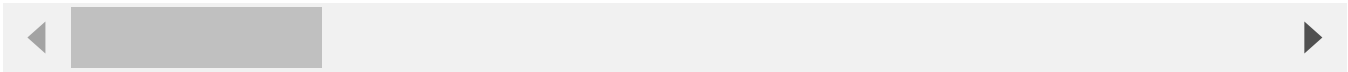


In [ ]:
```
app_test = pd.read_csv('/kaggle/input/processed-data-hcdr/app_test.csv')
app_test.head()
```

Out[ ]:

	SK_ID_CURR	NAME_CONTRACT_TYPE	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	A
0	100001	0	0	1	0	
1	100005	0	0	1	0	
2	100013	0	1	1	0	
3	100028	0	0	1	2	
4	100038	0	1	0	1	

5 rows × 239 columns



In [ ]:
```
ds_names = ("application_train", "application_test", "bureau","bureau_balance","cr
            "previous_application","POS_CASH_balance")

ds_names = ("application_train", "application_test", "bureau","bureau_balance","cr
            "previous_application","POS_CASH_balance")

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

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

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

5 rows × 121 columns

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

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 1716428 entries, 0 to 1716427

Data columns (total 17 columns):

 #   Column                  Dtype

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

 0   SK_ID_CURR              int64

 1   SK_ID_BUREAU            int64

 2   CREDIT_ACTIVE           object

 3   CREDIT_CURRENCY         object

 4   DAYS_CREDIT             int64

 5   CREDIT_DAY_OVERDUE      int64

 6   DAYS_CREDIT_ENDDATE     float64

 7   DAYS_ENDDATE_FACT       float64

 8   AMT_CREDIT_MAX_OVERDUE  float64

 9   CNT_CREDIT_PROLONG      int64

 10  AMT_CREDIT_SUM          float64

 11  AMT_CREDIT_SUM_DEBT     float64

 12  AMT_CREDIT_SUM_LIMIT    float64

 13  AMT_CREDIT_SUM_OVERDUE  float64

 14  CREDIT_TYPE             object

 15  DAYS_CREDIT_UPDATE      int64

 16  AMT_ANNUITY             float64

dtypes: float64(8), int64(6), object(3)

memory usage: 222.6+ MB

None
```

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

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

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

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

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	AMT_CREDIT_LIMIT_ACTUAL	AM
0	2562384	378907	-6	56.970	135000	
1	2582071	363914	-1	63975.555	45000	
2	1740877	371185	-7	31815.225	450000	
3	1389973	337855	-4	236572.110	225000	
4	1891521	126868	-1	453919.455	450000	

5 rows × 23 columns

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

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER	DAYS_INS
0	1054186	161674	1.0	6	
1	1330831	151639	0.0	34	
2	2085231	193053	2.0	1	
3	2452527	199697	1.0	3	
4	2714724	167756	1.0	2	

```
previous_application: shape is (1670214, 37)

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 1670214 entries, 0 to 1670213

Data columns (total 37 columns):

 #   Column                       Non-Null Count    Dtype

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

 0   SK_ID_PREV                   1670214 non-null  int64

 1   SK_ID_CURR                   1670214 non-null  int64

 2   NAME_CONTRACT_TYPE           1670214 non-null  object

 3   AMT_ANNUITY                  1297979 non-null  float64

 4   AMT_APPLICATION              1670214 non-null  float64

 5   AMT_CREDIT                   1670213 non-null  float64

 6   AMT_DOWN_PAYMENT             774370 non-null   float64

 7   AMT_GOODS_PRICE              1284699 non-null  float64

 8   WEEKDAY_APPR_PROCESS_START   1670214 non-null  object

 9   HOUR_APPR_PROCESS_START      1670214 non-null  int64

 10  FLAG_LAST_APPL_PER_CONTRACT  1670214 non-null  object

 11  NFLAG_LAST_APPL_IN_DAY       1670214 non-null  int64

 12  RATE_DOWN_PAYMENT            774370 non-null   float64

 13  RATE_INTEREST_PRIMARY        5951 non-null     float64

 14  RATE_INTEREST_PRIVILEGED     5951 non-null     float64

 15  NAME_CASH_LOAN_PURPOSE       1670214 non-null  object

 16  NAME_CONTRACT_STATUS         1670214 non-null  object

 17  DAYS_DECISION                1670214 non-null  int64

 18  NAME_PAYMENT_TYPE            1670214 non-null  object

 19  CODE_REJECT_REASON           1670214 non-null  object

 20  NAME_TYPE_SUITE              849809 non-null   object

 21  NAME_CLIENT_TYPE             1670214 non-null  object

 22  NAME_GOODS_CATEGORY          1670214 non-null  object

 23  NAME_PORTFOLIO               1670214 non-null  object

 24  NAME_PRODUCT_TYPE            1670214 non-null  object

 25  CHANNEL_TYPE                 1670214 non-null  object
```

```
 26  SELLERPLACE_AREA             1670214 non-null  int64
 27  NAME_SELLER_INDUSTRY         1670214 non-null  object
 28  CNT_PAYMENT                  1297984 non-null  float64
 29  NAME_YIELD_GROUP             1670214 non-null  object
 30  PRODUCT_COMBINATION          1669868 non-null  object
 31  DAYS_FIRST_DRAWING           997149 non-null   float64
 32  DAYS_FIRST_DUE               997149 non-null   float64
 33  DAYS_LAST_DUE_1ST_VERSION    997149 non-null   float64
 34  DAYS_LAST_DUE                997149 non-null   float64
 35  DAYS_TERMINATION             997149 non-null   float64
 36  NFLAG_INSURED_ON_APPROVAL    997149 non-null   float64
dtypes: float64(15), int64(6), object(16)
memory usage: 471.5+ MB
None
```

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CI
0	2030495	271877	Consumer loans	1730.430	17145.0	17
1	2802425	108129	Cash loans	25188.615	607500.0	679
2	2523466	122040	Cash loans	15060.735	112500.0	136
3	2819243	176158	Cash loans	47041.335	450000.0	470
4	1784265	202054	Cash loans	31924.395	337500.0	404

5 rows × 37 columns

```
POS_CASH_balance: shape is (10001358, 8)

<class 'pandas.core.frame.DataFrame'>

RangeIndex: 10001358 entries, 0 to 10001357

Data columns (total 8 columns):

 #   Column                 Dtype

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

 0   SK_ID_PREV             int64

 1   SK_ID_CURR             int64

 2   MONTHS_BALANCE         int64

 3   CNT_INSTALMENT         float64

 4   CNT_INSTALMENT_FUTURE  float64

 5   NAME_CONTRACT_STATUS   object

 6   SK_DPD                 int64

 7   SK_DPD_DEF             int64

dtypes: float64(2), int64(5), object(1)

memory usage: 610.4+ MB

None
```

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

In [ ]:
```
prevApps = pd.read_csv('/kaggle/input/home-credit-default-risk/previous_applicatio
prevApps.head()
```

Out[ ]:

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_C
0	2030495	271877	Consumer loans	1730.430	17145.0	17
1	2802425	108129	Cash loans	25188.615	607500.0	679
2	2523466	122040	Cash loans	15060.735	112500.0	136
3	2819243	176158	Cash loans	47041.335	450000.0	470
4	1784265	202054	Cash loans	31924.395	337500.0	404

5 rows × 37 columns

In [ ]:
```
app_train.shape, app_test.shape
```

Out[ ]:
```
((307511, 240), (48744, 239))
```

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

Out[ ]:

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

1 rows × 37 columns

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

Out[ ]:
```
6    0.0
Name: AMT_CREDIT, dtype: float64
```

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

Out[ ]:

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

1 rows × 37 columns

Missing values in prevApps

In [ ]:
```
prevApps.isna().sum()
```

Out[ ]:
```
SK_ID_PREV                       0
SK_ID_CURR                       0
NAME_CONTRACT_TYPE               0
AMT_ANNUITY                 372235
AMT_APPLICATION                  0
AMT_CREDIT                       1
AMT_DOWN_PAYMENT            895844
AMT_GOODS_PRICE             385515
WEEKDAY_APPR_PROCESS_START       0
HOUR_APPR_PROCESS_START          0
FLAG_LAST_APPL_PER_CONTRACT      0
NFLAG_LAST_APPL_IN_DAY           0
RATE_DOWN_PAYMENT           895844
RATE_INTEREST_PRIMARY      1664263
RATE_INTEREST_PRIVILEGED   1664263
NAME_CASH_LOAN_PURPOSE           0
NAME_CONTRACT_STATUS             0
DAYS_DECISION                    0
NAME_PAYMENT_TYPE                0
CODE_REJECT_REASON               0
NAME_TYPE_SUITE             820405
NAME_CLIENT_TYPE                 0
NAME_GOODS_CATEGORY              0
NAME_PORTFOLIO                   0
NAME_PRODUCT_TYPE                0
CHANNEL_TYPE                     0
SELLERPLACE_AREA                 0
NAME_SELLER_INDUSTRY             0
CNT_PAYMENT                 372230
NAME_YIELD_GROUP                 0
PRODUCT_COMBINATION            346
DAYS_FIRST_DRAWING          673065
DAYS_FIRST_DUE              673065
DAYS_LAST_DUE_1ST_VERSION   673065
DAYS_LAST_DUE               673065
DAYS_TERMINATION            673065
NFLAG_INSURED_ON_APPROVAL   673065
dtype: int64
```

In [ ]:
```
prevApps.columns
```

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

feature engineering for prevApp table

In [ ]:
```
list(prevApps.columns)
```

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

In [ ]:
```
features = features = ['AMT_ANNUITY', 'AMT_APPLICATION', 'AMT_CREDIT', 'AMT_DOWN_PA
            '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_1
            'DAYS_LAST_DUE', 'DAYS_TERMINATION']
print(f"{prevApps[features].describe()}")
agg_ops = ["min", "max", "mean"]
result = prevApps.groupby(["SK_ID_CURR"], as_index=False).agg("mean") #group by ID
display(result.head())
print("-"*50)
result = prevApps.groupby(["SK_ID_CURR"], as_index=False).agg({'AMT_ANNUITY' : agg_
result.columns = result.columns.map('_'.join)
display(result)
result['range_AMT_APPLICATION'] = result['AMT_APPLICATION_max'] - result['AMT_APPLI
print(f"result.shape: {result.shape}")
result[0:10]
```

	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	\
count	1.297979e+06	1.670214e+06	1.670213e+06	7.743700e+05	
mean	1.595512e+04	1.752339e+05	1.961140e+05	6.697402e+03	
std	1.478214e+04	2.927798e+05	3.185746e+05	2.092150e+04	
min	0.000000e+00	0.000000e+00	0.000000e+00	-9.000000e-01	
25%	6.321780e+03	1.872000e+04	2.416050e+04	0.000000e+00	
50%	1.125000e+04	7.104600e+04	8.054100e+04	1.638000e+03	
75%	2.065842e+04	1.803600e+05	2.164185e+05	7.740000e+03	
max	4.180581e+05	6.905160e+06	6.905160e+06	3.060045e+06	

	AMT_GOODS_PRICE	RATE_DOWN_PAYMENT	RATE_INTEREST_PRIMARY	\
count	1.284699e+06	774370.000000	5951.000000	
mean	2.278473e+05	0.079637	0.188357	
std	3.153966e+05	0.107823	0.087671	
min	0.000000e+00	-0.000015	0.034781	
25%	5.084100e+04	0.000000	0.160716	
50%	1.123200e+05	0.051605	0.189122	
75%	2.340000e+05	0.108909	0.193330	
max	6.905160e+06	1.000000	1.000000	

	RATE_INTEREST_PRIVILEGED	DAYS_DECISION	CNT_PAYMENT	\
count	5951.000000	1.670214e+06	1.297984e+06	
mean	0.773503	-8.806797e+02	1.605408e+01	
std	0.100879	7.790997e+02	1.456729e+01	
min	0.373150	-2.922000e+03	0.000000e+00	
25%	0.715645	-1.300000e+03	6.000000e+00	
50%	0.835095	-5.810000e+02	1.200000e+01	
75%	0.852537	-2.800000e+02	2.400000e+01	
max	1.000000	-1.000000e+00	8.400000e+01	

	DAYS_FIRST_DRAWING	DAYS_FIRST_DUE	DAYS_LAST_DUE_1ST_VERSION	\
count	997149.000000	997149.000000	997149.000000	

```
mean        342209.855039    13826.269337              33767.774054
std          88916.115834    72444.869708             106857.034789
min          -2922.000000    -2892.000000              -2801.000000
25%         365243.000000    -1628.000000              -1242.000000
50%         365243.000000     -831.000000               -361.000000
75%         365243.000000     -411.000000                129.000000
max         365243.000000   365243.000000             365243.000000
```

```
       DAYS_LAST_DUE  DAYS_TERMINATION
count  997149.000000     997149.000000
mean    76582.403064      81992.343838
std    149647.415123     153303.516729
min     -2889.000000      -2874.000000
25%     -1314.000000      -1270.000000
50%      -537.000000       -499.000000
75%       -74.000000        -44.000000
max    365243.000000     365243.000000
```

	SK_ID_CURR	SK_ID_PREV	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAY
0	100001	1.369693e+06	3951.000	24835.50	23787.00	
1	100002	1.038818e+06	9251.775	179055.00	179055.00	
2	100003	2.281150e+06	56553.990	435436.50	484191.00	
3	100004	1.564014e+06	5357.250	24282.00	20106.00	
4	100005	2.176837e+06	4813.200	22308.75	20076.75	

5 rows × 21 columns

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

	SK_ID_CURR_	AMT_ANNUITY_min	AMT_ANNUITY_max	AMT_ANNUITY_mean	AMT_APPLI
0	100001	3951.000	3951.000	3951.000000	
1	100002	9251.775	9251.775	9251.775000	
2	100003	6737.310	98356.995	56553.990000	
3	100004	5357.250	5357.250	5357.250000	
4	100005	4813.200	4813.200	4813.200000	
...	...	...	...	...	
338852	456251	6605.910	6605.910	6605.910000	
338853	456252	10074.465	10074.465	10074.465000	
338854	456253	3973.095	5567.715	4770.405000	
338855	456254	2296.440	19065.825	10681.132500	
338856	456255	2250.000	54022.140	20775.391875	

338857 rows × 7 columns

```
result.shape: (338857, 8)
```

Out[ ]:

	SK_ID_CURR_	AMT_ANNUITY_min	AMT_ANNUITY_max	AMT_ANNUITY_mean	AMT_APPLICATIO
0	100001	3951.000	3951.000	3951.000000	
1	100002	9251.775	9251.775	9251.775000	1
2	100003	6737.310	98356.995	56553.990000	
3	100004	5357.250	5357.250	5357.250000	
4	100005	4813.200	4813.200	4813.200000	
5	100006	2482.920	39954.510	23651.175000	
6	100007	1834.290	22678.785	12278.805000	
7	100008	8019.090	25309.575	15839.696250	
8	100009	7435.845	17341.605	10051.412143	

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

Out[ ]:
```
SK_ID_CURR_              0
AMT_ANNUITY_min        480
AMT_ANNUITY_max        480
AMT_ANNUITY_mean       480
AMT_APPLICATION_min      0
AMT_APPLICATION_max      0
AMT_APPLICATION_mean     0
range_AMT_APPLICATION    0
dtype: int64
```

In [ ]:
```
result.shape
```

Out[ ]:
```
(338857, 8)
```

Performing Label Encoding and OHE on the previous_application dataset for smoother feature engineering and dataset merging.

In [ ]:
```
# Label Encoding
# Create a label encoder object
le_prev = LabelEncoder()
le_prev_count = 0

# Iterate through the columns
for col in prevApps:
    if prevApps[col].dtype == 'object':
        print(prevApps[col].head())
        # If 2 or fewer unique categories
        # if len(list(prevApps[col].unique())) <= 2:
            # Train on the training data
        prevApps[col] = prevApps[col].fillna('NaN')
        le_prev.fit(prevApps[col])
        # Transform the prevApps dataframe
        prevApps[col] = le_prev.transform(prevApps[col])
        # app_test[col] = le_prev.transform(app_test[col])

        # Keep track of how many columns were label encoded
        le_prev_count += 1

print('%d columns were label encoded.' % le_prev_count)
```

```
0    Consumer loans
1        Cash loans
2        Cash loans
3        Cash loans
4        Cash loans
Name: NAME_CONTRACT_TYPE, dtype: object
0    SATURDAY
1    THURSDAY
2     TUESDAY
3      MONDAY
4    THURSDAY
Name: WEEKDAY_APPR_PROCESS_START, dtype: object
0    Y
1    Y
2    Y
3    Y
4    Y
Name: FLAG_LAST_APPL_PER_CONTRACT, dtype: object
0        XAP
1        XNA
2        XNA
3        XNA
4    Repairs
Name: NAME_CASH_LOAN_PURPOSE, dtype: object
0    Approved
1    Approved
2    Approved
3    Approved
4     Refused
Name: NAME_CONTRACT_STATUS, dtype: object
0    Cash through the bank
1                      XNA
```

```
2    Cash through the bank
3    Cash through the bank
4    Cash through the bank
Name: NAME_PAYMENT_TYPE, dtype: object
0    XAP
1    XAP
2    XAP
3    XAP
4     HC
Name: CODE_REJECT_REASON, dtype: object
0                NaN
1      Unaccompanied
2    Spouse, partner
3                NaN
4                NaN
Name: NAME_TYPE_SUITE, dtype: object
0    Repeater
1    Repeater
2    Repeater
3    Repeater
4    Repeater
Name: NAME_CLIENT_TYPE, dtype: object
0    Mobile
1       XNA
2       XNA
3       XNA
4       XNA
Name: NAME_GOODS_CATEGORY, dtype: object
0     POS
1    Cash
2    Cash
3    Cash
```

```
4    Cash
Name: NAME_PORTFOLIO, dtype: object
0        XNA
1     x-sell
2     x-sell
3     x-sell
4    walk-in
Name: NAME_PRODUCT_TYPE, dtype: object
0               Country-wide
1             Contact center
2    Credit and cash offices
3    Credit and cash offices
4    Credit and cash offices
Name: CHANNEL_TYPE, dtype: object
0    Connectivity
1             XNA
2             XNA
3             XNA
4             XNA
Name: NAME_SELLER_INDUSTRY, dtype: object
0        middle
1    low_action
2          high
3        middle
4          high
Name: NAME_YIELD_GROUP, dtype: object
0    POS mobile with interest
1            Cash X-Sell: low
2           Cash X-Sell: high
3         Cash X-Sell: middle
4           Cash Street: high
Name: PRODUCT_COMBINATION, dtype: object
```

```
16 columns were label encoded.
```

feature transformer for prevApp table

In [ ]:

```
class prevAppsFeaturesAggregater(BaseEstimator, TransformerMixin):
    def __init__(self, features=None): # no *args or **kargs
        self.features = features
        self.agg_op_features = {}
        for f in features:
#             self.agg_op_features[f] = {f"{f}_{func}":func for func in ["min", "mc
            self.agg_op_features[f] =  ["min", "max", "mean"]

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

    def transform(self, X, y=None):
        #from IPython.core.debugger import Pdb as pdb;    pdb().set_trace() #breakp
        result = X.groupby(["SK_ID_CURR"]).agg('mean')
#         result.columns = result.columns.droplevel()
#         result.columns = ["_".join(x) for x in result.columns.ravel()]

        result = result.reset_index(level=["SK_ID_CURR"])
#         result['range_AMT_APPLICATION'] = result['AMT_APPLICATION_max'] - result[
        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', 'AMT_CREDIT', 'AMT_DOWN_PAYMENT', 'AM
            '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_1
            'DAYS_LAST_DUE', 'DAYS_TERMINATION']

res = test_driver_prevAppsFeaturesAggregater(prevApps, features)
print(f"HELLO")
print(f"Test driver: \n{res[0:10]}")
# print(f"input[features][0:10]: \n{prevApps[0:10]}")
```

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

```
df[['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_F
IRST_DUE', 'DAYS_LAST_DUE_1ST_VERSION', 'DAYS_LAST_DUE', 'DAYS_TERMINATION']][0:
5]:
```

	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT	\
0	1730.430	17145.0	17145.0	0.0	
1	25188.615	607500.0	679671.0	NaN	
2	15060.735	112500.0	136444.5	NaN	
3	47041.335	450000.0	470790.0	NaN	
4	31924.395	337500.0	404055.0	NaN	

	AMT_GOODS_PRICE	RATE_DOWN_PAYMENT	RATE_INTEREST_PRIMARY	\
0	17145.0	0.0	0.182832	
1	607500.0	NaN	NaN	
2	112500.0	NaN	NaN	
3	450000.0	NaN	NaN	
4	337500.0	NaN	NaN	

	RATE_INTEREST_PRIVILEGED	DAYS_DECISION	NAME_PAYMENT_TYPE	CNT_PAYMENT	\
0	0.867336	-73	0	12.0	
1	NaN	-164	3	36.0	
2	NaN	-301	0	12.0	
3	NaN	-512	0	12.0	
4	NaN	-781	0	24.0	

	DAYS_FIRST_DRAWING	DAYS_FIRST_DUE	DAYS_LAST_DUE_1ST_VERSION	\
0	365243.0	-42.0	300.0	
1	365243.0	-134.0	916.0	
2	365243.0	-271.0	59.0	
3	365243.0	-482.0	-152.0	
4	NaN	NaN	NaN	

```
   DAYS_LAST_DUE  DAYS_TERMINATION
0          -42.0             -37.0
1       365243.0          365243.0
2       365243.0          365243.0
3         -182.0            -177.0
4            NaN               NaN
HELLO
Test driver:
   SK_ID_CURR    SK_ID_PREV  NAME_CONTRACT_TYPE   AMT_ANNUITY  \
0      100001  1.369693e+06            1.000000   3951.000000
1      100002  1.038818e+06            1.000000   9251.775000
2      100003  2.281150e+06            0.666667  56553.990000
3      100004  1.564014e+06            1.000000   5357.250000
4      100005  2.176837e+06            0.500000   4813.200000
5      100006  1.932462e+06            0.666667  23651.175000
6      100007  2.157812e+06            0.333333  12278.805000
7      100008  1.936735e+06            0.600000  15839.696250
8      100009  1.881798e+06            1.000000  10051.412143
9      100010  2.349489e+06            1.000000  27463.410000

   AMT_APPLICATION     AMT_CREDIT  AMT_DOWN_PAYMENT  AMT_GOODS_PRICE  \
0     24835.500000   23787.000000       2520.000000     24835.500000
1    179055.000000  179055.000000          0.000000    179055.000000
2    435436.500000  484191.000000       3442.500000    435436.500000
3     24282.000000   20106.000000       4860.000000     24282.000000
4     22308.750000   20076.750000       4464.000000     44617.500000
5    272203.260000  291695.500000      34840.170000    408304.890000
6    150530.250000  166638.750000       3390.750000    150530.250000
7    155701.800000  162767.700000       5548.500000    194627.250000
8     76741.714286   70137.642857       9203.142857     76741.714286
9    247212.000000  260811.000000          0.000000    247212.000000
```

	WEEKDAY_APPR_PROCESS_START	HOUR_APPR_PROCESS_START	...	\
0	0.000000	13.000000	...	
1	2.000000	9.000000	...	
2	1.666667	14.666667	...	
3	0.000000	5.000000	...	
4	2.000000	10.500000	...	
5	3.777778	14.666667	...	
6	2.166667	12.333333	...	
7	1.400000	12.000000	...	
8	3.285714	13.714286	...	
9	5.000000	16.000000	...	

	NAME_SELLER_INDUSTRY	CNT_PAYMENT	NAME_YIELD_GROUP	PRODUCT_COMBINATION	\
0	2.000000	8.000000	1.000000	14.000000	
1	0.000000	24.000000	3.000000	16.000000	
2	6.333333	10.000000	3.666667	9.666667	
3	2.000000	4.000000	4.000000	15.000000	
4	6.000000	12.000000	0.500000	8.000000	
5	8.555556	23.000000	1.333333	5.222222	
6	5.666667	20.666667	2.500000	8.500000	
7	4.800000	14.000000	2.600000	8.800000	
8	3.714286	8.000000	2.714286	10.714286	
9	5.000000	10.000000	2.000000	13.000000	

	DAYS_FIRST_DRAWING	DAYS_FIRST_DUE	DAYS_LAST_DUE_1ST_VERSION	\
0	365243.0	-1709.000000	-1499.000000	
1	365243.0	-565.000000	125.000000	
2	365243.0	-1274.333333	-1004.333333	
3	365243.0	-784.000000	-694.000000	
4	365243.0	-706.000000	-376.000000	
5	365243.0	91066.500000	91584.000000	

```
6            365243.0    -1263.200000                -837.200000
7            365243.0    -1434.500000               -1044.500000
8            365243.0     -688.285714                -478.285714
9            365243.0    -1039.000000                -769.000000


   DAYS_LAST_DUE  DAYS_TERMINATION  NFLAG_INSURED_ON_APPROVAL
0   -1619.000000      -1612.000000                   0.000000
1     -25.000000        -17.000000                   0.000000
2   -1054.333333      -1047.333333                   0.666667
3    -724.000000       -714.000000                   0.000000
4    -466.000000       -460.000000                   0.000000
5  182477.500000     182481.750000                   0.000000
6   72136.200000      72143.800000                   0.600000
7   -1209.500000       -872.750000                   0.250000
8   51666.857143      51672.857143                   0.000000
9    -769.000000       -762.000000                   0.000000


[10 rows x 37 columns]
```

Mergeing the engineered features with train and test datasets:

In [ ]:
```
app_train_res = app_train.merge(res, on = 'SK_ID_CURR', how = 'left')
app_train_res.head()
```

Out[ ]:

	SK_ID_CURR	NAME_CONTRACT_TYPE_x	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN
0	100002	0	0	1	0
1	100003	0	0	0	0
2	100004	1	1	1	0
3	100006	0	0	1	0
4	100007	0	0	1	0

5 rows × 276 columns

In [ ]:
```
app_test_res = app_test.merge(res, on = 'SK_ID_CURR', how = 'left')
app_test_res.head()
```

Out[ ]:

	SK_ID_CURR	NAME_CONTRACT_TYPE_x	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN
0	100001	0	0	1	0
1	100005	0	0	1	0
2	100013	0	1	1	0
3	100028	0	0	1	2
4	100038	0	1	0	1

5 rows × 275 columns

In [ ]:
```
print('Prev_Features Features shape: ', prevApps.shape)
prevApps.head()
```

```
Prev_Features Features shape:  (1670214, 37)
```

Out[ ]:

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CI
0	2030495	271877	1	1730.430	17145.0	17
1	2802425	108129	0	25188.615	607500.0	679
2	2523466	122040	0	15060.735	112500.0	136
3	2819243	176158	0	47041.335	450000.0	470
4	1784265	202054	0	31924.395	337500.0	404

5 rows × 37 columns

Creating polynomial features from the train and test dataset for columns with high correlation with the TARGET feature:

In [ ]:
```
from sklearn.preprocessing import PolynomialFeatures
from sklearn.impute import SimpleImputer

# Make a new dataframe for polynomial features
poly_features = app_train_res[['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3', 'DA
poly_features_test = app_test_res[['EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3',

# imputer for handling missing values
# from sklearn.preprocessing import Imputer
imputer = SimpleImputer(strategy = 'median')

poly_target = poly_features['TARGET']

poly_features = poly_features.drop(columns = ['TARGET'])

# Need to impute missing values
poly_features = imputer.fit_transform(poly_features)
poly_features_test = imputer.transform(poly_features_test)

# Create the polynomial object with specified degree
poly_transformer = PolynomialFeatures(degree = 3)
```

In [ ]:
```
# Train the polynomial features
poly_transformer.fit(poly_features)

# Transform the features
poly_features = poly_transformer.transform(poly_features)
poly_features_test = poly_transformer.transform(poly_features_test)
print('Polynomial Features shape: ', poly_features.shape)
```

```
Polynomial Features shape:  (307511, 35)
```

In [ ]:
```
poly_transformer.get_feature_names(input_features = ['EXT_SOURCE_1', 'EXT_SOURCE_2
```

Out[ ]:
```
['1',
 'EXT_SOURCE_1',
 'EXT_SOURCE_2',
 'EXT_SOURCE_3',
 'DAYS_BIRTH',
 'EXT_SOURCE_1^2',
 'EXT_SOURCE_1 EXT_SOURCE_2',
 'EXT_SOURCE_1 EXT_SOURCE_3',
 'EXT_SOURCE_1 DAYS_BIRTH',
 'EXT_SOURCE_2^2',
 'EXT_SOURCE_2 EXT_SOURCE_3',
 'EXT_SOURCE_2 DAYS_BIRTH',
 'EXT_SOURCE_3^2',
 'EXT_SOURCE_3 DAYS_BIRTH',
 'DAYS_BIRTH^2',
 'EXT_SOURCE_1^3',
 'EXT_SOURCE_1^2 EXT_SOURCE_2',
 'EXT_SOURCE_1^2 EXT_SOURCE_3',
 'EXT_SOURCE_1^2 DAYS_BIRTH',
 'EXT_SOURCE_1 EXT_SOURCE_2^2']
```

In [ ]:
```
# Create a dataframe of the features
poly_features = pd.DataFrame(poly_features,
                             columns = poly_transformer.get_feature_names(['EXT_SOU
                                                                           'EXT_SOU

# Add in the target
poly_features['TARGET'] = poly_target

# Find the correlations with the target
poly_corrs = poly_features.corr()['TARGET'].sort_values()

# Display most negative and most positive
print(poly_corrs.head(10))
print(poly_corrs.tail(5))
```

```
EXT_SOURCE_2 EXT_SOURCE_3                -0.193939

EXT_SOURCE_1 EXT_SOURCE_2 EXT_SOURCE_3   -0.189605

EXT_SOURCE_2 EXT_SOURCE_3 DAYS_BIRTH     -0.181283

EXT_SOURCE_2^2 EXT_SOURCE_3              -0.176428

EXT_SOURCE_2 EXT_SOURCE_3^2              -0.172282

EXT_SOURCE_1 EXT_SOURCE_2                -0.166625

EXT_SOURCE_1 EXT_SOURCE_3                -0.164065

EXT_SOURCE_2                             -0.160295

EXT_SOURCE_2 DAYS_BIRTH                  -0.156873

EXT_SOURCE_1 EXT_SOURCE_2^2              -0.156867

Name: TARGET, dtype: float64

DAYS_BIRTH     -0.078239

DAYS_BIRTH^2   -0.076672

DAYS_BIRTH^3   -0.074273

TARGET          1.000000

1                    NaN

Name: TARGET, dtype: float64
```

Merging the polynomial features with train and test dataset and aligning the two datasets to match their shapes:

In [ ]:
```
# Put test features into dataframe
poly_features_test = pd.DataFrame(poly_features_test,
                                  columns = poly_transformer.get_feature_names(['E
                                                                                'E
                                                                                'D


# Merge polynomial features into training dataframe
poly_features['SK_ID_CURR'] = app_train_res['SK_ID_CURR']
app_train_poly = app_train_res.merge(poly_features, on = 'SK_ID_CURR', how = 'left

# Merge polnomial features into testing dataframe
poly_features_test['SK_ID_CURR'] = app_test_res['SK_ID_CURR']
app_test_poly = app_test.merge(poly_features_test, on = 'SK_ID_CURR', how = 'left'

# Align the dataframes
app_train_poly, app_test_poly = app_train_poly.align(app_test_poly, join = 'inner'

# Print out the new shapes
print('Training data with polynomial features shape: ', app_train_poly.shape)
print('Testing data with polynomial features shape:  ', app_test_poly.shape)
```

```
Training data with polynomial features shape:  (307511, 269)

Testing data with polynomial features shape:   (48744, 269)
```

In [ ]:
```
app_train_poly['TARGET'] = app_train['TARGET']
app_train_poly.head()
```

Out[ ]:

	SK_ID_CURR	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	REGI
0	100002	0	1	0	202500.0	
1	100003	0	0	0	270000.0	
2	100004	1	1	0	67500.0	
3	100006	0	1	0	135000.0	
4	100007	0	1	0	121500.0	

5 rows × 270 columns

In [ ]:
```
app_train_poly['TARGET'].head()
```

Out[ ]:
```
0    1
1    0
2    0
3    0
4    0
Name: TARGET, dtype: int64
```

In [ ]:
```
app_test_poly.head()
```

Out[ ]:

	SK_ID_CURR	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	REGI
0	100001	0	1	0	135000.0	
1	100005	0	1	0	99000.0	
2	100013	1	1	0	202500.0	
3	100028	0	1	2	315000.0	
4	100038	1	0	1	180000.0	

5 rows × 269 columns

Feature Aggregating for bureau.csv dataset

In [ ]:
```
bureau_data = pd.read_csv('/kaggle/input/home-credit-default-risk/bureau.csv')
print(bureau_data.shape)
bureau_data.head()
```

```
(1716428, 17)
```

Out[ ]:

	SK_ID_CURR	SK_ID_BUREAU	CREDIT_ACTIVE	CREDIT_CURRENCY	DAYS_CREDIT	CREDIT_DAY_O
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	

In [ ]:
```
# Find the intersection of app_train and app_test.
print(f'Number of train applicants with previous applications is {len(np.intersect
```

```
Number of train applicants with previous applications is 263,491
```

In [ ]:
```
# Groupby the client id (SK_ID_CURR), count the number of previous loans, and renam
previous_loan_counts = bureau_data.groupby('SK_ID_CURR', as_index=False)['SK_ID_BU
previous_loan_counts.head()
```

Out[ ]:

	SK_ID_CURR	previous_loan_counts
0	100001	7
1	100002	8
2	100003	4
3	100004	2
4	100005	3

In [ ]:
```
app_train_poly_temp = app_train_poly.merge(previous_loan_counts, on = 'SK_ID_CURR',
app_train_poly_temp.head()
```

Out[ ]:

	SK_ID_CURR	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	REGI
0	100002	0	1	0	202500.0	
1	100003	0	0	0	270000.0	
2	100004	1	1	0	67500.0	
3	100006	0	1	0	135000.0	
4	100007	0	1	0	121500.0	

5 rows × 271 columns

In [ ]:
```
print('Missing Value counts in previous_loan_counts in the merged dataset:', app_tr
app_train_poly_temp['previous_loan_counts'].fillna(0, inplace=True)
print('Missing Value counts in previous_loan_counts in the merged dataset after cle
```

```
Missing Value counts in previous_loan_counts in the merged dataset: 44020

Missing Value counts in previous_loan_counts in the merged dataset after cleaning:
0
```

In [ ]:
```
app_test_poly_temp = app_test_poly.merge(previous_loan_counts, on = 'SK_ID_CURR', l
app_test_poly_temp.head()
```

Out[ ]:

	SK_ID_CURR	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	REGI
0	100001	0	1	0	135000.0	
1	100005	0	1	0	99000.0	
2	100013	1	1	0	202500.0	
3	100028	0	1	2	315000.0	
4	100038	1	0	1	180000.0	

5 rows × 270 columns

In [ ]:
```
print('Missing Value counts in previous_loan_counts in the test merged dataset:', 
app_test_poly_temp['previous_loan_counts'].fillna(0, inplace=True)
print('Missing Value counts in previous_loan_counts in the test merged dataset afte
```

```
Missing Value counts in previous_loan_counts in the test merged dataset: 6424

Missing Value counts in previous_loan_counts in the test merged dataset after clea
ning: 0
```

In [ ]:
```
bureau_agg = bureau_data.drop(columns = ['SK_ID_BUREAU']).groupby('SK_ID_CURR', as_
bureau_agg.head()
```

Out[ ]:

	SK_ID_CURR	DAYS_CREDIT				CREDIT_DAY_OVERDUE				...	
		count	mean	max	min	sum	count	mean	max	min	...	count	
0	100001	7	-735.000000	-49	-1572	-5145	7	0.0	0	0	...	7	-9
1	100002	8	-874.000000	-103	-1437	-6992	8	0.0	0	0	...	8	-49
2	100003	4	-1400.750000	-606	-2586	-5603	4	0.0	0	0	...	4	-81
3	100004	2	-867.000000	-408	-1326	-1734	2	0.0	0	0	...	2	-53
4	100005	3	-190.666667	-62	-373	-572	3	0.0	0	0	...	3	-5

5 rows × 61 columns

Assigning appropriate column names for the new features

In [ ]:
```
# List of column names
columns = ['SK_ID_CURR']

# Iterate through the variables names
for var in bureau_agg.columns.levels[0]:
    # Skip the id name
```

```
    if var != 'SK_ID_CURR':
        # Iterate through the stat names
        for stat in bureau_agg.columns.levels[1][:-1]:
            # Make a new column name for the variable and stat
            columns.append('bureau_%s_%s' % (var, stat))
```

In [ ]:
```
# Assign the list of columns names as the dataframe column names
bureau_agg.columns = columns
bureau_agg.head()
```

Out[ ]:

	SK_ID_CURR	bureau_DAYS_CREDIT_count	bureau_DAYS_CREDIT_mean	bureau_DAYS_CREDIT_max
0	100001	7	-735.000000	-49
1	100002	8	-874.000000	-103
2	100003	4	-1400.750000	-606
3	100004	2	-867.000000	-408
4	100005	3	-190.666667	-62

5 rows × 61 columns

Join the aggregated bureau dataset with the labeled dataset

In [ ]:
```
# Merge with the training data
app_train_poly_temp = app_train_poly_temp.merge(bureau_agg, on = 'SK_ID_CURR', how
app_train_poly_temp.head()
```

Out[ ]:

	SK_ID_CURR	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	REGI
0	100002	0	1	0	202500.0	
1	100003	0	0	0	270000.0	
2	100004	1	1	0	67500.0	
3	100006	0	1	0	135000.0	
4	100007	0	1	0	121500.0	

5 rows × 331 columns

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

In [ ]:
```
# Merge with the test data
app_test_poly_temp = app_test_poly_temp.merge(bureau_agg, on = 'SK_ID_CURR', how =
app_test_poly_temp.head()
```

In [ ]:
```
app_train_poly_temp.shape, app_test_poly_temp.shape
```

Processing pipeline

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

In [ ]:
```
class DataFrameSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attribute_names):
        self.attribute_names = attribute_names
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.attribute_names].values
```

In [ ]:
```
print(app_train_poly_temp.dtypes[app_train_poly_temp.dtypes == 'float64'].count())
print(app_train_poly_temp.dtypes[app_train_poly_temp.dtypes == 'int64'].count())
app_train_poly_temp.shape
```

Out[ ]:
```
160

171
(307511, 331)
```

In [ ]:
```
num_attributes = []
cat_attributes = []
num_attributes_int = app_train_poly_temp.select_dtypes(include=['int'])
for col in num_attributes_int:
    if len(list(num_attributes_int[col].unique())) > 3:
        num_attributes.append(col)
    else:
        cat_attributes.append(col)

num_attributes_float = app_train_poly_temp.select_dtypes(include=['float']).columns
for col in num_attributes_float:
    num_attributes.append(col)
print(num_attributes)
print(cat_attributes)
```

```
['SK_ID_CURR', 'CNT_CHILDREN', 'DAYS_ID_PUBLISH', 'AMT_INCOME_TOTAL', 'REGION_POPU
LATION_RELATIVE', 'DAYS_BIRTH_x', 'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'OWN_CAR_A
GE', 'CNT_FAM_MEMBERS', 'EXT_SOURCE_1_x', 'EXT_SOURCE_2_x', 'EXT_SOURCE_3_x', 'APA
RTMENTS_AVG', 'BASEMENTAREA_AVG', 'YEARS_BEGINEXPLUATATION_AVG', 'YEARS_BUILD_AV
G', 'COMMONAREA_AVG', 'ELEVATORS_AVG', 'ENTRANCES_AVG', 'FLOORSMAX_AVG', 'FLOORSMI
N_AVG', 'LANDAREA_AVG', 'LIVINGAPARTMENTS_AVG', 'LIVINGAREA_AVG', 'NONLIVINGAPARTM
ENTS_AVG', 'NONLIVINGAREA_AVG', 'APARTMENTS_MODE', 'BASEMENTAREA_MODE', 'YEARS_BEG
INEXPLUATATION_MODE', 'YEARS_BUILD_MODE', 'COMMONAREA_MODE', 'ELEVATORS_MODE', 'EN
TRANCES_MODE', 'FLOORSMAX_MODE', 'FLOORSMIN_MODE', 'LANDAREA_MODE', 'LIVINGAPARTME
NTS_MODE', 'LIVINGAREA_MODE', 'NONLIVINGAPARTMENTS_MODE', 'NONLIVINGAREA_MODE', 'A
PARTMENTS_MEDI', 'BASEMENTAREA_MEDI', 'YEARS_BEGINEXPLUATATION_MEDI', 'YEARS_BUILD
_MEDI', 'COMMONAREA_MEDI', 'ELEVATORS_MEDI', 'ENTRANCES_MEDI', 'FLOORSMAX_MEDI',
'FLOORSMIN_MEDI', 'LANDAREA_MEDI', 'LIVINGAPARTMENTS_MEDI', 'LIVINGAREA_MEDI', 'NO
NLIVINGAPARTMENTS_MEDI', 'NONLIVINGAREA_MEDI', 'TOTALAREA_MODE', 'OBS_30_CNT_SOCIA
L_CIRCLE', 'DEF_30_CNT_SOCIAL_CIRCLE', 'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOC
IAL_CIRCLE', 'DAYS_LAST_PHONE_CHANGE', 'AMT_REQ_CREDIT_BUREAU_HOUR', 'AMT_REQ_CRED
IT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK', 'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_RE
Q_CREDIT_BUREAU_QRT', 'AMT_REQ_CREDIT_BUREAU_YEAR', '1', 'EXT_SOURCE_1_y', 'EXT_SO
URCE_2_y', 'EXT_SOURCE_3_y', 'DAYS_BIRTH_y', 'EXT_SOURCE_1^2', 'EXT_SOURCE_1 EXT_S
OURCE_2', 'EXT_SOURCE_1 EXT_SOURCE_3', 'EXT_SOURCE_1 DAYS_BIRTH', 'EXT_SOURCE_2^
2', 'EXT_SOURCE_2 EXT_SOURCE_3', 'EXT_SOURCE_2 DAYS_BIRTH', 'EXT_SOURCE_3^2', 'EXT
_SOURCE_3 DAYS_BIRTH', 'DAYS_BIRTH^2', 'EXT_SOURCE_1^3', 'EXT_SOURCE_1^2 EXT_SOURC
E_2', 'EXT_SOURCE_1^2 EXT_SOURCE_3', 'EXT_SOURCE_1^2 DAYS_BIRTH', 'EXT_SOURCE_1 EX
T_SOURCE_2^2', 'EXT_SOURCE_1 EXT_SOURCE_2 EXT_SOURCE_3', 'EXT_SOURCE_1 EXT_SOURCE_
2 DAYS_BIRTH', 'EXT_SOURCE_1 EXT_SOURCE_3^2', 'EXT_SOURCE_1 EXT_SOURCE_3 DAYS_BIRT
H', 'EXT_SOURCE_1 DAYS_BIRTH^2', 'EXT_SOURCE_2^3', 'EXT_SOURCE_2^2 EXT_SOURCE_3',
'EXT_SOURCE_2^2 DAYS_BIRTH', 'EXT_SOURCE_2 EXT_SOURCE_3^2', 'EXT_SOURCE_2 EXT_SOUR
CE_3 DAYS_BIRTH', 'EXT_SOURCE_2 DAYS_BIRTH^2', 'EXT_SOURCE_3^3', 'EXT_SOURCE_3^2 D
AYS_BIRTH', 'EXT_SOURCE_3 DAYS_BIRTH^2', 'DAYS_BIRTH^3', 'previous_loan_counts',
'bureau_DAYS_CREDIT_count', 'bureau_DAYS_CREDIT_mean', 'bureau_DAYS_CREDIT_max',
'bureau_DAYS_CREDIT_min', 'bureau_DAYS_CREDIT_sum', 'bureau_CREDIT_DAY_OVERDUE_cou
nt', 'bureau_CREDIT_DAY_OVERDUE_mean', 'bureau_CREDIT_DAY_OVERDUE_max', 'bureau_CR
EDIT_DAY_OVERDUE_min', 'bureau_CREDIT_DAY_OVERDUE_sum', 'bureau_DAYS_CREDIT_ENDDAT
E_count', 'bureau_DAYS_CREDIT_ENDDATE_mean', 'bureau_DAYS_CREDIT_ENDDATE_max', 'bu
reau_DAYS_CREDIT_ENDDATE_min', 'bureau_DAYS_CREDIT_ENDDATE_sum', 'bureau_DAYS_ENDD
ATE_FACT_count', 'bureau_DAYS_ENDDATE_FACT_mean', 'bureau_DAYS_ENDDATE_FACT_max',
'bureau_DAYS_ENDDATE_FACT_min', 'bureau_DAYS_ENDDATE_FACT_sum', 'bureau_AMT_CREDIT
_MAX_OVERDUE_count', 'bureau_AMT_CREDIT_MAX_OVERDUE_mean', 'bureau_AMT_CREDIT_MAX_
OVERDUE_max', 'bureau_AMT_CREDIT_MAX_OVERDUE_min', 'bureau_AMT_CREDIT_MAX_OVERDUE_
sum', 'bureau_CNT_CREDIT_PROLONG_count', 'bureau_CNT_CREDIT_PROLONG_mean', 'bureau
_CNT_CREDIT_PROLONG_max', 'bureau_CNT_CREDIT_PROLONG_min', 'bureau_CNT_CREDIT_PROL
ONG_sum', 'bureau_AMT_CREDIT_SUM_count', 'bureau_AMT_CREDIT_SUM_mean', 'bureau_AMT
_CREDIT_SUM_max', 'bureau_AMT_CREDIT_SUM_min', 'bureau_AMT_CREDIT_SUM_sum', 'burea
u_AMT_CREDIT_SUM_DEBT_count', 'bureau_AMT_CREDIT_SUM_DEBT_mean', 'bureau_AMT_CREDI
T_SUM_DEBT_max', 'bureau_AMT_CREDIT_SUM_DEBT_min', 'bureau_AMT_CREDIT_SUM_DEBT_su
m', 'bureau_AMT_CREDIT_SUM_LIMIT_count', 'bureau_AMT_CREDIT_SUM_LIMIT_mean', 'bure
au_AMT_CREDIT_SUM_LIMIT_max', 'bureau_AMT_CREDIT_SUM_LIMIT_min', 'bureau_AMT_CREDI
T_SUM_LIMIT_sum', 'bureau_AMT_CREDIT_SUM_OVERDUE_count', 'bureau_AMT_CREDIT_SUM_OV
ERDUE_mean', 'bureau_AMT_CREDIT_SUM_OVERDUE_max', 'bureau_AMT_CREDIT_SUM_OVERDUE_m
in', 'bureau_AMT_CREDIT_SUM_OVERDUE_sum', 'bureau_DAYS_CREDIT_UPDATE_count', 'bure
au_DAYS_CREDIT_UPDATE_mean', 'bureau_DAYS_CREDIT_UPDATE_max', 'bureau_DAYS_CREDIT_
UPDATE_min', 'bureau_DAYS_CREDIT_UPDATE_sum', 'bureau_AMT_ANNUITY_count', 'bureau_
AMT_ANNUITY_mean', 'bureau_AMT_ANNUITY_max', 'bureau_AMT_ANNUITY_min', 'bureau_AMT
_ANNUITY_sum']
```

```
['FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'FLAG_MOBIL', 'FLAG_EMP_PHONE', 'FLAG_WORK_PHO
NE', 'FLAG_CONT_MOBILE', 'FLAG_PHONE', 'FLAG_EMAIL', 'REGION_RATING_CLIENT', 'REGI
ON_RATING_CLIENT_W_CITY', 'REG_REGION_NOT_LIVE_REGION', 'REG_REGION_NOT_WORK_REGIO
N', 'LIVE_REGION_NOT_WORK_REGION', 'REG_CITY_NOT_LIVE_CITY', 'REG_CITY_NOT_WORK_CI
TY', 'LIVE_CITY_NOT_WORK_CITY', 'FLAG_DOCUMENT_2', 'FLAG_DOCUMENT_3', 'FLAG_DOCUME
NT_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', 'FL
AG_DOCUMENT_13', 'FLAG_DOCUMENT_14', 'FLAG_DOCUMENT_15', 'FLAG_DOCUMENT_16', 'FLAG
_DOCUMENT_17', 'FLAG_DOCUMENT_18', 'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20', 'FLAG_D
```

```
OCUMENT_21', 'CODE_GENDER_F', 'CODE_GENDER_M', 'NAME_TYPE_SUITE_Children', 'NAME_T
YPE_SUITE_Family', 'NAME_TYPE_SUITE_Group of people', 'NAME_TYPE_SUITE_Other_A',
'NAME_TYPE_SUITE_Other_B', 'NAME_TYPE_SUITE_Spouse, partner', 'NAME_TYPE_SUITE_Una
ccompanied', 'NAME_INCOME_TYPE_Businessman', 'NAME_INCOME_TYPE_Commercial associat
e', 'NAME_INCOME_TYPE_Pensioner', 'NAME_INCOME_TYPE_State servant', 'NAME_INCOME_T
YPE_Student', 'NAME_INCOME_TYPE_Unemployed', 'NAME_INCOME_TYPE_Working', 'NAME_EDU
CATION_TYPE_Academic degree', 'NAME_EDUCATION_TYPE_Higher education', 'NAME_EDUCAT
ION_TYPE_Incomplete higher', 'NAME_EDUCATION_TYPE_Lower secondary', 'NAME_EDUCATIO
N_TYPE_Secondary / secondary special', 'NAME_FAMILY_STATUS_Civil marriage', 'NAME_
FAMILY_STATUS_Married', 'NAME_FAMILY_STATUS_Separated', 'NAME_FAMILY_STATUS_Single
/ not married', 'NAME_FAMILY_STATUS_Widow', 'NAME_HOUSING_TYPE_Co-op apartment',
'NAME_HOUSING_TYPE_House / apartment', 'NAME_HOUSING_TYPE_Municipal apartment', 'N
AME_HOUSING_TYPE_Office apartment', 'NAME_HOUSING_TYPE_Rented apartment', 'NAME_HO
USING_TYPE_With parents', 'OCCUPATION_TYPE_Accountants', 'OCCUPATION_TYPE_Cleaning
staff', 'OCCUPATION_TYPE_Cooking staff', 'OCCUPATION_TYPE_Core staff', 'OCCUPATION
_TYPE_Drivers', 'OCCUPATION_TYPE_HR staff', 'OCCUPATION_TYPE_High skill tech staf
f', 'OCCUPATION_TYPE_IT staff', 'OCCUPATION_TYPE_Laborers', 'OCCUPATION_TYPE_Low-s
kill Laborers', 'OCCUPATION_TYPE_Managers', 'OCCUPATION_TYPE_Medicine staff', 'OCC
UPATION_TYPE_Private service staff', 'OCCUPATION_TYPE_Realty agents', 'OCCUPATION_
TYPE_Sales staff', 'OCCUPATION_TYPE_Secretaries', 'OCCUPATION_TYPE_Security staf
f', 'OCCUPATION_TYPE_Waiters/barmen staff', 'WEEKDAY_APPR_PROCESS_START_FRIDAY',
'WEEKDAY_APPR_PROCESS_START_MONDAY', 'WEEKDAY_APPR_PROCESS_START_SATURDAY', 'WEEKD
AY_APPR_PROCESS_START_SUNDAY', 'WEEKDAY_APPR_PROCESS_START_THURSDAY', 'WEEKDAY_APP
R_PROCESS_START_TUESDAY', 'WEEKDAY_APPR_PROCESS_START_WEDNESDAY', 'ORGANIZATION_TY
PE_Advertising', 'ORGANIZATION_TYPE_Agriculture', 'ORGANIZATION_TYPE_Bank', 'ORGAN
IZATION_TYPE_Business Entity Type 1', 'ORGANIZATION_TYPE_Business Entity Type 2',
'ORGANIZATION_TYPE_Business Entity Type 3', 'ORGANIZATION_TYPE_Cleaning', 'ORGANIZ
ATION_TYPE_Construction', 'ORGANIZATION_TYPE_Culture', 'ORGANIZATION_TYPE_Electric
ity', 'ORGANIZATION_TYPE_Emergency', 'ORGANIZATION_TYPE_Government', 'ORGANIZATION
_TYPE_Hotel', 'ORGANIZATION_TYPE_Housing', 'ORGANIZATION_TYPE_Industry: type 1',
'ORGANIZATION_TYPE_Industry: type 10', 'ORGANIZATION_TYPE_Industry: type 11', 'ORG
ANIZATION_TYPE_Industry: type 12', 'ORGANIZATION_TYPE_Industry: type 13', 'ORGANIZ
ATION_TYPE_Industry: type 2', 'ORGANIZATION_TYPE_Industry: type 3', 'ORGANIZATION_
TYPE_Industry: type 4', 'ORGANIZATION_TYPE_Industry: type 5', 'ORGANIZATION_TYPE_I
ndustry: type 6', 'ORGANIZATION_TYPE_Industry: type 7', 'ORGANIZATION_TYPE_Industr
y: type 8', 'ORGANIZATION_TYPE_Industry: type 9', 'ORGANIZATION_TYPE_Insurance',
'ORGANIZATION_TYPE_Kindergarten', 'ORGANIZATION_TYPE_Legal Services', 'ORGANIZATIO
N_TYPE_Medicine', 'ORGANIZATION_TYPE_Military', 'ORGANIZATION_TYPE_Mobile', 'ORGAN
IZATION_TYPE_Other', 'ORGANIZATION_TYPE_Police', 'ORGANIZATION_TYPE_Postal', 'ORGA
NIZATION_TYPE_Realtor', 'ORGANIZATION_TYPE_Religion', 'ORGANIZATION_TYPE_Restauran
t', 'ORGANIZATION_TYPE_School', 'ORGANIZATION_TYPE_Security', 'ORGANIZATION_TYPE_S
ecurity Ministries', 'ORGANIZATION_TYPE_Self-employed', 'ORGANIZATION_TYPE_Service
s', 'ORGANIZATION_TYPE_Telecom', 'ORGANIZATION_TYPE_Trade: type 1', 'ORGANIZATION_
TYPE_Trade: type 2', 'ORGANIZATION_TYPE_Trade: type 3', 'ORGANIZATION_TYPE_Trade:
type 4', 'ORGANIZATION_TYPE_Trade: type 5', 'ORGANIZATION_TYPE_Trade: type 6', 'OR
GANIZATION_TYPE_Trade: type 7', 'ORGANIZATION_TYPE_Transport: type 1', 'ORGANIZATI
ON_TYPE_Transport: type 2', 'ORGANIZATION_TYPE_Transport: type 3', 'ORGANIZATION_T
YPE_Transport: type 4', 'ORGANIZATION_TYPE_University', 'ORGANIZATION_TYPE_XNA',
'FONDKAPREMONT_MODE_not specified', 'FONDKAPREMONT_MODE_org spec account', 'FONDKA
PREMONT_MODE_reg oper account', 'FONDKAPREMONT_MODE_reg oper spec account', 'HOUSE
TYPE_MODE_block of flats', 'HOUSETYPE_MODE_specific housing', 'HOUSETYPE_MODE_terr
aced house', 'WALLSMATERIAL_MODE_Block', 'WALLSMATERIAL_MODE_Mixed', 'WALLSMATERIA
L_MODE_Monolithic', 'WALLSMATERIAL_MODE_Others', 'WALLSMATERIAL_MODE_Panel', 'WALL
SMATERIAL_MODE_Stone, brick', 'WALLSMATERIAL_MODE_Wooden', 'EMERGENCYSTATE_MODE_N
o', 'EMERGENCYSTATE_MODE_Yes', 'TARGET']
```

In [ ]:

```
# # Split the provided training data into training, validation, and test
# # The kaggle evaluation test set has no labels

num_attributes = []
cat_attributes = []
num_attributes_int = app_train_poly_temp.select_dtypes(include=['int'])
for col in num_attributes_int:
    if len(list(num_attributes_int[col].unique())) > 3:
```

```
        num_attributes.append(col)
    else:
        cat_attributes.append(col)

num_attributes_float = app_train_poly_temp.select_dtypes(include=['float']).columns
for col in num_attributes_float:
    num_attributes.append(col)

train_dataset = app_train_poly_temp
class_labels = ["No Default","Default"]

from sklearn.model_selection import train_test_split

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


cat_attribs = ['FLAG_OWN_REALTY','FLAG_OWN_CAR','NAME_CONTRACT_TYPE', 'FLAG_OWN_CAR

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

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

selected_features = num_attributes + cat_attributes
total_features = f"{len(selected_features)}:   Num:{len(num_attributes)},    Cat:{
#Total Feature selected for processing
total_features
```

Out[ ]:
```
'331:   Num:163,    Cat:168'
```

In [ ]:
```
list(selected_features)
```

Out[ ]:
```
['SK_ID_CURR',
 'CNT_CHILDREN',
 'DAYS_ID_PUBLISH',
 'AMT_INCOME_TOTAL',
 'REGION_POPULATION_RELATIVE',
 'DAYS_BIRTH_x',
 'DAYS_EMPLOYED',
 'DAYS_REGISTRATION',
 'OWN_CAR_AGE',
 'CNT_FAM_MEMBERS',
 'EXT_SOURCE_1_x',
 'EXT_SOURCE_2_x',
 'EXT_SOURCE_3_x',
 'APARTMENTS_AVG',
 'BASEMENTAREA_AVG',
 'YEARS_BEGINEXPLUATATION_AVG',
 'YEARS_BUILD_AVG',
 'COMMONAREA_AVG',
 'ELEVATORS_AVG',
 'ENTRANCES_AVG',
 'FLOORSMAX_AVG',
 'FLOORSMIN_AVG',
 'LANDAREA_AVG',
 'LIVINGAPARTMENTS_AVG',
 'LIVINGAREA_AVG',
 'NONLIVINGAPARTMENTS_AVG',
 'NONLIVINGAREA_AVG',
 'APARTMENTS_MODE',
 'BASEMENTAREA_MODE',
 'YEARS_BEGINEXPLUATATION_MODE',
 'YEARS_BUILD_MODE',
 'COMMONAREA_MODE',
 'ELEVATORS_MODE',
 'ENTRANCES_MODE',
 'FLOORSMAX_MODE',
 'FLOORSMIN_MODE',
 'LANDAREA_MODE',
 'LIVINGAPARTMENTS_MODE',
 'LIVINGAREA_MODE',
 'NONLIVINGAPARTMENTS_MODE',
 'NONLIVINGAREA_MODE',
 'APARTMENTS_MEDI',
 'BASEMENTAREA_MEDI',
 'YEARS_BEGINEXPLUATATION_MEDI',
 'YEARS_BUILD_MEDI',
 'COMMONAREA_MEDI',
 'ELEVATORS_MEDI',
 'ENTRANCES_MEDI',
 'FLOORSMAX_MEDI',
 'FLOORSMIN_MEDI',
 'LANDAREA_MEDI',
 'LIVINGAPARTMENTS_MEDI',
 'LIVINGAREA_MEDI',
 'NONLIVINGAPARTMENTS_MEDI',
 'NONLIVINGAREA_MEDI',
 'TOTALAREA_MODE',
 'OBS_30_CNT_SOCIAL_CIRCLE',
 'DEF_30_CNT_SOCIAL_CIRCLE',
 'OBS_60_CNT_SOCIAL_CIRCLE',
 'DEF_60_CNT_SOCIAL_CIRCLE',
 'DAYS_LAST_PHONE_CHANGE',
 'AMT_REQ_CREDIT_BUREAU_HOUR',
 'AMT_REQ_CREDIT_BUREAU_DAY',
 'AMT_REQ_CREDIT_BUREAU_WEEK',
```

```
'AMT_REQ_CREDIT_BUREAU_MON',
'AMT_REQ_CREDIT_BUREAU_QRT',
'AMT_REQ_CREDIT_BUREAU_YEAR',
'1',
'EXT_SOURCE_1_y',
'EXT_SOURCE_2_y',
'EXT_SOURCE_3_y',
'DAYS_BIRTH_y',
'EXT_SOURCE_1^2',
'EXT_SOURCE_1 EXT_SOURCE_2',
'EXT_SOURCE_1 EXT_SOURCE_3',
'EXT_SOURCE_1 DAYS_BIRTH',
'EXT_SOURCE_2^2',
'EXT_SOURCE_2 EXT_SOURCE_3',
'EXT_SOURCE_2 DAYS_BIRTH',
'EXT_SOURCE_3^2',
'EXT_SOURCE_3 DAYS_BIRTH',
'DAYS_BIRTH^2',
'EXT_SOURCE_1^3',
'EXT_SOURCE_1^2 EXT_SOURCE_2',
'EXT_SOURCE_1^2 EXT_SOURCE_3',
'EXT_SOURCE_1^2 DAYS_BIRTH',
'EXT_SOURCE_1 EXT_SOURCE_2^2',
'EXT_SOURCE_1 EXT_SOURCE_2 EXT_SOURCE_3',
'EXT_SOURCE_1 EXT_SOURCE_2 DAYS_BIRTH',
'EXT_SOURCE_1 EXT_SOURCE_3^2',
'EXT_SOURCE_1 EXT_SOURCE_3 DAYS_BIRTH',
'EXT_SOURCE_1 DAYS_BIRTH^2',
'EXT_SOURCE_2^3',
'EXT_SOURCE_2^2 EXT_SOURCE_3',
'EXT_SOURCE_2^2 DAYS_BIRTH',
'EXT_SOURCE_2 EXT_SOURCE_3^2',
'EXT_SOURCE_2 EXT_SOURCE_3 DAYS_BIRTH',
'EXT_SOURCE_2 DAYS_BIRTH^2',
'EXT_SOURCE_3^3',
'EXT_SOURCE_3^2 DAYS_BIRTH',
'EXT_SOURCE_3 DAYS_BIRTH^2',
'DAYS_BIRTH^3',
'previous_loan_counts',
'bureau_DAYS_CREDIT_count',
'bureau_DAYS_CREDIT_mean',
'bureau_DAYS_CREDIT_max',
'bureau_DAYS_CREDIT_min',
'bureau_DAYS_CREDIT_sum',
'bureau_CREDIT_DAY_OVERDUE_count',
'bureau_CREDIT_DAY_OVERDUE_mean',
'bureau_CREDIT_DAY_OVERDUE_max',
'bureau_CREDIT_DAY_OVERDUE_min',
'bureau_CREDIT_DAY_OVERDUE_sum',
'bureau_DAYS_CREDIT_ENDDATE_count',
'bureau_DAYS_CREDIT_ENDDATE_mean',
'bureau_DAYS_CREDIT_ENDDATE_max',
'bureau_DAYS_CREDIT_ENDDATE_min',
'bureau_DAYS_CREDIT_ENDDATE_sum',
'bureau_DAYS_ENDDATE_FACT_count',
'bureau_DAYS_ENDDATE_FACT_mean',
'bureau_DAYS_ENDDATE_FACT_max',
'bureau_DAYS_ENDDATE_FACT_min',
'bureau_DAYS_ENDDATE_FACT_sum',
'bureau_AMT_CREDIT_MAX_OVERDUE_count',
'bureau_AMT_CREDIT_MAX_OVERDUE_mean',
'bureau_AMT_CREDIT_MAX_OVERDUE_max',
'bureau_AMT_CREDIT_MAX_OVERDUE_min',
'bureau_AMT_CREDIT_MAX_OVERDUE_sum',
```

```
                'bureau_CNT_CREDIT_PROLONG_count',
                'bureau_CNT_CREDIT_PROLONG_mean',
                'bureau_CNT_CREDIT_PROLONG_max',
                'bureau_CNT_CREDIT_PROLONG_min',
                'bureau_CNT_CREDIT_PROLONG_sum',
                'bureau_AMT_CREDIT_SUM_count',
                'bureau_AMT_CREDIT_SUM_mean',
                'bureau_AMT_CREDIT_SUM_max',
                'bureau_AMT_CREDIT_SUM_min',
                'bureau_AMT_CREDIT_SUM_sum',
                'bureau_AMT_CREDIT_SUM_DEBT_count',
                'bureau_AMT_CREDIT_SUM_DEBT_mean',
                'bureau_AMT_CREDIT_SUM_DEBT_max',
                'bureau_AMT_CREDIT_SUM_DEBT_min',
                'bureau_AMT_CREDIT_SUM_DEBT_sum',
                'bureau_AMT_CREDIT_SUM_LIMIT_count',
                'bureau_AMT_CREDIT_SUM_LIMIT_mean',
                'bureau_AMT_CREDIT_SUM_LIMIT_max',
                'bureau_AMT_CREDIT_SUM_LIMIT_min',
                'bureau_AMT_CREDIT_SUM_LIMIT_sum',
                'bureau_AMT_CREDIT_SUM_OVERDUE_count',
                'bureau_AMT_CREDIT_SUM_OVERDUE_mean',
                'bureau_AMT_CREDIT_SUM_OVERDUE_max',
                'bureau_AMT_CREDIT_SUM_OVERDUE_min',
                'bureau_AMT_CREDIT_SUM_OVERDUE_sum',
                'bureau_DAYS_CREDIT_UPDATE_count',
                'bureau_DAYS_CREDIT_UPDATE_mean',
                'bureau_DAYS_CREDIT_UPDATE_max',
                'bureau_DAYS_CREDIT_UPDATE_min',
                'bureau_DAYS_CREDIT_UPDATE_sum',
                'bureau_AMT_ANNUITY_count',
                'bureau_AMT_ANNUITY_mean',
                'bureau_AMT_ANNUITY_max',
                'bureau_AMT_ANNUITY_min',
                'bureau_AMT_ANNUITY_sum',
                'FLAG_OWN_CAR',
                'FLAG_OWN_REALTY',
                'FLAG_MOBIL',
                'FLAG_EMP_PHONE',
                'FLAG_WORK_PHONE',
                'FLAG_CONT_MOBILE',
                'FLAG_PHONE',
                'FLAG_EMAIL',
                'REGION_RATING_CLIENT',
                'REGION_RATING_CLIENT_W_CITY',
                'REG_REGION_NOT_LIVE_REGION',
                'REG_REGION_NOT_WORK_REGION',
                'LIVE_REGION_NOT_WORK_REGION',
                'REG_CITY_NOT_LIVE_CITY',
                'REG_CITY_NOT_WORK_CITY',
                'LIVE_CITY_NOT_WORK_CITY',
                'FLAG_DOCUMENT_2',
                'FLAG_DOCUMENT_3',
                'FLAG_DOCUMENT_4',
                'FLAG_DOCUMENT_5',
                'FLAG_DOCUMENT_6',
                'FLAG_DOCUMENT_7',
                'FLAG_DOCUMENT_8',
                'FLAG_DOCUMENT_9',
                'FLAG_DOCUMENT_10',
                'FLAG_DOCUMENT_11',
                'FLAG_DOCUMENT_12',
                'FLAG_DOCUMENT_13',
                'FLAG_DOCUMENT_14',
```

```
'FLAG_DOCUMENT_15',
'FLAG_DOCUMENT_16',
'FLAG_DOCUMENT_17',
'FLAG_DOCUMENT_18',
'FLAG_DOCUMENT_19',
'FLAG_DOCUMENT_20',
'FLAG_DOCUMENT_21',
'CODE_GENDER_F',
'CODE_GENDER_M',
'NAME_TYPE_SUITE_Children',
'NAME_TYPE_SUITE_Family',
'NAME_TYPE_SUITE_Group of people',
'NAME_TYPE_SUITE_Other_A',
'NAME_TYPE_SUITE_Other_B',
'NAME_TYPE_SUITE_Spouse, partner',
'NAME_TYPE_SUITE_Unaccompanied',
'NAME_INCOME_TYPE_Businessman',
'NAME_INCOME_TYPE_Commercial associate',
'NAME_INCOME_TYPE_Pensioner',
'NAME_INCOME_TYPE_State servant',
'NAME_INCOME_TYPE_Student',
'NAME_INCOME_TYPE_Unemployed',
'NAME_INCOME_TYPE_Working',
'NAME_EDUCATION_TYPE_Academic degree',
'NAME_EDUCATION_TYPE_Higher education',
'NAME_EDUCATION_TYPE_Incomplete higher',
'NAME_EDUCATION_TYPE_Lower secondary',
'NAME_EDUCATION_TYPE_Secondary / secondary special',
'NAME_FAMILY_STATUS_Civil marriage',
'NAME_FAMILY_STATUS_Married',
'NAME_FAMILY_STATUS_Separated',
'NAME_FAMILY_STATUS_Single / not married',
'NAME_FAMILY_STATUS_Widow',
'NAME_HOUSING_TYPE_Co-op apartment',
'NAME_HOUSING_TYPE_House / apartment',
'NAME_HOUSING_TYPE_Municipal apartment',
'NAME_HOUSING_TYPE_Office apartment',
'NAME_HOUSING_TYPE_Rented apartment',
'NAME_HOUSING_TYPE_With parents',
'OCCUPATION_TYPE_Accountants',
'OCCUPATION_TYPE_Cleaning staff',
'OCCUPATION_TYPE_Cooking staff',
'OCCUPATION_TYPE_Core staff',
'OCCUPATION_TYPE_Drivers',
'OCCUPATION_TYPE_HR staff',
'OCCUPATION_TYPE_High skill tech staff',
'OCCUPATION_TYPE_IT staff',
'OCCUPATION_TYPE_Laborers',
'OCCUPATION_TYPE_Low-skill Laborers',
'OCCUPATION_TYPE_Managers',
'OCCUPATION_TYPE_Medicine staff',
'OCCUPATION_TYPE_Private service staff',
'OCCUPATION_TYPE_Realty agents',
'OCCUPATION_TYPE_Sales staff',
'OCCUPATION_TYPE_Secretaries',
'OCCUPATION_TYPE_Security staff',
'OCCUPATION_TYPE_Waiters/barmen staff',
'WEEKDAY_APPR_PROCESS_START_FRIDAY',
'WEEKDAY_APPR_PROCESS_START_MONDAY',
'WEEKDAY_APPR_PROCESS_START_SATURDAY',
'WEEKDAY_APPR_PROCESS_START_SUNDAY',
'WEEKDAY_APPR_PROCESS_START_THURSDAY',
'WEEKDAY_APPR_PROCESS_START_TUESDAY',
'WEEKDAY_APPR_PROCESS_START_WEDNESDAY',
```

```
'ORGANIZATION_TYPE_Advertising',
'ORGANIZATION_TYPE_Agriculture',
'ORGANIZATION_TYPE_Bank',
'ORGANIZATION_TYPE_Business Entity Type 1',
'ORGANIZATION_TYPE_Business Entity Type 2',
'ORGANIZATION_TYPE_Business Entity Type 3',
'ORGANIZATION_TYPE_Cleaning',
'ORGANIZATION_TYPE_Construction',
'ORGANIZATION_TYPE_Culture',
'ORGANIZATION_TYPE_Electricity',
'ORGANIZATION_TYPE_Emergency',
'ORGANIZATION_TYPE_Government',
'ORGANIZATION_TYPE_Hotel',
'ORGANIZATION_TYPE_Housing',
'ORGANIZATION_TYPE_Industry: type 1',
'ORGANIZATION_TYPE_Industry: type 10',
'ORGANIZATION_TYPE_Industry: type 11',
'ORGANIZATION_TYPE_Industry: type 12',
'ORGANIZATION_TYPE_Industry: type 13',
'ORGANIZATION_TYPE_Industry: type 2',
'ORGANIZATION_TYPE_Industry: type 3',
'ORGANIZATION_TYPE_Industry: type 4',
'ORGANIZATION_TYPE_Industry: type 5',
'ORGANIZATION_TYPE_Industry: type 6',
'ORGANIZATION_TYPE_Industry: type 7',
'ORGANIZATION_TYPE_Industry: type 8',
'ORGANIZATION_TYPE_Industry: type 9',
'ORGANIZATION_TYPE_Insurance',
'ORGANIZATION_TYPE_Kindergarten',
'ORGANIZATION_TYPE_Legal Services',
'ORGANIZATION_TYPE_Medicine',
'ORGANIZATION_TYPE_Military',
'ORGANIZATION_TYPE_Mobile',
'ORGANIZATION_TYPE_Other',
'ORGANIZATION_TYPE_Police',
'ORGANIZATION_TYPE_Postal',
'ORGANIZATION_TYPE_Realtor',
'ORGANIZATION_TYPE_Religion',
'ORGANIZATION_TYPE_Restaurant',
'ORGANIZATION_TYPE_School',
'ORGANIZATION_TYPE_Security',
'ORGANIZATION_TYPE_Security Ministries',
'ORGANIZATION_TYPE_Self-employed',
'ORGANIZATION_TYPE_Services',
'ORGANIZATION_TYPE_Telecom',
'ORGANIZATION_TYPE_Trade: type 1',
'ORGANIZATION_TYPE_Trade: type 2',
'ORGANIZATION_TYPE_Trade: type 3',
'ORGANIZATION_TYPE_Trade: type 4',
'ORGANIZATION_TYPE_Trade: type 5',
'ORGANIZATION_TYPE_Trade: type 6',
'ORGANIZATION_TYPE_Trade: type 7',
'ORGANIZATION_TYPE_Transport: type 1',
'ORGANIZATION_TYPE_Transport: type 2',
'ORGANIZATION_TYPE_Transport: type 3',
'ORGANIZATION_TYPE_Transport: type 4',
'ORGANIZATION_TYPE_University',
'ORGANIZATION_TYPE_XNA',
'FONDKAPREMONT_MODE_not specified',
'FONDKAPREMONT_MODE_org spec account',
'FONDKAPREMONT_MODE_reg oper account',
'FONDKAPREMONT_MODE_reg oper spec account',
'HOUSETYPE_MODE_block of flats',
'HOUSETYPE_MODE_specific housing',
```

```
 'HOUSETYPE_MODE_terraced house',
 'WALLSMATERIAL_MODE_Block',
 'WALLSMATERIAL_MODE_Mixed',
 'WALLSMATERIAL_MODE_Monolithic',
 'WALLSMATERIAL_MODE_Others',
 'WALLSMATERIAL_MODE_Panel',
 'WALLSMATERIAL_MODE_Stone, brick',
 'WALLSMATERIAL_MODE_Wooden',
 'EMERGENCYSTATE_MODE_No',
 'EMERGENCYSTATE_MODE_Yes',
 'TARGET']
```

Setting up the train, validation, and test datasets

In [ ]:
```
selected_features.remove('TARGET')
X_train = train_dataset[selected_features]
y_train = train_dataset["TARGET"]
X_kaggle_test = app_test_poly_temp[selected_features]

subsample_rate = 0.3

X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, stratify=y_t
                                                    test_size=subsample_rate, rand

X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train,stratify=y_
                                                      test_size=0.15, random_state

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

```
X train           shape: (182968, 330)

X validation      shape: (32289, 330)

X test            shape: (92254, 330)

X X_kaggle_test   shape: (48744, 330)
```

In [ ]:
```
#####----- Saving the training files for future reference if kernel fails -----####
# X_train.to_csv('X_train.csv', index=False)
# X_valid.to_csv('X_valid.csv', index=False)
# X_test.to_csv('X_test.csv', index=False)
# X_kaggle_test.to_csv('X_kaggle_test.csv', index=False)
# y_train.to_csv('y_train.csv', index=False)
# y_valid.to_csv('y_valid.csv', index=False)
# y_test.to_csv('y_test.csv', index=False)
```

In [ ]:
```
import time
import pandas as pd
import numpy as np
import os
import zipfile
from sklearn.metrics import accuracy_score, f1_score, confusion_matrix, plot_roc_c
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier, A
from sklearn.tree import DecisionTreeClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.preprocessing import LabelEncoder
from sklearn.base import BaseEstimator, TransformerMixin
import matplotlib.pyplot as plt
import seaborn as sns
```

```
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
from sklearn.decomposition import PCA
from xgboost import XGBClassifier
from catboost import CatBoostClassifier
from lightgbm import LGBMClassifier
import pytorch_lightning
import torch
import torch.nn as nn
from torchmetrics import Accuracy
import warnings
warnings.filterwarnings('ignore')

from torch.utils.data import Dataset, TensorDataset, DataLoader
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")
if torch.cuda.is_available():
  print("GPU NAME --> ", torch.cuda.get_device_name(0))
```

In [ ]:
```
X_train = pd.read_csv('/kaggle/input/train-files/X_train.csv')
X_valid = pd.read_csv('/kaggle/input/train-files/X_valid.csv')
X_test = pd.read_csv('/kaggle/input/train-files/X_test.csv')
X_kaggle_test = pd.read_csv('/kaggle/input/train-files/X_kaggle_test.csv')
y_train = pd.read_csv('/kaggle/input/train-files/y_train.csv')
y_valid = pd.read_csv('/kaggle/input/train-files/y_valid.csv')
y_test = pd.read_csv('/kaggle/input/train-files/y_test.csv')
```

In [ ]:
```
X_train.shape, y_train.shape
```

Out[ ]:
```
((182968, 330), (182968, 1))
```

In [ ]:
```
X_valid.shape, y_valid.shape
```

Out[ ]:
```
((32289, 330), (32289, 1))
```

In [ ]:
```
X_test.shape, y_test.shape
```

Out[ ]:
```
((92254, 330), (92254, 1))
```

In [ ]:
```
X_kaggle_test.shape
```

Out[ ]:
```
(48744, 330)
```

In [ ]:
```
class DataFrameSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attribute_names):
        self.attribute_names = attribute_names
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.attribute_names].values
```

In [ ]:
```
# # Split the provided training data into training, validation, and test
# # The kaggle evaluation test set has no labels
```

```
num_attributes = []
cat_attributes = []
num_attributes_int = X_train.select_dtypes(include=['int'])
for col in num_attributes_int:
    if len(list(num_attributes_int[col].unique())) > 3:
        num_attributes.append(col)
    else:
        cat_attributes.append(col)

num_attributes_float = X_train.select_dtypes(include=['float']).columns
for col in num_attributes_float:
    num_attributes.append(col)

class_labels = ["No Default","Default"]

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

cat_pipeline = Pipeline([
        ('selector', DataFrameSelector(cat_attributes)),
        ('imputer', SimpleImputer(strategy='most_frequent'))
    ])

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

selected_features = num_attributes + cat_attributes
selected_features.remove('SK_ID_CURR')
total_features = f"Total:{len(selected_features)}:, Num:{len(num_attributes)}, Cat
# Total Feature selected for processing
print(total_features)
print(selected_features)
```

```
Total:329:, Num:163, Cat:167
['CNT_CHILDREN', 'DAYS_ID_PUBLISH', 'AMT_INCOME_TOTAL', 'REGION_POPULATION_RELATIV
E', 'DAYS_BIRTH_x', 'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'OWN_CAR_AGE', 'CNT_FAM_
MEMBERS', 'EXT_SOURCE_1_x', 'EXT_SOURCE_2_x', 'EXT_SOURCE_3_x', 'APARTMENTS_AVG',
'BASEMENTAREA_AVG', 'YEARS_BEGINEXPLUATATION_AVG', 'YEARS_BUILD_AVG', 'COMMONAREA_
AVG', 'ELEVATORS_AVG', 'ENTRANCES_AVG', 'FLOORSMAX_AVG', 'FLOORSMIN_AVG', 'LANDARE
A_AVG', 'LIVINGAPARTMENTS_AVG', 'LIVINGAREA_AVG', 'NONLIVINGAPARTMENTS_AVG', 'NONL
IVINGAREA_AVG', 'APARTMENTS_MODE', 'BASEMENTAREA_MODE', 'YEARS_BEGINEXPLUATATION_M
ODE', 'YEARS_BUILD_MODE', 'COMMONAREA_MODE', 'ELEVATORS_MODE', 'ENTRANCES_MODE',
'FLOORSMAX_MODE', 'FLOORSMIN_MODE', 'LANDAREA_MODE', 'LIVINGAPARTMENTS_MODE', 'LIV
INGAREA_MODE', 'NONLIVINGAPARTMENTS_MODE', 'NONLIVINGAREA_MODE', 'APARTMENTS_MED
I', 'BASEMENTAREA_MEDI', 'YEARS_BEGINEXPLUATATION_MEDI', 'YEARS_BUILD_MEDI', 'COMM
ONAREA_MEDI', 'ELEVATORS_MEDI', 'ENTRANCES_MEDI', 'FLOORSMAX_MEDI', 'FLOORSMIN_MED
I', 'LANDAREA_MEDI', 'LIVINGAPARTMENTS_MEDI', 'LIVINGAREA_MEDI', 'NONLIVINGAPARTME
NTS_MEDI', 'NONLIVINGAREA_MEDI', 'TOTALAREA_MODE', 'OBS_30_CNT_SOCIAL_CIRCLE', 'DE
F_30_CNT_SOCIAL_CIRCLE', 'OBS_60_CNT_SOCIAL_CIRCLE', 'DEF_60_CNT_SOCIAL_CIRCLE',
'DAYS_LAST_PHONE_CHANGE', 'AMT_REQ_CREDIT_BUREAU_HOUR', 'AMT_REQ_CREDIT_BUREAU_DA
Y', 'AMT_REQ_CREDIT_BUREAU_WEEK', 'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUR
EAU_QRT', 'AMT_REQ_CREDIT_BUREAU_YEAR', '1', 'EXT_SOURCE_1_y', 'EXT_SOURCE_2_y',
'EXT_SOURCE_3_y', 'DAYS_BIRTH_y', 'EXT_SOURCE_1^2', 'EXT_SOURCE_1 EXT_SOURCE_2',
'EXT_SOURCE_1 EXT_SOURCE_3', 'EXT_SOURCE_1 DAYS_BIRTH', 'EXT_SOURCE_2^2', 'EXT_SOU
RCE_2 EXT_SOURCE_3', 'EXT_SOURCE_2 DAYS_BIRTH', 'EXT_SOURCE_3^2', 'EXT_SOURCE_3 DA
YS_BIRTH', 'DAYS_BIRTH^2', 'EXT_SOURCE_1^3', 'EXT_SOURCE_1^2 EXT_SOURCE_2', 'EXT_S
OURCE_1^2 EXT_SOURCE_3', 'EXT_SOURCE_1^2 DAYS_BIRTH', 'EXT_SOURCE_1 EXT_SOURCE_2^
2', 'EXT_SOURCE_1 EXT_SOURCE_2 EXT_SOURCE_3', 'EXT_SOURCE_1 EXT_SOURCE_2 DAYS_BIRT
H', 'EXT_SOURCE_1 EXT_SOURCE_3^2', 'EXT_SOURCE_1 EXT_SOURCE_3 DAYS_BIRTH', 'EXT_SO
URCE_1 DAYS_BIRTH^2', 'EXT_SOURCE_2^3', 'EXT_SOURCE_2^2 EXT_SOURCE_3', 'EXT_SOURCE
_2^2 DAYS_BIRTH', 'EXT_SOURCE_2 EXT_SOURCE_3^2', 'EXT_SOURCE_2 EXT_SOURCE_3 DAYS_B
IRTH', 'EXT_SOURCE_2 DAYS_BIRTH^2', 'EXT_SOURCE_3^3', 'EXT_SOURCE_3^2 DAYS_BIRTH',
'EXT_SOURCE_3 DAYS_BIRTH^2', 'DAYS_BIRTH^3', 'previous_loan_counts', 'bureau_DAYS_
CREDIT_count', 'bureau_DAYS_CREDIT_mean', 'bureau_DAYS_CREDIT_max', 'bureau_DAYS_C
REDIT_min', 'bureau_DAYS_CREDIT_sum', 'bureau_CREDIT_DAY_OVERDUE_count', 'bureau_C
REDIT_DAY_OVERDUE_mean', 'bureau_CREDIT_DAY_OVERDUE_max', 'bureau_CREDIT_DAY_OVERD
UE_min', 'bureau_CREDIT_DAY_OVERDUE_sum', 'bureau_DAYS_CREDIT_ENDDATE_count', 'bur
eau_DAYS_CREDIT_ENDDATE_mean', 'bureau_DAYS_CREDIT_ENDDATE_max', 'bureau_DAYS_CRED
IT_ENDDATE_min', 'bureau_DAYS_CREDIT_ENDDATE_sum', 'bureau_DAYS_ENDDATE_FACT_coun
t', 'bureau_DAYS_ENDDATE_FACT_mean', 'bureau_DAYS_ENDDATE_FACT_max', 'bureau_DAYS_
ENDDATE_FACT_min', 'bureau_DAYS_ENDDATE_FACT_sum', 'bureau_AMT_CREDIT_MAX_OVERDUE_
count', 'bureau_AMT_CREDIT_MAX_OVERDUE_mean', 'bureau_AMT_CREDIT_MAX_OVERDUE_max',
'bureau_AMT_CREDIT_MAX_OVERDUE_min', 'bureau_AMT_CREDIT_MAX_OVERDUE_sum', 'bureau_
CNT_CREDIT_PROLONG_count', 'bureau_CNT_CREDIT_PROLONG_mean', 'bureau_CNT_CREDIT_PR
OLONG_max', 'bureau_CNT_CREDIT_PROLONG_min', 'bureau_CNT_CREDIT_PROLONG_sum', 'bur
eau_AMT_CREDIT_SUM_count', 'bureau_AMT_CREDIT_SUM_mean', 'bureau_AMT_CREDIT_SUM_ma
x', 'bureau_AMT_CREDIT_SUM_min', 'bureau_AMT_CREDIT_SUM_sum', 'bureau_AMT_CREDIT_S
UM_DEBT_count', 'bureau_AMT_CREDIT_SUM_DEBT_mean', 'bureau_AMT_CREDIT_SUM_DEBT_ma
x', 'bureau_AMT_CREDIT_SUM_DEBT_min', 'bureau_AMT_CREDIT_SUM_DEBT_sum', 'bureau_AM
T_CREDIT_SUM_LIMIT_count', 'bureau_AMT_CREDIT_SUM_LIMIT_mean', 'bureau_AMT_CREDIT_
SUM_LIMIT_max', 'bureau_AMT_CREDIT_SUM_LIMIT_min', 'bureau_AMT_CREDIT_SUM_LIMIT_su
m', 'bureau_AMT_CREDIT_SUM_OVERDUE_count', 'bureau_AMT_CREDIT_SUM_OVERDUE_mean',
'bureau_AMT_CREDIT_SUM_OVERDUE_max', 'bureau_AMT_CREDIT_SUM_OVERDUE_min', 'bureau_
AMT_CREDIT_SUM_OVERDUE_sum', 'bureau_DAYS_CREDIT_UPDATE_count', 'bureau_DAYS_CREDI
T_UPDATE_mean', 'bureau_DAYS_CREDIT_UPDATE_max', 'bureau_DAYS_CREDIT_UPDATE_min',
'bureau_DAYS_CREDIT_UPDATE_sum', 'bureau_AMT_ANNUITY_count', 'bureau_AMT_ANNUITY_m
ean', 'bureau_AMT_ANNUITY_max', 'bureau_AMT_ANNUITY_min', 'bureau_AMT_ANNUITY_su
m', 'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'FLAG_MOBIL', 'FLAG_EMP_PHONE', 'FLAG_WORK_
PHONE', 'FLAG_CONT_MOBILE', 'FLAG_PHONE', 'FLAG_EMAIL', 'REGION_RATING_CLIENT', 'R
EGION_RATING_CLIENT_W_CITY', 'REG_REGION_NOT_LIVE_REGION', 'REG_REGION_NOT_WORK_RE
GION', 'LIVE_REGION_NOT_WORK_REGION', 'REG_CITY_NOT_LIVE_CITY', 'REG_CITY_NOT_WORK
_CITY', 'LIVE_CITY_NOT_WORK_CITY', 'FLAG_DOCUMENT_2', 'FLAG_DOCUMENT_3', 'FLAG_DOC
UMENT_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', 'F
LAG_DOCUMENT_17', 'FLAG_DOCUMENT_18', 'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20', 'FLA
G_DOCUMENT_21', 'CODE_GENDER_F', 'CODE_GENDER_M', 'NAME_TYPE_SUITE_Children', 'NAM
```

```
E_TYPE_SUITE_Family', 'NAME_TYPE_SUITE_Group of people', 'NAME_TYPE_SUITE_Other_
A', 'NAME_TYPE_SUITE_Other_B', 'NAME_TYPE_SUITE_Spouse, partner', 'NAME_TYPE_SUITE
_Unaccompanied', 'NAME_INCOME_TYPE_Businessman', 'NAME_INCOME_TYPE_Commercial asso
ciate', 'NAME_INCOME_TYPE_Pensioner', 'NAME_INCOME_TYPE_State servant', 'NAME_INCO
ME_TYPE_Student', 'NAME_INCOME_TYPE_Unemployed', 'NAME_INCOME_TYPE_Working', 'NAME
_EDUCATION_TYPE_Academic degree', 'NAME_EDUCATION_TYPE_Higher education', 'NAME_ED
UCATION_TYPE_Incomplete higher', 'NAME_EDUCATION_TYPE_Lower secondary', 'NAME_EDUC
ATION_TYPE_Secondary / secondary special', 'NAME_FAMILY_STATUS_Civil marriage', 'N
AME_FAMILY_STATUS_Married', 'NAME_FAMILY_STATUS_Separated', 'NAME_FAMILY_STATUS_Si
ngle / not married', 'NAME_FAMILY_STATUS_Widow', 'NAME_HOUSING_TYPE_Co-op apartmen
t', 'NAME_HOUSING_TYPE_House / apartment', 'NAME_HOUSING_TYPE_Municipal apartmen
t', 'NAME_HOUSING_TYPE_Office apartment', 'NAME_HOUSING_TYPE_Rented apartment', 'N
AME_HOUSING_TYPE_With parents', 'OCCUPATION_TYPE_Accountants', 'OCCUPATION_TYPE_Cl
eaning staff', 'OCCUPATION_TYPE_Cooking staff', 'OCCUPATION_TYPE_Core staff', 'OCC
UPATION_TYPE_Drivers', 'OCCUPATION_TYPE_HR staff', 'OCCUPATION_TYPE_High skill tec
h staff', 'OCCUPATION_TYPE_IT staff', 'OCCUPATION_TYPE_Laborers', 'OCCUPATION_TYPE
_Low-skill Laborers', 'OCCUPATION_TYPE_Managers', 'OCCUPATION_TYPE_Medicine staf
f', 'OCCUPATION_TYPE_Private service staff', 'OCCUPATION_TYPE_Realty agents', 'OCC
UPATION_TYPE_Sales staff', 'OCCUPATION_TYPE_Secretaries', 'OCCUPATION_TYPE_Securit
y staff', 'OCCUPATION_TYPE_Waiters/barmen staff', 'WEEKDAY_APPR_PROCESS_START_FRID
AY', 'WEEKDAY_APPR_PROCESS_START_MONDAY', 'WEEKDAY_APPR_PROCESS_START_SATURDAY',
'WEEKDAY_APPR_PROCESS_START_SUNDAY', 'WEEKDAY_APPR_PROCESS_START_THURSDAY', 'WEEKD
AY_APPR_PROCESS_START_TUESDAY', 'WEEKDAY_APPR_PROCESS_START_WEDNESDAY', 'ORGANIZAT
ION_TYPE_Advertising', 'ORGANIZATION_TYPE_Agriculture', 'ORGANIZATION_TYPE_Bank',
'ORGANIZATION_TYPE_Business Entity Type 1', 'ORGANIZATION_TYPE_Business Entity Typ
e 2', 'ORGANIZATION_TYPE_Business Entity Type 3', 'ORGANIZATION_TYPE_Cleaning', 'O
RGANIZATION_TYPE_Construction', 'ORGANIZATION_TYPE_Culture', 'ORGANIZATION_TYPE_El
ectricity', 'ORGANIZATION_TYPE_Emergency', 'ORGANIZATION_TYPE_Government', 'ORGANI
ZATION_TYPE_Hotel', 'ORGANIZATION_TYPE_Housing', 'ORGANIZATION_TYPE_Industry: type
1', 'ORGANIZATION_TYPE_Industry: type 10', 'ORGANIZATION_TYPE_Industry: type 11',
'ORGANIZATION_TYPE_Industry: type 12', 'ORGANIZATION_TYPE_Industry: type 13', 'ORG
ANIZATION_TYPE_Industry: type 2', 'ORGANIZATION_TYPE_Industry: type 3', 'ORGANIZAT
ION_TYPE_Industry: type 4', 'ORGANIZATION_TYPE_Industry: type 5', 'ORGANIZATION_TY
PE_Industry: type 6', 'ORGANIZATION_TYPE_Industry: type 7', 'ORGANIZATION_TYPE_Ind
ustry: type 8', 'ORGANIZATION_TYPE_Industry: type 9', 'ORGANIZATION_TYPE_Insuranc
e', 'ORGANIZATION_TYPE_Kindergarten', 'ORGANIZATION_TYPE_Legal Services', 'ORGANIZ
ATION_TYPE_Medicine', 'ORGANIZATION_TYPE_Military', 'ORGANIZATION_TYPE_Mobile', 'O
RGANIZATION_TYPE_Other', 'ORGANIZATION_TYPE_Police', 'ORGANIZATION_TYPE_Postal',
'ORGANIZATION_TYPE_Realtor', 'ORGANIZATION_TYPE_Religion', 'ORGANIZATION_TYPE_Rest
aurant', 'ORGANIZATION_TYPE_School', 'ORGANIZATION_TYPE_Security', 'ORGANIZATION_T
YPE_Security Ministries', 'ORGANIZATION_TYPE_Self-employed', 'ORGANIZATION_TYPE_Se
rvices', 'ORGANIZATION_TYPE_Telecom', 'ORGANIZATION_TYPE_Trade: type 1', 'ORGANIZA
TION_TYPE_Trade: type 2', 'ORGANIZATION_TYPE_Trade: type 3', 'ORGANIZATION_TYPE_Tr
ade: type 4', 'ORGANIZATION_TYPE_Trade: type 5', 'ORGANIZATION_TYPE_Trade: type
6', 'ORGANIZATION_TYPE_Trade: type 7', 'ORGANIZATION_TYPE_Transport: type 1', 'ORG
ANIZATION_TYPE_Transport: type 2', 'ORGANIZATION_TYPE_Transport: type 3', 'ORGANIZ
ATION_TYPE_Transport: type 4', 'ORGANIZATION_TYPE_University', 'ORGANIZATION_TYPE_
XNA', 'FONDKAPREMONT_MODE_not specified', 'FONDKAPREMONT_MODE_org spec account',
'FONDKAPREMONT_MODE_reg oper account', 'FONDKAPREMONT_MODE_reg oper spec account',
'HOUSETYPE_MODE_block of flats', 'HOUSETYPE_MODE_specific housing', 'HOUSETYPE_MOD
E_terraced house', 'WALLSMATERIAL_MODE_Block', 'WALLSMATERIAL_MODE_Mixed', 'WALLSM
ATERIAL_MODE_Monolithic', 'WALLSMATERIAL_MODE_Others', 'WALLSMATERIAL_MODE_Panel',
'WALLSMATERIAL_MODE_Stone, brick', 'WALLSMATERIAL_MODE_Wooden', 'EMERGENCYSTATE_MO
DE_No', 'EMERGENCYSTATE_MODE_Yes']
```

In [ ]:
```
# X_train_imputed = pd.DataFrame(data_prep_pipeline.fit_transform(X_train), columns
# X_valid_imputed = pd.DataFrame(data_prep_pipeline.fit_transform(X_valid), columns
# X_test_imputed = pd.DataFrame(data_prep_pipeline.fit_transform(X_test), columns=
# X_kaggle_test_imputed = pd.DataFrame(data_prep_pipeline.fit_transform(X_kaggle_te
```

Baseline Models

To get a baseline, we will use some of the features after being preprocessed through the pipeline. The baseline model is a logistic regression model

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

In [ ]:
```
try:
    del expLog
    expLog
except NameError:
    expLog = pd.DataFrame(columns=["exp_name",
                                   "Model name",
                                   "Train Acc",
                                   "Valid Acc",
                                   "Test  Acc",
                                   "Train AUC",
                                   "Valid AUC",
                                   "Test  AUC",
                                   "Train F1",
                                   "Valid F1",
                                   "Test F1",
                                   "Fit Time"
                                  ])
expLog
```

Out[ ]:

exp_name	Model name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC	Train F1	Valid F1	Test F1	Fit Time

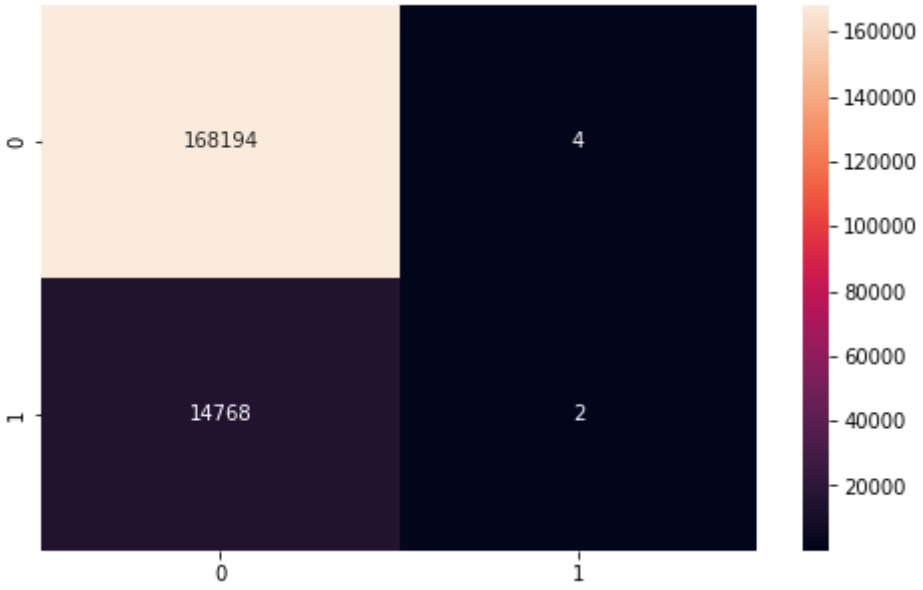
Performing a simple logistic regression run to check if the datasets are properly compatible with the models.

In [ ]:
```
np.random.seed(42)
start_time = time.time()
full_pipeline_with_predictor = Pipeline([
        ("preparation", data_prep_pipeline),
        ("pca", PCA(n_components=60)),
        ("linear", LogisticRegression(solver='saga'))
    ])
model = full_pipeline_with_predictor.fit(X_train, y_train)
fit_time = time.time() - start_time
```

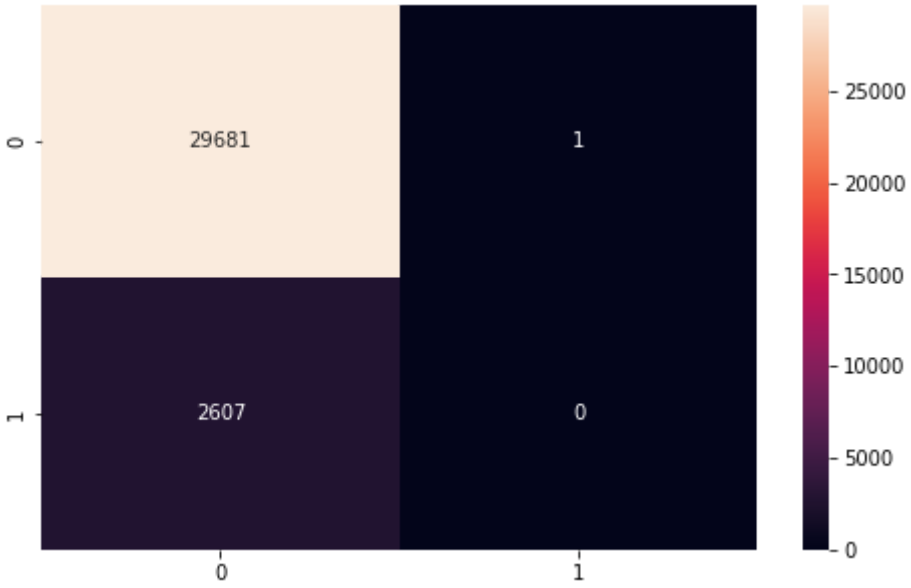
In [ ]:
```
print('Accuracy Score on Train Dataset:', np.round(accuracy_score(y_train, model.pr
print('F1 Score on Train Dataset:', np.round(f1_score(y_train, model.predict(X_tra
cf_train = confusion_matrix(y_train, model.predict(X_train))
cf_val = confusion_matrix(y_valid, model.predict(X_valid))
cf_test = confusion_matrix(y_test, model.predict(X_test))
plt.figure(figsize=(8,5))
print('Confusion Matrix for Training Set')
sns.heatmap(cf_train, annot=True, fmt='g')
plt.show()
plt.figure(figsize=(8,5))
print('Confusion Matrix for Validation Set')
sns.heatmap(cf_val, annot=True, fmt='g')
plt.show()
plt.figure(figsize=(8,5))
print('Confusion Matrix for Test Set')
sns.heatmap(cf_test, annot=True, fmt='g')
plt.show()
plt.figure(figsize=(10,8))
```

```
print('AUC-ROC for Train Set')
plot_roc_curve(model, X_train, y_train);
plt.show()
plt.figure(figsize=(10,8))
print('AUC-ROC for Valid Set')
plot_roc_curve(model, X_valid, y_valid);
plt.show()
plt.figure(figsize=(10,8))
print('AUC-ROC for Test Set')
plot_roc_curve(model, X_test, y_test);
plt.show()
```

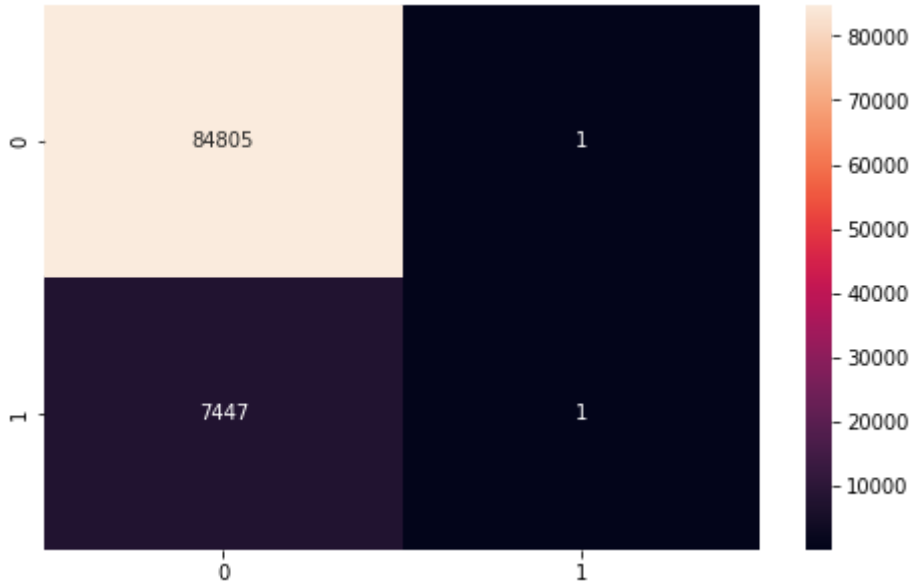
```
Accuracy Score on Train Dataset: 0.919
F1 Score on Train Dataset: 0.0
Confusion Matrix for Training Set
```



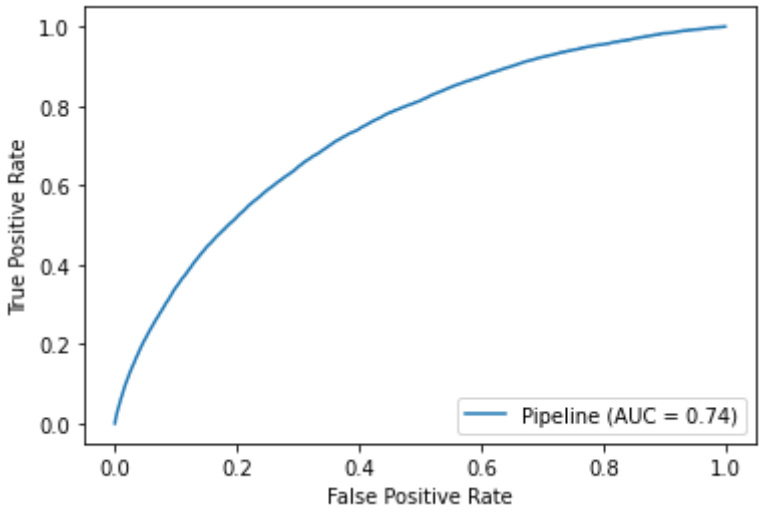

```
Confusion Matrix for Validation Set
```



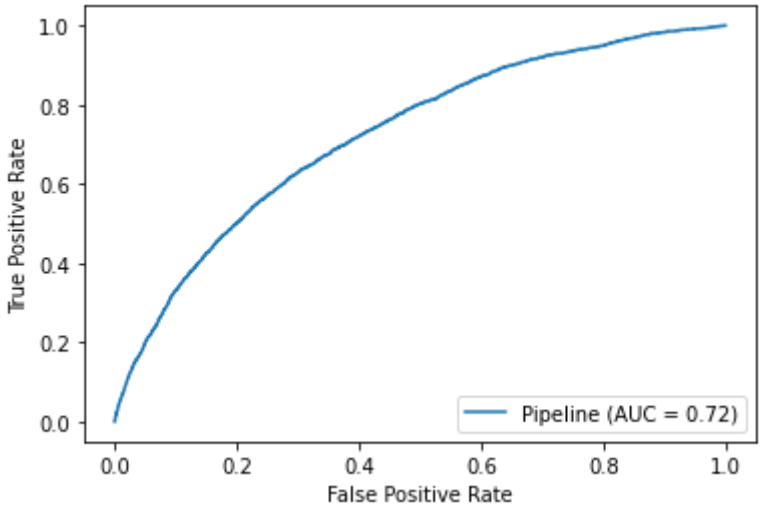

```
Confusion Matrix for Test Set
```



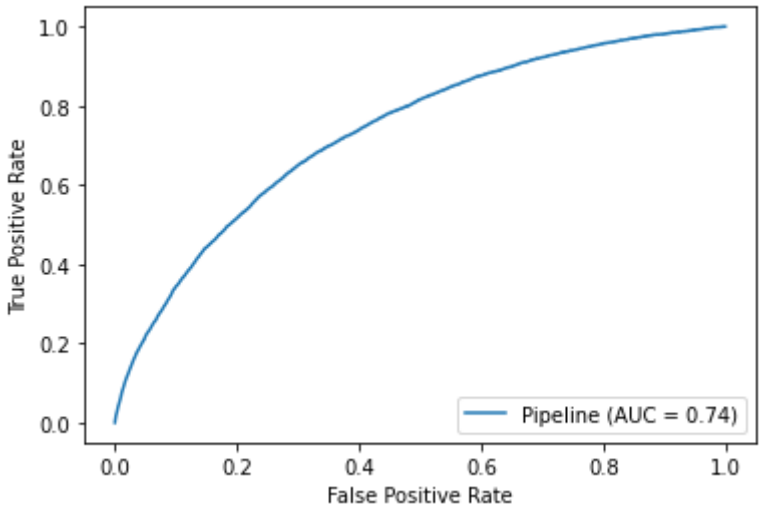

```
AUC-ROC for Train Set
<Figure size 720x576 with 0 Axes>
```




```
AUC-ROC for Valid Set
<Figure size 720x576 with 0 Axes>
```




```
AUC-ROC for Test Set
<Figure size 720x576 with 0 Axes>
```



Evaluation metrics

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

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

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

In [ ]:
```
from sklearn.metrics import roc_auc_score
print('Accuracy Score on Train Dataset:', roc_auc_score(y_train, model.predict_prol
```

```
Accuracy Score on Train Dataset: 0.7373977386631482
```

In [ ]:
```
roc_auc_score(y_test, model.predict_proba(X_test)[:, 1])
```

Out[ ]:
```
0.7372946577027398
```

THE BIG RACE (Baseline Models)

In [ ]:
```
del expLog
try:
    expLog
except NameError:
    expLog = pd.DataFrame(columns=["exp_name",
                                   "Model name",
                                   "Train Acc",
                                   "Valid Acc",
                                   "Test  Acc",
                                   "Train AUC",
                                   "Valid AUC",
                                   "Test  AUC",
                                   "Train F1",
                                   "Valid F1",
                                   "Test F1",
```

```
                                    "Fit Time (seconds)"
                                   ])
expLog
```

Out[ ]:

exp_name	Model name	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC	Train F1	Valid F1	Test F1	Fit Time (seconds)

Using Non-Ensemble Models

In [ ]:

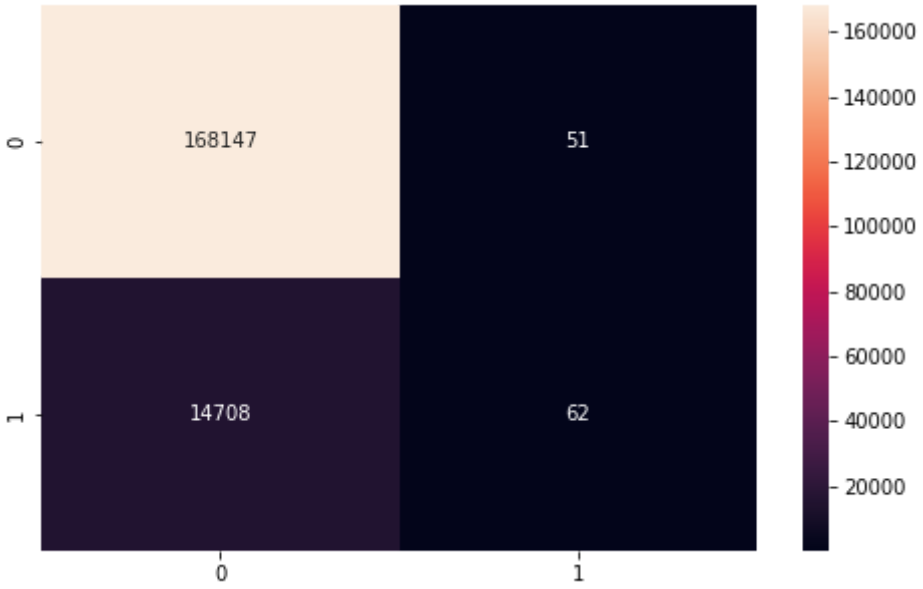
```
clfs = [LogisticRegression(solver='saga'),
        DecisionTreeClassifier(),
        GaussianNB()]

for clf in clfs:
    start_time = time.time()
    full_pipeline_with_predictor = Pipeline([
        ("preparation", data_prep_pipeline),
        ("model", clf)
    ])
    model_name = "Baseline {}".format(type(full_pipeline_with_predictor['model']).
    model = full_pipeline_with_predictor.fit(X_train, y_train)
    fit_time = time.time() - start_time
    print('Fit Time for {} is: {} seconds'.format(model_name, fit_time))
    exp_name = f"Baseline_{len(selected_features)}_features"
    print(model_name)
    expLog.loc[len(expLog)] = [f"{exp_name}"] + [model_name] + list(np.round(
                  [accuracy_score(y_train, model.predict(X_train)),
                   accuracy_score(y_valid, model.predict(X_valid)),
                   accuracy_score(y_test, model.predict(X_test)),
                   roc_auc_score(y_train, model.predict_proba(X_train)[:, 1]),
                   roc_auc_score(y_valid, model.predict_proba(X_valid)[:, 1]),
                   roc_auc_score(y_test, model.predict_proba(X_test)[:, 1]),
                   f1_score(y_train, model.predict(X_train)),
                   f1_score(y_valid, model.predict(X_valid)),
                   f1_score(y_test, model.predict(X_test)),
                   fit_time], 4))
    cf_train = confusion_matrix(y_train, model.predict(X_train))
    cf_val = confusion_matrix(y_valid, model.predict(X_valid))
    cf_test = confusion_matrix(y_test, model.predict(X_test))
    plt.figure(figsize=(8,5))
    print('Confusion Matrix for Training Set')
    sns.heatmap(cf_train, annot=True, fmt='g')
    plt.show()
    plt.figure(figsize=(8,5))
    print('Confusion Matrix for Validation Set')
    sns.heatmap(cf_val, annot=True, fmt='g')
    plt.show()
    plt.figure(figsize=(8,5))
    print('Confusion Matrix for Test Set')
    sns.heatmap(cf_test, annot=True, fmt='g')
    plt.show()
    plt.figure(figsize=(10,8))
    print('AUC-ROC for Train Set')
    plot_roc_curve(model, X_train, y_train);
    plt.show()
    plt.figure(figsize=(10,8))
    print('AUC-ROC for Valid Set')
    plot_roc_curve(model, X_valid, y_valid);
    plt.show()
    plt.figure(figsize=(10,8))
    print('AUC-ROC for Test Set')
    plot_roc_curve(model, X_test, y_test);
```

```
    plt.show()

expLog
```

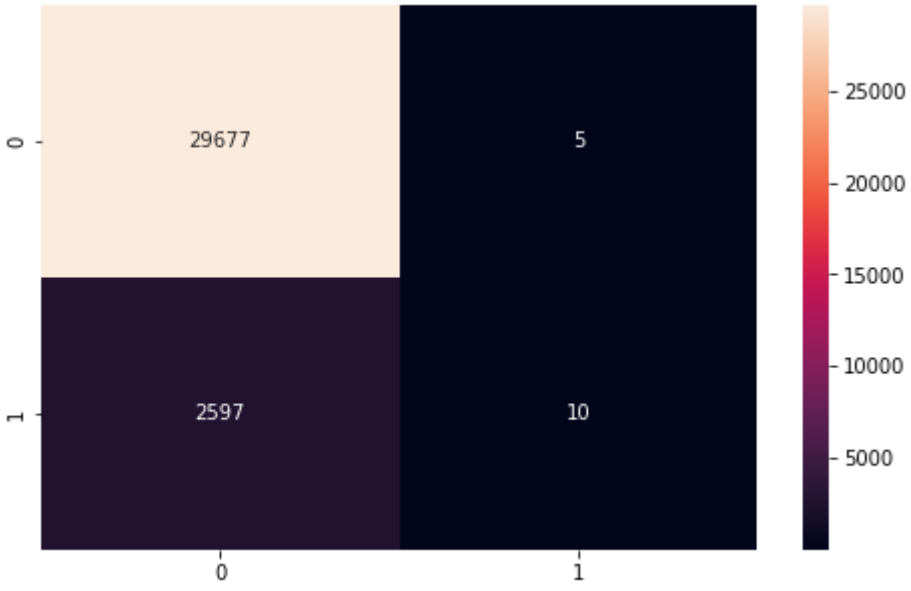
```
Fit Time for Baseline LogisticRegression is: 4.688223361968994 seconds
```

```
Baseline LogisticRegression
```

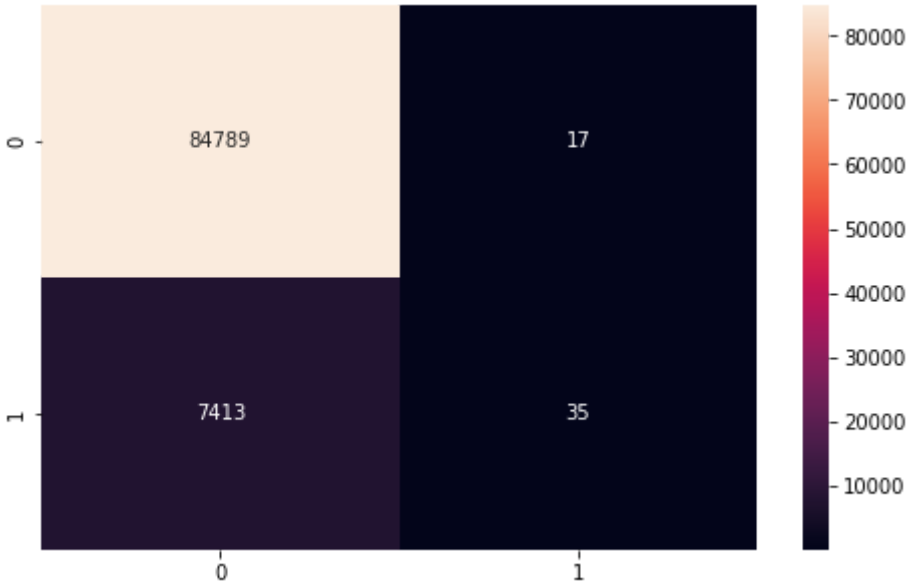
```
Confusion Matrix for Training Set
```



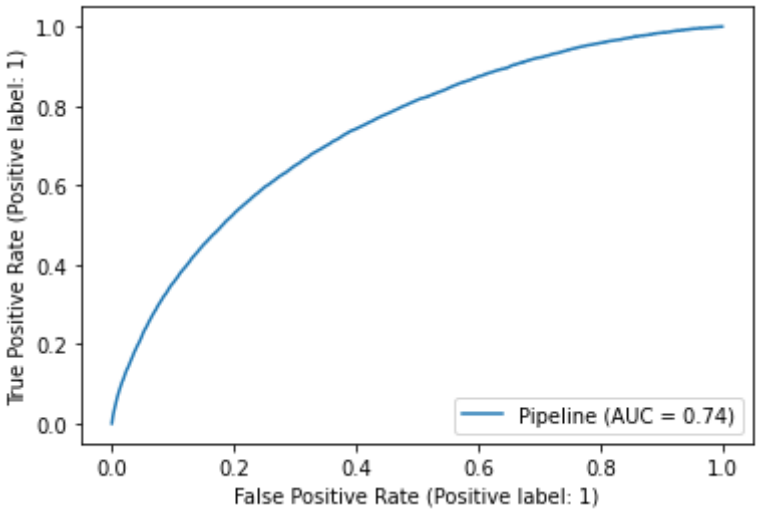

```
Confusion Matrix for Validation Set
```



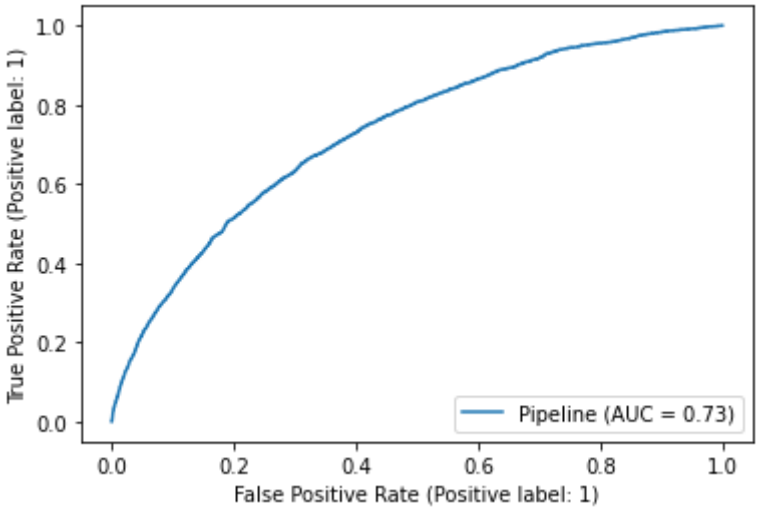

```
Confusion Matrix for Test Set
```



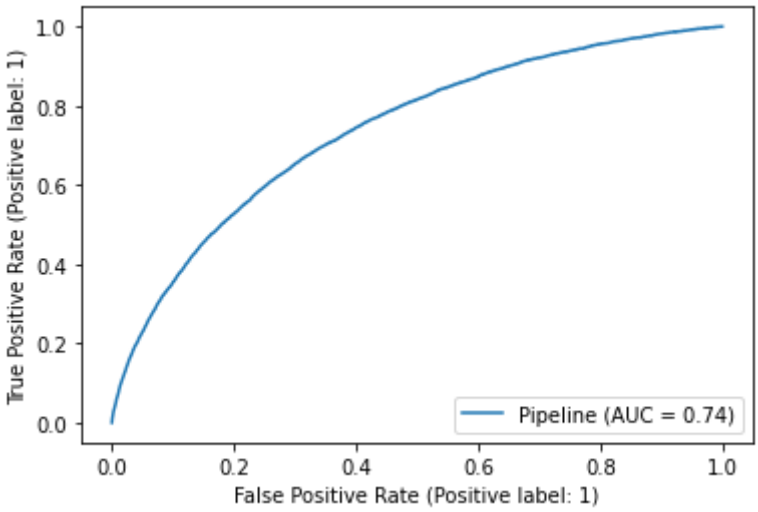
AUC-ROC for Train Set
<Figure size 720x576 with 0 Axes>



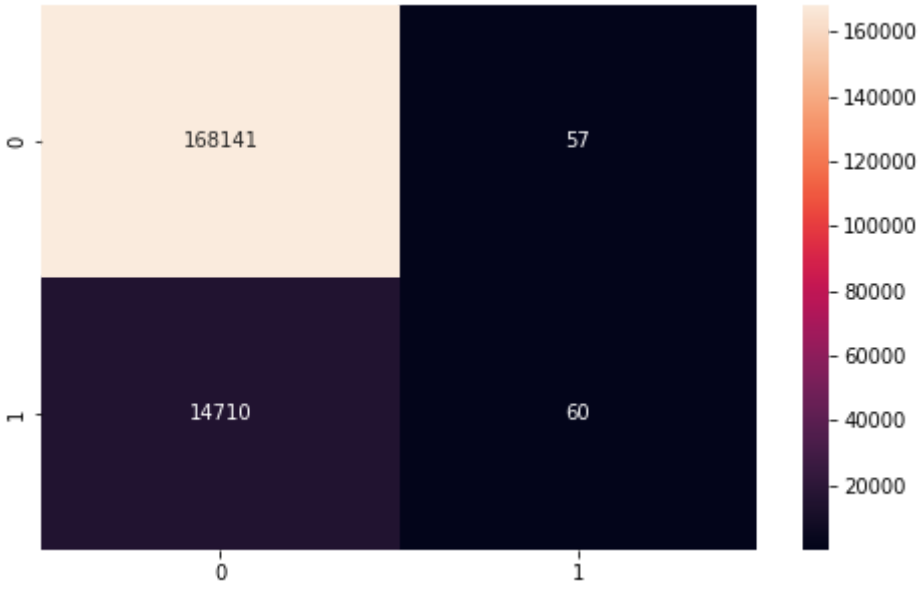
AUC-ROC for Valid Set
<Figure size 720x576 with 0 Axes>



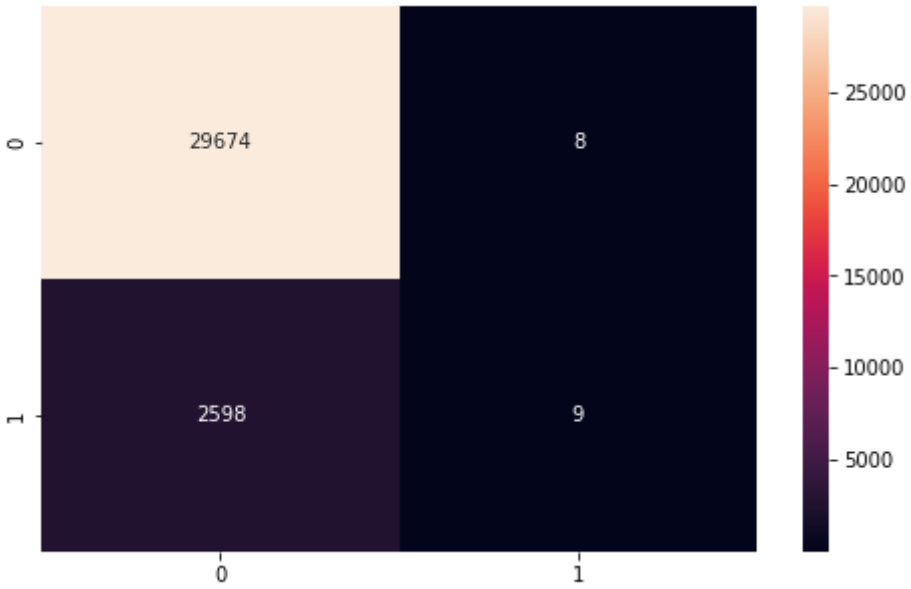
AUC-ROC for Test Set
<Figure size 720x576 with 0 Axes>



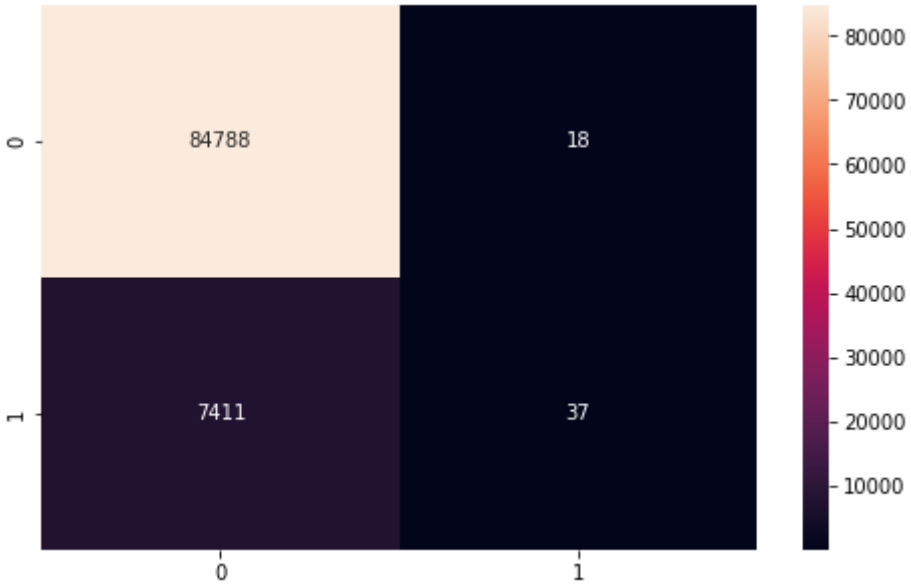

```
Fit Time for Baseline LogisticRegression is: 4.942737102508545 seconds


Baseline LogisticRegression


Confusion Matrix for Training Set
```



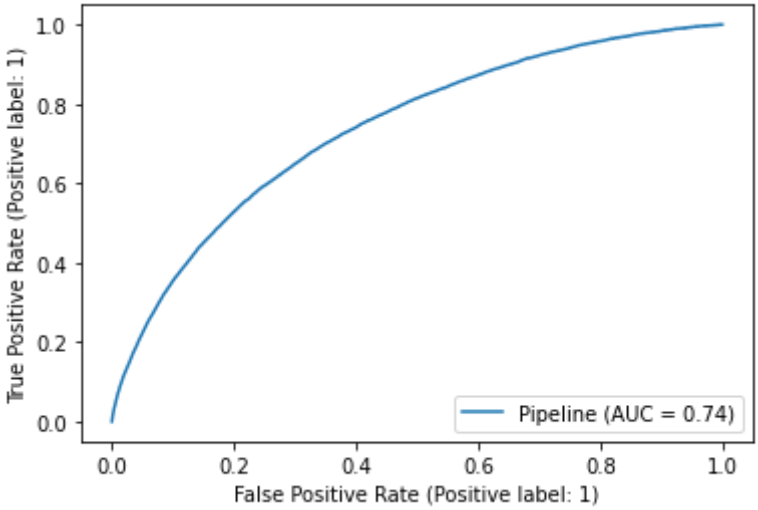

```
Confusion Matrix for Validation Set
```



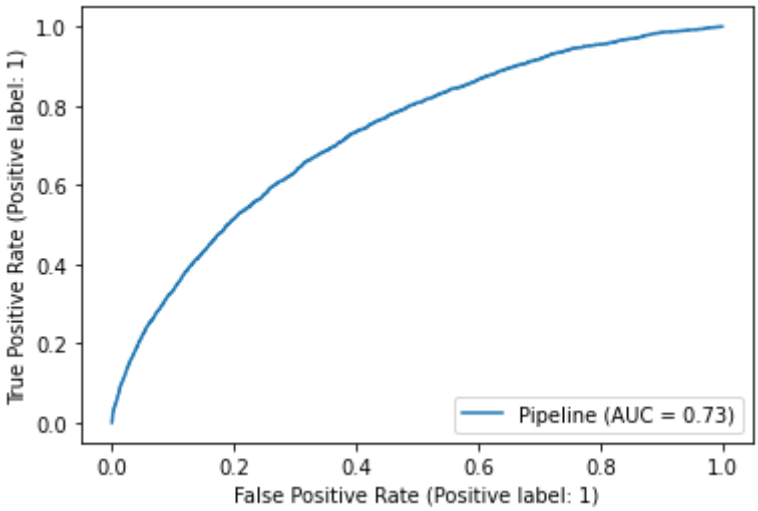
```
Confusion Matrix for Test Set
```



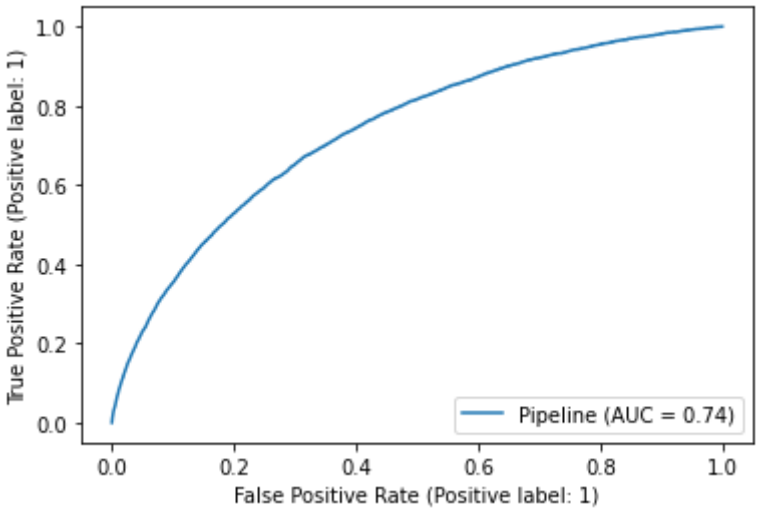

```
AUC-ROC for Train Set
<Figure size 720x576 with 0 Axes>
```



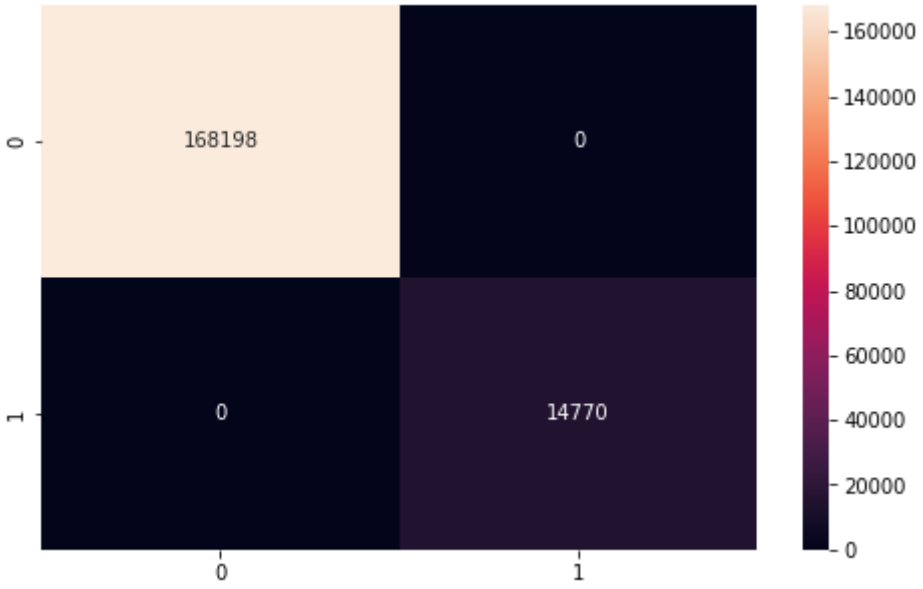

```
AUC-ROC for Valid Set
<Figure size 720x576 with 0 Axes>
```



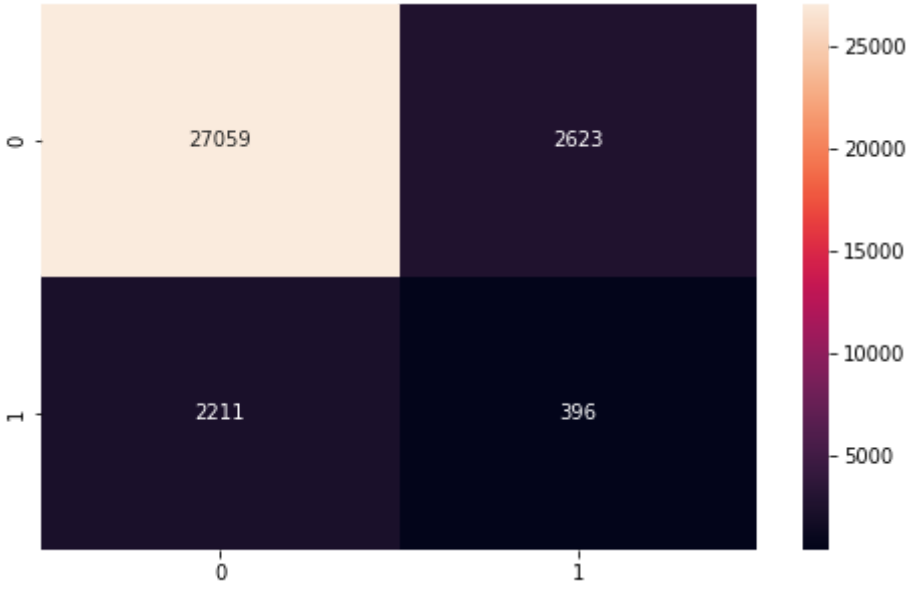

```
AUC-ROC for Test Set
<Figure size 720x576 with 0 Axes>
```



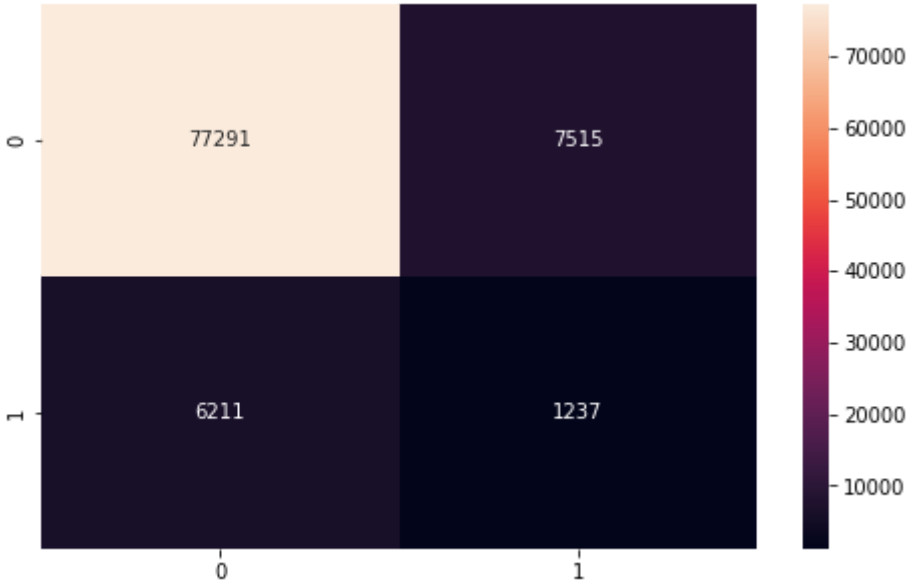

```
Fit Time for Baseline DecisionTreeClassifier is: 21.227879524230957 seconds


Baseline DecisionTreeClassifier


Confusion Matrix for Training Set
```



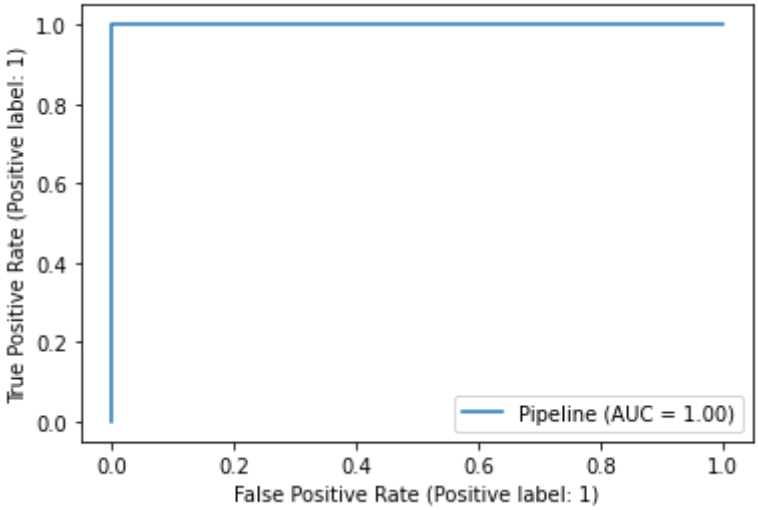

```
Confusion Matrix for Validation Set
```



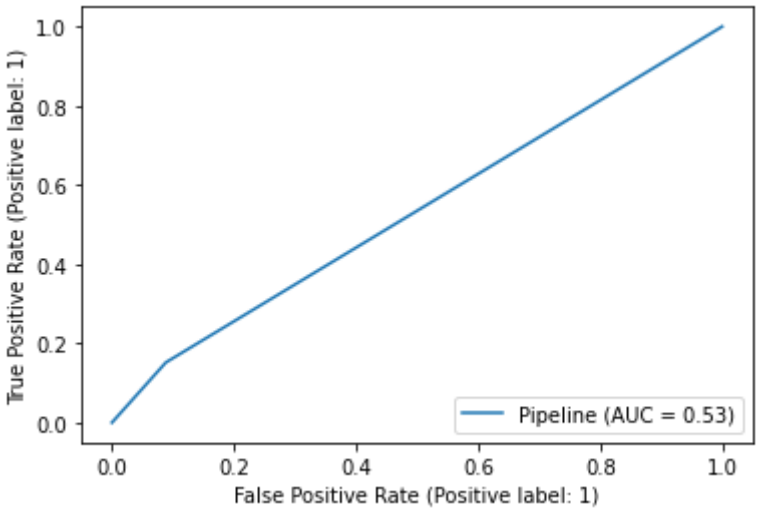
```
Confusion Matrix for Test Set
```



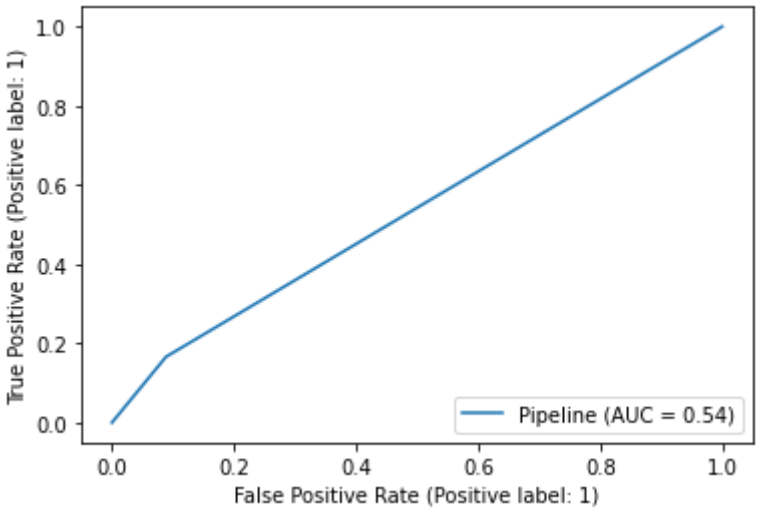

```
AUC-ROC for Train Set
<Figure size 720x576 with 0 Axes>
```



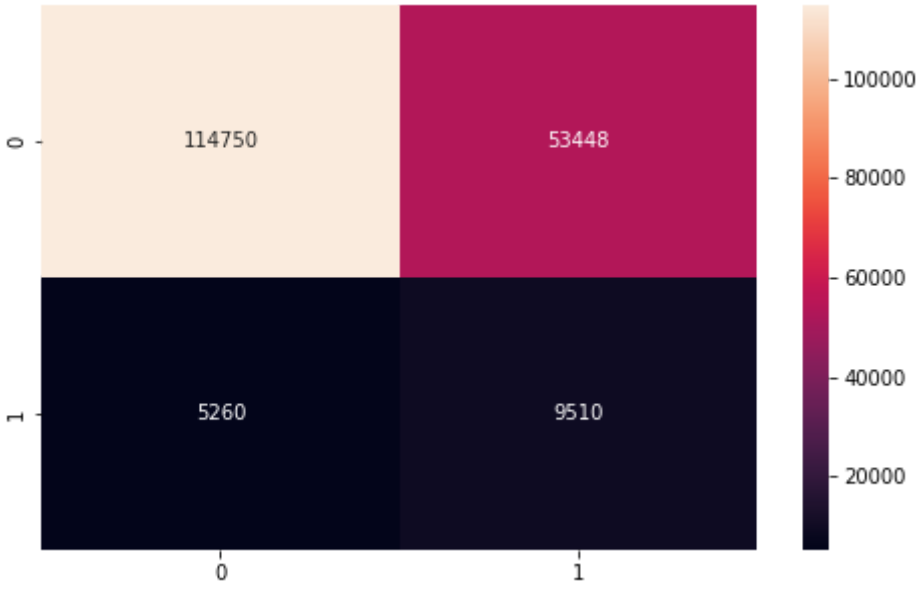

```
AUC-ROC for Valid Set
<Figure size 720x576 with 0 Axes>
```



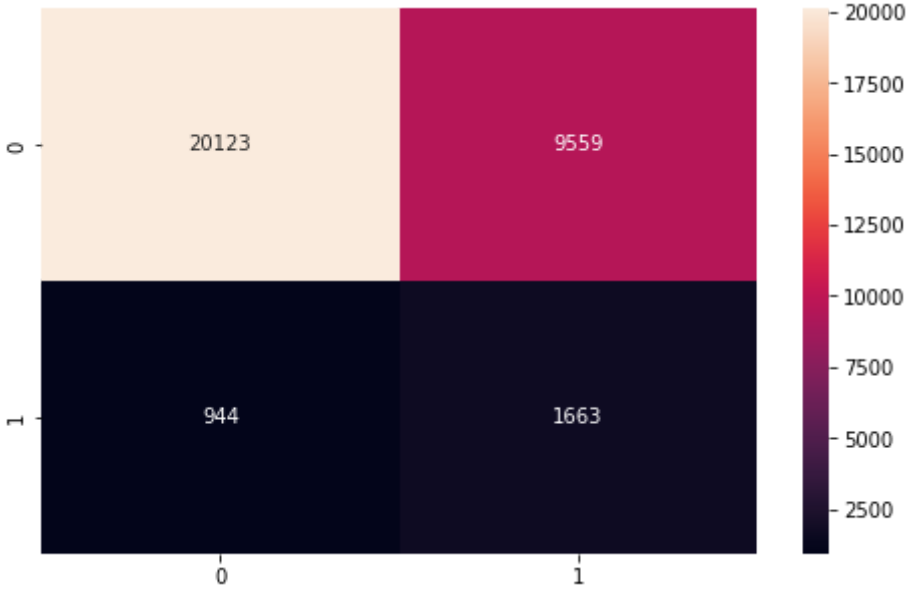

```
AUC-ROC for Test Set
<Figure size 720x576 with 0 Axes>
```



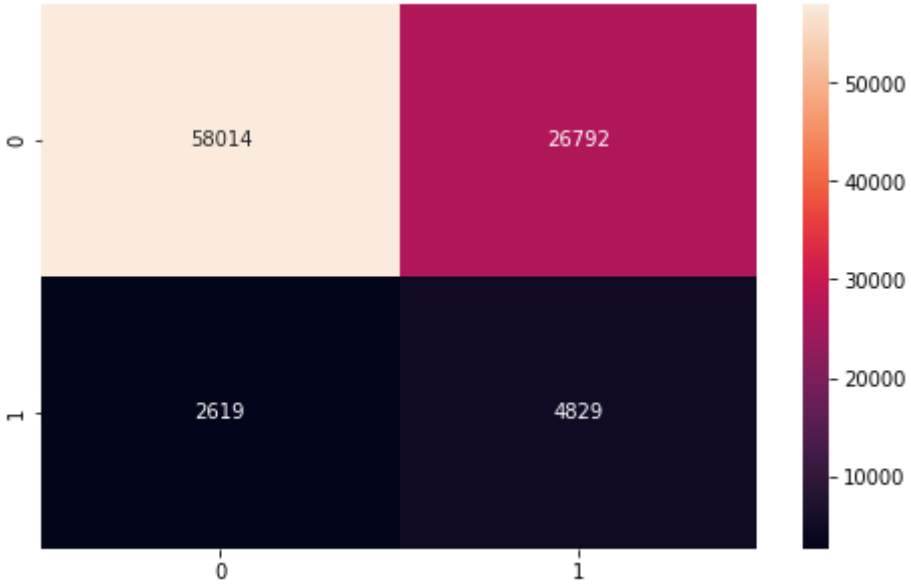

```
Fit Time for Baseline GaussianNB is: 1.784973382949829 seconds


Baseline GaussianNB


Confusion Matrix for Training Set
```



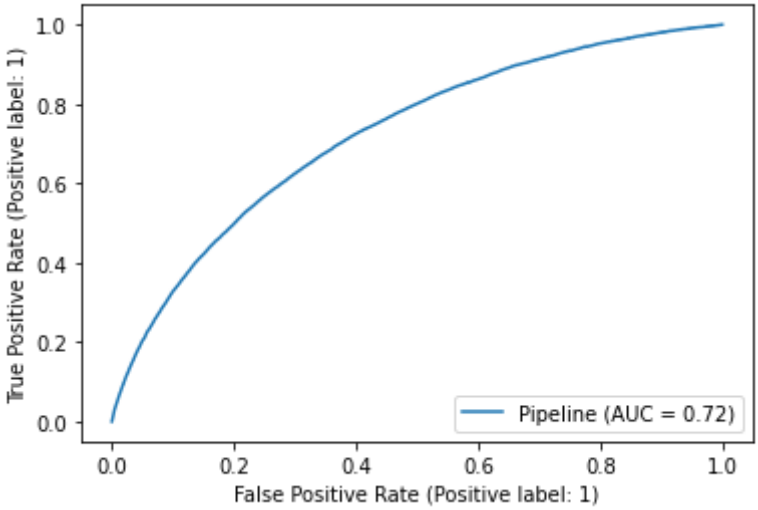

```
Confusion Matrix for Validation Set
```



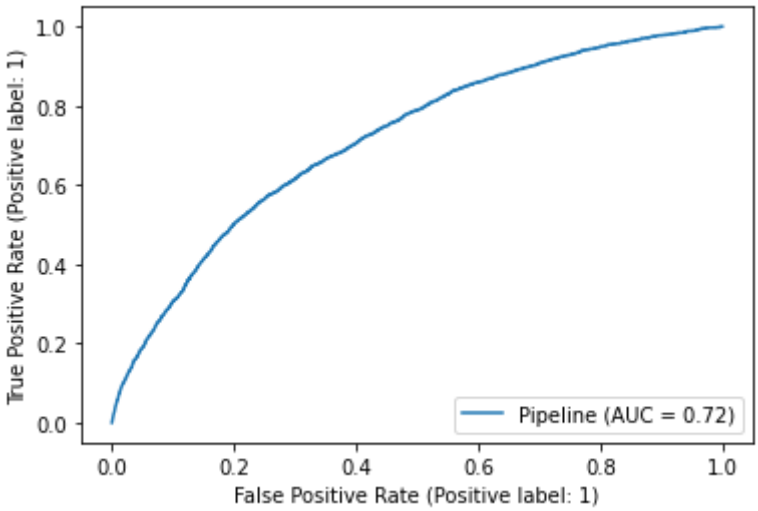
```
Confusion Matrix for Test Set
```



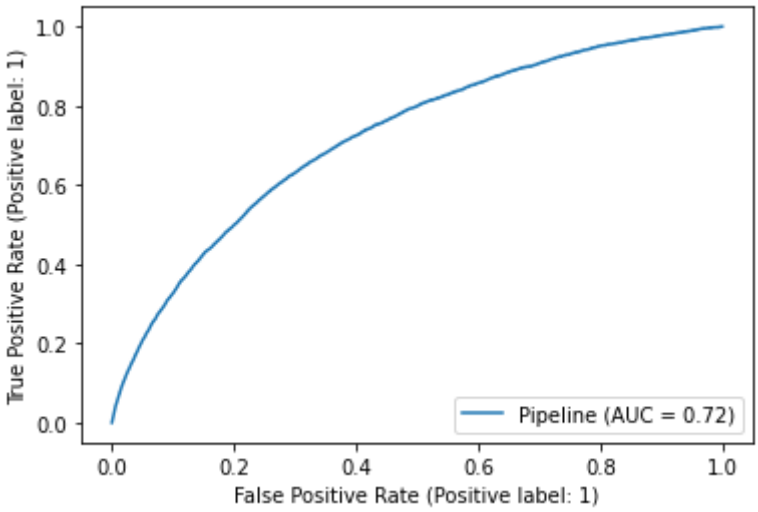

```
AUC-ROC for Train Set
<Figure size 720x576 with 0 Axes>
```




```
AUC-ROC for Valid Set
<Figure size 720x576 with 0 Axes>
```




```
AUC-ROC for Test Set
<Figure size 720x576 with 0 Axes>
```

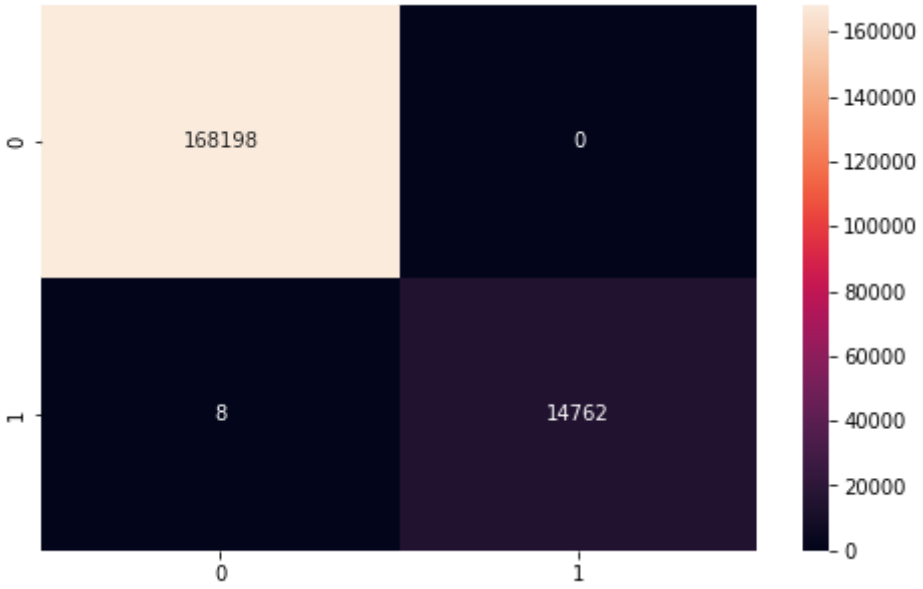


Using Ensemble Models

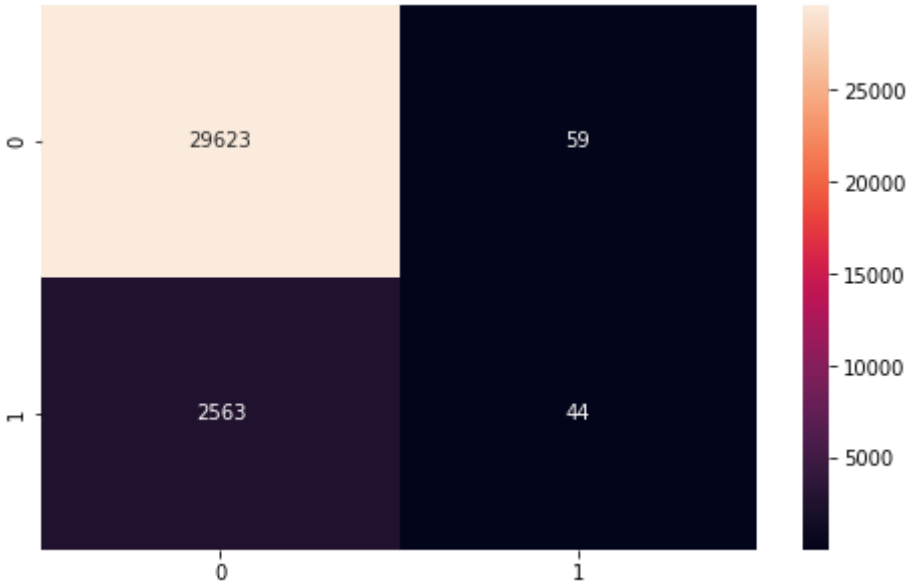
In [ ]:
```
clfs = [RandomForestClassifier(), XGBClassifier()]

for clf in clfs:
    start_time = time.time()
    full_pipeline_with_predictor = Pipeline([
        ("preparation", data_prep_pipeline),
        ("model", clf)
    ])
    model_name = "Baseline {}".format(type(full_pipeline_with_predictor['model']).
    model = full_pipeline_with_predictor.fit(X_train, y_train)
    fit_time = time.time() - start_time
    print('Fit Time for {} is: {} seconds'.format(model_name, fit_time))
    exp_name = f"Baseline_{len(selected_features)}_features"
    print(model_name)
    expLog.loc[len(expLog)] = [f"{exp_name}"] + [model_name] + list(np.round(
                  [accuracy_score(y_train, model.predict(X_train)),
                   accuracy_score(y_valid, model.predict(X_valid)),
                   accuracy_score(y_test, model.predict(X_test)),
                   roc_auc_score(y_train, model.predict_proba(X_train)[:, 1]),
                   roc_auc_score(y_valid, model.predict_proba(X_valid)[:, 1]),
                   roc_auc_score(y_test, model.predict_proba(X_test)[:, 1]),
                   f1_score(y_train, model.predict(X_train)),
                   f1_score(y_valid, model.predict(X_valid)),
                   f1_score(y_test, model.predict(X_test)),
                   fit_time], 4))
    cf_train = confusion_matrix(y_train, model.predict(X_train))
    cf_val = confusion_matrix(y_valid, model.predict(X_valid))
    cf_test = confusion_matrix(y_test, model.predict(X_test))
    plt.figure(figsize=(8,5))
    print('Confusion Matrix for Training Set')
    sns.heatmap(cf_train, annot=True, fmt='g')
    plt.show()
    plt.figure(figsize=(8,5))
    print('Confusion Matrix for Validation Set')
    sns.heatmap(cf_val, annot=True, fmt='g')
    plt.show()
    plt.figure(figsize=(8,5))
    print('Confusion Matrix for Test Set')
    sns.heatmap(cf_test, annot=True, fmt='g')
    plt.show()
    plt.figure(figsize=(10,8))
    print('AUC-ROC for Train Set')
    plot_roc_curve(model, X_train, y_train);
    plt.show()
```

```
    plt.figure(figsize=(10,8))
    print('AUC-ROC for Valid Set')
    plot_roc_curve(model, X_valid, y_valid);
    plt.show()
    plt.figure(figsize=(10,8))
    print('AUC-ROC for Test Set')
    plot_roc_curve(model, X_test, y_test);
    plt.show()
```

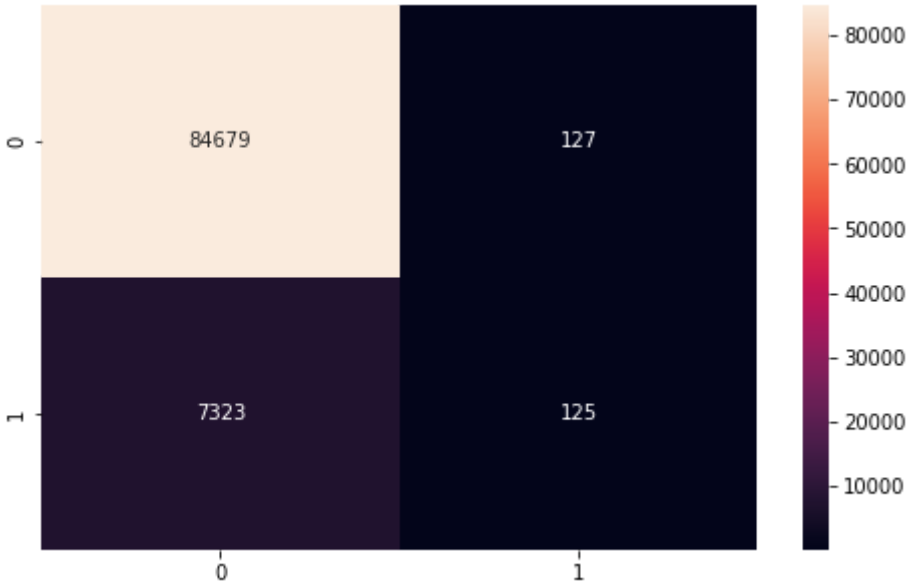
```
Fit Time for Baseline RandomForestClassifier is: 133.82825803756714 seconds


Baseline RandomForestClassifier


Confusion Matrix for Training Set
```



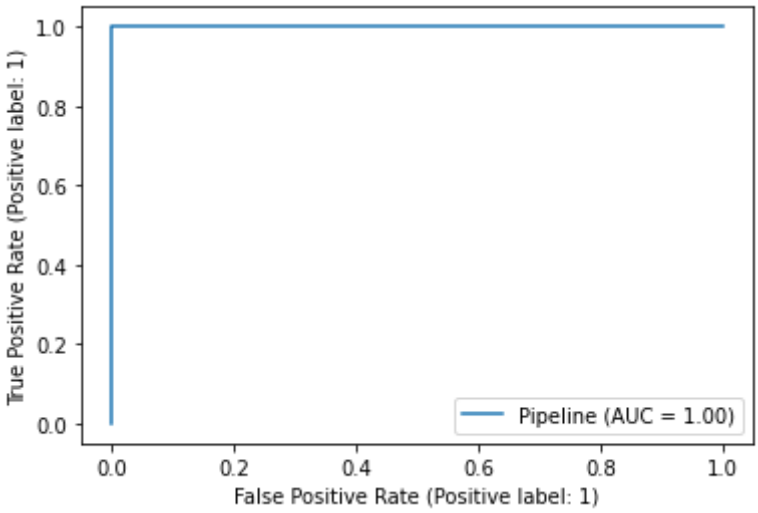

```
Confusion Matrix for Validation Set
```



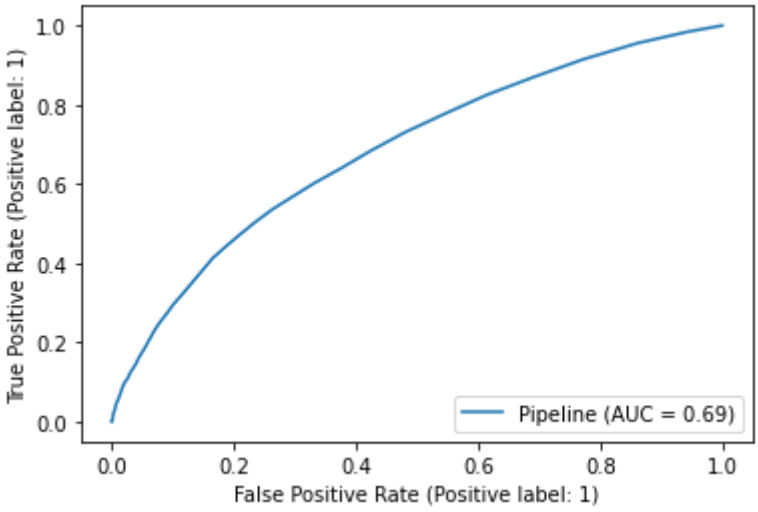

```
Confusion Matrix for Test Set
```



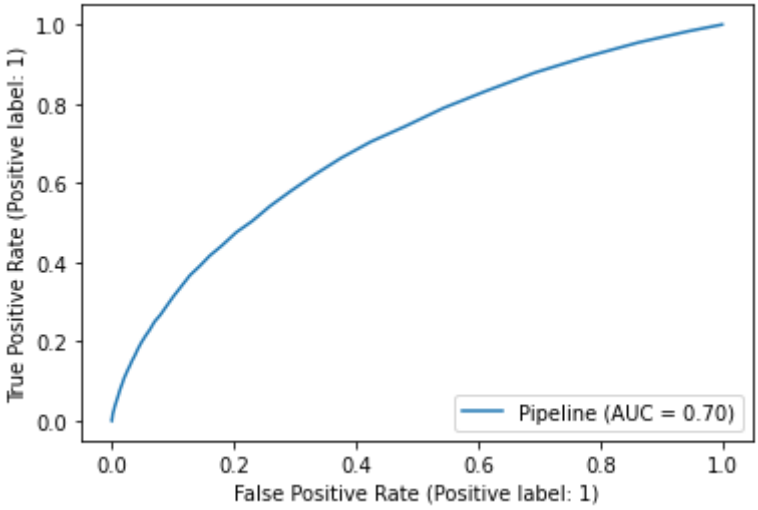

```
AUC-ROC for Train Set
<Figure size 720x576 with 0 Axes>
```



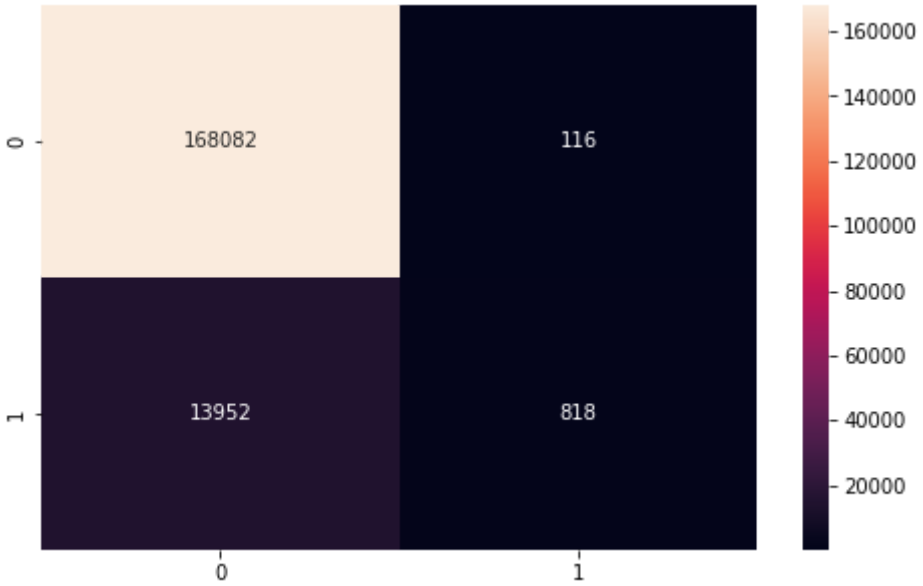

```
AUC-ROC for Valid Set
<Figure size 720x576 with 0 Axes>
```



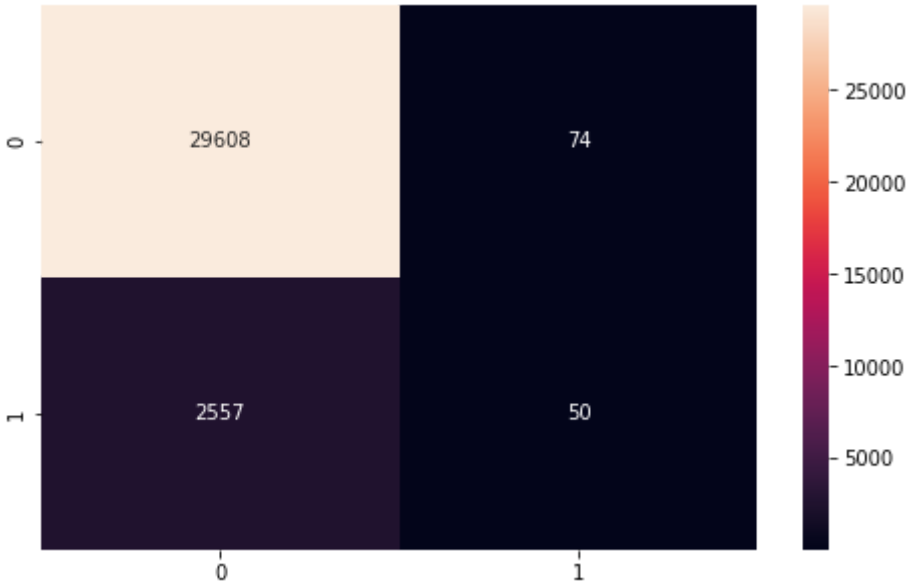

```
AUC-ROC for Test Set
<Figure size 720x576 with 0 Axes>
```



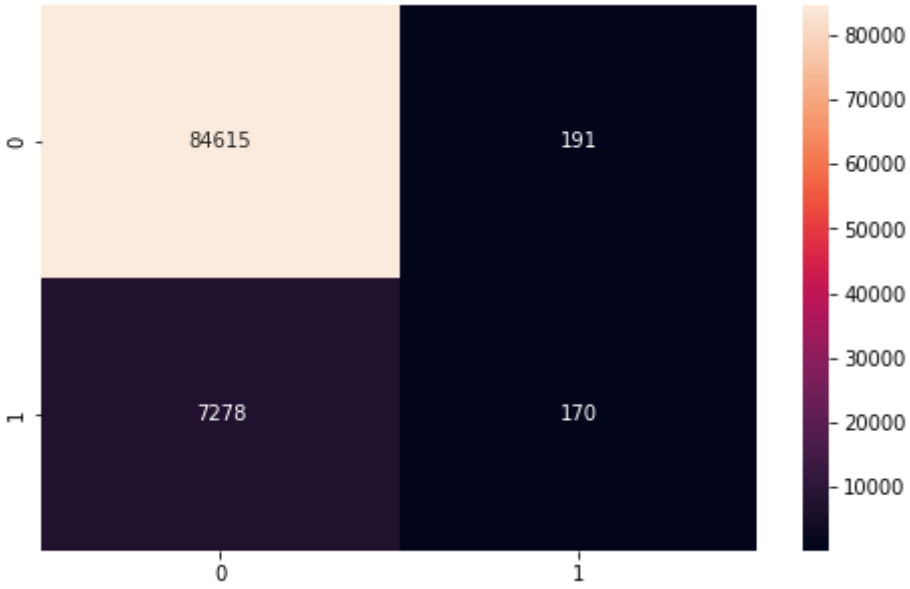

```
[01:07:22] WARNING: ../src/learner.cc:1115: Starting in XGBoost 1.3.0, the default
evaluation metric used with the objective 'binary:logistic' was changed from 'erro
r' to 'logloss'. Explicitly set eval_metric if you'd like to restore the old behav
ior.


Fit Time for Baseline XGBClassifier is: 15.154890298843384 seconds


Baseline XGBClassifier


Confusion Matrix for Training Set
```




```
Confusion Matrix for Validation Set
```

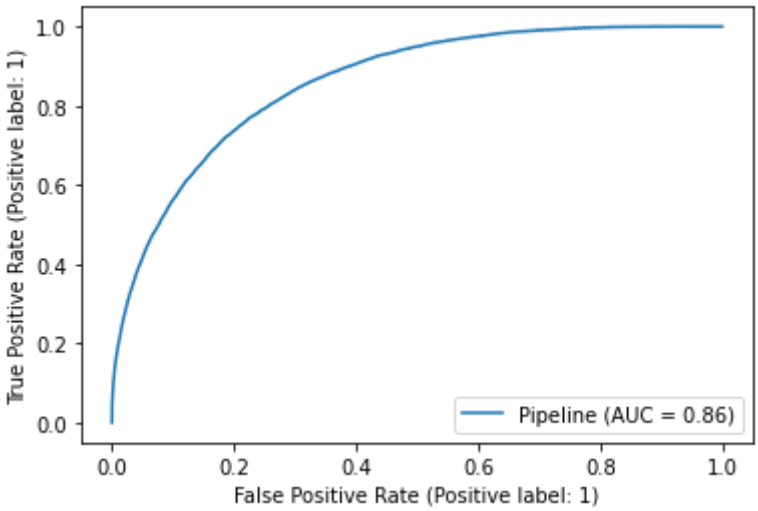


Confusion Matrix for Test Set



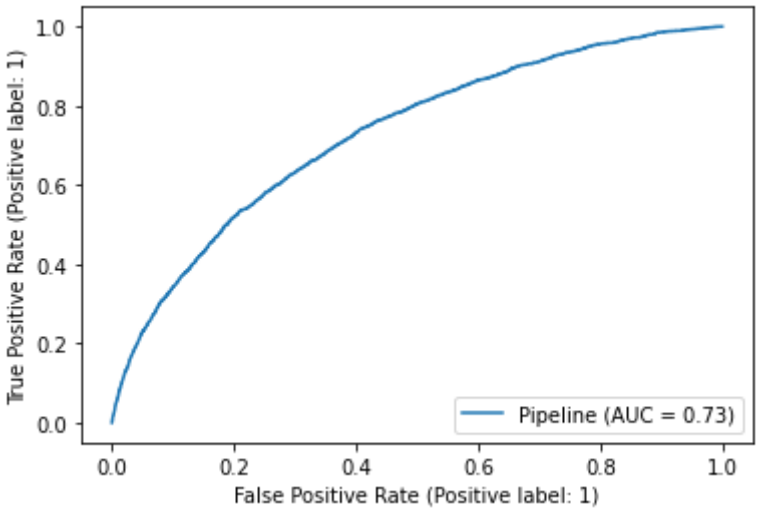
AUC-ROC for Train Set

<Figure size 720x576 with 0 Axes>

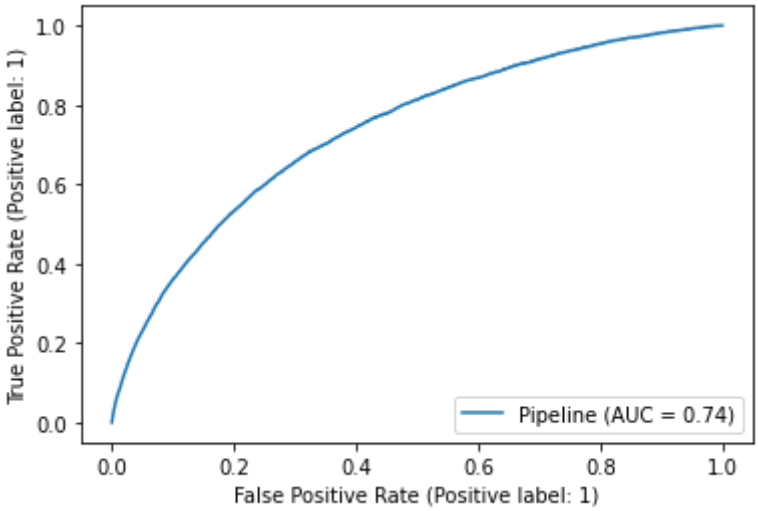


AUC-ROC for Valid Set

<Figure size 720x576 with 0 Axes>




```
AUC-ROC for Test Set
<Figure size 720x576 with 0 Axes>
```



KNeighborsClassifier, SVC, and Logistic Regression with L1 regularization with solver='liblinear' take a lot of time to run and crash the kernel. So we were unable to train the dataset on these models.

In [ ]:
```
# from sklearn.svm import SVC
# from sklearn.neighbors import KNeighborsClassifier
# clfs = [SVC(), KNeighborsClassifier()]

# for clf in clfs:
#     start_time = time.time()
#     full_pipeline_with_predictor = Pipeline([
#         ("preparation", data_prep_pipeline),
#         ("model", clf)
#     ])
#     model_name = "Baseline {}".format(type(full_pipeline_with_predictor['model']),
#     model = full_pipeline_with_predictor.fit(X_train, y_train)
#     fit_time = time.time() - start_time
#     print('Fit Time for {} is: {} seconds'.format(model_name, fit_time))
#     exp_name = f"Baseline_{len(selected_features)}_features"

#     expLog.loc[len(expLog)] = [f"{exp_name}"] + [model_name] + list(np.round(
#                     [accuracy_score(y_train, model.predict(X_train)),
#                      accuracy_score(y_valid, model.predict(X_valid)),
#                      accuracy_score(y_test, model.predict(X_test)),
#                      roc_auc_score(y_train, model.predict_proba(X_train)[:, 1]),
#                      roc_auc_score(y_valid, model.predict_proba(X_valid)[:, 1]),
#                      roc_auc_score(y_test, model.predict_proba(X_test)[:, 1]),
```

```
#                     f1_score(y_train, model.predict(X_train)),
#                     f1_score(y_valid, model.predict(X_valid)),
#                     f1_score(y_test, model.predict(X_test)),
#                     fit_time], 4))
# explog
```

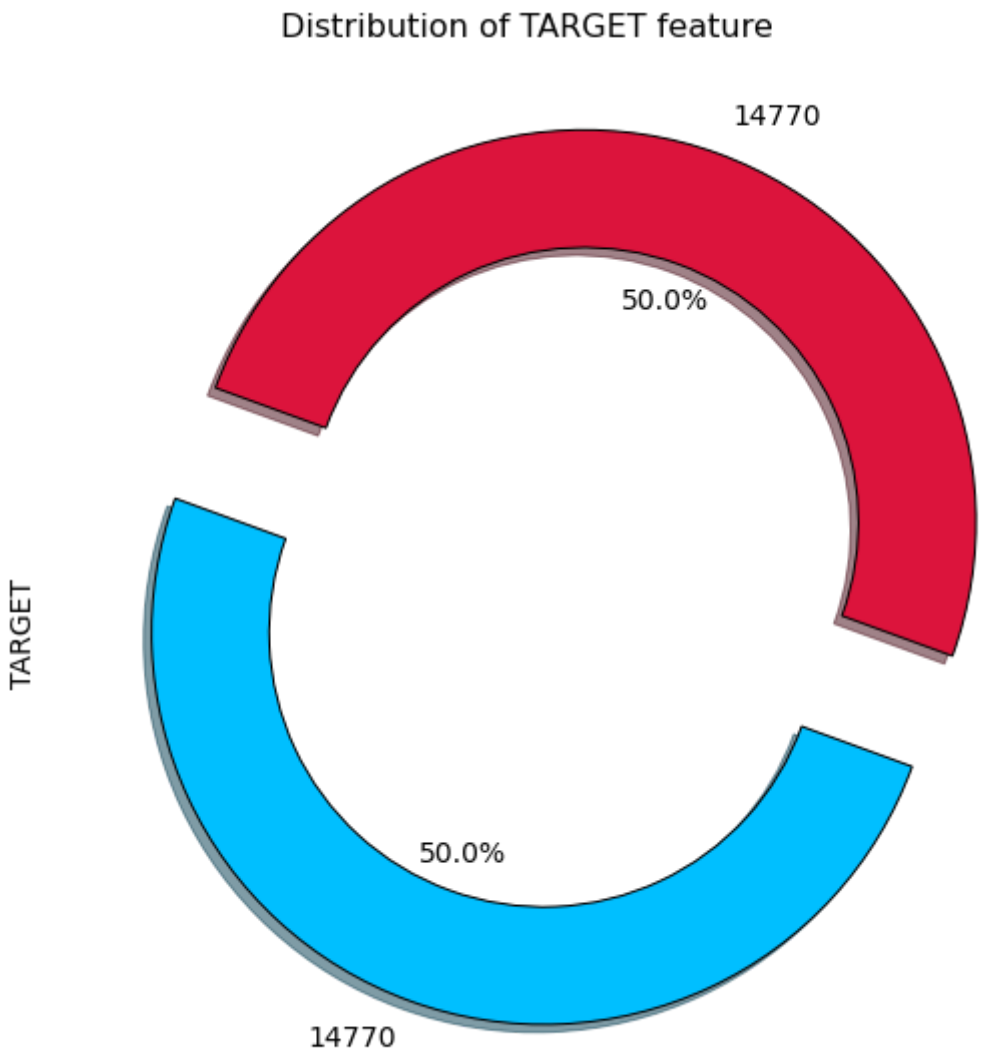
Undersampling using RandomUnderSampler

In [ ]:
```
from imblearn.under_sampling import RandomUnderSampler
# define undersample strategy
random_undersampler = RandomUnderSampler(sampling_strategy='majority')
# fit and apply the transform
X_under, y_under = random_undersampler.fit_resample(X_train, y_train)
```

In [ ]:
```
# summarize class distribution
y_under.value_counts()
```

Out[ ]:
```
TARGET
0         14770
1         14770
dtype: int64
```

In [ ]:
```
plt.figure(figsize=(9, 9))
plt.pie(x=y_under.value_counts(),
        radius=1.3-0.3,
        labels=y_under.value_counts(),
        autopct='%1.1f%%',
        colors=['deepskyblue', 'crimson'],
        explode=[0,0.3],
        wedgeprops={"edgecolor":"0", "width":0.3},
        startangle=160,
        shadow=True,
        textprops={'fontsize': 14})
plt.ylabel('TARGET', fontsize=14)
plt.suptitle('Distribution of TARGET feature', fontsize=16)
plt.show()
```



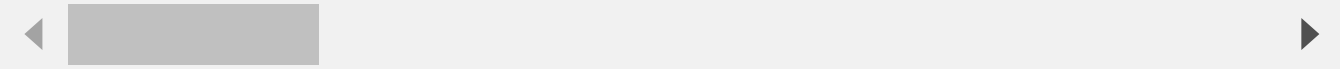
Using RandomForest Feature Importance on the Undersampled data

In [ ]:
```
X_under[:10]
```

Out[ ]:

	SK_ID_CURR	CNT_CHILDREN	DAYS_ID_PUBLISH	AMT_INCOME_TOTAL	REGION_POPULATION_RE
0	393418	1	-4269	81000.0	0
1	308883	0	-4141	225000.0	0
2	133379	1	-4087	225000.0	0
3	212640	0	-4738	292500.0	0
4	275183	1	-4268	112500.0	0
5	178287	1	-3993	312750.0	0
6	376198	1	-2262	225000.0	0
7	228325	0	-2324	112500.0	0
8	215709	1	-1616	121500.0	0
9	290335	0	-1889	180000.0	0

10 rows × 330 columns



In [ ]:

```
model = RandomForestClassifier()
clf_pipe = Pipeline([("preparation", data_prep_pipeline),
                     ("model", model)])
# fit the model
clf_pipe.fit(X_under, y_under)
```

Out[ ]:
```
Pipeline(steps=[('preparation',
                 FeatureUnion(transformer_list=[('num_pipeline',
                                                 Pipeline(steps=[('selector',
                                                                  DataFrameSelecto
r(attribute_names=['SK_ID_CURR',

'CNT_CHILDREN',

'DAYS_ID_PUBLISH',

'AMT_INCOME_TOTAL',

'REGION_POPULATION_RELATIVE',

'DAYS_BIRTH_x',

'DAYS_EMPLOYED',

'DAYS_REGISTRATION',

'OWN_CAR_AGE',

'CNT_FAM_MEMBERS',

'EXT_SOURCE_1_x',

'EXT_SOURCE_2_x...

'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', ...])),
                                                                 ('imputer',
                                                                  SimpleImputer(st
rategy='most_frequent'))]))])),
                ('model', RandomForestClassifier())])
```

In [ ]:
```
# get importance
importance = clf_pipe['model'].feature_importances_
# summarize feature importance
important_features = pd.DataFrame(zip(X_under.columns, importance), columns=['Featu
```

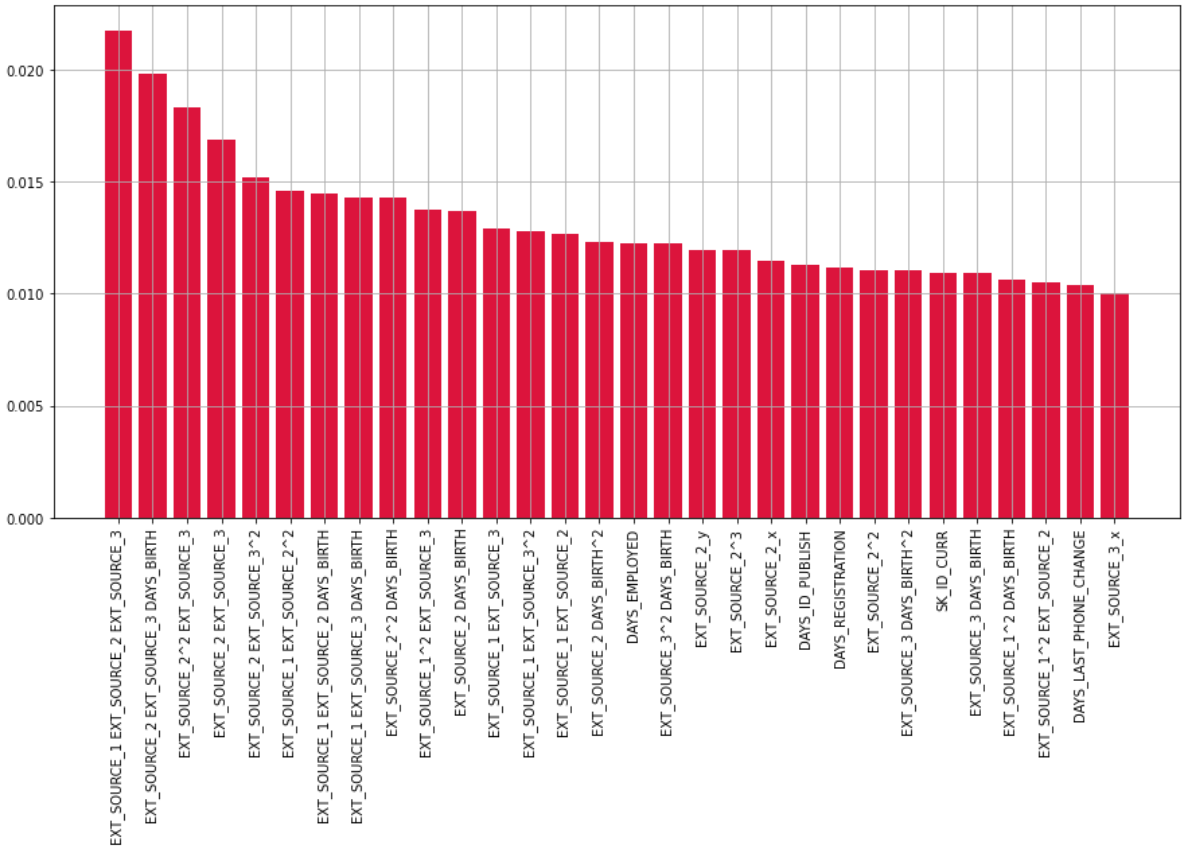
```
important_features.sort_values(by='Feature Score', ascending=False, inplace=True)
important_features.head(20)
```

Out[ ]:

	Feature Name	Feature Score
87	EXT_SOURCE_1 EXT_SOURCE_2 EXT_SOURCE_3	0.021752
96	EXT_SOURCE_2 EXT_SOURCE_3 DAYS_BIRTH	0.019794
93	EXT_SOURCE_2^2 EXT_SOURCE_3	0.018275
77	EXT_SOURCE_2 EXT_SOURCE_3	0.016836
95	EXT_SOURCE_2 EXT_SOURCE_3^2	0.015181
86	EXT_SOURCE_1 EXT_SOURCE_2^2	0.014589
88	EXT_SOURCE_1 EXT_SOURCE_2 DAYS_BIRTH	0.014479
90	EXT_SOURCE_1 EXT_SOURCE_3 DAYS_BIRTH	0.014307
94	EXT_SOURCE_2^2 DAYS_BIRTH	0.014294
84	EXT_SOURCE_1^2 EXT_SOURCE_3	0.013764
78	EXT_SOURCE_2 DAYS_BIRTH	0.013660
74	EXT_SOURCE_1 EXT_SOURCE_3	0.012923
89	EXT_SOURCE_1 EXT_SOURCE_3^2	0.012795
73	EXT_SOURCE_1 EXT_SOURCE_2	0.012657
97	EXT_SOURCE_2 DAYS_BIRTH^2	0.012278
6	DAYS_EMPLOYED	0.012232
99	EXT_SOURCE_3^2 DAYS_BIRTH	0.012225
69	EXT_SOURCE_2_y	0.011958
92	EXT_SOURCE_2^3	0.011914
11	EXT_SOURCE_2_x	0.011462

In [ ]:

```
# plot feature importance
plt.figure(figsize=(15, 7))
plt.bar(x=important_features['Feature Name'].head(30),
        height=important_features['Feature Score'].head(30),
        color='crimson')
plt.xticks(rotation=90)
plt.grid(b=True)
plt.show()
```



In [ ]:
```
filtered_features = important_features[important_features['Feature Score']>=0.001]
print(list(filtered_features['Feature Name']))
```

```
['EXT_SOURCE_1 EXT_SOURCE_2 EXT_SOURCE_3', 'EXT_SOURCE_2 EXT_SOURCE_3 DAYS_BIRTH',
'EXT_SOURCE_2^2 EXT_SOURCE_3', 'EXT_SOURCE_2 EXT_SOURCE_3', 'EXT_SOURCE_2 EXT_SOUR
CE_3^2', 'EXT_SOURCE_1 EXT_SOURCE_2^2', 'EXT_SOURCE_1 EXT_SOURCE_2 DAYS_BIRTH', 'E
XT_SOURCE_1 EXT_SOURCE_3 DAYS_BIRTH', 'EXT_SOURCE_2^2 DAYS_BIRTH', 'EXT_SOURCE_1^2
EXT_SOURCE_3', 'EXT_SOURCE_2 DAYS_BIRTH', 'EXT_SOURCE_1 EXT_SOURCE_3', 'EXT_SOURCE
_1 EXT_SOURCE_3^2', 'EXT_SOURCE_1 EXT_SOURCE_2', 'EXT_SOURCE_2 DAYS_BIRTH^2', 'DAY
S_EMPLOYED', 'EXT_SOURCE_3^2 DAYS_BIRTH', 'EXT_SOURCE_2_y', 'EXT_SOURCE_2^3', 'EXT
_SOURCE_2_x', 'DAYS_ID_PUBLISH', 'DAYS_REGISTRATION', 'EXT_SOURCE_2^2', 'EXT_SOURC
E_3 DAYS_BIRTH^2', 'SK_ID_CURR', 'EXT_SOURCE_3 DAYS_BIRTH', 'EXT_SOURCE_1^2 DAYS_B
IRTH', 'EXT_SOURCE_1^2 EXT_SOURCE_2', 'DAYS_LAST_PHONE_CHANGE', 'EXT_SOURCE_3_x',
'REGION_POPULATION_RELATIVE', 'EXT_SOURCE_3_y', 'DAYS_BIRTH_x', 'bureau_DAYS_CREDI
T_max', 'AMT_INCOME_TOTAL', 'DAYS_BIRTH_y', 'EXT_SOURCE_1 DAYS_BIRTH', 'DAYS_BIRTH
^3', 'DAYS_BIRTH^2', 'EXT_SOURCE_1 DAYS_BIRTH^2', 'bureau_DAYS_CREDIT_ENDDATE_ma
x', 'bureau_DAYS_CREDIT_ENDDATE_mean', 'EXT_SOURCE_3^3', 'bureau_DAYS_CREDIT_mea
n', 'bureau_DAYS_CREDIT_ENDDATE_sum', 'bureau_AMT_CREDIT_SUM_mean', 'EXT_SOURCE_3^
2', 'bureau_AMT_CREDIT_SUM_sum', 'bureau_AMT_CREDIT_SUM_max', 'bureau_DAYS_ENDDATE
_FACT_max', 'bureau_DAYS_CREDIT_min', 'bureau_DAYS_CREDIT_UPDATE_mean', 'bureau_DA
YS_CREDIT_UPDATE_min', 'bureau_DAYS_CREDIT_ENDDATE_min', 'bureau_DAYS_ENDDATE_FACT
_min', 'bureau_DAYS_CREDIT_UPDATE_max', 'bureau_DAYS_CREDIT_sum', 'bureau_DAYS_CRE
DIT_UPDATE_sum', 'bureau_DAYS_ENDDATE_FACT_mean', 'bureau_AMT_CREDIT_SUM_min', 'bu
reau_AMT_CREDIT_SUM_DEBT_mean', 'bureau_AMT_CREDIT_SUM_DEBT_max', 'EXT_SOURCE_1_
x', 'bureau_DAYS_ENDDATE_FACT_sum', 'bureau_AMT_CREDIT_SUM_DEBT_sum', 'OWN_CAR_AG
E', 'EXT_SOURCE_1_y', 'EXT_SOURCE_1^2', 'EXT_SOURCE_1^3', 'TOTALAREA_MODE', 'AMT_R
EQ_CREDIT_BUREAU_YEAR', 'OBS_30_CNT_SOCIAL_CIRCLE', 'LIVINGAREA_MEDI', 'YEARS_BEGI
NEXPLUATATION_MODE', 'LIVINGAREA_MODE', 'YEARS_BEGINEXPLUATATION_MEDI', 'LIVINGARE
A_AVG', 'OBS_60_CNT_SOCIAL_CIRCLE', 'YEARS_BEGINEXPLUATATION_AVG', 'APARTMENTS_MOD
E', 'APARTMENTS_MEDI', 'APARTMENTS_AVG', 'LANDAREA_AVG', 'bureau_AMT_CREDIT_SUM_LI
MIT_count', 'LANDAREA_MEDI', 'bureau_AMT_ANNUITY_max', 'BASEMENTAREA_AVG', 'BASEME
NTAREA_MEDI', 'bureau_AMT_CREDIT_SUM_DEBT_count', 'LANDAREA_MODE', 'previous_loan_
counts', 'bureau_AMT_ANNUITY_mean', 'BASEMENTAREA_MODE', 'bureau_AMT_CREDIT_MAX_OV
ERDUE_mean', 'bureau_DAYS_CREDIT_ENDDATE_count', 'bureau_AMT_CREDIT_MAX_OVERDUE_co
unt', 'bureau_DAYS_ENDDATE_FACT_count', 'COMMONAREA_MODE', 'bureau_AMT_CREDIT_MAX_
OVERDUE_sum', 'bureau_DAYS_CREDIT_UPDATE_count', 'NONLIVINGAREA_MEDI', 'COMMONAREA
_AVG', 'NONLIVINGAREA_AVG', 'bureau_AMT_CREDIT_SUM_count', 'CNT_FAM_MEMBERS', 'bur
eau_AMT_CREDIT_SUM_OVERDUE_count', 'bureau_CREDIT_DAY_OVERDUE_count', 'bureau_AMT_
CREDIT_MAX_OVERDUE_max', 'bureau_DAYS_CREDIT_count', 'bureau_CNT_CREDIT_PROLONG_co
unt', 'COMMONAREA_MEDI', 'NONLIVINGAREA_MODE', 'YEARS_BUILD_MEDI', 'YEARS_BUILD_AV
G', 'LIVINGAPARTMENTS_AVG', 'YEARS_BUILD_MODE', 'LIVINGAPARTMENTS_MODE', 'bureau_A
MT_ANNUITY_sum', 'LIVINGAPARTMENTS_MEDI', 'ENTRANCES_AVG', 'ENTRANCES_MEDI', 'bure
au_AMT_CREDIT_SUM_LIMIT_mean', 'bureau_AMT_CREDIT_SUM_LIMIT_max', 'bureau_AMT_ANNU
ITY_count', 'bureau_AMT_CREDIT_SUM_LIMIT_sum', 'ENTRANCES_MODE', 'NAME_EDUCATION_T
YPE_Higher education', 'CODE_GENDER_M', 'NAME_EDUCATION_TYPE_Secondary / secondary
special', 'CODE_GENDER_F', 'CNT_CHILDREN', 'REGION_RATING_CLIENT', 'REGION_RATING_
CLIENT_W_CITY', 'bureau_AMT_CREDIT_SUM_DEBT_min', 'FLOORSMAX_AVG', 'FLAG_DOCUMENT_
3', 'FLOORSMAX_MEDI', 'FLOORSMAX_MODE', 'FLOORSMIN_MEDI', 'DEF_30_CNT_SOCIAL_CIRCL
E', 'AMT_REQ_CREDIT_BUREAU_QRT', 'FLOORSMIN_AVG', 'bureau_AMT_ANNUITY_min', 'FLAG_
OWN_CAR', 'OCCUPATION_TYPE_Laborers', 'NAME_FAMILY_STATUS_Married', 'AMT_REQ_CREDI
T_BUREAU_MON', 'FLOORSMIN_MODE', 'FLAG_OWN_REALTY', 'DEF_60_CNT_SOCIAL_CIRCLE', 'b
ureau_AMT_CREDIT_MAX_OVERDUE_min', 'NAME_INCOME_TYPE_Working', 'ORGANIZATION_TYPE_
Business Entity Type 3', 'FLAG_PHONE', 'NAME_TYPE_SUITE_Unaccompanied', 'FLAG_WORK
_PHONE', 'ORGANIZATION_TYPE_Self-employed', 'REG_CITY_NOT_WORK_CITY', 'WEEKDAY_APP
R_PROCESS_START_TUESDAY', 'REG_CITY_NOT_LIVE_CITY', 'WEEKDAY_APPR_PROCESS_START_MO
NDAY', 'NAME_FAMILY_STATUS_Single / not married', 'NONLIVINGAPARTMENTS_MEDI', 'ELE
VATORS_AVG', 'WEEKDAY_APPR_PROCESS_START_WEDNESDAY', 'WEEKDAY_APPR_PROCESS_START_F
RIDAY', 'NONLIVINGAPARTMENTS_AVG', 'NAME_TYPE_SUITE_Family', 'WEEKDAY_APPR_PROCESS
_START_THURSDAY', 'LIVE_CITY_NOT_WORK_CITY', 'NAME_INCOME_TYPE_Commercial associat
e', 'ELEVATORS_MEDI', 'WEEKDAY_APPR_PROCESS_START_SATURDAY', 'NONLIVINGAPARTMENTS_
MODE', 'OCCUPATION_TYPE_Drivers', 'OCCUPATION_TYPE_Core staff']
```

In [ ]:
```
selected_features = list(filtered_features['Feature Name'])
selected_features
```

Out[ ]:
```
['EXT_SOURCE_1 EXT_SOURCE_2 EXT_SOURCE_3',
 'EXT_SOURCE_2 EXT_SOURCE_3 DAYS_BIRTH',
 'EXT_SOURCE_2^2 EXT_SOURCE_3',
 'EXT_SOURCE_2 EXT_SOURCE_3',
 'EXT_SOURCE_2 EXT_SOURCE_3^2',
 'EXT_SOURCE_1 EXT_SOURCE_2^2',
 'EXT_SOURCE_1 EXT_SOURCE_2 DAYS_BIRTH',
 'EXT_SOURCE_1 EXT_SOURCE_3 DAYS_BIRTH',
 'EXT_SOURCE_2^2 DAYS_BIRTH',
 'EXT_SOURCE_1^2 EXT_SOURCE_3',
 'EXT_SOURCE_2 DAYS_BIRTH',
 'EXT_SOURCE_1 EXT_SOURCE_3',
 'EXT_SOURCE_1 EXT_SOURCE_3^2',
 'EXT_SOURCE_1 EXT_SOURCE_2',
 'EXT_SOURCE_2 DAYS_BIRTH^2',
 'DAYS_EMPLOYED',
 'EXT_SOURCE_3^2 DAYS_BIRTH',
 'EXT_SOURCE_2_y',
 'EXT_SOURCE_2^3',
 'EXT_SOURCE_2_x',
 'DAYS_ID_PUBLISH',
 'DAYS_REGISTRATION',
 'EXT_SOURCE_2^2',
 'EXT_SOURCE_3 DAYS_BIRTH^2',
 'SK_ID_CURR',
 'EXT_SOURCE_3 DAYS_BIRTH',
 'EXT_SOURCE_1^2 DAYS_BIRTH',
 'EXT_SOURCE_1^2 EXT_SOURCE_2',
 'DAYS_LAST_PHONE_CHANGE',
 'EXT_SOURCE_3_x',
 'REGION_POPULATION_RELATIVE',
 'EXT_SOURCE_3_y',
 'DAYS_BIRTH_x',
 'bureau_DAYS_CREDIT_max',
 'AMT_INCOME_TOTAL',
 'DAYS_BIRTH_y',
 'EXT_SOURCE_1 DAYS_BIRTH',
 'DAYS_BIRTH^3',
 'DAYS_BIRTH^2',
 'EXT_SOURCE_1 DAYS_BIRTH^2',
 'bureau_DAYS_CREDIT_ENDDATE_max',
 'bureau_DAYS_CREDIT_ENDDATE_mean',
 'EXT_SOURCE_3^3',
 'bureau_DAYS_CREDIT_mean',
 'bureau_DAYS_CREDIT_ENDDATE_sum',
 'bureau_AMT_CREDIT_SUM_mean',
 'EXT_SOURCE_3^2',
 'bureau_AMT_CREDIT_SUM_sum',
 'bureau_AMT_CREDIT_SUM_max',
 'bureau_DAYS_ENDDATE_FACT_max',
 'bureau_DAYS_CREDIT_min',
 'bureau_DAYS_CREDIT_UPDATE_mean',
 'bureau_DAYS_CREDIT_UPDATE_min',
 'bureau_DAYS_CREDIT_ENDDATE_min',
 'bureau_DAYS_ENDDATE_FACT_min',
 'bureau_DAYS_CREDIT_UPDATE_max',
 'bureau_DAYS_CREDIT_sum',
 'bureau_DAYS_CREDIT_UPDATE_sum',
 'bureau_DAYS_ENDDATE_FACT_mean',
 'bureau_AMT_CREDIT_SUM_min',
 'bureau_AMT_CREDIT_SUM_DEBT_mean',
 'bureau_AMT_CREDIT_SUM_DEBT_max',
 'EXT_SOURCE_1_x',
 'bureau_DAYS_ENDDATE_FACT_sum',
```

```
'bureau_AMT_CREDIT_SUM_DEBT_sum',
'OWN_CAR_AGE',
'EXT_SOURCE_1_y',
'EXT_SOURCE_1^2',
'EXT_SOURCE_1^3',
'TOTALAREA_MODE',
'AMT_REQ_CREDIT_BUREAU_YEAR',
'OBS_30_CNT_SOCIAL_CIRCLE',
'LIVINGAREA_MEDI',
'YEARS_BEGINEXPLUATATION_MODE',
'LIVINGAREA_MODE',
'YEARS_BEGINEXPLUATATION_MEDI',
'LIVINGAREA_AVG',
'OBS_60_CNT_SOCIAL_CIRCLE',
'YEARS_BEGINEXPLUATATION_AVG',
'APARTMENTS_MODE',
'APARTMENTS_MEDI',
'APARTMENTS_AVG',
'LANDAREA_AVG',
'bureau_AMT_CREDIT_SUM_LIMIT_count',
'LANDAREA_MEDI',
'bureau_AMT_ANNUITY_max',
'BASEMENTAREA_AVG',
'BASEMENTAREA_MEDI',
'bureau_AMT_CREDIT_SUM_DEBT_count',
'LANDAREA_MODE',
'previous_loan_counts',
'bureau_AMT_ANNUITY_mean',
'BASEMENTAREA_MODE',
'bureau_AMT_CREDIT_MAX_OVERDUE_mean',
'bureau_DAYS_CREDIT_ENDDATE_count',
'bureau_AMT_CREDIT_MAX_OVERDUE_count',
'bureau_DAYS_ENDDATE_FACT_count',
'COMMONAREA_MODE',
'bureau_AMT_CREDIT_MAX_OVERDUE_sum',
'bureau_DAYS_CREDIT_UPDATE_count',
'NONLIVINGAREA_MEDI',
'COMMONAREA_AVG',
'NONLIVINGAREA_AVG',
'bureau_AMT_CREDIT_SUM_count',
'CNT_FAM_MEMBERS',
'bureau_AMT_CREDIT_SUM_OVERDUE_count',
'bureau_CREDIT_DAY_OVERDUE_count',
'bureau_AMT_CREDIT_MAX_OVERDUE_max',
'bureau_DAYS_CREDIT_count',
'bureau_CNT_CREDIT_PROLONG_count',
'COMMONAREA_MEDI',
'NONLIVINGAREA_MODE',
'YEARS_BUILD_MEDI',
'YEARS_BUILD_AVG',
'LIVINGAPARTMENTS_AVG',
'YEARS_BUILD_MODE',
'LIVINGAPARTMENTS_MODE',
'bureau_AMT_ANNUITY_sum',
'LIVINGAPARTMENTS_MEDI',
'ENTRANCES_AVG',
'ENTRANCES_MEDI',
'bureau_AMT_CREDIT_SUM_LIMIT_mean',
'bureau_AMT_CREDIT_SUM_LIMIT_max',
'bureau_AMT_ANNUITY_count',
'bureau_AMT_CREDIT_SUM_LIMIT_sum',
'ENTRANCES_MODE',
'NAME_EDUCATION_TYPE_Higher education',
'CODE_GENDER_M',
```

```
'NAME_EDUCATION_TYPE_Secondary / secondary special',
'CODE_GENDER_F',
'CNT_CHILDREN',
'REGION_RATING_CLIENT',
'REGION_RATING_CLIENT_W_CITY',
'bureau_AMT_CREDIT_SUM_DEBT_min',
'FLOORSMAX_AVG',
'FLAG_DOCUMENT_3',
'FLOORSMAX_MEDI',
'FLOORSMAX_MODE',
'FLOORSMIN_MEDI',
'DEF_30_CNT_SOCIAL_CIRCLE',
'AMT_REQ_CREDIT_BUREAU_QRT',
'FLOORSMIN_AVG',
'bureau_AMT_ANNUITY_min',
'FLAG_OWN_CAR',
'OCCUPATION_TYPE_Laborers',
'NAME_FAMILY_STATUS_Married',
'AMT_REQ_CREDIT_BUREAU_MON',
'FLOORSMIN_MODE',
'FLAG_OWN_REALTY',
'DEF_60_CNT_SOCIAL_CIRCLE',
'bureau_AMT_CREDIT_MAX_OVERDUE_min',
'NAME_INCOME_TYPE_Working',
'ORGANIZATION_TYPE_Business Entity Type 3',
'FLAG_PHONE',
'NAME_TYPE_SUITE_Unaccompanied',
'FLAG_WORK_PHONE',
'ORGANIZATION_TYPE_Self-employed',
'REG_CITY_NOT_WORK_CITY',
'WEEKDAY_APPR_PROCESS_START_TUESDAY',
'REG_CITY_NOT_LIVE_CITY',
'WEEKDAY_APPR_PROCESS_START_MONDAY',
'NAME_FAMILY_STATUS_Single / not married',
'NONLIVINGAPARTMENTS_MEDI',
'ELEVATORS_AVG',
'WEEKDAY_APPR_PROCESS_START_WEDNESDAY',
'WEEKDAY_APPR_PROCESS_START_FRIDAY',
'NONLIVINGAPARTMENTS_AVG',
'NAME_TYPE_SUITE_Family',
'WEEKDAY_APPR_PROCESS_START_THURSDAY',
'LIVE_CITY_NOT_WORK_CITY',
'NAME_INCOME_TYPE_Commercial associate',
'ELEVATORS_MEDI',
'WEEKDAY_APPR_PROCESS_START_SATURDAY',
'NONLIVINGAPARTMENTS_MODE',
'OCCUPATION_TYPE_Drivers',
'OCCUPATION_TYPE_Core staff']
```

In [ ]:

Hyperparameter Tuning using GridSearchCV

In [ ]:
```
from sklearn.metrics import roc_auc_score
# Hyperparameter Tuning using GridSearchCV on UnderSampled Data

sub_x_train = X_under
sub_y_train = y_under
```

```
# Craeting a general pipeline for performing hyperparameter tuning
clf_pipe = Pipeline([("preparation", data_prep_pipeline),
                     ("pca", PCA(n_components=0.85)),
                     ("classifier", None)]) # Placeholder Estimator
```

In [ ]:
```
try:
    del expLog
    expLog
except NameError:
    expLog = pd.DataFrame(columns=["Model name",
                                   "Best Model Parameters",
                                   "Train Acc",
                                   "Valid Acc",
                                   "Test  Acc",
                                   "Train AUC",
                                   "Valid AUC",
                                   "Test  AUC",
                                   "Train F1",
                                   "Valid F1",
                                   "Test F1",
                                   "Tune Time"
                                  ])
expLog
```

Out[ ]:

Model name	Best Model Parameters	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC	Train F1	Valid F1	Test F1	Tune Time

In [ ]:
```
parameters = [{'classifier__penalty':['l1','l2','elasticnet'], 'classifier__solver
start_time = time.time()
clf_gridsearch = GridSearchCV(clf_pipe, param_grid = parameters, cv = 5, scoring='
clf_gridsearch.fit(sub_x_train, sub_y_train)
tune_time = time.time() - start_time
model_name = "Tuned {}".format(type(clf_gridsearch.best_estimator_['classifier']).
print('Tune Time for {} is: {} seconds'.format(model_name, tune_time))
expLog.loc[len(expLog)] = [model_name] + [clf_gridsearch.best_params_] + list(np.r
               [accuracy_score(sub_y_train, clf_gridsearch.predict(sub_x_train)),
                accuracy_score(y_valid, clf_gridsearch.predict(X_valid)),
                accuracy_score(y_test, clf_gridsearch.predict(X_test)),
                roc_auc_score(sub_y_train, clf_gridsearch.predict_proba(sub_x_trai
                roc_auc_score(y_valid, clf_gridsearch.predict_proba(X_valid)[:, 1]
                roc_auc_score(y_test, clf_gridsearch.predict_proba(X_test)[:, 1]),
                f1_score(sub_y_train, clf_gridsearch.predict(sub_x_train)),
                f1_score(y_valid, clf_gridsearch.predict(X_valid)),
                f1_score(y_test, clf_gridsearch.predict(X_test)),
                tune_time], 4))
expLog
```

```
Tune Time for Tuned LogisticRegression is: 31.09601092338562 seconds
```

Out[ ]:

	Model name	Best Model Parameters	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC	Train F1
0	Tuned LogisticRegression	{'classifier': LogisticRegression(C=1, class_w...	0.6679	0.64	0.6467	0.7289	0.7181	0.7269	0.6765

In [ ]:
```
y_kaggle_test_scores = clf_gridsearch.predict_proba(X_kaggle_test)[:, 1]
y_kaggle_test = X_kaggle_test[['SK_ID_CURR']]
y_kaggle_test['TARGET'] = y_kaggle_test_scores
```

```
print(y_kaggle_test.head())
y_kaggle_test.to_csv("lr_tuned_submission.csv", index=False)
```

```
   SK_ID_CURR    TARGET
0      100001  0.370432
1      100005  0.587965
2      100013  0.226698
3      100028  0.328114
4      100038  0.654021
```

GradientBoostingClassifier

In [ ]:
```
parameters = [
    {'classifier__n_iter_no_change': [5], 'classifier__n_estimators': [100,200,300
]
start_time = time.time()
clf_gridsearch = GridSearchCV(clf_pipe, param_grid = parameters, cv = 5, scoring='
clf_gridsearch.fit(sub_x_train, sub_y_train)
tune_time = time.time() - start_time
model_name = "Tuned {}".format(type(clf_gridsearch.best_estimator_['classifier']).
print('Tune Time for {} is: {} seconds'.format(model_name, tune_time))
expLog.loc[len(expLog)] = [model_name] + [clf_gridsearch.best_params_] + list(np.r
               [accuracy_score(sub_y_train, clf_gridsearch.predict(sub_x_train)),
                accuracy_score(y_valid, clf_gridsearch.predict(X_valid)),
                accuracy_score(y_test, clf_gridsearch.predict(X_test)),
                roc_auc_score(sub_y_train, clf_gridsearch.predict_proba(sub_x_trai
                roc_auc_score(y_valid, clf_gridsearch.predict_proba(X_valid)[:, 1]
                roc_auc_score(y_test, clf_gridsearch.predict_proba(X_test)[:, 1]),
                f1_score(sub_y_train, clf_gridsearch.predict(sub_x_train)),
                f1_score(y_valid, clf_gridsearch.predict(X_valid)),
                f1_score(y_test, clf_gridsearch.predict(X_test)),
                tune_time], 4))
expLog
```

```
Tune Time for Tuned GradientBoostingClassifier is: 188.27889823913574 seconds
```

Out[ ]:

	Model name	Best Model Parameters	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC
0	Tuned LogisticRegression	{'classifier': LogisticRegression(C=1, class_w...	0.6679	0.6400	0.6467	0.7289	0.7181
1	Tuned LogisticRegression	{'classifier': LogisticRegression(C=1, class_w...	0.6679	0.6400	0.6467	0.7289	0.7181
2	Tuned GradientBoostingClassifier	{'classifier': GradientBoostingClassifier(max_...	0.7056	0.6606	0.6665	0.7803	0.7204

In [ ]:
```
y_kaggle_test_scores = clf_gridsearch.predict_proba(X_kaggle_test)[:, 1]
y_kaggle_test = X_kaggle_test[['SK_ID_CURR']]
y_kaggle_test['TARGET'] = y_kaggle_test_scores
print(y_kaggle_test.head())
y_kaggle_test.to_csv("gbc_tuned_submission.csv", index=False)
```

```
   SK_ID_CURR    TARGET
0      100001  0.258293
1      100005  0.574519
2      100013  0.239819
3      100028  0.195836
4      100038  0.665051
```

DecisionTreeClassifier

In [ ]:
```
parameters = [{'classifier__max_depth': [5,7,10, None], 'classifier__min_samples_l
start_time = time.time()
clf_gridsearch = GridSearchCV(clf_pipe, param_grid = parameters, cv = 5, scoring='
clf_gridsearch.fit(sub_x_train, sub_y_train)
tune_time = time.time() - start_time
model_name = "Tuned {}".format(type(clf_gridsearch.best_estimator_['classifier']).
print('Tune Time for {} is: {} seconds'.format(model_name, tune_time))
expLog.loc[len(expLog)] = [model_name] + [clf_gridsearch.best_params_] + list(np.r
               [accuracy_score(sub_y_train, clf_gridsearch.predict(sub_x_train)),
                accuracy_score(y_valid, clf_gridsearch.predict(X_valid)),
                accuracy_score(y_test, clf_gridsearch.predict(X_test)),
                roc_auc_score(sub_y_train, clf_gridsearch.predict_proba(sub_x_trai
                roc_auc_score(y_valid, clf_gridsearch.predict_proba(X_valid)[:, 1]
                roc_auc_score(y_test, clf_gridsearch.predict_proba(X_test)[:, 1]),
                f1_score(sub_y_train, clf_gridsearch.predict(sub_x_train)),
                f1_score(y_valid, clf_gridsearch.predict(X_valid)),
                f1_score(y_test, clf_gridsearch.predict(X_test)),
                tune_time], 4))
expLog
```

```
Tune Time for Tuned DecisionTreeClassifier is: 11.041849374771118 seconds
```

Out[ ]:

	Model name	Best Model Parameters	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC
0	Tuned LogisticRegression	{'classifier': LogisticRegression(C=1, class_w...	0.6679	0.6400	0.6467	0.7289	0.7181
1	Tuned LogisticRegression	{'classifier': LogisticRegression(C=1, class_w...	0.6679	0.6400	0.6467	0.7289	0.7181
2	Tuned GradientBoostingClassifier	{'classifier': GradientBoostingClassifier(max_...	0.7056	0.6606	0.6665	0.7803	0.7204
3	Tuned DecisionTreeClassifier	{'classifier': DecisionTreeClassifier(max_dept...	0.7312	0.6168	0.6182	0.8095	0.6629

In [ ]:
```
y_kaggle_test_scores = clf_gridsearch.predict_proba(X_kaggle_test)[:, 1]
y_kaggle_test = X_kaggle_test[['SK_ID_CURR']]
y_kaggle_test['TARGET'] = y_kaggle_test_scores
print(y_kaggle_test.head())
y_kaggle_test.to_csv("dt_tuned_submission.csv", index=False)
```

```
   SK_ID_CURR    TARGET
0      100001  0.171429
1      100005  0.090909
2      100013  0.155797
3      100028  0.428571
4      100038  0.814286
```

RandomForestClassifier

In [ ]:
```
parameters = [{'classifier__n_estimators': [100,200,300], 'classifier__max_depth':
start_time = time.time()
clf_gridsearch = GridSearchCV(clf_pipe, param_grid = parameters, cv = 5, scoring='
clf_gridsearch.fit(sub_x_train, sub_y_train)
tune_time = time.time() - start_time
model_name = "Tuned {}".format(type(clf_gridsearch.best_estimator_['classifier']).
print('Tune Time for {} is: {} seconds'.format(model_name, tune_time))
```

```
expLog.loc[len(expLog)] = [model_name] + [clf_gridsearch.best_params_] + list(np.r
               [accuracy_score(sub_y_train, clf_gridsearch.predict(sub_x_train)),
                accuracy_score(y_valid, clf_gridsearch.predict(X_valid)),
                accuracy_score(y_test, clf_gridsearch.predict(X_test)),
                roc_auc_score(sub_y_train, clf_gridsearch.predict_proba(sub_x_trai
                roc_auc_score(y_valid, clf_gridsearch.predict_proba(X_valid)[:, 1]
                roc_auc_score(y_test, clf_gridsearch.predict_proba(X_test)[:, 1]),
                f1_score(sub_y_train, clf_gridsearch.predict(sub_x_train)),
                f1_score(y_valid, clf_gridsearch.predict(X_valid)),
                f1_score(y_test, clf_gridsearch.predict(X_test)),
                tune_time], 4))
expLog
```

```
Tune Time for Tuned RandomForestClassifier is: 89.051 seconds
```

Out[ ]:

	Model name	Best Model Parameters	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC
0	Tuned LogisticRegression	{'classifier': LogisticRegression(C=1, class_w...	0.6679	0.6400	0.6467	0.7289	0.718
1	Tuned LogisticRegression	{'classifier': LogisticRegression(C=1, class_w...	0.6679	0.6400	0.6467	0.7289	0.718
2	Tuned GradientBoostingClassifier	{'classifier': GradientBoostingClassifier(max_...	0.7056	0.6606	0.6665	0.7803	0.720
3	Tuned DecisionTreeClassifier	{'classifier': DecisionTreeClassifier(max_dept...	0.7312	0.6168	0.6182	0.8095	0.662
4	Tuned RandomForestClassifier	{'classifier': RandomForestClassifier(min_samp...	0.9649	0.6669	0.6702	0.9956	0.713

In [ ]:

```
y_kaggle_test_scores = clf_gridsearch.predict_proba(X_kaggle_test)[:, 1]
y_kaggle_test = X_kaggle_test[['SK_ID_CURR']]
y_kaggle_test['TARGET'] = y_kaggle_test_scores
print(y_kaggle_test.head())
y_kaggle_test.to_csv("rfc_tuned_submission.csv", index=False)
```

```
   SK_ID_CURR    TARGET
0      100001  0.321775
1      100005  0.611232
2      100013  0.226083
3      100028  0.326351
4      100038  0.743761
```

LGBMClassifier

In [ ]:

```
# Hyperparameter Tuning using GridSearchCV on UnderSampled Data
# Craeting a general pipeline for performing hyperparameter tuning
clf_pipe_lgbm = Pipeline([("preparation", data_prep_pipeline),
                          ("pca", PCA(n_components=0.85)),
                          ("classifier", LGBMClassifier(random_state=42,
                                                        n_estimators=50))]) # Defining
```

In [ ]:

```
parameters = [{'classifier__learning_rate':[0.01,0.05,0.1],'classifier__num_leaves
start_time = time.time()
clf_gridsearch = GridSearchCV(clf_pipe_lgbm, param_grid = parameters, cv = 3, scor
clf_gridsearch.fit(sub_x_train, sub_y_train.values.ravel())
tune_time = time.time() - start_time
model_name = "Tuned {}".format(type(clf_gridsearch.best_estimator_['classifier']).
print('Tune Time for {} is: {} seconds'.format(model_name, tune_time))
```

```
expLog.loc[len(expLog)] = [model_name] + [clf_gridsearch.best_params_] + list(np.r
               [accuracy_score(sub_y_train, clf_gridsearch.predict(sub_x_train)),
                accuracy_score(y_valid, clf_gridsearch.predict(X_valid)),
                accuracy_score(y_test, clf_gridsearch.predict(X_test)),
                roc_auc_score(sub_y_train, clf_gridsearch.predict_proba(sub_x_trai
                roc_auc_score(y_valid, clf_gridsearch.predict_proba(X_valid)[:, 1]
                roc_auc_score(y_test, clf_gridsearch.predict_proba(X_test)[:, 1]),
                f1_score(sub_y_train, clf_gridsearch.predict(sub_x_train)),
                f1_score(y_valid, clf_gridsearch.predict(X_valid)),
                f1_score(y_test, clf_gridsearch.predict(X_test)),
                tune_time], 4))
expLog
```

```
Tune Time for Tuned LGBMClassifier is: 34.474247217178345 seconds
```

Out[ ]:

	Model name	Best Model Parameters	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC
0	Tuned LogisticRegression	{'classifier': LogisticRegression(C=1, class_w...	0.6679	0.6400	0.6467	0.7289	0.718
1	Tuned LogisticRegression	{'classifier': LogisticRegression(C=1, class_w...	0.6679	0.6400	0.6467	0.7289	0.718
2	Tuned GradientBoostingClassifier	{'classifier': GradientBoostingClassifier(max_...	0.7056	0.6606	0.6665	0.7803	0.720
3	Tuned DecisionTreeClassifier	{'classifier': DecisionTreeClassifier(max_dept...	0.7312	0.6168	0.6182	0.8095	0.662
4	Tuned RandomForestClassifier	{'classifier': RandomForestClassifier(min_samp...	0.9649	0.6669	0.6702	0.9956	0.713
5	Tuned CatBoostClassifier	{'classifier': <catboost.core.CatBoostClassifi...	0.6568	0.6568	0.6616	0.7164	0.707
6	Tuned XGBClassifier	{'classifier': XGBClassifier(base_score=None, ...	0.7378	0.6562	0.6598	0.8192	0.712
7	Tuned LGBMClassifier	{'classifier__learning_rate': 0.1, 'classifier...	0.7025	0.6691	0.6744	0.7754	0.722

In [ ]:

```
y_kaggle_test_scores = clf_gridsearch.predict_proba(X_kaggle_test)[:, 1]
y_kaggle_test = X_kaggle_test[['SK_ID_CURR']]
y_kaggle_test['TARGET'] = y_kaggle_test_scores
print(y_kaggle_test.head())
y_kaggle_test.to_csv("lgbm_tuned_submission.csv", index=False)
```

```
   SK_ID_CURR    TARGET
0      100001  0.254851
1      100005  0.537879
2      100013  0.250800
3      100028  0.255759
4      100038  0.695114
```

In [ ]:

```
expLog
```

Out[ ]:

	Model name	Best Model Parameters	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC
0	Tuned LogisticRegression	{'classifier': LogisticRegression(C=1, class_w...	0.6679	0.6400	0.6467	0.7289	0.718
1	Tuned LogisticRegression	{'classifier': LogisticRegression(C=1, class_w...	0.6679	0.6400	0.6467	0.7289	0.718
2	Tuned GradientBoostingClassifier	{'classifier': GradientBoostingClassifier(max_...	0.7056	0.6606	0.6665	0.7803	0.720
3	Tuned DecisionTreeClassifier	{'classifier': DecisionTreeClassifier(max_dept...	0.7312	0.6168	0.6182	0.8095	0.662
4	Tuned RandomForestClassifier	{'classifier': RandomForestClassifier(min_samp...	0.9649	0.6669	0.6702	0.9956	0.713
5	Tuned CatBoostClassifier	{'classifier': <catboost.core.CatBoostClassifi...	0.6568	0.6568	0.6616	0.7164	0.707
6	Tuned XGBClassifier	{'classifier': XGBClassifier(base_score=None, ...	0.7378	0.6562	0.6598	0.8192	0.712
7	Tuned LGBMClassifier	{'classifier__learning_rate': 0.1, 'classifier...	0.7025	0.6691	0.6744	0.7754	0.722

In [ ]:
```
expLog.to_csv('expLog.csv', index=False)
```

CatBoostClassifier

In [ ]:
```
parameters = [{'classifier__max_depth': [4,5,6,7,8,9,10],'classifier__learning_rat
start_time = time.time()
clf_gridsearch = GridSearchCV(clf_pipe, param_grid = parameters, cv = 5, scoring='
clf_gridsearch.fit(sub_x_train, sub_y_train.values.ravel())
tune_time = time.time() - start_time
model_name = "Tuned {}".format(type(clf_gridsearch.best_estimator_['classifier']).
print('Tune Time for {} is: {} seconds'.format(model_name, tune_time))
expLog.loc[len(expLog)] = [model_name] + [clf_gridsearch.best_params_] + list(np.r
               [accuracy_score(sub_y_train, clf_gridsearch.predict(sub_x_train)),
                accuracy_score(y_valid, clf_gridsearch.predict(X_valid)),
                accuracy_score(y_test, clf_gridsearch.predict(X_test)),
                roc_auc_score(sub_y_train, clf_gridsearch.predict_proba(sub_x_trai
                roc_auc_score(y_valid, clf_gridsearch.predict_proba(X_valid)[:, 1]
                roc_auc_score(y_test, clf_gridsearch.predict_proba(X_test)[:, 1]),
                f1_score(sub_y_train, clf_gridsearch.predict(sub_x_train)),
                f1_score(y_valid, clf_gridsearch.predict(X_valid)),
                f1_score(y_test, clf_gridsearch.predict(X_test)),
                tune_time], 4))
expLog
```

```
Tune Time for Tuned CatBoostClassifier is: 1716.3430893421173 seconds
```

Out[ ]:

	Model name	Best Model Parameters	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC
0	Tuned LogisticRegression	{'classifier': LogisticRegression(C=1, class_w...	0.6679	0.6400	0.6467	0.7289	0.718
1	Tuned LogisticRegression	{'classifier': LogisticRegression(C=1, class_w...	0.6679	0.6400	0.6467	0.7289	0.718
2	Tuned GradientBoostingClassifier	{'classifier': GradientBoostingClassifier(max_...	0.7056	0.6606	0.6665	0.7803	0.720
3	Tuned DecisionTreeClassifier	{'classifier': DecisionTreeClassifier(max_dept...	0.7312	0.6168	0.6182	0.8095	0.662
4	Tuned RandomForestClassifier	{'classifier': RandomForestClassifier(min_samp...	0.9649	0.6669	0.6702	0.9956	0.713
5	Tuned CatBoostClassifier	{'classifier':	0.6568	0.6568	0.6616	0.7164	0.707

In [ ]:
```
y_kaggle_test_scores = clf_gridsearch.predict_proba(X_kaggle_test)[:, 1]
y_kaggle_test = X_kaggle_test[['SK_ID_CURR']]
y_kaggle_test['TARGET'] = y_kaggle_test_scores
print(y_kaggle_test.head())
y_kaggle_test.to_csv("cbc_tuned_submission.csv", index=False)
```

```
   SK_ID_CURR    TARGET
0      100001  0.457818
1      100005  0.534093
2      100013  0.429211
3      100028  0.433899
4      100038  0.545063
```

In [ ]:
```
expLog.to_csv('expLog.csv', index=False)
```

XGBoostClassifier

In [ ]:
```
parameters = [
    {'classifier__n_estimators': [50, 100, 200], 'classifier__max_depth': [3,5,7],
]
start_time = time.time()
clf_gridsearch = GridSearchCV(clf_pipe, param_grid = parameters, cv = 5, scoring='
clf_gridsearch.fit(sub_x_train, sub_y_train.values.ravel())
tune_time = time.time() - start_time
model_name = "Tuned {}".format(type(clf_gridsearch.best_estimator_['classifier']).
print('Tune Time for {} is: {} seconds'.format(model_name, tune_time))
expLog.loc[len(expLog)] = [model_name] + [clf_gridsearch.best_params_] + list(np.r
               [accuracy_score(sub_y_train, clf_gridsearch.predict(sub_x_train)),
                accuracy_score(y_valid, clf_gridsearch.predict(X_valid)),
                accuracy_score(y_test, clf_gridsearch.predict(X_test)),
                roc_auc_score(sub_y_train, clf_gridsearch.predict_proba(sub_x_trai
                roc_auc_score(y_valid, clf_gridsearch.predict_proba(X_valid)[:, 1]
                roc_auc_score(y_test, clf_gridsearch.predict_proba(X_test)[:, 1]),
                f1_score(sub_y_train, clf_gridsearch.predict(sub_x_train)),
                f1_score(y_valid, clf_gridsearch.predict(X_valid)),
                f1_score(y_test, clf_gridsearch.predict(X_test)),
                tune_time], 4))
expLog
```

```
Tune Time for Tuned XGBClassifier is: 3296.56604886055 seconds
```

Out[ ]:

	Model name	Best Model Parameters	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC
0	Tuned LogisticRegression	{'classifier': LogisticRegression(C=1, class_w...	0.6679	0.6400	0.6467	0.7289	0.718
1	Tuned LogisticRegression	{'classifier': LogisticRegression(C=1, class_w...	0.6679	0.6400	0.6467	0.7289	0.718
2	Tuned GradientBoostingClassifier	{'classifier': GradientBoostingClassifier(max_...	0.7056	0.6606	0.6665	0.7803	0.720
3	Tuned DecisionTreeClassifier	{'classifier': DecisionTreeClassifier(max_dept...	0.7312	0.6168	0.6182	0.8095	0.662
4	Tuned RandomForestClassifier	{'classifier': RandomForestClassifier(min_samp...	0.9649	0.6669	0.6702	0.9956	0.713
5	Tuned CatBoostClassifier	{'classifier': <catboost.core.CatBoostClassifi...	0.6568	0.6568	0.6616	0.7164	0.707
6	Tuned XGBClassifier	{'classifier':	0.7278	0.6562	0.6598	0.8192	0.712

In [ ]:
```
y_kaggle_test_scores = clf_gridsearch.predict_proba(X_kaggle_test)[:, 1]
y_kaggle_test = X_kaggle_test[['SK_ID_CURR']]
y_kaggle_test['TARGET'] = y_kaggle_test_scores
print(y_kaggle_test.head())
y_kaggle_test.to_csv("xgb_tuned_submission.csv", index=False)
```

```
   SK_ID_CURR    TARGET
0      100001  0.441820
1      100005  0.509447
2      100013  0.161926
3      100028  0.283194
4      100038  0.728463
```

In [ ]:
```
expLog.to_csv('expLog.csv', index=False)
```

AdaBoost Classifier

In [ ]:
```
from sklearn.ensemble import AdaBoostClassifier
parameters = [{'classifier__n_estimators': [10, 50, 100, 500], 'classifier__learni
start_time = time.time()
clf_gridsearch = GridSearchCV(clf_pipe, param_grid = parameters, cv = 5, scoring='
clf_gridsearch.fit(sub_x_train, sub_y_train)
tune_time = time.time() - start_time
model_name = "Tuned {}".format(type(clf_gridsearch.best_estimator_['classifier']).
print('Tune Time for {} is: {} seconds'.format(model_name, tune_time))
expLog.loc[len(expLog)] = [model_name] + [clf_gridsearch.best_params_] + list(np.r
                [accuracy_score(sub_y_train, clf_gridsearch.predict(sub_x_train)),
                 accuracy_score(y_valid, clf_gridsearch.predict(X_valid)),
                 accuracy_score(y_test, clf_gridsearch.predict(X_test)),
                 roc_auc_score(sub_y_train, clf_gridsearch.predict_proba(sub_x_trai
                 roc_auc_score(y_valid, clf_gridsearch.predict_proba(X_valid)[:, 1]
                 roc_auc_score(y_test, clf_gridsearch.predict_proba(X_test)[:, 1]),
                 f1_score(sub_y_train, clf_gridsearch.predict(sub_x_train)),
                 f1_score(y_valid, clf_gridsearch.predict(X_valid)),
                 f1_score(y_test, clf_gridsearch.predict(X_test)),
                 tune_time], 4))
expLog
```

In [ ]:
```
y_kaggle_test_scores = clf_gridsearch.predict_proba(X_kaggle_test)[:, 1]
y_kaggle_test = X_kaggle_test[['SK_ID_CURR']]
y_kaggle_test['TARGET'] = y_kaggle_test_scores
print(y_kaggle_test.head())
y_kaggle_test.to_csv("adab_tuned_submission.csv", index=False)
```

In [ ]:
```
expLog
```

Out[ ]:

	Model name	Best Model Parameters	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC
0	Tuned LogisticRegression	{'classifier': LogisticRegression(C=1, class_w...	0.6679	0.6400	0.6467	0.7289	0.718
1	Tuned LogisticRegression	{'classifier': LogisticRegression(C=1, class_w...	0.6679	0.6400	0.6467	0.7289	0.718
2	Tuned GradientBoostingClassifier	{'classifier': GradientBoostingClassifier(max_...	0.7056	0.6606	0.6665	0.7803	0.720
3	Tuned DecisionTreeClassifier	{'classifier': DecisionTreeClassifier(max_dept...	0.7312	0.6168	0.6182	0.8095	0.662
4	Tuned RandomForestClassifier	{'classifier': RandomForestClassifier(min_samp...	0.9649	0.6669	0.6702	0.9956	0.713
5	Tuned CatBoostClassifier	{'classifier': <catboost.core.CatBoostClassifi...	0.6568	0.6568	0.6616	0.7164	0.707
6	Tuned XGBClassifier	{'classifier': XGBClassifier(base_score=None, ...	0.7378	0.6562	0.6598	0.8192	0.712
7	Tuned LGBMClassifier	{'classifier__learning_rate': 0.1, 'classifier...	0.7025	0.6691	0.6744	0.7754	0.722
8	Tuned AdaBoostClassifier	{'classifier': AdaBoostClassifier(algorithm='S...	0.6547	0.6363	0.6398	0.6646	0.658

PyTorch NN

- For our MLPs, we will use BinaryCrossEntropy Loss for optimization

$$BinaryCrossEntropy = H_p(q) = -\frac{1}{N}\sum_{i=1}^{n} y_i . log(p(y_i)) + (1 - y_i) . log(1 - p(y_i))$$

- We will create a basic architecture of the MLP based on sampled rows and columns and already imputed data, which is converted to tensors before being inputted to MLP.

In [ ]:
```
import pandas as pd
import numpy as np
from torchsummary import summary
import torch
import torch.utils.data
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
from sklearn.metrics import f1_score
```

```
from sklearn.model_selection import GridSearchCV
import warnings
warnings.filterwarnings('ignore')
```

In [ ]:
```
X_train = pd.read_csv('Processed_Data/X_train.csv')
X_valid = pd.read_csv('Processed_Data/X_valid.csv')
X_test = pd.read_csv('Processed_Data/X_test.csv')
y_train = pd.read_csv('Processed_Data/y_train.csv')
y_valid = pd.read_csv('Processed_Data/y_valid.csv')
y_test = pd.read_csv('Processed_Data/y_test.csv')
X_kaggle_test = pd.read_csv('Processed_Data/X_kaggle_test.csv')

from imblearn.under_sampling import RandomUnderSampler
# define undersample strategy
random_undersampler = RandomUnderSampler(sampling_strategy='majority')
# fit and apply the transform
X_under, y_under = random_undersampler.fit_resample(X_train, y_train)
```

Creating the Network

In [ ]:
```
#using code from HW11-PyTorch-Basics-Student.ipynb

from sklearn.pipeline import Pipeline
from sklearn.compose import ColumnTransformer
from sklearn.impute import SimpleImputer

torch.manual_seed(0)
device = torch.device("cuda:0" if torch.cuda.is_available() else "cpu")

num_attributes = []
cat_attributes = []
num_attributes_int = X_train.select_dtypes(include=['int'])
for col in num_attributes_int:
    if len(list(num_attributes_int[col].unique())) > 3:
        num_attributes.append(col)
    else:
        cat_attributes.append(col)

num_attributes_float = X_train.select_dtypes(include=['float']).columns
for col in num_attributes_float:
    num_attributes.append(col)

class_labels = ["No Default","Default"]

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

cat_pipeline = Pipeline([
        ('selector', DataFrameSelector(cat_attributes)),
        ('imputer', SimpleImputer(strategy='most_frequent'))
    ])

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

X_train = data_prep_pipeline.fit_transform(X_under)
X_valid = data_prep_pipeline.transform(X_valid)
```

```
X_test = data_prep_pipeline.transform(X_test)
X_kaggle_test = data_prep_pipeline.transform(X_kaggle_test)


y_train = np.array(y_under)
y_valid = np.array(y_valid)
y_test = np.array(y_test)

# convert numpy arrays to tensors
X_train_tensor = torch.from_numpy(X_train)
X_valid_tensor = torch.from_numpy(X_valid)
X_test_tensor = torch.from_numpy(X_test)
X_kaggle_test_tensor = torch.from_numpy(X_kaggle_test)
y_train_tensor = torch.from_numpy(y_train)
y_valid_tensor = torch.from_numpy(y_valid)
y_test_tensor = torch.from_numpy(y_test)

# create TensorDataset in PyTorch
hcdr_train = torch.utils.data.TensorDataset(X_train_tensor, y_train_tensor)
hcdr_valid = torch.utils.data.TensorDataset(X_valid_tensor, y_valid_tensor)
hcdr_test = torch.utils.data.TensorDataset(X_test_tensor, y_test_tensor)

# create dataloader
# DataLoader is implemented in PyTorch, which will return an iterator to iterate tr
train_batch_size = 96
valid_test_batch_size = 64
trainloader_hcdr = torch.utils.data.DataLoader(hcdr_train, batch_size=train_batch_s
validloader_hcdr = torch.utils.data.DataLoader(hcdr_valid, batch_size=valid_test_ba
testloader_hcdr = torch.utils.data.DataLoader(hcdr_test, batch_size=valid_test_bat
```

In [ ]:

```
import matplotlib.pyplot as plt

def run_hcdr_model(layer_type,
    hidden_layer_neurons=[32, 16, 8],
    opt=optim.SGD,
    epochs=5,
    learning_rate=1e-3
):

    D_in = X_test.shape[1]  # Input layer neurons depend on the input dataset shape
    D_out = 1  # Output layer neurons - depend on what you're trying to predict, he

    str_neurons = [str(h) for h in hidden_layer_neurons]
    arch_string = f"{D_in}-{'-'.join(str_neurons)}-{D_out}"

    layers = [
        torch.nn.Linear(D_in, hidden_layer_neurons[0]),
        layer_type,
    ]

    # Add hidden layers
    for i in range(1, len(hidden_layer_neurons)):
        prev, curr = hidden_layer_neurons[i - 1], hidden_layer_neurons[i]
        layers.append(torch.nn.Linear(prev, curr))
        layers.append(layer_type)


    # Add final layer
    layers.append(nn.Linear(hidden_layer_neurons[-1], D_out))
    layers.append(nn.Sigmoid()) #outputs 0 or 1

    # Use the nn package to define our model and loss function.
    # use the sequential API makes things simple
```

```
    model = torch.nn.Sequential(*layers)

    model.to(device)

    # use Cross Entropy and SGD optimizer.
    loss_fn = nn.CrossEntropyLoss()  #for classfication -- use binary cross entropy
    optimizer = opt(model.parameters(), lr=learning_rate)

    #summary(model, (4, 20))
    print('-'*50)
    print('Model:')
    print(model)
    print('-'*50)

    '''
    Training Process:
        Load a batch of data.
        Zero the grad.
        Predict the batch of the data through net i.e forward pass.
        Calculate the loss value by predict value and true value.
        Backprop i.e get the gradient with respect to parameters
        Update optimizer i.e gradient update
    '''

    loss_history = []
    acc_history = []
    def train_epoch(epoch, model, loss_fn, opt, train_loader):

        running_loss = 0.0
        count = 0
        y_pred = []
        epoch_target = []
        # dataset API gives us pythonic batching
        for batch_id, data in enumerate(train_loader):
            inputs, target = data[0].to(device), data[1].to(device)
            # 1:zero the grad, 2:forward pass, 3:calculate loss,  and 4:backprop!
            opt.zero_grad()
            preds = model(inputs.float()) #prediction over the input data
            target = target.float()

            # compute loss and gradients
            # loss = loss_fn(preds, target)    #mean loss for this batch
            loss = F.binary_cross_entropy(preds, target)

            loss.backward() #calculate nabla_w
            loss_history.append(loss.item())

            opt.step()  #update W
            # y_pred.extend(torch.argmax(preds, dim=1).tolist())
            y_pred.extend(torch.round(preds).tolist())

            epoch_target.extend(target.tolist())
            #from IPython.core.debugger import Pdb as pdb;    pdb().set_trace() #br

            running_loss += loss.item()
            count += 1

        loss = np.round(running_loss/count, 3)

        #accuracy

        # correct = (y_pred.detach().numpy() == np.array(epoch_target))
        # accuracy = correct.sum() / correct.size
        # accuracy = np.round(accuracy, 3)
```

```
        f1 = f1_score(epoch_target, y_pred)

        y_pred = np.array(y_pred)
        train_predicted_positive.append((y_pred==1).sum())

        return loss, f1



    #from IPython.core.debugger import Pdb as pdb;    pdb().set_trace() #breakpoint
    def evaluate_model(epoch, model, loss_fn, opt, data_loader, tag = "Test"):
        overall_loss = 0.0
        count = 0
        y_pred = []
        epoch_target = []
        for i,data in enumerate(data_loader):
            inputs, target = data[0].to(device), data[1].to(device)
            preds = model(inputs.float())
            target = target.float()


            # loss = loss_fn(preds, target)           # compute loss value
            loss = F.binary_cross_entropy(preds, target)

            overall_loss += (loss.item())  # compute total loss to save to logs
            # y_pred.extend(torch.argmax(preds, dim=1).tolist())
            y_pred.extend(torch.round(preds).tolist())

            epoch_target.extend(target.tolist())
            count += 1

        # compute mean loss
        loss = np.round(overall_loss/count, 3)
        #accuracy
        # correct = (y_pred.detach().numpy() == np.array(epoch_target))
        # accuracy = correct.sum() / correct.size
        # accuracy = np.round(accuracy, 3)
        f1 = f1_score(epoch_target, y_pred)

        y_pred = np.array(y_pred)
        if tag=="Test":
            test_predicted_positive.append((y_pred==1).sum())
        else:
            valid_predicted_positive.append((y_pred==1).sum())

        return loss, f1

    def prediction(model, data_loader):
        y_pred = []
        with torch.no_grad():
            for i,data in enumerate(data_loader):
                inputs = data
                preds = model(inputs.float())
                y_pred.extend(preds.tolist())

        return y_pred


    epoch_loss = []
    for epoch in range(epochs):
        # print(f"Epoch {epoch+1}")
        train_loss, train_f1, = train_epoch(epoch, model, loss_fn, optimizer, trai
        valid_loss, valid_f1 = evaluate_model(epoch, model, loss_fn, optimizer, va
        epoch_loss.append([train_loss,valid_loss])
```

```
        print(f"Epoch {epoch+1}: Train F1: {train_f1}\t Validation F1: {valid_f1}"
    print("-"*50)
    test_loss, test_f1 = evaluate_model(epoch, model, loss_fn, opt, testloader_hcdr
    predictions = prediction(model, X_kaggle_test_tensor)

    return arch_string, train_f1, valid_f1, test_f1, epoch_loss, predictions
```

In [ ]:

```
del hcdrLog
torch.manual_seed(0)

try: hcdrLog
except : hcdrLog = pd.DataFrame(
columns=[
"Architecture string",
"Optimizer",
"Epochs",
"Train F1",
"Valid F1",
"Test F1",
]
)

'''
(hidden_layers_neurons) - A list of the number of neurons in the hidden layers in c
(opt) - The optimizer function to use: SGD, Adam, etc.,  DEFAULT: optim.SGD
(epochs) - The total number of epochs to train your model for,  DEFAULT: 5
(learning_rate) - The learning rate to take the gradient descent step with
'''

optimizers = [optim.Adagrad, optim.Adam]
layer_types = [nn.ReLU(), nn.Sigmoid()]

hidden_layer_neurons = [16,8]
# opt = optim.SGD   # optim.SGD, Optim.Adam, etc.
epochs = 40
learning_rate = 1e-2

for layer_type in layer_types:
    for opt in optimizers:
        hidden_layer_neurons = [16,8]
        for i in range(4):
            hidden_layer_neurons.insert(0,32*2**i)


            arch_string, train_f1, valid_f1, test_f1, epoch_loss, predictions = ru
                layer_type,
                hidden_layer_neurons,
                opt,
                epochs,
                learning_rate
            )



            hcdrLog.loc[len(hcdrLog)] = [
                arch_string,
                f"{opt}, {layer_type}",
                f"{epochs}",
                f"{train_f1}",
                f"{valid_f1}",
                f"{test_f1}",
            ]
```

```
--------------------------------------------------
Model:
Sequential(
  (0): Linear(in_features=330, out_features=32, bias=True)
  (1): ReLU()
  (2): Linear(in_features=32, out_features=16, bias=True)
  (3): ReLU()
  (4): Linear(in_features=16, out_features=8, bias=True)
  (5): ReLU()
  (6): Linear(in_features=8, out_features=1, bias=True)
  (7): Sigmoid()
)
--------------------------------------------------
Epoch 1: Train F1: 0.6737726967047747    Validation F1: 0.2442726231386025
Epoch 2: Train F1: 0.6848476696482616    Validation F1: 0.254985754985755
Epoch 3: Train F1: 0.6861155398758438    Validation F1: 0.24655802191626858
Epoch 4: Train F1: 0.6892253497273312    Validation F1: 0.24617972802467403
Epoch 5: Train F1: 0.689034536625123     Validation F1: 0.2476108971616032
Epoch 6: Train F1: 0.6916624214637459    Validation F1: 0.25339710012905187
Epoch 7: Train F1: 0.6929872247893449    Validation F1: 0.2535863717872086
Epoch 8: Train F1: 0.6927984170845699    Validation F1: 0.2473669228579562
Epoch 9: Train F1: 0.6951402123604683    Validation F1: 0.2521020909293846
Epoch 10: Train F1: 0.6947418672850031   Validation F1: 0.24817931132008764
Epoch 11: Train F1: 0.697250204192758    Validation F1: 0.2503836877877659
Epoch 12: Train F1: 0.6978642361583458   Validation F1: 0.2514585764294049
Epoch 13: Train F1: 0.7000713872930618   Validation F1: 0.2542334949645655
Epoch 14: Train F1: 0.699081944916695    Validation F1: 0.24811046511627904
Epoch 15: Train F1: 0.6995985575287473   Validation F1: 0.24953867991483322
Epoch 16: Train F1: 0.702284375954652    Validation F1: 0.25283183270403714
Epoch 17: Train F1: 0.7032765755376161   Validation F1: 0.25703125000000004
Epoch 18: Train F1: 0.7040097949188858   Validation F1: 0.253905357006782
Epoch 19: Train F1: 0.704079899446275    Validation F1: 0.2518593400245505
Epoch 20: Train F1: 0.7051925162689805   Validation F1: 0.2499287140005703
Epoch 21: Train F1: 0.7060699623384113   Validation F1: 0.25092883681051725
Epoch 22: Train F1: 0.7075295792792488   Validation F1: 0.25390859891761874
Epoch 23: Train F1: 0.7081440292940937   Validation F1: 0.2467014843320506
Epoch 24: Train F1: 0.7105494952232536   Validation F1: 0.25012570935995976
Epoch 25: Train F1: 0.7089491525423728   Validation F1: 0.2519345238095238
Epoch 26: Train F1: 0.7109756097560976   Validation F1: 0.24992887624466573
Epoch 27: Train F1: 0.7120822622107968   Validation F1: 0.24445964432284542
Epoch 28: Train F1: 0.712213636979802    Validation F1: 0.24969151484357988
Epoch 29: Train F1: 0.7128451543042772   Validation F1: 0.24943998843847098
Epoch 30: Train F1: 0.7127013735797862   Validation F1: 0.24680073126142593
Epoch 31: Train F1: 0.714561659154834    Validation F1: 0.25358630263766774
Epoch 32: Train F1: 0.7144215746212893   Validation F1: 0.2438560760353021
Epoch 33: Train F1: 0.7166322349036476   Validation F1: 0.25012746740476366
Epoch 34: Train F1: 0.7162973770825557   Validation F1: 0.2511993022241605
Epoch 35: Train F1: 0.7167052101754864   Validation F1: 0.24574722339378602
Epoch 36: Train F1: 0.7178703609385081   Validation F1: 0.24711415134672937
Epoch 37: Train F1: 0.7183742591024556   Validation F1: 0.2520441077188508
Epoch 38: Train F1: 0.7194659075504949   Validation F1: 0.25526544135047724
Epoch 39: Train F1: 0.7204779199619836   Validation F1: 0.24927665257066547
Epoch 40: Train F1: 0.719434725005945    Validation F1: 0.24847781965787183
--------------------------------------------------
--------------------------------------------------
Model:
Sequential(
  (0): Linear(in_features=330, out_features=64, bias=True)
  (1): ReLU()
  (2): Linear(in_features=64, out_features=32, bias=True)
  (3): ReLU()
  (4): Linear(in_features=32, out_features=16, bias=True)
  (5): ReLU()
  (6): Linear(in_features=16, out_features=8, bias=True)
```

```
  (7): ReLU()
  (8): Linear(in_features=8, out_features=1, bias=True)
  (9): Sigmoid()
)
---------------------------------------------------
Epoch 1: Train F1: 0.6781804266080995    Validation F1: 0.2407268256028417
Epoch 2: Train F1: 0.6902467295909919    Validation F1: 0.2530344776230546
Epoch 3: Train F1: 0.6926230874362478    Validation F1: 0.24357923497267758
Epoch 4: Train F1: 0.6954434852171595    Validation F1: 0.24903777619387027
Epoch 5: Train F1: 0.6988194258116448    Validation F1: 0.24497358128045016
Epoch 6: Train F1: 0.7005433688871       Validation F1: 0.24747439559673934
Epoch 7: Train F1: 0.703803071969951     Validation F1: 0.2555823777911889
Epoch 8: Train F1: 0.7031265775922997    Validation F1: 0.2550203835120037
Epoch 9: Train F1: 0.7042826625072055    Validation F1: 0.2551662174303684
Epoch 10: Train F1: 0.7062690613351406   Validation F1: 0.24566178972439603
Epoch 11: Train F1: 0.7100743745774173   Validation F1: 0.2582545341807472
Epoch 12: Train F1: 0.7112738119449529   Validation F1: 0.2535971223021583
Epoch 13: Train F1: 0.7139271419849399   Validation F1: 0.25543687082148625
Epoch 14: Train F1: 0.7149207641928806   Validation F1: 0.2521883489284636
Epoch 15: Train F1: 0.7197858062766894   Validation F1: 0.25185294989623486
Epoch 16: Train F1: 0.7191171962807806   Validation F1: 0.24778137765882519
Epoch 17: Train F1: 0.7225990233315248   Validation F1: 0.2579207920792079
Epoch 18: Train F1: 0.7227314546631682   Validation F1: 0.2457098830286475
Epoch 19: Train F1: 0.7250076585316042   Validation F1: 0.2565096094234346
Epoch 20: Train F1: 0.7283602378929481   Validation F1: 0.25301204819277107
Epoch 21: Train F1: 0.7284560602446588   Validation F1: 0.2533005146565227
Epoch 22: Train F1: 0.731394674939488    Validation F1: 0.24329331046312175
Epoch 23: Train F1: 0.7351840511669048   Validation F1: 0.2542091534266066
Epoch 24: Train F1: 0.7360480595282795   Validation F1: 0.24926285508809784
Epoch 25: Train F1: 0.7376229201401885   Validation F1: 0.25403942790877465
Epoch 26: Train F1: 0.7393902147563932   Validation F1: 0.2395569836603541
Epoch 27: Train F1: 0.7425280305353917   Validation F1: 0.2465831031796615
Epoch 28: Train F1: 0.7452785163973547   Validation F1: 0.24436063473991315
Epoch 29: Train F1: 0.7485778519603501   Validation F1: 0.24588902106220697
Epoch 30: Train F1: 0.748458314878539    Validation F1: 0.24180100055586437
Epoch 31: Train F1: 0.7498466780238502   Validation F1: 0.24543899657924745
Epoch 32: Train F1: 0.7547850963830802   Validation F1: 0.2423585519184912
Epoch 33: Train F1: 0.7535905570907105   Validation F1: 0.2441211052322164
Epoch 34: Train F1: 0.7566885927541666   Validation F1: 0.24243295707815085
Epoch 35: Train F1: 0.7602040816326532   Validation F1: 0.2493220478264442
Epoch 36: Train F1: 0.7588625915205176   Validation F1: 0.24632237871674492
Epoch 37: Train F1: 0.7643807574206757   Validation F1: 0.23394898856640287
Epoch 38: Train F1: 0.7648499693815065   Validation F1: 0.23812772925764192
Epoch 39: Train F1: 0.7660733126850685   Validation F1: 0.24436860068259383
Epoch 40: Train F1: 0.7677993251763745   Validation F1: 0.2394981674654638
---------------------------------------------------
---------------------------------------------------
Model:
Sequential(
  (0): Linear(in_features=330, out_features=128, bias=True)
  (1): ReLU()
  (2): Linear(in_features=128, out_features=64, bias=True)
  (3): ReLU()
  (4): Linear(in_features=64, out_features=32, bias=True)
  (5): ReLU()
  (6): Linear(in_features=32, out_features=16, bias=True)
  (7): ReLU()
  (8): Linear(in_features=16, out_features=8, bias=True)
  (9): ReLU()
  (10): Linear(in_features=8, out_features=1, bias=True)
  (11): Sigmoid()
)
---------------------------------------------------
Epoch 1: Train F1: 0.6717039425078917    Validation F1: 0.25055687840847995
```

```
Epoch 2: Train F1: 0.6835348711174338    Validation F1: 0.2347067745197169
Epoch 3: Train F1: 0.6924109239750386    Validation F1: 0.2580140734949179
Epoch 4: Train F1: 0.6956521739130435    Validation F1: 0.24556712637207992
Epoch 5: Train F1: 0.6993391701049948    Validation F1: 0.2506636057105962
Epoch 6: Train F1: 0.6996895667431502    Validation F1: 0.25446394063538685
Epoch 7: Train F1: 0.7023117480059483    Validation F1: 0.2444
Epoch 8: Train F1: 0.7044840626688277    Validation F1: 0.24213878200875683
Epoch 9: Train F1: 0.7072070544344582    Validation F1: 0.24940914778256637
Epoch 10: Train F1: 0.7108294930875576   Validation F1: 0.2534814814814815
Epoch 11: Train F1: 0.710284820882333    Validation F1: 0.2576443491382734
Epoch 12: Train F1: 0.7146168530133624   Validation F1: 0.26049586776859507
Epoch 13: Train F1: 0.7171023406799931   Validation F1: 0.250892922224652
Epoch 14: Train F1: 0.7219455624531005   Validation F1: 0.2553382782647786
Epoch 15: Train F1: 0.7258713593888549   Validation F1: 0.24545454545454545
Epoch 16: Train F1: 0.727608242490517    Validation F1: 0.24296657853704834
Epoch 17: Train F1: 0.7313463514902363   Validation F1: 0.25987841945288753
Epoch 18: Train F1: 0.7352589300974193   Validation F1: 0.24953736654804268
Epoch 19: Train F1: 0.7365927350133901   Validation F1: 0.24442793462109957
Epoch 20: Train F1: 0.7397581228680702   Validation F1: 0.24415473997449344
Epoch 21: Train F1: 0.7454526699446412   Validation F1: 0.24323957059383072
Epoch 22: Train F1: 0.7490209549982823   Validation F1: 0.24133050247699933
Epoch 23: Train F1: 0.751607772629832    Validation F1: 0.24266796089579146
Epoch 24: Train F1: 0.7573613172701853   Validation F1: 0.24018258709807347
Epoch 25: Train F1: 0.7594403178441872   Validation F1: 0.23863557257222415
Epoch 26: Train F1: 0.7650983510215369   Validation F1: 0.24593301435406698
Epoch 27: Train F1: 0.7685692489076912   Validation F1: 0.24048064918851436
Epoch 28: Train F1: 0.772165949771056    Validation F1: 0.24156412772703253
Epoch 29: Train F1: 0.7741644396626522   Validation F1: 0.23635272945410918
Epoch 30: Train F1: 0.7770063871146904   Validation F1: 0.23491664922926694
Epoch 31: Train F1: 0.7828328572423884   Validation F1: 0.23588709677419356
Epoch 32: Train F1: 0.7860307713777344   Validation F1: 0.24188701923076925
Epoch 33: Train F1: 0.7884796648629778   Validation F1: 0.2322570702011251
Epoch 34: Train F1: 0.7934600335382895   Validation F1: 0.23005356639413585
Epoch 35: Train F1: 0.7972595078299777   Validation F1: 0.2374633544206141
Epoch 36: Train F1: 0.8007848083526031   Validation F1: 0.23347547974413643
Epoch 37: Train F1: 0.8035814213766089   Validation F1: 0.2408702408702409
Epoch 38: Train F1: 0.8064210858364584   Validation F1: 0.23485355088939414
Epoch 39: Train F1: 0.8111290209545169   Validation F1: 0.23604874919820398
Epoch 40: Train F1: 0.8144951939942469   Validation F1: 0.2304607901137185
--------------------------------------------------
--------------------------------------------------
Model:
Sequential(
  (0): Linear(in_features=330, out_features=256, bias=True)
  (1): ReLU()
  (2): Linear(in_features=256, out_features=128, bias=True)
  (3): ReLU()
  (4): Linear(in_features=128, out_features=64, bias=True)
  (5): ReLU()
  (6): Linear(in_features=64, out_features=32, bias=True)
  (7): ReLU()
  (8): Linear(in_features=32, out_features=16, bias=True)
  (9): ReLU()
  (10): Linear(in_features=16, out_features=8, bias=True)
  (11): ReLU()
  (12): Linear(in_features=8, out_features=1, bias=True)
  (13): Sigmoid()
)
--------------------------------------------------
Epoch 1: Train F1: 0.671713788020903     Validation F1: 0.23598593799936082
Epoch 2: Train F1: 0.6880868349204236    Validation F1: 0.24015485248965424
Epoch 3: Train F1: 0.6912055833305372    Validation F1: 0.24514098001642484
Epoch 4: Train F1: 0.6956900596554729    Validation F1: 0.25062362435803376
Epoch 5: Train F1: 0.6994918049338673    Validation F1: 0.24356288436845308
```

```
Epoch 6: Train F1: 0.7038808785877915    Validation F1: 0.24772057792116706
Epoch 7: Train F1: 0.7055704697986578    Validation F1: 0.24943502824858757
Epoch 8: Train F1: 0.7114754098360656    Validation F1: 0.2550164719976041
Epoch 9: Train F1: 0.7182497331910352    Validation F1: 0.257900635039223
Epoch 10: Train F1: 0.722123537889198    Validation F1: 0.24848695652173913
Epoch 11: Train F1: 0.7295807583096711   Validation F1: 0.24755195491370202
Epoch 12: Train F1: 0.7362630034675913   Validation F1: 0.23433044111401008
Epoch 13: Train F1: 0.7455471585045111   Validation F1: 0.24950841162333406
Epoch 14: Train F1: 0.7530006317119393   Validation F1: 0.22316002700877785
Epoch 15: Train F1: 0.7621750032994589   Validation F1: 0.23131006901889958
Epoch 16: Train F1: 0.7748193605859646   Validation F1: 0.2435897435897436
Epoch 17: Train F1: 0.7801718440185063   Validation F1: 0.2453307234320202
Epoch 18: Train F1: 0.7890041630872927   Validation F1: 0.22211847096585935
Epoch 19: Train F1: 0.8008431314428746   Validation F1: 0.24341711549970077
Epoch 20: Train F1: 0.8086801896733403   Validation F1: 0.22411328204779088
Epoch 21: Train F1: 0.8167132752097384   Validation F1: 0.21922823460496446
Epoch 22: Train F1: 0.8289984855468493   Validation F1: 0.23194197750148016
Epoch 23: Train F1: 0.836885030255196    Validation F1: 0.20464700625558532
Epoch 24: Train F1: 0.8459107072303437   Validation F1: 0.21151960784313725
Epoch 25: Train F1: 0.853454521504314    Validation F1: 0.23242510913743789
Epoch 26: Train F1: 0.8604689401164128   Validation F1: 0.21365866253550217
Epoch 27: Train F1: 0.8670979606581064   Validation F1: 0.22121604139715395
Epoch 28: Train F1: 0.8741228792620656   Validation F1: 0.21556589147286823
Epoch 29: Train F1: 0.8818477938999112   Validation F1: 0.21825223311887063
Epoch 30: Train F1: 0.887528389453935    Validation F1: 0.21674140508221226
Epoch 31: Train F1: 0.8931872758972333   Validation F1: 0.21865433022654604
Epoch 32: Train F1: 0.9002571033027885   Validation F1: 0.1974573978901812
Epoch 33: Train F1: 0.9045955033955299   Validation F1: 0.2154873975098694
Epoch 34: Train F1: 0.9122552574329225   Validation F1: 0.22186815375423222
Epoch 35: Train F1: 0.9142725203681102   Validation F1: 0.20774500209702224
Epoch 36: Train F1: 0.9190224570673712   Validation F1: 0.21783332254095708
Epoch 37: Train F1: 0.9246247732145803   Validation F1: 0.2194001276324186
Epoch 38: Train F1: 0.9266618258630933   Validation F1: 0.21782458124249063
Epoch 39: Train F1: 0.930965036812044    Validation F1: 0.20193827754484844
Epoch 40: Train F1: 0.9338886685177838   Validation F1: 0.2212722793798
--------------------------------------------------
--------------------------------------------------
Model:
Sequential(
  (0): Linear(in_features=330, out_features=32, bias=True)
  (1): ReLU()
  (2): Linear(in_features=32, out_features=16, bias=True)
  (3): ReLU()
  (4): Linear(in_features=16, out_features=8, bias=True)
  (5): ReLU()
  (6): Linear(in_features=8, out_features=1, bias=True)
  (7): Sigmoid()
)
--------------------------------------------------
Epoch 1: Train F1: 0.6752672318466642    Validation F1: 0.24515162860351927
Epoch 2: Train F1: 0.6818151078650166    Validation F1: 0.2580540715557714
Epoch 3: Train F1: 0.6830004804062864    Validation F1: 0.25875165125495375
Epoch 4: Train F1: 0.6871253895948214    Validation F1: 0.2467782128871485
Epoch 5: Train F1: 0.6935322052036653    Validation F1: 0.2610223374884966
Epoch 6: Train F1: 0.6894473738272957    Validation F1: 0.2576292964985544
Epoch 7: Train F1: 0.693037542662116     Validation F1: 0.24767977329082538
Epoch 8: Train F1: 0.69666133478542      Validation F1: 0.25206580808456786
Epoch 9: Train F1: 0.6976362330243219    Validation F1: 0.2497729336966394
Epoch 10: Train F1: 0.701890914055567    Validation F1: 0.2395806368107688
Epoch 11: Train F1: 0.7048773016575337   Validation F1: 0.242396068534998
Epoch 12: Train F1: 0.7072178922399187   Validation F1: 0.2308854638863705
Epoch 13: Train F1: 0.7165904004809781   Validation F1: 0.23739509055341704
Epoch 14: Train F1: 0.7105835260015874   Validation F1: 0.2539428388439677
Epoch 15: Train F1: 0.7116906351063098   Validation F1: 0.24906640688423445
```

```
Epoch 16: Train F1: 0.7133104619329799   Validation F1: 0.23687150837988827
Epoch 17: Train F1: 0.7184644131136403   Validation F1: 0.24849047840222943
Epoch 18: Train F1: 0.7167416396415245   Validation F1: 0.2517057131113851
Epoch 19: Train F1: 0.7169598078688987   Validation F1: 0.24708410067526088
Epoch 20: Train F1: 0.7188295165394403   Validation F1: 0.24769230769230768
Epoch 21: Train F1: 0.7197009398036543   Validation F1: 0.2436010591350397
Epoch 22: Train F1: 0.7271336161925298   Validation F1: 0.24258549182252134
Epoch 23: Train F1: 0.7322449258993051   Validation F1: 0.24421633215872526
Epoch 24: Train F1: 0.7327366609294319   Validation F1: 0.24775162274184717
Epoch 25: Train F1: 0.7271560940841055   Validation F1: 0.2367900106593574
Epoch 26: Train F1: 0.7282025959367945   Validation F1: 0.24156447864069885
Epoch 27: Train F1: 0.730479840999432    Validation F1: 0.23394280356305674
Epoch 28: Train F1: 0.7349389030974708   Validation F1: 0.23968474733426054
Epoch 29: Train F1: 0.7358316148759155   Validation F1: 0.24438659290595507
Epoch 30: Train F1: 0.7385014187129996   Validation F1: 0.21990701842827534
Epoch 31: Train F1: 0.7434764309764309   Validation F1: 0.22723983210305396
Epoch 32: Train F1: 0.738984742620847    Validation F1: 0.2296211251435132
Epoch 33: Train F1: 0.7426349953634354   Validation F1: 0.23005001087192867
Epoch 34: Train F1: 0.7414809844980401   Validation F1: 0.23681761786600497
Epoch 35: Train F1: 0.7443133352288881   Validation F1: 0.23815544812197842
Epoch 36: Train F1: 0.7468103203855968   Validation F1: 0.23127085228996058
Epoch 37: Train F1: 0.746933276607814    Validation F1: 0.23402824478816409
Epoch 38: Train F1: 0.7488655700510493   Validation F1: 0.23893391521197008
Epoch 39: Train F1: 0.7500440373436673   Validation F1: 0.2268312276345578
Epoch 40: Train F1: 0.7505960217770344   Validation F1: 0.23693273130787482
--------------------------------------------------
--------------------------------------------------
Model:
Sequential(
  (0): Linear(in_features=330, out_features=64, bias=True)
  (1): ReLU()
  (2): Linear(in_features=64, out_features=32, bias=True)
  (3): ReLU()
  (4): Linear(in_features=32, out_features=16, bias=True)
  (5): ReLU()
  (6): Linear(in_features=16, out_features=8, bias=True)
  (7): ReLU()
  (8): Linear(in_features=8, out_features=1, bias=True)
  (9): Sigmoid()
)
--------------------------------------------------
Epoch 1: Train F1: 0.6725954762064569    Validation F1: 0.24222057934000943
Epoch 2: Train F1: 0.6865522412931179    Validation F1: 0.24775268581451435
Epoch 3: Train F1: 0.6890186602228489    Validation F1: 0.24693399476367645
Epoch 4: Train F1: 0.6933875754443295    Validation F1: 0.24635888980489146
Epoch 5: Train F1: 0.6958055925432757    Validation F1: 0.24726003490401396
Epoch 6: Train F1: 0.70200477960701      Validation F1: 0.23669010547463584
Epoch 7: Train F1: 0.7054812657222296    Validation F1: 0.243403123961449
Epoch 8: Train F1: 0.7080335335665722    Validation F1: 0.2486237892829768
Epoch 9: Train F1: 0.7093119009190026    Validation F1: 0.23959548813691173
Epoch 10: Train F1: 0.7129498999311      Validation F1: 0.238432158286895
Epoch 11: Train F1: 0.7083615666079414   Validation F1: 0.23675252103539085
Epoch 12: Train F1: 0.7131871453037375   Validation F1: 0.2508661417322835
Epoch 13: Train F1: 0.7167948416563119   Validation F1: 0.23896207056331503
Epoch 14: Train F1: 0.7181380852361271   Validation F1: 0.23560547379292537
Epoch 15: Train F1: 0.7245076881575397   Validation F1: 0.24817843866171002
Epoch 16: Train F1: 0.7220438554546075   Validation F1: 0.2528827674567585
Epoch 17: Train F1: 0.7178028307519405   Validation F1: 0.2596014492753624
Epoch 18: Train F1: 0.7179468772433596   Validation F1: 0.25649556400506973
Epoch 19: Train F1: 0.7259475218658892   Validation F1: 0.2553699284009547
Epoch 20: Train F1: 0.7306331104994102   Validation F1: 0.25384225637561225
Epoch 21: Train F1: 0.7300739671657946   Validation F1: 0.2464499107627842
Epoch 22: Train F1: 0.7333094041636755   Validation F1: 0.24646693046919166
Epoch 23: Train F1: 0.7331815511163335   Validation F1: 0.25352818371607516
```

```
Epoch 24: Train F1: 0.7347948036610571   Validation F1: 0.2450174896282437
Epoch 25: Train F1: 0.7362090726854258   Validation F1: 0.24342383489807998
Epoch 26: Train F1: 0.7400848821893751   Validation F1: 0.24434893346068132
Epoch 27: Train F1: 0.7427083333333332   Validation F1: 0.2516902551158388
Epoch 28: Train F1: 0.7397219426739723   Validation F1: 0.24056647721978944
Epoch 29: Train F1: 0.7466706346253998   Validation F1: 0.24469227591904172
Epoch 30: Train F1: 0.7546864662206866   Validation F1: 0.23584684049800325
Epoch 31: Train F1: 0.7523222114884447   Validation F1: 0.242239512312505
Epoch 32: Train F1: 0.7575903972648558   Validation F1: 0.23664122137404578
Epoch 33: Train F1: 0.7578892720732748   Validation F1: 0.24138847073408304
Epoch 34: Train F1: 0.762290513173052    Validation F1: 0.2389580028387743
Epoch 35: Train F1: 0.7663767683177186   Validation F1: 0.23964106624439166
Epoch 36: Train F1: 0.7644877317849785   Validation F1: 0.24096181525696383
Epoch 37: Train F1: 0.7715383754337525   Validation F1: 0.23878055011496208
Epoch 38: Train F1: 0.7731574632983083   Validation F1: 0.24529592884023263
Epoch 39: Train F1: 0.7785618778784728   Validation F1: 0.24099926524614254
Epoch 40: Train F1: 0.7780175621372227   Validation F1: 0.23788265306122447
--------------------------------------------------
--------------------------------------------------
Model:
Sequential(
  (0): Linear(in_features=330, out_features=128, bias=True)
  (1): ReLU()
  (2): Linear(in_features=128, out_features=64, bias=True)
  (3): ReLU()
  (4): Linear(in_features=64, out_features=32, bias=True)
  (5): ReLU()
  (6): Linear(in_features=32, out_features=16, bias=True)
  (7): ReLU()
  (8): Linear(in_features=16, out_features=8, bias=True)
  (9): ReLU()
  (10): Linear(in_features=8, out_features=1, bias=True)
  (11): Sigmoid()
)
--------------------------------------------------
Epoch 1: Train F1: 0.6760357486959699    Validation F1: 0.2527531767423951
Epoch 2: Train F1: 0.6868915855447393    Validation F1: 0.2364588656106285
Epoch 3: Train F1: 0.6903672023789635    Validation F1: 0.22331015299026424
Epoch 4: Train F1: 0.692374523825436     Validation F1: 0.24584389969005355
Epoch 5: Train F1: 0.6921031096563012    Validation F1: 0.2525668890054711
Epoch 6: Train F1: 0.692391899288451     Validation F1: 0.24536068560631621
Epoch 7: Train F1: 0.6987549036329525    Validation F1: 0.26138802766515623
Epoch 8: Train F1: 0.7055133144669099    Validation F1: 0.25218887974257276
Epoch 9: Train F1: 0.7058622807616187    Validation F1: 0.24100320031350006
Epoch 10: Train F1: 0.7159688974907474   Validation F1: 0.2386589033207499
Epoch 11: Train F1: 0.7184896900177523   Validation F1: 0.24812030075187969
Epoch 12: Train F1: 0.7261986536612172   Validation F1: 0.2533508948632525
Epoch 13: Train F1: 0.7262964630893408   Validation F1: 0.24745946784578388
Epoch 14: Train F1: 0.734107855037281    Validation F1: 0.2407882867254181
Epoch 15: Train F1: 0.7371504289002022   Validation F1: 0.23916554508748322
Epoch 16: Train F1: 0.7418362352283058   Validation F1: 0.23778740419860048
Epoch 17: Train F1: 0.7512415349887134   Validation F1: 0.23709505161979355
Epoch 18: Train F1: 0.7555137451656736   Validation F1: 0.24665455146208493
Epoch 19: Train F1: 0.7619381776078032   Validation F1: 0.23974901502991391
Epoch 20: Train F1: 0.7609157924581798   Validation F1: 0.24040017654847723
Epoch 21: Train F1: 0.7702854732105191   Validation F1: 0.23632916192643158
Epoch 22: Train F1: 0.7796646072374227   Validation F1: 0.23440776699029126
Epoch 23: Train F1: 0.7751475083528826   Validation F1: 0.2355086226544101
Epoch 24: Train F1: 0.7821261054817572   Validation F1: 0.23543654750035609
Epoch 25: Train F1: 0.7917708810370943   Validation F1: 0.23499591169255926
Epoch 26: Train F1: 0.7953396666069187   Validation F1: 0.23068245910885504
Epoch 27: Train F1: 0.805955405023991    Validation F1: 0.23068173200879655
Epoch 28: Train F1: 0.8149636838022707   Validation F1: 0.2262495505213952
Epoch 29: Train F1: 0.8202797699353831   Validation F1: 0.22712933753943218
```

```
Epoch 30: Train F1: 0.8246875110654722   Validation F1: 0.2295269401171931
Epoch 31: Train F1: 0.8311843866694851   Validation F1: 0.22821885221528393
Epoch 32: Train F1: 0.8409035195237261   Validation F1: 0.21798029556650242
Epoch 33: Train F1: 0.8423861521716232   Validation F1: 0.23168023686158404
Epoch 34: Train F1: 0.847971310034456    Validation F1: 0.22247143758323404
Epoch 35: Train F1: 0.8521109431574507   Validation F1: 0.22328384536234017
Epoch 36: Train F1: 0.8506637557660481   Validation F1: 0.22544137618832052
Epoch 37: Train F1: 0.8579271511525604   Validation F1: 0.224891949307743
Epoch 38: Train F1: 0.8595807437058884   Validation F1: 0.2160888589700438
Epoch 39: Train F1: 0.8636807590344634   Validation F1: 0.22370497919357155
Epoch 40: Train F1: 0.8652066086835027   Validation F1: 0.22243821160788 67
--------------------------------------------------
--------------------------------------------------
Model:
Sequential(
  (0): Linear(in_features=330, out_features=256, bias=True)
  (1): ReLU()
  (2): Linear(in_features=256, out_features=128, bias=True)
  (3): ReLU()
  (4): Linear(in_features=128, out_features=64, bias=True)
  (5): ReLU()
  (6): Linear(in_features=64, out_features=32, bias=True)
  (7): ReLU()
  (8): Linear(in_features=32, out_features=16, bias=True)
  (9): ReLU()
  (10): Linear(in_features=16, out_features=8, bias=True)
  (11): ReLU()
  (12): Linear(in_features=8, out_features=1, bias=True)
  (13): Sigmoid()
)
--------------------------------------------------
Epoch 1: Train F1: 0.6724812030075188    Validation F1: 0.23932916351639288
Epoch 2: Train F1: 0.6856819166305688    Validation F1: 0.23930165397742192
Epoch 3: Train F1: 0.6879225103420442    Validation F1: 0.245427893340779
Epoch 4: Train F1: 0.6881496182179018    Validation F1: 0.22186477342060015
Epoch 5: Train F1: 0.6921741409244276    Validation F1: 0.21090731230129617
Epoch 6: Train F1: 0.6871909503488451    Validation F1: 0.23197299785210188
Epoch 7: Train F1: 0.7005891452456966    Validation F1: 0.2485957079072447
Epoch 8: Train F1: 0.6881466564821888    Validation F1: 0.2526740967910839
Epoch 9: Train F1: 0.6927435694907106    Validation F1: 0.25590215960987694
Epoch 10: Train F1: 0.6907857941834451   Validation F1: 0.2546816479400749
Epoch 11: Train F1: 0.6959619952494063   Validation F1: 0.2573816155988858
Epoch 12: Train F1: 0.702665425330161    Validation F1: 0.2220574817518248
Epoch 13: Train F1: 0.6854691689008043   Validation F1: 0.2492597948082352
Epoch 14: Train F1: 0.7017045454545455   Validation F1: 0.24073948209066812
Epoch 15: Train F1: 0.6914513906067781   Validation F1: 0.24453967376278685
Epoch 16: Train F1: 0.7005224660397075   Validation F1: 0.2454452703687509
Epoch 17: Train F1: 0.6967416767232201   Validation F1: 0.2398568781547505
Epoch 18: Train F1: 0.7216536493401562   Validation F1: 0.23767527441152209
Epoch 19: Train F1: 0.7091244439622436   Validation F1: 0.26017320048421644
Epoch 20: Train F1: 0.7082223276522754   Validation F1: 0.2530417915711515
Epoch 21: Train F1: 0.7152637366535419   Validation F1: 0.24769230769230768
Epoch 22: Train F1: 0.710279038193576    Validation F1: 0.24839549002601907
Epoch 23: Train F1: 0.7119412259220002   Validation F1: 0.23720650210716437
Epoch 24: Train F1: 0.7156411730879815   Validation F1: 0.25478393675321176
Epoch 25: Train F1: 0.7180551437889119   Validation F1: 0.249533544252454
Epoch 26: Train F1: 0.706164332259756    Validation F1: 0.23740010655301014
Epoch 27: Train F1: 0.6919282347352651   Validation F1: 0.25021720243266726
Epoch 28: Train F1: 0.6782095578244497   Validation F1: 0.24689566199000162
Epoch 29: Train F1: 0.7081436519979766   Validation F1: 0.24934291221307167
Epoch 30: Train F1: 0.6990932785140046   Validation F1: 0.22536436806613008
Epoch 31: Train F1: 0.6116188252345102   Validation F1: 0.23699772554965884
Epoch 32: Train F1: 0.6965711925089507   Validation F1: 0.24598214285714287
Epoch 33: Train F1: 0.6953470521583391   Validation F1: 0.25170068027210885
```

```
Epoch 34: Train F1: 0.6767668999461663   Validation F1: 0.24694207246711286
Epoch 35: Train F1: 0.6772908366533864   Validation F1: 0.23220349843019156
Epoch 36: Train F1: 0.730755438680373    Validation F1: 0.2098956719347175
Epoch 37: Train F1: 0.6972516156522973   Validation F1: 0.23641963174103925
Epoch 38: Train F1: 0.6957759848125296   Validation F1: 0.24997853893038033
Epoch 39: Train F1: 0.703775521843276    Validation F1: 0.2416036574643925
Epoch 40: Train F1: 0.7205887960950311   Validation F1: 0.24448930157448523
--------------------------------------------------
--------------------------------------------------
Model:
Sequential(
  (0): Linear(in_features=330, out_features=32, bias=True)
  (1): Sigmoid()
  (2): Linear(in_features=32, out_features=16, bias=True)
  (3): Sigmoid()
  (4): Linear(in_features=16, out_features=8, bias=True)
  (5): Sigmoid()
  (6): Linear(in_features=8, out_features=1, bias=True)
  (7): Sigmoid()
)
--------------------------------------------------
Epoch 1: Train F1: 0.6650975117686617    Validation F1: 0.23561895996318452
Epoch 2: Train F1: 0.6871493630783447    Validation F1: 0.24072258787284692
Epoch 3: Train F1: 0.6883548001593414    Validation F1: 0.2429008908685969
Epoch 4: Train F1: 0.6889786112316061    Validation F1: 0.24031111707120828
Epoch 5: Train F1: 0.6901131168874504    Validation F1: 0.24096871846619577
Epoch 6: Train F1: 0.6915120779394102    Validation F1: 0.24258395837556593
Epoch 7: Train F1: 0.6909333778594089    Validation F1: 0.24590630784564999
Epoch 8: Train F1: 0.6914647632451996    Validation F1: 0.2447894810463002
Epoch 9: Train F1: 0.6929654619547064    Validation F1: 0.248651045901397
Epoch 10: Train F1: 0.6923722272559483   Validation F1: 0.24683179723502302
Epoch 11: Train F1: 0.6921008403361344   Validation F1: 0.24456711620431834
Epoch 12: Train F1: 0.6934828593492932   Validation F1: 0.24476740143244555
Epoch 13: Train F1: 0.6935500117516705   Validation F1: 0.24376080193570687
Epoch 14: Train F1: 0.6941749202618768   Validation F1: 0.24349480263604864
Epoch 15: Train F1: 0.6947333109694733   Validation F1: 0.24809718969555036
Epoch 16: Train F1: 0.6939708799892398   Validation F1: 0.24690825424216278
Epoch 17: Train F1: 0.6951666610608456   Validation F1: 0.24680008485962804
Epoch 18: Train F1: 0.6951543926995992   Validation F1: 0.2448187426353365
Epoch 19: Train F1: 0.6965371309575791   Validation F1: 0.24700513405590416
Epoch 20: Train F1: 0.6977056218972225   Validation F1: 0.2514952153110048
Epoch 21: Train F1: 0.6949403759347841   Validation F1: 0.24800000000000003
Epoch 22: Train F1: 0.6954822625745375   Validation F1: 0.24664500635683007
Epoch 23: Train F1: 0.6955965526528413   Validation F1: 0.24539877300613497
Epoch 24: Train F1: 0.6959420777908739   Validation F1: 0.24693658187029738
Epoch 25: Train F1: 0.6968188849283744   Validation F1: 0.24773413897280966
Epoch 26: Train F1: 0.6974753092661881   Validation F1: 0.24586618293448687
Epoch 27: Train F1: 0.6965150699677072   Validation F1: 0.24730029321318744
Epoch 28: Train F1: 0.6975445677766566   Validation F1: 0.24985388661601404
Epoch 29: Train F1: 0.697364435019691    Validation F1: 0.2507387706855792
Epoch 30: Train F1: 0.6966762274337777   Validation F1: 0.2476420112048791
Epoch 31: Train F1: 0.6974424638609024   Validation F1: 0.24579685980269556
Epoch 32: Train F1: 0.6971662913039094   Validation F1: 0.24924100043371403
Epoch 33: Train F1: 0.6973892538319016   Validation F1: 0.24798573975044563
Epoch 34: Train F1: 0.6974410774410775   Validation F1: 0.24815864022662892
Epoch 35: Train F1: 0.6981646741875736   Validation F1: 0.24739933479583892
Epoch 36: Train F1: 0.6977668496749638   Validation F1: 0.2486610012140256
Epoch 37: Train F1: 0.6988097639701433   Validation F1: 0.25244589386302996
Epoch 38: Train F1: 0.6970177986423047   Validation F1: 0.24803233354605403
Epoch 39: Train F1: 0.6980236355678261   Validation F1: 0.2500358834505526
Epoch 40: Train F1: 0.698269476802909    Validation F1: 0.24965567234505254
--------------------------------------------------
--------------------------------------------------
Model:
```

```
Sequential(
  (0): Linear(in_features=330, out_features=64, bias=True)
  (1): Sigmoid()
  (2): Linear(in_features=64, out_features=32, bias=True)
  (3): Sigmoid()
  (4): Linear(in_features=32, out_features=16, bias=True)
  (5): Sigmoid()
  (6): Linear(in_features=16, out_features=8, bias=True)
  (7): Sigmoid()
  (8): Linear(in_features=8, out_features=1, bias=True)
  (9): Sigmoid()
)
--------------------------------------------------
Epoch 1: Train F1: 0.6441645208434824    Validation F1: 0.23116154532704325
Epoch 2: Train F1: 0.6845872225337248    Validation F1: 0.23738044633368757
Epoch 3: Train F1: 0.6877614715452552    Validation F1: 0.24042290265000685
Epoch 4: Train F1: 0.6894733328883252    Validation F1: 0.24594536149354862
Epoch 5: Train F1: 0.6898513857276742    Validation F1: 0.24358074926336468
Epoch 6: Train F1: 0.6901937127153295    Validation F1: 0.24259618045945197
Epoch 7: Train F1: 0.6935483870967741    Validation F1: 0.24581242490635377
Epoch 8: Train F1: 0.6929598230740877    Validation F1: 0.24372959077329256
Epoch 9: Train F1: 0.6921143469955426    Validation F1: 0.24595375722543353
Epoch 10: Train F1: 0.6926124916051041   Validation F1: 0.24353268428372743
Epoch 11: Train F1: 0.6942935693690068   Validation F1: 0.2487967419474269
Epoch 12: Train F1: 0.6925198439391901   Validation F1: 0.2455293783707068
Epoch 13: Train F1: 0.6938267451640034   Validation F1: 0.24437968359700252
Epoch 14: Train F1: 0.6930320150659133   Validation F1: 0.24471806352315656
Epoch 15: Train F1: 0.6939942161544151   Validation F1: 0.2455375966268447
Epoch 16: Train F1: 0.6944762161071116   Validation F1: 0.2473725602345035
Epoch 17: Train F1: 0.6954504186421869   Validation F1: 0.24916168537687708
Epoch 18: Train F1: 0.695040710584752    Validation F1: 0.25193536264562194
Epoch 19: Train F1: 0.6950301967002935   Validation F1: 0.24459327473140785
Epoch 20: Train F1: 0.6954678510144343   Validation F1: 0.2472716829408386
Epoch 21: Train F1: 0.6947758678800134   Validation F1: 0.24291609353507562
Epoch 22: Train F1: 0.6960441334768569   Validation F1: 0.24519878109276452
Epoch 23: Train F1: 0.6963421194190424   Validation F1: 0.25208945109555003
Epoch 24: Train F1: 0.6950101146325016   Validation F1: 0.24553224553224554
Epoch 25: Train F1: 0.6959776121919148   Validation F1: 0.24213706908391705
Epoch 26: Train F1: 0.6977275783245933   Validation F1: 0.24992627543497495
Epoch 27: Train F1: 0.6955934948377083   Validation F1: 0.24884927303280485
Epoch 28: Train F1: 0.6973958683803243   Validation F1: 0.25495924957711824
Epoch 29: Train F1: 0.6963188112124282   Validation F1: 0.24737945492662472
Epoch 30: Train F1: 0.6958397950239363   Validation F1: 0.2487815523386921
Epoch 31: Train F1: 0.6966914247130317   Validation F1: 0.24474557522123894
Epoch 32: Train F1: 0.6975773889636608   Validation F1: 0.25102177305491563
Epoch 33: Train F1: 0.696986079199892    Validation F1: 0.24503127171646977
Epoch 34: Train F1: 0.6982021412699482   Validation F1: 0.24839930151338765
Epoch 35: Train F1: 0.6975221119438255   Validation F1: 0.24740184757505776
Epoch 36: Train F1: 0.697823519487093    Validation F1: 0.2453374621718629
Epoch 37: Train F1: 0.6981214798826346   Validation F1: 0.24565355106637576
Epoch 38: Train F1: 0.6985936393376276   Validation F1: 0.2467774276711544
Epoch 39: Train F1: 0.699673301673908    Validation F1: 0.25076589703355
Epoch 40: Train F1: 0.6976383265856949   Validation F1: 0.25033427425345417
--------------------------------------------------
--------------------------------------------------
Model:
Sequential(
  (0): Linear(in_features=330, out_features=128, bias=True)
  (1): Sigmoid()
  (2): Linear(in_features=128, out_features=64, bias=True)
  (3): Sigmoid()
  (4): Linear(in_features=64, out_features=32, bias=True)
  (5): Sigmoid()
  (6): Linear(in_features=32, out_features=16, bias=True)
```

```
  (7): Sigmoid()
  (8): Linear(in_features=16, out_features=8, bias=True)
  (9): Sigmoid()
  (10): Linear(in_features=8, out_features=1, bias=True)
  (11): Sigmoid()
)
--------------------------------------------------
Epoch 1: Train F1: 0.6149351834946138    Validation F1: 0.22672455378678247
Epoch 2: Train F1: 0.6819920844327177    Validation F1: 0.2335881887687846
Epoch 3: Train F1: 0.6827210522819371    Validation F1: 0.2385786802030457
Epoch 4: Train F1: 0.6856571714142928    Validation F1: 0.2401877133105802
Epoch 5: Train F1: 0.688355021688355     Validation F1: 0.23869676261475806
Epoch 6: Train F1: 0.6899839957321953    Validation F1: 0.23899717611405744
Epoch 7: Train F1: 0.6915184222799654    Validation F1: 0.24679511207736715
Epoch 8: Train F1: 0.6904841850782352    Validation F1: 0.24603584335127957
Epoch 9: Train F1: 0.6927565392354125    Validation F1: 0.2418115088982757
Epoch 10: Train F1: 0.6918922545369159   Validation F1: 0.2514347758647433
Epoch 11: Train F1: 0.6916133378241833   Validation F1: 0.2388197325956662
Epoch 12: Train F1: 0.6914757842412843   Validation F1: 0.24890277467827124
Epoch 13: Train F1: 0.6924683373753705   Validation F1: 0.2422500513241634
Epoch 14: Train F1: 0.6944734098018769   Validation F1: 0.24482372691662002
Epoch 15: Train F1: 0.694955209806695    Validation F1: 0.24386172358628366
Epoch 16: Train F1: 0.6961697548508592   Validation F1: 0.2406207827260459
Epoch 17: Train F1: 0.6949689776099272   Validation F1: 0.24555010511562717
Epoch 18: Train F1: 0.6972909305064782   Validation F1: 0.24146734748723458
Epoch 19: Train F1: 0.698390247187984    Validation F1: 0.24311801580812212
Epoch 20: Train F1: 0.6973298764268157   Validation F1: 0.2480720720720721
Epoch 21: Train F1: 0.699572980061195    Validation F1: 0.25081336882579114
Epoch 22: Train F1: 0.698631060759323    Validation F1: 0.2426880696503877
Epoch 23: Train F1: 0.7002423915970912   Validation F1: 0.2542087542087542
Epoch 24: Train F1: 0.6991436855235655   Validation F1: 0.24442437295907737
Epoch 25: Train F1: 0.7006536828627266   Validation F1: 0.2434591021757092
Epoch 26: Train F1: 0.7018725582648524   Validation F1: 0.2500550943950635
Epoch 27: Train F1: 0.7013065937405044   Validation F1: 0.23960944715661697
Epoch 28: Train F1: 0.702303961196443    Validation F1: 0.24929709465791938
Epoch 29: Train F1: 0.7017094305269901   Validation F1: 0.2510381000947039
Epoch 30: Train F1: 0.7032811234134485   Validation F1: 0.24873310810810811
Epoch 31: Train F1: 0.7022632981414646   Validation F1: 0.24457002791584395
Epoch 32: Train F1: 0.701889338731444    Validation F1: 0.2464275707481087
Epoch 33: Train F1: 0.7027154823020805   Validation F1: 0.24190552016985137
Epoch 34: Train F1: 0.7029449423815621   Validation F1: 0.2470803433234839
Epoch 35: Train F1: 0.7038374565459515   Validation F1: 0.24906313058518303
Epoch 36: Train F1: 0.7053025956053599   Validation F1: 0.24785658612626657
Epoch 37: Train F1: 0.7050281970756086   Validation F1: 0.2506724827335514
Epoch 38: Train F1: 0.7058068872383524   Validation F1: 0.25187161811578085
Epoch 39: Train F1: 0.7068913124219198   Validation F1: 0.2539876364191406
Epoch 40: Train F1: 0.7054565136497412   Validation F1: 0.24811020840692335
--------------------------------------------------
--------------------------------------------------
Model:
Sequential(
  (0): Linear(in_features=330, out_features=256, bias=True)
  (1): Sigmoid()
  (2): Linear(in_features=256, out_features=128, bias=True)
  (3): Sigmoid()
  (4): Linear(in_features=128, out_features=64, bias=True)
  (5): Sigmoid()
  (6): Linear(in_features=64, out_features=32, bias=True)
  (7): Sigmoid()
  (8): Linear(in_features=32, out_features=16, bias=True)
  (9): Sigmoid()
  (10): Linear(in_features=16, out_features=8, bias=True)
  (11): Sigmoid()
  (12): Linear(in_features=8, out_features=1, bias=True)
```

```
  (13): Sigmoid()
)
--------------------------------------------------
Epoch 1: Train F1: 0.6528617802836152    Validation F1: 0.23964414790102864
Epoch 2: Train F1: 0.6830117987313898    Validation F1: 0.2308078904384973
Epoch 3: Train F1: 0.690601491621735     Validation F1: 0.2361357222550371
Epoch 4: Train F1: 0.6858978605020865    Validation F1: 0.24807361412433604
Epoch 5: Train F1: 0.6820650351994636    Validation F1: 0.2543537306278958
Epoch 6: Train F1: 0.6838279830405815    Validation F1: 0.2470718232044199
Epoch 7: Train F1: 0.6811618821131307    Validation F1: 0.24638208130317793
Epoch 8: Train F1: 0.6833762362823466    Validation F1: 0.24395665462628507
Epoch 9: Train F1: 0.6825418424275412    Validation F1: 0.24212585148715182
Epoch 10: Train F1: 0.6839248150912669   Validation F1: 0.24423837447156688
Epoch 11: Train F1: 0.6853718332879325   Validation F1: 0.25232853513971215
Epoch 12: Train F1: 0.68498081428911     Validation F1: 0.2570447832681408
Epoch 13: Train F1: 0.6863332879138847   Validation F1: 0.2544164159530973
Epoch 14: Train F1: 0.6858096600585871   Validation F1: 0.24983369059058322
Epoch 15: Train F1: 0.6866862090122027   Validation F1: 0.24555061179087878
Epoch 16: Train F1: 0.6865935861646354   Validation F1: 0.2459409854581392
Epoch 17: Train F1: 0.6876744027768326   Validation F1: 0.2455020707447892
Epoch 18: Train F1: 0.6872294741147199   Validation F1: 0.2489208633093525
Epoch 19: Train F1: 0.6890088621778547   Validation F1: 0.25793147208121825
Epoch 20: Train F1: 0.689396777819408    Validation F1: 0.24899126989949377
Epoch 21: Train F1: 0.6882350935806089   Validation F1: 0.2493425261212595
Epoch 22: Train F1: 0.6919925512104284   Validation F1: 0.2444216990788127
Epoch 23: Train F1: 0.691044776119403    Validation F1: 0.24587561261820945
Epoch 24: Train F1: 0.6911584951785956   Validation F1: 0.25231274246493585
Epoch 25: Train F1: 0.6930679618245423   Validation F1: 0.24747081712062255
Epoch 26: Train F1: 0.6928931434563473   Validation F1: 0.24385259859682584
Epoch 27: Train F1: 0.6930115908628753   Validation F1: 0.2607274213322436
Epoch 28: Train F1: 0.692781690140845    Validation F1: 0.25012599899200805
Epoch 29: Train F1: 0.6951842701952688   Validation F1: 0.25990059323392656
Epoch 30: Train F1: 0.6934336247204717   Validation F1: 0.251155401502022
Epoch 31: Train F1: 0.6944500914820085   Validation F1: 0.25098324836125274
Epoch 32: Train F1: 0.6968185494742518   Validation F1: 0.244253859348199
Epoch 33: Train F1: 0.693594922100404    Validation F1: 0.24578612876485217
Epoch 34: Train F1: 0.695983286157164    Validation F1: 0.2568044552513923
Epoch 35: Train F1: 0.6947033108543157   Validation F1: 0.24511593321165415
Epoch 36: Train F1: 0.6985645933014354   Validation F1: 0.25640628628938605
Epoch 37: Train F1: 0.6959105798129199   Validation F1: 0.2518616825186168
Epoch 38: Train F1: 0.6969953592357983   Validation F1: 0.24753590537876657
Epoch 39: Train F1: 0.6972600890808477   Validation F1: 0.25386165211551376
Epoch 40: Train F1: 0.697633595516997    Validation F1: 0.24842857142857144
--------------------------------------------------
--------------------------------------------------
Model:
Sequential(
  (0): Linear(in_features=330, out_features=32, bias=True)
  (1): Sigmoid()
  (2): Linear(in_features=32, out_features=16, bias=True)
  (3): Sigmoid()
  (4): Linear(in_features=16, out_features=8, bias=True)
  (5): Sigmoid()
  (6): Linear(in_features=8, out_features=1, bias=True)
  (7): Sigmoid()
)
--------------------------------------------------
Epoch 1: Train F1: 0.6743352450469239    Validation F1: 0.23659201169668373
Epoch 2: Train F1: 0.6842298819832169    Validation F1: 0.2387785859380119
Epoch 3: Train F1: 0.6894817992894013    Validation F1: 0.24752111641571795
Epoch 4: Train F1: 0.693609022556391     Validation F1: 0.25760486711495356
Epoch 5: Train F1: 0.6964963552689039    Validation F1: 0.2542425657792309
Epoch 6: Train F1: 0.7016147905452842    Validation F1: 0.24178924037880314
Epoch 7: Train F1: 0.7054696114136790    Validation F1: 0.24442925495557075
```

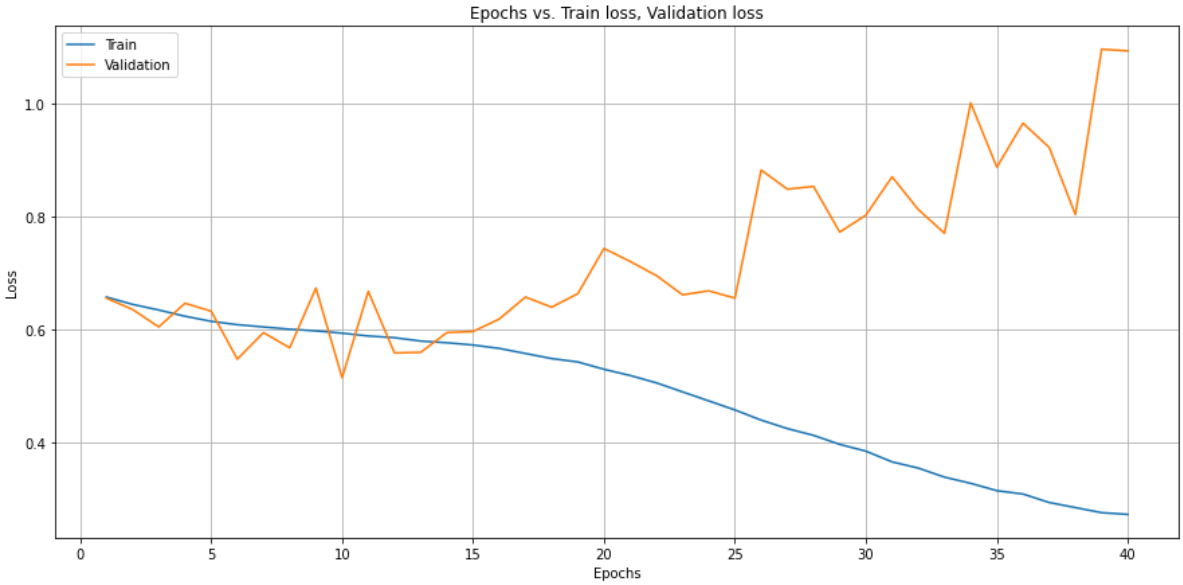
```
Epoch 8: Train F1: 0.7097204616718927    Validation F1: 0.2506981073533974
Epoch 9: Train F1: 0.7138053863311029    Validation F1: 0.245606453471622
Epoch 10: Train F1: 0.7174057797309895   Validation F1: 0.25299365104878246
Epoch 11: Train F1: 0.7215063096748348   Validation F1: 0.23992309276934698
Epoch 12: Train F1: 0.7253235977431132   Validation F1: 0.23239909629358246
Epoch 13: Train F1: 0.7292956720421206   Validation F1: 0.25185555833541823
Epoch 14: Train F1: 0.733526069185525    Validation F1: 0.23755145399017394
Epoch 15: Train F1: 0.7353692098544381   Validation F1: 0.22176754759874967
Epoch 16: Train F1: 0.7386609071274298   Validation F1: 0.2351012261191902
Epoch 17: Train F1: 0.7402670929496786   Validation F1: 0.2312960330792092
Epoch 18: Train F1: 0.7438510306369702   Validation F1: 0.2365901072791418
Epoch 19: Train F1: 0.749402072814244    Validation F1: 0.23410271585454068
Epoch 20: Train F1: 0.7510966369799282   Validation F1: 0.22910557184750732
Epoch 21: Train F1: 0.7539461467038068   Validation F1: 0.2342901474010861
Epoch 22: Train F1: 0.7562997347480106   Validation F1: 0.23305139882888745
Epoch 23: Train F1: 0.7591663065518731   Validation F1: 0.2243202416918429
Epoch 24: Train F1: 0.760936721474838    Validation F1: 0.23174109857442063
Epoch 25: Train F1: 0.7632782971951462   Validation F1: 0.23995440296380738
Epoch 26: Train F1: 0.7667175775283883   Validation F1: 0.23056852513045867
Epoch 27: Train F1: 0.7697455605618871   Validation F1: 0.22555918572505654
Epoch 28: Train F1: 0.7696441009652062   Validation F1: 0.2330770800734744
Epoch 29: Train F1: 0.7728568587151098   Validation F1: 0.2205488829423113
Epoch 30: Train F1: 0.7727618984532839   Validation F1: 0.22975062176514083
Epoch 31: Train F1: 0.7773923958643015   Validation F1: 0.22168580375782881
Epoch 32: Train F1: 0.7779733634461458   Validation F1: 0.22547702230583175
Epoch 33: Train F1: 0.7811227024341777   Validation F1: 0.21739647786768204
Epoch 34: Train F1: 0.7811590367822175   Validation F1: 0.21957878315132603
Epoch 35: Train F1: 0.7824851288830139   Validation F1: 0.23158528327439618
Epoch 36: Train F1: 0.7826605687015312   Validation F1: 0.21836653912125611
Epoch 37: Train F1: 0.7864521255462852   Validation F1: 0.22443701507601083
Epoch 38: Train F1: 0.788334216101695    Validation F1: 0.22037867130527863
Epoch 39: Train F1: 0.7898490628628296   Validation F1: 0.2247969103742176
Epoch 40: Train F1: 0.7900989443727457   Validation F1: 0.22311810190998613
--------------------------------------------------
--------------------------------------------------
Model:
Sequential(
  (0): Linear(in_features=330, out_features=64, bias=True)
  (1): Sigmoid()
  (2): Linear(in_features=64, out_features=32, bias=True)
  (3): Sigmoid()
  (4): Linear(in_features=32, out_features=16, bias=True)
  (5): Sigmoid()
  (6): Linear(in_features=16, out_features=8, bias=True)
  (7): Sigmoid()
  (8): Linear(in_features=8, out_features=1, bias=True)
  (9): Sigmoid()
)
--------------------------------------------------
Epoch 1: Train F1: 0.6476942812774289    Validation F1: 0.22491349480968859
Epoch 2: Train F1: 0.6792817403471366    Validation F1: 0.2728952772073922
Epoch 3: Train F1: 0.6814759546903616    Validation F1: 0.25579211020663745
Epoch 4: Train F1: 0.6877264262739122    Validation F1: 0.25572131531672265
Epoch 5: Train F1: 0.6924587056593555    Validation F1: 0.22816196542311193
Epoch 6: Train F1: 0.6950114911450588    Validation F1: 0.24264606208485834
Epoch 7: Train F1: 0.7020232573771593    Validation F1: 0.24931782277753842
Epoch 8: Train F1: 0.7086301922301383    Validation F1: 0.25
Epoch 9: Train F1: 0.7110254589714982    Validation F1: 0.25798190199760973
Epoch 10: Train F1: 0.7175593128817792   Validation F1: 0.24025201260063006
Epoch 11: Train F1: 0.7264928624086857   Validation F1: 0.23382343416024354
Epoch 12: Train F1: 0.730378422171961    Validation F1: 0.24028506118058354
Epoch 13: Train F1: 0.7363954238381369   Validation F1: 0.23639514731369152
Epoch 14: Train F1: 0.7401793266790728   Validation F1: 0.24441665177773805
Epoch 15: Train F1: 0.7467632289140164   Validation F1: 0.2352104664391354
```

```
Epoch 16: Train F1: 0.7578061363019276   Validation F1: 0.23938826466916358
Epoch 17: Train F1: 0.7632525622751647   Validation F1: 0.22309445429222585
Epoch 18: Train F1: 0.7697178064472205   Validation F1: 0.2253441180644246
Epoch 19: Train F1: 0.7748030421881927   Validation F1: 0.22268352519343268
Epoch 20: Train F1: 0.7805177529929398   Validation F1: 0.22298751614291862
Epoch 21: Train F1: 0.7852999385791306   Validation F1: 0.21626591230551628
Epoch 22: Train F1: 0.7922912205567452   Validation F1: 0.21837110391671427
Epoch 23: Train F1: 0.7973909777003724   Validation F1: 0.21855467378327129
Epoch 24: Train F1: 0.7996459333401423   Validation F1: 0.21750420039712848
Epoch 25: Train F1: 0.8070499026539605   Validation F1: 0.22315663012464332
Epoch 26: Train F1: 0.8090340656712339   Validation F1: 0.21495213861626075
Epoch 27: Train F1: 0.8152548738434223   Validation F1: 0.21970644375699333
Epoch 28: Train F1: 0.8214285714285715   Validation F1: 0.2146292710350428
Epoch 29: Train F1: 0.8231083844580778   Validation F1: 0.2171184576567251
Epoch 30: Train F1: 0.8301655007677872   Validation F1: 0.20960173264348456
Epoch 31: Train F1: 0.8294486086867235   Validation F1: 0.21117675018673188
Epoch 32: Train F1: 0.8318916886047385   Validation F1: 0.215064015254699
Epoch 33: Train F1: 0.8344483888585471   Validation F1: 0.20704875275864373
Epoch 34: Train F1: 0.8381212638770282   Validation F1: 0.2191007367623769
Epoch 35: Train F1: 0.8426759410801965   Validation F1: 0.217545266252422
Epoch 36: Train F1: 0.8449092209694451   Validation F1: 0.20649136633947124
Epoch 37: Train F1: 0.8474172004906637   Validation F1: 0.21318528799444827
Epoch 38: Train F1: 0.8487698493832209   Validation F1: 0.20977448581443026
Epoch 39: Train F1: 0.8490849494673586   Validation F1: 0.20494834842494578
Epoch 40: Train F1: 0.8548803697661772   Validation F1: 0.21449784017278617
--------------------------------------------------
--------------------------------------------------
Model:
Sequential(
  (0): Linear(in_features=330, out_features=128, bias=True)
  (1): Sigmoid()
  (2): Linear(in_features=128, out_features=64, bias=True)
  (3): Sigmoid()
  (4): Linear(in_features=64, out_features=32, bias=True)
  (5): Sigmoid()
  (6): Linear(in_features=32, out_features=16, bias=True)
  (7): Sigmoid()
  (8): Linear(in_features=16, out_features=8, bias=True)
  (9): Sigmoid()
  (10): Linear(in_features=8, out_features=1, bias=True)
  (11): Sigmoid()
)
--------------------------------------------------
Epoch 1: Train F1: 0.6556439003293522    Validation F1: 0.21439272614558041
Epoch 2: Train F1: 0.672787018530438     Validation F1: 0.23612167300380227
Epoch 3: Train F1: 0.6796227686089592    Validation F1: 0.2273219064973521
Epoch 4: Train F1: 0.6881706008583691    Validation F1: 0.23696215016644065
Epoch 5: Train F1: 0.6896855516400108    Validation F1: 0.24238339174979778
Epoch 6: Train F1: 0.6923499142367067    Validation F1: 0.2560966047204579
Epoch 7: Train F1: 0.6962059991781948    Validation F1: 0.2528119180633147
Epoch 8: Train F1: 0.7028000543699878    Validation F1: 0.2709384979911404
Epoch 9: Train F1: 0.7077684381778743    Validation F1: 0.2522851919561243
Epoch 10: Train F1: 0.7159056083358871   Validation F1: 0.23569065601398165
Epoch 11: Train F1: 0.7233536191479958   Validation F1: 0.25203960396039604
Epoch 12: Train F1: 0.7281625994154034   Validation F1: 0.24584331663399409
Epoch 13: Train F1: 0.7401021063664335   Validation F1: 0.23746556473829203
Epoch 14: Train F1: 0.7450515184381779   Validation F1: 0.23401518075389166
Epoch 15: Train F1: 0.7542891949511015   Validation F1: 0.23714846304774362
Epoch 16: Train F1: 0.7698885634447697   Validation F1: 0.23556851311953353
Epoch 17: Train F1: 0.7813037466698142   Validation F1: 0.22237238159592115
Epoch 18: Train F1: 0.7906047991335838   Validation F1: 0.21728001820146753
Epoch 19: Train F1: 0.7973918037771547   Validation F1: 0.23153439153439154
Epoch 20: Train F1: 0.8056544289636938   Validation F1: 0.2268330108746414
Epoch 21: Train F1: 0.8103489200094943   Validation F1: 0.22551235097015507
```

```
Epoch 22: Train F1: 0.8204377574292814   Validation F1: 0.2271505376344086
Epoch 23: Train F1: 0.8289290681502085   Validation F1: 0.21486570893191756
Epoch 24: Train F1: 0.8379250602982641   Validation F1: 0.21994326860135285
Epoch 25: Train F1: 0.8431292793708902   Validation F1: 0.21592287556662199
Epoch 26: Train F1: 0.8477622138708576   Validation F1: 0.21227684030325264
Epoch 27: Train F1: 0.8568425335684254   Validation F1: 0.21311122280462408
Epoch 28: Train F1: 0.8627038043478261   Validation F1: 0.2190374616717771
Epoch 29: Train F1: 0.8683627941827594   Validation F1: 0.2098447149109659
Epoch 30: Train F1: 0.8730656486468061   Validation F1: 0.21282573575893798
Epoch 31: Train F1: 0.8789899952358265   Validation F1: 0.20957658425689119
Epoch 32: Train F1: 0.8832563196520794   Validation F1: 0.21548192338101058
Epoch 33: Train F1: 0.8883836155699596   Validation F1: 0.21989316383798765
Epoch 34: Train F1: 0.892224945533769    Validation F1: 0.21256291260866725
Epoch 35: Train F1: 0.8941768365230989   Validation F1: 0.21172807472755575
Epoch 36: Train F1: 0.8982305410657665   Validation F1: 0.21360824742268042
Epoch 37: Train F1: 0.9000374136934117   Validation F1: 0.20833623269083568
Epoch 38: Train F1: 0.9037294606940751   Validation F1: 0.20554210384718863
Epoch 39: Train F1: 0.9056091267146543   Validation F1: 0.20789232181095135
Epoch 40: Train F1: 0.910580158379499    Validation F1: 0.20571351387857095
--------------------------------------------------
--------------------------------------------------
Model:
Sequential(
  (0): Linear(in_features=330, out_features=256, bias=True)
  (1): Sigmoid()
  (2): Linear(in_features=256, out_features=128, bias=True)
  (3): Sigmoid()
  (4): Linear(in_features=128, out_features=64, bias=True)
  (5): Sigmoid()
  (6): Linear(in_features=64, out_features=32, bias=True)
  (7): Sigmoid()
  (8): Linear(in_features=32, out_features=16, bias=True)
  (9): Sigmoid()
  (10): Linear(in_features=16, out_features=8, bias=True)
  (11): Sigmoid()
  (12): Linear(in_features=8, out_features=1, bias=True)
  (13): Sigmoid()
)
--------------------------------------------------
Epoch 1: Train F1: 0.6373255737054427    Validation F1: 0.2316191515482199
Epoch 2: Train F1: 0.6728619786676886    Validation F1: 0.23749905152135978
Epoch 3: Train F1: 0.6559461480927449    Validation F1: 0.23742126713597628
Epoch 4: Train F1: 0.6602564102564102    Validation F1: 0.22628145386766077
Epoch 5: Train F1: 0.6627287626331603    Validation F1: 0.23219512195121955
Epoch 6: Train F1: 0.6725852746812631    Validation F1: 0.25052699854061944
Epoch 7: Train F1: 0.6691519105312209    Validation F1: 0.2551948051948052
Epoch 8: Train F1: 0.6724679719603578    Validation F1: 0.2495630367049168
Epoch 9: Train F1: 0.6710792168884554    Validation F1: 0.25013176718620583
Epoch 10: Train F1: 0.6723075315992896   Validation F1: 0.285091437568124
Epoch 11: Train F1: 0.6814287688268619   Validation F1: 0.25194522253345786
Epoch 12: Train F1: 0.6799403747870527   Validation F1: 0.2724080958281702
Epoch 13: Train F1: 0.688725149450855    Validation F1: 0.27246632810775007
Epoch 14: Train F1: 0.6879123964621648   Validation F1: 0.26362101146564754
Epoch 15: Train F1: 0.6894655196728247   Validation F1: 0.25140896839010046
Epoch 16: Train F1: 0.6927747203976566   Validation F1: 0.2626614158805752
Epoch 17: Train F1: 0.7018148564997185   Validation F1: 0.2568627450980392
Epoch 18: Train F1: 0.7130132368075516   Validation F1: 0.2474179829890644
Epoch 19: Train F1: 0.7226594205455763   Validation F1: 0.24336084021005253
Epoch 20: Train F1: 0.7315368215022194   Validation F1: 0.23444883163958832
Epoch 21: Train F1: 0.7386920400482851   Validation F1: 0.24723334118201082
Epoch 22: Train F1: 0.7468277945619336   Validation F1: 0.24507083394187235
Epoch 23: Train F1: 0.7649669730542061   Validation F1: 0.23939508506616258
Epoch 24: Train F1: 0.7777933646629726   Validation F1: 0.24129701259155625
Epoch 25: Train F1: 0.787700291013639    Validation F1: 0.23889690556357224
```

```
Epoch 26: Train F1: 0.7991316830643186   Validation F1: 0.21635012386457472
Epoch 27: Train F1: 0.8100543099846818   Validation F1: 0.22534673622598297
Epoch 28: Train F1: 0.816526854397995    Validation F1: 0.22991566799822458
Epoch 29: Train F1: 0.8286486296803173   Validation F1: 0.22367870589875888
Epoch 30: Train F1: 0.8337844012512741   Validation F1: 0.23314518657886485
Epoch 31: Train F1: 0.8463191459864134   Validation F1: 0.22701635645798085
Epoch 32: Train F1: 0.8523461939520335   Validation F1: 0.22978035701167998
Epoch 33: Train F1: 0.8613895610093328   Validation F1: 0.2291060617973111
Epoch 34: Train F1: 0.8663183029380213   Validation F1: 0.20794038886948424
Epoch 35: Train F1: 0.875227780642943    Validation F1: 0.21483305966064586
Epoch 36: Train F1: 0.8804303138276005   Validation F1: 0.21841217379329342
Epoch 37: Train F1: 0.8853114550656693   Validation F1: 0.21472595930810523
Epoch 38: Train F1: 0.8909065990543411   Validation F1: 0.22459276344241025
Epoch 39: Train F1: 0.8969684485185567   Validation F1: 0.22066343254228402
Epoch 40: Train F1: 0.8974754221902511   Validation F1: 0.20461029327855756
--------------------------------------------------
```

In [ ]:
```
plt.figure(figsize=(15, 7))
plt.plot(range(1, 1+len(epoch_loss)),epoch_loss)
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Epochs vs. Train loss, Validation loss')
plt.legend(['Train', 'Validation'])
plt.grid(b=True)
plt.show()
```



In [ ]:
```
hcdrLog
```

Out[ ]:

	Architecture string	Optimizer	Epochs	Train F1	Valid F1
0	330-32-16-8-1	<class 'torch.optim.adagrad.Adagrad'>, ReLU()	40	0.719434725005945	0.24847781965787183
1	330-64-32-16-8-1	<class 'torch.optim.adagrad.Adagrad'>, ReLU()	40	0.7677993251763745	0.2394981674654638
2	330-128-64-32-16-8-1	<class 'torch.optim.adagrad.Adagrad'>, ReLU()	40	0.8144951939942469	0.2304607901137185
3	330-256-128-64-32-16-8-1	<class 'torch.optim.adagrad.Adagrad'>, ReLU()	40	0.9338886685177838	0.2212722793798
4	330-32-16-8-1	<class 'torch.optim.adam.Adam'>, ReLU()	40	0.7505960217770344	0.23693273130787482
5	330-64-32-16-8-1	<class 'torch.optim.adam.Adam'>, ReLU()	40	0.7780175621372227	0.23788265306122447
6	330-128-64-32-16-8-1	<class 'torch.optim.adam.Adam'>, ReLU()	40	0.8652066086835027	0.2224382116078867
7	330-256-128-64-32-16-8-1	<class 'torch.optim.adam.Adam'>, ReLU()	40	0.7205887960950311	0.24448930157448523
8	330-32-16-8-1	<class 'torch.optim.adagrad.Adagrad'>, Sigmoid()	40	0.698269476802909	0.24965567234505254
9	330-64-32-16-8-1	<class 'torch.optim.adagrad.Adagrad'>, Sigmoid()	40	0.6976383265856949	0.25033427425345417
10	330-128-64-32-16-8-1	<class 'torch.optim.adagrad.Adagrad'>, Sigmoid()	40	0.7054565136497412	0.24811020840692335
11	330-256-128-64-32-16-8-1	<class 'torch.optim.adagrad.Adagrad'>, Sigmoid()	40	0.697633595516997	0.24842857142857144
12	330-32-16-8-1	<class 'torch.optim.adam.Adam'>, Sigmoid()	40	0.7900989443727457	0.22311810190998613
13	330-64-32-16-8-1	<class 'torch.optim.adam.Adam'>, Sigmoid()	40	0.8548803697661772	0.21449784017278617
14	330-128-64-32-16-8-1	<class 'torch.optim.adam.Adam'>, Sigmoid()	40	0.910580158379499	0.20571351387857095
15	330-256-128-64-32-16-8-1	<class 'torch.optim.adam.Adam'>, Sigmoid()	40	0.8974754221902511	0.20461029327855756

PyTorch NN Experiments

- In our experiments, we tested different two different types of layers (ReLU and Sigmoid), two different optimizers (Adam and Adagrad), and several different hidden layer architectures, which were

$$32 - 16 - 8$$

$$64 - 32 - 16 - 8$$

$$128 - 64 - 32 - 16 - 8$$

$$256 - 128 - 64 - 32 - 16 - 8$$

  with 330 input neurons and 1 output neuron.

- Our architecture alternated with Linear nodes and either ReLU or Sigmoid, whichever one was being tested. Every model finished with a Sigmoid node to output a number between 0 and 1. The equations for these nodes are seen here.

$$ReLU : f(x) = max(0, x)$$

$$Sigmoid : f(x) = \frac{1}{1 + \exp(-x)}$$

- The difference between these two activation functions is that if ReLU sees a negative value, it will become 0, while Sigmoid transforms it into a value between 0 and 1.

- With two different types of layers, two different optimizers, and four different hidden layer architectures, we had 16 different possible neural network architecture combinations.

- We also kept track of the training and validation loss over the epochs so that we could determine the best number of epochs to run. We can see the epoch curve of the 6 hidden layers, Sigmoid activiation functions, and Adam optimizer experiment below.

- We also have the experimental log of all of the different architectures. As seen in the plot, the validation loss stops reducing around the 10 epochs mark.

- In the log, we find that the best parameters were the Adagrad optimizer, the Sigmoid activation function, and the $64 - 32 - 16 - 8$ hidden layers. We can use this architecture to test on the Kaggle test cases.

In [ ]:
```
plt.figure(figsize=(15, 7))
plt.plot(range(1, 1+len(epoch_loss)),epoch_loss)
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Epochs vs. Train loss, Validation loss')
plt.legend(['Train', 'Validation'])
plt.grid(b=True)
plt.show()
```


Epochs vs. Train loss, Validation loss
Train
Validation
Loss
Epochs
1.0
0.8
0.6
0.4
0
5
10
15
20
25
30
35
40

In [ ]:
```
hcdrLog
```

Out[ ]:

	Architecture string	Optimizer	Epochs	Train F1	Valid F1
0	330-32-16-8-1	<class 'torch.optim.adagrad.Adagrad'>, ReLU()	40	0.719434725005945	0.24847781965787183
1	330-64-32-16-8-1	<class 'torch.optim.adagrad.Adagrad'>, ReLU()	40	0.7677993251763745	0.2394981674654638
2	330-128-64-32-16-8-1	<class 'torch.optim.adagrad.Adagrad'>, ReLU()	40	0.8144951939942469	0.2304607901137185
3	330-256-128-64-32-16-8-1	<class 'torch.optim.adagrad.Adagrad'>, ReLU()	40	0.9338886685177838	0.2212722793798
4	330-32-16-8-1	<class 'torch.optim.adam.Adam'>, ReLU()	40	0.7505960217770344	0.23693273130787482
5	330-64-32-16-8-1	<class 'torch.optim.adam.Adam'>, ReLU()	40	0.7780175621372227	0.23788265306122447
6	330-128-64-32-16-8-1	<class 'torch.optim.adam.Adam'>, ReLU()	40	0.8652066086835027	0.2224382116078867
7	330-256-128-64-32-16-8-1	<class 'torch.optim.adam.Adam'>, ReLU()	40	0.7205887960950311	0.24448930157448523
8	330-32-16-8-1	<class 'torch.optim.adagrad.Adagrad'>, Sigmoid()	40	0.698269476802909	0.24965567234505254
9	330-64-32-16-8-1	<class 'torch.optim.adagrad.Adagrad'>, Sigmoid()	40	0.6976383265856949	0.25033427425345417
10	330-128-64-32-16-8-1	<class 'torch.optim.adagrad.Adagrad'>, Sigmoid()	40	0.7054565136497412	0.24811020840692335
11	330-256-128-64-32-16-8-1	<class 'torch.optim.adagrad.Adagrad'>, Sigmoid()	40	0.697633595516997	0.24842857142857144
12	330-32-16-8-1	<class 'torch.optim.adam.Adam'>, Sigmoid()	40	0.7900989443727457	0.22311810190998613
13	330-64-32-16-8-1	<class 'torch.optim.adam.Adam'>, Sigmoid()	40	0.8548803697661772	0.21449784017278617
14	330-128-64-32-16-8-1	<class 'torch.optim.adam.Adam'>, Sigmoid()	40	0.910580158379499	0.20571351387857095
15	330-256-128-64-32-16-8-1	<class 'torch.optim.adam.Adam'>, Sigmoid()	40	0.8974754221902511	0.20461029327855756

In [ ]:
```
# hcdrLog.to_csv('hcdrLog.csv', index=False)
```

In [ ]:
```
hcdrLog = pd.read_csv('/kaggle/input/nn-preds/hcdrLog.csv')
hcdrLog.head()
```

Out[ ]:

	Architecture string	Optimizer	Epochs	Train F1	Valid F1	Test F1
0	330-32-16-8-1	<class 'torch.optim.adagrad.Adagrad'>, ReLU()	40	0.719435	0.248478	0.255566
1	330-64-32-16-8-1	<class 'torch.optim.adagrad.Adagrad'>, ReLU()	40	0.767799	0.239498	0.243441
2	330-128-64-32-16-8-1	<class 'torch.optim.adagrad.Adagrad'>, ReLU()	40	0.814495	0.230461	0.229859
3	330-256-128-64-32-16-8-1	<class 'torch.optim.adagrad.Adagrad'>, ReLU()	40	0.933889	0.221272	0.224393
4	330-32-16-8-1	<class 'torch.optim.adam.Adam'>, ReLU()	40	0.750596	0.236933	0.238545

In [ ]:
```
hcdrLog.sort_values(by='Test F1', ascending=False).head(5)
```

Out[ ]:

	Architecture string	Optimizer	Epochs	Train F1	Valid F1	Test F1
9	330-64-32-16-8-1	<class 'torch.optim.adagrad.Adagrad'>, Sigmoid()	40	0.697638	0.250334	0.262235
8	330-32-16-8-1	<class 'torch.optim.adagrad.Adagrad'>, Sigmoid()	40	0.698269	0.249656	0.259600
11	330-256-128-64-32-16-8-1	<class 'torch.optim.adagrad.Adagrad'>, Sigmoid()	40	0.697634	0.248429	0.257255
0	330-32-16-8-1	<class 'torch.optim.adagrad.Adagrad'>, ReLU()	40	0.719435	0.248478	0.255566
10	330-128-64-32-16-8-1	<class 'torch.optim.adagrad.Adagrad'>, Sigmoid()	40	0.705457	0.248110	0.255508

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.
```

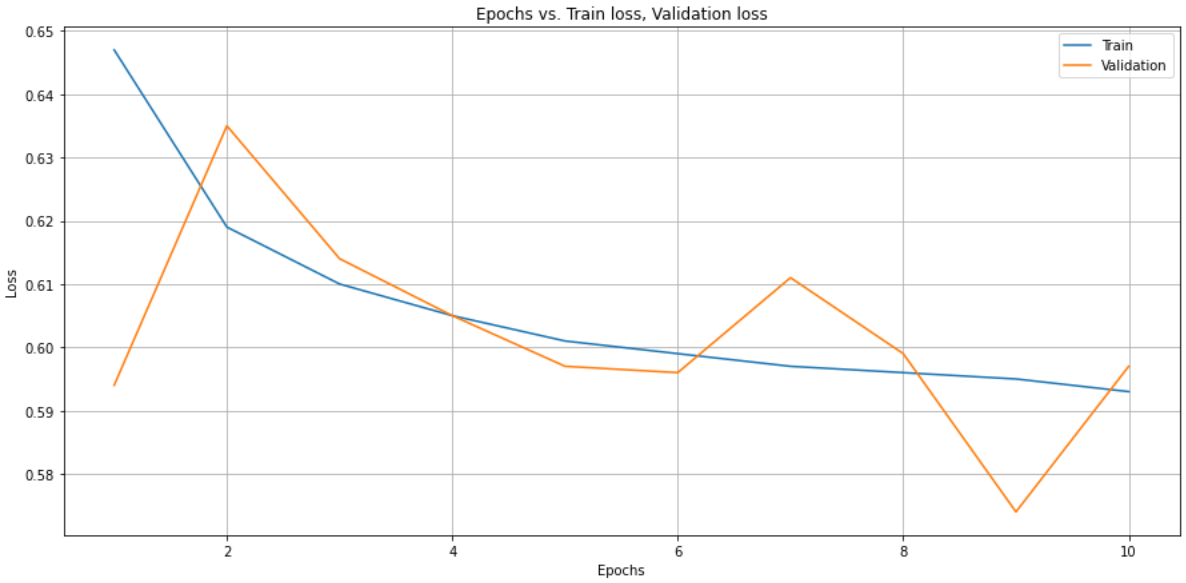
- We will use the Pytorch MLP as our best model since it gives the best F1-score on the test set (approximately 0.26).
- We will re-train the best MLP amongst the above models with Sigmoid activation, Adagrad Optimizer over 10 epochs with a learning rate of 0.01 and the architecture looks like this: [64,32,16,8]

image

In [ ]:
```
arch_string, train_f1, valid_f1, test_f1, epoch_loss, predictions = run_hcdr_model
                layer_type=nn.Sigmoid(),
                hidden_layer_neurons=[64,32,16,8],
                opt=optim.Adagrad,
                epochs=10,
                learning_rate=1e-2
            )
```

```
--------------------------------------------------
Model:
Sequential(
  (0): Linear(in_features=330, out_features=64, bias=True)
  (1): Sigmoid()
  (2): Linear(in_features=64, out_features=32, bias=True)
  (3): Sigmoid()
  (4): Linear(in_features=32, out_features=16, bias=True)
  (5): Sigmoid()
  (6): Linear(in_features=16, out_features=8, bias=True)
  (7): Sigmoid()
  (8): Linear(in_features=8, out_features=1, bias=True)
  (9): Sigmoid()
)
--------------------------------------------------
Epoch 1: Train F1: 0.6565708243800948    Validation F1: 0.25704863973619124
Epoch 2: Train F1: 0.6748403049660004    Validation F1: 0.243075420358613
Epoch 3: Train F1: 0.6834368740876532    Validation F1: 0.24676258992805758
Epoch 4: Train F1: 0.6872395921547373    Validation F1: 0.25096835489293284
Epoch 5: Train F1: 0.6915862627353081    Validation F1: 0.25223626820433204
Epoch 6: Train F1: 0.6918243243243243    Validation F1: 0.25131354991489674
Epoch 7: Train F1: 0.6925806996788912    Validation F1: 0.24839460870792462
Epoch 8: Train F1: 0.6939822471227514    Validation F1: 0.2511099788922047
Epoch 9: Train F1: 0.6949460916442048    Validation F1: 0.25593523779870786
Epoch 10: Train F1: 0.69286487950176     Validation F1: 0.25178910471739446
--------------------------------------------------
```

In [ ]:
```
plt.figure(figsize=(15, 7))
plt.plot(range(1, 1+len(epoch_loss)),epoch_loss)
plt.xlabel('Epochs')
plt.ylabel('Loss')
plt.title('Epochs vs. Train loss, Validation loss')
plt.legend(['Train', 'Validation'])
plt.grid(b=True)
plt.show()
```



In [ ]:
```
# y_kaggle_test_scores = clf_gridsearch.predict_proba(X_kaggle_test)[:, 1]
y_kaggle_test = X_kaggle_test[['SK_ID_CURR']]
y_kaggle_test['TARGET'] = predictions
print(y_kaggle_test.head())
y_kaggle_test.to_csv("nn_tuned_submission.csv", index=False)
```

```
   SK_ID_CURR    TARGET
0      100001  0.373540
1      100005  0.728149
2      100013  0.240932
3      100028  0.277968
4      100038  0.707481
```

Using the best model for our submission on Kaggle

Kaggle submission via the command line API

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

Report submission

Click on this link

Home-Credit-Default-Risk

Final Project for Fall 2022 course: CSCI-P 556 - Applied Machine Learning

Abstract

In this project, we are attempting to predict whether a specific client will repay the loan they have taken out. The main goal of this phase is to build a Multi-Layer Perception model in PyTorch for loan default classification. The initial phase was all about understanding the project requirements, understanding the HCDR dataset and building project plan. In phase 2, we performed data preprocessing (such as using imputer for missing values), Exploratory Data Analysis, some feature engineering (Polynomial Features) and created baseline models for prediction. Phase 3 involved more feature engineering (aggregated features based on existing numerical and categorical features) and Hyperparameter tuning using GridSearchCV. In this phase, we first performed Random Undersampling to remove the imbalance in the data. Feature importance was performed to derive the relative importance of features during prediction (using Random Forest Algorithm and feature*importances* property). We have created separate pipelines for each model which have been fine tuned further for comparative analysis. Furthermore, we have trained an ANN model based on the features extracted in previous phases. We were able to extract the best performance with the developed neural network (F1 Score = 0.262235) after testing on the entire training data. Upon Kaggle Submission, the best performing model was again the developed neural network with an AUC private score of 0.732 and public score of 0.7361.

Project Description

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

Some of the challenges

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

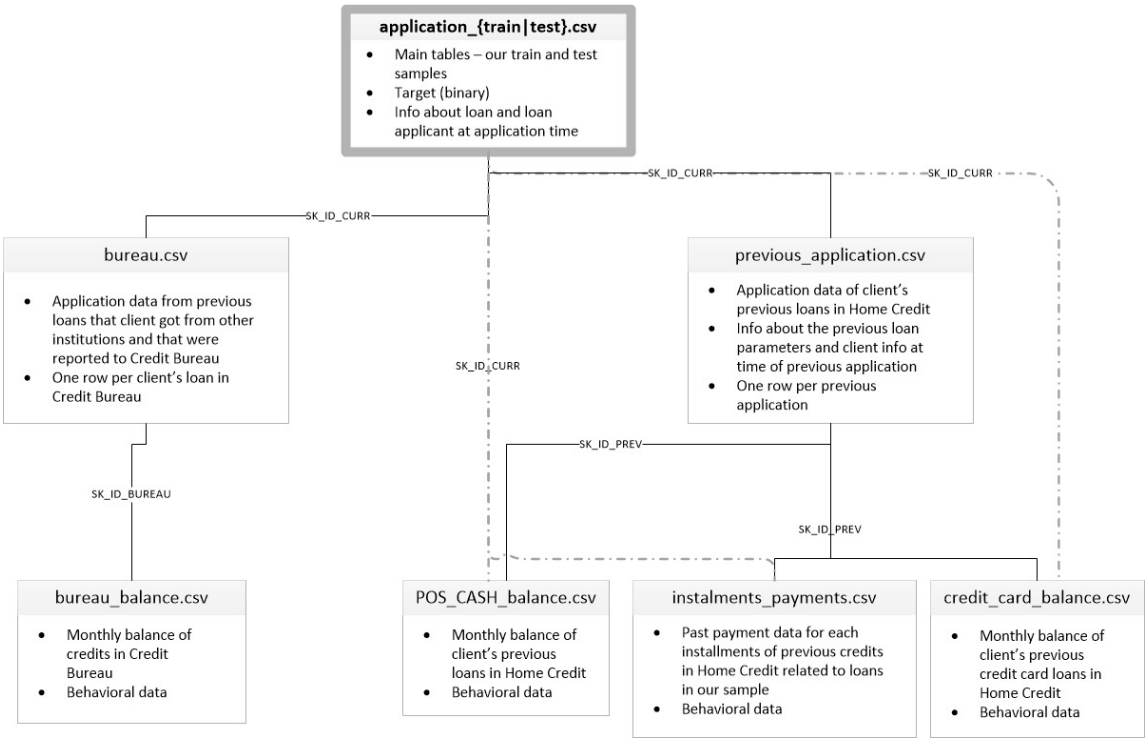
- Dealing with missing data
- Imbalanced datasets
- Summarizing transaction data

Dataset and Project Description

The main training section of the data is compiled in a CSV labeled “application_train.csv” and the counterpart test data in “application_test.csv”. The training data set’s columns differ from the test set only in that it includes the target column for whether the client repaid the loan. There are 121 feature columns in the application dataset which include both categorical, such as gender, and numerical features, such as income.

There are several other datasets that correlate to the application dataset, and they build off each other. The "bureau.csv" dataset contains a row for each previous credit the client had before the application date corresponding to the loan in the application dataset. The "bureau_balance.csv" dataset contains each month of history there is data for the previous credits mentioned in the bureau dataset. In this way, these subsidiary datasets build off the application dataset and each other. These subsidiary datasets contain categorical values and positive and negative numerical values. There are 6 subsidiary datasets in total, but the main dataset we will be looking at is the application set, as this is what the test set is based on.

The following diagram shows the relation between various datasets provided in the problem:



application_{train|test}.csv This is the main dataset consisting of test and training samples. It consists of personal details about the loan applicant and connects with other sets using SK_ID_CURR attribute. **previous_application.csv** This dataset consists of information about applicant's previous loans in Home credit. It contains attributes such as type of loan, status of loan, down-payment etc. **instalments_payments.csv** This dataset contains information about repayment of previous loans. **credit_card_balance.csv** This dataset consists of transaction information related to Home Credit credit cards. **POS_CASH_balance.csv** This dataset consists of entries that define the status of previous credit of the individual at Home Credit which includes consumer credit and cash loans. **bureau.csv** This dataset consists of information of the individual's previous credit history at other financial institutions that were reported to the credit bureau.

Exploratory Data Analysis

- Proper EDA can be viewed at this **link**

Feature Engineering

- This phase involves preparing features from the application_train.csv and application_test.csv datasets by introducing 3rd-degree polynomial features. We took a subset of that new data to train and evaluate the baseline models. We also used other secondary tables that are provided to us in the HCDR dataset. We have primarily focused on previous_application.csv and bureau.csv tables in this phase.

- We will divide this phase of feature engineering into three parts:

    - Feature Engineering in **previous_application.csv** table.
    - Feature Engineering in **brueau.csv** table.
    - Merging the two secondary tables with **application_train.csv** and **application_test.csv** tables.

For previous_application.csv table:

- In this table, we have created three types of aggregated features (min, max, mean) for the following columns:

```
'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'
```

- For `AMT_APPLICATION`, we also created a `range_AMT_APPLICATION` which takes the range from the maximum and minimum of the said feature.
- Then we performed a simple merge function of the newly prepared aggregated table with the `application_train.csv` and `application_test.csv` tables based on the `SK_ID_CURR` identifier.

For bureau.csv table:

- We see that 263,491 applicants have their data in this table.
- We first created a previous_loan_count feature for all these applicants based on the `SK_ID_BUREAU` feature.
- Then we created five types of aggregated features (count, mean, max, min, sum) for the rest of the columns by performing a groupby on the `SK_ID_CURR` feature:
- After appropriately renaming the columns, we performed a merge of the aggregated bureau table with the `application_train.csv` and `application_test.csv` tables based on the `SK_ID_CURR` identifier.

- At the end of this series of operations, we are left with 331 columns in the training dataset (merged application_train.csv) and 330 columns in the testing dataset (merged application_test.csv).

Hyperparameter Tuning

- In order to efficiently perform a hyperparameter tuning grid search, we needed to use a roundabout method.
- Our goal for the hyperparameter tuning was to find which classifier worked the best, and with what hyperparameters.
- We tested logistic regression (LogisticRegression), random forests (RandomForestClassifier), decision trees (DecisionTreeClassifier), the GradientBoostingClassifier, CatBoostClassifier, LightGBMClassifier, AdaBoostClassifier, and lastly the XGBoostClassifier.
- We trained these models on the downsampled dataset
- Additionally, we tested many hyperparameters for each of these models. We used the Scikit-Learn library's GridsearchCV to run our grid search.

- **Logistic Regression**: we tested the penalty type and the penalty coefficient.

- **Random Forest**: we tested the number of trees, the max depth of each tree, and the minimum number of samples allowed for each leaf node.
- For decision trees, we tested the max depth and the minimum samples for leaf nodes.
- **Gradient Boosting Classifier**: we tested the number of trees, the minimum samples for leaf nodes, the max depth, and the subsample parameter that controls the proportion of samples to be used to fit each tree.
- **Catboost classifier**: we tried a combination of n_estimators, learning rates, number of iterations, and max_depth.
- **Light gradient boost classifier**: tried different values of l1 and l2 regularizations, as well as the max_depth of the model.
- **XGBoost classifier**: we tested the number of trees, the max depth, the subsample, a 'colsample_bytree' parameter which controls the proportion of columns used to construct each tree, the alpha determining the amount of L1 regularization, the lambda for L2 regularization, and gamma, which determines how much loss reduction is needed to partition further down the tree.

- Some of these classifiers take a lot of time to train but eventually they yielded good results.

Data Leakage

- Data Leakage has been a concern in this dataset since the beginning of this project.

- There are multiple ways to ensure that there is no leakage of data among the subsets of data.

    - After performing feature engineering, we dropped the SK_ID_CURR feature among the X_train, X_valid, X_test, and X_kaggle_test sets and performed an intersection among the datasets to check for any overlaps of data.
    - The goal is to check if there are any rows that are same in the either dataset. Fortunately, **there were none**, indicating that there is no such row that might affect the held-out datasets.

- We already had a held-out dataset whose score is calculated on kaggle, but we still performed segregation on the engineered datasets by creating X_valid and X_test sets.
- Among these two, X_valid was used as an evaluation set, that may cause some leakage between X_valid and X_train features, but ensured there is no leakage between X_test and X_train datasets, and subsequently no leakage between X_kaggle_test and X_train datasets.
- There might be some information leakage from X_train set over to the other datasets due to imputation and PCA operations but that is as far as the data leakage goes.

- As for **cardinal sins of machine learning**, we are confident to check that we have not made such mistakes and assumptions on the data that might affect the EDA, interpretation, or the modelling and predictions of the plethora of models that we have implemented.

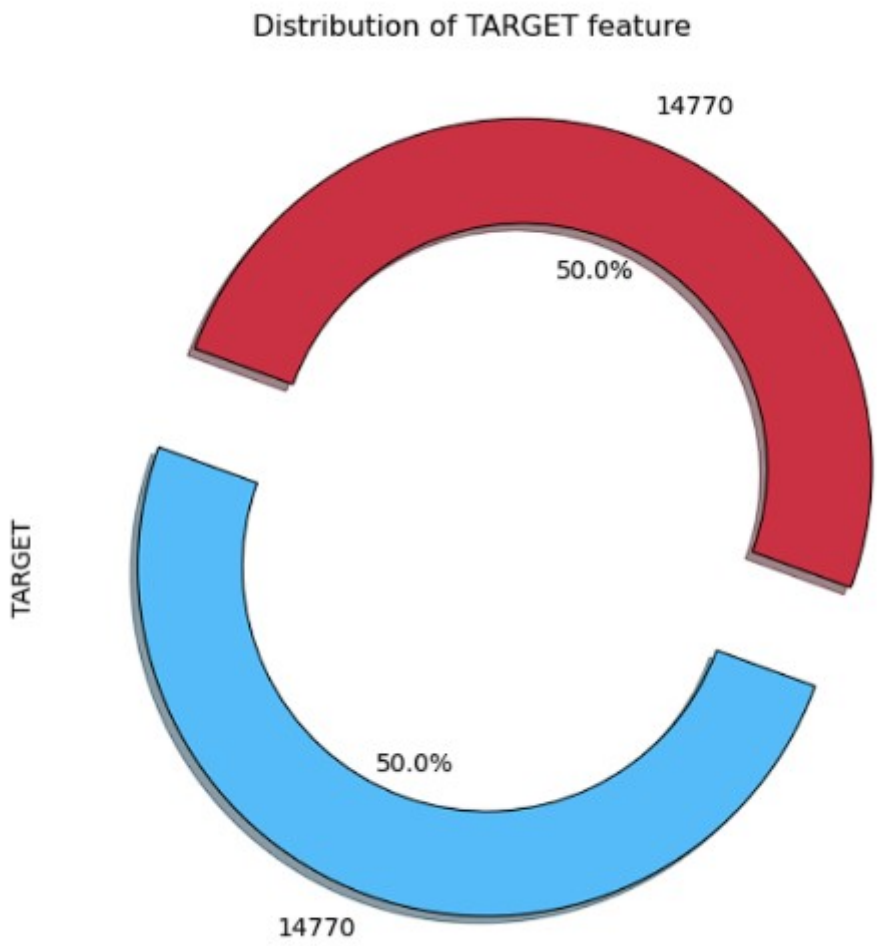
Pipelines

- Just like last phase, creating separate pipelines will help us to fill in missing values and perform scaling on numerical features.
- Our pipeline uses SimpleImputer to fill in the missing values with the median being used for the numerical pipeline and the most frequent value being used for the categorical pipeline.
- One Hot Encoding is used in the categorical pipeline to convert categorical values into integers for improving the performance of our models.
- Standard Scaler is used in the numerical pipeline for scaling purposes. - - The two pipelines are then combined into a single ' `data_prep_pipeline` '

- Additionally, to avoid imbalance class issues, we have undersampled the data (using RandomUnderSampler class) in this phase to make it balanced.

```
# get importance
importance = clf_pipe['model'].feature_importances_
# summarize feature importance
important_features = pd.DataFrame(zip(X_under.columns, importance), columns=['Feature Name', 'Feature Score'])
important_features.sort_values(by='Feature Score', ascending=False, inplace=True)
important_features.head(20)
```

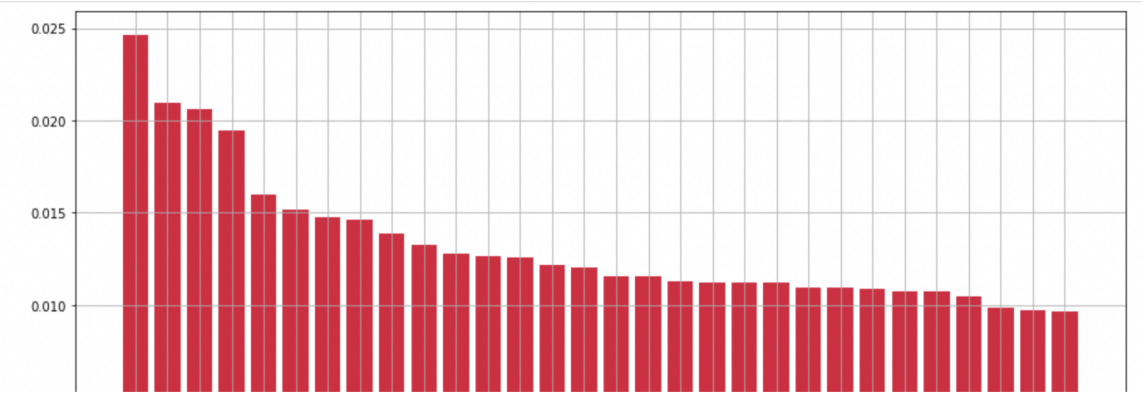
- The target feature exhibits the following distribution:



- We have then performed Random Forest feature importance (assigning scores to input features) on the undersampled data.
- This has been done to determine the relative importance of features while predicting the target variable.
- RandomForestClassifier class has been used to implement Random Forest algorithm for feature importance. After training the model, we have used the feature*importances* property to retrieve the relative importance scores for each input feature.

```
model = RandomForestClassifier()
clf_pipe = Pipeline([("preparation", data_prep_pipeline),
                     ("model", model)])
# fit the model
clf_pipe.fit(X_under, y_under)
```

- Some of the important features are as follows:



- Before sending the data for modeling and experiments, we introduce Principal Component Analysis to the pipeline with an explained variance of 0.85.
- This is then followed by hyperparameter tuning of the models.

- Multilayer Perceptron Pipeline:
  - First data imputation is performed followed by converting data into Tensors and then finally passed to the MLP model.
  - For our MLPs, we will use BinaryCrossEntropy Loss for optimization:

$$BinaryCrossEntropy = H_p(q) = -\frac{1}{N}\sum_{i=1}^{n} y_i . log(p(y_i)) + (1-y_i) . log(1-p(y_i))$$

- We primarily focus on these two performance metrics and loss functions:

$$F1 = \frac{2 * Precision * Recall}{Precision + Recall} = \frac{2 * TP}{2 * TP + FP + FN}$$

$$Sensitivity = Recall = \frac{TP}{TP + FN}$$

$$Specificity = \frac{TN}{FP + TN}$$

AUC is calculated as the Area Under the $Sensitivity$(TPR)-$(1 - Specificity)$(FPR) Curve.

Experimental results

- After Downsampling the training dataset and appropriately tuning the models, we get the following performacne report:

[178]:
```
expLog
```

[178...

	Model name	Best Model Parameters	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC	Train F1	Valid F1	Test F1	Tune Time
0	Tuned LogisticRegression	{'classifier': LogisticRegression(C=1, class_w...	0.6679	0.6400	0.6467	0.7289	0.7181	0.7269	0.6765	0.2340	0.2411	31.0960
1	Tuned LogisticRegression	{'classifier': LogisticRegression(C=1, class_w...	0.6679	0.6400	0.6467	0.7289	0.7181	0.7269	0.6765	0.2340	0.2411	31.0960
2	Tuned GradientBoostingClassifier	{'classifier': GradientBoostingClassifier(max_...	0.7056	0.6606	0.6665	0.7803	0.7204	0.7289	0.7076	0.2405	0.2467	188.2789
3	Tuned DecisionTreeClassifier	{'classifier': DecisionTreeClassifier(max_dept...	0.7312	0.6168	0.6182	0.8095	0.6629	0.6730	0.7363	0.2133	0.2160	11.0418
4	Tuned RandomForestClassifier	{'classifier': RandomForestClassifier(min_samp...	0.9649	0.6669	0.6702	0.9956	0.7134	0.7211	0.9649	0.2408	0.2446	89.0510
5	Tuned CatBoostClassifier	{'classifier': <catboost.core.CatBoostClassifi...	0.6568	0.6568	0.6616	0.7164	0.7071	0.7158	0.6555	0.2330	0.2387	1716.3431
6	Tuned XGBClassifier	{'classifier': XGBClassifier(base_score=None, ...	0.7378	0.6562	0.6598	0.8192	0.7129	0.7212	0.7407	0.2371	0.2407	3296.5660
7	Tuned LGBMClassifier	{'classifier__learning_rate': 0.1, 'classifier...	0.7025	0.6691	0.6744	0.7754	0.7226	0.7307	0.7030	0.2426	0.2503	34.4742
8	Tuned AdaBoostClassifier	{'classifier': AdaBoostClassifier(algorithm='S...	0.6547	0.6363	0.6398	0.6646	0.6583	0.6632	0.6598	0.2270	0.2311	3460.4755

- Moving a step further, we created various MLPs (more than 15) and these are the top 5 models based on best Test F1 score:

- We observe that some MLPs overfit after 10-15 epochs so we limit them to 10 epochs. This is how the training/validation loss curve looks like over the epochs:

- After selecting the best model, we see that our best performing model looks like this:

.png?token=GHSAT0AAAAAAAAASPN2UETHAEAY7WAZNZ5UY6AMY4A)

- The best scores were obtained for **PyTorch FCNN**, **LightGBM**, and **LogisticRegression**.

Discussion

- This phase of the project primarily focused on two things:
  - Reduce the complexity of the dataset and tune the classical and ensemble models
  - Prepare a Multi-Layer Perceptron using PyTorch library and select the best set of hyperparameters.
- **Focus #1** involved making an important decision - to reduce the number of rows of the training dataset and select the important features.
- RandomUnderSampler and RandomForestClassifier's feature*importances* enables us to achieve these with ease.
- **LightGBM** and **LogisticRegression** performed very well over this data and gave a score of above **0.725** on the kaggle scoreboard.
- **Focus #2** pushed our skills to the limit. Even though we had a reference code from HW11, we were unable to improvise on the F1 score for a long time.
- A lot of tweaking enabled us to finally perfect the architecture and run the code on the entire dataset with ease.
- The model started overfitting after 10 epochs due to complexity of dataset so we grounded the networks to 10-15 epochs.

- Our **best FCNN** was then determined and after another set of tweaks with the model architecture, we were able to score a kaggle public score of **0.736**, best so far amongst all the models.

Conclusion

- In this project, we are attempting to predict whether a specific client will repay the loan they have taken out. We had performed Exploratory Data Analysis, some feature engineering and created baseline models in the initial two phases. In the previous phase (i.e. phase 3), we merged the datasets, performed some more feature engineering (deriving polynomial and aggregated features) and hyperparameter tuning obtaining the best F1 score of 0.148820 for Decision Tree Classifier.
- Upon Kaggle submission in **past** phases, Logistic Regression was the best performing model with an ROC-AUC score of approximately 72%.
- In the **current phase**, we performed Random Undersampling to remove the imbalance in the dataset. We also performed Feature Importance to derive the relative importance scores of input features during prediction (using Random Forest Algorithm) and reduce the complexity of the dataset.
- In this last phase, we have implemented a Multi-Layer Perception (MLP) model for loan default classification. We obtained the best performance with the developed MLP model with an F1 Score of 0.262235.
- Upon Kaggle Submission, the best performing model was again the developed neural network with an AUC private score of 73.21% and public score of 73.61%. We were able to obtain our best performance by implementing the MLP model in the current phase.
- For **Future** scope, we can try improving the accuracy by increasing the complexity of the neural network and performing more hyperparameter tuning.

Kaggle Submission

- These are the final submissions of all the tuned models:

- Our Final best Predicting Model yields the best final score:

References

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

- The above notebook was referred for the phase
- Implementation of imblearn.RandomUnderSampler was referred from the following link: https://machinelearningmastery.com/random-oversampling-and-undersampling-for-imbalanced-classification/

- Implementation of RandomForestClassifier().feature*importances* was referred from the following link: https://machinelearningmastery.com/calculate-feature-importance-with-python/

- PyTorch implementation has been referred from **HW11-PyTorch-Basics**.

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/