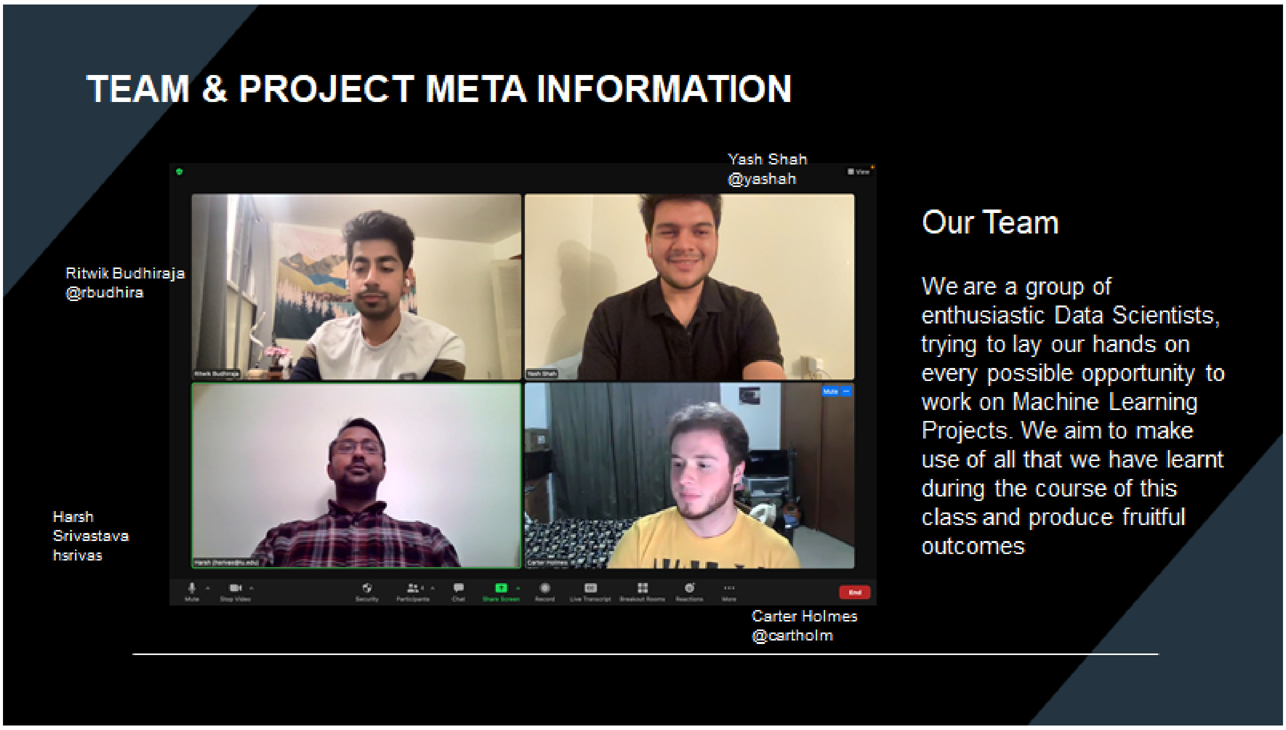
Project Title: Home Credit Default Risk

Team and Project Meta-Information

Group 32

- Carter Holmes `cartholm@iu.edu`
- Harsh Srivastava `hsrivas@iu.edu`
- Ritwik Budhiraja `rbudhira@iu.edu`
- Yash Shah `yashah@iu.edu`



ABSTRACT

Many people have a tough time getting loans because their credit histories are poor or non-existent. Untrustworthy lenders, unfortunately, may often take advantage of this demographic. Home Credit strives to enhance financial inclusion for the unbanked by promoting an enjoyable and secure borrowing experience. To guarantee that this underserved group has a positive loan experience, Home Credit employs a variety of alternative data, such as the person's background information, and machine learning algorithms to anticipate their clients' repayment potential. The goal of this project is to create an effective classification system for determining a client's financial ability to pay their debts. We have selected this data source as our final project for the course. To find the best solution, we will investigate a variety of machine learning methods, from traditional machine learning to deep learning, feature engineering, and hyperparameter searches; while using what we have learnt during the course – put to its best use. In Phase 1, we built a very basic model with two algorithms, logistic regression and random forest. In Phase 2, we find that the hyperparameter searches and feature engineering help us find a better model with an increased accuracy score. In Phase 3, we finish our final goal of building a model with an Artificial Neural Network. Through the last two phases we have tackled the problems like data processing issues and memory issues, that we have tackled by performing various data transformations and saving data into temporary files. As the final result of our Phase 3, we found out that the neural network we built gives an accuracy score of around 91.8% which we have selected for our final Kaggle submission. In this submission we got a score of 0.722 (public) and 0.709 (private).

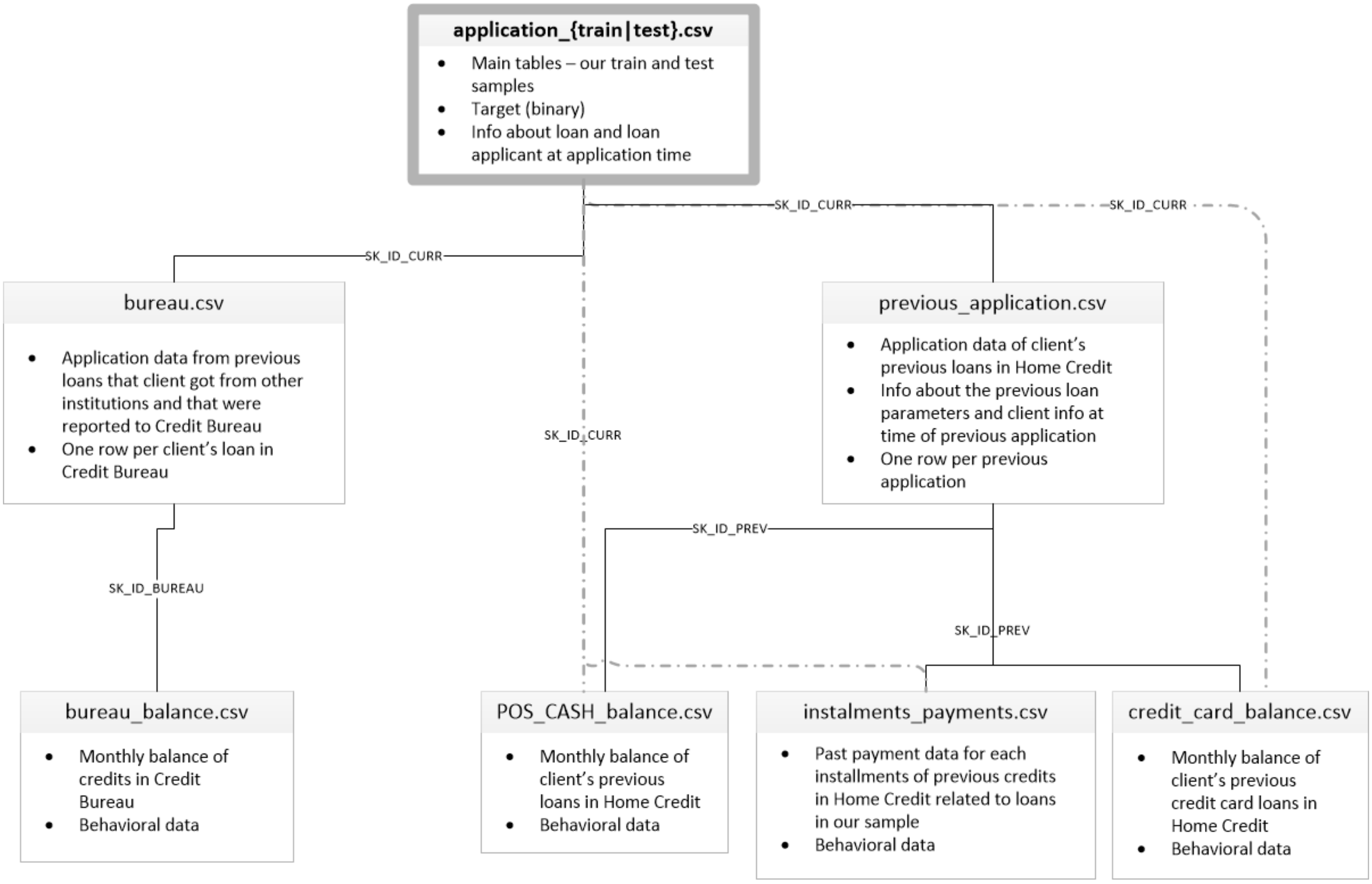
PROJECT DESCRIPTION

In this phase of the project, we primarily focused on building an Artificial Neural Network for attaining better accuracies on our training model. This was the main task that has been tackled in this phase. To find a model that was best suited for us, we had to work on various permutations and combinations of the number of layers, number of perceptrons, and the number of features which posed as a problem that we faced while building the ANN. We tried to implement five different models for the same (out of which three are not present in the code log), and decided to proceed with 18 features. We shall discuss the results of this model in the further sections.

DATA DESCRIPTION

The HCDR data source consists of a cluster of nine '.csv' files, as highlighted in the below image. Additionally, we have a large chunk of the remaining files that provide us with data about financial records of the applicants. To help with the labelling of data columns, there is also a column description 'HomeCredit_columns_description.csv' file present in the data source.

Here's a link to the dataset: https://www.kaggle.com/c/home-credit-default-risk/data



The files and their contents can be summarized as follows-

- ***application.csv*** \ This is the core dataset, which is split into train and test datasets and includes information about loans and loan applicants at the time of application.
- ***bureau.csv*** \ This file provides information on clients' loan histories that were reported to the Credit Bureau by institutes. In addition, in the Credit Bureau, there is one row dedicated to each client's loan.
- **_bureau*balance.csv*** \ This file contains the monthly balances of previous Credit Bureau credits.
- **_previous*application.csv*** \ This file contains information about the applicant's previous loan in Home credit, as well as past loan parameters and the applicant's personal information at the time.
- **_POS_CASH*balance.csv*** \ This file contains monthly balances from the applicant's previous point of sales (POS) and loans in the form of cash from Home Credit.
- **_installments*payments.csv*** \ This file comprises clients' previous payment history for each installment for previous Home Credit credits related to the loan granted to them.
- **_credit_card*balance.csv*** \ This file contains snapshots of monthly balances from a client's previous Home Credit credit card history.

Mount Google Drive on Colab

In [ ]:

```
from google.colab import drive
drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

Getting the Dataset from Kaggle

In [ ]:

```
!pip install kaggle
```

```
Requirement already satisfied: kaggle in /usr/local/lib/python3.9/site-packages (1.5.12)
Requirement already satisfied: requests in /usr/local/lib/python3.9/site-packages (from kaggle) (2.26.0)
Requirement already satisfied: tqdm in /usr/local/lib/python3.9/site-packages (from kaggle) (4.62.3)
Requirement already satisfied: python-slugify in /usr/local/lib/python3.9/site-packages (from kaggle) (5.0.2)
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.9/site-packages (from kaggle) (2.8.2)
Requirement already satisfied: certifi in /usr/local/lib/python3.9/site-packages (from kaggle) (2021.10.8)
```

```
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.9/site-packages (from kaggle) (1.15.0)
Requirement already satisfied: urllib3 in /usr/local/lib/python3.9/site-packages (from kaggle) (1.26.7)
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.9/site-packages (from python-slugify->kaggle) (1.3)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/site-packages (from requests->kaggle) (3.3)
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/python3.9/site-packages (from requests->kaggle) (2.0.4)
WARNING: Running pip as the 'root' user can result in broken permissions and conflicting behaviour with the system package manager. It is recommended to use a virtual environment instead: https://pip.pypa.io/warnings/venv
WARNING: You are using pip version 21.3.1; however, version 22.0.4 is available.
You should consider upgrading via the '/usr/local/bin/python -m pip install --upgrade pip' command.
```

In [ ]:

```
!pwd
```

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

In [ ]:

```
!mkdir ~/.kaggle
!cp ./kaggle.json ~/.kaggle
!chmod 600 ~/.kaggle/kaggle.json
```

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

In [ ]:

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

```
name                                 size  creationDate
----------------------------------  -----  -------------------
bureau.csv                          162MB  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
HomeCredit_columns_description.csv   37KB  2019-12-11 02:55:35
application_test.csv                 25MB  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
installments_payments.csv           690MB  2019-12-11 02:55:35
application_train.csv               158MB  2019-12-11 02:55:35
bureau_balance.csv                  358MB  2019-12-11 02:55:35
```

In [ ]:

```
DATA_DIR = "../data"   #same level as course repo in the data directory
#DATA_DIR = os.path.join('./ddddd/')
!mkdir $DATA_DIR
```

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

In [ ]:

```
!ls -l $DATA_DIR
```

```
total 704704
-rw-r--r-- 1 root root 721616255 Apr 12 00:08 home-credit-default-risk.zip
```

In [ ]:

```
! kaggle competitions download home-credit-default-risk -p $DATA_DIR
```

```
Downloading home-credit-default-risk.zip to ../data
100%|████████████████████████████████████████| 688M/688M [03:25<00:00, 2.37MB/s]
100%|████████████████████████████████████████| 688M/688M [03:26<00:00, 3.50MB/s]
```

In [ ]:

```
import zipfile
```

In [ ]:

```
unzippingReq = True
if unzippingReq: #please modify this code
    zip_ref = zipfile.ZipFile(DATA_DIR + '/home-credit-default-risk.zip', 'r')
    zip_ref.extractall('datasets')
    zip_ref.close()
```

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
```

```python
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')
```

Loading the Dataset

In [ ]:
```python
datasets = {}
```

In [ ]:
```python
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

def load_datasets():
    HOME = "/content/drive/MyDrive/Group32_AML"
    DATA_DIR = HOME + "/datasets/"
    ds_names = ("application_train", "application_test", "bureau","bureau_balance","credit_card_balance","installments_payr
                "previous_application","POS_CASH_balance")

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

load_datasets()
```

```
application_train: shape is (307511, 122)
application_test: shape is (48744, 121)
bureau: shape is (1716428, 17)
bureau_balance: shape is (27299925, 3)
credit_card_balance: shape is (3840312, 23)
installments_payments: shape is (13605401, 8)
previous_application: shape is (1670214, 37)
POS_CASH_balance: shape is (10001358, 8)
```

In [ ]:
```python
for ds_name in datasets.keys():
    print(f'dataset {ds_name:24}: [ {datasets[ds_name].shape[0]:10,}, {datasets[ds_name].shape[1]}]')
```

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

In [ ]:
```python
df_train = datasets["application_train"]
df_test = datasets["application_test"]
```

~~Make a copy of this data and use it for processing~~

In [ ]:
```python
# df_train_copy = df_train_original.copy()
# df_test_copy = df_test_original.copy()
```

In [ ]:
```python
print("\n".join(df_train.columns))
```

```
SK_ID_CURR
TARGET
NAME_CONTRACT_TYPE
CODE_GENDER
FLAG_OWN_CAR
FLAG_OWN_REALTY
CNT_CHILDREN
AMT_INCOME_TOTAL
AMT_CREDIT
AMT_ANNUITY
AMT_GOODS_PRICE
NAME_TYPE_SUITE
NAME_INCOME_TYPE
NAME_EDUCATION_TYPE
NAME_FAMILY_STATUS
NAME_HOUSING_TYPE
```

```
REGION_POPULATION_RELATIVE
DAYS_BIRTH
DAYS_EMPLOYED
DAYS_REGISTRATION
DAYS_ID_PUBLISH
OWN_CAR_AGE
FLAG_MOBIL
FLAG_EMP_PHONE
FLAG_WORK_PHONE
FLAG_CONT_MOBILE
FLAG_PHONE
FLAG_EMAIL
OCCUPATION_TYPE
CNT_FAM_MEMBERS
REGION_RATING_CLIENT
REGION_RATING_CLIENT_W_CITY
WEEKDAY_APPR_PROCESS_START
HOUR_APPR_PROCESS_START
REG_REGION_NOT_LIVE_REGION
REG_REGION_NOT_WORK_REGION
LIVE_REGION_NOT_WORK_REGION
REG_CITY_NOT_LIVE_CITY
REG_CITY_NOT_WORK_CITY
LIVE_CITY_NOT_WORK_CITY
ORGANIZATION_TYPE
EXT_SOURCE_1
EXT_SOURCE_2
EXT_SOURCE_3
APARTMENTS_AVG
BASEMENTAREA_AVG
YEARS_BEGINEXPLUATATION_AVG
YEARS_BUILD_AVG
COMMONAREA_AVG
ELEVATORS_AVG
ENTRANCES_AVG
FLOORSMAX_AVG
FLOORSMIN_AVG
LANDAREA_AVG
LIVINGAPARTMENTS_AVG
LIVINGAREA_AVG
NONLIVINGAPARTMENTS_AVG
NONLIVINGAREA_AVG
APARTMENTS_MODE
BASEMENTAREA_MODE
YEARS_BEGINEXPLUATATION_MODE
YEARS_BUILD_MODE
COMMONAREA_MODE
ELEVATORS_MODE
ENTRANCES_MODE
FLOORSMAX_MODE
FLOORSMIN_MODE
LANDAREA_MODE
LIVINGAPARTMENTS_MODE
LIVINGAREA_MODE
NONLIVINGAPARTMENTS_MODE
NONLIVINGAREA_MODE
APARTMENTS_MEDI
BASEMENTAREA_MEDI
YEARS_BEGINEXPLUATATION_MEDI
YEARS_BUILD_MEDI
COMMONAREA_MEDI
ELEVATORS_MEDI
ENTRANCES_MEDI
FLOORSMAX_MEDI
FLOORSMIN_MEDI
LANDAREA_MEDI
LIVINGAPARTMENTS_MEDI
LIVINGAREA_MEDI
NONLIVINGAPARTMENTS_MEDI
NONLIVINGAREA_MEDI
FONDKAPREMONT_MODE
HOUSETYPE_MODE
TOTALAREA_MODE
WALLSMATERIAL_MODE
EMERGENCYSTATE_MODE
OBS_30_CNT_SOCIAL_CIRCLE
DEF_30_CNT_SOCIAL_CIRCLE
OBS_60_CNT_SOCIAL_CIRCLE
DEF_60_CNT_SOCIAL_CIRCLE
DAYS_LAST_PHONE_CHANGE
FLAG_DOCUMENT_2
FLAG_DOCUMENT_3
FLAG_DOCUMENT_4
FLAG_DOCUMENT_5
FLAG_DOCUMENT_6
FLAG_DOCUMENT_7
```

```
FLAG_DOCUMENT_8
FLAG_DOCUMENT_9
FLAG_DOCUMENT_10
FLAG_DOCUMENT_11
FLAG_DOCUMENT_12
FLAG_DOCUMENT_13
FLAG_DOCUMENT_14
FLAG_DOCUMENT_15
FLAG_DOCUMENT_16
FLAG_DOCUMENT_17
FLAG_DOCUMENT_18
FLAG_DOCUMENT_19
FLAG_DOCUMENT_20
FLAG_DOCUMENT_21
AMT_REQ_CREDIT_BUREAU_HOUR
AMT_REQ_CREDIT_BUREAU_DAY
AMT_REQ_CREDIT_BUREAU_WEEK
AMT_REQ_CREDIT_BUREAU_MON
AMT_REQ_CREDIT_BUREAU_QRT
AMT_REQ_CREDIT_BUREAU_YEAR
```

Checkout the `application_train` data

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

Out[ ]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AI
0	100002	1	Cash loans	M	N	Y	0	202500.0	
1	100003	0	Cash loans	F	N	N	0	270000.0	
2	100004	0	Revolving loans	M	Y	Y	0	67500.0	
3	100006	0	Cash loans	F	N	Y	0	135000.0	
4	100007	0	Cash loans	M	N	Y	0	121500.0	

5 rows × 122 columns

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

Out[ ]:

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

8 rows × 106 columns

In [ ]:
```
df_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
```

Checkout the `application_test`

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

Out[ ]:

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDI
0	100001	Cash loans	F	N	Y	0	135000.0	568800.
1	100005	Cash loans	M	N	Y	0	99000.0	222768.

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDI
2	100013	Cash loans	M	Y	Y	0	202500.0	663264.
3	100028	Cash loans	F	N	Y	2	315000.0	1575000.
4	100038	Cash loans	M	Y	N	1	180000.0	625500.

5 rows × 121 columns

In [ ]:

```
df_test.describe()
```

Out[ ]:

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULATION_RELATIVE
count	48744.000000	48744.000000	4.874400e+04	4.874400e+04	48720.000000	4.874400e+04	48744.000000
mean	277796.676350	0.397054	1.784318e+05	5.167404e+05	29426.240209	4.626188e+05	0.021226
std	103169.547296	0.709047	1.015226e+05	3.653970e+05	16016.368315	3.367102e+05	0.014428
min	100001.000000	0.000000	2.694150e+04	4.500000e+04	2295.000000	4.500000e+04	0.000253
25%	188557.750000	0.000000	1.125000e+05	2.606400e+05	17973.000000	2.250000e+05	0.010006
50%	277549.000000	0.000000	1.575000e+05	4.500000e+05	26199.000000	3.960000e+05	0.018850
75%	367555.500000	1.000000	2.250000e+05	6.750000e+05	37390.500000	6.300000e+05	0.028663
max	456250.000000	20.000000	4.410000e+06	2.245500e+06	180576.000000	2.245500e+06	0.072508

8 rows × 105 columns

In [ ]:

```
df_test.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48744 entries, 0 to 48743
Columns: 121 entries, SK_ID_CURR to AMT_REQ_CREDIT_BUREAU_YEAR
dtypes: float64(65), int64(40), object(16)
memory usage: 45.0+ MB
```

Find and Handle Missing Data

In [ ]:

```
train_count = len(df_train)
train_count
```

Out[ ]:

```
307511
```

Find missing value counts with `isnull()`

In [ ]:

```
missing_counts = df_train.isnull().sum().sort_values(ascending=False) # * 100.0 / train_count
missing_counts
```

Out[ ]:

```
COMMONAREA_MEDI             214865
COMMONAREA_AVG              214865
COMMONAREA_MODE             214865
NONLIVINGAPARTMENTS_MODE    213514
NONLIVINGAPARTMENTS_AVG     213514
                             ...
NAME_HOUSING_TYPE                0
NAME_FAMILY_STATUS               0
NAME_EDUCATION_TYPE              0
NAME_INCOME_TYPE                 0
SK_ID_CURR                       0
Length: 122, dtype: int64
```

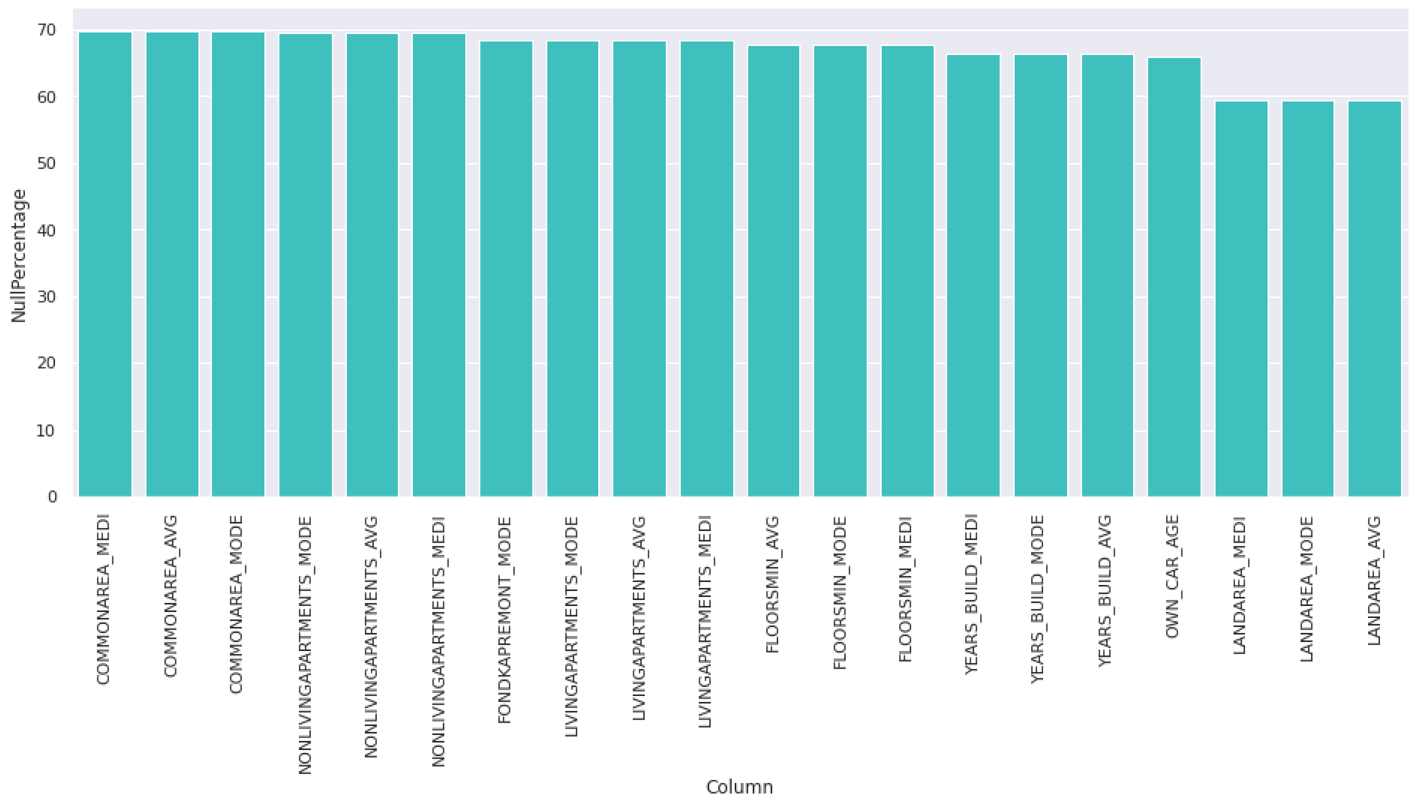
In [ ]:

```
missing_counts_percent = missing_counts * 100.0 / train_count
missing_counts_percent_top_20 = missing_counts_percent.head(20)
missing_counts_percent_top_20 = pd.DataFrame(data=zip(missing_counts_percent_top_20.index, missing_counts_percent_top_20.va
                                             columns=["Column", "NullPercentage"])

sns.set(rc={'figure.figsize':(16,6)})
missing_counts_percent_top_20_barplot = sns.barplot(x="Column", y="NullPercentage", data=missing_counts_percent_top_20,
                                                     color="aqua", saturation=.5)
missing_counts_percent_top_20_barplot.set_xticklabels(missing_counts_percent_top_20_barplot.get_xticklabels(), rotation=90)
missing_counts_percent_top_20_barplot
```

Out[ ]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x7fc30ec8f490>
```



Drop all columns that have a threshold of `missing_counts.max()`

In [ ]:
```
threshold_cutoff_non_na = train_count - missing_counts.max() / 2
```

In [ ]:
```
df_train.dropna(axis=1, thresh=threshold_cutoff_non_na, inplace=True)
```

Drop Columns

Drop zero valued columns above 70% threshold

In [ ]:
```
def drop_zeroes_threshold(df, threshold):
    num_zeroes = pd.DataFrame()

    columns = []
    percentage = []

    for col in df.columns:
        if col == 'TARGET':
            continue
        count = (df[col] == 0).sum()
        columns.append(col)
        percentage.append(count / len(df[col]))

    num_zeroes['Column'] = columns
    num_zeroes['Percentage'] = percentage
    num_zeroes = num_zeroes[num_zeroes['Percentage'] > threshold]
    return num_zeroes
```

In [ ]:
```
above_75_zeroes = drop_zeroes_threshold(df_train, 0.75)
above_75_zeroes
```

Out[ ]:

	Column	Percentage
22	FLAG_WORK_PHONE	0.800632
25	FLAG_EMAIL	0.943280
32	REG_REGION_NOT_LIVE_REGION	0.984856
33	REG_REGION_NOT_WORK_REGION	0.949231
34	LIVE_REGION_NOT_WORK_REGION	0.959341
35	REG_CITY_NOT_LIVE_CITY	0.921827
36	REG_CITY_NOT_WORK_CITY	0.769546
37	LIVE_CITY_NOT_WORK_CITY	0.820445
42	DEF_30_CNT_SOCIAL_CIRCLE	0.882323

	Column	Percentage
44	DEF_60_CNT_SOCIAL_CIRCLE	0.912881
46	FLAG_DOCUMENT_2	0.999958
48	FLAG_DOCUMENT_4	0.999919
49	FLAG_DOCUMENT_5	0.984885
50	FLAG_DOCUMENT_6	0.911945
51	FLAG_DOCUMENT_7	0.999808
52	FLAG_DOCUMENT_8	0.918624
53	FLAG_DOCUMENT_9	0.996104
54	FLAG_DOCUMENT_10	0.999977
55	FLAG_DOCUMENT_11	0.996088
56	FLAG_DOCUMENT_12	0.999993
57	FLAG_DOCUMENT_13	0.996475
58	FLAG_DOCUMENT_14	0.997064
59	FLAG_DOCUMENT_15	0.998790
60	FLAG_DOCUMENT_16	0.990072
61	FLAG_DOCUMENT_17	0.999733
62	FLAG_DOCUMENT_18	0.991870
63	FLAG_DOCUMENT_19	0.999405
64	FLAG_DOCUMENT_20	0.999493
65	FLAG_DOCUMENT_21	0.999665
66	AMT_REQ_CREDIT_BUREAU_HOUR	0.859696
67	AMT_REQ_CREDIT_BUREAU_DAY	0.860142
68	AMT_REQ_CREDIT_BUREAU_WEEK	0.837225

In [ ]:

```
df_train.drop(columns = above_75_zeroes['Column'], inplace = True)
```

Drop nan/null valued columns above 35% threshold

In [ ]:

```
def drop_nulls_threshold(df, threshold):
    num_nulls = pd.DataFrame()

    columns = []
    percentage = []

    for col in df.columns:
        if col == 'TARGET':
            continue
        count = df[col].isnull().sum()
        columns.append(col)
        percentage.append(count / len(df[col]))

    num_nulls['Column'] = columns
    num_nulls['Percentage'] = percentage
    num_nulls = num_nulls[num_nulls['Percentage'] > threshold]
    return num_nulls
```

Get all columns above 30% nulls

In [ ]:

```
above_35_nulls = drop_nulls_threshold(df_train, 0.35)
above_35_nulls
```

Out[ ]:

Column	Percentage

In [ ]:

```
df_train.drop(columns = above_35_nulls['Column'], inplace = True)
```

In [ ]:

```
df_train.describe()
```

Out[ ]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULAT

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

8 rows × 29 columns

Check and drop any rows that have values not present in test data since we won't be able to convert them properly during test

In [ ]:

```
def drop_non_existing_in_test(df_train, df_test):
    for col in df_train.columns:
        if col in df_test.columns and df_train[col].dtype == np.dtype("O"):
            unique_in_train = df_train[col].unique()
            unique_in_test = df_test[col].unique()
            for value in unique_in_train:
                if value not in unique_in_test:
                    df_train = df_train[df_train[col] != value]
    return df_train
```

In [ ]:

```
df_train = drop_non_existing_in_test(df_train, df_test)
df_train
```

Out[ ]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTA
0	100002	1	Cash loans	M	N	Y	0	20250C
1	100003	0	Cash loans	F	N	N	0	27000C
2	100004	0	Revolving loans	M	Y	Y	0	6750C
3	100006	0	Cash loans	F	N	Y	0	13500C
4	100007	0	Cash loans	M	N	Y	0	12150C
...	...	...	...	...	...	...	...	
307506	456251	0	Cash loans	M	N	N	0	15750C
307507	456252	0	Cash loans	F	N	Y	0	7200C
307508	456253	0	Cash loans	F	N	Y	0	15300C
307509	456254	1	Cash loans	F	N	Y	0	17100C
307510	456255	0	Cash loans	F	N	N	0	15750C

307500 rows × 41 columns

According to the above result a total of `11 rows` got removed from the dataset

Summarize the columns that remain

In [ ]:

```
df_train.dtypes
```

Out[ ]:

```
SK_ID_CURR                      int64
TARGET                          int64
NAME_CONTRACT_TYPE             object
CODE_GENDER                    object
FLAG_OWN_CAR                   object
FLAG_OWN_REALTY                object
CNT_CHILDREN                    int64
AMT_INCOME_TOTAL              float64
AMT_CREDIT                    float64
AMT_ANNUITY                   float64
AMT_GOODS_PRICE               float64
NAME_TYPE_SUITE                object
NAME_INCOME_TYPE               object
NAME_EDUCATION_TYPE            object
```

```
NAME_FAMILY_STATUS             object
NAME_HOUSING_TYPE              object
REGION_POPULATION_RELATIVE    float64
DAYS_BIRTH                      int64
DAYS_EMPLOYED                   int64
DAYS_REGISTRATION             float64
DAYS_ID_PUBLISH                 int64
FLAG_MOBIL                      int64
FLAG_EMP_PHONE                  int64
FLAG_CONT_MOBILE                int64
FLAG_PHONE                      int64
OCCUPATION_TYPE                object
CNT_FAM_MEMBERS               float64
REGION_RATING_CLIENT            int64
REGION_RATING_CLIENT_W_CITY     int64
WEEKDAY_APPR_PROCESS_START     object
HOUR_APPR_PROCESS_START         int64
ORGANIZATION_TYPE              object
EXT_SOURCE_2                  float64
EXT_SOURCE_3                  float64
OBS_30_CNT_SOCIAL_CIRCLE      float64
OBS_60_CNT_SOCIAL_CIRCLE      float64
DAYS_LAST_PHONE_CHANGE        float64
FLAG_DOCUMENT_3                 int64
AMT_REQ_CREDIT_BUREAU_MON     float64
AMT_REQ_CREDIT_BUREAU_QRT     float64
AMT_REQ_CREDIT_BUREAU_YEAR    float64
dtype: object
```

In [ ]:

```
df_train.describe().T
```

Out[ ]:

	count	mean	std	min	25%	50%	75%	r
SK_ID_CURR	307500.0	278181.087798	102789.822017	1.000020e+05	189146.750000	278202.500000	367143.250000	4.562550e
TARGET	307500.0	0.080725	0.272413	0.000000e+00	0.000000	0.000000	0.000000	1.000000e
CNT_CHILDREN	307500.0	0.417034	0.722108	0.000000e+00	0.000000	0.000000	1.000000	1.900000e
AMT_INCOME_TOTAL	307500.0	168797.123450	237126.307223	2.565000e+04	112500.000000	147150.000000	202500.000000	1.170000e
AMT_CREDIT	307500.0	599025.945351	402493.590146	4.500000e+04	270000.000000	513531.000000	808650.000000	4.050000e
AMT_ANNUITY	307488.0	27108.477604	14493.600189	1.615500e+03	16524.000000	24903.000000	34596.000000	2.580255e
AMT_GOODS_PRICE	307224.0	538394.285593	369445.877860	4.050000e+04	238500.000000	450000.000000	679500.000000	4.050000e
REGION_POPULATION_RELATIVE	307500.0	0.020868	0.013831	2.900000e-04	0.010006	0.018850	0.028663	7.250800e
DAYS_BIRTH	307500.0	-16037.069246	4363.988872	-2.522900e+04	-19682.000000	-15750.000000	-12413.000000	-7.489000e
DAYS_EMPLOYED	307500.0	63817.429333	141277.730537	-1.791200e+04	-2760.000000	-1213.000000	-289.000000	3.652430e
DAYS_REGISTRATION	307500.0	-4986.152449	3522.883278	-2.467200e+04	-7479.250000	-4504.000000	-2010.000000	0.000000e
DAYS_ID_PUBLISH	307500.0	-2994.203493	1509.452794	-7.197000e+03	-4299.000000	-3254.000000	-1720.000000	0.000000e
FLAG_MOBIL	307500.0	0.999997	0.001803	0.000000e+00	1.000000	1.000000	1.000000	1.000000e
FLAG_EMP_PHONE	307500.0	0.819883	0.384286	0.000000e+00	1.000000	1.000000	1.000000	1.000000e
FLAG_CONT_MOBILE	307500.0	0.998133	0.043165	0.000000e+00	1.000000	1.000000	1.000000	1.000000e
FLAG_PHONE	307500.0	0.281054	0.449514	0.000000e+00	0.000000	0.000000	1.000000	1.000000e
CNT_FAM_MEMBERS	307500.0	2.152637	0.910668	1.000000e+00	2.000000	2.000000	3.000000	2.000000e
REGION_RATING_CLIENT	307500.0	2.052462	0.509034	1.000000e+00	2.000000	2.000000	2.000000	3.000000e
REGION_RATING_CLIENT_W_CITY	307500.0	2.031519	0.502736	1.000000e+00	2.000000	2.000000	2.000000	3.000000e
HOUR_APPR_PROCESS_START	307500.0	12.063343	3.265828	0.000000e+00	10.000000	12.000000	14.000000	2.300000e
EXT_SOURCE_2	306840.0	0.514391	0.191061	8.173617e-08	0.392455	0.565956	0.663617	8.549997e
EXT_SOURCE_3	246541.0	0.510856	0.194843	5.272652e-04	0.370650	0.535276	0.669057	8.960095e
OBS_30_CNT_SOCIAL_CIRCLE	306479.0	1.422202	2.400947	0.000000e+00	0.000000	0.000000	2.000000	3.480000e
OBS_60_CNT_SOCIAL_CIRCLE	306479.0	1.405248	2.379760	0.000000e+00	0.000000	0.000000	2.000000	3.440000e
DAYS_LAST_PHONE_CHANGE	307499.0	-962.865681	826.813694	-4.292000e+03	-1570.000000	-757.000000	-274.000000	0.000000e
FLAG_DOCUMENT_3	307500.0	0.710049	0.453740	0.000000e+00	0.000000	1.000000	1.000000	1.000000e
AMT_REQ_CREDIT_BUREAU_MON	265986.0	0.267390	0.915997	0.000000e+00	0.000000	0.000000	0.000000	2.700000e
AMT_REQ_CREDIT_BUREAU_QRT	265986.0	0.265476	0.794062	0.000000e+00	0.000000	0.000000	0.000000	2.610000e
AMT_REQ_CREDIT_BUREAU_YEAR	265986.0	1.899961	1.869288	0.000000e+00	0.000000	1.000000	3.000000	2.500000e

EXPLORATORY DATA ANALYSIS + FEATURE ENGINEERING

What is the distribution of loan applications according to gender?

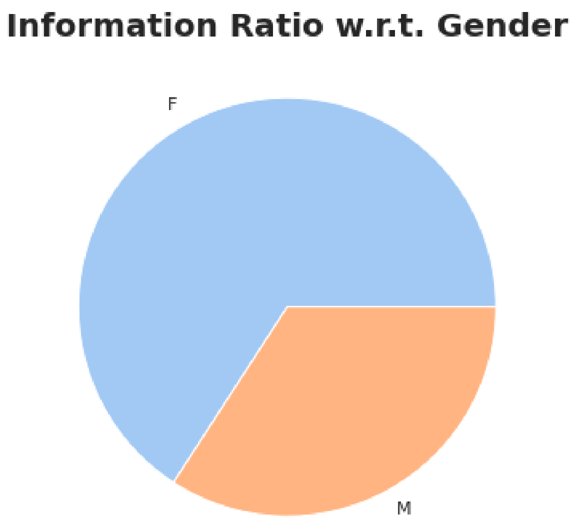
In [ ]:

```
df_gender = df_train["CODE_GENDER"]
df_gender_labels = df_gender.unique()
df_gender_labels.sort()
print(df_gender.value_counts())
plt.pie(df_gender.value_counts(), colors=sns.color_palette("pastel"), labels=df_gender_labels)
plt.title("Information Ratio w.r.t. Gender", fontweight="bold", fontsize="20")
```

```
F    202443
M    105057
Name: CODE_GENDER, dtype: int64
```

Out[ ]:

```
Text(0.5, 1.0, 'Information Ratio w.r.t. Gender')
```



Inference

The number of females applying for a loan is much higher than that of men according to the pie chart above.

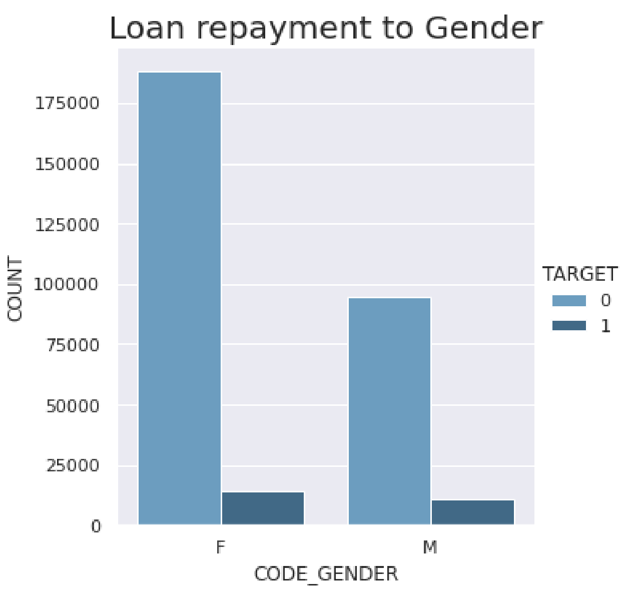
What is the distribution of loan repayment according to gender?

In [ ]:

```
af = df_train[["CODE_GENDER", "TARGET"]]
af = af.value_counts()
af = pd.DataFrame(data=[(x[0], x[1], af[x]) for x in af.index], columns=["CODE_GENDER", "TARGET", "COUNT"])
sns.catplot(x="CODE_GENDER", y="COUNT", hue="TARGET", data=af, kind="bar", palette="Blues_d", height=5, aspect=1)
plt.title("Loan repayment to Gender", fontsize="20")
```

Out[ ]:

```
Text(0.5, 1.0, 'Loan repayment to Gender')
```



In [ ]:

```
df_target_f = af[af["CODE_GENDER"] == "F"]
df_target_m = af[af["CODE_GENDER"] == "M"]
```

```
per_f_repaid = df_target_f[df_target_f["TARGET"] == 0]["COUNT"] / df_target_f["COUNT"].sum()
per_m_repaid = df_target_m[df_target_m["TARGET"] == 0]["COUNT"] / df_target_m["COUNT"].sum()
print("Percentage of females who repaid the loan =", np.round(per_f_repaid[0] * 100.0, 3))
print("Percentage of males   who repaid the loan =", np.round(per_m_repaid[1] * 100.0, 3))
```

```
Percentage of females who repaid the loan = 93.001
Percentage of males   who repaid the loan = 89.858
```

Inference

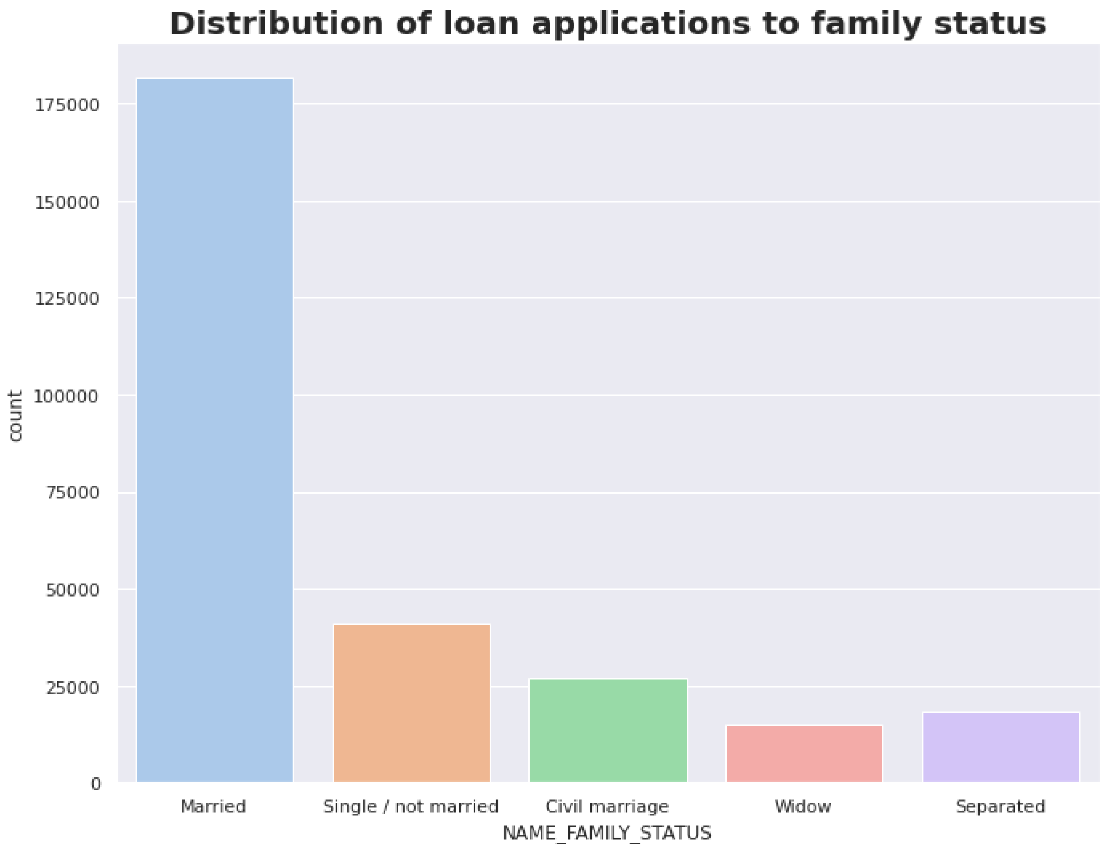
The number of females repaying the loan is higher than the number of males.

What is the distribution of loan applications to family status?

In [ ]:

```
df_family_status = df_train[df_train["TARGET"] == 0]["NAME_FAMILY_STATUS"]
df_family_status_labels = df_family_status.unique()
df_family_status_labels.sort()
print(df_family_status.value_counts())
plt.figure(figsize=(11, 8.5))
sns.countplot(df_family_status, palette="pastel")
_ = plt.title("Distribution of loan applications to family status", fontweight="bold", fontsize="20")
```

```
Married                 181576
Single / not married     40987
Civil marriage           26813
Separated                18150
Widow                    15151
Name: NAME_FAMILY_STATUS, dtype: int64
```



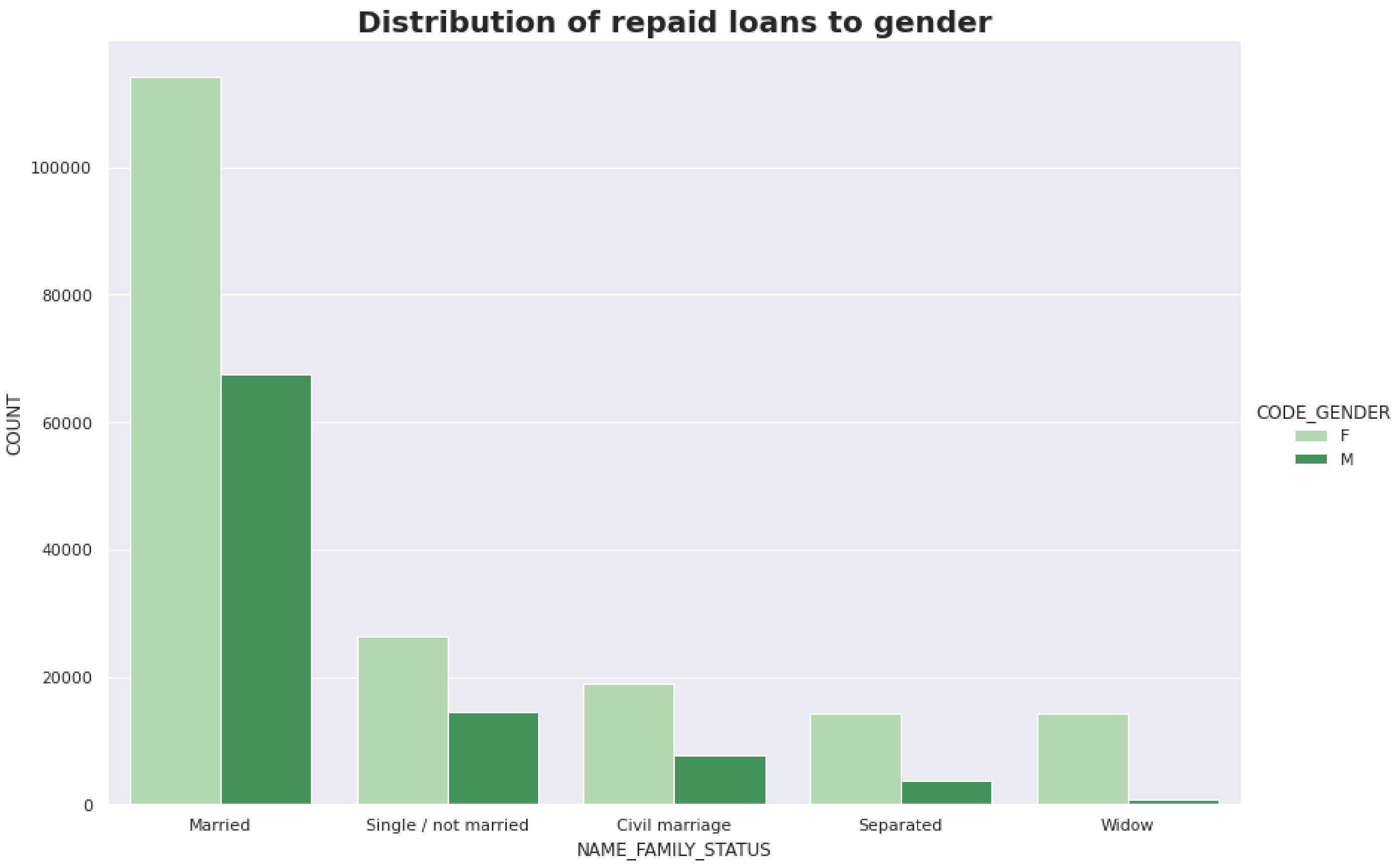
Inference

The majority of clients who are married have repaid the loan.

What is the distribution of loan applications to family status by gender?

In [ ]:

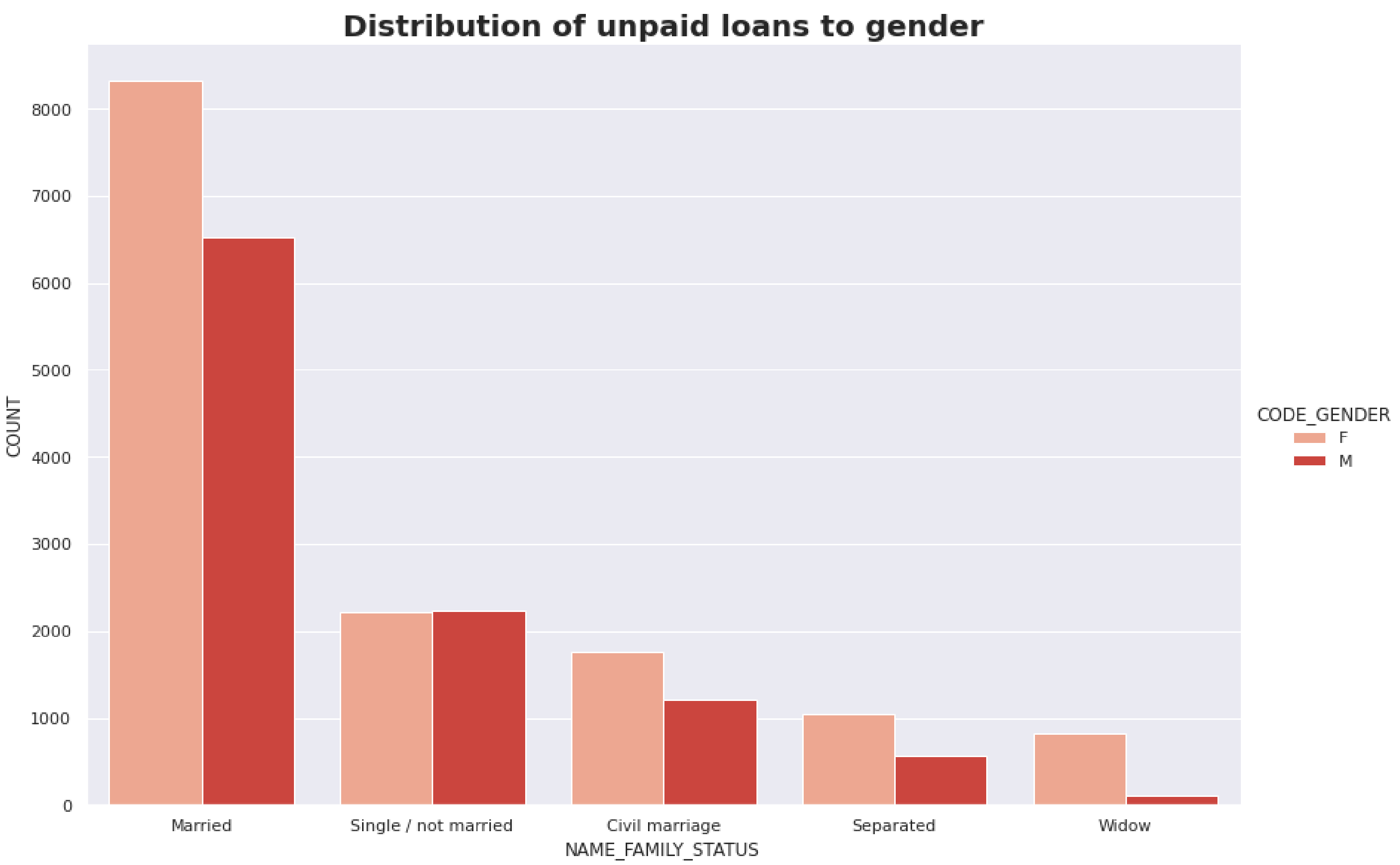
```
df_family_status_gender = df_train[df_train["TARGET"] == 0][["NAME_FAMILY_STATUS", "CODE_GENDER"]]
df_family_status_gender = df_family_status_gender.value_counts()
df_family_status_gender = pd.DataFrame(data=[(x[0], x[1], df_family_status_gender[x]) for x in df_family_status_gender.inde
sns.catplot(x="NAME_FAMILY_STATUS", y="COUNT", hue="CODE_GENDER", data=df_family_status_gender, kind="bar", palette="Greens
_ = plt.title("Distribution of repaid loans to gender", fontweight="bold", fontsize="20")
```



Inference

Under each family status, the females have the highest repayment percentage.

In [ ]:
```
df_family_status_gender = df_train[df_train["TARGET"] == 1][["NAME_FAMILY_STATUS", "CODE_GENDER"]]
df_family_status_gender = df_family_status_gender.value_counts()
df_family_status_gender = pd.DataFrame(data=[(x[0], x[1], df_family_status_gender[x]) for x in df_family_status_gender.inde
sns.catplot(x="NAME_FAMILY_STATUS", y="COUNT", hue="CODE_GENDER", data=df_family_status_gender, kind="bar", palette="Reds",
_ = plt.title("Distribution of unpaid loans to gender", fontweight="bold", fontsize="20")
```



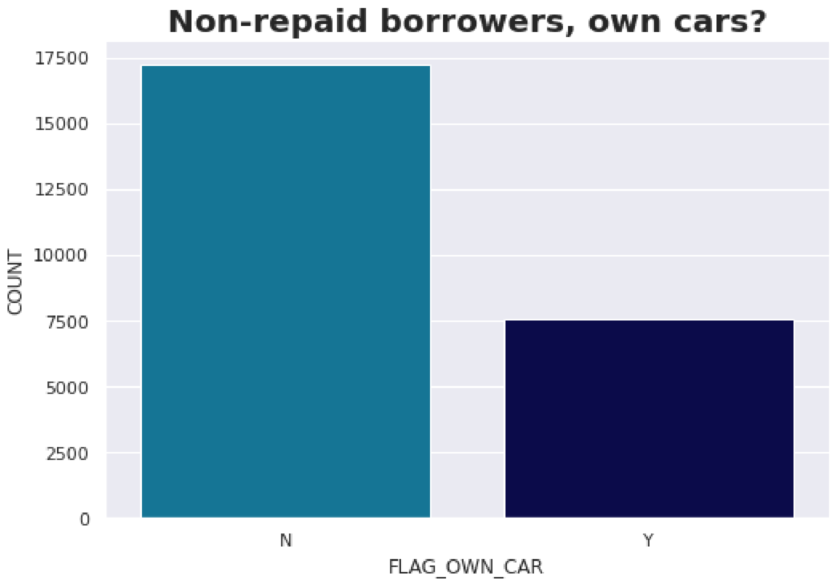
Inference

Under each family status, the females have the highest repayment percentage.

How many percent of borrowers who haven't repaid, own cars?

In [ ]:

```
af = df_train[df_train["TARGET"] == 1]["FLAG_OWN_CAR"].value_counts()
af = pd.DataFrame(data=[(x[0], af[x]) for x in af.index], columns=["FLAG_OWN_CAR", "COUNT"])
sns.catplot(x="FLAG_OWN_CAR", y="COUNT", data=af, kind="bar", palette="ocean_r", height=5, aspect=1.5)
_ = plt.title("Non-repaid borrowers, own cars?", fontweight="bold", fontsize="20")
```



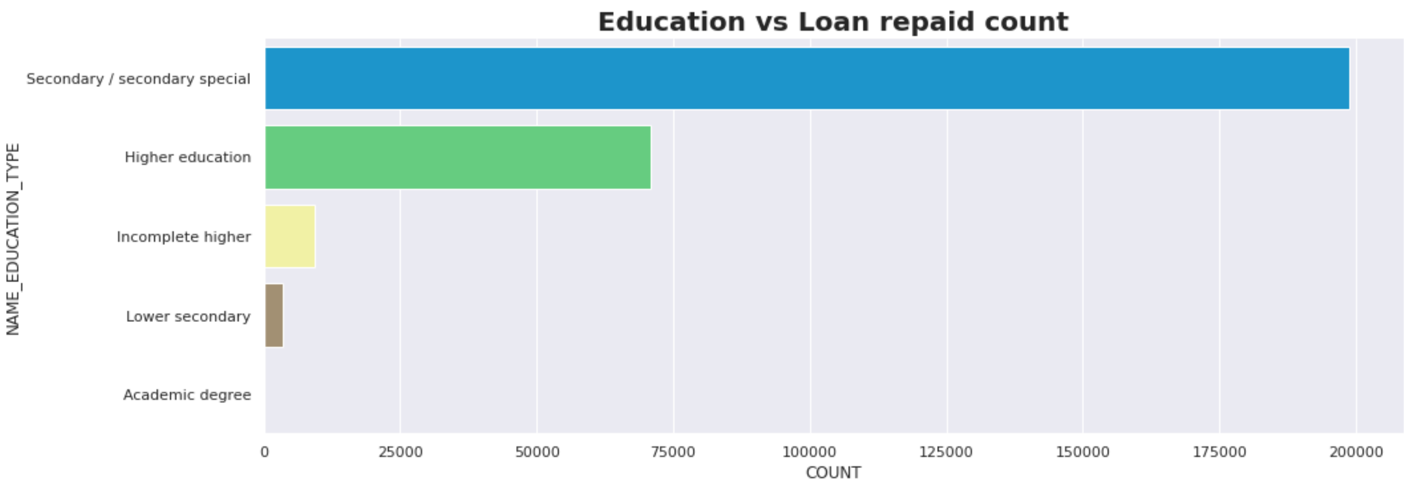
Inference

Out of 25000 people who have not repaid the loan, 70% of people do not own a car!

Do educated borrowers pay the loans?

In [ ]:

```
af = df_train[df_train["TARGET"] == 0]["NAME_EDUCATION_TYPE"].value_counts()
af = pd.DataFrame(data=[(x, af[x]) for x in af.index], columns=["NAME_EDUCATION_TYPE", "COUNT"])
sns.catplot(y="NAME_EDUCATION_TYPE", x="COUNT", data=af, kind="bar", palette="terrain", height=5, aspect=3)
_ = plt.title("Education vs Loan repaid count", fontweight="bold", fontsize="20")
```



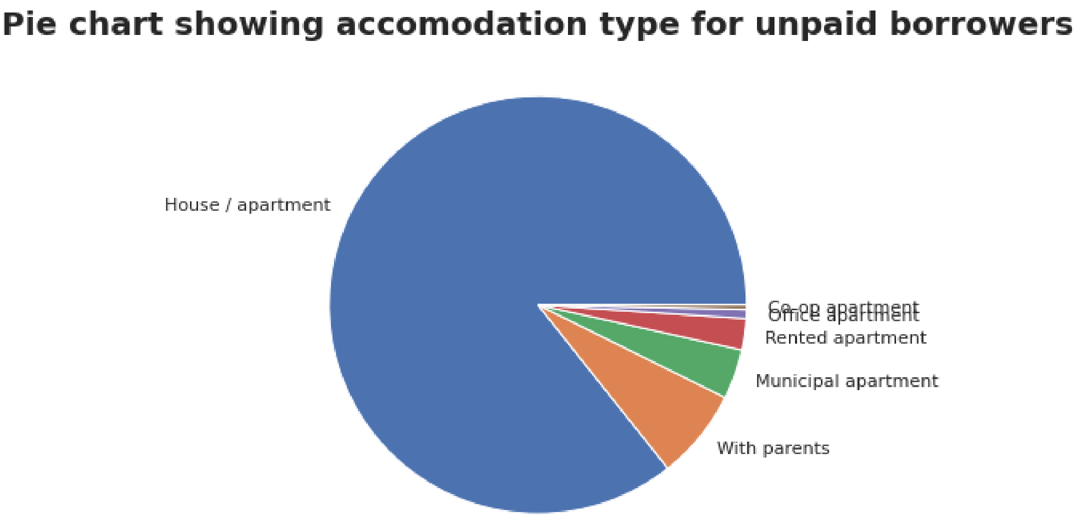
Inference

Clients with a secondary/special degree, are more likely to repay the loans compared to the others.

Where do clients having unpaid loans live?

In [ ]:

```
af = df_train[df_train["TARGET"] == 1]["NAME_HOUSING_TYPE"].value_counts()
plt.pie(af, labels=af.index)
_ = plt.title("Pie chart showing accomodation type for unpaid borrowers", fontweight="bold", fontsize="20")
```



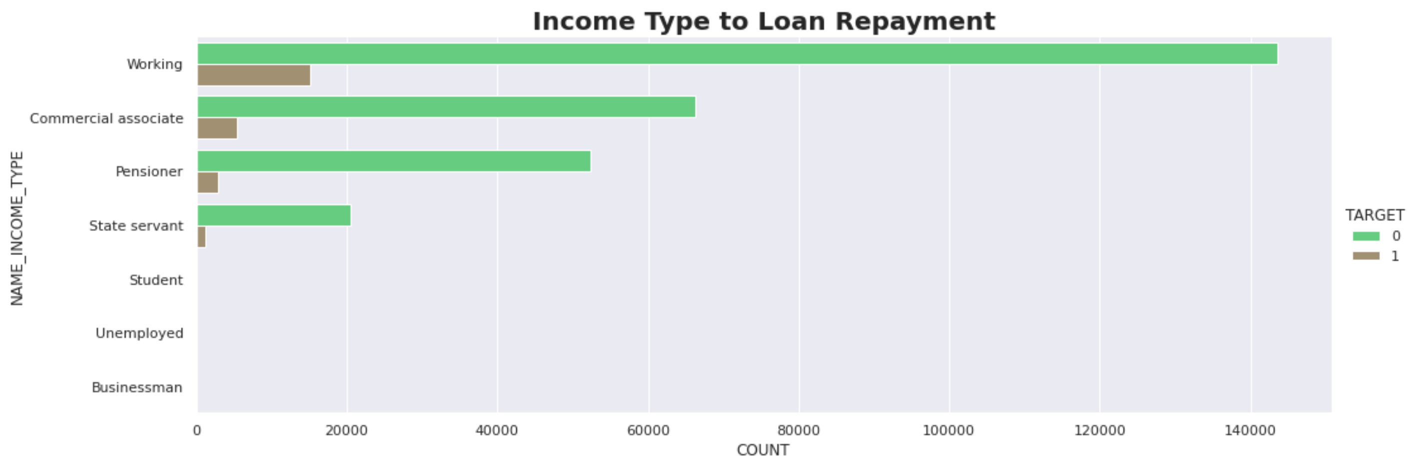
Inference

Although the loans are not repaid, maximun number of people still live in their own houses/apartments.

Finding the income type relation to loan target code

In [ ]:

```
af = df_train[["NAME_INCOME_TYPE", "TARGET"]].value_counts()
af = pd.DataFrame(data=[(x[0], x[1], af[x]) for x in af.index], columns=["NAME_INCOME_TYPE", "TARGET", "COUNT"])
sns.catplot(y="NAME_INCOME_TYPE", x="COUNT", hue="TARGET", data=af, kind="bar", palette="terrain", height=5, aspect=3)
_ = plt.title("Income Type to Loan Repayment", fontweight="bold", fontsize="20")
```



Inference

According to bar plot above borrowers with working income type have repaid the most loans.

Columns

In [ ]:

```
df_train.columns
```

Out[ ]:

```
Index(['SK_ID_CURR', 'TARGET', 'NAME_CONTRACT_TYPE', 'CODE_GENDER',
       'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL',
       'AMT_CREDIT', 'AMT_ANNUITY', 'AMT_GOODS_PRICE', 'NAME_TYPE_SUITE',
       'NAME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS',
       'NAME_HOUSING_TYPE', 'REGION_POPULATION_RELATIVE', 'DAYS_BIRTH',
       'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'DAYS_ID_PUBLISH', 'FLAG_MOBIL',
       'FLAG_EMP_PHONE', 'FLAG_CONT_MOBILE', 'FLAG_PHONE', 'OCCUPATION_TYPE',
       'CNT_FAM_MEMBERS', 'REGION_RATING_CLIENT',
       'REGION_RATING_CLIENT_W_CITY', 'WEEKDAY_APPR_PROCESS_START',
       'HOUR_APPR_PROCESS_START', 'ORGANIZATION_TYPE', 'EXT_SOURCE_2',
       'EXT_SOURCE_3', 'OBS_30_CNT_SOCIAL_CIRCLE', 'OBS_60_CNT_SOCIAL_CIRCLE',
       'DAYS_LAST_PHONE_CHANGE', 'FLAG_DOCUMENT_3',
       'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QRT',
       'AMT_REQ_CREDIT_BUREAU_YEAR'],
      dtype='object')
```

In [ ]:

```
columns_to_dummy = [col for col in df_train.columns if df_train.dtypes[col] == np.dtype("O")]
```

```
df_temp_dummies = pd.get_dummies(df_train, columns=columns_to_dummy)
df_temp_dummies
```

Out[ ]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULATION_RE
0	100002	1	0	202500.0	406597.5	24700.5	351000.0	0
1	100003	0	0	270000.0	1293502.5	35698.5	1129500.0	0
2	100004	0	0	67500.0	135000.0	6750.0	135000.0	0
3	100006	0	0	135000.0	312682.5	29686.5	297000.0	0
4	100007	0	0	121500.0	513000.0	21865.5	513000.0	0
...	...	...	...	...	...	...	...	
307506	456251	0	0	157500.0	254700.0	27558.0	225000.0	0
307507	456252	0	0	72000.0	269550.0	12001.5	225000.0	0
307508	456253	0	0	153000.0	677664.0	29979.0	585000.0	0
307509	456254	1	0	171000.0	370107.0	20205.0	319500.0	0
307510	456255	0	0	157500.0	675000.0	49117.5	675000.0	0

307500 rows × 150 columns

Remove nulls

Find the columns with non-numerical values and still having null values and encode it

In [ ]:

```
columns_nullcount_types = pd.DataFrame(data=zip(df_temp_dummies.columns, df_temp_dummies.isnull().sum(), df_temp_dummies.d
                                       columns=["Column", "NullCount", "Type"])
columns_nullcount = columns_nullcount_types[columns_nullcount_types["NullCount"] > 0]
columns_nullcount
```

Out[ ]:

	Column	NullCount	Type
5	AMT_ANNUITY	12	float64
6	AMT_GOODS_PRICE	276	float64
20	EXT_SOURCE_2	660	float64
21	EXT_SOURCE_3	60959	float64
22	OBS_30_CNT_SOCIAL_CIRCLE	1021	float64
23	OBS_60_CNT_SOCIAL_CIRCLE	1021	float64
24	DAYS_LAST_PHONE_CHANGE	1	float64
26	AMT_REQ_CREDIT_BUREAU_MON	41514	float64
27	AMT_REQ_CREDIT_BUREAU_QRT	41514	float64
28	AMT_REQ_CREDIT_BUREAU_YEAR	41514	float64

~~Converting categorical data into numerical forms for all such columns to convert~~

In [ ]:

```
# columns_to_convert = ["NAME_TYPE_SUITE"]

# for column in columns_to_convert:
#     unique_name_type_suite = [x for x in df_train[column].unique() if str(x) != 'nan']
#     df_train[column].replace(unique_name_type_suite, list(range(len(unique_name_type_suite))), inplace=True)

# df_train["NAME_TYPE_SUITE"].unique()
```

In [ ]:

```
df_train_interpolated = df_temp_dummies.interpolate(method='cubic', limit_direction="forward")
df_train_interpolated
```

Out[ ]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULATION_RE
0	100002	1	0	202500.0	406597.5	24700.5	351000.0	0
1	100003	0	0	270000.0	1293502.5	35698.5	1129500.0	0
2	100004	0	0	67500.0	135000.0	6750.0	135000.0	0
3	100006	0	0	135000.0	312682.5	29686.5	297000.0	0

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULATION_RE
4	100007	0	0	121500.0	513000.0	21865.5	513000.0	0
...	...	...	...	...	...	...	...	
307506	456251	0	0	157500.0	254700.0	27558.0	225000.0	0
307507	456252	0	0	72000.0	269550.0	12001.5	225000.0	0
307508	456253	0	0	153000.0	677664.0	29979.0	585000.0	0
307509	456254	1	0	171000.0	370107.0	20205.0	319500.0	0
307510	456255	0	0	157500.0	675000.0	49117.5	675000.0	0

307500 rows × 150 columns

In [ ]:

```
df_train_interpolated.describe()
```

Out[ ]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULAT
count	307500.000000	307500.000000	307500.000000	3.075000e+05	3.075000e+05	307500.000000	3.075000e+05	
mean	278181.087798	0.080725	0.417034	1.687971e+05	5.990259e+05	27108.650858	5.383484e+05	
std	102789.822017	0.272413	0.722108	2.371263e+05	4.024936e+05	14493.662612	3.694725e+05	
min	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04	1407.291066	-5.349644e+05	
25%	189146.750000	0.000000	0.000000	1.125000e+05	2.700000e+05	16524.000000	2.385000e+05	
50%	278202.500000	0.000000	0.000000	1.471500e+05	5.135310e+05	24903.000000	4.500000e+05	
75%	367143.250000	0.000000	1.000000	2.025000e+05	8.086500e+05	34596.000000	6.795000e+05	
max	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06	258025.500000	4.050000e+06	

8 rows × 150 columns

In [ ]:

```
df_train_interpolated.isnull().sum()
```

Out[ ]:

```
SK_ID_CURR                          0
TARGET                              0
CNT_CHILDREN                        0
AMT_INCOME_TOTAL                    0
AMT_CREDIT                          0
                                   ..
ORGANIZATION_TYPE_Transport: type 2    0
ORGANIZATION_TYPE_Transport: type 3    0
ORGANIZATION_TYPE_Transport: type 4    0
ORGANIZATION_TYPE_University           0
ORGANIZATION_TYPE_XNA                  0
Length: 150, dtype: int64
```

In [ ]:

```
dumm_corr = df_train_interpolated.corr().abs()
dumm_corr
```

Out[ ]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE
SK_ID_CURR	1.000000	0.002137	0.001140	0.001808	0.000346	0.000438	0.000267
TARGET	0.002137	1.000000	0.019143	0.003970	0.030390	0.012817	0.039610
CNT_CHILDREN	0.001140	0.019143	1.000000	0.012897	0.002139	0.021378	0.001845
AMT_INCOME_TOTAL	0.001808	0.003970	0.012897	1.000000	0.156873	0.191644	0.159536
AMT_CREDIT	0.000346	0.030390	0.002139	0.156873	1.000000	0.770101	0.986120
...	...	...	...	...	...	...	...
ORGANIZATION_TYPE_Transport: type 2	0.002597	0.000968	0.020947	0.002397	0.000053	0.001084	0.000825
ORGANIZATION_TYPE_Transport: type 3	0.000670	0.017553	0.004211	0.001778	0.009990	0.001484	0.010514
ORGANIZATION_TYPE_Transport: type 4	0.002658	0.005931	0.011963	0.012324	0.012154	0.021421	0.011763

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE
ORGANIZATION_TYPE_University	0.000496	0.007671	0.000852	0.005098	0.017552	0.015298	0.017220
ORGANIZATION_TYPE_XNA	0.001365	0.045983	0.240720	0.064037	0.065595	0.103611	0.063400

150 rows × 150 columns

In [ ]:

```
dumm_corr_target = dumm_corr["TARGET"]
dumm_corr_target_top_16 = dumm_corr_target.sort_values(ascending=False).head(16).index
dumm_corr_target_top_16
```

Out[ ]:

```
Index(['TARGET', 'EXT_SOURCE_2', 'EXT_SOURCE_3', 'DAYS_BIRTH',
       'REGION_RATING_CLIENT_W_CITY', 'REGION_RATING_CLIENT',
       'NAME_INCOME_TYPE_Working', 'NAME_EDUCATION_TYPE_Higher education',
       'DAYS_LAST_PHONE_CHANGE', 'CODE_GENDER_M', 'CODE_GENDER_F',
       'DAYS_ID_PUBLISH', 'NAME_EDUCATION_TYPE_Secondary / secondary special',
       'NAME_INCOME_TYPE_Pensioner', 'ORGANIZATION_TYPE_XNA',
       'FLAG_EMP_PHONE'],
      dtype='object')
```

~~Top 10 Feature Correlations Heatmap~~

~~Get the top 11 features having highest correlation with `TARGET` (including `TARGET` itself) and plot the heatmap to find the linear relationships between these variables.~~

In [ ]:

```
# def cartesian_product(*arrays):
#     la = len(arrays)
#     dtype = np.result_type(*arrays)
#     arr = np.empty([len(a) for a in arrays] + [la], dtype=dtype)
#     for i, a in enumerate(np.ix_(*arrays)):
#         arr[...,i] = a
#     return arr.reshape(-1, la)

# dumm_corr_target_top_16_pairs = cartesian_product(dumm_corr_target_top_16, dumm_corr_target_top_16)
# dumm_corr_target_top_16_matrix = np.zeros((len(dumm_corr_target_top_16), len(dumm_corr_target_top_16)))
# for i in range(len(dumm_corr_target_top_16)):
#     for j in range(len(dumm_corr_target_top_16)):
#         r = dumm_corr_target_top_16[i]
#         c = dumm_corr_target_top_16[j]
#         dumm_corr_target_top_16_matrix[i][j] = dumm_corr[r][c]
```

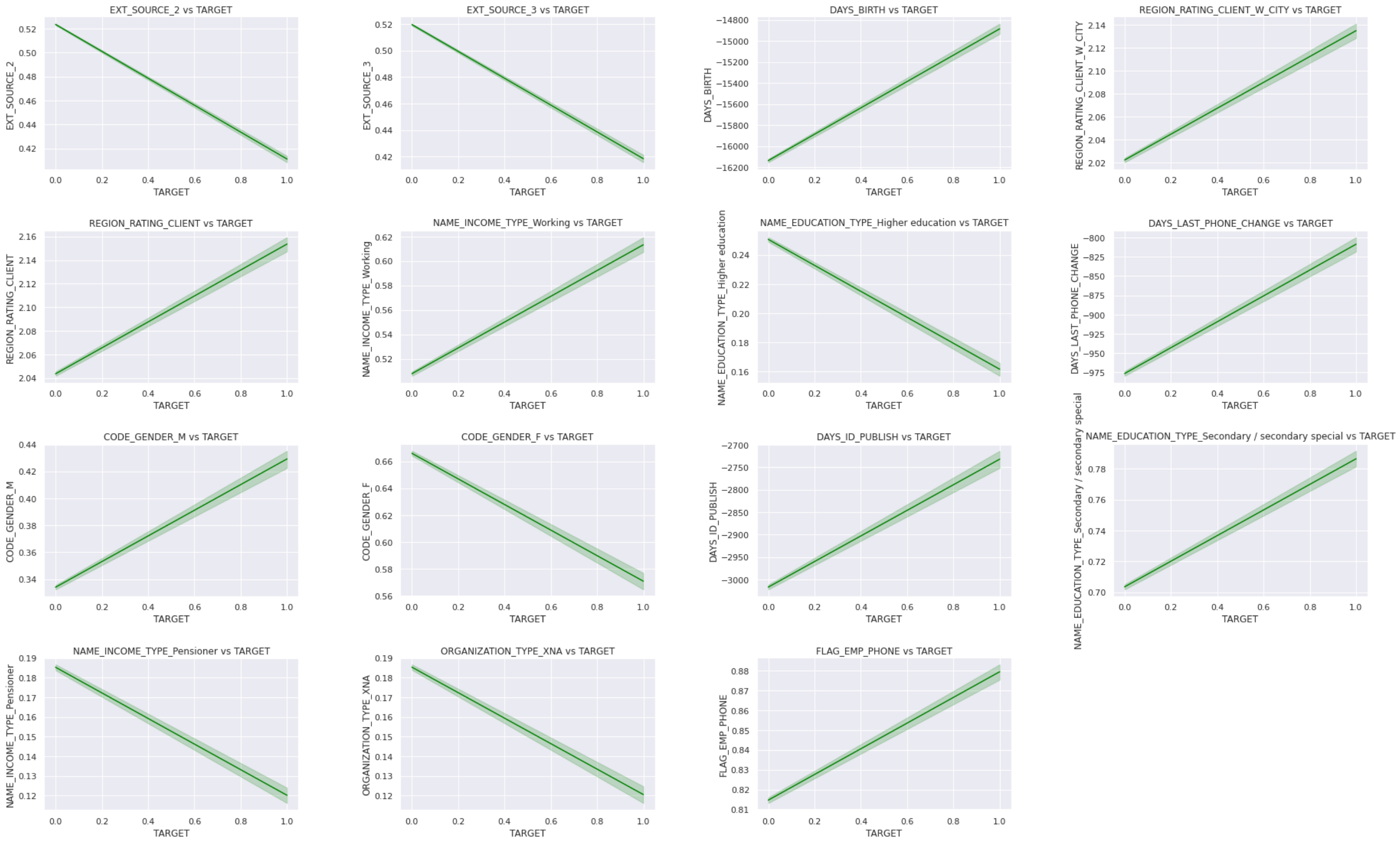
In [ ]:

```
# triangle_mask = np.zeros_like(dumm_corr_target_top_16_matrix, dtype=np.bool_)
# triangle_mask[np.triu_indices_from(triangle_mask)] = True
# func, axes = plt.subplots(figsize=(10, 10))
# sns.heatmap(dumm_corr_target_top_16_matrix, mask=triangle_mask, cmap="vlag", square=True, linewidths=1.0, ax=axes,
#             xticklabels=dumm_corr_target_top_16, yticklabels=dumm_corr_target_top_16)
# _ = plt.title("Top 15 feature correlations with TARGET", fontweight="bold", fontsize="20")
```

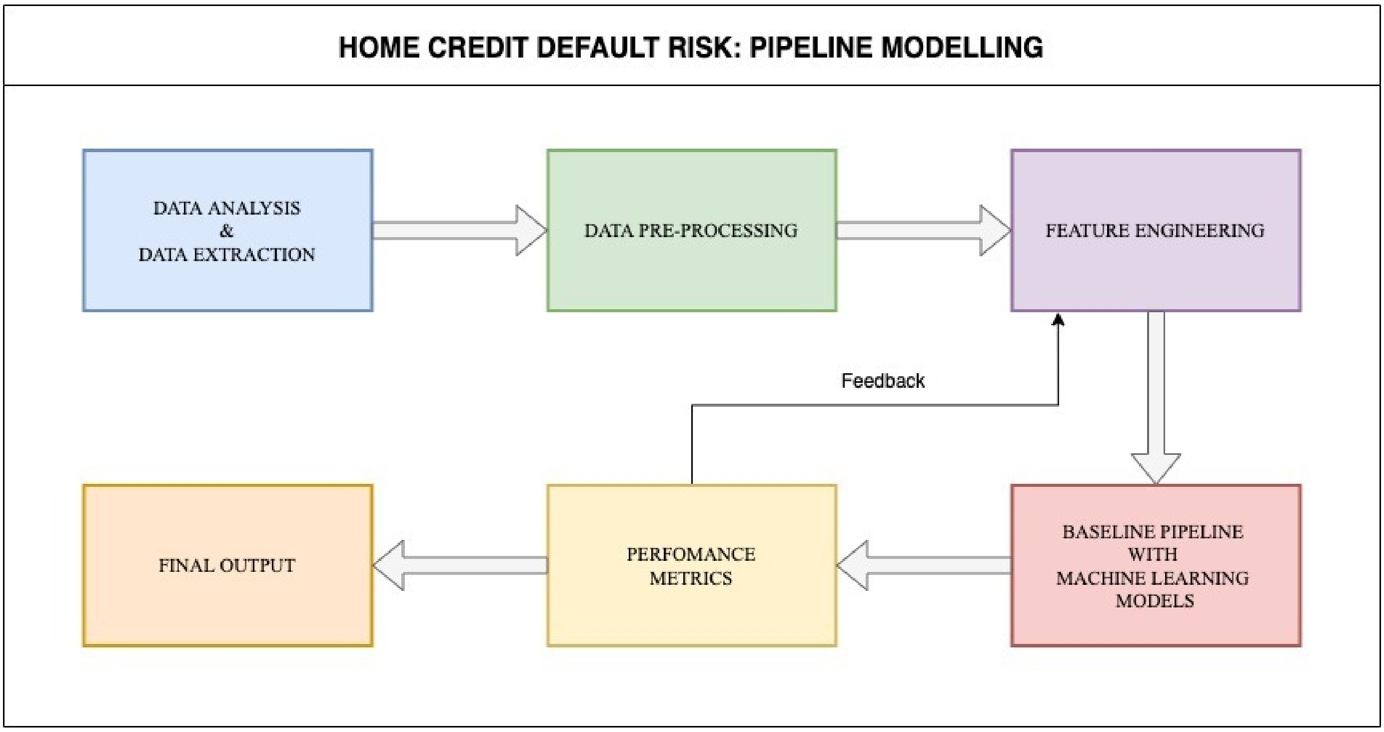
Plot each highly correlated feature against `TARGET`

In [ ]:

```
func, ax = plt.subplots(4, 4, figsize=(30, 18), facecolor='w', edgecolor='k')
plt.subplots_adjust(left=0.1, bottom=0.1, right=0.9, top=0.9, wspace=0.4, hspace=0.4)
for i in range(1, len(dumm_corr_target_top_16)):
    feature = dumm_corr_target_top_16[i]
    curr_ax = ax[(i - 1) // 4][(i - 1) % 4]
    sns.lineplot(x='TARGET', y=feature, data=df_train_interpolated, color="green", ax=curr_ax)
    curr_ax.title.set_text(feature + " vs TARGET")
ax[-1][-1].set_visible(False)
```



Modeling Pipelines



In [ ]:

```
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import OneHotEncoder
from sklearn.compose import ColumnTransformer, make_column_transformer
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
```

In [ ]:

```
df_train_interpolated_X = df_train_interpolated.copy()
df_train_interpolated_X.drop(columns = ['TARGET'], inplace=True)
```

In [ ]:

```
# Create a basic numerical pipeline
num_pipeline = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean'))
])
```

```python
# Column transform pipeline
data_pipeline = ColumnTransformer([
    ("num_pipeline", num_pipeline, df_train_interpolated_X.columns)
], n_jobs = -1)

# Run the transform
data_transformed = data_pipeline.fit_transform(df_train_interpolated_X)
```

Make the transformed data into a DataFrame

In [ ]:

```python
df_transformed = pd.DataFrame(data_transformed,  columns=df_train_interpolated_X.columns)
```

Check the DataFrame

In [ ]:

```python
df_transformed
```

Out[ ]:

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULATION_RELATIVE
0	100002.0	0.0	202500.0	406597.5	24700.5	351000.0	0.018801
1	100003.0	0.0	270000.0	1293502.5	35698.5	1129500.0	0.003541
2	100004.0	0.0	67500.0	135000.0	6750.0	135000.0	0.010032
3	100006.0	0.0	135000.0	312682.5	29686.5	297000.0	0.008019
4	100007.0	0.0	121500.0	513000.0	21865.5	513000.0	0.028663
...	...	...	...	...	...	...	...
307495	456251.0	0.0	157500.0	254700.0	27558.0	225000.0	0.032561
307496	456252.0	0.0	72000.0	269550.0	12001.5	225000.0	0.025164
307497	456253.0	0.0	153000.0	677664.0	29979.0	585000.0	0.005002
307498	456254.0	0.0	171000.0	370107.0	20205.0	319500.0	0.005313
307499	456255.0	0.0	157500.0	675000.0	49117.5	675000.0	0.046220

307500 rows × 149 columns

In [ ]:

```python
df_transformed.describe()
```

Out[ ]:

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	REGION_POPULATION_RELATIVE
count	307500.000000	307500.000000	3.075000e+05	3.075000e+05	307500.000000	3.075000e+05	307500.000000
mean	278181.087798	0.417034	1.687971e+05	5.990259e+05	27108.650858	5.383484e+05	0.020868
std	102789.822017	0.722108	2.371263e+05	4.024936e+05	14493.662612	3.694725e+05	0.013831
min	100002.000000	0.000000	2.565000e+04	4.500000e+04	1407.291066	-5.349644e+05	0.000290
25%	189146.750000	0.000000	1.125000e+05	2.700000e+05	16524.000000	2.385000e+05	0.010006
50%	278202.500000	0.000000	1.471500e+05	5.135310e+05	24903.000000	4.500000e+05	0.018850
75%	367143.250000	1.000000	2.025000e+05	8.086500e+05	34596.000000	6.795000e+05	0.028663
max	456255.000000	19.000000	1.170000e+08	4.050000e+06	258025.500000	4.050000e+06	0.072508

8 rows × 149 columns

Results of Pipeline

In [ ]:

```python
from sklearn.metrics import log_loss
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
from sklearn import metrics
```

In [ ]:

```python
model_results = []
X = df_transformed.values
```

```
y = df_train_interpolated["TARGET"].values
print("Shape of X", X.shape)
print("Shape of y", y.shape)
```

```
Shape of X (307500, 149)
Shape of y (307500,)
```

In [ ]:

```
X_train, X_validation, y_train, y_validation = train_test_split(X, y, test_size=0.25, random_state=42)
```

LogisticRegression

In [ ]:

```
pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', LogisticRegression(solver='lbfgs', max_iter=1000))
])
pipe.fit(X_train, y_train)
print("Accuracy score (training):", str(pipe.score(X_train, y_train)))
```

```
Accuracy score (training): 0.9195924119241192
```

In [ ]:

```
y_pred = pipe.predict(X_validation)
print("Accuracy score (validation):", str(accuracy_score(y_validation, y_pred)))
print("Log loss:", log_loss(y_validation, y_pred))
print("Confusion Matrix:", "\n", confusion_matrix(y_validation, y_pred))
print("ROC_AUC:", roc_auc_score(y_validation, pipe.predict_proba(X_validation)[:, 1]))
```

```
Accuracy score (validation): 0.9179186991869919
Log loss: 2.8349881942447235
Confusion Matrix:
 [[70530    48]
 [ 6262    35]]
ROC_AUC: 0.7257233296392955
```

In [ ]:

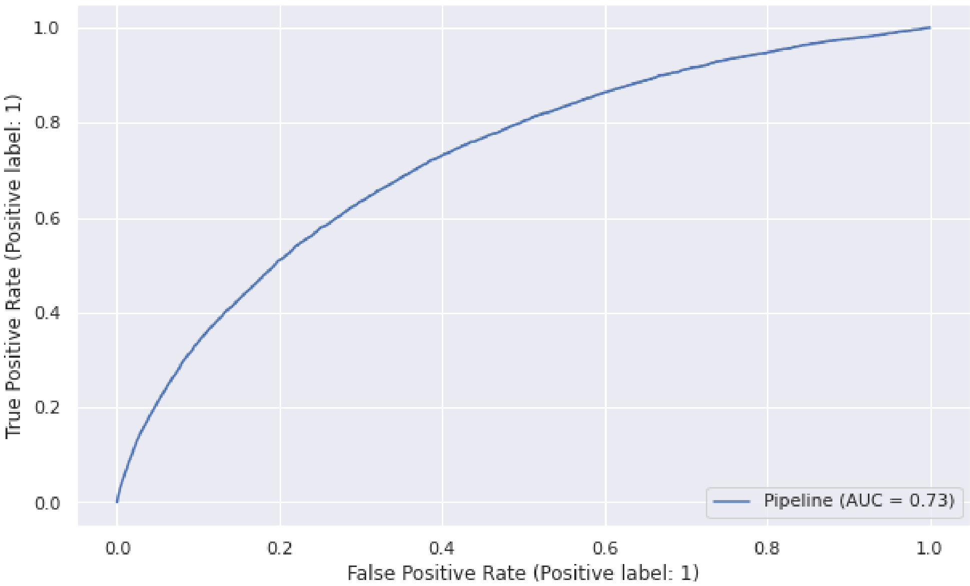
```
model_results.append(["Logistic Regression", pipe.score(X_train, y_train), accuracy_score(y_validation, y_pred)])
```

In [ ]:

```
func, ax = plt.subplots(1, 1, figsize=(10, 6))
metrics.plot_roc_curve(pipe, X_validation, y_validation, ax=ax)
```

Out[ ]:

```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7fc30e61ed50>
```



Naive Bayes (Gaussian)

In [ ]:

```
pipe_naive_bayes = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', GaussianNB())
])

pipe_naive_bayes.fit(X_train, y_train)
print("Accuracy score (training):", str(pipe_naive_bayes.score(X_train, y_train)))
```

```
Accuracy score (training): 0.13572249322493224
```

In [ ]:

```
y_pred = pipe_naive_bayes.predict(X_validation)
print("Accuracy score (validation):", str(accuracy_score(y_validation, y_pred)))
print("Log loss:", log_loss(y_validation, y_pred))
```

```
print("Confusion Matrix:", "\n", confusion_matrix(y_validation, y_pred))
print("ROC_AUC:", roc_auc_score(y_validation, pipe_naive_bayes.predict_proba(X_validation)[:, 1]))
```

```
Accuracy score (validation): 0.13668943089430893
Log loss: 29.81837815666981
Confusion Matrix:
 [[ 4485 66093]
 [  274  6023]]
ROC_AUC: 0.6180994159827307
```

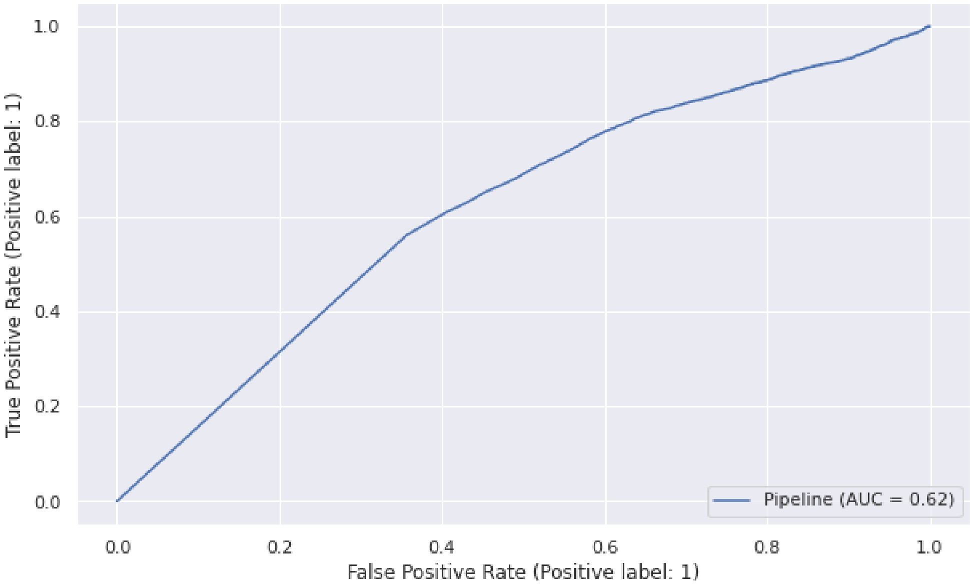
In [ ]:

```
model_results.append(["Naive Bayes (Gaussian)", pipe_naive_bayes.score(X_train, y_train), accuracy_score(y_validation, y_pı
```

In [ ]:

```
func, ax = plt.subplots(1, 1, figsize=(10, 6))
metrics.plot_roc_curve(pipe_naive_bayes, X_validation, y_validation, ax=ax)
```

Out[ ]: `<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7fc30bcf8110>`



Random Forest

In [ ]:

```
pipe_rf = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', RandomForestClassifier())
])

pipe_rf.fit(X_train, y_train)
print("Accuracy score (training):", str(pipe_rf.score(X_train, y_train)))
```

```
Accuracy score (training): 0.9999392953929539
```

In [ ]:

```
y_pred = pipe_rf.predict(X_validation)
print("Accuracy score (validation):", str(accuracy_score(y_validation, y_pred)))
print("Log loss:", log_loss(y_validation, y_pred))
print("Confusion Matrix:", "\n", confusion_matrix(y_validation, y_pred))
print("ROC_AUC:", roc_auc_score(y_validation, pipe_rf.predict_proba(X_validation)[:, 1]))
```

```
Accuracy score (validation): 0.9180878048780488
Log loss: 2.829147012135902
Confusion Matrix:
 [[70576     2]
 [ 6295     2]]
ROC_AUC: 0.7055574874697946
```

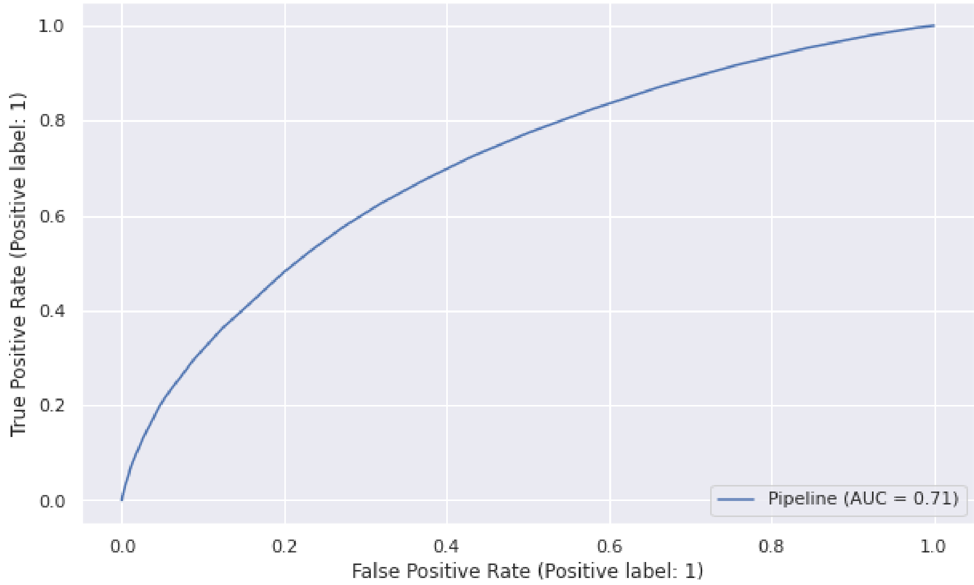
In [ ]:

```
model_results.append(["Random Forest", pipe_rf.score(X_train, y_train), accuracy_score(y_validation, y_pred)])
```

In [ ]:

```
func, ax = plt.subplots(1, 1, figsize=(10, 6))
metrics.plot_roc_curve(pipe_rf, X_validation, y_validation, ax=ax)
```

Out[ ]: `<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7fc30ba93a10>`



Build data frame of model results

In [ ]:

```
pd.DataFrame(model_results, columns = ["Model Name", "Training Accuracy", "Validation Accuracy"])
```

Out[ ]:

	Model Name	Training Accuracy	Validation Accuracy
0	Logistic Regression	0.919592	0.917919
1	Naive Bayes (Gaussian)	0.135722	0.136689
2	Random Forest	0.999939	0.918088

- From the above three pipeline tests we can see that the highest accuracy is obatined by using the Random Forest model, with accuracy of 91.826%. We also get the Logistic Regression model with accuracy of 91.802%. Both these models seem to be good models to start with.
- The ROC value for Logistic Regression and Random Forest is 0.72 and 0.71 which show a significant amount of True Positive values and indicate a good fit. The Naive Bayes is however not so great with only 0.62, so we cannot move forward with Naive Bayes on this ground as well.
- We need to check if Logistic Regression or Random Forest is a better model and whether either of them are overfitting.

Now we cleanup the memory a little bit to reclaim what is already used up, this is to optimize the colab runtime we are using.

In [ ]:

```
del missing_counts_percent_top_20
del above_75_zeroes
del above_35_nulls
del af
del df_target_f
del df_target_m
del df_family_status
del df_family_status_gender
del df_temp_dummies
del columns_nullcount_types
del df_train_interpolated_X
del df_train_interpolated
del ax
del data_transformed
del X_train
del X_validation
del y_train
del y_validation
```

In [ ]:

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

Out[ ]:

```
7075
```

Merge Datasets

In [ ]:

```
datasets.keys()
```

Out[ ]:

```
dict_keys(['application_train', 'application_test', 'bureau', 'bureau_balance', 'credit_card_balance', 'installments_paymen
ts', 'previous_application', 'POS_CASH_balance'])
```

In [ ]:

```python
datasets['installments_payments'] = datasets['installments_payments'].select_dtypes(exclude='object')
datasets['POS_CASH_balance'] = datasets['POS_CASH_balance'].select_dtypes(exclude='object')
datasets['previous_application'] = datasets['previous_application'].select_dtypes(exclude='object')
datasets['credit_card_balance'] = datasets['credit_card_balance'].select_dtypes(exclude='object')
```

In [ ]:

```python
datasets['installments_payments'] = datasets['installments_payments'][datasets['installments_payments']["SK_ID_CURR"].isin
datasets['POS_CASH_balance'] = datasets['POS_CASH_balance'][datasets['POS_CASH_balance']["SK_ID_CURR"].isin(df_train["SK_II
datasets['previous_application'] = datasets['previous_application'][datasets['previous_application']["SK_ID_CURR"].isin(df_
datasets['credit_card_balance'] = datasets['credit_card_balance'][datasets['credit_card_balance']["SK_ID_CURR"].isin(df_tra
```

In [ ]:

```python
datasets['installments_payments'] = datasets['installments_payments'].groupby("SK_ID_CURR", as_index=False).median()
datasets['POS_CASH_balance'] = datasets['POS_CASH_balance'].groupby("SK_ID_CURR", as_index=False).median()
datasets['previous_application'] = datasets['previous_application'].groupby("SK_ID_CURR", as_index=False).median()
datasets['credit_card_balance'] = datasets['credit_card_balance'].groupby("SK_ID_CURR", as_index=False).median()
```

In [ ]:

```python
display(datasets['installments_payments'].columns)
display(datasets['POS_CASH_balance'].columns)
display(datasets['previous_application'].columns)
display(datasets['credit_card_balance'].columns)
display(datasets['bureau_balance'].columns)
display(datasets['bureau'].columns)
```

```
Index(['SK_ID_CURR', 'SK_ID_PREV', 'NUM_INSTALMENT_VERSION',
       'NUM_INSTALMENT_NUMBER', 'DAYS_INSTALMENT', 'DAYS_ENTRY_PAYMENT',
       'AMT_INSTALMENT', 'AMT_PAYMENT'],
      dtype='object')
Index(['SK_ID_CURR', 'SK_ID_PREV', 'MONTHS_BALANCE', 'CNT_INSTALMENT',
       'CNT_INSTALMENT_FUTURE', 'SK_DPD', 'SK_DPD_DEF'],
      dtype='object')
Index(['SK_ID_CURR', 'SK_ID_PREV', 'AMT_ANNUITY', 'AMT_APPLICATION',
       'AMT_CREDIT', 'AMT_DOWN_PAYMENT', 'AMT_GOODS_PRICE',
       'HOUR_APPR_PROCESS_START', 'NFLAG_LAST_APPL_IN_DAY',
       'RATE_DOWN_PAYMENT', 'RATE_INTEREST_PRIMARY',
       'RATE_INTEREST_PRIVILEGED', 'DAYS_DECISION', 'SELLERPLACE_AREA',
       'CNT_PAYMENT', 'DAYS_FIRST_DRAWING', 'DAYS_FIRST_DUE',
       'DAYS_LAST_DUE_1ST_VERSION', 'DAYS_LAST_DUE', 'DAYS_TERMINATION',
       'NFLAG_INSURED_ON_APPROVAL'],
      dtype='object')
Index(['SK_ID_CURR', 'SK_ID_PREV', 'MONTHS_BALANCE', 'AMT_BALANCE',
       'AMT_CREDIT_LIMIT_ACTUAL', 'AMT_DRAWINGS_ATM_CURRENT',
       'AMT_DRAWINGS_CURRENT', 'AMT_DRAWINGS_OTHER_CURRENT',
       'AMT_DRAWINGS_POS_CURRENT', 'AMT_INST_MIN_REGULARITY',
       'AMT_PAYMENT_CURRENT', 'AMT_PAYMENT_TOTAL_CURRENT',
       'AMT_RECEIVABLE_PRINCIPAL', 'AMT_RECIVABLE', 'AMT_TOTAL_RECEIVABLE',
       'CNT_DRAWINGS_ATM_CURRENT', 'CNT_DRAWINGS_CURRENT',
       'CNT_DRAWINGS_OTHER_CURRENT', 'CNT_DRAWINGS_POS_CURRENT',
       'CNT_INSTALMENT_MATURE_CUM', 'SK_DPD', 'SK_DPD_DEF'],
      dtype='object')
Index(['SK_ID_BUREAU', 'MONTHS_BALANCE', 'STATUS'], dtype='object')
Index(['SK_ID_CURR', 'SK_ID_BUREAU', 'CREDIT_ACTIVE', 'CREDIT_CURRENCY',
       'DAYS_CREDIT', 'CREDIT_DAY_OVERDUE', 'DAYS_CREDIT_ENDDATE',
       'DAYS_ENDDATE_FACT', 'AMT_CREDIT_MAX_OVERDUE', 'CNT_CREDIT_PROLONG',
       'AMT_CREDIT_SUM', 'AMT_CREDIT_SUM_DEBT', 'AMT_CREDIT_SUM_LIMIT',
       'AMT_CREDIT_SUM_OVERDUE', 'CREDIT_TYPE', 'DAYS_CREDIT_UPDATE',
       'AMT_ANNUITY'],
      dtype='object')
```

In [ ]:

```python
df_train.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 307500 entries, 0 to 307510
Data columns (total 41 columns):
 #   Column                       Non-Null Count   Dtype
---  ------                       --------------   -----
 0   SK_ID_CURR                   307500 non-null  int64
 1   TARGET                       307500 non-null  int64
 2   NAME_CONTRACT_TYPE           307500 non-null  object
 3   CODE_GENDER                  307500 non-null  object
 4   FLAG_OWN_CAR                 307500 non-null  object
 5   FLAG_OWN_REALTY              307500 non-null  object
 6   CNT_CHILDREN                 307500 non-null  int64
 7   AMT_INCOME_TOTAL             307500 non-null  float64
 8   AMT_CREDIT                   307500 non-null  float64
 9   AMT_ANNUITY                  307488 non-null  float64
 10  AMT_GOODS_PRICE              307224 non-null  float64
 11  NAME_TYPE_SUITE              306210 non-null  object
 12  NAME_INCOME_TYPE             307500 non-null  object
 13  NAME_EDUCATION_TYPE          307500 non-null  object
 14  NAME_FAMILY_STATUS           307500 non-null  object
 15  NAME_HOUSING_TYPE            307500 non-null  object
```

```
 16  REGION_POPULATION_RELATIVE   307500 non-null  float64
 17  DAYS_BIRTH                   307500 non-null  int64
 18  DAYS_EMPLOYED                307500 non-null  int64
 19  DAYS_REGISTRATION            307500 non-null  float64
 20  DAYS_ID_PUBLISH              307500 non-null  int64
 21  FLAG_MOBIL                   307500 non-null  int64
 22  FLAG_EMP_PHONE               307500 non-null  int64
 23  FLAG_CONT_MOBILE             307500 non-null  int64
 24  FLAG_PHONE                   307500 non-null  int64
 25  OCCUPATION_TYPE              211112 non-null  object
 26  CNT_FAM_MEMBERS              307500 non-null  float64
 27  REGION_RATING_CLIENT         307500 non-null  int64
 28  REGION_RATING_CLIENT_W_CITY  307500 non-null  int64
 29  WEEKDAY_APPR_PROCESS_START   307500 non-null  object
 30  HOUR_APPR_PROCESS_START      307500 non-null  int64
 31  ORGANIZATION_TYPE            307500 non-null  object
 32  EXT_SOURCE_2                 306840 non-null  float64
 33  EXT_SOURCE_3                 246541 non-null  float64
 34  OBS_30_CNT_SOCIAL_CIRCLE     306479 non-null  float64
 35  OBS_60_CNT_SOCIAL_CIRCLE     306479 non-null  float64
 36  DAYS_LAST_PHONE_CHANGE       307499 non-null  float64
 37  FLAG_DOCUMENT_3              307500 non-null  int64
 38  AMT_REQ_CREDIT_BUREAU_MON    265986 non-null  float64
 39  AMT_REQ_CREDIT_BUREAU_QRT    265986 non-null  float64
 40  AMT_REQ_CREDIT_BUREAU_YEAR   265986 non-null  float64
dtypes: float64(15), int64(14), object(12)
memory usage: 98.5+ MB
```

In [ ]:
```
for ds_name in datasets.keys():
    print(f'dataset {ds_name:24}: [ {datasets[ds_name].shape[0]:10,}, {datasets[ds_name].shape[1]}]')
```

```
dataset application_train       : [    307,511, 41]
dataset application_test        : [     48,744, 121]
dataset bureau                  : [  1,716,428, 17]
dataset bureau_balance          : [ 27,299,925, 3]
dataset credit_card_balance     : [     86,904, 22]
dataset installments_payments   : [    291,637, 8]
dataset previous_application    : [    291,051, 21]
dataset POS_CASH_balance        : [    289,438, 7]
```

Prepare `df_bureau_balance` because we do not have any `SK_ID_CURR` in that dataset and so we need to merge it with dataset `bureau`.

In [ ]:
```
# Merge df_bureau and df_bureau_balance
datasets['bureau_balance'] = pd.merge(left=datasets['bureau'], right=datasets['bureau_balance'], how='left', left_on='SK_II
```

In [ ]:
```
datasets['bureau_balance'].columns
```

Out[ ]:
```
Index(['SK_ID_CURR', 'SK_ID_BUREAU', 'CREDIT_ACTIVE', 'CREDIT_CURRENCY',
       'DAYS_CREDIT', 'CREDIT_DAY_OVERDUE', 'DAYS_CREDIT_ENDDATE',
       'DAYS_ENDDATE_FACT', 'AMT_CREDIT_MAX_OVERDUE', 'CNT_CREDIT_PROLONG',
       'AMT_CREDIT_SUM', 'AMT_CREDIT_SUM_DEBT', 'AMT_CREDIT_SUM_LIMIT',
       'AMT_CREDIT_SUM_OVERDUE', 'CREDIT_TYPE', 'DAYS_CREDIT_UPDATE',
       'AMT_ANNUITY', 'MONTHS_BALANCE', 'STATUS'],
      dtype='object')
```

In [ ]:
```
# Remove data in bureau balance, with SK_ID_CURR not present in df_train
del datasets['bureau'] # Delete bureau as no longer needed
datasets['bureau_balance'] = datasets['bureau_balance'].drop_duplicates()
```

In [ ]:
```
# Group SK_ID_CURR
datasets['bureau_balance'] = datasets['bureau_balance'].groupby(['SK_ID_CURR','SK_ID_BUREAU']).min()
```

In [ ]:
```
datasets['bureau_balance'] = datasets['bureau_balance'][['DAYS_CREDIT','DAYS_ENDDATE_FACT','AMT_CREDIT_SUM','DAYS_CREDIT_UI
datasets['bureau_balance'] = datasets['bureau_balance'].reset_index()
datasets['bureau_balance'] = datasets['bureau_balance'].groupby('SK_ID_CURR').median()
datasets['bureau_balance'] = datasets['bureau_balance'].reset_index()
```

In [ ]:
```
datasets['bureau_balance'] = datasets['bureau_balance'].select_dtypes(exclude="object")
```

In [ ]:
```
datasets['bureau_balance']
```

Out[ ]:

	SK_ID_CURR	SK_ID_BUREAU	DAYS_CREDIT	DAYS_ENDDATE_FACT	AMT_CREDIT_SUM	DAYS_CREDIT_UPDATE	MONTHS_BALANCE
0	100001	5896633.0	-857.0	-715.0	168345.00	-155.0	-28.0
1	100002	6158905.5	-1042.5	-939.0	54130.50	-402.5	-34.0

	SK_ID_CURR	SK_ID_BUREAU	DAYS_CREDIT	DAYS_ENDDATE_FACT	AMT_CREDIT_SUM	DAYS_CREDIT_UPDATE	MONTHS_BALANCE
2	100003	5885878.5	-1205.5	-621.0	92576.25	-545.0	NaN
3	100004	6829133.5	-867.0	-532.5	94518.90	-532.0	NaN
4	100005	6735201.0	-137.0	-123.0	58500.00	-31.0	-4.0
...	...	...	...	...	...	...	...
305806	456249	5371700.0	-1680.0	-1279.0	248692.50	-909.0	NaN
305807	456250	6817237.0	-824.0	-760.0	483349.50	-31.0	-27.0
305808	456253	6098498.5	-919.0	-794.0	675000.00	-153.5	-30.0
305809	456254	6669849.0	-1104.0	-859.0	45000.00	-401.0	-36.0
305810	456255	5126332.0	-1020.0	-869.5	436032.00	-700.0	-33.0

305811 rows × 7 columns

In [ ]:

```
# Cache the current grouped dataset as it takes time
datasets['installments_payments'].to_csv("/content/drive/MyDrive/Group32_AML/datasets/df_installments_payments_grouped.csv'
datasets['POS_CASH_balance'].to_csv("/content/drive/MyDrive/Group32_AML/datasets/df_POS_CASH_balance_grouped.csv")
datasets['previous_application'].to_csv("/content/drive/MyDrive/Group32_AML/datasets/df_previous_application_grouped.csv")
datasets['credit_card_balance'].to_csv("/content/drive/MyDrive/Group32_AML/datasets/df_credit_card_balance_grouped.csv")
datasets['bureau_balance'].to_csv("/content/drive/MyDrive/Group32_AML/datasets/df_bureau_balance_grouped.csv")
```

In [ ]:

```
# # ONLY RUN if dynamically loading from cache
# # Done to avoid computation times
# datasets['installments_payments'] = pd.read_csv("/content/drive/MyDrive/Group32_AML/datasets/df_installments_payments_gro
# datasets['POS_CASH_balance'] = pd.read_csv("/content/drive/MyDrive/Group32_AML/datasets/df_POS_CASH_balance_grouped.csv")
# datasets['previous_application'] = pd.read_csv("/content/drive/MyDrive/Group32_AML/datasets/df_previous_application_grouμ
# datasets['credit_card_balance'] = pd.read_csv("/content/drive/MyDrive/Group32_AML/datasets/df_credit_card_balance_groupe
# datasets['bureau_balance'] = pd.read_csv("/content/drive/MyDrive/Group32_AML/datasets/df_bureau_balance_grouped.csv")
```

In [ ]:

```
pd.DataFrame(data=[["installments_payments", len(datasets['installments_payments'])],
                   ["POS_CASH_balance", len(datasets['POS_CASH_balance'])],
                   ["previous_application", len(datasets['previous_application'])],
                   ["credit_card_balance", len(datasets['credit_card_balance'])],
                   ["bureau_balance", len(datasets['bureau_balance'])]], columns=["Table", "Rows"])
```

Out[ ]:

	Table	Rows
0	installments_payments	291643
1	POS_CASH_balance	289444
2	previous_application	291057
3	credit_card_balance	86905
4	bureau_balance	305811

In [ ]:

```
datasets["installments_payments"].columns
```

Out[ ]:

```
Index(['Unnamed: 0', 'SK_ID_CURR', 'SK_ID_PREV', 'NUM_INSTALMENT_VERSION',
       'NUM_INSTALMENT_NUMBER', 'DAYS_INSTALMENT', 'DAYS_ENTRY_PAYMENT',
       'AMT_INSTALMENT', 'AMT_PAYMENT'],
      dtype='object')
```

Measure memory resources

In [ ]:

```
import sys

def print_memory_usage():
    to_measure = [x for x in (list(locals().keys()) + list(globals().keys())) if x.startswith("df_")]
    measured = [[x, sys.getsizeof(eval(x)) // 1024 // 1024] for x in to_measure]
    try:
        measured += [["dataset[" + k + "]", sys.getsizeof(datasets[k]) // 1024 // 1024] for k in datasets]
    except Exception as e:
        pass
    memory_usage = pd.DataFrame(data=measured,
                                columns=["DataFrame", "Size"]).sort_values("Size")
    display(memory_usage)
    print("Total:", memory_usage["Size"].sum())
```

The above shows that we might have much more data in installments_payments table

In [ ]:

```python
# Measure memory footprint
print_memory_usage()
```

	DataFrame	Size
8	dataset[credit_card_balance]	15
11	dataset[POS_CASH_balance]	17
7	dataset[bureau_balance]	18
9	dataset[installments_payments]	20
10	dataset[previous_application]	48
3	df_test	84
5	dataset[application_test]	84
1	df_merged_dummies	256
0	df_merged	452
6	dataset[bureau]	512
2	df_train	536
4	dataset[application_train]	536

```
Total: 2578
```

Filter out invalid data values compared to test dataset

In [ ]:

```python
df_train = drop_non_existing_in_test(df_train, df_test)
df_train
```

Out[ ]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTA
0	100002	1	Cash loans	M	N	Y	0	202500
1	100003	0	Cash loans	F	N	N	0	270000
2	100004	0	Revolving loans	M	Y	Y	0	67500
3	100006	0	Cash loans	F	N	Y	0	135000
4	100007	0	Cash loans	M	N	Y	0	121500
...	...	...	...	...	...	...	...	
307506	456251	0	Cash loans	M	N	N	0	157500
307507	456252	0	Cash loans	F	N	Y	0	72000
307508	456253	0	Cash loans	F	N	Y	0	153000
307509	456254	1	Cash loans	F	N	Y	0	171000
307510	456255	0	Cash loans	F	N	N	0	157500

307500 rows × 122 columns

Merge all tables

In [ ]:

```python
key = "SK_ID_CURR"
```

In [ ]:

```python
df_train.columns
```

Out[ ]:

```
Index(['SK_ID_CURR', 'TARGET', 'NAME_CONTRACT_TYPE', 'CODE_GENDER',
       'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN', 'AMT_INCOME_TOTAL',
       'AMT_CREDIT', 'AMT_ANNUITY',
       ...
       'FLAG_DOCUMENT_18', 'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20',
       'FLAG_DOCUMENT_21', 'AMT_REQ_CREDIT_BUREAU_HOUR',
       'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK',
       'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QRT',
       'AMT_REQ_CREDIT_BUREAU_YEAR'],
      dtype='object', length=122)
```

In [ ]:

```python
df_merged = pd.merge(left=df_train, right=datasets['previous_application'], how='left', left_on=key, right_on=key)
df_merged
```

Out[ ]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOT
0	100002	1	Cash loans	M	N	Y	0	20250
1	100003	0	Cash loans	F	N	N	0	27000
2	100004	0	Revolving loans	M	Y	Y	0	6750
3	100006	0	Cash loans	F	N	Y	0	13500
4	100007	0	Cash loans	M	N	Y	0	12150
...	...	...	...	...	...	...	...	
307495	456251	0	Cash loans	M	N	N	0	15750
307496	456252	0	Cash loans	F	N	Y	0	7200
307497	456253	0	Cash loans	F	N	Y	0	15300
307498	456254	1	Cash loans	F	N	Y	0	17100
307499	456255	0	Cash loans	F	N	N	0	15750

307500 rows × 143 columns

In [ ]:

```
df_merged = pd.merge(left=df_merged, right=datasets['POS_CASH_balance'], how='left', left_on=key, right_on=key)
df_merged
```

Out[ ]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOT
0	100002	1	Cash loans	M	N	Y	0	20250
1	100003	0	Cash loans	F	N	N	0	27000
2	100004	0	Revolving loans	M	Y	Y	0	6750
3	100006	0	Cash loans	F	N	Y	0	13500
4	100007	0	Cash loans	M	N	Y	0	12150
...	...	...	...	...	...	...	...	
307495	456251	0	Cash loans	M	N	N	0	15750
307496	456252	0	Cash loans	F	N	Y	0	7200
307497	456253	0	Cash loans	F	N	Y	0	15300
307498	456254	1	Cash loans	F	N	Y	0	17100
307499	456255	0	Cash loans	F	N	N	0	15750

307500 rows × 150 columns

In [ ]:

```
df_merged = pd.merge(left=df_merged, right=datasets['credit_card_balance'], how='left', left_on=key, right_on=key)
df_merged
```

Out[ ]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOT
0	100002	1	Cash loans	M	N	Y	0	20250
1	100003	0	Cash loans	F	N	N	0	27000
2	100004	0	Revolving loans	M	Y	Y	0	6750
3	100006	0	Cash loans	F	N	Y	0	13500
4	100007	0	Cash loans	M	N	Y	0	12150
...	...	...	...	...	...	...	...	
307495	456251	0	Cash loans	M	N	N	0	15750
307496	456252	0	Cash loans	F	N	Y	0	7200
307497	456253	0	Cash loans	F	N	Y	0	15300
307498	456254	1	Cash loans	F	N	Y	0	17100
307499	456255	0	Cash loans	F	N	N	0	15750

307500 rows × 172 columns

In [ ]:

```python
df_merged = pd.merge(left=df_merged, right=datasets['installments_payments'], how='left', left_on=key, right_on=key)
df_merged
```

Out[ ]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTA
0	100002	1	Cash loans	M	N	Y	0	20250
1	100003	0	Cash loans	F	N	N	0	27000
2	100004	0	Revolving loans	M	Y	Y	0	6750
3	100006	0	Cash loans	F	N	Y	0	13500
4	100007	0	Cash loans	M	N	Y	0	12150
...	...	...	...	...	...	...	...	
307495	456251	0	Cash loans	M	N	N	0	15750
307496	456252	0	Cash loans	F	N	Y	0	7200
307497	456253	0	Cash loans	F	N	Y	0	15300
307498	456254	1	Cash loans	F	N	Y	0	17100
307499	456255	0	Cash loans	F	N	N	0	15750

307500 rows × 180 columns

In [ ]:

```python
df_merged = pd.merge(left=df_merged, right=datasets['bureau_balance'], how='left', left_on=key, right_on=key)
df_merged
```

Out[ ]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTA
0	100002	1	Cash loans	M	N	Y	0	20250
1	100003	0	Cash loans	F	N	N	0	27000
2	100004	0	Revolving loans	M	Y	Y	0	6750
3	100006	0	Cash loans	F	N	Y	0	13500
4	100007	0	Cash loans	M	N	Y	0	12150
...	...	...	...	...	...	...	...	
307495	456251	0	Cash loans	M	N	N	0	15750
307496	456252	0	Cash loans	F	N	Y	0	7200
307497	456253	0	Cash loans	F	N	Y	0	15300
307498	456254	1	Cash loans	F	N	Y	0	17100
307499	456255	0	Cash loans	F	N	N	0	15750

307500 rows × 187 columns

In [ ]:

```python
# Cache the current grouped dataset as it takes time
df_merged.to_csv("/content/drive/MyDrive/Group32_AML/datasets/df_merged.csv")
```

In [ ]:

```python
# # ONLY RUN if dynamically loading from cache
# # Done to avoid computation times
# df_merged = pd.read_csv("/content/drive/MyDrive/Group32_AML/datasets/df_merged.csv")
```

In [ ]:

```python
df_merged.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 307500 entries, 0 to 307499
Columns: 187 entries, SK_ID_CURR to MONTHS_BALANCE
dtypes: float64(130), int64(41), object(16)
memory usage: 441.1+ MB
```

Convert categorical data to numerical and overwrite merged dataset

This helps to reduce the memory footprint as a lot of `string` data is converted to numerical data

In [ ]:
```
df_merged_dummies = pd.get_dummies(df_merged)
df_merged_dummies
```

Out[ ]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT_x	AMT_ANNUITY_x	AMT_GOODS_PRICE_x	REGION_POPULATI
0	100002	1	0	202500.0	406597.5	24700.5	351000.0	
1	100003	0	0	270000.0	1293502.5	35698.5	1129500.0	
2	100004	0	0	67500.0	135000.0	6750.0	135000.0	
3	100006	0	0	135000.0	312682.5	29686.5	297000.0	
4	100007	0	0	121500.0	513000.0	21865.5	513000.0	
...	...	...	...	...	...	...	...	
307495	456251	0	0	157500.0	254700.0	27558.0	225000.0	
307496	456252	0	0	72000.0	269550.0	12001.5	225000.0	
307497	456253	0	0	153000.0	677664.0	29979.0	585000.0	
307498	456254	1	0	171000.0	370107.0	20205.0	319500.0	
307499	456255	0	0	157500.0	675000.0	49117.5	675000.0	

307500 rows × 308 columns

In [ ]:
```
df_merged_dummies = df_merged_dummies.interpolate(method='cubic', limit_direction="forward")
df_merged_dummies
```

Out[ ]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT_x	AMT_ANNUITY_x	AMT_GOODS_PRICE_x	REGION_POPULATI
0	100002	1	0	202500.0	406597.5	24700.5	351000.0	
1	100003	0	0	270000.0	1293502.5	35698.5	1129500.0	
2	100004	0	0	67500.0	135000.0	6750.0	135000.0	
3	100006	0	0	135000.0	312682.5	29686.5	297000.0	
4	100007	0	0	121500.0	513000.0	21865.5	513000.0	
...	...	...	...	...	...	...	...	
307495	456251	0	0	157500.0	254700.0	27558.0	225000.0	
307496	456252	0	0	72000.0	269550.0	12001.5	225000.0	
307497	456253	0	0	153000.0	677664.0	29979.0	585000.0	
307498	456254	1	0	171000.0	370107.0	20205.0	319500.0	
307499	456255	0	0	157500.0	675000.0	49117.5	675000.0	

307500 rows × 308 columns

In [ ]:
```
# Cache the current grouped dataset as it takes time
df_merged_dummies.to_csv("/content/drive/MyDrive/Group32_AML/datasets/df_merged_interpolated.csv")
```

In [ ]:
```
# # ONLY RUN if dynamically loading from cache
# # Done to avoid computation times
# df_merged_dummies = pd.read_csv("/content/drive/MyDrive/Group32_AML/datasets/df_merged_interpolated.csv")
```

In [ ]:
```
df_merged_dummies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307500 entries, 0 to 307499
Columns: 309 entries, Unnamed: 0 to EMERGENCYSTATE_MODE_Yes
dtypes: float64(130), int64(179)
memory usage: 724.9 MB
```

Transform the current data using our old pipeline

In [ ]:
```
duplicated_columns = df_merged_dummies.columns[df_merged_dummies.columns.duplicated()]
duplicated_columns
```

```
Index([], dtype='object')
```

Out[ ]:

In [ ]:

```
# https://stackoverflow.com/questions/14984119/python-pandas-remove-duplicate-columns
df_merged_dummies = df_merged_dummies.loc[:, ~df_merged_dummies.columns.duplicated()]
```

In [ ]:

```
# Check if duplicates removed
df_merged_dummies.columns[df_merged_dummies.columns.duplicated()]
```

Out[ ]:

```
Index([], dtype='object')
```

In [ ]:

```
from sklearn.compose import ColumnTransformer

# Create a basic numerical pipeline
num_pipeline = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean'))
])

# Column transform pipeline
data_pipeline = ColumnTransformer([
    ("num_pipeline", num_pipeline, df_merged_dummies.columns)
], n_jobs = -1)

# Run the transform
data_transformed = data_pipeline.fit_transform(df_merged_dummies)
```

In [ ]:

```
df_transformed = pd.DataFrame(data_transformed,  columns=df_merged_dummies.columns)
del data_transformed
df_transformed
```

Out[ ]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT_x	AMT_ANNUITY_x	AMT_GOODS_PRICE_x	REGION_POPULATIO
0	100002.0	1.0	0.0	202500.0	406597.5	24700.5	351000.0	
1	100003.0	0.0	0.0	270000.0	1293502.5	35698.5	1129500.0	
2	100004.0	0.0	0.0	67500.0	135000.0	6750.0	135000.0	
3	100006.0	0.0	0.0	135000.0	312682.5	29686.5	297000.0	
4	100007.0	0.0	0.0	121500.0	513000.0	21865.5	513000.0	
...	...	...	...	...	...	...	...	
307495	456251.0	0.0	0.0	157500.0	254700.0	27558.0	225000.0	
307496	456252.0	0.0	0.0	72000.0	269550.0	12001.5	225000.0	
307497	456253.0	0.0	0.0	153000.0	677664.0	29979.0	585000.0	
307498	456254.0	1.0	0.0	171000.0	370107.0	20205.0	319500.0	
307499	456255.0	0.0	0.0	157500.0	675000.0	49117.5	675000.0	

307500 rows × 304 columns

NEW FEATURES

We have created a total of eleven new features based on certain computable properties that we realised were missing/would add value to our dataset, based on relations between certain columns.

We've added one to wherever we might get zero values, and hence avoid division by zero error.

_The features `F1` to `F7` are ratios of different amount values that are either constant or reflect an important property about the loan/lender.

1. `F1_RCV:BAL` - Ratio of total receivable amount to balance amount
2. `F2_TRC:RCV` - Ratio of total receivable amount to receivable amount
3. `F3_TRC:RCP` - Ratio of total receivable amount to receivable principal
4. `F4_BAL:RCV` - Ratio of balance amount to receivable amount
5. `F5_BAL:RCP` - Ratio of balance amount to receivable principal
6. `F6_RCV:RCP` - Ratio of receivable amount to receivable principal
7. `F7_EXT:RCV` - Ratio of all EXT sources to total receivable amount

The features `F8` *to* `F11` *are sum/product/max values of the respective amount columns in use.*

1. `F8_SUM_EXT` - Sum of all EXT sources
2. `F9_PRD_EXT` - Product of all EXT sources
3. `F10_EXT_RATIO` - Weighted sum of all EXT sources
4. `F11_MAX_EXT` - Maximum EXT source

Based on the new features above, we create new columns for each new feature by using their respective formulae.

In [ ]:

```
# We've added one to certain terms to avoid division by zero error
df_transformed['F1_RCV:BAL'] = df_transformed['AMT_TOTAL_RECEIVABLE'] / (df_transformed['AMT_BALANCE'] + 1)
df_transformed['F2_TRC:RCV'] = df_transformed['AMT_TOTAL_RECEIVABLE'] / (df_transformed['AMT_RECIVABLE'] + 1)
df_transformed['F3_TRC:RCP'] = df_transformed['AMT_TOTAL_RECEIVABLE'] / (df_transformed['AMT_RECEIVABLE_PRINCIPAL'] + 1)
df_transformed['F4_BAL:RCV'] = df_transformed['AMT_BALANCE'] / (df_transformed['AMT_RECIVABLE'] + 1)
df_transformed['F5_BAL:RCP'] = df_transformed['AMT_BALANCE'] / (df_transformed['AMT_RECEIVABLE_PRINCIPAL'] + 1)
df_transformed['F6_RCV:RCP'] = df_transformed['AMT_RECIVABLE'] / (df_transformed['AMT_RECEIVABLE_PRINCIPAL'] + 1)
df_transformed['F7_EXT:RCV'] = (df_transformed['EXT_SOURCE_1'] + df_transformed['EXT_SOURCE_2'] + df_transformed['EXT_SOUR(
df_transformed['F8_SUM_EXT'] = (df_transformed['EXT_SOURCE_1'] + df_transformed['EXT_SOURCE_2'] + df_transformed['EXT_SOUR(
df_transformed['F9_PRD_EXT'] = (df_transformed['EXT_SOURCE_1'] * df_transformed['EXT_SOURCE_2'] * df_transformed['EXT_SOUR(
df_transformed['F10_EXT_RATIO'] = (df_transformed['EXT_SOURCE_1'] + df_transformed['EXT_SOURCE_2'] * 2 + df_transformed['E>
df_transformed['F11_MAX_EXT'] = [max(x, y, z) for x, y, z in zip(df_transformed['EXT_SOURCE_1'], df_transformed['EXT_SOURCI
```

Fill NaN values

In [ ]:

```
df_transformed = df_transformed.apply(lambda x: x.fillna(x.median()), axis=0)
df_transformed.drop(columns=["Unnamed: 0"], inplace=True)
df_transformed
```

Out[ ]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT_x	AMT_ANNUITY_x	AMT_GOODS_PRICE_x	REGION_POPULATIO
0	100002.0	1.0	0.0	202500.0	406597.5	24700.5	351000.0	
1	100003.0	0.0	0.0	270000.0	1293502.5	35698.5	1129500.0	
2	100004.0	0.0	0.0	67500.0	135000.0	6750.0	135000.0	
3	100006.0	0.0	0.0	135000.0	312682.5	29686.5	297000.0	
4	100007.0	0.0	0.0	121500.0	513000.0	21865.5	513000.0	
...	...	...	...	...	...	...	...	
307495	456251.0	0.0	0.0	157500.0	254700.0	27558.0	225000.0	
307496	456252.0	0.0	0.0	72000.0	269550.0	12001.5	225000.0	
307497	456253.0	0.0	0.0	153000.0	677664.0	29979.0	585000.0	
307498	456254.0	1.0	0.0	171000.0	370107.0	20205.0	319500.0	
307499	456255.0	0.0	0.0	157500.0	675000.0	49117.5	675000.0	

307500 rows × 314 columns

In [ ]:

```
# Cache the current grouped dataset as it takes time
df_transformed.to_csv("/content/drive/MyDrive/Group32_AML/datasets/df_transformed.csv")
```

In [ ]:

```
# # ONLY RUN if dynamically loading from cache
# # Done to avoid computation times
# df_transformed = pd.read_csv("/content/drive/MyDrive/Group32_AML/datasets/df_transformed.csv")
```

See corr in new features with TARGET

In [ ]:

```
df_new_features = df_transformed[['F1_RCV:BAL', 'F2_TRC:RCV', 'F3_TRC:RCP', 'F4_BAL:RCV', 'F5_BAL:RCP', 'F6_RCV:RCP', 'F7_I
df_new_features_corr = df_new_features.corr()['TARGET'].sort_values(key=abs,ascending=False).reset_index()
df_new_features_corr.columns = ["Feature", "Correlation"]
columns_new_highly_correlated = df_new_features_corr[abs(df_new_features_corr["Correlation"]) > 0.0]
display(columns_new_highly_correlated)
del df_new_features
del df_new_features_corr
del columns_new_highly_correlated
```

	Feature	Correlation
0	TARGET	1.000000
1	F10_EXT_RATIO	-0.191394

	Feature	Correlation
2	F8_SUM_EXT	-0.170612
3	F9_PRD_EXT	-0.138242
4	F11_MAX_EXT	-0.104619
5	F7_EXT:RCV	-0.023481
6	F2_TRC:RCV	0.010480
7	F6_RCV:RCP	0.001826
8	F3_TRC:RCP	0.001768
9	F5_BAL:RCP	0.001643
10	F1_RCV:BAL	-0.001039
11	F4_BAL:RCV	0.000315

In [ ]:

```
df_all_features_corr = df_transformed.corr()['TARGET'].sort_values(key=abs,ascending=False).reset_index()
df_all_features_corr.columns = ["Feature", "Correlation"]
```

In [ ]:

```
columns_all_highly_correlated = df_all_features_corr[abs(df_all_features_corr["Correlation"]) > 0.05][1:]
display(columns_all_highly_correlated)
del df_all_features_corr
```

	Feature	Correlation
1	F10_EXT_RATIO	-0.191394
2	F8_SUM_EXT	-0.170612
3	EXT_SOURCE_2	-0.160066
4	F9_PRD_EXT	-0.138242
5	EXT_SOURCE_3	-0.133961
6	F11_MAX_EXT	-0.104619
7	DAYS_BIRTH	0.078236
8	DAYS_CREDIT	0.068278
9	REGION_RATING_CLIENT_W_CITY	0.060875
10	REGION_RATING_CLIENT	0.058882
11	NAME_INCOME_TYPE_Working	0.057504
12	NAME_EDUCATION_TYPE_Higher education	-0.056578
13	DAYS_LAST_PHONE_CHANGE	0.055229
14	CODE_GENDER_M	0.054729
15	CODE_GENDER_F	-0.054729
16	DAYS_CREDIT_UPDATE	0.053450
17	DAYS_ID_PUBLISH	0.051455
18	REG_CITY_NOT_WORK_CITY	0.050981

In [ ]:

```
columns_greater_than_5 = columns_all_highly_correlated["Feature"]
columns_greater_than_5
```

Out[ ]:

```
1                          F10_EXT_RATIO
2                             F8_SUM_EXT
3                           EXT_SOURCE_2
4                             F9_PRD_EXT
5                           EXT_SOURCE_3
6                            F11_MAX_EXT
7                             DAYS_BIRTH
8                            DAYS_CREDIT
9            REGION_RATING_CLIENT_W_CITY
10                  REGION_RATING_CLIENT
11              NAME_INCOME_TYPE_Working
12    NAME_EDUCATION_TYPE_Higher education
13                DAYS_LAST_PHONE_CHANGE
14                         CODE_GENDER_M
15                         CODE_GENDER_F
16                    DAYS_CREDIT_UPDATE
17                       DAYS_ID_PUBLISH
```

```
18                REG_CITY_NOT_WORK_CITY
Name: Feature, dtype: object
```

Reduce Bias in TARGET values

This section is not used.

Method 1

Remove rows with NaN values based on un-transformed values in original dataset and filter in `SK_ID_CURR`

In [ ]:
```
# df_transformed_ones = df_transformed[df_transformed["TARGET"] == 1.0]
# df_transformed_zeroes = df_transformed[df_transformed["TARGET"] == 0.0]
# display(len(df_transformed_ones))
# display(len(df_transformed_zeroes))
```

In [ ]:
```
# df_zeroes_merged = df_merged[df_merged["SK_ID_CURR"].isin(df_transformed_zeroes["SK_ID_CURR"])]
# df_zeroes_merged.dropna(thresh=100, inplace=True)
# df_transformed_zeroes = df_transformed_zeroes[df_transformed_zeroes["SK_ID_CURR"].isin(df_zeroes_merged["SK_ID_CURR"])]
# df_transformed_zeroes
```

In [ ]:
```
# df_transformed = df_transformed_zeroes.append(df_transformed_ones)
# del df_transformed_ones
# del df_transformed_zeroes
# del df_zeroes_merged
# df_transformed
```

Doesn't work, so we try method 2.

Method 2

Resample the minority `TARGET` population from the dataset into higher number of rows.

This solution to resample is borrowed from here.

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

# # Separate zeroes and ones
# df_majority = df_transformed[(df_transformed['TARGET'] == 0)]
# df_minority = df_transformed[(df_transformed['TARGET'] == 1)]

# # Upsample minority class
# df_minority_upsampled = resample(df_minority, replace=True, n_samples=len(df_majority), random_state=42)

# # Combine majority class with upsampled minority class
# df_transformed = pd.concat([df_minority_upsampled, df_majority])
```

In [ ]:
```
# df_transformed.sample(n=20)
```

We don't need this anymore as the problem was solved by removing TARGET from the X dataset

Check memory usage

In [ ]:
```
print_memory_usage()
```

Delete unused variables

In [ ]:
```
del df_minority
del df_majority
del df_minority_upsampled
```

HYPERPARAMETER TUNING (using GridSearch)

In [ ]:
```
from sklearn.metrics import log_loss
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
from sklearn.pipeline import make_pipeline
```

```
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

In [ ]:

```
# Get just the TARGET column
y = df_transformed["TARGET"]

# Get all other column
X = df_transformed[columns_greater_than_5]
```

In [ ]:

```
X
```

Out[ ]:

	F10_EXT_RATIO	F8_SUM_EXT	EXT_SOURCE_2	F9_PRD_EXT	EXT_SOURCE_3	F11_MAX_EXT	DAYS_BIRTH	DAYS_CREDIT	REGION_RATING_C
0	1.027061	0.485361	0.262949	0.003043	0.139376	0.262949	-9461.0	-1042.500000	
1	3.129262	1.458014	0.622246	0.101588	0.524501	0.622246	-16765.0	-1205.500000	
2	3.788411	1.773366	0.555912	0.197875	0.729567	0.729567	-19046.0	-867.000000	
3	4.289370	2.058418	0.650442	0.317518	0.790255	0.790255	-19005.0	-944.359225	
4	3.577815	1.770581	0.322738	0.169027	0.742248	0.742248	-19932.0	-1149.000000	
...	...	...	...	...	...	...	...	...	
307495	2.961330	1.311368	0.681632	0.048042	0.484165	0.681632	-9327.0	-1672.777265	
307496	1.192956	0.726770	0.115992	0.008849	0.175097	0.435682	-20775.0	-1218.686197	
307497	2.472047	1.498607	0.535722	0.087235	0.218859	0.744026	-14966.0	-919.000000	
307498	3.911762	2.075552	0.514163	0.306011	0.661024	0.900366	-11961.0	-1104.000000	
307499	2.493365	1.556951	0.708569	0.059287	0.113922	0.734460	-16856.0	-1020.000000	

307500 rows × 18 columns

In [ ]:

```
y
```

Out[ ]:

```
0         1.0
1         0.0
2         0.0
3         0.0
4         0.0
         ...
307495    0.0
307496    0.0
307497    0.0
307498    1.0
307499    0.0
Name: TARGET, Length: 307500, dtype: float64
```

In [ ]:

```
# Get actual values
X = X.values
y = y.values
```

In [ ]:

```
# Train and Test split
X_train, X_validation, y_train, y_validation = train_test_split(X, y, test_size=0.25, random_state=42)
```

In [ ]:

```
display(len(y_train))
display(len(y_validation))
```

```
230625
76875
```

New Pipeline + GridSearch

Hyperparameter tuning

- After selecting the optimal features for our models, we use hyperparameter tuning to find the most optimal setting for running our model.
- For hyperparameter tuning, we have decided to use GridSearchCV. The GridSearchCV is a library of sklearn's model selection package. It helps to fit our model on our data by using the best running conditions and the best parameters possible.

- By just specifying the model, input parameters, and accuracy we want, we can easily obtain the best running conditions and features using GridSearchCV.
- In our code, for logistic regression, we have created a function for GridSearchCV with 3 cross-validations for the hyperparameters and 1000 max iterations.

Logistic Regression

In [ ]:

```
pipeline = Pipeline([
    ('scaler', StandardScaler()),

    # use GridSearchCV for LogisticRegression with LBGFS solver, we choose different concordance parameter which reflects t
    ('classifier', GridSearchCV(LogisticRegression(solver='lbfgs', max_iter=2000), param_grid={'C': [ 0.1, 1, 5, 10.]}, cv=
])

# Fit the train data
pipeline.fit(X_train, y_train)
```

Out[ ]:

```
Pipeline(steps=[('scaler', StandardScaler()),
                ('classifier',
                 GridSearchCV(cv=5, estimator=LogisticRegression(max_iter=2000),
                              param_grid={'C': [0.1, 1, 5, 10.0]}))])
```

In [ ]:

```
pipeline.named_steps['classifier'].best_params_
```

Out[ ]:

```
{'C': 0.1}
```

In [ ]:

```
print("Training set accuracy score: " + str(pipeline.score(X_train, y_train)))
y_pred = pipeline.predict(X_validation)
print("Validation set accuracy score:", str(accuracy_score(y_validation, y_pred)))
print("Log loss:", log_loss(y_validation, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_validation, y_pred))
print("Score ROC_AUC:", roc_auc_score(y_validation, pipeline.predict_proba(X_validation)[:, 1]))
```

```
Training set accuracy score: 0.9195794037940379
Validation set accuracy score: 0.9178926829268292
Log loss: 2.8358865040054266
Confusion Matrix:
 [[70555    23]
 [ 6289     8]]
Score ROC_AUC: 0.7164539034169695
```

In [ ]:

```
print(len(y_validation), " => ", len(y_validation[y_validation == 0.0]))
```

```
76875  =>  70578
```

In [ ]:

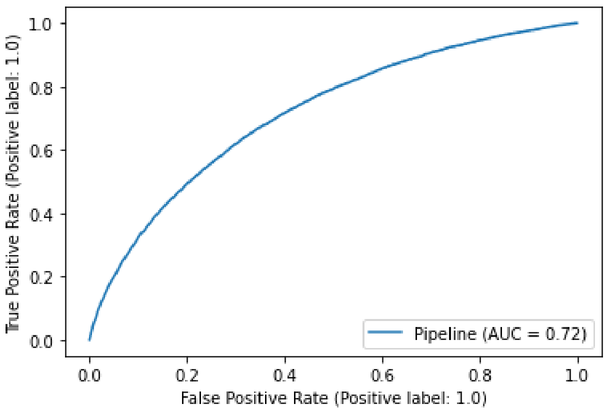
```
model_results.append(['Logistic Regression (Phase 2)', pipeline.score(X_train,y_train), accuracy_score(y_validation, y_pred
```

In [ ]:

```
metrics.plot_roc_curve(pipeline, X_validation, y_validation)
```

Out[ ]:

```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f9ac15a0510>
```



Random Forest

In [ ]:

```
pipeline_rf = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', GridSearchCV(RandomForestClassifier(), param_grid={
        'n_estimators': [40, 50, 60],
        'max_features': ['auto'],
        'max_depth' : [15, 20, 25],
        'criterion' :['entropy']
    }, cv=4, refit=True))
```

```
])
pipeline_rf.fit(X_train, y_train)
```

Out[ ]:
```
Pipeline(steps=[('scaler', StandardScaler()),
                ('classifier',
                 GridSearchCV(cv=4, estimator=RandomForestClassifier(),
                              param_grid={'criterion': ['entropy'],
                                          'max_depth': [15, 20, 25],
                                          'max_features': ['auto'],
                                          'n_estimators': [40, 50, 60]}))])
```

In [ ]:
```
pipeline_rf.named_steps['classifier'].best_params_
```

Out[ ]:
```
{'criterion': 'entropy',
 'max_depth': 25,
 'max_features': 'auto',
 'n_estimators': 60}
```

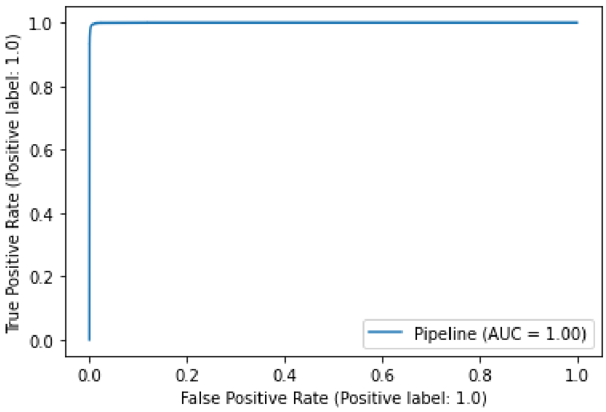
In [ ]:
```
print("Training set accuracy score: " + str(pipeline_rf.score(X_train, y_train)))
y_pred = pipeline_rf.predict(X_validation)
print("Validation set accuracy score:", str(accuracy_score(y_validation, y_pred)))
print("Log loss:", log_loss(y_validation, y_pred))
print("Confusion Matrix:\n", confusion_matrix(y_validation, y_pred))
print("Score ROC_AUC:", roc_auc_score(y_validation, pipeline_rf.predict_proba(X_validation)[:, 1]))
```

```
Training set accuracy score: 0.9977784538321671
Validation set accuracy score: 0.9802678590379431
Log loss: 0.6815396238505677
Confusion Matrix:
 [[67863  2761]
 [   28 70691]]
Score ROC_AUC: 0.9997644441407906
```

In [ ]:
```
model_results.append(['Random Forest (Phase 2)', pipeline_rf.score(X_train, y_train), accuracy_score(y_validation, y_pred)]
```

In [ ]:
```
metrics.plot_roc_curve(pipeline_rf, X_validation, y_validation)
```

Out[ ]:
```
<sklearn.metrics._plot.roc_curve.RocCurveDisplay at 0x7f3ddc711790>
```



In [ ]:
```
pd.DataFrame(model_results, columns = ["Model Name", "Training Accuracy", "Validation Accuracy"])
```

Out[ ]:

	Model Name	Training Accuracy	Validation Accuracy
0	Logistic Regression	0.919592	0.917919
1	Naive Bayes (Gaussian)	0.135722	0.136689
2	Random Forest	0.999939	0.918088
3	Logistic Regression (Phase 2)	0.919579	0.917892
4	Random Forest (Phase 2)	0.997778	0.980267

IMPACT OF NEW FEATURES

With the new features we have created columns which are much highly correlated with the `TARGET` and selected the top 18 features to train our model on.

The validation accuracy of Logistic Regression is quite good at 0.9195 but Random Forest attains an accuracy of 0.999, which we are assuming is because of an overfit. However, we will proceed to make the Kaggle submission for Phase 2 with this.

KAGGLE SUBMISSION - PHASE 2

Prepare data to be merged with `df_test`

In [ ]:
```
print_memory_usage()
```

	DataFrame	Size
9	dataset[credit_card_balance]	15
12	dataset[POS_CASH_balance]	17
8	dataset[bureau_balance]	18
10	dataset[installments_payments]	20
11	dataset[previous_application]	48
3	df_test	84
6	dataset[application_test]	84
1	df_merged_dummies	434
7	dataset[bureau]	512
5	dataset[application_train]	536
2	df_train	539
0	df_merged	691
4	df_transformed	736

```
Total: 3734
```

Model validation and prediction

Training Dataset

We now test the complete training dataset

In [ ]:
```
X = df_transformed[columns_greater_than_5].values
y = df_merged_dummies["TARGET"].values
```

Find the accuracy with the complete training dataset using Logistic Regression and the best estimator parameters- `C=0.1`

In [ ]:
```
pipe = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', LogisticRegression(solver='lbfgs', max_iter=1000, C=0.1))
])
pipe.fit(X, y)
print("Accuracy score (training):", str(pipe.score(X, y)))
```

```
Accuracy score (training): 0.9191577235772358
```

Find the accuracy with the complete training dataset using Random Forest and the best estimator parameters- `criterion='entropy', max_depth=25, max_features='auto', n_estimators=60`

In [ ]:
```
pipe_rf = Pipeline([
    ('scaler', StandardScaler()),
    ('classifier', RandomForestClassifier(criterion='entropy', max_depth=25, max_features='auto', n_estimators=60))
])

pipe_rf.fit(X, y)
print("Accuracy score (training):", str(pipe_rf.score(X, y)))
```

```
Accuracy score (training): 0.9727154471544716
```

Just like Phase-2, we see that Random Forest classifier is giving a much higher accuracy score, hence we will proceed with the Kaggle submission with this.

Cache both models

In [ ]:
```
import joblib
display(joblib.dump(pipe, "/content/drive/MyDrive/Group32_AML/LogisticRegression.model"))
display(joblib.dump(pipe_rf, "/content/drive/MyDrive/Group32_AML/RandomForest.model"))
```

```
['/content/drive/MyDrive/Group32_AML/LogisticRegression.model']
['/content/drive/MyDrive/Group32_AML/RandomForest.model']
```

In [ ]:

```
# # Load models from cache, only if required dynamically
# pipe = joblib.load("/content/drive/MyDrive/Group32_AML/LogisticRegression.model")
# pipe_rf = joblib.load("/content/drive/MyDrive/Group32_AML/RandomForest.model")
```

Test Dataset

Here we get the same final columns as for the training dataset in the test dataset.

In [ ]:
```
del df_merged
del df_merged_dummies
del df_transformed
del datasets
```

In [ ]:
```
import gc
gc.collect()
```

Out[ ]:
```
473
```

In [ ]:
```
# First handle memory
print_memory_usage()
```

	DataFrame	Size
1	df_test	84
0	df_train	539

```
Total: 623
```

In [ ]:
```
del X_train
del X_validation
del y_train
del y_validation
```

Reload datasets for test data preparation

In [ ]:
```
try:
    del datasets
except Exception as e:
    pass

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

HOME = "/content/drive/MyDrive/Group32_AML"
DATA_DIR = HOME + "/datasets/"
datasets = {}
ds_names = ("application_train", "application_test", "bureau","bureau_balance","credit_card_balance","installments_payments
            "previous_application","POS_CASH_balance")

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

```
application_train: shape is (307511, 122)
application_test: shape is (48744, 121)
bureau: shape is (1716428, 17)
bureau_balance: shape is (27299925, 3)
credit_card_balance: shape is (3840312, 23)
installments_payments: shape is (13605401, 8)
previous_application: shape is (1670214, 37)
POS_CASH_balance: shape is (10001358, 8)
```

In [ ]:
```
df_test = datasets["application_test"]
```

In [ ]:
```
# Filter out all object types
datasets['installments_payments'] = datasets['installments_payments'].select_dtypes(exclude='object')
datasets['POS_CASH_balance'] = datasets['POS_CASH_balance'].select_dtypes(exclude='object')
datasets['previous_application'] = datasets['previous_application'].select_dtypes(exclude='object')
datasets['credit_card_balance'] = datasets['credit_card_balance'].select_dtypes(exclude='object')
```

In [ ]:
```
# Filter all data with SK_ID_CURR present in df_test
```

```
datasets['installments_payments'] = datasets['installments_payments'][datasets['installments_payments']["SK_ID_CURR"].isin
datasets['POS_CASH_balance'] = datasets['POS_CASH_balance'][datasets['POS_CASH_balance']["SK_ID_CURR"].isin(df_test["SK_ID
datasets['previous_application'] = datasets['previous_application'][datasets['previous_application']["SK_ID_CURR"].isin(df
datasets['credit_card_balance'] = datasets['credit_card_balance'][datasets['credit_card_balance']["SK_ID_CURR"].isin(df_te
```

In [ ]:

```
# Group-by SK_ID_CURR
datasets['installments_payments'] = datasets['installments_payments'].groupby("SK_ID_CURR", as_index=False).median()
datasets['POS_CASH_balance'] = datasets['POS_CASH_balance'].groupby("SK_ID_CURR", as_index=False).median()
datasets['previous_application'] = datasets['previous_application'].groupby("SK_ID_CURR", as_index=False).median()
datasets['credit_card_balance'] = datasets['credit_card_balance'].groupby("SK_ID_CURR", as_index=False).median()
```

In [ ]:

```
# Merge df_bureau and df_bureau_balance
datasets['bureau_balance'] = pd.merge(left=datasets['bureau'], right=datasets['bureau_balance'], how='left', left_on='SK_I

# Remove data in bureau balance, with SK_ID_CURR not present in df_train
del datasets['bureau'] # Delete bureau as no longer needed
datasets['bureau_balance'] = datasets['bureau_balance'].drop_duplicates()

# Group SK_ID_CURR and call min()
datasets['bureau_balance'] = datasets['bureau_balance'].groupby(['SK_ID_CURR','SK_ID_BUREAU']).min()
```

In [ ]:

```
# Select only required columns and group-by SK_ID_CURR
datasets['bureau_balance'] = datasets['bureau_balance'][['DAYS_CREDIT','DAYS_ENDDATE_FACT','AMT_CREDIT_SUM','DAYS_CREDIT_U
datasets['bureau_balance'] = datasets['bureau_balance'].reset_index()
datasets['bureau_balance'] = datasets['bureau_balance'].groupby('SK_ID_CURR').median()
datasets['bureau_balance'] = datasets['bureau_balance'].reset_index()
datasets['bureau_balance'] = datasets['bureau_balance'].select_dtypes(exclude="object")
```

In [ ]:

```
# Filter new bureau_balance merged table on SK_ID_CURR
datasets['bureau_balance'] = datasets['bureau_balance'][datasets['bureau_balance']["SK_ID_CURR"].isin(df_test["SK_ID_CURR"]
```

In [ ]:

```
# Cache the current grouped dataset as it takes time
datasets['installments_payments'].to_csv("/content/drive/MyDrive/Group32_AML/datasets/df_test_installments_payments_groupe
datasets['POS_CASH_balance'].to_csv("/content/drive/MyDrive/Group32_AML/datasets/df_test_POS_CASH_balance_grouped.csv")
datasets['previous_application'].to_csv("/content/drive/MyDrive/Group32_AML/datasets/df_test_previous_application_grouped.
datasets['credit_card_balance'].to_csv("/content/drive/MyDrive/Group32_AML/datasets/df_test_credit_card_balance_grouped.cs
datasets['bureau_balance'].to_csv("/content/drive/MyDrive/Group32_AML/datasets/df_test_bureau_balance_grouped.csv")
```

In [ ]:

```
# # ONLY RUN if dynamically loading from cache
# # Done to avoid computation times
# datasets['installments_payments'] = pd.read_csv("/content/drive/MyDrive/Group32_AML/datasets/df_test_installments_payment
# datasets['POS_CASH_balance'] = pd.read_csv("/content/drive/MyDrive/Group32_AML/datasets/df_test_POS_CASH_balance_grouped
# datasets['previous_application'] = pd.read_csv("/content/drive/MyDrive/Group32_AML/datasets/df_test_previous_application_
# datasets['credit_card_balance'] = pd.read_csv("/content/drive/MyDrive/Group32_AML/datasets/df_test_credit_card_balance_gr
# datasets['bureau_balance'] = pd.read_csv("/content/drive/MyDrive/Group32_AML/datasets/df_test_bureau_balance_grouped.csv
```

In [ ]:

```
pd.DataFrame(data=[["installments_payments", len(datasets['installments_payments'])],
                   ["POS_CASH_balance", len(datasets['POS_CASH_balance'])],
                   ["previous_application", len(datasets['previous_application'])],
                   ["credit_card_balance", len(datasets['credit_card_balance'])],
                   ["bureau_balance", len(datasets['bureau_balance'])]], columns=["Table", "Rows"])
```

Out[ ]:

	Table	Rows
0	installments_payments	47944
1	POS_CASH_balance	47808
2	previous_application	47800
3	credit_card_balance	16653
4	bureau_balance	42320

Merge all datasets for test

In [ ]:

```
key = "SK_ID_CURR"
```

In [ ]:

```
df_test_merged = pd.merge(left=df_test, right=datasets['previous_application'], how='left', left_on=key, right_on=key)
df_test_merged = pd.merge(left=df_test_merged, right=datasets['POS_CASH_balance'], how='left', left_on=key, right_on=key)
df_test_merged = pd.merge(left=df_test_merged, right=datasets['credit_card_balance'], how='left', left_on=key, right_on=ke
df_test_merged = pd.merge(left=df_test_merged, right=datasets['installments_payments'], how='left', left_on=key, right_on=
df_test_merged = pd.merge(left=df_test_merged, right=datasets['bureau_balance'], how='left', left_on=key, right_on=key)
```

```python
df_test_merged.head(5)
```

	SK_ID_CURR	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDI
0	100001	Cash loans	F	N	Y	0	135000.0	56880
1	100005	Cash loans	M	N	Y	0	99000.0	22276
2	100013	Cash loans	M	Y	Y	0	202500.0	66326
3	100028	Cash loans	F	N	Y	2	315000.0	157500
4	100038	Cash loans	M	Y	N	1	180000.0	62550

5 rows × 186 columns

```python
# Cache the current grouped dataset as it takes time
df_test_merged.to_csv("/content/drive/MyDrive/Group32_AML/datasets/df_test_merged.csv")
```

```python
# # ONLY RUN if dynamically loading from cache
# # Done to avoid computation times
# df_test_merged = pd.read_csv("/content/drive/MyDrive/Group32_AML/datasets/df_test_merged.csv")
```

Convert categorical data to numerical and overwrite merged dataset

```python
df_test_dummies = pd.get_dummies(df_test_merged)
df_test_dummies
```

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT_x	AMT_ANNUITY_x	AMT_GOODS_PRICE_x	REGION_POPULATION_RELATIV
0	100001	0	135000.0	568800.0	20560.5	450000.0	0.01885
1	100005	0	99000.0	222768.0	17370.0	180000.0	0.03579
2	100013	0	202500.0	663264.0	69777.0	630000.0	0.01910
3	100028	2	315000.0	1575000.0	49018.5	1575000.0	0.02639
4	100038	1	180000.0	625500.0	32067.0	625500.0	0.01003
...	...	...	...	...	...	...	
48739	456221	0	121500.0	412560.0	17473.5	270000.0	0.00204
48740	456222	2	157500.0	622413.0	31909.5	495000.0	0.03579
48741	456223	1	202500.0	315000.0	33205.5	315000.0	0.02639
48742	456224	0	225000.0	450000.0	25128.0	450000.0	0.01885
48743	456250	0	135000.0	312768.0	24709.5	270000.0	0.00662

48744 rows × 307 columns

Remove unnecessary columns not present in `df_merged_dummies`

```python
columns_not_present = df_test_dummies.columns[~df_test_dummies.columns.isin(df_merged_dummies)]
columns_not_present
```

```
Index([], dtype='object')
```

```python
df_test_dummies.drop(columns=columns_not_present, inplace=True)
df_test_dummies
```

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT_x	AMT_ANNUITY_x	AMT_GOODS_PRICE_x	REGION_POPULATION_RELATIV
0	100001	0	135000.0	568800.0	20560.5	450000.0	0.01885
1	100005	0	99000.0	222768.0	17370.0	180000.0	0.03579
2	100013	0	202500.0	663264.0	69777.0	630000.0	0.01910
3	100028	2	315000.0	1575000.0	49018.5	1575000.0	0.02639

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT_x	AMT_ANNUITY_x	AMT_GOODS_PRICE_x	REGION_POPULATION_RELATIV
4	100038	1	180000.0	625500.0	32067.0	625500.0	0.01003
...	...	...	...	...	...	...	
48739	456221	0	121500.0	412560.0	17473.5	270000.0	0.00204
48740	456222	2	157500.0	622413.0	31909.5	495000.0	0.03579
48741	456223	1	202500.0	315000.0	33205.5	315000.0	0.02639
48742	456224	0	225000.0	450000.0	25128.0	450000.0	0.01885
48743	456250	0	135000.0	312768.0	24709.5	270000.0	0.00662

48744 rows × 307 columns

In [ ]:

```
duplicated_columns = df_test_dummies.columns[df_test_dummies.columns.duplicated()]
duplicated_columns
```

Out[ ]:

```
Index(['Unnamed: 0_x', 'SK_ID_PREV_x', 'Unnamed: 0_y', 'SK_ID_PREV_y'], dtype='object')
```

In [ ]:

```
# https://stackoverflow.com/questions/14984119/python-pandas-remove-duplicate-columns
df_test_dummies = df_test_dummies.loc[:, ~df_test_dummies.columns.duplicated()]
```

In [ ]:

```
df_test_dummies.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 48744 entries, 0 to 48743
Columns: 303 entries, SK_ID_CURR to EMERGENCYSTATE_MODE_Yes
dtypes: float64(126), int64(40), uint8(137)
memory usage: 68.5 MB
```

Cache the new data

In [ ]:

```
# Cache the current grouped dataset as it takes time
df_test_dummies.to_csv("/content/drive/MyDrive/Group32_AML/datasets/df_test_merged_interpolated.csv")
```

In [ ]:

```
# # ONLY RUN if dynamically loading from cache
# # Done to avoid computation times
# df_test_dummies = pd.read_csv("/content/drive/MyDrive/Group32_AML/datasets/df_test_merged_interpolated.csv")
```

Apply the Column Transformer on the merged test data

In [ ]:

```
from sklearn.compose import ColumnTransformer

# Create a basic numerical pipeline
num_pipeline = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean'))
])

# Column transform pipeline
data_pipeline = ColumnTransformer([
    ("num_pipeline", num_pipeline, df_test_dummies.columns)
], n_jobs = -1)

# Run the transform
data_transformed = data_pipeline.fit_transform(df_test_dummies)
```

In [ ]:

```
df_test_transformed = pd.DataFrame(data_transformed,  columns=df_test_dummies.columns)
del data_transformed
df_test_transformed
```

Out[ ]:

	Unnamed: 0	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT_x	AMT_ANNUITY_x	AMT_GOODS_PRICE_x	REGION_POPULAT
0	0.0	100001.0	0.0	135000.0	568800.0	20560.5	450000.0	
1	1.0	100005.0	0.0	99000.0	222768.0	17370.0	180000.0	
2	2.0	100013.0	0.0	202500.0	663264.0	69777.0	630000.0	
3	3.0	100028.0	2.0	315000.0	1575000.0	49018.5	1575000.0	
4	4.0	100038.0	1.0	180000.0	625500.0	32067.0	625500.0	

	Unnamed: 0	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT_x	AMT_ANNUITY_x	AMT_GOODS_PRICE_x	REGION_POPULAT
...	...	...	...	...	...	...	...	
48739	48739.0	456221.0	0.0	121500.0	412560.0	17473.5	270000.0	
48740	48740.0	456222.0	2.0	157500.0	622413.0	31909.5	495000.0	
48741	48741.0	456223.0	1.0	202500.0	315000.0	33205.5	315000.0	
48742	48742.0	456224.0	0.0	225000.0	450000.0	25128.0	450000.0	
48743	48743.0	456250.0	0.0	135000.0	312768.0	24709.5	270000.0	

48744 rows × 308 columns

Create new features for test data

In [ ]:

```
# We've added one to certain terms to avoid division by zero error
df_test_transformed['F1_RCV:BAL'] = df_test_transformed['AMT_TOTAL_RECEIVABLE'] / (df_test_transformed['AMT_BALANCE'] + 1)
df_test_transformed['F2_TRC:RCV'] = df_test_transformed['AMT_TOTAL_RECEIVABLE'] / (df_test_transformed['AMT_RECIVABLE'] + 1
df_test_transformed['F3_TRC:RCP'] = df_test_transformed['AMT_TOTAL_RECEIVABLE'] / (df_test_transformed['AMT_RECEIVABLE_PRI
df_test_transformed['F4_BAL:RCV'] = df_test_transformed['AMT_BALANCE'] / (df_test_transformed['AMT_RECIVABLE'] + 1)
df_test_transformed['F5_BAL:RCP'] = df_test_transformed['AMT_BALANCE'] / (df_test_transformed['AMT_RECEIVABLE_PRINCIPAL'] +
df_test_transformed['F6_RCV:RCP'] = df_test_transformed['AMT_RECIVABLE'] / (df_test_transformed['AMT_RECEIVABLE_PRINCIPAL']
df_test_transformed['F7_EXT:RCV'] = (df_test_transformed['EXT_SOURCE_1'] + df_test_transformed['EXT_SOURCE_2'] + df_test_t
df_test_transformed['F8_SUM_EXT'] = (df_test_transformed['EXT_SOURCE_1'] + df_test_transformed['EXT_SOURCE_2'] + df_test_t
df_test_transformed['F9_PRD_EXT'] = (df_test_transformed['EXT_SOURCE_1'] * df_test_transformed['EXT_SOURCE_2'] * df_test_t
df_test_transformed['F10_EXT_RATIO'] = (df_test_transformed['EXT_SOURCE_1'] + df_test_transformed['EXT_SOURCE_2'] * 2 + df_
df_test_transformed['F11_MAX_EXT'] = [max(x, y, z) for x, y, z in zip(df_test_transformed['EXT_SOURCE_1'], df_test_transfo
```

Prediction

In [ ]:

```
# Get the X (features)
X = df_test_transformed[columns_greater_than_5]
```

In [ ]:

```
# Get the values
X = X.values
```

Predict using the best model (Random Forest classifer)

In [ ]:

```
# Predict for the new test data
result = pipe_rf.predict(X)

# Get the prob
result_prob = pipe.predict_proba(X)

# Create data frame for result
df_result = pd.DataFrame(result, columns=['result'])

# Prepare the final submission
df_result[['class_0_prob', 'class_1_prob']] = result_prob
final_submission = pd.DataFrame()

# Set columns SK_ID_CURR and TARGET with respective values
final_submission['SK_ID_CURR'] = df_test['SK_ID_CURR']
final_submission['TARGET'] = df_result['class_1_prob']

# Set the index as SK_ID_CURR
final_submission = final_submission.set_index('SK_ID_CURR')
```

In [ ]:

```
final_submission
```

Out[ ]:

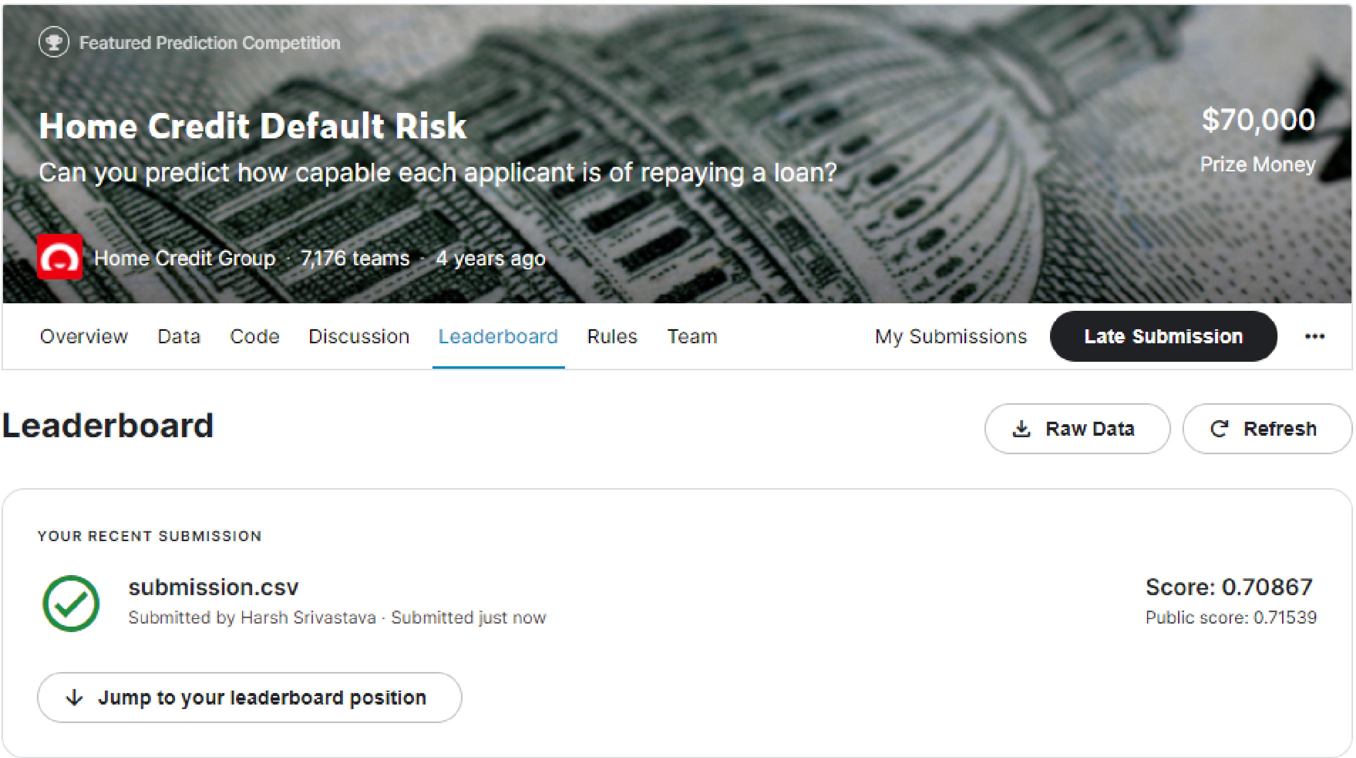
	TARGET
SK_ID_CURR	
100001	0.045167
100005	0.166401
100013	0.027187
100028	0.046405
100038	0.123737

	TARGET
SK_ID_CURR	
...	...
456221	0.041753
456222	0.051881
456223	0.070979
456224	0.052969
456250	0.112996

48744 rows × 1 columns

In [ ]:

```
final_submission.to_csv("submission.csv")
```



PHASE 3 - Neural Networks

We are going to build a neural network model with four layers and modifiable neuron counts per layer.

1. In this phase we will first run a basic neural network with the original columns for the Phase 2 filtered dataset. Based on previous correlations we will increase the threshold on correlation to filter out columns that do not contribute to our model.
2. We will then build a different model on the filtered columns and then perform validation. If the accuracy is higher we will continue, else conclude that the model we have is the best so far.

Make Elementary Neural Network Model

Scale and normalize the training data

In [ ]:

```
df_train_scaled = df_transformed[columns_greater_than_5]
```

In [ ]:

```
pd.DataFrame(data=zip(df_train_scaled.columns, df_train_scaled.min(), df_train_scaled.max()), columns=["Column", "Min", "Ma
```

Out[ ]:

	Column	Min	Max
0	F10_EXT_RATIO	-2.014893	6.937189
1	F8_SUM_EXT	-0.830280	3.945919
2	EXT_SOURCE_2	-0.194183	1.015776
3	F9_PRD_EXT	-0.821023	1.672735

	Column	Min	Max
4	EXT_SOURCE_3	-0.801118	1.985320
5	F11_MAX_EXT	0.025899	2.477270
6	DAYS_BIRTH	-25229.000000	-7489.000000
7	DAYS_CREDIT	-5425.810119	2620.706962
8	REGION_RATING_CLIENT_W_CITY	1.000000	3.000000
9	REGION_RATING_CLIENT	1.000000	3.000000
10	NAME_INCOME_TYPE_Working	0.000000	1.000000
11	NAME_EDUCATION_TYPE_Higher education	0.000000	1.000000
12	DAYS_LAST_PHONE_CHANGE	-4292.000000	0.000000
13	CODE_GENDER_M	0.000000	1.000000
14	CODE_GENDER_F	0.000000	1.000000
15	DAYS_CREDIT_UPDATE	-41890.000000	2480.240088
16	DAYS_ID_PUBLISH	-7197.000000	0.000000
17	REG_CITY_NOT_WORK_CITY	0.000000	1.000000

In [ ]:

```
# Now we do selective column scaling
df_train_scaled.std()
```

Out[ ]:

```
F10_EXT_RATIO                             0.821975
F8_SUM_EXT                                0.420202
EXT_SOURCE_2                              0.191119
F9_PRD_EXT                                0.118901
EXT_SOURCE_3                              0.205426
F11_MAX_EXT                               0.164373
DAYS_BIRTH                             4363.988872
DAYS_CREDIT                             647.889361
REGION_RATING_CLIENT_W_CITY               0.502736
REGION_RATING_CLIENT                      0.509034
NAME_INCOME_TYPE_Working                  0.499734
NAME_EDUCATION_TYPE_Higher education      0.429160
DAYS_LAST_PHONE_CHANGE                  826.816032
CODE_GENDER_M                             0.474263
CODE_GENDER_F                             0.474263
DAYS_CREDIT_UPDATE                      494.223739
DAYS_ID_PUBLISH                        1509.452794
REG_CITY_NOT_WORK_CITY                    0.421125
dtype: float64
```

Since the standard deviation is looking uniform just like test data (look below during Kaggle submission), we simply can use standard scaler and then train our model on it.

In [ ]:

```
standard_scaler = StandardScaler()
df_train_scaled = pd.DataFrame(standard_scaler.fit_transform(df_train_scaled), columns=df_train_scaled.columns)
```

Build the neural network.

In [ ]:

```
import torch
import torch.nn as nn
import torch.nn.functional as F
from torch.utils.tensorboard import SummaryWriter
```

In [ ]:

```
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
writer = SummaryWriter("/content/drive/MyDrive/Group32_AML/runs/")
```

In [ ]:

```
X = df_train_scaled.values
y = df_merged_dummies["TARGET"].values
```

In [ ]:

```
X_train, X_validation, y_train, y_validation = train_test_split(X, y, test_size=0.2, random_state=42)

X_train = torch.FloatTensor(X_train)
X_validation = torch.FloatTensor(X_validation)
y_train = torch.tensor(y_train, dtype=torch.long, device=device)
y_validation = torch.tensor(y_validation, dtype=torch.long, device=device)
```

In [ ]:

```
class NeuralNetworkModel(nn.Module):
```

```
    def __init__(self, input_features=18, hidden_layer1=50, hidden_layer2=80, hidden_layer3=32, hidden_layer4=10, out_featu
        super().__init__()
        self.f_connected1 = nn.Linear(input_features, hidden_layer1)
        self.f_connected2 = nn.Linear(hidden_layer1, hidden_layer2)
        self.f_connected3 = nn.Linear(hidden_layer2, hidden_layer3)
        self.f_connected4 = nn.Linear(hidden_layer3, hidden_layer4)
        self.out = nn.Linear(hidden_layer4, out_features)

    def forward(self,x):
        x = F.leaky_relu(self.f_connected1(x))
        x = F.dropout(x, p=0.4)
        x = F.leaky_relu(self.f_connected2(x))
        x = F.leaky_relu(self.f_connected3(x))
        x = F.leaky_relu(self.f_connected4(x))
        x = self.out(x)
        return x
```

Step 1 - Elementaty Model

In [ ]:

```
torch.manual_seed(20)
model = NeuralNetworkModel().to(device)
```

In [ ]:

```
loss_function = nn.CrossEntropyLoss()
#optimizer = torch.optim.Adadelta(model.parameters(),lr=0.001)

optimizer = torch.optim.Adam(model.parameters(),lr=0.001)
# optimizer = torch.optim.Adamax(model.parameters(),lr=0.001)
```

In [ ]:

```
row_count = y_train.shape[0]
epochs = 1000
final_losses = []
last_accuracy = 0.0
for i in range(epochs):
    i += 1
    y_pred = model.forward(X_train)
    loss = loss_function(y_pred, y_train)
    final_losses.append(loss.item())
    _,predicted = torch.max(y_pred, 1)
    running_correct = (predicted == y_train).sum().item()
    accuracy = running_correct / row_count

    if i % 100 == 1:
        print("Epoch: {} and the loss: {} accuracy : {}".format(i, loss.item(), accuracy))
        writer.add_scalar('Training loss', loss.item(), i)
        writer.add_scalar('Accuracy', running_correct / row_count, i)

        if np.round(accuracy, 5) == np.round(last_accuracy):
            break

    last_accuracy = accuracy

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

```
Epoch: 1 and the loss: 0.25475820899009705 accuracy : 0.9195243902439024
Epoch: 101 and the loss: 0.25448888540267944 accuracy : 0.9195243902439024
Epoch: 201 and the loss: 0.25433599948883057 accuracy : 0.9195243902439024
Epoch: 301 and the loss: 0.2541849911212921 accuracy : 0.9195243902439024
Epoch: 401 and the loss: 0.2540402114391327 accuracy : 0.9195243902439024
Epoch: 501 and the loss: 0.2538731098175049 accuracy : 0.9195243902439024
Epoch: 601 and the loss: 0.2537517547607422 accuracy : 0.9195243902439024
Epoch: 701 and the loss: 0.25354281067848206 accuracy : 0.9195243902439024
Epoch: 801 and the loss: 0.2535628080368042 accuracy : 0.9195243902439024
Epoch: 901 and the loss: 0.25340819358825684 accuracy : 0.9195243902439024
```

In [ ]:

```
torch.save(model, "/content/drive/MyDrive/Group32_AML/NeuralNetwork18Features.model")
```

In [ ]:

```
# # Load model from cache
# model = torch.load("/content/drive/MyDrive/Group32_AML/NeuralNetwork18Features.model")
```

In [ ]:

```
predictions = []
with torch.no_grad():
    for i, data in enumerate(X_validation):
        y_pred = model(data)
        predictions.append(y_pred.argmax().item())
```

In [ ]:
```
from sklearn.metrics import confusion_matrix, roc_auc_score
cm = confusion_matrix(y_validation, predictions)
cm
```

Out[ ]:
```
array([[56474,     0],
       [ 5026,     0]])
```

In [ ]:
```
from sklearn.metrics import accuracy_score
score = accuracy_score(y_validation, predictions)
score
```

Out[ ]:
```
0.9182764227642276
```

In [ ]:
```
model_results.append(["Neural Network (18 features)", score, last_accuracy])
```

In [ ]:
```
writer.close()
writer = SummaryWriter("/content/drive/MyDrive/Group32_AML/runs/")
```

Our accuracy is still quite good, but less than that of Logistic Regression from Phase 2 (0.9195). So now we move to step 2 and refine our columns.

Step 2 - Refine Columns (Iteration 1)

Using our previously formed df_transformed with all columns, we find the highly correlated features once again.

In [ ]:
```
all_corr_target = df_transformed.corr()["TARGET"]
all_corr_target
```

Out[ ]:
```
Unnamed: 0         -0.002140
SK_ID_CURR         -0.002137
TARGET              1.000000
CNT_CHILDREN        0.019143
AMT_INCOME_TOTAL   -0.003970
                      ...
F7_EXT:RCV         -0.023481
F8_SUM_EXT         -0.170612
F9_PRD_EXT         -0.138242
F10_EXT_RATIO      -0.191394
F11_MAX_EXT        -0.104619
Name: TARGET, Length: 315, dtype: float64
```

In [ ]:
```
all_corr_target = all_corr_target.sort_values(key=abs,ascending=False).reset_index()
all_corr_target.columns = ["Feature", "Correlation"]
all_corr_target
```

Out[ ]:

	Feature	Correlation
0	TARGET	1.000000
1	F10_EXT_RATIO	-0.191394
2	F8_SUM_EXT	-0.170612
3	EXT_SOURCE_2	-0.160066
4	F9_PRD_EXT	-0.138242
...	...	...
310	AMT_REQ_CREDIT_BUREAU_HOUR	-0.000139
311	ORGANIZATION_TYPE_Advertising	0.000118
312	ORGANIZATION_TYPE_Industry: type 7	-0.000093
313	AMT_DRAWINGS_POS_CURRENT	-0.000083
314	SK_ID_PREV_y	-0.000005

315 rows × 2 columns

In [ ]:
```
columns_all_highly_correlated = all_corr_target[abs(all_corr_target["Correlation"]) > 0.1][1:]
columns_all_highly_correlated
```

Out[ ]:

	Feature	Correlation
1	F10_EXT_RATIO	-0.191394
2	F8_SUM_EXT	-0.170612

	Feature	Correlation
3	EXT_SOURCE_2	-0.160066
4	F9_PRD_EXT	-0.138242
5	EXT_SOURCE_3	-0.133961
6	F11_MAX_EXT	-0.104619

New model with less columns above 0.06 correlation

In [ ]:
```
# Get just the TARGET column
y = df_transformed["TARGET"].values
```

In [ ]:
```
# Get the X (features) from df_test_transformed
X_new = df_train_scaled[columns_all_highly_correlated["Feature"]]
X_new
```

Out[ ]:

	F10_EXT_RATIO	F8_SUM_EXT	EXT_SOURCE_2	F9_PRD_EXT	EXT_SOURCE_3	F11_MAX_EXT
0	-2.480035	-2.482697	-1.315775	-1.128997	-1.810437	-2.584569
1	0.077469	-0.167967	0.564198	-0.300201	0.064327	-0.398705
2	0.879380	0.582511	0.217116	0.509607	1.062574	0.254206
3	1.488839	1.260881	0.711729	1.515850	1.358001	0.623416
4	0.623172	0.575884	-1.002934	0.266983	1.124304	0.331353
...	...	...	...	...	...	...
307495	-0.126834	-0.516958	0.874930	-0.750545	-0.132025	-0.037413
307496	-2.278210	-1.908189	-2.084705	-1.080171	-1.636549	-1.533710
307497	-0.722088	-0.071363	0.111473	-0.420911	-1.423517	0.342174
307498	1.029447	1.301656	-0.001331	1.419068	0.728911	1.293299
307499	-0.696153	0.067484	1.015872	-0.655966	-1.934342	0.283973

307500 rows × 6 columns

In [ ]:
```
X_new = X_new.values
```

Build new model

In [ ]:
```
class NeuralNetworkModel6Columns(nn.Module):
    def __init__(self, input_features=6, hidden_layer1=50, hidden_layer2=32, hidden_layer3=16, out_features=2):
        super().__init__()
        self.f_connected1 = nn.Linear(input_features, hidden_layer1)
        self.f_connected2 = nn.Linear(hidden_layer1, hidden_layer2)
        self.f_connected3 = nn.Linear(hidden_layer2, hidden_layer3)
        self.out = nn.Linear(hidden_layer3, out_features)

    def forward(self,x):
        x = F.leaky_relu(self.f_connected1(x))
        x = F.dropout(x, p=0.4)
        x = F.leaky_relu(self.f_connected2(x))
        x = F.leaky_relu(self.f_connected3(x))
        x = self.out(x)
        return x
```

Train model on the newly filtered data columns.

In [ ]:
```
X_train, X_validation, y_train, y_validation = train_test_split(X_new, y, test_size=0.2, random_state=42)

X_train = torch.FloatTensor(X_train)
X_validation = torch.FloatTensor(X_validation)
y_train = torch.tensor(y_train, dtype=torch.long, device=device)
y_validation = torch.tensor(y_validation, dtype=torch.long, device=device)
```

In [ ]:
```
torch.manual_seed(20)
# new_model = NeuralNetworkModel(input_features=9, hidden_layer1=100, hidden_layer2=80, hidden_layer3=40, hidden_layer4=16,
new_model = NeuralNetworkModel6Columns().to(device)
```

In [ ]:
```
new_loss_function = nn.CrossEntropyLoss()
```

```
#optimizer = torch.optim.Adadelta(model.parameters(),lr=0.001)

new_optimizer = torch.optim.Adam(new_model.parameters(),lr=0.001)
# optimizer = torch.optim.Adamax(model.parameters(),lr=0.001)
```

In [ ]:

```
row_count = y_train.shape[0]
epochs = 1000
final_losses = []
last_accuracy = 0.0
for i in range(epochs):
    i += 1
    y_pred = new_model.forward(X_train)
    loss = loss_function(y_pred, y_train)
    final_losses.append(loss.item())
    _,predicted = torch.max(y_pred, 1)
    running_correct = (predicted == y_train).sum().item()
    accuracy = running_correct / row_count

    if i % 100 == 1:
        print("Epoch: {} and the loss: {} accuracy : {}".format(i, loss.item(), accuracy))
        writer.add_scalar('Training loss', loss.item(), i)
        writer.add_scalar('Accuracy', running_correct / row_count, i)

        if np.round(accuracy, 5) == np.round(last_accuracy):
            break

    last_accuracy = accuracy

    optimizer.zero_grad()
    loss.backward()
    optimizer.step()
```

```
Epoch: 1 and the loss: 0.7414618134498596 accuracy : 0.10036585365853659
Epoch: 101 and the loss: 0.7415406107902527 accuracy : 0.10043089430894309
Epoch: 201 and the loss: 0.7414897084236145 accuracy : 0.10036991869918699
Epoch: 301 and the loss: 0.7415586709976196 accuracy : 0.09965853658536586
Epoch: 401 and the loss: 0.7415090203285217 accuracy : 0.10021138211382113
Epoch: 501 and the loss: 0.7415147423744202 accuracy : 0.10031707317073171
Epoch: 601 and the loss: 0.7415100932121277 accuracy : 0.1001869918699187
Epoch: 701 and the loss: 0.741520881652832 accuracy : 0.10044715447154472
Epoch: 801 and the loss: 0.7415565252304077 accuracy : 0.10013821138211382
Epoch: 901 and the loss: 0.741543710231781 accuracy : 0.10043089430894309
```

In [105…

```
torch.save(new_model, "/content/drive/MyDrive/Group32_AML/NeuralNetwork6Features.model")
```

In [106…

```
# # Load model from cache
# new_model = torch.load("/content/drive/MyDrive/Group32_AML/NeuralNetwork9Features.model")
```

In [107…

```
predictions = []
with torch.no_grad():
    for i, data in enumerate(X_validation):
        y_pred = new_model(data)
        predictions.append(y_pred.argmax().item())
```

In [108…

```
from sklearn.metrics import confusion_matrix, roc_auc_score
cm = confusion_matrix(y_validation, predictions)
cm
```

Out[108…

```
array([[ 1650, 54824],
       [  492,  4534]])
```

In [109…

```
from sklearn.metrics import accuracy_score
score = accuracy_score(y_validation, predictions)
score
```

Out[109…

```
0.10055284552845528
```

In [112…

```
model_results.append(["Neural Network (6 features)", score, last_accuracy])
```

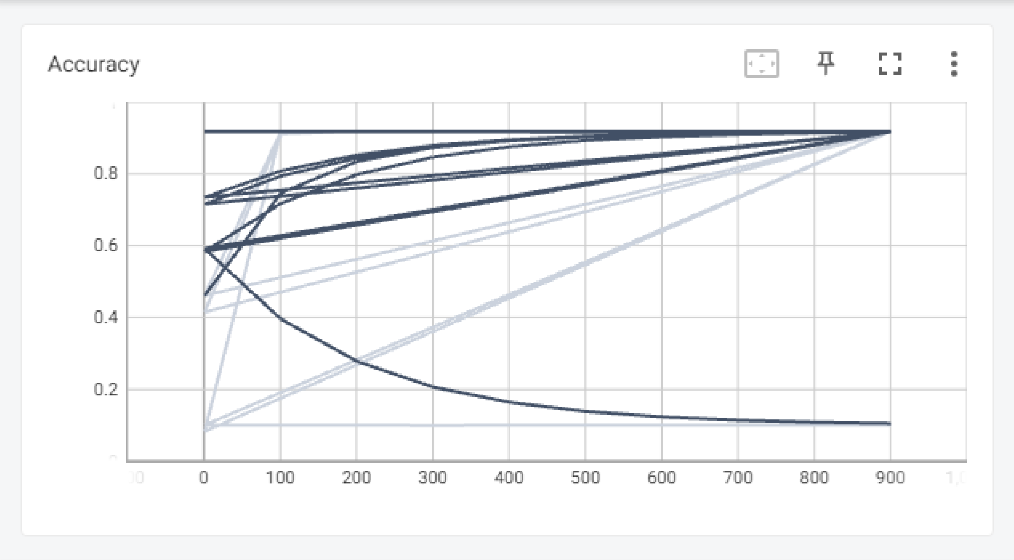
Generate tensorboard plots

In [ ]:

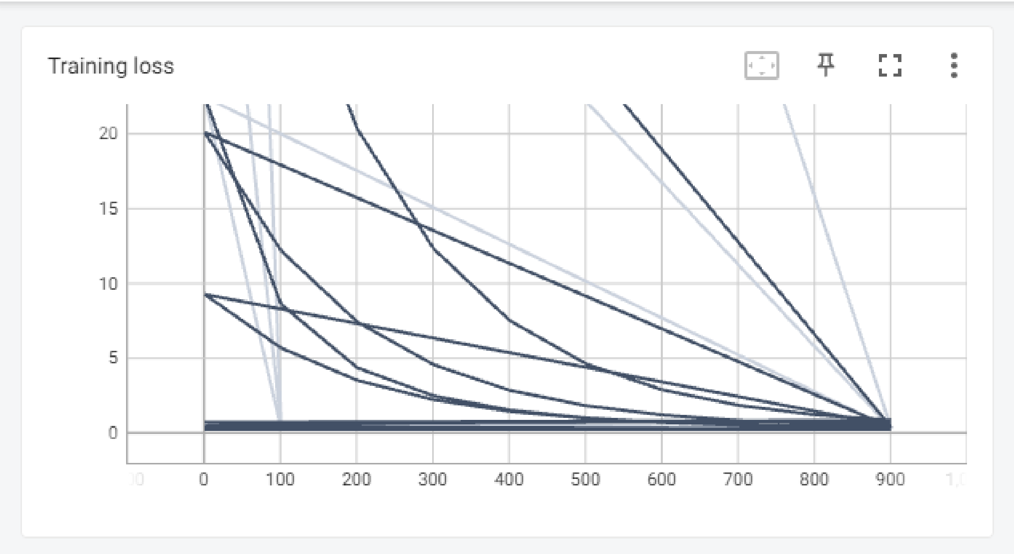
```
writer.close()
```

TensorBoard graphs (for all runs we did)

Accuracy

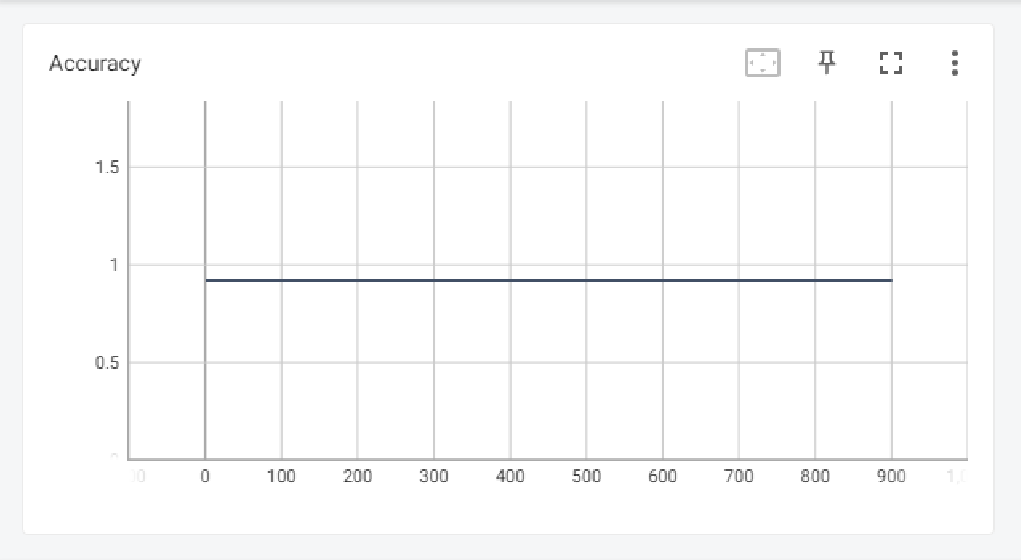


Training loss

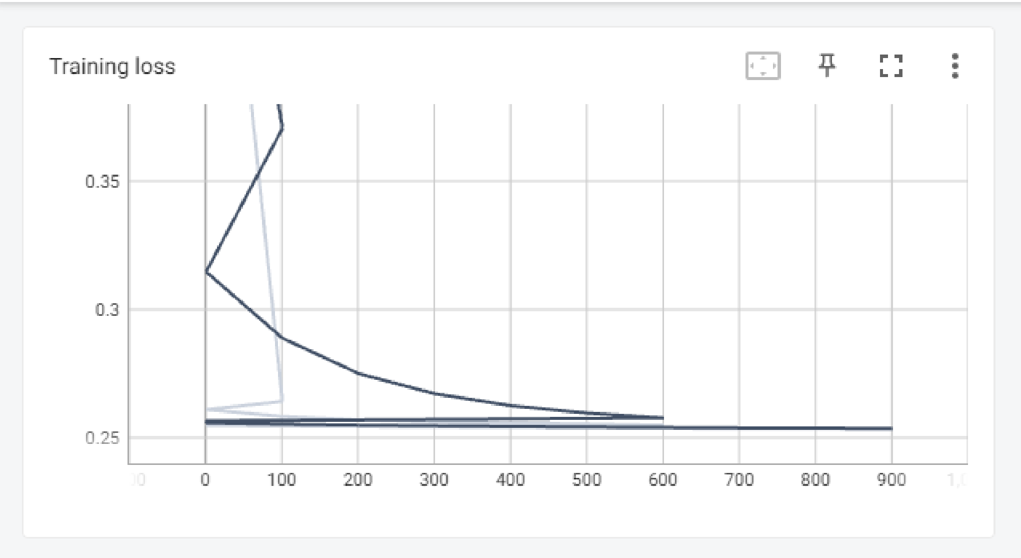


Most successful results on TensorBoard

Accuracy

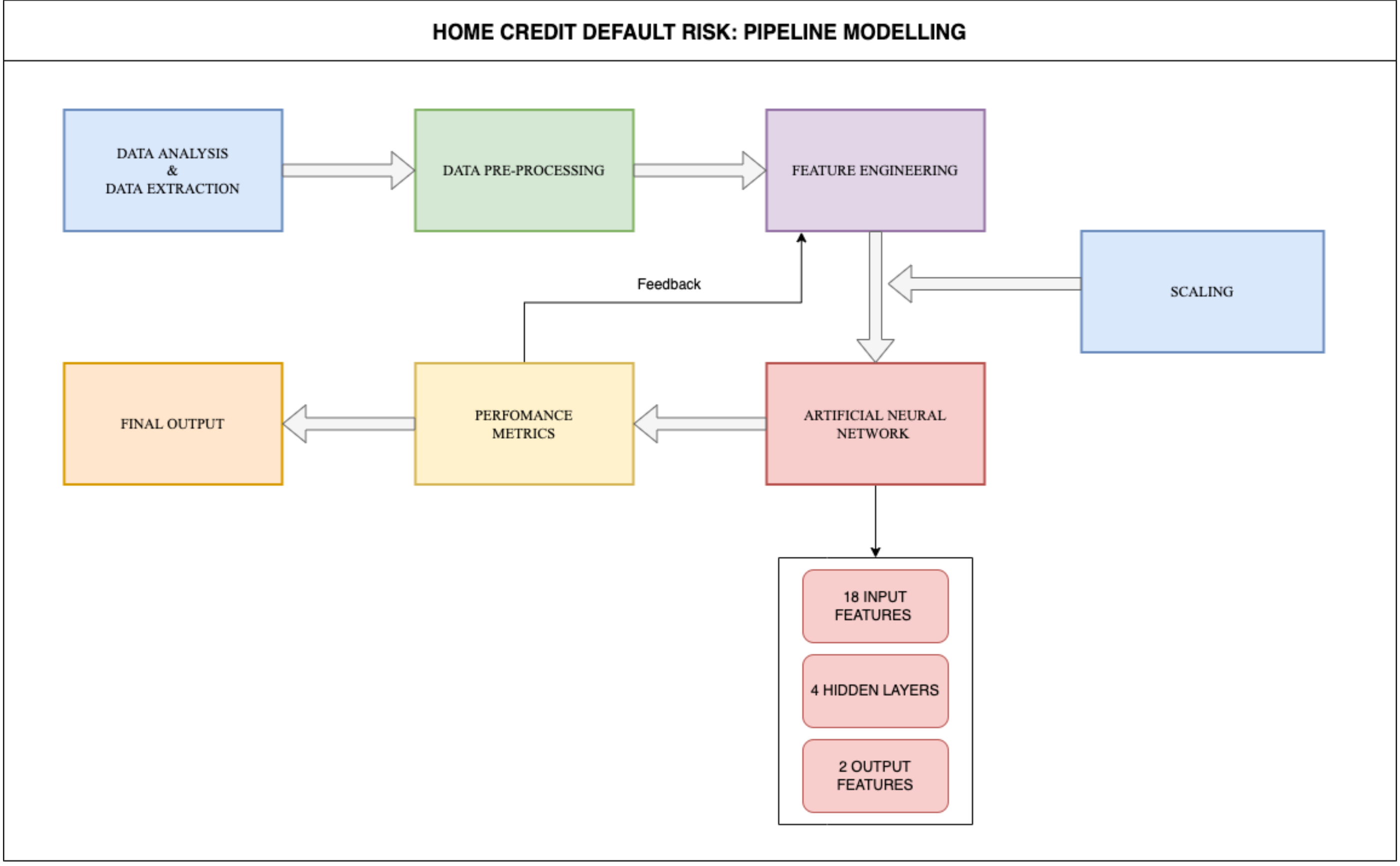


Training loss



Results and Discussion:

Final Pipeline(Diagram)



**Number of input features*

We've taken 18 features as input and hidden layers as 50, 80, 32, 10 and 2 output features

Loss function used: Cross entropy loss

Number of experiments conducted

A total of 14 experiments were conducted, but got overwritten as we only had sections for two models once we ran the final notebook. We ran different models with different layers of perceptrons and layer counts and chose the best as the one with 18 input features and 50, 80, 32, 10 hidden layers.

In [116…

```
pd.DataFrame(model_results, columns = ["Model Name", "Training Accuracy", "Validation Accuracy"])
```

Out[116…

	Model Name	Training Accuracy	Validation Accuracy
0	Logistic Regression	0.919592	0.917919
1	Naive Bayes (Gaussian)	0.135722	0.136689
2	Random Forest	0.999939	0.918088
3	Logistic Regression (Phase 2)	0.919579	0.917892
4	Random Forest (Phase 2)	0.997778	0.980267
5	Neural Network (18 features)	0.919524	0.918276
6	Neural Network (6 features)	0.100430	0.100552

Inference for Phase 3

We have tried two models each with different layers of perceptrons and with different feature counts. Upon validating both models we see that the first model gives a better validation accuracy (0.918), and the second model (with 6 features), doesn't give a good accuracy.

Our inference is that the first model is the best fit and is not overfitting because we also tried it with lesser columns but it gives us the best accuracy at the 18 columns from Phase 2. Now second model is definitely under-fitting because of the accuracy.

> Therefore we will proceed with the Neural Network model to get the Kaggle score.

Based on our results, we can see that the accuracy of logistic regression and Neural Network (18 features) is very close (~0.919). Based on this we can conclude both models to be a good choice. However, when tested with the validation set we get a slightly higher accuracy with the Neural Network.

KAGGLE SUBMISSION - PHASE 3

We will load the saved `df_test_merged_interpolated.csv` into df_test_dummies (which was the table from Phase 2 with the final filtered 18 columns)

In [ ]:
```
# Reload test data from saved cache
df_test_dummies = pd.read_csv("/content/drive/MyDrive/Group32_AML/datasets/df_test_merged_interpolated.csv")
```

Transform the data

In [ ]:
```
from sklearn.compose import ColumnTransformer

# Create a basic numerical pipeline
num_pipeline = Pipeline(steps=[
    ('imputer', SimpleImputer(strategy='mean'))
])

# Column transform pipeline
data_pipeline = ColumnTransformer([
    ("num_pipeline", num_pipeline, df_test_dummies.columns)
], n_jobs = -1)

# Run the transform
data_transformed = data_pipeline.fit_transform(df_test_dummies)
df_test_transformed = pd.DataFrame(data_transformed,  columns=df_test_dummies.columns)
del data_transformed
df_test_transformed
```

Out[ ]:

	Unnamed: 0	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT_x	AMT_ANNUITY_x	AMT_GOODS_PRICE_x	REGION_POPULAT
0	0.0	100001.0	0.0	135000.0	568800.0	20560.5	450000.0	
1	1.0	100005.0	0.0	99000.0	222768.0	17370.0	180000.0	
2	2.0	100013.0	0.0	202500.0	663264.0	69777.0	630000.0	
3	3.0	100028.0	2.0	315000.0	1575000.0	49018.5	1575000.0	
4	4.0	100038.0	1.0	180000.0	625500.0	32067.0	625500.0	
...	...	...	...	...	...	...	...	
48739	48739.0	456221.0	0.0	121500.0	412560.0	17473.5	270000.0	
48740	48740.0	456222.0	2.0	157500.0	622413.0	31909.5	495000.0	
48741	48741.0	456223.0	1.0	202500.0	315000.0	33205.5	315000.0	
48742	48742.0	456224.0	0.0	225000.0	450000.0	25128.0	450000.0	
48743	48743.0	456250.0	0.0	135000.0	312768.0	24709.5	270000.0	

48744 rows × 308 columns

In [ ]:
```
# Recreate new features
df_test_transformed['F1_RCV:BAL'] = df_test_transformed['AMT_TOTAL_RECEIVABLE'] / (df_test_transformed['AMT_BALANCE'] + 1)
df_test_transformed['F2_TRC:RCV'] = df_test_transformed['AMT_TOTAL_RECEIVABLE'] / (df_test_transformed['AMT_RECIVABLE'] + 1
df_test_transformed['F3_TRC:RCP'] = df_test_transformed['AMT_TOTAL_RECEIVABLE'] / (df_test_transformed['AMT_RECEIVABLE_PRI
df_test_transformed['F4_BAL:RCV'] = df_test_transformed['AMT_BALANCE'] / (df_test_transformed['AMT_RECIVABLE'] + 1)
df_test_transformed['F5_BAL:RCP'] = df_test_transformed['AMT_BALANCE'] / (df_test_transformed['AMT_RECEIVABLE_PRINCIPAL'] +
df_test_transformed['F6_RCV:RCP'] = df_test_transformed['AMT_RECIVABLE'] / (df_test_transformed['AMT_RECEIVABLE_PRINCIPAL']
df_test_transformed['F7_EXT:RCV'] = (df_test_transformed['EXT_SOURCE_1'] + df_test_transformed['EXT_SOURCE_2'] + df_test_tr
df_test_transformed['F8_SUM_EXT'] = (df_test_transformed['EXT_SOURCE_1'] + df_test_transformed['EXT_SOURCE_2'] + df_test_tr
df_test_transformed['F9_PRD_EXT'] = (df_test_transformed['EXT_SOURCE_1'] * df_test_transformed['EXT_SOURCE_2'] * df_test_tr
df_test_transformed['F10_EXT_RATIO'] = (df_test_transformed['EXT_SOURCE_1'] + df_test_transformed['EXT_SOURCE_2'] * 2 + df_
df_test_transformed['F11_MAX_EXT'] = [max(x, y, z) for x, y, z in zip(df_test_transformed['EXT_SOURCE_1'], df_test_transfor
```

In [ ]:
```
df_test_transformed.drop(columns=["Unnamed: 0"], inplace=True)
```

Scale the data

In [ ]:
```
df_test_scaled = df_test_transformed[columns_greater_than_5].round(6)
```

In [ ]:
```
pd.DataFrame(data=zip(df_test_scaled.columns, df_test_scaled.min(), df_test_scaled.max()), columns=["Column", "Min", "Max"]
```

Out[ ]:

	Column	Min	Max
0	F10_EXT_RATIO	0.184264	4.895597
1	F8_SUM_EXT	0.074324	2.463086
2	EXT_SOURCE_2	0.000008	0.855000
3	F9_PRD_EXT	0.000000	0.548405
4	EXT_SOURCE_3	0.000527	0.882530
5	F11_MAX_EXT	0.053478	0.939145
6	DAYS_BIRTH	-25195.000000	-7338.000000
7	DAYS_CREDIT	-2921.000000	0.000000
8	REGION_RATING_CLIENT_W_CITY	-1.000000	3.000000
9	REGION_RATING_CLIENT	1.000000	3.000000
10	NAME_INCOME_TYPE_Working	0.000000	1.000000
11	NAME_EDUCATION_TYPE_Higher education	0.000000	1.000000
12	DAYS_LAST_PHONE_CHANGE	-4361.000000	0.000000
13	CODE_GENDER_M	0.000000	1.000000
14	CODE_GENDER_F	0.000000	1.000000
15	DAYS_CREDIT_UPDATE	-2907.000000	0.000000
16	DAYS_ID_PUBLISH	-6348.000000	0.000000
17	REG_CITY_NOT_WORK_CITY	0.000000	1.000000

In [ ]:
```
df_test_scaled.std()
```

Out[ ]:
```
F10_EXT_RATIO                             0.700897
F8_SUM_EXT                                0.327164
EXT_SOURCE_2                              0.181263
F9_PRD_EXT                                0.086696
EXT_SOURCE_3                              0.171825
F11_MAX_EXT                               0.114472
DAYS_BIRTH                             4325.900393
DAYS_CREDIT                             541.857595
REGION_RATING_CLIENT_W_CITY               0.515804
REGION_RATING_CLIENT                      0.522694
NAME_INCOME_TYPE_Working                  0.499994
NAME_EDUCATION_TYPE_Higher education      0.436856
DAYS_LAST_PHONE_CHANGE                  878.920740
CODE_GENDER_M                             0.470073
CODE_GENDER_F                             0.470073
DAYS_CREDIT_UPDATE                      408.703135
DAYS_ID_PUBLISH                        1569.276709
REG_CITY_NOT_WORK_CITY                    0.417365
dtype: float64
```

In [ ]:
```
standard_scaler = StandardScaler()
df_test_scaled = pd.DataFrame(standard_scaler.fit_transform(df_test_scaled), columns=df_train_scaled.columns)
```

In [ ]:
```
df_test_scaled
```

Out[ ]:

	F10_EXT_RATIO	F8_SUM_EXT	EXT_SOURCE_2	F9_PRD_EXT	EXT_SOURCE_3	F11_MAX_EXT	DAYS_BIRTH	DAYS_CREDIT	REGION_RATING_C
0	-0.323955	0.557773	1.498570	-0.462033	-1.982191	1.325525	-0.733477	3.606020e-01	
1	-0.842291	-0.702100	-1.248832	-0.732613	-0.390773	-0.637109	-0.461392	1.689378e+00	
2	0.993292	0.894519	1.002783	0.916162	0.645347	0.540459	-0.917718	-1.444319e+00	
3	0.493176	0.393716	-0.046033	0.338172	0.655316	-0.220286	0.483623	-1.032767e+00	
4	-0.690127	-1.196257	-0.509397	-1.059170	0.000002	-1.203927	0.699997	-6.451606e-10	
...	...	...	...	...	...	...	...	...	
48739	0.984274	0.835899	0.720252	0.855386	0.831789	0.093078	-0.901998	8.330557e-01	
48740	0.475323	0.509152	0.918976	0.423667	0.000002	0.407752	1.128582	-6.451606e-10	
48741	-0.267319	0.399432	0.633058	-0.036646	-1.259400	0.834997	0.033770	1.298127e+00	

	F10_EXT_RATIO	F8_SUM_EXT	EXT_SOURCE_2	F9_PRD_EXT	EXT_SOURCE_3	F11_MAX_EXT	DAYS_BIRTH	DAYS_CREDIT	REGION_RATING_C
48742	0.019008	-0.321125	-0.398982	-0.413437	0.554934	-0.370963	0.485473	-6.802725e-01	
48743	-1.151218	-0.884741	-0.339179	-0.837324	-1.326784	-1.194545	0.486860	4.215043e-01	

48744 rows × 18 columns

Prediction

In [ ]:

```
# Initialize X and y
X = df_test_scaled # [columns_all_highly_correlated["Feature"]]
X_test = torch.FloatTensor(X.values)
```

In [ ]:

```
predictions = []
probs = []

with torch.no_grad():
    for i, data in enumerate(X_test):
        # Predict item
        y_pred = model(data)

        # Get probability and prediction
        probs.append(F.softmax(y_pred)[1].item())
        predictions.append(y_pred.argmax().item())
```

In [ ]:

```
# Prepare submission
final_submission = pd.DataFrame()

# Set columns SK_ID_CURR and TARGET with respective values
final_submission['SK_ID_CURR'] = df_test_transformed['SK_ID_CURR'].astype(int)
final_submission['TARGET'] = probs

# Set the index as SK_ID_CURR
final_submission = final_submission.set_index('SK_ID_CURR')
```

In [ ]:

```
final_submission
```

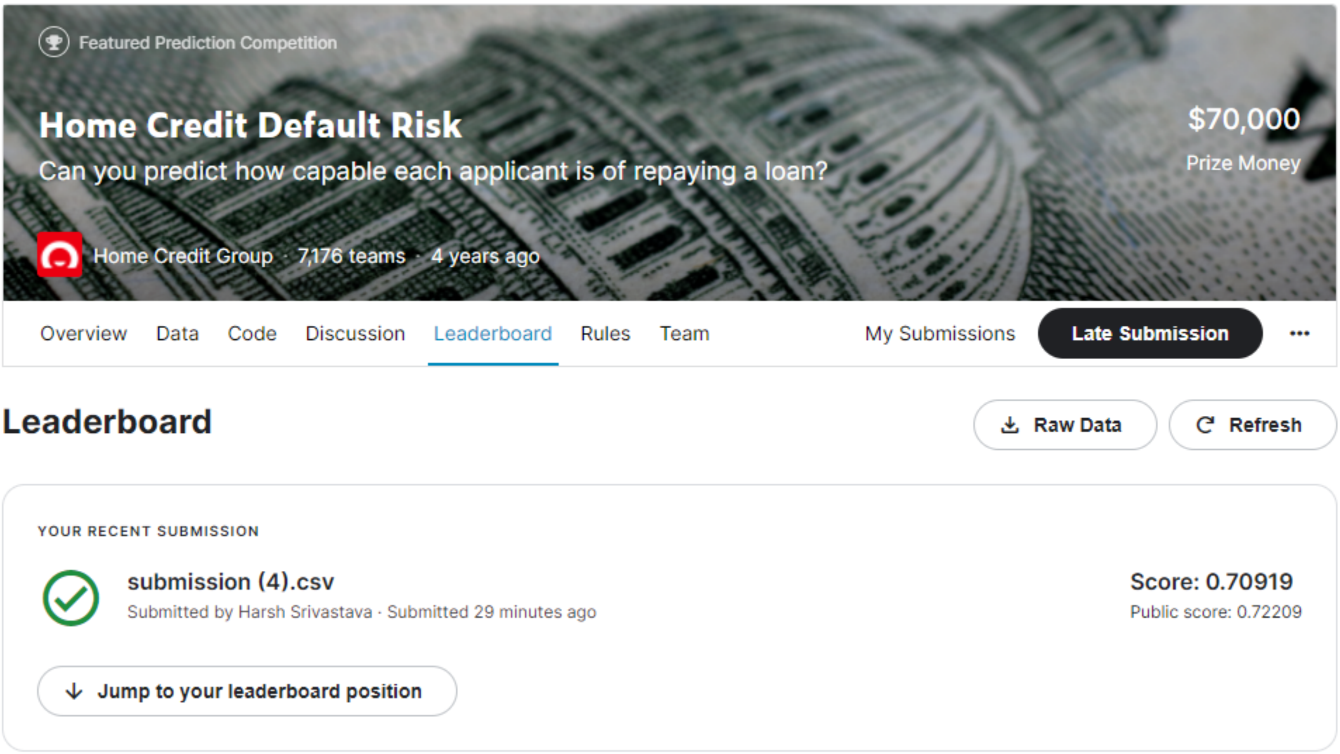
Out[ ]:

	TARGET
SK_ID_CURR	
100001	0.038358
100005	0.171255
100013	0.027056
100028	0.046264
100038	0.154696
...	...
456221	0.032200
456222	0.056499
456223	0.081611
456224	0.094527
456250	0.165215

48744 rows × 1 columns

In [ ]:

```
final_submission.to_csv("submission.csv")
```



Conclusion

In conclusion, we started experimenting with the different data columns, and selected most correlated columns to `TARGET`. After this we converted object typed / categorical types to numerical values by expanding our columns with multiple columns encoded in one-hot encoding for a certain class label corresponding to each unique value. After this we interpolated this data to fill any 'nan' values that might be remaining. It was after this stage that we were able to form any meaningful conclusions. We also made graphs/charts/plots to understand different relations in the columns that we had filtered out of the initial 122 columns. At the end we ended up with 127 unique columns (after categorical to numerical conversion). Using this encoded data we tested three models, 1. Logistic Regression, 2. Naive Bayes, 3. Random Forest models and found that both 1 and 3 gave the best accuracy. The ROC for 1 and 3 were also very similar but ROC for 2 it was quite low and so we finally rejected model 2. Moving ahead we tested the accuracy on the testing dataset with model 1 and 3. Model 3 had higher accuracy so we decided to submit this one to Kaggle. In Phase 2, we tackled the problem of high number of columns, but also introduced new columns for getting highly correlated features. Using machine learning we can predict the target values based on these newly engineered features. We ended up with a high accuracy on Random Forest classifier in the transformed-merged test data augmented with the different datasets provided. In Phase-3 we tried and tested different Neural Network models with different columns, but at the end we realised that the 18 column model was the best fit among others. This is the final model we've submitted to the final Kaggle submission and obtained a public score of 0.722. For future scope of our project, we need more visualization functions to draw out more insights and implement more bagging and boosting methods to possibly get a better result.

References:

https://www.kaggle.com/c/home-credit-default-risk/data