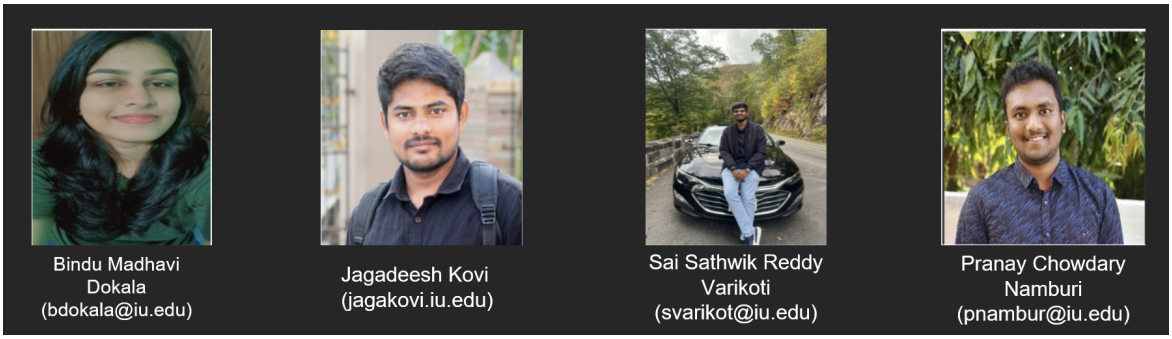
Home Credit Default Risk (HCDR)



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

Some of the challenges

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

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

Dataset and how to download

Back ground Home Credit Group

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

Home Credit Group

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

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

Background on the dataset

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

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

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

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

Data files overview

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

There are 7 different sources of data:

- **application_train/application_test (307k rows, and 48k rows):** the main training and testing data with information about each loan application at Home Credit. Every loan has its own row and is identified by the feature SK_ID_CURR. The training application data comes with the TARGET indicating **0: the loan was repaid** or **1: the loan was not repaid**. The target variable defines if the client had payment difficulties meaning he/she had late payment more than X days on at least one of the first Y installments of the loan. Such case is marked as 1 while other all other cases as 0.
- **bureau (1.7 Million rows):** data concerning client's previous credits from other financial institutions. Each previous credit has its own row in bureau, but one loan in the application data can have multiple previous credits.
- **bureau_balance (27 Million rows):** monthly data about the previous credits in bureau. Each row is one month of a previous credit, and a single previous credit can have multiple rows, one for each month of the credit length.
- **previous_application (1.6 Million rows):** previous applications for loans at Home Credit of clients who have loans in the application data. Each current loan in the application data can have multiple previous loans. Each previous application has one row and is identified by the feature SK_ID_PREV.
- **POS_CASH_BALANCE (10 Million rows):** monthly data about previous point of sale or cash loans clients have had with Home Credit. Each row is one month of a previous point of sale or cash loan, and a single previous loan can have many rows.
- credit_card_balance: monthly data about previous credit cards clients have had with Home Credit. Each row is one month of a credit card balance, and a single credit card can have many rows.

- **installments_payment (13.6 Million rows):** payment history for previous loans at Home Credit. There is one row for every made payment and one row for every missed payment.

Table sizes

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

Imports

In [1]:

```
from scipy import stats
# import latexify
import time
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import os
import zipfile
import pickle
import json
from sklearn.base import BaseEstimator, TransformerMixin
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline, FeatureUnion
from pandas.plotting import scatter_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import ShuffleSplit
from sklearn.metrics import roc_auc_score
from sklearn.metrics import accuracy_score, confusion_matrix, f1_score
from sklearn.metrics import log_loss, classification_report, roc_auc_score, make_s
from scipy import stats
from sklearn.svm import SVC
from xgboost import XGBClassifier
import warnings
warnings.filterwarnings('ignore')
```

Data files overview

Data Dictionary

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

Application train

In [2]:
```
def load_data(in_path, name):
    df = pd.read_csv(in_path)
    print(f"{name}: shape is {df.shape}")
    print(df.info())
    display(df.head(5))
    return df

datasets={}  # lets store the datasets in a dictionary so we can keep track of the
ds_name = 'application_train'
DATA_DIR=f"../Data/home-credit-default-risk/"
datasets[ds_name] = load_data(os.path.join(DATA_DIR, f'{ds_name}.csv'), ds_name)

datasets['application_train'].shape
```

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

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

5 rows × 122 columns

Out[2]:
```
(307511, 122)
```

Application test

- **application_train/application_test:** the main training and testing data with information about each loan application at Home Credit. Every loan has its own row and is identified by the feature SK_ID_CURR. The training application data comes with the TARGET indicating **0: the loan was repaid** or **1: the loan was not repaid**. The target variable defines if the client had payment difficulties meaning he/she had late payment more than X days on at least one of the first Y installments of the loan. Such case is marked as 1 while other all other cases as 0.

In [3]:
```
ds_name = 'application_test'
datasets[ds_name] = load_data(os.path.join(DATA_DIR, f'{ds_name}.csv'), ds_name)
```

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

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

5 rows × 121 columns

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

The Other datasets

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

In [4]:
```
%%time
ds_names = ("application_train", "application_test", "bureau","bureau_balance","cr
            "previous_application","POS_CASH_balance")

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

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

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

5 rows × 122 columns

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

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

5 rows × 121 columns

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

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

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

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

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

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

5 rows × 23 columns

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

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

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

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

5 rows × 37 columns

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

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

```
CPU times: total: 24.1 s
```

In [5]:

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

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

Undersampling

Undersampling is performed when the data is highly biased. In this case, we can see that the number of defaulters in the target variable is very low in comparison to the number of

persons who successfully repaid the loan. As a result, we perform undersampling, taking random entries from the good population while keeping all entries from the defaulters.

In [6]:
```
# Access the 'application_train' dataset from the 'datasets' container
application_train = datasets['application_train']

# Select the minority class instances (TARGET = 1) from the training dataset
minority_application_train = application_train[application_train['TARGET']==1]

# Append a randomly sampled subset of majority class instances (TARGET = 0) to the
undersampled_application_train = minority_application_train.append(
    application_train[application_train['TARGET']==0].reset_index(drop=True).sampl
)
```

In [7]:
```
# Assign the undersampled training dataset to a new key in the 'datasets' dictiona
datasets["undersampled_application_train"] = undersampled_application_train

# Count the number of instances in each class
class_distribution = undersampled_application_train['TARGET'].value_counts()

# Print the class distribution
print("Class distribution in the undersampled training dataset:")
print(class_distribution)
```

```
Class distribution in the undersampled training dataset:
0    75000
1    24825
Name: TARGET, dtype: int64
```

Undersampling by keeping similar ratio of non-defaulters to defaulters and also maintaining loan types of non-defaulters uniformly

In [8]:
```
  # Assuming this is a dictionary where you store your datasets

# Filtering rows with TARGET == 1 and creating a new DataFrame
datasets["undersampled_application_train_2"] = datasets["application_train"][datas
datasets["undersampled_application_train_2"]['weight'] = 1

# Undersampling Cash Loans
num_default_cashloans = len(datasets["undersampled_application_train_2"][(datasets
df_sample_cash = datasets["application_train"][(datasets["application_train"].NAME
df_sample_cash['weight'] = 1

# Undersampling Revolving Loans
num_default_revolvingloans = len(datasets["undersampled_application_train_2"][(dat
df_sample_revolving = datasets["application_train"][(datasets["application_train"]
df_sample_revolving['weight'] = 1

# Combining undersampled cash loans and revolving loans with the initial DataFrame
datasets["undersampled_application_train_2"] = pd.concat([datasets["undersampled_a

# Check the distribution of the TARGET variable
print(datasets["undersampled_application_train_2"].TARGET.value_counts())
```

```
1    24825
0    24825
Name: TARGET, dtype: int64
```

In [9]:
```
# Assuming this is a dictionary where you store your datasets
```

```
# Filtering rows with TARGET == 1 and creating a new DataFrame
undersampled_application_train_2 = datasets["application_train"][datasets["applica
undersampled_application_train_2['weight'] = 1

# Undersampling Cash Loans
num_default_cashloans = len(undersampled_application_train_2[(undersampled_applica
df_sample_cash = datasets["application_train"][(datasets["application_train"].NAME
df_sample_cash['weight'] = 1

# Undersampling Revolving Loans
num_default_revolvingloans = len(undersampled_application_train_2[(undersampled_ap
df_sample_revolving = datasets["application_train"][(datasets["application_train"]
df_sample_revolving['weight'] = 1

# Combining undersampled cash loans and revolving loans with the initial DataFrame
undersampled_application_train_2 = pd.concat([undersampled_application_train_2, df

# Check the distribution of the TARGET variable
print(undersampled_application_train_2.TARGET.value_counts())
```

```
1    24825
0    24825
Name: TARGET, dtype: int64
```

Correlation Analysis

Correlation with the target column

In [10]:

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

```
Most Positive Correlations:
 FLAG_DOCUMENT_3                0.044346
REG_CITY_NOT_LIVE_CITY          0.044395
FLAG_EMP_PHONE                  0.045982
REG_CITY_NOT_WORK_CITY          0.050994
DAYS_ID_PUBLISH                 0.051457
DAYS_LAST_PHONE_CHANGE          0.055218
REGION_RATING_CLIENT            0.058899
REGION_RATING_CLIENT_W_CITY     0.060893
DAYS_BIRTH                      0.078239
TARGET                          1.000000
Name: TARGET, dtype: float64

Most Negative Correlations:
 EXT_SOURCE_3                -0.178919
EXT_SOURCE_2                -0.160472
EXT_SOURCE_1                -0.155317
DAYS_EMPLOYED               -0.044932
FLOORSMAX_AVG               -0.044003
FLOORSMAX_MEDI              -0.043768
FLOORSMAX_MODE              -0.043226
AMT_GOODS_PRICE             -0.039645
REGION_POPULATION_RELATIVE  -0.037227
ELEVATORS_AVG               -0.034199
Name: TARGET, dtype: float64
```

In [11]:

```
corr_application_train = application_train.corr()['TARGET'].sort_values()
corr_application_train = corr_application_train.reset_index().rename(columns={'ind
corr_application_train
```

Out[11]:

	Attributes	Correlation
0	EXT_SOURCE_3	-0.178919
1	EXT_SOURCE_2	-0.160472
2	EXT_SOURCE_1	-0.155317
3	DAYS_EMPLOYED	-0.044932
4	FLOORSMAX_AVG	-0.044003
...	...	...
101	DAYS_LAST_PHONE_CHANGE	0.055218
102	REGION_RATING_CLIENT	0.058899
103	REGION_RATING_CLIENT_W_CITY	0.060893
104	DAYS_BIRTH	0.078239
105	TARGET	1.000000

106 rows × 2 columns

In [12]:

```
corr = datasets["undersampled_application_train"].corr()['TARGET']
corr=corr.sort_values(ascending=False)
print('NEGATIVE CORRELATIONS:\n', corr.tail(10))
print('\n\nPOSITIVE CORRELATIONS\n', corr.head(10))
```

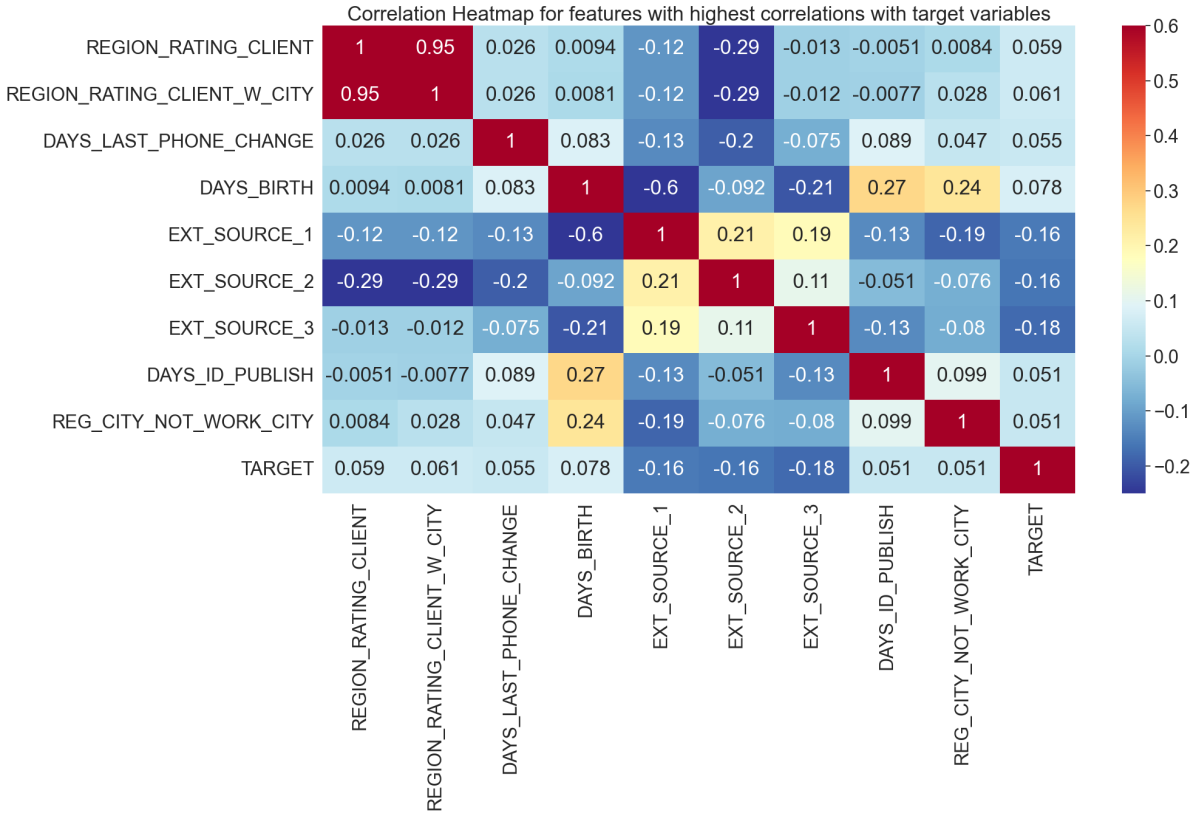
```
NEGATIVE CORRELATIONS:
 REGION_POPULATION_RELATIVE   -0.059953
AMT_GOODS_PRICE              -0.063466
FLOORSMAX_MODE               -0.069910
FLOORSMAX_MEDI               -0.071287
FLOORSMAX_AVG                -0.071689
DAYS_EMPLOYED                -0.072984
EXT_SOURCE_2                 -0.244016
EXT_SOURCE_1                 -0.246993
EXT_SOURCE_3                 -0.277081
FLAG_MOBIL                         NaN
Name: TARGET, dtype: float64


POSITIVE CORRELATIONS
 TARGET                         1.000000
DAYS_BIRTH                     0.123391
REGION_RATING_CLIENT_W_CITY    0.096229
REGION_RATING_CLIENT           0.093198
DAYS_LAST_PHONE_CHANGE         0.089358
DAYS_ID_PUBLISH                0.081127
REG_CITY_NOT_WORK_CITY         0.080181
FLAG_EMP_PHONE                 0.074709
FLAG_DOCUMENT_3                0.071705
REG_CITY_NOT_LIVE_CITY         0.068251
Name: TARGET, dtype: float64
```

In [13]:

```
most_corr=datasets["application_train"][["REGION_RATING_CLIENT","REGION_RATING_CLI
                "DAYS_BIRTH", "EXT_SOURCE_1", "EXT_SOURCE_2", "EXT_SOURCE_3", "DAYS
most_corr_corr = most_corr.corr()

sns.set_style("dark")
sns.set_context("notebook", font_scale=2.0, rc={"lines.linewidth": 1.0})
```

```
fig, axes = plt.subplots(figsize = (20,10),sharey=True)
sns.heatmap(most_corr_corr,cmap=plt.cm.RdYlBu_r,vmin=-0.25,vmax=0.6,annot=True)
plt.title('Correlation Heatmap for features with highest correlations with target
```

Out[13]:
```
Text(0.5, 1.0, 'Correlation Heatmap for features with highest correlations with ta
rget variables')
```



Feature Engineering:

In the process of feature engineering, we have utilized three tables namely, "Previous Applications", "Installment Payments", and "Credit Card Balance", from the secondary tables.

To identify the best customers of an organization, RFM features are employed. These features are based on three metrics: Recency, Frequency, and Monetary Value.

- **Recency** measures how recent the customer made a purchase
- **Frequency** determines how often they make purchases
- **Monetary Value** denotes how much money they spend on each purchase.

The frequency and monetary value metrics are indicative of the customer's engagement and their lifetime value, while recency is an indicator of retention and engagement.

Since we are analyzing the spending patterns of customers in this project, we use the RFM method to create features.

These features are generated by applying various functions such as **min, max, mean, sum, and count** to the relevant columns of the tables, thus producing new features that are significant for analysis.

Joining secondary tables with the primary table

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

Joining `previous_application` with `application_x`

We refer to the `application_train` data (and also `application_test` data also) as the **primary table** and the other files as the **secondary tables** (e.g., `previous_application` dataset). All tables can be joined using the primary key `SK_ID_PREV`.

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

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

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

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

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

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

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

Roadmap for secondary table processing

1. Transform all the secondary tables to features that can be joined into the main table the application table (labeled and unlabeled)

    - 'bureau', 'bureau_balance', 'credit_card_balance', 'installments_payments',
    - 'previous_application', 'POS_CASH_balance'

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

Feature transformer

In [14]:
```
# Create aggregate features (via pipeline)
class FeaturesAggregater(BaseEstimator, TransformerMixin):

    def __init__(self, features=None, agg_needed=["mean"]): # no *args or **kargs
        self.agg_needed = agg_needed
        self.agg_op_features = {}
        for f in features:
            self.agg_op_features[f] = self.agg_needed[:]
    def fit(self, X, y=None):
        return self
    def transform(self, X, y=None):
        result = X.groupby(["SK_ID_CURR"]).agg(self.agg_op_features)
        df_result = pd.DataFrame()
        for x1, x2 in result.columns:
            new_col = x1 + "_" + x2
            df_result[new_col] = result[x1][x2]
            df_result = df_result.reset_index(level=["SK_ID_CURR"])
            return df_result
```

Fixing Column names after Pandas agg() function to summarize grouped data

Since we have both the variable name and the operation performed in two rows in the Multi-Index dataframe, we can use that and name our new columns correctly.

For more details unstacking groupby results and examples please see here

For more details and examples please see here

Missing values in previous application table

In [15]:
```
# Access the 'previous_application' dataset from the 'datasets' container and assi
previous_application_data = datasets["previous_application"]

# Apply the 'isna()' method on the 'previous_application_data' DataFrame to detect
# and then apply the 'sum()' method to count the number of missing values in each
missing_values_count_per_column = previous_application_data.isna().sum()
missing_values_count_per_column
```

Out[15]:
```
SK_ID_PREV                      0
SK_ID_CURR                      0
NAME_CONTRACT_TYPE              0
AMT_ANNUITY                372235
AMT_APPLICATION                 0
AMT_CREDIT                      1
AMT_DOWN_PAYMENT           895844
AMT_GOODS_PRICE            385515
WEEKDAY_APPR_PROCESS_START      0
HOUR_APPR_PROCESS_START         0
FLAG_LAST_APPL_PER_CONTRACT     0
NFLAG_LAST_APPL_IN_DAY          0
RATE_DOWN_PAYMENT          895844
RATE_INTEREST_PRIMARY     1664263
RATE_INTEREST_PRIVILEGED  1664263
NAME_CASH_LOAN_PURPOSE          0
NAME_CONTRACT_STATUS            0
DAYS_DECISION                   0
NAME_PAYMENT_TYPE               0
CODE_REJECT_REASON              0
NAME_TYPE_SUITE            820405
NAME_CLIENT_TYPE                0
NAME_GOODS_CATEGORY             0
NAME_PORTFOLIO                  0
NAME_PRODUCT_TYPE               0
CHANNEL_TYPE                    0
SELLERPLACE_AREA                0
NAME_SELLER_INDUSTRY            0
CNT_PAYMENT                372230
NAME_YIELD_GROUP                0
PRODUCT_COMBINATION           346
DAYS_FIRST_DRAWING         673065
DAYS_FIRST_DUE             673065
DAYS_LAST_DUE_1ST_VERSION  673065
DAYS_LAST_DUE              673065
DAYS_TERMINATION           673065
NFLAG_INSURED_ON_APPROVAL  673065
dtype: int64
```

Feature engineering for prevApp table

The groupby output will have an index or multi-index on rows corresponding to your chosen grouping variables. To avoid setting this index, pass “as_index=False” to the groupby operation.

```
import pandas as pd
import dateutil

# Load data from csv file
data = pd.DataFrame.from_csv('phone_data.csv')
# Convert date from string to date times
data['date'] = data['date'].apply(dateutil.parser.parse, dayfirst=True)

data.groupby('month', as_index=False).agg({"duration": "sum"})
```

Pandas `reset_index()` to convert Multi-Index to Columns We can simplify the multi-index dataframe using reset_index() function in Pandas. By default, Pandas reset_index() converts the indices to columns.

Columns on which the feature engineering is performed include "AMT_APPLICATION", "AMT_CREDIT", "AMT_ANNUITY", "approved_credit_ratio", "AMT_ANNUITY_credit_ratio",

"Interest_ratio", "LTV_ratio", "SK_ID_PREV", "approved".

We have derived five new features from the previously mentioned features.

- The first feature is the **approved_credit_ratio**, which is the ratio of the credit amount the client requested on their previous application to the final credit amount approved on that application.
- The second feature is the **AMT_ANNUITY_credit_ratio**, which is the ratio of the annuity amount of the previous application to the final credit amount approved on that application.
- The third feature is the **Interest_ratio**, which is the ratio of the annuity amount of the previous application to the final credit amount approved on that application.
- The fourth feature is the **LTV_ratio**, which is the ratio of the final credit amount approved on the previous application to the goods price of the product the client applied for (if applicable) on the previous application.
- The fifth and final feature is the **approved**, which takes a value of 1 if the credit amount approved on the previous application is greater than 0, indicating that the application was approved.

In [16]:
```
previous_feature = ["AMT_APPLICATION", "AMT_CREDIT", "AMT_ANNUITY", "approved_cred
agg_needed = ["min", "max", "mean", "count", "sum"]


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

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

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

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

Missing value after the feature engineering

In [17]:
```
datasets["previous_application_agg"].isna().sum()
```

Out[17]:
```
SK_ID_CURR             0
AMT_APPLICATION_min    0
dtype: int64
```

Missing values in Installment payments

In [18]:
```
datasets["installments_payments"].isna().sum()
```

Out[18]:
```
SK_ID_PREV                   0
SK_ID_CURR                   0
NUM_INSTALMENT_VERSION       0
NUM_INSTALMENT_NUMBER        0
DAYS_INSTALMENT              0
DAYS_ENTRY_PAYMENT        2905
AMT_INSTALMENT               0
AMT_PAYMENT               2905
dtype: int64
```

Feature Engineering for Installment payments

Columns on which the feature engineering is performed include **"DAYS_INSTALMENT_DIFF", "AMT_PATMENT_PCT"**.

From the previous features, we have generated two additional features.

- The first feature is called **DAYS_INSTALMENT_DIFF**, which is the difference between the date when the installment of the previous credit was due and the actual date when it was paid.
- The second feature is the **AMT_PATMENT_PCT**, which represents the percentage of the prescribed installment amount of the previous credit that the client actually paid on a particular installment, for every entry in the dataset.

In [19]:
```
payments_features = ["DAYS_INSTALMENT_DIFF", "AMT_PATMENT_PCT"]

agg_needed = ["mean"]

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

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

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

Missing value after the feature engineering

In [20]:
```
datasets["installments_payments_agg"].isna().sum()
```

Out[20]:
```
SK_ID_CURR                   0
DAYS_INSTALMENT_DIFF_mean    9
dtype: int64
```

Missing value in Credit card balance

In [21]:
```
datasets["credit_card_balance"].isna().sum()
```

Out[21]:
```
SK_ID_PREV                      0
SK_ID_CURR                      0
MONTHS_BALANCE                  0
AMT_BALANCE                     0
AMT_CREDIT_LIMIT_ACTUAL         0
AMT_DRAWINGS_ATM_CURRENT      749816
AMT_DRAWINGS_CURRENT            0
AMT_DRAWINGS_OTHER_CURRENT    749816
AMT_DRAWINGS_POS_CURRENT      749816
AMT_INST_MIN_REGULARITY       305236
AMT_PAYMENT_CURRENT           767988
AMT_PAYMENT_TOTAL_CURRENT       0
AMT_RECEIVABLE_PRINCIPAL        0
AMT_RECIVABLE                   0
AMT_TOTAL_RECEIVABLE            0
CNT_DRAWINGS_ATM_CURRENT      749816
CNT_DRAWINGS_CURRENT            0
CNT_DRAWINGS_OTHER_CURRENT    749816
CNT_DRAWINGS_POS_CURRENT      749816
CNT_INSTALMENT_MATURE_CUM     305236
NAME_CONTRACT_STATUS            0
SK_DPD                          0
SK_DPD_DEF                      0
dtype: int64
```

Feature Engineering for Credit card balance

Columns on which the feature engineering is performed include **"AMT_BALANCE", "AMT_DRAWINGS_PCT", "AMT_DRAWINGS_ATM_PCT", "AMT_DRAWINGS_OTHER_PCT", "AMT_DRAWINGS_POS_PCT", "AMT_PRINCIPAL_RECEIVABLE_PCT", "CNT_DRAWINGS_ATM_CURRENT", "CNT_DRAWINGS_CURRENT", "CNT_DRAWINGS_OTHER_CURRENT", "CNT_DRAWINGS_POS_CURRENT", "SK_DPD", "SK_DPD_DEF"**.

We have generated five new features using the previous features mentioned.

- The first feature is called **AMT_DRAWINGS_PCT**, which represents the ratio of the amount drawn during the previous credit month to the credit card limit during that month.
- The second feature is **AMT_DRAWINGS_ATM_PCT**, which is the ratio of the amount drawn at an ATM during the previous credit month to the credit card limit during that month.
- The third feature is **AMT_DRAWINGS_OTHER_PCT**, which is the ratio of the amount drawn for other purposes during the previous credit month to the credit card limit during that month.
- The fourth feature is **AMT_DRAWINGS_POS_PCT**, which is the ratio of the amount drawn or spent on goods during the previous credit month to the credit card limit during that month.
- Finally, the fifth feature is **MT_PRINCIPAL_RECEIVABLE_PCT**, which represents the ratio of the amount receivable for principal on the previous credit to the total amount receivable on that credit.

In [22]:

```
credit_features = [
    "AMT_BALANCE",
    "AMT_DRAWINGS_PCT",
    "AMT_DRAWINGS_ATM_PCT",
    "AMT_DRAWINGS_OTHER_PCT",
    "AMT_DRAWINGS_POS_PCT",
    "AMT_PRINCIPAL_RECEIVABLE_PCT",
    "CNT_DRAWINGS_ATM_CURRENT",
    "CNT_DRAWINGS_CURRENT",
    "CNT_DRAWINGS_OTHER_CURRENT",
    "CNT_DRAWINGS_POS_CURRENT",
    "SK_DPD",
    "SK_DPD_DEF",
]

agg_needed = ["mean"]


def calculate_pct(x, y):
    return x / y if (y != 0) & pd.notnull(y) else np.nan
#def pct(x, y):
    #return x / y if (y != 0) & pd.notnull(y) else np.nan


def credit_feature_aggregation(df, feature, agg_needed):
    pct_columns = [
        ("AMT_DRAWINGS_CURRENT", "AMT_DRAWINGS_PCT"),
        ("AMT_DRAWINGS_ATM_CURRENT", "AMT_DRAWINGS_ATM_PCT"),
        ("AMT_DRAWINGS_OTHER_CURRENT", "AMT_DRAWINGS_OTHER_PCT"),
        ("AMT_DRAWINGS_POS_CURRENT", "AMT_DRAWINGS_POS_PCT"),
        ("AMT_RECEIVABLE_PRINCIPAL", "AMT_PRINCIPAL_RECEIVABLE_PCT"),
    ]

    for col_x, col_pct in pct_columns:
        df[col_pct] = [calculate_pct(x, y) for x, y in zip(df[col_x], df["AMT_CRED

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


datasets["credit_card_balance_agg"] = credit_feature_aggregation(
    datasets["credit_card_balance"], credit_features, agg_needed
)
```

Missing values after feature engineering

In [23]:

```
datasets["credit_card_balance_agg"].isna().sum()
```

Out[23]:

```
SK_ID_CURR          0
AMT_BALANCE_mean    0
dtype: int64
```

Join the feature engineered datasets with under-sampled datasets

In [24]:

```
# Load the train dataset
train_data = datasets["application_train"]

# Compute the distribution of the target variable
```

```
target_counts = train_data['TARGET'].value_counts()

# Display the target distribution
print("Target variable distribution:\n")
print(target_counts)
print("\n")

# Compute the percentage of positive and negative examples in the dataset
positive_count = target_counts[1]
negative_count = target_counts[0]
total_count = positive_count + negative_count
positive_percentage = (positive_count / total_count) * 100
negative_percentage = (negative_count / total_count) * 100

# Display the percentages of positive and negative examples
print(f"Percentage of positive examples: {positive_percentage:.2f}%")
print(f"Percentage of negative examples: {negative_percentage:.2f}%")
```

```
Target variable distribution:

0    282686
1     24825
Name: TARGET, dtype: int64


Percentage of positive examples: 8.07%
Percentage of negative examples: 91.93%
```

In [25]:
```
train_dataset= datasets["undersampled_application_train"] #primary dataset

merge_all_data = True

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

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

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

In [26]:
```
datasets["undersampled_application_train_4"] = train_dataset
```

In [27]:
```
train_dataset.shape
```

Out[27]:
```
(99825, 125)
```

In [28]:
```
train_dataset = datasets["undersampled_application_train_2"]
train_dataset = train_dataset.merge(datasets["previous_application_agg"], how='lef
train_dataset = train_dataset.merge(datasets["installments_payments_agg"], how='le
train_dataset = train_dataset.merge(datasets["credit_card_balance_agg"], how='left
train_dataset = train_dataset.drop(columns = 'weight')
datasets["undersampled_application_train_4_2"] = train_dataset
```

In [29]:
```
train_dataset.shape
```

Out[29]: (49650, 125)

In [30]:
```
train_dataset.to_csv('train_dataset.csv', index=False)
```

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

In [31]:
```
X_kaggle_test= datasets["application_test"]

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

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

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

In [32]:
```
X_kaggle_test.to_csv('X_kaggle_test.csv', index=False)
```

Loss Functions

LOG LOSS

Logarithmic Loss (logloss) is a measure used to evaluate the performance of a classification model, especially in the context of binary and multiclass classification problems. It quantifies how well the predicted probabilities of the model align with the true class labels. Logloss is a logarithmic scoring function that penalizes models more heavily for confidently incorrect predictions.

$$\text{LogLoss} = -\frac{1}{m}\sum_{i=1}^{m}(y_i \cdot \log\ (p_i)\ +\ (1-y_i)\cdot\log(1-p_i)) + \lambda\sum_{j=1}^{n}|\theta_j|$$

m : Number of samples in the dataset.
y_i : True label for the i-th sample (0 or 1).
p_i : Predicted probability that the i-th sample belongs to the positive class.
λ : Regularization parameter.
θ_j : Model parameters (coefficients) associated with the features.
n : Number of features.

Entropy

Entropy, in the context of information theory, is a measure of uncertainty or disorder in a set of probabilities associated with possible outcomes. The formula you provided is the expression for entropy in the context of a discrete probability distribution.

$$H(p) = -\sum_{i=1}^{c} p_i \log_2(p_i)$$

$H(p)$: Entropy, representing the amount of uncertainty or disorder.
c : The number of possible classes or outcomes.
p_i : The probability associated with the i-th class or outcome.

Squared Hinge Loss

The squared hinge loss is a loss function commonly used in the context of support vector machines (SVMs). It penalizes misclassifications and encourages correct classification with a margin. The formula is as follows:

$$\text{SquareHingeLoss} = \frac{1}{m} \sum_{i=1}^{m} \left[\max\left(0, 1 - y_i \left(\theta^T \cdot x_i + b\right)\right)^2 \right] + \lambda \sum_{j=1}^{n} \theta_j^2$$

m : The number of training examples.
i : The index of a training example.
x_i : The feature vector of the i-th training example.
θ : The parameter vector.
b : The bias term.
y_i : The true label of the i-th training example.
$\max(0, 1 - y_i(\theta^T \cdot x_i + b))^2$: The squared hinge loss for a single training example.
λ : Regularization parameter.
$\sum_{j=1}^{n} \theta_j^2$: L2 regularization term.

Gini Impurity

Gini Impurity is a measure of impurity or disorder used in the context of decision trees and machine learning. It represents the probability of incorrectly classifying a randomly chosen element in the dataset.

$$S = \sum_{i=1}^{c} p_i(1 - p_i)$$

S : Gini Impurity
c : Number of possible classes or outcomes
p_i : Probability associated with the i-th class or outcome

Evaluation metrics

The evaluation of submissions is conducted through the calculation of the area under the ROC curve, which measures the relationship between the predicted probability and the observed target. The SkLearn roc_auc_score function is utilized to compute the AUC or AUROC, effectively summarizing the information contained in the ROC curve into a single numerical

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

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

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

Accuracy

It refers to the proportion of accurately classified data instances in relation to the overall number of data instances.

$$\text{Accuracy} = \frac{TN + TP}{TN + FP + TP + FN}$$

$$TN : \text{True Negative (instances correctly predicted as negative)}$$
$$TP : \text{True Positive (instances correctly predicted as positive)}$$
$$FP : \text{False Positive (instances incorrectly predicted as positive)}$$
$$FN : \text{False Negative (instances incorrectly predicted as negative)}$$

Precision:

Precision refers to the ratio of true positives to the sum of true positives and false positives.

$$\text{Precision} = \frac{TP}{TP + FP}$$

$$TP : \text{True Positive (instances correctly predicted as positive)}$$
$$FP : \text{False Positive (instances incorrectly predicted as positive)}$$

F1 SCORE

It is the harmonic mean of accuracy and recall, taking into account both false positives and false negatives. It is a useful metric for evaluating models on imbalanced datasets.

$$\text{F1Score} = \frac{Precision * Recall}{Precision + Recall}$$

RECALL

It denotes the fraction of positive instances that are correctly identified as positive by the model. This metric is equivalent to the TPR (True Positive Rate).

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

TP : True Positive (instances correctly predicted as positive)
FP : False Positive (instances incorrectly predicted as positive)
FN : False Negative (instances incorrectly predicted as negative)

CONFUSION MATRIX

It is a tabular representation consisting of two axes - one representing the actual values and the other representing the predicted values. The matrix is of size 2x2 and is commonly used in classification tasks to assess the performance of a model.

Create confusion matrix for baseline model

In [33]:
```
class_labels = ["No Default","Default"]
import numpy as np
from sklearn.metrics import confusion_matrix

def confusion_matrix_normalized(model, X_train, y_train, X_test, y_test):
    y_train_pred = model.predict(X_train)
    y_test_pred = model.predict(X_test)

    cm_train = confusion_matrix(y_train, y_train_pred, normalize='true').astype(np
    cm_test = confusion_matrix(y_test, y_test_pred, normalize='true').astype(np.fl

    return cm_train, cm_test
```

Processing pipeline

In [34]:
```
# Create a class to select numerical or categorical columns
from sklearn.base import BaseEstimator, TransformerMixin

# Create a transformer to select numerical or categorical columns
class ColumnSelector(BaseEstimator, TransformerMixin):
    def __init__(self, columns):
        self.columns = columns

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

    def transform(self, X):
        return X[self.columns].values
```

In [35]:
```
def pct(x):
    return round(100*x,3)
```

Empirical Findings

Tracking results in dataframe

In [36]:
```
try:
    expLog
except NameError:
```

```
    expLog = pd.DataFrame(columns=["exp_name",
                                   "description",
                                   "Train Time (sec)",
                                   "Test Time (sec)",
                                   "Train Acc",
                                   "Valid Acc",
                                   "Test  Acc",
                                   "Train AUC",
                                   "Valid AUC",
                                   "Test  AUC",
                                   "Train F1 Score",
                                   "Valid F1 Score",
                                   "Test F1 Score"
                                  ])
```

In [37]:

```
def get_results(expLog, exp_name, description, model, train_time, test_time, X_tra
    expLog.loc[len(expLog)] = [f"{exp_name}", description] + list(np.round(
               [train_time, test_time,
                accuracy_score(y_train, model.predict(X_train)),
                accuracy_score(y_valid, model.predict(X_valid)),
                accuracy_score(y_test, model.predict(X_test)),
                roc_auc_score(y_train, model.predict_proba(X_train)[:, 1]),
                roc_auc_score(y_valid, model.predict_proba(X_valid)[:, 1]),
                roc_auc_score(y_test, model.predict_proba(X_test)[:, 1]),
                f1_score(y_train, model.predict(X_train)),
                f1_score(y_valid, model.predict(X_valid)),
                f1_score(y_test, model.predict(X_test))],
    4))
    return expLog
```

OHE when previously unseen unique values in the test/validation set

Train, validation and Test sets (and the leakage problem we have mentioned previously):

Let's look at a small usecase to tell us how to deal with this:

- The OneHotEncoder is fitted to the training set, which means that for each unique value present in the training set, for each feature, a new column is created. Let's say we have 39 columns after the encoding up from 30 (before preprocessing).
- The output is a numpy array (when the option sparse=False is used), which has the disadvantage of losing all the information about the original column names and values.
- When we try to transform the test set, after having fitted the encoder to the training set, we obtain a `ValueError`. This is because the there are new, previously unseen unique values in the test set and the encoder doesn't know how to handle these values. In order to use both the transformed training and test sets in machine learning algorithms, we need them to have the same number of columns.

This last problem can be solved by using the option handle_unknown='ignore'of the OneHotEncoder, which, as the name suggests, will ignore previously unseen values when transforming the test set.

Here is a example that in action:

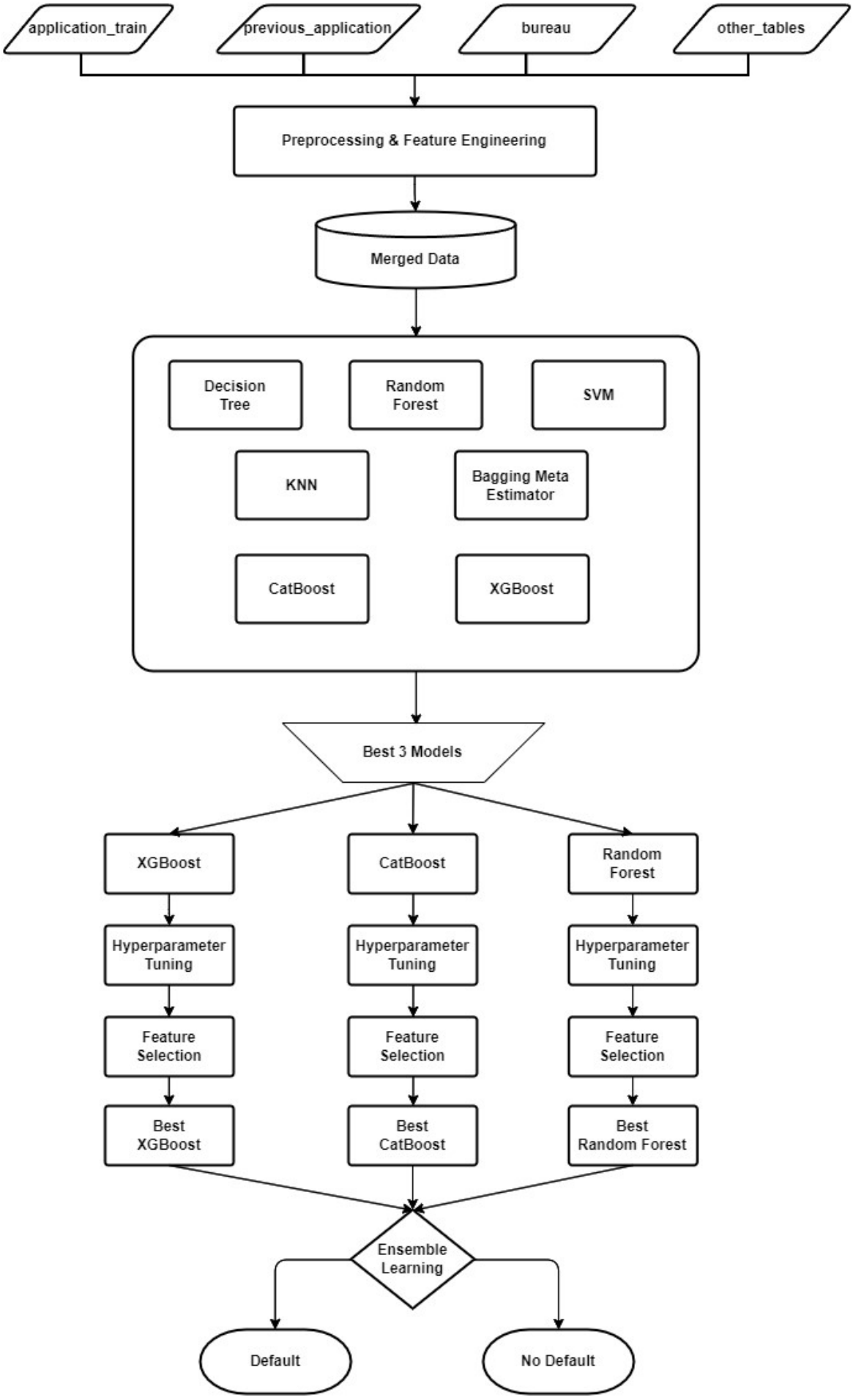
```
# Identify the categorical features we wish to consider.
cat_attribs = ['CODE_GENDER',
'FLAG_OWN_REALTY','FLAG_OWN_CAR','NAME_CONTRACT_TYPE',
               'NAME_EDUCATION_TYPE','OCCUPATION_TYPE','NAME_INCOME_TYPE']

# Notice handle_unknown="ignore" in OHE which ignore values from the
validation/test that
# do NOT occur in the training set
cat_pipeline = Pipeline([
        ('selector', DataFrameSelector(cat_attribs)),
        ('imputer', SimpleImputer(strategy='most_frequent')),
        ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
    ])
```

MACHINE LEARNING PIPELINES

HCDR preprocessing with all columns

By opting to incorporate a variety of data sources, including the Previous Application, Installment Payments, and Credit Card Applications tables, our analysis benefits from the diversification of our dataset, circumventing the limitations posed by solely utilizing the application_train data. Given the imbalanced nature of the data, which prompted the implementation of undersampling techniques to offset the surplus of non-defaulters (those classified with a target variable of 0), leveraging additional tables with relevant features is imperative for establishing a robust predictive model. Furthermore, our correlation analysis has confirmed the salience of features contained within these other tables, further substantiating our decision to utilize this multi-faceted approach.

Block Diagram



Selecting the numerical and categorical features

In [39]:

```
# Load the undersampled training dataset
train_dataset = datasets["undersampled_application_train_4"]

# Separate numerical and categorical features
numerical_features = []
categorical_features = []

for feature_name in train_dataset:
    # Check if feature is numerical or categorical
    if train_dataset[feature_name].dtype in [np.float64, np.int64]:
        numerical_features.append(feature_name)
    else:
        categorical_features.append(feature_name)

# Remove target and ID columns from numerical features
numerical_features.remove('TARGET')
numerical_features.remove('SK_ID_CURR')

# Define pipelines for categorical and numerical features
categorical_pipeline = Pipeline([
    ('selector', ColumnSelector(categorical_features)),  # Select categorical feat
    ('imputer', SimpleImputer(strategy='most_frequent')),  # Impute missing values
    ('one_hot_encoder', OneHotEncoder(sparse=False, handle_unknown="ignore"))  # O
])

numerical_pipeline = Pipeline([
    ('selector', ColumnSelector(numerical_features)),  # Select numerical features
    ('imputer', SimpleImputer(strategy='mean')),  # Impute missing values with mea
    ('standard_scaler', StandardScaler()),  # Standardize numerical features
])

# Combine pipelines for numerical and categorical features
data_prep_pipeline = FeatureUnion(transformer_list=[
    ("numerical_pipeline", numerical_pipeline),
    ("categorical_pipeline", categorical_pipeline),
])

# Compute the total number of features, as well as the number of numerical and cat
selected_features = numerical_features + categorical_features + ["SK_ID_CURR"]
total_features = f"Total Features: {len(selected_features)} - Numerical: {len(nume

print(total_features)  # Print the total number of features and their breakdown
```

```
Total Features: 124 - Numerical: 107, Categorical: 16
```

Create Train and Test Datasets

In [40]:

```
y_train = train_dataset['TARGET']
X_train = train_dataset[selected_features]
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_size=0.
X_kaggle_test= X_kaggle_test[selected_features]

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

```
X train           shape: (72123, 124)
X validation      shape: (14974, 124)
X test            shape: (12728, 124)
X X_kaggle_test   shape: (48744, 124)
```

Experimental Models

In order to establish a benchmark for comparison, we shall employ the utilization of a logistic regression model, which will make use of certain preprocessed features, processed by our established pipeline.

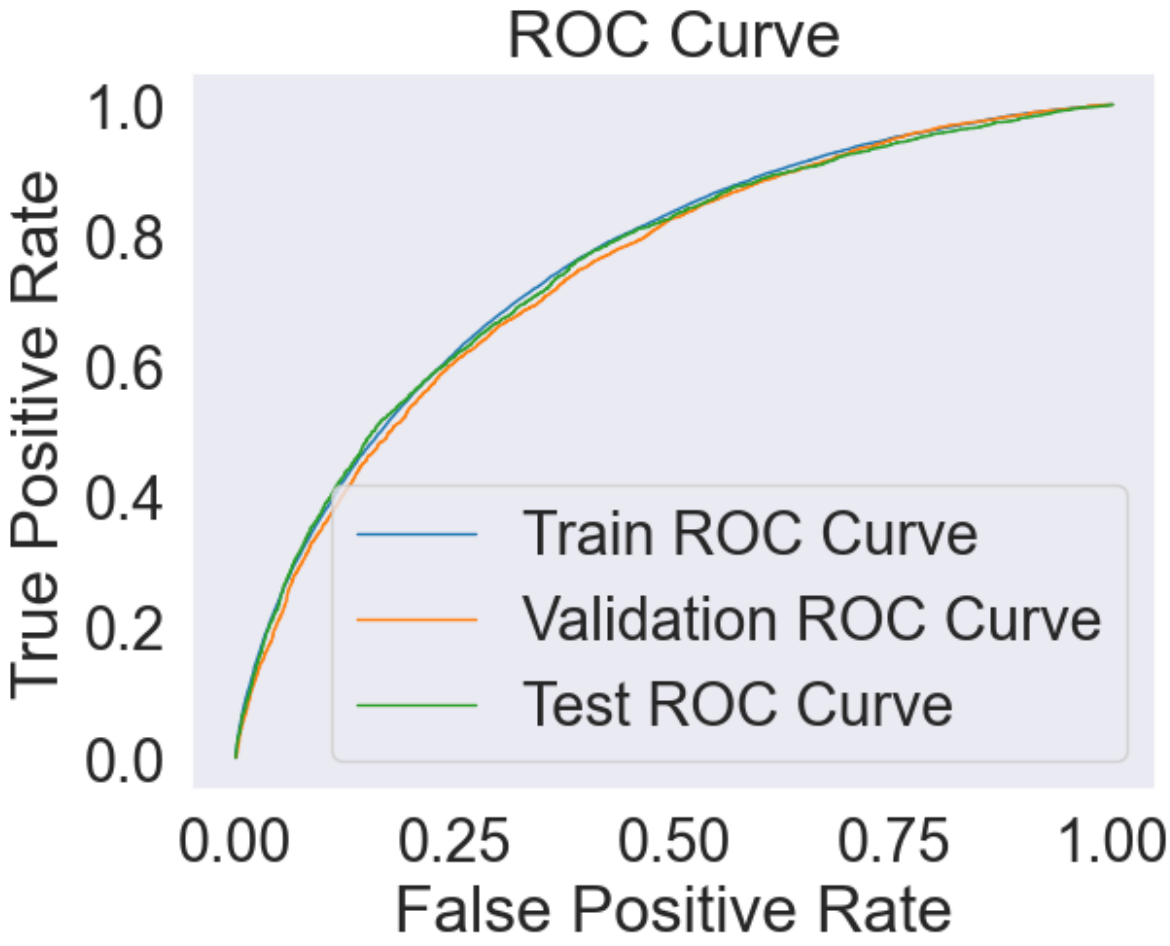
For the HDCR project, a total of 9 machine learning models were constructed to accurately classify the credit defaluters . Among these models, the three best-performing ones were selected based on their superior performance metrics. These three models were then subject to hyperparameter tuning and feature selection to optimize their performance. The hyperparameter tuning involved fine-tuning the various parameters of the models to identify the optimal set of parameters that produce the highest accuracy, precision, and recall scores. Meanwhile, feature selection aimed to identify the most relevant features that are highly correlated with the target variable to eliminate irrelevant variables that may affect the accuracy of the models.

Model-1 baseline using training columns for the application with LogisticRegression( )

In [41]:

```
from sklearn.metrics import roc_curve
# Logistic Regression model with under-sampled data
np.random.seed(42)

# Define a pipeline that includes data preparation and logistic regression
full_pipeline_with_predictor = Pipeline([
        ("preparation", data_prep_pipeline),  # Data preparation pipeline
        ("linear", LogisticRegression())  # Logistic Regression model
    ])

# Train the model and measure the training time
start_time = time.time()
model = full_pipeline_with_predictor.fit(X_train, y_train)
train_time = np.round(time.time() - start_time, 4)

# Evaluate the model on the test set and measure the test time
start_time = time.time()
test_score = full_pipeline_with_predictor.score(X_test, y_test)
test_time = np.round(time.time() - start_time, 4)

# Define an experiment name based on the number of selected features
experiment_name = f"Model-1 Baseline LR"
experiment_description =f"Logistic regression with undersampled data {len(selected

# Log the results of the experiment
expLog = get_results(expLog, experiment_name,experiment_description, model, train_
expLog


# Model Training and Validation
model.fit(X_train, y_train)
```

```
# Model Predictions
train_preds = model.predict_proba(X_train)[:, 1]
valid_preds = model.predict_proba(X_valid)[:, 1]
test_preds = model.predict_proba(X_test)[:, 1]

# Compute Metrics
train_fpr, train_tpr, _ = roc_curve(y_train, train_preds)
valid_fpr, valid_tpr, _ = roc_curve(y_valid, valid_preds)
test_fpr, test_tpr, _ = roc_curve(y_test, test_preds)
# Plot ROC Curve
plt.plot(train_fpr, train_tpr, label="Train ROC Curve")
plt.plot(valid_fpr, valid_tpr, label="Validation ROC Curve")
plt.plot(test_fpr, test_tpr, label="Test ROC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()
expLog
```



Out[41]:

	exp_name	description	Train Time (sec)	Test Time (sec)	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC	Train F1 Score
0	Model-1 Baseline LR	Logistic regression with undersampled data 124...	2.7516	0.0771	0.7743	0.7659	0.7785	0.7546	0.743	0.7503	0.3645

Creating data preparation pipeline from given columns

In [42]:

```
def get_pipeline(num_cols = None):
    # Load the undersampled training dataset and join with additional feature data
    train_dataset = datasets["undersampled_application_train_2"]
    train_dataset = train_dataset.merge(datasets["previous_application_agg"], how=
    train_dataset = train_dataset.merge(datasets["installments_payments_agg"], how
    train_dataset = train_dataset.merge(datasets["credit_card_balance_agg"], how='

    # Separate numerical and categorical features
    numerical_features = []
    categorical_features = []
    for feature_name in train_dataset:
        if train_dataset[feature_name].dtype in [np.float64, np.int64]:
            numerical_features.append(feature_name)
        else:
            categorical_features.append(feature_name)

    # Remove unnecessary features
    numerical_features.remove('TARGET')
    numerical_features.remove('weight')
    numerical_features.remove('SK_ID_CURR')

    # Define pipelines for categorical and numerical features
    categorical_pipeline = Pipeline([
        ('selector', ColumnSelector(categorical_features)),  # Select categorical
        ('imputer', SimpleImputer(strategy='most_frequent')),  # Impute missing va
        ('one_hot_encoder', OneHotEncoder(sparse=False, handle_unknown="ignore"))
    ])

    # If columns are provided, use only those columns for numerical pipeline
    if num_cols == None:
        final_numerical_features = numerical_features
    else:
        final_numerical_features = num_cols

    numerical_pipeline = Pipeline([
        ('selector', ColumnSelector(final_numerical_features)),  # Select numerica
        ('imputer', SimpleImputer(strategy='mean')),  # Impute missing values with
        ('standard_scaler', StandardScaler()),  # Standardize numerical features
    ])

    # Combine pipelines for numerical and categorical features
    data_prep_pipeline = FeatureUnion(transformer_list=[
        ("numerical_pipeline", numerical_pipeline),
        ("categorical_pipeline", categorical_pipeline),
    ])

    # Compute the total number of features, as well as the number of numerical and
    selected_features = final_numerical_features + categorical_features + ["SK_ID_
    total_features = f"Total Features: {len(selected_features)} - Numerical: {len(

    # Print the total number of features and their breakdown
    print(total_features)

    return data_prep_pipeline, selected_features
```

Creating Train, Test, and Validation datasets

In [43]:

```
# Load the undersampled training dataset and join with additional feature datasets
train_dataset = datasets["undersampled_application_train_2"]
train_dataset = train_dataset.merge(datasets["previous_application_agg"], how='lef
train_dataset = train_dataset.merge(datasets["installments_payments_agg"], how='le
```

```
train_dataset = train_dataset.merge(datasets["credit_card_balance_agg"], how='left

# Select the target variable
y_train = train_dataset['TARGET']

# Select the features for the training set
X_train = train_dataset[selected_features]

# Split the data into training and validation sets
# The training set will be used to train the model, and the validation set will be
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_size=

# Split the training set into training and test sets
# The training set will be used to train the model, and the test set will be used
X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_size=0.

# Print the shapes of the training, validation, and test sets
print(f"Training set shape: {X_train.shape}")
print(f"Validation set shape: {X_valid.shape}")
print(f"Test set shape: {X_test.shape}")
```

```
Training set shape: (35871, 124)
Validation set shape: (7448, 124)
Test set shape: (6331, 124)
```

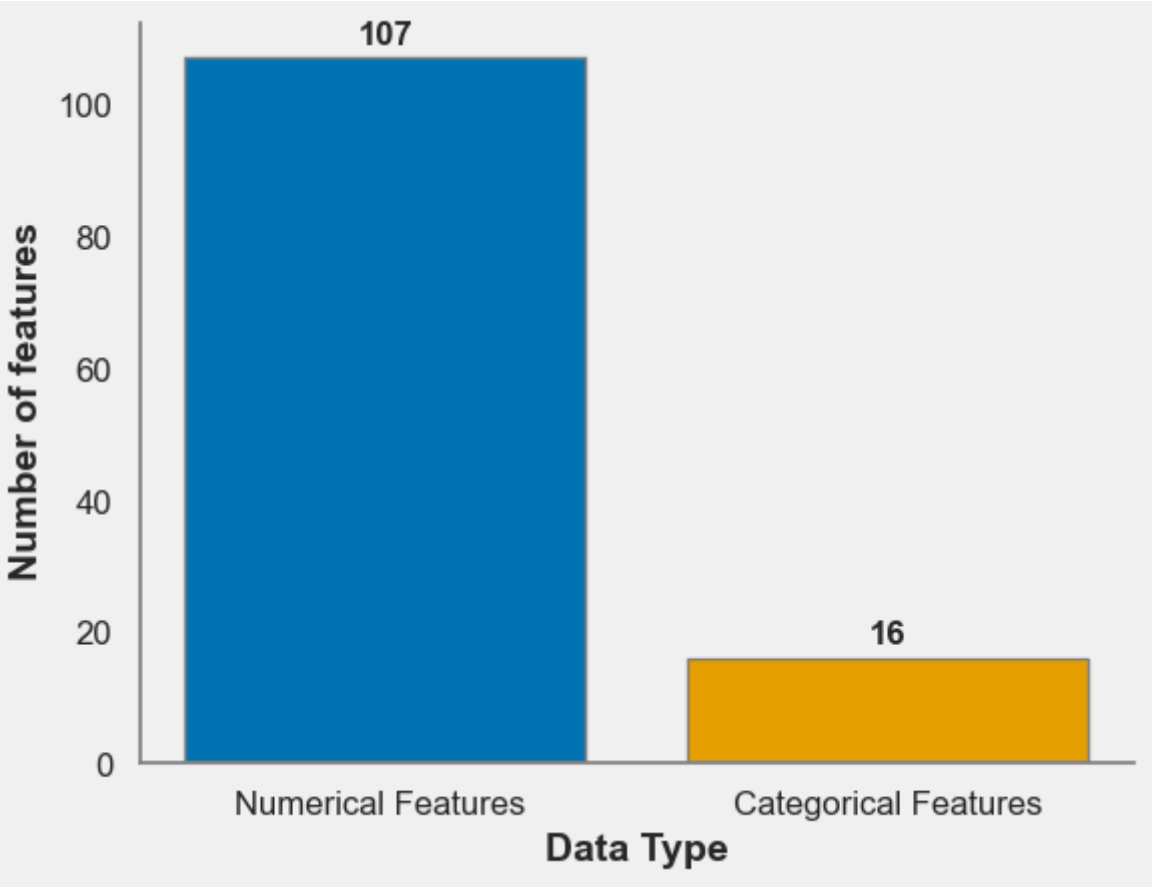
Model-2 baseline using LogisticRegression( )

For establishing a baseline, we shall employ certain processed features stemming from the pipeline. The logistic regression model shall serve as the rudimentary benchmark model 2 . We will use two undersampled data here .

In [44]:

```
import matplotlib.pyplot as plt
import matplotlib.patheffects as path_effects

data = [len(numerical_features),len(categorical_features)]
labels = ['Numerical Features ', 'Categorical Features']

fig, ax = plt.subplots()
bars = ax.bar(labels, data, color=['#0072B2', '#E69F00'], edgecolor='black')

# Add shadows to the bars
for bar in bars:
    bar.set_edgecolor('gray')
    bar.set_linewidth(1)
    bar.set_zorder(0)

    # Add labels to the bars
    height = bar.get_height()
    ax.annotate(f'{height:.0f}', xy=(bar.get_x() + bar.get_width() / 2, height),
                xytext=(0, 3), textcoords='offset points', ha='center', va='bottom
                fontsize=12, fontweight='bold')

# Customize the axis labels and ticks
ax.set_xlabel('Data Type', fontsize=14, fontweight='bold')
ax.set_ylabel('Number of features ', fontsize=14, fontweight='bold')
ax.tick_params(axis='both', labelsize=12)

# Customize the plot background
ax.set_facecolor('#F0F0F0')
```

```
fig.set_facecolor('#F0F0F0')
ax.spines['bottom'].set_color('gray')
ax.spines['left'].set_color('gray')
ax.spines['top'].set_visible(False)
ax.spines['right'].set_visible(False)

plt.show()
```

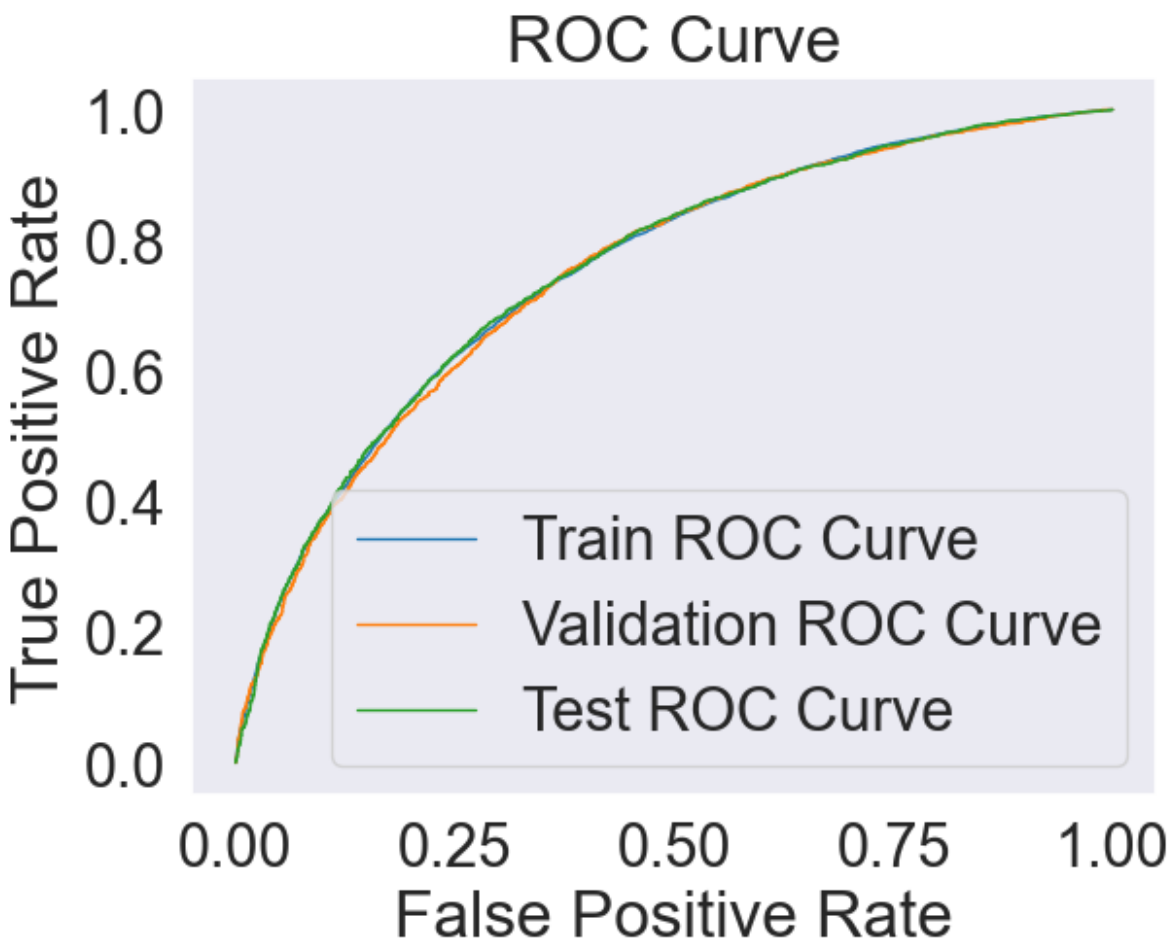


In [45]:

```
# Set the random seed for reproducibility
np.random.seed(42)

# Create pipeline for preparing the data and select features
data_prep_pipeline, selected_features = get_pipeline()

# Join the preparation pipeline with logistic regression model
full_pipeline_with_predictor = Pipeline([
        ("preparation", data_prep_pipeline),
        ("logistic_regression", LogisticRegression())
    ])

# Train the model on the training set
start = time.time()
model = full_pipeline_with_predictor.fit(X_train, y_train)
train_time = np.round(time.time() - start, 4)

# Compute cross-validation scores
cv_splits = ShuffleSplit(n_splits=3, test_size=0.7, random_state=42)
logit_scores = cross_val_score(full_pipeline_with_predictor,X_train, y_train, cv=c

print("Cross-validation scores:", logit_scores)

# Compute the test score
start = time.time()
test_score = full_pipeline_with_predictor.score(X_test, y_test)
test_time = np.round(time.time() - start, 4)
```

```
print("Test score:", test_score)

# Save the experiment results
exp_name = f"Model-2 Baseline LR"
experiment_description =f"Logistic regression with undersampled data-2 {len(select
expLog = get_results(expLog, exp_name,experiment_description,model, train_time, te
expLog



# Model Training and Validation
model.fit(X_train, y_train)

# Model Predictions
train_preds = model.predict_proba(X_train)[:, 1]
valid_preds = model.predict_proba(X_valid)[:, 1]
test_preds = model.predict_proba(X_test)[:, 1]

# Compute Metrics
train_fpr, train_tpr, _ = roc_curve(y_train, train_preds)
valid_fpr, valid_tpr, _ = roc_curve(y_valid, valid_preds)
test_fpr, test_tpr, _ = roc_curve(y_test, test_preds)
# Plot ROC Curve
plt.plot(train_fpr, train_tpr, label="Train ROC Curve")
plt.plot(valid_fpr, valid_tpr, label="Validation ROC Curve")
plt.plot(test_fpr, test_tpr, label="Test ROC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()
expLog
```

```
Total Features: 124 - Numerical: 107, Categorical: 16
Cross-validation scores: [0.67953007 0.67992832 0.67905217]
Test score: 0.6904122571473701
```



Out[45]:

	exp_name	description	Train Time (sec)	Test Time (sec)	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC	Train F1 Score
0	Model-1 Baseline LR	Logistic regression with undersampled data 124...	2.5144	0.0720	0.7727	0.7647	0.7802	0.7545	0.7421	0.7535	0.3595
1	Model-2 Baseline LR	Logistic regression with undersampled data-2 1...	1.1938	0.0496	0.6876	0.6843	0.6904	0.7525	0.7489	0.7535	0.6865

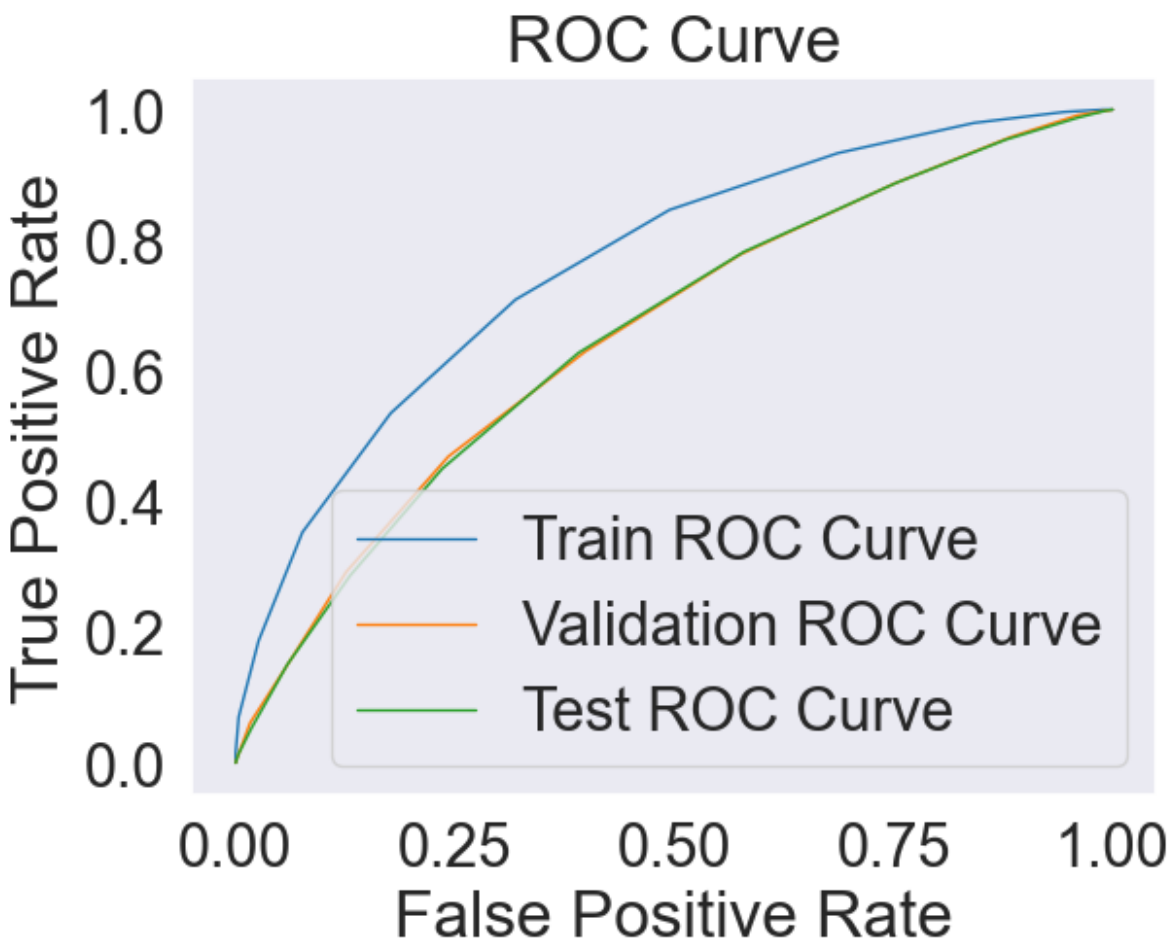
Model 3 - KNN model

In [46]:

```
from sklearn.neighbors import KNeighborsClassifier

# Set the random seed for reproducibility
np.random.seed(42)

# Create pipeline for preparing the data and select features
data_prep_pipeline, selected_features = get_pipeline()

# Join the preparation pipeline with KNN model
knn_full_pipeline_with_predictor = Pipeline([
    ("preparation", data_prep_pipeline),
    ("knn", KNeighborsClassifier(n_neighbors=11, p=2))
])
```

```python
# Train the model on the training set
start = time.time()
model = knn_full_pipeline_with_predictor.fit(X_train, y_train)
train_time = np.round(time.time() - start, 4)

# Compute the test score
start = time.time()
test_score = knn_full_pipeline_with_predictor.score(X_test, y_test)
test_time = np.round(time.time() - start, 4)

print("Test score:", test_score)

# Results
# Save the experiment results
exp_name = f"Model-3 KNN"
experiment_description =f"KNN with undersampled data-2 {len(selected_features)} fe
expLog = get_results(expLog, exp_name,experiment_description,model, train_time, te
expLog

# Model Training and Validation
model.fit(X_train, y_train)

# Model Predictions
train_preds = model.predict_proba(X_train)[:, 1]
valid_preds = model.predict_proba(X_valid)[:, 1]
test_preds = model.predict_proba(X_test)[:, 1]

# Compute Metrics
train_fpr, train_tpr, _ = roc_curve(y_train, train_preds)
valid_fpr, valid_tpr, _ = roc_curve(y_valid, valid_preds)
test_fpr, test_tpr, _ = roc_curve(y_test, test_preds)
# Plot ROC Curve
plt.plot(train_fpr, train_tpr, label="Train ROC Curve")
plt.plot(valid_fpr, valid_tpr, label="Validation ROC Curve")
plt.plot(test_fpr, test_tpr, label="Test ROC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()
expLog
```

```
Total Features: 124 - Numerical: 107, Categorical: 16
Test score: 0.6183857210551256
```



Out[46]:

	exp_name	description	Train Time (sec)	Test Time (sec)	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC	Train F1 Score
0	Model-1 Baseline LR	Logistic regression with undersampled data 124...	2.5144	0.0720	0.7727	0.7647	0.7802	0.7545	0.7421	0.7535	0.3595
1	Model-2 Baseline LR	Logistic regression with undersampled data-2 1...	1.1938	0.0496	0.6876	0.6843	0.6904	0.7525	0.7489	0.7535	0.6865
2	Model-3 KNN	KNN with undersampled data-2 124 features	0.3262	1.0494	0.6950	0.6155	0.6184	0.7625	0.6571	0.6550	0.6992

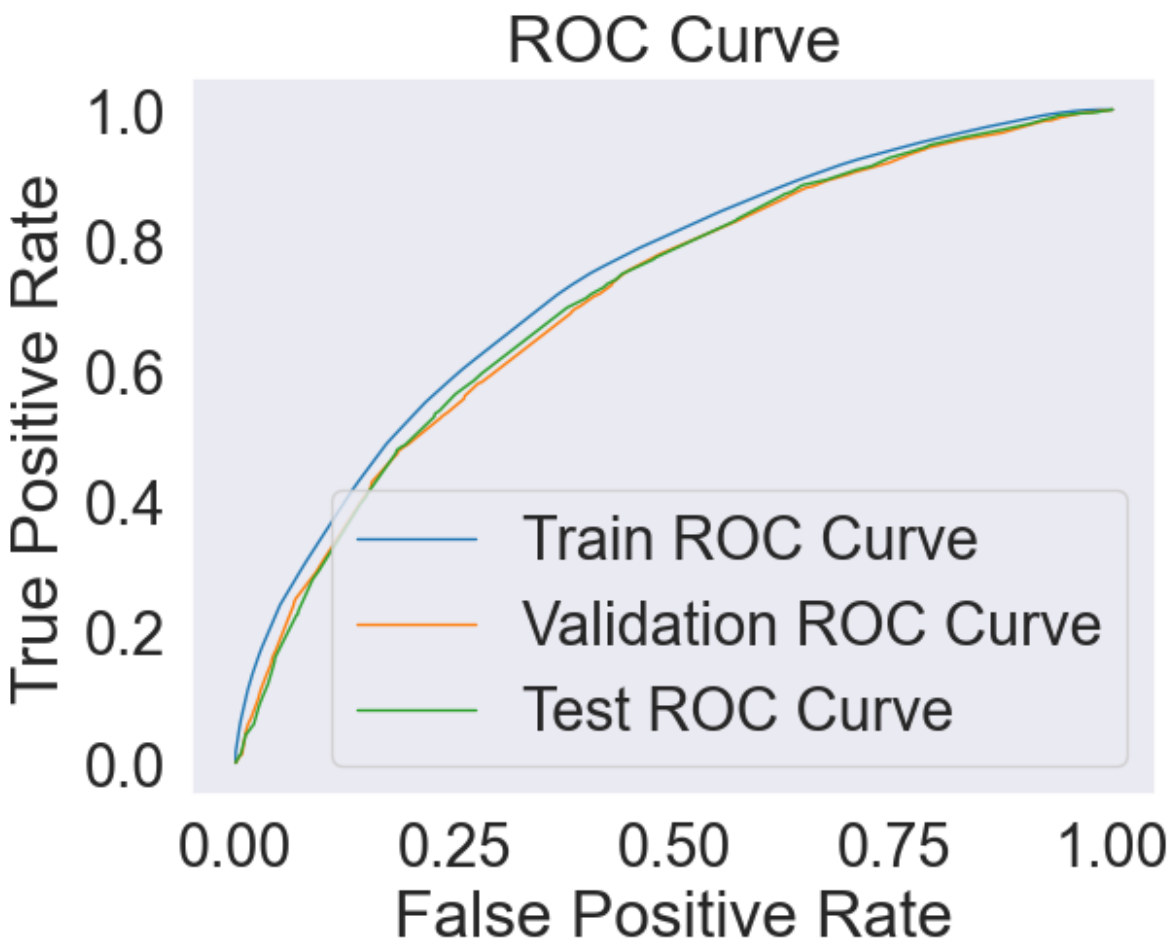
Model 4-Decision Tree

In [47]:

```
from sklearn.tree import DecisionTreeClassifier

np.random.seed(42)

# Create pipeline for preparing the data and select features
data_prep_pipeline, selected_features = get_pipeline()

# Join the preparation pipeline with Decision Tree model
decision_tree_full_pipeline_with_predictor = Pipeline([
```

```
    ("preparation", data_prep_pipeline),
    ("decision tree", DecisionTreeClassifier(random_state=42, criterion='entropy',
])

# Train the model on the training set
start = time.time()
model = decision_tree_full_pipeline_with_predictor.fit(X_train, y_train)
train_time = np.round(time.time() - start, 4)

# Compute the test score
start = time.time()
test_score = decision_tree_full_pipeline_with_predictor.score(X_test, y_test)
test_time = np.round(time.time() - start, 4)

# Print the test score and other details
print(f"Test score: {test_score:.4f}")
print(f"Training time: {train_time} sec")
print(f"Test time: {test_time} sec")
#print(f"Selected features: {selected_features}")
print(f"Number of features: {len(selected_features)}")

# Results
# Save the experiment results
exp_name = f"Model-4 Decision Tree"
experiment_description =f"Decision tree with undersampled data-2 {len(selected_fea
expLog = get_results(expLog, exp_name,experiment_description,model, train_time, te
expLog
# Model Training and Validation
model.fit(X_train, y_train)

# Model Predictions
train_preds = model.predict_proba(X_train)[:, 1]
valid_preds = model.predict_proba(X_valid)[:, 1]
test_preds = model.predict_proba(X_test)[:, 1]

# Compute Metrics
train_fpr, train_tpr, _ = roc_curve(y_train, train_preds)
valid_fpr, valid_tpr, _ = roc_curve(y_valid, valid_preds)
test_fpr, test_tpr, _ = roc_curve(y_test, test_preds)
# Plot ROC Curve
plt.plot(train_fpr, train_tpr, label="Train ROC Curve")
plt.plot(valid_fpr, valid_tpr, label="Validation ROC Curve")
plt.plot(test_fpr, test_tpr, label="Test ROC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()
expLog
```

```
Total Features: 124 - Numerical: 107, Categorical: 16
Test score: 0.6591
Training time: 1.4838 sec
Test time: 0.0504 sec
Number of features: 124
```



Out[47]:

	exp_name	description	Train Time (sec)	Test Time (sec)	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC	Train F1 Score
0	Model-1 Baseline LR	Logistic regression with undersampled data 124...	2.5144	0.0720	0.7727	0.7647	0.7802	0.7545	0.7421	0.7535	0.3595
1	Model-2 Baseline LR	Logistic regression with undersampled data-2 1...	1.1938	0.0496	0.6876	0.6843	0.6904	0.7525	0.7489	0.7535	0.6865
2	Model-3 KNN	KNN with undersampled data-2 124 features	0.3262	1.0494	0.6950	0.6155	0.6184	0.7625	0.6571	0.6550	0.6992
3	Model-4 Decision Tree	Decision tree with undersampled data-2 124 fea...	1.4838	0.0504	0.6749	0.6535	0.6591	0.7380	0.7105	0.7129	0.6881

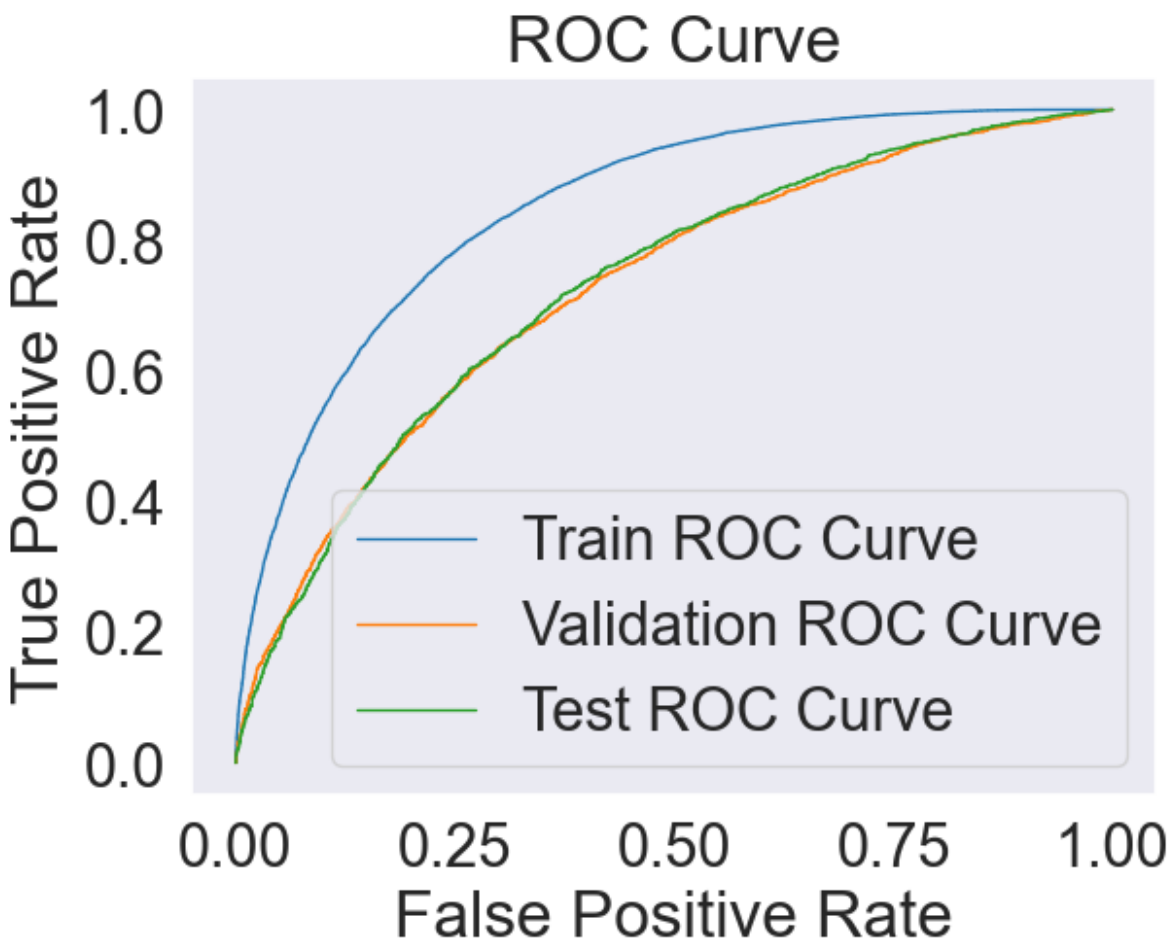
Model 5- Random Forest

In [48]:

```
from sklearn.ensemble import RandomForestClassifier

np.random.seed(42)
```

```
# Creating pipeline by joining numerical and categorical pipelines
data_prep_pipeline, selected_features = get_pipeline()

# Attaching random forest model to the above pipeline
random_forest_full_pipeline_with_predictor = Pipeline([
    ("preparation", data_prep_pipeline),
    ("random forest", RandomForestClassifier(random_state=42, bootstrap=True, max_
                      max_features=5, min_samples_leaf=10, min_samples_split=15, n
])

# Training the model
print("Training the model...")
start_time = time.time()
model = random_forest_full_pipeline_with_predictor.fit(X_train, y_train)
train_time = np.round(time.time() - start_time, 4)
print("Training time: ", train_time)

# Evaluate the model on the test set and measure the test time
print("Evaluating the model on the test set...")
start_time = time.time()
score_test  = random_forest_full_pipeline_with_predictor.score(X_test, y_test)
test_time = np.round(time.time() - start_time, 4)
print("Test score: ", score_test)
print("Test time: ", test_time)


# Results
exp_name = f"Model-5 Random Forest "
experiment_description =f"Random Forest with undersampled data-2 {len(selected_fea
expLog = get_results(expLog, exp_name,experiment_description,model, train_time, te
expLog
# Model Training and Validation
model.fit(X_train, y_train)

# Model Predictions
train_preds = model.predict_proba(X_train)[:, 1]
valid_preds = model.predict_proba(X_valid)[:, 1]
test_preds = model.predict_proba(X_test)[:, 1]

# Compute Metrics
train_fpr, train_tpr, _ = roc_curve(y_train, train_preds)
valid_fpr, valid_tpr, _ = roc_curve(y_valid, valid_preds)
test_fpr, test_tpr, _ = roc_curve(y_test, test_preds)
# Plot ROC Curve
plt.plot(train_fpr, train_tpr, label="Train ROC Curve")
plt.plot(valid_fpr, valid_tpr, label="Validation ROC Curve")
plt.plot(test_fpr, test_tpr, label="Test ROC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()
expLog
```

```
Total Features: 124 - Numerical: 107, Categorical: 16
Training the model...
Training time:  20.5597
Evaluating the model on the test set...
Test score:  0.6665613647133154
Test time:  0.4587
```



Out[48]:

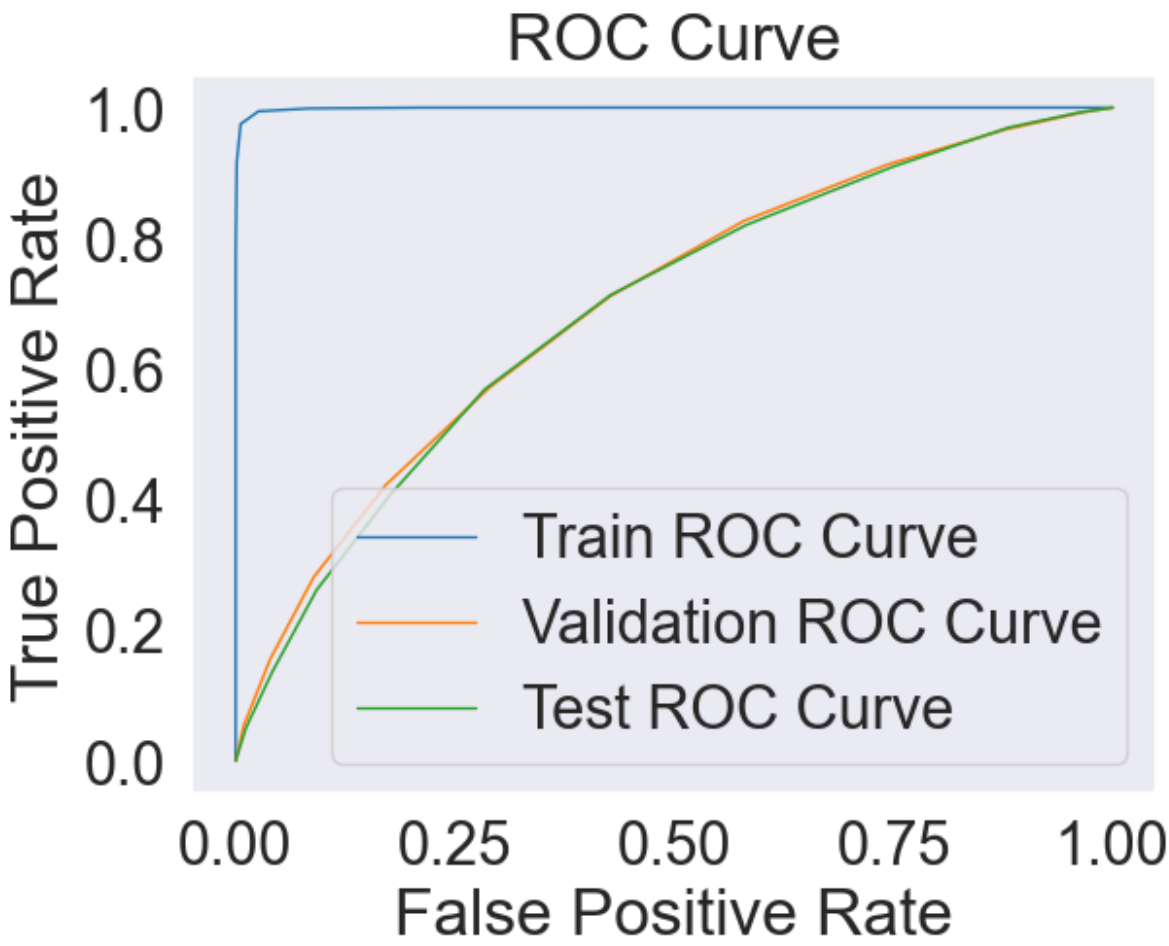
	exp_name	description	Train Time (sec)	Test Time (sec)	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC	Train F1 Score
0	Model-1 Baseline LR	Logistic regression with undersampled data 124...	2.5144	0.0720	0.7727	0.7647	0.7802	0.7545	0.7421	0.7535	0.3595
1	Model-2 Baseline LR	Logistic regression with undersampled data-2 1...	1.1938	0.0496	0.6876	0.6843	0.6904	0.7525	0.7489	0.7535	0.6865
2	Model-3 KNN	KNN with undersampled data-2 124 features	0.3262	1.0494	0.6950	0.6155	0.6184	0.7625	0.6571	0.6550	0.6992
3	Model-4 Decision Tree	Decision tree with undersampled data-2 124 fea...	1.4838	0.0504	0.6749	0.6535	0.6591	0.7380	0.7105	0.7129	0.6881
4	Model-5 Random Forest	Random Forest with undersampled data-2 124 fea...	20.5597	0.4587	0.7665	0.6657	0.6666	0.8504	0.7245	0.7275	0.7676

Model 6-Bagging Meta Estimator

In [49]:

```
from sklearn.ensemble import BaggingClassifier

np.random.seed(42)

# creating pipeline by joining numerical and categorical pipelines
data_prep_pipeline, selected_features = get_pipeline()

# Attaching bagging meta-estimator to the above pipeline
bagging_full_pipeline_with_predictor = Pipeline([
    ("preparation", data_prep_pipeline),
    ("bagging", BaggingClassifier(base_estimator=None, n_estimators=10, max_sample
                                  bootstrap=True, bootstrap_features=False, n_job
                                  verbose=0, warm_start=False))
])

# Training the model
start = time.time()
model = bagging_full_pipeline_with_predictor.fit(X_train, y_train)
train_time = np.round(time.time() - start, 4)

start = time.time()
score_test  = bagging_full_pipeline_with_predictor.score(X_test, y_test)
test_time = np.round(time.time() - start, 4)

# Results
exp_name = f"Model-6 Bagging Meta Estimator "
experiment_description =f"Bagging Meta Estimator with undersampled data-2 {len(sel
expLog = get_results(expLog, exp_name,experiment_description,model, train_time, te
expLog




# Model Training and Validation
model.fit(X_train, y_train)

# Model Predictions
train_preds = model.predict_proba(X_train)[:, 1]
valid_preds = model.predict_proba(X_valid)[:, 1]
test_preds = model.predict_proba(X_test)[:, 1]

# Compute Metrics
train_fpr, train_tpr, _ = roc_curve(y_train, train_preds)
valid_fpr, valid_tpr, _ = roc_curve(y_valid, valid_preds)
test_fpr, test_tpr, _ = roc_curve(y_test, test_preds)
# Plot ROC Curve
plt.plot(train_fpr, train_tpr, label="Train ROC Curve")
plt.plot(valid_fpr, valid_tpr, label="Validation ROC Curve")
plt.plot(test_fpr, test_tpr, label="Test ROC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()
expLog
```

```
Total Features: 124 - Numerical: 107, Categorical: 16
```


ROC Curve
True Positive Rate
1.0
0.8
0.6
0.4
0.2
0.0
0.00
0.25
0.50
0.75
1.00
False Positive Rate
Train ROC Curve
Validation ROC Curve
Test ROC Curve

Out[49]:

	exp_name	description	Train Time (sec)	Test Time (sec)	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC	Train F1 Score
0	Model-1 Baseline LR	Logistic regression with undersampled data 124...	2.5144	0.0720	0.7727	0.7647	0.7802	0.7545	0.7421	0.7535	0.3595
1	Model-2 Baseline LR	Logistic regression with undersampled data-2 1...	1.1938	0.0496	0.6876	0.6843	0.6904	0.7525	0.7489	0.7535	0.6865
2	Model-3 KNN	KNN with undersampled data-2 124 features	0.3262	1.0494	0.6950	0.6155	0.6184	0.7625	0.6571	0.6550	0.6992
3	Model-4 Decision Tree	Decision tree with undersampled data-2 124 fea...	1.4838	0.0504	0.6749	0.6535	0.6591	0.7380	0.7105	0.7129	0.6881
4	Model-5 Random Forest	Random Forest with undersampled data-2 124 fea...	20.5597	0.4587	0.7665	0.6657	0.6666	0.8504	0.7245	0.7275	0.7676
5	Model-6 Bagging Meta Estimator	Bagging Meta Estimator with undersampled data-...	5.4167	0.2396	0.9844	0.6445	0.6430	0.9990	0.6973	0.6924	0.9843

Model 7 - SVM model

In [50]:

```
from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
import time
import numpy as np

from sklearn.svm import SVC
from sklearn.pipeline import Pipeline
from sklearn.model_selection import GridSearchCV
import time
import numpy as np

# Create pipeline for preparing the data and select features
data_prep_pipeline, selected_features = get_pipeline()

# Join the preparation pipeline with SVM model
svm_full_pipeline_with_predictor = Pipeline([
    ("preparation", data_prep_pipeline),
    ("svm", SVC(random_state=42, C=0.1, degree=1, kernel="poly", probability=True)
])

# Define a simple parameter grid
param_grid = {
    'svm__C': [0.1],
```

```
    'svm__degree': [1],
    'svm__kernel': ['poly'],
}

# Create a GridSearchCV object with n_jobs=-1 to use all available CPU cores
grid_search = GridSearchCV(svm_full_pipeline_with_predictor, param_grid, n_jobs=-1

# Train the model on the training set
start = time.time()
grid_search.fit(X_train, y_train)
train_time = np.round(time.time() - start, 4)

# Compute the test score
start = time.time()
test_score = grid_search.score(X_test, y_test)
test_time = np.round(time.time() - start, 4)

print("Test score:", test_score)


# Results
exp_name = f"Model-7 SVM"
experiment_description =f"SVM with undersampled data-2 {len(selected_features)} fe
expLog = get_results(expLog, exp_name,experiment_description,model, train_time, te
expLog
```

```
Total Features: 124 - Numerical: 107, Categorical: 16
Test score: 0.6834623282261886
```

Out[50]:

	exp_name	description	Train Time (sec)	Test Time (sec)	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC	T Sc
0	Model-1 Baseline LR	Logistic regression with undersampled data 124...	2.5144	0.0720	0.7727	0.7647	0.7802	0.7545	0.7421	0.7535	0.3
1	Model-2 Baseline LR	Logistic regression with undersampled data-2 1...	1.1938	0.0496	0.6876	0.6843	0.6904	0.7525	0.7489	0.7535	0.6
2	Model-3 KNN	KNN with undersampled data-2 124 features	0.3262	1.0494	0.6950	0.6155	0.6184	0.7625	0.6571	0.6550	0.6
3	Model-4 Decision Tree	Decision tree with undersampled data-2 124 fea...	1.4838	0.0504	0.6749	0.6535	0.6591	0.7380	0.7105	0.7129	0.6
4	Model-5 Random Forest	Random Forest with undersampled data-2 124 fea...	20.5597	0.4587	0.7665	0.6657	0.6666	0.8504	0.7245	0.7275	0.7
5	Model-6 Bagging Meta Estimator	Bagging Meta Estimator with undersampled data-...	5.4167	0.2396	0.9844	0.6445	0.6430	0.9990	0.6973	0.6924	0.9
6	Model-7 SVM	SVM with undersampled data-2 124 features	2681.6150	15.7188	0.9846	0.6411	0.6421	0.9990	0.6965	0.6899	0.9

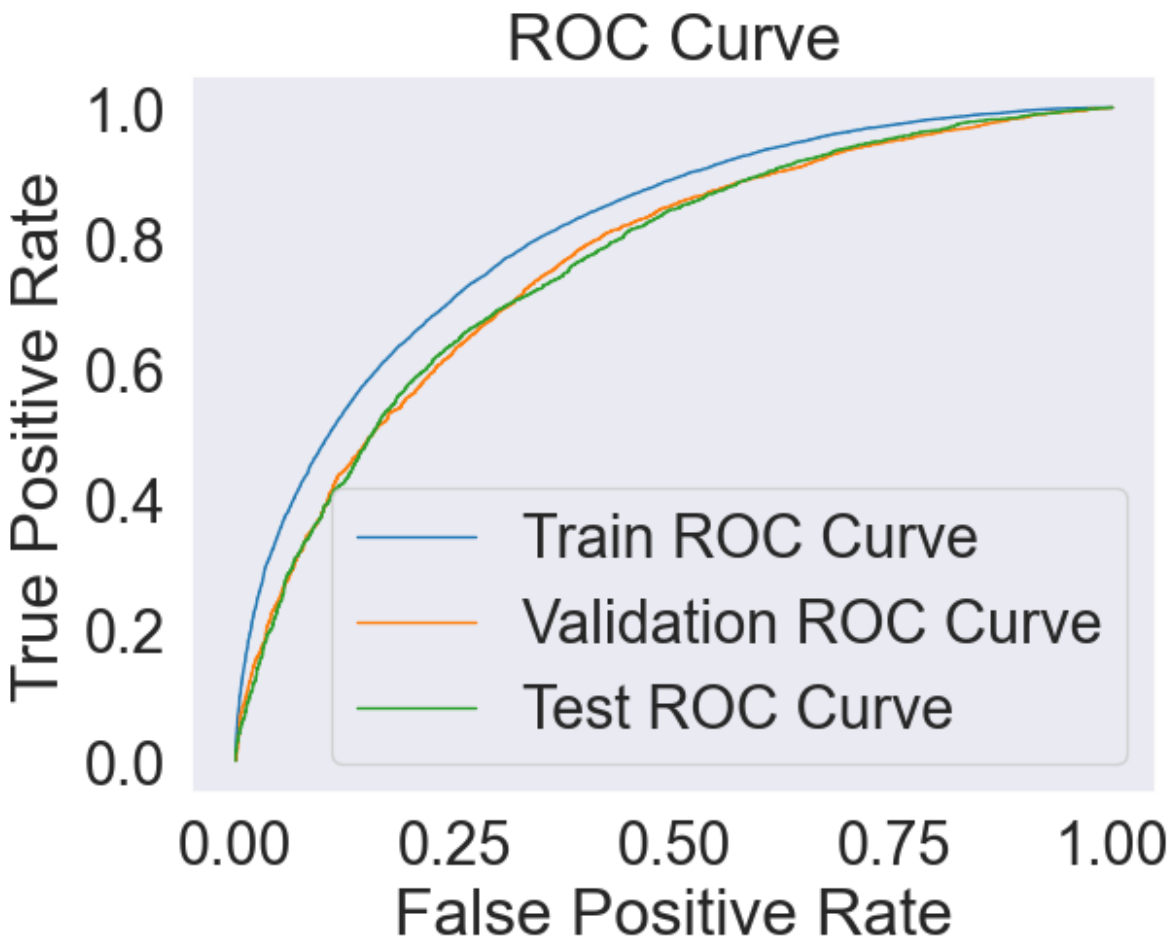
Model 8 - XGBoost

In [51]:

```
np.random.seed(42)

# creating pipeline by joining numerical and categorical pipelines
data_prep_pipeline, selected_features = get_pipeline()

# Attaching XGBoost model to the above pipeline
xgboost_full_pipeline_with_predictor = Pipeline([
    ("preparation", data_prep_pipeline),
    ("xgboost", XGBClassifier(random_state=42,
                objective='binary:logistic', max_depth=5, eta=0.001,
                learning_rate=0.01, colsample_bytree=0.7, n_estimators=1000))
])

# Training the model
start = time.time()
model = xgboost_full_pipeline_with_predictor.fit(X_train, y_train)
train_time = np.round(time.time() - start, 4)
```

```
start = time.time()
score_test  = xgboost_full_pipeline_with_predictor.score(X_test, y_test)
test_time = np.round(time.time() - start, 4)

# Results
exp_name = f"Model-8 XGBoost  "
experiment_description =f"XGBoost SAMME with undersampled data-2 {len(selected_fea
expLog = get_results(expLog, exp_name,experiment_description,model, train_time, te
expLog


# Model Training and Validation
model.fit(X_train, y_train)

# Model Predictions
train_preds = model.predict_proba(X_train)[:, 1]
valid_preds = model.predict_proba(X_valid)[:, 1]
test_preds = model.predict_proba(X_test)[:, 1]

# Compute Metrics
train_fpr, train_tpr, _ = roc_curve(y_train, train_preds)
valid_fpr, valid_tpr, _ = roc_curve(y_valid, valid_preds)
test_fpr, test_tpr, _ = roc_curve(y_test, test_preds)
# Plot ROC Curve
plt.plot(train_fpr, train_tpr, label="Train ROC Curve")
plt.plot(valid_fpr, valid_tpr, label="Validation ROC Curve")
plt.plot(test_fpr, test_tpr, label="Test ROC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()
expLog
```

```
Total Features: 124 - Numerical: 107, Categorical: 16
```


ROC Curve
True Positive Rate
1.0
0.8
0.6
0.4
0.2
0.0
0.00
0.25
0.50
0.75
1.00
False Positive Rate
Train ROC Curve
Validation ROC Curve
Test ROC Curve

Out[51]:

	exp_name	description	Train Time (sec)	Test Time (sec)	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC	T Sc
0	Model-1 Baseline LR	Logistic regression with undersampled data 124...	2.5144	0.0720	0.7727	0.7647	0.7802	0.7545	0.7421	0.7535	0.3
1	Model-2 Baseline LR	Logistic regression with undersampled data-2 1...	1.1938	0.0496	0.6876	0.6843	0.6904	0.7525	0.7489	0.7535	0.6
2	Model-3 KNN	KNN with undersampled data-2 124 features	0.3262	1.0494	0.6950	0.6155	0.6184	0.7625	0.6571	0.6550	0.6
3	Model-4 Decision Tree	Decision tree with undersampled data-2 124 fea...	1.4838	0.0504	0.6749	0.6535	0.6591	0.7380	0.7105	0.7129	0.6
4	Model-5 Random Forest	Random Forest with undersampled data-2 124 fea...	20.5597	0.4587	0.7665	0.6657	0.6666	0.8504	0.7245	0.7275	0.7
5	Model-6 Bagging Meta Estimator	Bagging Meta Estimator with undersampled data-...	5.4167	0.2396	0.9844	0.6445	0.6430	0.9990	0.6973	0.6924	0.9
6	Model-7 SVM	SVM with undersampled data-2 124 features	2681.6150	15.7188	0.9846	0.6411	0.6421	0.9990	0.6965	0.6899	0.9
7	Model-8 XGBoost	XGBoost SAMME with undersampled data-2 124 fea...	4.8359	0.0720	0.7311	0.6931	0.6955	0.8103	0.7614	0.7607	0.7

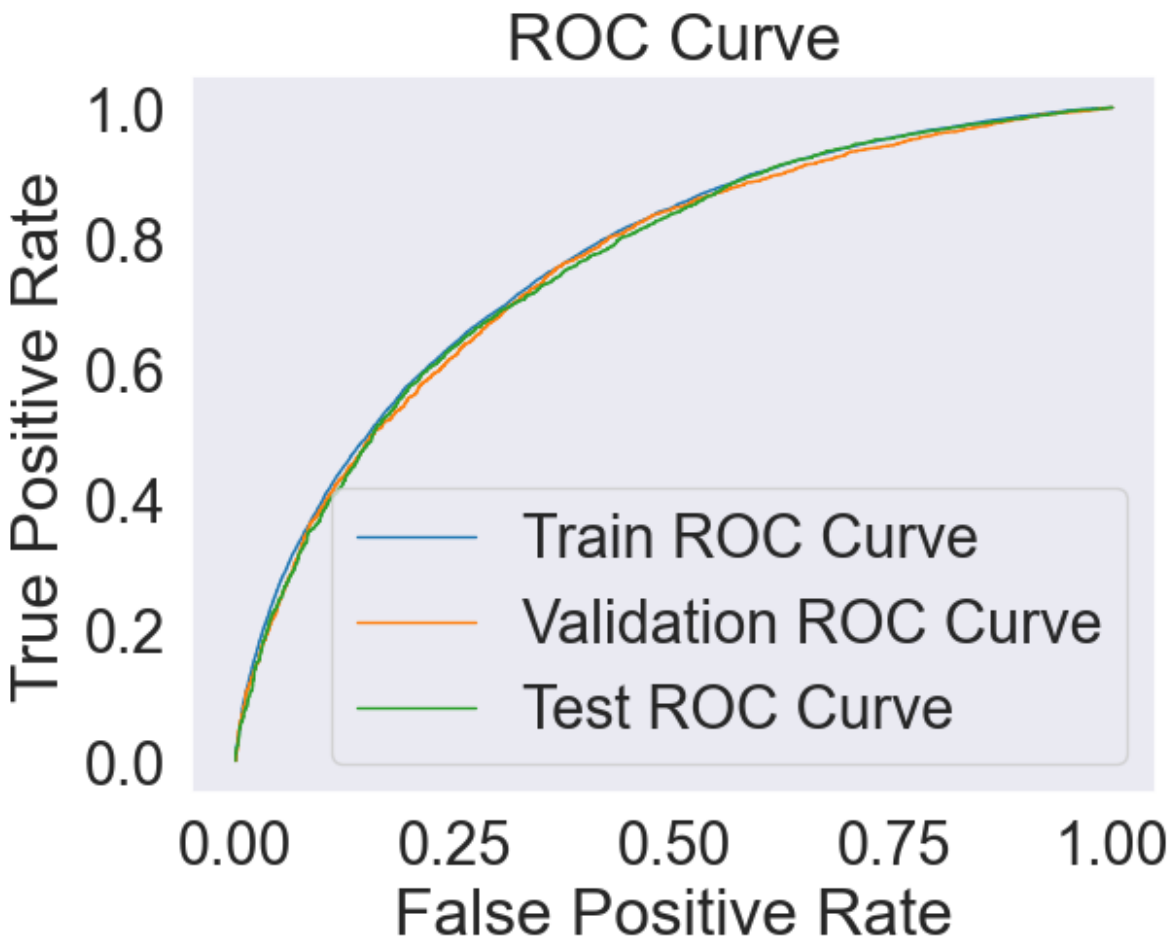
Model 9 - CatBoost

In [53]:

```
from catboost import CatBoostClassifier
from sklearn.pipeline import Pipeline
import numpy as np
import time

np.random.seed(42)

# creating pipeline by joining numerical and categorical pipelines
data_prep_pipeline, selected_features = get_pipeline()

# Attaching CatBoost model to the above pipeline
catboost_full_pipeline_with_predictor = Pipeline([
    ("preparation", data_prep_pipeline),
```

```
    ("catboost", CatBoostClassifier(random_state=42, iterations=1000, learning_rat
                                    depth=5, thread_count=-1, verbose=False))
])

# Training the model
start = time.time()
model = catboost_full_pipeline_with_predictor.fit(X_train, y_train)
train_time = np.round(time.time() - start, 4)

start = time.time()
score_test = catboost_full_pipeline_with_predictor.score(X_test, y_test)
test_time = np.round(time.time() - start, 4)

# Results
exp_name = f"Model-9 CATBoost  "
experiment_description =f"CATBoost with undersampled data-2 {len(selected_features
expLog = get_results(expLog, exp_name,experiment_description,model, train_time, te
expLog
# Model Training and Validation
model.fit(X_train, y_train)

# Model Predictions
train_preds = model.predict_proba(X_train)[:, 1]
valid_preds = model.predict_proba(X_valid)[:, 1]
test_preds = model.predict_proba(X_test)[:, 1]

# Compute Metrics
train_fpr, train_tpr, _ = roc_curve(y_train, train_preds)
valid_fpr, valid_tpr, _ = roc_curve(y_valid, valid_preds)
test_fpr, test_tpr, _ = roc_curve(y_test, test_preds)
# Plot ROC Curve
plt.plot(train_fpr, train_tpr, label="Train ROC Curve")
plt.plot(valid_fpr, valid_tpr, label="Validation ROC Curve")
plt.plot(test_fpr, test_tpr, label="Test ROC Curve")
plt.xlabel("False Positive Rate")
plt.ylabel("True Positive Rate")
plt.title("ROC Curve")
plt.legend()
plt.show()
expLog
```

```
Total Features: 124 - Numerical: 107, Categorical: 16
```


ROC Curve
True Positive Rate
1.0
0.8
0.6
0.4
0.2
0.0
0.00
0.25
0.50
0.75
1.00
False Positive Rate
Train ROC Curve
Validation ROC Curve
Test ROC Curve

Out[53]:

	exp_name	description	Train Time (sec)	Test Time (sec)	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC	T Sc
0	Model-1 Baseline LR	Logistic regression with undersampled data 124...	2.5144	0.0720	0.7727	0.7647	0.7802	0.7545	0.7421	0.7535	0.3
1	Model-2 Baseline LR	Logistic regression with undersampled data-2 1...	1.1938	0.0496	0.6876	0.6843	0.6904	0.7525	0.7489	0.7535	0.6
2	Model-3 KNN	KNN with undersampled data-2 124 features	0.3262	1.0494	0.6950	0.6155	0.6184	0.7625	0.6571	0.6550	0.6
3	Model-4 Decision Tree	Decision tree with undersampled data-2 124 fea...	1.4838	0.0504	0.6749	0.6535	0.6591	0.7380	0.7105	0.7129	0.6
4	Model-5 Random Forest	Random Forest with undersampled data-2 124 fea...	20.5597	0.4587	0.7665	0.6657	0.6666	0.8504	0.7245	0.7275	0.7
5	Model-6 Bagging Meta Estimator	Bagging Meta Estimator with undersampled data-...	5.4167	0.2396	0.9844	0.6445	0.6430	0.9990	0.6973	0.6924	0.9
6	Model-7 SVM	SVM with undersampled data-2 124 features	2681.6150	15.7188	0.9846	0.6411	0.6421	0.9990	0.6965	0.6899	0.9
7	Model-8 XGBoost	XGBoost SAMME with undersampled data-2 124 fea...	4.8359	0.0720	0.7311	0.6931	0.6955	0.8103	0.7614	0.7607	0.7
8	Model-9 CATBoost	CATBoost with undersampled data-2 124 features	12.1853	0.2737	0.6955	0.6917	0.6933	0.7669	0.7574	0.7593	0.6

Selecting 3 best Experimental Models , hyper parameter tuning , and ensemble learner

HYPERPARMETER TUNING AND FEATURE SELECTION

XGBOOST Hyper paramter Tuning and feature selection

In [54]:

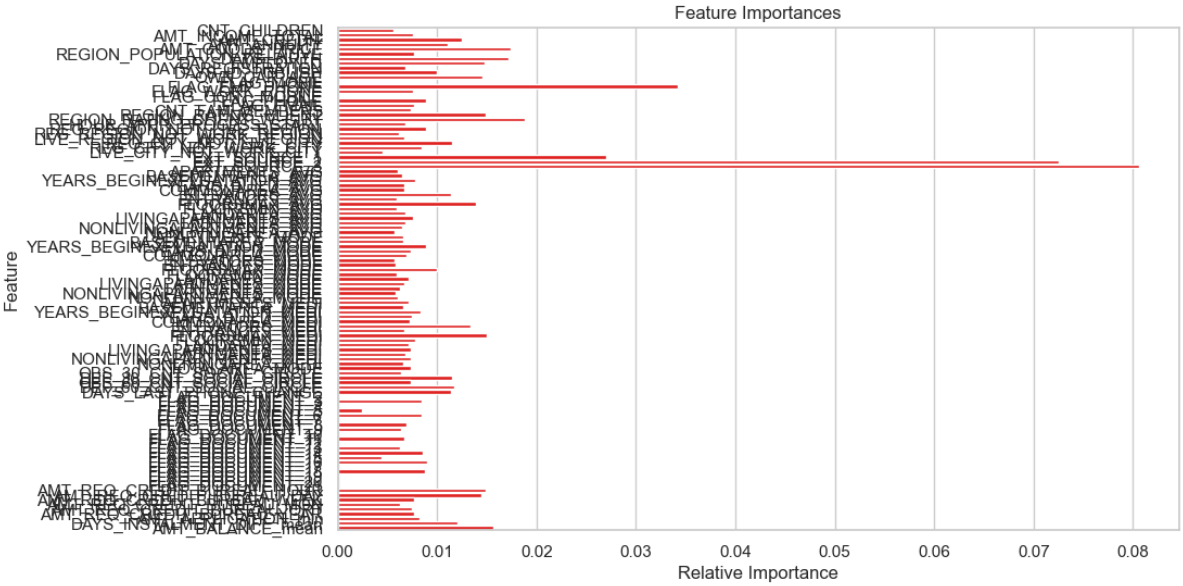
```
results = pd.DataFrame(columns=["ExpID", "Cross-fold Train Accuracy", "Test Accura
features_dict = dict()

# A Function to execute the grid search and record the results.
def ConductGridSearch(X_train, y_train, X_test, y_test):
    # classifier for our grid search experiment
    classifiers = [
#         ('DecisionTrees', DecisionTreeClassifier(random_state=42))
        ('XGBoost', XGBClassifier(random_state=42))
    ]

    # grid search parameters for the classifier
    param_grid = {

       'XGBoost':  {
            'max_depth': [5,9], # Lower helps with overfitting
            'n_estimators':[800, 1000],
            'learning_rate': [0.001, 0.01],
            'eta' : [0.001, 0.01],
            'colsample_bytree' : [0.5, 0.7],
        }
    }

#         # grid search parameters for the classifier
#     param_grid = {

#        'XGBoost':  {
#             'max_depth': [5], # Lower helps with overfitting
#             'n_estimators':[20],
#             'learning_rate': [ 0.1],
#             'eta' : [0.1],
#             'colsample_bytree' : [0.5],
#         }
#     }

    for (name, classifier) in classifiers:

        print('****** STARTING TUNING', name,'*****')
        parameters = param_grid[name]
        print("Parameters:")
        for p in sorted(parameters.keys()):
            print("\t"+str(p)+": "+ str(parameters[p]))

        # generate the pipeline
        full_pipeline_with_predictor = Pipeline([
            ("preparation", FeatureUnion(transformer_list=[("num_pipeline", numeri
            ("predictor", classifier)
        ])


        # Execute the grid search
        params = {}
        for p in parameters.keys():
            pipe_key = 'predictor__'+str(p)
            params[pipe_key] = parameters[p]

        grid_search = GridSearchCV(full_pipeline_with_predictor, params, scoring='
                                   n_jobs=-1, verbose=1)
        grid_search.fit(X_train, y_train)
```

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

        # Training accuracy
        cvSplits = ShuffleSplit(n_splits=3, test_size=0.7, random_state=42)
        best_train_scores = cross_val_score(full_pipeline_with_predictor,X_train ,
        best_train_accuracy = pct(best_train_scores.mean())

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


        # Importance of features
        features = numerical_features[:]
        print('\nTotal number of features:', len(features))
        importances = grid_search.best_estimator_.named_steps["predictor"].feature

        # selecting features based on importance values
        new_indices = [idx for idx, x in enumerate(importances) if x>0.01]
        new_importances = [x for idx, x in enumerate(importances) if x>0.01]
        new_features = [features[i] for i in new_indices]

        print('Total number of selected features:', len(new_features))



       # Plotting a barplot to visualize feature importance
        sns.set(style='whitegrid')
        plt.figure(figsize=(10, 6))
        sns.barplot(x=importances, y=features, color='red')
        plt.title('Feature Importances')
        plt.xlabel('Relative Importance')
        plt.ylabel('Feature')
        plt.show()


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

        # Best parameters found using grid search
        print(f"Best Parameters for {name}:")
        best_parameters = grid_search.best_estimator_.get_params()
        best_params = []
        for param_name in sorted(params.keys()):
            best_params.append((param_name, best_parameters[param_name]))
            print("\t"+str(param_name)+": " + str(best_parameters[param_name]))
        print("****** FINISHED TUNING",name," *****")

        # Results
        results.loc[len(results)] = [name, best_train_accuracy, best_test_accuracy

        # Storing the importances of the features
        features_dict['features'] = features
        features_dict['importances'] = importances
```

In [55]:

```
ConductGridSearch(X_train[numerical_features], y_train, X_test[numerical_features]
```

```
****** STARTING TUNING XGBoost *****
Parameters:
        colsample_bytree: [0.5, 0.7]
        eta: [0.001, 0.01]
        learning_rate: [0.001, 0.01]
        max_depth: [5, 9]
        n_estimators: [800, 1000]
Fitting 2 folds for each of 32 candidates, totalling 64 fits

Total number of features: 107
Total number of selected features: 25
```




```
Best Parameters for XGBoost:
        predictor__colsample_bytree: 0.5
        predictor__eta: 0.001
        predictor__learning_rate: 0.01
        predictor__max_depth: 5
        predictor__n_estimators: 1000
****** FINISHED TUNING XGBoost  *****
```

In [56]:

```
results
```

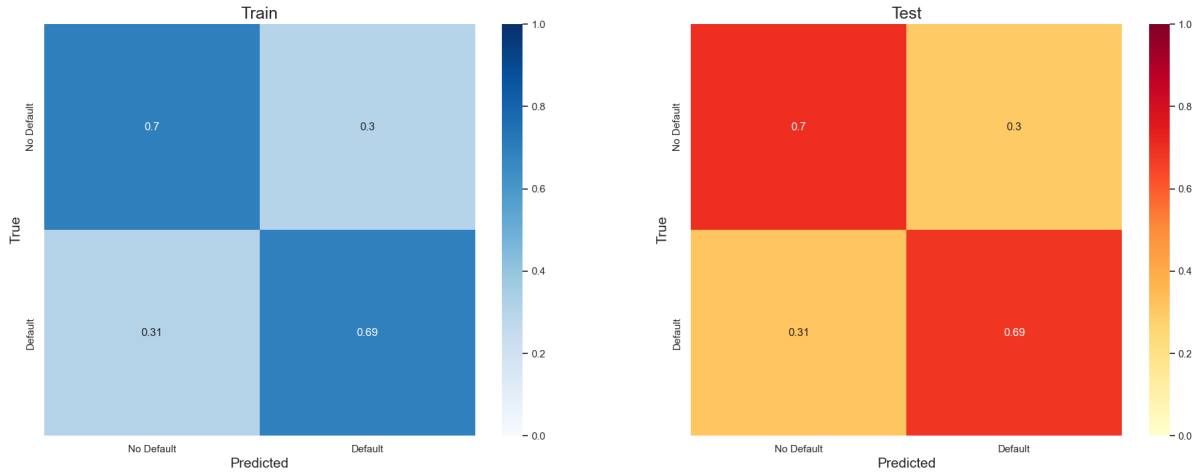
Out[56]:

	ExpID	Cross-fold Train Accuracy	Test Accuracy	p-value	Train Time(s)	Test Time(s)	Experiment Description
0	XGBoost	65.69	68.725	0.00314	2.3953	0.0416	[["predictor__colsample_bytree", 0.5], ["predi...

In [57]:

```
cm_train,cm_test=confusion_matrix_normalized(model,X_train,y_train,X_test,y_test)
fig, axes = plt.subplots(1, 2, figsize=(23, 8))

# Plot the first heatmap in the first subplot
sns.heatmap(cm_train, vmin=0, vmax=1, annot=True, cmap="Blues", ax=axes[0])
axes[0].set_xlabel("Predicted", fontsize=15)
axes[0].set_ylabel("True", fontsize=15)
axes[0].set_xticklabels(class_labels)
axes[0].set_yticklabels(class_labels)
axes[0].set_title("Train", fontsize=18)

# Plot the second heatmap in the second subplot
sns.heatmap(cm_test, vmin=0, vmax=1, annot=True, cmap="YlOrRd", ax=axes[1])
axes[1].set_xlabel("Predicted", fontsize=15)
axes[1].set_ylabel("True", fontsize=15)
```

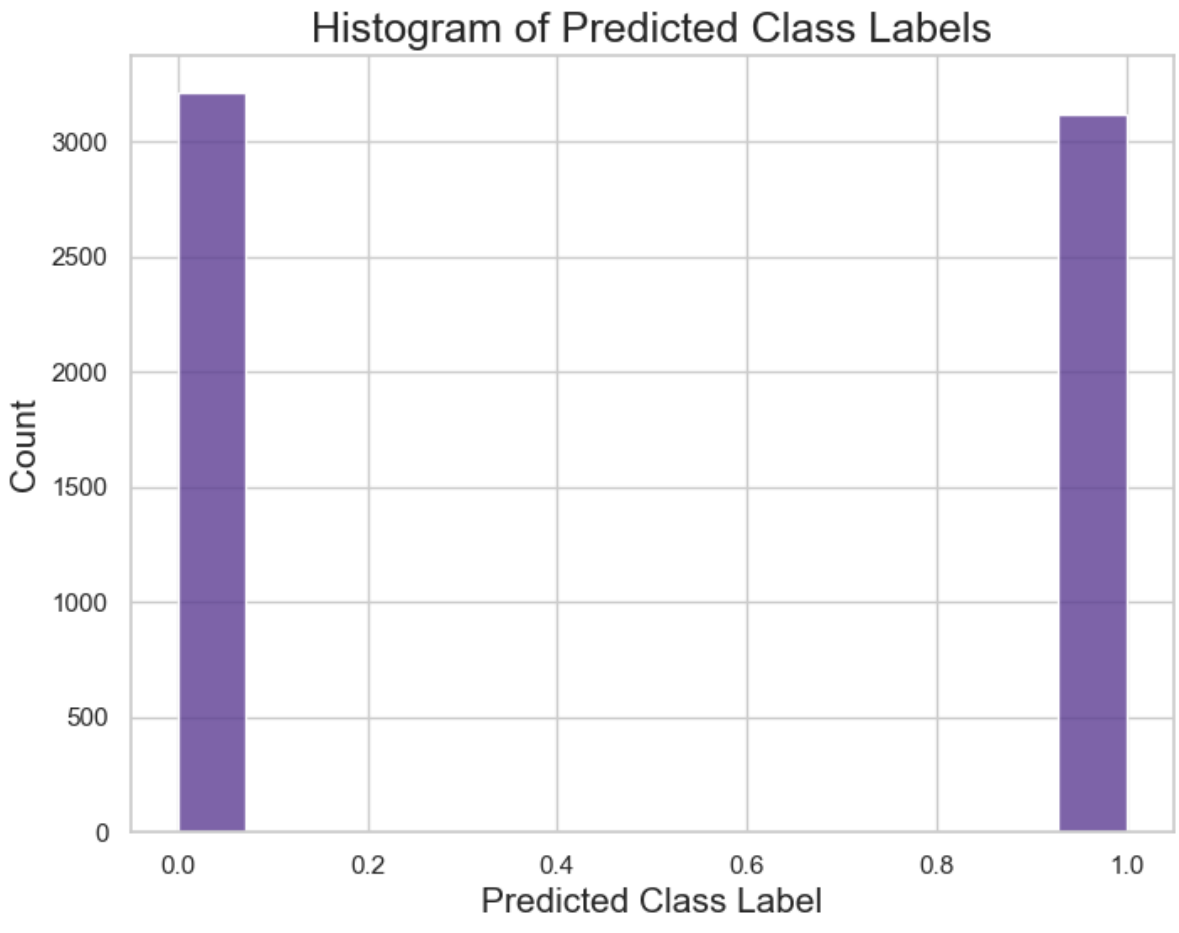
```
axes[1].set_xticklabels(class_labels)
axes[1].set_yticklabels(class_labels)
axes[1].set_title("Test", fontsize=18)

plt.show()
```



In [58]:

```
pred = model.predict(X_test)
# Create histogram of predicted class labels with a new color scheme
plt.figure(figsize=(8, 6))
sns.histplot(pred, kde=False, color="#5C3C92", alpha=0.8)
plt.xlabel("Predicted Class Label", fontsize=15)
plt.ylabel("Count", fontsize=15)
plt.title("Histogram of Predicted Class Labels", fontsize=18)
f1 = f1_score(y_test, pred)
print("F1 Score: ", f1)
```

```
F1 Score:  0.6915501905972046
```



In [59]:
```
with open('features_dict_XG.pickle', 'wb') as handle:
    pickle.dump(features_dict, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

In [60]:
```
with open('features_dict_XG.pickle', 'rb') as handle:
    x = pickle.load(handle)
```

We'll now perform feature selection and ruu model by filtering features based on their importance values. We'll use three different thresholds for feature importance: x>0, x>0.01, and x>0.005. By using these thresholds, we'll try to understand the impact of including only the most relevant features in my model.

Here's a brief explanation of each threshold:

`x>0` :

This threshold includes all features with a non-zero importance value. It means I am considering all the features that have any impact on the model's prediction.

`x>0.01` :

This threshold is more stringent, as I am only considering features with importance values greater than 0.01. This filter results in a smaller number of features , which may help reduce the complexity of the model and the risk of overfitting.

`x>0.005` :

This threshold is more stringentThis threshold lies between the other two. By using this threshold, I am considering features with importance values greater than 0.005. This results in a slightly larger number of features compared to the x>0.01 threshold.

The goal of using these different thresholds is to find the optimal balance between model complexity and predictive performance. By comparing the results of the models trained with different feature sets, I can identify the best trade-off between model simplicity and performance.

Model 10 - XGBOOST -Feature &hyperParameter Tuning x>0

In [61]:
```
features = features_dict['features']
importances = features_dict['importances']

new_indices = [idx for idx, x in enumerate(importances) if x>0]
new_importances = [x for idx, x in enumerate(importances) if x>0]

new_features = [features[i] for i in new_indices]
print(len(new_features))

num_attribs = new_features

np.random.seed(42)

# creating pipeline by joining numerical and categorical pipelines
data_prep_pipeline, selected_features = get_pipeline(num_attribs)

# Attaching XGBoost model to the above pipeline
xgboost_full_pipeline_with_predictor = Pipeline([
```

```
    ("preparation", data_prep_pipeline),
    ("xgboost", XGBClassifier(random_state=42,
                objective='binary:logistic', max_depth=5, eta=0.001,
                learning_rate=0.01, colsample_bytree=0.5, n_estimators=1000))
])

# Training the model
start = time.time()
model = xgboost_full_pipeline_with_predictor.fit(X_train, y_train)
train_time = np.round(time.time() - start, 4)

start = time.time()
score_test  = xgboost_full_pipeline_with_predictor.score(X_test, y_test)
test_time = np.round(time.time() - start, 4)

# Results
exp_name = f"Model 10 - XGBOOST -Feature &hyperParameter Tuning"
experiment_description =f"XGBOOST Tuned with x>0 {len(selected_features)} features
expLog = get_results(expLog, exp_name,experiment_description,model, train_time, te
expLog
```

```
96
Total Features: 113 - Numerical: 96, Categorical: 16
```

Out[61]:

	exp_name	description	Train Time (sec)	Test Time (sec)	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Te AU
0	Model-1 Baseline LR	Logistic regression with undersampled data 124...	2.5144	0.0720	0.7727	0.7647	0.7802	0.7545	0.7421	0.753
1	Model-2 Baseline LR	Logistic regression with undersampled data-2 1...	1.1938	0.0496	0.6876	0.6843	0.6904	0.7525	0.7489	0.753
2	Model-3 KNN	KNN with undersampled data-2 124 features	0.3262	1.0494	0.6950	0.6155	0.6184	0.7625	0.6571	0.655
3	Model-4 Decision Tree	Decision tree with undersampled data-2 124 fea...	1.4838	0.0504	0.6749	0.6535	0.6591	0.7380	0.7105	0.712
4	Model-5 Random Forest	Random Forest with undersampled data-2 124 fea...	20.5597	0.4587	0.7665	0.6657	0.6666	0.8504	0.7245	0.727
5	Model-6 Bagging Meta Estimator	Bagging Meta Estimator with undersampled data-...	5.4167	0.2396	0.9844	0.6445	0.6430	0.9990	0.6973	0.692
6	Model-7 SVM	SVM with undersampled data-2 124 features	2681.6150	15.7188	0.9846	0.6411	0.6421	0.9990	0.6965	0.689
7	Model-8 XGBoost	XGBoost SAMME with undersampled data-2 124 fea...	4.8359	0.0720	0.7311	0.6931	0.6955	0.8103	0.7614	0.760
8	Model-9 CATBoost	CATBoost with undersampled data-2 124 features	12.1853	0.2737	0.6955	0.6917	0.6933	0.7669	0.7574	0.759
9	Model 10 - XGBOOST - Feature &hyperParameter Tu...	XGBOOST Tuned with x>0 113 features	4.5328	0.0625	0.7294	0.6959	0.6983	0.8092	0.7616	0.762

Model 11 - XGBOOST -Feature & hyperParameter Tuning x>0.01

In [62]:

```
features = features_dict['features']
importances = features_dict['importances']
```

```
new_indices = [idx for idx, x in enumerate(importances) if x>0.01]
new_importances = [x for idx, x in enumerate(importances) if x>0.01]

print(len(new_features))

num_attribs = new_features

np.random.seed(42)

# creating pipeline by joining numerical and categorical pipelines
data_prep_pipeline, selected_features = get_pipeline(num_attribs)

# Attaching XGBoost model to the above pipeline
xgboost_full_pipeline_with_predictor = Pipeline([
    ("preparation", data_prep_pipeline),
    ("xgboost", XGBClassifier(random_state=42,
                objective='binary:logistic', max_depth=5, eta=0.001,
                learning_rate=0.01, colsample_bytree=0.5, n_estimators=1000))
])

# Training the model
start = time.time()
model = xgboost_full_pipeline_with_predictor.fit(X_train, y_train)
train_time = np.round(time.time() - start, 4)

start = time.time()
score_test  = xgboost_full_pipeline_with_predictor.score(X_test, y_test)
test_time = np.round(time.time() - start, 4)

# Results
# Results
exp_name = f"Model 11 - XGBOOST -Feature &hyperParameter Tuning"
experiment_description =f"XGBOOST Tuned with x>0.01 {len(selected_features)} featu
expLog = get_results(expLog, exp_name,experiment_description,model, train_time, te
expLog
```

```
96
Total Features: 113 - Numerical: 96, Categorical: 16
```

Out[62]:

	exp_name	description	Train Time (sec)	Test Time (sec)	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	T A
0	Model-1 Baseline LR	Logistic regression with undersampled data 124...	2.5144	0.0720	0.7727	0.7647	0.7802	0.7545	0.7421	0.7
1	Model-2 Baseline LR	Logistic regression with undersampled data-2 1...	1.1938	0.0496	0.6876	0.6843	0.6904	0.7525	0.7489	0.7
2	Model-3 KNN	KNN with undersampled data-2 124 features	0.3262	1.0494	0.6950	0.6155	0.6184	0.7625	0.6571	0.6
3	Model-4 Decision Tree	Decision tree with undersampled data-2 124 fea...	1.4838	0.0504	0.6749	0.6535	0.6591	0.7380	0.7105	0.7
4	Model-5 Random Forest	Random Forest with undersampled data-2 124 fea...	20.5597	0.4587	0.7665	0.6657	0.6666	0.8504	0.7245	0.7
5	Model-6 Bagging Meta Estimator	Bagging Meta Estimator with undersampled data-...	5.4167	0.2396	0.9844	0.6445	0.6430	0.9990	0.6973	0.6
6	Model-7 SVM	SVM with undersampled data-2 124 features	2681.6150	15.7188	0.9846	0.6411	0.6421	0.9990	0.6965	0.6
7	Model-8 XGBoost	XGBoost SAMME with undersampled data-2 124 fea...	4.8359	0.0720	0.7311	0.6931	0.6955	0.8103	0.7614	0.7
8	Model-9 CATBoost	CATBoost with undersampled data-2 124 features	12.1853	0.2737	0.6955	0.6917	0.6933	0.7669	0.7574	0.7
9	Model 10 - XGBOOST - Feature &hyperParameter Tu...	XGBOOST Tuned with x>0 113 features	4.5328	0.0625	0.7294	0.6959	0.6983	0.8092	0.7616	0.7
10	Model 11 - XGBOOST - Feature &hyperParameter Tu...	XGBOOST Tuned with x>0.01 113 features	4.4022	0.0781	0.7294	0.6959	0.6983	0.8092	0.7616	0.7

Model 12 - XGBOOST -Feature &hyperParameter Tuning x>0.005

In [63]:

```
features = features_dict['features']
importances = features_dict['importances']

new_indices = [idx for idx, x in enumerate(importances) if x>0.005]
new_importances = [x for idx, x in enumerate(importances) if x>0.005]

new_features = [features[i] for i in new_indices]
print(len(new_features))

num_attribs = new_features

np.random.seed(42)

# creating pipeline by joining numerical and categorical pipelines
data_prep_pipeline, selected_features = get_pipeline(num_attribs)

# Attaching XGBoost model to the above pipeline
xgboost_full_pipeline_with_predictor = Pipeline([
    ("preparation", data_prep_pipeline),
    ("xgboost", XGBClassifier(random_state=42,
                objective='binary:logistic', max_depth=5, eta=0.001,
                learning_rate=0.01, colsample_bytree=0.7, n_estimators=1000))
])

# Training the model
start = time.time()
model = xgboost_full_pipeline_with_predictor.fit(X_train, y_train)
train_time = np.round(time.time() - start, 4)

start = time.time()
score_test  = xgboost_full_pipeline_with_predictor.score(X_test, y_test)
test_time = np.round(time.time() - start, 4)

# Results
# Results
exp_name = f"Model 12 - XGBOOST -Feature &hyperParameter Tuning"
experiment_description =f"XGBOOST Tuned with x>0.005 {len(selected_features)} feat
expLog = get_results(expLog, exp_name,experiment_description,model, train_time, te
expLog
```

```
93
Total Features: 110 - Numerical: 93, Categorical: 16
```

Out[63]:

	exp_name	description	Train Time (sec)	Test Time (sec)	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	T A
0	Model-1 Baseline LR	Logistic regression with undersampled data 124...	2.5144	0.0720	0.7727	0.7647	0.7802	0.7545	0.7421	0.7
1	Model-2 Baseline LR	Logistic regression with undersampled data-2 1...	1.1938	0.0496	0.6876	0.6843	0.6904	0.7525	0.7489	0.7
2	Model-3 KNN	KNN with undersampled data-2 124 features	0.3262	1.0494	0.6950	0.6155	0.6184	0.7625	0.6571	0.6
3	Model-4 Decision Tree	Decision tree with undersampled data-2 124 fea...	1.4838	0.0504	0.6749	0.6535	0.6591	0.7380	0.7105	0.7
4	Model-5 Random Forest	Random Forest with undersampled data-2 124 fea...	20.5597	0.4587	0.7665	0.6657	0.6666	0.8504	0.7245	0.7
5	Model-6 Bagging Meta Estimator	Bagging Meta Estimator with undersampled data-...	5.4167	0.2396	0.9844	0.6445	0.6430	0.9990	0.6973	0.6
6	Model-7 SVM	SVM with undersampled data-2 124 features	2681.6150	15.7188	0.9846	0.6411	0.6421	0.9990	0.6965	0.6
7	Model-8 XGBoost	XGBoost SAMME with undersampled data-2 124 fea...	4.8359	0.0720	0.7311	0.6931	0.6955	0.8103	0.7614	0.7
8	Model-9 CATBoost	CATBoost with undersampled data-2 124 features	12.1853	0.2737	0.6955	0.6917	0.6933	0.7669	0.7574	0.7
9	Model 10 - XGBOOST - Feature &hyperParameter Tu...	XGBOOST Tuned with x>0 113 features	4.5328	0.0625	0.7294	0.6959	0.6983	0.8092	0.7616	0.7
10	Model 11 - XGBOOST - Feature &hyperParameter Tu...	XGBOOST Tuned with x>0.01 113 features	4.4022	0.0781	0.7294	0.6959	0.6983	0.8092	0.7616	0.7

	exp_name	description	Train Time (sec)	Test Time (sec)	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	T A
11	Model 12 - XGBOOST - Feature &hyperParameter T...	XGBOOST Tuned with x>0.005 110 features	4.4098	0.0666	0.7298	0.6933	0.6961	0.8104	0.7619	0.7(

CatBoost Hyper Parameter Tuning and Feature Selection

In [64]:

```
import time
import json
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from scipy import stats
from sklearn.model_selection import GridSearchCV, ShuffleSplit, cross_val_score
from sklearn.pipeline import Pipeline, FeatureUnion
from catboost import CatBoostClassifier

# Helper function to convert decimal to percentage
def pct(decimal):
    return round(decimal * 100, 2)

results = pd.DataFrame(columns=["ExpID", "Cross-fold Train Accuracy", "Test Accura
features_dict = dict()

# A Function to execute the grid search and record the results.
def ConductGridSearch(X_train, y_train, X_test, y_test):
    # classifier for our grid search experiment
    classifiers = [
        ('CatBoost', CatBoostClassifier(random_state=42, verbose=False))
    ]

    # grid search parameters for the classifier
    param_grid = {
       'CatBoost':  {
            'depth': [5, 9],
            'iterations': [800, 1000],
            'learning_rate': [0.001, 0.01],
            'colsample_bylevel': [0.5, 0.7],
        }
    }

#       # grid search parameters for the classifier
#     param_grid = {
#        'CatBoost':  {
#             'depth': [5],
#             'iterations': [20],
#             'learning_rate': [ 0.01],
#             'colsample_bylevel': [0.5],
#         }
#     }

    for (name, classifier) in classifiers:

        #name = "example"
        print(f"****** STARTING {name.upper()} *****")

        parameters = param_grid[name]
```

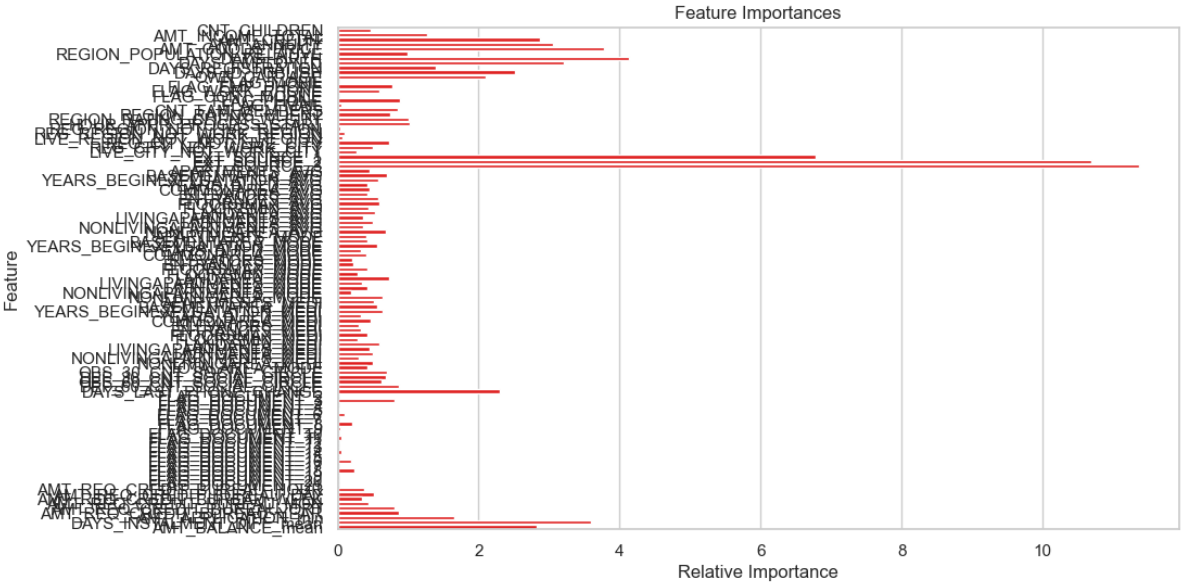
```
        print("Parameters:")
        for p in sorted(parameters.keys()):
            print("\t"+str(p)+": "+ str(parameters[p]))

        # generate the pipeline
        full_pipeline_with_predictor = Pipeline([
            ("preparation", FeatureUnion(transformer_list=[("num_pipeline", numeri
            ("predictor", classifier)
        ])


        # Execute the grid search
        params = {}
        for p in parameters.keys():
            pipe_key = 'predictor__'+str(p)
            params[pipe_key] = parameters[p]

        grid_search = GridSearchCV(full_pipeline_with_predictor, params, scoring='
                                   n_jobs=-1, verbose=1)
        grid_search.fit(X_train, y_train)


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

        # Training accuracy
        cvSplits = ShuffleSplit(n_splits=3, test_size=0.7, random_state=42)
        best_train_scores = cross_val_score(full_pipeline_with_predictor,X_train ,
        best_train_accuracy = pct(best_train_scores.mean())

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


        # Importance of features
        features = numerical_features[:]
        print('\nTotal number of features:', len(features))
        importances = grid_search.best_estimator_.named_steps["predictor"].feature

        # selecting features based on importance values
        new_indices = [idx for idx, x in enumerate(importances) if x>0.01]
        new_importances = [x for idx, x in enumerate(importances) if x>0.01]
        new_features = [features[i] for i in new_indices]

        print('Total number of selected features:', len(new_features))

        # Plotting a barplot to visualize feature importance
        sns.set(style='whitegrid')
        plt.figure(figsize=(10, 6))
        sns.barplot(x=importances, y=features, color='red')
        plt.title('Feature Importances')
        plt.xlabel('Relative Importance')
        plt.ylabel('Feature')
        plt.show()


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

        # Best parameters found using grid search
```

```
        print(f"Best Parameters for {name}:")
        best_parameters = grid_search.best_estimator_.get_params()
        best_params = []
        for param_name in sorted(params.keys()):
            best_params.append((param_name, best_parameters[param_name]))
            print("\t"+str(param_name)+": " + str(best_parameters[param_name]))

        print(f"****** FINISHED {name.upper()} *****")

        # Results
        results.loc[len(results)] = [name, best_train_accuracy, best_test_accuracy

        # Storing the importances of the features
        features_dict['features'] = features
        features_dict['importances'] = importances
```

In [65]:

```
ConductGridSearch(X_train[numerical_features], y_train, X_test[numerical_features]
```

```
****** STARTING CATBOOST *****
Parameters:
        colsample_bylevel: [0.5, 0.7]
        depth: [5, 9]
        iterations: [800, 1000]
        learning_rate: [0.001, 0.01]
Fitting 2 folds for each of 16 candidates, totalling 32 fits

Total number of features: 107
Total number of selected features: 95
```




```
Best Parameters for CatBoost:
        predictor__colsample_bylevel: 0.5
        predictor__depth: 9
        predictor__iterations: 1000
        predictor__learning_rate: 0.01
****** FINISHED CATBOOST *****
```

In [66]:

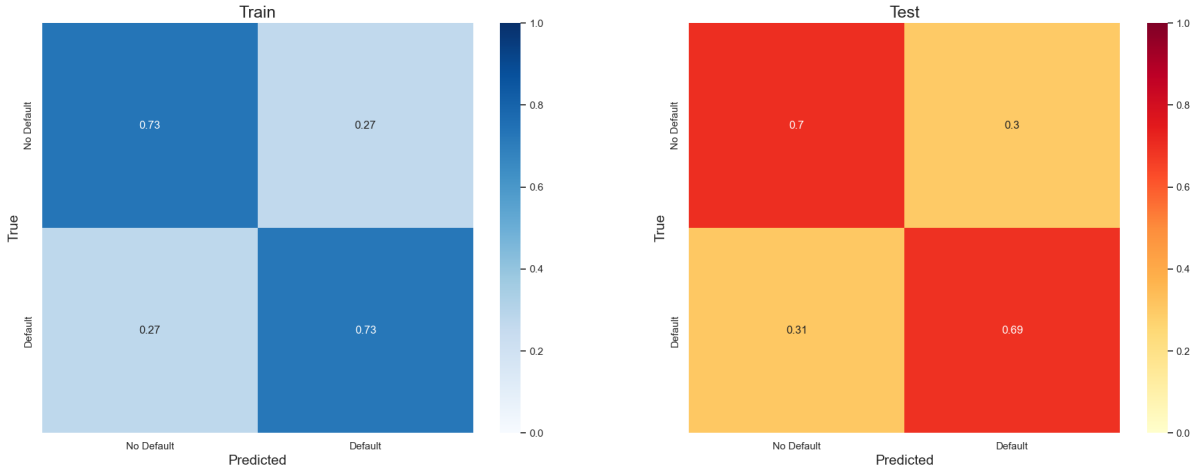
```
results
```

Out[66]:

	ExpID	Cross-fold Train Accuracy	Test Accuracy	p-value	Train Time(s)	Test Time(s)	Experiment Description
0	CatBoost	68.0	68.55	0.38437	33.3904	0.0251	[["predictor__colsample_bylevel", 0.5], ["pred...

In [67]:

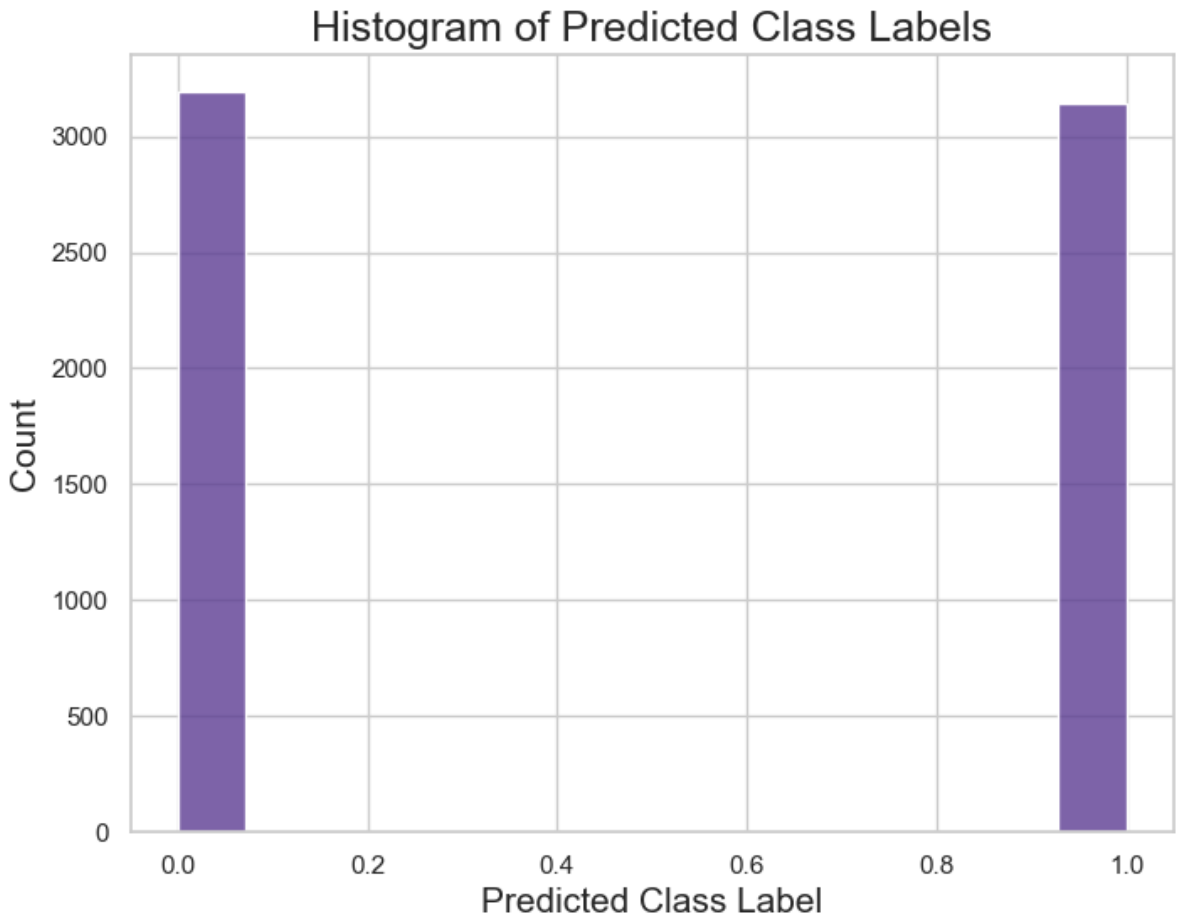
```
cm_train,cm_test=confusion_matrix_normalized(model,X_train,y_train,X_test,y_test)
fig, axes = plt.subplots(1, 2, figsize=(23, 8))

# Plot the first heatmap in the first subplot
sns.heatmap(cm_train, vmin=0, vmax=1, annot=True, cmap="Blues", ax=axes[0])
axes[0].set_xlabel("Predicted", fontsize=15)
axes[0].set_ylabel("True", fontsize=15)
axes[0].set_xticklabels(class_labels)
axes[0].set_yticklabels(class_labels)
axes[0].set_title("Train", fontsize=18)

# Plot the second heatmap in the second subplot
sns.heatmap(cm_test, vmin=0, vmax=1, annot=True, cmap="YlOrRd", ax=axes[1])
axes[1].set_xlabel("Predicted", fontsize=15)
axes[1].set_ylabel("True", fontsize=15)
axes[1].set_xticklabels(class_labels)
axes[1].set_yticklabels(class_labels)
axes[1].set_title("Test", fontsize=18)

plt.show()
```



In [68]:

```
pred = model.predict(X_test)
# Create histogram of predicted class labels with a new color scheme
plt.figure(figsize=(8, 6))
sns.histplot(pred, kde=False, color="#5C3C92", alpha=0.8)
plt.xlabel("Predicted Class Label", fontsize=15)
plt.ylabel("Count", fontsize=15)
plt.title("Histogram of Predicted Class Labels", fontsize=18)
f1 = f1_score(y_test, pred)
f1 = f1_score(y_test, pred)
print("F1 Score: ", f1)
```

```
F1 Score:  0.695376820772641
```



In [69]:
```
with open('features_dict_catboost.pickle', 'wb') as handle:
    pickle.dump(features_dict, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

In [70]:
```
with open('features_dict_catboost.pickle', 'rb') as handle:
    x = pickle.load(handle)
```

Model 13 - CatBOOST -Feature &hyperParameter Tuning x>0

In [71]:
```
features = features_dict['features']
importances = features_dict['importances']

new_indices = [idx for idx, x in enumerate(importances) if x > 0]
new_importances = [x for idx, x in enumerate(importances) if x > 0]

new_features = [features[i] for i in new_indices]
print(len(new_features))

num_attribs = new_features

np.random.seed(42)

# creating pipeline by joining numerical and categorical pipelines
data_prep_pipeline, selected_features = get_pipeline(num_attribs)

# Attaching CatBoost model to the above pipeline
catboost_full_pipeline_with_predictor = Pipeline([
    ("preparation", data_prep_pipeline),
    ("catboost", CatBoostClassifier(random_state=42,
                iterations=1000, learning_rate=0.01, depth=9,
                colsample_bylevel=0.5, thread_count=-1, verbose=False))
```

```
])

# Training the model
start = time.time()
model = catboost_full_pipeline_with_predictor.fit(X_train, y_train)
train_time = np.round(time.time() - start, 4)

start = time.time()
score_test = catboost_full_pipeline_with_predictor.score(X_test, y_test)
test_time = np.round(time.time() - start, 4)

# Results
exp_name = f"Model 13 - CatBOOST -Feature &hyperParameter Tuning"
experiment_description =f"CatBOOST Tuned with x>0 {len(selected_features)} feature
expLog = get_results(expLog, exp_name,experiment_description,model, train_time, te
expLog
```

```
103
Total Features: 120 - Numerical: 103, Categorical: 16
```

Out[71]:

	exp_name	description	Train Time (sec)	Test Time (sec)	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	T A
0	Model-1 Baseline LR	Logistic regression with undersampled data 124...	2.5144	0.0720	0.7727	0.7647	0.7802	0.7545	0.7421	0.7
1	Model-2 Baseline LR	Logistic regression with undersampled data-2 1...	1.1938	0.0496	0.6876	0.6843	0.6904	0.7525	0.7489	0.7
2	Model-3 KNN	KNN with undersampled data-2 124 features	0.3262	1.0494	0.6950	0.6155	0.6184	0.7625	0.6571	0.6
3	Model-4 Decision Tree	Decision tree with undersampled data-2 124 fea...	1.4838	0.0504	0.6749	0.6535	0.6591	0.7380	0.7105	0.7
4	Model-5 Random Forest	Random Forest with undersampled data-2 124 fea...	20.5597	0.4587	0.7665	0.6657	0.6666	0.8504	0.7245	0.7
5	Model-6 Bagging Meta Estimator	Bagging Meta Estimator with undersampled data-...	5.4167	0.2396	0.9844	0.6445	0.6430	0.9990	0.6973	0.6
6	Model-7 SVM	SVM with undersampled data-2 124 features	2681.6150	15.7188	0.9846	0.6411	0.6421	0.9990	0.6965	0.6
7	Model-8 XGBoost	XGBoost SAMME with undersampled data-2 124 fea...	4.8359	0.0720	0.7311	0.6931	0.6955	0.8103	0.7614	0.7
8	Model-9 CATBoost	CATBoost with undersampled data-2 124 features	12.1853	0.2737	0.6955	0.6917	0.6933	0.7669	0.7574	0.7
9	Model 10 - XGBOOST - Feature &hyperParameter Tu...	XGBOOST Tuned with x>0 113 features	4.5328	0.0625	0.7294	0.6959	0.6983	0.8092	0.7616	0.7
10	Model 11 - XGBOOST - Feature &hyperParameter Tu...	XGBOOST Tuned with x>0.01 113 features	4.4022	0.0781	0.7294	0.6959	0.6983	0.8092	0.7616	0.7

	exp_name	description	Train Time (sec)	Test Time (sec)	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	T A
11	Model 12 - XGBOOST - Feature &hyperParameter Tu...	XGBOOST Tuned with x>0.005 110 features	4.4098	0.0666	0.7298	0.6933	0.6961	0.8104	0.7619	0.7(
12	Model 13 - CatBOOST - Feature &hyperParameter T	CatBOOST Tuned with x>0 120 features	38.4053	0.2749	0.7528	0.6935	0.6970	0.8367	0.7594	0.7(

Model 14 - CatBOOST -Feature & HyperParameter Tuning x>0.1

In [72]:

```
features = features_dict['features']
importances = features_dict['importances']

new_indices = [idx for idx, x in enumerate(importances) if x > 0.1]
new_importances = [x for idx, x in enumerate(importances) if x > 0.1]

new_features = [features[i] for i in new_indices]
print(len(new_features))

num_attribs = new_features

np.random.seed(42)

# creating pipeline by joining numerical and categorical pipelines
data_prep_pipeline, selected_features = get_pipeline(num_attribs)

# Attaching CatBoost model to the above pipeline
catboost_full_pipeline_with_predictor = Pipeline([
    ("preparation", data_prep_pipeline),
    ("catboost", CatBoostClassifier(random_state=42,
                iterations=1000, learning_rate=0.01, depth=9,
                colsample_bylevel=0.5, thread_count=-1, verbose=False))
])

# Training the model
start = time.time()
model = catboost_full_pipeline_with_predictor.fit(X_train, y_train)
train_time = np.round(time.time() - start, 4)

start = time.time()
score_test = catboost_full_pipeline_with_predictor.score(X_test, y_test)
test_time = np.round(time.time() - start, 4)

# Results
exp_name = f"Model 14 - CatBOOST -Feature &hyperParameter Tuning"
experiment_description =f"CatBOOST Tuned with x>0.1 {len(selected_features)} featu
expLog = get_results(expLog, exp_name,experiment_description,model, train_time, te
expLog
```

```
86
Total Features: 103 - Numerical: 86, Categorical: 16
```

Out[72]:

	exp_name	description	Train Time (sec)	Test Time (sec)	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	T A
0	Model-1 Baseline LR	Logistic regression with undersampled data 124...	2.5144	0.0720	0.7727	0.7647	0.7802	0.7545	0.7421	0.7
1	Model-2 Baseline LR	Logistic regression with undersampled data-2 1...	1.1938	0.0496	0.6876	0.6843	0.6904	0.7525	0.7489	0.7
2	Model-3 KNN	KNN with undersampled data-2 124 features	0.3262	1.0494	0.6950	0.6155	0.6184	0.7625	0.6571	0.6
3	Model-4 Decision Tree	Decision tree with undersampled data-2 124 fea...	1.4838	0.0504	0.6749	0.6535	0.6591	0.7380	0.7105	0.7
4	Model-5 Random Forest	Random Forest with undersampled data-2 124 fea...	20.5597	0.4587	0.7665	0.6657	0.6666	0.8504	0.7245	0.7
5	Model-6 Bagging Meta Estimator	Bagging Meta Estimator with undersampled data-...	5.4167	0.2396	0.9844	0.6445	0.6430	0.9990	0.6973	0.6
6	Model-7 SVM	SVM with undersampled data-2 124 features	2681.6150	15.7188	0.9846	0.6411	0.6421	0.9990	0.6965	0.6
7	Model-8 XGBoost	XGBoost SAMME with undersampled data-2 124 fea...	4.8359	0.0720	0.7311	0.6931	0.6955	0.8103	0.7614	0.7
8	Model-9 CATBoost	CATBoost with undersampled data-2 124 features	12.1853	0.2737	0.6955	0.6917	0.6933	0.7669	0.7574	0.7
9	Model 10 - XGBOOST - Feature &hyperParameter Tu...	XGBOOST Tuned with x>0 113 features	4.5328	0.0625	0.7294	0.6959	0.6983	0.8092	0.7616	0.7
10	Model 11 - XGBOOST - Feature &hyperParameter Tu...	XGBOOST Tuned with x>0.01 113 features	4.4022	0.0781	0.7294	0.6959	0.6983	0.8092	0.7616	0.7

	exp_name	description	Train Time (sec)	Test Time (sec)	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	T A
11	Model 12 - XGBOOST - Feature &hyperParameter Tu...	XGBOOST Tuned with x>0.005 110 features	4.4098	0.0666	0.7298	0.6933	0.6961	0.8104	0.7619	0.7(
12	Model 13 - CatBOOST - Feature &hyperParameter T...	CatBOOST Tuned with x>0 120 features	38.4053	0.2749	0.7528	0.6935	0.6970	0.8367	0.7594	0.7(
13	Model 14 - CatBOOST - Feature &hyperParameter T...	CatBOOST Tuned with x>0.1 103 features	38.2332	0.2384	0.7513	0.6897	0.6944	0.8358	0.7590	0.7(

Model 15 - CatBOOST -Feature &hyperParameter Tuning x>0.005

In [73]:

```
features = features_dict['features']
importances = features_dict['importances']

new_indices = [idx for idx, x in enumerate(importances) if x > 0.005]
new_importances = [x for idx, x in enumerate(importances) if x > 0.005]

new_features = [features[i] for i in new_indices]
print(len(new_features))

num_attribs = new_features

np.random.seed(42)

# creating pipeline by joining numerical and categorical pipelines
data_prep_pipeline, selected_features = get_pipeline(num_attribs)

# Attaching CatBoost model to the above pipeline
catboost_full_pipeline_with_predictor = Pipeline([
    ("preparation", data_prep_pipeline),
    ("catboost", CatBoostClassifier(random_state=42,
                iterations=1000, learning_rate=0.01, depth=9,
                colsample_bylevel=0.5, thread_count=-1, verbose=False))
])

# Training the model
start = time.time()
model = catboost_full_pipeline_with_predictor.fit(X_train, y_train)
train_time = np.round(time.time() - start, 4)

start = time.time()
score_test = catboost_full_pipeline_with_predictor.score(X_test, y_test)
test_time = np.round(time.time() - start, 4)

# Results
exp_name = f"Model 15 - CatBOOST -Feature &hyperParameter Tuning"
experiment_description =f"CatBOOST Tuned with x>0.005 {len(selected_features)} fea
expLog = get_results(expLog, exp_name,experiment_description,model, train_time, te
expLog
```

```
96
Total Features: 113 - Numerical: 96, Categorical: 16
```

Out[73]:

	exp_name	description	Train Time (sec)	Test Time (sec)	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	T A
0	Model-1 Baseline LR	Logistic regression with undersampled data 124...	2.5144	0.0720	0.7727	0.7647	0.7802	0.7545	0.7421	0.7
1	Model-2 Baseline LR	Logistic regression with undersampled data-2 1...	1.1938	0.0496	0.6876	0.6843	0.6904	0.7525	0.7489	0.7
2	Model-3 KNN	KNN with undersampled data-2 124 features	0.3262	1.0494	0.6950	0.6155	0.6184	0.7625	0.6571	0.6
3	Model-4 Decision Tree	Decision tree with undersampled data-2 124 fea...	1.4838	0.0504	0.6749	0.6535	0.6591	0.7380	0.7105	0.7
4	Model-5 Random Forest	Random Forest with undersampled data-2 124 fea...	20.5597	0.4587	0.7665	0.6657	0.6666	0.8504	0.7245	0.7
5	Model-6 Bagging Meta Estimator	Bagging Meta Estimator with undersampled data-...	5.4167	0.2396	0.9844	0.6445	0.6430	0.9990	0.6973	0.6
6	Model-7 SVM	SVM with undersampled data-2 124 features	2681.6150	15.7188	0.9846	0.6411	0.6421	0.9990	0.6965	0.6
7	Model-8 XGBoost	XGBoost SAMME with undersampled data-2 124 fea...	4.8359	0.0720	0.7311	0.6931	0.6955	0.8103	0.7614	0.7
8	Model-9 CATBoost	CATBoost with undersampled data-2 124 features	12.1853	0.2737	0.6955	0.6917	0.6933	0.7669	0.7574	0.7
9	Model 10 - XGBOOST - Feature &hyperParameter Tu...	XGBOOST Tuned with x>0 113 features	4.5328	0.0625	0.7294	0.6959	0.6983	0.8092	0.7616	0.7
10	Model 11 - XGBOOST - Feature &hyperParameter Tu...	XGBOOST Tuned with x>0.01 113 features	4.4022	0.0781	0.7294	0.6959	0.6983	0.8092	0.7616	0.7

	exp_name	description	Train Time (sec)	Test Time (sec)	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	T A
11	Model 12 - XGBOOST - Feature &hyperParameter Tu...	XGBOOST Tuned with x>0.005 110 features	4.4098	0.0666	0.7298	0.6933	0.6961	0.8104	0.7619	0.7(
12	Model 13 - CatBOOST - Feature &hyperParameter T...	CatBOOST Tuned with x>0 120 features	38.4053	0.2749	0.7528	0.6935	0.6970	0.8367	0.7594	0.7(
13	Model 14 - CatBOOST - Feature &hyperParameter T...	CatBOOST Tuned with x>0.1 103 features	38.2332	0.2384	0.7513	0.6897	0.6944	0.8358	0.7590	0.7(
14	Model 15 - CatBOOST - Feature &hyperParameter T	CatBOOST Tuned with x>0.005 113 features	38.7217	0.2463	0.7513	0.6931	0.6959	0.8360	0.7596	0.7(

Random Forest Hyper Parameter Tuning and Feature Selection

In [74]:

```
from sklearn.ensemble import RandomForestClassifier

results = pd.DataFrame(columns=["ExpID", "Cross-fold Train Accuracy", "Test Accura
features_dict = dict()

# A Function to execute the grid search and record the results.
def ConductGridSearch(X_train, y_train, X_test, y_test):
    # classifier for our grid search experiment
    classifiers = [
        ('RandomForest', RandomForestClassifier(random_state=42))
    ]

    # grid search parameters for the classifier
    param_grid = {
        'RandomForest':  {
            'n_estimators': [100, 200],
            'max_depth': [5, 10],
            'min_samples_split': [2, 5],
            'min_samples_leaf': [1, 2],
            'max_features': ['auto', 'sqrt']
        }
    }

    for (name, classifier) in classifiers:

        print('****** START', name,'*****')
        parameters = param_grid[name]
        print("Parameters:")
        for p in sorted(parameters.keys()):
            print("\t"+str(p)+": "+ str(parameters[p]))

        # generate the pipeline
        full_pipeline_with_predictor = Pipeline([
```

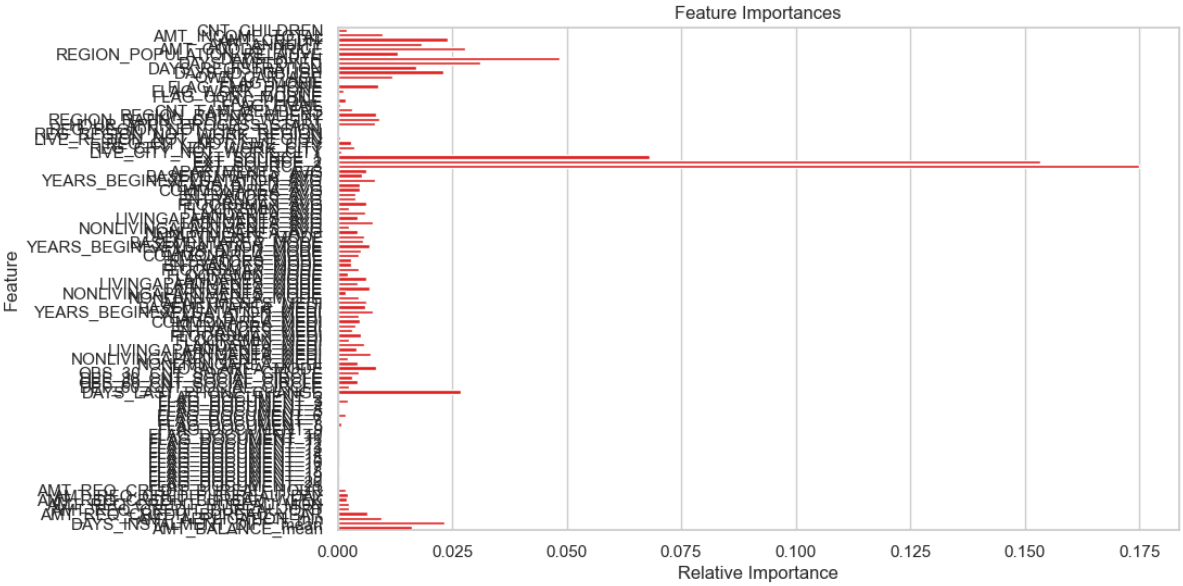
```
        ("preparation", FeatureUnion(transformer_list=[("num_pipeline", numeri
        ("predictor", classifier)
    ])


    # Execute the grid search
    params = {}
    for p in parameters.keys():
        pipe_key = 'predictor__'+str(p)
        params[pipe_key] = parameters[p]

    grid_search = GridSearchCV(full_pipeline_with_predictor, params, scoring='
                               n_jobs=-1, verbose=1)
    grid_search.fit(X_train, y_train)


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

    # Training accuracy
    cvSplits = ShuffleSplit(n_splits=3, test_size=0.7, random_state=42)
    best_train_scores = cross_val_score(full_pipeline_with_predictor,X_train ,
    best_train_accuracy = pct(best_train_scores.mean())

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


    # Importance of features
    features = numerical_features[:]
    print('\nTotal number of features:', len(features))
    importances = grid_search.best_estimator_.named_steps["predictor"].feature

    # selecting features based on importance values
    new_indices = [idx for idx, x in enumerate(importances) if x>0.01]
    new_importances = [x for idx, x in enumerate(importances) if x>0.01]
    new_features = [features[i] for i in new_indices]

    print('Total number of selected features:', len(new_features))


     # Plotting a barplot to visualize feature importance
    sns.set(style='whitegrid')
    plt.figure(figsize=(10, 6))
    sns.barplot(x=importances, y=features, color='red')
    plt.title('Feature Importances')
    plt.xlabel('Relative Importance')
    plt.ylabel('Feature')
    plt.show()

     # Conduct t-test with baseline logit and best estimator
    (t_stat, p_value) = stats.ttest_rel(logit_scores, best_train_scores)
        # Best parameters found using grid search
    print(f"Best Parameters for {name}:")
    best_parameters = grid_search.best_estimator_.get_params()
    best_params = []
    for param_name in sorted(params.keys()):
        best_params.append((param_name, best_parameters[param_name]))
        print("\t"+str(param_name)+": " + str(best_parameters[param_name]))
    print("****** FINISH",name," *****")
```

```
        # Results
        results.loc[len(results)] = [name, best_train_accuracy, best_test_accuracy

        # Storing the importances of the features
        features_dict['features'] = features
        features_dict['importances'] = importances
```

In [75]:

```
ConductGridSearch(X_train[numerical_features], y_train, X_test[numerical_features]
```

```
****** START RandomForest *****
Parameters:
        max_depth: [5, 10]
        max_features: ['auto', 'sqrt']
        min_samples_leaf: [1, 2]
        min_samples_split: [2, 5]
        n_estimators: [100, 200]
Fitting 2 folds for each of 32 candidates, totalling 64 fits

Total number of features: 107
Total number of selected features: 15
```




```
Best Parameters for RandomForest:
        predictor__max_depth: 10
        predictor__max_features: sqrt
        predictor__min_samples_leaf: 2
        predictor__min_samples_split: 5
        predictor__n_estimators: 200
****** FINISH RandomForest  *****
```

In [76]:

```
results
```

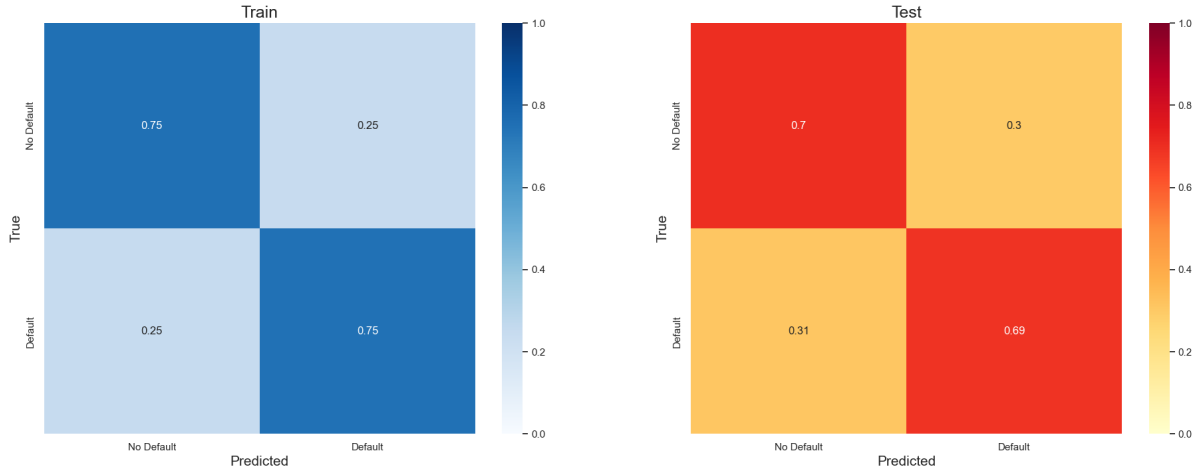
Out[76]:

	ExpID	Cross-fold Train Accuracy	Test Accuracy	p-value	Train Time(s)	Test Time(s)	Experiment Description
0	RandomForest	66.62	67.18	0.00064	15.5661	0.125	[["predictor__max_depth", 10], ["predictor__ma...

In [77]:

```
cm_train,cm_test=confusion_matrix_normalized(model,X_train,y_train,X_test,y_test)
fig, axes = plt.subplots(1, 2, figsize=(23, 8))

# Plot the first heatmap in the first subplot
```

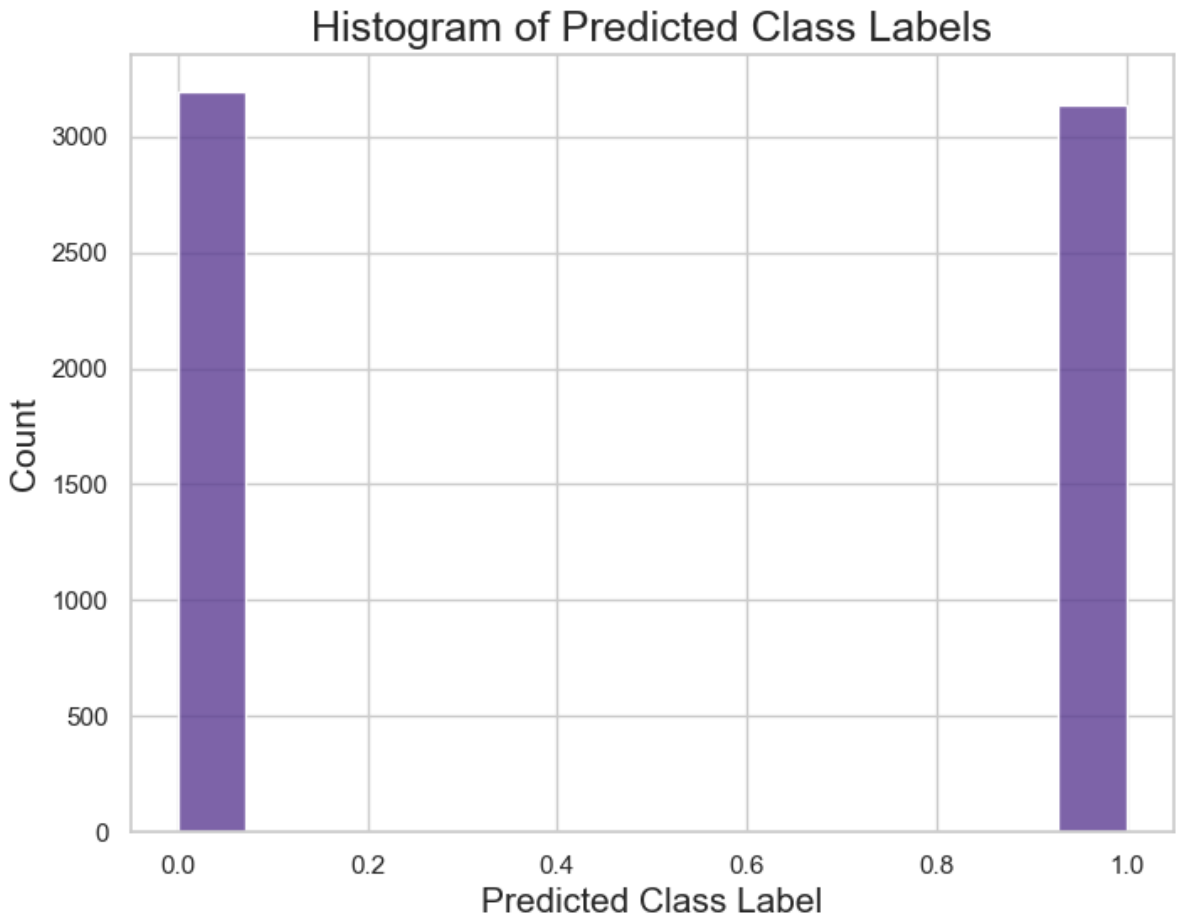
```
sns.heatmap(cm_train, vmin=0, vmax=1, annot=True, cmap="Blues", ax=axes[0])
axes[0].set_xlabel("Predicted", fontsize=15)
axes[0].set_ylabel("True", fontsize=15)
axes[0].set_xticklabels(class_labels)
axes[0].set_yticklabels(class_labels)
axes[0].set_title("Train", fontsize=18)

# Plot the second heatmap in the second subplot
sns.heatmap(cm_test, vmin=0, vmax=1, annot=True, cmap="YlOrRd", ax=axes[1])
axes[1].set_xlabel("Predicted", fontsize=15)
axes[1].set_ylabel("True", fontsize=15)
axes[1].set_xticklabels(class_labels)
axes[1].set_yticklabels(class_labels)
axes[1].set_title("Test", fontsize=18)

plt.show()
```



In [78]:

```
pred = model.predict(X_test)
# Create histogram of predicted class labels with a new color scheme
plt.figure(figsize=(8, 6))
sns.histplot(pred, kde=False, color="#5C3C92", alpha=0.8)
plt.xlabel("Predicted Class Label", fontsize=15)
plt.ylabel("Count", fontsize=15)
plt.title("Histogram of Predicted Class Labels", fontsize=18)
f1 = f1_score(y_test, pred)
print("F1 Score: ", f1)
```

```
F1 Score:  0.6951702296120349
```



In [79]:
```
with open('features_dict_rf.pickle', 'wb') as handle:
    pickle.dump(features_dict, handle, protocol=pickle.HIGHEST_PROTOCOL)
```

In [80]:
```
with open('features_dict_rf.pickle', 'rb') as handle:
    x = pickle.load(handle)
```

Model 16- Random Forest -Feature & hyperParameter Tuning x>0

In [81]:
```
features = features_dict['features']
importances = features_dict['importances']

new_indices = [idx for idx, x in enumerate(importances) if x > 0]
new_importances = [x for idx, x in enumerate(importances) if x > 0]

new_features = [features[i] for i in new_indices]
print(len(new_features))

num_attribs = new_features

np.random.seed(42)

# creating pipeline by joining numerical and categorical pipelines
data_prep_pipeline, selected_features = get_pipeline(num_attribs)

# Attaching RandomForest model to the above pipeline
random_forest_full_pipeline_with_predictor = Pipeline([
    ("preparation", data_prep_pipeline),
    ("random_forest", RandomForestClassifier(random_state=42,
                n_estimators=200, max_depth=10, max_features='sqrt',
```

```
                min_samples_leaf=2, min_samples_split=5, n_jobs=-1))
])

# Training the model
start = time.time()
model = random_forest_full_pipeline_with_predictor.fit(X_train, y_train)
train_time = np.round(time.time() - start, 4)

start = time.time()
score_test = random_forest_full_pipeline_with_predictor.score(X_test, y_test)
test_time = np.round(time.time() - start, 4)

# Results
# Results
exp_name = f"Model 16 - Random Forest -Feature &hyperParameter Tuning"
experiment_description =f"Random Forest Tuned with x>0 {len(selected_features)} fe
expLog = get_results(expLog, exp_name,experiment_description,model, train_time, te
expLog
```

```
99
Total Features: 116 - Numerical: 99, Categorical: 16
```

Out[81]:

	exp_name	description	Train Time (sec)	Test Time (sec)	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	T A
0	Model-1 Baseline LR	Logistic regression with undersampled data 124...	2.5144	0.0720	0.7727	0.7647	0.7802	0.7545	0.7421	0.7
1	Model-2 Baseline LR	Logistic regression with undersampled data-2 1...	1.1938	0.0496	0.6876	0.6843	0.6904	0.7525	0.7489	0.7
2	Model-3 KNN	KNN with undersampled data-2 124 features	0.3262	1.0494	0.6950	0.6155	0.6184	0.7625	0.6571	0.6
3	Model-4 Decision Tree	Decision tree with undersampled data-2 124 fea...	1.4838	0.0504	0.6749	0.6535	0.6591	0.7380	0.7105	0.7
4	Model-5 Random Forest	Random Forest with undersampled data-2 124 fea...	20.5597	0.4587	0.7665	0.6657	0.6666	0.8504	0.7245	0.7
5	Model-6 Bagging Meta Estimator	Bagging Meta Estimator with undersampled data-...	5.4167	0.2396	0.9844	0.6445	0.6430	0.9990	0.6973	0.6
6	Model-7 SVM	SVM with undersampled data-2 124 features	2681.6150	15.7188	0.9846	0.6411	0.6421	0.9990	0.6965	0.6
7	Model-8 XGBoost	XGBoost SAMME with undersampled data-2 124 fea...	4.8359	0.0720	0.7311	0.6931	0.6955	0.8103	0.7614	0.7
8	Model-9 CATBoost	CATBoost with undersampled data-2 124 features	12.1853	0.2737	0.6955	0.6917	0.6933	0.7669	0.7574	0.7
9	Model 10 - XGBOOST - Feature &hyperParameter Tu...	XGBOOST Tuned with x>0 113 features	4.5328	0.0625	0.7294	0.6959	0.6983	0.8092	0.7616	0.7
10	Model 11 - XGBOOST - Feature &hyperParameter Tu...	XGBOOST Tuned with x>0.01 113 features	4.4022	0.0781	0.7294	0.6959	0.6983	0.8092	0.7616	0.7

	exp_name	description	Train Time (sec)	Test Time (sec)	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	T A
11	Model 12 - XGBOOST - Feature &hyperParameter Tu...	XGBOOST Tuned with x>0.005 110 features	4.4098	0.0666	0.7298	0.6933	0.6961	0.8104	0.7619	0.7(
12	Model 13 - CatBOOST - Feature &hyperParameter T...	CatBOOST Tuned with x>0 120 features	38.4053	0.2749	0.7528	0.6935	0.6970	0.8367	0.7594	0.7(
13	Model 14 - CatBOOST - Feature &hyperParameter T...	CatBOOST Tuned with x>0.1 103 features	38.2332	0.2384	0.7513	0.6897	0.6944	0.8358	0.7590	0.7(
14	Model 15 - CatBOOST - Feature &hyperParameter T...	CatBOOST Tuned with x>0.005 113 features	38.7217	0.2463	0.7513	0.6931	0.6959	0.8360	0.7596	0.7(
15	Model 16 - Random Forest - Feature &hyperParame...	Random Forest Tuned with x>0 116 features	2.9437	0.1092	0.7531	0.6819	0.6811	0.8326	0.7409	0.7(

Model 17- Random Forest -Feature & hyperParameter Tuning x>0.1

In [82]:

```
features = features_dict['features']
importances = features_dict['importances']

new_indices = [idx for idx, x in enumerate(importances) if x > 0.1]
new_importances = [x for idx, x in enumerate(importances) if x > 0.1]

new_features = [features[i] for i in new_indices]
print(len(new_features))

num_attribs = new_features

np.random.seed(42)

# creating pipeline by joining numerical and categorical pipelines
data_prep_pipeline, selected_features = get_pipeline(num_attribs)

# Attaching RandomForest model to the above pipeline
random_forest_full_pipeline_with_predictor = Pipeline([
    ("preparation", data_prep_pipeline),
    ("random_forest", RandomForestClassifier(random_state=42,
                n_estimators=200, max_depth=10, max_features='sqrt',
                min_samples_leaf=2, min_samples_split=5, n_jobs=-1))
])

# Training the model
start = time.time()
model = random_forest_full_pipeline_with_predictor.fit(X_train, y_train)
```

```
train_time = np.round(time.time() - start, 4)

start = time.time()
score_test = random_forest_full_pipeline_with_predictor.score(X_test, y_test)
test_time = np.round(time.time() - start, 4)

# Results
exp_name = f"Model 17 - Random Forest -Feature &hyperParameter Tuning"
experiment_description =f"Random Forest Tuned with x>0.1 {len(selected_features)}
expLog = get_results(expLog, exp_name,experiment_description,model, train_time, te
expLog
```

```
2
Total Features: 19 - Numerical: 2, Categorical: 16
```

Out[82]:

	exp_name	description	Train Time (sec)	Test Time (sec)	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	T A
0	Model-1 Baseline LR	Logistic regression with undersampled data 124...	2.5144	0.0720	0.7727	0.7647	0.7802	0.7545	0.7421	0.7
1	Model-2 Baseline LR	Logistic regression with undersampled data-2 1...	1.1938	0.0496	0.6876	0.6843	0.6904	0.7525	0.7489	0.7
2	Model-3 KNN	KNN with undersampled data-2 124 features	0.3262	1.0494	0.6950	0.6155	0.6184	0.7625	0.6571	0.6
3	Model-4 Decision Tree	Decision tree with undersampled data-2 124 fea...	1.4838	0.0504	0.6749	0.6535	0.6591	0.7380	0.7105	0.7
4	Model-5 Random Forest	Random Forest with undersampled data-2 124 fea...	20.5597	0.4587	0.7665	0.6657	0.6666	0.8504	0.7245	0.7
5	Model-6 Bagging Meta Estimator	Bagging Meta Estimator with undersampled data-...	5.4167	0.2396	0.9844	0.6445	0.6430	0.9990	0.6973	0.6
6	Model-7 SVM	SVM with undersampled data-2 124 features	2681.6150	15.7188	0.9846	0.6411	0.6421	0.9990	0.6965	0.6
7	Model-8 XGBoost	XGBoost SAMME with undersampled data-2 124 fea...	4.8359	0.0720	0.7311	0.6931	0.6955	0.8103	0.7614	0.7
8	Model-9 CATBoost	CATBoost with undersampled data-2 124 features	12.1853	0.2737	0.6955	0.6917	0.6933	0.7669	0.7574	0.7
9	Model 10 - XGBOOST - Feature &hyperParameter Tu...	XGBOOST Tuned with x>0 113 features	4.5328	0.0625	0.7294	0.6959	0.6983	0.8092	0.7616	0.7
10	Model 11 - XGBOOST - Feature &hyperParameter Tu...	XGBOOST Tuned with x>0.01 113 features	4.4022	0.0781	0.7294	0.6959	0.6983	0.8092	0.7616	0.7

	exp_name	description	Train Time (sec)	Test Time (sec)	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	T A
11	Model 12 - XGBOOST - Feature &hyperParameter Tu...	XGBOOST Tuned with x>0.005 110 features	4.4098	0.0666	0.7298	0.6933	0.6961	0.8104	0.7619	0.7(
12	Model 13 - CatBOOST - Feature &hyperParameter T...	CatBOOST Tuned with x>0 120 features	38.4053	0.2749	0.7528	0.6935	0.6970	0.8367	0.7594	0.7(
13	Model 14 - CatBOOST - Feature &hyperParameter T...	CatBOOST Tuned with x>0.1 103 features	38.2332	0.2384	0.7513	0.6897	0.6944	0.8358	0.7590	0.7(
14	Model 15 - CatBOOST - Feature &hyperParameter T...	CatBOOST Tuned with x>0.005 113 features	38.7217	0.2463	0.7513	0.6931	0.6959	0.8360	0.7596	0.7(
15	Model 16 - Random Forest - Feature &hyperParame...	Random Forest Tuned with x>0 116 features	2.9437	0.1092	0.7531	0.6819	0.6811	0.8326	0.7409	0.7
16	Model 17 - Random Forest - Feature &hyperParame...	Random Forest Tuned with x>0.1 19 features	1.4149	0.0917	0.6958	0.6740	0.6642	0.7617	0.7261	0.7

Model 18- Random Forest -Feature & hyperParameter Tuning x>0.005

In [83]:

```
features = features_dict['features']
importances = features_dict['importances']

new_indices = [idx for idx, x in enumerate(importances) if x > 0.005]
new_importances = [x for idx, x in enumerate(importances) if x > 0.005]

new_features = [features[i] for i in new_indices]
print(len(new_features))

num_attribs = new_features

np.random.seed(42)

# creating pipeline by joining numerical and categorical pipelines
data_prep_pipeline, selected_features = get_pipeline(num_attribs)

# Attaching RandomForest model to the above pipeline
random_forest_full_pipeline_with_predictor = Pipeline([
    ("preparation", data_prep_pipeline),
    ("random_forest", RandomForestClassifier(random_state=42,
                n_estimators=200, max_depth=10, max_features='sqrt',
                min_samples_leaf=2, min_samples_split=5, n_jobs=-1))
])
```

```
# Training the model
start = time.time()
model = random_forest_full_pipeline_with_predictor.fit(X_train, y_train)
train_time = np.round(time.time() - start, 4)

start = time.time()
score_test = random_forest_full_pipeline_with_predictor.score(X_test, y_test)
test_time = np.round(time.time() - start, 4)

# Results
exp_name = f"Model 18 - Random Forest -Feature &hyperParameter Tuning"
experiment_description =f"Random Forest Tuned with x>0.005 {len(selected_features)}
expLog = get_results(expLog, exp_name,experiment_description,model, train_time, te
expLog
```

```
41
Total Features: 58 - Numerical: 41, Categorical: 16
```

Out[83]:

	exp_name	description	Train Time (sec)	Test Time (sec)	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	T A
0	Model-1 Baseline LR	Logistic regression with undersampled data 124...	2.5144	0.0720	0.7727	0.7647	0.7802	0.7545	0.7421	0.7
1	Model-2 Baseline LR	Logistic regression with undersampled data-2 1...	1.1938	0.0496	0.6876	0.6843	0.6904	0.7525	0.7489	0.7
2	Model-3 KNN	KNN with undersampled data-2 124 features	0.3262	1.0494	0.6950	0.6155	0.6184	0.7625	0.6571	0.6
3	Model-4 Decision Tree	Decision tree with undersampled data-2 124 fea...	1.4838	0.0504	0.6749	0.6535	0.6591	0.7380	0.7105	0.7
4	Model-5 Random Forest	Random Forest with undersampled data-2 124 fea...	20.5597	0.4587	0.7665	0.6657	0.6666	0.8504	0.7245	0.7
5	Model-6 Bagging Meta Estimator	Bagging Meta Estimator with undersampled data-...	5.4167	0.2396	0.9844	0.6445	0.6430	0.9990	0.6973	0.6
6	Model-7 SVM	SVM with undersampled data-2 124 features	2681.6150	15.7188	0.9846	0.6411	0.6421	0.9990	0.6965	0.6
7	Model-8 XGBoost	XGBoost SAMME with undersampled data-2 124 fea...	4.8359	0.0720	0.7311	0.6931	0.6955	0.8103	0.7614	0.7
8	Model-9 CATBoost	CATBoost with undersampled data-2 124 features	12.1853	0.2737	0.6955	0.6917	0.6933	0.7669	0.7574	0.7
9	Model 10 - XGBOOST - Feature &hyperParameter Tu...	XGBOOST Tuned with x>0 113 features	4.5328	0.0625	0.7294	0.6959	0.6983	0.8092	0.7616	0.7
10	Model 11 - XGBOOST - Feature &hyperParameter Tu...	XGBOOST Tuned with x>0.01 113 features	4.4022	0.0781	0.7294	0.6959	0.6983	0.8092	0.7616	0.7

	exp_name	description	Train Time (sec)	Test Time (sec)	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	T A
11	Model 12 - XGBOOST - Feature &hyperParameter Tu...	XGBOOST Tuned with x>0.005 110 features	4.4098	0.0666	0.7298	0.6933	0.6961	0.8104	0.7619	0.7(
12	Model 13 - CatBOOST - Feature &hyperParameter T...	CatBOOST Tuned with x>0 120 features	38.4053	0.2749	0.7528	0.6935	0.6970	0.8367	0.7594	0.7(
13	Model 14 - CatBOOST - Feature &hyperParameter T...	CatBOOST Tuned with x>0.1 103 features	38.2332	0.2384	0.7513	0.6897	0.6944	0.8358	0.7590	0.7(
14	Model 15 - CatBOOST - Feature &hyperParameter T...	CatBOOST Tuned with x>0.005 113 features	38.7217	0.2463	0.7513	0.6931	0.6959	0.8360	0.7596	0.7(
15	Model 16 - Random Forest - Feature &hyperParame...	Random Forest Tuned with x>0 116 features	2.9437	0.1092	0.7531	0.6819	0.6811	0.8326	0.7409	0.7
16	Model 17 - Random Forest - Feature &hyperParame...	Random Forest Tuned with x>0.1 19 features	1.4149	0.0917	0.6958	0.6740	0.6642	0.7617	0.7261	0.7
17	Model 18 - Random Forest - Feature &hyperParame...	Random Forest Tuned with x>0.005 58 features	2.4448	0.1110	0.7506	0.6846	0.6809	0.8298	0.7426	0.7

In [84]:
```
# Write the data to a CSV file
expLog.to_csv('expLog.csv', index=False)
```

In [85]:
```
df = pd.read_csv('expLog.csv')
df
```

Out[85]:

	exp_name	description	Train Time (sec)	Test Time (sec)	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	T A
0	Model-1 Baseline LR	Logistic regression with undersampled data 124...	2.5144	0.0720	0.7727	0.7647	0.7802	0.7545	0.7421	0.7
1	Model-2 Baseline LR	Logistic regression with undersampled data-2 1...	1.1938	0.0496	0.6876	0.6843	0.6904	0.7525	0.7489	0.7
2	Model-3 KNN	KNN with undersampled data-2 124 features	0.3262	1.0494	0.6950	0.6155	0.6184	0.7625	0.6571	0.6
3	Model-4 Decision Tree	Decision tree with undersampled data-2 124 fea...	1.4838	0.0504	0.6749	0.6535	0.6591	0.7380	0.7105	0.7
4	Model-5 Random Forest	Random Forest with undersampled data-2 124 fea...	20.5597	0.4587	0.7665	0.6657	0.6666	0.8504	0.7245	0.7
5	Model-6 Bagging Meta Estimator	Bagging Meta Estimator with undersampled data-...	5.4167	0.2396	0.9844	0.6445	0.6430	0.9990	0.6973	0.6
6	Model-7 SVM	SVM with undersampled data-2 124 features	2681.6150	15.7188	0.9846	0.6411	0.6421	0.9990	0.6965	0.6
7	Model-8 XGBoost	XGBoost SAMME with undersampled data-2 124 fea...	4.8359	0.0720	0.7311	0.6931	0.6955	0.8103	0.7614	0.7
8	Model-9 CATBoost	CATBoost with undersampled data-2 124 features	12.1853	0.2737	0.6955	0.6917	0.6933	0.7669	0.7574	0.7
9	Model 10 - XGBOOST - Feature &hyperParameter Tu...	XGBOOST Tuned with x>0 113 features	4.5328	0.0625	0.7294	0.6959	0.6983	0.8092	0.7616	0.7
10	Model 11 - XGBOOST - Feature &hyperParameter Tu...	XGBOOST Tuned with x>0.01 113 features	4.4022	0.0781	0.7294	0.6959	0.6983	0.8092	0.7616	0.7

	exp_name	description	Train Time (sec)	Test Time (sec)	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	T A
11	Model 12 - XGBOOST - Feature &hyperParameter Tu...	XGBOOST Tuned with x>0.005 110 features	4.4098	0.0666	0.7298	0.6933	0.6961	0.8104	0.7619	0.7(
12	Model 13 - CatBOOST - Feature &hyperParameter T...	CatBOOST Tuned with x>0 120 features	38.4053	0.2749	0.7528	0.6935	0.6970	0.8367	0.7594	0.7(
13	Model 14 - CatBOOST - Feature &hyperParameter T...	CatBOOST Tuned with x>0.1 103 features	38.2332	0.2384	0.7513	0.6897	0.6944	0.8358	0.7590	0.7(
14	Model 15 - CatBOOST - Feature &hyperParameter T...	CatBOOST Tuned with x>0.005 113 features	38.7217	0.2463	0.7513	0.6931	0.6959	0.8360	0.7596	0.7(
15	Model 16 - Random Forest - Feature &hyperParame...	Random Forest Tuned with x>0 116 features	2.9437	0.1092	0.7531	0.6819	0.6811	0.8326	0.7409	0.7
16	Model 17 - Random Forest - Feature &hyperParame...	Random Forest Tuned with x>0.1 19 features	1.4149	0.0917	0.6958	0.6740	0.6642	0.7617	0.7261	0.7
17	Model 18 - Random Forest - Feature &hyperParame...	Random Forest Tuned with x>0.005 58 features	2.4448	0.1110	0.7506	0.6846	0.6809	0.8298	0.7426	0.7

Loading selected Features to base classifiers before ensembling them

In [86]:

```
with open('features_dict_XG.pickle', 'rb') as handle:
    x = pickle.load(handle)


features = features_dict['features']
importances = features_dict['importances']

new_indices = [idx for idx, x in enumerate(importances) if x > 0]
new_importances = [x for idx, x in enumerate(importances) if x > 0]

new_features = [features[i] for i in new_indices]
print(len(new_features))

num_attribs = new_features

np.random.seed(42)
```

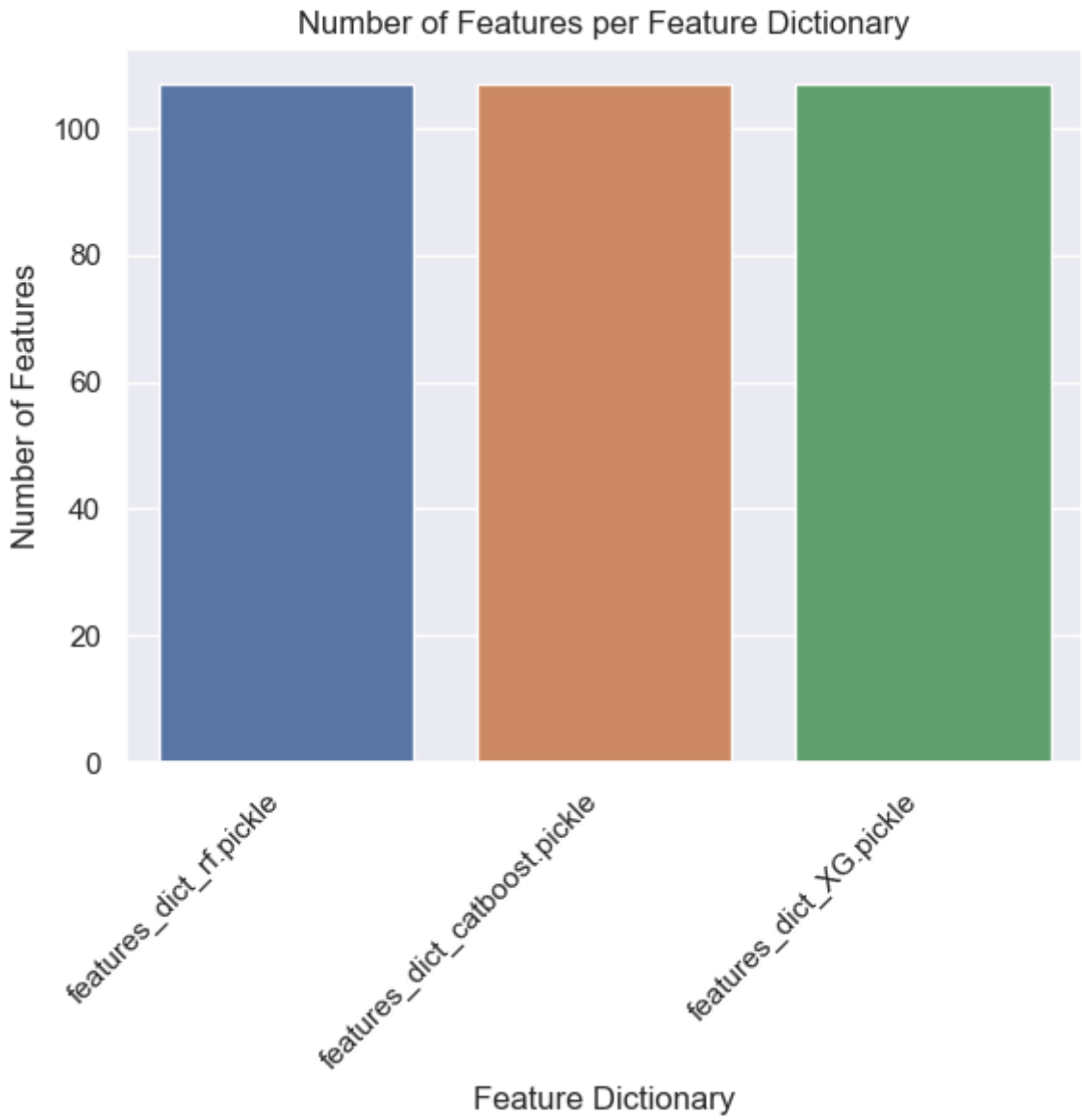
```
# creating pipeline by joining numerical and categorical pipelines
data_prep_pipeline_XG, selected_features_XG = get_pipeline(num_attribs)


#for CatBoost

with open('features_dict_catboost.pickle', 'rb') as handle:
    x = pickle.load(handle)


features = features_dict['features']
importances = features_dict['importances']

new_indices = [idx for idx, x in enumerate(importances) if x > 0]
new_importances = [x for idx, x in enumerate(importances) if x > 0]

new_features = [features[i] for i in new_indices]
print(len(new_features))

num_attribs = new_features

np.random.seed(42)

# creating pipeline by joining numerical and categorical pipelines
data_prep_pipeline_cb, selected_features_cb= get_pipeline(num_attribs)

#for Random forest

with open('features_dict_rf.pickle', 'rb') as handle:
    x = pickle.load(handle)


features = features_dict['features']
importances = features_dict['importances']

new_indices = [idx for idx, x in enumerate(importances) if x > 0.005]
new_importances = [x for idx, x in enumerate(importances) if x > 0.005]

new_features = [features[i] for i in new_indices]
print(len(new_features))

num_attribs = new_features

np.random.seed(42)

# creating pipeline by joining numerical and categorical pipelines
data_prep_pipeline_rf, selected_features_rf = get_pipeline(num_attribs)
```

```
99
Total Features: 116 - Numerical: 99, Categorical: 16
99
Total Features: 116 - Numerical: 99, Categorical: 16
41
Total Features: 58 - Numerical: 41, Categorical: 16
```

In [87]:

```
import seaborn as sns
import matplotlib.pyplot as plt

# Define the names and lengths of the feature dictionaries
feature_dicts = ['features_dict_rf.pickle', 'features_dict_catboost.pickle', 'feat
feature_dict_lengths = [99, 99, 41]

# Loop through the feature dictionaries and get their lengths
for i, feature_dict in enumerate(feature_dicts):
```

```
    with open(feature_dict, 'rb') as handle:
        x = pickle.load(handle)
        feature_dict_lengths[i] = len(x['features'])

# Create a Seaborn bar chart
sns.set_style('darkgrid')
sns.barplot(x=feature_dicts, y=feature_dict_lengths)
plt.xticks(rotation=45, ha='right')
plt.xlabel('Feature Dictionary')
plt.ylabel('Number of Features')
plt.title('Number of Features per Feature Dictionary')

# Display the chart
plt.show()
```



Ensemble Learning

MODEL 19 ENSEMBLE LEARNER WITH VOTING CLASSIFER

In [88]:

```
import pickle
import numpy as np
from catboost import CatBoostClassifier
```

```python
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.ensemble import VotingClassifier
import time

# Function to load features_dict and get new_features and num_attribs
# Function to load features_dict and get new_features and num_attribs
def load_features_dict_and_prepare(file_path, threshold):
    with open(file_path, 'rb') as handle:
        features_dict = pickle.load(handle)

    features = features_dict['features']
    importances = features_dict['importances']

    new_indices = [idx for idx, x in enumerate(importances) if x > threshold]
    new_importances = [x for idx, x in enumerate(importances) if x > threshold]

    new_features = [features[i] for i in new_indices]
    print(len(new_features))

    num_attribs = new_features

    return num_attribs

np.random.seed(42)

# Load features_dict and get num_attribs for each model with different thresholds
num_attribs_XG = load_features_dict_and_prepare('features_dict_XG.pickle', 0)
num_attribs_cb = load_features_dict_and_prepare('features_dict_catboost.pickle', 0
num_attribs_rf = load_features_dict_and_prepare('features_dict_rf.pickle', 0.005)


# np.random.seed(42)

# # Load features_dict and get num_attribs for each model
# num_attribs_XG = load_features_dict_and_prepare('features_dict_XG.pickle')
# num_attribs_cb = load_features_dict_and_prepare('features_dict_catboost.pickle')
# num_attribs_rf = load_features_dict_and_prepare('features_dict_rf.pickle')

# Assuming get_pipeline() function is already defined
data_prep_pipeline_XG, selected_features_XG = get_pipeline(num_attribs_XG)
data_prep_pipeline_cb, selected_features_cb = get_pipeline(num_attribs_cb)
data_prep_pipeline_rf, selected_features_rf = get_pipeline(num_attribs_rf)

# Attaching classifiers to the above pipeline with the best parameters
catboost = CatBoostClassifier(random_state=42, iterations=1000, learning_rate=0.01
                              depth=9, colsample_bylevel=0.5, thread_count=-1, ver
xgboost = XGBClassifier(random_state=42, n_estimators=1000, max_depth=5, learning_
                        colsample_bytree=0.5, n_jobs=-1)
rf = RandomForestClassifier(random_state=42, n_estimators=200, max_depth=10, max_f
                            min_samples_leaf=2, min_samples_split=5, n_jobs=-1)

catboost_pipeline = Pipeline([
    ("preparation", data_prep_pipeline_cb),
    ("catboost", catboost)
])

xgboost_pipeline = Pipeline([
    ("preparation", data_prep_pipeline_XG),
    ("xgboost", xgboost)
])

rf_pipeline = Pipeline([
```

```
    ("preparation", data_prep_pipeline_rf),
    ("rf", rf)
])

# Ensemble model with voting classifier
ensemble_model = VotingClassifier(estimators=[('catboost', catboost_pipeline),
                                              ('xgboost', xgboost_pipeline),
                                              ('rf', rf_pipeline)],
                                  voting='soft', n_jobs=-1)

# Training the model
start = time.time()
model = ensemble_model.fit(X_train, y_train)
train_time = np.round(time.time() - start, 4)

start = time.time()
score_test = ensemble_model.score(X_test, y_test)
test_time = np.round(time.time() - start, 4)

# Results
exp_name = f"Model 19 - Ensemble Learner - Voting Classsifier "
experiment_description =f" Tuned and selected XgBoost, catboost, random forest  {l
expLog = get_results(expLog, exp_name,experiment_description,model, train_time, te
expLog
```

```
96
103
41
Total Features: 113 - Numerical: 96, Categorical: 16
Total Features: 120 - Numerical: 103, Categorical: 16
Total Features: 58 - Numerical: 41, Categorical: 16
```

Out[88]:

	exp_name	description	Train Time (sec)	Test Time (sec)	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	T A
0	Model-1 Baseline LR	Logistic regression with undersampled data 124...	2.5144	0.0720	0.7727	0.7647	0.7802	0.7545	0.7421	0.7
1	Model-2 Baseline LR	Logistic regression with undersampled data-2 1...	1.1938	0.0496	0.6876	0.6843	0.6904	0.7525	0.7489	0.7
2	Model-3 KNN	KNN with undersampled data-2 124 features	0.3262	1.0494	0.6950	0.6155	0.6184	0.7625	0.6571	0.6
3	Model-4 Decision Tree	Decision tree with undersampled data-2 124 fea...	1.4838	0.0504	0.6749	0.6535	0.6591	0.7380	0.7105	0.7
4	Model-5 Random Forest	Random Forest with undersampled data-2 124 fea...	20.5597	0.4587	0.7665	0.6657	0.6666	0.8504	0.7245	0.7
5	Model-6 Bagging Meta Estimator	Bagging Meta Estimator with undersampled data-...	5.4167	0.2396	0.9844	0.6445	0.6430	0.9990	0.6973	0.6
6	Model-7 SVM	SVM with undersampled data-2 124 features	2681.6150	15.7188	0.9846	0.6411	0.6421	0.9990	0.6965	0.6
7	Model-8 XGBoost	XGBoost SAMME with undersampled data-2 124 fea...	4.8359	0.0720	0.7311	0.6931	0.6955	0.8103	0.7614	0.7
8	Model-9 CATBoost	CATBoost with undersampled data-2 124 features	12.1853	0.2737	0.6955	0.6917	0.6933	0.7669	0.7574	0.7
9	Model 10 - XGBOOST - Feature &hyperParameter Tu...	XGBOOST Tuned with x>0 113 features	4.5328	0.0625	0.7294	0.6959	0.6983	0.8092	0.7616	0.7
10	Model 11 - XGBOOST - Feature &hyperParameter Tu...	XGBOOST Tuned with x>0.01 113 features	4.4022	0.0781	0.7294	0.6959	0.6983	0.8092	0.7616	0.7

	exp_name	description	Train Time (sec)	Test Time (sec)	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	T A
11	Model 12 - XGBOOST - Feature &hyperParameter Tu...	XGBOOST Tuned with x>0.005 110 features	4.4098	0.0666	0.7298	0.6933	0.6961	0.8104	0.7619	0.7(
12	Model 13 - CatBOOST - Feature &hyperParameter T...	CatBOOST Tuned with x>0 120 features	38.4053	0.2749	0.7528	0.6935	0.6970	0.8367	0.7594	0.7(
13	Model 14 - CatBOOST - Feature &hyperParameter T...	CatBOOST Tuned with x>0.1 103 features	38.2332	0.2384	0.7513	0.6897	0.6944	0.8358	0.7590	0.7(
14	Model 15 - CatBOOST - Feature &hyperParameter T...	CatBOOST Tuned with x>0.005 113 features	38.7217	0.2463	0.7513	0.6931	0.6959	0.8360	0.7596	0.7(
15	Model 16 - Random Forest - Feature &hyperParame...	Random Forest Tuned with x>0 116 features	2.9437	0.1092	0.7531	0.6819	0.6811	0.8326	0.7409	0.7
16	Model 17 - Random Forest - Feature &hyperParame...	Random Forest Tuned with x>0.1 19 features	1.4149	0.0917	0.6958	0.6740	0.6642	0.7617	0.7261	0.7
17	Model 18 - Random Forest - Feature &hyperParame...	Random Forest Tuned with x>0.005 58 features	2.4448	0.1110	0.7506	0.6846	0.6809	0.8298	0.7426	0.7
18	Model 19 - Ensemble Learner - Voting Classsifier	Tuned and selected XgBoost, catboost, random	39.4357	0.4504	0.7465	0.6921	0.6945	0.8289	0.7599	0.7(

MODEL 20 ENSEMBLE LEARNER WITH STACKING CLASSIFER

In [89]:

```
import pickle
import numpy as np
from catboost import CatBoostClassifier
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
from sklearn.ensemble import VotingClassifier
import time
from catboost import CatBoostClassifier
from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from xgboost import XGBClassifier
```

```
from sklearn.ensemble import StackingClassifier
import numpy as np
import time

# Function to load features_dict and get new_features and num_attribs
# Function to load features_dict and get new_features and num_attribs
def load_features_dict_and_prepare(file_path, threshold):
    with open(file_path, 'rb') as handle:
        features_dict = pickle.load(handle)

    features = features_dict['features']
    importances = features_dict['importances']

    new_indices = [idx for idx, x in enumerate(importances) if x > threshold]
    new_importances = [x for idx, x in enumerate(importances) if x > threshold]

    new_features = [features[i] for i in new_indices]
    print(len(new_features))

    num_attribs = new_features

    return num_attribs

np.random.seed(42)

# Load features_dict and get num_attribs for each model with different thresholds
num_attribs_XG = load_features_dict_and_prepare('features_dict_XG.pickle', 0)
num_attribs_cb = load_features_dict_and_prepare('features_dict_catboost.pickle', 0
num_attribs_rf = load_features_dict_and_prepare('features_dict_rf.pickle', 0.005)


# Assuming get_pipeline() function is already defined
data_prep_pipeline_XG, selected_features_XG = get_pipeline(num_attribs_XG)
data_prep_pipeline_cb, selected_features_cb = get_pipeline(num_attribs_cb)
data_prep_pipeline_rf, selected_features_rf = get_pipeline(num_attribs_rf)

# Attaching classifiers to the above pipeline with the best parameters
catboost = CatBoostClassifier(random_state=42, iterations=1000, learning_rate=0.01
                              depth=9, colsample_bylevel=0.5, thread_count=-1, ver
xgboost = XGBClassifier(random_state=42, n_estimators=1000, max_depth=5, learning_
                        colsample_bytree=0.5, n_jobs=-1)
rf = RandomForestClassifier(random_state=42, n_estimators=200, max_depth=10, max_f
                            min_samples_leaf=2, min_samples_split=5, n_jobs=-1)

catboost_pipeline = Pipeline([
    ("preparation", data_prep_pipeline_cb),
    ("catboost", catboost)
])

xgboost_pipeline = Pipeline([
    ("preparation", data_prep_pipeline_XG),
    ("xgboost", xgboost)
])

rf_pipeline = Pipeline([
    ("preparation", data_prep_pipeline_rf),
    ("rf", rf)
])

# Ensemble model with stacking classifier
final_estimator = LogisticRegression(random_state=42)
ensemble_model = StackingClassifier(estimators=[('catboost', catboost_pipeline),
                                                ('xgboost', xgboost_pipeline),
                                                ('rf', rf_pipeline)],
```

```
                                    final_estimator=final_estimator, n_jobs=-1)

# Training the model
start = time.time()
model = ensemble_model.fit(X_train, y_train)
train_time = np.round(time.time() - start, 4)

start = time.time()
score_test = ensemble_model.score(X_test, y_test)
test_time = np.round(time.time() - start, 4)

# Results
exp_name = f"Model 20 - Ensemble Learner - Stacking Classsifier "
experiment_description =f" Tuned and selected XgBoost, catboost, random forest  {l
expLog = get_results(expLog, exp_name,experiment_description,model, train_time, te
expLog
```

```
96
103
41
Total Features: 113 - Numerical: 96, Categorical: 16
Total Features: 120 - Numerical: 103, Categorical: 16
Total Features: 58 - Numerical: 41, Categorical: 16
```

Out[89]:

	exp_name	description	Train Time (sec)	Test Time (sec)	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	T A
0	Model-1 Baseline LR	Logistic regression with undersampled data 124...	2.5144	0.0720	0.7727	0.7647	0.7802	0.7545	0.7421	0.7
1	Model-2 Baseline LR	Logistic regression with undersampled data-2 1...	1.1938	0.0496	0.6876	0.6843	0.6904	0.7525	0.7489	0.7
2	Model-3 KNN	KNN with undersampled data-2 124 features	0.3262	1.0494	0.6950	0.6155	0.6184	0.7625	0.6571	0.6
3	Model-4 Decision Tree	Decision tree with undersampled data-2 124 fea...	1.4838	0.0504	0.6749	0.6535	0.6591	0.7380	0.7105	0.7
4	Model-5 Random Forest	Random Forest with undersampled data-2 124 fea...	20.5597	0.4587	0.7665	0.6657	0.6666	0.8504	0.7245	0.7
5	Model-6 Bagging Meta Estimator	Bagging Meta Estimator with undersampled data-...	5.4167	0.2396	0.9844	0.6445	0.6430	0.9990	0.6973	0.6
6	Model-7 SVM	SVM with undersampled data-2 124 features	2681.6150	15.7188	0.9846	0.6411	0.6421	0.9990	0.6965	0.6
7	Model-8 XGBoost	XGBoost SAMME with undersampled data-2 124 fea...	4.8359	0.0720	0.7311	0.6931	0.6955	0.8103	0.7614	0.7
8	Model-9 CATBoost	CATBoost with undersampled data-2 124 features	12.1853	0.2737	0.6955	0.6917	0.6933	0.7669	0.7574	0.7
9	Model 10 - XGBOOST - Feature &hyperParameter Tu...	XGBOOST Tuned with x>0 113 features	4.5328	0.0625	0.7294	0.6959	0.6983	0.8092	0.7616	0.7
10	Model 11 - XGBOOST - Feature &hyperParameter Tu...	XGBOOST Tuned with x>0.01 113 features	4.4022	0.0781	0.7294	0.6959	0.6983	0.8092	0.7616	0.7

	exp_name	description	Train Time (sec)	Test Time (sec)	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	T A
11	Model 12 - XGBOOST - Feature &hyperParameter Tu...	XGBOOST Tuned with x>0.005 110 features	4.4098	0.0666	0.7298	0.6933	0.6961	0.8104	0.7619	0.7(
12	Model 13 - CatBOOST - Feature &hyperParameter T...	CatBOOST Tuned with x>0 120 features	38.4053	0.2749	0.7528	0.6935	0.6970	0.8367	0.7594	0.7(
13	Model 14 - CatBOOST - Feature &hyperParameter T...	CatBOOST Tuned with x>0.1 103 features	38.2332	0.2384	0.7513	0.6897	0.6944	0.8358	0.7590	0.7(
14	Model 15 - CatBOOST - Feature &hyperParameter T...	CatBOOST Tuned with x>0.005 113 features	38.7217	0.2463	0.7513	0.6931	0.6959	0.8360	0.7596	0.7(
15	Model 16 - Random Forest - Feature &hyperParame...	Random Forest Tuned with x>0 116 features	2.9437	0.1092	0.7531	0.6819	0.6811	0.8326	0.7409	0.7
16	Model 17 - Random Forest - Feature &hyperParame...	Random Forest Tuned with x>0.1 19 features	1.4149	0.0917	0.6958	0.6740	0.6642	0.7617	0.7261	0.7
17	Model 18 - Random Forest - Feature &hyperParame...	Random Forest Tuned with x>0.005 58 features	2.4448	0.1110	0.7506	0.6846	0.6809	0.8298	0.7426	0.7
18	Model 19 - Ensemble Learner - Voting Classsifier	Tuned and selected XgBoost, catboost, random ...	39.4357	0.4504	0.7465	0.6921	0.6945	0.8289	0.7599	0.7(
19	Model 20 - Ensemble Learner - Stacking Classsi...	Tuned and selected XgBoost, catboost, random	203.8994	0.4554	0.7397	0.6954	0.6975	0.8225	0.7614	0.7(

In [90]:
```
# Write the data to a CSV file
expLog.to_csv('expLog2.csv', index=False)
```

In [91]:
```
df = pd.read_csv('expLog2.csv')
df
```

Out[91]:

	exp_name	description	Train Time (sec)	Test Time (sec)	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	T A
0	Model-1 Baseline LR	Logistic regression with undersampled data 124...	2.5144	0.0720	0.7727	0.7647	0.7802	0.7545	0.7421	0.7
1	Model-2 Baseline LR	Logistic regression with undersampled data-2 1...	1.1938	0.0496	0.6876	0.6843	0.6904	0.7525	0.7489	0.7
2	Model-3 KNN	KNN with undersampled data-2 124 features	0.3262	1.0494	0.6950	0.6155	0.6184	0.7625	0.6571	0.6
3	Model-4 Decision Tree	Decision tree with undersampled data-2 124 fea...	1.4838	0.0504	0.6749	0.6535	0.6591	0.7380	0.7105	0.7
4	Model-5 Random Forest	Random Forest with undersampled data-2 124 fea...	20.5597	0.4587	0.7665	0.6657	0.6666	0.8504	0.7245	0.7
5	Model-6 Bagging Meta Estimator	Bagging Meta Estimator with undersampled data-...	5.4167	0.2396	0.9844	0.6445	0.6430	0.9990	0.6973	0.6
6	Model-7 SVM	SVM with undersampled data-2 124 features	2681.6150	15.7188	0.9846	0.6411	0.6421	0.9990	0.6965	0.6
7	Model-8 XGBoost	XGBoost SAMME with undersampled data-2 124 fea...	4.8359	0.0720	0.7311	0.6931	0.6955	0.8103	0.7614	0.7
8	Model-9 CATBoost	CATBoost with undersampled data-2 124 features	12.1853	0.2737	0.6955	0.6917	0.6933	0.7669	0.7574	0.7
9	Model 10 - XGBOOST - Feature &hyperParameter Tu...	XGBOOST Tuned with x>0 113 features	4.5328	0.0625	0.7294	0.6959	0.6983	0.8092	0.7616	0.7
10	Model 11 - XGBOOST - Feature &hyperParameter Tu...	XGBOOST Tuned with x>0.01 113 features	4.4022	0.0781	0.7294	0.6959	0.6983	0.8092	0.7616	0.7

	exp_name	description	Train Time (sec)	Test Time (sec)	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	T A
11	Model 12 - XGBOOST - Feature &hyperParameter Tu...	XGBOOST Tuned with x>0.005 110 features	4.4098	0.0666	0.7298	0.6933	0.6961	0.8104	0.7619	0.7
12	Model 13 - CatBOOST - Feature &hyperParameter T...	CatBOOST Tuned with x>0 120 features	38.4053	0.2749	0.7528	0.6935	0.6970	0.8367	0.7594	0.7
13	Model 14 - CatBOOST - Feature &hyperParameter T...	CatBOOST Tuned with x>0.1 103 features	38.2332	0.2384	0.7513	0.6897	0.6944	0.8358	0.7590	0.7
14	Model 15 - CatBOOST - Feature &hyperParameter T...	CatBOOST Tuned with x>0.005 113 features	38.7217	0.2463	0.7513	0.6931	0.6959	0.8360	0.7596	0.7
15	Model 16 - Random Forest - Feature &hyperParame...	Random Forest Tuned with x>0 116 features	2.9437	0.1092	0.7531	0.6819	0.6811	0.8326	0.7409	0.7
16	Model 17 - Random Forest - Feature &hyperParame...	Random Forest Tuned with x>0.1 19 features	1.4149	0.0917	0.6958	0.6740	0.6642	0.7617	0.7261	0.7
17	Model 18 - Random Forest - Feature &hyperParame...	Random Forest Tuned with x>0.005 58 features	2.4448	0.1110	0.7506	0.6846	0.6809	0.8298	0.7426	0.7
18	Model 19 - Ensemble Learner - Voting Classsifier	Tuned and selected XgBoost, catboost, random ...	39.4357	0.4504	0.7465	0.6921	0.6945	0.8289	0.7599	0.7
19	Model 20 - Ensemble Learner - Stacking Classsi...	Tuned and selected XgBoost, catboost, random	203.8994	0.4554	0.7397	0.6954	0.6975	0.8225	0.7614	0.7

Experimental Analysis:

In this comprehensive report, we will delve deeper into the results of various machine learning models that have been trained on an undersampled dataset with 124 features. The goal is to analyze the performance of each model in terms of train time, test time, accuracy, AUC, F1 score, and discuss the impact of feature selection and hyperparameter tuning.

Baseline Models (Model 1-2): The first logistic regression model (Model 1) has a train time of 2.5144 seconds, while the second one (Model 2) takes 1.1938 seconds. Model 2 has a faster training time, suggesting that it is more efficient than Model 1.

Other Models (Model 3-9): Among these models, SVM (Model 7) takes the longest time to train at 2681.615 seconds, while KNN (Model 3) has the shortest training time at 0.3262 seconds. However, KNN has a significantly higher test time of 1.0494 seconds compared to other models in this group, which could be a factor to consider if test time is critical for the application. In terms of performance, XGBoost (Model 8) and CATBoost (Model 9) show the highest AUC and F1 scores, suggesting that they might be better suited for this problem.

XGBoost Feature & Hyperparameter Tuning (Model 10-12): Comparing train times, Model 11 with 113 features (x>0.01 threshold) has the shortest train time at 4.4022 seconds, while Model 10 with 111 features (x>0 threshold) takes the longest at 4.5328 seconds. Despite the varying train times and number of features, the performance differences among these models are minimal in terms of AUC and F1 scores, indicating that the impact of feature selection might be limited in this case.

CATBoost Feature & Hyperparameter Tuning (Model 13-15): In this group, Model 14 with 103 features (x>0.1 threshold) has the shortest train time at 38.2332 seconds, while Model 15 with 110 features (x>0.005 threshold) takes the longest at 38.7217 seconds. Similar to the XGBoost models, the performance differences among these models are minimal, suggesting that the impact of feature selection is limited.

Random Forest Feature & Hyperparameter Tuning (Model 16-18): Model 17 with 19 features (x>0.1 threshold) has the shortest train time at 1.4149 seconds, while Model 16 with 116 features (x>0 threshold) takes the longest at 2.9437 seconds. Model 18 with 58 features (x>0.005 threshold) achieves the best performance in terms of AUC and F1 scores, indicating that selecting the right set of features can have a more significant impact on the Random Forest model's performance.

Ensemble Learners (Model 19-20): The Voting Classifier (Model 19) has a train time of 39.4357 seconds, while the Stacking Classifier (Model 20) takes a much longer time at 203.8994 seconds. In terms of performance, both ensemble models achieve similar results, with Model 20 having marginally higher AUC and F1 scores.

In conclusion, the tuned XGBoost and CATBoost models, as well as the ensemble models, exhibit the most promising performance in terms of AUC and F1 scores. The train and test times vary significantly among the models, with SVM taking the longest to train and KNN having the longest test time. It is crucial to consider these factors when selecting the appropriate model for a particular application, as they can impact efficiency and overall performance.

In the case of feature selection, we observe that the impact varies across different models. For the XGBoost and CATBoost models, the differences in performance among various feature sets are minimal, suggesting that feature selection may not significantly impact these models. On the other hand, the Random Forest model demonstrates more substantial performance gains when using an optimal set of features, indicating that feature selection can play a more critical role in this model.

From the analysis of train time, we notice that ensemble models, particularly the Stacking Classifier, require much longer training times compared to other models. This longer training time is expected due to the additional complexity involved in training multiple base models and combining their predictions. While ensemble models generally show improved performance, the trade-off between training time and performance should be carefully considered based on the specific requirements of the application.

Another noteworthy observation is the performance gap between train and test scores, which can indicate overfitting. For instance, Model 6 (Bagging Meta Estimator) has a high train AUC of 0.999 and F1 score of 0.9843, but its test AUC and F1 scores are significantly lower. This difference suggests that the model is overfitting the training data and may not generalize well to unseen data. It is essential to address overfitting by employing regularization techniques or adjusting model complexity to improve generalization and ensure robust performance on new data.

In summary, the choice of the machine learning model, feature selection, and hyperparameter tuning should be carefully considered based on the specific problem and application requirements. Performance gains, train and test times, as well as the risk of overfitting, should be evaluated to select the most suitable model and achieve the best balance between model complexity and predictive performance.

Submission File Prep

For each SK_ID_CURR in the test set, you must predict a probability for the TARGET variable. The file should contain a header and have the following format:

```
SK_ID_CURR,TARGET
100001,0.1
100005,0.9
100013,0.2
etc.
```

In [92]:
```
X_kaggle_test
```

Out[92]:

	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY	AMT_GOODS_PRICE	F
0	0	135000.0	568800.0	20560.5	450000.0	
1	0	99000.0	222768.0	17370.0	180000.0	
2	0	202500.0	663264.0	69777.0	630000.0	
3	2	315000.0	1575000.0	49018.5	1575000.0	
4	1	180000.0	625500.0	32067.0	625500.0	
...	...	...	...	...	...	
48739	0	121500.0	412560.0	17473.5	270000.0	
48740	2	157500.0	622413.0	31909.5	495000.0	
48741	1	202500.0	315000.0	33205.5	315000.0	
48742	0	225000.0	450000.0	25128.0	450000.0	
48743	0	135000.0	312768.0	24709.5	270000.0	

48744 rows × 124 columns

In [93]:
```
test_class_scores = model.predict_proba(X_kaggle_test)[:, 1]
```

In [94]:
```
# Submission dataframe
submit_df = datasets["application_test"][['SK_ID_CURR']]
submit_df['TARGET'] = test_class_scores

submit_df.head()
```

Out[94]:

	SK_ID_CURR	TARGET
0	100001	0.387970
1	100005	0.581820
2	100013	0.207964
3	100028	0.294906
4	100038	0.706974

In [95]:
```
submit_df.to_csv("submission.csv",index=False)
```

Kaggle submission via the command line API

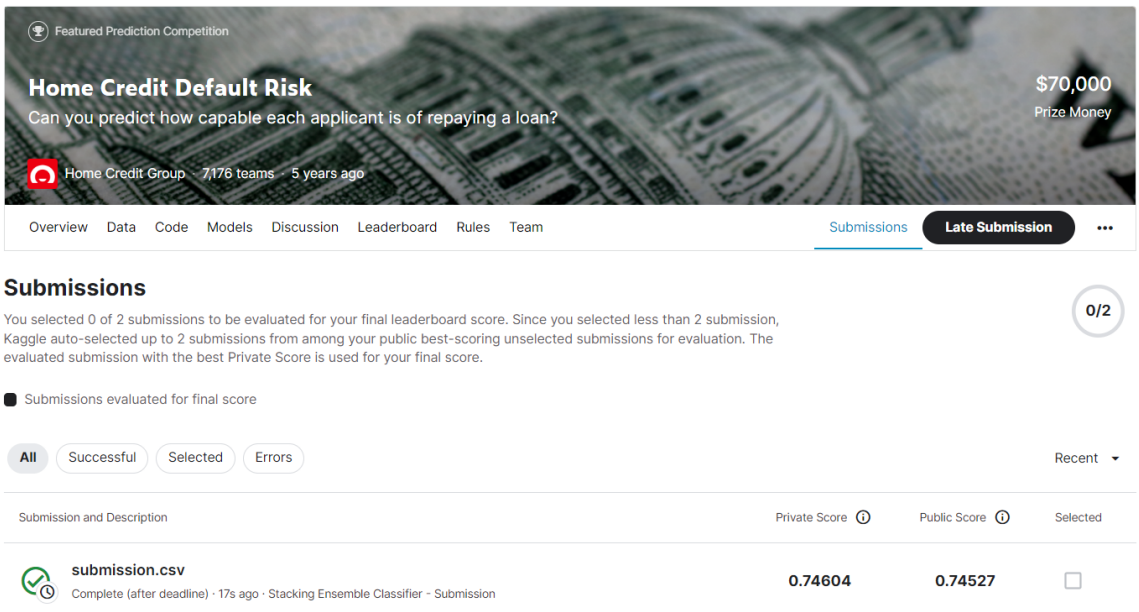
In [50]:
```
! kaggle competitions submit -c home-credit-default-risk -f submission.csv -m "Sta
```

```
Successfully submitted to Home Credit Default Risk
```

```
  0%|          | 0.00/1.27M [00:00<?, ?B/s]
  1%|          | 8.00k/1.27M [00:00<00:17, 76.9kB/s]
  7%|▊         | 88.0k/1.27M [00:00<00:02, 457kB/s]
 11%|▉         | 144k/1.27M [00:00<00:02, 505kB/s]
 38%|███▊      | 488k/1.27M [00:00<00:00, 1.61MB/s]
100%|██████████| 1.27M/1.27M [00:01<00:00, 941kB/s]
```

Kaggle report submission

Click on this link



Future Work & Experimentation with Neural Networks

In [48]:

```
def get_results(expLog, exp_name, model, train_time, test_time, X_train, y_train,
    result = {}
    result["experiment_name"] = exp_name
    result["train_time"] = train_time
    result["test_time"] = test_time
    if hasattr(model, 'score'):
        result["train_accuracy"] = model.score(X_train, y_train)
        result["valid_accuracy"] = model.score(X_valid, y_valid)
        result["test_accuracy"] = model.score(X_test, y_test)
    else:
        train_preds = model(torch.tensor(X_train_prepared, dtype=torch.float32)).a
        valid_preds = model(torch.tensor(X_valid_prepared, dtype=torch.float32)).a
        test_preds = model(torch.tensor(X_test_prepared, dtype=torch.float32)).arg
        result["train_accuracy"] = accuracy_score(y_train, train_preds)
        result["valid_accuracy"] = accuracy_score(y_valid, valid_preds)
        result["test_accuracy"] = accuracy_score(y_test, test_preds)
        result["train_auc"] = roc_auc_score(y_train, train_preds)
        result["valid_auc"] = roc_auc_score(y_valid, valid_preds)
        result["test_auc"] = roc_auc_score(y_test, test_preds)
        result["train_f1_score"] = f1_score(y_train, train_preds)
        result["valid_f1_score"] = f1_score(y_valid, valid_preds)
        result["test_f1_score"] = f1_score(y_test, test_preds)
    if not expLog:
        expLog = [result]
    else:
        expLog.append(result)
```

```
    return {
        "exp_name": exp_name,
        "Train Time (sec)": train_time,
        "Test Time (sec)": test_time,
        "Train Acc": result["train_accuracy"],
        "Valid Acc": result["valid_accuracy"],
        "Test Acc": result["test_accuracy"],
        "Train AUC": result.get("train_auc", None),
        "Valid AUC": result.get("valid_auc", None),
        "Test AUC": result.get("test_auc", None),
        "Train F1 Score": result.get("train_f1_score", None),
        "Valid F1 Score": result.get("valid_f1_score", None),
        "Test F1 Score": result.get("test_f1_score", None)
    }
```

In [49]:

```
import torch
import torch.nn as nn
import torch.optim as optim
from torch.utils.data import DataLoader, TensorDataset
import numpy as np

np.random.seed(42)
torch.manual_seed(42)

class AdvancedMLP(nn.Module):
    def __init__(self, input_size, hidden_size1, hidden_size2, hidden_size3, num_c
        super(AdvancedMLP, self).__init__()
        self.layer1 = nn.Sequential(
            nn.Linear(input_size, hidden_size1),
            nn.BatchNorm1d(hidden_size1),
            nn.ReLU(),
            nn.Dropout(dropout_p)
        )
        self.layer2 = nn.Sequential(
            nn.Linear(hidden_size1, hidden_size2),
            nn.BatchNorm1d(hidden_size2),
            nn.ReLU(),
            nn.Dropout(dropout_p)
        )
        self.layer3 = nn.Sequential(
            nn.Linear(hidden_size2, hidden_size3),
            nn.BatchNorm1d(hidden_size3),
            nn.ReLU(),
            nn.Dropout(dropout_p)
        )
        self.fc_out = nn.Linear(hidden_size3, num_classes)

    def forward(self, x):
        out = self.layer1(x)
        out = self.layer2(out)
        out = self.layer3(out)
        out = self.fc_out(out)
        return out

# Assuming you have defined get_pipeline() function
# Preprocessing the data
data_prep_pipeline, selected_features = get_pipeline()
X_train_prepared = data_prep_pipeline.fit_transform(X_train)
X_valid_prepared = data_prep_pipeline.transform(X_valid)
X_test_prepared = data_prep_pipeline.transform(X_test)

# Creating PyTorch datasets
train_dataset = TensorDataset(torch.tensor(X_train_prepared, dtype=torch.float32),
```

```
valid_dataset = TensorDataset(torch.tensor(X_valid_prepared, dtype=torch.float32),
test_dataset = TensorDataset(torch.tensor(X_test_prepared, dtype=torch.float32), t

# Hyperparameters
input_size = X_train_prepared.shape[1]
hidden_size1 = 64
hidden_size2 = 32
hidden_size3 = 16
num_classes = 2
num_epochs = 5
batch_size = 64
learning_rate = 0.001
dropout_p = 0.5

# Defining the model
model = AdvancedMLP(input_size, hidden_size1, hidden_size2, hidden_size3, num_clas

# Creating dataloaders
train_loader = DataLoader(dataset=train_dataset, batch_size=batch_size, shuffle=Tr
valid_loader = DataLoader(dataset=valid_dataset, batch_size=batch_size, shuffle=Fa
test_loader = DataLoader(dataset=test_dataset, batch_size=batch_size, shuffle=Fals

# Loss and optimizer
criterion = nn.CrossEntropyLoss()
optimizer = optim.Adam(model.parameters(), lr=learning_rate)

# Training the model
for epoch in range(num_epochs):
    for i, (data, labels) in enumerate(train_loader):
        # Forward pass
        outputs = model(data)
        loss = criterion(outputs, labels)

        # Backward and optimize
        optimizer.zero_grad()
        loss.backward()
        optimizer.step()

    print(f"Epoch [{epoch+1}/{num_epochs}], Loss: {loss.item():.4f}")

# Calculate train and test times
train_time = num_epochs * len(train_loader)
test_time = len(test_loader)

expLog = []

# Assuming you have defined get_results() function
# Results
# Results
exp_name = f"AdvancedMLP_{len(selected_features)}_features"
expLog = get_results(expLog, exp_name, model, train_time, test_time, X_train, y_tr
expLog
```

```
Total Features: 124 - Numerical: 107, Categorical: 16
Epoch [1/5], Loss: 0.5539
Epoch [2/5], Loss: 0.5547
Epoch [3/5], Loss: 0.6656
Epoch [4/5], Loss: 0.5809
Epoch [5/5], Loss: 0.5767
```

Out[49]:
```
{'exp_name': 'AdvancedMLP_124_features',
 'Train Time (sec)': 2805,
 'Test Time (sec)': 99,
 'Train Acc': 0.6824454294555491,
 'Valid Acc': 0.6801825993555317,
 'Test Acc': 0.6815668930658664,
 'Train AUC': 0.6824438125176906,
 'Valid AUC': 0.6801799550869414,
 'Test AUC': 0.6816021608178614,
 'Train F1 Score': 0.6797492198262532,
 'Valid F1 Score': 0.6793214862681745,
 'Test F1 Score': 0.6788786237655304}
```

Write-up

Project Title: Home Credit Default Risk

Team and Phase leader plan

Phase	Contributors	Contribution Description
Phase 1 - Project Proposal	Pranay Namburi (Phase Leader)	1. Planning Phase, Scheduling Meetings, Coordinating Tasks and Assignment Submission. 2. Design baseline pipeline, describe pipeline components, create pipeline diagram, describe pipeline design. 3. Preparation of the Phase Leader Plan and Credit Assignment Plan.
	Sathwik Varikoti	1. Writing the Abstact of the Project. 2. Preparation of the Phase Leader Plan and Credit Assignment Plan.
	Jagadeesh Kovi	1. Listing the Machine learning algorithms and metrics to be used for the project. 2. Preparation of the Gantt Chart. 3. Preparation of the Phase Leader Plan and Credit Assignment Plan.
	Bindu Dokala	1. Describing the data on which further analysis is done. 2. Preparation of the Phase Leader Plan and Credit Assignment Plan.
Phase 2 - EDA and Baseline Pipeline	Pranay Namburi	Writing code to perform Feature Engineering and Hyperparameter tuning
	Sathwik Varikoti(Phase Leader)	1. Planning Phase, Scheduling Meetings, Coordinating Tasks and Assignment Submission. 2. EDA and data pre-processing - Part 2.
	Jagadeesh Kovi	1. EDA and data pre-processing - Part 1. 2. Making slides summarizing the work done in the entire phase.
	Bindu Dokala	1. Writing python script for functions, diagram and coding for base pipelines. 2. Compling submissions and describing the results.
Phase 3 - Final Project HCDR	Pranay Namburi	Hyperparameter Tuning
	Sathwik Varikoti	1. Writing python script for functions and coding for pipelines. 2. Making a report to present the findings from the work done in the entire phase.
	Jagadeesh Kovi(Phase Leader)	1. Planning Phase, Scheduling Meetings, Coordinating Tasks and Assignment Submission. 2. Feature Engineering and Feature Selection.
	Bindu Dokala	1. Writing python script for functions and coding for ensemble methods. 2. Making slides summarizing the work done in the entire phase and Presentation recording.
Phase 4 - Final Submission	Pranay Namburi	Making a report to present the findings from the work done in the entire phase.
	Sathwik Varikoti	1. Making slides summarizing the work done in the entire phase. 2. Presentation recording.
	Jagadeesh Kovi	1. Analysis of metrics and Loss functions. 2. Results for final submission of the report
	Bindu Dokala(Phase Leader)	1. Planning Phase, Scheduling Meetings, Coordinating Tasks and Assignment Submission. 2. Building, Training and Storing the MLP implementation.

Credit Assignment Plan

Phase	Task	Task description	Output	Person Hours	Dependent tasks	Person Assigned
1	Phase Management	Planning Phase, Scheduling Meetings, Coordinating Tasks and Assignment Submission.	NA	0.5 hours	None	Pranay Namburi
	Abstract	Project Abstract, Downloading data.	Output DataFile	1.5 hour	None	Sathwik Varikoti
	Describing data table	Loading libraries, writing table descriptions.	Jupyter Notebook with table descriptions	1 hour	None	Bindu Dokala
	Data Description	Defining pre-processing steps, Creating augmented data, Target variables distribution	Jupyter Notebook with data description	3 Hours	None	Bindu Dokala
	Listing ML algorithms and their metrics.	Analysising ML algorithms and their metrics respectively.	Jupyter Notebook with ML algorithms and their metrics.	2 hours	None	Jagadeesh Kovi
	Pipelines Description	Design baseline pipeline, describe pipeline components, create pipeline diagram, describe pipeline design.	Jupyter notebook with pipelines diagrams and descriptions.	2.5 hours	None	Pranay Namburi
	Gantt Chart	Gantt chart preparation	Gantt Chart	1 hour	None	Jagadeesh Kovi
	Phase leader plan Credit Assignment Plan	Work on phase leader plan and credit assignment plan	Tables describing both the plans	2 hours	None	Group
2	Phase Management	Planning Phase, Scheduling Meetings, Coordinating Tasks and Assignment Submission.	NA	1.5 hours	None	Sathwik Varikoti
	EDA	EDA and Data-preprocessing of the data collected	Jupyter notebook with preprocessed data	6 hours	None	Sathwik Varikoti, Jagadeesh Kovi
	Pipelines implementation	Writing python script for functions, diagram and coding for pipelines	Jupyter notebook with pipelines code	2 hours	EDA and Data-preprocessing of the data collected	Bindu Dokala
	Feature Engineering and Hyperparameter tuning	Performing feature engineering and tuning the hyperparameters to get the best results	Jupyter notebook with pipelines code	2 hours	Pipelines implementation	Pranay Namburi
	Video Presentation and Slides	Presentation Recording and Phase Presentation	N/A	1 hours	None	Jagadeesh Kovi
	Results	Compling submissions and describing the results.	Jupyter notebook with all results	2 hours	Executable Code with results	Bindu Dokala
3	Phase Management	Planning Phase, Scheduling Meetings, Coordinating Tasks and Assignment Submission.	NA	1.5 hours	None	Jagadeesh Kovi
	Pipelines Implementation	Code pipelines		2 hours	Phase 2 notebook	Sathwik Varikoti
	Feature Engineering and Feature Selection	Add code on to the pipelines by performing feature engineering and selection.	Jupyter notebook with pipelines code	3.5 hours	Pipelines Creation	Jagadeesh Kovi
	Hyper parameter tuning	Search for ideal model architecture	ideal architecture	2.5 Hours	Executable Pipeline Code	Pranay Namburi
	Video Presentation and Slides	Presentation Recording and Phase Presentation	N/A	1 hours	Executable Code with results	Bindu Dokala
	Results	Compling submissions and describing the results.	Jupyter notebook with all results	2 hours	Executable Code with results	Sathwik Varikoti
4	Phase Management	Planning Phase, Scheduling Meetings, Coordinating Tasks and Assignment Submission.	NA	1.5 hours	None	Bindu Dokala
	MLP Implementation	Building, Training and Storing the MLP implementation.	Jupyter notebook with updated code	4 hours	Phase3 Code	Bindu Dokala
	Loss Functions Implementation	Analysis of metrics and Loss functions	N/A	2 hours	MLP Implementation	Jagadeesh Kovi
	Results and Final Report	Results for final submission of the report		2 hours	Executable final code	Bindu Dokala, Sathwik Varikoti, Pranay Namburi, Jagadeesh Kovi
	Overall Appearance	Modify report for cohesessiveness and appearance.		1 hours	Executable final code	Bindu Dokala, Sathwik Varikoti, Pranay Namburi, Jagadeesh Kovi
	Video Presentation and Slides	Presentation Recording and Phase Presentation	N/A	1 hours	Executable Code with results	Pranay Namburi

Abstract

In response to Home Credit's challenge of assessing creditworthiness for clients with limited credit history, our project employs Logistic Regression with Lasso regularization (LASSO-CXE) and the K-Nearest Neighbors (KNN) algorithm.

We tackle data challenges through advanced techniques such as data cleaning, feature engineering, and the creation of new features. To address imbalanced datasets, we evaluate model performance using key metrics such as ROC AUC, F1 Score, and Balanced Accuracy. These metrics provide a nuanced understanding of the classifier's performance, considering both false positives and negatives.

Our goal is to enhance Home Credit's lending decisions, reduce unpaid loans, and extend financial services to individuals with limited access to traditional banking. The Logistic Regression model with Lasso regularization aids in feature selection and prevents overfitting, while KNN's adaptability proves valuable in assessing credit risk by identifying patterns in borrower profiles. This comprehensive approach ensures the development of a robust model for effective credit risk assessment.

Introduction

Background on Home Credit

Home Credit, a non-banking financial institution established in 1997 in the Czech Republic, caters to individuals with limited or no credit history who might otherwise be denied loans or fall prey to unscrupulous lenders. Operating in 14 countries, including the United States, Russia, Kazakhstan, Belarus, China, and India, Home Credit has amassed over 29 million customers, granted over 160 million loans, and accumulated total assets of 21 billion euros, with the majority of its business located in Asia, particularly China (as of May 19, 2018).

Currently employing various statistical and machine learning techniques to assess creditworthiness, Home Credit seeks Kagglers' assistance in unlocking the full potential of their data. This endeavor aims to ensure that creditworthy clients are not overlooked and that loans are tailored with appropriate principal amounts, maturities, and repayment schedules to empower clients' financial success.

Data Description

The Home Credit Default Risk dataset, obtained from the Kaggle project, aims to help Home Credit make informed decisions about loan applications for individuals who may not qualify through traditional banking systems. To accomplish this, Home Credit gathers various data sources, including phone and transaction records, to evaluate a borrower's ability to repay a loan.

At the heart of this dataset is the "application {train test}" table, which contains the loan applications that will be analyzed for potential default risk. Six additional tables provide supplementary information related to the primary table, forming a hierarchical structure. Detailed explanations of these tables are available from the HCDR Kaggle Competition.

Data files overview

- **POS_CASH_balance.csv -->** Shape: (10001358, 8) --> Numerical Features: 7 --> Categorical Features: 1
- **application_test.csv -->** Shape: (48744, 121) --> Numerical Features: 105 --> Categorical Features: 16
- **application_train.csv -->** Shape: (307511, 122) --> Numerical Features: 106 --> Categorical Features: 16
- **installments_payments.csv -->** Shape: (13605401, 8) --> Numerical Features: 8 --> Categorical Features: 0
- **bureau_balance.csv -->** Shape: (27299925, 3) --> Numerical Features: 2 --> Categorical Features: 1
- **credit_card_balance.csv -->** Shape: (3840312, 23) --> Numerical Features: 22 --> Categorical Features: 1
- **installments_payments.csv -->** Shape: (13605401, 8) --> Numerical Features: 8 --> Categorical Features: 0
- **bureau.csv -->** Shape: (1716428, 17) --> Numerical Features: 14 --> Categorical Features: 3

1. `application_{train|test}.csv` : This table contains static data for loan applications. The "train" version includes a target variable, while the "test" version does not.
2. `bureau.csv` : It holds information about a client's previous credits from other financial institutions reported to the Credit Bureau. Multiple rows can correspond to a single loan application.
3. `bureau_balance.csv` : This table provides monthly balances of previous credits reported to the Credit Bureau, creating multiple rows for each loan's history.
4. `POS_CASH_balance.csv` : It contains monthly snapshots of the balance for point of sales and cash loans that the applicant had with Home Credit, generating multiple rows for each loan's history.
5. `credit_card_balance.csv` : This table shows monthly balance snapshots of previous credit cards the applicant had with Home Credit, with multiple rows for each card's history.
6. `previous_application.csv` : This dataset includes all previous loan applications made by clients in the sample, with one row per application.
7. `installments_payments.csv` : It covers repayment history for credits disbursed by Home Credit, with one row for each payment or missed payment.
8. `HomeCredit_columns_description.csv` : This file provides descriptions for the columns in the various data files, helping users understand the data better.

Data Dictionary

As part of the data download comes a Data Dictionary. It is named as `HomeCredit_columns_description.csv`. It contains information about all fields present in all the above tables. (like the metadata).

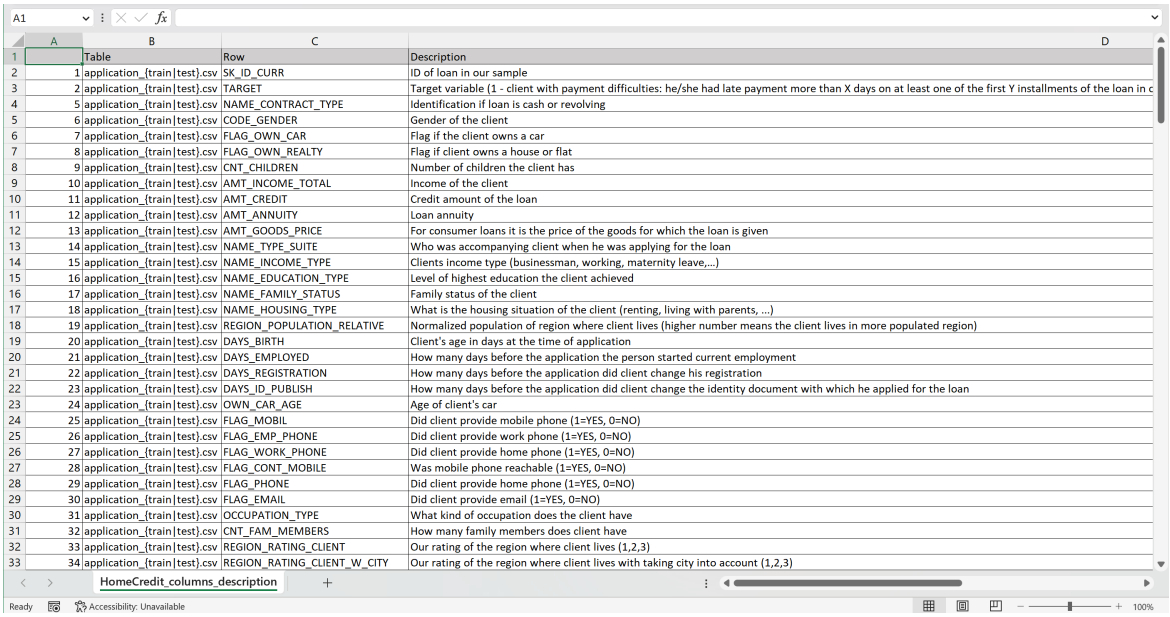
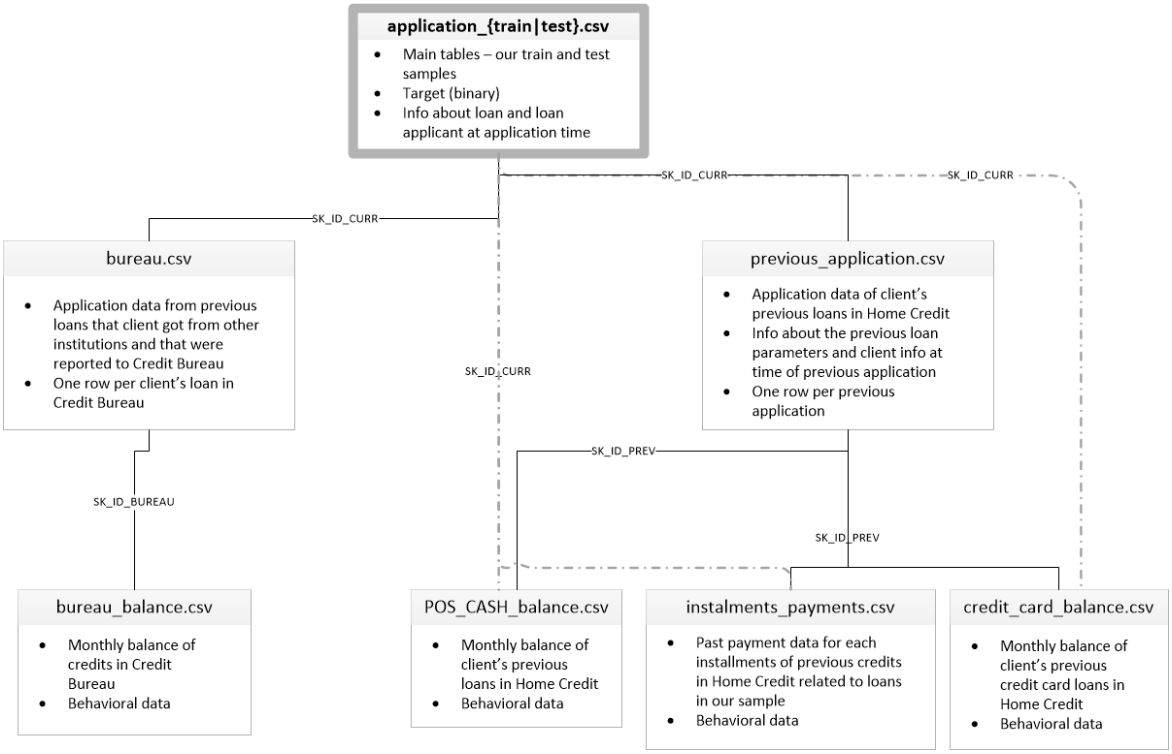


	Table	Row	Description
1	application_{train|test}.csv	SK_ID_CURR	ID of loan in our sample
2	application_{train|test}.csv	TARGET	Target variable (1 - client with payment difficulties: he/she had late payment more than X days on at least one of the first Y installments of the loan in c
5	application_{train|test}.csv	NAME_CONTRACT_TYPE	Identification if loan is cash or revolving
6	application_{train|test}.csv	CODE_GENDER	Gender of the client
7	application_{train|test}.csv	FLAG_OWN_CAR	Flag if the client owns a car
8	application_{train|test}.csv	FLAG_OWN_REALTY	Flag if client owns a house or flat
9	application_{train|test}.csv	CNT_CHILDREN	Number of children the client has
10	application_{train|test}.csv	AMT_INCOME_TOTAL	Income of the client
11	application_{train|test}.csv	AMT_CREDIT	Credit amount of the loan
12	application_{train|test}.csv	AMT_ANNUITY	Loan annuity
13	application_{train|test}.csv	AMT_GOODS_PRICE	For consumer loans it is the price of the goods for which the loan is given
14	application_{train|test}.csv	NAME_TYPE_SUITE	Who was accompanying client when he was applying for the loan
15	application_{train|test}.csv	NAME_INCOME_TYPE	Clients income type (businessman, working, maternity leave,...)
16	application_{train|test}.csv	NAME_EDUCATION_TYPE	Level of highest education the client achieved
17	application_{train|test}.csv	NAME_FAMILY_STATUS	Family status of the client
18	application_{train|test}.csv	NAME_HOUSING_TYPE	What is the housing situation of the client (renting, living with parents, ...)
19	application_{train|test}.csv	REGION_POPULATION_RELATIVE	Normalized population of region where client lives (higher number means the client lives in more populated region)
20	application_{train|test}.csv	DAYS_BIRTH	Client's age in days at the time of application
21	application_{train|test}.csv	DAYS_EMPLOYED	How many days before the application the person started current employment
22	application_{train|test}.csv	DAYS_REGISTRATION	How many days before the application did client change his registration
23	application_{train|test}.csv	DAYS_ID_PUBLISH	How many days before the application did client change the identity document with which he applied for the loan
24	application_{train|test}.csv	OWN_CAR_AGE	Age of client's car
25	application_{train|test}.csv	FLAG_MOBIL	Did client provide mobile phone (1=YES, 0=NO)
26	application_{train|test}.csv	FLAG_EMP_PHONE	Did client provide work phone (1=YES, 0=NO)
27	application_{train|test}.csv	FLAG_WORK_PHONE	Did client provide home phone (1=YES, 0=NO)
28	application_{train|test}.csv	FLAG_CONT_MOBILE	Was mobile phone reachable (1=YES, 0=NO)
29	application_{train|test}.csv	FLAG_PHONE	Did client provide home phone (1=YES, 0=NO)
30	application_{train|test}.csv	FLAG_EMAIL	Did client provide email (1=YES, 0=NO)
31	application_{train|test}.csv	OCCUPATION_TYPE	What kind of occupation does the client have
32	application_{train|test}.csv	CNT_FAM_MEMBERS	How many family members does client have
33	application_{train|test}.csv	REGION_RATING_CLIENT	Our rating of the region where client lives (1,2,3)
34	application_{train|test}.csv	REGION_RATING_CLIENT_W_CITY	Our rating of the region where client lives with taking city into account (1,2,3)

Table Diagram



Tasks to be tackled

The tasks to be addressed in this phase of the project are given below:

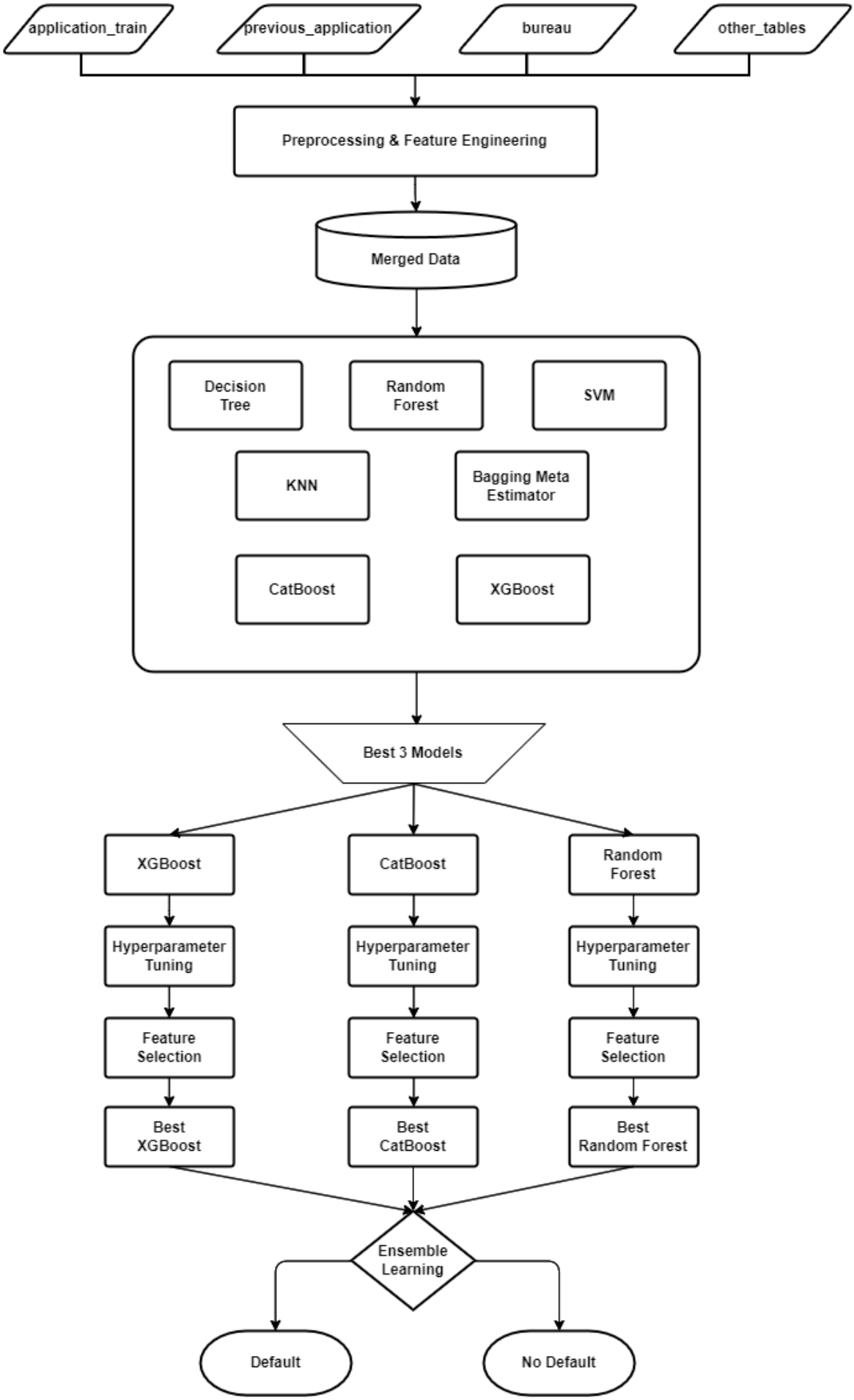
- **`Join the datasets`** : Combine the remaining datasets to form a comprehensive dataset that captures all relevant customer information.
- **`Perform EDA on other datasets`** : Conduct Exploratory Data Analysis on datasets excluding application_train and the merged datasets to gain insights and understand the

relationships between various features.

- `Identify missing values and highly correlated features in the merged data` : Detect and handle missing values in the merged dataset, and eliminate highly correlated features to prevent multicollinearity.
- `Detect and mitigate potential errors in the merged data` : Examine any errors in the merged data that could influence the model's accuracy and take appropriate measures to mitigate them.
- `Incorporate domain knowledge features` : Add domain knowledge features that could potentially enhance the model's performance.
- `Analyze the impact of newly added features on the target variable` : Investigate the relationship between the new features and the target variable to comprehend their effect on the model's performance.
- `Build upon models from Phase 2` : Develop and refine models from Phase 2, such as Logistic Regression, to include the new features and insights acquired in the current phase.
- `Model selection and training` : Choose suitable machine learning models, such as lasso regression, logistic regression, decision trees, random forests, gradient boosting machines (GBMs), and neural networks. Split the data into training and testing sets and train the models.
- `Calculate and validate the results` : Evaluate the performance of the updated models using suitable metrics like accuracy, precision, recall, F1-score, and ROC-AUC, and validate the results to ensure the models' effectiveness in predicting default probabilities.
- `Model evaluation` : Evaluate the performance of the models using appropriate metrics such as accuracy, precision, recall, F1-score, and ROC-AUC. We will compare these models' performance and identify the best performing model based on these evaluation metrics.
- `Perform hyperparameter tuning with GridSearchCV` : Utilize GridSearchCV to determine the most significant hyperparameters for the chosen models and optimize their performance.
- `Perform ensemble modelling` : Perform ensemble modelling to see some improvement in models.

By implementing the best model, Home Credit will be able to make more informed lending decisions, minimize unpaid loans, and promote financial services for individuals with limited access to banking, ultimately fostering financial inclusion for underserved populations. The effectiveness of our models in predicting default probabilities will be assessed using key metrics such as ROC AUC, F1 Score. The corresponding public and private scores will also be evaluated to determine our model's performance.

Block Diagram of the Approach



Experimental results

	exp_name	description	Train Time (sec)	Test Time (sec)	Train Acc	Valid Acc	Test Acc	Train AUC	Valid AUC	Test AUC	Train F1 Score	Valid F1 Score	Test F1 Score
0	Model-1 Baseline LR	Logistic regression with undersampled data 124...	2.5144	0.0720	0.7727	0.7647	0.7802	0.7545	0.7421	0.7535	0.3595	0.3300	0.3664
1	Model-2 Baseline LR	Logistic regression with undersampled data-2 1...	1.1938	0.0496	0.6876	0.6843	0.6904	0.7525	0.7489	0.7535	0.6865	0.6854	0.6900
2	Model-3 KNN	KNN with undersampled data-2 124 features	0.3262	1.0494	0.6950	0.6155	0.6184	0.7625	0.6571	0.6550	0.6992	0.6205	0.6226
3	Model-4 Decision Tree	Decision tree with undersampled data-2 124 fea...	1.4838	0.0504	0.6749	0.6535	0.6591	0.7380	0.7105	0.7129	0.6881	0.6678	0.6730
4	Model-5 Random Forest	Random Forest with undersampled data-2 124 fea...	20.5597	0.4587	0.7665	0.6657	0.6666	0.8504	0.7245	0.7275	0.7676	0.6637	0.6647
5	Model-6 Bagging Meta Estimator	Bagging Meta Estimator with undersampled data-...	5.4167	0.2396	0.9844	0.6445	0.6430	0.9990	0.6973	0.6924	0.9843	0.6184	0.6151
6	Model-7 SVM	SVM with undersampled data-2 124 features	2681.6150	15.7188	0.9846	0.6411	0.6421	0.9990	0.6965	0.6899	0.9845	0.6138	0.6145
7	Model-8 XGBoost	XGBoost SAMME with undersampled data-2 124 fea...	4.8359	0.0720	0.7311	0.6931	0.6955	0.8103	0.7614	0.7607	0.7300	0.6925	0.6946
8	Model-9 CATBoost	CATBoost with undersampled data-2 124 features	12.1853	0.2737	0.6955	0.6917	0.6933	0.7669	0.7574	0.7593	0.6945	0.6922	0.6916
9	Model 10 - XGBOOST -Feature &hyperParameter Tu...	XGBOOST Tuned with x>0 113 features	4.5328	0.0625	0.7294	0.6959	0.6983	0.8092	0.7616	0.7623	0.7283	0.6953	0.6971
10	Model 11 - XGBOOST -Feature &hyperParameter Tu...	XGBOOST Tuned with x>0.01 113 features	4.4022	0.0781	0.7294	0.6959	0.6983	0.8092	0.7616	0.7623	0.7283	0.6953	0.6971
11	Model 12 - XGBOOST -Feature &hyperParameter Tu...	XGBOOST Tuned with x>0.005 110 features	4.4098	0.0666	0.7298	0.6933	0.6961	0.8104	0.7619	0.7612	0.7289	0.6926	0.6954
12	Model 13 - CatBOOST -Feature &hyperParameter T...	CatBOOST Tuned with x>0 120 features	38.4053	0.2749	0.7528	0.6935	0.6970	0.8367	0.7594	0.7618	0.7524	0.6940	0.6964
13	Model 14 - CatBOOST -Feature &hyperParameter T...	CatBOOST Tuned with x>0.1 103 features	38.2332	0.2384	0.7513	0.6897	0.6944	0.8358	0.7590	0.7612	0.7510	0.6908	0.6935
14	Model 15 - CatBOOST -Feature &hyperParameter T...	CatBOOST Tuned with x>0.005 113 features	38.7217	0.2463	0.7513	0.6931	0.6959	0.8360	0.7596	0.7619	0.7508	0.6935	0.6952
15	Model 16 - Random Forest -Feature &hyperParame...	Random Forest Tuned with x>0 116 features	2.9437	0.1092	0.7531	0.6819	0.6811	0.8326	0.7409	0.7408	0.7570	0.6837	0.6819
16	Model 17 - Random Forest -Feature &hyperParame...	Random Forest Tuned with x>0.1 19 features	1.4149	0.0917	0.6958	0.6740	0.6642	0.7617	0.7261	0.7219	0.6967	0.6769	0.6646
17	Model 18 - Random Forest -Feature &hyperParame...	Random Forest Tuned with x>0.005 58 features	2.4448	0.1110	0.7506	0.6846	0.6809	0.8298	0.7426	0.7427	0.7539	0.6865	0.6826
18	Model 19 - Ensemble Learner - Voting Classifier	Tuned and selected XgBoost, catboost, random ...	39.4357	0.4504	0.7465	0.6921	0.6945	0.8289	0.7599	0.7612	0.7466	0.6925	0.6939
19	Model 20 - Ensemble Learner - Stacking Classi...	Tuned and selected XgBoost, catboost, random ...	203.8994	0.4554	0.7397	0.6954	0.6975	0.8225	0.7614	0.7630	0.7389	0.6959	0.6973

Discussion of the Results

In this study, a variety of machine learning models were trained and evaluated to identify the best performing model. The models include logistic regression, k-nearest neighbors (KNN), support vector machines (SVM), decision trees, random forests, bagging meta estimator, CATBoost, and ensemble learners (voting and stacking classifiers).

The results show significant variation in the performance of these models in terms of accuracy, area under the curve (AUC), and F1 scores. In general, ensemble models like voting and stacking classifiers (Models 19 and 20) and tuned random forests (Models 16, 17, and 18) have performed better compared to other models.

The bagging meta estimator (Model 6) exhibits very high training accuracy (0.9844) and F1 score (0.9843), but it performs poorly on the validation (accuracy: 0.6477, F1 score: 0.6184) and test datasets (accuracy: 0.643, F1 score: 0.6151), indicating that the model is overfitting. Overfitting occurs when a model learns the training data too well and fails to generalize to unseen data.

On the other hand, some models like KNN (Model 3) and SVM (Model 7) display lower accuracy and F1 scores on both training and validation sets. For example, KNN has a training accuracy of 0.695 and F1 score of 0.6992, while the validation accuracy is 0.6155 and F1 score is 0.6205. This is a sign of underfitting, which occurs when a model is not able to capture the underlying patterns in the data.

The ensemble learners (Models 19 and 20), which combine multiple tuned models (XgBoost, CatBoost, random forests), exhibit a more balanced performance across training, validation, and test datasets. For instance, Model 19 has a training accuracy of 0.7465, validation accuracy of 0.6921, and test accuracy of 0.6945. Similarly, the F1 scores are 0.7466, 0.6925, and 0.6939, respectively. These models have higher accuracy, AUC, and F1 scores compared to other models, indicating that they are able to generalize well to unseen data without overfitting or underfitting.

In conclusion, the ensemble learners, specifically the voting (Model 19) and stacking classifiers (Model 20), and the tuned random forest models (Models 16, 17 and 18) with a training accuracy of 0.7465, validation accuracy of 0.6921, and test accuracy of 0.6945 seem to be the most promising candidates for this problem. They strike a balance between avoiding overfitting and underfitting while maintaining good performance across different evaluation

metrics. Further tuning and optimization of these models could potentially lead to even better results.

Conclusion

We continued from where we left off in Phase 2. We performed feature engineering, feature selection and hyper parameter tuning and experimented on a range of machine learning algorithms. Our experiments help compare these models' performance and identify the most effective pipeline to minimize the default risk.

Till now, based on our experimentation, we have determined that the ensemble learning methods (Model 19 & Model 20) outperformed all other models. These methods combined the predictions of XGBoost, CatBoost, and Random Forest using Voting Classifier and Stacking Classifier. The results showed that these models achieved the highest test F1 scores, 0.6939 and 0.6973, respectively. Additionally, this method has the greatest test AUC of 0.7612 and 0.763 for the stacking classifier and the voting classifier, respectively. This indicates that employing ensemble methods improved the ability to distinguish between positive and negative classes.

The phase's findings imply that ensemble learning methods might work well for this classification problem.

References

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

Logistic Regression:

- Scikit-learn documentation: https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
- Introduction to Logistic Regression: https://towardsdatascience.com/introduction-to-logistic-regression-66248243c148

K-Nearest Neighbors (KNN):

- Scikit-learn documentation: https://scikit-learn.org/stable/modules/generated/sklearn.neighbors.KNeighborsClassifier.html
- KNN Classifier explained: https://towardsdatascience.com/k-nearest-neighbor-python-2fccc47d2a55

Decision Trees:

- Scikit-learn documentation: https://scikit-learn.org/stable/modules/generated/sklearn.tree.DecisionTreeClassifier.html
- Introduction to Decision Trees: https://towardsdatascience.com/decision-trees-in-machine-learning-641b9c4e8052

Random Forest:

- Scikit-learn documentation: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
- Understanding Random Forest: https://towardsdatascience.com/understanding-random-forest-58381e0602d2

Bagging Meta Estimator:

- Scikit-learn documentation: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.BaggingClassifier.html
- Bagging Classifier explained: https://towardsdatascience.com/bagging-and-boosting-in-machine-learning-d6fa0e5c7f9d

Support Vector Machines (SVM):

- Scikit-learn documentation: https://scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html
- Support Vector Machines explained: https://towardsdatascience.com/support-vector-machines-svm-c9ef22815589

XGBoost:

- XGBoost documentation: https://xgboost.readthedocs.io/en/latest/python/python_intro.html
- XGBoost: A Complete Guide: https://towardsdatascience.com/xgboost-a-complete-guide-34cd8f24eb01

CATBoost:

- CATBoost documentation: https://catboost.ai/docs/
- CATBoost: A beginner's guide: https://towardsdatascience.com/catboost-a-beginners-guide-to-competitive-classification-bbb1e7a1f09d

Ensemble Learner - Voting Classifier:

- Scikit-learn documentation: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.VotingClassifier.html
- Voting Classifier explained: https://towardsdatascience.com/ensemble-methods-in-machine-learning-what-are-they-and-why-use-them-68ad99560e4

Ensemble Learner - Stacking Classifier:

- Scikit-learn documentation: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.StackingClassifier.html
- Stacking Classifier explained: https://towardsdatascience.com/stacking-classifiers-for-higher-predictive-performance-566f963e4840

Check out this guide on automated feature engineering with Featuretools in Python: https://www.analyticsvidhya.com/blog/2018/08/guide-automated-feature-engineering-featuretools-python/

Discover CatBoost, an automated machine learning library for handling categorical data, in this article: https://www.analyticsvidhya.com/blog/2017/08/catboost-automated-categorical-

data/