# Home Credit Default Risk (HCDR)

The 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

# Kaggle API setup

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

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

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

1. Install library

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

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

```
!pip install kaggle
```

```
Looking in indexes: https://pypi.org/simple, https://us-python.pkg.dev/colab-
Requirement already satisfied: kaggle in /usr/local/lib/python3.9/dist-packag
Requirement already satisfied: python-slugify in /usr/local/lib/python3.9/dis
Requirement already satisfied: python-dateutil in /usr/local/lib/python3.9/di
Requirement already satisfied: requests in /usr/local/lib/python3.9/dist-pack
Requirement already satisfied: six>=1.10 in /usr/local/lib/python3.9/dist-pac
Requirement already satisfied: urllib3 in /usr/local/lib/python3.9/dist-packa
Requirement already satisfied: tqdm in /usr/local/lib/python3.9/dist-packages
Requirement already satisfied: certifi in /usr/local/lib/python3.9/dist-packa
Requirement already satisfied: text-unidecode>=1.3 in /usr/local/lib/python3.
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.9/dist-
Requirement already satisfied: charset-normalizer~=2.0.0 in /usr/local/lib/py
```

```
!pwd
```

```
/content
```

```
!pwd
```

```
/content
```

```
!ls -l  ./kaggle.json
```

```
-rw-r--r-- 1 root root 68 Apr 25 05:37 ./kaggle.json
```

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

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

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

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

# Dataset and how to download

## Back ground Home Credit Group

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

### Home Credit Group

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

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

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

## Background on the dataset

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

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

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

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

## Data files overview

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

There are 7 different sources of data:

- **application_train/application_test (307k rows, and 48k rows):** the main training and testing data with information about each loan application at Home Credit. Every loan has its own row and is identified by the feature SK_ID_CURR. The training application data comes with the TARGET indicating **0: the loan was repaid** or **1: the loan was not repaid**. The target variable defines if the client had payment difficulties meaning he/she had late payment more than X days on at least one of the first Y installments of the loan. Such case is marked as 1 while other all other cases as 0.

- **bureau (1.7 Million rows):** data concerning client's previous credits from other financial institutions. Each previous credit has its own row in bureau, but one loan in the application data can have multiple previous credits.
- **bureau_balance (27 Million rows):** monthly data about the previous credits in bureau. Each row is one month of a previous credit, and a single previous credit can have multiple rows, one for each month of the credit length.
- **previous_application (1.6 Million rows):** previous applications for loans at Home Credit of clients who have loans in the application data. Each current loan in the application data can have multiple previous loans. Each previous application has one row and is identified by the feature SK_ID_PREV.
- **POS_CASH_BALANCE (10 Million rows):** monthly data about previous point of sale or cash loans clients have had with Home Credit. Each row is one month of a previous point of sale or cash loan, and a single previous loan can have many rows.
- credit_card_balance: monthly data about previous credit cards clients have had with Home Credit. Each row is one month of a credit card balance, and a single credit card can have many rows.
- **installments_payment (13.6 Million rows):** payment history for previous loans at Home Credit. There is one row for every made payment and one row for every missed payment.

## Table sizes

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

Double-click (or enter) to edit

## Downloading the files via Kaggle API

Create a base directory:

```
DATA_DIR = "Data/home-credit-default-risk"   #same level as course repo in the data di
```

Please download the project data files and data dictionary and unzip them using either of the following approaches:

1. Click on the `Download` button on the following Data Webpage and unzip the zip file to the `BASE_DIR`
2. If you plan to use the Kaggle API, please use the following steps.

```
DATA_DIR = "Data/home-credit-default-risk"   #same level as course repo in the da
#DATA_DIR = os.path.join('./ddddd/')
!mkdir DATA_DIR
```

```
!ls -l DATA_DIR
```

```
total 0
```

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

```
Downloading home-credit-default-risk.zip to Data/home-credit-default-risk
100% 687M/688M [00:36<00:00, 22.1MB/s]
100% 688M/688M [00:36<00:00, 19.9MB/s]
```

```
!pwd
```

```
/content
```

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

```
total 704708
-rw-r--r-- 1 root root 721616255 Apr 25 20:52 home-credit-default-risk.zip
```

```
#!rm -r  DATA_DIR
```

## Imports

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import os
import zipfile
from sklearn.base import BaseEstimator, TransformerMixin
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline, FeatureUnion
from pandas.plotting import scatter_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
import warnings
warnings.filterwarnings('ignore')
```

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

# Data files overview

## Data Dictionary

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

Double-click (or enter) to edit

# Application train

```
DATA_DIR = "Data/home-credit-default-risk"
```

```
ls -l Data/home-credit-default-risk/application_train.csv
```

```
-rw-r--r-- 1 root root 166133370 Apr 25 20:52 Data/home-credit-default-risk/a
```

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import os
import zipfile
from sklearn.base import BaseEstimator, TransformerMixin
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline, FeatureUnion
from pandas.plotting import scatter_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
import warnings
warnings.filterwarnings('ignore')
```

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

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

datasets['credit_card_balance'].shape
```

```
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_ACTUA |
|---|---|---|---|---|---|

| | | | | | |
|---|---|---|---|---|---|
| **0** | 2562384 | 378907 | -6 | 56.970 | 13500 |
| **1** | 2582071 | 363914 | -1 | 63975.555 | 4500 |
| **2** | 1740877 | 371185 | -7 | 31815.225 | 45000 |
| **3** | 1389973 | 337855 | -4 | 236572.110 | 22500 |
| **4** | 1891521 | 126868 | -1 | 453919.455 | 45000 |

5 rows × 23 columns

```
(3840312, 23)
```

```
DATA_DIR
```

```
'Data/home-credit-default-risk'
```

## 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.

```
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 |
|---|---|---|---|---|---|
| **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.

```
%%time
ds_names = ("application_train", "application_test", "bureau","bureau_balance","c
            "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_na
```

```
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_ |
|---|---|---|---|---|---|

| | | | | | |
|---|---|---|---|---|---|
| **0** | 100002 | 1 | Cash loans | M | N |
| **1** | 100003 | 0 | Cash loans | F | N |
| **2** | 100004 | 0 | Revolving loans | M | Y |
| **3** | 100006 | 0 | Cash loans | F | N |
| **4** | 100007 | 0 | Cash loans | M | N |

5 rows × 122 columns

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

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

5 rows × 121 columns

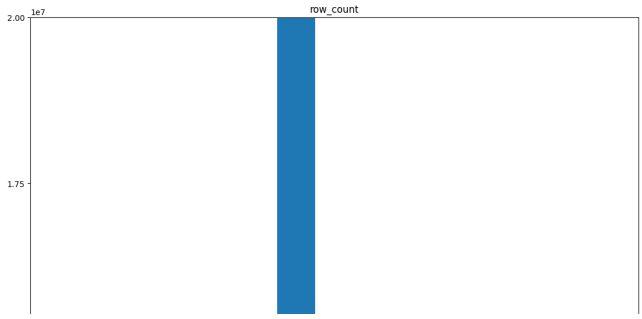
```
bureau: shape is (1716428, 17)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1716428 entries, 0 to 1716427
Data columns (total 17 columns):
 #   Column                  Dtype
---  ------                  -----
 0   SK_ID_CURR              int64
 1   SK_ID_BUREAU            int64
 2   CREDIT_ACTIVE           object
 3   CREDIT_CURRENCY         object
 4   DAYS_CREDIT             int64
 5   CREDIT_DAY_OVERDUE      int64
 6   DAYS_CREDIT_ENDDATE     float64
 7   DAYS_ENDDATE_FACT       float64
 8   AMT_CREDIT_MAX_OVERDUE  float64
 9   CNT_CREDIT_PROLONG      int64
 10  AMT_CREDIT_SUM          float64
 11  AMT_CREDIT_SUM_DEBT     float64
 12  AMT_CREDIT_SUM_LIMIT    float64
 13  AMT_CREDIT_SUM_OVERDUE  float64
 14  CREDIT_TYPE             object
```

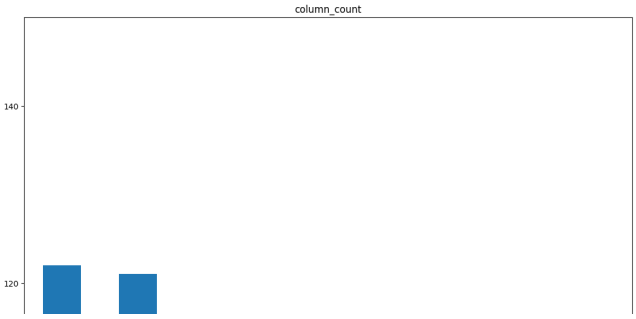
```
for ds_name in datasets.keys():
    print(f'dataset {ds_name:24}: [ {datasets[ds_name].shape[0]:10,}, {datasets[d
```

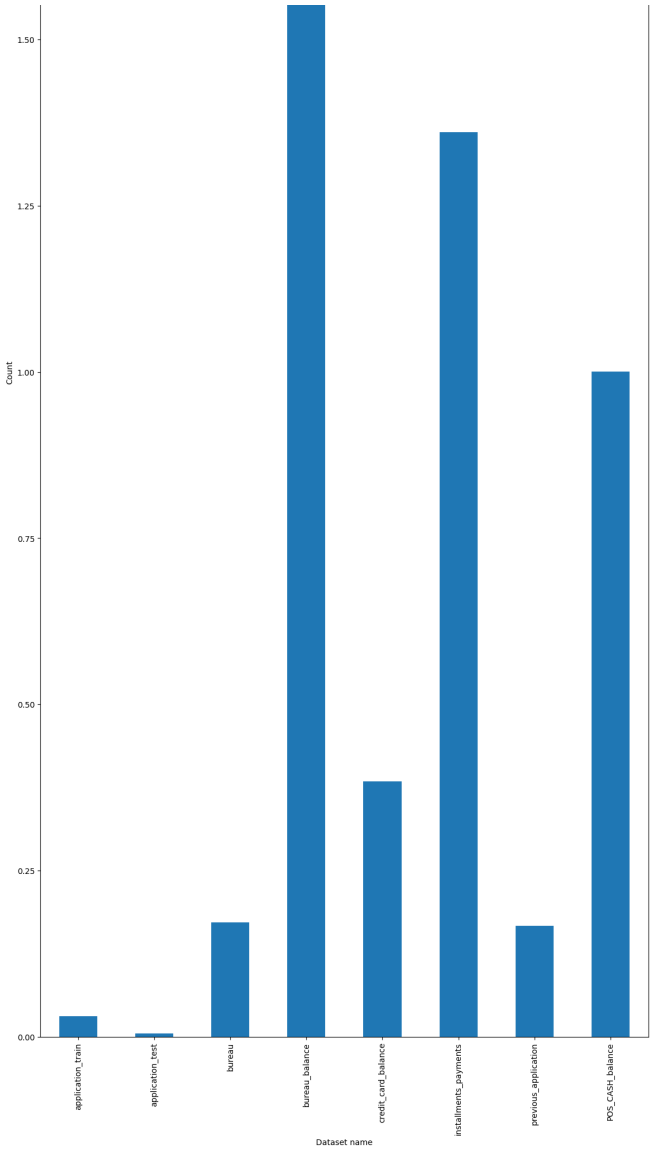
```
dataset credit_card_balance     : [  3,840,312, 23]
dataset application_test        : [     48,744, 121]
dataset application_train       : [    307,511, 122]
dataset bureau                  : [  1,716,428, 17]
dataset bureau_balance          : [ 27,299,925, 3]
dataset installments_payments   : [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

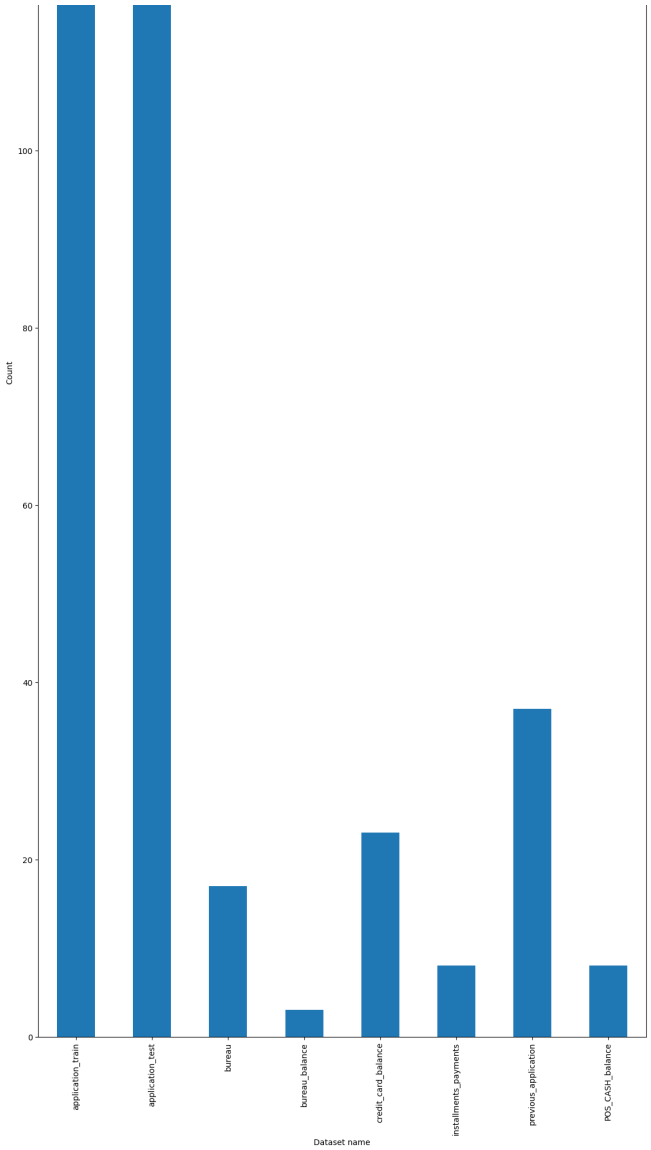
```
#Visualizing the number of rows and columns in each dataset
pd.options.display.float_format = '{:.4f}'.format
ds_info = pd.DataFrame(columns=['row_count','column_count'],index=ds_names)
for ds_name in ds_names:
    ds_info['row_count'][ds_name] = datasets[ds_name].shape[0]
    ds_info['column_count'][ds_name] = datasets[ds_name].shape[1]
print(ds_info)
fig = plt.figure(figsize=(30,30))
ax1 = fig.add_subplot(1,2,1)
ax2 = fig.add_subplot(1,2,2)
ylim = [20000000,150]

axes = [ax1,ax2]
for i in range(len(ds_info.columns)):
    ds_info.iloc[:,i].plot(kind = 'bar',ax=axes[i],title = ds_info.columns[i],xla
```

```
                      row_count column_count
application_train        307511          122
application_test          48744          121
bureau                  1716428           17
bureau_balance         27299925            3
credit_card_balance     3840312           23
installments_payments  13605401            8
previous_application    1670214           37
POS_CASH_balance       10001358            8
```




Count
1.50
1.25
1.00
0.75
0.50
0.25
0.00
application_train
application_test
bureau
bureau_balance
credit_card_balance
installments_payments
previous_application
POS_CASH_balance
Dataset name


Count
100
80
60
40
20
0
application_train
application_test
bureau
bureau_balance
credit_card_balance
installments_payments
previous_application
POS_CASH_balance
Dataset name

# Observation:

From the graph, it is clear that the dataset with the highest number of rows in the HCDR dataset is 'bureau_balance', with over 27 million rows, but it has the least number of features with only 3. The dataset 'install_payments' has the second-highest number of rows, with over 13 million. On the other hand, the 'application' dataset has the highest number of features, with 121, followed by the 'previous_applications' dataset with 37 features.

```
Data columns (total 37 columns):
```

# Exploratory Data Analysis

## Summary of Datasets

```
1   SK_ID_CURR                    1670214 non-null  int64
```

```
from IPython.display import display, HTML
pd.set_option("display.max_rows", None, "display.max_columns", None)

def dataset_summary(df, name):
  print(f"Summary of the dataset '{name}':\n")
  print("Basic Info:")
  print(datasets[name].info(verbose=True, null_counts=True))
  print(f"\nDescription of the dataset {name}:\n")
  print(display(HTML(np.round(datasets[name].describe(),2).to_html())))
  print("\nData Types feature counts:\n",df.dtypes.value_counts())
  print("\nDataframe Shape: ", df.shape)
  print("\nUnique elements count in each object:")
  print(df.select_dtypes('object').apply(pd.Series.nunique, axis = 0))
  df_dtypes = df.columns.to_series().groupby(df.dtypes).groups
  print(f"\n\nList of Categorical and Numerical(int + float) features  of {name}:
  for k, v in df_dtypes.items():
      print({k.name: v}, '\n')
```

```
23  NAME_PORTFOLIO                1670214 non-null  object
```

```
dataset_summary(datasets['application_train'], 'application_train')
```

```
Summary of the dataset 'application_train':

Basic Info:
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 307511 entries, 0 to 307510
Data columns (total 122 columns):
 #    Column                        Non-Null Count   Dtype
---   ------                        --------------   -----
 0    SK_ID_CURR                    307511 non-null  int64
 1    TARGET                        307511 non-null  int64
 2    NAME_CONTRACT_TYPE            307511 non-null  object
 3    CODE_GENDER                   307511 non-null  object
 4    FLAG_OWN_CAR                  307511 non-null  object
 5    FLAG_OWN_REALTY               307511 non-null  object
 6    CNT_CHILDREN                  307511 non-null  int64
 7    AMT_INCOME_TOTAL              307511 non-null  float64
 8    AMT_CREDIT                    307511 non-null  float64
 9    AMT_ANNUITY                   307499 non-null  float64
 10   AMT_GOODS_PRICE               307233 non-null  float64
 11   NAME_TYPE_SUITE               306219 non-null  object
 12   NAME_INCOME_TYPE              307511 non-null  object
 13   NAME_EDUCATION_TYPE           307511 non-null  object
 14   NAME_FAMILY_STATUS            307511 non-null  object
 15   NAME_HOUSING_TYPE             307511 non-null  object
 16   REGION_POPULATION_RELATIVE    307511 non-null  float64
 17   DAYS_BIRTH                    307511 non-null  int64
 18   DAYS_EMPLOYED                 307511 non-null  int64
 19   DAYS_REGISTRATION             307511 non-null  float64
 20   DAYS_ID_PUBLISH               307511 non-null  int64
 21   OWN_CAR_AGE                   104582 non-null  float64
 22   FLAG_MOBIL                    307511 non-null  int64
 23   FLAG_EMP_PHONE                307511 non-null  int64
 24   FLAG_WORK_PHONE               307511 non-null  int64
 25   FLAG_CONT_MOBILE              307511 non-null  int64
 26   FLAG_PHONE                    307511 non-null  int64
 27   FLAG_EMAIL                    307511 non-null  int64
 28   OCCUPATION_TYPE               211120 non-null  object
 29   CNT_FAM_MEMBERS               307509 non-null  float64
 30   REGION_RATING_CLIENT          307511 non-null  int64
 31   REGION_RATING_CLIENT_W_CITY   307511 non-null  int64
 32   WEEKDAY_APPR_PROCESS_START    307511 non-null  object
 33   HOUR_APPR_PROCESS_START       307511 non-null  int64
 34   REG_REGION_NOT_LIVE_REGION    307511 non-null  int64
 35   REG_REGION_NOT_WORK_REGION    307511 non-null  int64
 36   LIVE_REGION_NOT_WORK_REGION   307511 non-null  int64
 37   REG_CITY_NOT_LIVE_CITY        307511 non-null  int64
 38   REG_CITY_NOT_WORK_CITY        307511 non-null  int64
 39   LIVE_CITY_NOT_WORK_CITY       307511 non-null  int64
 40   ORGANIZATION_TYPE             307511 non-null  object
 41   EXT_SOURCE_1                  134133 non-null  float64
 42   EXT_SOURCE_2                  306851 non-null  float64
 43   EXT_SOURCE_3                  246546 non-null  float64
 44   APARTMENTS_AVG                151450 non-null  float64
 45   BASEMENTAREA_AVG              127568 non-null  float64
```

```
 46   YEARS_BEGINEXPLUATATION_AVG   157504 non-null  float64
 47   YEARS_BUILD_AVG               103023 non-null  float64
 48   COMMONAREA_AVG                92646 non-null   float64
 49   ELEVATORS_AVG                 143620 non-null  float64
 50   ENTRANCES_AVG                 152683 non-null  float64
```

## Observation:

From the dataset description, it is observed that there are features in application train dataset that have negative values, which is unexpected for certain features. Those features are DAYS_BIRTH, DAYS_EMPLOYED, DAYS_REGISTRATION, DAYS_ID_PUBLISH, DAYS_LAST_PHONE_CHARGE.

```
 61   YEARS_BUILD_MODE              103023 non-null  float64
```

```
dataset_summary(datasets['application_test'], 'application_test')
```

```
Summary of the dataset 'application_test':

Basic Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 48744 entries, 0 to 48743
Data columns (total 121 columns):
 #    Column                        Non-Null Count  Dtype
---   ------                        --------------  -----
 0    SK_ID_CURR                    48744 non-null  int64
 1    NAME_CONTRACT_TYPE            48744 non-null  object
 2    CODE_GENDER                   48744 non-null  object
 3    FLAG_OWN_CAR                  48744 non-null  object
 4    FLAG_OWN_REALTY               48744 non-null  object
 5    CNT_CHILDREN                  48744 non-null  int64
 6    AMT_INCOME_TOTAL              48744 non-null  float64
 7    AMT_CREDIT                    48744 non-null  float64
 8    AMT_ANNUITY                   48720 non-null  float64
 9    AMT_GOODS_PRICE               48744 non-null  float64
 10   NAME_TYPE_SUITE               47833 non-null  object
 11   NAME_INCOME_TYPE              48744 non-null  object
 12   NAME_EDUCATION_TYPE           48744 non-null  object
 13   NAME_FAMILY_STATUS            48744 non-null  object
 14   NAME_HOUSING_TYPE             48744 non-null  object
 15   REGION_POPULATION_RELATIVE    48744 non-null  float64
 16   DAYS_BIRTH                    48744 non-null  int64
 17   DAYS_EMPLOYED                 48744 non-null  int64
 18   DAYS_REGISTRATION             48744 non-null  float64
 19   DAYS_ID_PUBLISH               48744 non-null  int64
 20   OWN_CAR_AGE                   16432 non-null  float64
 21   FLAG_MOBIL                    48744 non-null  int64
 22   FLAG_EMP_PHONE                48744 non-null  int64
 23   FLAG_WORK_PHONE               48744 non-null  int64
```

```
 23   FLAG_WORK_PHONE               48744 non-null  int64
 24   FLAG_CONT_MOBILE              48744 non-null  int64
 25   FLAG_PHONE                    48744 non-null  int64
 26   FLAG_EMAIL                    48744 non-null  int64
 27   OCCUPATION_TYPE               33139 non-null  object
 28   CNT_FAM_MEMBERS               48744 non-null  float64
 29   REGION_RATING_CLIENT          48744 non-null  int64
 30   REGION_RATING_CLIENT_W_CITY   48744 non-null  int64
 31   WEEKDAY_APPR_PROCESS_START    48744 non-null  object
 32   HOUR_APPR_PROCESS_START       48744 non-null  int64
 33   REG_REGION_NOT_LIVE_REGION    48744 non-null  int64
 34   REG_REGION_NOT_WORK_REGION    48744 non-null  int64
 35   LIVE_REGION_NOT_WORK_REGION   48744 non-null  int64
 36   REG_CITY_NOT_LIVE_CITY        48744 non-null  int64
 37   REG_CITY_NOT_WORK_CITY        48744 non-null  int64
 38   LIVE_CITY_NOT_WORK_CITY       48744 non-null  int64
 39   ORGANIZATION_TYPE             48744 non-null  object
 40   EXT_SOURCE_1                  28212 non-null  float64
 41   EXT_SOURCE_2                  48736 non-null  float64
 42   EXT_SOURCE_3                  40076 non-null  float64
 43   APARTMENTS_AVG                24857 non-null  float64
 44   BASEMENTAREA_AVG              21103 non-null  float64
 45   YEARS_BEGINEXPLUATATION_AVG   25888 non-null  float64
 46   YEARS_BUILD_AVG               16926 non-null  float64
 47   COMMONAREA_AVG                15249 non-null  float64
 48   ELEVATORS_AVG                 23555 non-null  float64
 49   ENTRANCES_AVG                 25165 non-null  float64
 50   FLOORSMAX_AVG                 25423 non-null  float64
```

## Observation:

Similar to the application train dataset, the test dataset also have negative values for the same features, which is unexpected for certain features.

```
 58   BASEMENTAREA_MODE             21103 non-null  float64
```

```
dataset_summary(datasets['bureau'], 'bureau')
```

```
Summary of the dataset 'bureau':

Basic Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1716428 entries, 0 to 1716427
Data columns (total 17 columns):
 #   Column                  Non-Null Count    Dtype
---  ------                  --------------    -----
 0   SK_ID_CURR              1716428 non-null  int64
 1   SK_ID_BUREAU            1716428 non-null  int64
 2   CREDIT_ACTIVE           1716428 non-null  object
 3   CREDIT_CURRENCY         1716428 non-null  object
```

```
 4   DAYS_CREDIT              1716428 non-null  int64
 5   CREDIT_DAY_OVERDUE       1716428 non-null  int64
 6   DAYS_CREDIT_ENDDATE      1610875 non-null  float64
 7   DAYS_ENDDATE_FACT        1082775 non-null  float64
 8   AMT_CREDIT_MAX_OVERDUE  591940 non-null   float64
 9   CNT_CREDIT_PROLONG       1716428 non-null  int64
 10  AMT_CREDIT_SUM           1716415 non-null  float64
 11  AMT_CREDIT_SUM_DEBT      1458759 non-null  float64
 12  AMT_CREDIT_SUM_LIMIT     1124648 non-null  float64
 13  AMT_CREDIT_SUM_OVERDUE   1716428 non-null  float64
 14  CREDIT_TYPE              1716428 non-null  object
 15  DAYS_CREDIT_UPDATE       1716428 non-null  int64
 16  AMT_ANNUITY              489637 non-null   float64
dtypes: float64(8), int64(6), object(3)
memory usage: 222.6+ MB
None

Description of the dataset bureau:
```

| | SK_ID_CURR | SK_ID_BUREAU | DAYS_CREDIT | CREDIT_DAY_OVERDUE | DAYS_CREDIT_ |
|---|---|---|---|---|---|
| **count** | 1716428.0000 | 1716428.0000 | 1716428.0000 | 1716428.0000 | 1610 |
| **mean** | 278214.9300 | 5924434.4900 | -1142.1100 | 0.8200 | |
| **std** | 102938.5600 | 532265.7300 | 795.1600 | 36.5400 | 4 |
| **min** | 100001.0000 | 5000000.0000 | -2922.0000 | 0.0000 | -42 |
| **25%** | 188866.7500 | 5463953.7500 | -1666.0000 | 0.0000 | -1 |
| **50%** | 278055.0000 | 5926303.5000 | -987.0000 | 0.0000 | - |
| **75%** | 367426.0000 | 6385681.2500 | -474.0000 | 0.0000 | |
| **max** | 456255.0000 | 6843457.0000 | 0.0000 | 2792.0000 | 31 |

```
None

Data Types feature counts:
 float64    8
int64      6
object     3
dtype: int64

Dataframe Shape:  (1716428, 17)

Unique elements count in each object:
CREDIT_ACTIVE       4
CREDIT_CURRENCY     4
CREDIT_TYPE        15
dtype: int64
```

```
List of Categorical and Numerical(int + float) features  of bureau:
------------------------------
{'int64': Index(['SK_ID_CURR', 'SK_ID_BUREAU', 'DAYS_CREDIT', 'CREDIT_DAY_OVE
       'CNT_CREDIT_PROLONG', 'DAYS_CREDIT_UPDATE'],
      dtype='object')}
------------------------------
{'float64': Index(['DAYS_CREDIT_ENDDATE', 'DAYS_ENDDATE_FACT', 'AMT_CREDIT_MA
       'AMT_CREDIT_SUM', 'AMT_CREDIT_SUM_DEBT', 'AMT_CREDIT_SUM_LIMIT',
       'AMT_CREDIT_SUM_OVERDUE', 'AMT_ANNUITY'],
      dtype='object')}
------------------------------
{'object': Index(['CREDIT_ACTIVE', 'CREDIT_CURRENCY', 'CREDIT_TYPE'], dtype='
```

```
dataset_summary(datasets['bureau_balance'], 'bureau_balance')
```

```
Summary of the dataset 'bureau_balance':

Basic Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 27299925 entries, 0 to 27299924
Data columns (total 3 columns):
 #   Column          Non-Null Count     Dtype
---  ------          --------------     -----
 0   SK_ID_BUREAU    27299925 non-null  int64
 1   MONTHS_BALANCE  27299925 non-null  int64
 2   STATUS          27299925 non-null  object
dtypes: int64(2), object(1)
memory usage: 624.8+ MB
None

Description of the dataset bureau_balance:
```

| | SK_ID_BUREAU | MONTHS_BALANCE |
|---|---|---|
| **count** | 27299925.0000 | 27299925.0000 |
| **mean** | 6036297.3300 | -30.7400 |
| **std** | 492348.8600 | 23.8600 |
| **min** | 5001709.0000 | -96.0000 |
| **25%** | 5730933.0000 | -46.0000 |
| **50%** | 6070821.0000 | -25.0000 |
| **75%** | 6431951.0000 | -11.0000 |

| | | |
|---|---|---|
| **max** | 6842888.0000 | 0.0000 |

```
None

Data Types feature counts:
 int64     2
object    1
dtype: int64

Dataframe Shape:  (27299925, 3)

Unique elements count in each object:
STATUS    8
dtype: int64


List of Categorical and Numerical(int + float) features  of bureau_balance:

{'int64': Index(['SK_ID_BUREAU', 'MONTHS_BALANCE'], dtype='object')}

{'object': Index(['STATUS'], dtype='object')}

        'LIVINGAPARTMENTS_AVG',  'LIVINGAREA_AVG',  'NONLIVINGAPARTMENTS_AVG'
```

```
dataset_summary(datasets['credit_card_balance'], 'credit_card_balance')
```

```
Summary of the dataset 'credit_card_balance':

Basic Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3840312 entries, 0 to 3840311
Data columns (total 23 columns):
 #   Column                      Non-Null Count    Dtype
---  ------                      --------------    -----
 0   SK_ID_PREV                  3840312 non-null  int64
 1   SK_ID_CURR                  3840312 non-null  int64
 2   MONTHS_BALANCE              3840312 non-null  int64
 3   AMT_BALANCE                 3840312 non-null  float64
 4   AMT_CREDIT_LIMIT_ACTUAL     3840312 non-null  int64
 5   AMT_DRAWINGS_ATM_CURRENT    3090496 non-null  float64
 6   AMT_DRAWINGS_CURRENT        3840312 non-null  float64
 7   AMT_DRAWINGS_OTHER_CURRENT  3090496 non-null  float64
 8   AMT_DRAWINGS_POS_CURRENT    3090496 non-null  float64
 9   AMT_INST_MIN_REGULARITY     3535076 non-null  float64
 10  AMT_PAYMENT_CURRENT         3072324 non-null  float64
 11  AMT_PAYMENT_TOTAL_CURRENT   3840312 non-null  float64
 12  AMT_RECEIVABLE_PRINCIPAL    3840312 non-null  float64
 13  AMT_RECIVABLE               3840312 non-null  float64
 14  AMT_TOTAL_RECEIVABLE        3840312 non-null  float64
 15  CNT_DRAWINGS_ATM_CURRENT    3090496 non-null  float64
```

```
 16  CNT_DRAWINGS_CURRENT         3840312 non-null  int64
 17  CNT_DRAWINGS_OTHER_CURRENT  3090496 non-null  float64
 18  CNT_DRAWINGS_POS_CURRENT    3090496 non-null  float64
 19  CNT_INSTALMENT_MATURE_CUM   3535076 non-null  float64
 20  NAME_CONTRACT_STATUS        3840312 non-null  object
 21  SK_DPD                      3840312 non-null  int64
 22  SK_DPD_DEF                  3840312 non-null  int64
dtypes: float64(15), int64(7), object(1)
memory usage: 673.9+ MB
None

Description of the dataset credit_card_balance:
```

| | SK_ID_PREV | SK_ID_CURR | MONTHS_BALANCE | AMT_BALANCE | AMT_CREDIT_LIMIT_ |
|---|---|---|---|---|---|
| **count** | 3840312.0000 | 3840312.0000 | 3840312.0000 | 3840312.0000 | 38403 |
| **mean** | 1904503.5900 | 278324.2100 | -34.5200 | 58300.1600 | 1538 |
| **std** | 536469.4700 | 102704.4800 | 26.6700 | 106307.0300 | 1651 |
| **min** | 1000018.0000 | 100006.0000 | -96.0000 | -420250.1800 | |
| **25%** | 1434385.0000 | 189517.0000 | -55.0000 | 0.0000 | 450 |
| **50%** | 1897122.0000 | 278396.0000 | -28.0000 | 0.0000 | 1125 |
| **75%** | 2369327.7500 | 367580.0000 | -11.0000 | 89046.6900 | 1800 |
| **max** | 2843496.0000 | 456250.0000 | -1.0000 | 1505902.1800 | 13500 |

```
None

Data Types feature counts:
 float64    15
int64       7
object      1
```

```
dataset_summary(datasets['installments_payments'], 'installments_payments')
```

```
Summary of the dataset 'installments_payments':

Basic Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 13605401 entries, 0 to 13605400
Data columns (total 8 columns):
 #   Column                  Non-Null Count     Dtype
---  ------                  --------------     -----
 0   SK_ID_PREV              13605401 non-null  int64
 1   SK_ID_CURR              13605401 non-null  int64
 2   NUM_INSTALMENT_VERSION  13605401 non-null  float64
 3   NUM_INSTALMENT_NUMBER   13605401 non-null  int64
 4   DAYS_INSTALMENT         13605401 non-null  float64
```

```
 5   DAYS_ENTRY_PAYMENT      13602496 non-null  float64
 6   AMT_INSTALMENT          13605401 non-null  float64
 7   AMT_PAYMENT             13602496 non-null  float64
dtypes: float64(5), int64(3)
memory usage: 830.4 MB
None

Description of the dataset installments_payments:
```

| | SK_ID_PREV | SK_ID_CURR | NUM_INSTALMENT_VERSION | NUM_INSTALMENT_NUMBE |
|---|---|---|---|---|
| **count** | 13605401.0000 | 13605401.0000 | 13605401.0000 | 13605401.000 |
| **mean** | 1903364.9700 | 278444.8800 | 0.8600 | 18.870 |
| **std** | 536202.9100 | 102718.3100 | 1.0400 | 26.660 |
| **min** | 1000001.0000 | 100001.0000 | 0.0000 | 1.000 |
| **25%** | 1434191.0000 | 189639.0000 | 0.0000 | 4.000 |
| **50%** | 1896520.0000 | 278685.0000 | 1.0000 | 8.000 |
| **75%** | 2369094.0000 | 367530.0000 | 1.0000 | 19.000 |
| **max** | 2843499.0000 | 456255.0000 | 178.0000 | 277.000 |

```
None

Data Types feature counts:
 float64    5
int64      3
dtype: int64

Dataframe Shape:  (13605401, 8)

Unique elements count in each object:
Series([], dtype: float64)


List of Categorical and Numerical(int + float) features  of installments_paym

{'int64': Index(['SK_ID_PREV', 'SK_ID_CURR', 'NUM_INSTALMENT_NUMBER'], dtype=

{'float64': Index(['NUM_INSTALMENT_VERSION', 'DAYS_INSTALMENT', 'DAYS_ENTRY_P
       'AMT_INSTALMENT', 'AMT_PAYMENT'],
      dtype='object')}
```

```
dataset_summary(datasets['previous_application'], 'previous_application')
```

```
Summary of the dataset 'previous_application':
```

```
Basic Info:
<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

Description of the dataset previous_application:
```

| | SK_ID_PREV | SK_ID_CURR | AMT_ANNUITY | AMT_APPLICATION | AMT_CREDIT | AMT_ |
|---|---|---|---|---|---|---|
| **count** | 1670214.0000 | 1670214.0000 | 1297979.0000 | 1670214.0000 | 1670213.0000 | |
| **mean** | 1923089.1400 | 278357.1700 | 15955.1200 | 175233.8600 | 196114.0200 | |
| **std** | 532597.9600 | 102814.8200 | 14782.1400 | 292779.7600 | 318574.6200 | |
| **min** | 1000001.0000 | 100001.0000 | 0.0000 | 0.0000 | 0.0000 | |

```
dataset_summary(datasets['POS_CASH_balance'], 'POS_CASH_balance')
```

```
Summary of the dataset 'POS_CASH_balance':

Basic Info:
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 10001358 entries, 0 to 10001357
Data columns (total 8 columns):
 #   Column                 Non-Null Count     Dtype  
---  ------                 --------------     -----  
 0   SK_ID_PREV             10001358 non-null  int64  
 1   SK_ID_CURR             10001358 non-null  int64  
 2   MONTHS_BALANCE         10001358 non-null  int64  
 3   CNT_INSTALMENT         9975287 non-null   float64
 4   CNT_INSTALMENT_FUTURE  9975271 non-null   float64
 5   NAME_CONTRACT_STATUS   10001358 non-null  object 
 6   SK_DPD                 10001358 non-null  int64  
 7   SK_DPD_DEF             10001358 non-null  int64  
dtypes: float64(2), int64(5), object(1)
memory usage: 610.4+ MB
None

Description of the dataset POS_CASH_balance:
```

| | SK_ID_PREV | SK_ID_CURR | MONTHS_BALANCE | CNT_INSTALMENT | CNT_INSTALME |
|---|---|---|---|---|---|
| **count** | 10001358.0000 | 10001358.0000 | 10001358.0000 | 9975287.0000 | 99 |
| **mean** | 1903216.6000 | 278403.8600 | -35.0100 | 17.0900 | |
| **std** | 535846.5300 | 102763.7500 | 26.0700 | 12.0000 | |
| **min** | 1000001.0000 | 100001.0000 | -96.0000 | 1.0000 | |
| **25%** | 1434405.0000 | 189550.0000 | -54.0000 | 10.0000 | |
| **50%** | 1896565.0000 | 278654.0000 | -28.0000 | 12.0000 | |
| **75%** | 2368963.0000 | 367429.0000 | -13.0000 | 24.0000 | |
| **max** | 2843499.0000 | 456255.0000 | -1.0000 | 92.0000 | |

```
None
```

```
Data Types feature counts:
 int64      5
float64    2
object     1
dtype: int64

Dataframe Shape:  (10001358, 8)

Unique elements count in each object:
NAME_CONTRACT_STATUS    9
dtype: int64


List of Categorical and Numerical(int + float) features  of POS_CASH_balance:

{'int64': Index(['SK_ID_PREV', 'SK_ID_CURR', 'MONTHS_BALANCE', 'SK_DPD', 'SK_

{'float64': Index(['CNT_INSTALMENT', 'CNT_INSTALMENT_FUTURE'], dtype='object'
```

## Missing data for application train

```
def missing_data_plot(df, name):
    percent = (df.isnull().sum()/df.isnull().count()*100).sort_values(ascending =
    missing_count = df.isna().sum().sort_values(ascending = False)
    missing_data  = pd.concat([percent, missing_count], axis=1, keys=['Percent',
    missing_data=missing_data[missing_data['Percent'] > 0]
    print(f"\n The summary of the missing data in the dataset '{name}': \n")
    if len(missing_data)==0:
      print("No missing Data")
    else:
      display(HTML(missing_data.to_html()))  # display all the rows
      return missing_data
```

```
app_train_missing_data = missing_data_plot(datasets['application_train'], 'applic
```

```
app_test_missing_data = missing_data_plot(datasets['application_test'], 'applicat
```

```
 The summary of the missing data in the dataset 'application_test':
```

| | Percent | Train Missing Count |
|---|---|---|
| **COMMONAREA_AVG** | 68.7200 | 33495 |

| | | |
|---|---|---|
| **COMMONAREA_MODE** | 68.7200 | 33495 |
| **COMMONAREA_MEDI** | 68.7200 | 33495 |
| **NONLIVINGAPARTMENTS_AVG** | 68.4100 | 33347 |
| **NONLIVINGAPARTMENTS_MODE** | 68.4100 | 33347 |
| **NONLIVINGAPARTMENTS_MEDI** | 68.4100 | 33347 |
| **FONDKAPREMONT_MODE** | 67.2800 | 32797 |
| **LIVINGAPARTMENTS_AVG** | 67.2500 | 32780 |
| **LIVINGAPARTMENTS_MODE** | 67.2500 | 32780 |
| **LIVINGAPARTMENTS_MEDI** | 67.2500 | 32780 |
| **FLOORSMIN_MEDI** | 66.6100 | 32466 |
| **FLOORSMIN_AVG** | 66.6100 | 32466 |
| **FLOORSMIN_MODE** | 66.6100 | 32466 |
| **OWN_CAR_AGE** | 66.2900 | 32312 |
| **YEARS_BUILD_AVG** | 65.2800 | 31818 |
| **YEARS_BUILD_MEDI** | 65.2800 | 31818 |
| **YEARS_BUILD_MODE** | 65.2800 | 31818 |
| **LANDAREA_MEDI** | 57.9600 | 28254 |
| **LANDAREA_AVG** | 57.9600 | 28254 |
| **LANDAREA_MODE** | 57.9600 | 28254 |
| **BASEMENTAREA_MEDI** | 56.7100 | 27641 |
| **BASEMENTAREA_AVG** | 56.7100 | 27641 |
| **BASEMENTAREA_MODE** | 56.7100 | 27641 |
| **NONLIVINGAREA_AVG** | 53.5100 | 26084 |
| **NONLIVINGAREA_MODE** | 53.5100 | 26084 |
| **NONLIVINGAREA_MEDI** | 53.5100 | 26084 |
| **ELEVATORS_MODE** | 51.6800 | 25189 |
| **ELEVATORS_MEDI** | 51.6800 | 25189 |
| **ELEVATORS_AVG** | 51.6800 | 25189 |

| | | |
|---|---|---|
| **WALLSMATERIAL_MODE** | 49.0200 | 23893 |

```
bureau_missing_data = missing_data_plot(datasets['bureau'], 'bureau')
```

The summary of the missing data in the dataset 'bureau':

| | Percent | Train Missing Count |
|---|---|---|
| **AMT_ANNUITY** | 71.4700 | 1226791 |
| **AMT_CREDIT_MAX_OVERDUE** | 65.5100 | 1124488 |
| **DAYS_ENDDATE_FACT** | 36.9200 | 633653 |
| **AMT_CREDIT_SUM_LIMIT** | 34.4800 | 591780 |
| **AMT_CREDIT_SUM_DEBT** | 15.0100 | 257669 |
| **DAYS_CREDIT_ENDDATE** | 6.1500 | 105553 |
| **FLOORSMAX_AVG** | 47.8400 | 23321 |

```
bureau_balance_missing_data = missing_data_plot(datasets['bureau_balance'], 'bure
```

The summary of the missing data in the dataset 'bureau_balance':

No missing Data

| | | |
|---|---|---|
| **YEARS_BEGINEXPLUATATION_MODE** | 46.8900 | 22856 |
| **TOTALAREA_MODE** | 46.4100 | 22624 |
| **EMERGENCYSTATE_MODE** | 45.5600 | 22209 |
| **EXT_SOURCE_1** | 42.1200 | 20532 |
| **OCCUPATION_TYPE** | 32.0100 | 15605 |
| **EXT_SOURCE_3** | 17.7800 | 8668 |
| **AMT_REQ_CREDIT_BUREAU_DAY** | 12.4100 | 6049 |
| **AMT_REQ_CREDIT_BUREAU_WEEK** | 12.4100 | 6049 |
| **AMT_REQ_CREDIT_BUREAU_HOUR** | 12.4100 | 6049 |
| **AMT_REQ_CREDIT_BUREAU_MON** | 12.4100 | 6049 |
| **AMT_REQ_CREDIT_BUREAU_QRT** | 12.4100 | 6049 |
| **AMT_REQ_CREDIT_BUREAU_YEAR** | 12.4100 | 6049 |

```
credit_card_balance_missing_data = missing_data_plot(datasets['credit_card_balanc
```

The summary of the missing data in the dataset 'credit_card_balance':

| | Percent | Train Missing Count |
|---|---|---|
| **AMT_PAYMENT_CURRENT** | 20.0000 | 767988 |
| **AMT_DRAWINGS_ATM_CURRENT** | 19.5200 | 749816 |
| **CNT_DRAWINGS_POS_CURRENT** | 19.5200 | 749816 |
| **AMT_DRAWINGS_OTHER_CURRENT** | 19.5200 | 749816 |
| **AMT_DRAWINGS_POS_CURRENT** | 19.5200 | 749816 |
| **CNT_DRAWINGS_OTHER_CURRENT** | 19.5200 | 749816 |
| **CNT_DRAWINGS_ATM_CURRENT** | 19.5200 | 749816 |
| **CNT_INSTALMENT_MATURE_CUM** | 7.9500 | 305236 |
| **AMT_INST_MIN_REGULARITY** | 7.9500 | 305236 |

```
installments_payments_missing_data = missing_data_plot(datasets['installments_pay
```

The summary of the missing data in the dataset 'installments_payments':

| | Percent | Train Missing Count |
|---|---|---|
| **DAYS_ENTRY_PAYMENT** | 0.0200 | 2905 |
| **AMT_PAYMENT** | 0.0200 | 2905 |

```
prev_app_missing_data = missing_data_plot(datasets['previous_application'], 'prev
```

The summary of the missing data in the dataset 'previous_application':

| | Percent | Train Missing Count |
|---|---|---|
| **RATE_INTEREST_PRIVILEGED** | 99.6400 | 1664263 |
| **RATE_INTEREST_PRIMARY** | 99.6400 | 1664263 |
| **AMT_DOWN_PAYMENT** | 53.6400 | 895844 |
| **RATE_DOWN_PAYMENT** | 53.6400 | 895844 |
| **NAME_TYPE_SUITE** | 49.1200 | 820405 |
| **NFLAG_INSURED_ON_APPROVAL** | 40.3000 | 673065 |
| **DAYS_TERMINATION** | 40.3000 | 673065 |
| **DAYS_LAST_DUE** | 40.3000 | 673065 |
| **DAYS_LAST_DUE_1ST_VERSION** | 40.3000 | 673065 |
| **DAYS_FIRST_DUE** | 40.3000 | 673065 |
| **DAYS_FIRST_DRAWING** | 40.3000 | 673065 |
| **AMT_GOODS_PRICE** | 23.0800 | 385515 |
| **AMT_ANNUITY** | 22.2900 | 372235 |
| **CNT_PAYMENT** | 22.2900 | 372230 |
| **PRODUCT_COMBINATION** | 0.0200 | 346 |

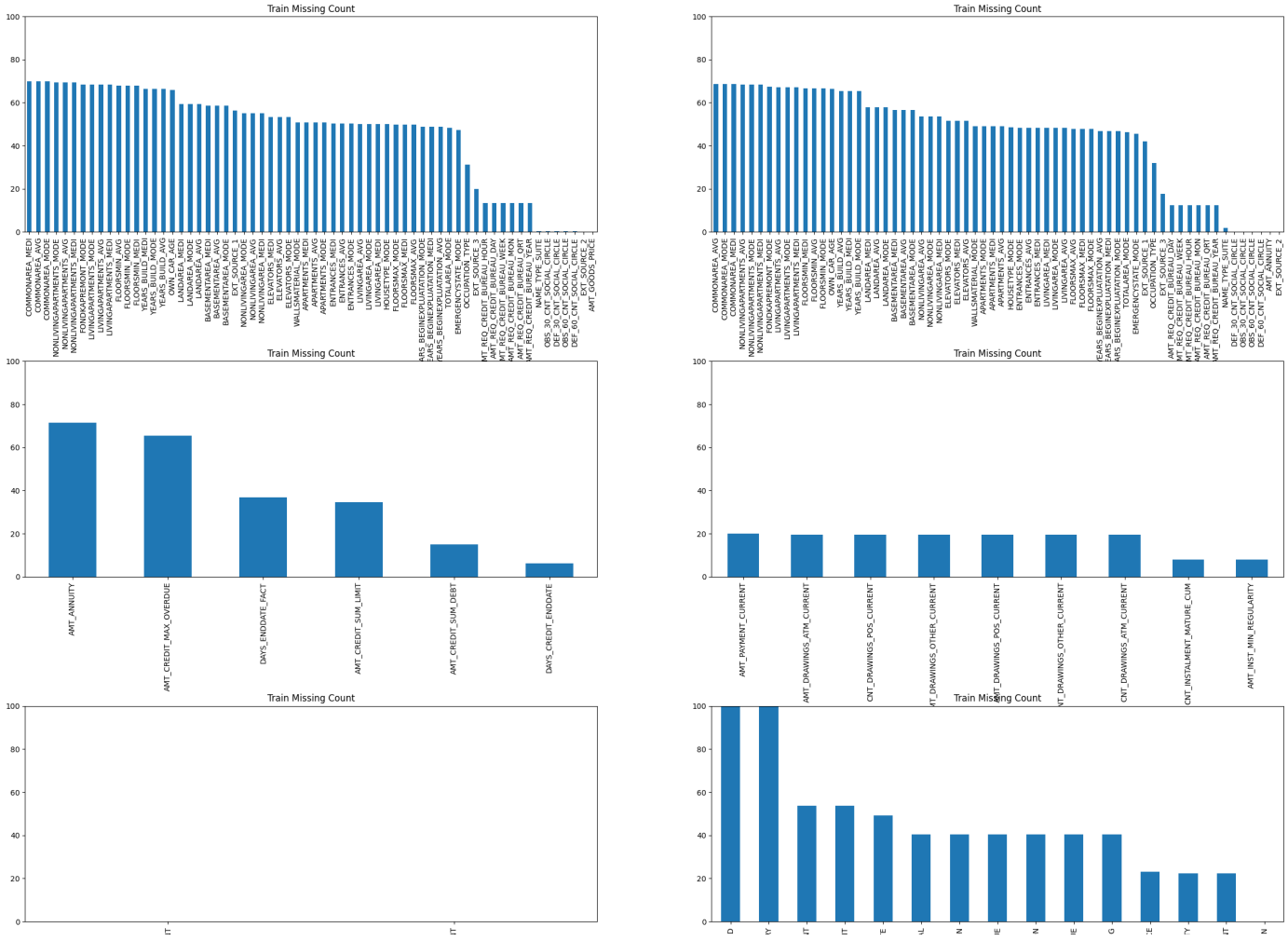
```
pos_cash_bal_missing_data = missing_data_plot(datasets['POS_CASH_balance'], 'POS_
```

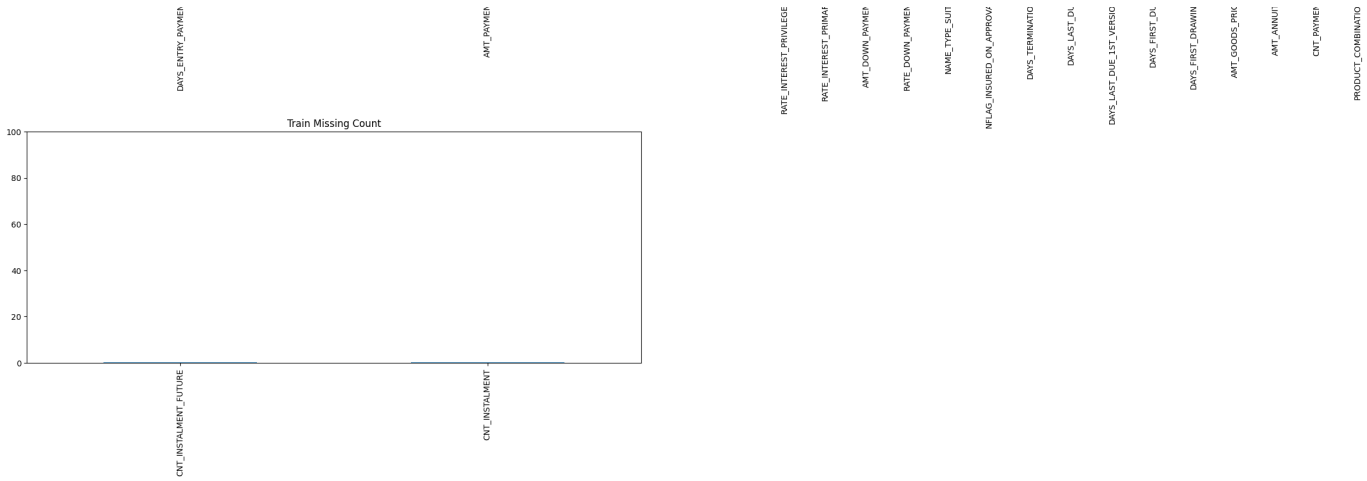
The summary of the missing data in the dataset 'POS_CASH_balance':

| | Percent | Train Missing Count |
|---|---|---|
| **CNT_INSTALMENT_FUTURE** | 0.2600 | 26087 |
| **CNT_INSTALMENT** | 0.2600 | 26071 |

```
#Plot to visualize the number of missing features in each table
```

```
#Plot to visualize the number of missing features in each table
fig = plt.figure(figsize=(30,30))
plot1 = fig.add_subplot(4,2,1)
plot2 = fig.add_subplot(4,2,2)
plot3 = fig.add_subplot(4,2,3)
plot4 = fig.add_subplot(4,2,4)
plot5 = fig.add_subplot(4,2,5)
plot6 = fig.add_subplot(4,2,6)
plot7 = fig.add_subplot(4,2,7)

axes = [plot1, plot2, plot3, plot4, plot5, plot6, plot7]

all_df = [app_train_missing_data, app_test_missing_data, bureau_missing_data,
          credit_card_balance_missing_data, installments_payments_missing_data,
          prev_app_missing_data, pos_cash_bal_missing_data]
for i in range(7):
    df = all_df[i]
    df.loc[(df.Percent > 0.0),'Percent'].plot(kind='bar',ax=axes[i],title=df.colu
    plt.subplots_adjust(hspace=0.6)
plt.show()
```



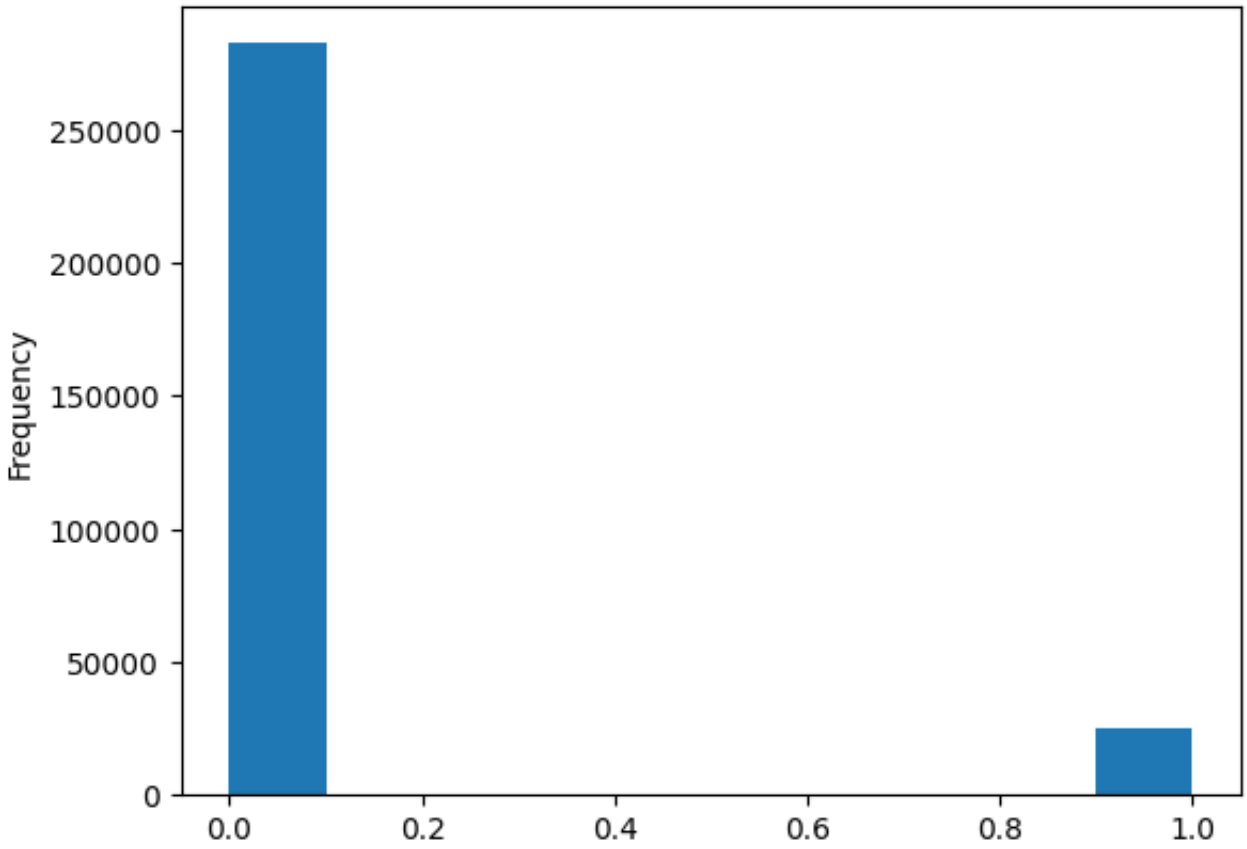



## Observation

The missing data in HCDR was found to be present in several features across the different tables, with some tables having a higher percentage of missing data than others. The missing data in application train dataset ranged from 0.0% to 69.9%

## Distribution of the target column

```
import matplotlib.pyplot as plt
%matplotlib inline

datasets["application_train"]['TARGET'].astype(int).plot.hist();
```




## Correlation with the target column

```
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               0.0443
REG_CITY_NOT_LIVE_CITY         0.0444
FLAG_EMP_PHONE                 0.0460
REG_CITY_NOT_WORK_CITY         0.0510
DAYS_ID_PUBLISH                0.0515
DAYS_LAST_PHONE_CHANGE         0.0552
REGION_RATING_CLIENT           0.0589
REGION_RATING_CLIENT_W_CITY    0.0609
DAYS_BIRTH                     0.0782
TARGET                         1.0000
Name: TARGET, dtype: float64

Most Negative Correlations:
 EXT_SOURCE_3                 -0.1789
EXT_SOURCE_2                  -0.1605
EXT_SOURCE_1                  -0.1553
DAYS_EMPLOYED                 -0.0449
FLOORSMAX_AVG                 -0.0440
FLOORSMAX_MEDI                -0.0438
FLOORSMAX_MODE                -0.0432
AMT_GOODS_PRICE               -0.0396
REGION_POPULATION_RELATIVE    -0.0372
ELEVATORS_AVG                 -0.0342
Name: TARGET, dtype: float64
```

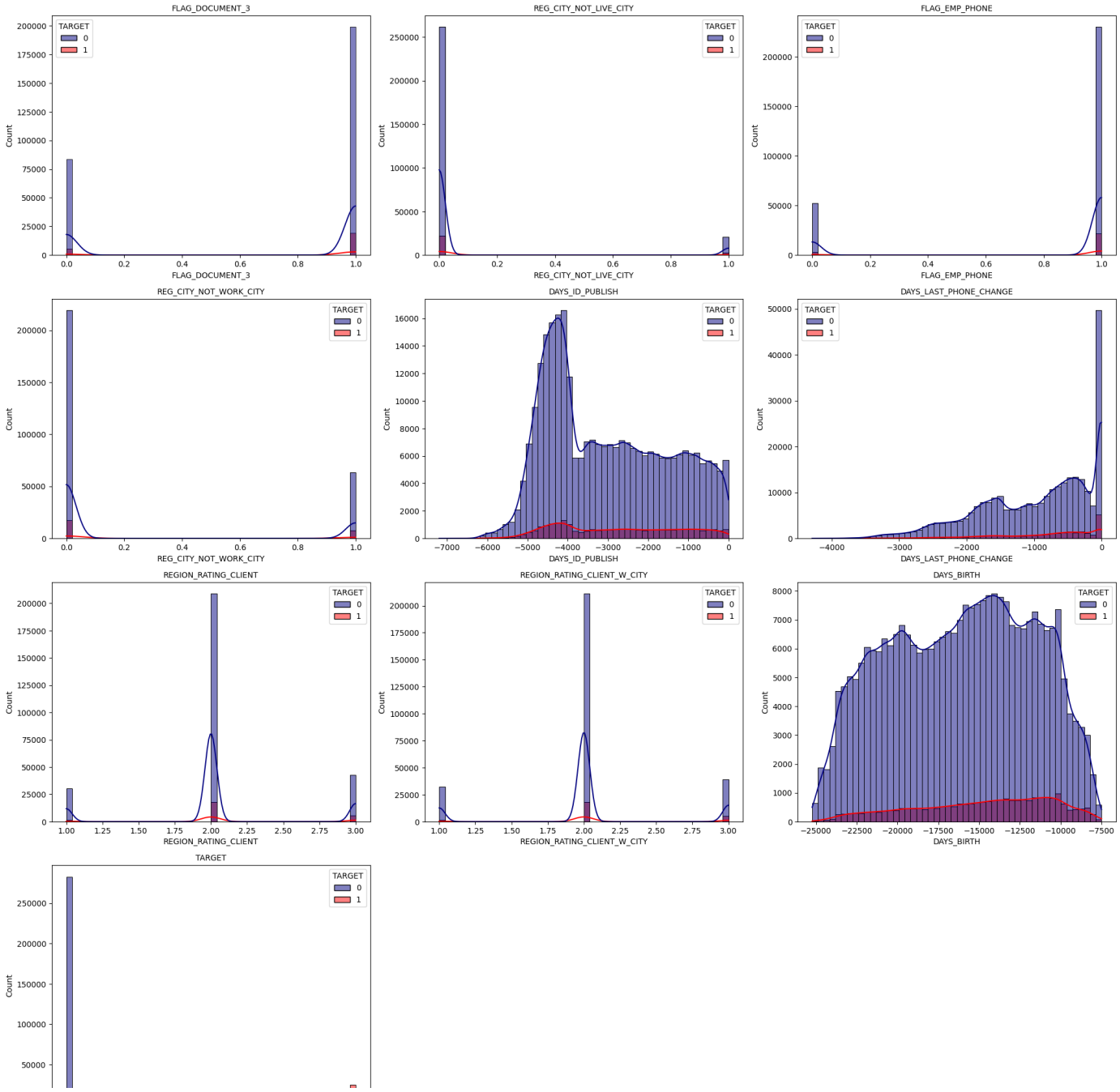
```
correlations
```

```
EXT_SOURCE_3                  -0.1789
EXT_SOURCE_2                  -0.1605
EXT_SOURCE_1                  -0.1553
DAYS_EMPLOYED                 -0.0449
FLOORSMAX_AVG                 -0.0440
FLOORSMAX_MEDI                -0.0438
FLOORSMAX_MODE                -0.0432
AMT_GOODS_PRICE               -0.0396
REGION_POPULATION_RELATIVE    -0.0372
ELEVATORS_AVG                 -0.0342
ELEVATORS_MEDI                -0.0339
FLOORSMIN_AVG                 -0.0336
FLOORSMIN_MEDI                -0.0334
LIVINGAREA_AVG                -0.0330
LIVINGAREA_MEDI               -0.0327
FLOORSMIN_MODE                -0.0327
TOTALAREA_MODE                -0.0326
```

```
TOTALAREA_MODE                -0.0326
ELEVATORS_MODE                -0.0321
LIVINGAREA_MODE               -0.0307
AMT_CREDIT                    -0.0304
APARTMENTS_AVG                -0.0295
APARTMENTS_MEDI               -0.0292
FLAG_DOCUMENT_6               -0.0286
APARTMENTS_MODE               -0.0273
LIVINGAPARTMENTS_AVG          -0.0250
LIVINGAPARTMENTS_MEDI         -0.0246
HOUR_APPR_PROCESS_START       -0.0242
FLAG_PHONE                    -0.0238
LIVINGAPARTMENTS_MODE         -0.0234
BASEMENTAREA_AVG              -0.0227
YEARS_BUILD_MEDI              -0.0223
YEARS_BUILD_AVG               -0.0221
BASEMENTAREA_MEDI             -0.0221
YEARS_BUILD_MODE              -0.0221
BASEMENTAREA_MODE             -0.0200
ENTRANCES_AVG                 -0.0192
ENTRANCES_MEDI                -0.0190
COMMONAREA_MEDI               -0.0186
COMMONAREA_AVG                -0.0185
ENTRANCES_MODE                -0.0174
COMMONAREA_MODE               -0.0163
NONLIVINGAREA_AVG             -0.0136
NONLIVINGAREA_MEDI            -0.0133
AMT_ANNUITY                   -0.0128
NONLIVINGAREA_MODE            -0.0127
AMT_REQ_CREDIT_BUREAU_MON     -0.0125
FLAG_DOCUMENT_16              -0.0116
FLAG_DOCUMENT_13              -0.0116
LANDAREA_MEDI                 -0.0113
LANDAREA_AVG                  -0.0109
LANDAREA_MODE                 -0.0102
YEARS_BEGINEXPLUATATION_MEDI  -0.0100
YEARS_BEGINEXPLUATATION_AVG   -0.0097
FLAG_DOCUMENT_14              -0.0095
YEARS_BEGINEXPLUATATION_MODE  -0.0090
FLAG_DOCUMENT_8               -0.0080
FLAG_DOCUMENT_18              -0.0080
FLAG_DOCUMENT_15              -0.0065
FLAG_DOCUMENT_9               -0.0044
FLAG_DOCUMENT_11              -0.0042
```

```
pos_corr = correlations.tail(10).index.values
neg_corr = correlations.head(10).index.values
```

```
# Distribution of top 10 positive correlation variables with respect to TARGET
numVar = pos_corr.shape[0]

plt.figure(figsize=(20,50))
for i,var in enumerate(pos_corr):
    plt.subplot(numVar,3,i+1)
    sns.histplot(datasets['application_train'], x=var, hue="TARGET", kde=True, bi
    plt.subplots_adjust(hspace=0.50)
    plt.title(var, fontsize = 10)
    plt.tight_layout()
plt.show()
```

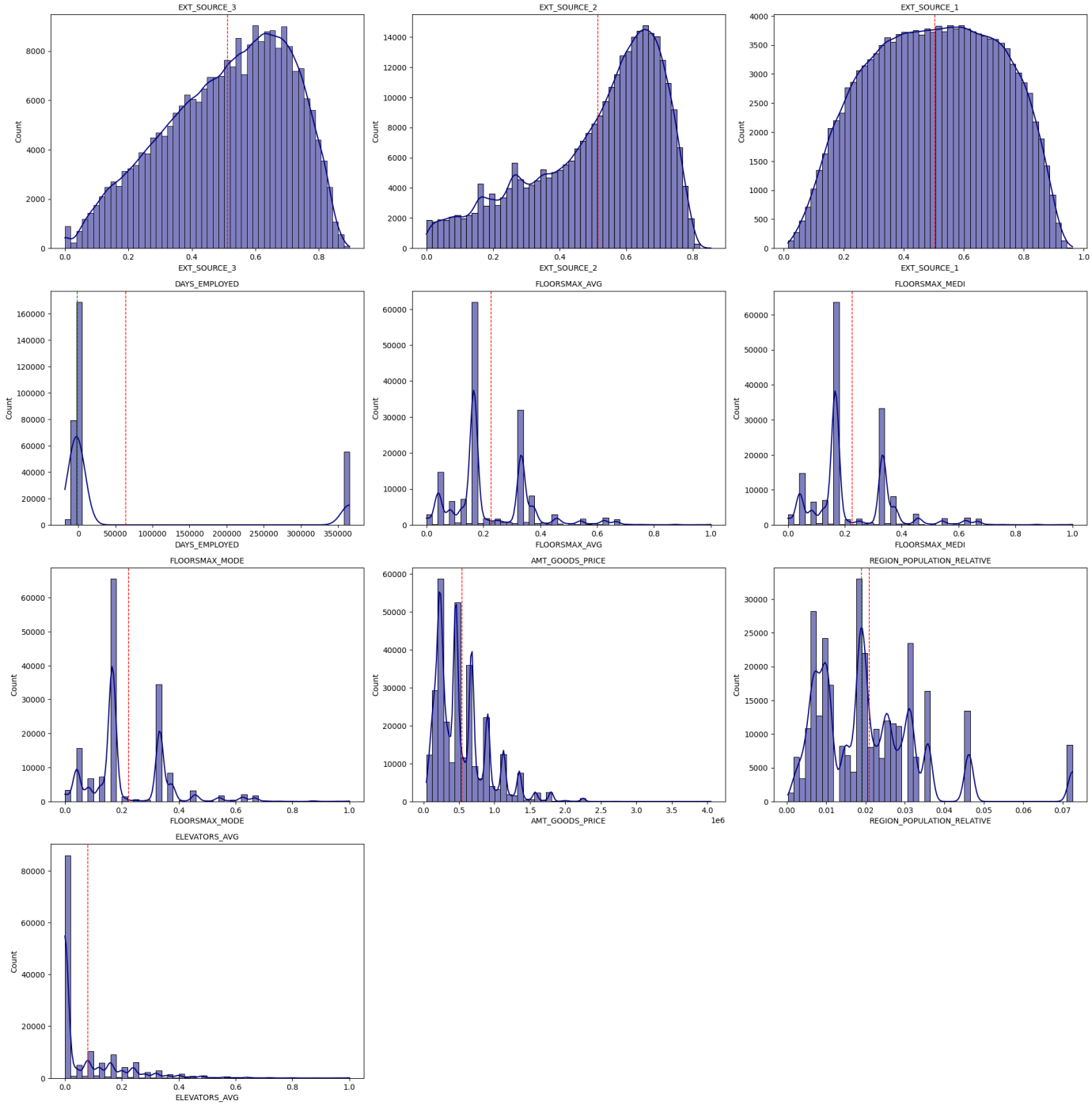



## Observation

Above plot shows us the histogram of the top 10 positively correlated values to the Target variable. The positive correlation distribution with target in HCDR dataset shows that the variables with the highest positive correlation with the target variable are the ones related to credit bureau and loan history, such as the number of days overdue on past credits, the number of previous loans, and the number of inquiries to the credit bureau.

```
# Distribution of top 10 negative correlation variables with target of HCDR
numVar = neg_corr.shape[0]

plt.figure(figsize=(20, 50))
for i, col in enumerate(neg_corr):
    defaulter = datasets["application_train"].loc[datasets["application_train"]['
    non_defaulter = datasets["application_train"].loc[datasets["application_train

    mu = np.mean(datasets['application_train'][col])
    median = np.median(datasets['application_train'][col])
    sigma = np.std(datasets['application_train'][col])

    plt.subplot(numVar, 3, i+1)
```

```
    plot = sns.histplot(data=datasets['application_train'][col], kde=True, bins=5
    plt.axvline(mu, color='red', linestyle='dashed', linewidth=1)
    plt.axvline(median, color='green', linestyle='dashed', linewidth=1)
    plt.subplots_adjust(hspace=0.50)
    plt.title(col, fontsize=10)
    plt.tight_layout()
plt.show()
```



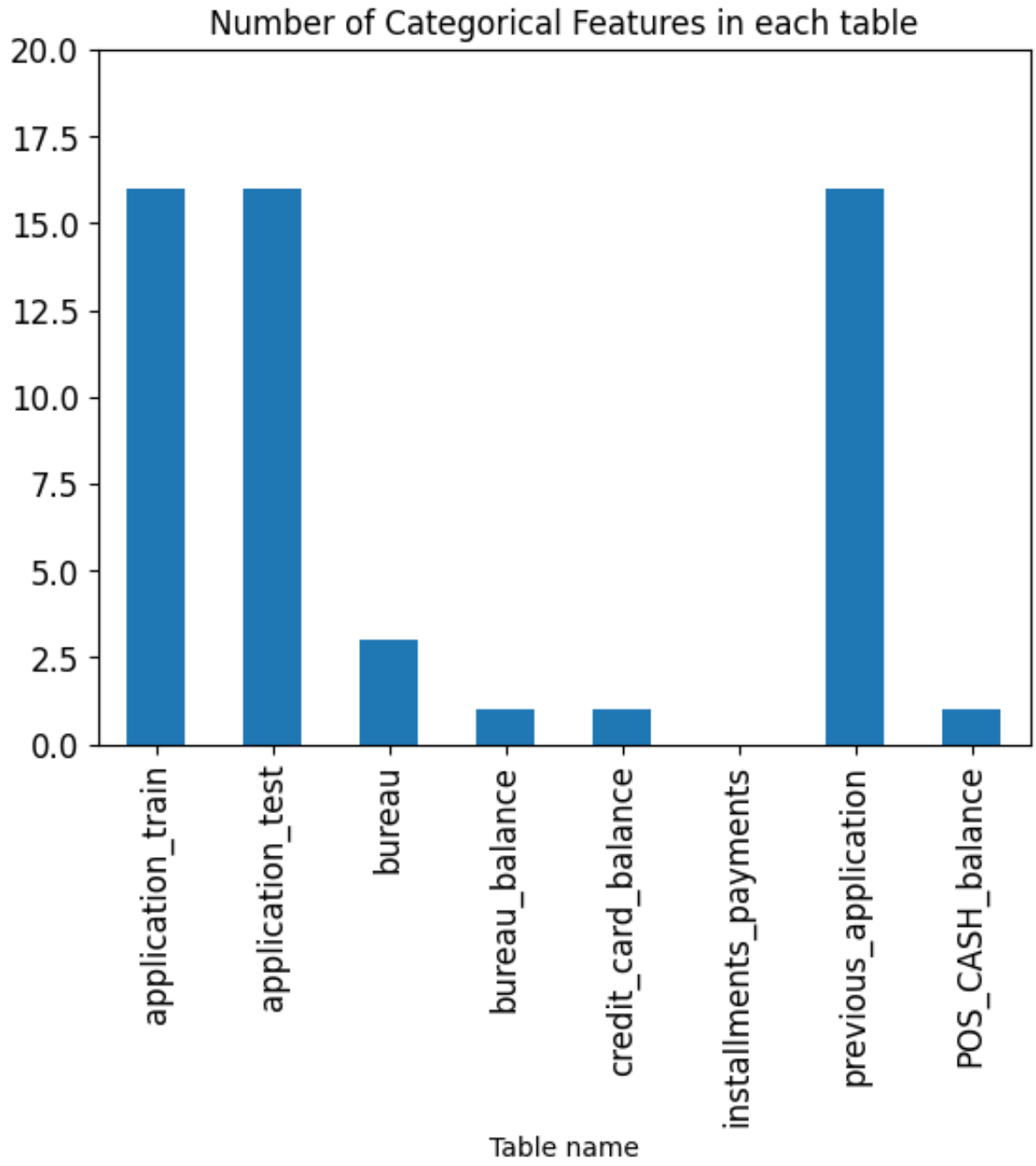
## Observation

Above plot shows us the histogram of the top 10 negatively correlated values to the Target variable. We have used the Kernel Density Estimation feature to visualize the probability distribution of the features. We can see that the distribution of 'EXT_SOURCE_2' and 'EXT_SOURCE_3' are right skewed, whereas 'EXT_SOURCE_1' approximately follows a normal distribution.

# Distribution of Numerical and Categorical Features in Application train

```
numerical_features_app = datasets['application_train'].select_dtypes(include = ['
categorical_features_app = datasets['application_train'].select_dtypes(include =
dtype_count = pd.DataFrame(index=ds_names,columns=['Categorical','Numerical'])
for ds_name in ds_names:
    categorical_features = datasets[ds_name].select_dtypes(include = ['object']).
    numerical_features = datasets[ds_name].select_dtypes(include = ['int64','floa
    dtype_count['Categorical'][ds_name] = len(categorical_features)
    dtype_count['Numerical'][ds_name] = len(numerical_features)
print(dtype_count)
dtype_count['Categorical'].plot(kind='bar',ylim = (0,20),fontsize=12,title='Numbe
```

```
                      Categorical Numerical
application_train              16       106
application_test               16       105
bureau                          3        14
bureau_balance                  1         2
credit_card_balance             1        22
installments_payments           0         8
previous_application            16        21
POS_CASH_balance                1         7
<Axes: title={'center': 'Number of Categorical Features in each table'},
xlabel='Table name'>
```

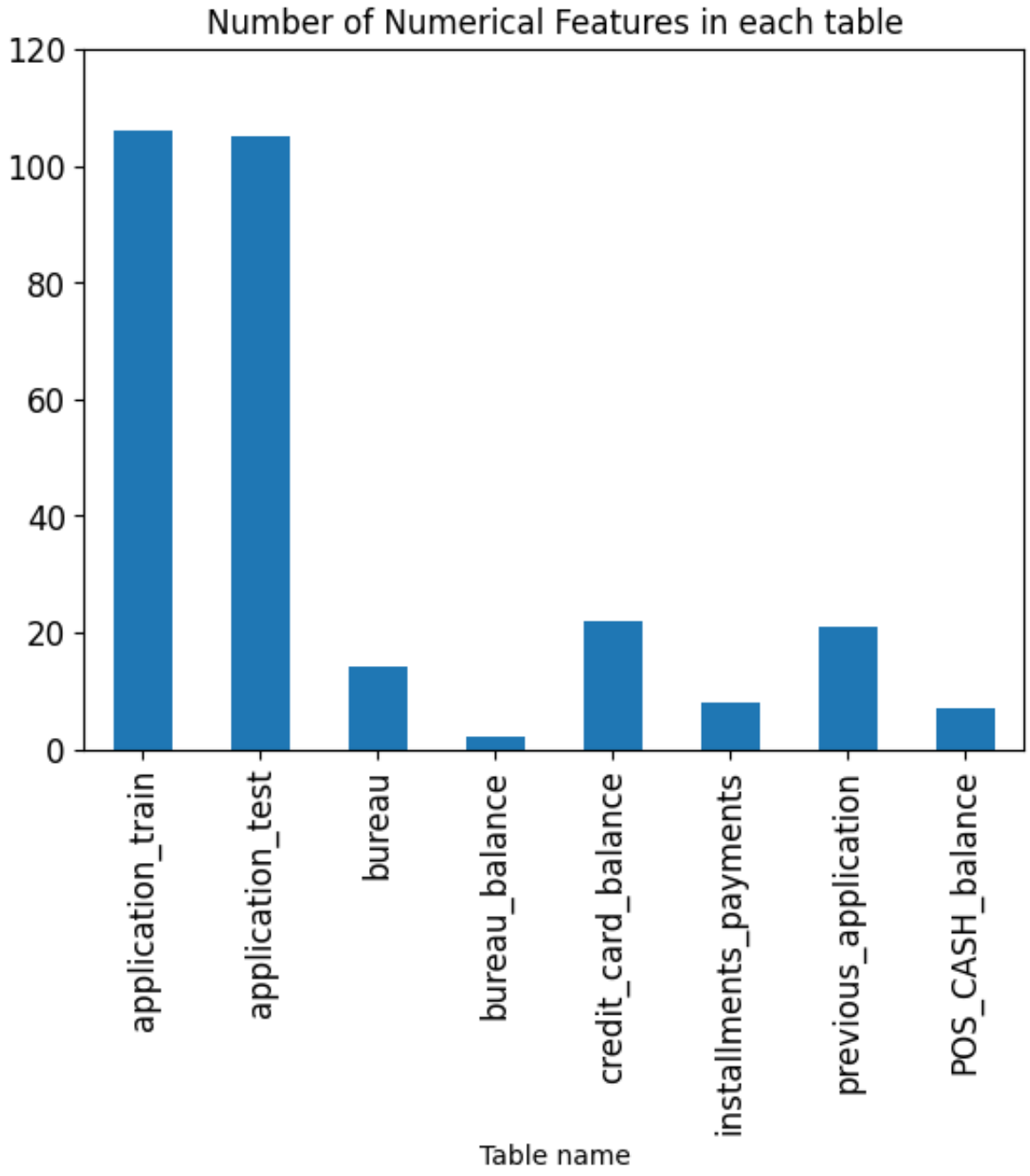


## Observation

We see that the application table and previous_application table have the maximum number (16) of Categorical Features. Hence we will have to use One Hot Encoding in order to pass the data to the learning algorithm

```
dtype_count['Numerical'].plot(kind='bar',ylim = (0,120),fontsize=12,title='Number
```

```
<Axes: title={'center': 'Number of Numerical Features in each table'},
xlabel='Table name'>
```




## Observation

Application table has the maximum number of numerical features, hence easier to work with

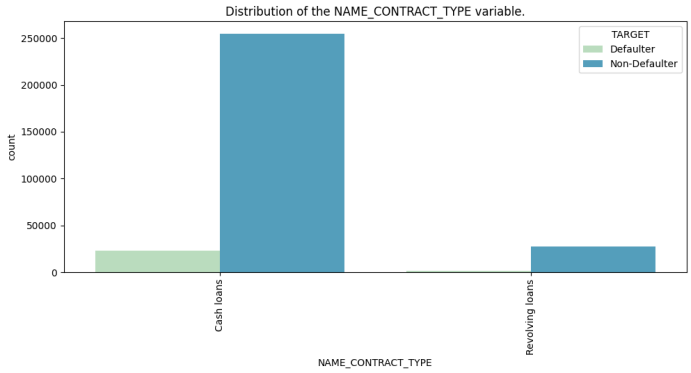
```
#Plot distribution of each numerical input variable
ax,fig = plt.subplots(21,5,figsize=(50,50))
numerical_fea = datasets['application_train'][numerical_features_app]
numerical_fea.loc[:, numerical_fea.columns != 'SK_ID_CURR'].hist(
    bins=10, figsize=(50,50),xrot = 45,legend=True,ax=ax)
plt.title('Histogram Plot for numerical features')
plt.show()
```

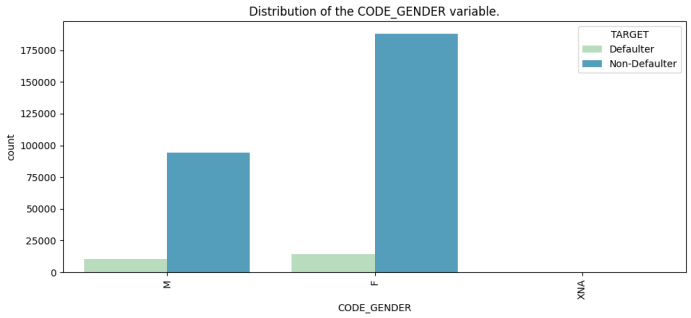
## Observation

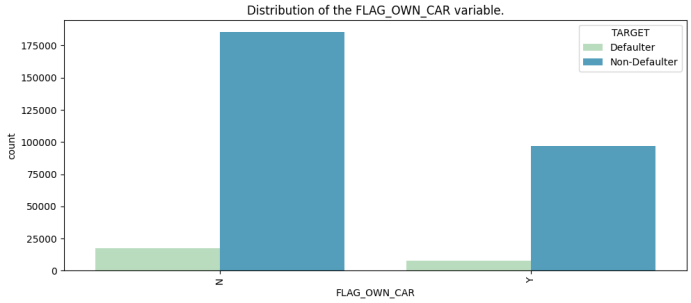
The histogram plot of each numerical feature provides a clear understanding of the distribution of the features, such as the range and spread of the values. We can observe the distribution of the features and estimate the location and spread of the data. Additionally, we can identify the outliers and understand whether the distribution is skewed or symmetric. These insights help in understanding the relationship between the features and the target variable and can be useful for feature engineering and model building.

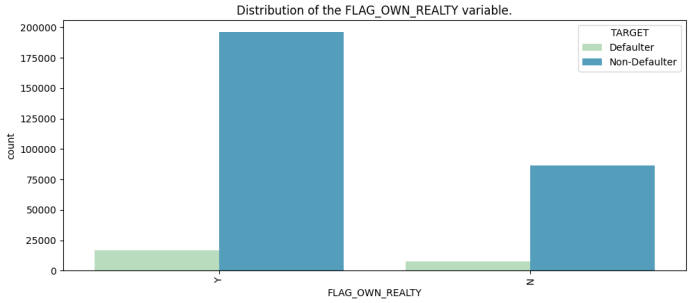
```
#Distribution of categorical variables

df_categorical = datasets['application_train'][categorical_features_app]
df_categorical['TARGET'] = datasets['application_train']['TARGET']
df_categorical['TARGET'].replace(0, 'Non-Defaulter', inplace=True)
df_categorical['TARGET'].replace(1, 'Defaulter', inplace=True)

num_cols = 2
num_rows = int(len(categorical_features_app) / num_cols)

fig, ax = plt.subplots(num_rows, num_cols, figsize=(20, 50))
col = 0

for i in range(num_rows):
    for j in range(num_cols):
```

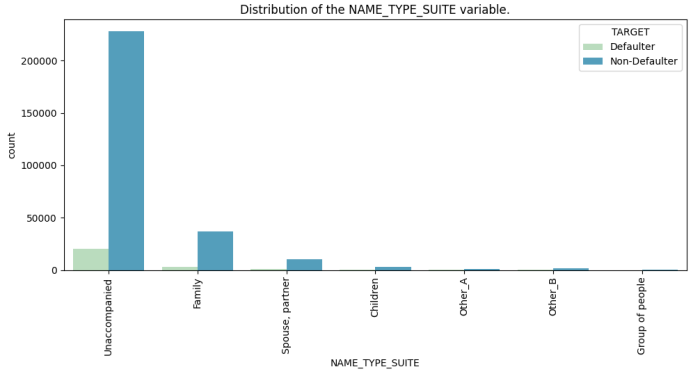
```
        if col < len(categorical_features_app):
            plot = sns.countplot(x=categorical_features_app[col],
                                 data=df_categorical, hue='TARGET', ax=ax[i][j],
            plot.set_title(f"Distribution of the {categorical_features_app[col]}
            plot.set_xticklabels(plot.get_xticklabels(), rotation=90)
            plt.subplots_adjust(hspace=0.45)
            col += 1

plt.tight_layout()
plt.show()
```

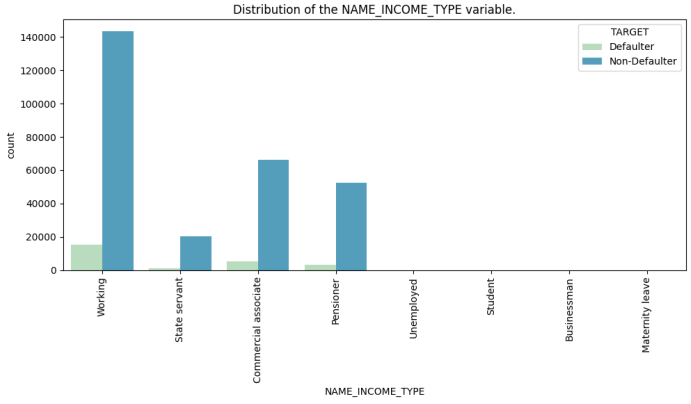


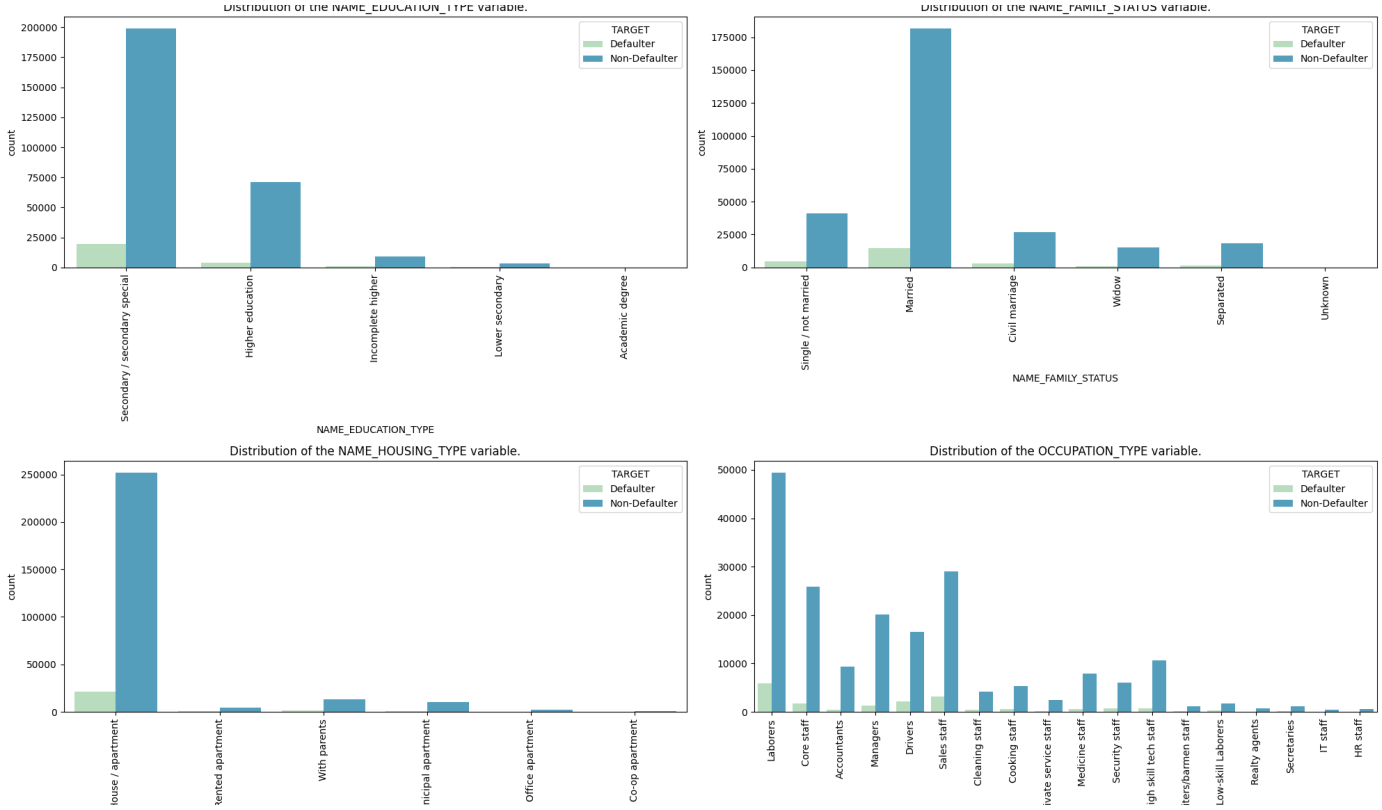





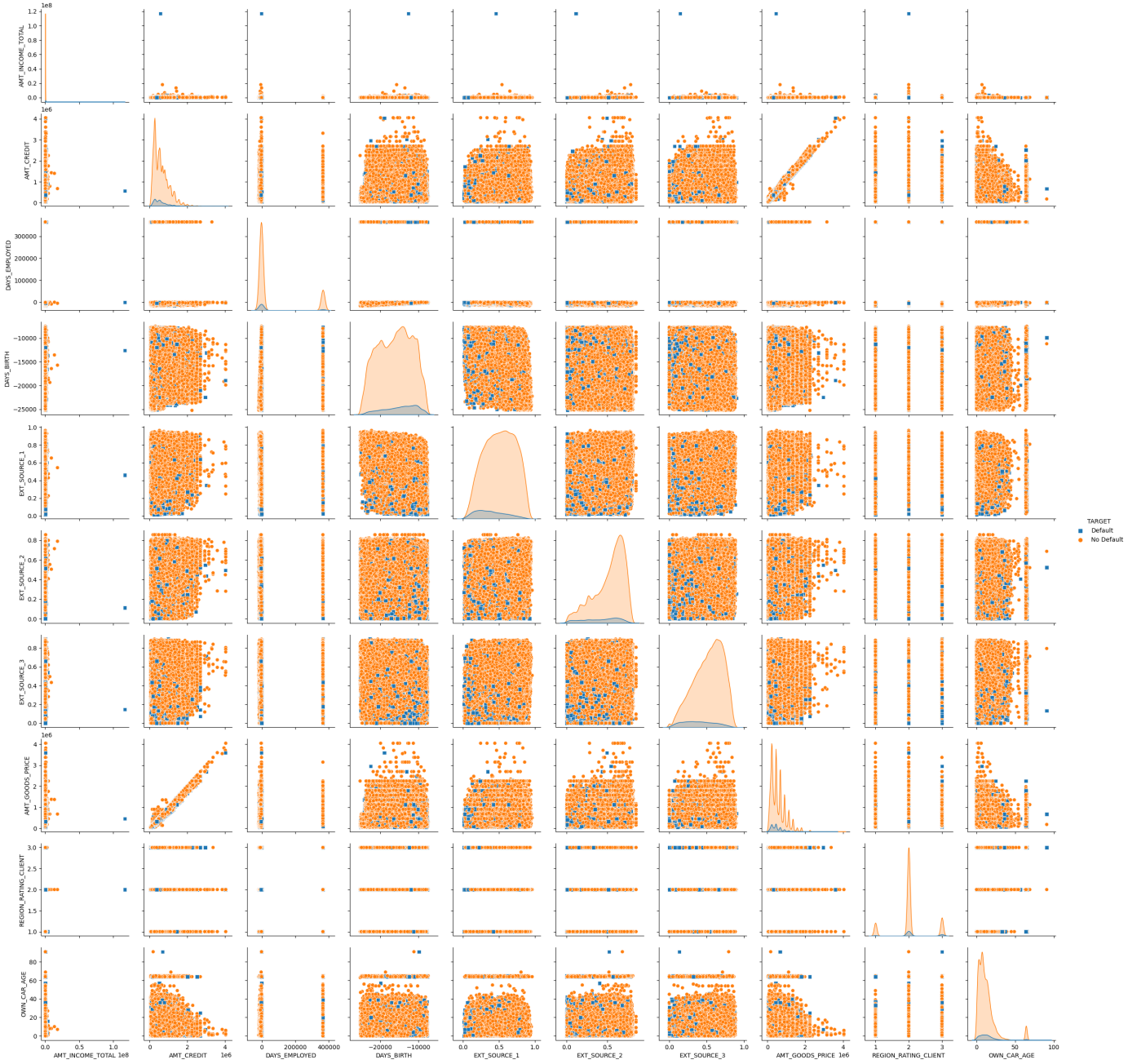

## Observation

The code plots a countplot for each categorical feature in the HCDR application dataset, showing the distribution of the target variable (Defaulter or Non-Defaulter) for each category. This allows us to see if certain categories have a higher proportion of defaulters, which can be useful in identifying potential risk factors for loan default. The code also sets the x-axis tick labels to be rotated 90 degrees to make them easier to read. Overall, the countplots provide a useful visualization of the distribution of categorical variables in the dataset.




```
#Pairplot
run = True
if run:
    df_name = 'application_train'
    num_attribs = ['TARGET', 'AMT_INCOME_TOTAL',  'AMT_CREDIT', 'DAYS_EMPLOYED',
                   'DAYS_BIRTH', 'EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3',
                   'AMT_GOODS_PRICE','REGION_RATING_CLIENT','OWN_CAR_AGE']
    df = datasets[df_name].copy()
    num_df = df[num_attribs]


    # Pair-plot
```

```
    num_df['TARGET'].replace(0, "No Default", inplace=True)
    num_df['TARGET'].replace(1, "Default", inplace=True)
    sns.pairplot(num_df, hue="TARGET", markers=["s", "o"])

# numerical_features_app = datasets['application_train'].select_dtypes(include =
# df_numerical = datasets['application_train'][numerical_features_app]
# sns.pairplot(df_numerical)
# plt.show()
```



## Observation

Some observations that can be made from the pairplot include:

1. Some features are highly correlated with each other. For example, there is a strong positive correlation between the 'DAYS_EMPLOYED' and 'DAYS_BIRTH' features, indicating that younger applicants tend to have shorter employment histories.
2. There are some features with non-linear relationships with the target variable, such as the 'EXT_SOURCE' features.
3. Some features have clear separations between the classes of the target variable, such as the 'CODE_GENDER' and 'NAME_EDUCATION_TYPE' features. This indicates that these features may be important for predicting the target variable.

# Applicants Age

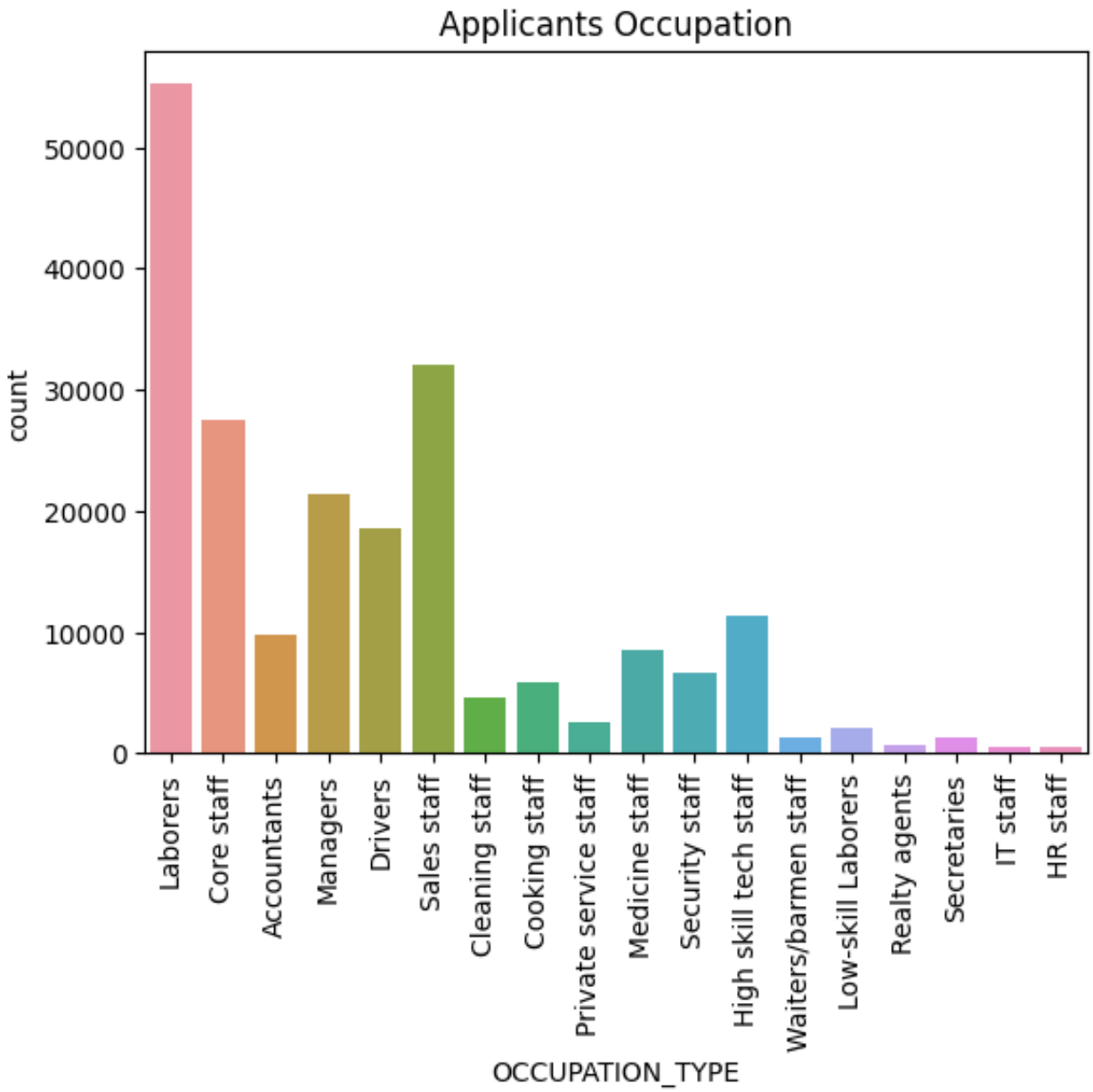
```
x = datasets['application_train']
# plt.hist(datasets["application_train"]['DAYS_BIRTH'] / -365, edgecolor = 'k', b
plt.hist([x[x['TARGET'] == 1]['DAYS_BIRTH'] / -365,x[x['TARGET'] == 0]['DAYS_BIRT
plt.title('Age of Client'); plt.xlabel('Age (years)'); plt.ylabel('Count');
```


## Observation

Based on this plot, it appears that the majority of defaulters are between the ages of 25 to 40, while non-defaulters are more evenly distributed across age ranges.

# Applicants occupations

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

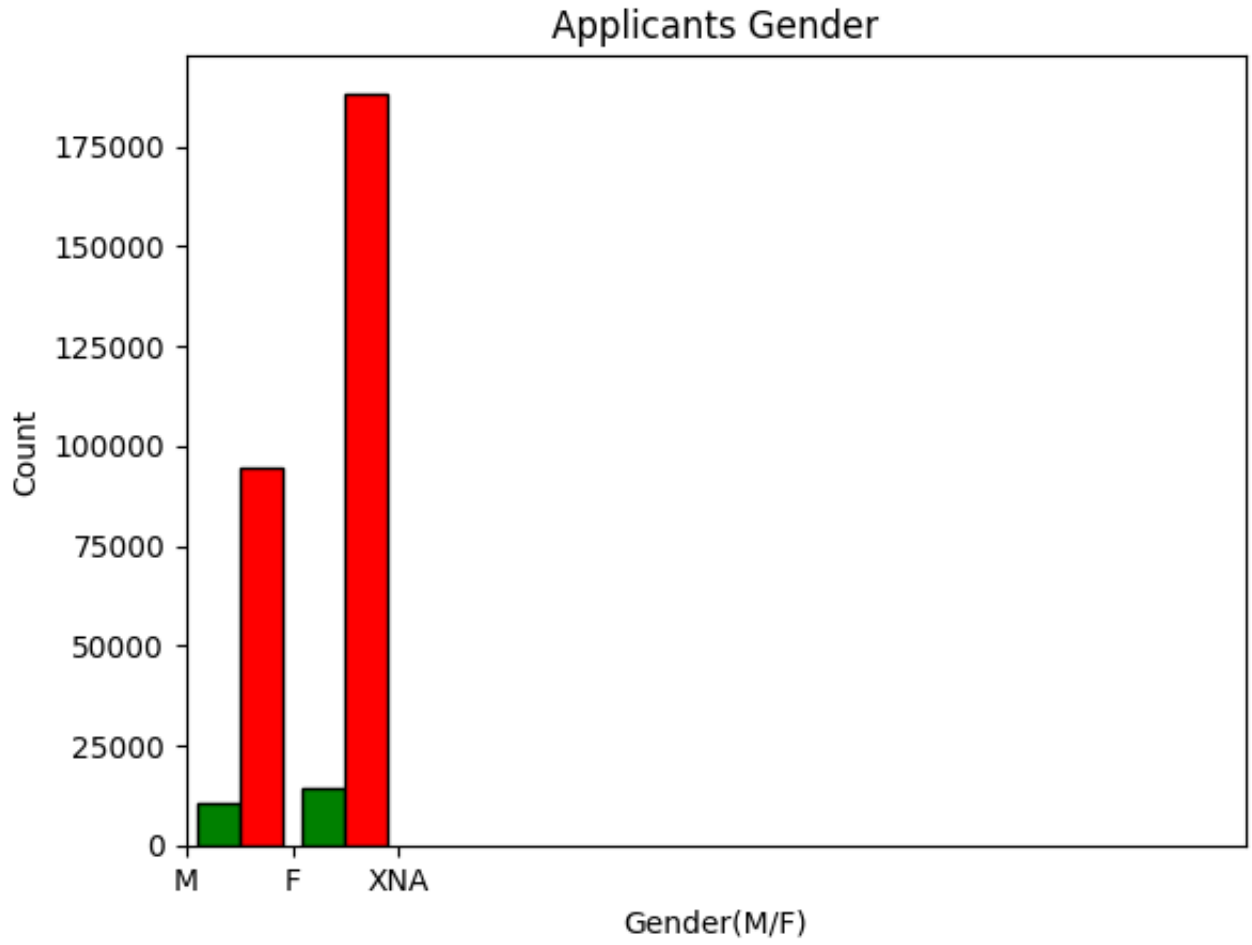



## Observation

The distribution of applicants' occupation shows that the majority of the applicants are laborers, followed by sales staff and core staff.

## Applicants Gender

```
#plt.hist(datasets["application_train"]['CODE_GENDER'] , edgecolor = 'k',stacked=
plt.hist([x[x['TARGET'] == 1]['CODE_GENDER'] ,x[x['TARGET'] == 0]['CODE_GENDER']]
plt.title('Applicants Gender'); plt.xlabel('Gender(M/F)'); plt.ylabel('Count');pl
```

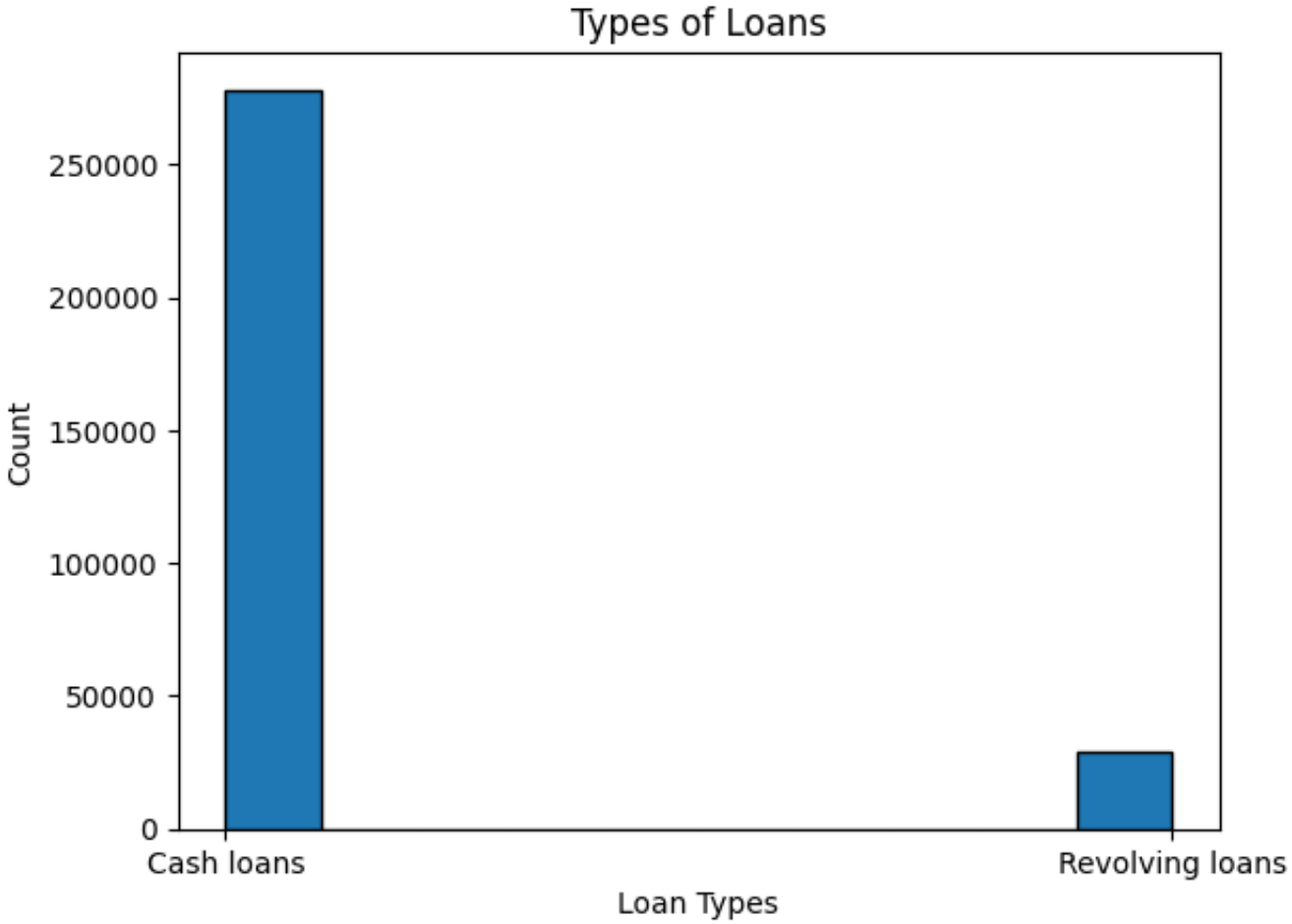



### Observation

Here we notice that number of Female applicants who are defaulters and is slightly higher than Male applicants.

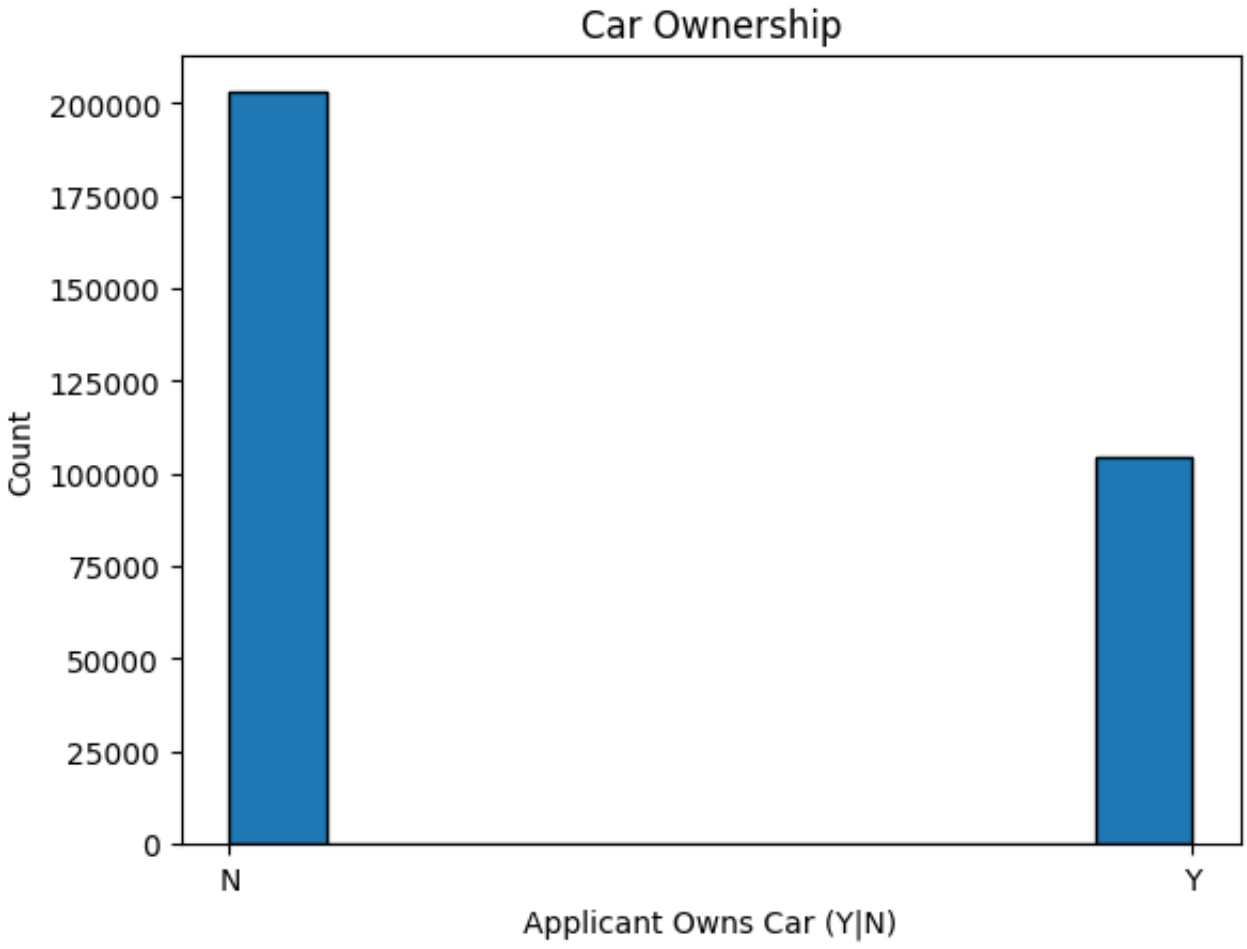
## Loan Types

```
plt.hist(datasets["application_train"]['NAME_CONTRACT_TYPE'] , edgecolor = 'k')
plt.title('Types of Loans'); plt.xlabel('Loan Types'); plt.ylabel('Count');
```




## Car Ownership

```
plt.hist(datasets["application_train"]['FLAG_OWN_CAR'] , edgecolor = 'k')
plt.title('Car Ownership'); plt.xlabel('Applicant Owns Car (Y|N)'); plt.ylabel('C
```




## Dataset questions

### Unique record for each SK_ID_CURR

```
list(datasets.keys())
```

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

```
len(datasets["application_train"]["SK_ID_CURR"].unique()) == datasets["applicatio
```

```
True
```

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

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

```
#
datasets["application_test"].shape
```

```
(48744, 121)
```

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

```
(307511, 122)
```

## previous applications for the submission file

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

```
appsDF = datasets["previous_application"]
display(appsDF.head())
print(f"{appsDF.shape[0]:,} rows, {appsDF.shape[1]:,} columns")
```

| | SK_ID_PREV | SK_ID_CURR | NAME_CONTRACT_TYPE | AMT_ANNUITY | AMT_APPLICATION | AM |
|---|---|---|---|---|---|---|
| **0** | 2030495 | 271877 | Consumer loans | 1730.4300 | 17145.0000 | |
| **1** | 2802425 | 108129 | Cash loans | 25188.6150 | 607500.0000 | 6 |
| **2** | 2523466 | 122040 | Cash loans | 15060.7350 | 112500.0000 | 1 |
| **3** | 2819243 | 176158 | Cash loans | 47041.3350 | 450000.0000 | 4 |
| **4** | 1784265 | 202054 | Cash loans | 31924.3950 | 337500.0000 | 4 |

5 rows × 37 columns

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

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

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

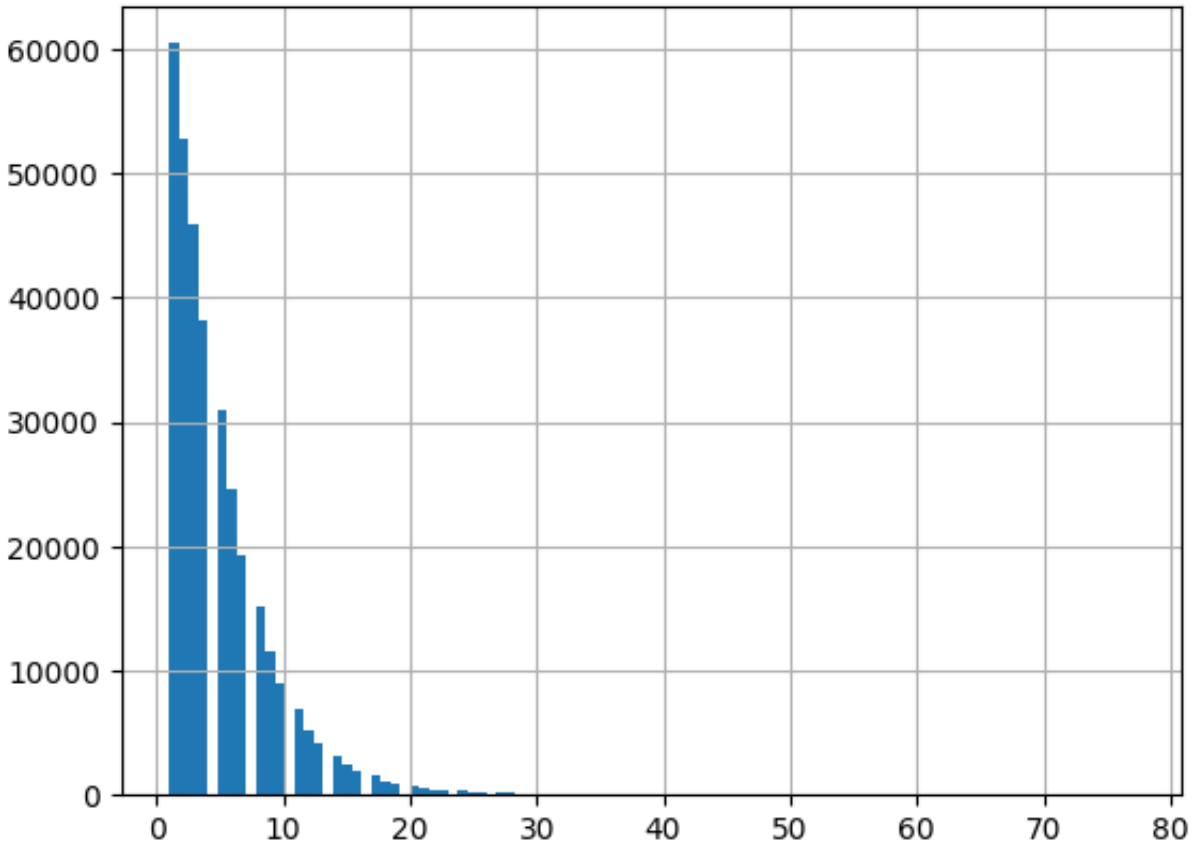
```
#Find the intersection of two arrays.
print(f'Number of train applicants with previous applications is {len(np.intersec
```

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

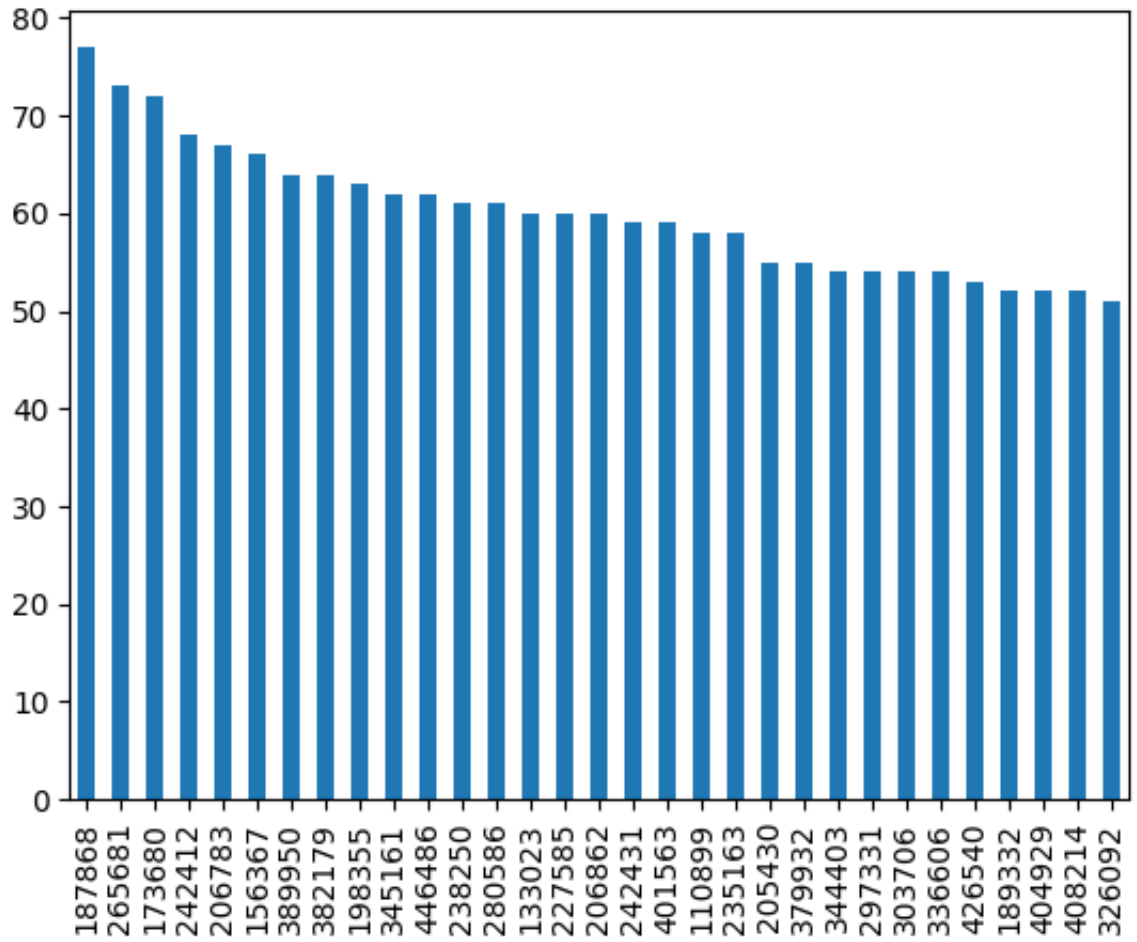
```
#Find the intersection of two arrays.
print(f'Number of train applicants with previous applications is {len(np.intersec
```

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

```
# How many previous applciations  per applicant in the previous_application
prevAppCounts = appsDF['SK_ID_CURR'].value_counts(dropna=False)
len(prevAppCounts[prevAppCounts >40])  #more than 40 previous applications
plt.hist(prevAppCounts[prevAppCounts>=0], bins=100)
plt.grid()
```



```
prevAppCounts[prevAppCounts >50].plot(kind='bar')
plt.xticks(rotation=90)
plt.show()
```



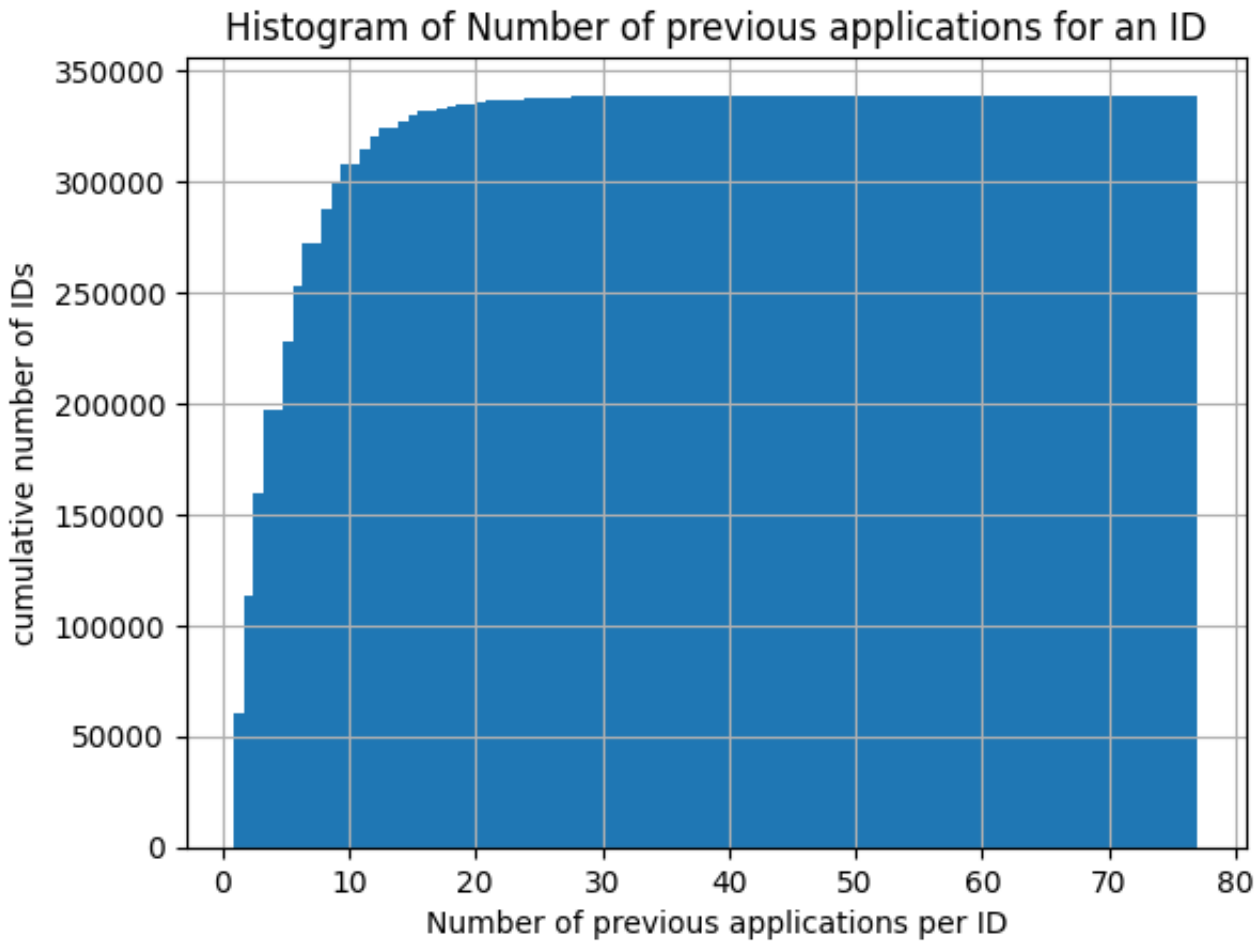

## Histogram of Number of previous applications for an ID

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

```
60458
```

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

```
Text(0.5, 1.0, 'Histogram of Number of previous applications for an ID')
```




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

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

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

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

```
# This is formatted as code
```

# 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

# agg detour

Aggregate using one or more operations over the specified axis.

For more details see agg

```
DataFrame.agg(func, axis=0, *args, **kwargs**)
```

Aggregate using one or more operations over the specified axis.

```
import pandas as pd
import numpy as np
df = pd.DataFrame([[1, 2, 3],
                   [4, 5, 6],
                  [7, 8, 9],
                  [np.nan, np.nan, np.nan]],
                  columns=['A', 'B', 'C'])
display(df)
```

| | A | B | C |
|---|---|---|---|
| **0** | 1.0 | 2.0 | 3.0 |
| **1** | 4.0 | 5.0 | 6.0 |
| **2** | 7.0 | 8.0 | 9.0 |
| **3** | NaN | NaN | NaN |

```
df.agg({'A' : ['sum', 'min'], 'B' : ['min', 'max']})
#        A    B
#max   NaN  8.0
#min   1.0  2.0
#sum  12.0  NaN
```

| | A | B |
|---|---|---|
| **sum** | 12.0 | NaN |
| **min** | 1.0 | 2.0 |
| **max** | NaN | 8.0 |

```
df = pd.DataFrame({'A': [1, 1, 2, 2],
                   'B': [1, 2, 3, 4],
                   'C': np.random.randn(4)})
display(df)
```

| | A | B | C |
|---|---|---|---|
| **0** | 1 | 1 | 0.387542 |
| **1** | 1 | 2 | 0.371771 |
| **2** | 2 | 3 | 0.539041 |
| **3** | 2 | 4 | -0.391709 |

```
# group by column A:
df.groupby('A').agg({'B': ['min', 'max'], 'C': 'sum'})
#    B             C
#  min max       sum
#A
#1   1   2  0.590716
#2   3   4  0.704907
```

| | B | | C |
|---|---|---|---|
| | min | max | sum |
| A | | | |
| 1 | 1 | 2 | 0.759312 |
| 2 | 3 | 4 | 0.147332 |

```
appsDF.columns
```

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

```
funcs = ["a","b","c"]
{f:f"{f}_max" for f in funcs}
```

```
{'a': 'a_max', 'b': 'b_max', 'c': 'c_max'}
```

## Multiple condition expressions in Pandas

So far, both our boolean selections have involved a single condition. You can, of course, have as many conditions as you would like. To do so, you will need to combine your boolean expressions using the three logical operators and, or and not.

Use &, | , ~ Although Python uses the syntax and, or, and not, these will not work when testing multiple conditions with pandas. The details of why are explained here.

You must use the following operators with pandas:

- & for and
- | for or
- ~ for not

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

| | SK_ID_PREV | SK_ID_CURR | NAME_CONTRACT_TYPE | AMT_ANNUITY | AMT_APPLICATION | AM |
|---|---|---|---|---|---|---|
| **6** | 2315218 | 175704 | Cash loans | NaN | 0.0000 | |

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

```
6   0.0000
Name: AMT_CREDIT, dtype: float64
```

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

| | SK_ID_PREV | SK_ID_CURR | NAME_CONTRACT_TYPE | AMT_ANNUITY | AMT_APPLICATION | AM |
|---|---|---|---|---|---|---|
| **6** | 2315218 | 175704 | Cash loans | NaN | 0.0000 | |

## Missing values in prevApps

```
appsDF.isna().sum()
```

```
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
```

```
appsDF.columns
```

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

## feature engineering for prevApp table

appsDF[agg_op_features].head()

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.

## 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

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

```
             AMT_ANNUITY  AMT_APPLICATION
```

```
count   1.297979e+06     1.670214e+06
mean    1.595512e+04     1.752339e+05
std     1.478214e+04     2.927798e+05
min     0.000000e+00     0.000000e+00
25%     6.321780e+03     1.872000e+04
50%     1.125000e+04     7.104600e+04
75%     2.065842e+04     1.803600e+05
max     4.180581e+05     6.905160e+06
```

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

5 rows × 21 columns

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

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

338857 rows × 7 columns

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

| | SK_ID_CURR_ | AMT_ANNUITY_min | AMT_ANNUITY_max | AMT_ANNUITY_mean | AMT_APPLIC |
|---|---|---|---|---|---|
| **0** | 100001 | 3951.000 | 3951.000 | 3951.000000 | |

| | | | | |
|---|---|---|---|---|
| **1** | 100002 | 9251.775 | 9251.775 | 9251.775000 |
| **2** | 100003 | 6737.310 | 98356.995 | 56553.990000 |
| **3** | 100004 | 5357.250 | 5357.250 | 5357.250000 |
| **4** | 100005 | 4813.200 | 4813.200 | 4813.200000 |

```
result.isna().sum()
```

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

```
# This is formatted as code
```

## feature transformer for prevApp table

```
# Create aggregate features (via pipeline)
class prevAppsFeaturesAggregater(BaseEstimator, TransformerMixin):
    def __init__(self, features=None): # no *args or **kargs
        self.features = features
        self.agg_op_features = {}
        self.agg_name_op_features = {}
        for f in features:
            self.agg_name_op_features[f] = {f"{f}_{func}":func for func in ["min"
            self.agg_op_features[f] = {np.min,np.max,np.mean}
    def fit(self, X, y=None):
        return self

    def transform(self, X, y=None):
        col=[]
        for i in self.agg_name_op_features.keys():
            for j in self.agg_name_op_features[i]:
                col.append(j)
        result = X.groupby(["SK_ID_CURR"]).agg(self.agg_op_features)
```

```
        result.columns = col
        result = result.reset_index(level=["SK_ID_CURR"])
        result['range_AMT_APPLICATION'] = result['AMT_APPLICATION_max'] - result[
        return result


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

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


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

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

df[['AMT_ANNUITY', 'AMT_APPLICATION']][0:5]:
   AMT_ANNUITY  AMT_APPLICATION
0    1730.4300       17145.0000
1   25188.6150      607500.0000
2   15060.7350      112500.0000
3   47041.3350      450000.0000
4   31924.3950      337500.0000
HELLO
Test driver:
   SK_ID_CURR  AMT_ANNUITY_min  AMT_ANNUITY_max  AMT_ANNUITY_mean  \
0      100001        3951.0000        3951.0000         3951.0000
1      100002        9251.7750        9251.7750         9251.7750
2      100003       56553.9900       98356.9950         6737.3100
3      100004        5357.2500        5357.2500         5357.2500
```

```
4      100005        4813.2000        4813.2000         4813.2000
5      100006       23651.1750       39954.5100         2482.9200
6      100007       12278.8050       22678.7850         1834.2900
7      100008       15839.6963       25309.5750         8019.0900
8      100009       10051.4121       17341.6050         7435.8450
9      100010       27463.4100       27463.4100        27463.4100

   AMT_APPLICATION_min  AMT_APPLICATION_max  AMT_APPLICATION_mean  \
0           24835.5000           24835.5000           24835.5000
1          179055.0000          179055.0000          179055.0000
2          435436.5000          900000.0000           68809.5000
3           24282.0000           24282.0000           24282.0000
4           22308.7500           44617.5000               0.0000
5          272203.2600          688500.0000               0.0000
6          150530.2500          247500.0000           17176.5000
7          155701.8000          450000.0000               0.0000
8           76741.7143          110160.0000           40455.0000
9          247212.0000          247212.0000          247212.0000

   range_AMT_APPLICATION
0                 0.0000
1                 0.0000
2            464563.5000
3                 0.0000
4             22308.7500
5            416296.7400
6             96969.7500
7            294298.2000
8             33418.2857
9                 0.0000
input[features][0:10]:
   SK_ID_PREV  SK_ID_CURR NAME_CONTRACT_TYPE  AMT_ANNUITY  AMT_APPLICATION  \
0     2030495      271877     Consumer loans    1730.4300       17145.0000
1     2802425      108129         Cash loans   25188.6150      607500.0000
2     2523466      122040         Cash loans   15060.7350      112500.0000
3     2819243      176158         Cash loans   47041.3350      450000.0000
4     1784265      202054         Cash loans   31924.3950      337500.0000
5     1383531      199383         Cash loans   23703.9300      315000.0000
6     2315218      175704         Cash loans          NaN           0.0000
7     1656711      296299         Cash loans          NaN           0.0000
8     2367563      342292         Cash loans          NaN           0.0000
9     2579447      334349         Cash loans          NaN           0.0000

   AMT_CREDIT  AMT_DOWN_PAYMENT  AMT_GOODS_PRICE WEEKDAY_APPR_PROCESS_START
```

```
res.head()
```

| | SK_ID_CURR | AMT_ANNUITY_min | AMT_ANNUITY_max | AMT_ANNUITY_mean | AMT_APPLICA |
|---|---|---|---|---|---|
| **0** | 100001 | 3951.000 | 3951.000 | 3951.000 | |
| **1** | 100002 | 9251.775 | 9251.775 | 9251.775 | |
| **2** | 100003 | 56553.990 | 98356.995 | 6737.310 | |
| **3** | 100004 | 5357.250 | 5357.250 | 5357.250 | |
| **4** | 100005 | 4813.200 | 4813.200 | 4813.200 | |

## Join the labeled dataset

```
~3==3
```

```
False
```

```
datasets.keys()
```

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

```
## Transform all secondary tables
class feature_Aggregater(BaseEstimator, TransformerMixin):
    def __init__(self, features=None,sk_id=None):
        self.features = features
        self.sk_id = sk_id
        self.agg_op_features = {}
        self.agg_name_op_features = {}
        for f in features:
            self.agg_name_op_features[f] = {f"{f}_{func}":func for func in ["min"
            self.agg_op_features[f] = {np.min,np.max,np.mean}
    def fit(self, X, y=None):
        return self

    def transform(self, X, y=None):
        col=[]
        for i in self.agg_name_op_features.keys():
            for j in self.agg_name_op_features[i]:
                col.append(j)
        result = X.groupby([self.sk_id]).agg(self.agg_op_features)
        result.columns = col
        result = result.reset_index(level=[self.sk_id])
        return result
```

```
## Previous App

prevAppsfeatures = ['AMT_ANNUITY', 'AMT_APPLICATION','AMT_CREDIT']
prevApps_feature_pipeline = Pipeline([('prevApps_aggregater', feature_Aggregater(
prevAppsDF = datasets['previous_application']
```

```
## POS_CASH_balance

POS_CASH_balance_features = ['MONTHS_BALANCE','CNT_INSTALMENT','CNT_INSTALMENT_FU
POS_CASH_balance_pipeline = Pipeline([('POS_CASH_balance_aggregater', feature_Agg

POS_CASH_balanceDF = datasets['POS_CASH_balance']
POS_CASH_balanceDF.dropna(inplace=True)
```

```
## installments_payments
installments_payments_features = ['DAYS_INSTALMENT','AMT_INSTALMENT']
installments_payments_pipeline = Pipeline([('installments_payments_aggregater', f
installments_paymentsDF = datasets['installments_payments']
installments_paymentsDF.dropna(inplace=True)
```

```
## credit_card_balance
credit_card_balance_features = ['AMT_BALANCE','AMT_DRAWINGS_CURRENT']
credit_card_balance_pipeline = Pipeline([('credit_card_balance_aggregater', featu
credit_card_balanceDF = datasets['credit_card_balance']
credit_card_balanceDF.dropna(inplace=True)
```

```
## bureau_balance
bureau_balance_features = ['MONTHS_BALANCE']
bureau_balance_pipeline = Pipeline([('bureau_balance_aggregater', feature_Aggrega
bureau_balanceDF = datasets['bureau_balance']
bureau_balanceDF.dropna(inplace=True)
```

```
## bureau
bureau_features = ['AMT_CREDIT_SUM']
bureau_pipeline = Pipeline([('bureau_aggregater', feature_Aggregater(bureau_featu
bureauDF = datasets['bureau']
bureauDF.dropna(inplace=True)
```

```
X_train= datasets["application_train"] #primary dataset
print(X_train.shape)
appsDF = datasets["previous_application"] #prev app

merge_all_data = True

if merge_all_data:
    prevApps_aggregated = prevApps_feature_pipeline.transform(appsDF)
    POS_CASH_balance_aggregated = POS_CASH_balance_pipeline.transform(POS_CASH_ba
    installments_payments_aggregated = installments_payments_pipeline.transform(i
    print(installments_payments_aggregated.columns)
    credit_card_balance_aggregated = credit_card_balance_pipeline.transform(credi
    bureau_balance_aggregated = bureau_balance_pipeline.transform(bureau_balanceD
    bureau_aggregated = bureau_pipeline.transform(bureauDF)
    bureauDF = bureauDF.merge(bureau_balance_aggregated, how='left', on="SK_ID_BU
    bureauDF = bureauDF.merge(bureau_aggregated, how='left', on="SK_ID_CURR")
    X_train = X_train.merge(prevApps_aggregated, how='left', on="SK_ID_CURR")
    X_train = X_train.merge(POS_CASH_balance_aggregated, how='left', on="SK_ID_CU
    X_train = X_train.merge(installments_payments_aggregated, how='left', on="SK_
    X_train = X_train.merge(credit_card_balance_aggregated, how='left', on="SK_ID
    X_train = X_train.merge(bureauDF, how='left', on="SK_ID_CURR")
```

```
(307511, 122)
Index(['SK_ID_CURR', 'DAYS_INSTALMENT_min', 'DAYS_INSTALMENT_max',
       'DAYS_INSTALMENT_mean', 'AMT_INSTALMENT_min', 'AMT_INSTALMENT_max',
       'AMT_INSTALMENT_mean'],
      dtype='object')
```

```
X_train.shape
```

```
(322187, 171)
```

```
with open("X_train_before", "w") as f:
    X_train.to_csv(f, index=False)

f.close()
```

## Feature Engineering

```
import pandas as pd
csv_file = 'X_train_before'

# Read the CSV file into a pandas DataFrame
#X_train = pd.read_csv(csv_file)
X_train.shape
```

```
(1519680, 201)
```

```
prevAppCounts = appsDF['SK_ID_CURR'].value_counts(dropna=False)
prevAppCounts_df = prevAppCounts.to_frame()
# Frequency feature
prevAppCounts_df.columns = ['Count']
prevAppCounts_df['SK_ID_CURR'] = prevAppCounts.index
# Monetary feature
prevAppCounts_df['AverageAppAmt'] = appsDF.groupby(['SK_ID_CURR'])['AMT_APPLICATI
# Recency feature
prevAppCounts_df['AVG_DAYS_BETWEEN_PAYMENTS'] = installments_paymentsDF.groupby('
prevAppCounts_df['AVG_DAYS_BETWEEN_PAYMENTS'] = np.where(prevAppCounts_df['AVG_DA
                                                         prevAppCounts_df['AVG_DA
X_train = X_train.merge(prevAppCounts_df,how='left',on='SK_ID_CURR')
X_train.shape
```

```
(322187, 174)
```

```
# Drop columns with more than 50% missing values
# threshold = 0.9
# null_counts = X_train.isnull().sum() / len(df)
# drop_cols = null_counts[null_counts > threshold].index
# X_train = X_train.drop(drop_cols, axis=1)


def drop_missing_cols(df, drop_percentage):
    percent = (df.isnull().sum()/df.isnull().count()*100).sort_values(ascending =
    missing_cols = percent[percent > drop_percentage].index.tolist()
    df.drop(columns=missing_cols, inplace=True)
    print(missing_cols)
    return df

X_train = drop_missing_cols(X_train, 90)
X_train.shape
```

```
[]
(322187, 174)
```

```
# dropping one of the highly correlated features:
# Create a correlation matrix
import numpy as np
def drop_highly_correlated_features(df, corr_coeff):
    corr_matrix = df.corr().abs()

    mask = np.triu(np.ones_like(corr_matrix, dtype=bool))

    # pairs of highly correlated features (correlation coefficient >= 0.995)
    corr_pairs = [(i, j) for i in range(len(corr_matrix.columns)) for j in range(

    for pair in corr_pairs:
        print(f"Highly correlated features: {corr_matrix.columns[pair[0]]}, {corr
        print(f"Correlation coefficient: {corr_matrix.iloc[pair[0], pair[1]]}\n")

    # Dropping one of the features from each highly correlated pair
    for pair in corr_pairs:
        feature1 = corr_matrix.columns[pair[0]]
        feature2 = corr_matrix.columns[pair[1]]
        if feature1 in df.columns and feature2 in df.columns:
            if df[feature1].dtype != 'object' and df[feature2].dtype != 'object':
                if abs(df[feature1].corr(df['TARGET'])) >= abs(df[feature2].corr(
                    df.drop(columns=[feature1], inplace=True)
                else:
```

```
                    df.drop(columns=[feature2], inplace=True)
    return df
X_train = drop_highly_correlated_features(X_train, 0.995)
X_train.shape
```

```
Highly correlated features: DAYS_EMPLOYED, FLAG_EMP_PHONE
Correlation coefficient: 0.9997501312628575

Highly correlated features: APARTMENTS_AVG, APARTMENTS_MEDI
Correlation coefficient: 0.9951638081592303

Highly correlated features: YEARS_BUILD_AVG, YEARS_BUILD_MEDI
Correlation coefficient: 0.9985106831314824

Highly correlated features: COMMONAREA_AVG, COMMONAREA_MEDI
Correlation coefficient: 0.9960542778645148

Highly correlated features: ELEVATORS_AVG, ELEVATORS_MEDI
Correlation coefficient: 0.9960713815440911

Highly correlated features: ENTRANCES_AVG, ENTRANCES_MEDI
Correlation coefficient: 0.9968730280305592

Highly correlated features: FLOORSMAX_AVG, FLOORSMAX_MEDI
Correlation coefficient: 0.9970462703773344

Highly correlated features: FLOORSMIN_AVG, FLOORSMIN_MEDI
Correlation coefficient: 0.9971684189793344

Highly correlated features: LIVINGAREA_AVG, LIVINGAREA_MEDI
Correlation coefficient: 0.9956324098885693

Highly correlated features: OBS_30_CNT_SOCIAL_CIRCLE, OBS_60_CNT_SOCIAL_CIRCL
Correlation coefficient: 0.9984733876311623

Highly correlated features: AMT_APPLICATION_min, AverageAppAmt
Correlation coefficient: 1.0

Highly correlated features: CNT_INSTALMENT_max, CNT_INSTALMENT_FUTURE_max
Correlation coefficient: 0.998029000848077

Highly correlated features: DAYS_CREDIT, MONTHS_BALANCE_mean_y
Correlation coefficient: 0.9999336025220297

(322187, 161)
```

```
#X_train.to_csv("X_train_after",index=False)
with open("X_train_final", "w") as f:
    X_train.to_csv(f, index=False)

f.close()
```

```
import pandas as pd
csv_file = 'X_train_final'

# Read the CSV file into a pandas DataFrame
X_train = pd.read_csv(csv_file)
X_train.shape
df=X_train
```

```
num_duplicates = X_train.duplicated().sum()

# Print the number of duplicate rows
print("Number of duplicate rows:", num_duplicates)
```

```
Number of duplicate rows: 0
```

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

```
X_kaggle_test= datasets["application_test"]
if merge_all_data:
    # 1. Join/Merge in prevApps Data
    X_kaggle_test = X_kaggle_test.merge(prevApps_aggregated, how='left', on='SK_I

    #Since the PrevAppsDF is already merged with other secondary tables, we are c
```

```
# Convert categorical features to numerical approximations (via pipeline)
class ClaimAttributesAdder(BaseEstimator, TransformerMixin):
    def fit(self, X, y=None):
        return self
    def transform(self, X, y=None):
        charlson_idx_dt = {'0': 0, '1-2': 2, '3-4': 4, '5+': 6}
        los_dt = {'1 day': 1, '2 days': 2, '3 days': 3, '4 days': 4, '5 days': 5,
          '1- 2 weeks': 11, '2- 4 weeks': 21, '4- 8 weeks': 42, '26+ weeks': 180}
        X['PayDelay'] = X['PayDelay'].apply(lambda x: int(x) if x != '162+' else
        X['DSFS'] = X['DSFS'].apply(lambda x: None if pd.isnull(x) else int(x[0])
        X['CharlsonIndex'] = X['CharlsonIndex'].apply(lambda x: charlson_idx_dt[x
        X['LengthOfStay'] = X['LengthOfStay'].apply(lambda x: None if pd.isnull(x
        return X
```

# Data processing and pipeline for HCDR

```
import pandas as pd
import numpy as np
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report
from sklearn.metrics import mean_squared_error,f1_score, roc_auc_score
from sklearn import metrics
from sklearn.model_selection import GridSearchCV


#logistic regssion import statement
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import accuracy_score
from time import time
```

**Splitting the data**

```
X_train.columns
```

```
Index(['SK_ID_CURR', 'TARGET', 'NAME_CONTRACT_TYPE', 'CODE_GENDER',
       'FLAG_OWN_CAR', 'FLAG_OWN_REALTY', 'CNT_CHILDREN',
'AMT_INCOME_TOTAL',
       'AMT_CREDIT', 'AMT_ANNUITY_x',
       ...
       'AMT_ANNUITY_y', 'MONTHS_BALANCE_min_y', 'MONTHS_BALANCE_max_y',
       'MONTHS_BALANCE_mean_y', 'AMT_CREDIT_SUM_min', 'AMT_CREDIT_SUM_max',
       'AMT_CREDIT_SUM_mean', 'Count', 'AverageAppAmt',
       'AVG_DAYS_BETWEEN_PAYMENTS'],
      dtype='object', length=161)
```

```
#df = datasets["application_train"]
df = X_train
#input_f = datasets["application_train"]
X = df.drop('TARGET', axis=1)  # Features
y = df['TARGET']  # Target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train,
                                                      test_size=0.2, random_state

print(f"X train           shape: {X_train.shape}")
print(f"X validation      shape: {X_valid.shape}")
print(f"X test            shape: {X_test.shape}")
#X_train_resampled, y_train_resampled = resample(X_train[y_train == 0], y_train[y
#X_train = pd.concat([X_train[y_train == 1], X_train_resampled])
#y_train = pd.concat([y_train[y_train == 1], y_train_resampled])
```

```
X train           shape: (219086, 160)
X validation      shape: (54772, 160)
X test            shape: (48329, 160)
```

**Numberical Features**

Identify the numeric features we wish to consider

```
#getting the numerical features from the application train table which contains i
numerical_features = df.select_dtypes(include=['int64', 'float64']).columns.tolis
```

```
numerical_features.remove("TARGET")
numerical_features
```

```
['SK_ID_CURR',
 'CNT_CHILDREN',
 'AMT_INCOME_TOTAL',
 'AMT_CREDIT',
 'AMT_ANNUITY_x',
 'AMT_GOODS_PRICE',
 'REGION_POPULATION_RELATIVE',
 'DAYS_BIRTH',
 'DAYS_EMPLOYED',
 'DAYS_REGISTRATION',
 'DAYS_ID_PUBLISH',
 'OWN_CAR_AGE',
 'FLAG_MOBIL',
 'FLAG_WORK_PHONE',
 'FLAG_CONT_MOBILE',
 'FLAG_PHONE',
 'FLAG_EMAIL',
 'CNT_FAM_MEMBERS',
 'REGION_RATING_CLIENT',
 'REGION_RATING_CLIENT_W_CITY',
 'HOUR_APPR_PROCESS_START',
 'REG_REGION_NOT_LIVE_REGION',
 'REG_REGION_NOT_WORK_REGION',
 'LIVE_REGION_NOT_WORK_REGION',
 'REG_CITY_NOT_LIVE_CITY',
 'REG_CITY_NOT_WORK_CITY',
 'LIVE_CITY_NOT_WORK_CITY',
 'EXT_SOURCE_1',
 'EXT_SOURCE_2',
 'EXT_SOURCE_3',
 'BASEMENTAREA_AVG',
 'YEARS_BEGINEXPLUATATION_AVG',
 'YEARS_BUILD_AVG',
 'COMMONAREA_AVG',
 'LANDAREA_AVG',
 'LIVINGAPARTMENTS_AVG',
 'NONLIVINGAPARTMENTS_AVG',
 'NONLIVINGAREA_AVG',
 'APARTMENTS_MODE',
 'BASEMENTAREA_MODE',
 'YEARS_BEGINEXPLUATATION_MODE',
 'YEARS_BUILD_MODE',
 'COMMONAREA_MODE',
 'ELEVATORS_MODE',
 'ENTRANCES_MODE',
 'FLOORSMAX_MODE',
 'FLOORSMIN_MODE',
 'LANDAREA_MODE',
 'LIVINGAPARTMENTS_MODE',
 'LIVINGAREA_MODE'
```

```
 'LIVINGAREA_MODE',
 'NONLIVINGAPARTMENTS_MODE',
 'NONLIVINGAREA_MODE',
 'APARTMENTS_MEDI',
 'BASEMENTAREA_MEDI',
 'YEARS_BEGINEXPLUATATION_MEDI',
 'ELEVATORS_MEDI',
 'ENTRANCES_MEDI',
 'FLOORSMAX_MEDI',
 'FLOORSMIN_MEDI',
 'LANDAREA_MEDI',
```

```
len(numerical_features)
```

```
141
```

### Categorical Features

Identify the categorical features we wish to consider

```
#getting the categorical features from the application train table which contains
categorical_features = df.select_dtypes(include=['object']).columns.tolist()
```

```
categorical_features
```

```
['NAME_CONTRACT_TYPE',
 'CODE_GENDER',
 'FLAG_OWN_CAR',
 'FLAG_OWN_REALTY',
 'NAME_TYPE_SUITE',
 'NAME_INCOME_TYPE',
 'NAME_EDUCATION_TYPE',
 'NAME_FAMILY_STATUS',
 'NAME_HOUSING_TYPE',
 'OCCUPATION_TYPE',
 'WEEKDAY_APPR_PROCESS_START',
 'ORGANIZATION_TYPE',
 'FONDKAPREMONT_MODE',
 'HOUSETYPE_MODE',
 'WALLSMATERIAL_MODE',
 'EMERGENCYSTATE_MODE',
 'CREDIT_ACTIVE',
 'CREDIT_CURRENCY',
 'CREDIT_TYPE']
```

```
len(categorical_features)
```

```
19
```

```
from sklearn.base import BaseEstimator, TransformerMixin
# Create a class to select numerical or categorical columns
# since Scikit-Learn doesn't handle DataFrames yet
class DataFrameSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attribute_names):
        self.attribute_names = attribute_names
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.attribute_names].values
```

### Numerical Pipeline Defination

```
# Create numerical pipeline
numerical_pipeline = Pipeline([
    ('selector', DataFrameSelector(numerical_features)),
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])
```

### Categorical Pipeline Definition

**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.

```
# Create categorical pipeline
categorical_pipeline = Pipeline([
    ('selector', DataFrameSelector(categorical_features)),
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('encoder', OneHotEncoder(handle_unknown= 'ignore', sparse=False))
])
```

**Data Preparation Pipeline**

with columnTransformer() we combine both numerical and categorical feature pipelines to form a pipeline

```
# Create column transformer
column_transformer_pipeline = ColumnTransformer(
    transformers=[
        ('num_pipeline', numerical_pipeline),
        ('cat_pipeline', categorical_pipeline)
    ], remainder="passthrough")

from sklearn.pipeline import FeatureUnion

data_prep_pipeline_FU = FeatureUnion(transformer_list=[
        ("num_pipeline", numerical_pipeline),
        ("cat_pipeline", categorical_pipeline),
    ])
```

**Number of selected features are**

```
selected_features = numerical_features + categorical_features
tot_features = f"{len(selected_features)}:   Num:{len(numerical_features)},    Ca
#Total Feature selected for processing
tot_features
```

```
'176:   Num:141,    Cat:35'
```

```
# Set feature selection settings
# Features removed each step
feature_selection_steps=10
# Number of features used
features_used=len(selected_features)
```

# Baseline Model

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

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

```
try:
    experimentLog
except NameError:
    experimentLog = pd.DataFrame(columns=["exp_name",
                                      "Train Acc",
                                      "Valid Acc",
                                      "Test  Acc",
                                      "Train AUC",
                                      "Valid AUC",
                                      "Test  AUC",
                                      "Train F1 Score",
                                      "Valid F1 Score",
                                      "Test F1 Score",
                                      "Train Log Loss",
                                      "Valid Log Loss",
                                      "Test Log Loss",
                                      "Train Time",
                                      "Valid Time",
                                      "Test Time",
                                      "Description"
                                     ])
```

### Defining Pipeline

### Logistic Regression

```
%%time
np.random.seed(42)
logistic_pipeline = Pipeline([
        ("preparation", column_transformer_pipeline),# combination of numerical,
        #("clf", MultinomialNB())  # classifier estimator you are using
        ("logistic_regression", LogisticRegression() )
    ])
```

```
CPU times: user 474 µs, sys: 0 ns, total: 474 µs
Wall time: 481 µs
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.feature_selection import RFE



classifiers = [    [('Logistic Regression', LogisticRegression(solver='saga', ran
    [('Support Vector Machine', SVC(random_state=42, probability=True),"SVM")],
    [('Decision Tree', DecisionTreeClassifier(random_state=42), "RFE")],
    [('Random Forest', RandomForestClassifier(random_state=42), "RFE")],
    [('Gradient Boosting', GradientBoostingClassifier(warm_start=True, random_sta
]
```

Perform cross-fold validation and Train the model Split the training data to 15 fold to perform Crossfold validation

```
from sklearn.model_selection import ShuffleSplit, cross_validate
cvSplits = ShuffleSplit(n_splits=5, test_size=0.3, random_state=0)
```

```
start = time()
logistic_pipeline.fit(X_train, y_train)
duration_LR_train = np.round((time() - start), 4)
np.random.seed(42)
```

```
# Time and score for valid predictions
start = time()
logit_score_valid  = logistic_pipeline.score(X_valid, y_valid)
valid_time = np.round(time() - start, 4)

# Time and score for test predictions
start = time()
logit_score_test  = logistic_pipeline.score(X_test, y_test)
test_time = np.round(time() - start, 4)
```

```
# Time and score for train predictions
start = time()
logit_score_train  = logistic_pipeline.score(X_train, y_train)
train_time = np.round(time() - start, 4)
```

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import log_loss

# obtained predicted probabilities or scores for train
predicted_probs = logistic_pipeline.predict_proba(X_train)[:, 1]
true_labels = y_train

# Calculate the AUC for train
auc_train = roc_auc_score(true_labels, predicted_probs)
# Calculate the log loss for train
logloss_train = log_loss(true_labels, predicted_probs)


# obtained predicted probabilities or scores for test predictions
predicted_probs = logistic_pipeline.predict_proba(X_test)[:, 1]
true_labels = y_test

# Calculate the AUC for test
auc_test = roc_auc_score(true_labels, predicted_probs)
# Calculate the log loss for test
logloss_test = log_loss(true_labels, predicted_probs)


# obtained predicted probabilities or scores for valid predictions
predicted_probs = logistic_pipeline.predict_proba(X_valid)[:, 1]
true_labels = y_valid

# Calculate the AUC for test
auc_valid = roc_auc_score(true_labels, predicted_probs)
# Calculate the log loss for valid
logloss_valid = log_loss(true_labels, predicted_probs)


#F1-Score for test
y_test_pred = logistic_pipeline.predict(X_test)
f1_test = f1_score(y_test, y_test_pred)
print("F1-Score for Logistic Regression is: " , np.round(f1_test, 4))
```

```
#F1-Score for train
y_train_pred = logistic_pipeline.predict(X_train)
f1_train = f1_score(y_train, y_train_pred)


#F1-Score for valid
y_valid_pred = logistic_pipeline.predict(X_valid)
f1_valid = f1_score(y_valid, y_valid_pred)
```

```
F1-Score for Logistic Regression is:  0.0414
```

```
exp_name = f"Baseline_{len(selected_features)}_features"
experimentLog = pd.DataFrame(columns=["exp_name",
                                      "Train Acc",
                                      "Valid Acc",
                                      "Test  Acc",
                                      "Train AUC",
                                      "Valid AUC",
                                      "Test  AUC",
                                      "Train F1 Score",
                                      "Valid F1 Score",
                                      "Test F1 Score",
                                      "Train Log Loss",
                                      "Valid Log Loss",
                                      "Test Log Loss",
                                      "Train Time",
                                      "Valid Time",
                                      "Test Time",
                                      "Description"
                                     ])
experimentLog.loc[len(experimentLog)] = [f"{exp_name}"],logit_score_train, logit_
#experimentLog.loc[len(experimentLog)] = ["Logistic Regression as Baseline", "HCD

experimentLog
```

| | exp_name | Train Acc | Valid Acc | Test Acc | Train AUC | Valid AUC | Test AUC | Tr Sc |
|---|---|---|---|---|---|---|---|---|
| **0** | [Baseline_176_features] | 0.913844 | 0.91348 | 0.913732 | 0.752923 | 0.752498 | 0.752081 | 0.04 |

# Evaluation metrics

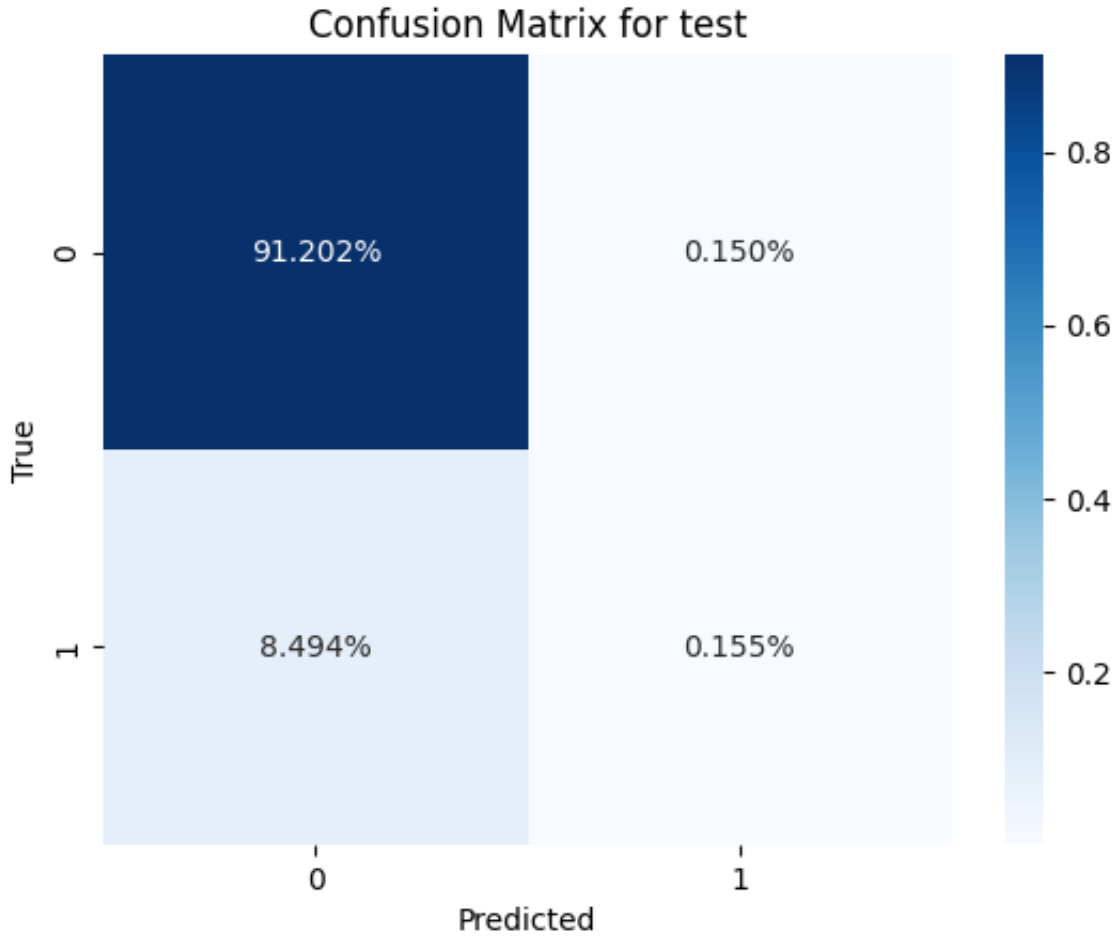
Submissions are evaluated on [area under the ROC curve](#) between the predicted probability and the observed target.

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

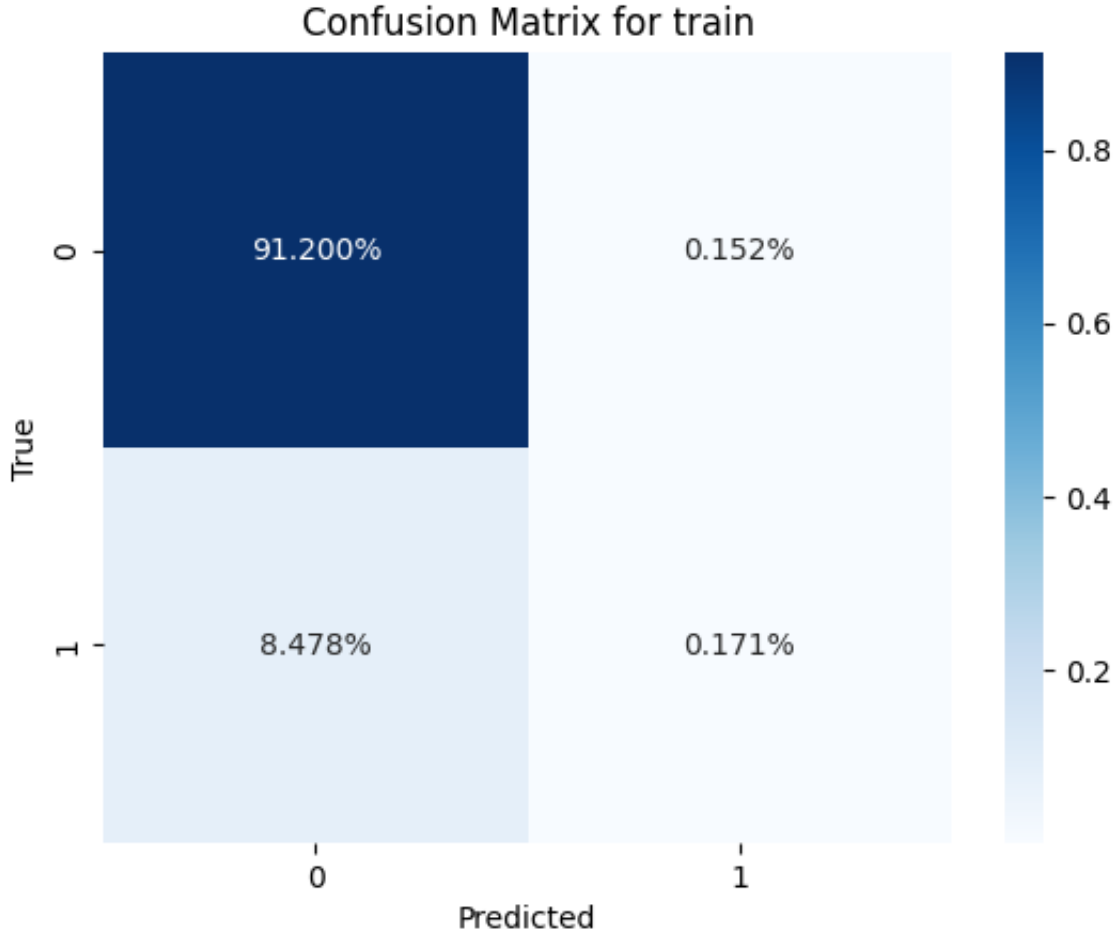
```
from sklearn.metrics import roc_auc_score
>>> y_true = np.array([0, 0, 1, 1])
>>> y_scores = np.array([0.1, 0.4, 0.35, 0.8])
>>> roc_auc_score(y_true, y_scores)
0.75
```

**Confusion Matrix**

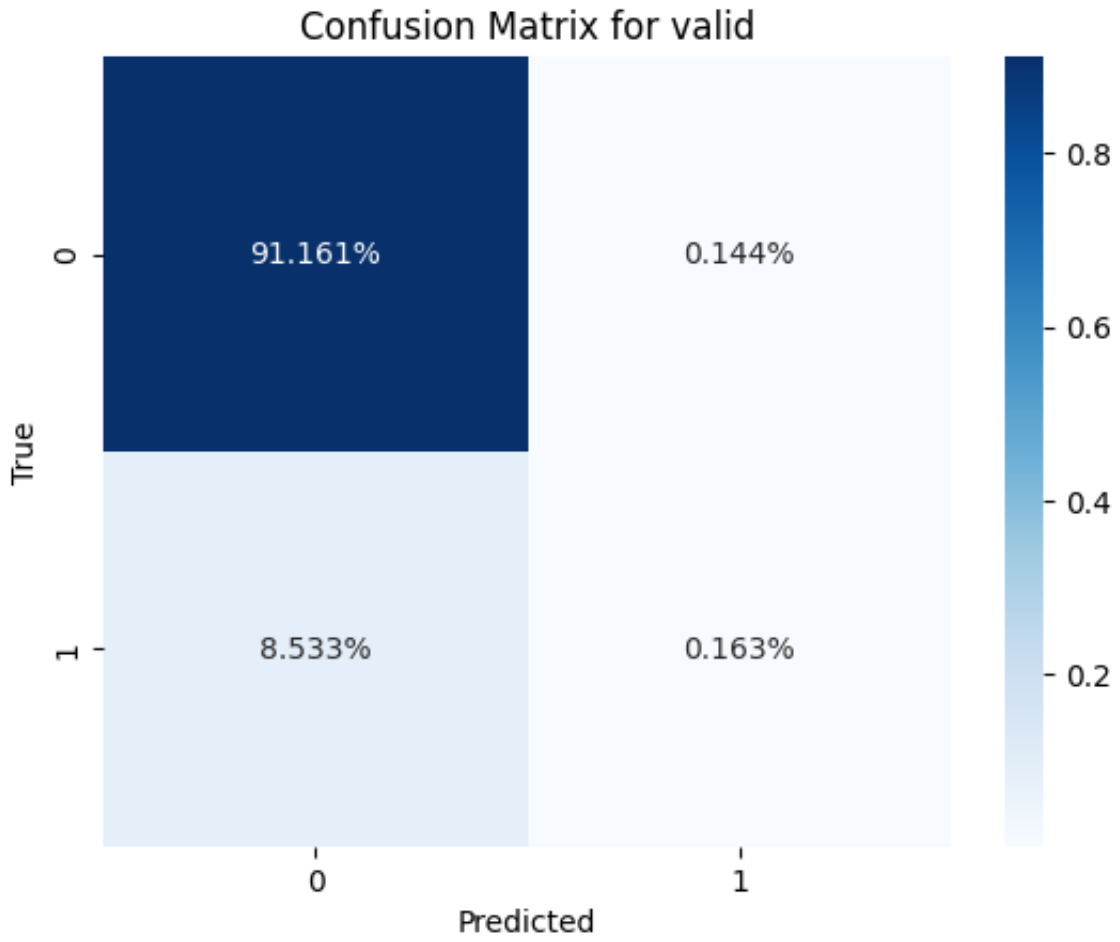
```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.pipeline import Pipeline


# Calculate the confusion matrix
cm = confusion_matrix(y_test, y_test_pred)

# Create a heatmap for the confusion matrix
sns.heatmap(cm/np.sum(cm), annot=True, fmt=".3%", cmap="Blues")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix for test')
plt.show()
```



```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.pipeline import Pipeline


# Calculate the confusion matrix
cm = confusion_matrix(y_train, y_train_pred)

# Create a heatmap for the confusion matrix
sns.heatmap(cm/np.sum(cm), annot=True, fmt=".3%", cmap="Blues")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix for train')
plt.show()
```

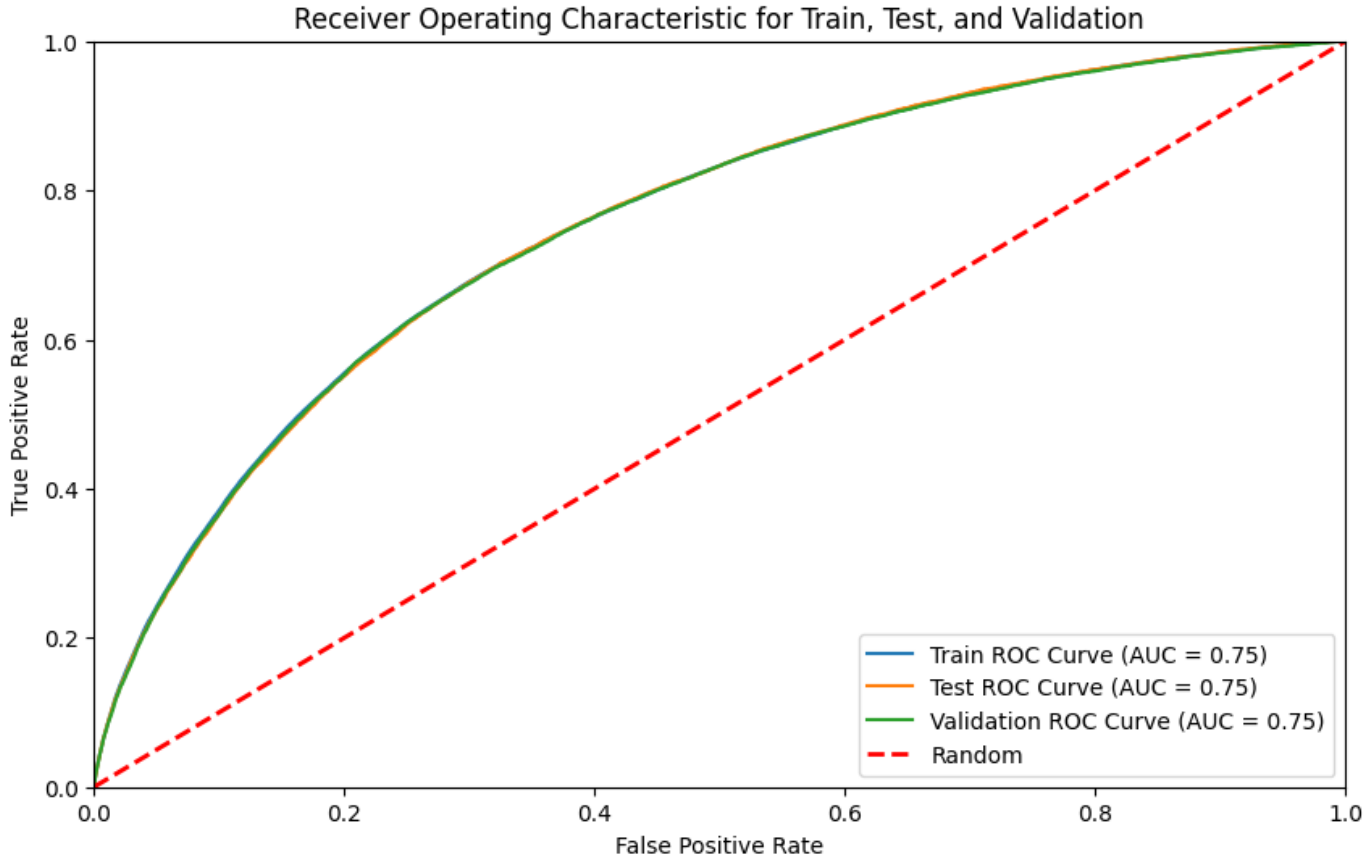


```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.pipeline import Pipeline


# Calculate the confusion matrix
cm = confusion_matrix(y_valid, y_valid_pred)

# Create a heatmap for the confusion matrix
sns.heatmap(cm/np.sum(cm), annot=True, fmt=".3%", cmap="Blues")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix for valid')
plt.show()
```




## ROC CURVE

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

# Obtain predicted probabilities for the positive class for each dataset
train_predicted_probs = logistic_pipeline.predict_proba(X_train)[:, 1]
test_predicted_probs = logistic_pipeline.predict_proba(X_test)[:, 1]
valid_predicted_probs = logistic_pipeline.predict_proba(X_valid)[:, 1]

# Assuming y_train, y_test, y_valid contain the true labels for your train, test,
train_true_labels = y_train
test_true_labels = y_test
valid_true_labels = y_valid

# Calculate the false positive rate, true positive rate, and thresholds for each
train_fpr, train_tpr, _ = roc_curve(train_true_labels, train_predicted_probs)
test_fpr, test_tpr, _ = roc_curve(test_true_labels, test_predicted_probs)
valid_fpr, valid_tpr, _ = roc_curve(valid_true_labels, valid_predicted_probs)

# Calculate the area under the ROC curve for each dataset
train_roc_auc = auc(train_fpr, train_tpr)
test_roc_auc = auc(test_fpr, test_tpr)
valid_roc_auc = auc(valid_fpr, valid_tpr)

# Plot the ROC curves for train, test, and validation datasets on a single plot
plt.figure(figsize=(10, 6))
plt.plot(train_fpr, train_tpr, label='Train ROC Curve (AUC = {:.2f})'.format(trai
plt.plot(test_fpr, test_tpr, label='Test ROC Curve (AUC = {:.2f})'.format(test_rc
plt.plot(valid_fpr, valid_tpr, label='Validation ROC Curve (AUC = {:.2f})'.format
plt.plot([0, 1], [0, 1], 'r--', label='Random', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic for Train, Test, and Validation')
plt.legend(loc='lower right')
plt.show()
```




## Decision Tree

```
from sklearn.tree import DecisionTreeClassifier
```

```
%%time
np.random.seed(42)
decision_tree_pipeline = Pipeline([
        ("preparation", column_transformer_pipeline),
        ("dt",DecisionTreeClassifier(max_depth=3))
    ])
model_DT = decision_tree_pipeline.fit(X_train, y_train)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_encoders.py:868
  warnings.warn(
CPU times: user 1min 38s, sys: 9.35 s, total: 1min 48s
Wall time: 1min 47s
```

```
start = time()
decision_tree_pipeline.fit(X_train, y_train)
duration_dt_train = np.round((time() - start), 4)
np.random.seed(42)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_encoders.py:868
  warnings.warn(
```

```
# Time and score for valid predictions
start = time()
# Predict on the valid data
y_pred_valid = model_DT.predict(X_valid)

# Calculate accuracy on the valid data
logit_score_valid = accuracy_score(y_valid, y_pred_valid)
valid_time = np.round(time() - start, 4)
print(valid_time)
```

```
5.2869
```

```
# Time and score for test predictions
start = time()
# Predict on the testing data
y_pred_test = model_DT.predict(X_test)

# Calculate accuracy on the testing data
logit_score_test = accuracy_score(y_test, y_pred_test)
test_time = np.round(time() - start, 4)
print(test_time)
```

```
    4.8941
```

```
# Time and score for train predictions
start = time()
# Predict on the training data
y_pred_train = model_DT.predict(X_train)

# Calculate accuracy on the training data
logit_score_train = accuracy_score(y_train, y_pred_train)
train_time = np.round(time() - start, 4)
print(train_time)
```

```
    21.8424
```

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import log_loss
```

```
# obtained predicted probabilities or scores for train
predicted_probs = model_DT.predict_proba(X_train)[:, 1]
true_labels = y_train

# Calculate the AUC for train
auc_train = roc_auc_score(true_labels, predicted_probs)
# Calculate the log loss for train
logloss_train = log_loss(true_labels, predicted_probs)
```

```
# obtained predicted probabilities or scores for test predictions
predicted_probs = model_DT.predict_proba(X_test)[:, 1]
true_labels = y_test

# Calculate the AUC for test
auc_test = roc_auc_score(true_labels, predicted_probs)
# Calculate the log loss for test
logloss_test = log_loss(true_labels, predicted_probs)
```

```
# obtained predicted probabilities or scores for valid predictions
predicted_probs = model_DT.predict_proba(X_valid)[:, 1]
true_labels = y_valid

# Calculate the AUC for valid
auc_valid = roc_auc_score(true_labels, predicted_probs)
# Calculate the log loss for valid
logloss_valid = log_loss(true_labels, predicted_probs)
```

```
#F1-Score for test
#y_test_pred = model_DT.predict(X_test)
f1_test = f1_score(y_test, y_pred_test)
print("F1-Score for DT is: " , np.round(f1_test, 4))


#F1-Score for train
#y_train_pred = model_DT.predict(X_train)
f1_train = f1_score(y_train, y_pred_train)


#F1-Score for valid
#y_valid_pred = model_DT.predict(X_valid)
f1_valid = f1_score(y_valid, y_pred_valid)
```

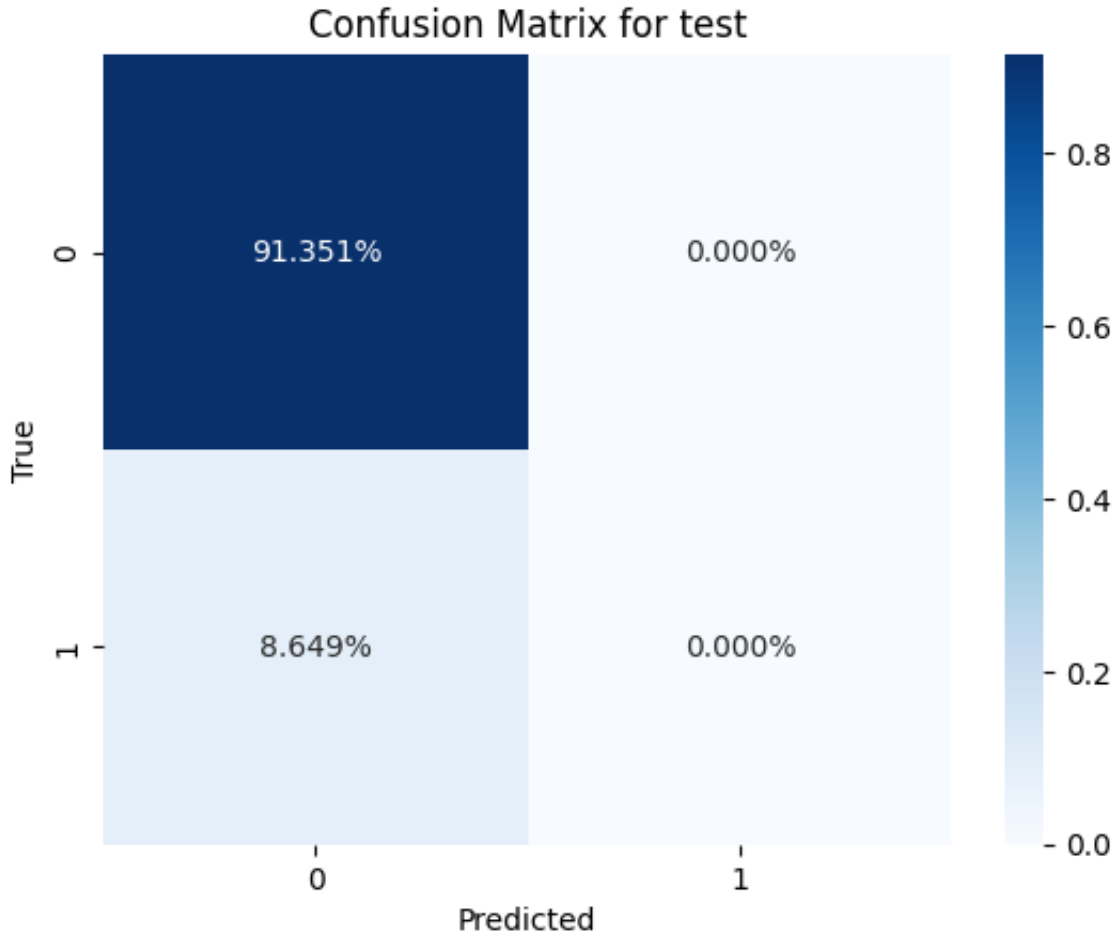
```
F1-Score for DT is:  0.0
```

```
exp_name = f"Baseline_{len(selected_features)}_features"
experimentLog = pd.DataFrame(columns=["exp_name",
                                      "Train Acc",
                                      "Valid Acc",
                                      "Test  Acc",
                                      "Train AUC",
                                      "Valid AUC",
                                      "Test  AUC",
                                      "Train F1 Score",
                                      "Valid F1 Score",
                                      "Test F1 Score",
                                      "Train Log Loss",
                                      "Valid Log Loss",
                                      "Test Log Loss",
                                      "Train Time",
                                      "Valid Time",
                                      "Test Time",
                                      "Description"
                                     ])
experimentLog.loc[len(experimentLog)] = [f"{exp_name}"],logit_score_train, logit_
#experimentLog.loc[len(experimentLog)] = ["Logistic Regression as Baseline", "HCD

experimentLog
```

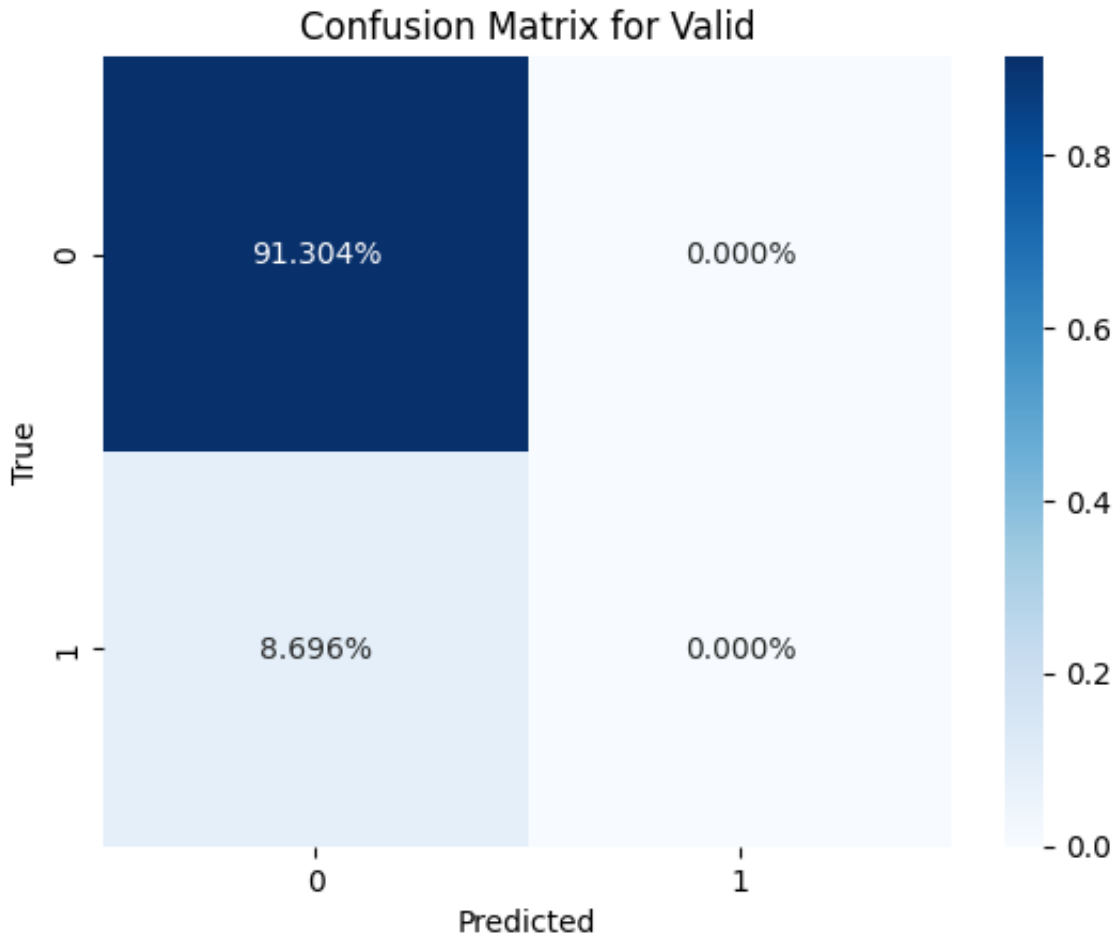
| | exp_name | Train Acc | Valid Acc | Test Acc | Train AUC | Valid AUC | Test AUC | Tra Sc |
|---|---|---|---|---|---|---|---|---|
| **0** | [Baseline_200_features] | 0.913516 | 0.913043 | 0.913512 | 0.686199 | 0.689019 | 0.685988 | |

## Metrics For Decision Tree

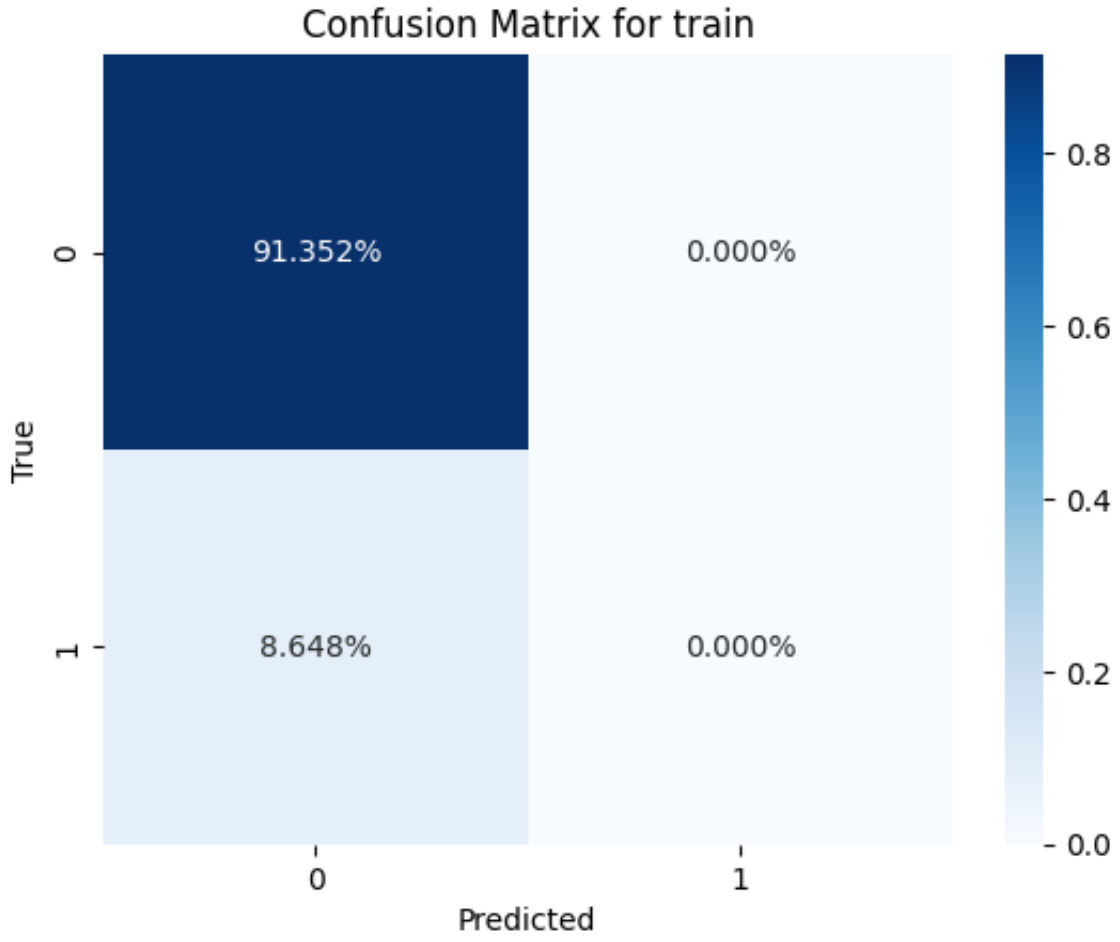
```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.pipeline import Pipeline


# Calculate the confusion matrix
cm = confusion_matrix(y_test, y_pred_test)

# Create a heatmap for the confusion matrix
sns.heatmap(cm/np.sum(cm), annot=True, fmt=".3%", cmap="Blues")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix for test')
plt.show()
```



```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.pipeline import Pipeline


# Calculate the confusion matrix
cm = confusion_matrix(y_valid, y_pred_valid)

# Create a heatmap for the confusion matrix
sns.heatmap(cm/np.sum(cm), annot=True, fmt=".3%", cmap="Blues")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix for Valid')
plt.show()
```

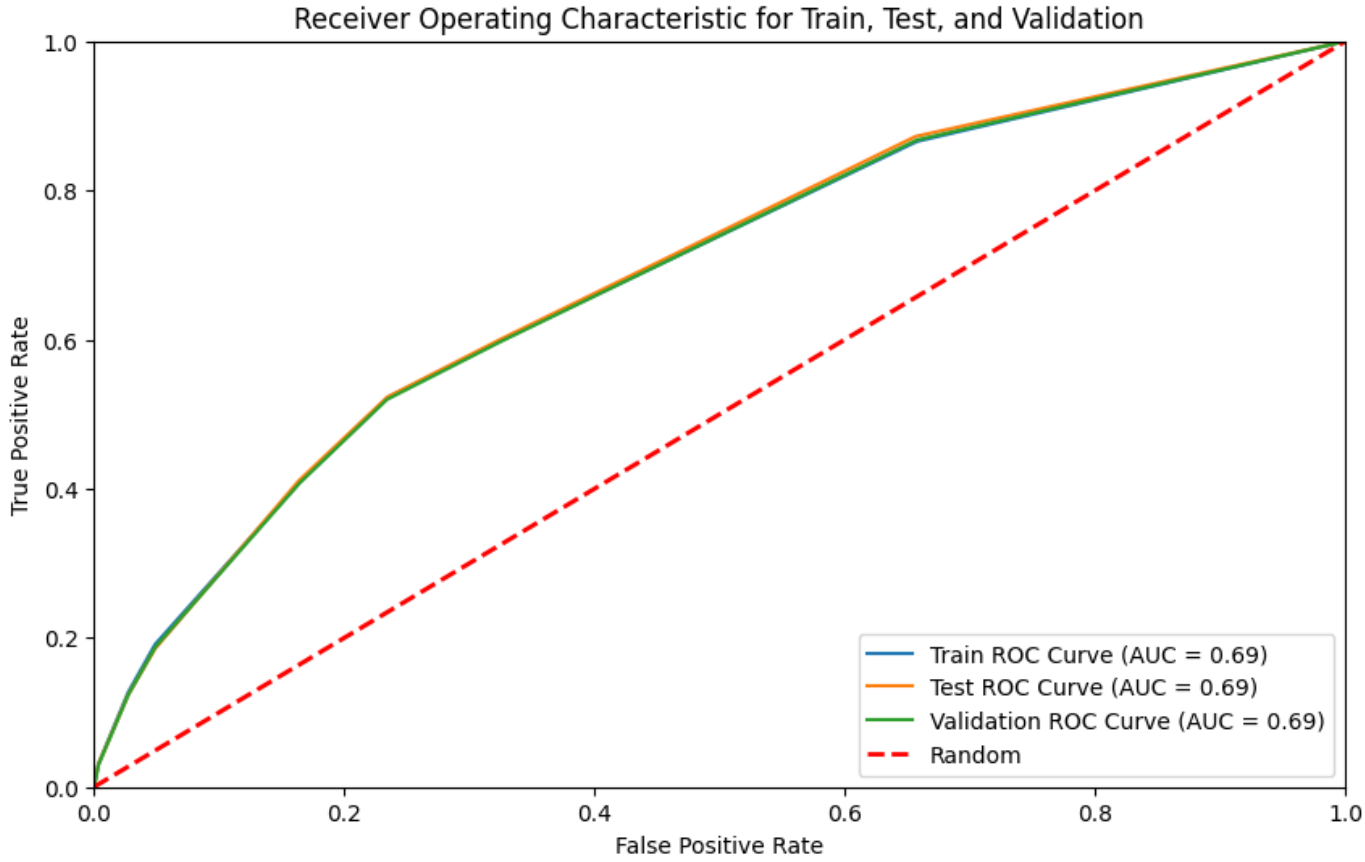


```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.pipeline import Pipeline


# Calculate the confusion matrix
cm = confusion_matrix(y_train, y_pred_train)

# Create a heatmap for the confusion matrix
sns.heatmap(cm/np.sum(cm), annot=True, fmt=".3%", cmap="Blues")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix for train')
plt.show()
```



## ROC CURVE

```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc

# Obtain predicted probabilities for the positive class for each dataset
train_predicted_probs = model_DT.predict_proba(X_train)[:, 1]
test_predicted_probs = model_DT.predict_proba(X_test)[:, 1]
valid_predicted_probs = model_DT.predict_proba(X_valid)[:, 1]

# Assuming y_train, y_test, y_valid contain the true labels for your train, test,
train_true_labels = y_train
test_true_labels = y_test
valid_true_labels = y_valid

# Calculate the false positive rate, true positive rate, and thresholds for each
train_fpr, train_tpr, _ = roc_curve(train_true_labels, train_predicted_probs)
test_fpr, test_tpr, _ = roc_curve(test_true_labels, test_predicted_probs)
valid_fpr, valid_tpr, _ = roc_curve(valid_true_labels, valid_predicted_probs)

# Calculate the area under the ROC curve for each dataset
train_roc_auc = auc(train_fpr, train_tpr)
test_roc_auc = auc(test_fpr, test_tpr)
valid_roc_auc = auc(valid_fpr, valid_tpr)

# Plot the ROC curves for train, test, and validation datasets on a single plot
plt.figure(figsize=(10, 6))
plt.plot(train_fpr, train_tpr, label='Train ROC Curve (AUC = {:.2f})'.format(trai
plt.plot(test_fpr, test_tpr, label='Test ROC Curve (AUC = {:.2f})'.format(test_ro
plt.plot(valid_fpr, valid_tpr, label='Validation ROC Curve (AUC = {:.2f})'.format
plt.plot([0, 1], [0, 1], 'r--', label='Random', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic for Train, Test, and Validation')
plt.legend(loc='lower right')
plt.show()
```




## Random Forest PIPELINE

```
from sklearn.ensemble import RandomForestClassifier
```

```
rf_pipeline = Pipeline([
        ("preparation", column_transformer_pipeline),
        ("classifier",RandomForestClassifier(max_depth=2))
    ])
start = time()
model_RF = rf_pipeline.fit(X_train, y_train)
duration_dt_train = np.round((time() - start), 4)
```

```
/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_encoders.py:868
  warnings.warn(
```

```
# Time and score for valid predictions
start = time()
# Predict on the valid data
y_pred_valid = model_RF.predict(X_valid)

# Calculate accuracy on the valid data
logit_score_valid = accuracy_score(y_valid, y_pred_valid)
valid_time = np.round(time() - start, 4)
print(valid_time)
```

```
9.5556
```

```
# Time and score for test predictions
start = time()
# Predict on the testing data
y_pred_test = model_RF.predict(X_test)

# Calculate accuracy on the testing data
logit_score_test = accuracy_score(y_test, y_pred_test)
test_time = np.round(time() - start, 4)
print(test_time)
```

```
6.4537
```

```
# Time and score for train predictions
start = time()
# Predict on the training data
y_pred_train = model_RF.predict(X_train)

# Calculate accuracy on the training data
logit_score_train = accuracy_score(y_train, y_pred_train)
train_time = np.round(time() - start, 4)
print(train_time)
```

```
    36.3243
```

```
from sklearn.metrics import roc_auc_score
from sklearn.metrics import log_loss
```

```
# obtained predicted probabilities or scores for train
predicted_probs = model_RF.predict_proba(X_train)[:, 1]
true_labels = y_train

# Calculate the AUC for train
auc_train = roc_auc_score(true_labels, predicted_probs)
# Calculate the log loss for train
logloss_train = log_loss(true_labels, predicted_probs)
```

```
# obtained predicted probabilities or scores for test predictions
predicted_probs = model_RF.predict_proba(X_test)[:, 1]
true_labels = y_test

# Calculate the AUC for test
auc_test = roc_auc_score(true_labels, predicted_probs)
# Calculate the log loss for test
logloss_test = log_loss(true_labels, predicted_probs)
```

```
# obtained predicted probabilities or scores for valid predictions
predicted_probs = model_RF.predict_proba(X_valid)[:, 1]
true_labels = y_valid

# Calculate the AUC for valid
auc_valid = roc_auc_score(true_labels, predicted_probs)
# Calculate the log loss for valid
logloss_valid = log_loss(true_labels, predicted_probs)
```

```
#F1-Score for test
#y_test_pred = model_DT.predict(X_test)
f1_test = f1_score(y_test, y_pred_test)
print("F1-Score for DT is: " , np.round(f1_test, 4))


#F1-Score for train
#y_train_pred = model_DT.predict(X_train)
f1_train = f1_score(y_train, y_pred_train)


#F1-Score for valid
#y_valid_pred = model_DT.predict(X_valid)
f1_valid = f1_score(y_valid, y_pred_valid)
```

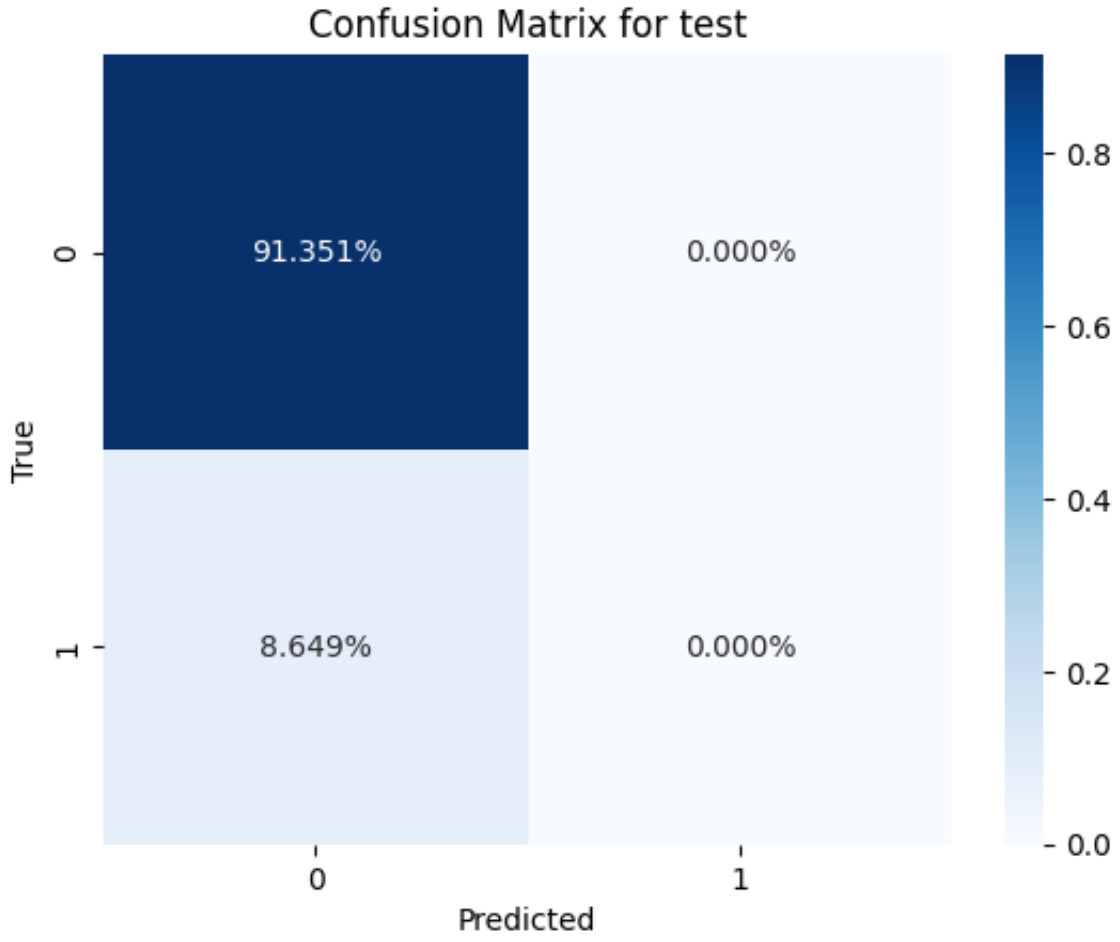
```
F1-Score for DT is:  0.0
```

```
exp_name = f"Baseline_{len(selected_features)}_features"
experimentLog = pd.DataFrame(columns=["exp_name",
                                      "Train Acc",
                                      "Valid Acc",
                                      "Test  Acc",
                                      "Train AUC",
                                      "Valid AUC",
                                      "Test  AUC",
                                      "Train F1 Score",
                                      "Valid F1 Score",
                                      "Test F1 Score",
                                      "Train Log Loss",
                                      "Valid Log Loss",
                                      "Test Log Loss",
                                      "Train Time",
                                      "Valid Time",
                                      "Test Time",
                                      "Description"
                                     ])
experimentLog.loc[len(experimentLog)] = [f"{exp_name}"],logit_score_train, logit_
#experimentLog.loc[len(experimentLog)] = ["Logistic Regression as Baseline", "HCD

experimentLog
```

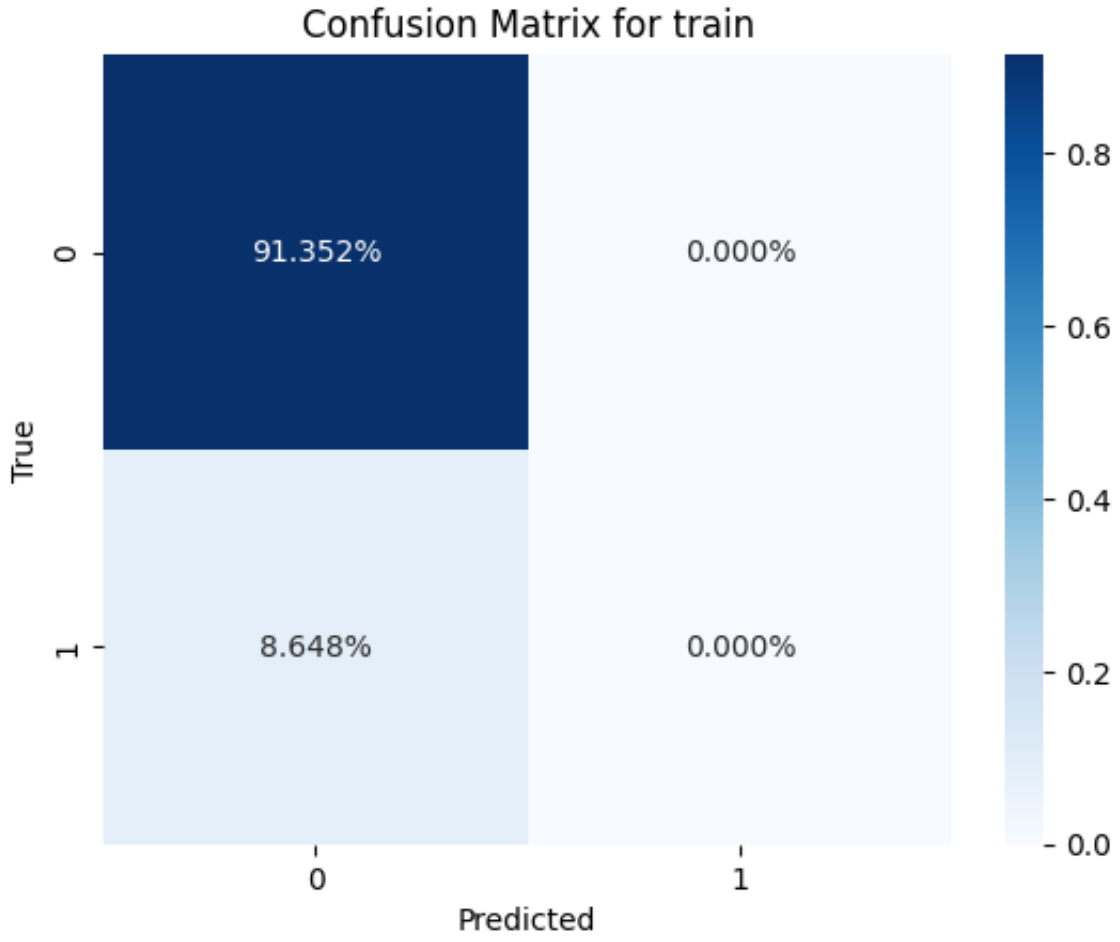
| | exp_name | Train Acc | Valid Acc | Test Acc | Train AUC | Valid AUC | Test AUC | Tra Sco |
|---|---|---|---|---|---|---|---|---|
| **0** | [Baseline_200_features] | 0.913516 | 0.913043 | 0.913512 | 0.701011 | 0.697197 | 0.699813 | |

## Metrics for Random Forest

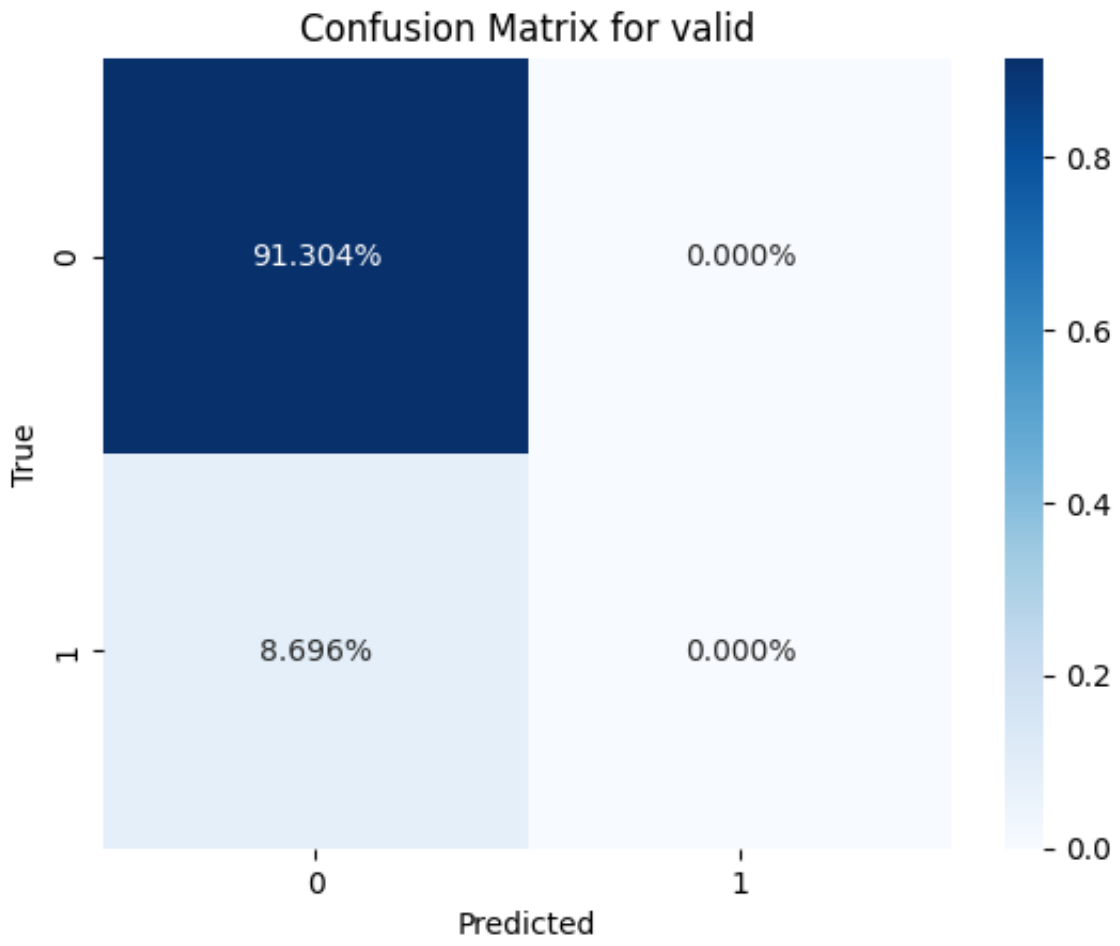
```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.pipeline import Pipeline


# Calculate the confusion matrix
cm = confusion_matrix(y_test, y_pred_test)

# Create a heatmap for the confusion matrix
sns.heatmap(cm/np.sum(cm), annot=True, fmt=".3%", cmap="Blues")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix for test')
plt.show()
```



```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.pipeline import Pipeline


# Calculate the confusion matrix
cm = confusion_matrix(y_train, y_pred_train)

# Create a heatmap for the confusion matrix
sns.heatmap(cm/np.sum(cm), annot=True, fmt=".3%", cmap="Blues")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix for train')
plt.show()
```

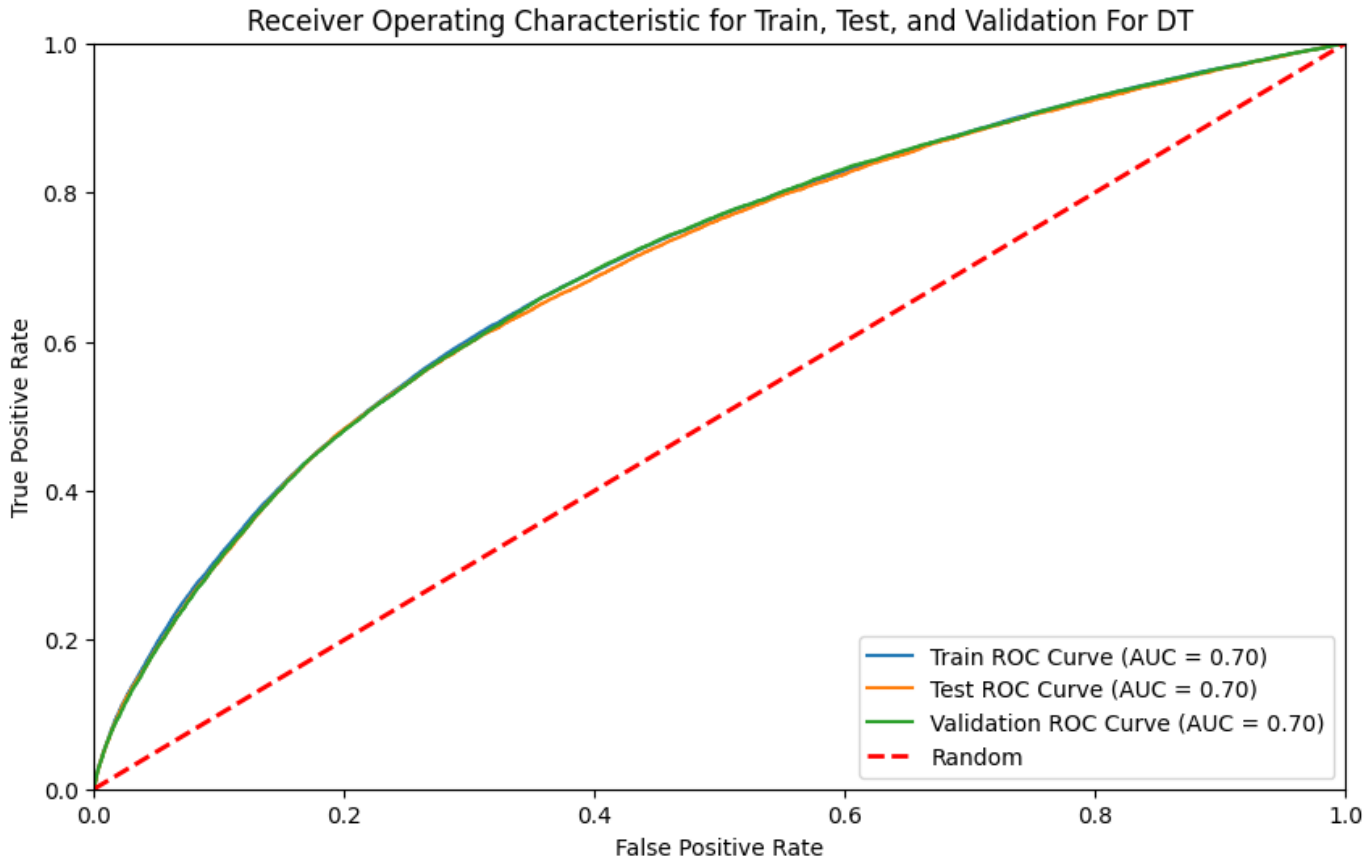


```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.pipeline import Pipeline


# Calculate the confusion matrix
cm = confusion_matrix(y_valid, y_pred_valid)

# Create a heatmap for the confusion matrix
sns.heatmap(cm/np.sum(cm), annot=True, fmt=".3%", cmap="Blues")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix for valid')
plt.show()
```




```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
```

```
# Obtain predicted probabilities for the positive class for each dataset
train_predicted_probs = model_RF.predict_proba(X_train)[:, 1]
test_predicted_probs = model_RF.predict_proba(X_test)[:, 1]
valid_predicted_probs = model_RF.predict_proba(X_valid)[:, 1]

# Assuming y_train, y_test, y_valid contain the true labels for your train, test,
train_true_labels = y_train
test_true_labels = y_test
valid_true_labels = y_valid

# Calculate the false positive rate, true positive rate, and thresholds for each
train_fpr, train_tpr, _ = roc_curve(train_true_labels, train_predicted_probs)
test_fpr, test_tpr, _ = roc_curve(test_true_labels, test_predicted_probs)
valid_fpr, valid_tpr, _ = roc_curve(valid_true_labels, valid_predicted_probs)

# Calculate the area under the ROC curve for each dataset
train_roc_auc = auc(train_fpr, train_tpr)
test_roc_auc = auc(test_fpr, test_tpr)
valid_roc_auc = auc(valid_fpr, valid_tpr)

# Plot the ROC curves for train, test, and validation datasets on a single plot
plt.figure(figsize=(10, 6))
plt.plot(train_fpr, train_tpr, label='Train ROC Curve (AUC = {:.2f})'.format(trai
plt.plot(test_fpr, test_tpr, label='Test ROC Curve (AUC = {:.2f})'.format(test_rc
plt.plot(valid_fpr, valid_tpr, label='Validation ROC Curve (AUC = {:.2f})'.format
plt.plot([0, 1], [0, 1], 'r--', label='Random', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic for Train, Test, and Validation For
plt.legend(loc='lower right')
plt.show()
```




## Doing experiment with different features

```
# Split the provided training data into training and validationa and test
# The kaggle evaluation test set has no labels
#
from sklearn.model_selection import train_test_split

use_application_data_ONLY = True #use joined data
if use_application_data_ONLY:
    # just selected a few features for a baseline experiment
    selected_features = ['AMT_INCOME_TOTAL',  'AMT_CREDIT','DAYS_EMPLOYED','DAYS_
        'EXT_SOURCE_2','EXT_SOURCE_3','CODE_GENDER', 'FLAG_OWN_REALTY','FLAG_OWN_
                  'NAME_EDUCATION_TYPE','OCCUPATION_TYPE','NAME_INCOME_TYPE']
    X_train = datasets["application_train"][selected_features]
    y_train = datasets["application_train"]['TARGET']
    X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, test_
    X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test_si
    X_kaggle_test= datasets["application_test"][selected_features]
    # y_test = datasets["application_test"]['TARGET']   #why no  TARGET?!! (hint:

X_train = datasets["application_train"]
selected_features = ['AMT_INCOME_TOTAL',  'AMT_CREDIT','DAYS_EMPLOYED','DAYS_BIRT
        'EXT_SOURCE_2','EXT_SOURCE_3','CODE_GENDER', 'FLAG_OWN_REALTY','FLAG_OWN_
                  'NAME_EDUCATION_TYPE','OCCUPATION_TYPE','NAME_INCOME_TYPE']
y_train = X_train['TARGET']
X_train = X_train[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]
# y_test = datasets["application_test"]['TARGET']   #why no  TARGET?!! (hint: kag


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: (222176, 14)
X validation      shape: (46127, 14)
X test            shape: (39208, 14)
X X_kaggle_test   shape: (48744, 14)
```

```
# Create a class to select numerical or categorical columns
# since Scikit-Learn doesn't handle DataFrames yet
class DataFrameSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attribute_names):
        self.attribute_names = attribute_names
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.attribute_names].values
```

```
# Identify the numeric features we wish to consider.
num_attribs = [
    'AMT_INCOME_TOTAL',  'AMT_CREDIT','DAYS_EMPLOYED','DAYS_BIRTH','EXT_SOURCE_1'
    'EXT_SOURCE_2','EXT_SOURCE_3']

num_pipeline = Pipeline([
        ('selector', DataFrameSelector(num_attribs)),
        ('imputer', SimpleImputer(strategy='mean')),
        ('std_scaler', StandardScaler()),
    ])
# Identify the categorical features we wish to consider.
cat_attribs = ['CODE_GENDER', 'FLAG_OWN_REALTY','FLAG_OWN_CAR','NAME_CONTRACT_TYP
               'NAME_EDUCATION_TYPE','OCCUPATION_TYPE','NAME_INCOME_TYPE']

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

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

```
list(datasets["application_train"].columns)
```

```
['SK_ID_CURR',
 'TARGET',
 'NAME_CONTRACT_TYPE',
 'CODE_GENDER',
 'FLAG_OWN_CAR',
 'FLAG_OWN_REALTY',
 'CNT_CHILDREN',
 'AMT_INCOME_TOTAL',
 'AMT_CREDIT',
 'AMT_ANNUITY',
 'AMT_GOODS_PRICE',
 'NAME_TYPE_SUITE',
 'NAME_INCOME_TYPE',
 'NAME_EDUCATION_TYPE',
 'NAME_FAMILY_STATUS',
 'NAME_HOUSING_TYPE',
 'REGION_POPULATION_RELATIVE',
 'DAYS_BIRTH',
 'DAYS_EMPLOYED',
 'DAYS_REGISTRATION',
 'DAYS_ID_PUBLISH',
 'OWN_CAR_AGE',
 'FLAG_MOBIL',
 'FLAG_EMP_PHONE',
 'FLAG_WORK_PHONE',
 'FLAG_CONT_MOBILE',
 'FLAG_PHONE',
 'FLAG_EMAIL',
 'OCCUPATION_TYPE',
 'CNT_FAM_MEMBERS',
 'REGION_RATING_CLIENT',
 'REGION_RATING_CLIENT_W_CITY',
 'WEEKDAY_APPR_PROCESS_START',
 'HOUR_APPR_PROCESS_START',
 'REG_REGION_NOT_LIVE_REGION',
 'REG_REGION_NOT_WORK_REGION',
 'LIVE_REGION_NOT_WORK_REGION',
 'REG_CITY_NOT_LIVE_CITY',
 'REG_CITY_NOT_WORK_CITY',
 'LIVE_CITY_NOT_WORK_CITY',
 'ORGANIZATION_TYPE',
 'EXT_SOURCE_1',
 'EXT_SOURCE_2',
 'EXT_SOURCE_3',
 'APARTMENTS_AVG',
 'BASEMENTAREA_AVG',
 'YEARS_BEGINEXPLUATATION_AVG',
 'YEARS_BUILD_AVG',
 'COMMONAREA_AVG',
 'ELEVATORS_AVG',
 'ENTRANCES_AVG',
```

```
        'ENTRANCES_AVG',
        'FLOORSMAX_AVG',
        'FLOORSMIN_AVG',
        'LANDAREA_AVG',
        'LIVINGAPARTMENTS_AVG',
        'LIVINGAREA_AVG',
        'NONLIVINGAPARTMENTS_AVG',
        'NONLIVINGAREA_AVG',
        'APARTMENTS_MODE',
        'BASEMENTAREA_MODE'
```

```
selected_features = num_attribs + cat_attribs
tot_features = f"{len(selected_features)}:   Num:{len(num_attribs)},    Cat:{len(
#Total Feature selected for processing
tot_features
```

```
'14:   Num:7,    Cat:7'
```

```
%%time
np.random.seed(42)
logistic_pipeline = Pipeline([
        ("preparation", data_prep_pipeline),# combination of numerical, categoric
        #("clf", MultinomialNB())  # classifier estimator you are using
        ("logistic_regression", LogisticRegression() )
    ])
```

```
CPU times: user 103 µs, sys: 19 µs, total: 122 µs
Wall time: 127 µs
```

```
start = time()
logistic_pipeline.fit(X_train, y_train)
duration_logistic_train = np.round((time() - start), 4)
np.random.seed(42)
```

```
# Time and score for valid predictions
start = time()
logit_score_valid  = logistic_pipeline.score(X_valid, y_valid)
valid_time = np.round(time() - start, 4)
```

```
# Time and score for valid predictions
start = time()
logit_score_valid  = logistic_pipeline.score(X_valid, y_valid)
valid_time = np.round(time() - start, 4)
```

```
# Time and score for test predictions
start = time()
logit_score_test  = logistic_pipeline.score(X_test, y_test)
test_time = np.round(time() - start, 4)

# Time and score for train predictions
start = time()
logit_score_train  = logistic_pipeline.score(X_train, y_train)
train_time = np.round(time() - start, 4)


from sklearn.metrics import roc_auc_score
from sklearn.metrics import log_loss

# obtained predicted probabilities or scores for train
predicted_probs = logistic_pipeline.predict_proba(X_train)[:, 1]
true_labels = y_train

# Calculate the AUC for train
auc_train = roc_auc_score(true_labels, predicted_probs)
# Calculate the log loss for train
logloss_train = log_loss(true_labels, predicted_probs)


# obtained predicted probabilities or scores for test predictions
predicted_probs = logistic_pipeline.predict_proba(X_test)[:, 1]
true_labels = y_test

# Calculate the AUC for test
auc_test = roc_auc_score(true_labels, predicted_probs)
# Calculate the log loss for test
logloss_test = log_loss(true_labels, predicted_probs)


# obtained predicted probabilities or scores for valid predictions
predicted_probs = logistic_pipeline.predict_proba(X_valid)[:, 1]
true_labels = y_valid

# Calculate the AUC for test
auc_valid = roc_auc_score(true_labels, predicted_probs)
# Calculate the log loss for valid
logloss_valid = log_loss(true_labels, predicted_probs)


#F1-Score for test
```

```
y_test_pred = logistic_pipeline.predict(X_test)
f1_test = f1_score(y_test, y_test_pred)
print("F1-Score for Logistic Regression is: " , np.round(f1_test, 4))


#F1-Score for train
y_train_pred = logistic_pipeline.predict(X_train)
f1_train = f1_score(y_train, y_train_pred)


#F1-Score for valid
y_valid_pred = logistic_pipeline.predict(X_valid)
f1_valid = f1_score(y_valid, y_valid_pred)
```

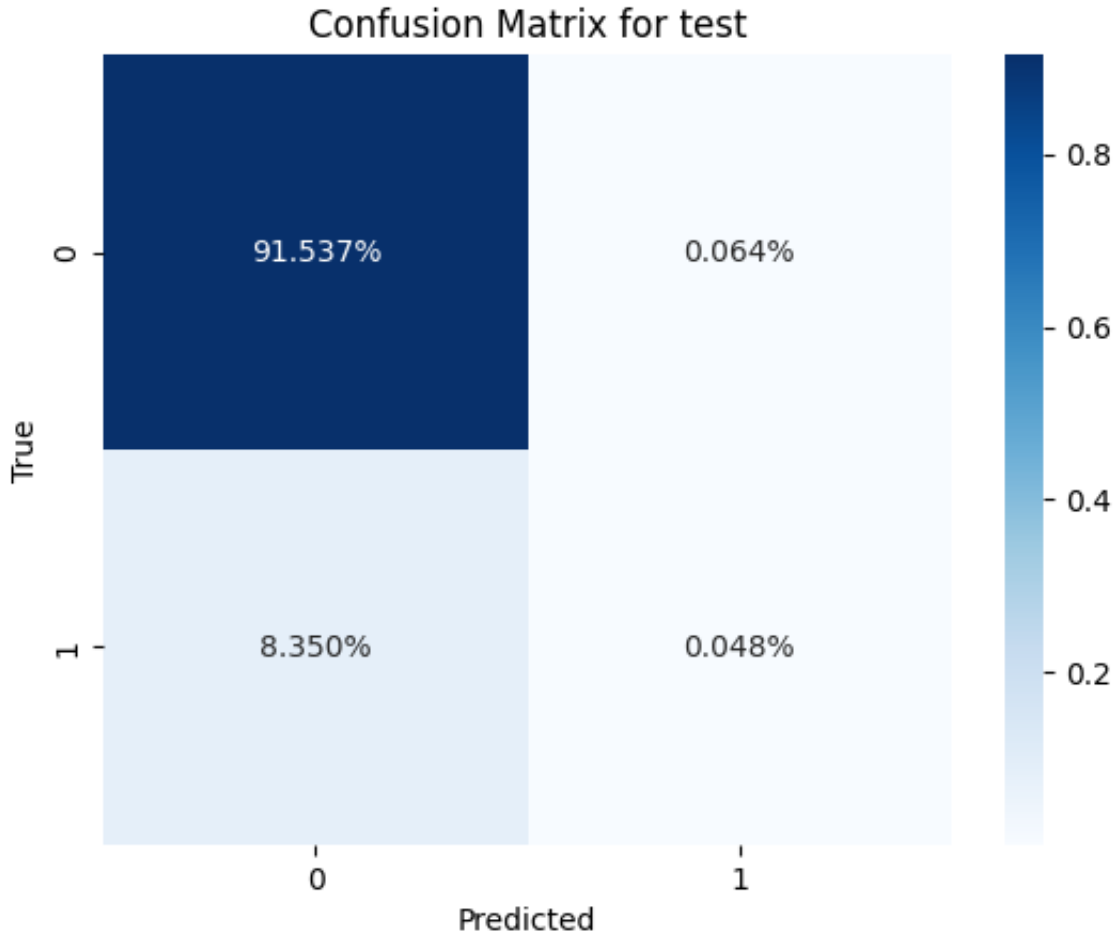
```
F1-Score for Logistic Regression is:  0.0114
```

```
exp_name = f"Baseline_{len(selected_features)}_features"
experimentLog.loc[len(experimentLog)] = [f"{exp_name}"],logit_score_train, logit_
#experimentLog.loc[len(experimentLog)] = ["Logistic Regression as Baseline", "HCD

experimentLog
```

| | exp_name | Train Acc | Valid Acc | Test Acc | Train AUC | Valid AUC | Test AUC | T s |
|---|---|---|---|---|---|---|---|---|
| **0** | [Baseline_200_features] | 0.913516 | 0.913043 | 0.913512 | 0.701011 | 0.697197 | 0.699813 | 0.0 |
| **1** | [Baseline_14_features] | 0.919780 | 0.919266 | 0.915859 | 0.735710 | 0.735608 | 0.735645 | 0.0 |

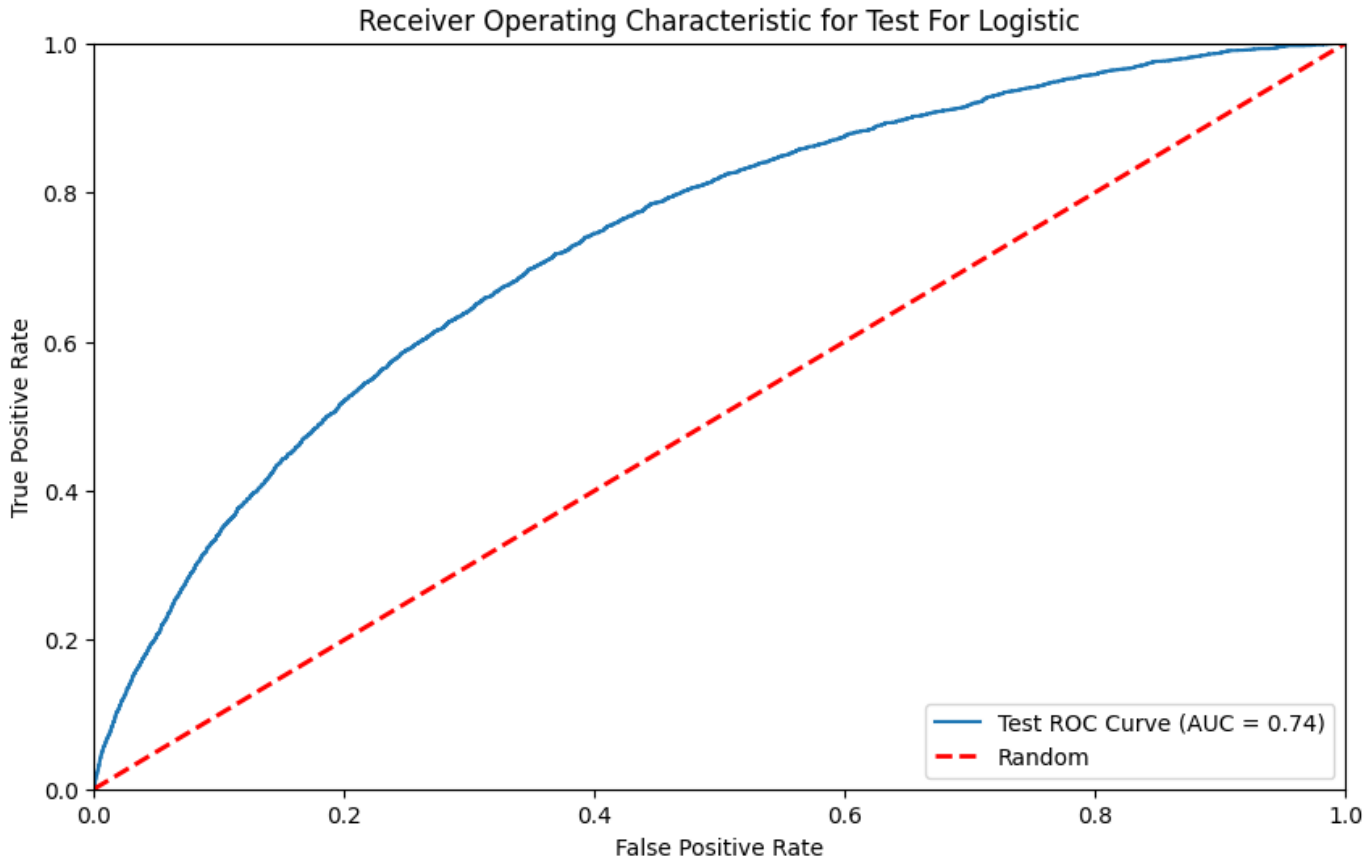
### Metrics for 14 features Logistic Regression

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.pipeline import Pipeline


# Calculate the confusion matrix
cm = confusion_matrix(y_test, y_test_pred)

# Create a heatmap for the confusion matrix
sns.heatmap(cm/np.sum(cm), annot=True, fmt=".3%", cmap="Blues")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix for test')
plt.show()
```




```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
```

```
# Obtain predicted probabilities for the positive class for each dataset
test_predicted_probs = logistic_pipeline.predict_proba(X_test)[:, 1]

test_true_labels = y_test

# Calculate the false positive rate, true positive rate, and thresholds for each
test_fpr, test_tpr, _ = roc_curve(test_true_labels, test_predicted_probs)

# Calculate the area under the ROC curve for each dataset
test_roc_auc = auc(test_fpr, test_tpr)

# Plot the ROC curves for train, test, and validation datasets on a single plot
plt.figure(figsize=(10, 6))
plt.plot(test_fpr, test_tpr, label='Test ROC Curve (AUC = {:.2f})'.format(test_rc
plt.plot([0, 1], [0, 1], 'r--', label='Random', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic for Test For Logistic')
plt.legend(loc='lower right')
plt.show()
```




## Decision Tree for 14 features

```
from sklearn.tree import DecisionTreeClassifier
```

```
%%time
np.random.seed(42)
decision_tree_pipeline = Pipeline([
        ("preparation", data_prep_pipeline),
        ("dt",DecisionTreeClassifier(max_depth=3))
    ])
model_DT = decision_tree_pipeline.fit(X_train, y_train)
```

```
CPU times: user 1.75 s, sys: 2.69 ms, total: 1.76 s
Wall time: 1.75 s
```

```
start = time()
decision_tree_pipeline.fit(X_train, y_train)
duration_dt_train = np.round((time() - start), 4)
np.random.seed(42)
```

```
# Time and score for valid predictions
start = time()
logit_score_valid  = decision_tree_pipeline.score(X_valid, y_valid)
valid_time = np.round(time() - start, 4)

# Time and score for test predictions
start = time()
logit_score_test  = decision_tree_pipeline.score(X_test, y_test)
test_time = np.round(time() - start, 4)

# Time and score for train predictions
start = time()
logit_score_train  = decision_tree_pipeline.score(X_train, y_train)
train_time = np.round(time() - start, 4)



from sklearn.metrics import roc_auc_score
from sklearn.metrics import log_loss

# obtained predicted probabilities or scores for train
predicted_probs = decision_tree_pipeline.predict_proba(X_train)[:, 1]
true_labels = y_train

# Calculate the AUC for train
auc_train = roc_auc_score(true_labels, predicted_probs)
# Calculate the log loss for train
logloss_train = log_loss(true_labels, predicted_probs)
```

```
# obtained predicted probabilities or scores for test predictions
predicted_probs = decision_tree_pipeline.predict_proba(X_test)[:, 1]
true_labels = y_test

# Calculate the AUC for test
auc_test = roc_auc_score(true_labels, predicted_probs)
# Calculate the log loss for test
logloss_test = log_loss(true_labels, predicted_probs)


# obtained predicted probabilities or scores for valid predictions
predicted_probs = decision_tree_pipeline.predict_proba(X_valid)[:, 1]
true_labels = y_valid

# Calculate the AUC for test
auc_valid = roc_auc_score(true_labels, predicted_probs)
# Calculate the log loss for valid
logloss_valid = log_loss(true_labels, predicted_probs)


#F1-Score for test
y_test_pred = decision_tree_pipeline.predict(X_test)
f1_test = f1_score(y_test, y_test_pred)
print("F1-Score for DT is: " , np.round(f1_test, 4))


#F1-Score for train
y_train_pred = decision_tree_pipeline.predict(X_train)
f1_train = f1_score(y_train, y_train_pred)


#F1-Score for valid
y_valid_pred = decision_tree_pipeline.predict(X_valid)
f1_valid = f1_score(y_valid, y_valid_pred)
```

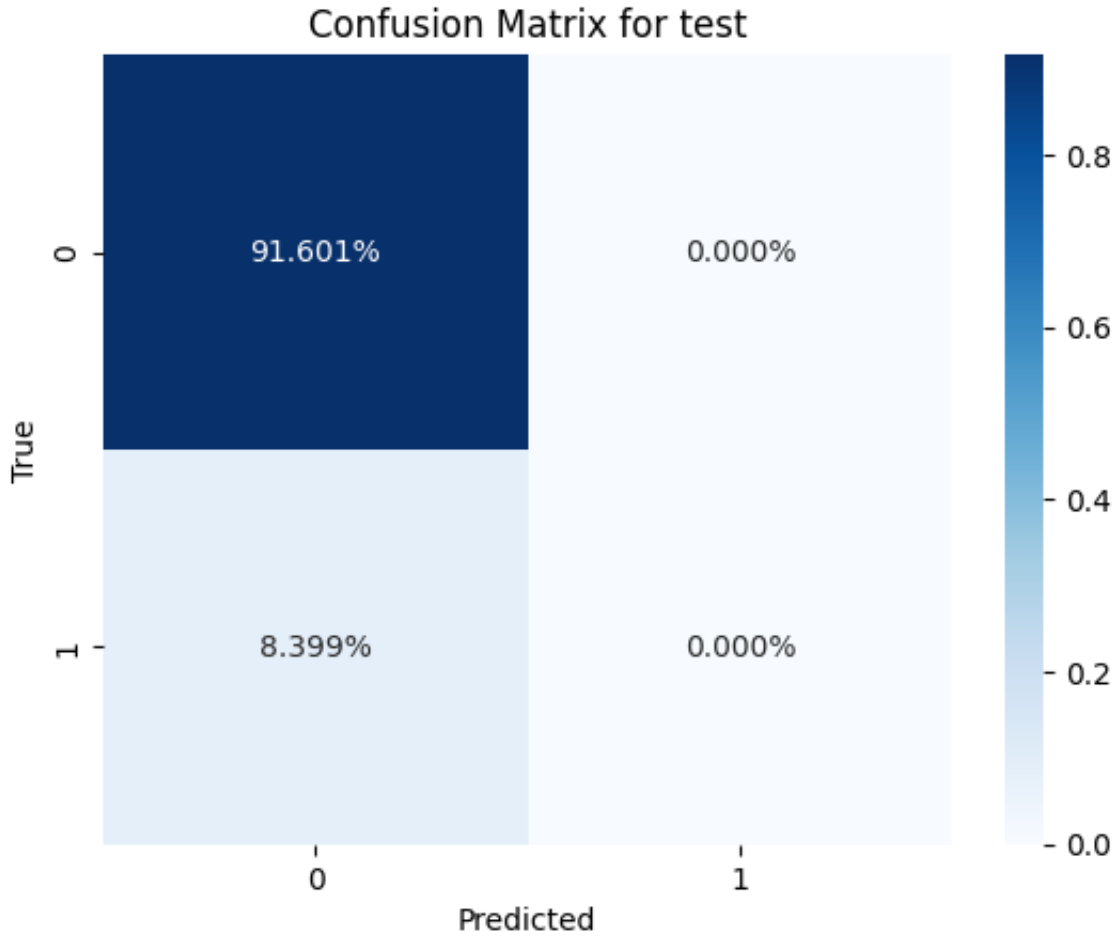
```
F1-Score for DT is:  0.0
```

```
exp_name = f"Baseline_{len(selected_features)}_features"
experimentLog.loc[len(experimentLog)] = [f"{exp_name}"],logit_score_train, logit_
#experimentLog.loc[len(experimentLog)] = ["Logistic Regression as Baseline", "HCD

experimentLog
```

| | exp_name | Train Acc | Valid Acc | Test Acc | Train AUC | Valid AUC | Test AUC | T s |
|---|---|---|---|---|---|---|---|---|
| **0** | [Baseline_200_features] | 0.913516 | 0.913043 | 0.913512 | 0.701011 | 0.697197 | 0.699813 | 0.0 |
| **1** | [Baseline_14_features] | 0.919780 | 0.919266 | 0.915859 | 0.735710 | 0.735608 | 0.735645 | 0.0 |
| **2** | [Baseline_14_features] | 0.919816 | 0.919418 | 0.916012 | 0.689067 | 0.682330 | 0.683875 | 0.0 |

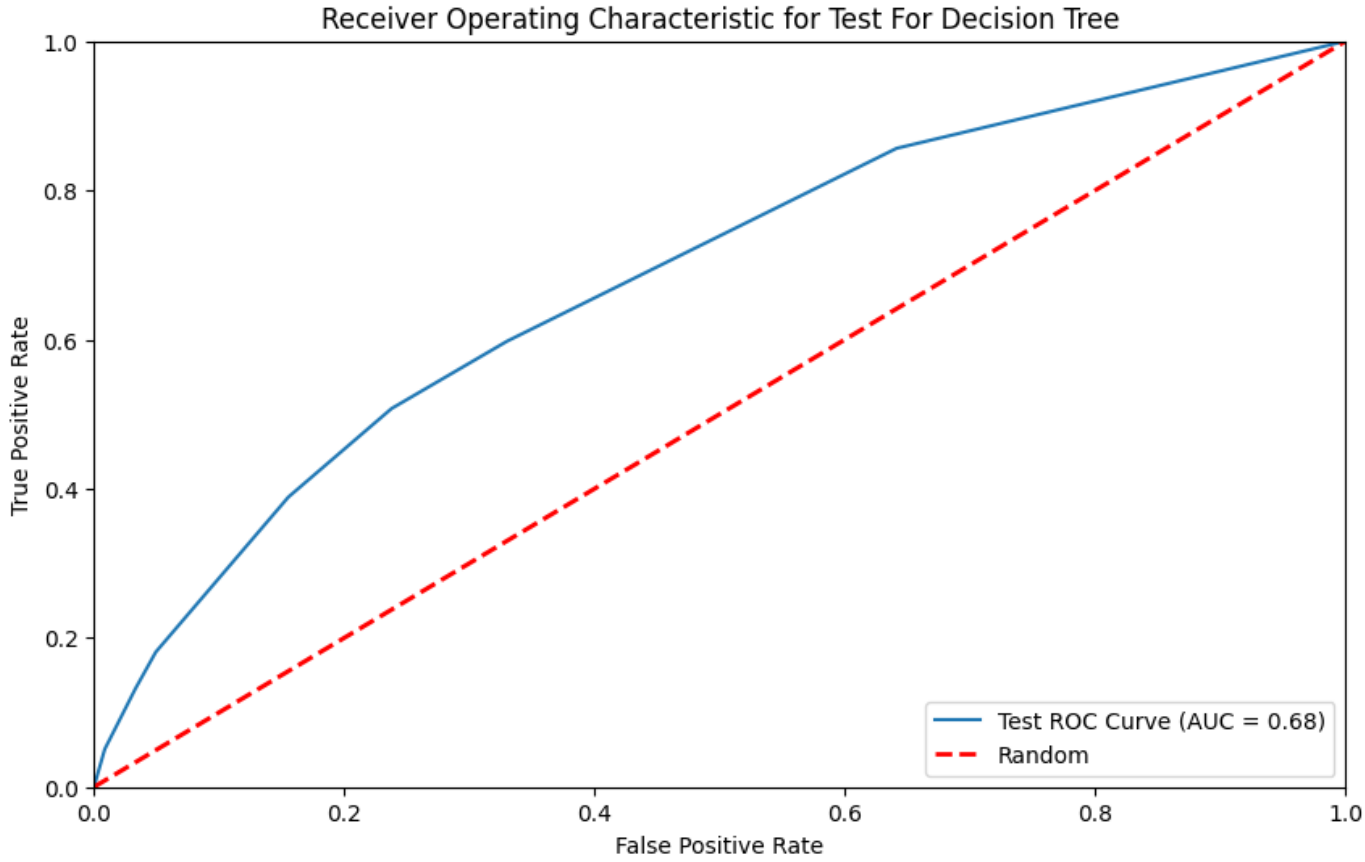
## Metrics for DT

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.pipeline import Pipeline


# Calculate the confusion matrix
cm = confusion_matrix(y_test, y_test_pred)

# Create a heatmap for the confusion matrix
sns.heatmap(cm/np.sum(cm), annot=True, fmt=".3%", cmap="Blues")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix for test')
plt.show()
```




```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
```

```
# Obtain predicted probabilities for the positive class for each dataset
test_predicted_probs = decision_tree_pipeline.predict_proba(X_test)[:, 1]

test_true_labels = y_test

# Calculate the false positive rate, true positive rate, and thresholds for each
test_fpr, test_tpr, _ = roc_curve(test_true_labels, test_predicted_probs)

# Calculate the area under the ROC curve for each dataset
test_roc_auc = auc(test_fpr, test_tpr)

# Plot the ROC curves for train, test, and validation datasets on a single plot
plt.figure(figsize=(10, 6))
plt.plot(test_fpr, test_tpr, label='Test ROC Curve (AUC = {:.2f})'.format(test_rc
plt.plot([0, 1], [0, 1], 'r--', label='Random', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic for Test For Decision Tree')
plt.legend(loc='lower right')
plt.show()
```




## Random Forest for 14 features

```
from sklearn.ensemble import RandomForestClassifier
```

```
rf_pipeline = Pipeline([
        ("preparation", data_prep_pipeline),
        ("classifier",RandomForestClassifier(max_depth=2))
    ])
start = time()
model_RF = rf_pipeline.fit(X_train, y_train)
duration_dt_train = np.round((time() - start), 4)
```

```
start = time()
rf_pipeline.fit(X_train, y_train)
duration_rf_train = np.round((time() - start), 4)
np.random.seed(42)
```

```
# Time and score for valid predictions
start = time()
logit_score_valid  = rf_pipeline.score(X_valid, y_valid)
valid_time = np.round(time() - start, 4)

# Time and score for test predictions
start = time()
logit_score_test  = rf_pipeline.score(X_test, y_test)
test_time = np.round(time() - start, 4)

# Time and score for train predictions
start = time()
logit_score_train  = rf_pipeline.score(X_train, y_train)
train_time = np.round(time() - start, 4)


from sklearn.metrics import roc_auc_score
from sklearn.metrics import log_loss

# obtained predicted probabilities or scores for train
predicted_probs = rf_pipeline.predict_proba(X_train)[:, 1]
true_labels = y_train

# Calculate the AUC for train
auc_train = roc_auc_score(true_labels, predicted_probs)
# Calculate the log loss for train
logloss_train = log_loss(true_labels, predicted_probs)


# obtained predicted probabilities or scores for test predictions
```

```
predicted_probs = rf_pipeline.predict_proba(X_test)[:, 1]
true_labels = y_test

# Calculate the AUC for test
auc_test = roc_auc_score(true_labels, predicted_probs)
# Calculate the log loss for test
logloss_test = log_loss(true_labels, predicted_probs)


# obtained predicted probabilities or scores for valid predictions
predicted_probs = rf_pipeline.predict_proba(X_valid)[:, 1]
true_labels = y_valid

# Calculate the AUC for test
auc_valid = roc_auc_score(true_labels, predicted_probs)
# Calculate the log loss for valid
logloss_valid = log_loss(true_labels, predicted_probs)


#F1-Score for test
y_test_pred1 = rf_pipeline.predict(X_test)
f1_test = f1_score(y_test, y_test_pred1)
print("F1-Score for DT is: " , np.round(f1_test, 4))


#F1-Score for train
y_train_pred = rf_pipeline.predict(X_train)
f1_train = f1_score(y_train, y_train_pred)


#F1-Score for valid
y_valid_pred = rf_pipeline.predict(X_valid)
f1_valid = f1_score(y_valid, y_valid_pred)
```

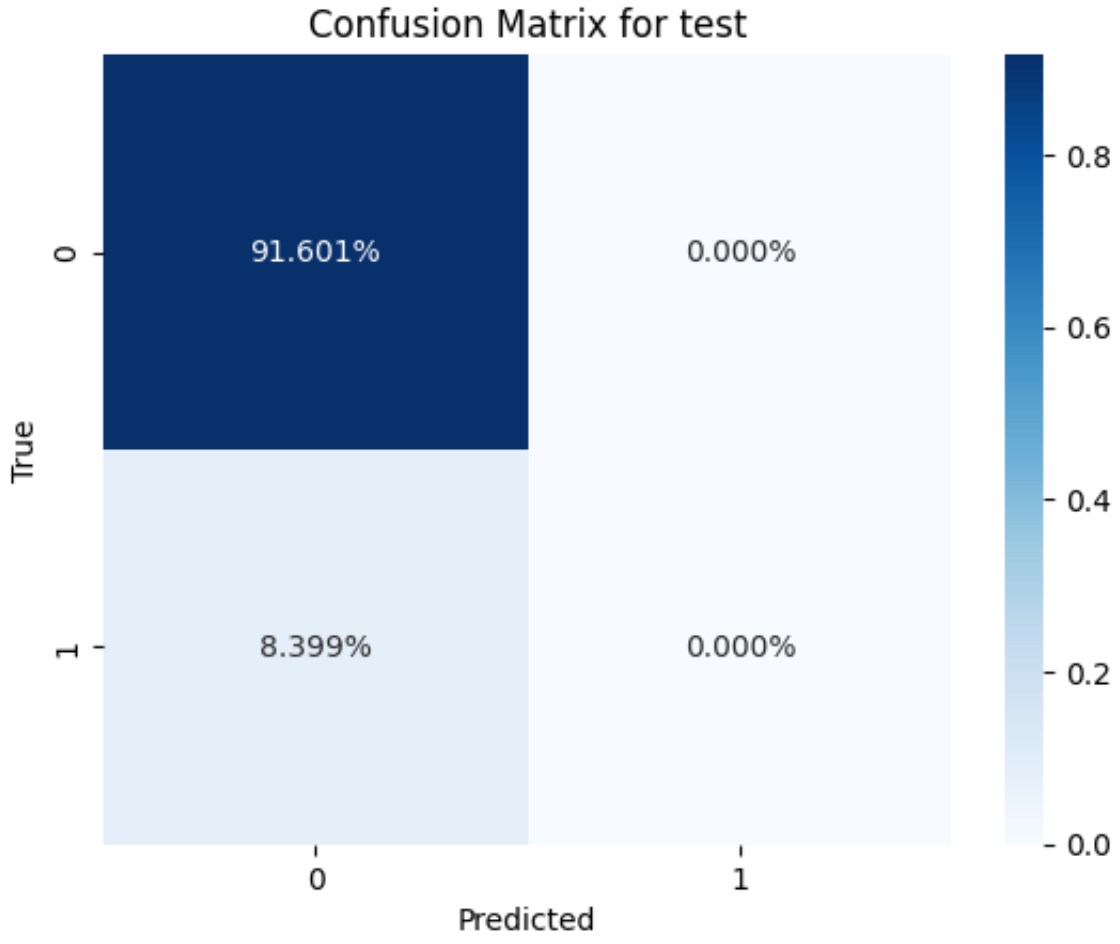
```
F1-Score for DT is:  0.0
```

```
exp_name = f"Baseline_{len(selected_features)}_features"
experimentLog.loc[len(experimentLog)] = [f"{exp_name}"],logit_score_train, logit_
#experimentLog.loc[len(experimentLog)] = ["Logistic Regression as Baseline", "HCD

experimentLog
```

| | exp_name | Train Acc | Valid Acc | Test Acc | Train AUC | Valid AUC | Test AUC | T s |
|---|---|---|---|---|---|---|---|---|
| **0** | [Baseline_200_features] | 0.913516 | 0.913043 | 0.913512 | 0.701011 | 0.697197 | 0.699813 | 0.0 |
| **1** | [Baseline_14_features] | 0.919780 | 0.919266 | 0.915859 | 0.735710 | 0.735608 | 0.735645 | 0.0 |
| **2** | [Baseline_14_features] | 0.919816 | 0.919418 | 0.916012 | 0.689067 | 0.682330 | 0.683875 | 0.0 |
| **3** | [Baseline_14_features] | 0.919816 | 0.919418 | 0.916012 | 0.713529 | 0.713784 | 0.718498 | 0.0 |
| **4** | [Baseline_14_features] | 0.919816 | 0.919418 | 0.916012 | 0.713529 | 0.713784 | 0.718498 | 0.0 |

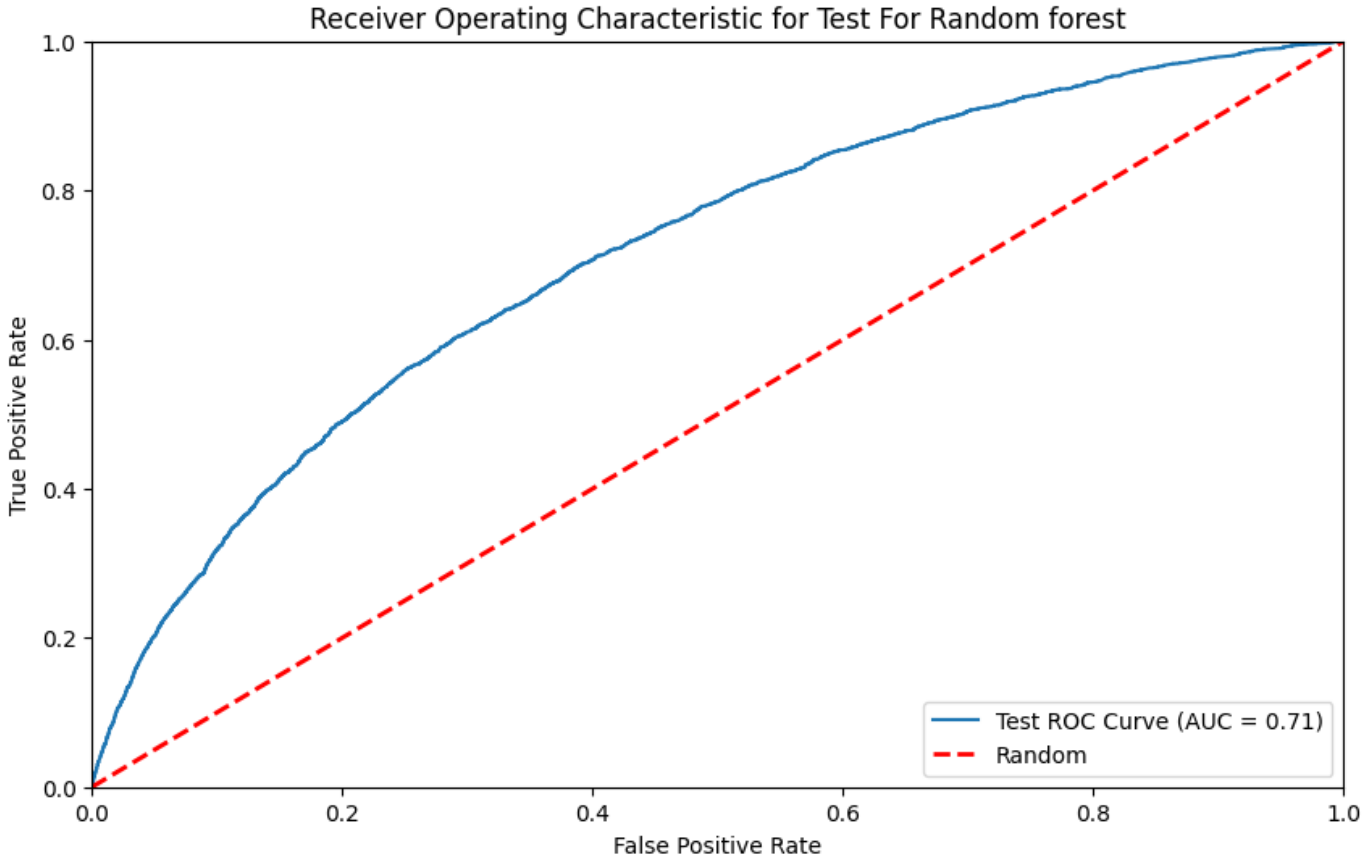
## Metrics for Random forest

```
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.metrics import confusion_matrix
from sklearn.pipeline import Pipeline


# Calculate the confusion matrix
cm = confusion_matrix(y_test, y_test_pred1)

# Create a heatmap for the confusion matrix
sns.heatmap(cm/np.sum(cm), annot=True, fmt=".3%", cmap="Blues")
plt.xlabel('Predicted')
plt.ylabel('True')
plt.title('Confusion Matrix for test')
plt.show()
```




```
import matplotlib.pyplot as plt
from sklearn.metrics import roc_curve, auc
```

```
# Obtain predicted probabilities for the positive class for each dataset
test_predicted_probs = rf_pipeline.predict_proba(X_test)[:, 1]

test_true_labels = y_test

# Calculate the false positive rate, true positive rate, and thresholds for each
test_fpr, test_tpr, _ = roc_curve(test_true_labels, test_predicted_probs)

# Calculate the area under the ROC curve for each dataset
test_roc_auc = auc(test_fpr, test_tpr)

# Plot the ROC curves for train, test, and validation datasets on a single plot
plt.figure(figsize=(10, 6))
plt.plot(test_fpr, test_tpr, label='Test ROC Curve (AUC = {:.2f})'.format(test_rc
plt.plot([0, 1], [0, 1], 'r--', label='Random', lw=2)
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.0])
plt.xlabel('False Positive Rate')
plt.ylabel('True Positive Rate')
plt.title('Receiver Operating Characteristic for Test For Random forest')
plt.legend(loc='lower right')
plt.show()
```




## HyperParameter Tuning

```
#df = datasets["application_train"]
df = X_train
#input_f = datasets["application_train"]
X = df.drop('TARGET', axis=1)  # Features
y = df['TARGET']  # Target variable
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.15, random_
X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train,
                                                      test_size=0.2, random_state

print(f"X train           shape: {X_train.shape}")
print(f"X validation      shape: {X_valid.shape}")
print(f"X test            shape: {X_test.shape}")
#X_train_resampled, y_train_resampled = resample(X_train[y_train == 0], y_train[y
#X_train = pd.concat([X_train[y_train == 1], X_train_resampled])
#y_train = pd.concat([y_train[y_train == 1], y_train_resampled])
```

```
X train           shape: (219086, 160)
X validation      shape: (54772, 160)
X test            shape: (48329, 160)
```

```
#getting the numerical features from the application train table which contains i
numerical_features = df.select_dtypes(include=['int64', 'float64']).columns.tolis
```

```
numerical_features.remove("TARGET")
numerical_features
```

```
['SK_ID_CURR',
 'CNT_CHILDREN',
 'AMT_INCOME_TOTAL',
 'AMT_CREDIT',
 'AMT_ANNUITY_x',
 'AMT_GOODS_PRICE',
 'REGION_POPULATION_RELATIVE',
 'DAYS_BIRTH',
 'DAYS_EMPLOYED',
 'DAYS_REGISTRATION',
 'DAYS_ID_PUBLISH',
 'OWN_CAR_AGE',
 'FLAG_MOBIL',
 'FLAG_WORK_PHONE',
 'FLAG_CONT_MOBILE',
 'FLAG_PHONE',
 'FLAG_EMAIL',
 'CNT_FAM_MEMBERS',
 'REGION_RATING_CLIENT',
 'REGION_RATING_CLIENT_W_CITY',
```

```
        'REGION_RATING_CLIENT_W_CITY',
        'HOUR_APPR_PROCESS_START',
        'REG_REGION_NOT_LIVE_REGION',
        'REG_REGION_NOT_WORK_REGION',
        'LIVE_REGION_NOT_WORK_REGION',
        'REG_CITY_NOT_LIVE_CITY',
        'REG_CITY_NOT_WORK_CITY',
        'LIVE_CITY_NOT_WORK_CITY',
        'EXT_SOURCE_1',
        'EXT_SOURCE_2',
        'EXT_SOURCE_3',
        'BASEMENTAREA_AVG',
        'YEARS_BEGINEXPLUATATION_AVG',
        'YEARS_BUILD_AVG',
        'COMMONAREA_AVG',
        'LANDAREA_AVG',
        'LIVINGAPARTMENTS_AVG',
        'NONLIVINGAPARTMENTS_AVG',
        'NONLIVINGAREA_AVG',
        'APARTMENTS_MODE',
        'BASEMENTAREA_MODE',
        'YEARS_BEGINEXPLUATATION_MODE',
        'YEARS_BUILD_MODE',
        'COMMONAREA_MODE',
        'ELEVATORS_MODE',
        'ENTRANCES_MODE',
        'FLOORSMAX_MODE',
        'FLOORSMIN_MODE',
        'LANDAREA_MODE',
        'LIVINGAPARTMENTS_MODE',
        'LIVINGAREA_MODE',
        'NONLIVINGAPARTMENTS_MODE',
        'NONLIVINGAREA_MODE',
        'APARTMENTS_MEDI',
        'BASEMENTAREA_MEDI',
        'YEARS_BEGINEXPLUATATION_MEDI',
        'ELEVATORS_MEDI',
        'ENTRANCES_MEDI',
        'FLOORSMAX_MEDI',
        'FLOORSMIN_MEDI',
        'LANDAREA_MEDI'
```

```
len(numerical_features)
```

```
141
```

```
#getting the categorical features from the application train table which contains
categorical_features = df.select_dtypes(include=['object']).columns.tolist()
```

```
categorical_features
```

```
['NAME_CONTRACT_TYPE',
 'CODE_GENDER',
 'FLAG_OWN_CAR',
 'FLAG_OWN_REALTY',
 'NAME_TYPE_SUITE',
 'NAME_INCOME_TYPE',
 'NAME_EDUCATION_TYPE',
 'NAME_FAMILY_STATUS',
 'NAME_HOUSING_TYPE',
 'OCCUPATION_TYPE',
 'WEEKDAY_APPR_PROCESS_START',
 'ORGANIZATION_TYPE',
 'FONDKAPREMONT_MODE',
 'HOUSETYPE_MODE',
 'WALLSMATERIAL_MODE',
 'EMERGENCYSTATE_MODE',
 'CREDIT_ACTIVE',
 'CREDIT_CURRENCY',
 'CREDIT_TYPE']
```

```
len(categorical_features)
```

```
19
```

```
from sklearn.base import BaseEstimator, TransformerMixin
# Create a class to select numerical or categorical columns
# since Scikit-Learn doesn't handle DataFrames yet
class DataFrameSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attribute_names):
        self.attribute_names = attribute_names
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.attribute_names].values
```

```
# Create numerical pipeline
numerical_pipeline = Pipeline([
    ('selector', DataFrameSelector(numerical_features)),
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])
```

```
# Create categorical pipeline
categorical_pipeline = Pipeline([
    ('selector', DataFrameSelector(categorical_features)),
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('encoder', OneHotEncoder(handle_unknown= 'ignore', sparse=False))
])
```

```
# Create column transformer
column_transformer_pipeline = ColumnTransformer(
    transformers=[
        ('num_pipeline', numerical_pipeline),
        ('cat_pipeline', categorical_pipeline)
    ], remainder="passthrough")

from sklearn.pipeline import FeatureUnion

data_prep_pipeline_FU = FeatureUnion(transformer_list=[
        ("num_pipeline", numerical_pipeline),
        ("cat_pipeline", categorical_pipeline),
    ])
```

```
selected_features = numerical_features + categorical_features
tot_features = f"{len(selected_features)}:   Num:{len(numerical_features)},    Ca
#Total Feature selected for processing
tot_features
```

```
'160:   Num:141,    Cat:19'
```

```
# Set feature selection settings
# Features removed each step
feature_selection_steps=10
# Number of features used
features_used=len(selected_features)
```

## Metrics

### ROC Curve

```
from sklearn.metrics import roc_curve, auc
import matplotlib.pyplot as plt

def plot_roc_curve(model, X_train, y_train, X_test, y_test, X_valid, y_valid, nam
    """
    Plots ROC curves for train, test, and validation sets
    """
    # Compute ROC curve and ROC area for train set
    fpr_train, tpr_train, _ = roc_curve(y_train, model.predict_proba(X_train)[:,
    roc_auc_train = auc(fpr_train, tpr_train)

    # Compute ROC curve and ROC area for test set
    fpr_test, tpr_test, _ = roc_curve(y_test, model.predict_proba(X_test)[:, 1])
    roc_auc_test = auc(fpr_test, tpr_test)

    # Compute ROC curve and ROC area for validation set
    fpr_valid, tpr_valid, _ = roc_curve(y_valid, model.predict_proba(X_valid)[:,
    roc_auc_valid = auc(fpr_valid, tpr_valid)

    # Plot ROC curves
    plt.figure(figsize=(10, 7))
    plt.plot(fpr_train, tpr_train, color='darkorange', lw=2, label='Train ROC cur
    plt.plot(fpr_test, tpr_test, color='blue', lw=2, label='Test ROC curve (area
    plt.plot(fpr_valid, tpr_valid, color='green', lw=2, label='Validation ROC cur
    plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('ROC curve - ' + name)
    plt.legend(loc="lower right")
    plt.show()
```

## Precision Recall curve

```
from sklearn.metrics import precision_recall_curve

def plot_precision_recall_all(model, X_train, y_train, X_test, y_test, X_valid, y
    # Get precision and recall values for train, test, and valid sets
    train_precision, train_recall, train_thresholds = precision_recall_curve(y_tr
    test_precision, test_recall, test_thresholds = precision_recall_curve(y_test,
    valid_precision, valid_recall, valid_thresholds = precision_recall_curve(y_va

    # Plot the precision-recall curves
    plt.plot(train_recall, train_precision, label='Train')
    plt.plot(test_recall, test_precision, label='Test')
    plt.plot(valid_recall, valid_precision, label='Valid')
    plt.xlabel('Recall')
    plt.ylabel('Precision')
    plt.title('Precision-Recall Curve - ' + name)
    plt.legend()
    plt.show()
```

## Confusion Matrix

```
import seaborn as sns
from sklearn.metrics import confusion_matrix

def plot_confusion_matrix(model, X_train, y_train, X_test, y_test, X_valid, y_val
    # Create confusion matrices for train, test and validation sets
    train_cm = confusion_matrix(y_train, model.predict(X_train))
    test_cm = confusion_matrix(y_test, model.predict(X_test))
    valid_cm = confusion_matrix(y_valid, model.predict(X_valid))

    # Plot confusion matrices
    fig, ax = plt.subplots(1, 3, figsize=(18, 5))
    sns.heatmap(train_cm/np.sum(train_cm), annot=True, fmt=".3%", cmap="Blues", a
    ax[0].set_title('Train Confusion Matrix - ' + name)
    ax[0].set_xlabel('Predicted')
    ax[0].set_ylabel('Actual')

    sns.heatmap(test_cm/np.sum(test_cm), annot=True, fmt=".3%", cmap="Blues",ax=a
    ax[1].set_title('Test Confusion Matrix - ' + name)
    ax[1].set_xlabel('Predicted')
    ax[1].set_ylabel('Actual')

    sns.heatmap(valid_cm/np.sum(valid_cm), annot=True, fmt=".3%", cmap='Blues', a
    ax[2].set_title('Validation Confusion Matrix - ' + name)
    ax[2].set_xlabel('Predicted')
    ax[2].set_ylabel('Actual')

    plt.show()
```

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

## Tuning

```
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier, GradientBoostingClassifier
from sklearn.feature_selection import RFE



classifiers = [    [('Logistic Regression', LogisticRegression(solver='saga', ran
    [('Support Vector Machine', SVC(random_state=42, probability=True),"SVM")],
    [('Decision Tree', DecisionTreeClassifier(random_state=42), "RFE")],
    [('Random Forest', RandomForestClassifier(random_state=42), "RFE")],
    [('Gradient Boosting', GradientBoostingClassifier(warm_start=True, random_sta
]
```

```
params_grid = {
    'Logistic Regression': {
        'penalty': ['l1', 'l2'],
        'tol': [0.0001, 0.00001],
        'C': [10, 1, 0.1, 0.01],
    },
    'Support Vector Machine': {
        'kernel': ['linear'],
        'degree': [4, 5],
        'C': [0.001, 0.01],   #Low C - allow for misclassification
        'gamma': [0.01, 0.1, 1]  #Low gamma - high variance and low bias
    },
    'Decision Tree': {
        'criterion': ['gini', 'entropy'],
        'max_depth': [5, 10, 15],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 3, 5]
    },
    'Random Forest': {
        'n_estimators': [1000],
        'max_features': ['auto', 'sqrt', 'log2'],
        'max_depth': [5, 10, 15],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 3, 5],
        'bootstrap': [True, False]
    },
    'Gradient Boosting': {
        'n_estimators': [1000],
        'max_depth': [5, 10, 15],
        'max_features': ['auto', 'sqrt', 'log2'],
        'min_samples_split': [2, 5, 10],
        'min_samples_leaf': [1, 3, 5],
        'learning_rate': [0.01, 0.1, 1]
    }
}
```

## Logistic Regression

```
params_grid_lr = {
        'lr__penalty': ['l1', 'l2'],
        'lr__tol': [0.0001, 0.00001],
        'lr__C': [10, 1, 0.1, 0.01],
    }


full_pipeline_with_predictor = Pipeline([
                ("preparation", data_prep_pipeline_FU),
                #('RFE', RFE(estimator=LogisticRegression(solver='saga', random_s
                ("lr", LogisticRegression(solver='saga', random_state=42))
                ])

grid_search = GridSearchCV(full_pipeline_with_predictor, params_grid_lr, cv=2, sc
                                       n_jobs=-1,verbose=1)
grid_search.fit(X_train, y_train)

# Best estimator score
best_train = pct(grid_search.best_score_)
print(grid_search.best_params_)
```

```
Fitting 2 folds for each of 16 candidates, totalling 32 fits
/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_encoders.py:868
  warnings.warn(
{'lr__C': 10, 'lr__penalty': 'l2', 'lr__tol': 0.0001}
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_sag.py:350: Conv
  warnings.warn(
```

```
from sklearn.metrics import log_loss

# get the best model from GridSearchCV
best_model = grid_search.best_estimator_

# calculate scores on training set
best_train_accuracy_lr = np.round(best_model.score(X_train, y_train), 4)
best_train_f1_lr = np.round(f1_score(y_train, best_model.predict(X_train)), 4)
best_train_logloss_lr = np.round(log_loss(y_train, best_model.predict_proba(X_tra
best_train_roc_auc_lr = np.round(roc_auc_score(y_train, best_model.predict_proba(

# calculate scores on validation set
best_valid_accuracy_lr = np.round(best_model.score(X_valid, y_valid), 4)
best_valid_f1_lr = np.round(f1_score(y_valid, best_model.predict(X_valid)), 4)
best_valid_logloss_lr = np.round(log_loss(y_valid, best_model.predict_proba(X_val
best_valid_roc_auc_lr = np.round(roc_auc_score(y_valid, best_model.predict_proba(

# calculate scores on test set
best_test_accuracy_lr = np.round(best_model.score(X_test, y_test), 4)
best_test_f1_lr = np.round(f1_score(y_test, best_model.predict(X_test)), 4)
best_test_logloss_lr = np.round(log_loss(y_test, best_model.predict_proba(X_test)
best_test_roc_auc_lr = np.round(roc_auc_score(y_test, best_model.predict_proba(X_
```

```
print(best_train_accuracy_lr,best_valid_accuracy_lr,best_test_accuracy_lr)
```

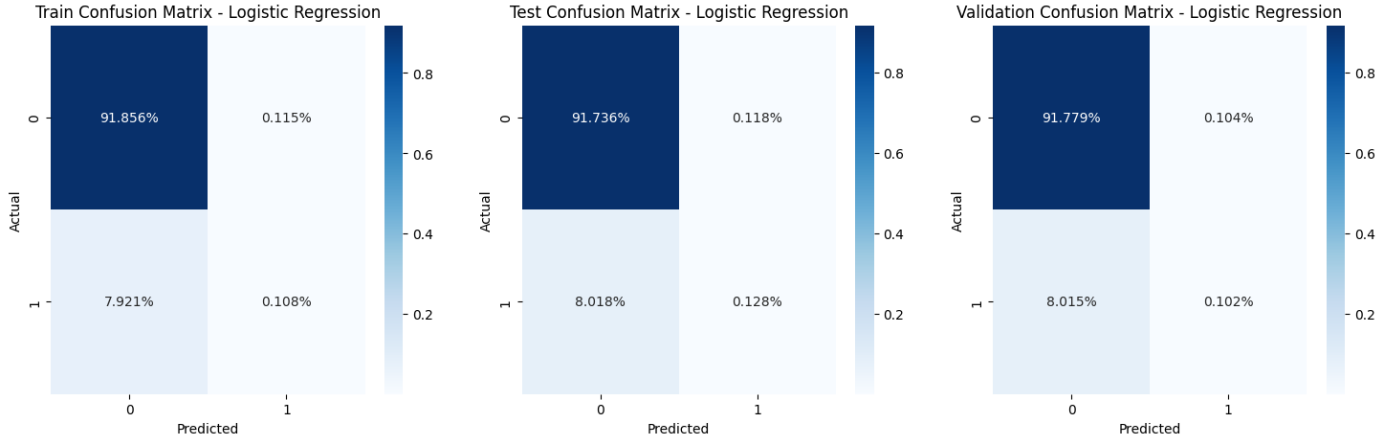
```
0.9196 0.9188 0.9186
```

```
print("Fit and Prediction with best estimator")
start = time()
model = grid_search.best_estimator_.fit(X_train, y_train)
train_time_lr = round(time() - start, 4)
print(train_time_lr)
```

```
Fit and Prediction with best estimator
/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_encoders.py:868
  warnings.warn(
84.9184
/usr/local/lib/python3.9/dist-packages/sklearn/linear_model/_sag.py:350: Conv
  warnings.warn(
```

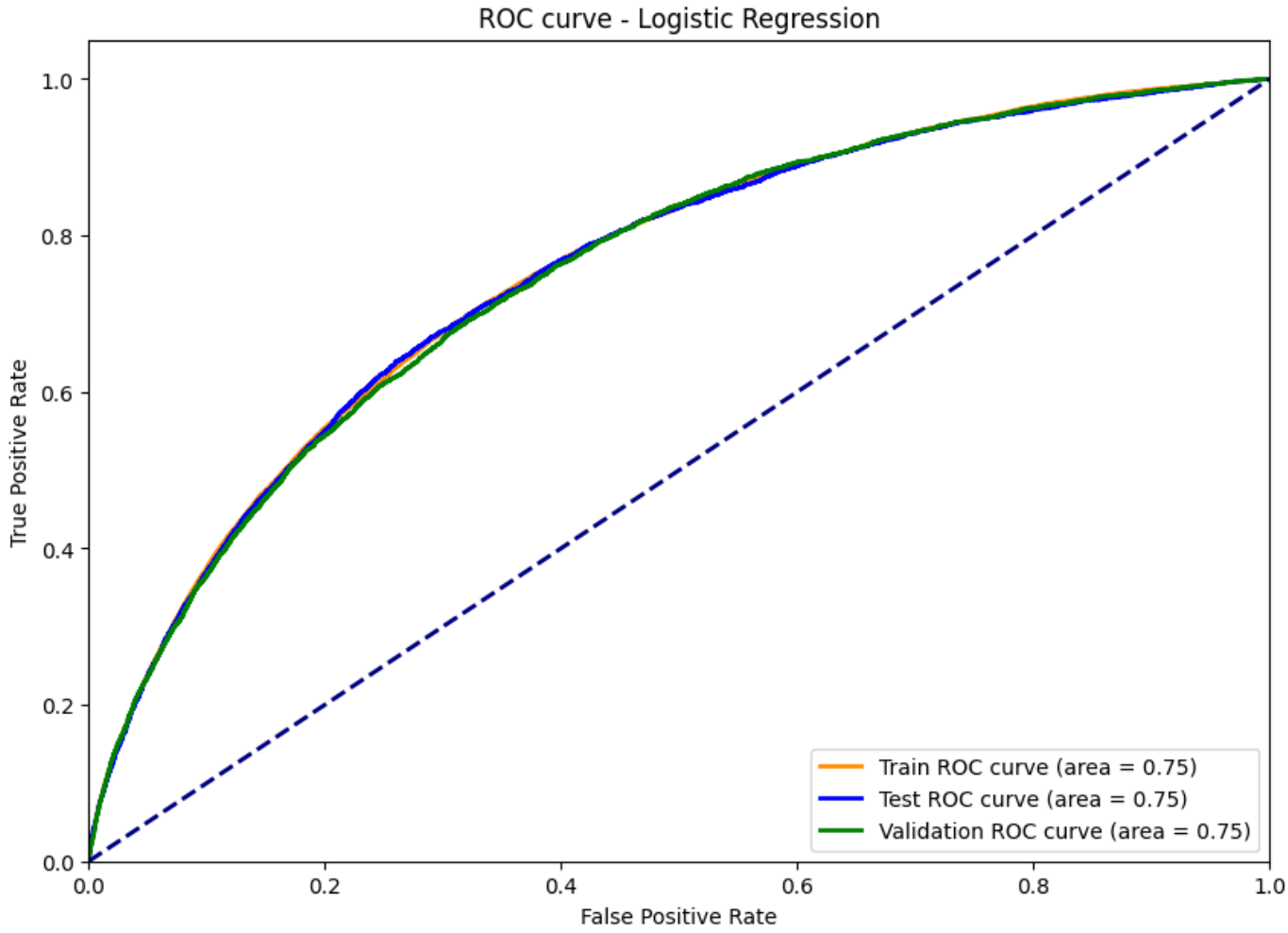
## Confusion Matrix

```
plot_confusion_matrix(best_model,X_train,y_train,X_test,y_test,X_valid, y_valid,"
```



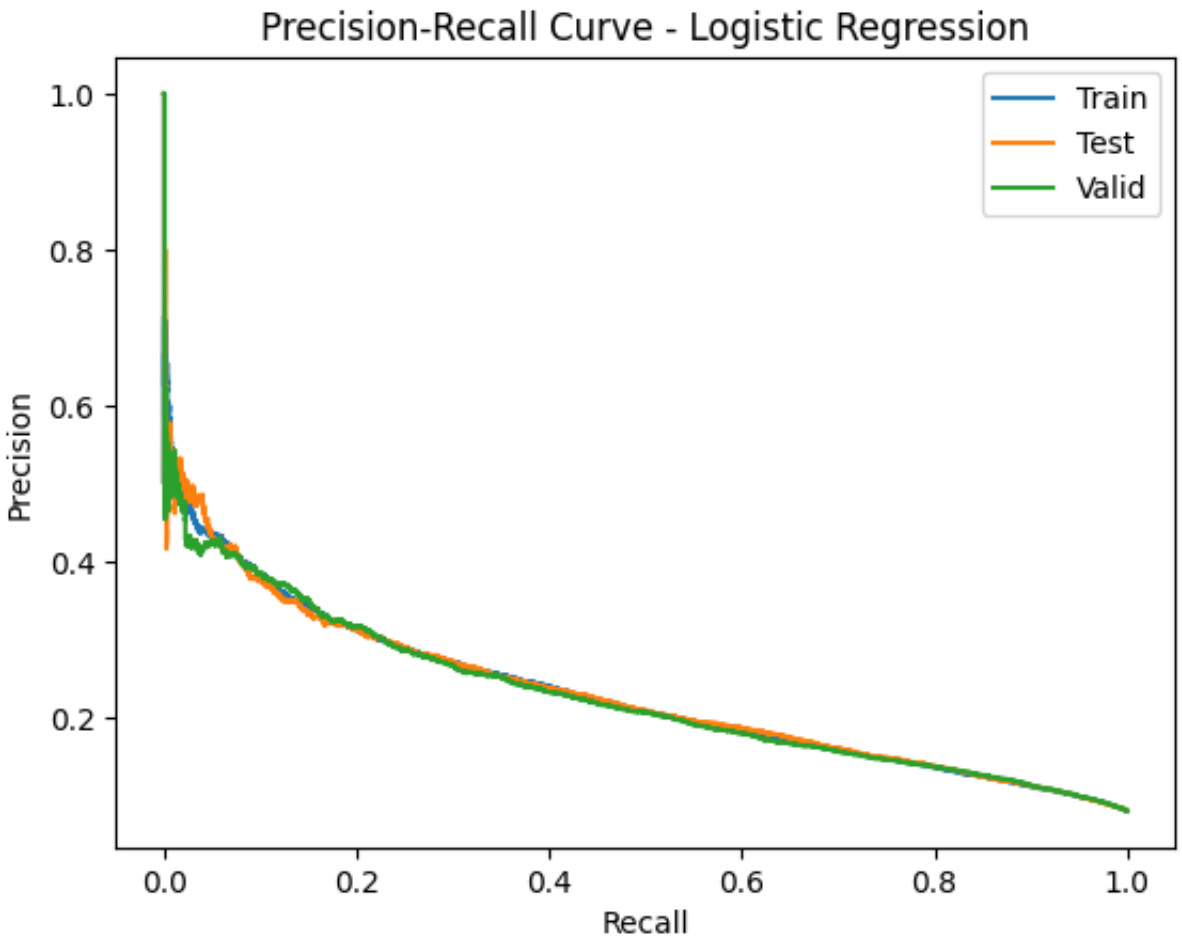

## ROC Curve

```
plot_roc_curve(best_model,X_train,y_train,X_test, y_test,X_valid, y_valid,"Logist
```




## Precision Curve

```
plot_precision_recall_all(best_model,X_train,y_train,X_test, y_test,X_valid, y_va
```




## Decision Tree

```
params_grid_dt = {
        'dt__criterion': ['gini', 'entropy'],
        'dt__max_depth': [5,10],
        'dt__min_samples_split': [2,5, 10],
        'dt__max_features': ['auto','sqrt','log2'],
        'dt__splitter':['best','random']
    }


full_pipeline_with_predictor_dt = Pipeline([
                ("preparation", data_prep_pipeline_FU),
                #('RFE', RFE(estimator=LogisticRegression(solver='saga', random_s
                ("dt", DecisionTreeClassifier(random_state=42))
                ])

grid_search_dt = GridSearchCV(full_pipeline_with_predictor_dt, params_grid_dt, cv
                                      n_jobs=-1,verbose=1)
grid_search_dt.fit(X_train, y_train)

# Best estimator score
best_train_dt = pct(grid_search_dt.best_score_)
print(grid_search_dt.best_params_)
```

```
Fitting 2 folds for each of 72 candidates, totalling 144 fits
/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_encoders.py:868
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/tree/_classes.py:269: FutureWa
  warnings.warn(
{'dt__criterion': 'entropy', 'dt__max_depth': 10, 'dt__max_features': 'auto',
```

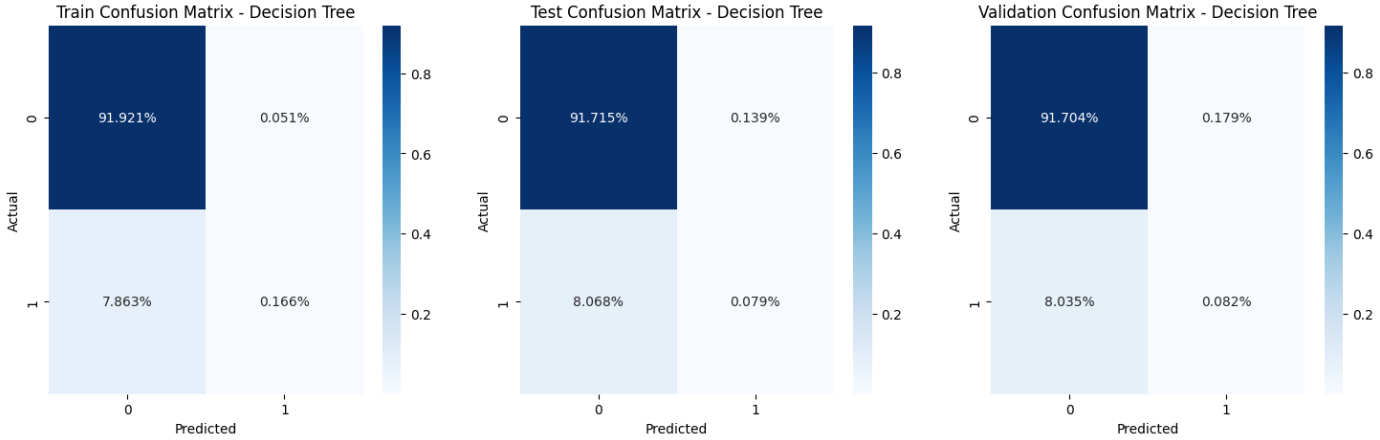
```
from sklearn.metrics import log_loss

# get the best model from GridSearchCV
best_model_dt = grid_search_dt.best_estimator_

# calculate scores on training set
best_train_accuracy_dt = np.round(best_model_dt.score(X_train, y_train), 4)
best_train_f1_dt = np.round(f1_score(y_train, best_model_dt.predict(X_train)), 4)
best_train_logloss_dt = np.round(log_loss(y_train, best_model_dt.predict_proba(X_
best_train_roc_auc_dt = np.round(roc_auc_score(y_train, best_model_dt.predict_pro

# calculate scores on validation set
best_valid_accuracy_dt = np.round(best_model_dt.score(X_valid, y_valid), 4)
best_valid_f1_dt = np.round(f1_score(y_valid, best_model_dt.predict(X_valid)), 4)
best_valid_logloss_dt = np.round(log_loss(y_valid, best_model_dt.predict_proba(X_
best_valid_roc_auc_dt = np.round(roc_auc_score(y_valid, best_model_dt.predict_pro

# calculate scores on test set
best_test_accuracy_dt = np.round(best_model_dt.score(X_test, y_test), 4)
best_test_f1_dt = np.round(f1_score(y_test, best_model_dt.predict(X_test)), 4)
best_test_logloss_dt = np.round(log_loss(y_test, best_model_dt.predict_proba(X_te
best_test_roc_auc_dt = np.round(roc_auc_score(y_test, best_model_dt.predict_proba
```

```
print("Fit and Prediction with best estimator")
start = time()
model = grid_search_dt.best_estimator_.fit(X_train, y_train)
train_time_dt = round(time() - start, 4)
print(train_time_dt)
```

```
Fit and Prediction with best estimator
/usr/local/lib/python3.9/dist-packages/sklearn/preprocessing/_encoders.py:868
  warnings.warn(
/usr/local/lib/python3.9/dist-packages/sklearn/tree/_classes.py:269: FutureWa
  warnings.warn(
6.9671
```

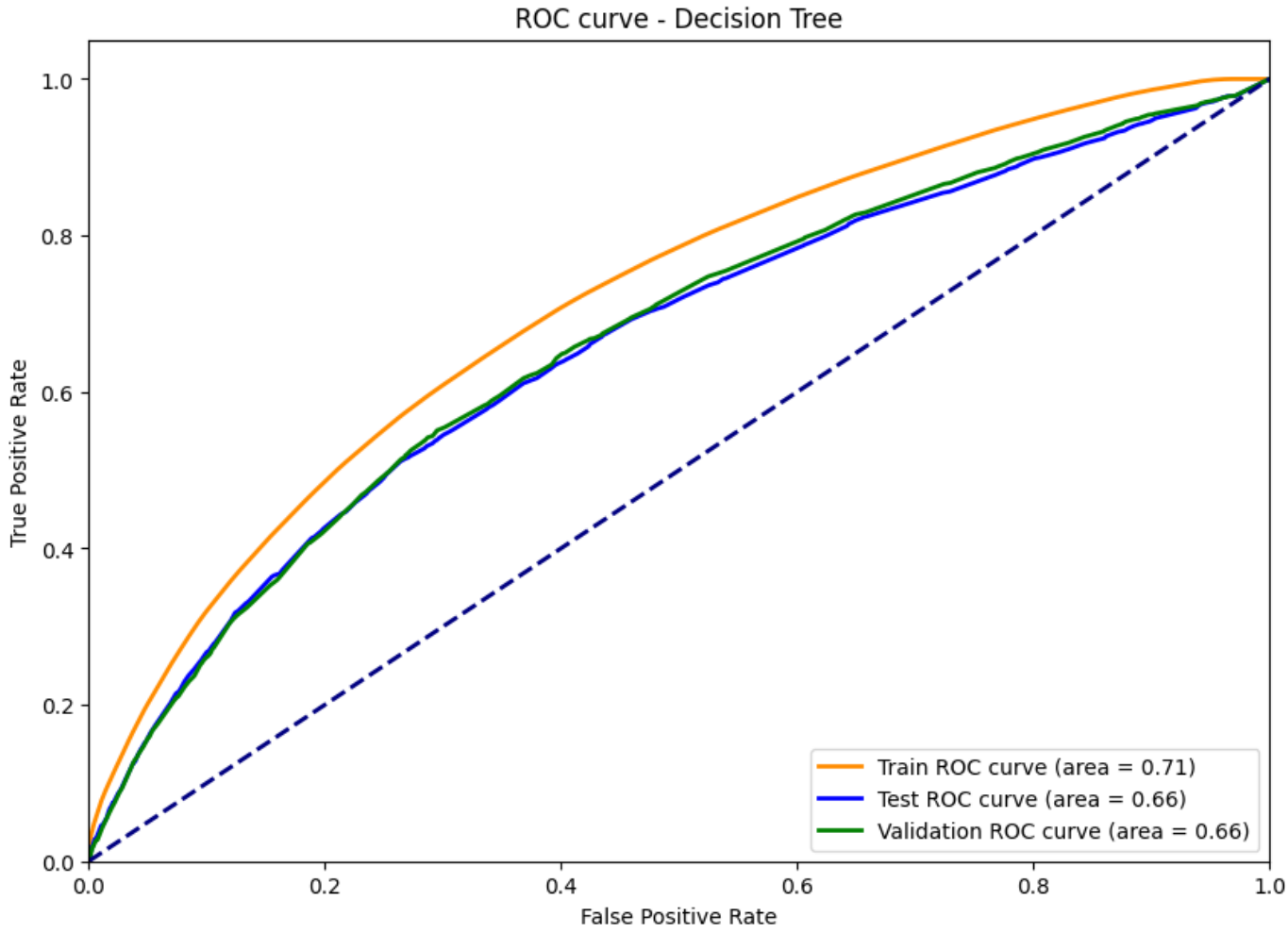
## Confusion Matrix

```
plot_confusion_matrix(best_model_dt,X_train,y_train,X_test,y_test,X_valid, y_vali
```



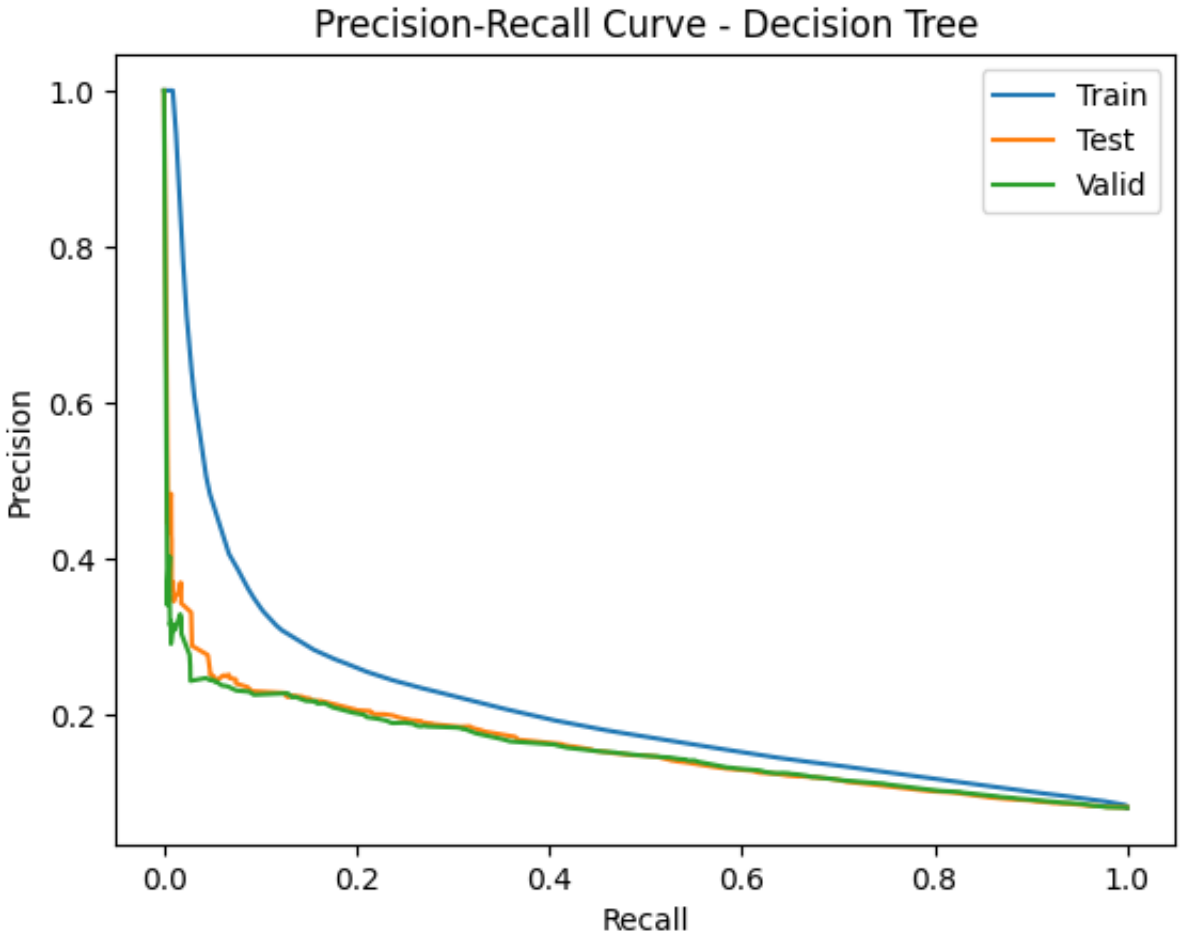

## ROC curve

```
plot_roc_curve(best_model_dt,X_train,y_train,X_test, y_test,X_valid, y_valid,"Dec
```




## Precision Curve

```
plot_precision_recall_all(best_model_dt,X_train,y_train,X_test, y_test,X_valid, y
```




```
print(best_train_accuracy_dt,best_valid_accuracy_dt,best_test_accuracy_dt)
```

```
0.9209 0.9179 0.9179
```

```
X_train.head()
```

| | SK_ID_CURR | NAME_CONTRACT_TYPE | CODE_GENDER | FLAG_OWN_CAR | FLAG_OWN_REA |
|---|---|---|---|---|---|
| **7100** | 107877 | Cash loans | F | Y | |
| **93236** | 203374 | Cash loans | F | N | |
| **269497** | 397459 | Cash loans | F | Y | |
| **153917** | 270361 | Cash loans | F | N | |
| **173591** | 292017 | Cash loans | M | Y | |

5 rows × 160 columns

## Random Forest

```
params_grid_rf = {
        'rf__n_estimators': [50,200],
        'rf__max_depth': [2,5],
        'rf__min_samples_split': [2, 5]
    }


full_pipeline_with_predictor_rf = Pipeline([
                ("preparation", data_prep_pipeline_FU),
                #('RFE', RFE(estimator=LogisticRegression(solver='saga', random_s
                ("rf", RandomForestClassifier(random_state=42))
                ])

grid_search_rf = GridSearchCV(full_pipeline_with_predictor_rf, params_grid_rf, cv
                                      n_jobs=-1,verbose=1)
grid_search_rf.fit(X_train, y_train)

# Best estimator score
best_train_rf = pct(grid_search_rf.best_score_)
print(grid_search_rf.best_params_)
```

```
Fitting 2 folds for each of 8 candidates, totalling 16 fits
{'rf__max_depth': 5, 'rf__min_samples_split': 2, 'rf__n_estimators': 200}
```

```
from sklearn.metrics import log_loss

# get the best model from GridSearchCV
best_model_rf = grid_search_rf.best_estimator_

# calculate scores on training set
best_train_accuracy_rf = np.round(grid_search_rf.score(X_train, y_train), 4)
best_train_f1_rf = np.round(f1_score(y_train, grid_search_rf.predict(X_train)), 4
best_train_logloss_rf = np.round(log_loss(y_train, grid_search_rf.predict_proba(X
best_train_roc_auc_rf = np.round(roc_auc_score(y_train, grid_search_rf.predict_pr

# calculate scores on validation set
best_valid_accuracy_rf = np.round(grid_search_rf.score(X_valid, y_valid), 4)
best_valid_f1_rf = np.round(f1_score(y_valid, grid_search_rf.predict(X_valid)), 4
best_valid_logloss_rf = np.round(log_loss(y_valid, grid_search_rf.predict_proba(X
best_valid_roc_auc_rf = np.round(roc_auc_score(y_valid, grid_search_rf.predict_pr

# calculate scores on test set
best_test_accuracy_rf = np.round(grid_search_rf.score(X_test, y_test), 4)
best_test_f1_rf = np.round(f1_score(y_test, grid_search_rf.predict(X_test)), 4)
best_test_logloss_rf = np.round(log_loss(y_test, grid_search_rf.predict_proba(X_t
best_test_roc_auc_rf = np.round(roc_auc_score(y_test, grid_search_rf.predict_prob
```

```
print("Fit and Prediction with best estimator")
start = time()
model = grid_search_rf.best_estimator_.fit(X_train, y_train)
train_time_rf = round(time() - start, 4)
print(train_time_rf)
```

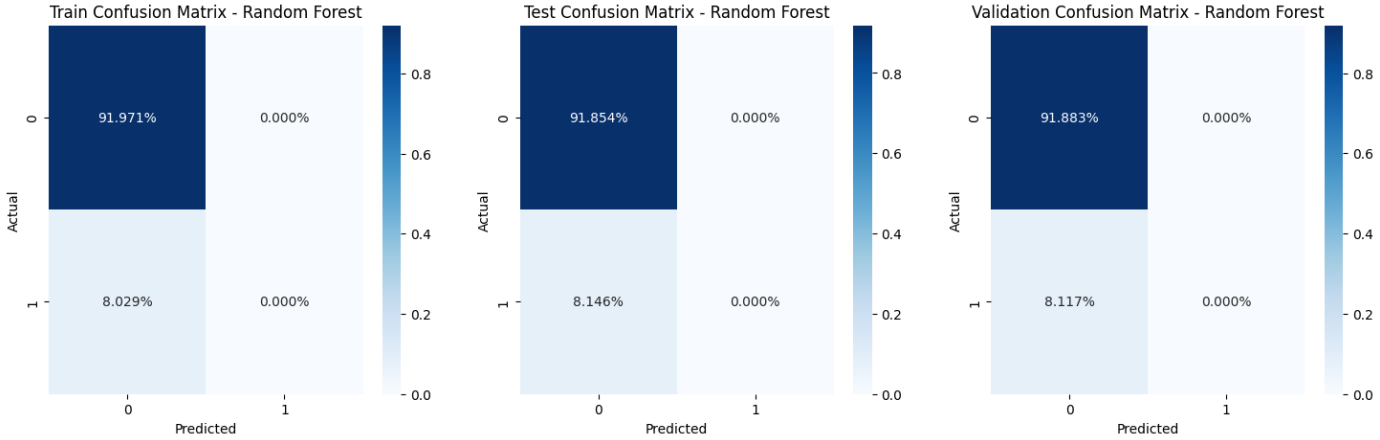
```
Fit and Prediction with best estimator
69.9169
```

```
print(best_train_accuracy_rf,best_valid_accuracy_rf,best_test_accuracy_rf)
print(best_valid_f1_rf)
```

```
0.7446 0.731 0.7349
0.0
```

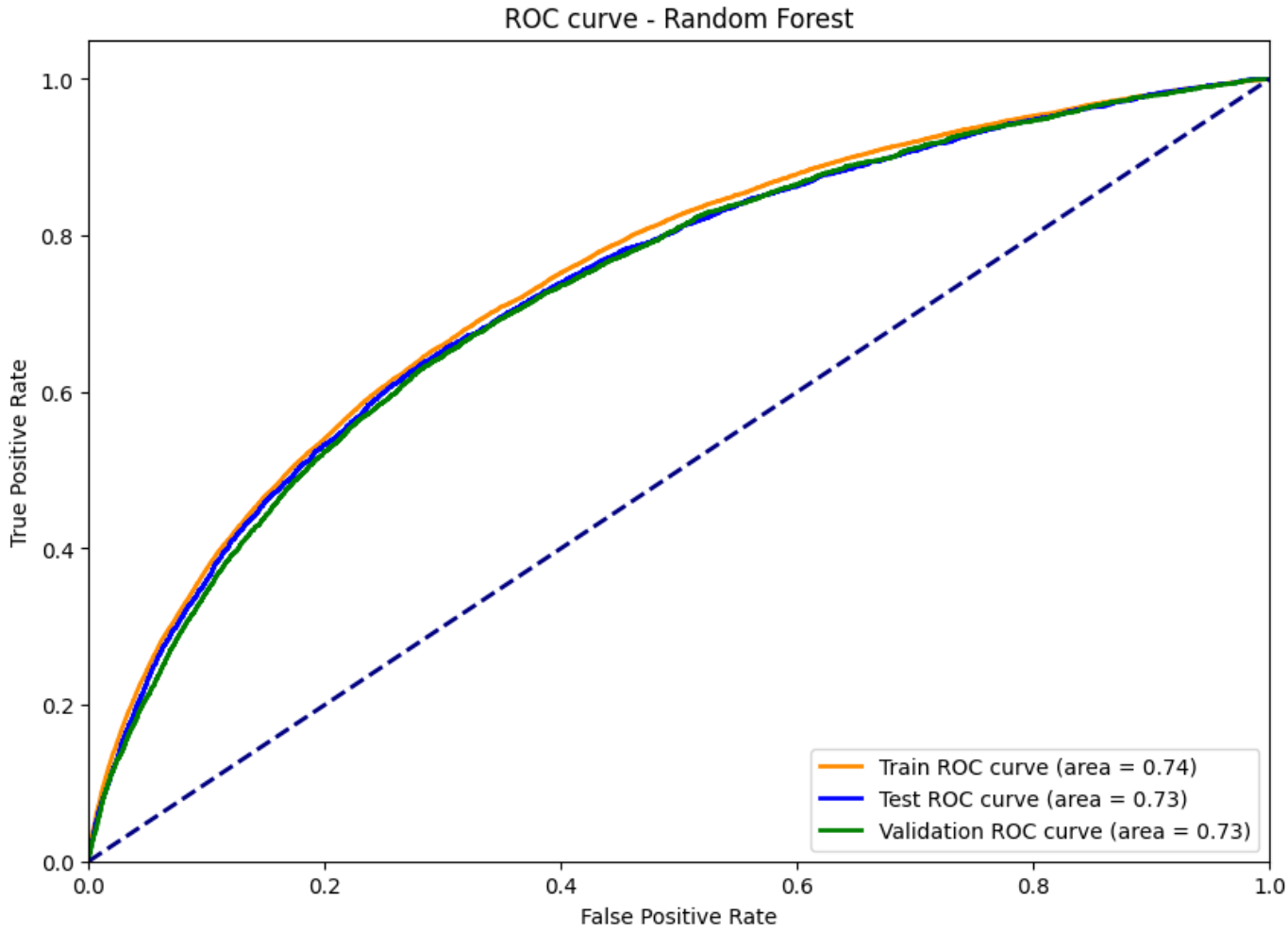
## Confusion Matrix

```
plot_confusion_matrix(best_model_rf,X_train,y_train,X_test,y_test,X_valid, y_vali
```



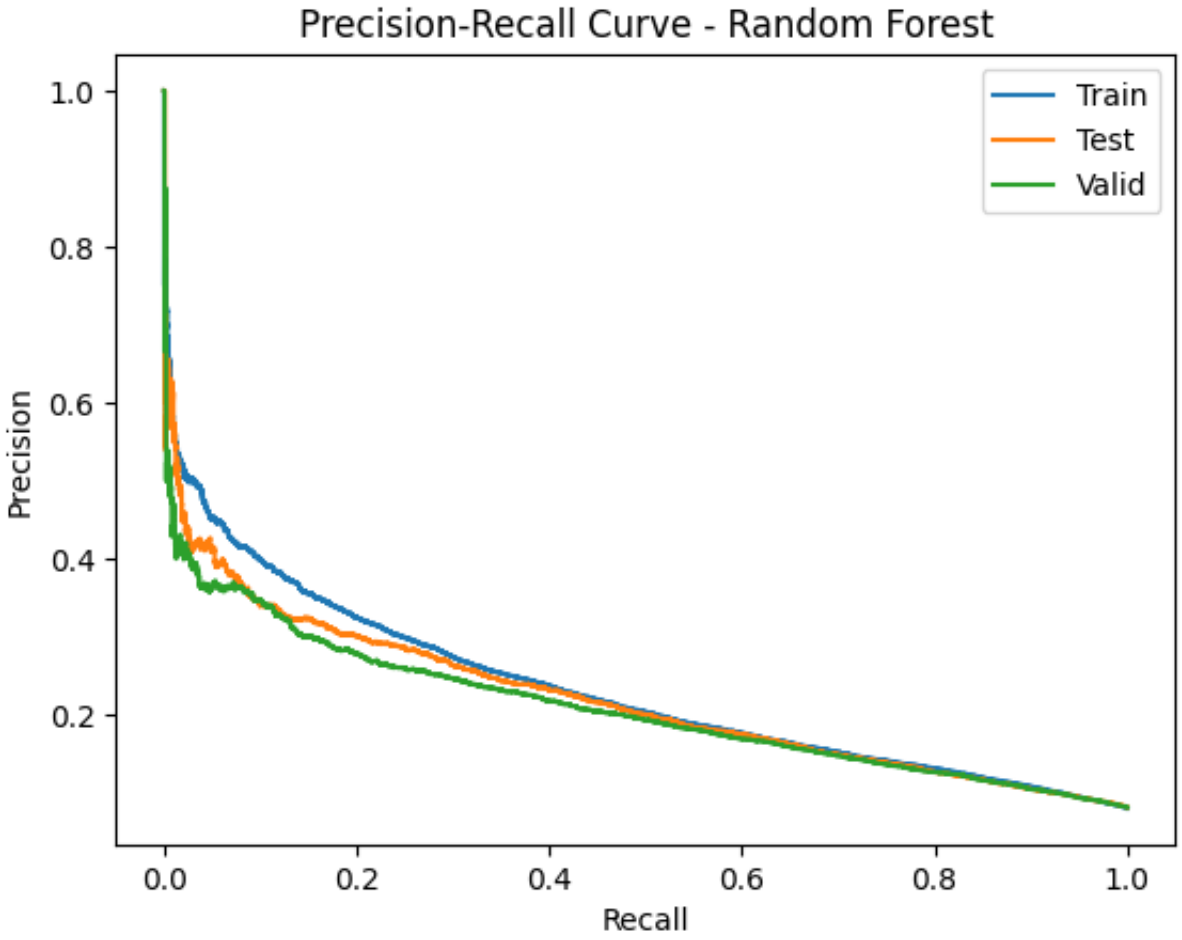

## ROC curve

```
plot_roc_curve(best_model_rf,X_train,y_train,X_test, y_test,X_valid, y_valid,"Ran
```




## Presicion Curve

```
plot_precision_recall_all(best_model_rf,X_train,y_train,X_test, y_test,X_valid, y
```




## Experiment Log

```
exp_name = f"Baseline_{len(selected_features)}_features"
experimentLog = pd.DataFrame(columns=["exp_name",
                                      "Train Acc",
                                      "Valid Acc",
                                      "Test  Acc",
                                      "Train AUC",
                                      "Valid AUC",
                                      "Test  AUC",
                                      "Train F1 Score",
                                      "Valid F1 Score",
                                      "Test F1 Score",
                                      "Train Log Loss",
                                      "Valid Log Loss",
                                      "Test Log Loss",
                                      "Train Time",
                                      "Description"
                                     ])
#Logistic Regression
exp_name = "Logistic Regression"
experimentLog.loc[len(experimentLog)] = [f"{exp_name}"],best_train_accuracy_lr, b



#Decision Tree
exp_name = "Decision Tree"
experimentLog.loc[len(experimentLog)] = [f"{exp_name}"],best_train_accuracy_dt, b


#Random Forest
exp_name = "Random Forest"
experimentLog.loc[len(experimentLog)] = [f"{exp_name}"],best_train_accuracy_rf, b
```

```
experimentLog
```

| | exp_name | Train Acc | Valid Acc | Test Acc | Train AUC | Valid AUC | Test AUC | Train F1 Score | Valid F1 Score | Test F1 Score | Tr L |
|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | [Logistic Regression] | 0.9196 | 0.9188 | 0.9186 | 0.7539 | 0.7508 | 0.7521 | 0.0262 | 0.0246 | 0.0306 | 0.2 |
| **1** | [Decision Tree] | 0.9209 | 0.9179 | 0.9179 | 0.7141 | 0.6638 | 0.6592 | 0.0402 | 0.0196 | 0.0188 | 0.2 |
| **2** | [Random Forest] | 0.7446 | 0.7310 | 0.7349 | 0.7446 | 0.7310 | 0.7349 | 0.0000 | 0.0000 | 0.0000 | 0.2 |

## Gradient Boosting

```
params_grid_gb = {
        'gb__n_estimators': [1000],
        'gb__max_depth': [5, 10],
        'gb__max_features': ['auto', 'sqrt', 'log2'],
        'gb__min_samples_split': [5, 10],
        'gb__learning_rate': [0.01, 0.1, 1]
    }


full_pipeline_with_predictor_gb = Pipeline([
                ("preparation", data_prep_pipeline_FU),
                #('RFE', RFE(estimator=LogisticRegression(solver='saga', random_s
                ("gb", GradientBoostingClassifier(random_state=42))
                ])

grid_search_gb = GridSearchCV(full_pipeline_with_predictor_gb, params_grid_gb, cv
                                      n_jobs=-1,verbose=1)
grid_search_gb.fit(X_train, y_train)

# Best estimator score
best_train_gb = pct(grid_search_gb.best_score_)
print(grid_search_gb.best_params_)
```

```
from sklearn.metrics import log_loss

# get the best model from GridSearchCV
best_model_gb = grid_search_gb.best_estimator_

# calculate scores on training set
best_train_accuracy_gb = np.round(grid_search_gb.score(X_train, y_train), 4)
best_train_f1_gb = np.round(f1_score(y_train, grid_search_gb.predict(X_train)), 4
best_train_logloss_gb = np.round(log_loss(y_train, grid_search_gb.predict_proba(X
best_train_roc_auc_gb = np.round(roc_auc_score(y_train, grid_search_gb.predict_pr

# calculate scores on validation set
best_valid_accuracy_gb = np.round(grid_search_gb.score(X_valid, y_valid), 4)
best_valid_f1_gb = np.round(f1_score(y_valid, grid_search_gb.predict(X_valid)), 4
best_valid_logloss_gb = np.round(log_loss(y_valid, grid_search_gb.predict_proba(X
best_valid_roc_auc_gb = np.round(roc_auc_score(y_valid, grid_search_gb.predict_pr

# calculate scores on test set
best_test_accuracy_gb = np.round(grid_search_gb.score(X_test, y_test), 4)
best_test_f1_gb = np.round(f1_score(y_test, grid_search_gb.predict(X_test)), 4)
best_test_logloss_gb = np.round(log_loss(y_test, grid_search_gb.predict_proba(X_t
best_test_roc_auc_gb = np.round(roc_auc_score(y_test, grid_search_gb.predict_prob
```

```
print("Fit and Prediction with best estimator")
start = time()
model = grid_search_gb.best_estimator_.fit(X_train, y_train)
train_time_gb = round(time() - start, 4)
print(train_time_gb)
```

```
print(best_train_accuracy_rf,best_valid_accuracy_rf,best_test_accuracy_rf)
print(best_valid_f1_rf)
```

## Confusion Matrix

```
plot_confusion_matrix(best_model_gb,X_train,y_train,X_test,y_test,X_valid, y_vali
```

## ROC Curve

```
plot_roc_curve(best_model_gb,X_train,y_train,X_test, y_test,X_valid, y_valid,"Gra
```

### Precision Curve

```
plot_precision_recall_all(best_model_gb,X_train,y_train,X_test, y_test,X_valid, y
```

Double-click (or enter) to edit

## Neural Networks

```
!pip install -q pytorch-lightning
```

```
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 719.0/719.0 kB 48.9 MB/s eta 0:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 519.2/519.2 kB 47.7 MB/s eta 0:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.0/1.0 MB 69.8 MB/s eta 0:00:0
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 114.2/114.2 kB 15.3 MB/s eta 0:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 269.3/269.3 kB 28.0 MB/s eta 0:00
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 158.8/158.8 kB 18.8 MB/s eta 0:00
```

```
import pandas as pd
import torch
import pytorch_lightning as pl
from torch import nn

from torchmetrics import Accuracy
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.impute import SimpleImputer
from sklearn.compose import ColumnTransformer
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
from numpy import vstack
from sklearn.metrics import accuracy_score
from sklearn.metrics import make_scorer, roc_auc_score
```

```
# Load the data
df = pd.read_csv('X_train_final')

# Split the dataset into features and target
X = df.drop('TARGET', axis=1)
y = df['TARGET']

# Split the dataset into training and testing sets
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_s
```

## Numerical features and categorical features

```
# Define the numerical features and categorical features
numerical_features = df.select_dtypes(include=['int64', 'float64']).columns.tolis
numerical_features.remove("TARGET")
categorical_features = df.select_dtypes(include=['object']).columns.tolist()
```

## Numerical and Categorical Pipeline

```
# Define the numerical pipeline
num_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='median')),
    ('scaler', StandardScaler())
])

# Define the categorical pipeline
cat_pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('onehot', OneHotEncoder(handle_unknown='ignore'))
])
```

## Combine numerical and categorical features

```
# Combine the numerical and categorical pipelines using ColumnTransformer
preprocessor = ColumnTransformer([
    ('num', num_pipeline, numerical_features),
    ('cat', cat_pipeline, categorical_features)
])
```

## Preprocess the data

```
# Preprocess the training data
X_train = preprocessor.fit_transform(X_train)

# Preprocess the testing data
X_test = preprocessor.transform(X_test)

# Convert the data to PyTorch tensors
X_train = torch.Tensor(X_train)
X_test = torch.Tensor(X_test)
y_train = torch.Tensor(y_train.values).unsqueeze(1)
y_test = torch.Tensor(y_test.values).unsqueeze(1)
```

## MLP class

```
class MLP(pl.LightningModule):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super().__init__()
        self.train_acc = Accuracy("binary", num_classes=2)
        self.valid_acc = Accuracy("binary", num_classes=2)
        self.test_acc = Accuracy("binary", num_classes=2)
        self.input_layer = nn.Linear(input_dim, hidden_dim[0])
        self.hidden_layers = nn.ModuleList()
        for i in range(len(hidden_dim)-1):
            self.hidden_layers.append(nn.Linear(hidden_dim[i], hidden_dim[i+1]))
        self.output_layer = nn.Linear(hidden_dim[-1], output_dim)
        nn.init.kaiming_uniform_(self.input_layer.weight)
        for hidden_layer in self.hidden_layers:
            nn.init.kaiming_uniform_(hidden_layer.weight)
        nn.init.xavier_uniform_(self.output_layer.weight)
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()
```

```
        self.loss = nn.BCELoss()
    # def __init__(self, input_dim, hidden_dim, output_dim):
    #     super().__init__()
    #     self.train_acc = Accuracy("binary", num_classes=2)
    #     self.valid_acc = Accuracy("binary", num_classes=2)
    #     self.test_acc = Accuracy("binary", num_classes=2)
    #     self.input_layer = nn.Linear(input_dim, hidden_dim[0])
    #     self.hidden_layers = nn.ModuleList()
    #     for i in range(len(hidden_dim)-1):
    #         self.hidden_layers.append(nn.Linear(hidden_dim[i], hidden_dim[i+1]))
    #     self.output_layer = nn.Linear(hidden_dim[-1], output_dim)
    #     self.relu = nn.ReLU()
    #     self.sigmoid = nn.Sigmoid()
    #     self.loss = nn.BCELoss()

    def forward(self, x):
        x = self.input_layer(x)
        x = self.relu(x)
        for hidden_layer in self.hidden_layers:
            x = hidden_layer(x)
            x = self.relu(x)
        x = self.output_layer(x)
        x = self.sigmoid(x)
        return x
    # def forward(self, x):
    #     x = self.relu(self.input_layer(x))
    #     for layer in self.hidden_layers:
    #         x = self.relu(layer(x))
    #     x = self.sigmoid(self.output_layer(x))
    #     return x

    def configure_optimizers(self):
        optimizer = torch.optim.Adam(self.parameters(), lr=0.01)
        return optimizer

    def training_step(self, batch, batch_idx):
        x, y = batch
        y_pred = self(x)
        loss = self.loss(y_pred, y)
        self.log('train_loss', loss)
        y_pred = torch.round(y_pred)
        correct = (y_pred == y).sum().item()
        total = y.shape[0]
        self.log('train_acc', correct / total, prog_bar=True)
        return loss
```

```
    def validation_step(self, batch, batch_idx):
        x, y = batch
        y_pred = self(x)
        loss = self.loss(y_pred, y)
        self.log('val_loss', loss)
        return loss
    def test_step(self, batch, batch_idx):
        x, y = batch
        logits = self(x)
        loss = nn.functional.cross_entropy(logits, y)
        preds = torch.argmax(logits, dim=1)
        #self.test_acc.update(preds, y)
        self.log("test_loss", loss, prog_bar=True)
        #self.log("test_acc", self.test_acc.compute(), prog_bar=True)
        acc = (preds == y).float().mean()
        self.log('test_acc', acc, on_step=True, on_epoch=True, prog_bar=True)
        return loss
```

Double-click (or enter) to edit

```
class MyDataModule(pl.LightningDataModule):
    def __init__(self, dataloader):
        super().__init__()
        self.dataloader = dataloader

    def train_dataloader(self):
        return self.dataloader

    def val_dataloader(self):
        return self.dataloader

    def test_dataloader(self):
        return self.dataloader
```

```
from pytorch_lightning.callbacks import ModelCheckpoint


callbacks = [ModelCheckpoint(save_top_k=1, mode='max', monitor="valid_acc")] # sa
```

# Experiment 1

```
input_dim = X_train.shape[1]
hidden_dim = [128, 64, 32, 16]
output_dim = 1
num_epochs = 400
batch_size = 100

# Create the MLP model
mlp = MLP(input_dim=input_dim, hidden_dim=hidden_dim, output_dim=1)

# Create the PyTorch Lightning trainer
if torch.cuda.is_available(): # if you have GPUs
    print("GPU is available")
    trainer = pl.Trainer(max_epochs=num_epochs, callbacks=callbacks, accelerator=
else:
    trainer = pl.Trainer(max_epochs=10, callbacks=callbacks)
#trainer = pl.Trainer(max_epochs=num_epochs, accelerator='auto')
val_dataset = torch.utils.data.TensorDataset(X_test, y_test)

# Define the validation data loader
val_loader = torch.utils.data.DataLoader(val_dataset, batch_size=batch_size, shuf

# Define the training dataset
train_dataset = torch.utils.data.TensorDataset(X_train, y_train)

# Define the training data loader
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size,

# Train the model
trainer.fit(mlp, train_loader)


# Get the final training accuracy
train_acc = trainer.callback_metrics['train_acc']
print(f'Train Accuracy: {train_acc:.4f}')
```

```
GPU is available
INFO:pytorch_lightning.utilities.rank_zero:GPU available: True (cuda), used: '
INFO:pytorch_lightning.utilities.rank_zero:TPU available: False, using: 0 TPU
INFO:pytorch_lightning.utilities.rank_zero:IPU available: False, using: 0 IPU
INFO:pytorch_lightning.utilities.rank_zero:HPU available: False, using: 0 HPU
/usr/local/lib/python3.9/dist-packages/pytorch_lightning/trainer/configuratio
  rank_zero_warn(
WARNING:pytorch_lightning.loggers.tensorboard:Missing logger folder: /content
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES
INFO:pytorch_lightning.callbacks.model_summary:
  | Name          | Type           | Params
-------------------------------------------------
0 | train_acc     | BinaryAccuracy | 0
1 | valid_acc     | BinaryAccuracy | 0
2 | test_acc      | BinaryAccuracy | 0
3 | input_layer   | Linear         | 38.0 K
4 | hidden_layers | ModuleList     | 10.9 K
5 | output_layer  | Linear         | 17
6 | relu          | ReLU           | 0
7 | sigmoid       | Sigmoid        | 0
8 | loss          | BCELoss        | 0
-------------------------------------------------
48.9 K    Trainable params
0         Non-trainable params
48.9 K    Total params
0.196     Total estimated model params size (MB)
/usr/local/lib/python3.9/dist-packages/pytorch_lightning/trainer/connectors/d
  rank_zero_warn(
```

Epoch 399: 100%  2578/2578 [00:10<00:00, 248.36it/s, v_num=0, train_acc=0.939]

```
/usr/local/lib/python3.9/dist-packages/pytorch_lightning/callbacks/model_chec
  warning_cache.warn(m)
INFO:pytorch_lightning.utilities.rank_zero:`Trainer.fit` stopped: `max_epochs
Train Accuracy: 0.9388
```

## Results

```
#test_results = trainer.test(mlp, datamodule=val_loader)

datamodule = MyDataModule(val_loader)
# Test the model
test_results = trainer.test(model=mlp, datamodule=datamodule)
print(test_results)
test_acc = test_results[0]['test_acc_epoch']
print(f'Test Accuracy: {test_acc:.4f}')



predictions, actuals = list(), list()
for inputs, targets in val_loader:
    yhat = mlp(inputs)
    yhat = yhat.detach().numpy()
    actual = targets.numpy()
    actual = actual.reshape((len(actual), 1))
    yhat = yhat.round()
    predictions.append(yhat)
    actuals.append(actual)
predictions, actuals = vstack(predictions), vstack(actuals)
acc = accuracy_score(actuals, predictions)
auc = roc_auc_score(actuals, predictions)
#print("Accuracy: ", acc)
print("AUC: ", auc)
```

```
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES
/usr/local/lib/python3.9/dist-packages/pytorch_lightning/trainer/connectors/d
  rank_zero_warn(
```

Testing DataLoader 0: 100% 645/645 [00:02<00:00, 277.93it/s]

| Test metric | DataLoader 0 |
|---|---|
| test_acc_epoch | 0.918588399887085 |
| test_loss | 0.0 |

```
[{'test_loss': 0.0, 'test_acc_epoch': 0.918588399887085}]
Test Accuracy: 0.9186
AUC:  0.5801253551641196
```

## Tensorboard for experiment 1

```
# Start tensorboard
%load_ext tensorboard
%tensorboard --logdir lightning_logs/
```

TensorBoard TIME SERIES SCALAR INACTIVE

Filter runs (reg

Filter tags (regex)

All Scalars Image Histogram

Run

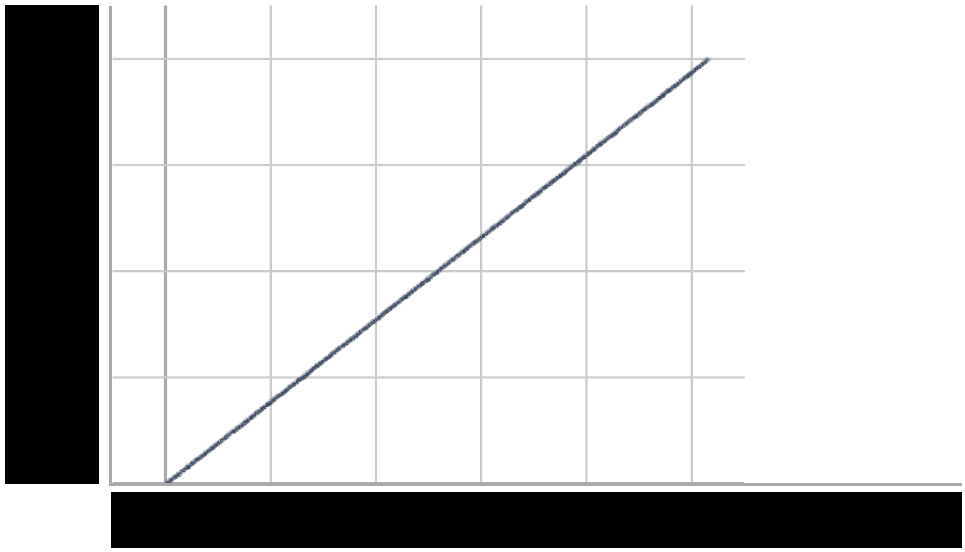
versio n_0

Pinned

*Pin cards for a quick view and comparison*

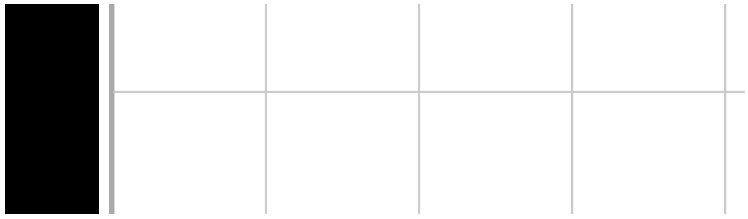
epoch

epoch



test_acc_epoch

test_acc_epoch



## Experiment 2

Using 4 hidden layers with BCEWithLogitsLoss loss function and Adam optimizer with 100 epoches and 128 batch size

```
class MLP1(pl.LightningModule):
    def __init__(self, input_dim, hidden_dim, output_dim):
        super().__init__()
        self.train_acc = Accuracy("binary", num_classes=2)
        self.valid_acc = Accuracy("binary", num_classes=2)
        self.test_acc = Accuracy("binary", num_classes=2)
        self.input_layer = nn.Linear(input_dim, hidden_dim[0])
        self.hidden_layers = nn.ModuleList()
        for i in range(len(hidden_dim)-1):
            self.hidden_layers.append(nn.Linear(hidden_dim[i], hidden_dim[i+1]))
        self.output_layer = nn.Linear(hidden_dim[-1], output_dim)
        nn.init.kaiming_uniform_(self.input_layer.weight)
        for hidden_layer in self.hidden_layers:
            nn.init.kaiming_uniform_(hidden_layer.weight)
        nn.init.xavier_uniform_(self.output_layer.weight)
        self.relu = nn.ReLU()
        self.sigmoid = nn.Sigmoid()
        # instantiate the loss function
        self.loss = nn.BCEWithLogitsLoss()
    # def __init__(self, input_dim, hidden_dim, output_dim):
    #     super().__init__()
    #     self.train_acc = Accuracy("binary", num_classes=2)
    #     self.valid_acc = Accuracy("binary", num_classes=2)
    #     self.test_acc = Accuracy("binary", num_classes=2)
    #     self.input_layer = nn.Linear(input_dim, hidden_dim[0])
    #     self.hidden_layers = nn.ModuleList()
    #     for i in range(len(hidden_dim)-1):
    #         self.hidden_layers.append(nn.Linear(hidden_dim[i], hidden_dim[i+1])
    #     self.output_layer = nn.Linear(hidden_dim[-1], output_dim)
    #     self.relu = nn.ReLU()
    #     self.sigmoid = nn.Sigmoid()
    #     self.loss = nn.BCELoss()
```

```
    def forward(self, x):
        x = self.input_layer(x)
        x = self.relu(x)
        for hidden_layer in self.hidden_layers:
            x = hidden_layer(x)
            x = self.relu(x)
        x = self.output_layer(x)
        x = self.sigmoid(x)
        return x
    # def forward(self, x):
    #     x = self.relu(self.input_layer(x))
    #     for layer in self.hidden_layers:
    #         x = self.relu(layer(x))
    #     x = self.sigmoid(self.output_layer(x))
    #     return x

    def configure_optimizers(self):
        optimizer = torch.optim.Adam(self.parameters(), lr=0.0001)
        return optimizer

    def training_step(self, batch, batch_idx):
        x, y = batch
        y_pred = self(x)
        loss = self.loss(y_pred, y)
        self.log('train_loss', loss)
        y_pred = torch.round(y_pred)
        correct = (y_pred == y).sum().item()
        total = y.shape[0]
        self.log('train_acc', correct / total, prog_bar=True)
        return loss

    def validation_step(self, batch, batch_idx):
        x, y = batch
        y_pred = self(x)
        loss = self.loss(y_pred, y)
        self.log('val_loss', loss)
        return loss
    def test_step(self, batch, batch_idx):
        x, y = batch
        logits = self(x)
        loss = nn.functional.cross_entropy(logits, y)
        preds = torch.argmax(logits, dim=1)
        #self.test_acc.update(preds, y)
        self.log("test_loss", loss, prog_bar=True)
        #self.log("test_acc", self.test_acc.compute(), prog_bar=True)
```

```
        acc = (preds == y).float().mean()
        self.log('test_acc', acc, on_step=True, on_epoch=True, prog_bar=True)
        return loss
```

Double-click (or enter) to edit

```
input_dim = X_train.shape[1]
hidden_dim = [128, 64, 32, 16]
output_dim = 1
num_epochs = 100
batch_size = 128

# Create the MLP1 model
mlp1 = MLP1(input_dim=input_dim, hidden_dim=hidden_dim, output_dim=1)

# Create the PyTorch Lightning trainer
if torch.cuda.is_available(): # if you have GPUs
    print("GPU is available")
    trainer = pl.Trainer(max_epochs=num_epochs, callbacks=callbacks, accelerator=
else:
    trainer = pl.Trainer(max_epochs=10, callbacks=callbacks)
#trainer = pl.Trainer(max_epochs=num_epochs, accelerator='auto')
val_dataset1 = torch.utils.data.TensorDataset(X_test, y_test)

# Define the validation data loader
val_loader1 = torch.utils.data.DataLoader(val_dataset1, batch_size=batch_size, sh

# Define the training dataset
train_dataset1 = torch.utils.data.TensorDataset(X_train, y_train)

# Define the training data loader
train_loader1 = torch.utils.data.DataLoader(train_dataset1, batch_size=batch_size

# Train the model
trainer.fit(mlp1, train_loader1)


# Get the final training accuracy
train_acc1 = trainer.callback_metrics['train_acc']
print(f'Train Accuracy: {train_acc1:.4f}')
```

```
INFO:pytorch_lightning.utilities.rank_zero:Trainer already configured with mo
INFO:pytorch_lightning.utilities.rank_zero:GPU available: True (cuda), used: 
INFO:pytorch_lightning.utilities.rank_zero:TPU available: False, using: 0 TPU
INFO:pytorch_lightning.utilities.rank_zero:IPU available: False, using: 0 IPU
INFO:pytorch_lightning.utilities.rank_zero:HPU available: False, using: 0 HPU
/usr/local/lib/python3.9/dist-packages/pytorch_lightning/trainer/configuratio
  rank_zero_warn(
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES
INFO:pytorch_lightning.callbacks.model_summary:
  | Name          | Type               | Params
-------------------------------------------------
0 | train_acc     | BinaryAccuracy     | 0
1 | valid_acc     | BinaryAccuracy     | 0
2 | test_acc      | BinaryAccuracy     | 0
3 | input_layer   | Linear             | 38.0 K
4 | hidden_layers | ModuleList         | 10.9 K
5 | output_layer  | Linear             | 17
6 | relu          | ReLU               | 0
7 | sigmoid       | Sigmoid            | 0
8 | loss          | BCEWithLogitsLoss  | 0
-------------------------------------------------
48.9 K    Trainable params
0         Non-trainable params
48.9 K    Total params
0.196     Total estimated model params size (MB)
GPU is available
/usr/local/lib/python3.9/dist-packages/pytorch_lightning/trainer/connectors/d
  rank_zero_warn(
```

Epoch 99: 100%  2014/2014 [00:08<00:00, 235.52it/s, v_num=6, train_acc=0.906]

```
INFO:pytorch_lightning.utilities.rank_zero:`Trainer.fit` stopped: `max_epochs
Train Accuracy: 0.9059
```

```
from sklearn.metrics import roc_auc_score

# Set the model to evaluation mode
mlp1.eval()

# Create lists to store the predicted and true labels
y_pred = []
y_true = []

# Iterate over the validation data loader and make predictions
for x_batch, y_batch in val_loader1:
    y_pred_batch = mlp1(x_batch)
    y_pred_batch = torch.sigmoid(y_pred_batch).detach().cpu().numpy()
    y_pred.extend(y_pred_batch)
    y_true.extend(y_batch.detach().cpu().numpy())

# Calculate the ROC-AUC score
roc_auc = roc_auc_score(y_true, y_pred)
print(f'ROC-AUC Score: {roc_auc:.4f}')
```

```
ROC-AUC Score: 0.5000
```

## Results

```
#test_results = trainer.test(mlp, datamodule=val_loader)

datamodule = MyDataModule(val_loader)
# Test the model
test_results1 = trainer.test(model=mlp1, datamodule=datamodule)
print(test_results1)
test_acc1 = test_results1[0]['test_acc_epoch']
print(f'Test Accuracy: {test_acc1:.4f}')



predictions, actuals = list(), list()
for inputs, targets in val_loader:
    yhat = mlp1(inputs)
    yhat = yhat.detach().numpy()
    actual = targets.numpy()
    actual = actual.reshape((len(actual), 1))
    yhat = yhat.round()
    predictions.append(yhat)
    actuals.append(actual)
predictions, actuals = vstack(predictions), vstack(actuals)
acc1 = accuracy_score(actuals, predictions)
auc1 = roc_auc_score(actuals, predictions)
#print("Accuracy: ", acc)
print("AUC: ", auc1)
```

```
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES
/usr/local/lib/python3.9/dist-packages/pytorch_lightning/trainer/connectors/d
  rank_zero_warn(
```

Testing DataLoader 0: 100% 504/504 [00:01<00:00, 360.58it/s]

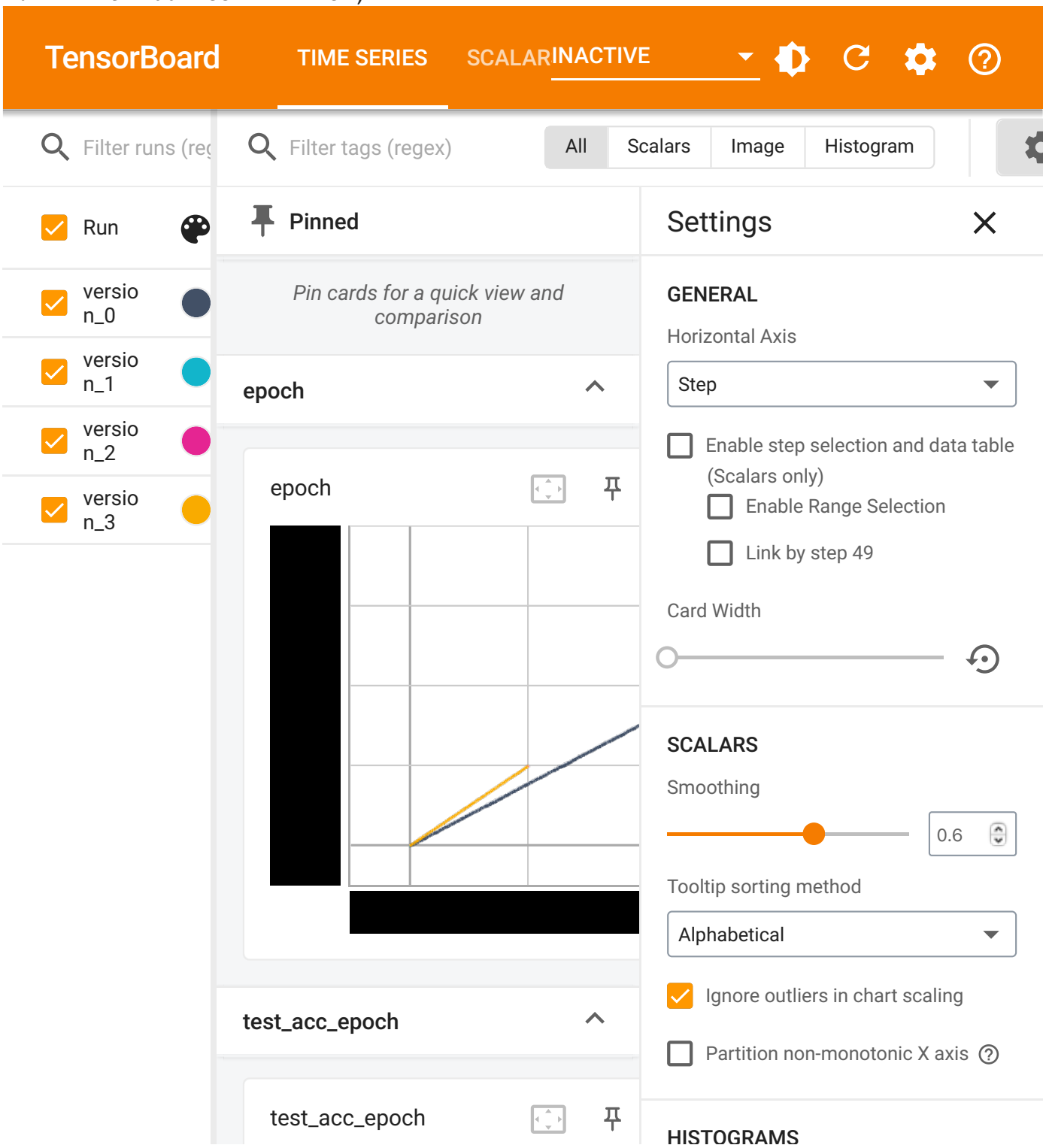
| Test metric | DataLoader 0 |
|---|---|
| test_acc_epoch | 0.918588399887085 |
| test_loss | 0.0 |

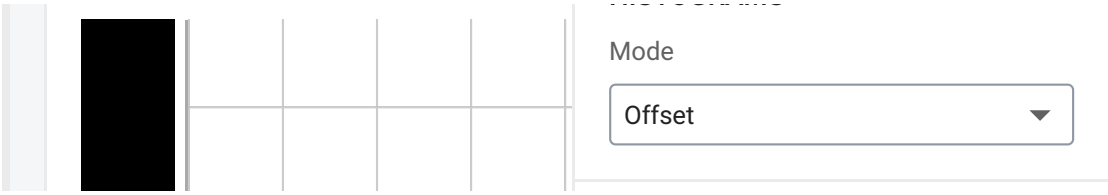
```
[{'test_loss': 0.0, 'test_acc_epoch': 0.918588399887085}]
Test Accuracy: 0.9186
AUC:  0.5
```

## Tensorboard for experiment 2

```
# Start tensorboard
%load_ext tensorboard
%tensorboard --logdir lightning_logs/
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
Reusing TensorBoard on port 6006 (pid 191200), started 1:15:27 ago. (Use
'!kill 191200' to kill it.)
```






## Experiment 3

```
input_dim = X_train.shape[1]
hidden_dim = [128, 64, 32]
output_dim = 2
num_epochs = 100
batch_size = 128

# Create the MLP1 model
mlp2 = MLP(input_dim=input_dim, hidden_dim=hidden_dim, output_dim=1)

# Create the PyTorch Lightning trainer
if torch.cuda.is_available(): # if you have GPUs
    print("GPU is available")
    trainer = pl.Trainer(max_epochs=num_epochs, callbacks=callbacks, accelerator=
else:
    trainer = pl.Trainer(max_epochs=10, callbacks=callbacks)
#trainer = pl.Trainer(max_epochs=num_epochs, accelerator='auto')
val_dataset = torch.utils.data.TensorDataset(X_test, y_test)

# Define the validation data loader
val_loader = torch.utils.data.DataLoader(val_dataset, batch_size=batch_size, shuf

# Define the training dataset
train_dataset = torch.utils.data.TensorDataset(X_train, y_train)

# Define the training data loader
train_loader = torch.utils.data.DataLoader(train_dataset, batch_size=batch_size,

# Train the model
trainer.fit(mlp2, train_loader1)


# Get the final training accuracy
train_acc2 = trainer.callback_metrics['train_acc']
print(f'Train Accuracy: {train_acc2:.4f}')
```

```
INFO:pytorch_lightning.utilities.rank_zero:Trainer already configured with mo
INFO:pytorch_lightning.utilities.rank_zero:GPU available: True (cuda), used: 
INFO:pytorch_lightning.utilities.rank_zero:TPU available: False, using: 0 TPU
INFO:pytorch_lightning.utilities.rank_zero:IPU available: False, using: 0 IPU
INFO:pytorch_lightning.utilities.rank_zero:HPU available: False, using: 0 HPU
/usr/local/lib/python3.9/dist-packages/pytorch_lightning/trainer/configuratio
  rank_zero_warn(
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES
INFO:pytorch_lightning.callbacks.model_summary:
  | Name          | Type           | Params
-------------------------------------------------
0 | train_acc     | BinaryAccuracy | 0
1 | valid_acc     | BinaryAccuracy | 0
2 | test_acc      | BinaryAccuracy | 0
3 | input_layer   | Linear         | 38.0 K
4 | hidden_layers | ModuleList     | 10.3 K
5 | output_layer  | Linear         | 33
6 | relu          | ReLU           | 0
7 | sigmoid       | Sigmoid        | 0
8 | loss          | BCELoss        | 0
-------------------------------------------------
48.4 K    Trainable params
0         Non-trainable params
48.4 K    Total params
0.194     Total estimated model params size (MB)
GPU is available
/usr/local/lib/python3.9/dist-packages/pytorch_lightning/trainer/connectors/d
  rank_zero_warn(
```

Epoch 99: 100%  2014/2014 [00:08<00:00, 245.16it/s, v_num=5, train_acc=0.941]

```
INFO:pytorch_lightning.utilities.rank_zero:`Trainer.fit` stopped: `max_epochs
Train Accuracy: 0.9412
```

## Results

```
#test_results = trainer.test(mlp, datamodule=val_loader)

datamodule = MyDataModule(val_loader)
# Test the model
test_results2 = trainer.test(model=mlp2, datamodule=datamodule)
print(test_results1)
test_acc2 = test_results2[0]['test_acc_epoch']
print(f'Test Accuracy: {test_acc2:.4f}')



predictions, actuals = list(), list()
for inputs, targets in val_loader:
    yhat = mlp2(inputs)
    yhat = yhat.detach().numpy()
    actual = targets.numpy()
    actual = actual.reshape((len(actual), 1))
    yhat = yhat.round()
    predictions.append(yhat)
    actuals.append(actual)
predictions, actuals = vstack(predictions), vstack(actuals)
#acc1 = accuracy_score(actuals, predictions)
auc2 = roc_auc_score(actuals, predictions)
#print("Accuracy: ", acc)
print("AUC: ", auc2)
```

```
INFO:pytorch_lightning.accelerators.cuda:LOCAL_RANK: 0 - CUDA_VISIBLE_DEVICES
/usr/local/lib/python3.9/dist-packages/pytorch_lightning/trainer/connectors/d
  rank_zero_warn(
```

Testing DataLoader 0: 100% 504/504 [00:01<00:00, 386.25it/s]

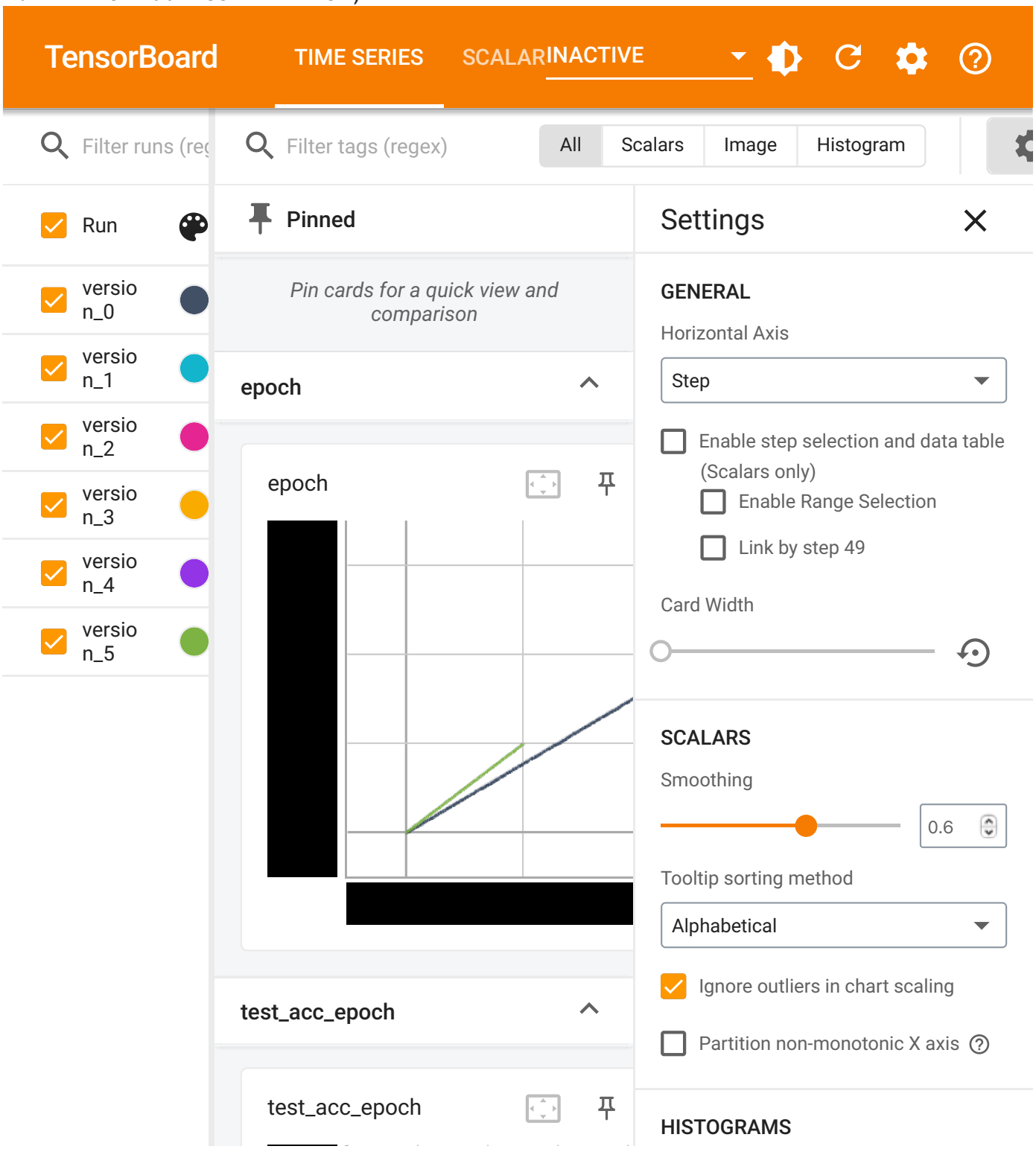
| Test metric | DataLoader 0 |
|---|---|
| test_acc_epoch | 0.918588399887085 |
| test_loss | 0.0 |

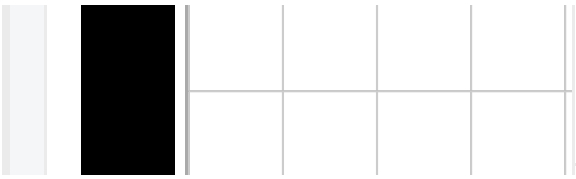
```
[{'test_loss': 0.0, 'test_acc_epoch': 0.918588399887085}]
Test Accuracy: 0.9186
AUC:  0.5306657967916345
```

## Tensorboard for experiment 3

```
# Start tensorboard
%load_ext tensorboard
%tensorboard --logdir lightning_logs/
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
Reusing TensorBoard on port 6006 (pid 191200), started 2:04:55 ago. (Use
'!kill 191200' to kill it.)
```






## Experiment log for neural network

```
exp_name = f"Baseline_{len(selected_features)}_features"
experimentLog1 = pd.DataFrame(columns=["DataSet",
                                       "Learning Rate",
                                       "Epochs",
                                       "Hidden layers",
                                       "Optimizer",
                                       "Loss Function",
                                       "Train Accuracy",
                                       "Test Accuracy",
                                       "AUC_ROC",
                                       "Description"
                                      ])
#Logistic Regression
exp_name = "HCDR"
experimentLog1.loc[len(experimentLog1)] = "HCDR",0.01, 400,4,"Adam","BCELoss",tra

print(experimentLog1.columns)


experimentLog1.loc[len(experimentLog1)] = "HCDR",0.001, 100,4,"Adam","BCEWithLogi


experimentLog1.loc[len(experimentLog1)] = "HCDR",0.01, 100,3,"Adam","BCELoss",tra
```

```
Index(['DataSet', 'Learning Rate', 'Epochs', 'Hidden layers', 'Optimizer',
       'Loss Function', 'Train Accuracy', 'Test Accuracy', 'AUC_ROC',
       'Description'],
      dtype='object')
```

```
experimentLog1
```

| | DataSet | Learning Rate | Epochs | Hidden layers | Optimizer | Loss Function | Train Accuracy | A |
|---|---|---|---|---|---|---|---|---|
| **0** | HCDR | 0.010 | 400 | 4 | Adam | BCELoss | tensor(0.9388) | |
| **1** | HCDR | 0.001 | 100 | 4 | Adam | BCEWithLogitsLoss | tensor(0.9059) | |

## 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.
```

```
test_class_scores[0:10]
```

```
test_results[0:10]
```

```
[{'test_loss': 0.0, 'test_acc_epoch': 0.918588399887085}]
```

```
predictions, actuals = list(), list()
for inputs, targets in val_loader:
    yhat = mlp(inputs)
    yhat = yhat.detach().numpy()
    actual = targets.numpy()
    actual = actual.reshape((len(actual), 1))
    yhat = yhat.round()
    predictions.append(yhat)
    actuals.append(actual)
predictions, actuals = vstack(predictions), vstack(actuals)
print("predictions are: ", predictions)
```

```
predictions are:  [[0.]
 [0.]
 [0.]
 ...
 [0.]
 [0.]
 [0.]]
```

```
# Submission dataframe
submit_df = datasets["application_test"][['SK_ID_CURR']]
submit_df['TARGET'] = pd.DataFrame(predictions)

submit_df.head()
```

| | SK_ID_CURR | TARGET |
|---|---|---|
| **0** | 100001 | 0.0 |
| **1** | 100005 | 0.0 |
| **2** | 100013 | 0.0 |
| **3** | 100028 | 0.0 |
| **4** | 100038 | 0.0 |

```
import pandas as pd
import numpy as np

# Convert the NumPy array to a DataFrame
my_dataframe = pd.DataFrame(predictions)

# Save the DataFrame to a CSV file
submit_df.to_csv('submission.csv', index=False)
```

## Kaggle submission via the command line API

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

```
100% 524k/524k [00:03<00:00, 166kB/s]
Successfully submitted to Home Credit Default Risk
```

## Write-up

# Abstract

The project focuses on predicting whether a client can repay a loan or not, using the “Home Credit Default Risk” Kaggle Competition dataset. The aim is to improve the loan experience for clients who have insufficient credit histories by using additional information, such as telco and transactional data. Following extensive exploratory data analysis(EDA), We performed feature engineering, introduced three new features, trained final features on various models such as Logistic Regression, Random Forest and Decision tree with hyper parameter tuning using gridsearch cv. For the final phase of the project, three neural networks were implemented using PyTorch Lightning for classification.The dataset was transformed into a tensor using the 'Data_pre_nn' function, and ReLU was used as the activation function, along with the Adam optimizer. The binary cross-entropy loss function was used, and backward propagation was implemented to minimize the loss. Neural Network 1, with 4 hidden layers, 400 epochs, and a batch size of 100, gave the best results. Neural Network 2 had the same architecture as Neural Network 1, but with 100 epochs and a batch size of 128. Neural Network 3 had a different architecture, with 3 hidden layers and 2 sigmoid output layers. The first neural network performed better compared to other 2 with accuracy of 0.938 and AUC score of 0.580

# Introduction

## Data Description

The HomeCredit_columns_description.csv acts as a data dictioanry.

There are 7 different sources of data:

1. application_train/application_test : This is the main table broken into two files for 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. The testing application data doesn't comes

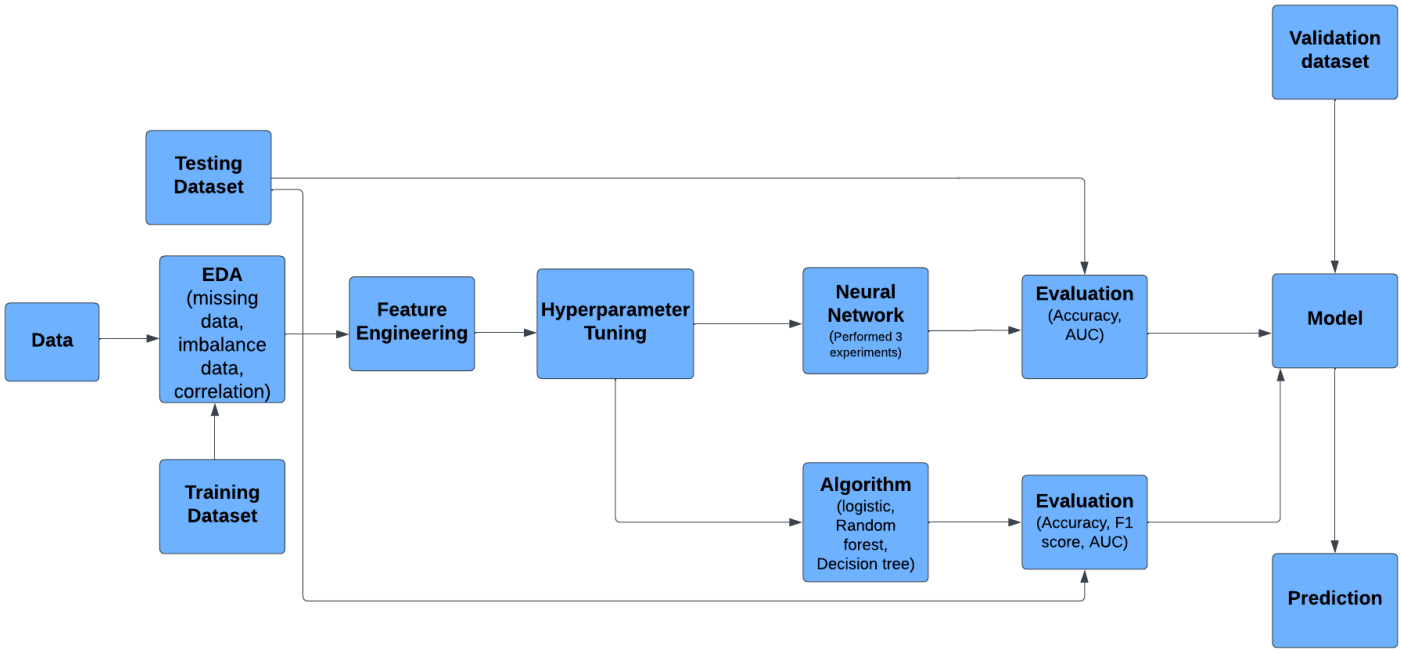
with the TARGET value.

2. bureau : data concerning about the client's previous credits from other financial institutions that were reported to Credit Bureau are recorded. Each prioir credit has its own row in bureau, but one loan in the application data can have multiple previous credits the client had in Credit Bureau before the application date.
3. bureau_balance : monthly balance about the previous credits in the 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.
4. POS_CASH_BALANCE : monthly balance snapshots of previous POS(point of sale) and cash loans that the clients 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.
5. credit_card_balance: monthly balance snapshots about previous credit cards that the clients had with Home Credit. Each row is one month of a credit card balance, and a single credit card can have many rows.
6. previous_application: previous loan applications by the clients at Home Credit of 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.
7. installments_payment: repayment history for the previously disbursed loans at Home Credit. There is one row for every payment made by the client and one row for every missed payment by the same client. One row is equivalent to one payment of one installment or one installment corresponding to one payment of one previous Home credit loans.

## Tasks to be Tackled

The major challenges we faced were choosing the correct architecture. In building an MLP architecture can effect the performance of the model hence choosing the correct parameters such as learning rate, number of epochs, optimizer and activation function plays a major role. To tackle this, we performed multiple experiments by changing the parameters and finally obtained the best AUC score. Along with this we also faced issues with computational complexity as training a neural network model can be time consuming especially with a greater number of epochs.

## Workflow



# Data Lineage

The data was initially downloaded from Kaggle website which was made open by the home credit group in an attempt to make better predictions while classifying a user as a potential defaulter/non-defaulter based on non-conventional user data.

Source Data

Loading Data into Dictionary

Perform EDA on data

Feature Engineering

Preprocessing Pipelines

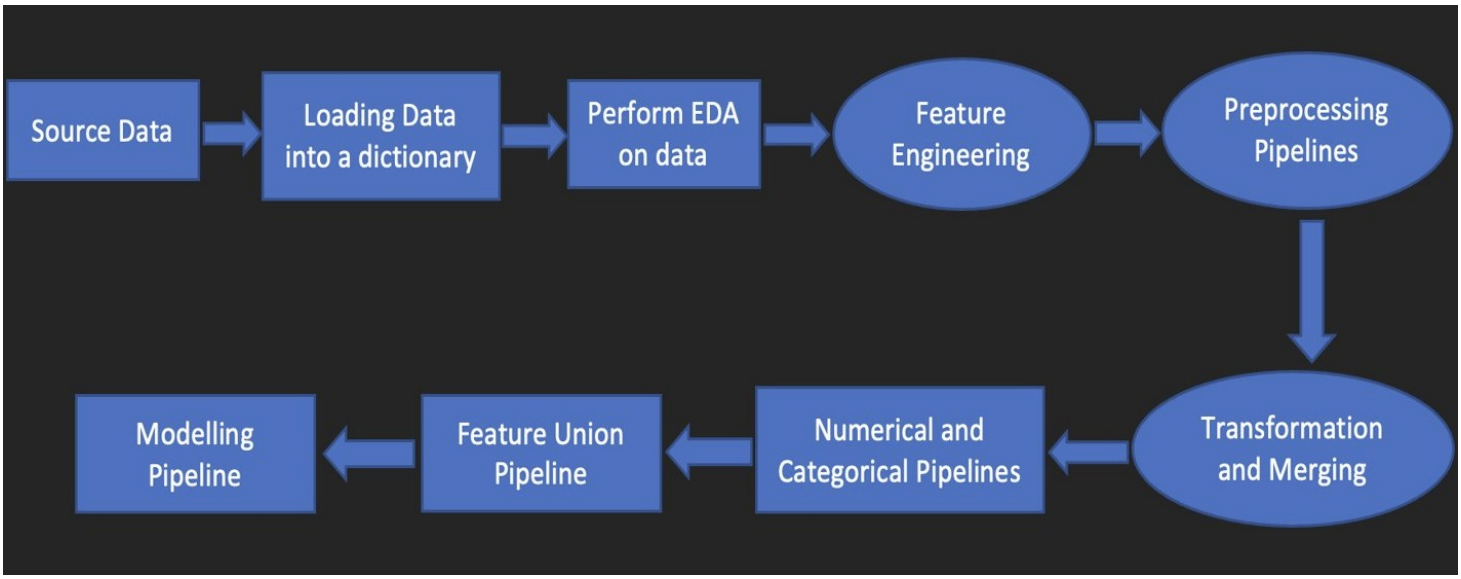
Transformation and Merging

Numerical and Categorical Pipelines

Feature Union Pipeline

Modelling Pipelines

Data lineage pic



# Neural Network

we implemented three Neural Networks using PyTorch Lightning for classification. For the classification of neural network, we transformed the dataset into a tensor using the 'Data_pre_nn' function. This Neural Network has 160 input features and one output feature.

The DataLoader function then converts the train, test and validation data with 160 features. We have used the ReLU (Rectified Linear Unit) function as the activation function and Adam algorithm as an optimizer with a learning rate of 0.01.

For the loss function, we have used the Binary cross entropy loss function, and are using Backward Propagation to minimize the loss.

$$BCE(t, p) = -(t \log(p) + (1 - t) \log(1 - p))$$

Neural Network 1 is implemented with 4 hidden layers, 400 epochs, 100 batch size and below is the string form of the architecture :

$$160 - 128 - \text{ReLU} - 64 - \text{ReLU} - 32 - \text{ReLU} - 16 - \text{ReLU} - 1 \text{ sigmoid}$$

Neural Network 2 is implemented with same 4 hidden layers but100 epochs, 128 batch size and below is the string form of the architecture :

$$160 \text{ - } 128 \text{ - Relu - } 64 \text{ - Relu - } 32 \text{ - Relu - } 16 \text{ - Relu - } 1 \text{ sigmoid}$$

Neural Network 3 is implemented with 3 hidden layers, 100 epochs, 128 batch size and below is the string form of the architecture :

$$160 \text{ - } 128 \text{ - Relu - } 64 \text{ - Relu - } 32 \text{ - Relu - } 2 \text{ sigmoid}$$

Overall, the performance of the model depends on the architecture like learning rate, epochs, the choice of optimizer and loss function. In this case, Neural Network 1 with higher learning rate, more epochs, and with 4 hidden layers gave the best results.

# Leakage

Data Leakage occurs when the training dataset contains the information that we are going to predict, also if the data is taken from outside the training set.

Steps we have taken to avoid data leakage:

We split the data to create a Validation set and kept it held out before working on the model.

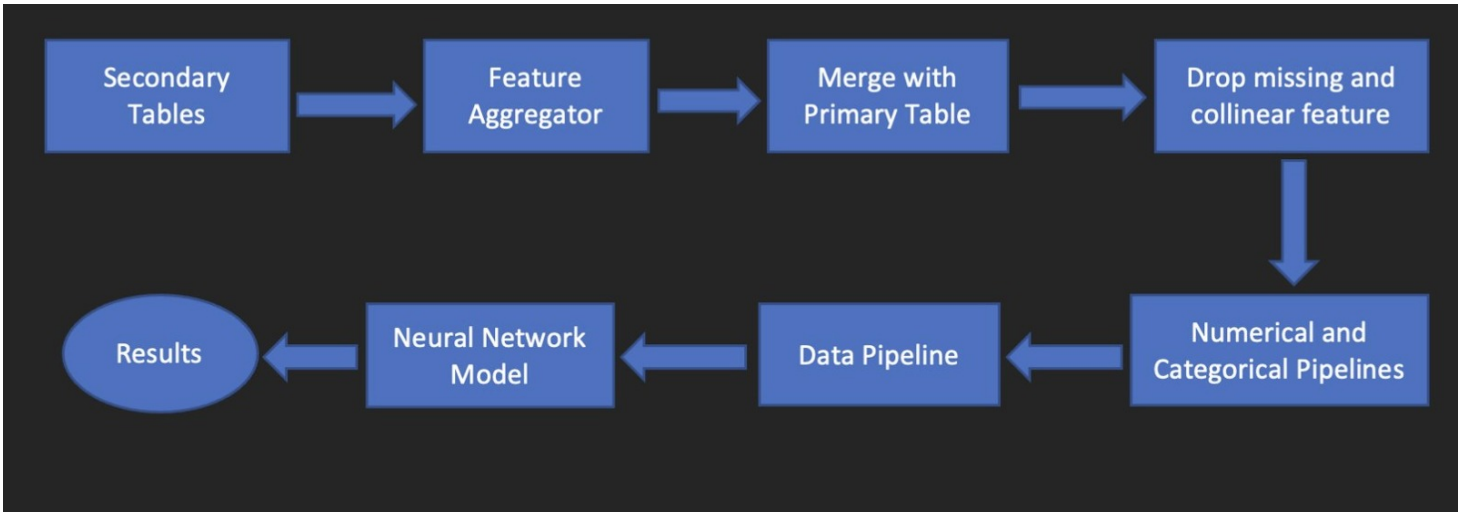
Also, we performed Standardization were done on the train and test set separately, to avoid knowing the distribution of the entire dataset.

We made sure that we did the correct evaluation by using the appropriate metrics, models and careful about the data leakage. Hence, we made sure we avoided any cardinal sins of ML

# Pipelines

There were five different pipelines used in this process:

1. The first pipeline, called the feature aggregator pipeline, was used to generate new features from the secondary tables and merged them with the primary data (application train and application test). After feature aggregation, there are two functions Drop Missing Features and Drop Collinear Features which are used to drop less important features.
2. The second pipeline, the numerical pipeline, was used to select and standardize the numerical features used for the model, using a simple imputer with a strategy as 'mean'.
3. The third pipeline, called the categorical pipeline, was used to select the categorical features, one-hot encode them, and deal with missing values using a simple imputer with a strategy as 'most frequent'.
4. The fourth pipeline, the data pipeline, was used to combine the output of the numerical and categorical pipelines using a feature union pipeline. Finally, the model pipeline was used to train the model with the output of the data pipeline.
5. Finally, we have implemented Neural networks and changed the parameters for each experiment.
6. The last pipeline is used to train the model with the output of the Neural networks.



## Family of input features:

We have splitted the data into numerical and categorical features. we have total of 160 features in which we have 141 numerical features and 19 categorical features for all the experiments conducted for the neural networks

## parameters and settings considered

In Experiment1 we used a higher learning rate of 0.01, trained for 400 epochs with 4 hidden layers, optimized with the Adam optimizer and trained with binary cross-entropy (BCE) loss function, n experiment 2 used a lower learning rate of 0.001, trained for 100 epochs with 4 hidden layers, optimized with the Adam optimizer and trained with binary cross-entropy with logits (BCEWithLogits) loss function, In Experiment 3 used a higher learning rate of 0.01, trained for 100 epochs with 3 hidden layers, optimized with the Adam optimizer and trained with binary cross-entropy (BCE) loss function. The goal of this process was to implement the MLP model and find the parameter/architecture that would give better results in predicting the target variable in the Home Credit Default Risk dataset.

## Loss function:

For the loss function, we have used the Binary cross entropy loss function, and are using Backward Propagation to minimize the loss.

$$BCE(t, p) = -(t \log(p) + (1 - t) \log(1 - p))$$

where:

‘t’ is the true label or target value of a binary classification problem, which can only take on values of 0 or 1.

‘p’ is the predicted probability of the positive class (i.e., the class with a label of 1) output by the model, which can range from 0 to 1.

# Number of Experiments

In this phase we have conducted three Experiments for the neural network by changing the parameters. For all the Experiments we have used the adam optimizer and for Experiment 2 and 3 we took the learning rate of 0.001 and 100 epochs. For Experiment 3 we took 400 epochs and learning rate of 0.01. As per the hidden layers we have taken 4 for the 1st two experiments and 3 hidden layers for the last experiment.

## Experimental results

```
1 experimentLog1
```

| | DataSet | Learning Rate | Epochs | Hidden layers | Optimizer | Loss Function | Train Accuracy | Test Accuracy | AUC_ROC | Description |
|---|---|---|---|---|---|---|---|---|---|---|
| 0 | HCDR | 0.010 | 400 | 4 | Adam | BCELoss | tensor(0.9388) | 0.918588 | 0.580125 | Multi layer perceptron exp 1 |
| 1 | HCDR | 0.001 | 100 | 4 | Adam | BCEWithLogitsLoss | tensor(0.9059) | 0.918588 | 0.500000 | Multi layer perceptron exp 2 |
| 2 | HCDR | 0.010 | 100 | 3 | Adam | BCELoss | tensor(0.9412) | 0.918588 | 0.530666 | Multi layer perceptron exp 3 |

# Discussion

We have used the 160 features and implemented MLP model using PyTorch lightning in which we have conducted 3 experiments with different parameters. In the result table we have displayed various metrics such as accuracy, AUC, Log loss and also parameters such as optimizer, number of epochs, hidden layers. From the results table, Experiment 1 had the highest AUC_ROC score of 0.580125, followed by experiment 3 with an AUC_ROC score of 0.530666. Experiment 2 had the lowest AUC_ROC score of 0.5. In terms of other metrics, all three experiments had the same test accuracy of 0.918588, however, the train accuracy was highest in experiment 1 with a value of 0.9388, while experiments 2 and 3 had train accuracies of 0.9059 and 0.9412, respectively. Based on these results, we can infer that the first Experiment achieved the highest AUC-ROC score, indicating that it performed the best in predicting the binary output for the given dataset. However, the other two models also achieved a high test accuracy, indicating that they were also able to learn the underlying patterns in the data. We can see that Model 2 has achieved the highest train accuracy, which may suggest that it overfits to the training data. Finally, the different loss functions and hyperparameters used in the models led to the differences in their performance.

# Four Ps

Past: The Home Credit Default Risk project aims to provide loans and financial services to unbanked and low-credit-scored customers. In the past, we denormalized and merged the primary and secondary tables and performed exploratory and visual data analysis. We also trained machine learning models such as Logistic Regression and Random forest and obtained accuracy and AUC scores. Later, we focused on feature engineering and hyperparameter tuning. We used functions to drop columns with missing values and collinear features and created three new features based on domain knowledge. We then trained a Logistic Regression model using grid search and cross-validation to find the best hyperparameters. Finally, we tested the model on the Kaggle test set and made a submission.

Present: In the present, we focused on implementing Neural Networks using PyTorch Lightning for classification. For the classification of neural network, we transformed the dataset into a tensor using the ‘Data_pre_nn’ function. We have used the ReLU (Rectified Linear Unit) function

as the activation function. We are using the Adam algorithm as an optimizer with a learning rate of 0.01. For the loss function, we have used the Binary cross entropy loss function, and are using Backward Propagation to minimize the loss.

Plan: Finally, the obtained results for the MLP were not as expected this may be due to various reasons. The logistic regression gave the best AUC score.

Problems - Some of the problems we faced were the large sizes of the datasets, which led to long running times and kernel crashes. Additionally, we had to deal with computational complexity and selecting appropriate parameters for the model for which have conducted experiments by changing the parameters.

## Conclusion

The Home Credit Default Risk Initiative project aims to predict whether a borrower will repay their debt accurately. Our hypothesis follows: ML pipelines used with custom features or by taking only appropriate features instead of all features, might give good predictions for the client's repaying capability. In the phase1 we have performed EDA where we got lot of insights into data and their relations, In the next phase we focused on feature engineering where we used functions to drop columns with missing values and collinear features and created three new features based on domain knowledge. We then conducted hyperparameter tuning using grid search and cross-validation to find the best hyperparameters. After this we implemented an MLP model using PyTorch lightining where we have conducted 3 experiments by changing the parameters. In this we got the best results for Exp1 with highest AUC score of 0.58 and test accuracy of 0.918 when we used the following parameters: 400 epochs, learning rate as 0.01, adam optimizer, BCE loss for the loss function. Overall, we didn’t get expected result using the MLP model as the obtained AUC score is not ideal and less when compared to AUC score obtained from logistic regression model in previous phase.

## References

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

# TODO: Predicting Loan Repayment with Automated Feature Engineering in Featuretools

Read the following:

- feature engineering via Featuretools library:
    - https://github.com/Featuretools/predict-loan-repayment/blob/master/Automated%20Loan%20Repayment.ipynb
- https://www.analyticsvidhya.com/blog/2018/08/guide-automated-feature-engineering-featuretools-python/
- feature engineering paper: https://dai.lids.mit.edu/wp-content/uploads/2017/10/DSAA_DSM_2015.pdf
- https://www.analyticsvidhya.com/blog/2017/08/catboost-automated-categorical-data/