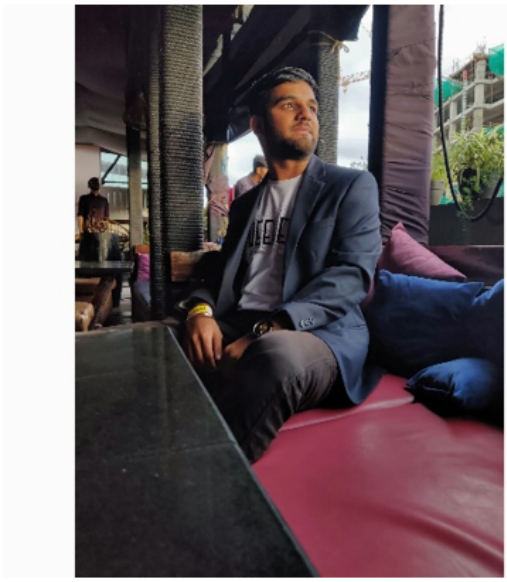
Home Credit Default Risk

Team Members

- Anuj Mahajan - anujmaha@iu.edu
- Shubham Jambhale - sjambhal@iu.edu
- Siddhant Patil - sidpatil@iu.edu
- Shashwati Diware - sdiware@iu.edu

Shubham Jambhale
sjambhal@iu.edu

Siddhant Patil
sidpatil@iu.edu



Anuj Mahajan
anujmaha@iu.edu

Shashwati Diware
sdiware@iu.edu

1.0 FP*GroupN* 11 HCDR

1.1 Phase Leader Plan

Phase	Contributor	Contribution Description
Phase 1: Project Planning	Anuj Mahajan	Download Data, go through data, and load libraries. Create a pipeline diagram and describe the pipeline design. Describe Preprocessing,
Phase 1: Project Planning	Shashwati Diware	Project Abstract, ML Algorithm Names, and describe Metrics.
Phase 1: Project Planning	Shubham Jambhale(Phase Leader)	Understanding the problem statement, and writing table descriptions. Schedule meetings, coordinate tasks, plan phase
Phase 1: Project Planning	Siddhant Patil	Machine Learning Pipeline Steps and describes pipeline components.
Phase 2: Base Line Modelling and EDA	Anuj Mahajan (Phase Leader)	Creating Block Diagram EDA and one slide of the presentation. Schedule meetings, coordinate tasks, plan phase
Phase 2: Base Line Modelling and EDA	Shashwati Diware	Result Analysis EDA and one slide of the presentation.
Phase 2: Base Line Modelling and EDA	Shubham jambhale	Result Analysis and two slides of the presentation
Phase 2: Base Line Modelling and EDA	Siddhant Patil	Result Analysis and two slides of the presentation
Phase 3: Hyperparameter Tuning	Shashwati Diware (Phase Leader)	Testing Accuracy matrix and Schedule meetings, coordinating tasks, the planning phase
Phase 3: Hyperparameter Tuning	Siddhant Patil	Create and develop code for Hyperparameter tuning
Phase 3: Hyperparameter Tuning	Shubham Jambhale	Run and create analysis by testing the confusion / AUC matrix. Coordinate Tasks and one slide of the presentation
Phase 3: Hyperparameter Tuning	Anuj Mahajan	Run and analyze Lasso and ridge regression losses. Coordinate tasks and one slide of the presentation
Phase 4: Final Report Generation	Siddhant Patil (Phase Leader)	Plan Phase Schedule Meetings and Coordinate Tasks, analyze and go through the final results
Phase 4: Final Report Generation	Anuj Mahajan	Rearrange everything and go through the final documentation, list down the final recordings
Phase 4: Final Report Generation	Shashwati Diware	Prepare the final presentation
Phase 4: Final Report Generation	Shubham Jambhale	Check everything and submit the assignment before the deadline

1.2 Credit Assignment Plan

Phase 1:

Task	Task Description	Hours spent	Assigned to	Start	End
Understanding problem statement	Go through the problem statement to understand the requirements	6	Shubham	11/05/22	11/07/22
Data Exploration	Explore and analyze the data for a better understanding	6	Anuj	11/07/22	11/09/22
Project Proposal	Creating the project proposal and preparing a basic report with Abstract, ML models, and Gantt diagram	20	Group	11/09/22	11/14/22

Phase 2:

Task	Task Description	Hours Spent	Assigned to	Start	End
Creating Block Diagram	Creating the block diagram of the basic flow of execution.	5	Anuj	11/13/22	11/15/22
Creating Pipeline Diagram	Creating the pipeline diagram of the machine learning model from analyzing the data till the result analysis	5	Shashwati	11/13/22	11/15/22
Result Analysis	Analyzing the Result	10	Group	11/26/22	11/29/22
PowerPoint Presentation	Simultaneously prepare the PowerPoint presentation and add the analyzed data into it as per need	10	Group	11/20/22	11/29/22

Phase 3:

Task	Task Description	Hours spent	Assigned to	Start	End
Create and develop code for hyperparameter tuning	Design and develop python helper function for hyperparameter tuning	16	Siddhant	11/20/22	11/25/22
Result Analysis	Analysis of Obtained Result	2	Group	12/02/22	12/03/22
Testing Accuracy matrix	Analyzing accuracy using accuracy matrix	2	Shashwati	12/03/22	12/04/22
Analyzing the Loss and AUC	Analyzing Loss and AUC matrix	2	Shubham	12/03/22	12/04/22
Creating Powerpoint presentation	Simultaneously prepare the PowerPoint presentation and add the analyzed data into it as per need	2	Anuj	12/03/22	12/04/22

Phase 4:

Task	Task Description	Hours Spent	Assigned To	Start	End
Build MLP using Pytorch	Build a neural network model using Pytorch	10	Group	12/03/22	12/08/22
Final Documentation	Rearrange everything and go through the final documentation, list down the final recordings	10	Anuj	12/03/22	12/13/22
Final Results	Analyze final results obtained after the final testing	6	Siddhant	12/05/22	12/12/22
Final Presentation	Prepare the final presentation	4	Shashwati	12/06/22	12/13/22
Assignment Submission	Check everything and submit the assignment before the deadline	1	Shubham	12/08/22	12/13/22

1.3 Abstract

The main objective of this project is to create the best machine learning model that can determine whether a loan application will be able to repay the loan. We examined the data in Phase 1 and completed our initial baseline models. We expanded our work from Phase 1 to Phase 2 and applied fundamental feature engineering, where we considered probable characteristics from other tables and constructed base models. In terms of performance on these data, logistic regression performed the best and decision tree and random forest came in second and third, respectively.

In Phase 3, we carefully chose the features from the generated features, analyzed the significance of the features, and applied hyper parameter tuning. We improved upon our baseline model using feature engineering, yielding 4 more characteristics which were Salary-to Credit Ratio, Total External Source AMT Credit to Annuity Ratio, Annuity to Salary Ratio. On baseline models, we performed hyper parameter adjustment, and Lasso and Ridge were also used, and we achieved the test accuracy of 92.13% for Decision Tree with AUC 71.8% for Lasso and Ridge. Our Kaggle Score were 0.71673(private) and 0.73086(public)

In Phase 4, we ran a deep learning algorithm to forecast the project's eventual aim. The Multi-Layer Perceptron was the Deep Learning algorithm that was employed. Implementing MLP and displaying the training model on TensorBoard were the primary objectives of this phase. Our accuracy increased to 92.4% along with Testing AUC of 60.13%. We found out that there is slight leakage in the pipeline and we think that is the reason our Kaggle Score was decreased to 0.504 even after receiving the testing AUC of 0.6013

1.4 Data and Task Description

- Data source:
  - We are planning to use the existing datasets provided by Kaggle. Source: https://www.kaggle.com/c/home-credit-default-risk/data
- POS_CASH_balance.csv:
  - This dataset gives information about previous credit information such as contract status, the number of installments left to pay, DPD(days past due), etc. of the current application.

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

- bureau.csv
  - This dataset gives information about the type of credit, debt, limit, overdue, maximum overdue, annuity, remaining days for previous credit, etc.

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

- bureau_balance.csv:
  - This dataset gives information about the Status of the Credit Bureau loan during the month, the Month of balance relative to the application date, Recoded ID of the Credit Bureau credit. 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.

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

  - credit_card_balance.csv:
    - This dataset gives information about financial transactions aggregated values such as amount received, drawings, number of transactions of previous credit, installments, etc. Each row is one month of a credit card balance, and a single credit card can have many rows.

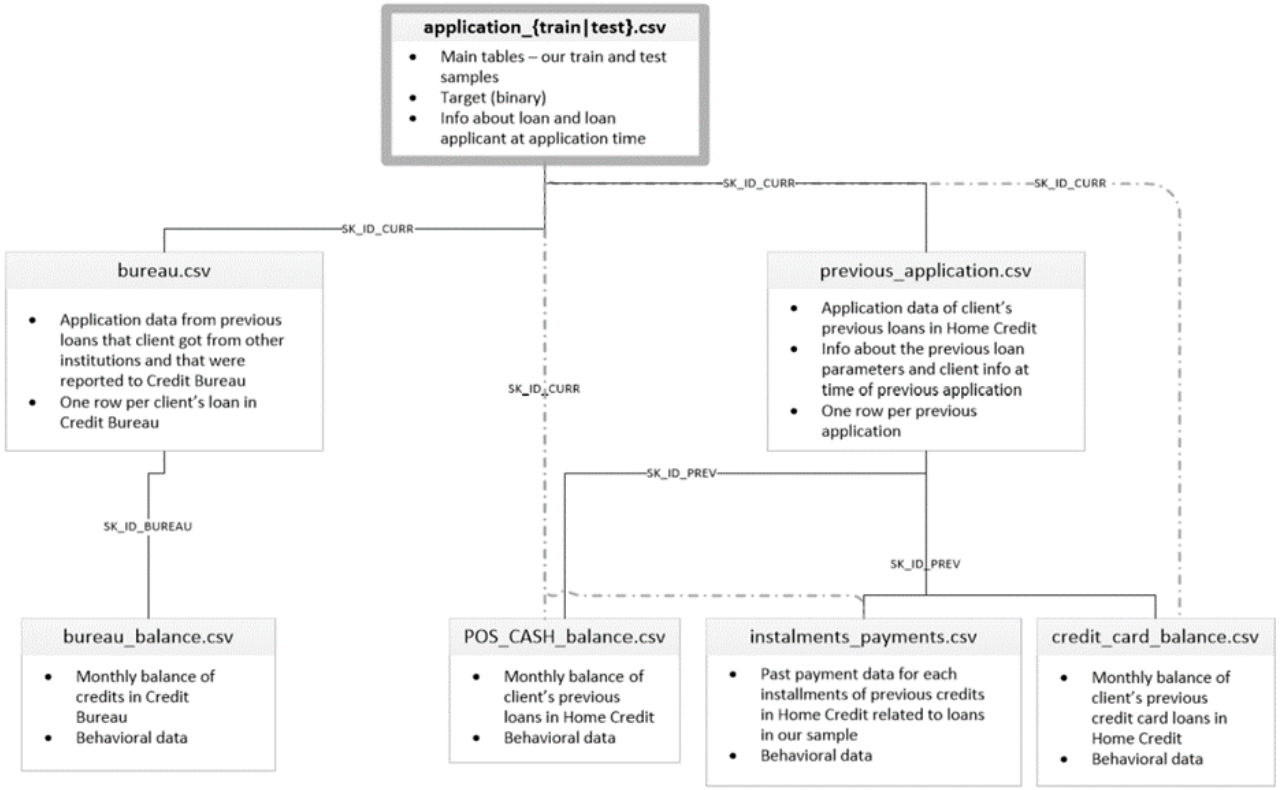
	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	AMT_CREDIT_LIMIT_ACTUAL	AMT_DRAWINGS_ATM_CURRENT	AMT_DRAWINGS_CURRENT
0	2562384	378907	-6	56.970	135000	0.0	877.5
1	2582071	363914	-1	63975.555	45000	2250.0	2250.0
2	1740877	371185	-7	31815.225	450000	0.0	0.0
3	1389973	337855	-4	236572.110	225000	2250.0	2250.0
4	1891521	126868	-1	453919.455	450000	0.0	11547.0

- installments_payments.csv:
  - This dataset gives information about payments, installments supposed to be paid, and their details. There is one row for every made payment and one row for every missed payment.

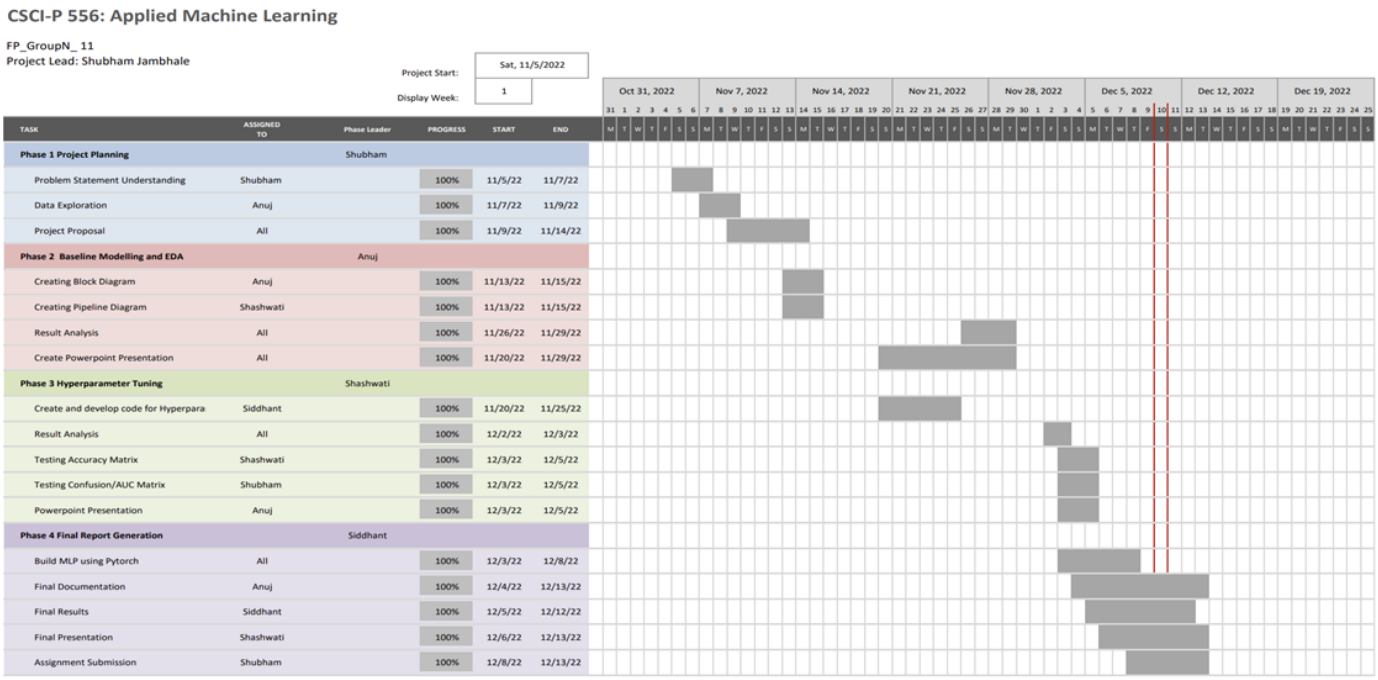
	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER	DAYS_INSTALMENT	DAYS_ENTRY_PAYMENT	AMT_INSTALMENT
0	1054186	161674	1.0	6	-1180.0	-1187.0	6948.360
1	1330831	151639	0.0	34	-2156.0	-2156.0	1716.525
2	2085231	193053	2.0	1	-63.0	-63.0	25425.000
3	2452527	199697	1.0	3	-2418.0	-2426.0	24350.130
4	2714724	167756	1.0	2	-1383.0	-1366.0	2165.040

- previous_application.csv
  - This dataset contains information about previous application details of an application. 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.

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN_PAYMENT
0	2030495	271877	Consumer loans	1730.430	17145.0	17145.0	0.0
1	2802425	108129	Cash loans	25188.615	607500.0	679671.0	NaN
2	2523466	122040	Cash loans	15060.735	112500.0	136444.5	NaN
3	2819243	176158	Cash loans	47041.335	450000.0	470790.0	NaN
4	1784265	202054	Cash loans	31924.395	337500.0	404055.0	NaN



1.5 Gantt Chart



TASK	ASSIGNED TO	Phase Leader	PROGRESS	START	END
Phase 1 Project Planning		Shubham			
Problem Statement Understanding	Shubham		100%	11/5/22	11/7/22
Data Exploration	Anuj		100%	11/7/22	11/9/22
Project Proposal	All		100%	11/9/22	11/14/22
Phase 2 Baseline Modelling and EDA		Anuj			
Creating Block Diagram	Anuj		100%	11/13/22	11/15/22
Creating Pipeline Diagram	Shashwati		100%	11/13/22	11/15/22
Result Analysis	All		100%	11/26/22	11/29/22
Create Powerpoint Presentation	All		100%	11/20/22	11/29/22
Phase 3 Hyperparameter Tuning		Shashwati			
Create and develop code for Hyperpara	Siddhant		100%	11/20/22	11/25/22
Result Analysis	All		100%	12/2/22	12/3/22
Testing Accuracy Matrix	Shashwati		100%	12/3/22	12/5/22
Testing Confusion/AUC Matrix	Shubham		100%	12/3/22	12/5/22
Powerpoint Presentation	Anuj		100%	12/3/22	12/5/22
Phase 4 Final Report Generation		Siddhant			
Build MLP using Pytorch	All		100%	12/3/22	12/8/22
Final Documentation	Anuj		100%	12/4/22	12/13/22
Final Results	Siddhant		100%	12/5/22	12/12/22
Final Presentation	Shashwati		100%	12/6/22	12/13/22
Assignment Submission	Shubham		100%	12/8/22	12/13/22

1.6 Machine Learning Algorithms and Metrics

The outcome of this project is to predict, whether the customer will repay the loan or not. That's why this is a classification task where the outcome is 0 or 1. To classify this problem we will be building the following machine-learning models:

1. Logistics Regression:
   • In our case, the number of features is relatively small i.e. <1000, and no. of examples is large. Hence logistic regression can be a good fit here for the classification.

2. Decision Tree:
   • Decision trees are better for categorical data and our target data is also categorical in nature that's why decision trees are a good fit.

3. Random Forest:
   • Random Forest works well with a mixture of numerical and categorical features. • As we have a good amount of mixture of both types of features random forest can be a good fit.

4. Lasso Regression:
   • The bias-variance trade-off is the basis for Lasso's superiority over least squares. The lasso solution can result in a decrease in variance at the cost of a slight increase in bias when the variance of the least squares estimates is very large. Consequently, this can produce predictions that are more accurate.

5. Ridge Regression:
   • Any data that exhibits multicollinearity can be analyzed using the model-tuning technique known as ridge regression. This technique carries out L2 regularization. Predicted values differ much from real values when the problem of multicollinearity arises, least-squares are unbiased, and variances are significant.

6. Multi-Layer Perceptron:
   • The perceptron can be used to define a linear decision boundary in a binary classification model. The separating hyperplane that reduces the separation between incorrectly classified points and the decision boundary is found. Input and output layers, as well as one or more hidden layers with a densely packed number of neurons, make up a multilayer perceptron.

1.6.1 Loss Function

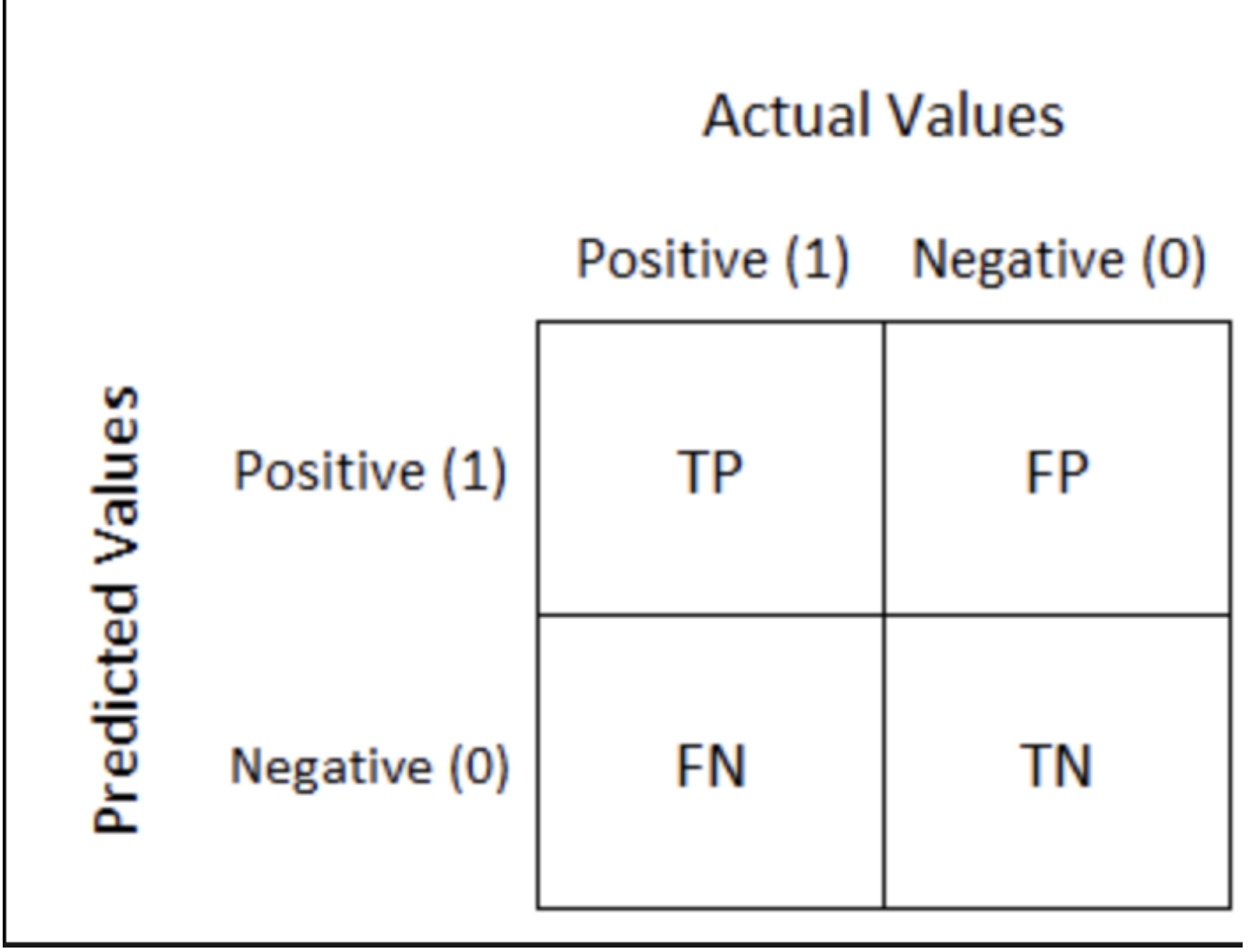
- BCE Loss
  - The binary cross entropy loss between the input and target (predicted and actual) probability is calculated using the BCELoss() function.

1.6.2 Metrics

In [2]:
```
!pip install latexify-py==0.2.0
import math
import latexify
```

```
Requirement already satisfied: latexify-py==0.2.0 in /Users/anujmahajan/opt/anaconda3/lib/
python3.9/site-packages (0.2.0)
Requirement already satisfied: dill>=0.3.2 in /Users/anujmahajan/opt/anaconda3/lib/python
3.9/site-packages (from latexify-py==0.2.0) (0.3.4)
```

1. Confusion Metrics:
   - A confusion matrix, also called an error matrix, is used in the field of machine learning and more specifically in the challenge of classification. Confusion matrices show counts between expected and observed values. The result "TN" stands for True Negative and displays the number of negatively classed cases that were correctly identified. Similar to this, "TP" stands for True Positive and denotes the quantity of correctly identified positive cases. The term "FP" denotes the number of real negative cases that were mistakenly categorized as positive, while "FN" denotes the number of real positive examples that were mistakenly classed as negative. Accuracy is one of the most often used metrics in classification.



1. AUC:
   - AUC stands for "Area under the ROC Curve." It measures the entire two-dimensional area underneath the entire ROC curve from (0,0) to (1,1). It is a widely used accuracy method for binary classification problems.
2. Accuracy:
   - The accuracy score is used to gauge the model's effectiveness by calculating the ratio of total true positives to total true negatives across all made predictions. Accuracy is generally used to calculate binary classification models.

In [2]:
```
@latexify.function(use_math_symbols=True)
def Accuracy():
    return(True_Positives + True_Negatives) / (True_Positives +
True_Negatives + False_Positives + False_Negatives)

Accuracy
```

Out[2]:

$$\text{Accuracy}() = \frac{True_{P}ositives + True_{N}egatives}{True_{P}ositives + True_{N}egatives + False_{P}ositives + False_{N}egatives}$$

In [3]:
```
@latexify.function(use_math_symbols=True)
def logloss():
    return (-1/m*(sum(y*np.log(p)+(1- y)*np.log(1-p))))
logloss
```

Out[3]:

$$\text{logloss}() = \frac{-1}{m}\sum\left(y\log\left(p\right) + \left(1-y\right)\log\left(1-p\right)\right)$$

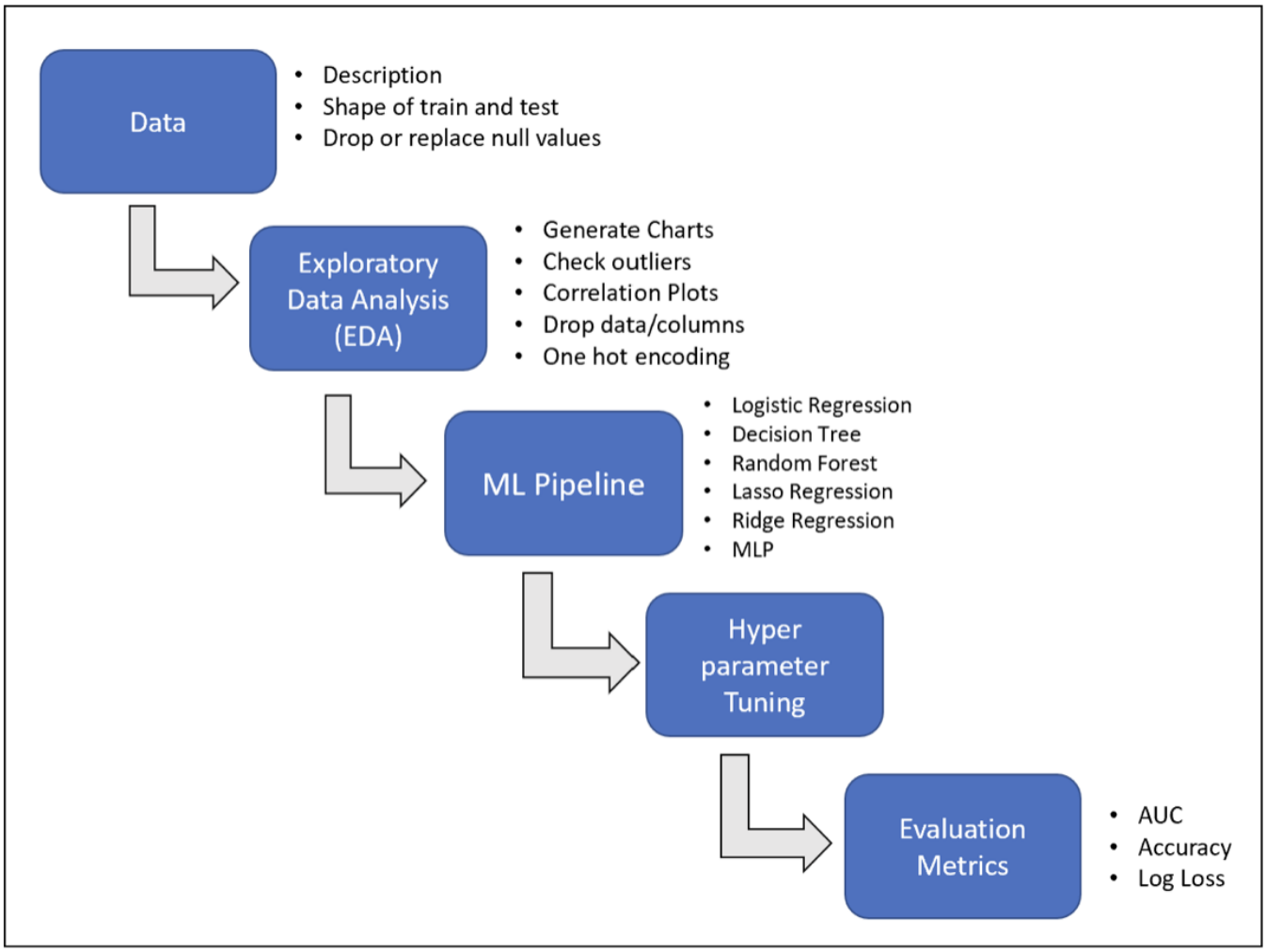
In [3]:

```
@latexify.function(use_math_symbols=True)
def BCELoss():
    return -w((y*np.log(x)+(1- y)*np.log(1-x)))
BCELoss
```

Out[3]:

$$\text{BCELoss}() = -\text{w}\left(y\log\left(x\right) + \left(1-y\right)\log\left(1-x\right)\right)$$

1.7 Machine Learning Pipeline Steps



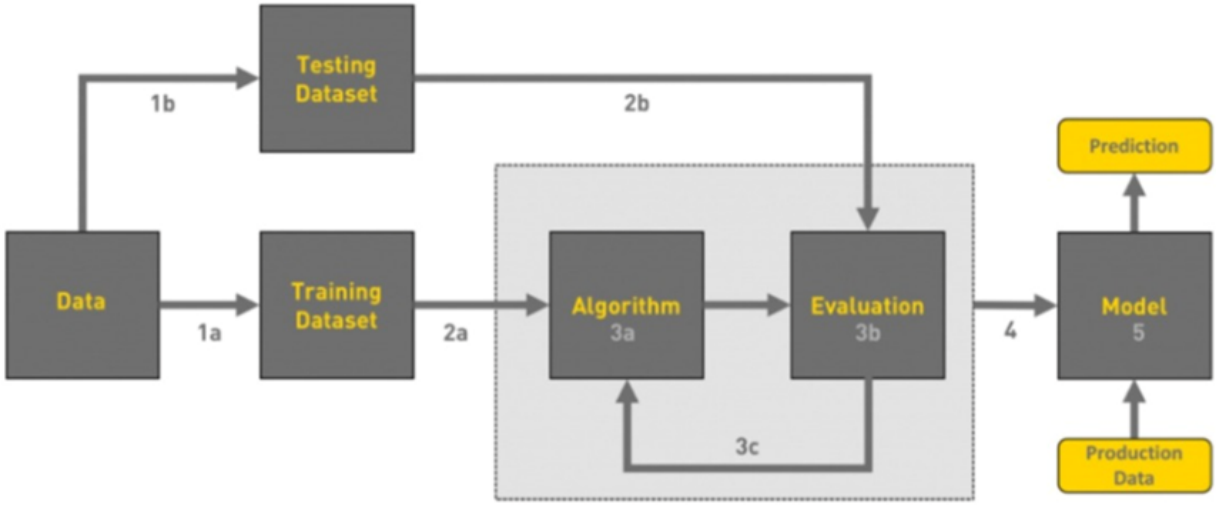
- Data Preprocessing:
  • Convert the raw data set into a clean data set for processing.
  • First, Obtain Kaggle's raw data.
  • On this Raw Data. Analyze exploratory data.

- Feature Engineering:
  • Create a suitable input dataset by performing feature engineering and other processing techniques.
  • Pipeline must not only select the features it wants to create from an unlimited pool of possibilities, but it must also process vast amounts of data to do so. This makes the data appropriate for the model.

- Model Selection:
  • Here, we try on different models for various option purposes.
  • Develop and test several candidate models, such as Random Forest, Decision Making Trees, Logistic Regression, Lasso Regression, Ridge Regression and MLP
  • Using the evaluation function, pick the top model with a good evaluation score.
  • For this selection purposes, employ many measures for evaluation criteria, including "Accuracy", "AUC".

- Prediction Generation:
  • The top performer is then chosen as the winning model when the models are tested on a new set of data that wasn't used during training.
  • Once the best model has been chosen, use it to forecast outcomes based on the fresh data.
  • It is then used to make predictions across all your objects.

1.8 Block Diagram



Overview of the Workflow of ML

Referenced from: https://towardsdatascience.com/workflow-of-a-machine-learning-project-ec1dba419b94

Previous Experiments

Phase 2: Baseline

We built the baseline models as part of phase 2

Below are the experiments we conducted in Phase 2 for baseline models.

	ExpID	Cross fold train accuracy	Test Accuracy	Validation Accuracy	AUC	Accuracy	Loss	Train Time(s)	Test Time(s)	Validation Time(s)	Experiment description
0	Baseline with 81 inputs	92.0	91.9	91.7	0.506443	91.917467	0.246888	12.5042	0.0495	0.0410	Selective Features - Baseline LogisticRegression
1	Baseline Decision Tree with 81 inputs	86.4	86.1	86.3	0.570952	86.148643	2.362000	29.5238	0.0733	0.0594	Selective Features - Baseline Decision Tree
2	Baseline Random Forest with 81 inputs	92.2	92.2	92.0	0.522292	92.187373	0.270439	245.7127	1.2979	1.0884	Selective Features - Baseline Random Forest

Phase 3: Hyper-Parameter Tuning

As part of Phase 3, we hypertuned the model used in phase 2.

We implemented different algorithms during this phase such as Logistic regression, Decison Tree, Random Forest, lasso Regression and Ridge Regression.

We conducted multiple experiments as part of phase 3, below are some of them.

Logistic Regression:

	ExpID	Cross fold train accuracy	Test Accuracy	Valid Accuracy	AUC	Loss	Accuracy	Train Time(s)	Test Time(s)	Experiment description	Best Hyper Parameters
0	Hypertuned Logisitc rgeression with 74 inputs	91.91%	91.98%	91.96%	0.5	0.287519	91.976152	120.5867	0.0364	Hypertuned LogisticRegression with 4 cv, logis...	{'logistic__C': 10, 'logistic__tol': 0.001}
1	Hypertuned Logisitc rgeression with 74 inputs	91.91%	91.98%	91.96%	0.5	0.287519	91.976152	206.4487	0.0150	Hypertuned LogisticRegression with 5 cv, logis...	{'logistic__C': 10, 'logistic__tol': 0.0001}
2	Hypertuned Logisitc rgeression with 74 inputs	91.91%	91.98%	91.96%	0.5	0.287519	91.976152	207.5919	0.0150	Hypertuned LogisticRegression with 10 cv, logi...	{'logistic__C': 10, 'logistic__tol': 0.001}
3	Hypertuned Logisitc rgeression with 74 inputs	91.91%	91.98%	91.96%	0.5	0.287519	91.976152	142.8426	0.0140	Hypertuned LogisticRegression with 5 cv, logis...	{'logistic__C': 10, 'logistic__tol': 0.001}

Decision Tree:

	ExpID	Cross fold train accuracy	Test Accuracy	Valid Accuracy	AUC	Loss	Accuracy	Train Time(s)	Test Time(s)	Experiment description	Best Hyper Parameters
0	Hypertuned Decision Tree with 74 inputs	92.19%	92.13%	92.03%	0.535858	0.248343	92.130081	71.7969	0.0530	Hypertuned Decision Tree with 2 cv, Max Depth ...	{'decisionTree__criterion': 'gini', 'decisionT...
1	Hypertuned Decision Tree with 74 inputs	91.91%	91.98%	91.96%	0.500000	0.257044	91.976152	56.4094	0.0515	Hypertuned Decision Tree with 3 cv, Max Depth ...	{'decisionTree__criterion': 'gini', 'decisionT...
2	Hypertuned Decision Tree with 74 inputs	91.91%	91.98%	91.96%	0.500000	0.257044	91.976152	109.6553	0.0590	Hypertuned Decision Tree with 5 cv, Max Depth ...	{'decisionTree__criterion': 'gini', 'decisionT...
3	Hypertuned Decision Tree with 74 inputs	91.91%	91.98%	91.96%	0.500000	0.257044	91.976152	55.6498	0.0525	Hypertuned Decision Tree with 5 cv, Max Depth ...	{'decisionTree__criterion': 'gini', 'decisionT...

Random Forest:

	ExpID	Cross fold train accuracy	Test Accuracy	Valid Accuracy	AUC	Loss	Accuracy	Train Time(s)	Test Time(s)	Experiment description	Best Hyper Parameters
0	Hypertuned random Forest with 74 inputs	91.91%	91.98%	91.96%	0.5	0.269295	91.976152	147.6134	0.125	Hypertuned Random Forest with 3 cv, Max Depth ...	{'randomForest__max_depth': 1, 'randomForest__...
1	Hypertuned random Forest with 74 inputs	91.91%	91.98%	91.96%	0.5	0.268306	91.976152	262.9420	0.191	Hypertuned Random Forest with 3 cv, Max Depth ...	{'randomForest__max_depth': 1, 'randomForest__...
2	Hypertuned random Forest with 74 inputs	91.91%	91.98%	91.96%	0.5	0.269295	91.976152	68.3533	0.126	Hypertuned Random Forest with 5 cv, Max Depth ...	{'randomForest__max_depth': 1, 'randomForest__...

Lasso and Ridge Regression:

	ExpID	Cross fold train accuracy	Test Accuracy	Valid Accuracy	AUC	Train Time(s)	Test Time(s)	Experiment description	Best Hyper Parameters
0	Hypertuned Lasso Regression with 74 inputs	-6.90%	-6.87%	-6.89%	0.755738	37.7049	0.0260	Hypertuned Lasso Regression with 5 cv and alph...	{'randomForest__max_depth': 1, 'randomForest__...
1	Hypertuned Lasso Regression with 74 inputs	-6.90%	-6.87%	-6.89%	0.755738	31.7134	0.0262	Hypertuned Lasso Regression with 4 cv and alph...	{'randomForest__max_depth': 1, 'randomForest__...
2	Hypertuned Lasso Regression with 74 inputs	-6.90%	-6.87%	-6.89%	0.755738	24.2078	0.0270	Hypertuned Lasso Regression with 3 cv and alph...	{'randomForest__max_depth': 1, 'randomForest__...
3	Hypertuned Ridge Regression with 74 inputs	-6.89%	-6.87%	-6.89%	0.756875	8.0231	0.0250	Hypertuned Ridge Regression with 5 cv and alph...	{'randomForest__max_depth': 1, 'randomForest__...
4	Hypertuned Ridge Regression with 74 inputs	-6.89%	-6.87%	-6.89%	0.756874	8.1149	0.0278	Hypertuned Ridge Regression with 4 cv and alph...	{'randomForest__max_depth': 1, 'randomForest__...
5	Hypertuned Ridge Regression with 74 inputs	-6.89%	-6.87%	-6.89%	0.756862	7.0266	0.0270	Hypertuned Ridge Regression with 3 cv and alph...	{'randomForest__max_depth': 1, 'randomForest__...

Phase 4: MLP

In [4]:

```
from sklearn.base import BaseEstimator, TransformerMixin
import seaborn as sns

from sklearn.preprocessing import MinMaxScaler

from pandas.plotting import scatter_matrix
from sklearn.preprocessing import StandardScaler

from sklearn.neural_network import MLPClassifier
from sklearn.compose import ColumnTransformer
import warnings
warnings.filterwarnings('ignore')

import warnings
warnings.simplefilter('ignore')
import seaborn as sea
import matplotlib.pyplot as plt
%matplotlib inline

from sklearn.model_selection import train_test_split

from sklearn.linear_model import LinearRegression

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
import re
```

```
from time import time
from scipy import stats
import json
from sklearn.pipeline import Pipeline, FeatureUnion

from IPython.display import display, Math, Latex
from numpy import vstack
from pandas import read_csv
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import accuracy_score
from torch.utils.data import Dataset
from torch.utils.data import DataLoader
from torch.utils.data import random_split
from torch import Tensor
from torch.nn import Linear
from torch.nn import ReLU
from sklearn.preprocessing import StandardScaler
from torch.nn import Sigmoid
from torch.nn import Module
from sklearn.preprocessing import OneHotEncoder

from sklearn.model_selection import ShuffleSplit


from sklearn.metrics import roc_auc_score, log_loss, accuracy_score
from sklearn.metrics import confusion_matrix

from torch.optim import SGD
from torch.nn import BCELoss
from torch.nn.init import kaiming_uniform_
from torch.nn.init import xavier_uniform_
from sklearn.metrics import make_scorer, roc_auc_score
from torch.utils.tensorboard import SummaryWriter
from sklearn.preprocessing import MinMaxScaler

import numpy as np
import pandas as pd
import os
import zipfile

#Loading of dataset in dictionary is referreed from professors base notebook from github

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

datasets['application_train'].shape
```

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

SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDR

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

5 rows × 122 columns

Out[4]:
```
(307511, 122)
```

In [5]:
```
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	CNT_CHILDREN	AMT
0	100001	Cash loans	F	N	Y	0	
1	100005	Cash loans	M	N	Y	0	
2	100013	Cash loans	M	Y	Y	0	
3	100028	Cash loans	F	N	Y	2	
4	100038	Cash loans	M	Y	N	1	

5 rows × 121 columns

In [6]:
```
ds_names = ("application_train", "application_test", "bureau","bureau_balance","credit_ca
            "previous_application","POS_CASH_balance")

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

```
application_train: shape is (307511, 122)
<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_REALTY	CNT_CHILDR
0	100002	1	Cash loans	M	N	Y	
1	100003	0	Cash loans	F	N	N	
2	100004	0	Revolving loans	M	Y	Y	
3	100006	0	Cash loans	F	N	Y	
4	100007	0	Cash loans	M	N	Y	

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	CNT_CHILDREN	AMT
0	100001	Cash loans	F	N	Y	0	
1	100005	Cash loans	M	N	Y	0	
2	100013	Cash loans	M	Y	Y	0	
3	100028	Cash loans	F	N	Y	2	
4	100038	Cash loans	M	Y	N	1	

5 rows × 121 columns

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

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

```
bureau_balance: shape is (27299925, 3)
<class 'pandas.core.frame.DataFrame'>
```

```
RangeIndex: 27299925 entries, 0 to 27299924
Data columns (total 3 columns):
 #   Column          Dtype
---  ------          -----
 0   SK_ID_BUREAU    int64
 1   MONTHS_BALANCE  int64
 2   STATUS          object
dtypes: int64(2), object(1)
memory usage: 624.8+ MB
None
```

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

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

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

5 rows × 23 columns

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

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER	DAYS_INSTALMENT	DAYS_EI
0	1054186	161674	1.0	6	-1180.0	
1	1330831	151639	0.0	34	-2156.0	
2	2085231	193053	2.0	1	-63.0	
3	2452527	199697	1.0	3	-2418.0	
4	2714724	167756	1.0	2	-1383.0	

```
previous_application: shape is (1670214, 37)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1670214 entries, 0 to 1670213
Data columns (total 37 columns):
 #   Column                       Non-Null Count    Dtype
---  ------                       --------------    -----
 0   SK_ID_PREV                   1670214 non-null  int64
 1   SK_ID_CURR                   1670214 non-null  int64
 2   NAME_CONTRACT_TYPE           1670214 non-null  object
 3   AMT_ANNUITY                  1297979 non-null  float64
 4   AMT_APPLICATION              1670214 non-null  float64
 5   AMT_CREDIT                   1670213 non-null  float64
 6   AMT_DOWN_PAYMENT             774370 non-null   float64
 7   AMT_GOODS_PRICE              1284699 non-null  float64
 8   WEEKDAY_APPR_PROCESS_START   1670214 non-null  object
 9   HOUR_APPR_PROCESS_START      1670214 non-null  int64
 10  FLAG_LAST_APPL_PER_CONTRACT  1670214 non-null  object
 11  NFLAG_LAST_APPL_IN_DAY       1670214 non-null  int64
 12  RATE_DOWN_PAYMENT            774370 non-null   float64
 13  RATE_INTEREST_PRIMARY        5951 non-null     float64
 14  RATE_INTEREST_PRIVILEGED     5951 non-null     float64
 15  NAME_CASH_LOAN_PURPOSE       1670214 non-null  object
 16  NAME_CONTRACT_STATUS         1670214 non-null  object
 17  DAYS_DECISION                1670214 non-null  int64
 18  NAME_PAYMENT_TYPE            1670214 non-null  object
 19  CODE_REJECT_REASON           1670214 non-null  object
 20  NAME_TYPE_SUITE              849809 non-null   object
 21  NAME_CLIENT_TYPE             1670214 non-null  object
 22  NAME_GOODS_CATEGORY          1670214 non-null  object
 23  NAME_PORTFOLIO               1670214 non-null  object
 24  NAME_PRODUCT_TYPE            1670214 non-null  object
 25  CHANNEL_TYPE                 1670214 non-null  object
 26  SELLERPLACE_AREA             1670214 non-null  int64
 27  NAME_SELLER_INDUSTRY         1670214 non-null  object
 28  CNT_PAYMENT                  1297984 non-null  float64
 29  NAME_YIELD_GROUP             1670214 non-null  object
```

```
 30  PRODUCT_COMBINATION          1669868 non-null  object
 31  DAYS_FIRST_DRAWING           997149 non-null   float64
 32  DAYS_FIRST_DUE               997149 non-null   float64
 33  DAYS_LAST_DUE_1ST_VERSION    997149 non-null   float64
 34  DAYS_LAST_DUE                997149 non-null   float64
 35  DAYS_TERMINATION             997149 non-null   float64
 36  NFLAG_INSURED_ON_APPROVAL    997149 non-null   float64
dtypes: float64(15), int64(6), object(16)
memory usage: 471.5+ MB
None
```

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_DOWN
0	2030495	271877	Consumer loans	1730.430	17145.0	17145.0	
1	2802425	108129	Cash loans	25188.615	607500.0	679671.0	
2	2523466	122040	Cash loans	15060.735	112500.0	136444.5	
3	2819243	176158	Cash loans	47041.335	450000.0	470790.0	
4	1784265	202054	Cash loans	31924.395	337500.0	404055.0	

5 rows × 37 columns

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

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

In [7]:
```
for ds_name in datasets.keys():
    print(f'dataset {ds_name:24}: [ {datasets[ds_name].shape[0]:10,}, {datasets[ds_name].s
```

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

In [8]:
```
data = datasets['application_train'].copy()
y = data['TARGET']
X = data.drop(['SK_ID_CURR','TARGET'], axis = 1)
```

FETCHING IMPORTANT RELAVENT FEATURES

In [9]:
```
data.corrwith(data["TARGET"])
```

Out[9]:
```
SK_ID_CURR                   -0.002108
TARGET                        1.000000
CNT_CHILDREN                  0.019187
AMT_INCOME_TOTAL             -0.003982
AMT_CREDIT                   -0.030369
                                ...
AMT_REQ_CREDIT_BUREAU_DAY     0.002704
AMT_REQ_CREDIT_BUREAU_WEEK    0.000788
AMT_REQ_CREDIT_BUREAU_MON    -0.012462
AMT_REQ_CREDIT_BUREAU_QRT    -0.002022
AMT_REQ_CREDIT_BUREAU_YEAR    0.019930
Length: 106, dtype: float64
```

In [10]:
```
#Fetched some of the important features from dtaset on the basis of their correlation
imp_ftr = ['ELEVATORS_MEDI', 'NAME_TYPE_SUITE', 'AMT_GOODS_PRICE', 'FLAG_DOCUMENT_3', 'YEA
imp_ftr += ['FLAG_OWN_CAR','AMT_CREDIT','AMT_ANNUITY','DAYS_EMPLOYED','OWN_CAR_AGE','CODE_
```

In [11]:
```
class DataFrameSelector(BaseEstimator, TransformerMixin):
    def __init__(self, attribute_names):
        self.attribute_names = attribute_names
    def fit(self, X, y=None):
        return self
    def transform(self, X):
        return X[self.attribute_names].values
```

In [12]:
```
def fillNullValues(NullValColumns, finalFtrs):
    for column in NullValColumns:
        if column == 'AMT_REQ_CREDIT_BUREAU_HOUR':
            finalFtrs[column].fillna(0,inplace=True)
        elif column == 'AMT_REQ_CREDIT_BUREAU_WEEK':
            finalFtrs[column].fillna(0,inplace=True)
        elif column == 'AMT_REQ_CREDIT_BUREAU_DAY':
            finalFtrs[column].fillna(0,inplace=True)
        elif column == 'AMT_REQ_CREDIT_BUREAU_MON':
            finalFtrs[column].fillna(0,inplace=True)
        elif column == 'AMT_REQ_CREDIT_BUREAU_YEAR':
            finalFtrs[column].fillna(0,inplace=True)
        elif column == 'AMT_REQ_CREDIT_BUREAU_QRT':
            finalFtrs[column].fillna(0,inplace=True)
        elif column == 'OBS_30_CNT_SOCIAL_CIRCLE':
            finalFtrs[column].fillna(0,inplace=True)
        elif column == 'DEF_30_CNT_SOCIAL_CIRCLE':
            finalFtrs[column].fillna(0,inplace=True)
        elif column == 'OBS_60_CNT_SOCIAL_CIRCLE':
            finalFtrs[column].fillna(0,inplace=True)
        elif column == 'DEF_60_CNT_SOCIAL_CIRCLE':
            finalFtrs[column].fillna(0,inplace=True)
        elif column == 'CNT_FAM_MEMBERS':
            finalFtrs[column].fillna(0,inplace=True)
    return finalFtrs
```

```
#Filling all null AMT_GOODS_PRICE values with median
def correct_AMT_GOODS_PRICE_val(c):
    if c['AMT_GOODS_PRICE'] == -np.inf:
        return AMTGoodPrice[AMTGoodPrice['NAME_FAMILY_STATUS'] == c['NAME_FAMILY_STATUS']]
    else:
        return c['AMT_GOODS_PRICE']

def correlationWithEXTSource2(allColumns):
    correlation_df = pd.DataFrame()
    for col in allColumns:
        if finalFtrs[col].dtype == 'int':
            l = [col, finalFtrs['EXT_SOURCE_2'].corr(finalFtrs[col])]
        else:
            l = [col, finalFtrs['EXT_SOURCE_2'].corr(pd.DataFrame(LabelEncoder().fit_trans
        correlation_df = correlation_df.append(pd.Series(l),ignore_index=True)
    correlation_df = correlation_df.rename(columns={0:'Column Name',1:'Correlation with EX
    correlation_df['Correlation with EXT 2'] = correlation_df['Correlation with EXT 2'] *
    return correlation_df


def correct_ext_source_2(e):
    if e['EXT_SOURCE_2'] != -np.inf:
        return e['EXT_SOURCE_2']
    else:
        return region_rating_client[region_rating_client['REGION_RATING_CLIENT'] == e['REG



def correlationWithEXTSource3(allColumns):
    correlation_df = pd.DataFrame()
    for col in allColumns:
        if finalFtrs[col].dtype == 'int':
            l = [col, finalFtrs['EXT_SOURCE_3'].corr(finalFtrs[col])]
        else:
            l = [col, finalFtrs['EXT_SOURCE_3'].corr(pd.DataFrame(LabelEncoder().fit_trans
        correlation_df = correlation_df.append(pd.Series(l),ignore_index=True)
    correlation_df = correlation_df.rename(columns={0:'Column Name',1:'Correlation with EX
    correlation_df['Correlation with EXT 3'] = correlation_df['Correlation with EXT 3'] *
    correlation_df = correlation_df.sort_values(by='Correlation with EXT 3',ascending = Fa
    return correlation_df


def get_cat_features(X):
    return X.select_dtypes(include=['object']).columns.tolist()

def get_num_features(X):
    return X.select_dtypes(include=np.number).columns.tolist()
```

In [13]:
```
X.isna().sum()
```

Out[13]:
```
NAME_CONTRACT_TYPE              0
CODE_GENDER                     0
FLAG_OWN_CAR                    0
FLAG_OWN_REALTY                 0
CNT_CHILDREN                    0
                              ...
AMT_REQ_CREDIT_BUREAU_DAY     41519
AMT_REQ_CREDIT_BUREAU_WEEK    41519
AMT_REQ_CREDIT_BUREAU_MON     41519
AMT_REQ_CREDIT_BUREAU_QRT     41519
AMT_REQ_CREDIT_BUREAU_YEAR    41519
Length: 120, dtype: int64
```

In [14]:

```
null_data = X.isna().sum().reset_index().rename(columns={'index':'column_name',0:'null_val
numberOfNullValues = (max(null_data['null_value_count']) * 30) / 100
null_data['count %'] = (null_data['null_value_count'] / len(X)) *100
null_data = null_data[null_data['null_value_count'] <= numberOfNullValues]
null_data
```

Out[14]:

	column_name	null_value_count	count %
0	NAME_CONTRACT_TYPE	0	0.000000
1	CODE_GENDER	0	0.000000
2	FLAG_OWN_CAR	0	0.000000
3	FLAG_OWN_REALTY	0	0.000000
4	CNT_CHILDREN	0	0.000000
...	...	...	...
115	AMT_REQ_CREDIT_BUREAU_DAY	41519	13.501631
116	AMT_REQ_CREDIT_BUREAU_WEEK	41519	13.501631
117	AMT_REQ_CREDIT_BUREAU_MON	41519	13.501631
118	AMT_REQ_CREDIT_BUREAU_QRT	41519	13.501631
119	AMT_REQ_CREDIT_BUREAU_YEAR	41519	13.501631

70 rows × 3 columns

In [15]:

```
selectiveFtr = null_data['column_name'].tolist()
selectiveFtr += ['TARGET']
print(selectiveFtr)
print(len(selectiveFtr))
```

```
['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', 'NA
ME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'REGION
_POPULATION_RELATIVE', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'DAYS_ID_PUBLIS
H', 'FLAG_MOBIL', 'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE', 'FLAG_CONT_MOBILE', 'FLAG_PHONE',
'FLAG_EMAIL', 'CNT_FAM_MEMBERS', 'REGION_RATING_CLIENT', 'REGION_RATING_CLIENT_W_CITY', 'W
EEKDAY_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_CIT
Y_NOT_WORK_CITY', 'LIVE_CITY_NOT_WORK_CITY', 'ORGANIZATION_TYPE', 'EXT_SOURCE_2', 'EXT_SOU
RCE_3', 'OBS_30_CNT_SOCIAL_CIRCLE', 'DEF_30_CNT_SOCIAL_CIRCLE', 'OBS_60_CNT_SOCIAL_CIRCL
E', 'DEF_60_CNT_SOCIAL_CIRCLE', 'DAYS_LAST_PHONE_CHANGE', 'FLAG_DOCUMENT_2', 'FLAG_DOCUMEN
T_3', 'FLAG_DOCUMENT_4', 'FLAG_DOCUMENT_5', 'FLAG_DOCUMENT_6', 'FLAG_DOCUMENT_7', 'FLAG_DO
CUMENT_8', 'FLAG_DOCUMENT_9', 'FLAG_DOCUMENT_10', 'FLAG_DOCUMENT_11', 'FLAG_DOCUMENT_12',
'FLAG_DOCUMENT_13', 'FLAG_DOCUMENT_14', 'FLAG_DOCUMENT_15', 'FLAG_DOCUMENT_16', 'FLAG_DOCU
MENT_17', 'FLAG_DOCUMENT_18', 'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20', 'FLAG_DOCUMENT_21',
'AMT_REQ_CREDIT_BUREAU_HOUR', 'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK',
'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QRT', 'AMT_REQ_CREDIT_BUREAU_YEAR', 'T
ARGET']
71
```

In [16]:

```
finalFtrs = data[selectiveFtr]
finalFtrs['NAME_TYPE_SUITE'].fillna('dummy', inplace=True)
print('Unique values of NAME_TYPE_SUITE')
finalFtrs.NAME_TYPE_SUITE.unique()
```

Out[16]:

```
Unique values of NAME_TYPE_SUITE
array(['Unaccompanied', 'Family', 'Spouse, partner', 'Children',
       'Other_A', 'dummy', 'Other_B', 'Group of people'], dtype=object)
```

In [17]:
```
finalFtrs.dropna(subset=['DAYS_LAST_PHONE_CHANGE'], inplace=True)
```

In [18]:
```
finalFtrs = finalFtrs.reset_index(drop=True)
```

In [19]:
```
NullValColumns = null_data[null_data['null_value_count'] != 0].reset_index(drop=True)['col
NullValColumns
```

Out[19]:
```
['AMT_ANNUITY',
 'AMT_GOODS_PRICE',
 'NAME_TYPE_SUITE',
 'CNT_FAM_MEMBERS',
 'EXT_SOURCE_2',
 'EXT_SOURCE_3',
 'OBS_30_CNT_SOCIAL_CIRCLE',
 'DEF_30_CNT_SOCIAL_CIRCLE',
 'OBS_60_CNT_SOCIAL_CIRCLE',
 'DEF_60_CNT_SOCIAL_CIRCLE',
 'DAYS_LAST_PHONE_CHANGE',
 'AMT_REQ_CREDIT_BUREAU_HOUR',
 'AMT_REQ_CREDIT_BUREAU_DAY',
 'AMT_REQ_CREDIT_BUREAU_WEEK',
 'AMT_REQ_CREDIT_BUREAU_MON',
 'AMT_REQ_CREDIT_BUREAU_QRT',
 'AMT_REQ_CREDIT_BUREAU_YEAR']
```

In [20]:
```
finalFtrs = fillNullValues(NullValColumns, finalFtrs)
finalFtrs
```

Out[20]:

	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_CHILDREN	AMT_INCOM
0	Cash loans	M	N	Y	0	
1	Cash loans	F	N	N	0	
2	Revolving loans	M	Y	Y	0	
3	Cash loans	F	N	Y	0	
4	Cash loans	M	N	Y	0	
...	...	...	...	...	...	
307505	Cash loans	M	N	N	0	
307506	Cash loans	F	N	Y	0	
307507	Cash loans	F	N	Y	0	
307508	Cash loans	F	N	Y	0	
307509	Cash loans	F	N	N	0	

307510 rows × 71 columns

In [21]:
```
AMTGoodPrice = finalFtrs[['AMT_GOODS_PRICE','NAME_FAMILY_STATUS']]
AMTGoodPrice = AMTGoodPrice.groupby('NAME_FAMILY_STATUS')
AMTGoodPrice.head()
```

Out[21]:

	AMT_GOODS_PRICE	NAME_FAMILY_STATUS

	AMT_GOODS_PRICE	NAME_FAMILY_STATUS
0	351000.0	Single / not married
1	1129500.0	Married
2	135000.0	Single / not married
3	297000.0	Civil marriage
4	513000.0	Single / not married
5	454500.0	Married
6	1395000.0	Married
7	1530000.0	Married
8	913500.0	Married
9	405000.0	Single / not married
15	247500.0	Single / not married
18	157500.0	Widow
26	702000.0	Widow
30	477000.0	Civil marriage
31	360000.0	Civil marriage
32	180000.0	Civil marriage
42	238500.0	Civil marriage
58	517500.0	Separated
59	1129500.0	Widow
67	270000.0	Separated
72	459000.0	Separated
74	675000.0	Widow
84	679500.0	Separated
99	675000.0	Widow
110	585000.0	Separated
41981	NaN	Unknown
187347	NaN	Unknown

In [22]:
```
AMTGoodPrice = finalFtrs[['AMT_GOODS_PRICE','NAME_FAMILY_STATUS']]
AMTGoodPrice = AMTGoodPrice.groupby('NAME_FAMILY_STATUS')['AMT_GOODS_PRICE'].median().rese
AMTGoodPrice['AMT_GOODS_PRICE'] = AMTGoodPrice['AMT_GOODS_PRICE'].fillna(AMTGoodPrice['AMT
AMTGoodPrice.head()
```

Out[22]:

	NAME_FAMILY_STATUS	AMT_GOODS_PRICE
0	Civil marriage	450000.0
1	Married	459000.0
2	Separated	450000.0
3	Single / not married	373500.0

	NAME_FAMILY_STATUS	AMT_GOODS_PRICE
4	Unknown	450000.0

In [23]:
```
finalFtrs.dropna(subset=['AMT_ANNUITY'], inplace=True)
```

In [24]:
```
finalFtrs = finalFtrs.reset_index(drop=True)
```

In [25]:
```
#We wanted to fill AMT_GOOD_PRICE with median value wrt group by NAME_FAMILY_STATUS
#.apply method of dataframe allows us to fill all the values depending on the index passed
#referred from - https://pandas.pydata.org/pandas-docs/stable/reference/api/pandas.DataFra
for col in NullValColumns:
    finalFtrs['AMT_GOODS_PRICE'] = finalFtrs['AMT_GOODS_PRICE'].fillna(-np.inf)
    if 'AMT_GOODS_PRICE' in col:
        finalFtrs['AMT_GOODS_PRICE'] = finalFtrs.apply(lambda c: correct_AMT_GOODS_PRICE_v
```

In [26]:
```
print(finalFtrs.columns.tolist())
finalFtrs.shape
```

```
['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', 'NA
ME_INCOME_TYPE', 'NAME_EDUCATION_TYPE', 'NAME_FAMILY_STATUS', 'NAME_HOUSING_TYPE', 'REGION
_POPULATION_RELATIVE', 'DAYS_BIRTH', 'DAYS_EMPLOYED', 'DAYS_REGISTRATION', 'DAYS_ID_PUBLIS
H', 'FLAG_MOBIL', 'FLAG_EMP_PHONE', 'FLAG_WORK_PHONE', 'FLAG_CONT_MOBILE', 'FLAG_PHONE',
'FLAG_EMAIL', 'CNT_FAM_MEMBERS', 'REGION_RATING_CLIENT', 'REGION_RATING_CLIENT_W_CITY', 'W
EEKDAY_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_CIT
Y_NOT_WORK_CITY', 'LIVE_CITY_NOT_WORK_CITY', 'ORGANIZATION_TYPE', 'EXT_SOURCE_2', 'EXT_SOU
RCE_3', 'OBS_30_CNT_SOCIAL_CIRCLE', 'DEF_30_CNT_SOCIAL_CIRCLE', 'OBS_60_CNT_SOCIAL_CIRCL
E', 'DEF_60_CNT_SOCIAL_CIRCLE', 'DAYS_LAST_PHONE_CHANGE', 'FLAG_DOCUMENT_2', 'FLAG_DOCUMEN
T_3', 'FLAG_DOCUMENT_4', 'FLAG_DOCUMENT_5', 'FLAG_DOCUMENT_6', 'FLAG_DOCUMENT_7', 'FLAG_DO
CUMENT_8', 'FLAG_DOCUMENT_9', 'FLAG_DOCUMENT_10', 'FLAG_DOCUMENT_11', 'FLAG_DOCUMENT_12',
'FLAG_DOCUMENT_13', 'FLAG_DOCUMENT_14', 'FLAG_DOCUMENT_15', 'FLAG_DOCUMENT_16', 'FLAG_DOCU
MENT_17', 'FLAG_DOCUMENT_18', 'FLAG_DOCUMENT_19', 'FLAG_DOCUMENT_20', 'FLAG_DOCUMENT_21',
'AMT_REQ_CREDIT_BUREAU_HOUR', 'AMT_REQ_CREDIT_BUREAU_DAY', 'AMT_REQ_CREDIT_BUREAU_WEEK',
'AMT_REQ_CREDIT_BUREAU_MON', 'AMT_REQ_CREDIT_BUREAU_QRT', 'AMT_REQ_CREDIT_BUREAU_YEAR', 'T
ARGET']
```

Out[26]:
```
(307498, 71)
```

In [27]:
```
correlation_df = pd.DataFrame()
allColumns = finalFtrs.columns.tolist()
```

In [28]:
```
correlation_df = correlationWithEXTSource3(allColumns)
```

In [29]:
```
print(correlation_df['Column Name'].tolist())
ext_source_3 = finalFtrs[correlation_df['Column Name'].tolist()+['EXT_SOURCE_3']]
for column in ext_source_3.columns.tolist():
    if column != 'EXT_SOURCE_3':
        ext_source_3[column] = LabelEncoder().fit_transform(finalFtrs[[column]])
print(ext_source_3)
```

```
['TARGET', 'DAYS_ID_PUBLISH', 'FLAG_EMP_PHONE', 'DAYS_REGISTRATION', 'AMT_INCOME_TOTAL',
'REG_CITY_NOT_WORK_CITY', 'REG_CITY_NOT_LIVE_CITY', 'DAYS_LAST_PHONE_CHANGE', 'AMT_REQ_CRE
DIT_BUREAU_YEAR']
        TARGET  DAYS_ID_PUBLISH  FLAG_EMP_PHONE  DAYS_REGISTRATION  \
```

```
0            1             4047                1              12039
1            0             5876                1              14501
2            0             3636                1              11427
3            0             3730                1               5854
4            0             2709                1              11376
...        ...              ...              ...                ...
307493       0             4185                1               7231
307494       0             2077                0              11299
307495       0             1017                1               8950
307496       1             5236                1              13125
307497       0             5757                1              10559

        AMT_INCOME_TOTAL  REG_CITY_NOT_WORK_CITY  REG_CITY_NOT_LIVE_CITY  \
0                   1754                       0                       0
1                   2064                       0                       0
2                    338                       0                       0
3                   1170                       0                       0
4                   1019                       1                       0
...                  ...                     ...                     ...
307493              1407                       0                       0
307494               387                       0                       0
307495              1371                       1                       0
307496              1519                       1                       1
307497              1407                       1                       0

        DAYS_LAST_PHONE_CHANGE  AMT_REQ_CREDIT_BUREAU_YEAR  EXT_SOURCE_3
0                         2638                           1      0.139376
1                         2944                           0           NaN
2                         2957                           0      0.729567
3                         3155                           0           NaN
4                         2666                           0           NaN
...                        ...                         ...           ...
307493                    3499                           0           NaN
307494                    3772                           0           NaN
307495                    1863                           1      0.218859
307496                    3450                           0      0.661024
307497                    2985                           1      0.113922

[307498 rows x 10 columns]
```

In [30]:

```
#Filling null values of EXT_SOURCE_3 with min, median or any other parameter will not be a
#Instead we tried to fill in the null values by predicting the values using linear regress
#We Split the data for EXT_SOURCE_3 into train(Not null values) and test(null values)
#built the model for training data and predicted the test values
def splitEXTSource3(ext_source_3):
    ext_source_3_train = ext_source_3[ext_source_3['EXT_SOURCE_3'].notnull()]
    ext_source_3_test = ext_source_3[ext_source_3['EXT_SOURCE_3'].isnull()]
    ext_source_3_train.shape, ext_source_3_test.shape
    ext_source_3_y_train = ext_source_3_train[['EXT_SOURCE_3']]
    ext_source_3_X_train = ext_source_3_train.drop(columns=['EXT_SOURCE_3'])
    ext_source_3_X_test = ext_source_3_test.drop(columns=['EXT_SOURCE_3'])
    model = LinearRegression().fit(ext_source_3_X_train, ext_source_3_y_train)
    ext_source_3_y_pred = model.predict(ext_source_3_X_test)

    ext_source_3_output = ext_source_3_X_test
    ext_source_3_output['EXT_SOURCE_3_Y'] = ext_source_3_y_pred
    return ext_source_3_output
```

In [31]:

```
ext_source_3_output = splitEXTSource3(ext_source_3)
```

In [32]:

```
ext_source_3_output
```

Out[32]:

	TARGET	DAYS_ID_PUBLISH	FLAG_EMP_PHONE	DAYS_REGISTRATION	AMT_INCOME_TOTAL	REG_CITY_NOT_W
1	0	5876	1	14501	2064	
3	0	3730	1	5854	1170	
4	0	2709	1	11376	1019	
9	0	2175	1	1373	1170	
14	0	4111	1	15072	1659	
...	...	...	...	...	...	
307471	0	6132	1	13156	2398	
307488	0	2387	1	14289	506	
307491	0	5908	1	5889	1371	
307493	0	4185	1	7231	1407	
307494	0	2077	0	11299	387	

60963 rows × 10 columns

In [ ]:

```
```

In [33]:

```
#We predicted values for EXT_SOURCE_3
#Now we need to fill those values in our original datatset
ext_source_3_output = ext_source_3_output.reset_index().rename(columns={'index':'Update In
ext_source3_y_columns = ext_source_3_output['Update Index'].tolist()
for column in ext_source3_y_columns:
    finalFtrs['EXT_SOURCE_3'].iloc[column] = ext_source_3_output[ext_source_3_output['Upda
```

In [34]:

```
correlation_df = pd.DataFrame()
allColumns = finalFtrs.columns.tolist()
```

In [35]:

```
correlation_df = correlationWithEXTSource2(allColumns)
correlation_df.sort_values(by='Correlation with EXT 2',ascending = False).head()
```

Out[35]:

	Column Name	Correlation with EXT 2
26	REGION_RATING_CLIENT	0.292903
27	REGION_RATING_CLIENT_W_CITY	0.288306
43	DAYS_LAST_PHONE_CHANGE	0.195766
70	TARGET	0.160471
11	NAME_EDUCATION_TYPE	0.119436

In [36]:

```
region_rating_client = finalFtrs.groupby('REGION_RATING_CLIENT')['EXT_SOURCE_2'].median()
finalFtrs['EXT_SOURCE_2'] = finalFtrs['EXT_SOURCE_2'].fillna(-np.inf)

finalFtrs['EXT_SOURCE_2'] = finalFtrs.apply(lambda e: correct_ext_source_2(e),axis=1)
region_rating_client.head()
```

Out[36]:

	REGION_RATING_CLIENT	EXT_SOURCE_2

	REGION_RATING_CLIENT	EXT_SOURCE_2
0	1	0.690127
1	2	0.569782
2	3	0.449345

In [37]:
```
finalFtrs.shape
```

Out[37]:
```
(307498, 71)
```

In [38]:
```
finalFtrs.isna().sum()
```

Out[38]:
```
NAME_CONTRACT_TYPE             0
CODE_GENDER                    0
FLAG_OWN_CAR                   0
FLAG_OWN_REALTY                0
CNT_CHILDREN                   0
                              ..
AMT_REQ_CREDIT_BUREAU_WEEK     0
AMT_REQ_CREDIT_BUREAU_MON      0
AMT_REQ_CREDIT_BUREAU_QRT      0
AMT_REQ_CREDIT_BUREAU_YEAR     0
TARGET                         0
Length: 71, dtype: int64
```

In [39]:
```
finalFtrs['Tot_EXTERNAL_SOURCE'] = finalFtrs['EXT_SOURCE_2'] + finalFtrs['EXT_SOURCE_3']
finalFtrs['Annuity_to_salary_ratio'] = finalFtrs['AMT_ANNUITY']/ finalFtrs['AMT_INCOME_TOT
finalFtrs['AMT_CREDIT_TO_ANNUITY_RATIO'] = finalFtrs['AMT_CREDIT'] / finalFtrs['AMT_ANNUIT
finalFtrs['Salary_to_credit'] = finalFtrs['AMT_INCOME_TOTAL']/ finalFtrs['AMT_CREDIT']
```

In [40]:
```
finalFtrs.columns.tolist()
```

Out[40]:
```
['NAME_CONTRACT_TYPE',
 'CODE_GENDER',
 'FLAG_OWN_CAR',
 'FLAG_OWN_REALTY',
 'CNT_CHILDREN',
 'AMT_INCOME_TOTAL',
 'AMT_CREDIT',
 'AMT_ANNUITY',
 'AMT_GOODS_PRICE',
 'NAME_TYPE_SUITE',
 'NAME_INCOME_TYPE',
 'NAME_EDUCATION_TYPE',
 'NAME_FAMILY_STATUS',
 'NAME_HOUSING_TYPE',
 'REGION_POPULATION_RELATIVE',
 'DAYS_BIRTH',
 'DAYS_EMPLOYED',
 'DAYS_REGISTRATION',
 'DAYS_ID_PUBLISH',
 'FLAG_MOBIL',
 'FLAG_EMP_PHONE',
 'FLAG_WORK_PHONE',
 'FLAG_CONT_MOBILE',
 'FLAG_PHONE',
 'FLAG_EMAIL',
 '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_2',
'EXT_SOURCE_3',
'OBS_30_CNT_SOCIAL_CIRCLE',
'DEF_30_CNT_SOCIAL_CIRCLE',
'OBS_60_CNT_SOCIAL_CIRCLE',
'DEF_60_CNT_SOCIAL_CIRCLE',
'DAYS_LAST_PHONE_CHANGE',
'FLAG_DOCUMENT_2',
'FLAG_DOCUMENT_3',
'FLAG_DOCUMENT_4',
'FLAG_DOCUMENT_5',
'FLAG_DOCUMENT_6',
'FLAG_DOCUMENT_7',
'FLAG_DOCUMENT_8',
'FLAG_DOCUMENT_9',
'FLAG_DOCUMENT_10',
'FLAG_DOCUMENT_11',
'FLAG_DOCUMENT_12',
'FLAG_DOCUMENT_13',
'FLAG_DOCUMENT_14',
'FLAG_DOCUMENT_15',
'FLAG_DOCUMENT_16',
'FLAG_DOCUMENT_17',
'FLAG_DOCUMENT_18',
'FLAG_DOCUMENT_19',
'FLAG_DOCUMENT_20',
'FLAG_DOCUMENT_21',
'AMT_REQ_CREDIT_BUREAU_HOUR',
'AMT_REQ_CREDIT_BUREAU_DAY',
'AMT_REQ_CREDIT_BUREAU_WEEK',
'AMT_REQ_CREDIT_BUREAU_MON',
'AMT_REQ_CREDIT_BUREAU_QRT',
'AMT_REQ_CREDIT_BUREAU_YEAR',
'TARGET',
'Tot_EXTERNAL_SOURCE',
'Annuity_to_salary_ratio',
'AMT_CREDIT_TO_ANNUITY_RATIO',
'Salary_to_credit']
```

In [41]:

```
features = finalFtrs.columns.tolist()
features.remove('TARGET')
len(features)
```

Out[41]:

```
74
```

In [42]:

```
lableEncD = {}
trainFtr = finalFtrs
for column in features:
    if trainFtr[column].dtype == 'object':
        lableEnc = LabelEncoder()
        trainFtr[column] = lableEnc.fit_transform(trainFtr[column])
        lableEncD['lableEnc{}'.format(column)] = lableEnc
```

In [43]:
```
X = trainFtr[features]
y = trainFtr['TARGET']
```

In [44]:
```
finalFtrs.to_csv('MLP_input_data.csv', index=False)
```

In [45]:
```
finalFtrs.shape
```

Out[45]:
```
(307498, 75)
```

In [46]:
```
expLog = pd.DataFrame(columns=["ExpID", "Train Time", "Test Time", "Accuracy", "AUC","Comm
```

In [47]:
```
#The MLP Execution is referenced from Professors HW 11 Notebook
#Along with that we referred lot of articles from pytorch official documentation to imple

writer = SummaryWriter()
scaler = MinMaxScaler()
#Define Datasets

class MLPdataset(Dataset):
    # load the dataset
    def __init__(self, path):
        df = read_csv(path).head(30000)
        numAttributes = []
        catAttributes = []

        for c in df.columns.tolist():
            if df[c].dtype in (['int','float']):
                numAttributes.append(c)
            else:
                catAttributes.append(c)
        lableEncD = {}
        for c in df.columns.tolist():
            if df[c].dtype == 'object':
                lableEnc = LabelEncoder()
                df[c] = df[col].fillna("NULL")
                df[c] = lableEnc.fit_transform(df[c])
                lableEncD['lableEnc{}'.format(c)] = lableEnc

        # Input Output storing
        self.X = df.drop(columns=['TARGET']).values[:, :]
        self.X = scaler.fit_transform(self.X)
        self.y = df['TARGET'].values[:]

        self.X = self.X.astype('float32')
        self.y = LabelEncoder().fit_transform(self.y)
        self.y = self.y.astype('float32')
        self.y = self.y.reshape((len(self.y), 1))

    def __len__(self):
        return len(self.X)

    def __getitem__(self, idx):
        return [self.X[idx], self.y[idx]]

    #splitting of data is referred from https://clay-atlas.com/us/blog/2021/08/25/pytorch-
    def get_splits(self, n_test=0.05):
        testLen = round(n_test * len(self.X))
        trainLen = len(self.X) - testLen
        return random_split(self, [trainLen, testLen])
```

```
#Defination of model
class MultiLayer(Module):
    # define model elements
    def __init__(self, n_inputs):
        #How the inbuilt forward method works
        #https://pytorch.org/docs/stable/generated/torch.nn.Module.html
        #https://discuss.pytorch.org/t/why-the-forward-function-is-never-be-called/109498
        super(MultiLayer, self).__init__()
        # Hidden Layer 1
        #Initializing weights by using kaiming_uniform_ and xavier_uniform_ is referenced
        #https://pytorch.org/docs/stable/_modules/torch/nn/init.html#kaiming_uniform
        self.hidden1 = Linear(n_inputs, 35)
        kaiming_uniform_(self.hidden1.weight,nonlinearity='relu')
        self.act1 = ReLU()
        # Hidden Layer 2
        self.hidden2 = Linear(35, 25)
        kaiming_uniform_(self.hidden2.weight,nonlinearity='relu')
        self.act2 = ReLU()

        self.hidden3 = Linear(25, 15)
        kaiming_uniform_(self.hidden3.weight)
        self.act3 = ReLU()
         # Hidden Layer 3
        self.hidden4 = Linear(15, 5)
        xavier_uniform_(self.hidden4.weight)
        self.act4 = Sigmoid()
         # Hidden Layer 4
        self.hidden5 = Linear(5, 1)
        xavier_uniform_(self.hidden5.weight)
        self.act5 = Sigmoid()

    def forward(self, X):
        # Hidden Layer 1
        X = self.hidden1(X)
        X = self.act1(X)
         # Hidden Layer 2
        X = self.hidden2(X)
        X = self.act2(X)
         # Hidden Layer 3
        X = self.hidden3(X)
        X = self.act3(X)
         # Hidden Layer 4
        X = self.hidden4(X)
        X = self.act4(X)
         # Hidden Layer 5
        X = self.hidden5(X)
        X = self.act5(X)
        return X

# train the model
def modelTrain(train_dl, model):
    # define the optimization
    #for BCE Loss - https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html
    criterion = BCELoss()
    optimizer = SGD(model.parameters(), lr=0.01, momentum=0.9)
    # enumerate epochs
    start = time()
    totalEpochs = 150
    for epoch in range(totalEpochs):
        print('Running epoch - > ', epoch , ' out of total - > ', totalEpochs , ' epochs')
        # enumerate mini batches
        for i, (inputs, targets) in enumerate(train_dl):
            # clear the gradients
            #How optimizer works
            #https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html
```

```
            optimizer.zero_grad()
            # compute the model output
            yhat = model(inputs)
            # calculate loss
            loss = criterion(yhat, targets)
            # plotting on tensorboard
            writer.add_scalar("Loss/train", loss, epoch)
            # credit assignment
            loss.backward()
            # update model weights
            optimizer.step()
    train_time = np.round(time() - start,4)
    return train_time

def MultiLayerModel(testData, model):
    predictions, actuals = list(), list()
    for i, (inputs, targets) in enumerate(testData):
        # evaluate the model on the test set
        y_hat = model(inputs)
        # retrieve numpy array
        y_hat = y_hat.detach().numpy()
        actual = targets.numpy()
        actual = actual.reshape((len(actual), 1))
        # round to class values
        y_hat = y_hat.round()
        # store
        predictions.append(y_hat)
        actuals.append(actual)
        return [predictions,actuals]

# evaluate the model
def modelEvaluation(testData, model):
    x=MultiLayerModel(testData,model)
    predictions,actuals=x[0],x[1]
    predictions, actuals = vstack(predictions), vstack(actuals)
    # calculate accuracy
    acc = accuracy_score(actuals, predictions)
    return acc

# make a class prediction for one row of data
def modelPrediction(testData, model):
    start = time()
    result_df = pd.DataFrame()
    x=MultiLayerModel(testData,model)
    predictions,actuals=x[0],x[1]
    predictions = predictions[0].reshape(len(predictions[0])).tolist()
    actuals = actuals[0].reshape(len(actuals[0])).tolist()
    result_df['pred'] = predictions
    result_df['actual'] = actuals
    test_time = np.round(time() - start,4)
    return result_df, test_time

# prepare the data
path ='MLP_input_data.csv'

dataset = MLPdataset(path)
train, test = dataset.get_splits()
trainDataLen = DataLoader(train, batch_size=32, shuffle=True)
testDataLen = DataLoader(test, batch_size=1024, shuffle=False)

#print(len(trainDataLen.dataset), len(testDataLen.dataset))
# Network Defination
model = MultiLayer(74)
#Model Train
train_time = modelTrain(trainDataLen, model)
output_df, test_time = modelPrediction(testDataLen, model)
```

```
# Model Evaluation
Accuracy = modelEvaluation(testDataLen, model)
print('Accuracy: %.3f' % Accuracy)
AUC = roc_auc_score(output_df['actual'],output_df['pred'])
print('AUC: %.3f' % AUC)


result_df = pd.DataFrame()
result_df = result_df.append(pd.Series(["Multi Layer Perceptron", train_time, test_time,
result_df.columns = expLog.columns
expLog = expLog.append(result_df,ignore_index=True)

#writer.flush()
#writer.close()
```

```
Running epoch - >  0  out of total - >  150  epochs
Running epoch - >  1  out of total - >  150  epochs
Running epoch - >  2  out of total - >  150  epochs
Running epoch - >  3  out of total - >  150  epochs
Running epoch - >  4  out of total - >  150  epochs
Running epoch - >  5  out of total - >  150  epochs
Running epoch - >  6  out of total - >  150  epochs
Running epoch - >  7  out of total - >  150  epochs
Running epoch - >  8  out of total - >  150  epochs
Running epoch - >  9  out of total - >  150  epochs
Running epoch - >  10  out of total - >  150  epochs
Running epoch - >  11  out of total - >  150  epochs
Running epoch - >  12  out of total - >  150  epochs
Running epoch - >  13  out of total - >  150  epochs
Running epoch - >  14  out of total - >  150  epochs
Running epoch - >  15  out of total - >  150  epochs
Running epoch - >  16  out of total - >  150  epochs
Running epoch - >  17  out of total - >  150  epochs
Running epoch - >  18  out of total - >  150  epochs
Running epoch - >  19  out of total - >  150  epochs
Running epoch - >  20  out of total - >  150  epochs
Running epoch - >  21  out of total - >  150  epochs
Running epoch - >  22  out of total - >  150  epochs
Running epoch - >  23  out of total - >  150  epochs
Running epoch - >  24  out of total - >  150  epochs
Running epoch - >  25  out of total - >  150  epochs
Running epoch - >  26  out of total - >  150  epochs
Running epoch - >  27  out of total - >  150  epochs
Running epoch - >  28  out of total - >  150  epochs
Running epoch - >  29  out of total - >  150  epochs
Running epoch - >  30  out of total - >  150  epochs
Running epoch - >  31  out of total - >  150  epochs
Running epoch - >  32  out of total - >  150  epochs
Running epoch - >  33  out of total - >  150  epochs
Running epoch - >  34  out of total - >  150  epochs
Running epoch - >  35  out of total - >  150  epochs
Running epoch - >  36  out of total - >  150  epochs
Running epoch - >  37  out of total - >  150  epochs
Running epoch - >  38  out of total - >  150  epochs
Running epoch - >  39  out of total - >  150  epochs
Running epoch - >  40  out of total - >  150  epochs
Running epoch - >  41  out of total - >  150  epochs
Running epoch - >  42  out of total - >  150  epochs
Running epoch - >  43  out of total - >  150  epochs
Running epoch - >  44  out of total - >  150  epochs
Running epoch - >  45  out of total - >  150  epochs
Running epoch - >  46  out of total - >  150  epochs
Running epoch - >  47  out of total - >  150  epochs
Running epoch - >  48  out of total - >  150  epochs
Running epoch - >  49  out of total - >  150  epochs
```

```
Running epoch - >  50  out of total - >  150  epochs
Running epoch - >  51  out of total - >  150  epochs
Running epoch - >  52  out of total - >  150  epochs
Running epoch - >  53  out of total - >  150  epochs
Running epoch - >  54  out of total - >  150  epochs
Running epoch - >  55  out of total - >  150  epochs
Running epoch - >  56  out of total - >  150  epochs
Running epoch - >  57  out of total - >  150  epochs
Running epoch - >  58  out of total - >  150  epochs
Running epoch - >  59  out of total - >  150  epochs
Running epoch - >  60  out of total - >  150  epochs
Running epoch - >  61  out of total - >  150  epochs
Running epoch - >  62  out of total - >  150  epochs
Running epoch - >  63  out of total - >  150  epochs
Running epoch - >  64  out of total - >  150  epochs
Running epoch - >  65  out of total - >  150  epochs
Running epoch - >  66  out of total - >  150  epochs
Running epoch - >  67  out of total - >  150  epochs
Running epoch - >  68  out of total - >  150  epochs
Running epoch - >  69  out of total - >  150  epochs
Running epoch - >  70  out of total - >  150  epochs
Running epoch - >  71  out of total - >  150  epochs
Running epoch - >  72  out of total - >  150  epochs
Running epoch - >  73  out of total - >  150  epochs
Running epoch - >  74  out of total - >  150  epochs
Running epoch - >  75  out of total - >  150  epochs
Running epoch - >  76  out of total - >  150  epochs
Running epoch - >  77  out of total - >  150  epochs
Running epoch - >  78  out of total - >  150  epochs
Running epoch - >  79  out of total - >  150  epochs
Running epoch - >  80  out of total - >  150  epochs
Running epoch - >  81  out of total - >  150  epochs
Running epoch - >  82  out of total - >  150  epochs
Running epoch - >  83  out of total - >  150  epochs
Running epoch - >  84  out of total - >  150  epochs
Running epoch - >  85  out of total - >  150  epochs
Running epoch - >  86  out of total - >  150  epochs
Running epoch - >  87  out of total - >  150  epochs
Running epoch - >  88  out of total - >  150  epochs
Running epoch - >  89  out of total - >  150  epochs
Running epoch - >  90  out of total - >  150  epochs
Running epoch - >  91  out of total - >  150  epochs
Running epoch - >  92  out of total - >  150  epochs
Running epoch - >  93  out of total - >  150  epochs
Running epoch - >  94  out of total - >  150  epochs
Running epoch - >  95  out of total - >  150  epochs
Running epoch - >  96  out of total - >  150  epochs
Running epoch - >  97  out of total - >  150  epochs
Running epoch - >  98  out of total - >  150  epochs
Running epoch - >  99  out of total - >  150  epochs
Running epoch - >  100  out of total - >  150  epochs
Running epoch - >  101  out of total - >  150  epochs
Running epoch - >  102  out of total - >  150  epochs
Running epoch - >  103  out of total - >  150  epochs
Running epoch - >  104  out of total - >  150  epochs
Running epoch - >  105  out of total - >  150  epochs
Running epoch - >  106  out of total - >  150  epochs
Running epoch - >  107  out of total - >  150  epochs
Running epoch - >  108  out of total - >  150  epochs
Running epoch - >  109  out of total - >  150  epochs
Running epoch - >  110  out of total - >  150  epochs
Running epoch - >  111  out of total - >  150  epochs
Running epoch - >  112  out of total - >  150  epochs
Running epoch - >  113  out of total - >  150  epochs
Running epoch - >  114  out of total - >  150  epochs
Running epoch - >  115  out of total - >  150  epochs
```

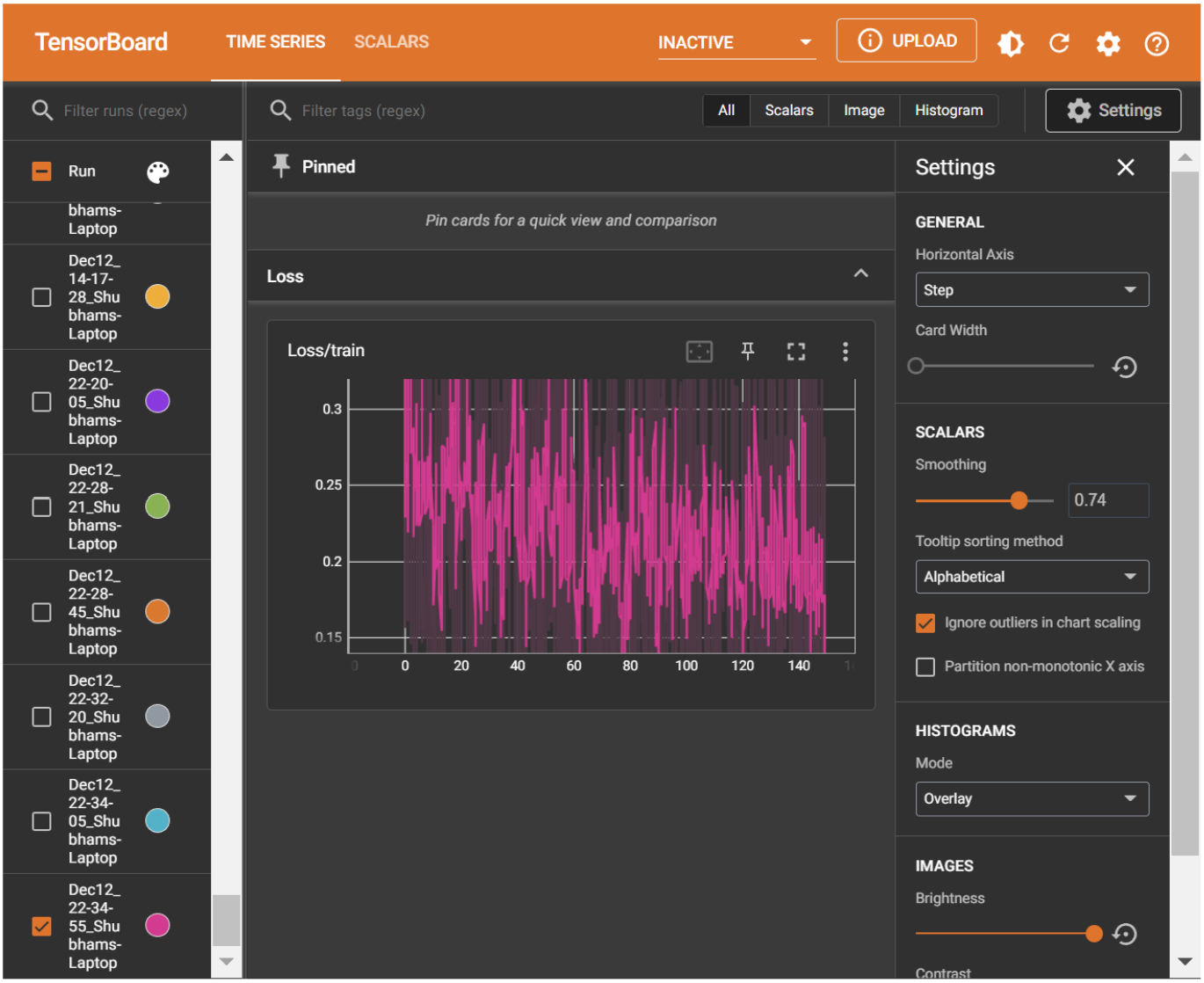
```
Running epoch - >  116  out of total - >  150  epochs
Running epoch - >  117  out of total - >  150  epochs
Running epoch - >  118  out of total - >  150  epochs
Running epoch - >  119  out of total - >  150  epochs
Running epoch - >  120  out of total - >  150  epochs
Running epoch - >  121  out of total - >  150  epochs
Running epoch - >  122  out of total - >  150  epochs
Running epoch - >  123  out of total - >  150  epochs
Running epoch - >  124  out of total - >  150  epochs
Running epoch - >  125  out of total - >  150  epochs
Running epoch - >  126  out of total - >  150  epochs
Running epoch - >  127  out of total - >  150  epochs
Running epoch - >  128  out of total - >  150  epochs
Running epoch - >  129  out of total - >  150  epochs
Running epoch - >  130  out of total - >  150  epochs
Running epoch - >  131  out of total - >  150  epochs
Running epoch - >  132  out of total - >  150  epochs
Running epoch - >  133  out of total - >  150  epochs
Running epoch - >  134  out of total - >  150  epochs
Running epoch - >  135  out of total - >  150  epochs
Running epoch - >  136  out of total - >  150  epochs
Running epoch - >  137  out of total - >  150  epochs
Running epoch - >  138  out of total - >  150  epochs
Running epoch - >  139  out of total - >  150  epochs
Running epoch - >  140  out of total - >  150  epochs
Running epoch - >  141  out of total - >  150  epochs
Running epoch - >  142  out of total - >  150  epochs
Running epoch - >  143  out of total - >  150  epochs
Running epoch - >  144  out of total - >  150  epochs
Running epoch - >  145  out of total - >  150  epochs
Running epoch - >  146  out of total - >  150  epochs
Running epoch - >  147  out of total - >  150  epochs
Running epoch - >  148  out of total - >  150  epochs
Running epoch - >  149  out of total - >  150  epochs
Accuracy: 0.915
AUC: 0.601
```

In [48]:

```
%load_ext tensorboard
%tensorboard --logdir runs
```

```
Reusing TensorBoard on port 6006 (pid 1476), started 2 days, 12:06:19 ago. (Use '!kill 147
6' to kill it.)
```

Note: We receivd error as port already in use so we rann the algorithm on different machine to generate the tensor board report. Attaching the results below.



In [49]:
```
expLog
```

Out[49]:

	ExpID	Train Time	Test Time	Accuracy	AUC	Comments
0	Multi Layer Perceptron	363.661	0.0234	0.915039	0.601393	150 - ReLU + Sigmoid

In [50]:
```
#The MLP Execution is referenced from Professors HW 11 Notebook
#Along with that we referred lot of articles from pytorch official documentation to implem

writer = SummaryWriter()
scaler = MinMaxScaler()
#Define Datasets

class MLPdataset(Dataset):
    # load the dataset
    def __init__(self, path):
        df = read_csv(path).head(30000)
        numAttributes = []
        catAttributes = []

        for c in df.columns.tolist():
            if df[c].dtype in (['int','float']):
                numAttributes.append(c)
            else:
                catAttributes.append(c)
        lableEncD = {}
```

```
        for c in df.columns.tolist():
            if df[c].dtype == 'object':
                lableEnc = LabelEncoder()
                df[c] = df[col].fillna("NULL")
                df[c] = lableEnc.fit_transform(df[c])
                lableEncD['lableEnc{}'.format(c)] = lableEnc

        # Input Output storing
        self.X = df.drop(columns=['TARGET']).values[:, :]
        self.X = scaler.fit_transform(self.X)
        self.y = df['TARGET'].values[:]

        self.X = self.X.astype('float32')
        self.y = LabelEncoder().fit_transform(self.y)
        self.y = self.y.astype('float32')
        self.y = self.y.reshape((len(self.y), 1))

    def __len__(self):
        return len(self.X)

    def __getitem__(self, idx):
        return [self.X[idx], self.y[idx]]

    #splitting of data is referred from https://clay-atlas.com/us/blog/2021/08/25/pytorch-
    def get_splits(self, n_test=0.05):
        testLen = round(n_test * len(self.X))
        trainLen = len(self.X) - testLen
        return random_split(self, [trainLen, testLen])

#Defination of model
class MultiLayer(Module):
    # define model elements
    def __init__(self, n_inputs):
        #How the inbuilt forward method works
        #https://pytorch.org/docs/stable/generated/torch.nn.Module.html
        #https://discuss.pytorch.org/t/why-the-forward-function-is-never-be-called/109498
        super(MultiLayer, self).__init__()
        # Hidden Layer 1
        #Initializing weights by using kaiming_uniform_ and xavier_uniform_ is referenced
        #https://pytorch.org/docs/stable/_modules/torch/nn/init.html#kaiming_uniform
        self.hidden1 = Linear(n_inputs, 35)
        kaiming_uniform_(self.hidden1.weight,nonlinearity='relu')
        self.act1 = ReLU()
        # Hidden Layer 2
        self.hidden2 = Linear(35, 25)
        kaiming_uniform_(self.hidden2.weight,nonlinearity='relu')
        self.act2 = ReLU()

        # Hidden Layer 3
        self.hidden3 = Linear(25, 15)
        kaiming_uniform_(self.hidden3.weight)
        self.act3 = ReLU()
         # Hidden Layer 4
        self.hidden4 = Linear(15, 5)
        xavier_uniform_(self.hidden4.weight)
        self.act4 = ReLU()
         # Hidden Layer 5
        self.hidden5 = Linear(5, 1)
        xavier_uniform_(self.hidden5.weight)
        self.act5 = ReLU()

    def forward(self, X):
        # Hidden Layer 1
        X = self.hidden1(X)
        X = self.act1(X)
         # Hidden Layer 2
```

```
        X = self.hidden2(X)
        X = self.act2(X)
         # Hidden Layer 3
        X = self.hidden3(X)
        X = self.act3(X)
        X = self.hidden4(X)
        X = self.act4(X)
        X = self.hidden5(X)
        X = self.act5(X)
        return X

# train the model
def modelTrain(train_dl, model):
    # define the optimization
    #for BCE Loss - https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html
    criterion = BCELoss()
    optimizer = SGD(model.parameters(), lr=0.01, momentum=0.9)
    # enumerate epochs
    start = time()
    totalEpochs = 150
    for epoch in range(totalEpochs):
        print('Running epoch - > ', epoch , ' out of total - > ', totalEpochs , ' epochs')
        # enumerate mini batches
        for i, (inputs, targets) in enumerate(train_dl):
            # clear the gradients
            #How optimizer works
            #https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html
            optimizer.zero_grad()
            # compute the model output
            yhat = model(inputs)
            # calculate loss
            loss = criterion(yhat, targets)
            # plotting on tensorboard
            writer.add_scalar("Loss/train", loss, epoch)
            # credit assignment
            loss.backward()
            # update model weights
            optimizer.step()
    train_time = np.round(time() - start,4)
    return train_time
# evaluate the model

def MultiLayerModel(testData, model):
    predictions, actuals = list(), list()
    for i, (inputs, targets) in enumerate(testData):
        # evaluate the model on the test set
        y_hat = model(inputs)
        # retrieve numpy array
        y_hat = y_hat.detach().numpy()
        actual = targets.numpy()
        actual = actual.reshape((len(actual), 1))
        # round to class values
        y_hat = y_hat.round()
        # store
        predictions.append(y_hat)
        actuals.append(actual)
        return [predictions,actuals]

def modelEvaluation(testData, model):
    x=MultiLayerModel(testData,model)
    predictions,actuals=x[0],x[1]
    predictions, actuals = vstack(predictions), vstack(actuals)
    # calculate accuracy
    acc = accuracy_score(actuals, predictions)
    return acc
```

```
# make a class prediction for one row of data
def modelPrediction(testData, model):
    start = time()
    result_df = pd.DataFrame()
    x=MultiLayerModel(testData,model)
    predictions,actuals=x[0],x[1]
    predictions = predictions[0].reshape(len(predictions[0])).tolist()
    actuals = actuals[0].reshape(len(actuals[0])).tolist()
    result_df['pred'] = predictions
    result_df['actual'] = actuals
    test_time = np.round(time() - start,4)
    return result_df, test_time

# prepare the data
path ='MLP_input_data.csv'

dataset = MLPdataset(path)
train, test = dataset.get_splits()
trainDataLen = DataLoader(train, batch_size=32, shuffle=True)
testDataLen = DataLoader(test, batch_size=1024, shuffle=False)

# Network Defination
model = MultiLayer(74)
#Model Train
train_time = modelTrain(trainDataLen, model)
output_df, test_time = modelPrediction(testDataLen, model)
# Model Evaluation
Accuracy = modelEvaluation(testDataLen, model)
print('Accuracy: %.3f' % Accuracy)
AUC = roc_auc_score(output_df['actual'],output_df['pred'])
print('AUC: %.3f' % AUC)


result_df = pd.DataFrame()
result_df = result_df.append(pd.Series(["Multi Layer Perceptron", train_time, test_time,
result_df.columns = expLog.columns
expLog = expLog.append(result_df,ignore_index=True)

#writer.flush()
#writer.close()
```

```
Running epoch - >  0  out of total - >  150  epochs
Running epoch - >  1  out of total - >  150  epochs
Running epoch - >  2  out of total - >  150  epochs
Running epoch - >  3  out of total - >  150  epochs
Running epoch - >  4  out of total - >  150  epochs
Running epoch - >  5  out of total - >  150  epochs
Running epoch - >  6  out of total - >  150  epochs
Running epoch - >  7  out of total - >  150  epochs
Running epoch - >  8  out of total - >  150  epochs
Running epoch - >  9  out of total - >  150  epochs
Running epoch - >  10  out of total - >  150  epochs
Running epoch - >  11  out of total - >  150  epochs
Running epoch - >  12  out of total - >  150  epochs
Running epoch - >  13  out of total - >  150  epochs
Running epoch - >  14  out of total - >  150  epochs
Running epoch - >  15  out of total - >  150  epochs
Running epoch - >  16  out of total - >  150  epochs
Running epoch - >  17  out of total - >  150  epochs
Running epoch - >  18  out of total - >  150  epochs
Running epoch - >  19  out of total - >  150  epochs
Running epoch - >  20  out of total - >  150  epochs
Running epoch - >  21  out of total - >  150  epochs
Running epoch - >  22  out of total - >  150  epochs
Running epoch - >  23  out of total - >  150  epochs
```

```
Running epoch - >  24  out of total - >  150  epochs
Running epoch - >  25  out of total - >  150  epochs
Running epoch - >  26  out of total - >  150  epochs
Running epoch - >  27  out of total - >  150  epochs
Running epoch - >  28  out of total - >  150  epochs
Running epoch - >  29  out of total - >  150  epochs
Running epoch - >  30  out of total - >  150  epochs
Running epoch - >  31  out of total - >  150  epochs
Running epoch - >  32  out of total - >  150  epochs
Running epoch - >  33  out of total - >  150  epochs
Running epoch - >  34  out of total - >  150  epochs
Running epoch - >  35  out of total - >  150  epochs
Running epoch - >  36  out of total - >  150  epochs
Running epoch - >  37  out of total - >  150  epochs
Running epoch - >  38  out of total - >  150  epochs
Running epoch - >  39  out of total - >  150  epochs
Running epoch - >  40  out of total - >  150  epochs
Running epoch - >  41  out of total - >  150  epochs
Running epoch - >  42  out of total - >  150  epochs
Running epoch - >  43  out of total - >  150  epochs
Running epoch - >  44  out of total - >  150  epochs
Running epoch - >  45  out of total - >  150  epochs
Running epoch - >  46  out of total - >  150  epochs
Running epoch - >  47  out of total - >  150  epochs
Running epoch - >  48  out of total - >  150  epochs
Running epoch - >  49  out of total - >  150  epochs
Running epoch - >  50  out of total - >  150  epochs
Running epoch - >  51  out of total - >  150  epochs
Running epoch - >  52  out of total - >  150  epochs
Running epoch - >  53  out of total - >  150  epochs
Running epoch - >  54  out of total - >  150  epochs
Running epoch - >  55  out of total - >  150  epochs
Running epoch - >  56  out of total - >  150  epochs
Running epoch - >  57  out of total - >  150  epochs
Running epoch - >  58  out of total - >  150  epochs
Running epoch - >  59  out of total - >  150  epochs
Running epoch - >  60  out of total - >  150  epochs
Running epoch - >  61  out of total - >  150  epochs
Running epoch - >  62  out of total - >  150  epochs
Running epoch - >  63  out of total - >  150  epochs
Running epoch - >  64  out of total - >  150  epochs
Running epoch - >  65  out of total - >  150  epochs
Running epoch - >  66  out of total - >  150  epochs
Running epoch - >  67  out of total - >  150  epochs
Running epoch - >  68  out of total - >  150  epochs
Running epoch - >  69  out of total - >  150  epochs
Running epoch - >  70  out of total - >  150  epochs
Running epoch - >  71  out of total - >  150  epochs
Running epoch - >  72  out of total - >  150  epochs
Running epoch - >  73  out of total - >  150  epochs
Running epoch - >  74  out of total - >  150  epochs
Running epoch - >  75  out of total - >  150  epochs
Running epoch - >  76  out of total - >  150  epochs
Running epoch - >  77  out of total - >  150  epochs
Running epoch - >  78  out of total - >  150  epochs
Running epoch - >  79  out of total - >  150  epochs
Running epoch - >  80  out of total - >  150  epochs
Running epoch - >  81  out of total - >  150  epochs
Running epoch - >  82  out of total - >  150  epochs
Running epoch - >  83  out of total - >  150  epochs
Running epoch - >  84  out of total - >  150  epochs
Running epoch - >  85  out of total - >  150  epochs
Running epoch - >  86  out of total - >  150  epochs
Running epoch - >  87  out of total - >  150  epochs
Running epoch - >  88  out of total - >  150  epochs
Running epoch - >  89  out of total - >  150  epochs
```

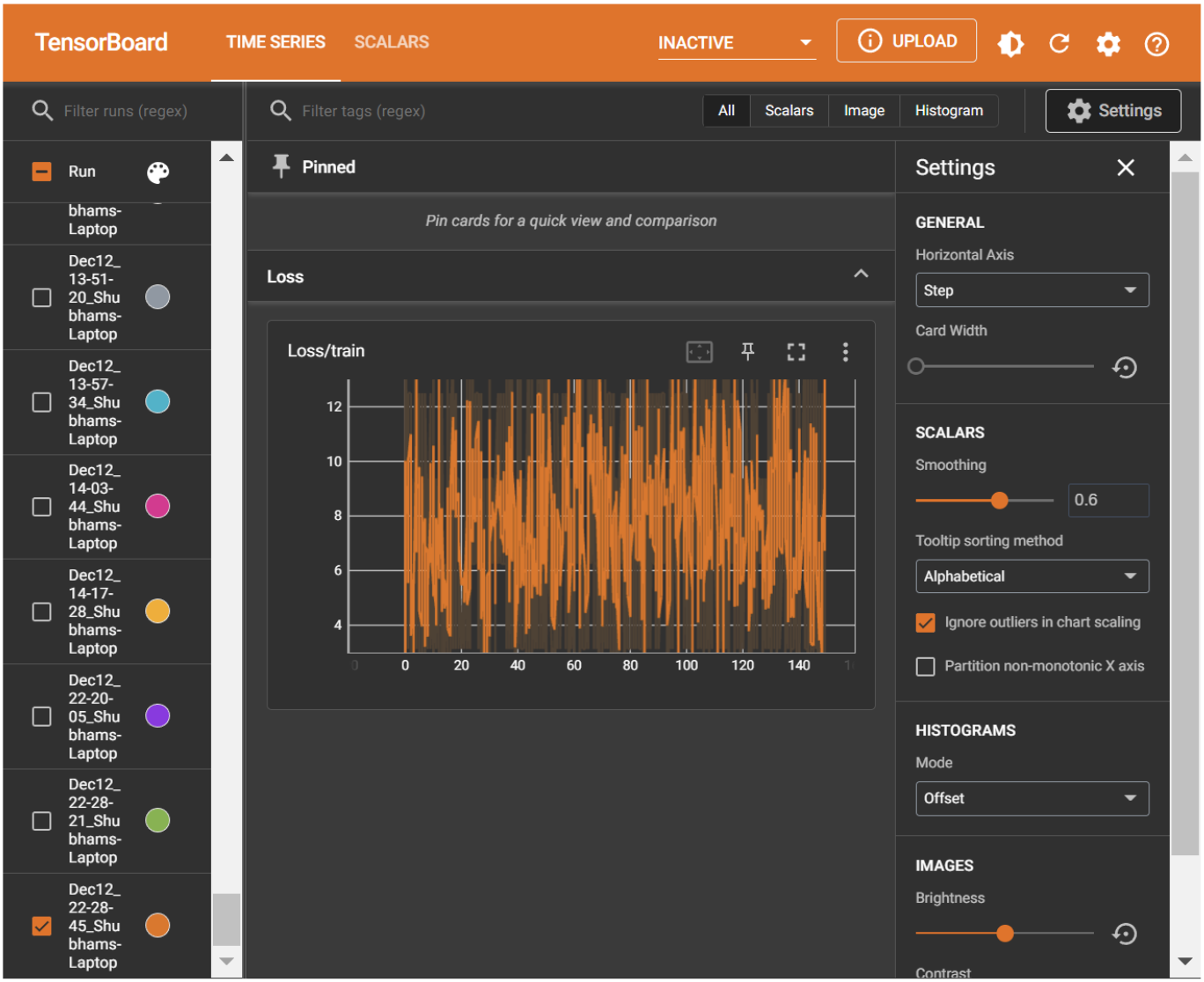
```
Running epoch - >  90  out of total - >  150  epochs
Running epoch - >  91  out of total - >  150  epochs
Running epoch - >  92  out of total - >  150  epochs
Running epoch - >  93  out of total - >  150  epochs
Running epoch - >  94  out of total - >  150  epochs
Running epoch - >  95  out of total - >  150  epochs
Running epoch - >  96  out of total - >  150  epochs
Running epoch - >  97  out of total - >  150  epochs
Running epoch - >  98  out of total - >  150  epochs
Running epoch - >  99  out of total - >  150  epochs
Running epoch - >  100  out of total - >  150  epochs
Running epoch - >  101  out of total - >  150  epochs
Running epoch - >  102  out of total - >  150  epochs
Running epoch - >  103  out of total - >  150  epochs
Running epoch - >  104  out of total - >  150  epochs
Running epoch - >  105  out of total - >  150  epochs
Running epoch - >  106  out of total - >  150  epochs
Running epoch - >  107  out of total - >  150  epochs
Running epoch - >  108  out of total - >  150  epochs
Running epoch - >  109  out of total - >  150  epochs
Running epoch - >  110  out of total - >  150  epochs
Running epoch - >  111  out of total - >  150  epochs
Running epoch - >  112  out of total - >  150  epochs
Running epoch - >  113  out of total - >  150  epochs
Running epoch - >  114  out of total - >  150  epochs
Running epoch - >  115  out of total - >  150  epochs
Running epoch - >  116  out of total - >  150  epochs
Running epoch - >  117  out of total - >  150  epochs
Running epoch - >  118  out of total - >  150  epochs
Running epoch - >  119  out of total - >  150  epochs
Running epoch - >  120  out of total - >  150  epochs
Running epoch - >  121  out of total - >  150  epochs
Running epoch - >  122  out of total - >  150  epochs
Running epoch - >  123  out of total - >  150  epochs
Running epoch - >  124  out of total - >  150  epochs
Running epoch - >  125  out of total - >  150  epochs
Running epoch - >  126  out of total - >  150  epochs
Running epoch - >  127  out of total - >  150  epochs
Running epoch - >  128  out of total - >  150  epochs
Running epoch - >  129  out of total - >  150  epochs
Running epoch - >  130  out of total - >  150  epochs
Running epoch - >  131  out of total - >  150  epochs
Running epoch - >  132  out of total - >  150  epochs
Running epoch - >  133  out of total - >  150  epochs
Running epoch - >  134  out of total - >  150  epochs
Running epoch - >  135  out of total - >  150  epochs
Running epoch - >  136  out of total - >  150  epochs
Running epoch - >  137  out of total - >  150  epochs
Running epoch - >  138  out of total - >  150  epochs
Running epoch - >  139  out of total - >  150  epochs
Running epoch - >  140  out of total - >  150  epochs
Running epoch - >  141  out of total - >  150  epochs
Running epoch - >  142  out of total - >  150  epochs
Running epoch - >  143  out of total - >  150  epochs
Running epoch - >  144  out of total - >  150  epochs
Running epoch - >  145  out of total - >  150  epochs
Running epoch - >  146  out of total - >  150  epochs
Running epoch - >  147  out of total - >  150  epochs
Running epoch - >  148  out of total - >  150  epochs
Running epoch - >  149  out of total - >  150  epochs
Accuracy: 0.933
AUC: 0.500
```

In [51]:

```
%load_ext tensorboard
%tensorboard --logdir runs
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
Reusing TensorBoard on port 6006 (pid 1476), started 2 days, 12:12:29 ago. (Use '!kill 147
6' to kill it.)
```

Note: We receivd error as port already in use so we rann the algorithm on different machine to generate the tensor board report. Attaching the results below.



In [52]:
```
expLog
```

Out[52]:

	ExpID	Train Time	Test Time	Accuracy	AUC	Comments
0	Multi Layer Perceptron	363.6610	0.0234	0.915039	0.601393	150 - ReLU + Sigmoid
1	Multi Layer Perceptron	366.6223	0.0326	0.932617	0.500000	150 - ReLU

In [53]:
```
#The MLP Execution is referenced from Professors HW 11 Notebook
#Along with that we referred lot of articles from pytorch official documentation to impler

writer = SummaryWriter()
scaler = MinMaxScaler()
#Define Datasets

class MLPdataset(Dataset):
    # load the dataset
    def __init__(self, path):
        df = read_csv(path).head(20000)
        numAttributes = []
        catAttributes = []

        for c in df.columns.tolist():
            if df[c].dtype in (['int','float']):
                numAttributes.append(c)
            else:
```

```
                catAttributes.append(c)
        lableEncD = {}
        for c in df.columns.tolist():
            if df[c].dtype == 'object':
                lableEnc = LabelEncoder()
                df[c] = df[col].fillna("NULL")
                df[c] = lableEnc.fit_transform(df[c])
                lableEncD['lableEnc{}'.format(c)] = lableEnc

        # Input Output storing
        self.X = df.drop(columns=['TARGET']).values[:, :]
        self.X = scaler.fit_transform(self.X)
        self.y = df['TARGET'].values[:]

        self.X = self.X.astype('float32')
        self.y = LabelEncoder().fit_transform(self.y)
        self.y = self.y.astype('float32')
        self.y = self.y.reshape((len(self.y), 1))

    def __len__(self):
        return len(self.X)

    def __getitem__(self, idx):
        return [self.X[idx], self.y[idx]]

    #splitting of data is referred from https://clay-atlas.com/us/blog/2021/08/25/pytorch-
    def get_splits(self, n_test=0.05):
        testLen = round(n_test * len(self.X))
        trainLen = len(self.X) - testLen
        return random_split(self, [trainLen, testLen])

#Defination of model
class MultiLayer(Module):
    # define model elements
    def __init__(self, n_inputs):
        #How the inbuilt forward method works
        #https://pytorch.org/docs/stable/generated/torch.nn.Module.html
        #https://discuss.pytorch.org/t/why-the-forward-function-is-never-be-called/109498
        super(MultiLayer, self).__init__()
        # Hidden Layer 1
        #Initializing weights by using kaiming_uniform_ and xavier_uniform_ is referenced
        #https://pytorch.org/docs/stable/_modules/torch/nn/init.html#kaiming_uniform
        self.hidden1 = Linear(n_inputs, 35)
        kaiming_uniform_(self.hidden1.weight)
        self.act1 = Sigmoid()
        # Hidden Layer 2
        self.hidden2 = Linear(35, 25)
        kaiming_uniform_(self.hidden2.weight)
        self.act2 = Sigmoid()

        self.hidden3 = Linear(25, 15)
        kaiming_uniform_(self.hidden3.weight)
        self.act3 = Sigmoid()
         # Hidden Layer 3
        self.hidden4 = Linear(15, 5)
        xavier_uniform_(self.hidden4.weight)
        self.act4 = Sigmoid()
         # Hidden Layer 4
        self.hidden5 = Linear(5, 1)
        xavier_uniform_(self.hidden5.weight)
        self.act5 = Sigmoid()

    def forward(self, X):
        # Hidden Layer 1
        X = self.hidden1(X)
        X = self.act1(X)
```

```
        # Hidden Layer 2
        X = self.hidden2(X)
        X = self.act2(X)
        # Hidden Layer 3
        X = self.hidden3(X)
        X = self.act3(X)
        X = self.hidden4(X)
        X = self.act4(X)
        X = self.hidden5(X)
        X = self.act5(X)
        return X

# prepare the dataset

# train the model
def modelTrain(train_dl, model):
    # define the optimization
    #for BCE Loss - https://pytorch.org/docs/stable/generated/torch.nn.BCELoss.html
    criterion = BCELoss()
    optimizer = SGD(model.parameters(), lr=0.01, momentum=0.9)
    # enumerate epochs
    start = time()
    totalEpochs = 100
    for epoch in range(totalEpochs):
        print('Running epoch - > ', epoch , ' out of total - > ', totalEpochs , ' epochs')
        # enumerate mini batches
        for i, (inputs, targets) in enumerate(train_dl):
            # clear the gradients
            #How optimizer works
            #https://pytorch.org/tutorials/beginner/blitz/cifar10_tutorial.html
            optimizer.zero_grad()
            # compute the model output
            yhat = model(inputs)
            # calculate loss
            loss = criterion(yhat, targets)
            # plotting on tensorboard
            writer.add_scalar("Loss/train", loss, epoch)
            # credit assignment
            loss.backward()
            # update model weights
            optimizer.step()
    train_time = np.round(time() - start,4)
    return train_time
# evaluate the model

def MultiLayerModel(testData, model):
    predictions, actuals = list(), list()
    for i, (inputs, targets) in enumerate(testData):
        # evaluate the model on the test set
        y_hat = model(inputs)
        # retrieve numpy array
        y_hat = y_hat.detach().numpy()
        actual = targets.numpy()
        actual = actual.reshape((len(actual), 1))
        # round to class values
        y_hat = y_hat.round()
        # store
        predictions.append(y_hat)
        actuals.append(actual)
        return [predictions,actuals]

def modelEvaluation(testData, model):
    x=MultiLayerModel(testData,model)
    predictions,actuals=x[0],x[1]
    predictions, actuals = vstack(predictions), vstack(actuals)
    # calculate accuracy
```

```
    acc = accuracy_score(actuals, predictions)
    return acc

# make a class prediction for one row of data
def modelPrediction(testData, model):
    start = time()
    result_df = pd.DataFrame()
    x=MultiLayerModel(testData,model)
    predictions,actuals=x[0],x[1]
    predictions = predictions[0].reshape(len(predictions[0])).tolist()
    actuals = actuals[0].reshape(len(actuals[0])).tolist()
    result_df['pred'] = predictions
    result_df['actual'] = actuals
    test_time = np.round(time() - start,4)
    return result_df, test_time

# prepare the data
path ='MLP_input_data.csv'

dataset = MLPdataset(path)
train, test = dataset.get_splits()

trainDataLen = DataLoader(train, batch_size=32, shuffle=True)

testDataLen = DataLoader(test, batch_size=1024, shuffle=False)

# Network Defination
model = MultiLayer(74)
#Model Train
train_time = modelTrain(trainDataLen, model)
output_df, test_time = modelPrediction(testDataLen, model)
# Model Evaluation
Accuracy = modelEvaluation(testDataLen, model)
print('Accuracy: %.3f' % Accuracy)
AUC = roc_auc_score(output_df['actual'],output_df['pred'])
print('AUC: %.3f' % AUC)


result_df = pd.DataFrame()
result_df = result_df.append(pd.Series(["Multi Layer Perceptron", train_time, test_time,
result_df.columns = expLog.columns
expLog = expLog.append(result_df,ignore_index=True)

#writer.flush()
#writer.close()
```

```
Running epoch - >  0  out of total - >  100  epochs
Running epoch - >  1  out of total - >  100  epochs
Running epoch - >  2  out of total - >  100  epochs
Running epoch - >  3  out of total - >  100  epochs
Running epoch - >  4  out of total - >  100  epochs
Running epoch - >  5  out of total - >  100  epochs
Running epoch - >  6  out of total - >  100  epochs
Running epoch - >  7  out of total - >  100  epochs
Running epoch - >  8  out of total - >  100  epochs
Running epoch - >  9  out of total - >  100  epochs
Running epoch - >  10  out of total - >  100  epochs
Running epoch - >  11  out of total - >  100  epochs
Running epoch - >  12  out of total - >  100  epochs
Running epoch - >  13  out of total - >  100  epochs
Running epoch - >  14  out of total - >  100  epochs
Running epoch - >  15  out of total - >  100  epochs
Running epoch - >  16  out of total - >  100  epochs
Running epoch - >  17  out of total - >  100  epochs
Running epoch - >  18  out of total - >  100  epochs
```

```
Running epoch - >  19  out of total - >  100  epochs
Running epoch - >  20  out of total - >  100  epochs
Running epoch - >  21  out of total - >  100  epochs
Running epoch - >  22  out of total - >  100  epochs
Running epoch - >  23  out of total - >  100  epochs
Running epoch - >  24  out of total - >  100  epochs
Running epoch - >  25  out of total - >  100  epochs
Running epoch - >  26  out of total - >  100  epochs
Running epoch - >  27  out of total - >  100  epochs
Running epoch - >  28  out of total - >  100  epochs
Running epoch - >  29  out of total - >  100  epochs
Running epoch - >  30  out of total - >  100  epochs
Running epoch - >  31  out of total - >  100  epochs
Running epoch - >  32  out of total - >  100  epochs
Running epoch - >  33  out of total - >  100  epochs
Running epoch - >  34  out of total - >  100  epochs
Running epoch - >  35  out of total - >  100  epochs
Running epoch - >  36  out of total - >  100  epochs
Running epoch - >  37  out of total - >  100  epochs
Running epoch - >  38  out of total - >  100  epochs
Running epoch - >  39  out of total - >  100  epochs
Running epoch - >  40  out of total - >  100  epochs
Running epoch - >  41  out of total - >  100  epochs
Running epoch - >  42  out of total - >  100  epochs
Running epoch - >  43  out of total - >  100  epochs
Running epoch - >  44  out of total - >  100  epochs
Running epoch - >  45  out of total - >  100  epochs
Running epoch - >  46  out of total - >  100  epochs
Running epoch - >  47  out of total - >  100  epochs
Running epoch - >  48  out of total - >  100  epochs
Running epoch - >  49  out of total - >  100  epochs
Running epoch - >  50  out of total - >  100  epochs
Running epoch - >  51  out of total - >  100  epochs
Running epoch - >  52  out of total - >  100  epochs
Running epoch - >  53  out of total - >  100  epochs
Running epoch - >  54  out of total - >  100  epochs
Running epoch - >  55  out of total - >  100  epochs
Running epoch - >  56  out of total - >  100  epochs
Running epoch - >  57  out of total - >  100  epochs
Running epoch - >  58  out of total - >  100  epochs
Running epoch - >  59  out of total - >  100  epochs
Running epoch - >  60  out of total - >  100  epochs
Running epoch - >  61  out of total - >  100  epochs
Running epoch - >  62  out of total - >  100  epochs
Running epoch - >  63  out of total - >  100  epochs
Running epoch - >  64  out of total - >  100  epochs
Running epoch - >  65  out of total - >  100  epochs
Running epoch - >  66  out of total - >  100  epochs
Running epoch - >  67  out of total - >  100  epochs
Running epoch - >  68  out of total - >  100  epochs
Running epoch - >  69  out of total - >  100  epochs
Running epoch - >  70  out of total - >  100  epochs
Running epoch - >  71  out of total - >  100  epochs
Running epoch - >  72  out of total - >  100  epochs
Running epoch - >  73  out of total - >  100  epochs
Running epoch - >  74  out of total - >  100  epochs
Running epoch - >  75  out of total - >  100  epochs
Running epoch - >  76  out of total - >  100  epochs
Running epoch - >  77  out of total - >  100  epochs
Running epoch - >  78  out of total - >  100  epochs
Running epoch - >  79  out of total - >  100  epochs
Running epoch - >  80  out of total - >  100  epochs
Running epoch - >  81  out of total - >  100  epochs
Running epoch - >  82  out of total - >  100  epochs
Running epoch - >  83  out of total - >  100  epochs
Running epoch - >  84  out of total - >  100  epochs
```

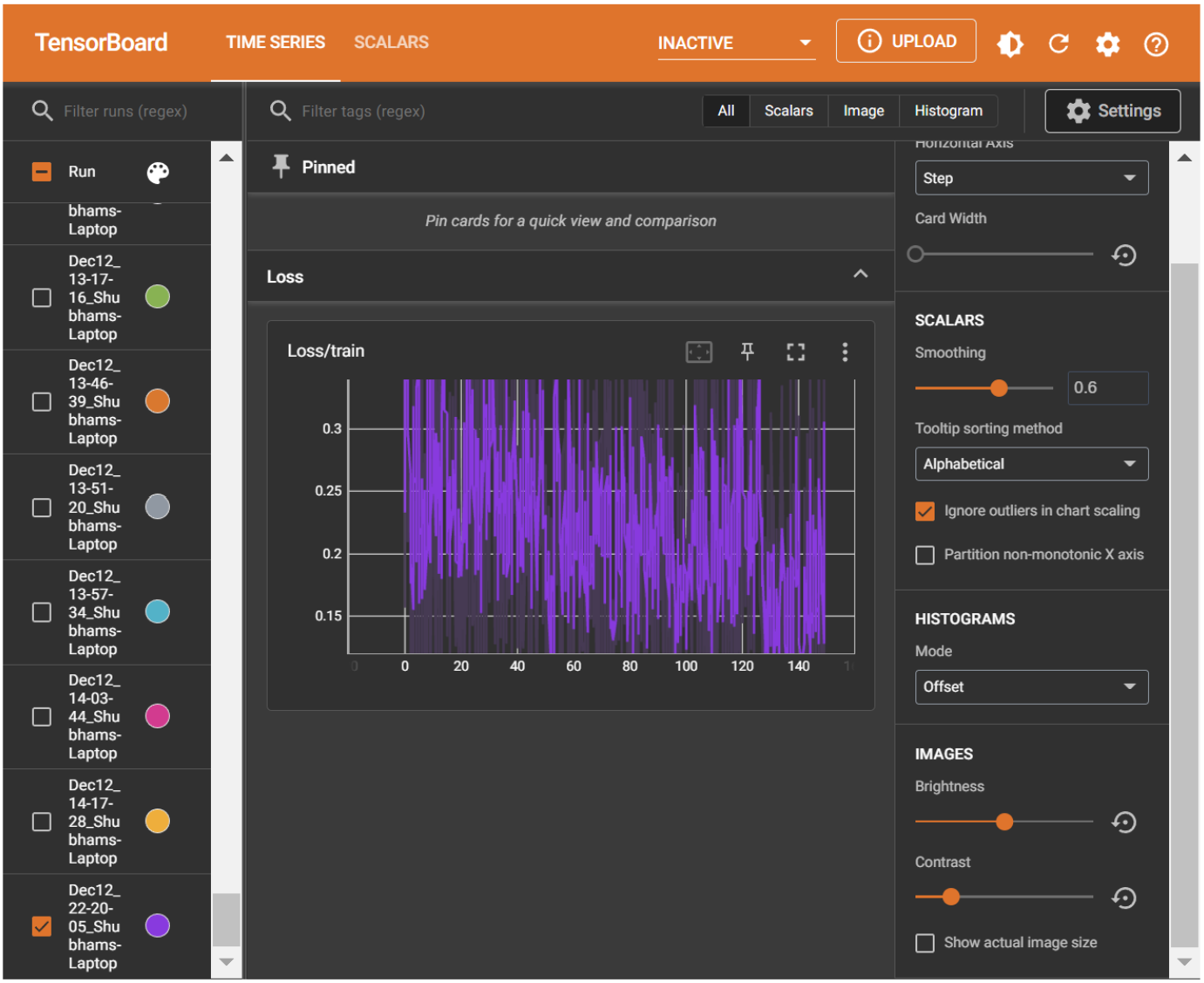
```
Running epoch - >  85  out of total - >  100  epochs
Running epoch - >  86  out of total - >  100  epochs
Running epoch - >  87  out of total - >  100  epochs
Running epoch - >  88  out of total - >  100  epochs
Running epoch - >  89  out of total - >  100  epochs
Running epoch - >  90  out of total - >  100  epochs
Running epoch - >  91  out of total - >  100  epochs
Running epoch - >  92  out of total - >  100  epochs
Running epoch - >  93  out of total - >  100  epochs
Running epoch - >  94  out of total - >  100  epochs
Running epoch - >  95  out of total - >  100  epochs
Running epoch - >  96  out of total - >  100  epochs
Running epoch - >  97  out of total - >  100  epochs
Running epoch - >  98  out of total - >  100  epochs
Running epoch - >  99  out of total - >  100  epochs
Accuracy: 0.920
AUC: 0.528
```

In [54]:

```
%load_ext tensorboard
%tensorboard --logdir runs
```

```
The tensorboard extension is already loaded. To reload it, use:
  %reload_ext tensorboard
Reusing TensorBoard on port 6006 (pid 1476), started 2 days, 12:15:13 ago. (Use '!kill 147
6' to kill it.)
```

Note: We receivd error as port already in use so we rann the algorithm on different machine to generate the tensor board report. Attaching the results below.



In [55]:
```
expLog
```

Out[55]:

	ExpID	Train Time	Test Time	Accuracy	AUC	Comments
0	Multi Layer Perceptron	363.6610	0.0234	0.915039	0.601393	150 - ReLU + Sigmoid
1	Multi Layer Perceptron	366.6223	0.0326	0.932617	0.500000	150 - ReLU
2	Multi Layer Perceptron	160.7423	0.0156	0.920000	0.527774	100 - Sigmoid

In [ ]:
```
```

In [57]:
```
########################################################################
```

In [58]:
```
experimentLog = pd.DataFrame(columns=["ExpID", 'Time', "Accuracy", "Valid Acc", "AUC","Cor
```

In [59]:
```
training_data = datasets['application_train']
testing_data = datasets['application_test']

#Calculate the applicants that got a loan and those which did not get.
ones = training_data[training_data['TARGET']==1]
zeros = training_data[training_data['TARGET']==0]
```

```
ones_len = ones.shape[0]
zeros_len = zeros.shape[0]
n = ones_len + zeros_len

ones_len_proportion = ones_len/n
zeros__len_proportion = zeros_len/n

#subset rows from data
ones_sample = ones.sample(n=int(10000*ones_len_proportion))
zeros_sample = zeros.sample(n=int(10000*zeros__len_proportion))

#finishing the subset
application_train = pd.concat([ones_sample,zeros_sample])

application_train.head()
```

Out[59]:

	SK_ID_CURR	TARGET	NAME_CONTRACT_TYPE	CODE_GENDER	FLAG_OWN_CAR	FLAG_OWN_REALTY	CNT_C
301117	448852	1	Cash loans	M	Y	Y	
172800	300244	1	Cash loans	M	Y	N	
5879	106884	1	Cash loans	M	N	Y	
73579	185324	1	Cash loans	F	N	Y	
192100	322755	1	Cash loans	M	N	N	

5 rows × 122 columns

In [60]:

```
X = application_train.loc[:, application_train.columns != 'TARGET']
y = application_train['TARGET']

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

#Create full set of categorical features
cat_features = get_cat_features(X)

#Numeric Features
num_features = get_num_features(X)
```

In [61]:

```
#pipeline

cat_pipe = Pipeline([
        ('selector', DataFrameSelector(cat_features)),
        ('imputer', SimpleImputer(strategy='most_frequent', fill_value='missing')),
        ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))])

num_pipe = Pipeline([
        ('selector', DataFrameSelector(num_features)),
        ('imputer', SimpleImputer(strategy='mean')),
        ('std_scaler', StandardScaler())])

data_pipe = ColumnTransformer(transformers= [
        ("num_pipeline", num_pipe, num_features),
        ("cat_pipeline", cat_pipe, cat_features)],
         remainder='drop',
         n_jobs=-1)
```

In [62]:

```
#Execution of MLPClassifier is referenced from
#https://scikit-learn.org/stable/modules/generated/sklearn.neural_network.MLPClassifier.h
```

```
##Single Layer perceptron

single_layer_pipe = Pipeline([
        ("data_prep", data_pipe),
        ('SLP', MLPClassifier(random_state=42))])

params_single_layer = {'SLP__hidden_layer_sizes':[1],
          'SLP__alpha':[0.1,0.5],
          'SLP__activation':['identity','tanh','relu']}

scoring='accuracy'
#running Single Layer Perceptron
start = time()
single_layer_gridsearch = GridSearchCV(single_layer_pipe, param_grid = params_single_layer
single_layer_gridsearch.fit(X_train, y_train)
train_time = np.round(time() - start,4)

print("Best parameters after running grid search:")
best_param = single_layer_gridsearch.best_estimator_.get_params()
for param_name in sorted(params_single_layer.keys()):
    print("\t%s: %r" % (param_name, best_param[param_name]))
print("-----------------------------------------------------------------")
print()
print("Best %s score is: %0.3f" %(scoring, single_layer_gridsearch.best_score_))
print("-----------------------------------------------------------------")
print()

pr_train = single_layer_gridsearch.predict(X_train)
print(f"Training accuracy given by the best pipeline is: {accuracy_score(pr_train, y_train
print()
print("-----------------------------------------------------------------")
print()


pr_test = single_layer_gridsearch.predict(X_test)
print(f"Testing accuracy given by the best pipeline is: {accuracy_score(pr_test, y_test):.
print()
print("-----------------------------------------------------------------")
print()

y_test_pred = single_layer_gridsearch.best_estimator_.predict(X_test)


print("Confusion matrix is:")
print(confusion_matrix(y_test_pred, y_test))
print()
print("-----------------------------------------------------------------")
print()

print(f"Overall accuracy is: {np.round(accuracy_score(y_test, y_test_pred), 3)*100}%")
print()
print("-----------------------------------------------------------------")
print()
```

```
Best parameters after running grid search:
        SLP__activation: 'identity'
        SLP__alpha: 0.5
        SLP__hidden_layer_sizes: 1
-----------------------------------------------------------------

Best accuracy score is: 0.924
-----------------------------------------------------------------

Training accuracy given by the best pipeline is: 0.924
```

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

Testing accuracy given by the best pipeline is: 0.910

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

Confusion matrix is:
[[2726  268]
 [   3    3]]

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

Overall accuracy is: 91.0%

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

In [63]:

```
result_df = pd.DataFrame()
result_df = result_df.append(pd.Series(["MLP (Single hiddne layer)", train_time, accuracy_
result_df.columns = experimentLog.columns
experimentLog = experimentLog.append(result_df,ignore_index=True)
experimentLog
```

Out[63]:

	ExpID	Time	Accuracy	Valid Acc	AUC	Comment
0	MLP (Single hiddne layer)	15.8018	0.924275	0.909667	0.778271	MLP (Single hiddne layer)

In [64]:

```
##MultiLayer Perceptron

scoring='accuracy'
multi_layer_pipe = Pipeline([
        ("data_prep", data_pipe),
        ('MLP', MLPClassifier(random_state=42))])

params_multi_layer = {'MLP__hidden_layer_sizes':[5,10],
          'MLP__alpha':[0.1,0.01],
          'MLP__activation':['identity','tanh','relu']}

multi_layer_gridsearch = GridSearchCV(multi_layer_pipe, param_grid = params_multi_layer, c
multi_layer_gridsearch.fit(X_train, y_train)

print("Best parameters after running grid search:")
best_param = multi_layer_gridsearch.best_estimator_.get_params()
for param_name in sorted(params_multi_layer.keys()):
    print("\t%s: %r" % (param_name, best_param[param_name]))
print("----------------------------------------------------------------")
print()
print("Best %s score is: %0.3f" %(scoring, multi_layer_gridsearch.best_score_))
print("----------------------------------------------------------------")
print()
pr_train = multi_layer_gridsearch.predict(X_train)
print(f"Training accuracy given by the best pipeline is: {accuracy_score(pr_train, y_train
print()
print("----------------------------------------------------------------")
print()
pr_test = multi_layer_gridsearch.predict(X_test)
print(f"Testing accuracy given by the best pipeline is: {accuracy_score(pr_test, y_test):.
print()
print("----------------------------------------------------------------")
print()
y_pred = multi_layer_gridsearch.best_estimator_.predict(X_test)
print("Confusion matrix is:")
print(confusion_matrix(y_test_pred, y_test))
```

```
print()
print("----------------------------------------------------------------------")
print()
print(f"Overall accuracy is: {np.round(accuracy_score(y_test, y_pred), 3)*100}%")
print()
print("----------------------------------------------------------------------")
print()
```

```
Best parameters after running grid search:
        MLP__activation: 'identity'
        MLP__alpha: 0.1
        MLP__hidden_layer_sizes: 10
----------------------------------------------------------------------

Best accuracy score is: 0.922
----------------------------------------------------------------------

Training accuracy given by the best pipeline is: 0.925

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

Testing accuracy given by the best pipeline is: 0.909

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

Confusion matrix is:
[[2726  268]
 [   3    3]]

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

Overall accuracy is: 90.9%

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

In [65]:

```
result_df = pd.DataFrame()
result_df = result_df.append(pd.Series(["MLP (multiple hidden layer)", train_time, accurac
result_df.columns = experimentLog.columns
experimentLog = experimentLog.append(result_df,ignore_index=True)
experimentLog
```

Out[65]:

	ExpID	Time	Accuracy	Valid Acc	AUC	Comment
0	MLP (Single hiddne layer)	15.8018	0.924275	0.909667	0.778271	MLP (Single hiddne layer)
1	MLP (multiple hidden layer)	15.8018	0.925275	0.909667	0.787373	MLP (Multiple hiddne layer)

In [66]:

```
y_pred = multi_layer_gridsearch.best_estimator_.predict(testing_data)
print(y_pred)
print(y_pred.shape)
```

```
[0 0 0 ... 0 0 0]
(48744,)
```

Result and Discussion

In Phase 1, we finalized the machine learning models and different project tracking tools allocated tasks to team members built the Phase leader plan the credit assignment plan and Gantt chart. Along with this we also performed some basic data exploration and explored the dataset.

In phase 2 we performed the baseline models of all the selected models and found out the training and testing accuracies.

In Phase 3 we performed feature engineering and hyper parameter tunning and reran the hyperparameter tuned models. We added 2 new models the Lasso Regression and Ridge regression which gave us the improved ROC AUC of 75.57 % than the baseline models.

- Some important finding in phase 3 were:

1. The hyper-tuned decision tree model's train (92.19) and test (92.13) accuracy have greatly risen in comparison to the model's baseline, indicating that it is functioning well on the input dataset.
2. The overall accuracy of the decision tree grew significantly and reached 92%. Hyper tuned Decision Tree is the best-fit algorithm since it surpasses others models in Phase 3.
3. We saw an improvement of 6% in test accuracy and 6% in total accuracy. Decision tree outperforms the other models since its log loss (0.24) is on the lower side and has greatly lowered when compared to its baseline model.

In this Phase, we implemented deep learning, and our model was a multi-layer perceptron. We then generated a visualization of the loss function and accuracy using Tensor Board to visualize our training model. We improved our Kaggle submission from 0.5 to 0.71 which is a 38% improvement. We received a score of 0.71 in our results. Hyperparameter tuning helped improve the Test Accuracy. Our accuracy for the multi-Layer perceptron was out to be 92.4% which is quite efficient and overall good for the given data.
We have performed various experiments and below is the brief description:
We can conclude that our best models are Decision Tree with accuracy on higher side (92.03), ROC score of 53.58% and Lasso Regression with accuracy on a lower side but ROC score on a higher side giving us ROC score of 75.57%. We also built a Deep learning model Multi-Layer Perceptron using PyTorch. The Test Accuracy was on a higher side but we were getting ROC score on the Lower side (60.13%) as compared to the baseline models.

- Following are more details below:

1. Logistic Regression: In Phase 2 we performed, baseline logistic regression with limited feature engineering and it gave us the Train Accuracy of 92 % and test accuracy of 91.9 %. In phase 3 we performed feature engineering and after running it on multiple experiments it gave us the train Accuracy of 91.91% and test accuracy of 91.98%.

2. Random Forest: In Phase 2 we performed, baseline Random regression with limited feature engineering and it gave us the Train Accuracy of 92.2 % and test accuracy of 92.2%. In phase 3 we performed feature engineering and after running it on multiple experiments it gave us the train Accuracy of 91.91% and test accuracy of 91.98%.

3. Decision Tree: In Phase 2 we performed, baseline Decision regression with limited feature engineering and it gave us the Train Accuracy of 86.4 % and test accuracy of 86.1 %. In phase 3 we performed feature engineering and after running it on multiple experiments it gave us the train Accuracy of 91.13% and test accuracy of 92.13%.

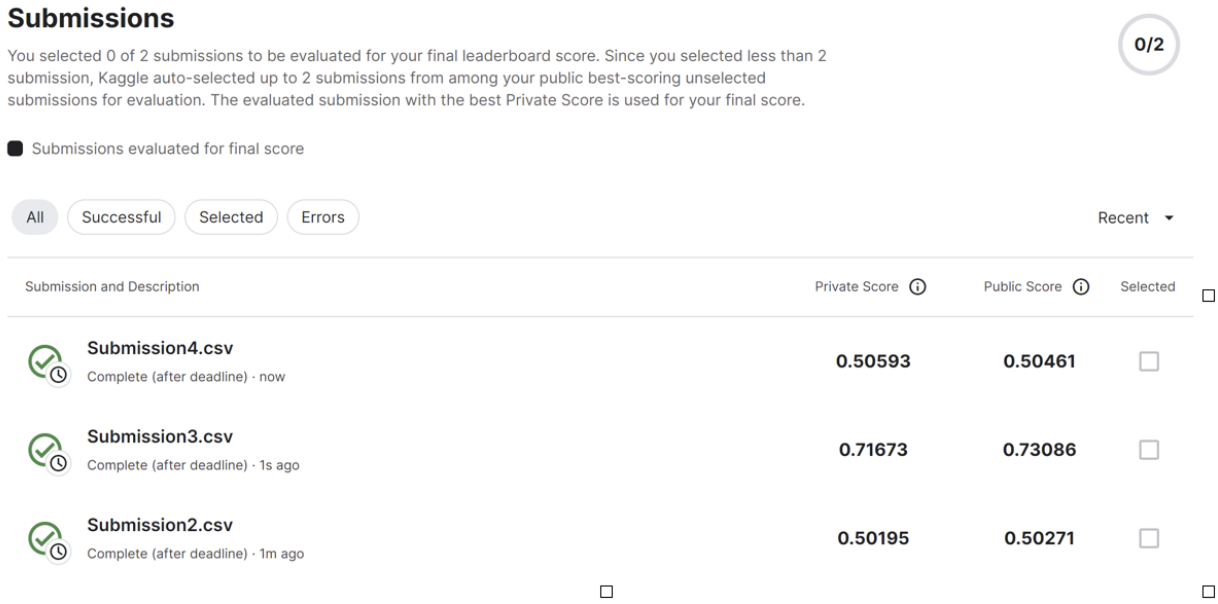
4. Lasso Regression: In Phase 3 after hyper-parameter tuning and addition feature engineering it gave us training and testing accuracy on a lower side but ROC on a higher side of 75.57%.

1. Ridge Regression: In Phase 3 after hyper-parameter tuning and addition feature engineering it gave us training and testing accuracy on a lower side but ROC on a higher side of 75.68%.

2. Deep Learning Model: In Phase 4 we built a Multi-layer perceptron model and experimented with different types of activation function like RuLU, Sigmoid etc. along with different epochs. Our best testing accuracy was 93.65 % along with ROC of 60.13%

- Below are our best kaggle submission from each Phase 2, 3, 4

Figure 24: Kaggle Submission



Conclusion

The major objective of this project is to develop a machine learning model that can forecast a loan applicant's ability to repay a loan. Without any statistical analysis, many deserving applicants with no credit history or default history are accepted. The HCDR dataset is used in our work to train the machine learning model. A user's average, minimum, and maximum balances as well as reported Bureau scores, salary, and other factors are used to generate a credit history, which serves as a gauge of their reliability.

We improved our classification model through feature engineering, feature selection, and hyperparameter tuning to more precisely forecast whether the loan applicant will be able to repay the loan or not. As part of this project, we developed machine learning pipelines, undertaking exploratory data analysis on the datasets provided by Kaggle, and evaluating the models using several evaluation measures before deploying one.

During Phase 2 we concluded that the best model for this dataset will be logistic regression having the highest accuracy of 91.9 %. In Phase 3, we expanded our feature engineering and added some more relevant features like AMT Credit to Annuity Ratio and Annuity to Salary Ratio. On top of that we discarded features with more than 30 % null values as compared to phase 2. Because of hyperparameter tuning our accuracy and AUC both increased significantly. Decision tree turned out the best model in phase 3 along with the best parameters as gini criterion, max_depth = 4 and min samples leaf = 4. We achieved accuracy of 92.13 % by decision tree. Along with that Lasso also turned out to be best model in terms of AUC 0.71.

As part of phase 4, we implemented Multi-Layer Perceptron with activation function ReLU and Sigmoid along with multiple hidden layer settings, we got the result that ReLU and Sigmoid together can give us the effective

result. We have determined that the Multi-Layer Perceptron's accuracy for the provided dataset using PyTorch was 93.65%, which is extremely effective and generally good. The test AUC for MLP has been fluctuating and has come down to 0.6 after performing multiple experiments.

With all the experiments taken into consideration Decision Tree and Lasso Regression turned out to be the best performing models. Additional experimentation on MLP model could have yielded better results.

Bibliography

- Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow, 2nd Edition
  - by Aurélien Géron
- Lab-End_to_end_Machine_Learning_Project
  - by James Shahnan
- HW 11 from assignments
  - by James Shahnan

In [ ]: