

# Home Credit Default Risk (HCDR) Group no. 15

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

## Some of the challenges

### 1. Dataset size

- (688 meg compressed) with millions of rows of data
- 2.71 Gig of data uncompressed
- Dealing with missing data
- Imbalanced datasets
- Summarizing transaction data

### Team Members:

Group Member	Member Picture	Group Member	Member Picture
Akash Gangadharan (akganga@iu.edu)		Laya Harwin (lharwin@iu.edu)	
Dhairya Shah (shahds@iu.edu)		Shobhit Sinha (shosinha@iu.edu)	

# Abstract

The main problems to tackle in this phase are merging the whole dataset, extracting relevant features, creating new features, treating missing values, applying OHE and imputing methods, engineering RFM features and optimizing the model's parameters using hyperparameter tuning. The main goal of the feature is to evaluate the model's performance on the test set to check for improvement. The key experiments that we will perform are first we will consider selected features and implement 5 models, second, we performed feature engineering without adding the new features and third we performed feature engineering, and we will add the newly generated result. Our results and finding are that with feature engineering and adding newly generated features, AdaBoost and MultinomialNB models achieved 91.83% accuracy. While on the other hand, Logistic Regression achieved 91.85% accuracy with feature engineering (in this case we did not consider the newly generated features).

# Phase Leader Plan

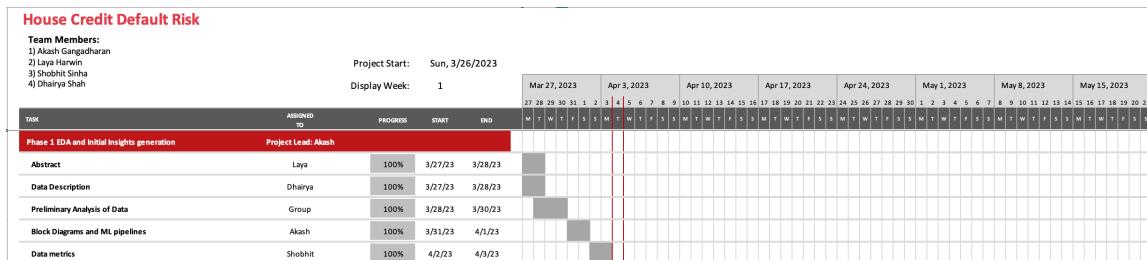
Phase Leader Plan				
Phase leader	Dhairya	Shobhit	Akash	Laya
Phase	Phase 3	Phase 4	Phase 1	Phase 2
Task	EDA and feature engineering	Creating model pipelines	Abstract	Data extraction
	Model Development- Logistic Regression, KNN	Visual representation of model performance	Data description	Exploratory Data analysis
	Pipeline update	Best model selection	Data analysis	Defining Metrics and block diagram
	Model Development- SVM, Random Forest	Validation	ML pipeline analysis	Create ML pipelines baseline models
	Hyperparameter Tuning	Summary	Credit Assessment	Combining codes
	Model Validation	Presentation	Overall Proposal	Slides and Video
	Presentation	Final Report	EDA	Presentation

# Credit assignment Table

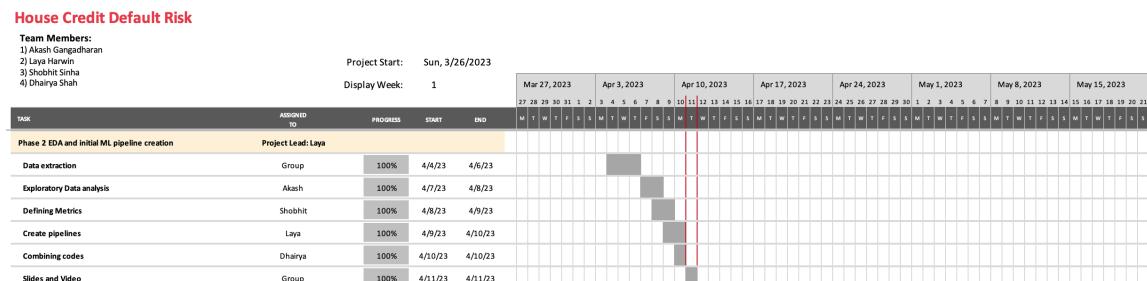
Names	Task	Description
Laya	Modeling Pipeline	Creating modeling pipelines for Logistic Regression, Decision Tree, Random Forest, and SVC
	Experimental Results	Performance comparison based on accuracy and AUC/ROC
	Discussion	Analysing the results obtained from experimental results
Akash	Feature Engineering of application_train, application_test, credit_card_balance	Performed Feature Engineering and did the entire EDA for these three files
	Missing value analysis, summary analysis, analysing input features with the target feature.	Performed Missing value analysis, summary statistics analysis and analysed distribution of input features with the Target feature.
	Correlation Analysis for Application train, application test and credit_card_balance.	Performed correlation analysis for all the three dataset and gave my insights.
	Project Description	Did the Data Description, Data Size, Tasks to tackle and creating block diagram.
	Credit Assignment Plan and Phase Leader plan	Divided the tasks equally among all group members
Dhairya	Hyper parameter Tuning on Jupyter Notebook	Did the Hyper parameter Tuning of base line models
	Gantt chart	Created Timelines for all the 4 phases and the Task that each member would be doing during that phase.
	Documentation for Report phase 3	Worked on content for abstract, loss function, machine learning pipeline etc.
Shobhit	Machine Learning and metrics	Identified the models that can be used and the metrics to consider to find the best model
	Feature Engineering of installments_payments, pos_cash_balance, previous_applicants, bureau and bureau_balance	Performed Feature Engineering and did the entire EDA for the mentioned files
	Missing value analysis, summary analysis, analysing input features with the target feature.	Performed Missing value analysis, summary statistics analysis and analysed distribution of input features with the Target feature.
	Correlation Analysis for installments_payments, pos_cash_balance, previous_applicants, bureau and bureau_balance	Performed correlation analysis for all the mentioned dataset and presented insights.
Group	Analysis of different ML models and accuracy metrics	Went deeply through all the CSV files and studied the relation between the different features
	Team photo	Through zoom call

# Gantt Chart for the Project timelines and work distribution

## Phase 1



## Phase 2



## Phase 3

House Credit Default Risk												
Team Members:		Project Details										
1) Akash Gangadharan 2) Laya Harwin 3) Shobhit Sinha 4) Dhairya Shah		Project Start: Sun, 3/26/2023			Display Week: 1			Timeline (Mar 27, 2023 - May 15, 2023)				
TASK	ASSIGNED TO	PROGRESS	START	END	Mar 27, 2023	Apr 3, 2023	Apr 10, 2023	Apr 17, 2023	Apr 24, 2023	May 1, 2023	May 8, 2023	May 15, 2023
Phase 3 Model Development and Feature Engineering	Project Lead: Dhairya				M	T	W	T	F	S	S	M
EDA and feature engineering	Group	100%	4/11/23	4/12/23								
Model Development- Logistic Regression, KNN	Akash	100%	4/12/23	4/13/23								
Pipeline update	Laya	100%	4/13/23	4/14/23								
Model Development- SVM, Random Forest	Shobhit	100%	4/16/23	4/17/23								
Hyperparameter Tuning	Dhairya	100%	4/17/23	4/18/23								

## Phase 4

House Credit Default Risk												
Team Members:		Project Details										
1) Akash Gangadharan 2) Laya Harwin 3) Shobhit Sinha 4) Dhairya Shah		Project Start: Sun, 3/26/2023			Display Week: 1			Timeline (Mar 27, 2023 - May 15, 2023)				
TASK	ASSIGNED TO	PROGRESS	START	END	Mar 27, 2023	Apr 3, 2023	Apr 10, 2023	Apr 17, 2023	Apr 24, 2023	May 1, 2023	May 8, 2023	May 15, 2023
Phase 4 Hyperparameter tuning and Model Performance	Project Lead: Shobhit				M	T	W	T	F	S	S	M
Creating model pipelines	Akash		4/18/23	4/19/23								
Visual representation of model performance	Laya		4/20/23	4/21/23								
Best model selection	Shobhit		4/22/23	4/23/23								
Validation	Dhairya		4/24/23	4/25/23								
Summary and final report	Group		4/25/23	4/25/23								

# Data Description:

A comprehensive dataset with a plethora of information on loan applicants and their credit histories is the Home Credit Default Risk (HCDR) dataset. The dataset comprises of various tables that are connected by a special identifier, making joining and data analysis simple. An overview of the tables found in the HCDR dataset is provided below:

**I. Application\_{train|test}.csv** : Basic information regarding loan applicants is provided in this table, including their age, gender, income, education level, marital status, and contact details. It also contains information on the loan they are requesting for, such as the type of loan, the amount of the loan, the interest rate, and the period.

**II.bureau.csv** : The applicant's previous credit, loan, and debt commitments are listed in this table. It contains details like the credit type, credit period, credit amount, amount still owed, and the condition of the credit.

**III.bureau\_balance.csv** : Monthly balance and status details for credits reported to the credit bureau throughout time are provided in this table. It has elements like the monthly balance, the number of days past due, and the credit status.

**IV.POS\_CASH\_balance.csv** : For the point of sale (POS) and cash loans, this table shows the monthly balance and status. It contains information on the monthly balance, how many days the loan is past due, and the status of the loan.

**V.credit\_card\_balance.csv** : For credit card loans, the monthly balance and status information is shown in this table. It contains information on the monthly balance, how many days the loan is past due, and the status of the loan.

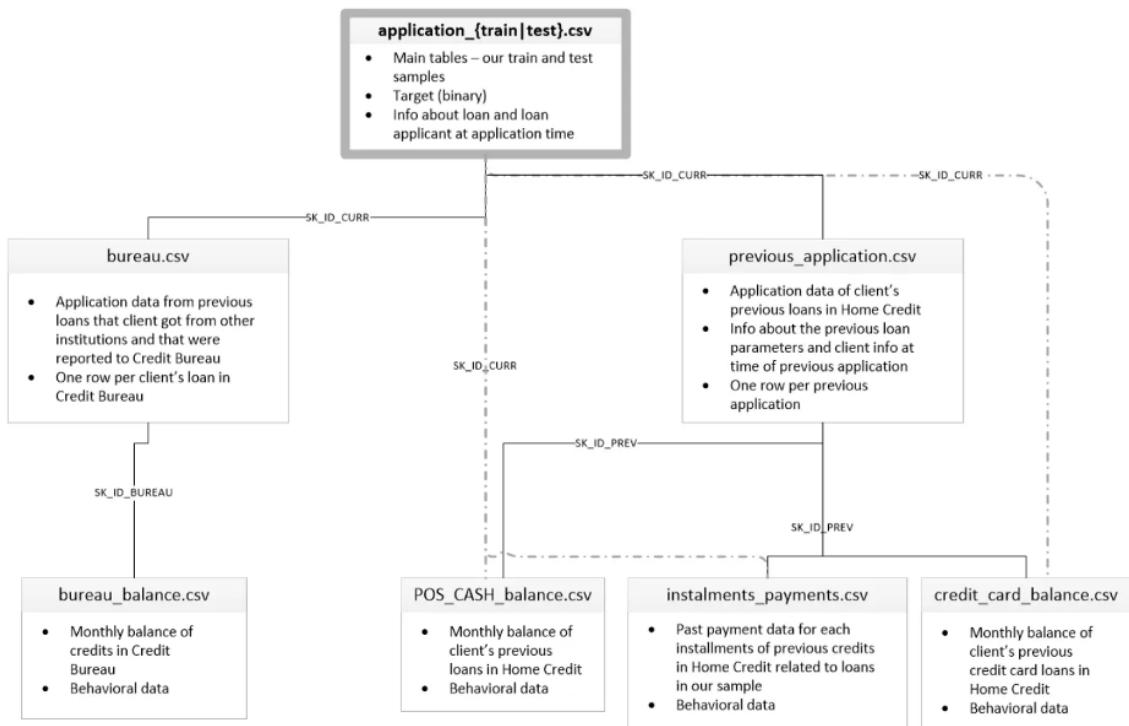
**VI.previous\_application.csv** : The applicant's past loan applications are detailed in this table. It contains information on the type of loan, the loan amount, the date of the

application, and the outcome of the prior application.

**VII. installments\_payments.csv** : Information on prior loans' repayment histories can be found in this table. It provides details like the payment amount, the payment date, and the amount of principal and interest still owed.

**VIII. HomeCredit\_columns\_description.csv** : This file is a metadata file that gives a thorough explanation of each column in the Home Credit Default Risk (HCDR) dataset. It contains details like the column name, data type, and a succinct summary of the contents in each column. A table from the dataset is represented by each of the file's several sections. Each section includes a list of the columns that are present in the associated table, as well as a description of the information that each column provides.

The data's relationships are depicted in the diagram below:



Overall, the HCDR dataset offers lenders a thorough understanding of an applicant's financial background and creditworthiness, making it a useful and complete resource for credit risk modeling.

## Machine Learning Algorithms

For the project, our team decided that all the team members will understand the data and brainstorms ideas to select models that could be employed for the provided dataset. We planned to create a data pipeline to handle highly correlated numerical and categorical features, as well as missing values in the dataset.

The target variable that needs to be predicted is binary. Hence, for this reason we decided to use Logistic Regression as the base model. We will decided to utilize Naive Bayes, KNN, Decision Tree, Random Forest and SVM (Support Vector Machine) to see

how unique feature of each model perform for the given dataset. Moreover, we will also make use of AdaBoost and XGBoost and Gradient Boosting Tree to compare and achieve the best results amongst all the models. These models will be trained and fine-tuned using hyper parameters and cross-fold validation through GridSearchCV.

## Metrics and Analysis

In order to evaluate the machine learning models for the project we decided to go forth with the metrics below to measure accuracy and effectiveness.

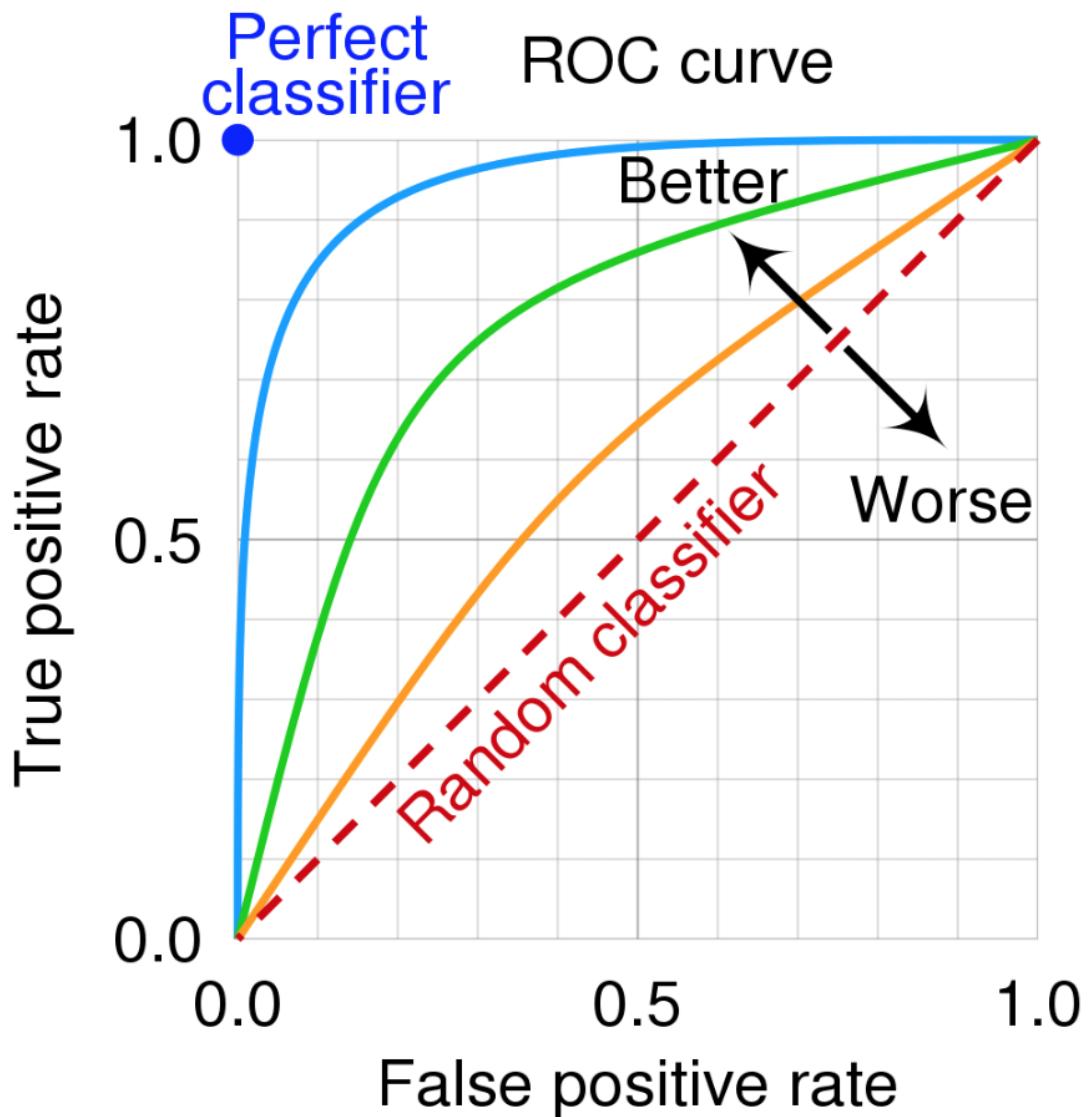
### Accuracy

This metric determines the total correctly classified labels.

$$\text{Accuracy} = \frac{\text{Number of correctly classified samples}}{\text{Total number of samples}} = \frac{TP + TN}{TP + TN + FP + FN}$$

### ROC/AUC Score

It ranks the probabilities of positive class label predictions and calculates the area under the curve plotted between True Positive Rates and False Positive Rates for each threshold value.



### Confusion Matrix

It measures the predictions made for the problem statement. Basically, measures the correct and incorrect predictions made on the dataset.

		True Class	
		Positive	Negative
Predicted Class	Positive	TP	FP
	Negative	FN	TN

Additionally, we will be using various loss functions such Mean Squared Error, Root Mean Squared Error, Categorical, Ridge, and Lasso to optimize each algorithm.

Overall, our aim is to build a model that accurately predicts credit repayment behavior and performs well on evaluations metrics.

## Kaggle API setup

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

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

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

### 1. Install library

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

For more detailed information on setting the Kaggle API see [here](#) and [here](#).

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

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

```
/Users/newuser/Courses/AML526/I526_AML_Student/Assignments/Unit-Project-Hom  
e-Credit-Default-Risk/Phase2
```

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

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

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

```
In [ ]: # ! kaggle competitions files home-credit-default-risk
```

## Dataset and how to download

### Back ground Home Credit Group

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

### Home Credit Group

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

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

### Background on the dataset

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

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

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

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

## Data files overview

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

There are 7 different sources of data:

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

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

## Table sizes

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

## Downloading the files via Kaggle API

Create a base directory:

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

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

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

```
In [ ]: DATA_DIR = ".../.../.../Data/home-credit-default-risk/home-credit-default-risk/  
#DATA_DIR = os.path.join('./ddddd/')  
# !mkdir DATA_DIR
```

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

```
total 5242728
-rw-r--r-- 1 newuser staff 37383 Apr 4 18:10 HomeCredit_columns_des
cription.csv
-rw-r--r-- 1 newuser staff 392703158 Apr 4 18:10 POS_CASH_balance.csv
-rw-r--r-- 1 newuser staff 26567651 Apr 4 18:10 application_test.csv
-rw-r--r-- 1 newuser staff 166133370 Apr 4 18:10 application_train.csv
-rw-r--r-- 1 newuser staff 170016717 Apr 4 18:10 bureau.csv
-rw-r--r-- 1 newuser staff 375592889 Apr 4 18:10 bureau_balance.csv
-rw-r--r-- 1 newuser staff 424582605 Apr 4 18:10 credit_card_balance.cs
v
-rw-r--r-- 1 newuser staff 723118349 Apr 4 18:10 installments_payments.
csv
-rw-r--r-- 1 newuser staff 404973293 Apr 4 18:10 previous_application.c
sv
-rw-r--r-- 1 newuser staff 536202 Apr 4 18:10 sample_submission.csv
```

In [ ]: !ls -l \$DATA\_DIR/home-credit-default-risk

```
ls: ../../Data/home-credit-default-risk/home-credit-default-risk//home-credit-default-risk: No such file or directory
```

## Imports

In [ ]:

```
import numpy as np
import pandas as pd
from sklearn.preprocessing import LabelEncoder
import os
import zipfile
import time
from sklearn.base import BaseEstimator, TransformerMixin
import matplotlib.pyplot as plt
import seaborn as sns
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.model_selection import KFold
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import GridSearchCV
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import MinMaxScaler
from sklearn.pipeline import Pipeline, FeatureUnion
from pandas.plotting import scatter_matrix
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import OneHotEncoder
import warnings
warnings.filterwarnings('ignore')
pd.set_option('display.max_columns', None)
pd.set_option('display.max_rows', None)
pd.set_option('display.width', None)
pd.set_option('display.max_colwidth', None)
```

## Data files overview

## Data Dictionary

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

```
HomeCredit_columns_description.csv
```

```
In [ ]: # import numpy as np
# import pandas as pd
# from sklearn.preprocessing import LabelEncoder
# import os
# import zipfile
# from sklearn.base import BaseEstimator, TransformerMixin
# import matplotlib.pyplot as plt
# import seaborn as sns
# from sklearn.linear_model import LogisticRegression
# from sklearn.model_selection import train_test_split
# from sklearn.model_selection import KFold
# from sklearn.model_selection import cross_val_score
# from sklearn.model_selection import GridSearchCV
# from sklearn.ensemble import AdaBoostClassifier
# from sklearn.impute import SimpleImputer
# from sklearn.preprocessing import MinMaxScaler
# from sklearn.pipeline import Pipeline, FeatureUnion
# from pandas.plotting import scatter_matrix
# from sklearn.preprocessing import StandardScaler
# from sklearn.preprocessing import OneHotEncoder
# import warnings
# warnings.filterwarnings('ignore')
# pd.set_option('display.max_columns', None)
# pd.set_option('display.max_rows', None)
# pd.set_option('display.width', None)
# pd.set_option('display.max_colwidth', None)

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

datasets={} # lets store the datasets in a dictionary so we can keep track
# ds_name = 'application_train'
DATA_DIR=f'{DATA_DIR}'
# datasets[ds_name] = load_data(os.path.join(DATA_DIR, f'{ds_name}.csv'), ds

# datasets['application_train'].shape
```

```
In [ ]: DATA_DIR
```

```
Out[ ]: '../../../../../Data/home-credit-default-risk/home-credit-default-risk/'
```

## Application test

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

```
In [ ]: ds_name = 'application_test'
datasets[ds_name] = load_data(os.path.join(DATA_DIR, f'{ds_name}.csv'), ds_r
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_REALM
0	100001	Cash loans	F	N	
1	100005	Cash loans	M	N	
2	100013	Cash loans	M	Y	
3	100028	Cash loans	F	N	
4	100038	Cash loans	M	Y	

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

## The Other datasets

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

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

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

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

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

```
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	CREDI
0	215354	5714462	Closed	currency 1	-497	
1	215354	5714463	Active	currency 1	-208	
2	215354	5714464	Active	currency 1	-203	
3	215354	5714465	Active	currency 1	-203	
4	215354	5714466	Active	currency 1	-629	

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

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

```
credit_card_balance: shape is (3840312, 23)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 3840312 entries, 0 to 3840311
Data columns (total 23 columns):
 #   Column            Dtype  
 --- 
 0   SK_ID_PREV        int64  
 1   SK_ID_CURR        int64  
 2   MONTHS_BALANCE    int64  
 3   AMT_BALANCE       float64 
 4   AMT_CREDIT_LIMIT_ACTUAL int64  
 5   AMT_DRAWINGS_ATM_CURRENT float64 
 6   AMT_DRAWINGS_CURRENT float64  
 7   AMT_DRAWINGS_OTHER_CURRENT float64 
 8   AMT_DRAWINGS_POS_CURRENT float64 
 9   AMT_INST_MIN_REGULARITY float64 
 10  AMT_PAYMENT_CURRENT float64 
 11  AMT_PAYMENT_TOTAL_CURRENT float64 
 12  AMT_RECEIVABLE_PRINCIPAL float64 
 13  AMT_RECVABLE        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_ACTL
0	2562384	378907	-6	56.970	1351
1	2582071	363914	-1	63975.555	450
2	1740877	371185	-7	31815.225	450
3	1389973	337855	-4	236572.110	2250
4	1891521	126868	-1	453919.455	450

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	DA
0	1054186	161674		1.0	6
1	1330831	151639		0.0	34
2	2085231	193053		2.0	1
3	2452527	199697		1.0	3
4	2714724	167756		1.0	2

```
previous_application: shape is (1670214, 37)
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1670214 entries, 0 to 1670213
Data columns (total 37 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   SK_ID_PREV      1670214 non-null   int64  
 1   SK_ID_CURR      1670214 non-null   int64  
 2   NAME_CONTRACT_TYPE 1670214 non-null   object  
 3   AMT_ANNUITY     1297979 non-null   float64 
 4   AMT_APPLICATION 1670214 non-null   float64 
 5   AMT_CREDIT       1670213 non-null   float64 
 6   AMT_DOWN_PAYMENT 774370 non-null   float64 
 7   AMT_GOODS_PRICE  1284699 non-null   float64 
 8   WEEKDAY_APPR_PROCESS_START 1670214 non-null   object  
 9   HOUR_APPR_PROCESS_START 1670214 non-null   int64  
 10  FLAG_LAST_APPL_PER_CONTRACT 1670214 non-null   object  
 11  NFLAG_LAST_APPL_IN_DAY    1670214 non-null   int64  
 12  RATE_DOWN_PAYMENT    774370 non-null   float64 
 13  RATE_INTEREST_PRIMARY 5951 non-null   float64 
 14  RATE_INTEREST_PRIVILEGED 5951 non-null   float64 
 15  NAME_CASH_LOAN_PURPOSE 1670214 non-null   object  
 16  NAME_CONTRACT_STATUS 1670214 non-null   object  
 17  DAYS_DECISION      1670214 non-null   int64  
 18  NAME_PAYMENT_TYPE   1670214 non-null   object  
 19  CODE_REJECT_REASON 1670214 non-null   object  
 20  NAME_TYPE_SUITE     849809 non-null   object  
 21  NAME_CLIENT_TYPE    1670214 non-null   object  
 22  NAME_GOODS_CATEGORY 1670214 non-null   object  
 23  NAME_PORTFOLIO      1670214 non-null   object  
 24  NAME_PRODUCT_TYPE   1670214 non-null   object  
 25  CHANNEL_TYPE        1670214 non-null   object  
 26  SELLERPLACE_AREA    1670214 non-null   int64  
 27  NAME_SELLER_INDUSTRY 1670214 non-null   object  
 28  CNT_PAYMENT         1297984 non-null   float64 
 29  NAME_YIELD_GROUP    1670214 non-null   object  
 30  PRODUCT_COMBINATION 1669868 non-null   object  
 31  DAYS_FIRST_DRAWING 997149 non-null   float64 
 32  DAYS_FIRST_DUE      997149 non-null   float64 
 33  DAYS_LAST_DUE_1ST_VERSION 997149 non-null   float64 
 34  DAYS_LAST_DUE       997149 non-null   float64 
 35  DAYS_TERMINATION    997149 non-null   float64 
 36  NFLAG_INSURED_ON_APPROVAL 997149 non-null   float64 

dtypes: float64(15), int64(6), object(16)
memory usage: 471.5+ MB
None
```

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

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

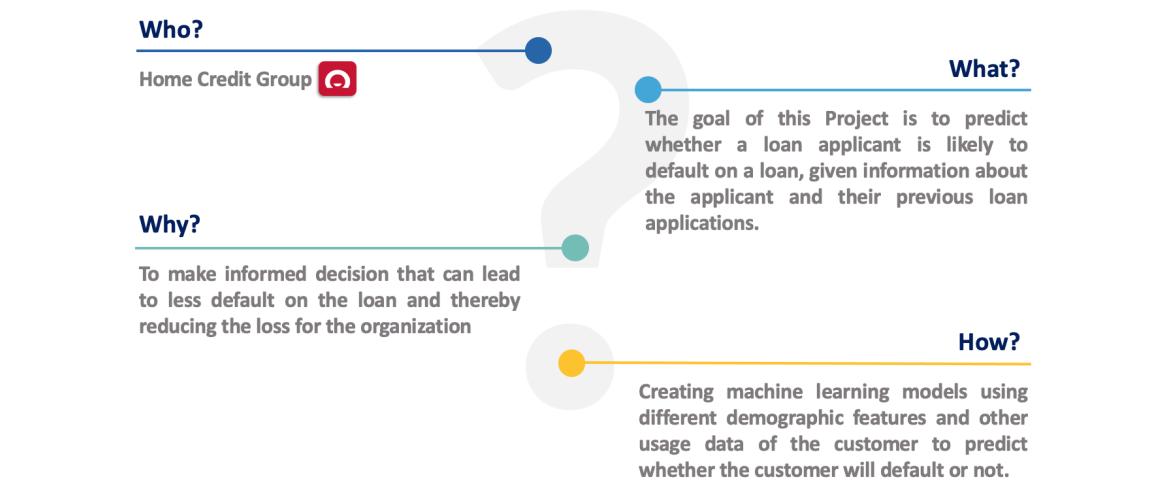
	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	CNT_INSTALMENT	CNT_INSTALMENT_FU
0	1803195	182943		-31	48.0
1	1715348	367990		-33	36.0
2	1784872	397406		-32	12.0
3	1903291	269225		-35	48.0
4	2341044	334279		-35	36.0
..	..	..	..	..	..

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

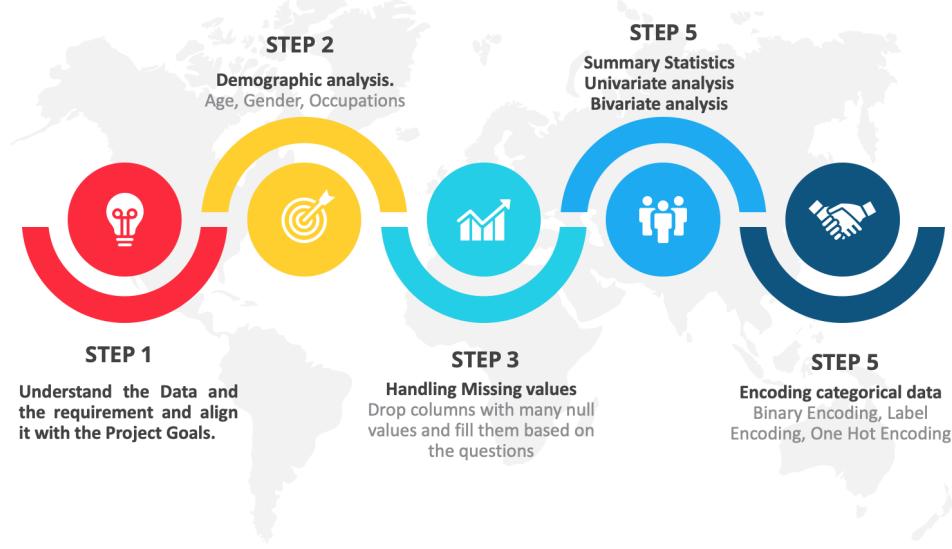
dataset application_test	: [ 48,744, 121]
dataset application_train	: [ 307,511, 122]
dataset bureau	: [ 1,716,428, 17]
dataset bureau_balance	: [ 27,299,925, 3]
dataset credit_card_balance	: [ 3,840,312, 23]
dataset installments_payments	: [ 13,605,401, 8]
dataset previous_application	: [ 1,670,214,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

# Project Description

## What are we aiming to solve?



## EDA and Data Preparation



## Data Description

The data consists of several tables giving information on loan applications, applicants' previous credit history, and credit card balance details.

Here's a quick rundown of the information in the Home Credit Default Risk competition:

\* **application {train|test}.csv** \*: This is the main table, which contains details about each loan application. Each row in this table represents one loan application, and

each column indicates a feature or variable associated with that loan application, such as the loan amount, the age of the applicant, or the type of loan.

\* **bureau.csv** \*: This table contains information on each loan obtained by the applicant from different financial institutions. Each row in this table represents one loan, and each column represents a feature or variable associated with that loan, such as the loan amount, the date the loan was obtained, or the loan's status.

\* **bureau\_balance.csv** \*: This table contains monthly information on each loan obtained by the applicant from different financial institutions. Each row in this table represents a month for a single loan, and each column reflects a feature or variable associated with that month, such as the amount paid, the number of delinquent payments, or the loan's status.

\* **previous\_application.csv** \*: This table contains information on prior loan applications submitted by the applicant to Home Credit. Each row in this table represents a previous loan application, and each column reflects a feature or variable associated with that loan application, such as the loan amount, the date the loan was accepted, or the loan application's outcome.

\* **POS\_CASH\_balance.csv** \*: This table contains monthly information regarding the applicant's previous point of sale or cash loan with Home Credit. Each row in this table represents a month for a single loan, and each column reflects a feature or variable associated with that month, such as the amount paid, the number of delinquent payments, or the loan's status.

\* **credit\_card\_balance.csv** \*: This table shows monthly statistics for each previous credit card with Home Credit that the applicant holds. Each row in this table represents one month for one credit card, and each column reflects a feature or variable associated with that month, such as the amount paid, the number of past-due payments, or the credit card's status.

\* **installments\_payments.csv** \*: This table offers information on the installments paid on past Home Credit loans. Each row in this table represents one payment, and each column indicates a feature or variable associated with that payment, such as the payment amount, date, or status.

## Data Size

The number of rows and columns in each file are as follows:

**application\_train.csv** : 307,511 rows, 122 columns

**application\_test.csv** : 48744 rows, 121 columns

**bureau.csv** : 1,716,428 rows, 17 columns

**bureau\_balance.csv** : 27,299,925 rows, 3 columns

**credit\_card\_balance.csv** : 38,403,12 rows, 23 columns

**installments\_payments.csv** : 136,054,01 rows, 8 columns

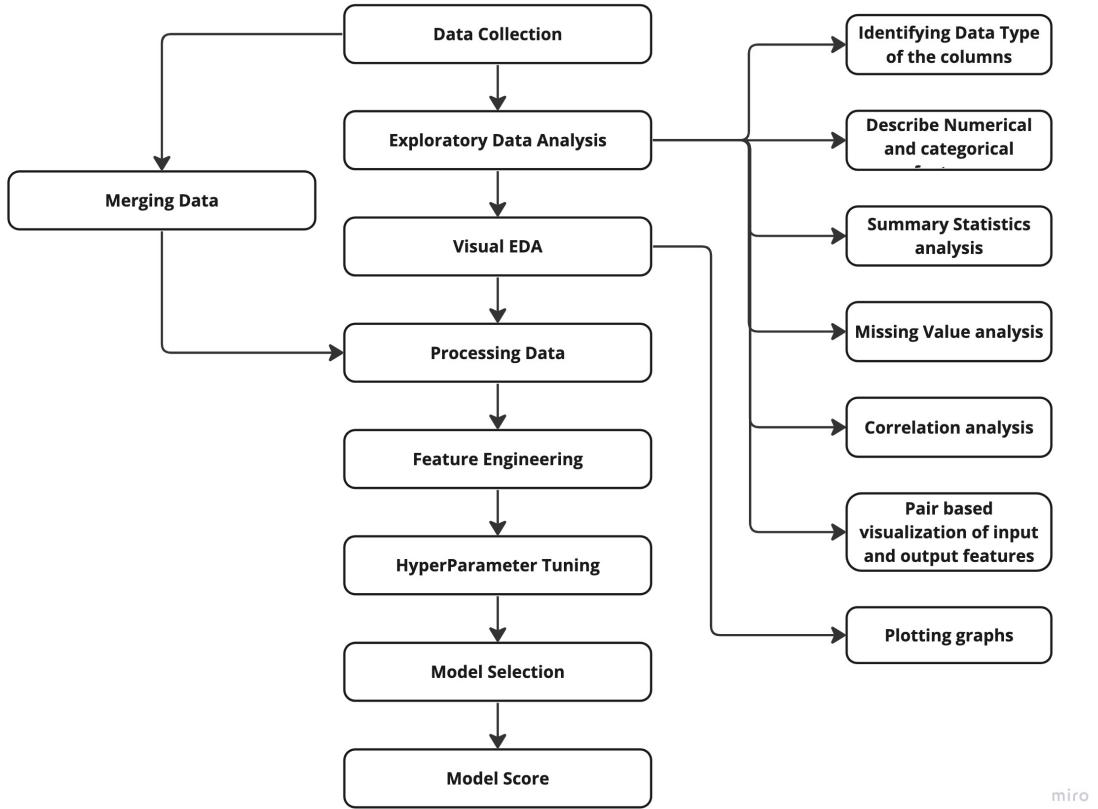
**POS\_CASH\_balance.csv** : 10,001,358 rows, 8 columns

**previous\_application.csv** : 1,670,214 rows, 37 columns

## Task to be tackled

- Merge the whole dataset
- Extract all possible features using feature Engineering
- Clean the data
- Performing feature selection.
- Provide analysis of feature importance.
- Identify the most relevant features to the Target variable and prediction
- Create new features from the most correlated features to contribute towards the prediction
- Apply One-Hot Encoding (OHE) to categorical features
- Apply imputing methods to fix missing values in the data
- Prepare the data for feeding into the model
- Perform hyperparameter tuning for the models using K-Fold cross validation and GridSearchCV
- Evaluate the model's performance on the test set to check if accuracy has improved

## Plan or methodology



miro

**Data collection** is the first step in any machine learning pipeline. It involves gathering and compiling relevant data from various sources, which may include databases, APIs, web scraping, or other means. The quality and relevance of the data collected is crucial to the success of the entire pipeline, as it forms the basis for all subsequent steps.

In the data collection stage, it is important to identify and understand the problem to be solved and the data required to solve it. This includes defining the variables of interest, identifying potential data sources, and collecting the necessary data from these sources.

Data collection can be an iterative process, and it is often necessary to refine the data collection process as new insights are gained through the EDA (Exploratory Data Analysis) process. It is important to carefully document the data collection process, including any assumptions or limitations, as this information will be important for interpreting the results of the subsequent analysis.

During data collection, it is also important to ensure that the data is properly cleaned and preprocessed before analysis. This includes handling missing values, dealing with outliers, and transforming the data as necessary to meet the assumptions of the analysis.

Overall, data collection is a critical step in the machine learning pipeline, and it requires careful planning and attention to detail to ensure that the resulting data is of high quality and relevance to the problem being solved.

**Exploratory Data Analysis** : Exploratory Data Analysis (EDA) is a critical phase in the Machine Learning (ML) pipeline in which data scientists or ML engineers evaluate and analyze the information to understand the underlying structure, trends, and correlations.

EDA employs a number of approaches, including data visualization, summary statistics, and statistical testing, to obtain insights into the dataset's features, detect possible faults or anomalies, and influence the next steps in the ML pipeline.

To prepare the data for modeling, data scientists may undertake activities such as data cleansing, missing value imputation, outlier identification, and feature engineering during EDA. They may also evaluate the balance of the classes or target variable in the dataset and decide whether to use data balancing techniques like oversampling or undersampling.

Overall, EDA is crucial in assuring data quality and assisting in the selection of the best relevant algorithms and strategies to develop a strong and accurate ML model.

**Visual EDA** : Visual EDA, or visual Exploratory Data Analysis, is a technique that involves using visualizations to understand the structure and patterns in a dataset. Visual EDA aims to provide a quick and intuitive understanding of the data, enabling data scientists to make informed decisions on data preprocessing, feature engineering, and modeling.

Visual EDA techniques typically involve creating plots and charts that illustrate the distribution, relationship, and variability of the data. Some common visualizations used in EDA include histograms, scatter plots, box plots, heatmaps, and bar charts.

By visualizing the data, we can identify any potential issues with the dataset, such as missing or outlier values, skewed distributions, or correlations between variables. Visual EDA can also help in selecting appropriate data preprocessing techniques, such as scaling or normalization, and identifying which features may be relevant for the ML model.

Overall, visual EDA is an important step in the ML pipeline that enables data scientists to gain a deeper understanding of the data and make informed decisions about data preprocessing, feature engineering, and modeling.

**Processing Data** : Processing data refers to the steps involved in preparing and transforming raw data into a format that is suitable for analysis or machine learning (ML) models. Processing data is a crucial step in the data science workflow, as raw data may contain errors, inconsistencies, or missing values that can adversely affect the accuracy and reliability of the analysis.

The first step in processing data is to clean the data, which involves identifying and correcting any errors, removing duplicate records, and dealing with missing data. Data

cleaning may also involve transforming data into a consistent format, such as converting categorical data into numerical data or changing the date format to a standardized format.

After cleaning the data, the next step is to preprocess the data, which involves transforming the data to make it more suitable for analysis or ML models. Preprocessing may involve feature scaling, which ensures that all features are on a similar scale to avoid bias in the analysis. Feature engineering, which involves creating new features that may be more relevant to the analysis, is another important preprocessing step.

**Modeling** : Modeling refers to the process of creating a machine learning (ML) model that can make predictions or decisions based on input data. The goal of modeling is to develop a model that can generalize well to new, unseen data and accurately predict the outcome of interest.

The modeling process typically involves the following steps:

Selecting a model: Choosing an appropriate ML model that is best suited for the problem being solved. This can involve considering factors such as the type of data, the number of features, and the desired output.

Training the model: Using a training dataset to fit the model to the data. This involves estimating the model parameters using a chosen optimization algorithm that minimizes the error between the predicted and actual outcomes.

Evaluating the model: Using a validation dataset to assess the performance of the model. This involves measuring metrics such as accuracy, precision, recall, or F1-score to evaluate how well the model is performing.

Tuning the model: Adjusting the model parameters to optimize its performance on the validation dataset. This involves techniques such as grid search, random search, or Bayesian optimization to find the best combination of parameters.

Testing the model: Using a separate test dataset to evaluate the final performance of the model. This is done to ensure that the model can generalize well to new, unseen data.

Once the model has been developed and tested, it can be used for making predictions or decisions on new data. The model can also be updated or retrained periodically to ensure that it remains accurate and up-to-date.

**Feature Engineering** : Feature engineering is the process of creating new features, or variables, from existing raw data that can improve the performance of machine learning (ML) models. Feature engineering involves extracting information from the raw data and representing it in a way that is more suitable for modeling.

The goal of feature engineering is to create features that are relevant to the problem being solved and that capture the underlying patterns in the data. This process may involve domain knowledge or intuition about the data, as well as experimentation and iteration to find the most effective features.

Feature engineering can involve a variety of techniques, such as:

Feature transformation: Transforming variables to fit a particular distribution or to normalize the data. Examples include log transformation, scaling, and standardization.

Feature extraction: Creating new variables by extracting information from existing variables. For example, extracting the day of the week or month from a date variable, or extracting the length of text.

Feature aggregation: Creating new variables by combining multiple variables. For example, calculating the average or sum of a group of variables, or computing the percentage of one variable relative to another.

Feature selection: Selecting a subset of the most important variables for modeling. This can involve techniques such as correlation analysis, principal component analysis, or feature importance ranking.

Feature generation: Creating new variables through mathematical operations or other data manipulations. This can include creating interaction terms between variables, or combining categorical variables into a single feature using one-hot encoding.

Overall, feature engineering is an essential step in the ML pipeline that can significantly improve the accuracy and performance of models. It involves creating new features that capture relevant information from the data, and can require a combination of domain knowledge, creativity, and experimentation to achieve the best results.

**Hyper Parameter Tuning** : Hyperparameter tuning refers to the process of optimizing the hyperparameters of a machine learning (ML) model to achieve better performance. Hyperparameters are parameters that are not learned during the training process but are set before training and can significantly impact the performance of the model.

Hyperparameter tuning involves searching for the optimal combination of hyperparameters that results in the best model performance. This is typically done using a validation dataset, where the performance of different hyperparameter configurations is evaluated.

Some common hyperparameters that may be tuned include:

Learning rate: A hyperparameter that determines how quickly the model learns during training.

Regularization parameters: Hyperparameters that control the level of regularization applied to the model, such as L1 or L2 regularization.

Number of layers: Hyperparameters that determine the number of layers in a neural network.

Number of hidden units: Hyperparameters that determine the number of neurons in each layer of a neural network.

Dropout rate: A hyperparameter that determines the amount of dropout regularization applied to a neural network.

Hyperparameter tuning can be done using a variety of techniques, including grid search, random search, and Bayesian optimization. Grid search involves evaluating the model performance for all possible combinations of hyperparameters within a predefined range. Random search involves randomly sampling the hyperparameter space and evaluating the performance. Bayesian optimization uses a probabilistic model to select the next set of hyperparameters to evaluate, based on the previous results.

## Exploratory Data Analysis and Visual EDA

The EDA covers both text based analysis and is shown Visually as well.

### Application\_Train

The \* **application\_{train|test}.csv** \* files contain the following columns:

**SK\_ID\_CURR** : ID of the loan applicant

**TARGET** : binary variable indicating whether the loan was repaid or not (only available in the train file)

**NAME\_CONTRACT\_TYPE** : type of loan (cash loan or revolving loan)

**CODE\_GENDER** : gender of the applicant

**FLAG\_OWN\_CAR** : whether the applicant owns a car

**FLAG\_OWN\_REALTY** : whether the applicant owns real estate

**CNT\_CHILDREN** : number of children the applicant has

**AMT\_INCOME\_TOTAL** : total income of the applicant

**AMT\_CREDIT** : total credit amount of the loan

**AMT\_ANNUITY** : loan annuity (monthly payment)

**AMT\_GOODS\_PRICE** : price of the goods for which the loan is given

**NAME\_TYPE\_SUITE** : who was accompanying the applicant when he/she applied for the loan

**NAME\_INCOME\_TYPE** : income category of the applicant

**NAME\_EDUCATION\_TYPE** : level of education of the applicant

**NAME\_FAMILY\_STATUS** : family status of the applicant  
**NAME\_HOUSING\_TYPE** : housing situation of the applicant  
**REGION\_POPULATION\_RELATIVE** : population density of the region where the applicant lives  
**DAYS\_BIRTH** : age of the applicant (in days, negative value)  
**DAYS\_EMPLOYED** : number of days the applicant has been employed (negative value for unemployed)  
**DAYS\_REGISTRATION** : number of days the applicant has been registered in the region (negative value for not registered)  
**DAYS\_ID\_PUBLISH** : number of days since the loan application was published (negative value for not yet published)  
**OWN\_CAR\_AGE** : age of the applicant's car (if applicable)  
**FLAG\_MOBIL** : whether the applicant provided a mobile phone number  
**FLAG\_EMP\_PHONE** : whether the applicant provided a work phone number  
**FLAG\_WORK\_PHONE** : whether the applicant provided a home phone number  
**FLAG\_CONT\_MOBILE** : whether the applicant's mobile phone can be contacted  
**FLAG\_PHONE** : whether the applicant provided a phone number  
**FLAG\_EMAIL** : whether the applicant provided an email address  
**OCCUPATION\_TYPE** : occupation of the applicant  
**CNT\_FAM\_MEMBERS** : number of family members of the applicant  
**REGION\_RATING\_CLIENT** : region rating where the applicant lives (1,2,3)  
**REGION\_RATING\_CLIENT\_W\_CITY** : region rating where the applicant lives with taking city into account (1,2,3)  
**WEEKDAY\_APPR\_PROCESS\_START** : day of the week when the application was started  
**HOUR\_APPR\_PROCESS\_START** : hour of the day when the application was started  
**REG\_REGION\_NOT\_LIVE\_REGION** : flag indicating if the region of the applicant's contact address does not match the region of the applicant's permanent address  
**REG\_REGION\_NOT\_WORK\_REGION** : flag indicating if the region of the applicant's contact address does not match the region where he/she works  
**LIVE\_REGION\_NOT\_WORK\_REGION** : flag indicating if the region of the applicant's permanent address does not match the region where he/she works  
**REG\_CITY\_NOT\_LIVE\_CITY** : flag indicating if the city of the applicant's contact address does not match the city of his/her permanent address  
**REG\_CITY\_NOT\_WORK\_CITY** : flag indicating if the city of the applicant's contact address does not match the

## Column wise data types

```
In [ ]: # Load the data
df = pd.DataFrame(datasets["application_train"])

print(df.dtypes)
```

SK_ID_CURR	int64
TARGET	int64
NAME_CONTRACT_TYPE	object
CODE_GENDER	object
FLAG_OWN_CAR	object
FLAG_OWN_REALTY	object
CNT_CHILDREN	int64
AMT_INCOME_TOTAL	float64
AMT_CREDIT	float64
AMT_ANNUITY	float64
AMT_GOODS_PRICE	float64
NAME_TYPE_SUITE	object
NAME_INCOME_TYPE	object
NAME_EDUCATION_TYPE	object
NAME_FAMILY_STATUS	object
NAME_HOUSING_TYPE	object
REGION_POPULATION_RELATIVE	float64
DAYS_BIRTH	int64
DAYS_EMPLOYED	int64
DAYS_REGISTRATION	float64
DAYS_ID_PUBLISH	int64
OWN_CAR_AGE	float64
FLAG_MOBIL	int64
FLAG_EMP_PHONE	int64
FLAG_WORK_PHONE	int64
FLAG_CONT_MOBILE	int64
FLAG_PHONE	int64
FLAG_EMAIL	int64
OCCUPATION_TYPE	object
CNT_FAM_MEMBERS	float64
REGION_RATING_CLIENT	int64
REGION_RATING_CLIENT_W_CITY	int64
WEEKDAY_APPR_PROCESS_START	object
HOUR_APPR_PROCESS_START	int64
REG_REGION_NOT_LIVE_REGION	int64
REG_REGION_NOT_WORK_REGION	int64
LIVE_REGION_NOT_WORK_REGION	int64
REG_CITY_NOT_LIVE_CITY	int64
REG_CITY_NOT_WORK_CITY	int64
LIVE_CITY_NOT_WORK_CITY	int64
ORGANIZATION_TYPE	object
EXT_SOURCE_1	float64
EXT_SOURCE_2	float64
EXT_SOURCE_3	float64
APARTMENTS_AVG	float64
BASEMENTAREA_AVG	float64
YEARS_BEGINEXPLUATATION_AVG	float64
YEARS_BUILD_AVG	float64
COMMONAREA_AVG	float64
ELEVATORS_AVG	float64
ENTRANCES_AVG	float64
FLOORSMAX_AVG	float64
FLOORSMIN_AVG	float64
LANDAREA_AVG	float64
LIVINGAPARTMENTS_AVG	float64
LIVINGAREA_AVG	float64

NONLIVINGAPARTMENTS_AVG	float64
NONLIVINGAREA_AVG	float64
APARTMENTS_MODE	float64
BASEMENTAREA_MODE	float64
YEARS_BEGINEXPLUATATION_MODE	float64
YEARS_BUILD_MODE	float64
COMMONAREA_MODE	float64
ELEVATORS_MODE	float64
ENTRANCES_MODE	float64
FLOORSMAX_MODE	float64
FLOORSMIN_MODE	float64
LANDAREA_MODE	float64
LIVINGAPARTMENTS_MODE	float64
LIVINGAREA_MODE	float64
NONLIVINGAPARTMENTS_MODE	float64
NONLIVINGAREA_MODE	float64
APARTMENTS_MEDI	float64
BASEMENTAREA_MEDI	float64
YEARS_BEGINEXPLUATATION_MEDI	float64
YEARS_BUILD_MEDI	float64
COMMONAREA_MEDI	float64
ELEVATORS_MEDI	float64
ENTRANCES_MEDI	float64
FLOORSMAX_MEDI	float64
FLOORSMIN_MEDI	float64
LANDAREA_MEDI	float64
LIVINGAPARTMENTS_MEDI	float64
LIVINGAREA_MEDI	float64
NONLIVINGAPARTMENTS_MEDI	float64
NONLIVINGAREA_MEDI	float64
FONDKAPREMONT_MODE	object
HOUSETYPE_MODE	object
TOTALAREA_MODE	float64
WALLSMATERIAL_MODE	object
EMERGENCYSTATE_MODE	object
OBS_30_CNT_SOCIAL_CIRCLE	float64
DEF_30_CNT_SOCIAL_CIRCLE	float64
OBS_60_CNT_SOCIAL_CIRCLE	float64
DEF_60_CNT_SOCIAL_CIRCLE	float64
DAYSLAST_PHONE_CHANGE	float64
FLAG_DOCUMENT_2	int64
FLAG_DOCUMENT_3	int64
FLAG_DOCUMENT_4	int64
FLAG_DOCUMENT_5	int64
FLAG_DOCUMENT_6	int64
FLAG_DOCUMENT_7	int64
FLAG_DOCUMENT_8	int64
FLAG_DOCUMENT_9	int64
FLAG_DOCUMENT_10	int64
FLAG_DOCUMENT_11	int64
FLAG_DOCUMENT_12	int64
FLAG_DOCUMENT_13	int64
FLAG_DOCUMENT_14	int64
FLAG_DOCUMENT_15	int64
FLAG_DOCUMENT_16	int64
FLAG_DOCUMENT_17	int64

```
FLAG_DOCUMENT_18           int64
FLAG_DOCUMENT_19           int64
FLAG_DOCUMENT_20           int64
FLAG_DOCUMENT_21           int64
AMT_REQ_CREDIT_BUREAU_HOUR float64
AMT_REQ_CREDIT_BUREAU_DAY float64
AMT_REQ_CREDIT_BUREAU_WEEK float64
AMT_REQ_CREDIT_BUREAU_MON float64
AMT_REQ_CREDIT_BUREAU_QRT float64
AMT_REQ_CREDIT_BUREAU_YEAR float64
dtype: object
```

## Data Size

```
In [ ]: print("Number of rows in the Application_train ",df.shape[0])
print("Number of Columns in the Application_train ",df.shape[1])
```

```
Number of rows in the Application_train  307511
Number of Columns in the Application_train  122
```

## Summary statistics of Application train

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

```
Out[ ]:
```

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT
<b>count</b>	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05
<b>mean</b>	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05
<b>std</b>	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05
<b>min</b>	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04
<b>25%</b>	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05
<b>50%</b>	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05
<b>75%</b>	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05
<b>max</b>	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06

```
In [ ]: datasets["application_train"].describe(include='all') #look at all categoric
```

Out[ ]:

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

## Correlation Analysis

In [ ]:

```
# Select a subset of variables
subset_vars = ['TARGET', 'AMT_INCOME_TOTAL', 'AMT_CREDIT', 'DAYS_BIRTH', 'DA

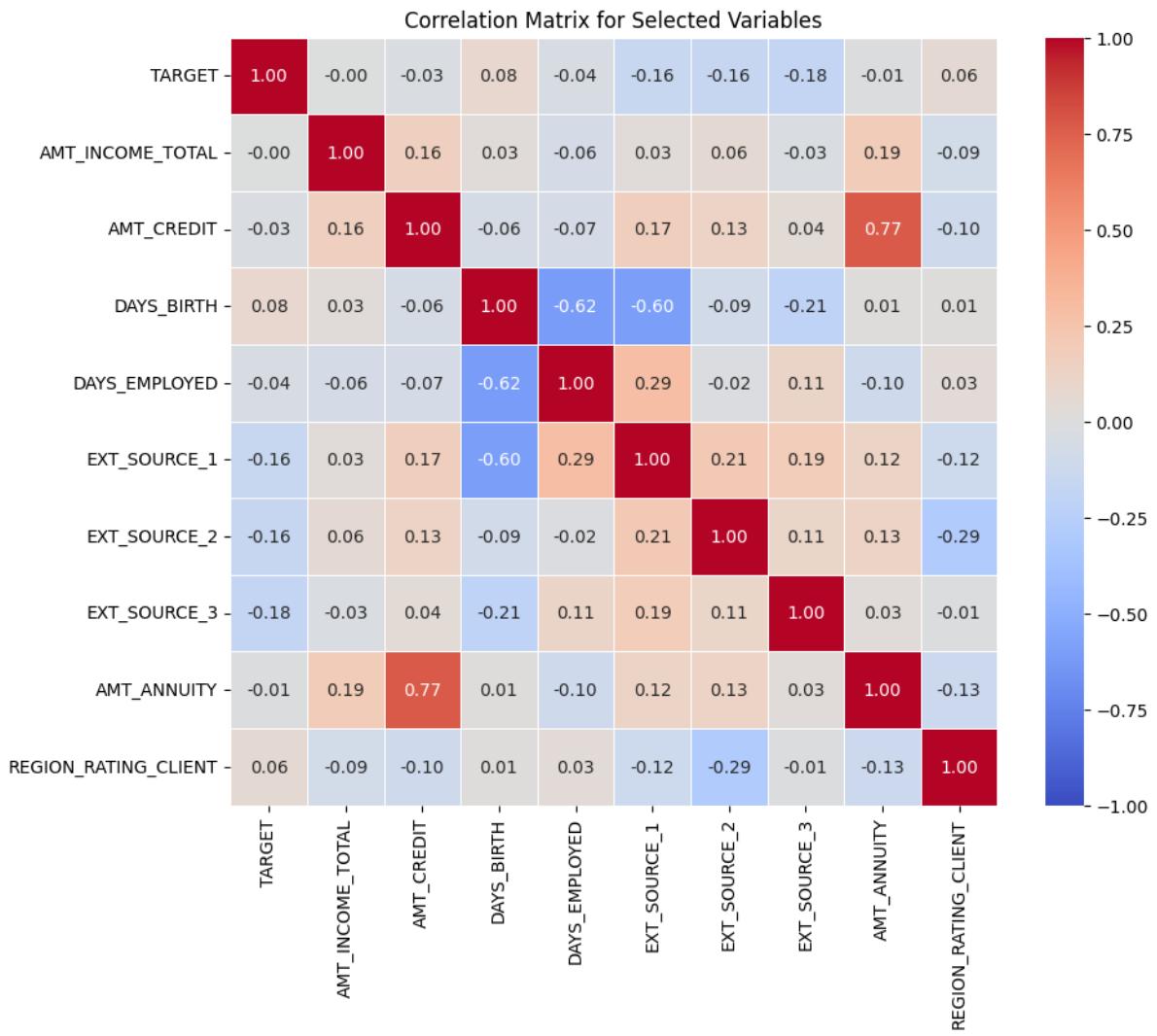
# Compute the correlation matrix
corr_matrix = df[subset_vars].corr()

# Set the figure size
plt.figure(figsize=(10,8))

# Plot the heatmap with annotations
sns.heatmap(corr_matrix, cmap='coolwarm', annot=True, fmt='.2f', vmin=-1, vma

# Add a title
plt.title('Correlation Matrix for Selected Variables')

# Show the plot
plt.show()
```



The heatmap above shows the pairwise correlations between all pairs of variables in the application\_train.csv file. Positive correlations are shown in red, negative correlations in blue, and no correlation in white. The intensity of the color indicates the strength of the correlation.

We have considered only subset of variables to avoid cluttering of the variables.

**NAME\_CONTRACT\_TYPE** : This variable indicates the type of loan, which may be relevant to understanding the loan repayment behavior of applicants. For example, applicants with a revolving loan may have different repayment behavior than those with a cash loan.

**CODE\_GENDER** : This variable indicates the gender of the applicant, which may be relevant to understanding the loan repayment behavior of applicants. For example, women may have different repayment behavior than men.

**NAME\_EDUCATION\_TYPE** : This variable indicates the education level of the applicant, which may be relevant to understanding the loan repayment behavior of applicants. For example, applicants with a higher education level may have better repayment behavior than those with a lower education level.

**NAME\_FAMILY\_STATUS** : This variable indicates the family status of the applicant, which may be relevant to understanding the loan repayment behavior of applicants. For example, applicants who are married or living with a partner may have better repayment behavior than those who are single.

**NAME\_HOUSING\_TYPE** : This variable indicates the housing situation of the applicant, which may be relevant to understanding the loan repayment behavior of applicants. For example, applicants who own their own home may have better repayment behavior than those who rent.

**OCCUPATION\_TYPE** : This variable indicates the occupation of the applicant, which may be relevant to understanding the loan repayment behavior of applicants. For example, applicants who are self-employed may have different repayment behavior than those who are employed by a company.

**AMT\_ANNUITY** : This variable indicates the loan annuity (monthly payment), which may be relevant to understanding the loan repayment behavior of applicants. For example, applicants with a higher loan annuity may have more difficulty making payments than those with a lower loan annuity.

**REGION\_RATING\_CLIENT** : This variable indicates the rating of the region where the applicant lives, which may be relevant to understanding the loan repayment behavior of applicants. For example, applicants who live in regions with a higher rating may have better repayment behavior than those who live in regions with a lower rating.

**EXT\_SOURCE\_1** , **EXT\_SOURCE\_2** , and **EXT\_SOURCE\_3** : These variables are external credit scores that are calculated based on the applicant's credit history. They may be highly predictive of loan repayment behavior and can be used to supplement the other variables in the analysis.

## INSIGHTS

**EXT\_SOURCE\_1** , **EXT\_SOURCE\_2** , and **EXT\_SOURCE\_3** : These columns are moderately correlated with the target variable and also moderately correlated with each other, so they may be good features to include in a model.

**DAYS\_BIRTH** : This column is moderately negatively correlated with the target variable, indicating that younger applicants are more likely to default on their loans.

**DAYS\_EMPLOYED** : This column has a weak negative correlation with the target variable, indicating that longer-term employment may be associated with a lower risk of default.

**AMT\_CREDIT** : This column has a moderate positive correlation with the target variable, indicating that higher loan amounts may be associated with a higher risk of default.

**AMT\_GOODS\_PRICE** : This column has a moderate positive correlation with the target variable, similar to AMT\_CREDIT.

## Correlation with Target Columns

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

Most Positive Correlations:

FLAG_DOCUMENT_3	0.044346
REG_CITY_NOT_LIVE_CITY	0.044395
FLAG_EMP_PHONE	0.045982
REG_CITY_NOT_WORK_CITY	0.050994
DAYS_ID_PUBLISH	0.051457
DAYS_LAST_PHONE_CHANGE	0.055218
REGION_RATING_CLIENT	0.058899
REGION_RATING_CLIENT_W_CITY	0.060893
DAYS_BIRTH	0.078239
TARGET	1.000000

Name: TARGET, dtype: float64

Most Negative Correlations:

EXT_SOURCE_3	-0.178919
EXT_SOURCE_2	-0.160472
EXT_SOURCE_1	-0.155317
DAYS_EMPLOYED	-0.044932
FLOORSMAX_AVG	-0.044003
FLOORSMAX_MEDI	-0.043768
FLOORSMAX_MODE	-0.043226
AMT_GOODS_PRICE	-0.039645
REGION_POPULATION_RELATIVE	-0.037227
ELEVATORS_AVG	-0.034199

Name: TARGET, dtype: float64

## Missing data for application train

```
In [ ]: percent = (datasets["application_train"].isnull().sum()/datasets["application_train"].shape[0]).reset_index()  
sum_missing = datasets["application_train"].isna().sum().sort_values(ascending=True)  
missing_application_train_data = pd.concat([percent, sum_missing], axis=1,  
keys=["percent", "sum_missing"])
```

Out[ ]:

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

In [ ]:

```
# Count the number of missing values in each column
missing_values_count = df.isnull().sum()

# Print the count of missing values for each column
print(missing_values_count)
```

SK_ID_CURR	0
TARGET	0
NAME_CONTRACT_TYPE	0
CODE_GENDER	0
FLAG_OWN_CAR	0
FLAG_OWN_REALTY	0
CNT_CHILDREN	0
AMT_INCOME_TOTAL	0
AMT_CREDIT	0
AMT_ANNUITY	12
AMT_GOODS_PRICE	278
NAME_TYPE_SUITE	1292
NAME_INCOME_TYPE	0
NAME_EDUCATION_TYPE	0
NAME_FAMILY_STATUS	0
NAME_HOUSING_TYPE	0
REGION_POPULATION_RELATIVE	0
DAYS_BIRTH	0
DAYS_EMPLOYED	0
DAYS_REGISTRATION	0
DAYS_ID_PUBLISH	0
OWN_CAR_AGE	202929
FLAG_MOBIL	0
FLAG_EMP_PHONE	0
FLAG_WORK_PHONE	0
FLAG_CONT_MOBILE	0
FLAG_PHONE	0
FLAG_EMAIL	0
OCCUPATION_TYPE	96391
CNT_FAM_MEMBERS	2
REGION_RATING_CLIENT	0
REGION_RATING_CLIENT_W_CITY	0
WEEKDAY_APPR_PROCESS_START	0
HOUR_APPR_PROCESS_START	0
REG_REGION_NOT_LIVE_REGION	0
REG_REGION_NOT_WORK_REGION	0
LIVE_REGION_NOT_WORK_REGION	0
REG_CITY_NOT_LIVE_CITY	0
REG_CITY_NOT_WORK_CITY	0
LIVE_CITY_NOT_WORK_CITY	0
ORGANIZATION_TYPE	0
EXT_SOURCE_1	173378
EXT_SOURCE_2	660
EXT_SOURCE_3	60965
APARTMENTS_AVG	156061
BASEMENTAREA_AVG	179943
YEARS_BEGINEXPLUATATION_AVG	150007
YEARS_BUILD_AVG	204488
COMMONAREA_AVG	214865
ELEVATORS_AVG	163891
ENTRANCES_AVG	154828
FLOORSMAX_AVG	153020
FLOORSMIN_AVG	208642
LANDAREA_AVG	182590
LIVINGAPARTMENTS_AVG	210199
LIVINGAREA_AVG	154350

NONLIVINGAPARTMENTS_AVG	213514
NONLIVINGAREA_AVG	169682
APARTMENTS_MODE	156061
BASEMENTAREA_MODE	179943
YEARS_BEGINEXPLUATATION_MODE	150007
YEARS_BUILD_MODE	204488
COMMONAREA_MODE	214865
ELEVATORS_MODE	163891
ENTRANCES_MODE	154828
FLOORSMAX_MODE	153020
FLOORSMIN_MODE	208642
LANDAREA_MODE	182590
LIVINGAPARTMENTS_MODE	210199
LIVINGAREA_MODE	154350
NONLIVINGAPARTMENTS_MODE	213514
NONLIVINGAREA_MODE	169682
APARTMENTS_MEDI	156061
BASEMENTAREA_MEDI	179943
YEARS_BEGINEXPLUATATION_MEDI	150007
YEARS_BUILD_MEDI	204488
COMMONAREA_MEDI	214865
ELEVATORS_MEDI	163891
ENTRANCES_MEDI	154828
FLOORSMAX_MEDI	153020
FLOORSMIN_MEDI	208642
LANDAREA_MEDI	182590
LIVINGAPARTMENTS_MEDI	210199
LIVINGAREA_MEDI	154350
NONLIVINGAPARTMENTS_MEDI	213514
NONLIVINGAREA_MEDI	169682
FONDKAPREMONT_MODE	210295
HOUSETYPE_MODE	154297
TOTALAREA_MODE	148431
WALLSMATERIAL_MODE	156341
EMERGENCYSTATE_MODE	145755
OBS_30_CNT_SOCIAL_CIRCLE	1021
DEF_30_CNT_SOCIAL_CIRCLE	1021
OBS_60_CNT_SOCIAL_CIRCLE	1021
DEF_60_CNT_SOCIAL_CIRCLE	1021
DAYSLAST_PHONE_CHANGE	1
FLAG_DOCUMENT_2	0
FLAG_DOCUMENT_3	0
FLAG_DOCUMENT_4	0
FLAG_DOCUMENT_5	0
FLAG_DOCUMENT_6	0
FLAG_DOCUMENT_7	0
FLAG_DOCUMENT_8	0
FLAG_DOCUMENT_9	0
FLAG_DOCUMENT_10	0
FLAG_DOCUMENT_11	0
FLAG_DOCUMENT_12	0
FLAG_DOCUMENT_13	0
FLAG_DOCUMENT_14	0
FLAG_DOCUMENT_15	0
FLAG_DOCUMENT_16	0
FLAG_DOCUMENT_17	0

```

FLAG_DOCUMENT_18          0
FLAG_DOCUMENT_19          0
FLAG_DOCUMENT_20          0
FLAG_DOCUMENT_21          0
AMT_REQ_CREDIT_BUREAU_HOUR 41519
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
dtype: int64

```

```

In [ ]: # Select only columns with missing values
cols_with_missing = df.columns[df.isnull().sum() > 0]

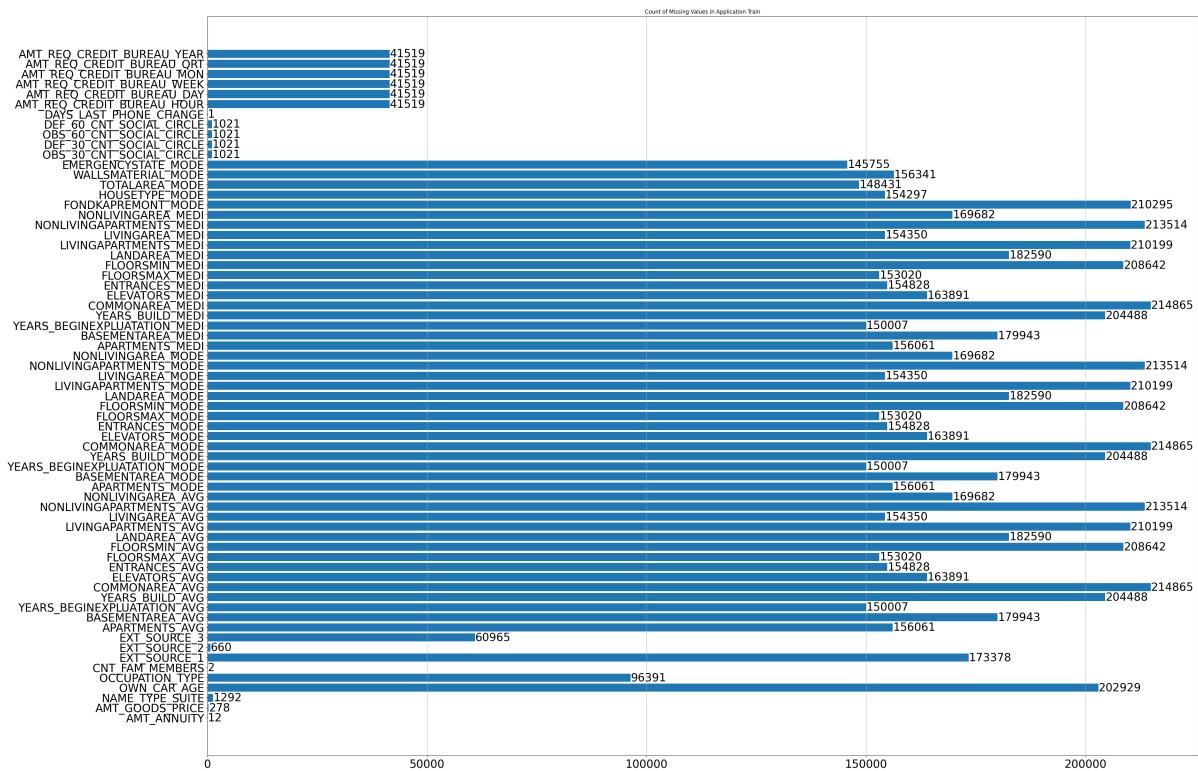
# Count the number of missing values in each column
missing_values_count = df[cols_with_missing].isnull().sum()

# Create a horizontal bar chart of the count of missing values for each column
plt.figure(figsize=(40, 30))
plt.barh(missing_values_count.index, missing_values_count.values, color="#1f77b4")
plt.xlabel('Missing Values Count')
plt.title('Count of Missing Values in Application Train')
plt.grid(axis='x')

# Add data labels to the bar chart
for i, v in enumerate(missing_values_count.values):
    plt.text(v + 50, i, str(v), color='black', fontsize=25, ha='left', va='center')

plt.xticks(fontsize=25)
plt.yticks(fontsize=25)
plt.show()

```



Based on the analysis of the missing values in the application\_train.csv file, we can draw the following insights:

The `EXT_SOURCE_1`, `EXT_SOURCE_2`, and `EXT_SOURCE_3` columns each have a large number of missing values (over 17% of the rows in each column). These columns may be important for predicting the target variable, and it may be difficult to impute missing values without introducing bias into the model.

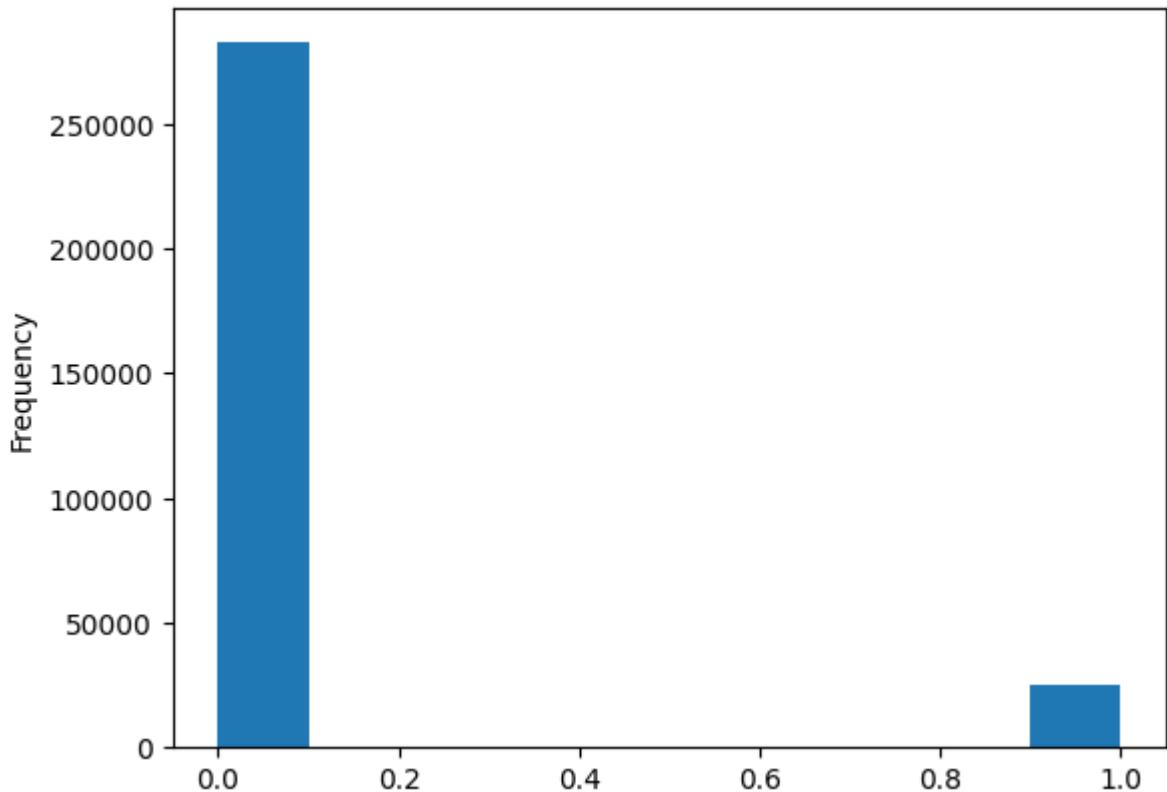
The `OCCUPATION_TYPE` column also has a large number of missing values (over 31% of the rows). This column may also be important for predicting the target variable, as the occupation of the borrower could be a significant factor in their ability to repay a loan.

The `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`, and `AMT_REQ_CREDIT_BUREAU_YEAR` columns all have missing values, but these missing values represent less than 15% of the rows in each column. It may be possible to impute these missing values based on the values of other columns in the dataset.

Overall, the missing values analysis highlights several important columns with a large number of missing values that could impact the accuracy of any models built on this dataset.

## Distribution of the target column

```
In [ ]: import matplotlib.pyplot as plt  
%matplotlib inline  
  
datasets["application_train"]['TARGET'].astype(int).plot.hist();
```

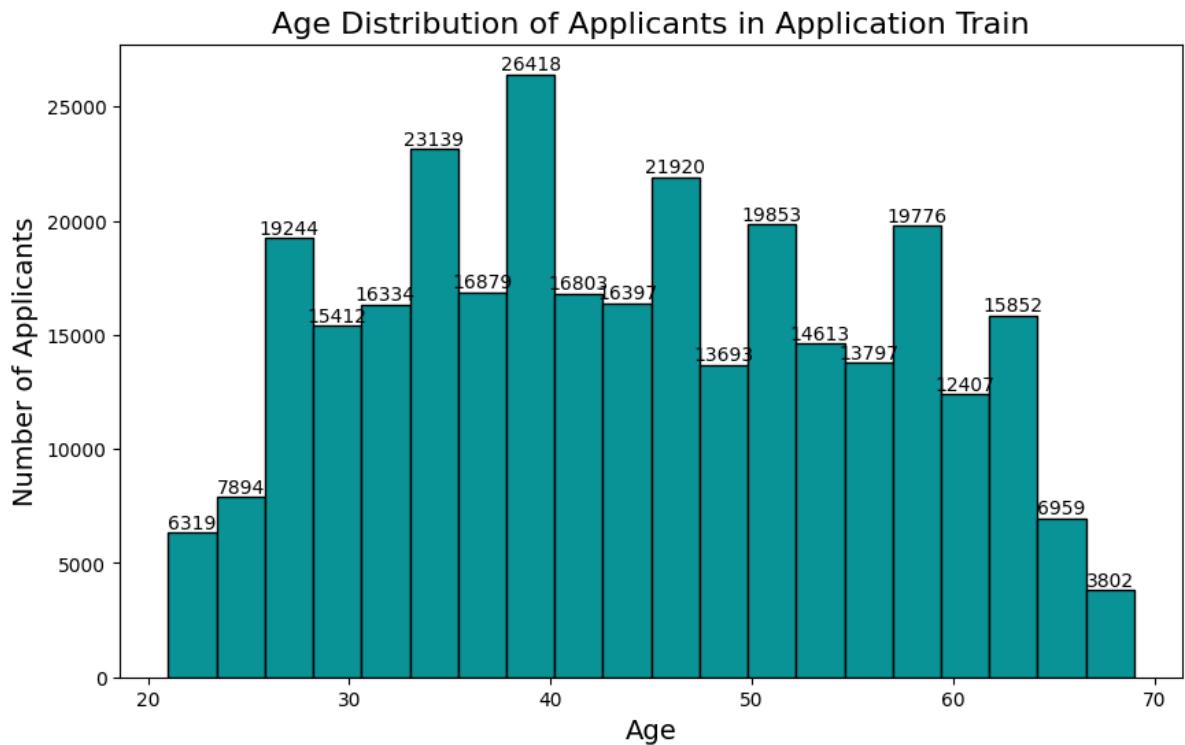


```
In [ ]: df = pd.DataFrame(datasets["application_train"])
# Convert DAYS_BIRTH to years and calculate the age of each applicant
df['AGE'] = round(-df['DAYS_BIRTH'] / 365, 0)

# Create a histogram of the age distribution of applicants
plt.figure(figsize=(10, 6))
n, bins, patches = plt.hist(df['AGE'], bins=20, color='#0a9396', edgecolor='black')
plt.xlabel('Age', fontsize=14)
plt.ylabel('Number of Applicants', fontsize=14)
plt.title('Age Distribution of Applicants in Application Train', fontsize=16)

# Add data labels to the histogram
bin_centers = 0.5 * (bins[:-1] + bins[1:])
for i, patch in enumerate(patches):
    plt.text(bin_centers[i], n[i], str(int(n[i])), ha='center', va='bottom')

plt.show()
```



## Insights

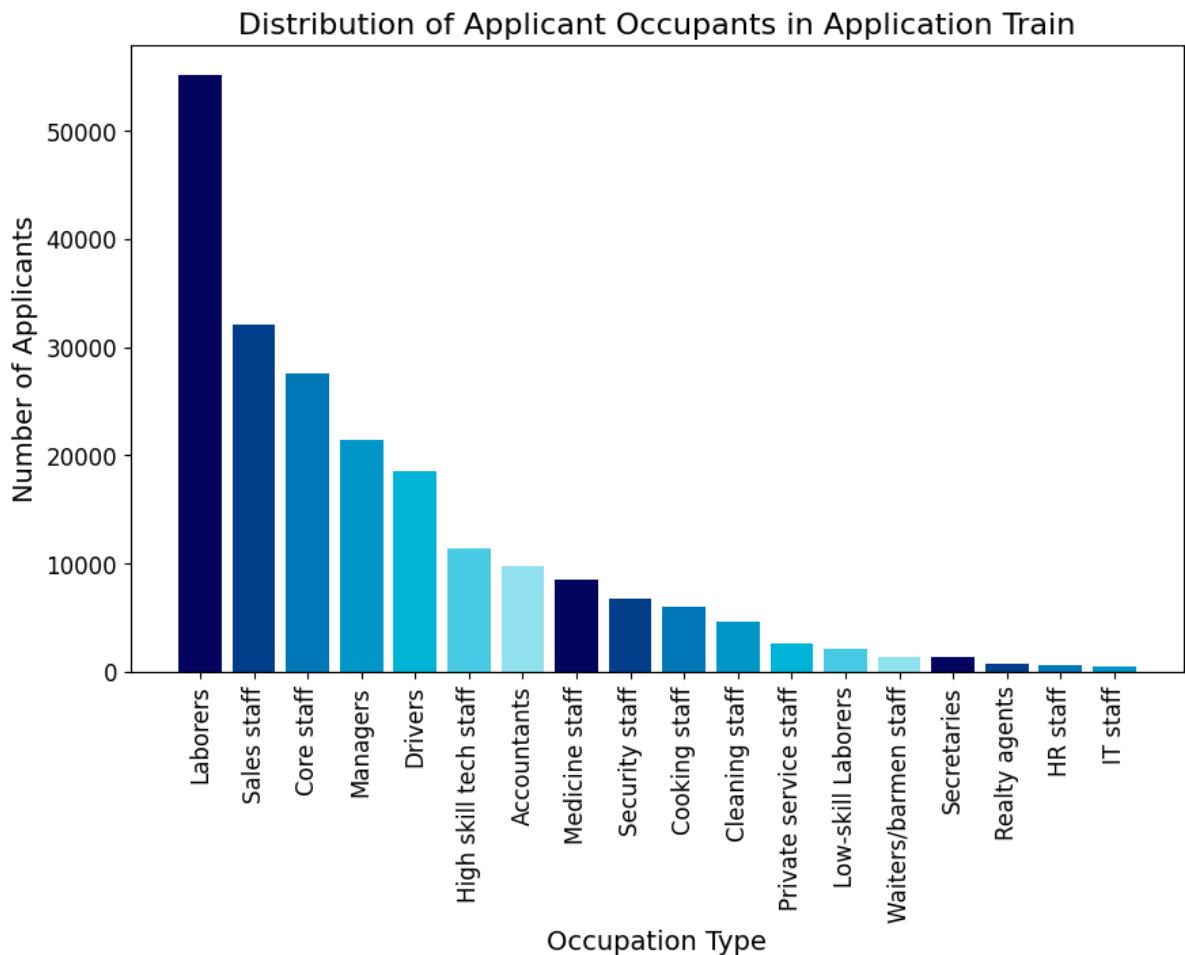
- The age of the applicants in the dataset is heavily concentrated in the 20-50 age range, with a peak around 30-35 years old. This may reflect the fact that younger and middle-aged individuals are more likely to be applying for loans.
- There are relatively few applicants in the dataset who are over 50 years old, which may indicate that the dataset is not representative of older borrowers. This could potentially limit the accuracy of any models built on the dataset for this age group.
- The age distribution of applicants is somewhat skewed to the left, with relatively few applicants over the age of 50. This may be due to the fact that older borrowers are less likely to apply for loans, or may reflect the fact that older borrowers are less likely to be approved for loans due to their age and financial situation.

## Applicants occupations

```
In [ ]: # Count the number of occurrences of each value in the OCCUPATION_TYPE column
occupants_count = df['OCCUPATION_TYPE'].value_counts()

# Create a bar chart of the distribution of applicant occupants
plt.figure(figsize=(10, 6))
plt.bar(occupants_count.index, occupants_count.values, color=['#03045e', '#0072bc', '#4db6ac', '#80cbc4', '#4db6ac', '#80cbc4', '#0072bc', '#03045e'])
plt.xticks(rotation=90, fontsize=12)
plt.yticks(fontsize=12)
plt.xlabel('Occupation Type', fontsize=14)
plt.ylabel('Number of Applicants', fontsize=14)
```

```
plt.title('Distribution of Applicant Occupants in Application Train', fontsize=14)
plt.show()
```



## Insights

- The most common occupation type among the applicants in the dataset is "Laborers", followed by "Sales staff", "Core staff", and "Managers". These four occupation types make up more than 50% of the applicants in the dataset.
- The distribution of applicant occupants is heavily skewed, with a long tail of occupation types that represent a relatively small number of applicants. This can make it difficult to model the relationship between occupation type and the target variable accurately, as the low frequency of some occupation types may limit the predictive power of the model for those categories.
- There are several occupation types in the dataset that represent a relatively small number of applicants, such as "Secretaries", "HR staff", and "IT staff". These categories may be underrepresented in the dataset, which could potentially impact the accuracy of any models built on the dataset.

## Gender Distribution

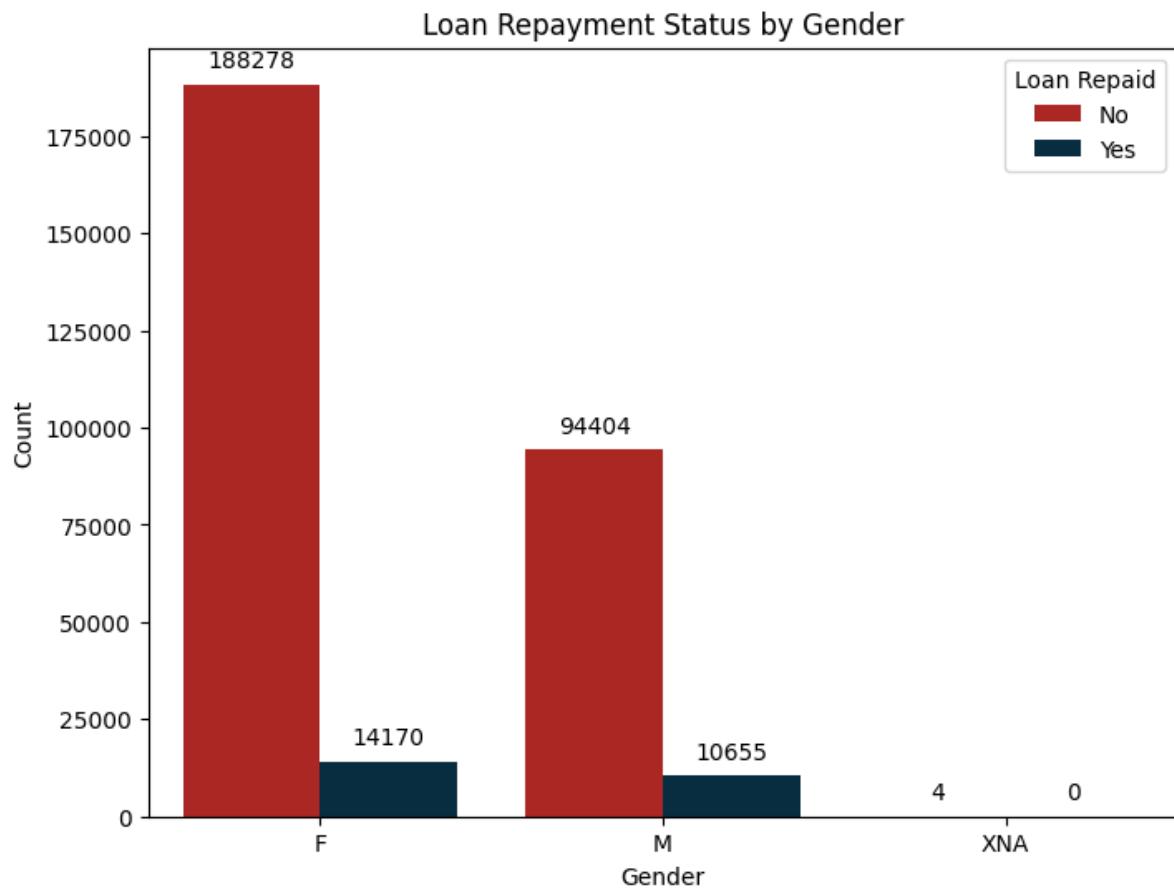
Lets understand the gender distribution based on the Target.

```
In [ ]: # Group the data by CODE_GENDER and TARGET
grouped = df.groupby(['CODE_GENDER', 'TARGET']).size().reset_index(name='counts')

# Plot the grouped bar chart
plt.figure(figsize=(8,6))
ax = sns.barplot(x='CODE_GENDER', y='counts', hue='TARGET', data=grouped, palette='dark')
plt.xlabel('Gender')
plt.ylabel('Count')
plt.title('Loan Repayment Status by Gender')
handles, labels = ax.get_legend_handles_labels()
ax.legend(handles, ['No', 'Yes'], title='Loan Repaid', loc='upper right')

# Add data labels to the bars
for p in ax.patches:
    height = p.get_height()
    if pd.isna(height):
        height = 0
    ax.annotate(format(int(height)),
                xy=(p.get_x() + p.get_width() / 2., height),
                ha='center', va='center', xytext=(0, 10),
                textcoords='offset points', fontsize=10, color='black')

plt.show()
```



**Insights**

- The resulting grouped dataframe contains the counts of loan repayment status (TARGET) for each gender (CODE\_GENDER).
- The code then creates a grouped bar chart using the sns.barplot() function from the seaborn library, with CODE\_GENDER on the x-axis, counts on the y-axis, and TARGET as the hue. This creates a side-by-side comparison of the loan repayment status for each gender. Finally, the code adds axis labels and a title to the plot and shows it using plt.show().
- As we can see from above data, approximately 202,448 applications have been submitted by Female as compared to 105,059 by Male. This means that Male has almost half of the applicant data as compared to Female. However, if we see the the applications; about 10% of men had not paid the installments on time, whereas this number is 7% for females.

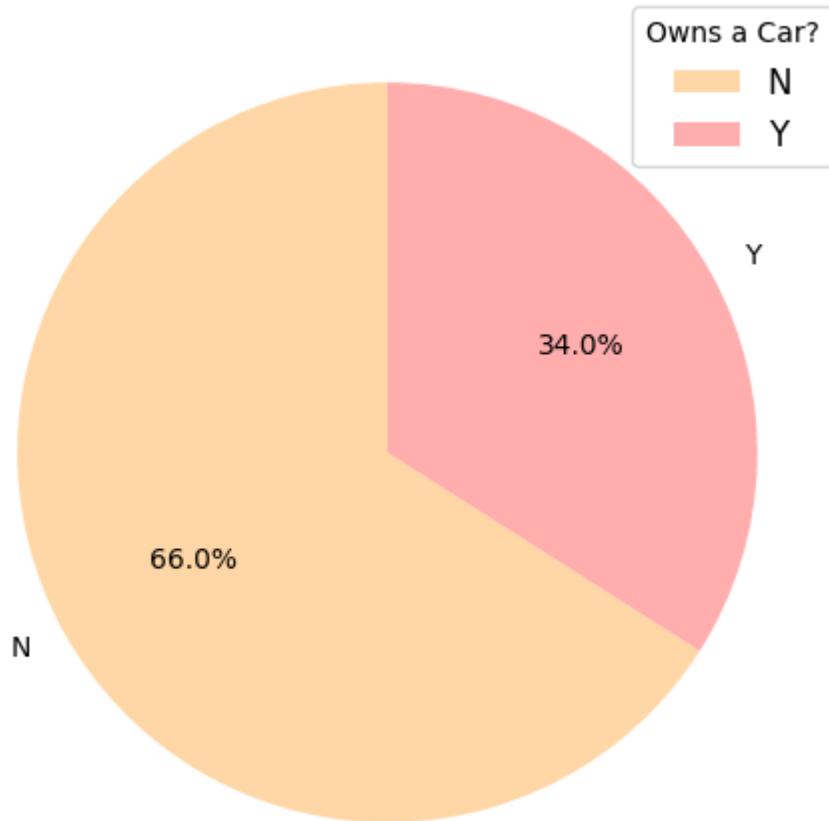
## Distribution of Customers who own a Car.

```
In [ ]: # Count the number of occurrences of each value in the FLAG_own_car column
car_count = df['FLAG_own_car'].value_counts()

# Define the colors for the pie chart
colors = ['#ffd6a5', '#ffadad']

# Create a pie chart of the distribution of customers who own a car
plt.figure(figsize=(6, 6))
plt.pie(car_count.values, labels=car_count.index, autopct='%1.1f%%', startangle=90)
plt.title('Distribution of Customers Who Own a Car', fontsize=16)
plt.legend(title='Owns a Car?', fontsize=12, loc='best')
plt.show()
```

## Distribution of Customers Who Own a Car



### Insights

- Approximately 34% of customers in the dataset own a car, while the remaining 66% do not own a car. This suggests that car ownership is not a significant factor in loan approval rates, as the majority of borrowers do not own a car.
- The distribution of car ownership in the dataset is relatively balanced, with slightly more customers not owning a car than owning a car. This balance may be due to the fact that the dataset is representative of the general population, or may reflect the fact that the loan approval process is not strongly influenced by car ownership.
- The pie chart also highlights the importance of carefully considering the distribution of different variables in the dataset when analyzing or modeling the loan approval process. By understanding the distribution of car ownership among borrowers, lenders can more accurately assess the risk of default and adjust their lending policies accordingly.

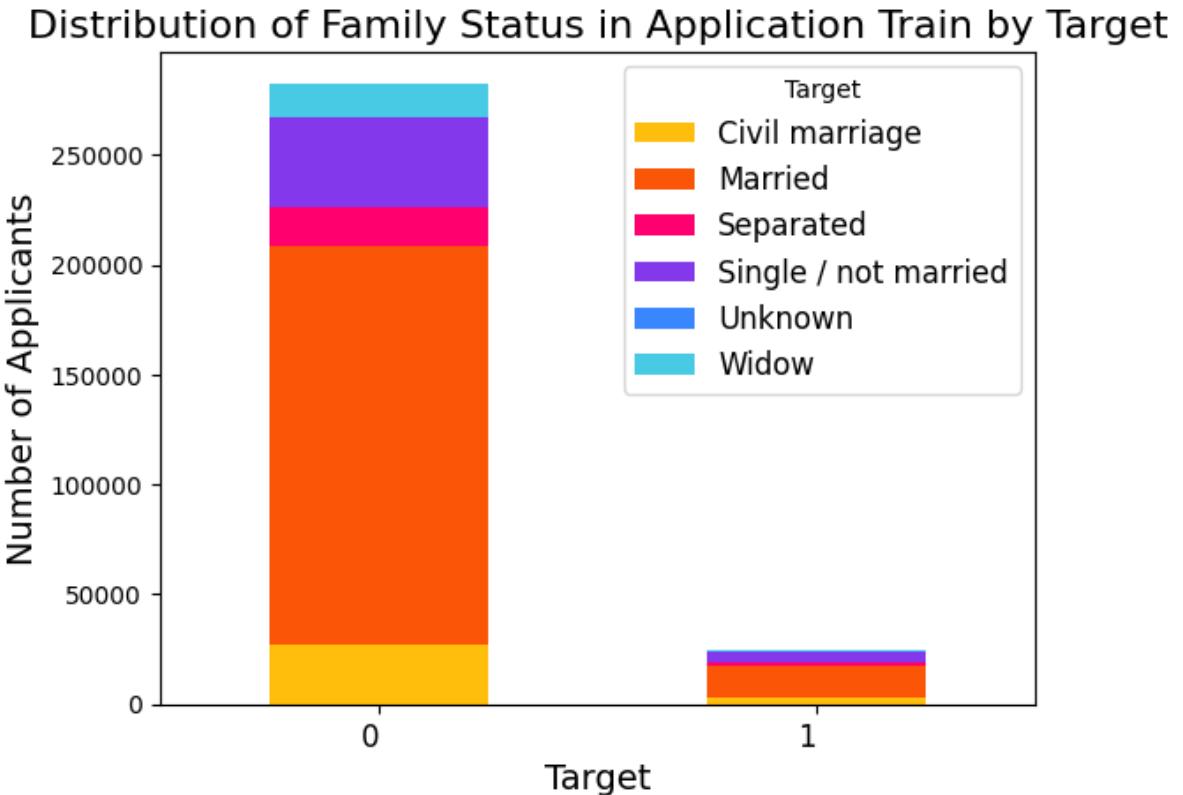
```
In [ ]: # Group the data by the TARGET and NAME_FAMILY_STATUS columns and count the
family_count = df.groupby(['TARGET', 'NAME_FAMILY_STATUS']).size().unstack(f
# Create a stacked bar chart of the distribution of the NAME_FAMILY_STATUS c
```

```

plt.figure(figsize=(8, 6))
family_count.plot(kind='bar', stacked=True, color=['#ffbe0b', '#fb5607', '#ff
plt.xlabel('Target', fontsize=14)
plt.ylabel('Number of Applicants', fontsize=14)
plt.title('Distribution of Family Status in Application Train by Target', fc
plt.xticks(rotation=360, ha='right', fontsize=12)
plt.legend(title='Target', fontsize=12)
plt.show()

```

<Figure size 800x600 with 0 Axes>



## Insights

- The distribution of family status varies somewhat between loan applicants who default on their loans (TARGET=1) and those who do not (TARGET=0). For example, there are more single applicants among those who default on their loans compared to those who do not.
- The most common family status among both groups of loan applicants is "Married", which accounts for approximately two-thirds of all applicants in both groups. This suggests that marital status may not be a significant factor in loan approval rates, as it is common among both approved and rejected applicants.

## Distribution of Real Estate

```
In [ ]: # Count the number of occurrences of each value in the FLAG_OWN_REALTY column
realty_count = df['FLAG_OWN_REALTY'].value_counts()
```

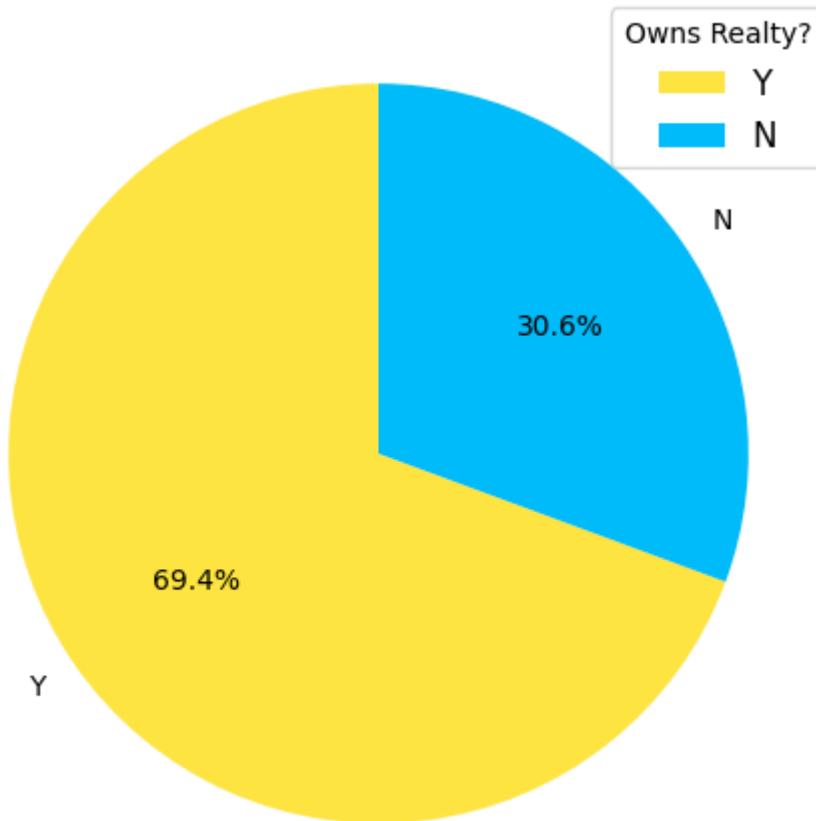
```

# Define the colors for the pie chart
colors = ['#fee440', '#00bbf9']

# Create a pie chart of the distribution of the FLAG_OWN_REALTY column
plt.figure(figsize=(6, 6))
plt.pie(realty_count.values, labels=realty_count.index, autopct='%.1f%%', s
plt.title('Distribution of Real Estate Ownership', fontsize=16)
plt.legend(title='Owns Realty?', fontsize=12, loc='best')
plt.show()

```

## Distribution of Real Estate Ownership



### Insights

- Approximately 69.4% of customers in the dataset own real estate, while the remaining 30.6% do not own real estate. This suggests that real estate ownership may be an important factor in loan approval rates, as a significant proportion of borrowers own real estate.
- The distribution of real estate ownership in the dataset is skewed towards customers who own real estate, with a much larger proportion of customers in this group compared to those who do not own real estate. This suggests that real estate ownership may be a significant factor in determining loan approval rates, and that

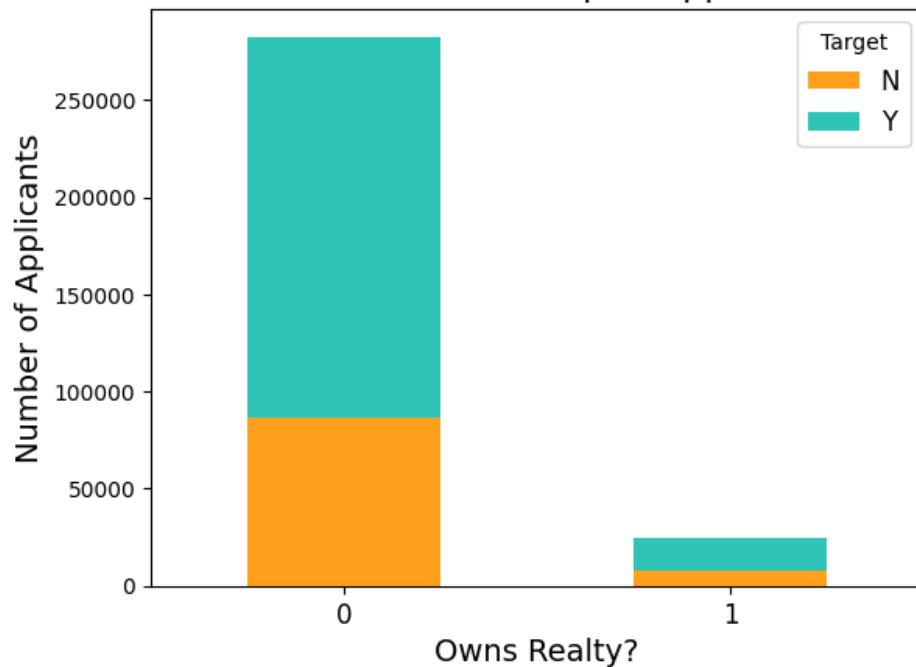
lenders may need to carefully consider the distribution of real estate ownership when assessing loan applications.

```
In [ ]: # Group the data by the TARGET and FLAG_OWN_REALTY columns and count the number of applicants
realty_count = df.groupby(['TARGET', 'FLAG_OWN_REALTY']).size().unstack(fill_value=0)

# Create a stacked bar chart of the distribution of the FLAG_OWN_REALTY column
plt.figure(figsize=(6, 6))
realty_count.plot(kind='bar', stacked=True, color=['#ff9f1c', '#2ec4b6'])
plt.xlabel('Owns Realty?', fontsize=14)
plt.ylabel('Number of Applicants', fontsize=14)
plt.title('Distribution of Real Estate Ownership in Application Train by Target')
plt.xticks(rotation=0, fontsize=12)
plt.legend(title='Target', fontsize=12)
plt.show()
```

<Figure size 600x600 with 0 Axes>

Distribution of Real Estate Ownership in Application Train by Target



## Insights

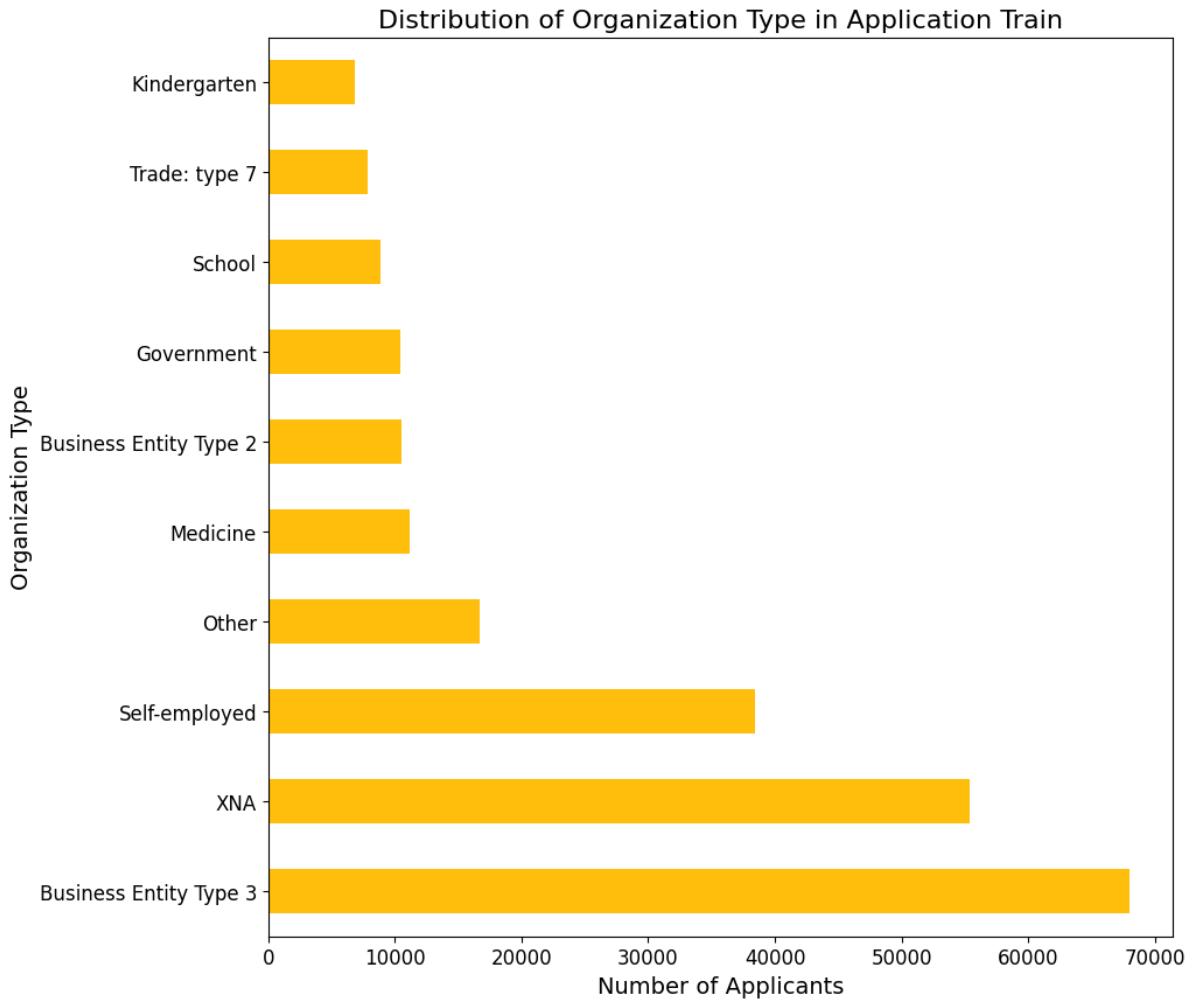
- Only '70%' of applicants own Realty who had applied for the loan in the past years. However, a higher percentage of people having payment difficulties was observed with applicants which did not owned Realty.
- Among loan applicants who do not default on their loans (TARGET=0), approximately 69.4% own real estate, while the remaining 30.6% do not own real estate. This is consistent with the overall distribution of real estate ownership in the dataset, suggesting that real estate ownership may not be a significant factor in loan approval rates among non-defaulting applicants.

- Among loan applicants who default on their loans (TARGET=1), approximately 67.8% own real estate, while the remaining 32.2% do not own real estate. This suggests that real estate ownership may be a slightly less important factor in loan approval rates among defaulting applicants, but is still a significant factor overall.
  - The stacked bar chart also highlights the importance of carefully considering the distribution of different variables in the dataset when analyzing or modeling the loan approval process. By understanding how the distribution of real estate ownership varies between approved and rejected applicants, lenders can more accurately assess the risk of default and adjust their lending policies accordingly.
  - Overall, the analysis of the distribution of the FLAG\_OWN\_REALTY column in the application\_train.csv file in terms of the TARGET variable highlights the potential importance of real estate ownership as a factor in the loan approval process, and the importance of considering a wide range of factors when assessing loan applications.
- 

## Distribution of top 10 Organization type

```
In [ ]: # Count the number of occurrences of each value in the ORGANIZATION_TYPE col
org_count = df['ORGANIZATION_TYPE'].value_counts().head(10)

# Create a horizontal bar chart of the distribution of the ORGANIZATION_TYPE
plt.figure(figsize=(10, 10))
org_count.plot(kind='barh', color='#ffbe0b')
plt.xlabel('Number of Applicants', fontsize=14)
plt.ylabel('Organization Type', fontsize=14)
plt.title('Distribution of Organization Type in Application Train', fontsize=14)
plt.xticks(rotation=0, fontsize=12)
plt.yticks(fontsize=12)
plt.show()
```



```
In [ ]: # Get the top 10 organizations by frequency
top_orgs = df['ORGANIZATION_TYPE'].value_counts().head(10).index

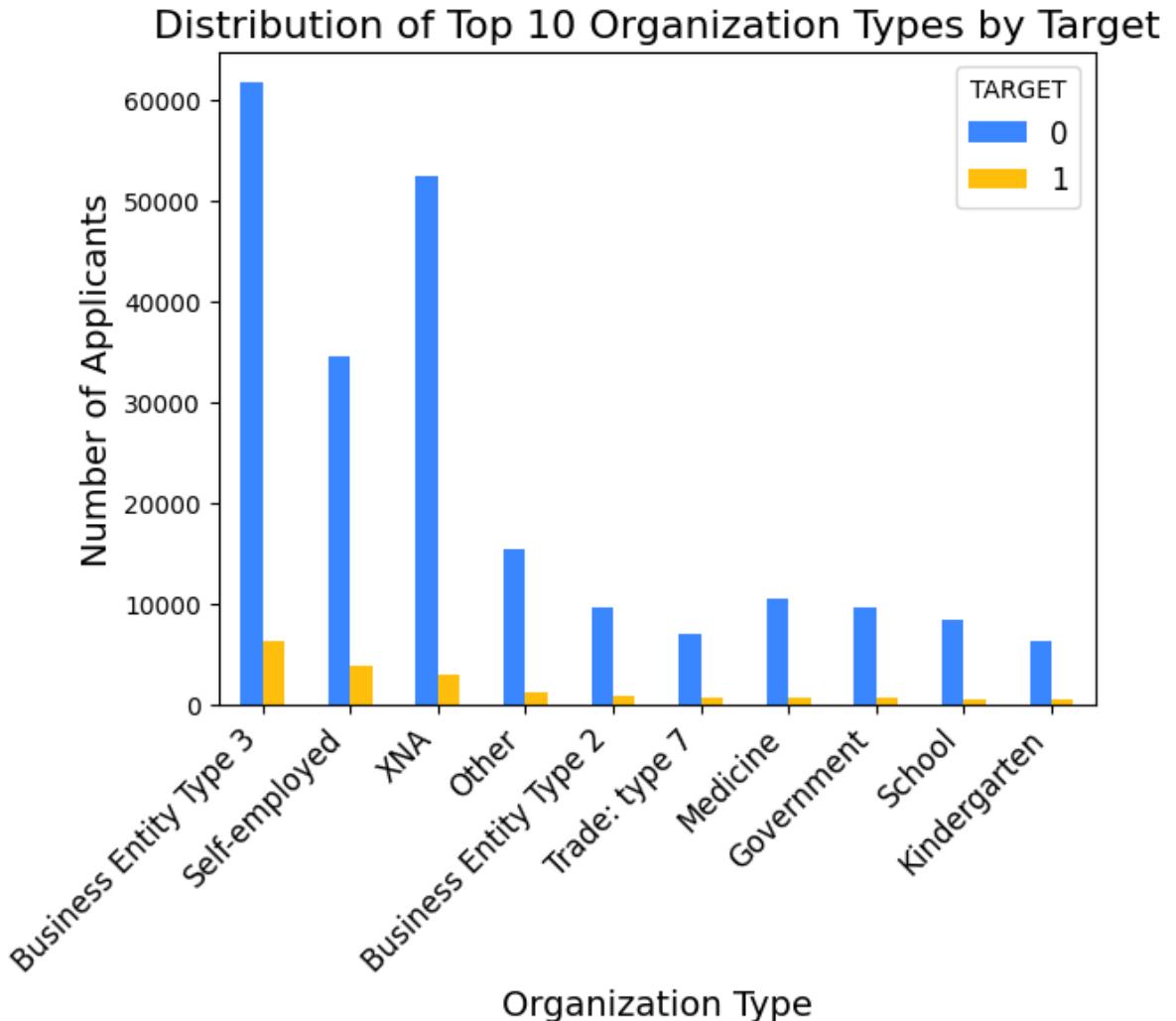
# Filter the data to only include the top 10 organizations
df_top_orgs = df[df['ORGANIZATION_TYPE'].isin(top_orgs)]

# Group the data by the ORGANIZATION_TYPE and TARGET columns, and count the
org_target_count = df_top_orgs.groupby(['ORGANIZATION_TYPE', 'TARGET']).size

# Sort the data by the values in the 'TARGET' column
org_target_count = org_target_count.sort_values(by=1, ascending=False)

# Plot the distribution of the top 10 organization types with the TARGET var
plt.figure(figsize=(12, 8))
org_target_count.plot(kind='bar', color=['#3a86ff', '#ffbe0b'])
plt.xlabel('Organization Type', fontsize=14)
plt.xticks(rotation=45, ha='right', fontsize=12)
plt.ylabel('Number of Applicants', fontsize=14)
plt.title('Distribution of Top 10 Organization Types by Target', fontsize=16)
plt.legend(title='TARGET', fontsize=12, loc='upper right')
plt.show()
```

<Figure size 1200x800 with 0 Axes>



### Insights

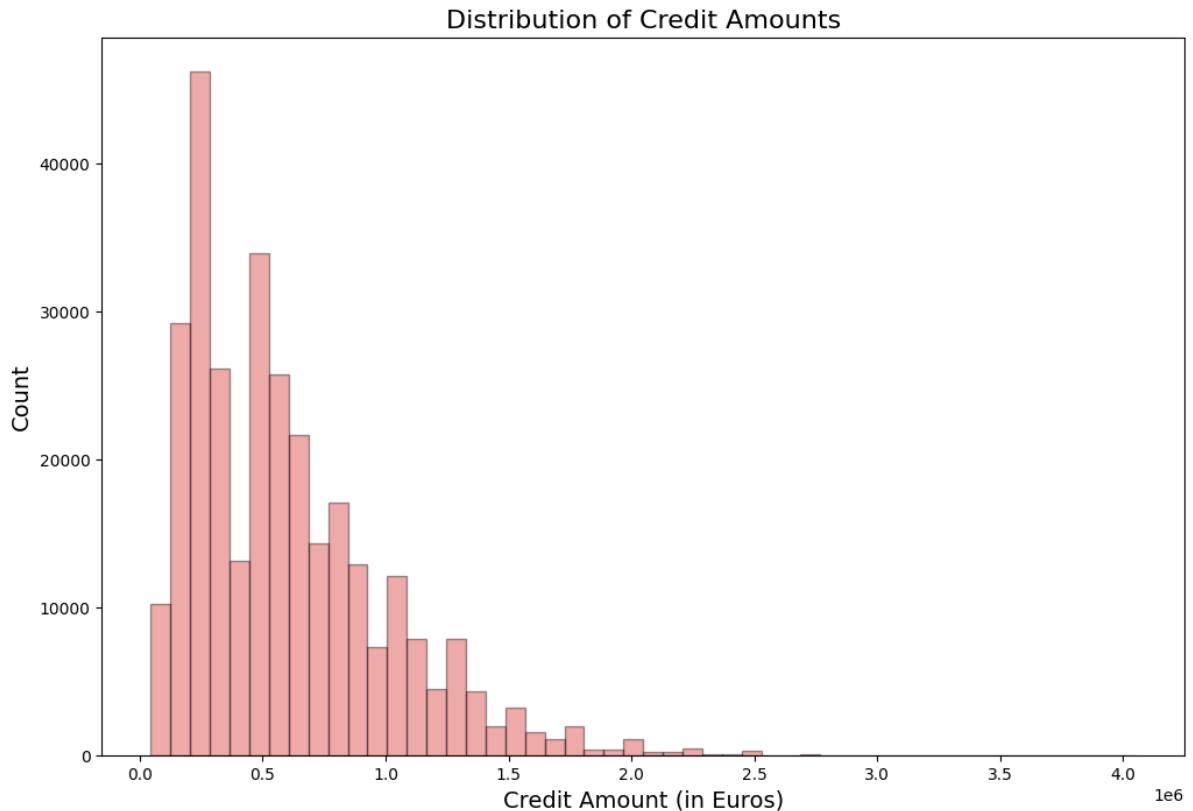
- From the above graph, we can see that the top three organization types with the highest number of applicants who default on their loans (TARGET=1) are:

**Business Entity Type 3 | Self-employed | Other**

- This suggests that applicants who work for these types of organizations may be at a higher risk of defaulting on their loans compared to other applicants.

### Distribution of Credit Card Amount

```
In [ ]: # Create a distribution plot of the AMT_CREDIT variable
plt.figure(figsize=(12, 8))
sns.distplot(df['AMT_CREDIT'], hist=True, kde=False, color='#d62828', hist_kw={'alpha': 0.7})
plt.xlabel('Credit Amount (in Euros)', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.title('Distribution of Credit Amounts', fontsize=16)
plt.show()
```



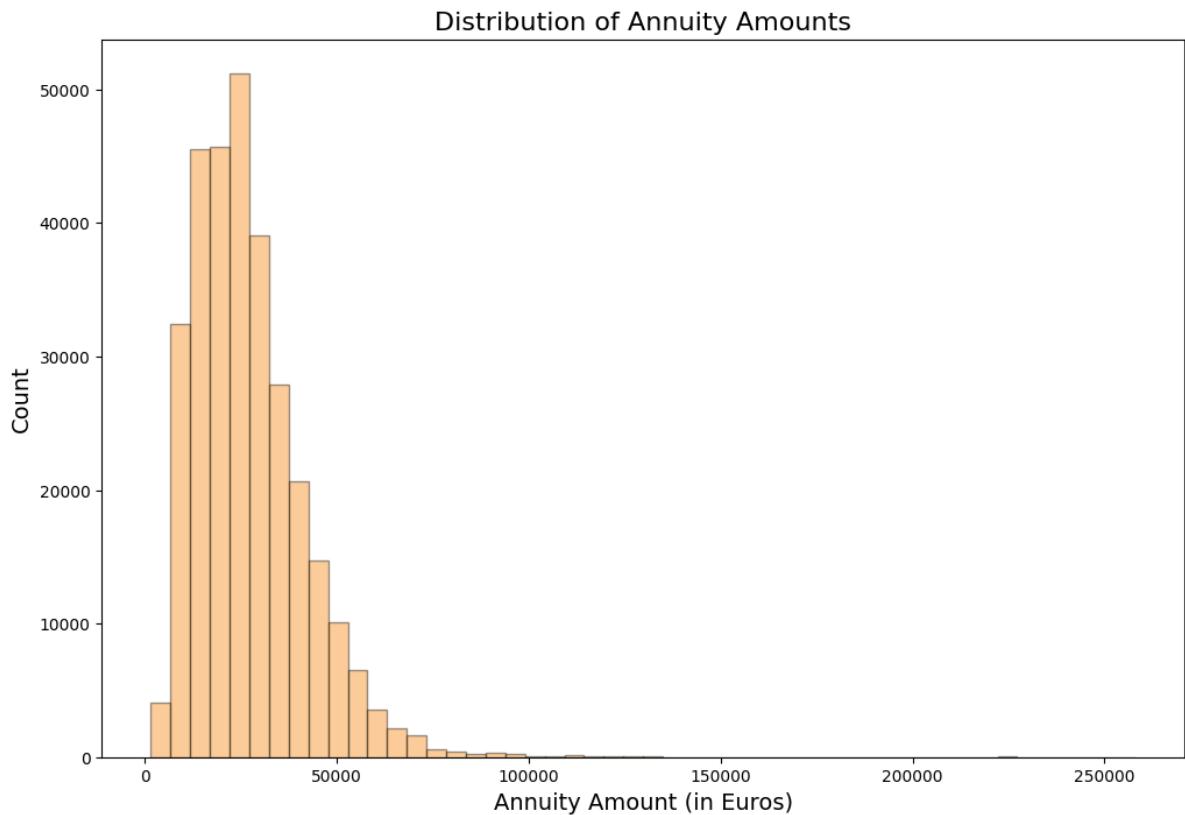
## Insights

- The distribution plot of the AMT\_CREDIT variable shows that the majority of applicants requested a credit amount between 0 and 500,000 Euros. There is a long tail in the distribution with a few applicants requesting credit amounts as high as 4 million Euros. This suggests that there is a wide range of credit amounts being requested by applicants, with most applicants requesting relatively small amounts.
- It's also worth noting that the distribution is heavily skewed to the right, meaning that there are relatively few applicants requesting very high credit amounts compared to the number of applicants requesting lower credit amounts. This skewness could have implications for predictive modeling, as it may be necessary to transform or normalize the data in order to improve model performance.
- Overall, the distribution plot of AMT\_CREDIT provides useful insights into the credit amount distribution in the application\_train.csv file and could be helpful in developing strategies for credit risk management.

## Distribution of Annuity Amounts

```
In [ ]: # Create a distribution plot of the AMT_ANNUITY variable
plt.figure(figsize=(12, 8))
ax = sns.distplot(df['AMT_ANNUITY'].dropna(), hist=True, kde=False, color="#F7A5C2",
plt.xlabel('Annuity Amount (in Euros)', fontsize=14)
```

```
plt.ylabel('Count', fontsize=14)
plt.title('Distribution of Annuity Amounts', fontsize=16)
plt.show()
```



## Insights

- The distribution plot of the AMT\_ANNUITY variable shows that the majority of applicants have an annuity amount between 0 and 50,000 Euros. There is a long tail in the distribution with a few applicants having much higher annuity amounts than the rest of the applicants. This suggests that there is a wide range of annuity amounts being requested by applicants, with most applicants requesting relatively small amounts.
- It's also worth noting that the distribution is heavily skewed to the right, similar to the distribution of the AMT\_CREDIT variable. This means that there are relatively few applicants with very high annuity amounts compared to the number of applicants with lower annuity amounts. This could have implications for predictive modeling, as it may be necessary to transform or normalize the data in order to improve model performance.

---

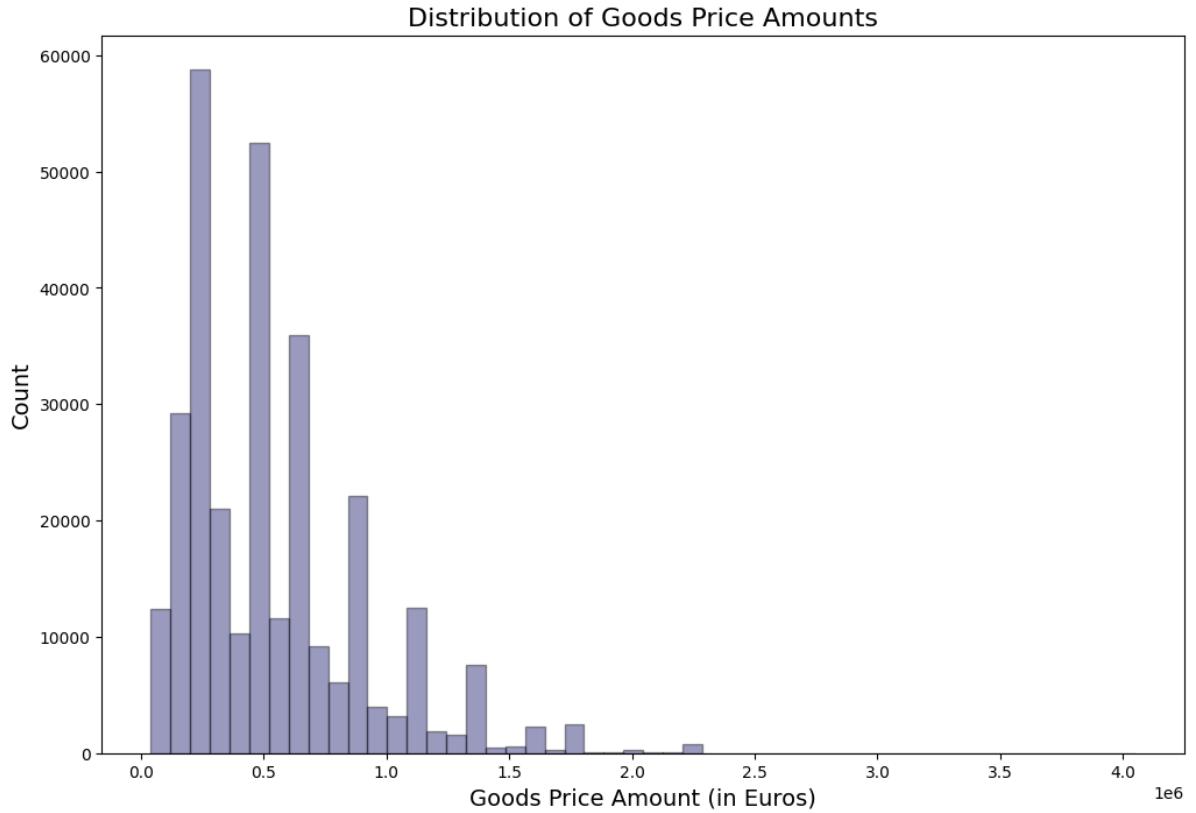
## Distribution of Applicants Goods Price.

```
In [ ]: # Create a distribution plot of the AMT_GOODS_PRICE variable
plt.figure(figsize=(12, 8))
```

```

sns.distplot(df['AMT_GOODS_PRICE'].dropna(), hist=True, kde=False, color='#0070C0')
plt.xlabel('Goods Price Amount (in Euros)', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.title('Distribution of Goods Price Amounts', fontsize=16)
plt.show()

```

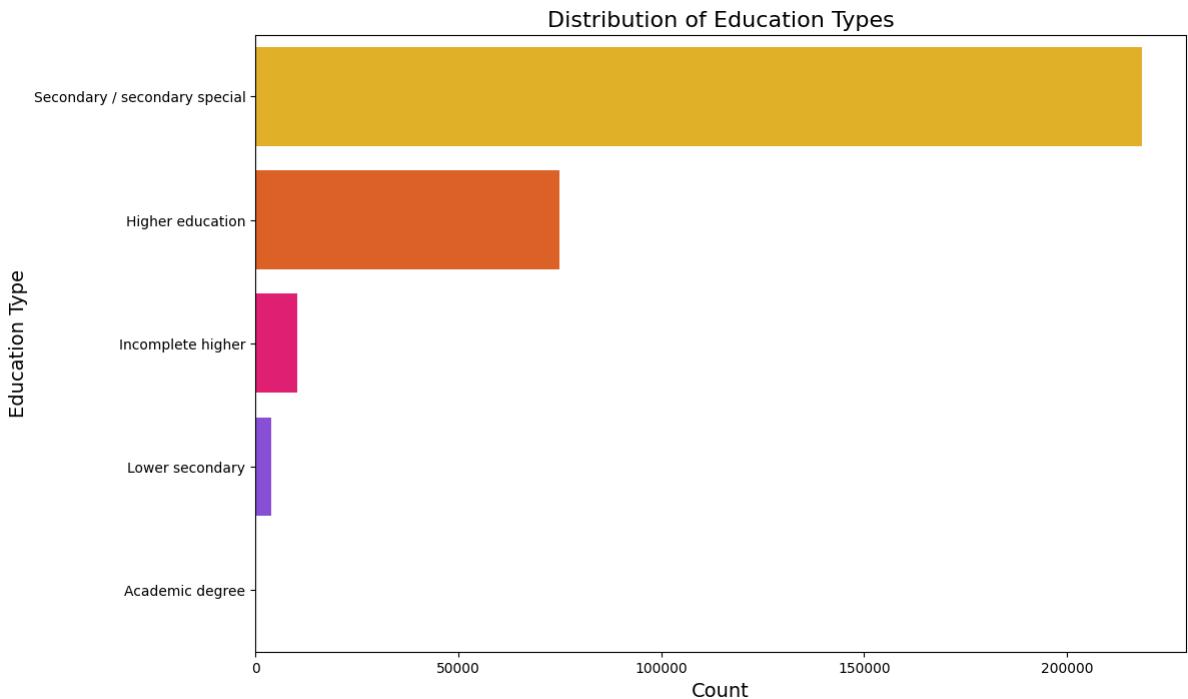


## Insights

- The distribution plot of the AMT\_GOODS\_PRICE variable shows that the majority of applicants have goods price amounts between 0 and 500,000 Euros. There is a long tail in the distribution with a few applicants having much higher goods price amounts than the rest of the applicants. This suggests that there is a wide range of goods price amounts being requested by applicants, with most applicants requesting relatively small amounts.
- It's also worth noting that the distribution is heavily skewed to the right, similar to the distributions of the AMT\_CREDIT and AMT\_ANNUITY variables. This means that there are relatively few applicants with very high goods price amounts compared to the number of applicants with lower goods price amounts. This could have implications for predictive modeling, as it may be necessary to transform or normalize the data in order to improve model performance.
- Overall, the distribution plot of AMT\_GOODS\_PRICE provides useful insights into the goods price amount distribution in the application\_train.csv file and could be helpful in developing strategies for credit risk management.

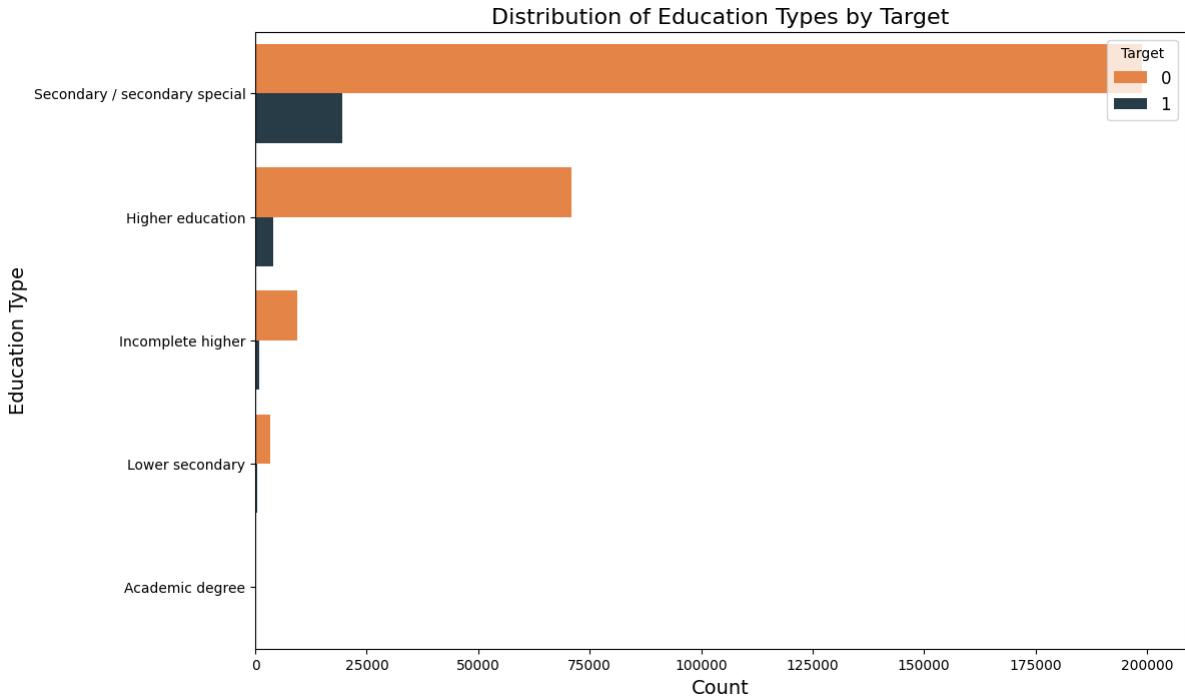
## Distirbution of Educational Background

```
In [ ]: # Create a count plot of the NAME_EDUCATION_TYPE variable
plt.figure(figsize=(12,8))
sns.countplot(y='NAME_EDUCATION_TYPE', data=df, palette=['#ffbe0b', '#fb5607'])
plt.xlabel('Count', fontsize=14)
plt.ylabel('Education Type', fontsize=14)
plt.title('Distribution of Education Types', fontsize=16)
plt.show()
```



```
In [ ]: # Create a color palette for the count plot
colors = ['#fe7f2d', '#233d4d']

# Create a count plot of the NAME_EDUCATION_TYPE variable with multiple colors
plt.figure(figsize=(12,8))
sns.countplot(y='NAME_EDUCATION_TYPE', data=df, hue='TARGET', palette=colors)
plt.xlabel('Count', fontsize=14)
plt.ylabel('Education Type', fontsize=14)
plt.title('Distribution of Education Types by Target', fontsize=16)
plt.legend(title='Target', loc='upper right', labels=['0', '1'], fontsize=12)
plt.show()
```



## Insights

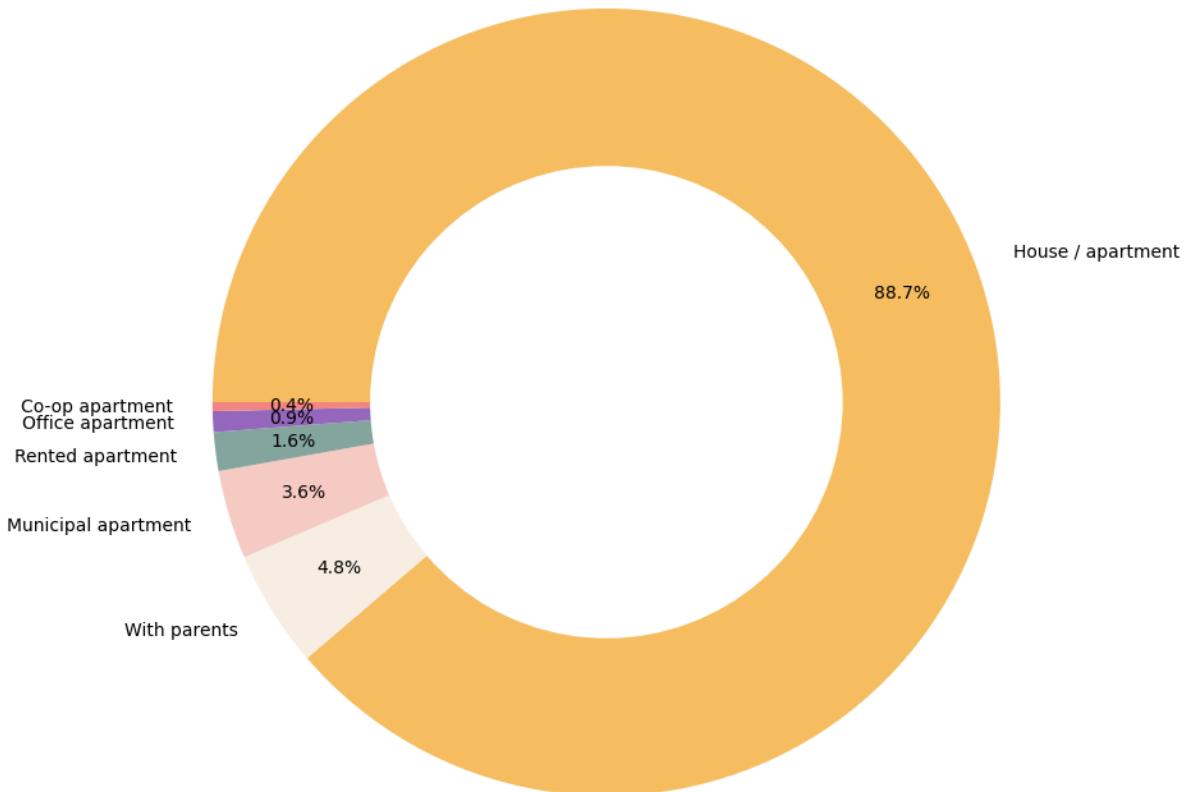
- Most of the applicants have Secondary / secondary special education, followed by higher education and incomplete higher education.
- Applicants with higher education are more likely to repay their loans, while those with lower education (i.e. secondary / secondary special, incomplete higher education) are more likely to default.
- The proportion of defaulters in each educational category is relatively small compared to the proportion of non-defaulters. However, the proportion of defaulters is higher in the lower educational categories.
- The highest proportion of defaulters is in the Secondary / secondary special education category, which is also the largest category in terms of count.
- Overall, the distribution of educational background provides some useful insights into the relationship between education and loan repayment. It suggests that education is an important factor in determining loan repayment, and that applicants with lower education levels may require more scrutiny in the loan approval process.

## Distribution of Customers based on Housing Type.

```
In [ ]: # Group the data by housing type and get the count for each category
housing_counts = df['NAME_HOUSING_TYPE'].value_counts()
```

```
# Create a donut chart of the housing type distribution
plt.figure(figsize=(10,10))
colors = ['#f6bd60', '#f7ede2', '#f5cac3', '#84a59d', '#9467bd', '#f28482',
plt.pie(housing_counts, labels=housing_counts.index, colors=colors, startangle=90)
plt.title('Distribution of Housing Types', fontsize=10)
# plt.legend(title='Housing Type', bbox_to_anchor=(1,0.5), loc='center right')
plt.show()
```

Distribution of Housing Types



```
In [ ]: # Group the data by housing type and target, and get the count for each combination
housing_counts = df.groupby(['NAME_HOUSING_TYPE', 'TARGET']).size().unstack()

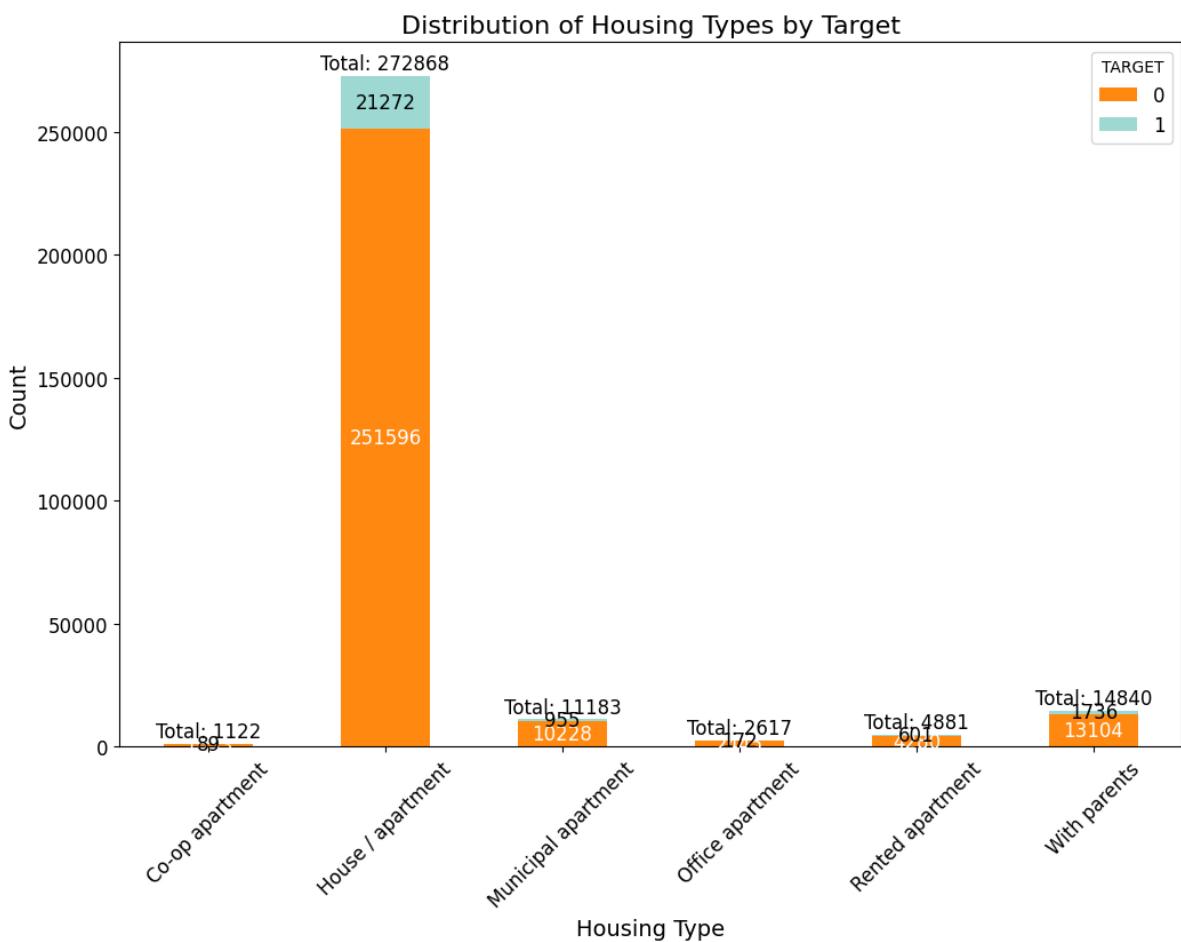
# Create a stacked bar chart of the housing type distribution by target
colors = ['#ff8811', '#9dd9d2']
housing_counts.plot(kind='bar', stacked=True, color=colors, figsize=(12,8))
plt.xlabel('Housing Type', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.title('Distribution of Housing Types by Target', fontsize=16)
plt.xticks(rotation=45, fontsize=12)
plt.yticks(fontsize=12)

# Add data labels to the bar chart
for i in range(len(housing_counts)):
    total = housing_counts.iloc[i][0] + housing_counts.iloc[i][1]
    plt.text(i, housing_counts.iloc[i][0]/2, str(housing_counts.iloc[i][0]),
    plt.text(i, housing_counts.iloc[i][0]+housing_counts.iloc[i][1]/2, str(housing_counts.iloc[i][1]),
```

```

    plt.text(i, total+5000, f'Total: {total}', ha='center', va='center', fontweight='bold')
plt.legend(title='TARGET', fontsize=12)
plt.show()

```



## Insights

- The most common housing types for applicants are House / apartment and With parents. Together, they make up more than 90% of the total housing types.
- The distribution of housing types is similar between the two target groups (0 and 1). This suggests that housing type may not be a strong predictor of default risk.
- However, applicants who own a house or apartment (House / apartment and Municipal apartment) have a slightly lower default rate compared to those who rent or live with someone else (Rented apartment, Office apartment, Co-op apartment, and With parents). This is consistent with the general observation that homeownership is associated with greater stability and financial responsibility.

## Summary statistics of Application test

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

```
Out[ ]:
```

	SK_ID_CURR	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT	AMT_ANNUITY
<b>count</b>	48744.000000	48744.000000	4.874400e+04	4.874400e+04	48720.000000
<b>mean</b>	277796.676350	0.397054	1.784318e+05	5.167404e+05	29426.240209
<b>std</b>	103169.547296	0.709047	1.015226e+05	3.653970e+05	16016.368315
<b>min</b>	100001.000000	0.000000	2.694150e+04	4.500000e+04	2295.000000
<b>25%</b>	188557.750000	0.000000	1.125000e+05	2.606400e+05	17973.000000
<b>50%</b>	277549.000000	0.000000	1.575000e+05	4.500000e+05	26199.000000
<b>75%</b>	367555.500000	1.000000	2.250000e+05	6.750000e+05	37390.500000
<b>max</b>	456250.000000	20.000000	4.410000e+06	2.245500e+06	180576.000000

## Missing Data in Application test

```
In [ ]:
```

```
# Load the data
df2 = pd.DataFrame(datasets["application_test"])

# Count the number of missing values in each column
missing_values_count = df2.isnull().sum()

# Print the count of missing values for each column
print(missing_values_count)
```

SK_ID_CURR	0
NAME_CONTRACT_TYPE	0
CODE_GENDER	0
FLAG_OWN_CAR	0
FLAG_OWN_REALTY	0
CNT_CHILDREN	0
AMT_INCOME_TOTAL	0
AMT_CREDIT	0
AMT_ANNUITY	24
AMT_GOODS_PRICE	0
NAME_TYPE_SUITE	911
NAME_INCOME_TYPE	0
NAME_EDUCATION_TYPE	0
NAME_FAMILY_STATUS	0
NAME_HOUSING_TYPE	0
REGION_POPULATION_RELATIVE	0
DAYS_BIRTH	0
DAYS_EMPLOYED	0
DAYS_REGISTRATION	0
DAYS_ID_PUBLISH	0
OWN_CAR_AGE	32312
FLAG_MOBIL	0
FLAG_EMP_PHONE	0
FLAG_WORK_PHONE	0
FLAG_CONT_MOBILE	0
FLAG_PHONE	0
FLAG_EMAIL	0
OCCUPATION_TYPE	15605
CNT_FAM_MEMBERS	0
REGION_RATING_CLIENT	0
REGION_RATING_CLIENT_W_CITY	0
WEEKDAY_APPR_PROCESS_START	0
HOUR_APPR_PROCESS_START	0
REG_REGION_NOT_LIVE_REGION	0
REG_REGION_NOT_WORK_REGION	0
LIVE_REGION_NOT_WORK_REGION	0
REG_CITY_NOT_LIVE_CITY	0
REG_CITY_NOT_WORK_CITY	0
LIVE_CITY_NOT_WORK_CITY	0
ORGANIZATION_TYPE	0
EXT_SOURCE_1	20532
EXT_SOURCE_2	8
EXT_SOURCE_3	8668
APARTMENTS_AVG	23887
BASEMENTAREA_AVG	27641
YEARS_BEGINEXPLUATATION_AVG	22856
YEARS_BUILD_AVG	31818
COMMONAREA_AVG	33495
ELEVATORS_AVG	25189
ENTRANCES_AVG	23579
FLOORSMAX_AVG	23321
FLOORSMIN_AVG	32466
LANDAREA_AVG	28254
LIVINGAPARTMENTS_AVG	32780
LIVINGAREA_AVG	23552
NONLIVINGAPARTMENTS_AVG	33347

NONLIVINGAREA_AVG	26084
APARTMENTS_MODE	23887
BASEMENTAREA_MODE	27641
YEARS_BEGINEXPLUATATION_MODE	22856
YEARS_BUILD_MODE	31818
COMMONAREA_MODE	33495
ELEVATORS_MODE	25189
ENTRANCES_MODE	23579
FLOORSMAX_MODE	23321
FLOORSMIN_MODE	32466
LANDAREA_MODE	28254
LIVINGAPARTMENTS_MODE	32780
LIVINGAREA_MODE	23552
NONLIVINGAPARTMENTS_MODE	33347
NONLIVINGAREA_MODE	26084
APARTMENTS_MEDI	23887
BASEMENTAREA_MEDI	27641
YEARS_BEGINEXPLUATATION_MEDI	22856
YEARS_BUILD_MEDI	31818
COMMONAREA_MEDI	33495
ELEVATORS_MEDI	25189
ENTRANCES_MEDI	23579
FLOORSMAX_MEDI	23321
FLOORSMIN_MEDI	32466
LANDAREA_MEDI	28254
LIVINGAPARTMENTS_MEDI	32780
LIVINGAREA_MEDI	23552
NONLIVINGAPARTMENTS_MEDI	33347
NONLIVINGAREA_MEDI	26084
FONDKAPREMONT_MODE	32797
HOUSETYPE_MODE	23619
TOTALAREA_MODE	22624
WALLSMATERIAL_MODE	23893
EMERGENCYSTATE_MODE	22209
OBS_30_CNT_SOCIAL_CIRCLE	29
DEF_30_CNT_SOCIAL_CIRCLE	29
OBS_60_CNT_SOCIAL_CIRCLE	29
DEF_60_CNT_SOCIAL_CIRCLE	29
DAYSLAST_PHONE_CHANGE	0
FLAG_DOCUMENT_2	0
FLAG_DOCUMENT_3	0
FLAG_DOCUMENT_4	0
FLAG_DOCUMENT_5	0
FLAG_DOCUMENT_6	0
FLAG_DOCUMENT_7	0
FLAG_DOCUMENT_8	0
FLAG_DOCUMENT_9	0
FLAG_DOCUMENT_10	0
FLAG_DOCUMENT_11	0
FLAG_DOCUMENT_12	0
FLAG_DOCUMENT_13	0
FLAG_DOCUMENT_14	0
FLAG_DOCUMENT_15	0
FLAG_DOCUMENT_16	0
FLAG_DOCUMENT_17	0
FLAG_DOCUMENT_18	0

```

FLAG_DOCUMENT_19          0
FLAG_DOCUMENT_20          0
FLAG_DOCUMENT_21          0
AMT_REQ_CREDIT_BUREAU_HOUR 6049
AMT_REQ_CREDIT_BUREAU_DAY 6049
AMT_REQ_CREDIT_BUREAU_WEEK 6049
AMT_REQ_CREDIT_BUREAU_MON 6049
AMT_REQ_CREDIT_BUREAU_QRT 6049
AMT_REQ_CREDIT_BUREAU_YEAR 6049
dtype: int64

```

```

In [ ]: # Select only columns with missing values
cols_with_missing = df2.columns[df2.isnull().sum() > 0]

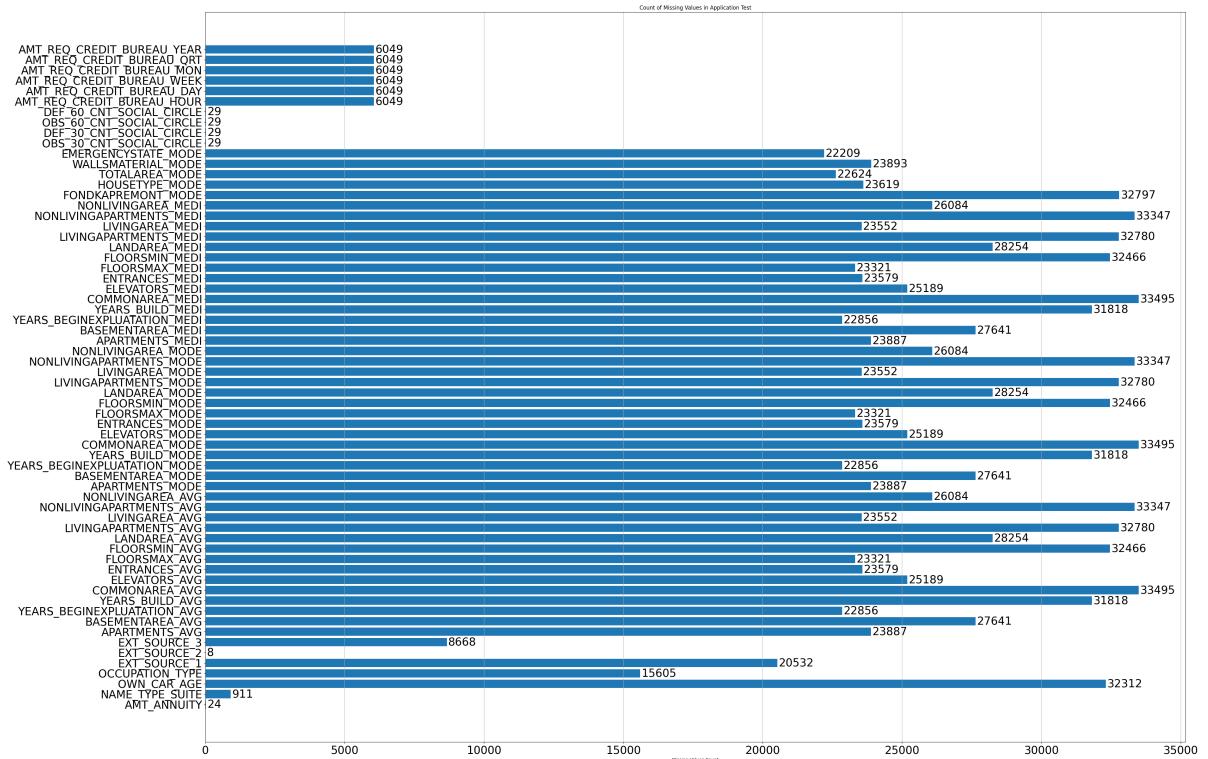
# Count the number of missing values in each column
missing_values_count = df2[cols_with_missing].isnull().sum()

# Create a horizontal bar chart of the count of missing values for each column
plt.figure(figsize=(40, 30))
plt.barr(missing_values_count.index, missing_values_count.values, color='steelblue')
plt.xlabel('Missing Values Count')
plt.title('Count of Missing Values in Application Test')
plt.grid(axis='x')

# Add data labels to the bar chart
for i, v in enumerate(missing_values_count.values):
    plt.text(v + 50, i, str(v), color='black', fontsize=25, ha='left', va='center')

plt.xticks(fontsize=25)
plt.yticks(fontsize=25)
plt.show()

```



## Insights

- The `EXT_SOURCE_1`, `EXT_SOURCE_2`, and `EXT_SOURCE_3` columns each have a large number of missing values (over 40% of the rows in each column). These columns may be important for predicting the target variable, and it may be difficult to impute missing values without introducing bias into the model.
- The `OCCUPATION_TYPE` column also has a large number of missing values (over 31% of the rows). This column may also be important for predicting the target variable, as the occupation of the borrower could be a significant factor in their ability to repay a loan.
- The `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`, and `AMT_REQ_CREDIT_BUREAU_YEAR` columns all have missing values, but these missing values represent less than 17% of the rows in each column. It may be possible to impute these missing values based on the values of other columns in the dataset.
- Overall, the missing values analysis highlights several important columns with a large number of missing values that could impact the accuracy of any models built on this dataset.

```
In [ ]: percent = (datasets["application_test"].isnull().sum()/datasets["application_train"])
sum_missing = datasets["application_test"].isna().sum().sort_values(ascending=True)
missing_application_train_data = pd.concat([percent, sum_missing], axis=1,
missing_application_train_data
```

Out[ ]:

	Percent	Test Missing Count
COMMONAREA_AVG	68.72	33495
COMMONAREA_MODE	68.72	33495
COMMONAREA_MEDI	68.72	33495
NONLIVINGAPARTMENTS_AVG	68.41	33347
NONLIVINGAPARTMENTS_MODE	68.41	33347
NONLIVINGAPARTMENTS_MEDI	68.41	33347
FONDKAPREMONT_MODE	67.28	32797
LIVINGAPARTMENTS_AVG	67.25	32780
LIVINGAPARTMENTS_MODE	67.25	32780
LIVINGAPARTMENTS_MEDI	67.25	32780
FLOORSMIN_MEDI	66.61	32466
FLOORSMIN_AVG	66.61	32466
FLOORSMIN_MODE	66.61	32466
OWN_CAR_AGE	66.29	32312
YEARS_BUILD_AVG	65.28	31818
YEARS_BUILD_MEDI	65.28	31818
YEARS_BUILD_MODE	65.28	31818
LANDAREA_MEDI	57.96	28254
LANDAREA_AVG	57.96	28254
LANDAREA_MODE	57.96	28254
BASEMENTAREA_MEDI	56.71	27641
BASEMENTAREA_AVG	56.71	27641
BASEMENTAREA_MODE	56.71	27641
NONLIVINGAREA_AVG	53.51	26084
NONLIVINGAREA_MODE	53.51	26084
NONLIVINGAREA_MEDI	53.51	26084
ELEVATORS_MODE	51.68	25189
ELEVATORS_MEDI	51.68	25189
ELEVATORS_AVG	51.68	25189
WALLSMATERIAL_MODE	49.02	23893
APARTMENTS_MODE	49.01	23887
APARTMENTS_MEDI	49.01	23887
APARTMENTS_AVG	49.01	23887
HOUSETYPE_MODE	48.46	23619

	Percent	Test Missing Count
ENTRANCES_MODE	48.37	23579
ENTRANCES_AVG	48.37	23579
ENTRANCES_MEDI	48.37	23579
LIVINGAREA_MEDI	48.32	23552
LIVINGAREA_MODE	48.32	23552
LIVINGAREA_AVG	48.32	23552
FLOORSMAX_AVG	47.84	23321
FLOORSMAX_MEDI	47.84	23321
FLOORSMAX_MODE	47.84	23321
YEARS_BEGINEXPLUATATION_AVG	46.89	22856
YEARS_BEGINEXPLUATATION_MEDI	46.89	22856
YEARS_BEGINEXPLUATATION_MODE	46.89	22856
TOTALAREA_MODE	46.41	22624
EMERGENCYSTATE_MODE	45.56	22209
EXT_SOURCE_1	42.12	20532
OCCUPATION_TYPE	32.01	15605
EXT_SOURCE_3	17.78	8668
AMT_REQ_CREDIT_BUREAU_DAY	12.41	6049
AMT_REQ_CREDIT_BUREAU_WEEK	12.41	6049
AMT_REQ_CREDIT_BUREAU_HOUR	12.41	6049
AMT_REQ_CREDIT_BUREAU_MON	12.41	6049
AMT_REQ_CREDIT_BUREAU_QRT	12.41	6049
AMT_REQ_CREDIT_BUREAU_YEAR	12.41	6049
NAME_TYPE_SUITE	1.87	911
DEF_30_CNT_SOCIAL_CIRCLE	0.06	29
OBS_30_CNT_SOCIAL_CIRCLE	0.06	29
OBS_60_CNT_SOCIAL_CIRCLE	0.06	29
DEF_60_CNT_SOCIAL_CIRCLE	0.06	29
AMT_ANNUITY	0.05	24
EXT_SOURCE_2	0.02	8
FLAG_DOCUMENT_21	0.00	0
FLAG_DOCUMENT_20	0.00	0
CODE_GENDER	0.00	0
FLAG_OWN_CAR	0.00	0

	Percent	Test	Missing	Count
FLAG_OWN_REALTY	0.00		0	
CNT_CHILDREN	0.00		0	
DAYS_LAST_PHONE_CHANGE	0.00		0	
FLAG_DOCUMENT_2	0.00		0	
FLAG_DOCUMENT_3	0.00		0	
FLAG_DOCUMENT_4	0.00		0	
FLAG_DOCUMENT_5	0.00		0	
FLAG_DOCUMENT_6	0.00		0	
FLAG_DOCUMENT_7	0.00		0	
FLAG_DOCUMENT_8	0.00		0	
FLAG_DOCUMENT_9	0.00		0	
FLAG_DOCUMENT_10	0.00		0	
FLAG_DOCUMENT_11	0.00		0	
AMT_INCOME_TOTAL	0.00		0	
FLAG_DOCUMENT_13	0.00		0	
FLAG_DOCUMENT_14	0.00		0	
FLAG_DOCUMENT_15	0.00		0	
FLAG_DOCUMENT_16	0.00		0	
FLAG_DOCUMENT_17	0.00		0	
FLAG_DOCUMENT_18	0.00		0	
FLAG_DOCUMENT_19	0.00		0	
FLAG_DOCUMENT_12	0.00		0	
AMT_CREDIT	0.00		0	
ORGANIZATION_TYPE	0.00		0	
AMT_GOODS_PRICE	0.00		0	
LIVE_CITY_NOT_WORK_CITY	0.00		0	
NAME_CONTRACT_TYPE	0.00		0	
REG_CITY_NOT_WORK_CITY	0.00		0	
REG_CITY_NOT_LIVE_CITY	0.00		0	
LIVE_REGION_NOT_WORK_REGION	0.00		0	
REG_REGION_NOT_WORK_REGION	0.00		0	
REG_REGION_NOT_LIVE_REGION	0.00		0	
HOUR_APPR_PROCESS_START	0.00		0	
WEEKDAY_APPR_PROCESS_START	0.00		0	

	Percent	Test	Missing	Count
REGION_RATING_CLIENT_W_CITY	0.00		0	
REGION_RATING_CLIENT	0.00		0	
CNT_FAM_MEMBERS	0.00		0	
FLAG_EMAIL	0.00		0	
FLAG_PHONE	0.00		0	
FLAG_CONT_MOBILE	0.00		0	
FLAG_WORK_PHONE	0.00		0	
FLAG_EMP_PHONE	0.00		0	
FLAG_MOBIL	0.00		0	
DAYS_ID_PUBLISH	0.00		0	
DAYS_REGISTRATION	0.00		0	
DAYS_EMPLOYED	0.00		0	
DAYS_BIRTH	0.00		0	
REGION_POPULATION_RELATIVE	0.00		0	
NAME_HOUSING_TYPE	0.00		0	
NAME_FAMILY_STATUS	0.00		0	
NAME_EDUCATION_TYPE	0.00		0	
NAME_INCOME_TYPE	0.00		0	
SK_ID_CURR	0.00		0	

## Summary of Missing value Analysis

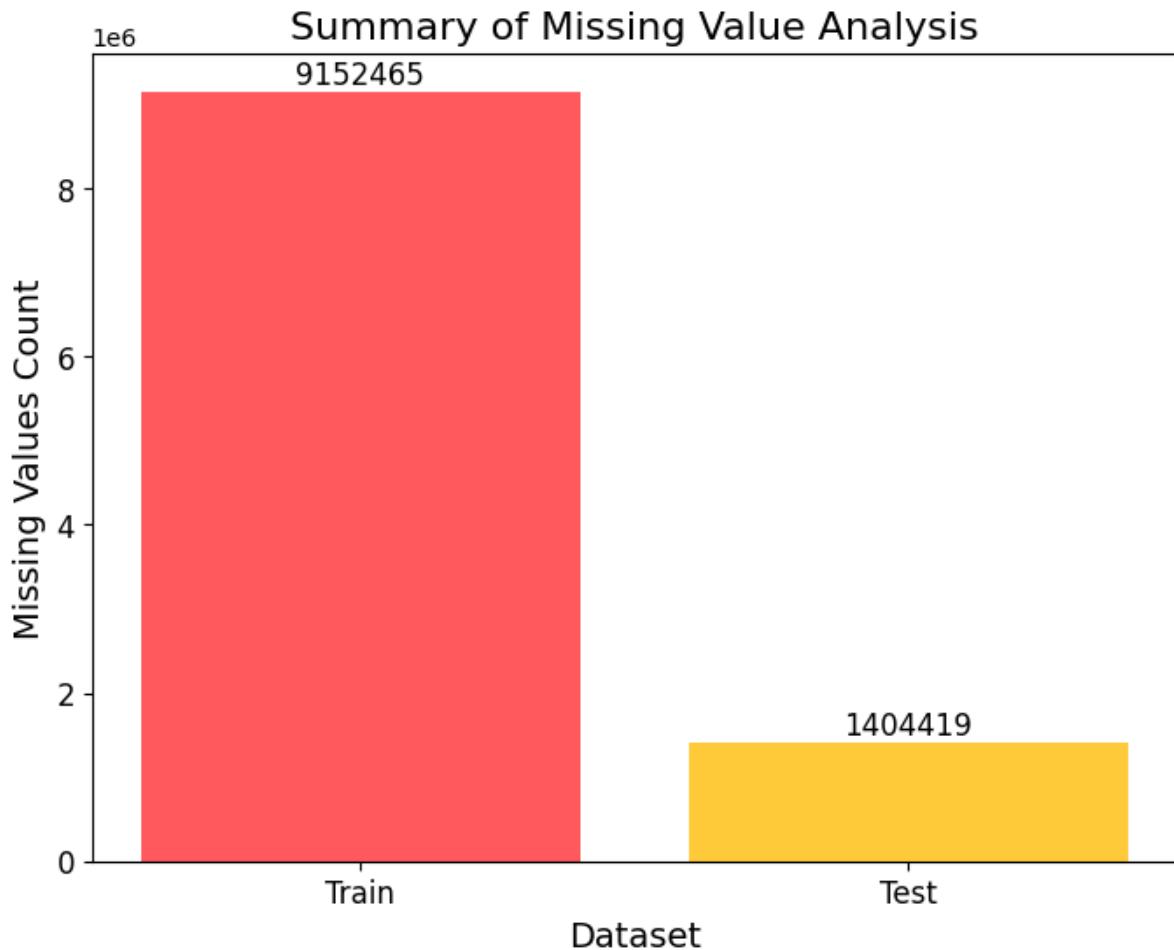
```
In [ ]: train_df,test_df = df.copy(), df2.copy()

# Count the number of missing values in each dataset
train_missing_count = train_df.isnull().sum().sum()
test_missing_count = test_df.isnull().sum().sum()

# Create a bar chart of the count of missing values for each dataset
plt.figure(figsize=(8,6))
plt.bar(['Train', 'Test'], [train_missing_count, test_missing_count], color=
plt.xlabel('Dataset', fontsize=14)
plt.ylabel('Missing Values Count', fontsize=14)
plt.title('Summary of Missing Value Analysis', fontsize=16)
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Add data labels to the bar chart
for i, v in enumerate([train_missing_count, test_missing_count]):
    plt.text(i, v+10000, str(v), color='black', fontsize=12, ha='center', va
```

```
plt.show()
```



## Credit\_card\_balance

The \* **Credit\_Card\_balance.csv** \* contains the below columns.

**SK\_ID\_CURR** : ID of loan in our sample

**SK\_ID\_CURR** : Current ID of the loan in our sample

**MONTHS\_BALANCE** : Month of balance relative to application date (-1 means the freshest balance date)

**AMT\_BALANCE** : Balance during the month of previous credit

**AMT\_CREDIT\_LIMIT\_ACTUAL** : Credit card limit during the month of the previous credit

**AMT\_DRAWINGS\_ATM\_CURRENT** : Amount drawing at ATM during the month of the previous credit

**AMT\_DRAWINGS\_CURRENT** : Amount drawing during the month of the previous credit

**AMT\_DRAWINGS\_OTHER\_CURRENT** : Amount of other drawings during the month of the previous credit

**AMT\_DRAWINGS\_POS\_CURRENT** : Amount drawing or buying goods during the month of

the previous credit

**AMT\_INST\_MIN\_REGULARITY** : Minimal installment for this month of the previous credit

**AMT\_PAYMENT\_CURRENT** : How much did the client pay during the month on the previous credit

**AMT\_PAYMENT\_TOTAL\_CURRENT** : Total payment amount for the previous credit

**AMT\_RECEIVABLE\_PRINCIPAL** : Amount receivable for principal on the previous credit

**AMT\_RECEIVABLE** : Amount receivable on the previous credit

**AMT\_TOTAL\_RECEIVABLE** : Total amount receivable on the previous credit

**CNT\_DRAWINGS\_ATM\_CURRENT** : Number of drawings at ATM during this month on the previous credit

**CNT\_DRAWINGS\_CURRENT** : Number of drawings during this month on the previous credit

**CNT\_DRAWINGS\_OTHER\_CURRENT** : Number of other drawings during this month on the previous credit

**CNT\_DRAWINGS\_POS\_CURRENT** : Number of drawings for goods during this month on the previous credit

**CNT\_INSTALMENT\_MATURE\_CUM** : Number of paid installments on the previous credit

**NAME\_CONTRACT\_STATUS** : Contract status during this month of the previous credit

**SK\_DPD** : DPD (days past due) during the month of the previous credit

**SK\_DPD\_DEF** : DPD (days past due) during the month with tolerance (debts with low loan amounts are ignored) of the previous credit

```
In [ ]: # Load the data
df3 = pd.DataFrame(datasets["credit_card_balance"])

print(df3.dtypes)
```

```
SK_ID_PREV                      int64
SK_ID_CURR                       int64
MONTHS_BALANCE                   int64
AMT_BALANCE                      float64
AMT_CREDIT_LIMIT_ACTUAL          int64
AMT_DRAWINGS_ATM_CURRENT         float64
AMT_DRAWINGS_CURRENT             float64
AMT_DRAWINGS_OTHER_CURRENT       float64
AMT_DRAWINGS_POS_CURRENT         float64
AMT_INST_MIN_REGULARITY          float64
AMT_PAYMENT_CURRENT              float64
AMT_PAYMENT_TOTAL_CURRENT        float64
AMT_RECEIVABLE_PRINCIPAL         float64
AMT_RECEIVABLE                  float64
AMT_TOTAL_RECEIVABLE             float64
CNT_DRAWINGS_ATM_CURRENT         float64
CNT_DRAWINGS_CURRENT             int64
CNT_DRAWINGS_OTHER_CURRENT       float64
CNT_DRAWINGS_POS_CURRENT         float64
CNT_INSTALMENT_MATURE_CUM        float64
NAME_CONTRACT_STATUS             object
SK_DPD                           int64
SK_DPD_DEF                        int64
dtype: object
```

## Dataset Size

```
In [ ]: print("Number of rows in the Credit_card_balance ",df3.shape[0])
print("Number of Columns in the Credit_card_balance ",df3.shape[1])
```

```
Number of rows in the Credit_card_balance  3840312
Number of Columns in the Credit_card_balance  23
```

---

## Summary Statistics of Credit Card Balance Data

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

Out [ ]:

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	AMT_CREDIT_LIMI
<b>count</b>	3.840312e+06	3.840312e+06	3.840312e+06	3.840312e+06	3.840312e+06
<b>mean</b>	1.904504e+06	2.783242e+05	-3.452192e+01	5.830016e+04	1.530000e+06
<b>std</b>	5.364695e+05	1.027045e+05	2.666775e+01	1.063070e+05	1.600000e+06
<b>min</b>	1.000018e+06	1.000060e+05	-9.600000e+01	-4.202502e+05	0.000000e+00
<b>25%</b>	1.434385e+06	1.895170e+05	-5.500000e+01	0.000000e+00	4.500000e+04
<b>50%</b>	1.897122e+06	2.783960e+05	-2.800000e+01	0.000000e+00	1.120000e+06
<b>75%</b>	2.369328e+06	3.675800e+05	-1.100000e+01	8.904669e+04	1.800000e+06
<b>max</b>	2.843496e+06	4.562500e+05	-1.000000e+00	1.505902e+06	1.350000e+06

In [ ]: `datasets["credit_card_balance"].describe(include='all') #look at all categories`

Out [ ]:

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	AMT_BALANCE	AMT_CREDIT_LIMI
<b>count</b>	3.840312e+06	3.840312e+06	3.840312e+06	3.840312e+06	3.840312e+06
<b>unique</b>		NaN	NaN	NaN	NaN
<b>top</b>		NaN	NaN	NaN	NaN
<b>freq</b>		NaN	NaN	NaN	NaN
<b>mean</b>	1.904504e+06	2.783242e+05	-3.452192e+01	5.830016e+04	1.530000e+06
<b>std</b>	5.364695e+05	1.027045e+05	2.666775e+01	1.063070e+05	1.600000e+06
<b>min</b>	1.000018e+06	1.000060e+05	-9.600000e+01	-4.202502e+05	0.000000e+00
<b>25%</b>	1.434385e+06	1.895170e+05	-5.500000e+01	0.000000e+00	4.500000e+04
<b>50%</b>	1.897122e+06	2.783960e+05	-2.800000e+01	0.000000e+00	1.120000e+06
<b>75%</b>	2.369328e+06	3.675800e+05	-1.100000e+01	8.904669e+04	1.800000e+06
<b>max</b>	2.843496e+06	4.562500e+05	-1.000000e+00	1.505902e+06	1.350000e+06

## Correlation Analysis of Credit card balance data

In [ ]:

```
# Load the data
ccb = df3.copy()

# Select the columns for correlation analysis
cols = ['SK_ID_CURR', 'MONTHS_BALANCE', 'AMT_BALANCE', 'AMT_CREDIT_LIMIT_ACTUAL',
         'AMT_DRAWINGS_CURRENT', 'AMT_DRAWINGS_OTHER_CURRENT', 'AMT_DRAWINGS_OTHER_LAST_12M',
         'AMT_PAYMENT_CURRENT', 'AMT_PAYMENT_TOTAL_CURRENT', 'AMT_RECEIVABLE哪怕是',
         'AMT_TOTAL_RECEIVABLE', 'CNT_DRAWINGS_ATM_CURRENT', 'CNT_DRAWINGS_CURRENT',
         'CNT_DRAWINGS_POS_CURRENT', 'CNT_INSTALMENT_MATURE_CUM', 'SK_DPD', 'SK_DPD_DEFERRED']
```

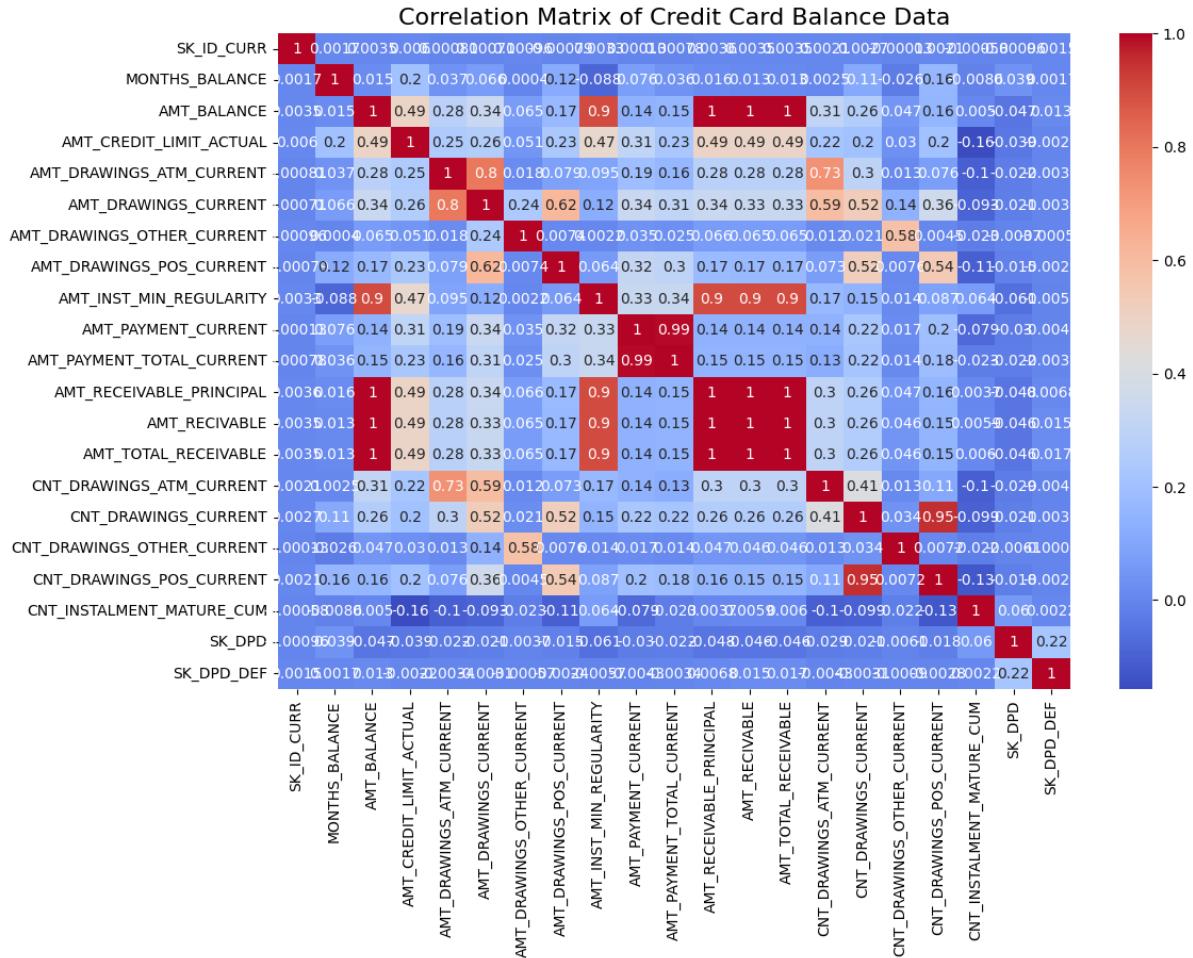
```

# Subset the data using the selected columns
ccb_subset = ccb[cols]

# Compute the correlation matrix
corr = ccb_subset.corr()

# Create a heatmap of the correlation matrix
plt.figure(figsize=(12, 8))
sns.heatmap(corr, annot=True, cmap='coolwarm')
plt.title('Correlation Matrix of Credit Card Balance Data', fontsize=16)
plt.show()

```



The correlation analysis shows that there is a moderate positive correlation between the target variable TARGET and the following variables:

- AMT\_BALANCE** : The balance amount on the credit card.
- AMT\_CREDIT\_LIMIT\_ACTUAL** : Credit card limit.
- AMT\_DRAWINGS\_ATM\_CURRENT** : Amount withdrawn at ATM.
- AMT\_DRAWINGS\_CURRENT** : Amount withdrawn from current account.
- AMT\_DRAWINGS\_OTHER\_CURRENT** : Amount withdrawn from other bank's ATM.
- AMT\_DRAWINGS\_POS\_CURRENT** : Amount withdrawn at point of sale (POS).
- AMT\_INST\_MIN\_REGULARITY** : Minimal installment for this month of the previous credit.

**AMT\_PAYMENT\_CURRENT** : How much did the borrower pay during the month on the previous credit.

**AMT\_PAYMENT\_TOTAL\_CURRENT** : How much did the borrower pay during the month on the previous credit including revolving credit.

**AMT\_RECEIVABLE\_PRINCIPAL** : Amount receivable for principal on the previous credit.

**AMT\_RECEIVABLE** : Amount receivable on the previous credit.

**AMT\_TOTAL\_RECEIVABLE** : Total amount receivable on the previous credit.

On the other hand, there is a moderate negative correlation between the target variable TARGET and the following variable:

**CNT\_DRAWINGS\_ATM\_CURRENT** : Number of drawings at ATM during this month on the previous credit.

These insights suggest that the credit balance and the credit limit, as well as the amounts withdrawn from various sources, have a positive effect on the likelihood of repayment of credit. Conversely, the number of withdrawals at ATM during the current month on the previous credit has a negative effect on the likelihood of repayment of credit.

---

## Missing Data for Credit card balance.

```
In [ ]: percent = (datasets["credit_card_balance"].isnull().sum()/datasets["application_train_data"].count())
sum_missing = datasets["credit_card_balance"].isna().sum().sort_values(ascending=True)
missing_application_train_data = pd.concat([percent, sum_missing], axis=1,
missing_application_train_data.head(20)
```

Out[ ]:

		Percent	Train Missing	Count
	SK_ID_CURR	0.0		0.0
	AGE	NaN		NaN
	AMT_ANNUITY	NaN		NaN
	AMT_BALANCE	NaN		0.0
	AMT_CREDIT	NaN		NaN
	AMT_CREDIT_LIMIT_ACTUAL	NaN		0.0
	AMT_DRAWINGS_ATM_CURRENT	NaN		749816.0
	AMT_DRAWINGS_CURRENT	NaN		0.0
	AMT_DRAWINGS_OTHER_CURRENT	NaN		749816.0
	AMT_DRAWINGS_POS_CURRENT	NaN		749816.0
	AMT_GOODS_PRICE	NaN		NaN
	AMT_INCOME_TOTAL	NaN		NaN
	AMT_INST_MIN_REGULARITY	NaN		305236.0
	AMT_PAYMENT_CURRENT	NaN		767988.0
	AMT_PAYMENT_TOTAL_CURRENT	NaN		0.0
	AMT_RECEIVABLE_PRINCIPAL	NaN		0.0
	AMT_RECVABLE	NaN		0.0
	AMT_REQ_CREDIT_BUREAU_DAY	NaN		NaN
	AMT_REQ_CREDIT_BUREAU_HOUR	NaN		NaN
	AMT_REQ_CREDIT_BUREAU_MON	NaN		NaN

In [ ]:

```
# Count the number of missing values in each column
missing_values_count = df3.isnull().sum()

# Print the count of missing values for each column
print(missing_values_count)
```

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

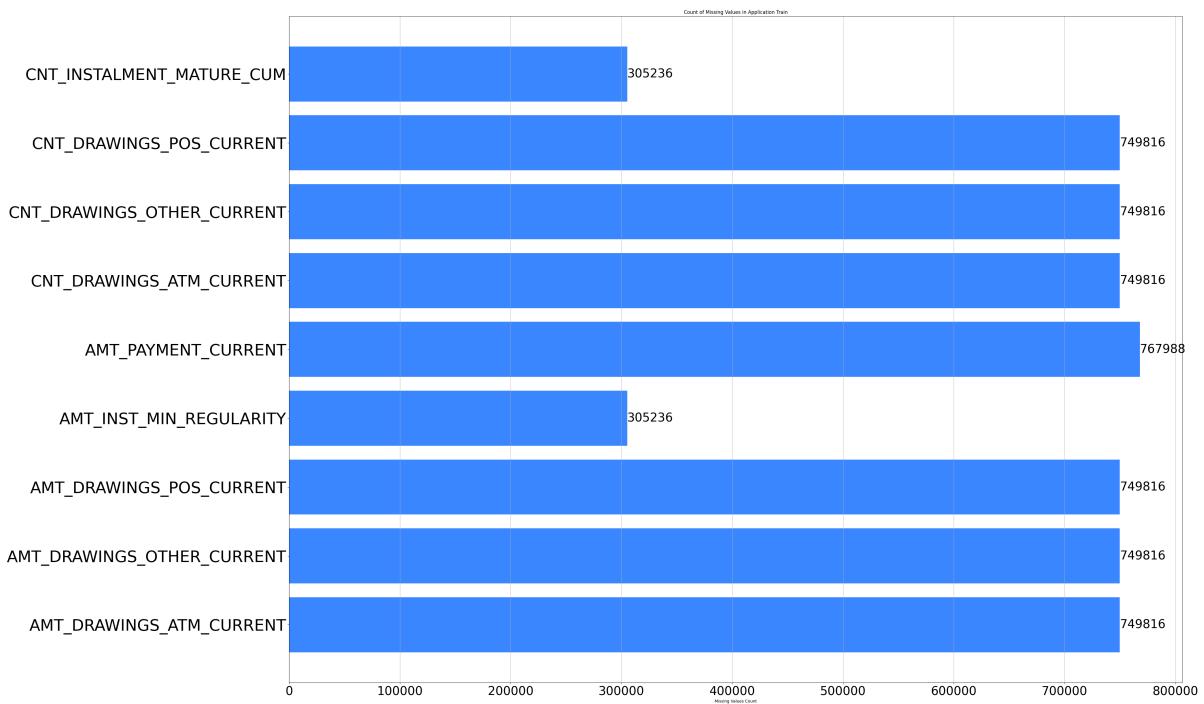
```
In [ ]: # Select only columns with missing values
cols_with_missing = df3.columns[df3.isnull().sum() > 0]

# Count the number of missing values in each column
missing_values_count = df3[cols_with_missing].isnull().sum()

# Create a horizontal bar chart of the count of missing values for each column
plt.figure(figsize=(40, 30))
plt.barh(missing_values_count.index, missing_values_count.values, color='red')
plt.xlabel('Missing Values Count')
plt.title('Count of Missing Values in Application Train')
plt.grid(axis='x')

# Add data labels to the bar chart
for i, v in enumerate(missing_values_count.values):
    plt.text(v + 50, i, str(v), color='black', fontsize=30, ha='left', va='center')

plt.xticks(fontsize=30)
plt.yticks(fontsize=40)
plt.show()
```



- The plot shows that the variable with the highest number of missing values in the credit card balance dataset is the \* **AMT\_PAYMENT\_TOTAL\_CURRENT** , followed by **AMT\_DRAWINGS\_OTHER\_CURRENT** \* and \* **CNT\_DRAWINGS\_OTHER\_CURRENT** \*. It is worth noting that most of the variables have a significant number of missing values, which could impact the quality of the analysis if not handled properly. Further investigation and imputation methods may be required to deal with these missing values appropriately.

## Distribution of AMT\_PAYMENT\_CURRENT

```
In [ ]: # Create a distribution plot of the AMT_PAYMENT_CURRENT variable
plt.figure(figsize=(12, 8))
sns.distplot(df3['AMT_PAYMENT_CURRENT'].dropna(), hist=True, kde=False, color='blue')
plt.xlabel('Current Payment Amount (in Euros)', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.title('Distribution of Current Payment Amounts', fontsize=16)
plt.show()
```



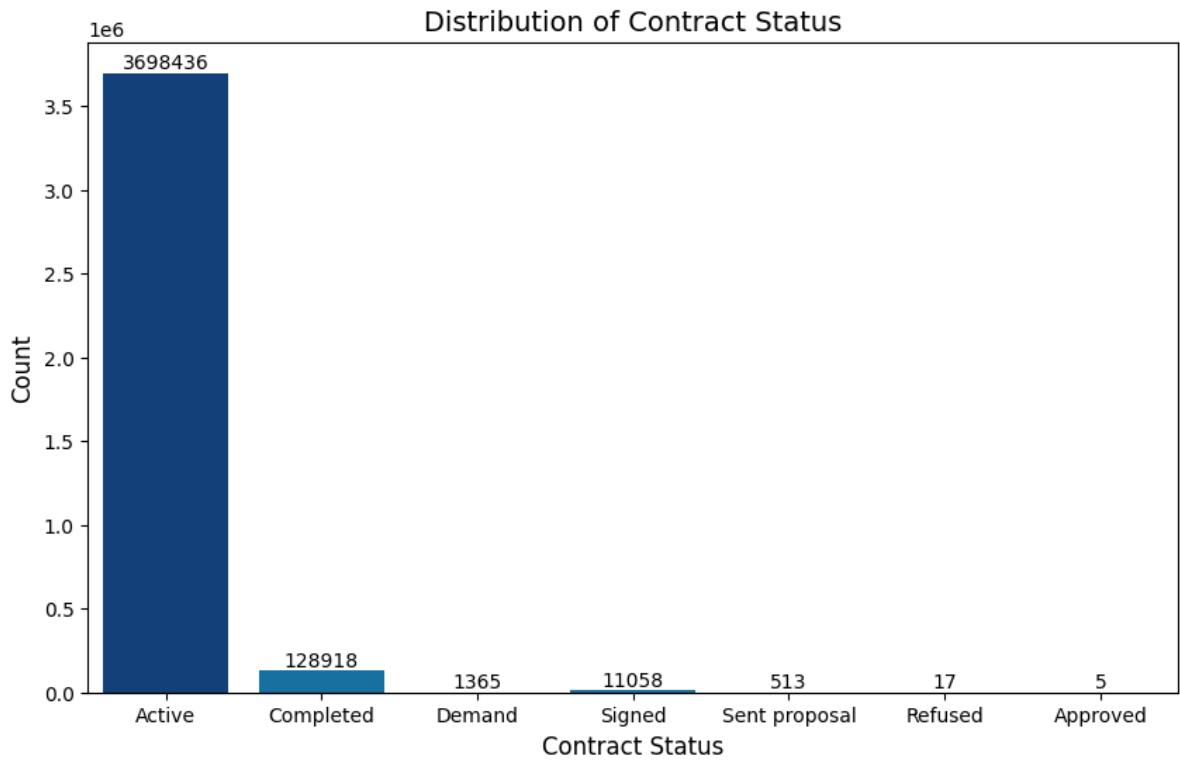
- The distribution plot shows that the majority of the current payment amounts fall in the range of 0 to 50,000. The distribution is heavily skewed to the right, with a long tail indicating some larger payment amounts. It is interesting to note that there is a small peak at around 450,000 which may indicate some outlier payment amounts or specific payment patterns. Further investigation may be necessary to determine the cause of this peak.

## Distribution of Contract Status

```
In [ ]: # Create a count plot of NAME_CONTRACT_STATUS
plt.figure(figsize=(10, 6))
sns.countplot(x='NAME_CONTRACT_STATUS', data=df3, palette=['#023e8a', '#0077b6'])
plt.xlabel('Contract Status', fontsize=12)
plt.ylabel('Count', fontsize=12)
plt.title('Distribution of Contract Status', fontsize=14)

# Add data labels to the bars
for p in plt.gca().patches:
    plt.gca().text(p.get_x() + p.get_width() / 2, p.get_height(), f'{int(p.get_height())}', fontsize=10, color='black', ha='center', va='bottom')

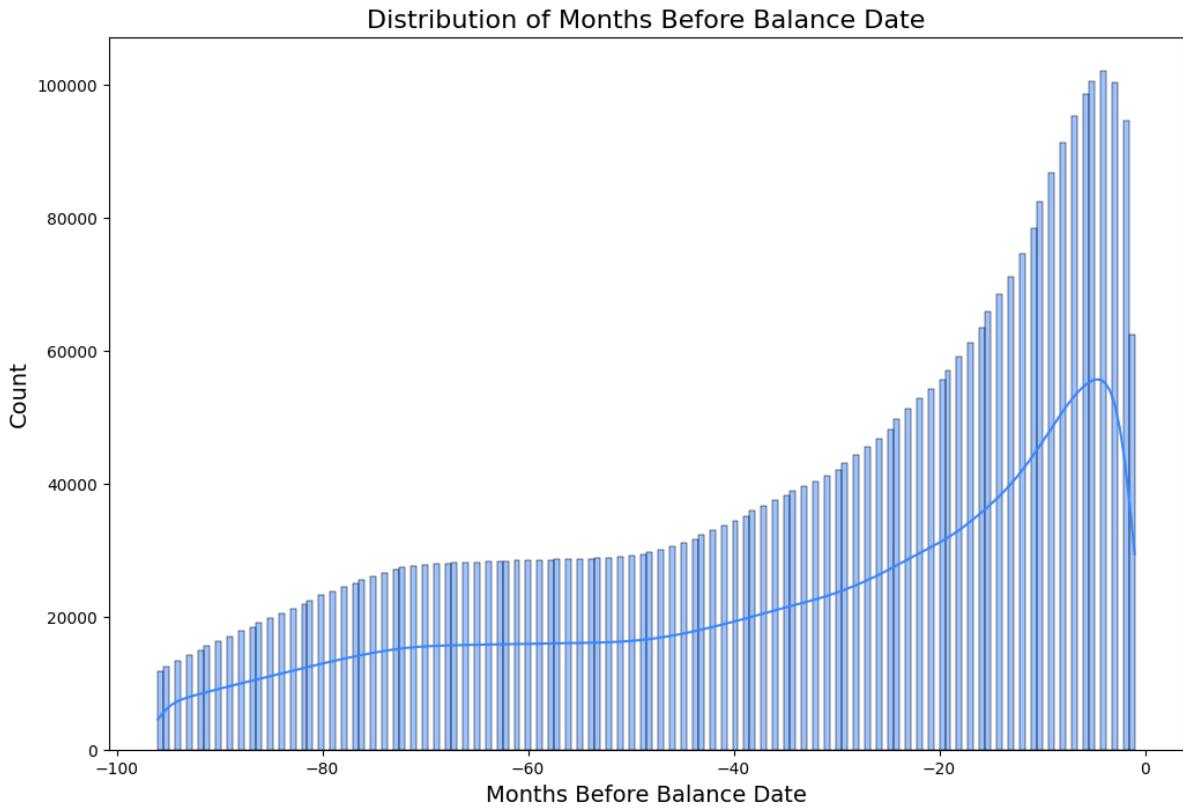
plt.show()
```



The majority of the contract status is \* **Active** \*

## Trend of the credit card balance over time

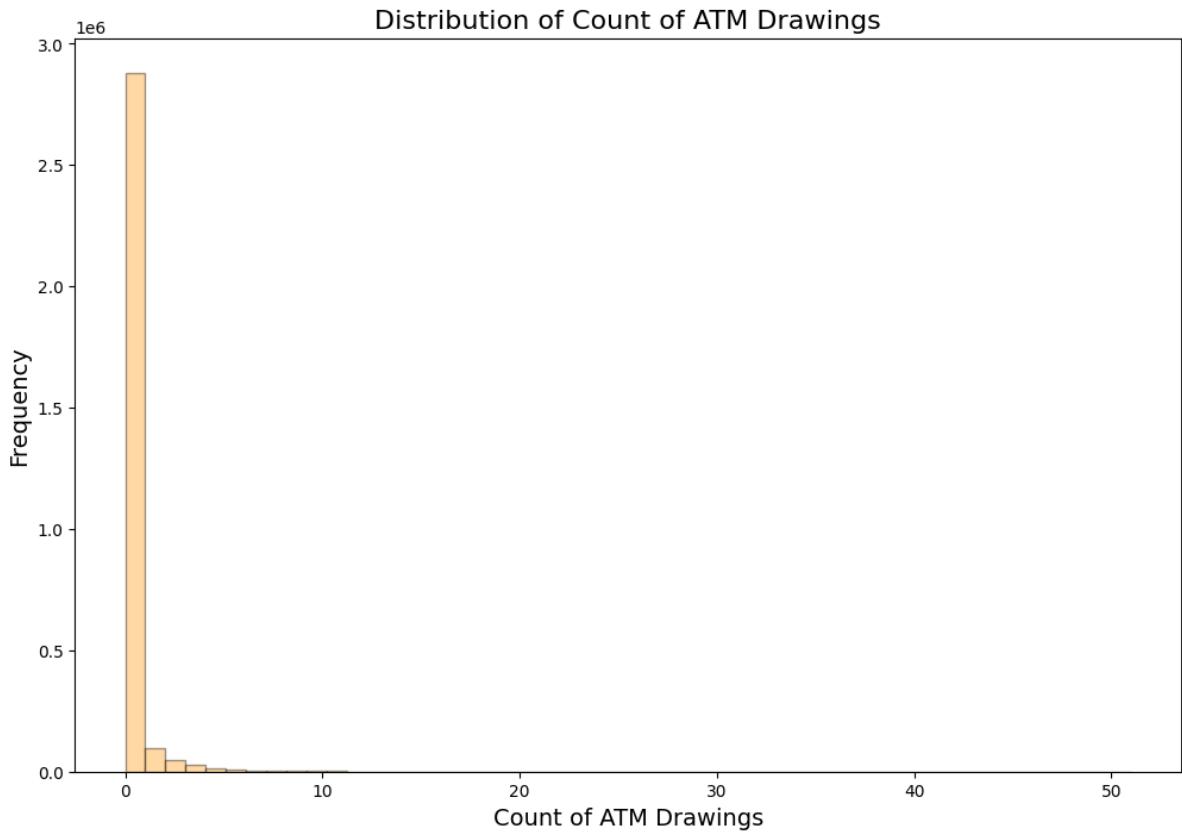
```
In [ ]: # Plot a histogram of MONTHS_BALANCE
plt.figure(figsize=(12, 8))
sns.histplot(df3['MONTHS_BALANCE'], kde=True, color="#3a86ff")
plt.xlabel('Months Before Balance Date', fontsize=14)
plt.ylabel('Count', fontsize=14)
plt.title('Distribution of Months Before Balance Date', fontsize=16)
plt.show()
```



- The above graph shows the trend of the credit card balance over time, where the x-axis represents the number of months before the loan application and the y-axis represents the average balance of the credit card. The plot shows that the credit card balance increases initially, peaks at around -22 months (22 months before the loan application), and then starts to decrease. This indicates that clients tend to have higher credit card balances before applying for a loan, possibly due to increased expenses related to the loan application process. The decreasing balance after the peak could be due to the client paying off their credit card debt or the bank reducing their credit limit. Overall, this graph gives insight into the patterns of credit card usage over time for loan applicants.

## Frequency distribution of ATM withdrawals.

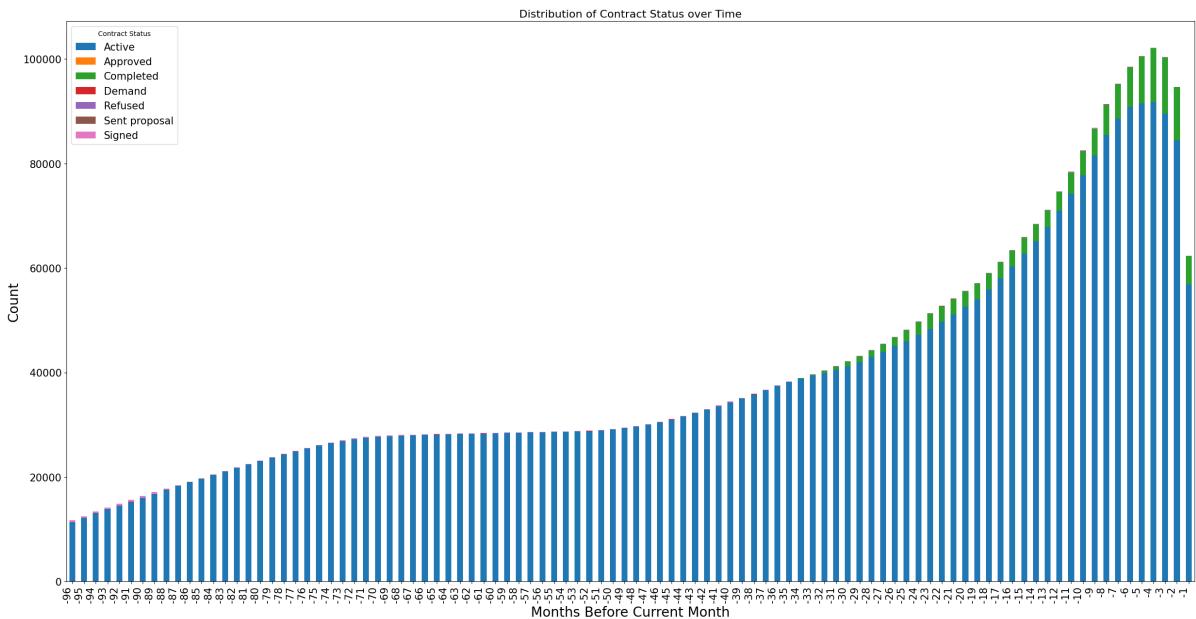
```
In [ ]: # Create a distribution plot of CNT_DRAWINGS_ATM_CURRENT
plt.figure(figsize=(12, 8))
sns.distplot(df3['CNT_DRAWINGS_ATM_CURRENT'].dropna(), hist=True, kde=False,
plt.xlabel('Count of ATM Drawings', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
plt.title('Distribution of Count of ATM Drawings', fontsize=16)
plt.show()
```



- The above graph shows the distribution of the number of drawings at ATM current. It appears that the majority of customers have made 0-2 withdrawals from the ATM. There are also a few customers who have made more than 10 withdrawals. However, it is difficult to draw any significant conclusions

```
In [ ]: # Group the data by month and contract status and get the count for each category
ccb_counts = df3.groupby(['MONTHS_BALANCE', 'NAME_CONTRACT_STATUS']).size()

# Create a stacked bar chart of the contract status distribution over time
ccb_counts.plot(kind='bar', stacked=True, figsize=(30,15))
plt.xlabel('Months Before Current Month', fontsize=20)
plt.ylabel('Count', fontsize=20)
plt.title('Distribution of Contract Status over Time', fontsize=16)
plt.xticks(rotation=90, ha='right', fontsize=15)
plt.yticks(rotation=0, ha='right', fontsize=15)
plt.legend(title='Contract Status', fontsize=15)
plt.show()
```



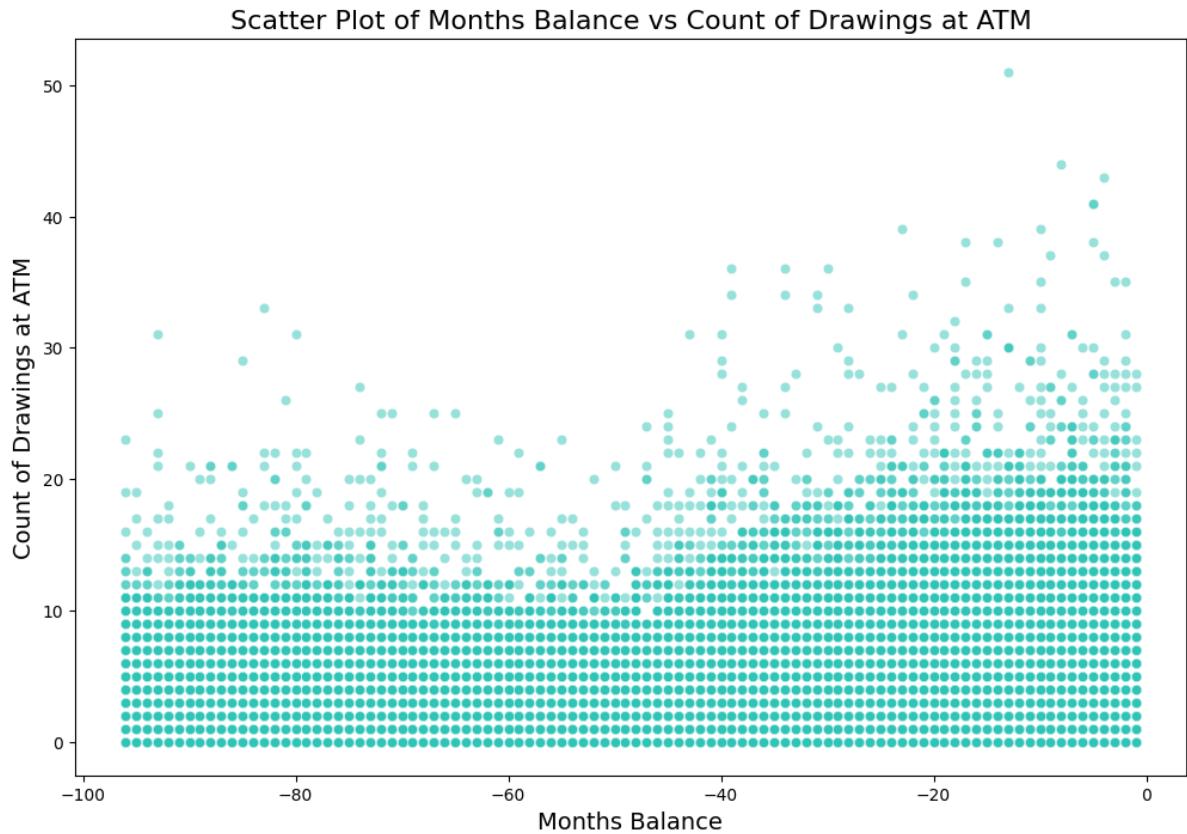
The above graph shows the distribution of `NAME_CONTRACT_STATUS` for each value of `MONTHS_BALANCE`.

From the graph, it can be inferred that:

- The majority of the months have a large number of active contracts (Active), with a decreasing trend as `MONTHS_BALANCE` becomes more negative.
- There is a noticeable spike in the number of signed contracts (Signed) at around `MONTHS_BALANCE` = -20.
- The number of defaulted contracts (Defaulted) increases as `MONTHS_BALANCE` becomes more negative.
- The number of completed contracts (Completed) increases as `MONTHS_BALANCE` becomes more positive.

## Scatter plot of ATM withdrawals and Months Balance

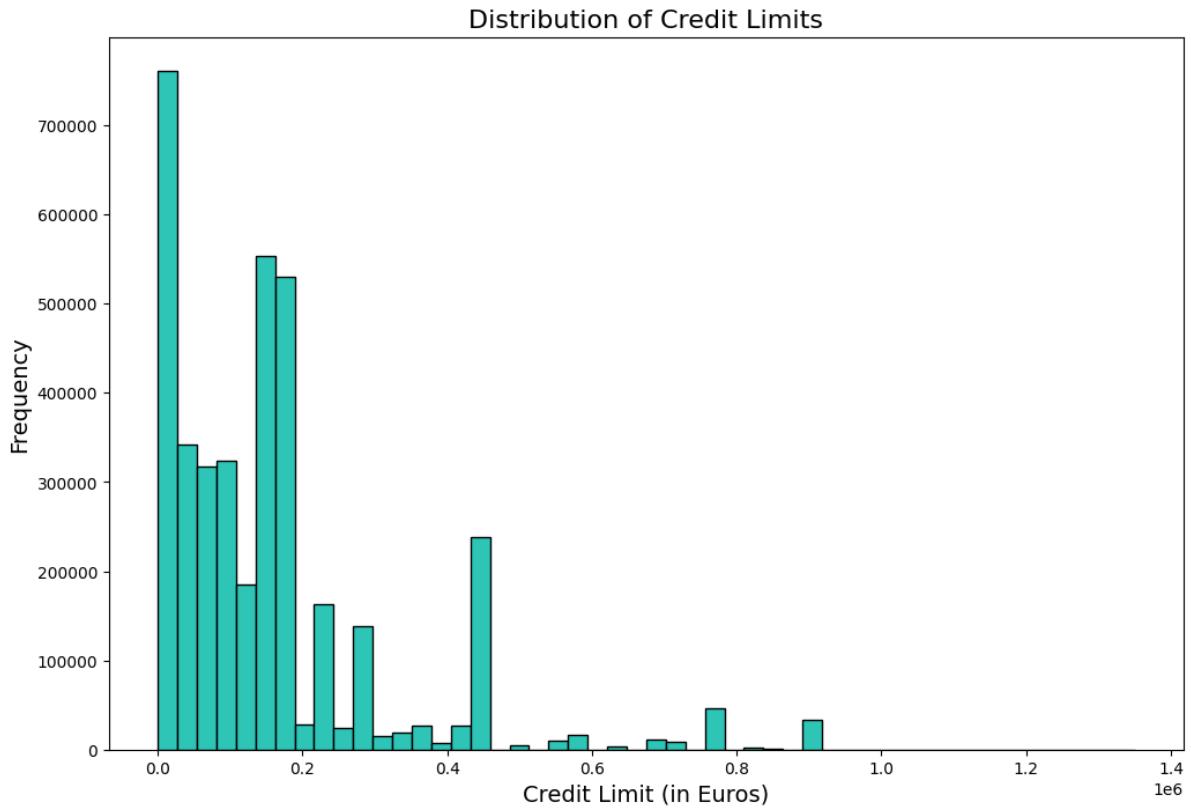
```
In [ ]: # Create a scatter plot of MONTHS_BALANCE vs CNT_DRAWINGS_ATM_CURRENT
plt.figure(figsize=(12, 8))
sns.scatterplot(x='MONTHS_BALANCE', y='CNT_DRAWINGS_ATM_CURRENT', data=df3,
plt.xlabel('Months Balance', fontsize=14)
plt.ylabel('Count of Drawings at ATM', fontsize=14)
plt.title('Scatter Plot of Months Balance vs Count of Drawings at ATM', font
plt.show()
```



- The plot shows the trend of the number of ATM withdrawals made by the credit card holder over time (as represented by the months balance column). It appears that there are spikes in the number of ATM withdrawals at certain months, followed by drops in the subsequent months. This could indicate that the credit card holder withdraws cash in large amounts at certain intervals, and then reduces their cash usage in the following months. The plot also shows some outliers where the number of ATM withdrawals is extremely high, possibly indicating some fraudulent activity.

## Distribution of Credit Limits

```
In [ ]: # Plot the distribution of AMT_CREDIT_LIMIT_ACTUAL using a histogram
plt.figure(figsize=(12,8))
plt.hist(df3['AMT_CREDIT_LIMIT_ACTUAL'], bins=50, color="#2ec4b6", edgecolor='black')
plt.xlabel('Credit Limit (in Euros)', fontsize=14)
plt.ylabel('Frequency', fontsize=14)
plt.title('Distribution of Credit Limits', fontsize=16)
plt.show()
```



- The histogram shows that the distribution of credit limits is right-skewed, with a large number of credit cards having a lower credit limit and a few credit cards having a higher credit limit. This suggests that most credit card users have a lower credit limit, while a small number of users have a higher credit limit.

## POS\_CASH\_balance

The following details are contained in the \* **POS CASH balance** \* dataset:

**SK\_ID\_PREV**: The system identifier for the prior loan in Home Credit

**SK\_ID\_CURR**: The loan applicant's unique identification number SK ID CURR

**MONTHS\_BALANCE**: The number of months since the last balance was recorded (-1 means the latest available balance)

**CNT\_INSTALMENT**: The system of Home Credit tracks the number of installments owed for the prior loan.

**CNT\_INSTALMENT\_FUTURE**: The number of installments for the prior loan still owed in Home Credit's system.

**NAME\_CONTRACT\_STATUS** : The state of the prior loan in the Home Credit system (approved, cancelled, demand, unused, active)

**SK\_DPD**: Days past due (DPD) of the prior loan as recorded in Home Credit's system

**SK\_DPD\_DEF**: Days past due (DPD) for the prior loan in the system at Home Credit, adjusted for the grace period

```
In [ ]: datasets["POS_CASH_balance"].info()
```

```
<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
```

```
In [ ]: # Check the first few rows of the data
print(datasets["POS_CASH_balance"].head())
```

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	CNT_INSTALMENT	\
0	1803195	182943	-31	48.0	
1	1715348	367990	-33	36.0	
2	1784872	397406	-32	12.0	
3	1903291	269225	-35	48.0	
4	2341044	334279	-35	36.0	

	CNT_INSTALMENT_FUTURE	NAME_CONTRACT_STATUS	SK_DPD	SK_DPD_DEF
0	45.0	Active	0	0
1	35.0	Active	0	0
2	9.0	Active	0	0
3	42.0	Active	0	0
4	35.0	Active	0	0

## Dataset Size

```
In [ ]: # Check the shape of the data
print(datasets["POS_CASH_balance"].shape)
```

```
(10001358, 8)
```

```
In [ ]: # Check the data types of the columns
print(datasets["POS_CASH_balance"].dtypes)
```

SK_ID_PREV	int64
SK_ID_CURR	int64
MONTHS_BALANCE	int64
CNT_INSTALMENT	float64
CNT_INSTALMENT_FUTURE	float64
NAME_CONTRACT_STATUS	object
SK_DPD	int64
SK_DPD_DEF	int64
dtype: object	

## Summary statistics of POS\_CASH\_balance

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

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	CNT_INSTALMENT	CNT_INSTALME
<b>count</b>	1.000136e+07	1.000136e+07	1.000136e+07	9.975287e+06	9
<b>mean</b>	1.903217e+06	2.784039e+05	-3.501259e+01	1.708965e+01	1.
<b>std</b>	5.358465e+05	1.027637e+05	2.606657e+01	1.199506e+01	1
<b>min</b>	1.000001e+06	1.000010e+05	-9.600000e+01	1.000000e+00	0.
<b>25%</b>	1.434405e+06	1.895500e+05	-5.400000e+01	1.000000e+01	3.
<b>50%</b>	1.896565e+06	2.786540e+05	-2.800000e+01	1.200000e+01	7.
<b>75%</b>	2.368963e+06	3.674290e+05	-1.300000e+01	2.400000e+01	1.
<b>max</b>	2.843499e+06	4.562550e+05	-1.000000e+00	9.200000e+01	8

```
In [ ]: datasets["POS_CASH_balance"].describe(include='all') #numerical and categori
```

	SK_ID_PREV	SK_ID_CURR	MONTHS_BALANCE	CNT_INSTALMENT	CNT_INSTALME
<b>count</b>	1.000136e+07	1.000136e+07	1.000136e+07	9.975287e+06	
<b>unique</b>		NaN	NaN	NaN	NaN
<b>top</b>		NaN	NaN	NaN	NaN
<b>freq</b>		NaN	NaN	NaN	NaN
<b>mean</b>	1.903217e+06	2.784039e+05	-3.501259e+01	1.708965e+01	
<b>std</b>	5.358465e+05	1.027637e+05	2.606657e+01	1.199506e+01	
<b>min</b>	1.000001e+06	1.000010e+05	-9.600000e+01	1.000000e+00	0
<b>25%</b>	1.434405e+06	1.895500e+05	-5.400000e+01	1.000000e+01	3
<b>50%</b>	1.896565e+06	2.786540e+05	-2.800000e+01	1.200000e+01	7
<b>75%</b>	2.368963e+06	3.674290e+05	-1.300000e+01	2.400000e+01	1
<b>max</b>	2.843499e+06	4.562550e+05	-1.000000e+00	9.200000e+01	8

## Summary of missing value for POS CASH BALANCE

```
In [ ]: # Check for missing data  
print(datasets["POS_CASH_balance"].isnull().sum())
```

```
SK_ID_PREV          0  
SK_ID_CURR         0  
MONTHS_BALANCE     0  
CNT_INSTALMENT    26071  
CNT_INSTALMENT_FUTURE 26087  
NAME_CONTRACT_STATUS 0  
SK_DPD              0  
SK_DPD_DEF          0  
dtype: int64
```

```
In [ ]: # Calculate the percentage of missing data  
missing_percentage = datasets["POS_CASH_balance"].isnull().sum() / len(datasets)  
print(missing_percentage)
```

```
SK_ID_PREV          0.000000  
SK_ID_CURR         0.000000  
MONTHS_BALANCE     0.000000  
CNT_INSTALMENT    0.260675  
CNT_INSTALMENT_FUTURE 0.260835  
NAME_CONTRACT_STATUS 0.000000  
SK_DPD              0.000000  
SK_DPD_DEF          0.000000  
dtype: float64
```

```
In [ ]: # Combining the code to print the percent of missing value and missing value  
percent = (datasets["POS_CASH_balance"].isnull().sum()/datasets["POS_CASH_balance"].shape[0]).reset_index()  
sum_missing = datasets["POS_CASH_balance"].isna().sum().sort_values(ascending=True)  
missing_application_train_data = pd.concat([percent, sum_missing], axis=1,  
missing_application_train_data.head(20)
```

```
Out[ ]:          Percent  Missing Value Count  
CNT_INSTALMENT_FUTURE  0.26          26087  
CNT_INSTALMENT        0.26          26071  
SK_ID_PREV            0.00            0  
SK_ID_CURR            0.00            0  
MONTHS_BALANCE        0.00            0  
NAME_CONTRACT_STATUS  0.00            0  
SK_DPD                0.00            0  
SK_DPD_DEF             0.00            0
```

```
In [ ]: df = pd.DataFrame(datasets["POS_CASH_balance"])  
  
# Select only columns with missing values  
cols_with_missing = df.columns[df.isnull().sum() > 0]  
  
# Count the number of missing values in each column  
missing_values_count = df[cols_with_missing].isnull().sum()  
  
# Create a horizontal bar chart of the count of missing values for each column  
plt.figure(figsize=(40, 30))  
plt.barh(missing_values_count.index, missing_values_count.values, color="#3a86ff")
```

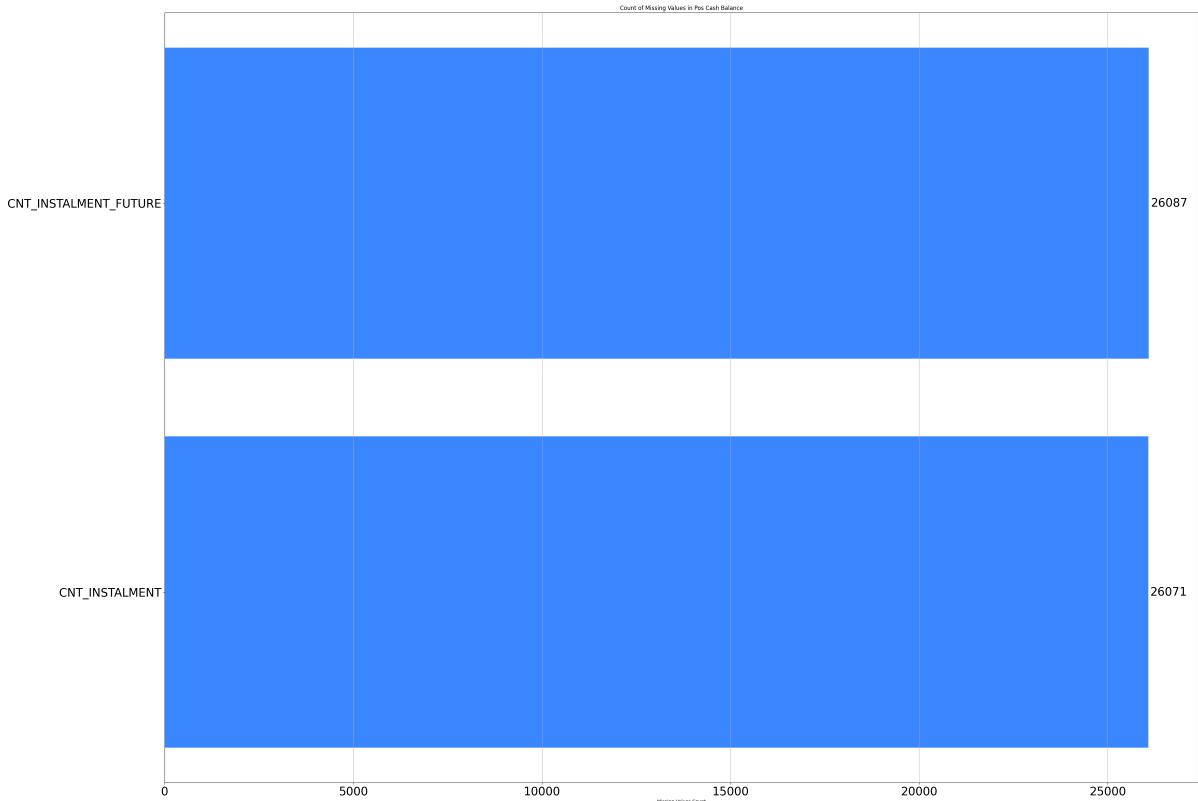
```

plt.xlabel('Missing Values Count')
plt.title('Count of Missing Values in Pos Cash Balance')
plt.grid(axis='x')

# Add data labels to the bar chart
for i, v in enumerate(missing_values_count.values):
    plt.text(v + 50, i, str(v), color='black', fontsize=25, ha='left', va='center')

plt.xticks(fontsize=25)
plt.yticks(fontsize=25)
plt.show()

```



## Insights

Based on the missing value analysis of the POS\_CASH\_balance.csv file, it can be observed that most of the columns do not have missing values. However, there are a few columns that have missing values, such as CNT\_INSTALMENT and CNT\_INSTALMENT\_FUTURE.

```

In [ ]: # Check the correlation between the variables
corr_matrix = datasets["POS_CASH_balance"].corr()
sns.heatmap(corr_matrix, annot=True, cmap='YlGnBu')

# Set the figure size
plt.figure(figsize=(17,12))

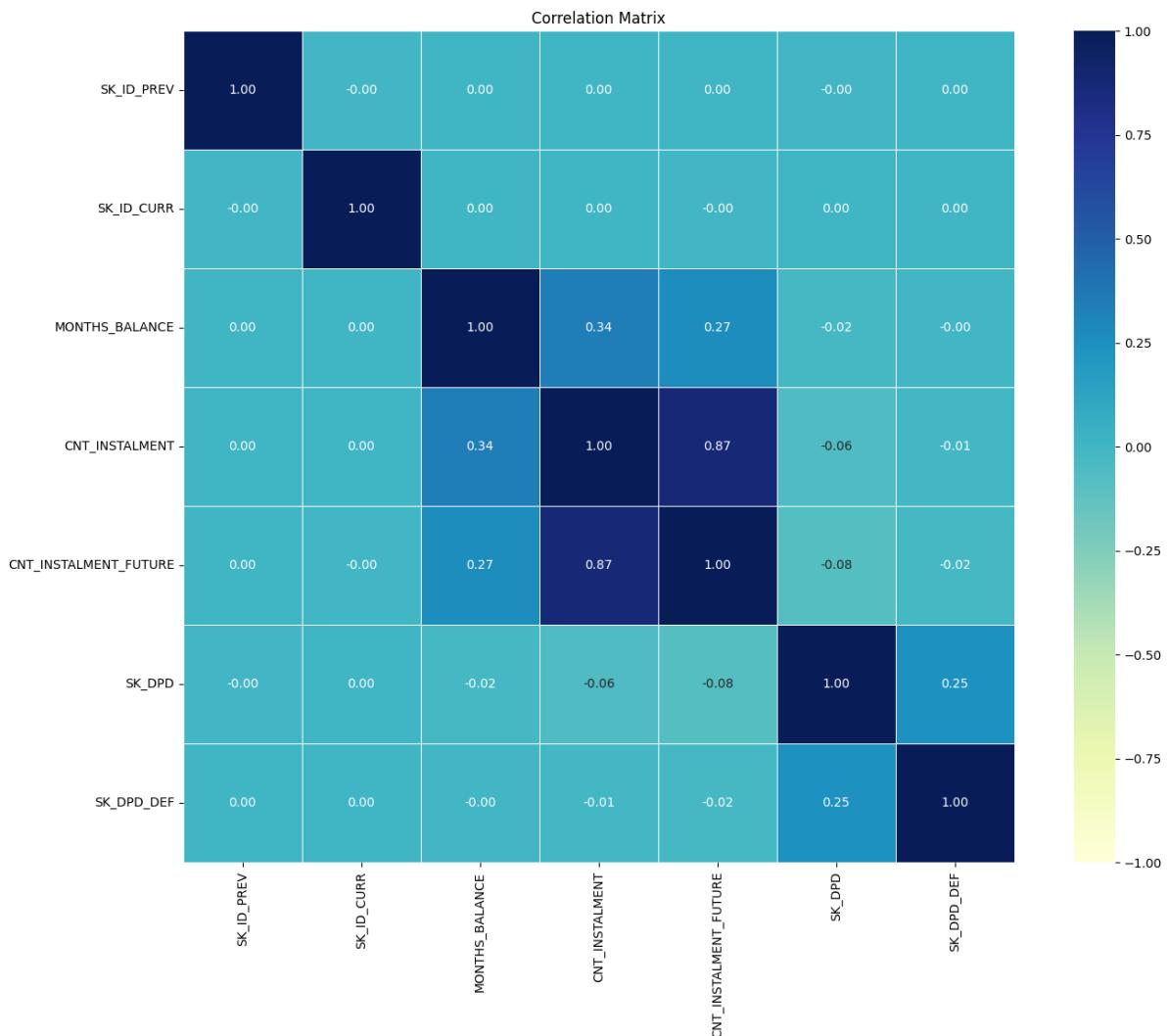
# Plot the heatmap with annotations
sns.heatmap(corr_matrix, cmap='YlGnBu', annot=True, fmt='.2f', vmin=-1, vmax=1)

# Add a title

```

```
plt.title('Correlation Matrix')
```

```
# Show the plot  
plt.show()
```



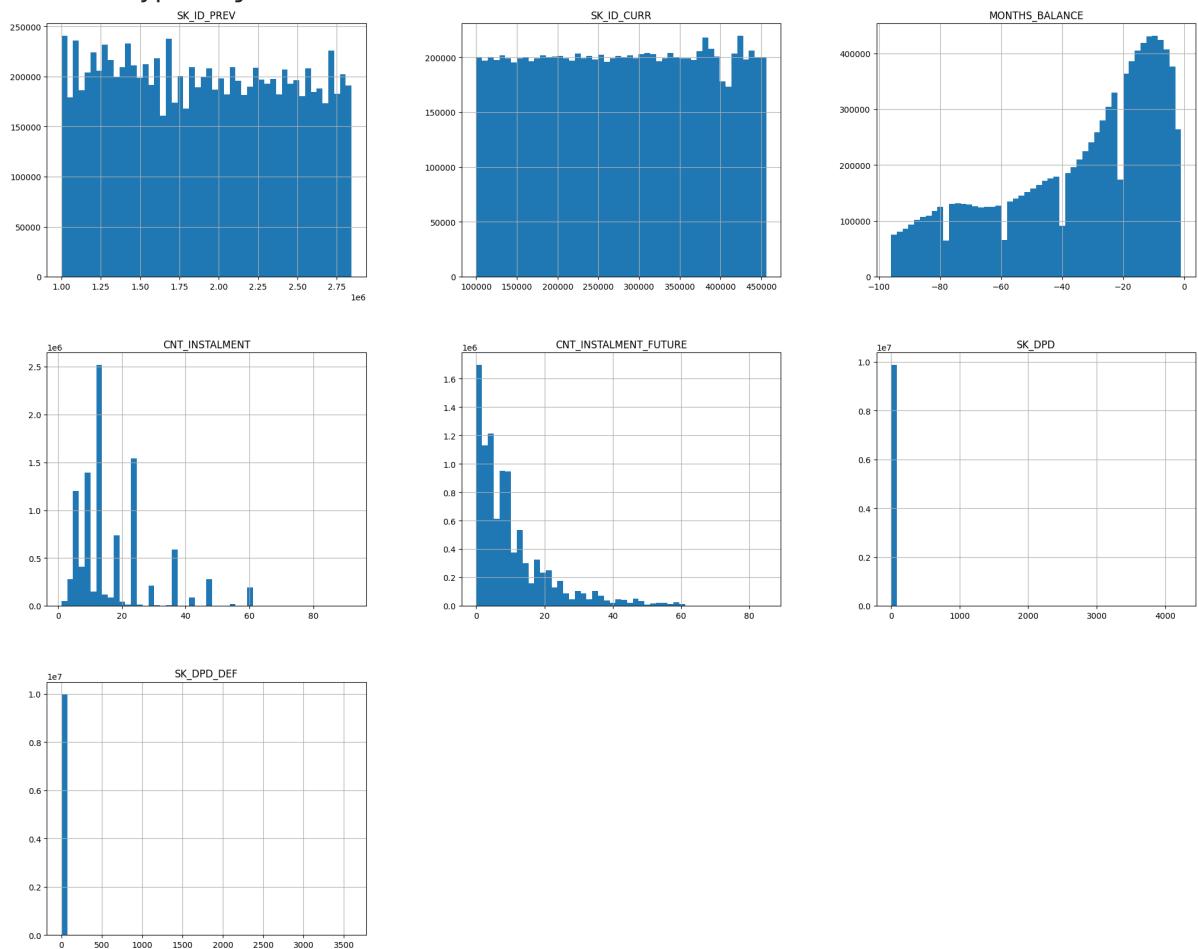
## Insights

- From EDA of the "POS\_CASH\_balance.csv" file, it appears that there is generally a weak correlation between the variables. However, the heatmap indicates that the strongest correlation is between "CNT\_INSTALMENT" and "CNT\_INSTALMENT\_FUTURE", with a positive correlation coefficient of 0.87. This indicates that as the number of months since the last update of the credit card balance increases, the number of future installments also tends to decrease. This makes intuitive sense, as customers who have fallen behind on their payments may have fewer installments left to pay.

```
In [ ]: # Checking the distribution of the variables  
datasets["POS_CASH_balance"].hist(bins=50, figsize=(25,20))
```

```
Out[ ]: array([ [,
   <Axes: title={'center': 'SK_ID_CURR'}>,
   <Axes: title={'center': 'MONTHS_BALANCE'}>],
  [

```



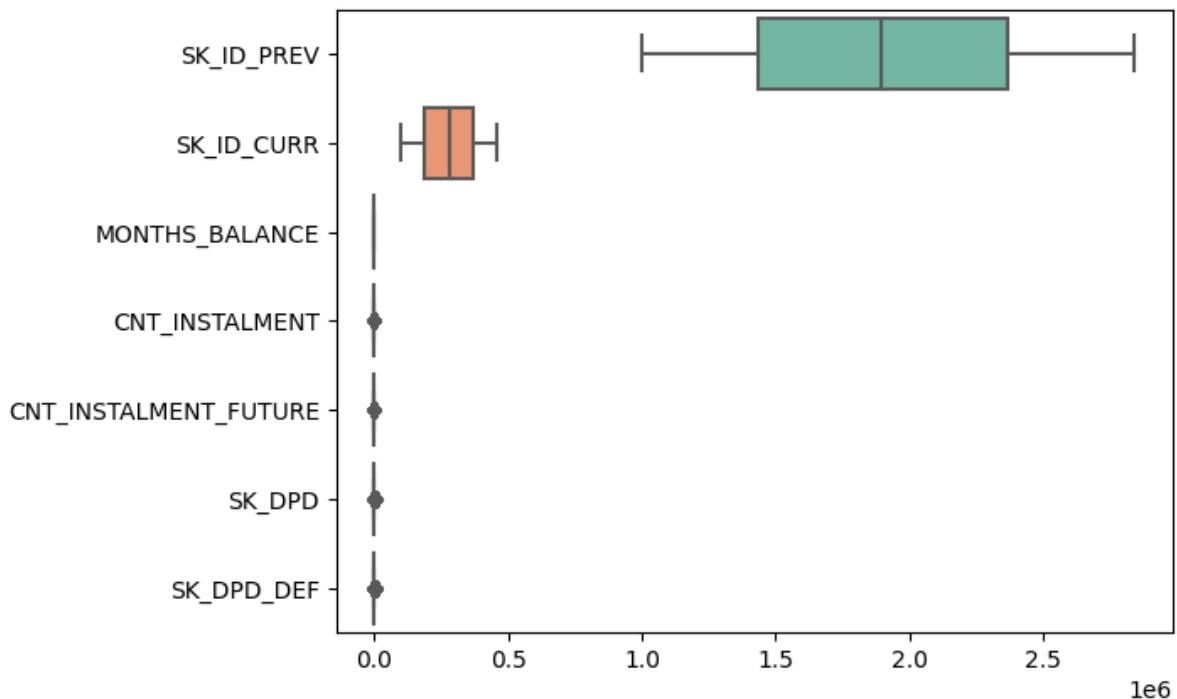
## Insights

- Most of the observations in the "MONTHS BALANCE" variable's negative skew range from -20 to 0. This shows that the most recent months have the greatest number of observations, while the older months have relatively less observations.
- The "CNT INSTALMENT" variable peaks at 12, indicating that 12 is likely to be the most typical number of installments due for the prior loan.
- The majority of the data in the "CNT INSTALMENT FUTURE" variable's positive skew range from 0 to 10. This shows that there are often less installments due on the majority of past loans in the Home Credit system.
- There is a long tail for the "SK DPD" variable, and some observations have DPD values as high as 3300. This shows that a small number of debtors have made significant payment arrears.

- The tail of the "SK DPD DEF" variable is also quite lengthy, but it is not as long as the tail of the "SK DPD" variable. This means that even after accounting for the grace period, some borrowers may still have fallen behind on their payments.
- The "POS CASH balance" dataset's variable distribution is diverse and occasionally intricate.

```
In [ ]: # Checking for outliers
sns.boxplot(data=datasets["POS_CASH_balance"], orient='h', palette='Set2')
```

Out[ ]: <Axes: >



## Insights

- The box plot shows that there are no outliers in the data that will be useful to us when training our model as only outliers visible in the box plot is in SK\_ID\_PREV and SK\_ID\_CURR which are nothing but identification number given to the customer.

## Visual EDA of POS cash balance

```
In [ ]: # Create a copy of the "POS_CASH_balance" dataset and assign it to "pos_cash_balance"
pos_cash_balance=datasets["POS_CASH_balance"].copy()

# Extract two columns, "SK_ID_CURR" and "TARGET", from the "application_train" dataset
extract_key_target=datasets["application_train"][['SK_ID_CURR', 'TARGET']]

# Merge "pos_cash_balance" and "extract_key_target" on the "SK_ID_CURR" column
```

```
merge_target = pd.merge(pos_cash_balance, extract_key_target, on='SK_ID_CURR')

# Calculate the absolute correlation between "TARGET" column and all other columns
correlation_pos_cash_balance = merge_target.corr()['TARGET'].abs().sort_values(ascending=False)
```

```
In [ ]: correlation_pos_cash_balance.head(6)
```

```
Out[ ]: TARGET           1.000000
        CNT_INSTALMENT_FUTURE    0.021972
        MONTHS_BALANCE          0.020147
        CNT_INSTALMENT           0.018506
        SK_DPD                  0.009866
        SK_DPD_DEF              0.008594
Name: TARGET, dtype: float64
```

```
In [ ]: # Extract the top 6 features with highest absolute correlation values from
top_feature = correlation_pos_cash_balance[0:6].index.tolist()

# create "pos_top_features", that contains only the columns with the top features
pos_top_features = merge_target[top_feature]

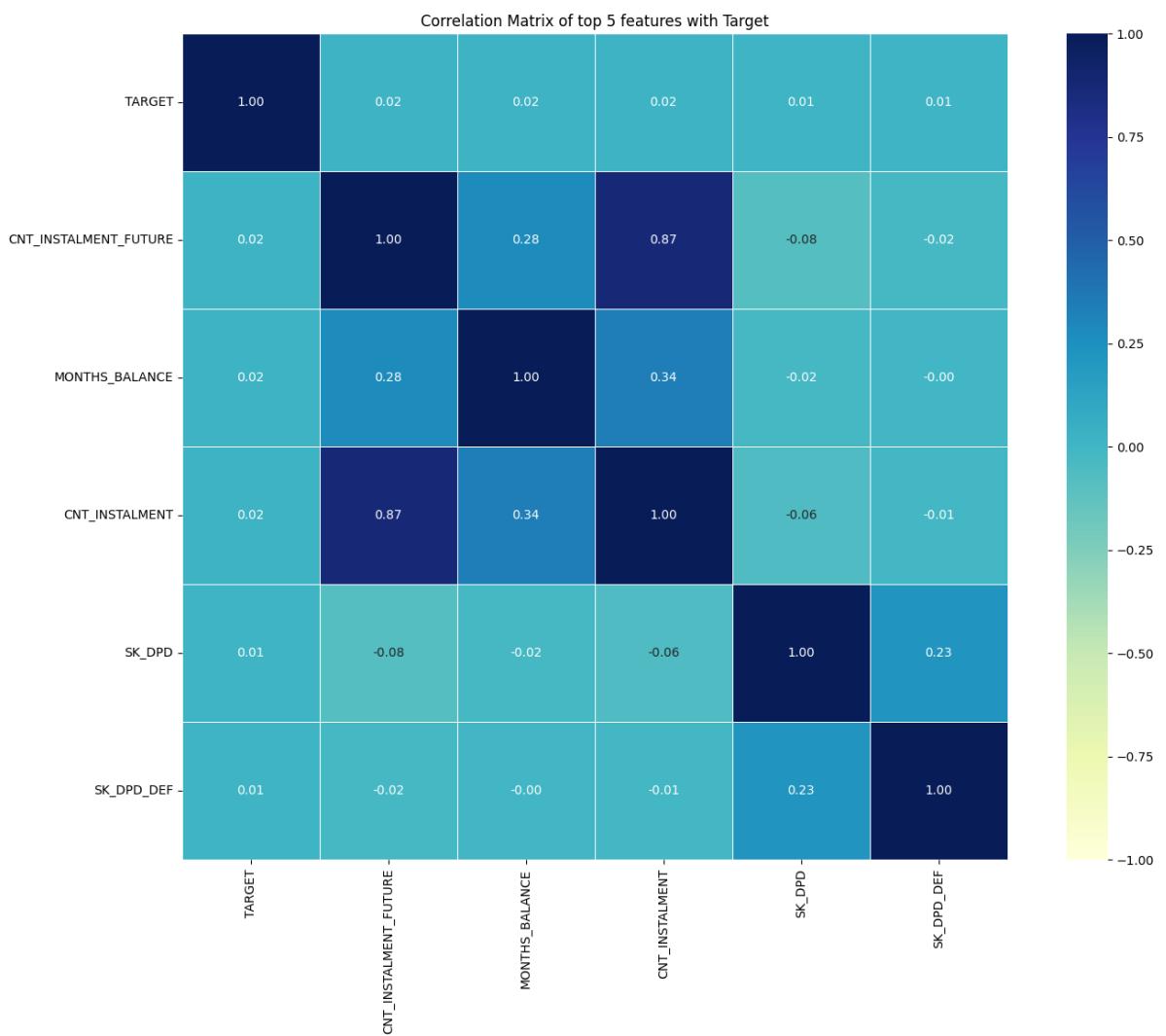
# Calculate the correlation matrix of "pos_top_features" and assign it to "corr_matrix"
corr_matrix = pos_top_features.corr()

# Set the figure size
plt.figure(figsize=(17,12))

# Plot the heatmap with annotations
sns.heatmap(corr_matrix, cmap='YlGnBu', annot=True, fmt='.2f', vmin=-1, vmax=1)

# Add a title
plt.title('Correlation Matrix of top 5 features with Target')

# Show the plot
plt.show()
```



## Pair based visualization

```
In [ ]: import seaborn as sns

# Identify the pairs of variables that have the highest positive and negative
highest_corr_pairs = corr_matrix.unstack().sort_values(ascending=False).drop_duplicates()
lowest_corr_pairs = corr_matrix.unstack().sort_values(ascending=True).drop_duplicates()

print("Pairs of variables with the highest positive correlation coefficients")
print(highest_corr_pairs)
print("\nPairs of variables with the highest negative correlation coefficients")
print(lowest_corr_pairs)

# Visualize the correlation matrix using a heatmap
sns.heatmap(corr_matrix, cmap='coolwarm')
```

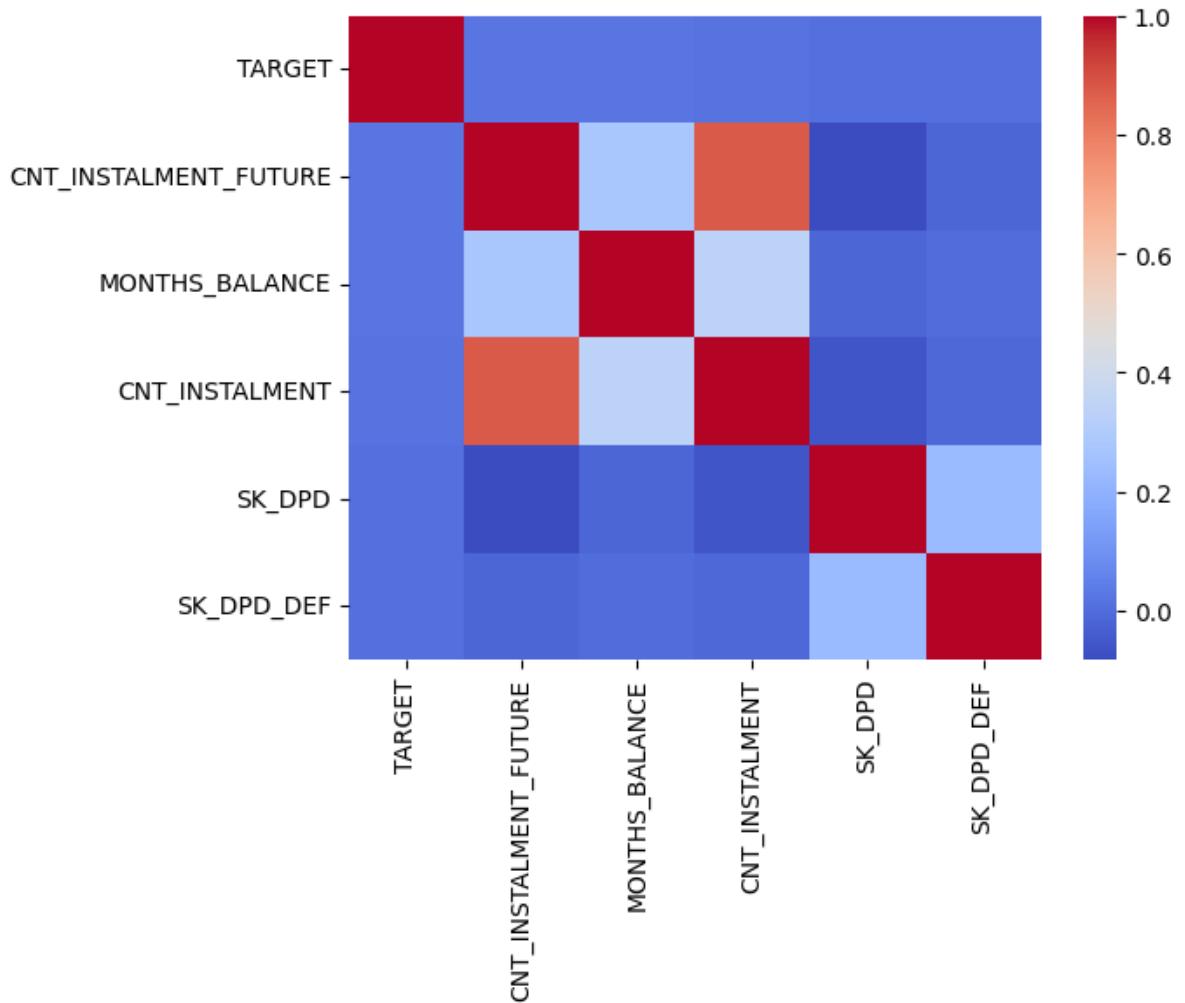
Pairs of variables with the highest positive correlation coefficients:

```
TARGET           TARGET      1.000000
CNT_INSTALMENT_FUTURE CNT_INSTALMENT  0.873742
MONTHS_BALANCE     CNT_INSTALMENT  0.340099
CNT_INSTALMENT_FUTURE MONTHS_BALANCE  0.277634
SK_DPD_DEF        SK_DPD       0.225918
dtype: float64
```

Pairs of variables with the highest negative correlation coefficients:

```
SK_DPD           CNT_INSTALMENT_FUTURE -0.082890
                  CNT_INSTALMENT      -0.061455
                  MONTHS_BALANCE      -0.017702
SK_DPD_DEF       CNT_INSTALMENT_FUTURE -0.016427
CNT_INSTALMENT   SK_DPD_DEF        -0.013403
dtype: float64
```

Out[ ]: <Axes: >



## Previous Applications

In [ ]: `datasets["previous_application"].info()`

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 1670214 entries, 0 to 1670213
Data columns (total 37 columns):
 #   Column           Non-Null Count   Dtype  
--- 
 0   SK_ID_PREV       1670214 non-null   int64  
 1   SK_ID_CURR       1670214 non-null   int64  
 2   NAME_CONTRACT_TYPE 1670214 non-null   object  
 3   AMT_ANNUITY      1297979 non-null   float64 
 4   AMT_APPLICATION  1670214 non-null   float64 
 5   AMT_CREDIT        1670213 non-null   float64 
 6   AMT_DOWN_PAYMENT  774370 non-null   float64 
 7   AMT_GOODS_PRICE   1284699 non-null   float64 
 8   WEEKDAY_APPR_PROCESS_START 1670214 non-null   object  
 9   HOUR_APPR_PROCESS_START 1670214 non-null   int64  
 10  FLAG_LAST_APPL_PER_CONTRACT 1670214 non-null   object  
 11  NFLAG_LAST_APPL_IN_DAY    1670214 non-null   int64  
 12  RATE_DOWN_PAYMENT     774370 non-null   float64 
 13  RATE_INTEREST_PRIMARY 5951 non-null    float64 
 14  RATE_INTEREST_PRIVILEGED 5951 non-null   float64 
 15  NAME_CASH_LOAN_PURPOSE 1670214 non-null   object  
 16  NAME_CONTRACT_STATUS  1670214 non-null   object  
 17  DAYS_DECISION       1670214 non-null   int64  
 18  NAME_PAYMENT_TYPE   1670214 non-null   object  
 19  CODE_REJECT_REASON  1670214 non-null   object  
 20  NAME_TYPE_SUITE     849809 non-null   object  
 21  NAME_CLIENT_TYPE    1670214 non-null   object  
 22  NAME_GOODS_CATEGORY 1670214 non-null   object  
 23  NAME_PORTFOLIO      1670214 non-null   object  
 24  NAME_PRODUCT_TYPE   1670214 non-null   object  
 25  CHANNEL_TYPE        1670214 non-null   object  
 26  SELLERPLACE_AREA    1670214 non-null   int64  
 27  NAME_SELLER_INDUSTRY 1670214 non-null   object  
 28  CNT_PAYMENT         1297984 non-null   float64 
 29  NAME_YIELD_GROUP   1670214 non-null   object  
 30  PRODUCT_COMBINATION 1669868 non-null   object  
 31  DAYS_FIRST_DRAWING 997149 non-null    float64 
 32  DAYS_FIRST_DUE     997149 non-null    float64 
 33  DAYS_LAST_DUE_1ST_VERSION 997149 non-null   float64 
 34  DAYS_LAST_DUE      997149 non-null    float64 
 35  DAYS_TERMINATION   997149 non-null    float64 
 36  NFLAG_INSURED_ON_APPROVAL 997149 non-null   float64 

dtypes: float64(15), int64(6), object(16)
memory usage: 471.5+ MB
```

```
In [ ]: # Check the first few rows of the data
print(datasets["previous_application"].head())
```

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION
\					
0	2030495	271877	Consumer loans	1730.430	17145.0
1	2802425	108129	Cash loans	25188.615	607500.0
2	2523466	122040	Cash loans	15060.735	112500.0
3	2819243	176158	Cash loans	47041.335	450000.0
4	1784265	202054	Cash loans	31924.395	337500.0
	AMT_CREDIT	AMT_DOWN_PAYMENT	AMT_GOODS_PRICE	WEEKDAY_APPR_PROCESS_START	
\					
0	17145.0	0.0	17145.0	SATURDAY	
1	679671.0	NaN	607500.0	THURSDAY	
2	136444.5	NaN	112500.0	TUESDAY	
3	470790.0	NaN	450000.0	MONDAY	
4	404055.0	NaN	337500.0	THURSDAY	
	HOUR_APPR_PROCESS_START	FLAG_LAST_APPL_PER_CONTRACT	\		
\					
0	15	Y			
1	11	Y			
2	11	Y			
3	7	Y			
4	9	Y			
	NFLAG_LAST_APPL_IN_DAY	RATE_DOWN_PAYMENT	RATE_INTEREST_PRIMARY	\	
\					
0	1	0.0	0.182832		
1	1	NaN	NaN		
2	1	NaN	NaN		
3	1	NaN	NaN		
4	1	NaN	NaN		
	RATE_INTEREST_PRIVILEGED	NAME_CASH_LOAN_PURPOSE	NAME_CONTRACT_STATUS	\	
\					
0	0.867336	XAP	Approved		
1	NaN	XNA	Approved		
2	NaN	XNA	Approved		
3	NaN	XNA	Approved		
4	NaN	Repairs	Refused		
	DAYS_DECISION	NAME_PAYMENT_TYPE	CODE_REJECT_REASON	NAME_TYPE_SUITE	
\					
0	-73	Cash through the bank	XAP	NaN	
1	-164	XNA	XAP	Unaccompanied	
2	-301	Cash through the bank	XAP	Spouse, partner	
3	-512	Cash through the bank	XAP	NaN	
4	-781	Cash through the bank	HC	NaN	
	NAME_CLIENT_TYPE	NAME_GOODS_CATEGORY	NAME_PORTFOLIO	NAME_PRODUCT_TYPE	\
\					
0	Repeater	Mobile	POS	XNA	
1	Repeater	XNA	Cash	x-sell	
2	Repeater	XNA	Cash	x-sell	
3	Repeater	XNA	Cash	x-sell	
4	Repeater	XNA	Cash	walk-in	
	CHANNEL_TYPE	SELLERPLACE_AREA	NAME_SELLER_INDUSTRY	\	
\					
0	Country-wide	35	Connectivity		
1	Contact center	-1	XNA		
2	Credit and cash offices	-1	XNA		

3	Credit and cash offices	-1	XNA	
4	Credit and cash offices	-1	XNA	
0	CNT_PAYMENT	NAME_YIELD_GROUP	PRODUCT_COMBINATION	DAYS_FIRST_DRAWI
3.0	\			NG
0	12.0	middle	POS mobile with interest	36524
1	36.0	low_action	Cash X-Sell: low	36524
2	12.0	high	Cash X-Sell: high	36524
3	12.0	middle	Cash X-Sell: middle	36524
3.0				
4	24.0	high	Cash Street: high	N
aN				
0	DAYS_FIRST_DUE	DAYS_LAST_DUE_1ST_VERSION	DAYS_LAST_DUE	DAYS_TERMINATI
7.0	\			ON
0	-42.0	300.0	-42.0	-3
1	-134.0	916.0	365243.0	36524
3.0				
2	-271.0	59.0	365243.0	36524
3.0				
3	-482.0	-152.0	-182.0	-17
7.0				
4	NaN	NaN	NaN	N
aN				
0	NFLAG_INSURED_ON_APPROVAL			
1		0.0		
2		1.0		
3		1.0		
4		1.0		
		NaN		

---

## Dataset Size

```
In [ ]: # Check the shape of the data
print(datasets["previous_application"].shape)

(1670214, 37)
```

```
In [ ]: # Check the data types of the columns
print(datasets["previous_application"].dtypes)
```

```
SK_ID_PREV                      int64
SK_ID_CURR                       int64
NAME_CONTRACT_TYPE                object
AMT_ANNUITY                        float64
AMT_APPLICATION                   float64
AMT_CREDIT                         float64
AMT_DOWN_PAYMENT                   float64
AMT_GOODS_PRICE                     float64
WEEKDAY_APPR_PROCESS_START         object
HOUR_APPR_PROCESS_START           int64
FLAG_LAST_APPL_PER_CONTRACT       object
NFLAG_LAST_APPL_IN_DAY             int64
RATE_DOWN_PAYMENT                  float64
RATE_INTEREST_PRIMARY              float64
RATE_INTEREST_PRIVILEGED           float64
NAME_CASH_LOAN_PURPOSE            object
NAME_CONTRACT_STATUS               object
DAYS_DECISION                      int64
NAME_PAYMENT_TYPE                  object
CODE_REJECT_REASON                 object
NAME_TYPE_SUITE                     object
NAME_CLIENT_TYPE                   object
NAME_GOODS_CATEGORY                 object
NAME_PORTFOLIO                      object
NAME_PRODUCT_TYPE                   object
CHANNEL_TYPE                        object
SELLERPLACE_AREA                    int64
NAME_SELLER_INDUSTRY               object
CNT_PAYMENT                         float64
NAME_YIELD_GROUP                   object
PRODUCT_COMBINATION                 object
DAYS_FIRST_DRAWING                  float64
DAYS_FIRST_DUE                      float64
DAYS_LAST_DUE_1ST_VERSION           float64
DAYS_LAST_DUE                        float64
DAYS_TERMINATION                     float64
NFLAG_INSURED_ON_APPROVAL           float64
dtype: object
```

---

## Summary statistics of the previous application

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

Out [ ]:

	SK_ID_PREV	SK_ID_CURR	AMT_ANNUITY	AMT_APPLICATION	AMT_CREDIT	AMT_GOODS_PRICE
<b>count</b>	1.670214e+06	1.670214e+06	1.297979e+06	1.670214e+06	1.670213e+06	1.670213e+06
<b>mean</b>	1.923089e+06	2.783572e+05	1.595512e+04	1.752339e+05	1.961140e+05	1.961140e+05
<b>std</b>	5.325980e+05	1.028148e+05	1.478214e+04	2.927798e+05	3.185746e+05	3.185746e+05
<b>min</b>	1.000001e+06	1.000010e+05	0.000000e+00	0.000000e+00	0.000000e+00	0.000000e+00
<b>25%</b>	1.461857e+06	1.893290e+05	6.321780e+03	1.872000e+04	2.416050e+04	2.416050e+04
<b>50%</b>	1.923110e+06	2.787145e+05	1.125000e+04	7.104600e+04	8.054100e+04	8.054100e+04
<b>75%</b>	2.384280e+06	3.675140e+05	2.065842e+04	1.803600e+05	2.164185e+05	2.164185e+05
<b>max</b>	2.845382e+06	4.562550e+05	4.180581e+05	6.905160e+06	6.905160e+06	6.905160e+06

In [ ]: `datasets["previous_application"].describe(include='all') #numerical and categorical`

Out [ ]:

	SK_ID_PREV	SK_ID_CURR	NAME_CONTRACT_TYPE	AMT_ANNUITY	AMT_APPLICATION	CNT_PAYMENT
<b>count</b>	1.670214e+06	1.670214e+06		1670214	1.297979e+06	1.670213e+06
<b>unique</b>		NaN	NaN		4	NaN
<b>top</b>		NaN	NaN	Cash loans		NaN
<b>freq</b>		NaN	NaN	747553		NaN
<b>mean</b>	1.923089e+06	2.783572e+05		NaN	1.595512e+04	1.752339e+05
<b>std</b>	5.325980e+05	1.028148e+05		NaN	1.478214e+04	2.927798e+05
<b>min</b>	1.000001e+06	1.000010e+05		NaN	0.000000e+00	0.000000e+00
<b>25%</b>	1.461857e+06	1.893290e+05		NaN	6.321780e+03	1.872000e+04
<b>50%</b>	1.923110e+06	2.787145e+05		NaN	1.125000e+04	7.104600e+04
<b>75%</b>	2.384280e+06	3.675140e+05		NaN	2.065842e+04	1.803600e+05
<b>max</b>	2.845382e+06	4.562550e+05		NaN	4.180581e+05	6.905160e+06

## Summary of Missing data for Previous Applications

In [ ]: `# Check for missing data  
print(datasets["previous_application"].isnull().sum())`

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

```
In [ ]: # Calculate the percentage of missing data
missing_percentage = datasets["previous_application"].isnull().sum() / len(datasets)
print(missing_percentage)
```

```
SK_ID_PREV           0.000000
SK_ID_CURR          0.000000
NAME_CONTRACT_TYPE  0.000000
AMT_ANNUITY         22.286665
AMT_APPLICATION    0.000000
AMT_CREDIT          0.000060
AMT_DOWN_PAYMENT   53.636480
AMT_GOODS_PRICE     23.081773
WEEKDAY_APPR_PROCESS_START 0.000000
HOUR_APPR_PROCESS_START 0.000000
FLAG_LAST_APPL_PER_CONTRACT 0.000000
NFLAG_LAST_APPL_IN_DAY 0.000000
RATE_DOWN_PAYMENT  53.636480
RATE_INTEREST_PRIMARY 99.643698
RATE_INTEREST_PRIVILEGED 99.643698
NAME_CASH_LOAN_PURPOSE 0.000000
NAME_CONTRACT_STATUS 0.000000
DAYS_DECISION       0.000000
NAME_PAYMENT_TYPE   0.000000
CODE_REJECT_REASON 0.000000
NAME_TYPE_SUITE     49.119754
NAME_CLIENT_TYPE   0.000000
NAME_GOODS_CATEGORY 0.000000
NAME_PORTFOLIO      0.000000
NAME_PRODUCT_TYPE   0.000000
CHANNEL_TYPE        0.000000
SELLERPLACE_AREA   0.000000
NAME_SELLER_INDUSTRY 0.000000
CNT_PAYMENT         22.286366
NAME_YIELD_GROUP   0.000000
PRODUCT_COMBINATION 0.020716
DAYS_FIRST_DRAWING 40.298129
DAYS_FIRST_DUE     40.298129
DAYS_LAST_DUE_1ST_VERSION 40.298129
DAYS_LAST_DUE      40.298129
DAYS_TERMINATION   40.298129
NFLAG_INSURED_ON_APPROVAL 40.298129
dtype: float64
```

```
In [ ]: # Combining the code to print the percent of missing value and missing value
percent = (datasets["previous_application"].isnull().sum()/datasets["previous_application"].shape[0]).sort_values(ascending=True)
sum_missing = datasets["previous_application"].isna().sum().sort_values(ascending=True)
missing_application_train_data = pd.concat([percent, sum_missing], axis=1, sort=False)
missing_application_train_data.head(20)
```

Out[ ]:

	Percent	Missing Value Count
RATE_INTEREST_PRIVILEGED	99.64	1664263
RATE_INTEREST_PRIMARY	99.64	1664263
AMT_DOWN_PAYMENT	53.64	895844
RATE_DOWN_PAYMENT	53.64	895844
NAME_TYPE_SUITE	49.12	820405
NFLAG_INSURED_ON_APPROVAL	40.30	673065
DAYS_TERMINATION	40.30	673065
DAYS_LAST_DUE	40.30	673065
DAYS_LAST_DUE_1ST_VERSION	40.30	673065
DAYS_FIRST_DUE	40.30	673065
DAYS_FIRST_DRAWING	40.30	673065
AMT_GOODS_PRICE	23.08	385515
AMT_ANNUITY	22.29	372235
CNT_PAYMENT	22.29	372230
PRODUCT_COMBINATION	0.02	346
AMT_CREDIT	0.00	1
NAME_YIELD_GROUP	0.00	0
NAME_PORTFOLIO	0.00	0
NAME_SELLER_INDUSTRY	0.00	0
SELLERPLACE_AREA	0.00	0

In [ ]:

```
df = pd.DataFrame(datasets["previous_application"])

# Select only columns with missing values
cols_with_missing = df.columns[df.isnull().sum() > 0]

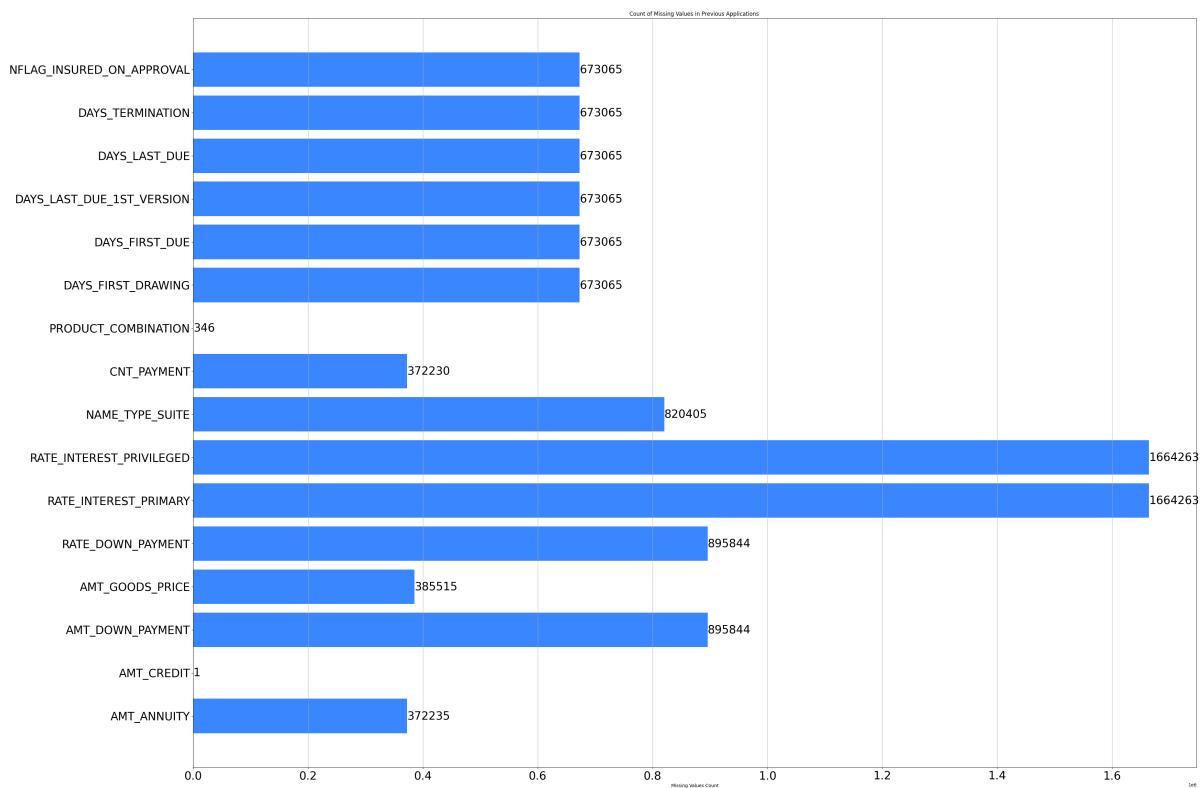
# Count the number of missing values in each column
missing_values_count = df[cols_with_missing].isnull().sum()

# Create a horizontal bar chart of the count of missing values for each column
plt.figure(figsize=(40, 30))
plt.barrh(missing_values_count.index, missing_values_count.values, color='red')
plt.xlabel('Missing Values Count')
plt.title('Count of Missing Values in Previous Applications ')
plt.grid(axis='x')

# Add data labels to the bar chart
for i, v in enumerate(missing_values_count.values):
    plt.text(v + 50, i, str(v), color='black', fontsize=25, ha='left', va='center')

plt.xticks(fontsize=25)
```

```
plt.yticks(fontsize=25)
plt.show()
```



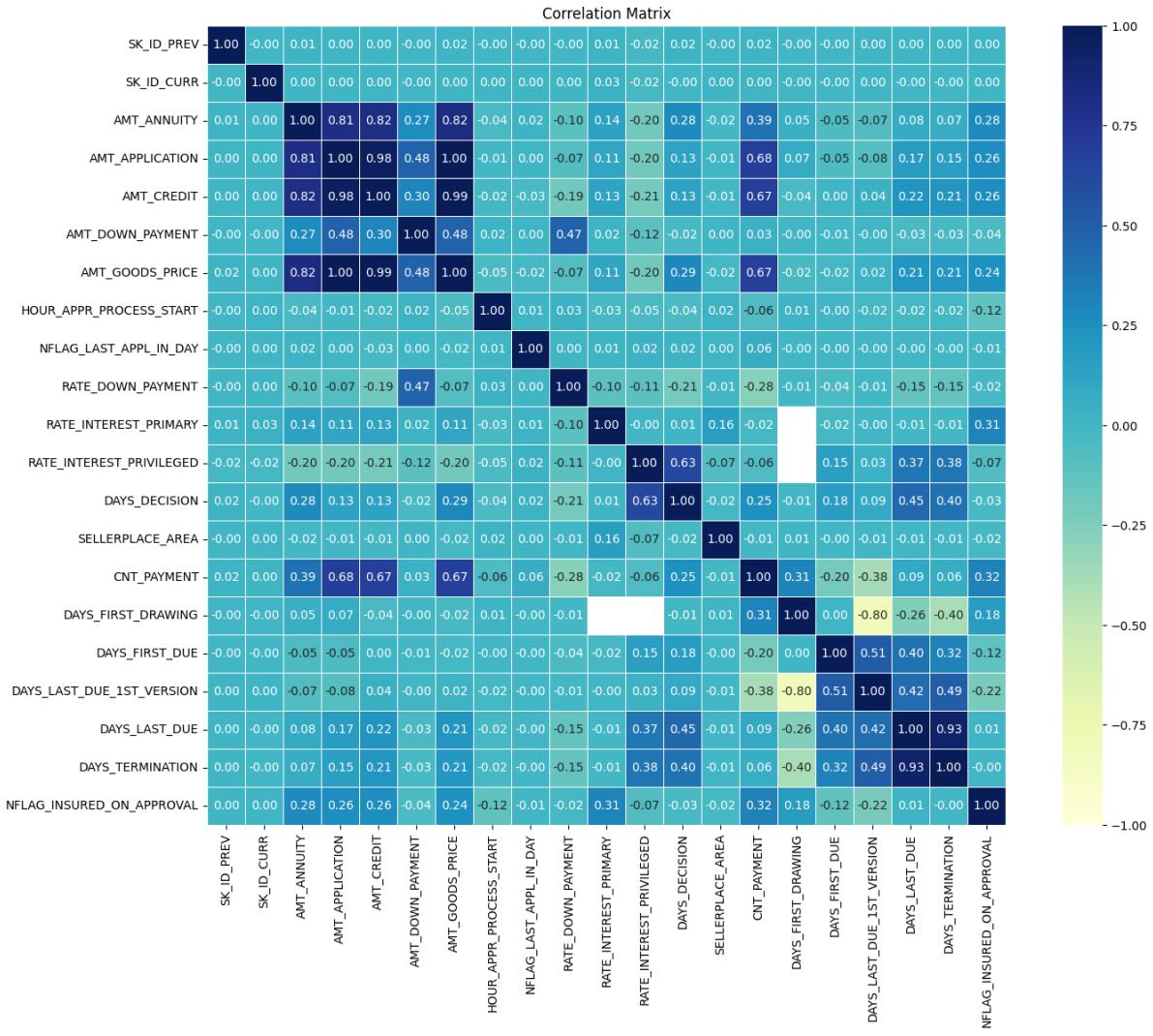
```
In [ ]: #Check the correlation between the variables
corr_matrix = datasets["previous_application"].corr()
# sns.heatmap(corr_matrix, annot=True, cmap='YlGnBu')

# Set the figure size
plt.figure(figsize=(17,12))

# Plot the heatmap with annotations
sns.heatmap(corr_matrix, cmap='YlGnBu', annot=True, fmt='.2f', vmin=-1, vmax=1)

# Add a title
plt.title('Correlation Matrix')

# Show the plot
plt.show()
```

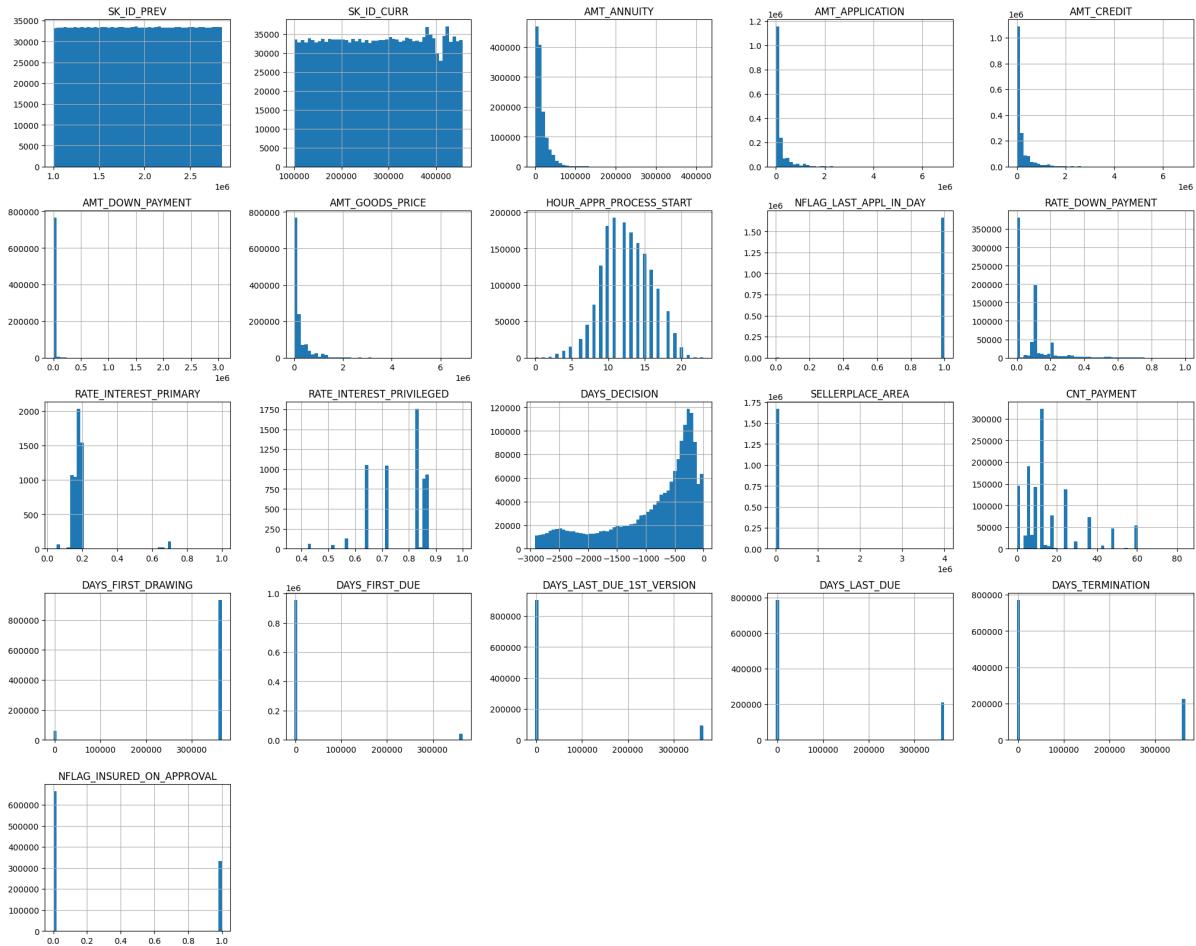


The above correlation matrix shows that the variables are diversely correlated. For example, the highest positive correlation is between AMT\_CREDIT and AMT\_GOODS\_PRICE that is 0.99, whereas there is also negative correlation between DAYS\_LAST\_DUE\_1st\_VERSION and DAYS\_FIRST\_DRAWING

```
In [ ]: # Check the distribution of the variables
datasets["previous_application"].hist(bins=50, figsize=(25,20))
```

```
Out[ ]: array([ [,
 <Axes: title={'center': 'SK_ID_CURR'}>,
 <Axes: title={'center': 'AMT_ANNUITY'}>,
 <Axes: title={'center': 'AMT_APPLICATION'}>,
 <Axes: title={'center': 'AMT_CREDIT'}>],
 [

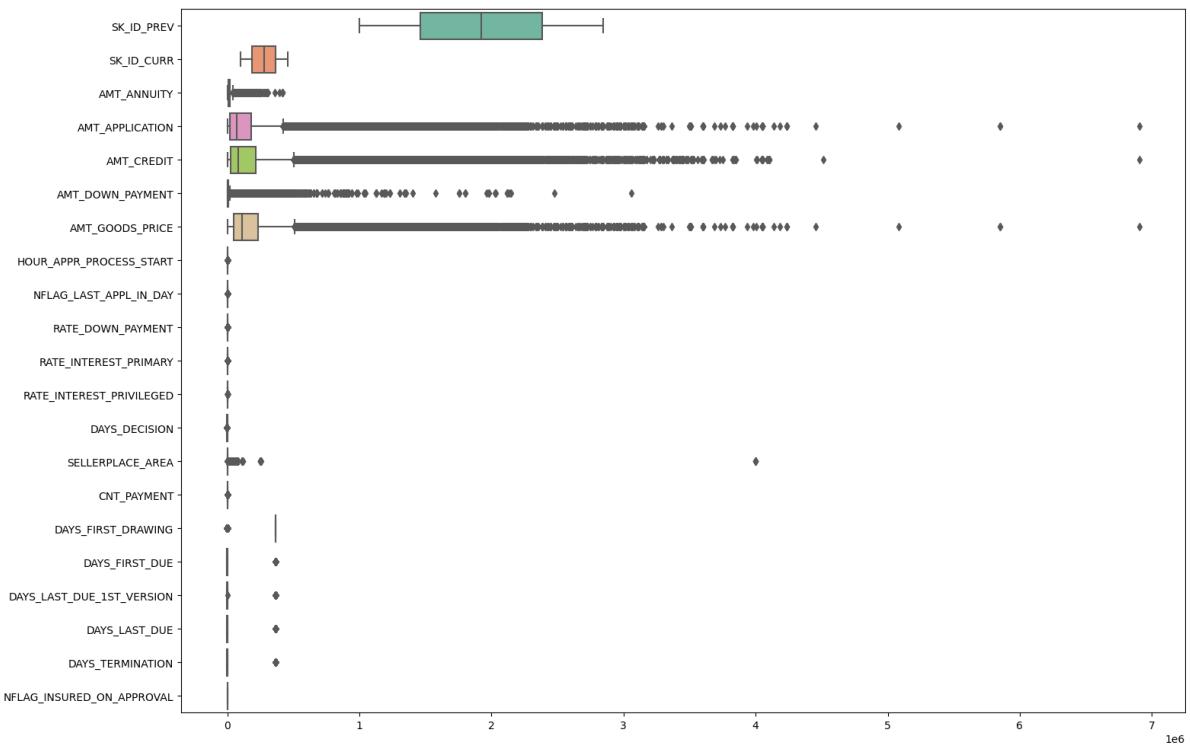
```



```
In [ ]: # Set the figure size
plt.figure(figsize=(17,12))
```

```
# Check for outliers
sns.boxplot(data=datasets["previous_application"], orient='h', palette='Set2')
```

Out[ ]: <Axes: >



## Insights

- The above graph shows the outliers of all the variables in previous\_application. We can infer that their are more outlier in the SK\_ID\_PREV and SK\_ID\_CURR but as they are only identification number we do not need to process those as we won't be using those features while training the model

## Visual EDA of Previous applicants

```
In [ ]: previous_application=datasets["previous_application"].copy()

merge_target = pd.merge(previous_application, extract_key_target, on='SK_ID_CURR')

correlation_previous_application = merge_target.corr() ['TARGET'].abs().sort_
```

In [ ]: correlation\_previous\_application.head(6)

Out[ ]:

	1.000000
TARGET	1.000000
DAYS_DECISION	0.039901
DAYS_FIRST_DRAWING	0.031154
CNT_PAYMENT	0.030480
RATE_INTEREST_PRIVILEGED	0.028640
HOUR_APPR_PROCESS_START	0.027809

Name: TARGET, dtype: float64

```
In [ ]: top_feature = correlation_previous_application[0:6].index.tolist()

previous_application_top_features = merge_target[top_feature]

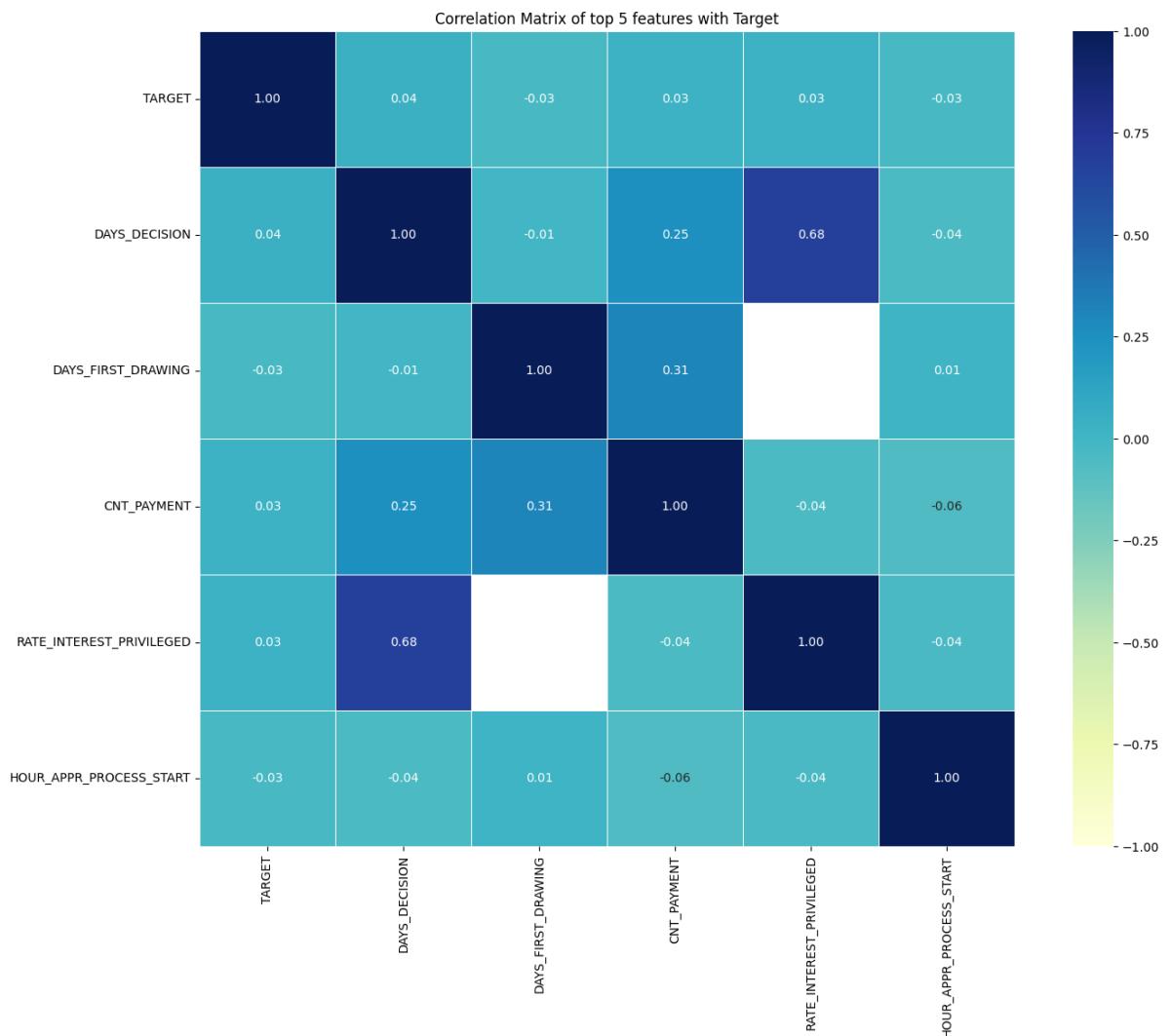
corr_matrix = previous_application_top_features.corr()

plt.figure(figsize=(17,12))

sns.heatmap(corr_matrix, cmap='YlGnBu', annot=True, fmt='.2f', vmin=-1, vmax=1)

plt.title('Correlation Matrix of top 5 features with Target')

plt.show()
```



## Pair based visualization

```
In [ ]: import seaborn as sns

# Identify the pairs of variables that have the highest positive and negative correlations
highest_corr_pairs = corr_matrix.unstack().sort_values(ascending=False).drop
```

```
lowest_corr_pairs = corr_matrix.unstack().sort_values(ascending=True).drop_d

print("Pairs of variables with the highest positive correlation coefficients")
print(highest_corr_pairs)
print("\nPairs of variables with the highest negative correlation coefficient")
print(lowest_corr_pairs)

# Visualize the correlation matrix using a heatmap
sns.heatmap(corr_matrix, cmap='coolwarm')
```

Pairs of variables with the highest positive correlation coefficients:

TARGET	TARGET	1.000000
RATE_INTEREST_PRIVILEGED	DAYS_DECISION	0.677183
DAYS_FIRST_DRAWING	CNT_PAYMENT	0.309294
DAYS_DECISION	CNT_PAYMENT	0.253460
	TARGET	0.039901

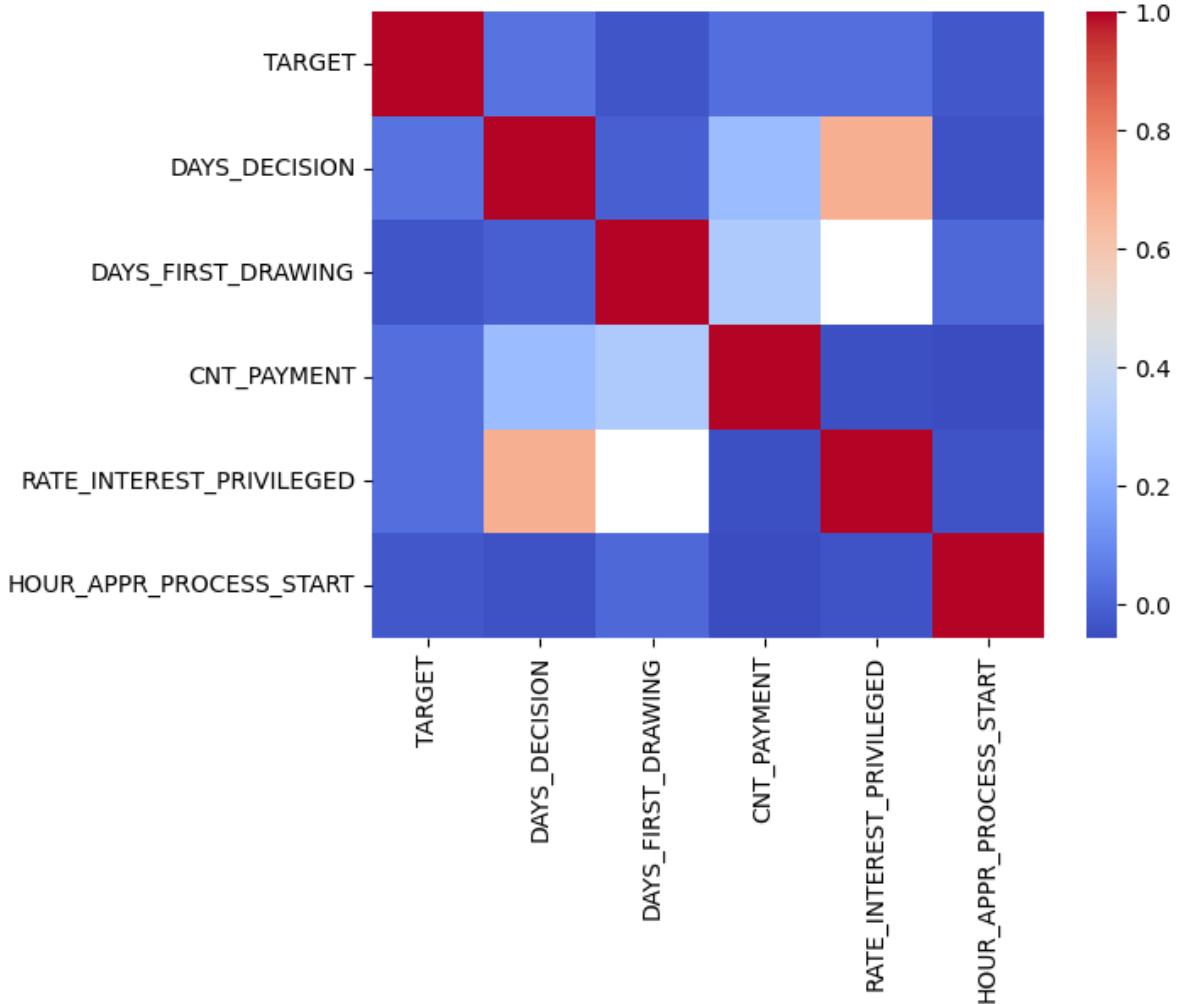
dtype: float64

Pairs of variables with the highest negative correlation coefficients:

CNT_PAYMENT	HOUR_APPR_PROCESS_START	-0.055198
RATE_INTEREST_PRIVILEGED	CNT_PAYMENT	-0.044796
HOUR_APPR_PROCESS_START	DAYS_DECISION	-0.040456
RATE_INTEREST_PRIVILEGED	HOUR_APPR_PROCESS_START	-0.038340
TARGET	DAYS_FIRST_DRAWING	-0.031154

dtype: float64

Out[ ]: <Axes: >



## Installments Payments

```
In [ ]: datasets["installments_payments"].info()

<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
```

```
In [ ]: # Check the first few rows of the data
```

```
print(datasets["installments_payments"].head( ))
```

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

  
	DAYS\_INSTALMENT	DAYS\_ENTRY\_PAYMENT	AMT\_INSTALMENT	AMT\_PAYMENT
0	-1180.0	-1187.0	6948.360	6948.360
1	-2156.0	-2156.0	1716.525	1716.525
2	-63.0	-63.0	25425.000	25425.000
3	-2418.0	-2426.0	24350.130	24350.130
4	-1383.0	-1366.0	2165.040	2160.585

---

## Dataset Size

```
In [ ]: # Check the shape of the data  
print(datasets["installments_payments"].shape)
```

(13605401, 8)

```
In [ ]: # Check the data types of the columns  
print(datasets["installments_payments"].dtypes)
```

SK_ID_PREV	int64
SK_ID_CURR	int64
NUM_INSTALMENT_VERSION	float64
NUM_INSTALMENT_NUMBER	int64
DAYS_INSTALMENT	float64
DAYS_ENTRY_PAYMENT	float64
AMT_INSTALMENT	float64
AMT_PAYMENT	float64
dtype:	object

---

## Summary statistics of the Installments Payments

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

Out [ ]:

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER
<b>count</b>	1.360540e+07	1.360540e+07	1.360540e+07	1.360540e+07
<b>mean</b>	1.903365e+06	2.784449e+05	8.566373e-01	1.887090e+00
<b>std</b>	5.362029e+05	1.027183e+05	1.035216e+00	2.666407e+00
<b>min</b>	1.000001e+06	1.000010e+05	0.000000e+00	1.000000e+00
<b>25%</b>	1.434191e+06	1.896390e+05	0.000000e+00	4.000000e+00
<b>50%</b>	1.896520e+06	2.786850e+05	1.000000e+00	8.000000e+00
<b>75%</b>	2.369094e+06	3.675300e+05	1.000000e+00	1.900000e+00
<b>max</b>	2.843499e+06	4.562550e+05	1.780000e+02	2.770000e+00

In [ ]: `datasets["installments_payments"].describe(include='all') #numerical and cat`

Out [ ]:

	SK_ID_PREV	SK_ID_CURR	NUM_INSTALMENT_VERSION	NUM_INSTALMENT_NUMBER
<b>count</b>	1.360540e+07	1.360540e+07	1.360540e+07	1.360540e+07
<b>mean</b>	1.903365e+06	2.784449e+05	8.566373e-01	1.887090e+00
<b>std</b>	5.362029e+05	1.027183e+05	1.035216e+00	2.666407e+00
<b>min</b>	1.000001e+06	1.000010e+05	0.000000e+00	1.000000e+00
<b>25%</b>	1.434191e+06	1.896390e+05	0.000000e+00	4.000000e+00
<b>50%</b>	1.896520e+06	2.786850e+05	1.000000e+00	8.000000e+00
<b>75%</b>	2.369094e+06	3.675300e+05	1.000000e+00	1.900000e+00
<b>max</b>	2.843499e+06	4.562550e+05	1.780000e+02	2.770000e+00

## Summary of Missing data for Installments Payments

In [ ]: `# Check for missing data  
print(datasets["installments_payments"].isnull().sum())`

```
SK_ID_PREV          0  
SK_ID_CURR         0  
NUM_INSTALMENT_VERSION 0  
NUM_INSTALMENT_NUMBER 0  
DAYS_INSTALMENT    0  
DAYS_ENTRY_PAYMENT 2905  
AMT_INSTALMENT      0  
AMT_PAYMENT         2905  
dtype: int64
```

In [ ]: `# Calculate the percentage of missing data  
missing_percentage = datasets["installments_payments"].isnull().sum() / len(`

```
print(missing_percentage)

SK_ID_PREV           0.000000
SK_ID_CURR           0.000000
NUM_INSTALMENT_VERSION 0.000000
NUM_INSTALMENT_NUMBER   0.000000
DAYS_INSTALMENT        0.000000
DAYS_ENTRY_PAYMENT     0.021352
AMT_INSTALMENT          0.000000
AMT_PAYMENT             0.021352
dtype: float64
```

In [ ]: # Combining the code to print the percent of missing value and missing value  
percent = (datasets["installments\_payments"].isnull().sum()/datasets["installments\_payments"].shape[0]).reset\_index()  
sum\_missing = datasets["installments\_payments"].isna().sum().sort\_values(ascending=True)  
missing\_application\_train\_data = pd.concat([percent, sum\_missing], axis=1, ignore\_index=True)  
missing\_application\_train\_data.head(20)

Out[ ]:

	Percent	Missing Value Count
DAYS_ENTRY_PAYMENT	0.02	2905
AMT_PAYMENT	0.02	2905
SK_ID_PREV	0.00	0
SK_ID_CURR	0.00	0
NUM_INSTALMENT_VERSION	0.00	0
NUM_INSTALMENT_NUMBER	0.00	0
DAYS_INSTALMENT	0.00	0
AMT_INSTALMENT	0.00	0

In [ ]: import pandas as pd  
df = pd.DataFrame(datasets["installments\_payments"])

# Select only columns with missing values
cols\_with\_missing = df.columns[df.isnull().sum() > 0]

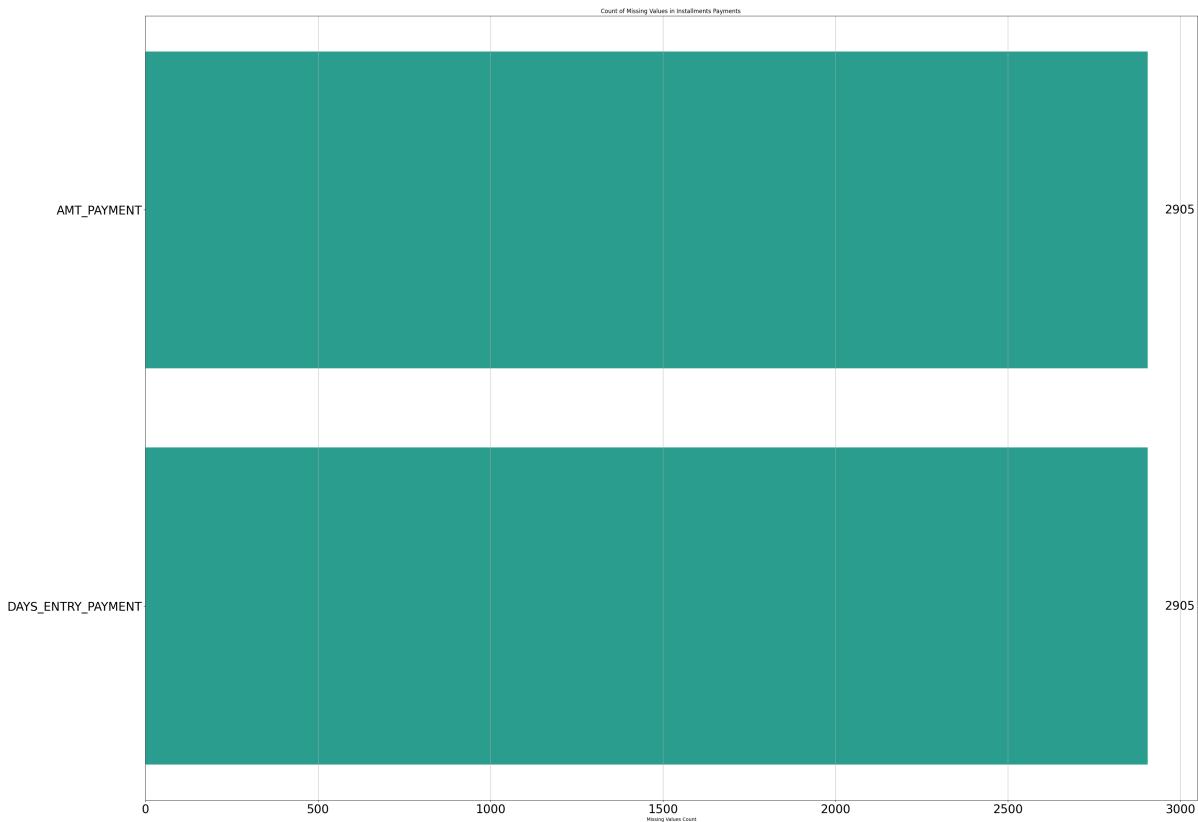
# Count the number of missing values in each column
missing\_values\_count = df[cols\_with\_missing].isnull().sum()

# Create a horizontal bar chart of the count of missing values for each column
plt.figure(figsize=(40, 30))
plt.barrh(missing\_values\_count.index, missing\_values\_count.values, color='red')
plt.xlabel('Missing Values Count')
plt.title('Count of Missing Values in Installments Payments ')
plt.grid(axis='x')

# Add data labels to the bar chart
for i, v in enumerate(missing\_values\_count.values):
 plt.text(v + 50, i, str(v), color='black', fontsize=25, ha='left', va='center')

plt.xticks(fontsize=25)

```
plt.yticks(fontsize=25)
plt.show()
```



## Insights

- The above plot shows that the installment payments file only contains 2 columns that have missing values the AMT\_PAYMENT and DAYS\_ENTRY\_PAYMENT that contains 2905 and 2905 colmuns respectively.

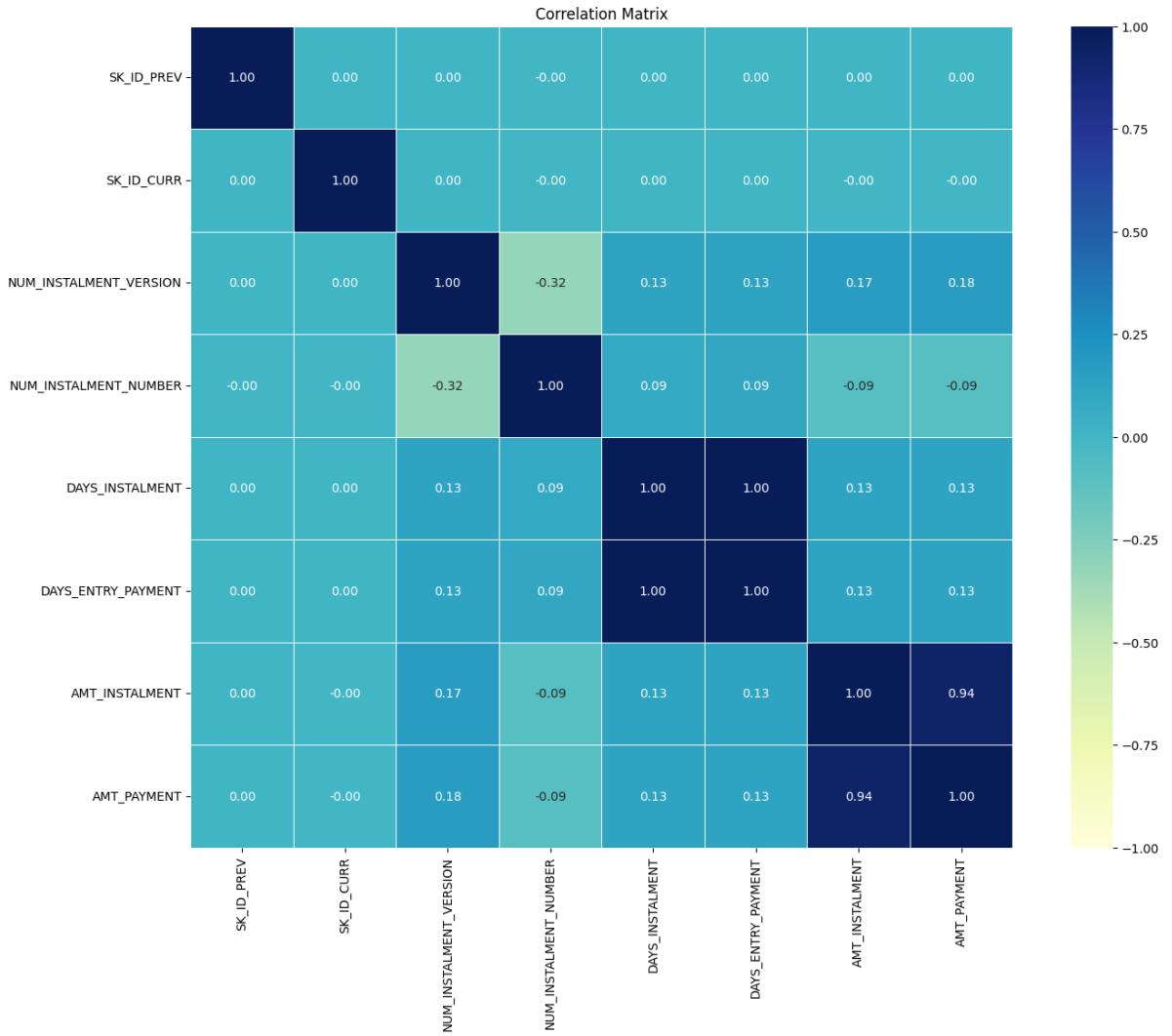
```
In [ ]: # Check the correlation between the variables
corr_matrix = datasets["installments_payments"].corr()
#sns.heatmap(corr_matrix, annot=True, cmap='YlGnBu')

# Set the figure size
plt.figure(figsize=(17,12))

# Plot the heatmap with annotations
sns.heatmap(corr_matrix, cmap='YlGnBu', annot=True, fmt='.2f', vmin=-1, vmax=1)

# Add a title
plt.title('Correlation Matrix')

# Show the plot
plt.show()
```

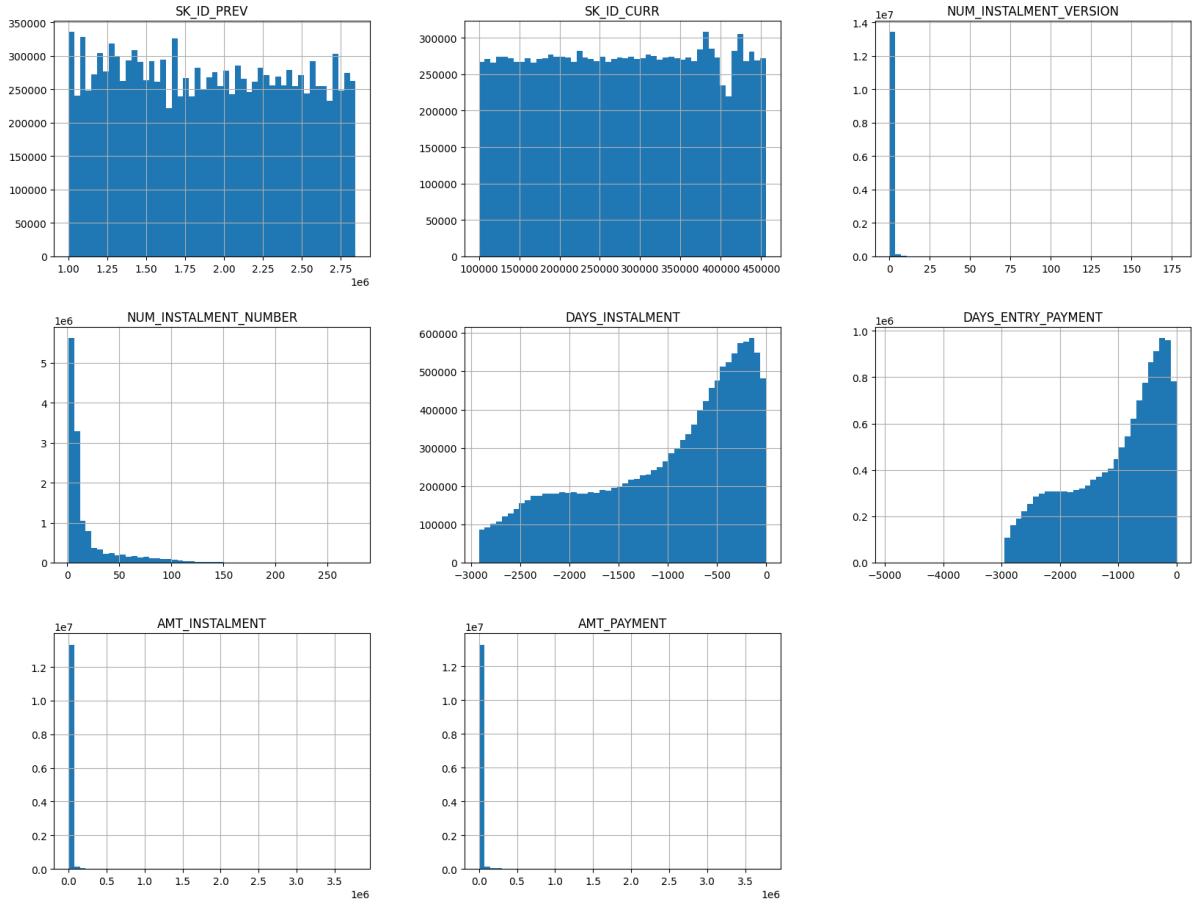


## Insights

- Based on the correlation matrix we can say that the variables are moderately correlated with each other. The greatest positive correlation is 0.94 between AMT\_INSTALMENT and AMT\_PAYMENT whereas the greatest negative correlation is -0.32 between NUM\_INSTALMENT\_VERSION and NUM\_INSTALMENT\_NUMBER

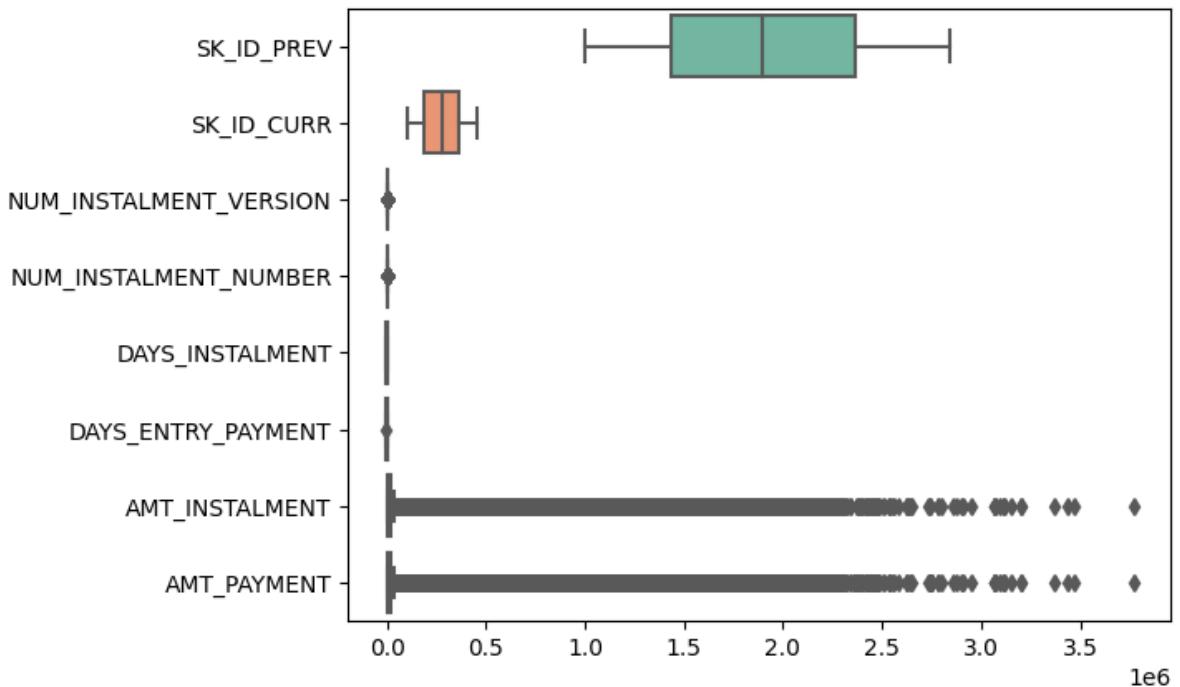
```
In [ ]: # Check the distribution of the variables
datasets["installments_payments"].hist(bins=50, figsize=(20,15))
```

```
Out[ ]: array([[[<Axes: title={'center': 'SK_ID_PREV'}>,
    <Axes: title={'center': 'SK_ID_CURR'}>,
    <Axes: title={'center': 'NUM_INSTALMENT_VERSION'}>],
   [<Axes: title={'center': 'NUM_INSTALMENT_NUMBER'}>,
    <Axes: title={'center': 'DAYS_INSTALMENT'}>,
    <Axes: title={'center': 'DAYS_ENTRY_PAYMENT'}>],
   [<Axes: title={'center': 'AMT_INSTALMENT'}>,
    <Axes: title={'center': 'AMT_PAYMENT'}>, <Axes: >]], dtype=object)
```



```
In [ ]: # Check for outliers
sns.boxplot(data=datasets["installments_payments"], orient='h', palette='Set1')
```

```
Out[ ]: <Axes: >
```



## Insights

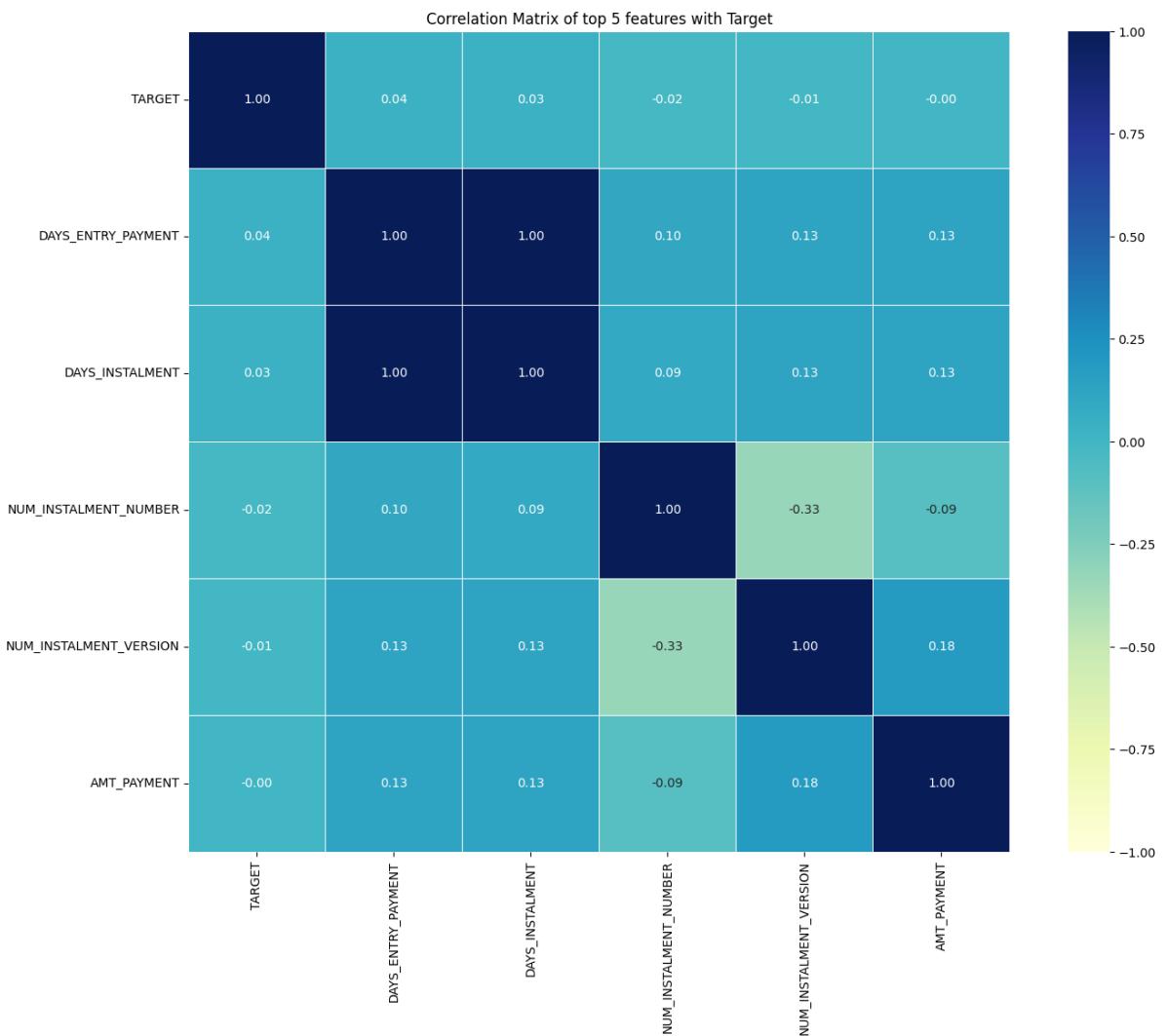
- The box plot above shows that there are outlier in SK\_ID\_PREV and there are no other outlier in any other column

```
In [ ]: installments_payments=datasets["installments_payments"].copy()  
merge_target = pd.merge(installments_payments,extract_key_target,on='SK_ID_C  
correlation_installments_payments = merge_target.corr() ['TARGET'].abs().sort
```

```
In [ ]: correlation_installments_payments.head(6)
```

```
Out[ ]: TARGET          1.000000  
DAYS_ENTRY_PAYMENT    0.035122  
DAYS_INSTALMENT        0.034974  
NUM_INSTALMENT_NUMBER  0.016190  
NUM_INSTALMENT_VERSION 0.009896  
AMT_PAYMENT            0.003623  
Name: TARGET, dtype: float64
```

```
In [ ]: top_feature = correlation_installments_payments[0:6].index.tolist()  
  
installments_payments_top_features = merge_target[top_feature]  
  
corr_matrix = installments_payments_top_features.corr()  
  
plt.figure(figsize=(17,12))  
  
sns.heatmap(corr_matrix, cmap='YlGnBu', annot=True, fmt='.2f', vmin=-1, vmax  
plt.title('Correlation Matrix of top 5 features with Target')  
plt.show()
```



## Pair based visualization

```
In [ ]: import seaborn as sns

# Identify the pairs of variables that have the highest positive and negative
highest_corr_pairs = corr_matrix.unstack().sort_values(ascending=False).drop_duplicates()
lowest_corr_pairs = corr_matrix.unstack().sort_values(ascending=True).drop_duplicates()

print("Pairs of variables with the highest positive correlation coefficients")
print(highest_corr_pairs)
print("\nPairs of variables with the highest negative correlation coefficients")
print(lowest_corr_pairs)

# Visualize the correlation matrix using a heatmap
sns.heatmap(corr_matrix, cmap='coolwarm')
```

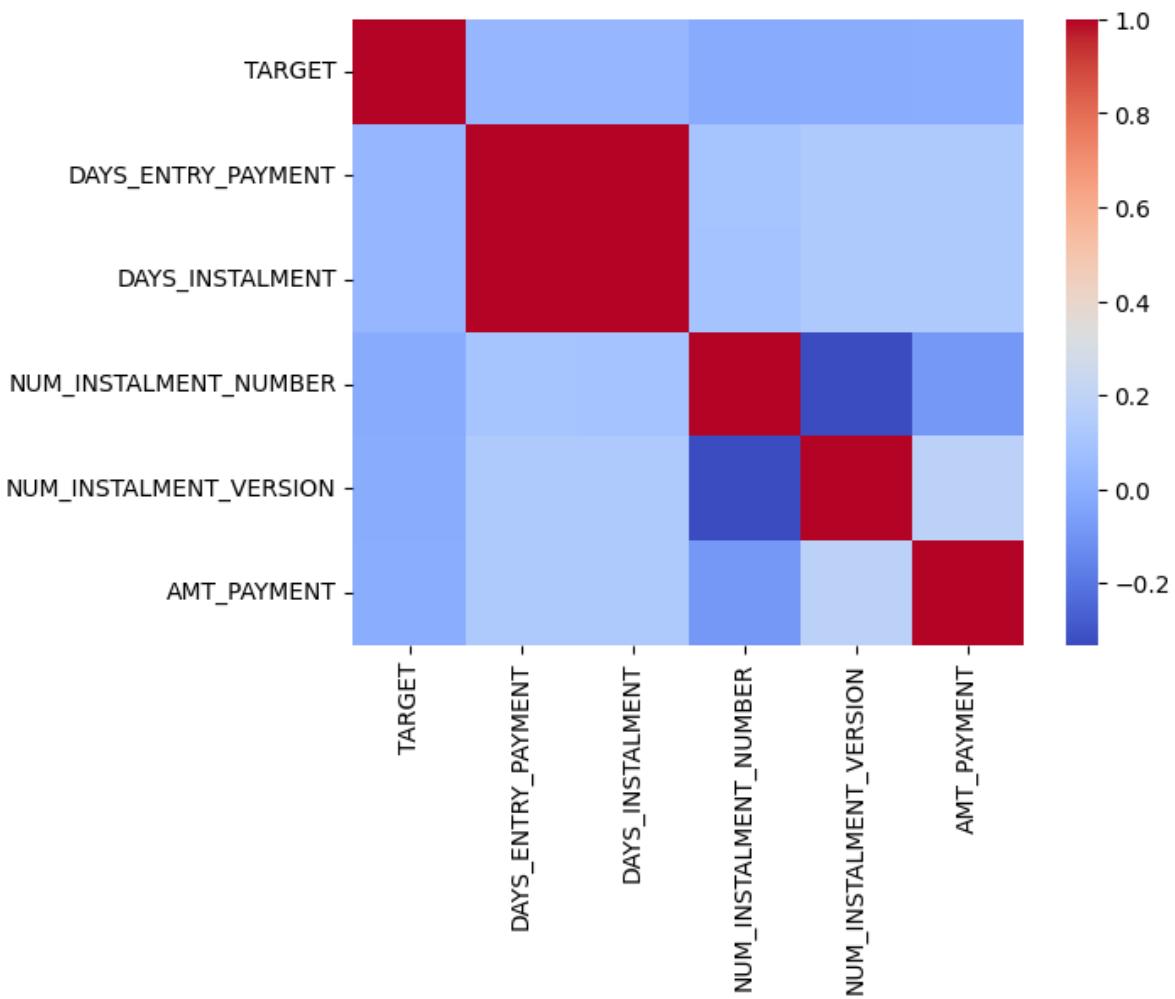
Pairs of variables with the highest positive correlation coefficients:

```
TARGET TARGET 1.000000
DAYS_ENTRY_PAYMENT DAYS_INSTALMENT 0.999472
NUM_INSTALMENT_VERSION AMT_PAYMENT 0.179931
                           DAYS_INSTALMENT 0.130761
DAYS_INSTALMENT AMT_PAYMENT 0.129411
dtype: float64
```

Pairs of variables with the highest negative correlation coefficients:

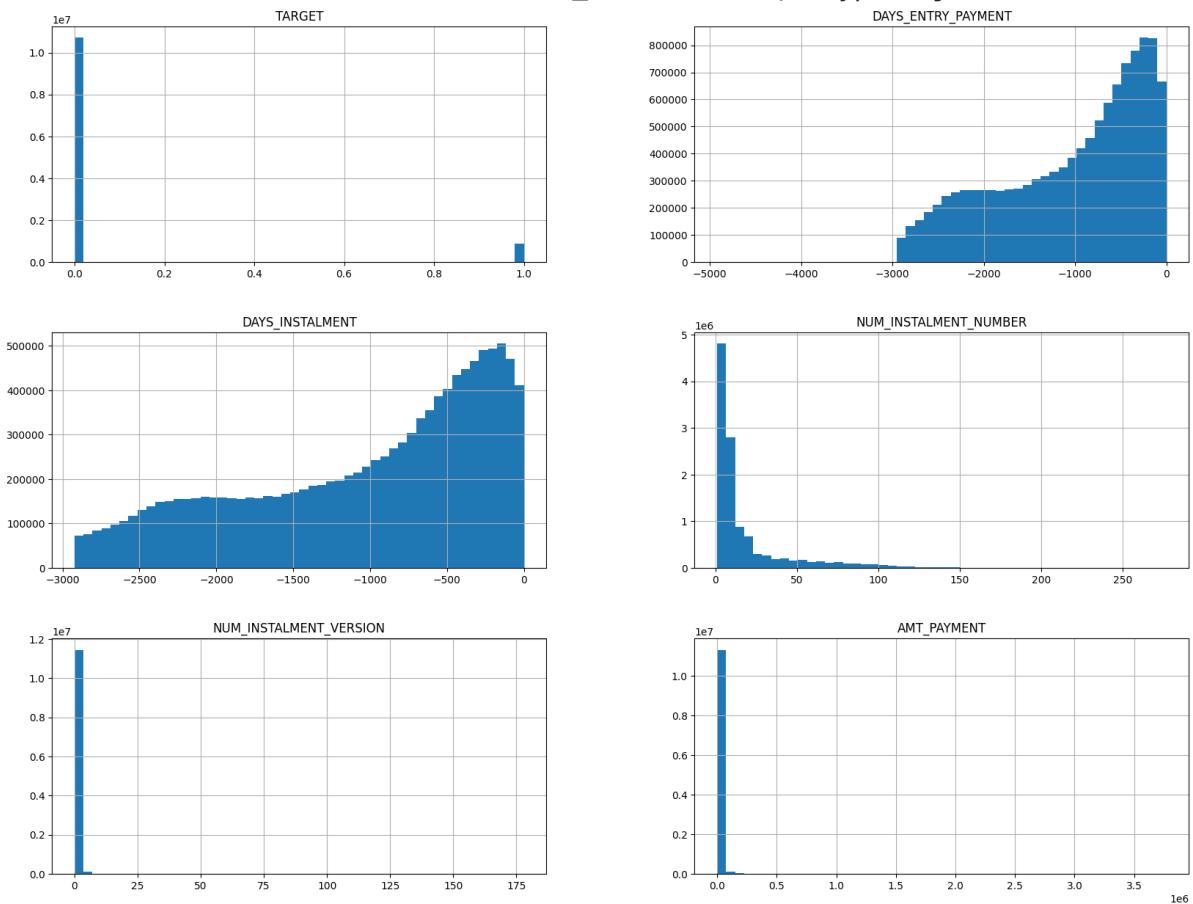
```
NUM_INSTALMENT_NUMBER NUM_INSTALMENT_VERSION -0.330851
AMT_PAYMENT NUM_INSTALMENT_NUMBER -0.086881
TARGET NUM_INSTALMENT_NUMBER -0.016190
                  NUM_INSTALMENT_VERSION -0.009896
                           AMT_PAYMENT -0.003623
dtype: float64
```

Out[ ]: <Axes: >



```
In [ ]: # Check the distribution of the top variables
installments_payments_top_features.hist(bins=50, figsize=(20,15))
```

```
Out[ ]: array([ [
```



---

## Bureau

```
In [ ]: datasets["bureau"].info()
```

```
<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
```

```
In [ ]: # Check the first few rows of the data
print(datasets["bureau"].head())
```

```

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

CREDIT_DAY_OVERDUE DAYS_CREDIT_ENDDATE DAYS_ENDDATE_FACT \
0 0 -153.0 -153.0
1 0 1075.0 NaN
2 0 528.0 NaN
3 0 NaN NaN
4 0 1197.0 NaN

AMT_CREDIT_MAX_OVERDUE CNT_CREDIT_PROLONG AMT_CREDIT_SUM \
0 NaN 0 91323.0
1 NaN 0 225000.0
2 NaN 0 464323.5
3 NaN 0 90000.0
4 77674.5 0 2700000.0

AMT_CREDIT_SUM_DEBT AMT_CREDIT_SUM_LIMIT AMT_CREDIT_SUM_OVERDUE \
0 0.0 NaN 0.0
1 171342.0 NaN 0.0
2 NaN NaN 0.0
3 NaN NaN 0.0
4 NaN NaN 0.0

CREDIT_TYPE DAYS_CREDIT_UPDATE AMT_ANNUITY
0 Consumer credit -131 NaN
1 Credit card -20 NaN
2 Consumer credit -16 NaN
3 Credit card -16 NaN
4 Consumer credit -21 NaN

```

---

## Dataset Size

```
In [ ]: # Check the shape of the data
print(datasets["bureau"].shape)
```

(1716428, 17)

```
In [ ]: # Check the data types of the columns
print(datasets["bureau"].dtypes)
```

```
SK_ID_CURR           int64
SK_ID_BUREAU         int64
CREDIT_ACTIVE        object
CREDIT_CURRENCY      object
DAYS_CREDIT          int64
CREDIT_DAY_OVERDUE  int64
DAYS_CREDIT_ENDDATE float64
DAYS_ENDDATE_FACT   float64
AMT_CREDIT_MAX_OVERDUE float64
CNT_CREDIT_PROLONG   int64
AMT_CREDIT_SUM        float64
AMT_CREDIT_SUM_DEBT  float64
AMT_CREDIT_SUM_LIMIT float64
AMT_CREDIT_SUM_OVERDUE float64
CREDIT_TYPE          object
DAYS_CREDIT_UPDATE   int64
AMT_ANNUITY          float64
dtype: object
```

---

## Summary statistics of the bureau

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

```
Out[ ]:
```

	SK_ID_CURR	SK_ID_BUREAU	DAYS_CREDIT	CREDIT_DAY_OVERDUE	DAYS_CREDIT
<b>count</b>	1.716428e+06	1.716428e+06	1.716428e+06	1.716428e+06	1.6
<b>mean</b>	2.782149e+05	5.924434e+06	-1.142108e+03	8.181666e-01	5.1
<b>std</b>	1.029386e+05	5.322657e+05	7.951649e+02	3.654443e+01	4.95
<b>min</b>	1.000010e+05	5.000000e+06	-2.922000e+03	0.000000e+00	-4.20
<b>25%</b>	1.888668e+05	5.463954e+06	-1.666000e+03	0.000000e+00	-1.13
<b>50%</b>	2.780550e+05	5.926304e+06	-9.870000e+02	0.000000e+00	-3.30
<b>75%</b>	3.674260e+05	6.385681e+06	-4.740000e+02	0.000000e+00	4.74
<b>max</b>	4.562550e+05	6.843457e+06	0.000000e+00	2.792000e+03	3.1

---

```
In [ ]: datasets["bureau"].describe(include='all') #numerical and categorical features
```

Out [ ]:

	SK_ID_CURR	SK_ID_BUREAU	CREDIT_ACTIVE	CREDIT_CURRENCY	DAY_S_CREDIT
<b>count</b>	1.716428e+06	1.716428e+06	1716428	1716428	1.716428e+06
<b>unique</b>	NaN	NaN	4	4	NaN
<b>top</b>	NaN	NaN	Closed	currency 1	NaN
<b>freq</b>	NaN	NaN	1079273	1715020	NaN
<b>mean</b>	2.782149e+05	5.924434e+06	NaN	NaN	-1.142108e+03
<b>std</b>	1.029386e+05	5.322657e+05	NaN	NaN	7.951649e+02
<b>min</b>	1.000010e+05	5.000000e+06	NaN	NaN	-2.922000e+03
<b>25%</b>	1.888668e+05	5.463954e+06	NaN	NaN	-1.666000e+03
<b>50%</b>	2.780550e+05	5.926304e+06	NaN	NaN	-9.870000e+02
<b>75%</b>	3.674260e+05	6.385681e+06	NaN	NaN	-4.740000e+02
<b>max</b>	4.562550e+05	6.843457e+06	NaN	NaN	0.000000e+00

## Summary of Missing data for Bureau

In [ ]:

```
# Check for missing data
print(datasets["bureau"].isnull().sum())
```

SK_ID_CURR	0
SK_ID_BUREAU	0
CREDIT_ACTIVE	0
CREDIT_CURRENCY	0
DAY_S_CREDIT	0
CREDIT_DAY_OVERDUE	0
DAY_S_CREDIT_ENDDATE	105553
DAY_S_ENDDATE_FACT	633653
AMT_CREDIT_MAX_OVERDUE	1124488
CNT_CREDIT_PROLONG	0
AMT_CREDIT_SUM	13
AMT_CREDIT_SUM_DEBT	257669
AMT_CREDIT_SUM_LIMIT	591780
AMT_CREDIT_SUM_OVERDUE	0
CREDIT_TYPE	0
DAY_S_CREDIT_UPDATE	0
AMT_ANNUITY	1226791
dtype: int64	

In [ ]:

```
# Calculate the percentage of missing data
missing_percentage = datasets["bureau"].isnull().sum() / len(datasets["bureau"])
print(missing_percentage)
```

```

SK_ID_CURR           0.000000
SK_ID_BUREAU         0.000000
CREDIT_ACTIVE        0.000000
CREDIT_CURRENCY      0.000000
DAYS_CREDIT          0.000000
CREDIT_DAY_OVERDUE  0.000000
DAYS_CREDIT_ENDDATE 6.149573
DAYS_ENDDATE_FACT   36.916958
AMT_CREDIT_MAX_OVERDUE 65.513264
CNT_CREDIT_PROLONG   0.000000
AMT_CREDIT_SUM        0.000757
AMT_CREDIT_SUM_DEBT   15.011932
AMT_CREDIT_SUM_LIMIT  34.477415
AMT_CREDIT_SUM_OVERDUE 0.000000
CREDIT_TYPE          0.000000
DAYS_CREDIT_UPDATE    0.000000
AMT_ANNUITY          71.473490
dtype: float64

```

```
In [ ]: # Combining the code to print the percent of missing value and missing value
percent = (datasets["bureau"].isnull().sum()/datasets["bureau"].isnull().count())
sum_missing = datasets["bureau"].isna().sum().sort_values(ascending = False)
missing_application_train_data = pd.concat([percent, sum_missing], axis=1,
missing_application_train_data.head(20)
```

Out[ ]:

	Percent	Missing Value Count
AMT_ANNUITY	71.47	1226791
AMT_CREDIT_MAX_OVERDUE	65.51	1124488
DAYS_ENDDATE_FACT	36.92	633653
AMT_CREDIT_SUM_LIMIT	34.48	591780
AMT_CREDIT_SUM_DEBT	15.01	257669
DAYS_CREDIT_ENDDATE	6.15	105553
AMT_CREDIT_SUM	0.00	13
CREDIT_ACTIVE	0.00	0
CREDIT_CURRENCY	0.00	0
DAYS_CREDIT	0.00	0
CREDIT_DAY_OVERDUE	0.00	0
SK_ID_BUREAU	0.00	0
CNT_CREDIT_PROLONG	0.00	0
AMT_CREDIT_SUM_OVERDUE	0.00	0
CREDIT_TYPE	0.00	0
DAYS_CREDIT_UPDATE	0.00	0
SK_ID_CURR	0.00	0

```
In [ ]: import pandas as pd
df = pd.DataFrame(datasets["bureau"])

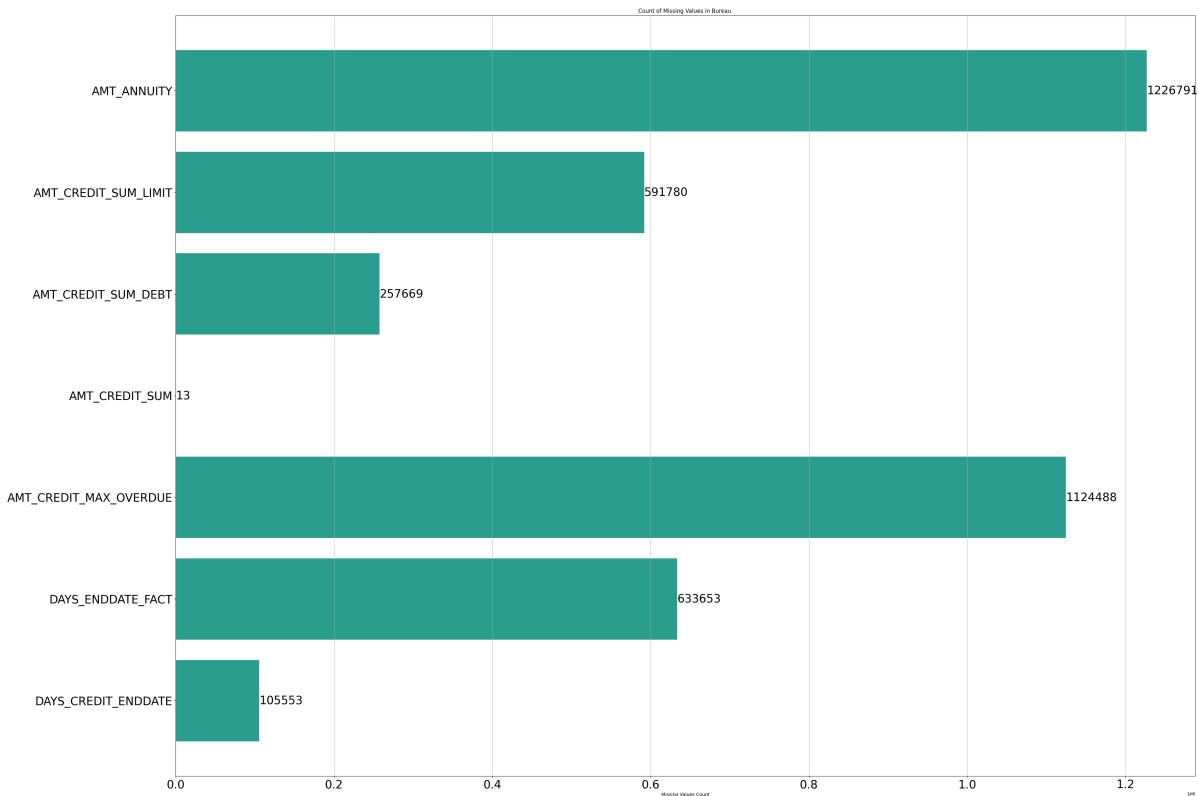
# Select only columns with missing values
cols_with_missing = df.columns[df.isnull().sum() > 0]

# Count the number of missing values in each column
missing_values_count = df[cols_with_missing].isnull().sum()

# Create a horizontal bar chart of the count of missing values for each column
plt.figure(figsize=(40, 30))
plt.barh(missing_values_count.index, missing_values_count.values, color="#2ca02c")
plt.xlabel('Missing Values Count')
plt.title('Count of Missing Values in Bureau ')
plt.grid(axis='x')

# Add data labels to the bar chart
for i, v in enumerate(missing_values_count.values):
    plt.text(v + 50, i, str(v), color='black', fontsize=25, ha='left', va='center')

plt.xticks(fontsize=25)
plt.yticks(fontsize=25)
plt.show()
```



## Insights

- The bureau file contains the following columns that have missing data  
AMT\_ANNUITY, AMT\_CREDIT\_SUM\_LIMIT,

AMT\_CREDIT\_SUM\_DEBT, AMT\_CREDIT\_SU, AMT\_CREDIT\_MAX\_OVERDUE, DAYS\_ENDDATE and DAYS\_CREDIT\_ENDDATE

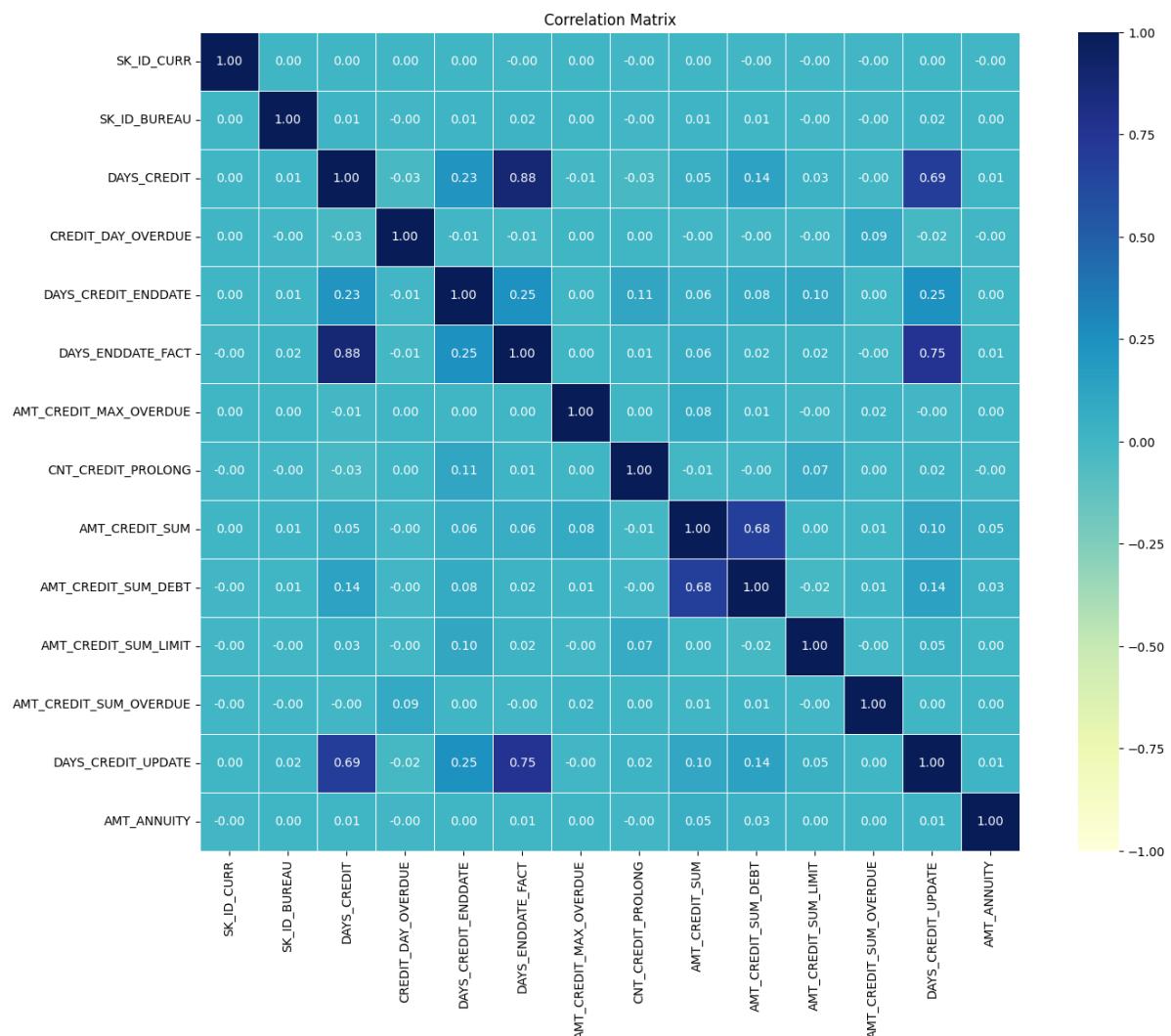
```
In [ ]: # Check the correlation between the variables
corr_matrix = datasets["bureau"].corr()
sns.heatmap(corr_matrix, annot=True, cmap='YlGnBu')

# Set the figure size
plt.figure(figsize=(17,12))

# Plot the heatmap with annotations
sns.heatmap(corr_matrix, cmap='YlGnBu', annot=True, fmt='.2f', vmin=-1, vmax=1)

# Add a title
plt.title('Correlation Matrix')

# Show the plot
plt.show()
```

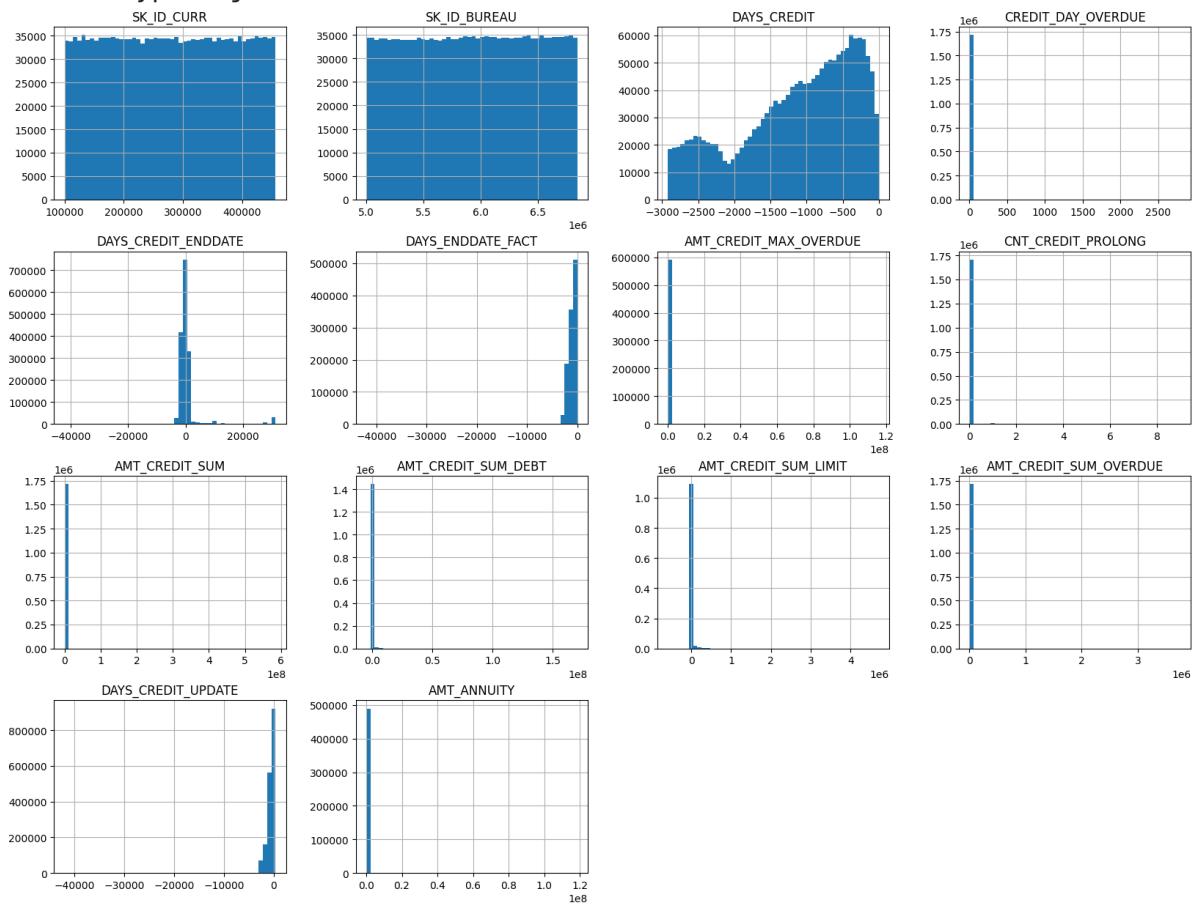


**Insights**

- Based on the correlation matrix we can say that the variables are less correlated with each other. The greatest positive correlation is 0.88 between DAYS\_ENDDATE\_FACT and DAYS\_CREDIT whereas the greatest negative correlation ranges between -0.01 and -0.03

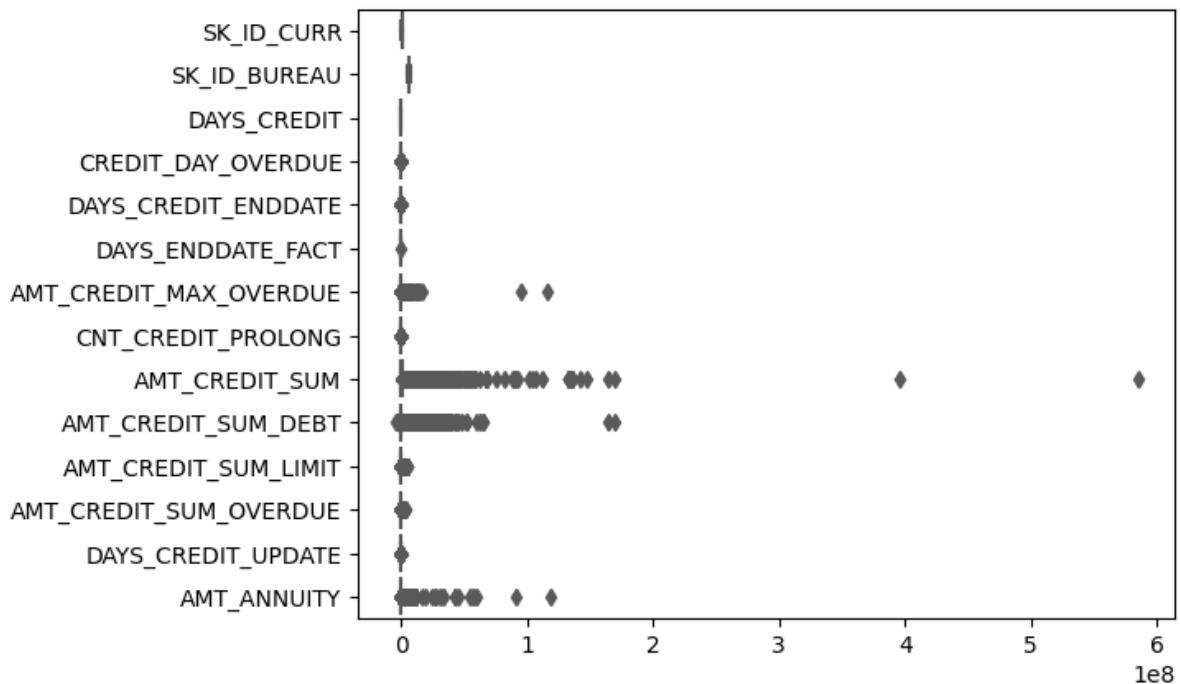
```
In [ ]: # Check the distribution of the variables
datasets["bureau"].hist(bins=50, figsize=(20,15))
```

```
Out[ ]: array([[[<Axes: title={'center': 'SK_ID_CURR'}>,
    <Axes: title={'center': 'SK_ID_BUREAU'}>,
    <Axes: title={'center': 'DAYS_CREDIT'}>,
    <Axes: title={'center': 'CREDIT_DAY_OVERDUE'}>],
   [<Axes: title={'center': 'DAYS_CREDIT_ENDDATE'}>,
    <Axes: title={'center': 'DAYS_ENDDATE_FACT'}>,
    <Axes: title={'center': 'AMT_CREDIT_MAX_OVERDUE'}>,
    <Axes: title={'center': 'CNT_CREDIT_PROLONG'}>],
   [<Axes: title={'center': 'AMT_CREDIT_SUM'}>,
    <Axes: title={'center': 'AMT_CREDIT_SUM_DEBT'}>,
    <Axes: title={'center': 'AMT_CREDIT_SUM_LIMIT'}>,
    <Axes: title={'center': 'AMT_CREDIT_SUM_OVERDUE'}>],
   [<Axes: title={'center': 'DAYS_CREDIT_UPDATE'}>,
    <Axes: title={'center': 'AMT_ANNUITY'}>, <Axes: >, <Axes: >]],
  dtype=object)
```



```
In [ ]: # Check for outliers
sns.boxplot(data=datasets["bureau"], orient='h', palette='Set2')
```

```
Out[ ]: <Axes: >
```



- The box plot shows that bureau files do not have any variables that have outlier

---

## Visual EDA of Bureau

```
In [ ]: bureau=datasets["bureau"].copy()

merge_target = pd.merge(bureau,extract_key_target,on='SK_ID_CURR')

correlation_bureau = merge_target.corr() ['TARGET'].abs().sort_values(ascending=False)
```

```
In [ ]: correlation_bureau.head(6)
```

```
Out[ ]: TARGET           1.000000
        DAYS_CREDIT      0.061556
        DAYS_CREDIT_UPDATE 0.041076
        DAYS_ENDDATE_FACT 0.039057
        DAYS_CREDIT_ENDDATE 0.026497
        AMT_CREDIT_SUM     0.010606
Name: TARGET, dtype: float64
```

```
In [ ]: top_feature = correlation_bureau[0:6].index.tolist()

bureau_top_features = merge_target[top_feature]

corr_matrix = bureau_top_features.corr()

plt.figure(figsize=(17,12))
```

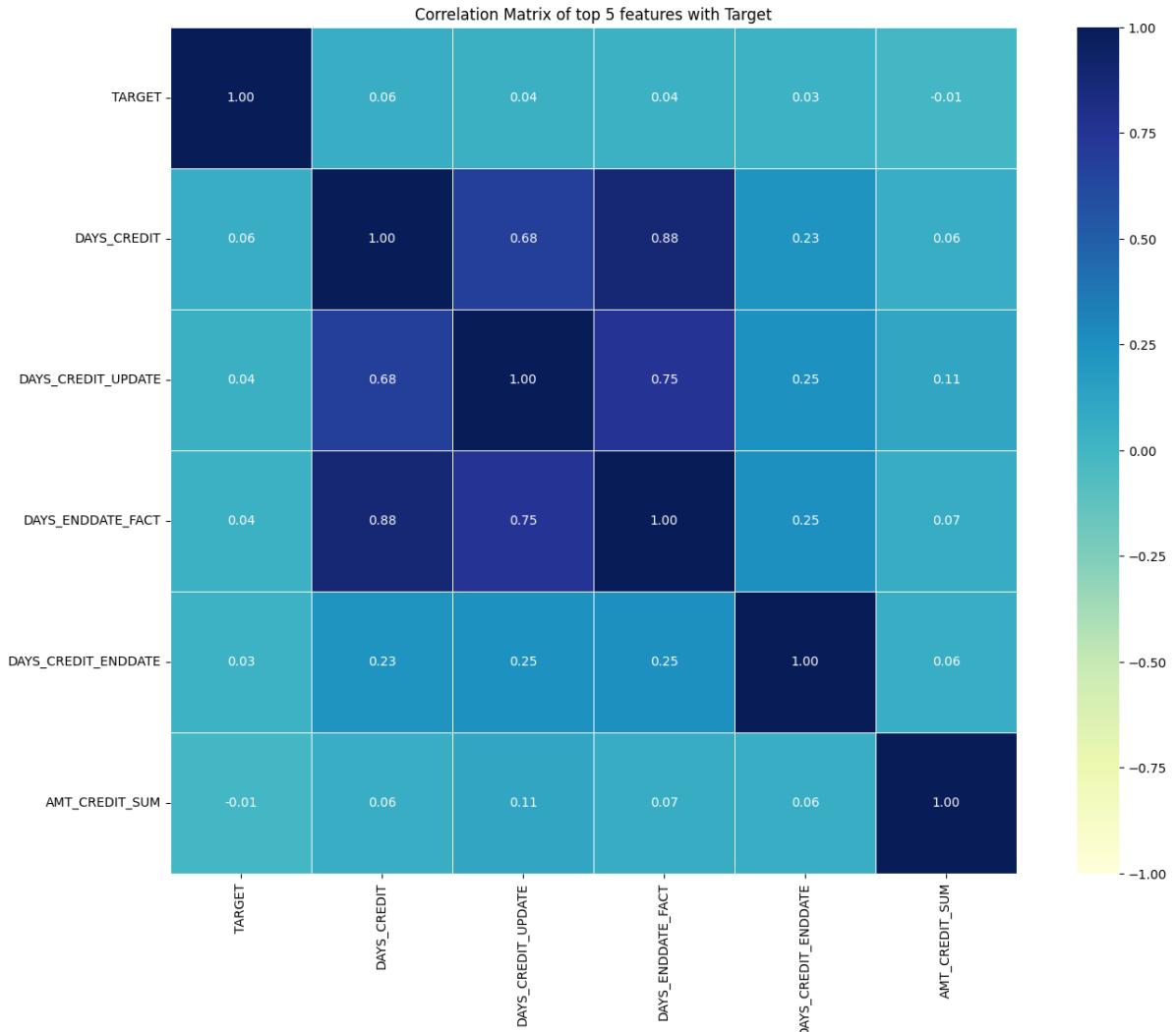
```

sns.heatmap(corr_matrix, cmap='YlGnBu', annot=True, fmt='.2f', vmin=-1, vmax=1)

plt.title('Correlation Matrix of top 5 features with Target')

plt.show()

```



## Pair based visualization

```

In [ ]: import seaborn as sns

# Identify the pairs of variables that have the highest positive and negative correlation coefficients
highest_corr_pairs = corr_matrix.unstack().sort_values(ascending=False).drop_duplicates()
lowest_corr_pairs = corr_matrix.unstack().sort_values(ascending=True).drop_duplicates()

print("Pairs of variables with the highest positive correlation coefficients")
print(highest_corr_pairs)
print("\nPairs of variables with the highest negative correlation coefficients")
print(lowest_corr_pairs)

# Visualize the correlation matrix using a heatmap
sns.heatmap(corr_matrix, cmap='coolwarm')

```

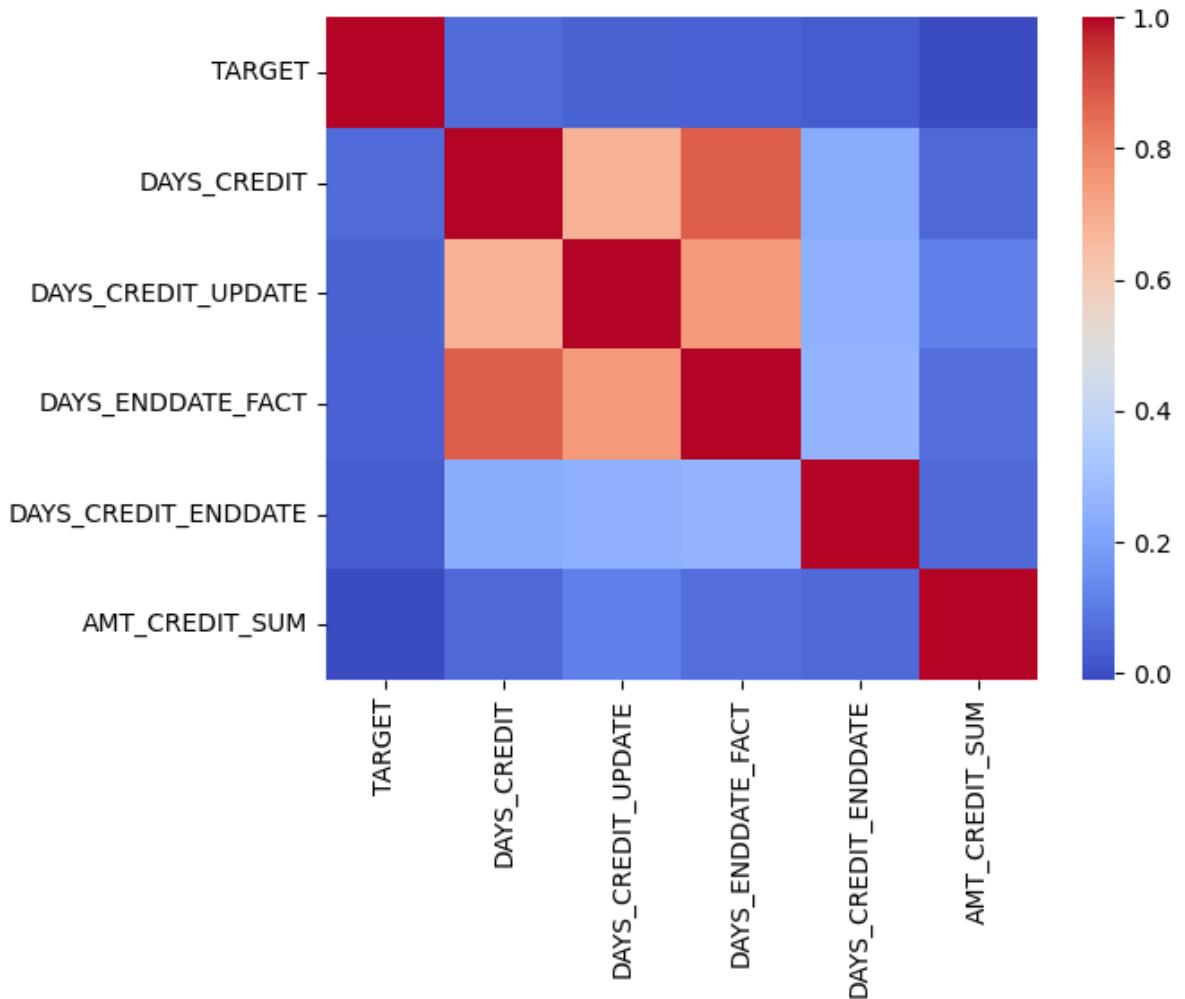
Pairs of variables with the highest positive correlation coefficients:

```
TARGET           TARGET      1.000000
DAYS_CREDIT     DAYS_ENDDATE_FACT  0.875291
DAYS_ENDDATE_FACT DAYS_CREDIT_UPDATE  0.745192
DAYS_CREDIT     DAYS_CREDIT_UPDATE  0.683189
DAYS_ENDDATE_FACT DAYS_CREDIT_ENDDATE  0.254893
dtype: float64
```

Pairs of variables with the highest negative correlation coefficients:

```
TARGET           AMT_CREDIT_SUM      -0.010606
                  DAYS_CREDIT_ENDDATE   0.026497
                  DAYS_ENDDATE_FACT    0.039057
                  DAYS_CREDIT_UPDATE    0.041076
AMT_CREDIT_SUM   DAYS_CREDIT        0.055663
dtype: float64
```

Out[ ]: <Axes: >



## Bureau Balance

In [ ]: `datasets["bureau_balance"].info()`

```
<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
```

```
In [ ]: # Check the first few rows of the data
print(datasets["bureau_balance"].head())
```

```
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
```

```
In [ ]: # Check the shape of the data
print(datasets["bureau_balance"].shape)
```

```
(27299925, 3)
```

```
In [ ]: # Check the data types of the columns
print(datasets["bureau_balance"].dtypes)
```

```
SK_ID_BUREAU      int64
MONTHS_BALANCE    int64
STATUS            object
dtype: object
```

---

## Summary statistics of the data

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

```
Out[ ]:      SK_ID_BUREAU  MONTHS_BALANCE
count    2.729992e+07    2.729992e+07
mean     6.036297e+06   -3.074169e+01
std      4.923489e+05    2.386451e+01
min      5.001709e+06   -9.600000e+01
25%     5.730933e+06   -4.600000e+01
50%     6.070821e+06   -2.500000e+01
75%     6.431951e+06   -1.100000e+01
max     6.842888e+06    0.000000e+00
```

```
In [ ]: datasets["bureau_balance"].describe(include='all') #numerical and categorical
```

Out [ ]:

	SK_ID_BUREAU	MONTHS_BALANCE	STATUS
<b>count</b>	2.729992e+07	2.729992e+07	27299925
<b>unique</b>	NaN	NaN	8
<b>top</b>	NaN	NaN	C
<b>freq</b>	NaN	NaN	13646993
<b>mean</b>	6.036297e+06	-3.074169e+01	NaN
<b>std</b>	4.923489e+05	2.386451e+01	NaN
<b>min</b>	5.001709e+06	-9.600000e+01	NaN
<b>25%</b>	5.730933e+06	-4.600000e+01	NaN
<b>50%</b>	6.070821e+06	-2.500000e+01	NaN
<b>75%</b>	6.431951e+06	-1.100000e+01	NaN
<b>max</b>	6.842888e+06	0.000000e+00	NaN

---

## Summary of Missing data for Bureau Balance

In [ ]:

```
# Check for missing data
print(datasets["bureau_balance"].isnull().sum())
```

SK_ID_BUREAU	0
MONTHS_BALANCE	0
STATUS	0
dtype:	int64

In [ ]:

```
# Calculate the percentage of missing data
missing_percentage = datasets["bureau_balance"].isnull().sum() / len(dataset)
print(missing_percentage)
```

SK_ID_BUREAU	0.0
MONTHS_BALANCE	0.0
STATUS	0.0
dtype:	float64

In [ ]:

```
# Combining the code to print the percent of missing value and missing value
percent = (datasets["bureau_balance"].isnull().sum()/datasets["bureau_balance"].count())
sum_missing = datasets["bureau_balance"].isna().sum().sort_values(ascending=True)
missing_application_train_data = pd.concat([percent, sum_missing], axis=1,
missing_application_train_data.head(20))
```

Out [ ]:

	Percent	Missing Value Count
SK_ID_BUREAU	0.0	0
MONTHS_BALANCE	0.0	0
STATUS	0.0	0

```
In [ ]: import pandas as pd
df = pd.DataFrame(datasets["bureau_balance"])

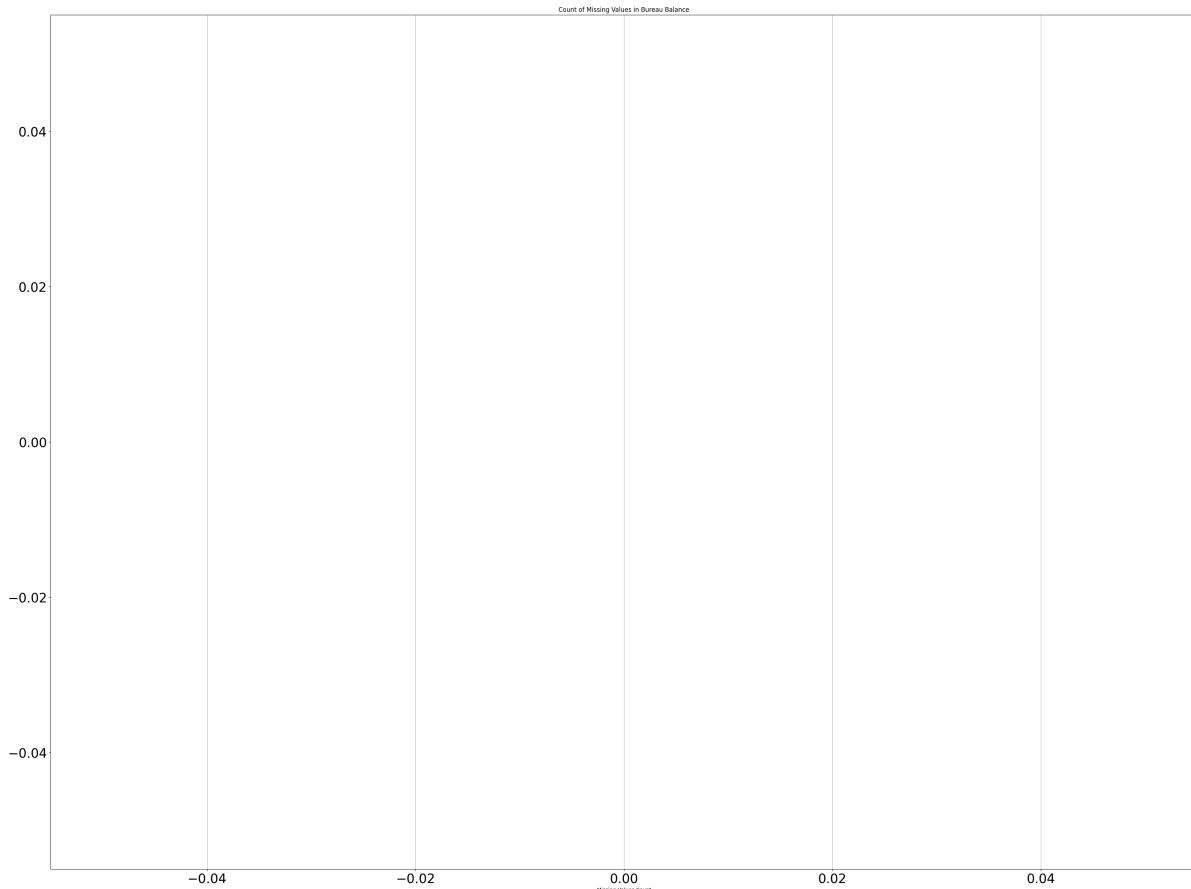
# Select only columns with missing values
cols_with_missing = df.columns[df.isnull().sum() > 0]

# Count the number of missing values in each column
missing_values_count = df[cols_with_missing].isnull().sum()

# Create a horizontal bar chart of the count of missing values for each column
plt.figure(figsize=(40, 30))
plt.barh(missing_values_count.index, missing_values_count.values, color="#2ca02c")
plt.xlabel('Missing Values Count')
plt.title('Count of Missing Values in Bureau Balance')
plt.grid(axis='x')

# Add data labels to the bar chart
for i, v in enumerate(missing_values_count.values):
    plt.text(v + 50, i, str(v), color='black', fontsize=25, ha='left', va='center')

plt.xticks(fontsize=25)
plt.yticks(fontsize=25)
plt.show()
```



- The bureau balance file does not contain any missing value in any of the variables

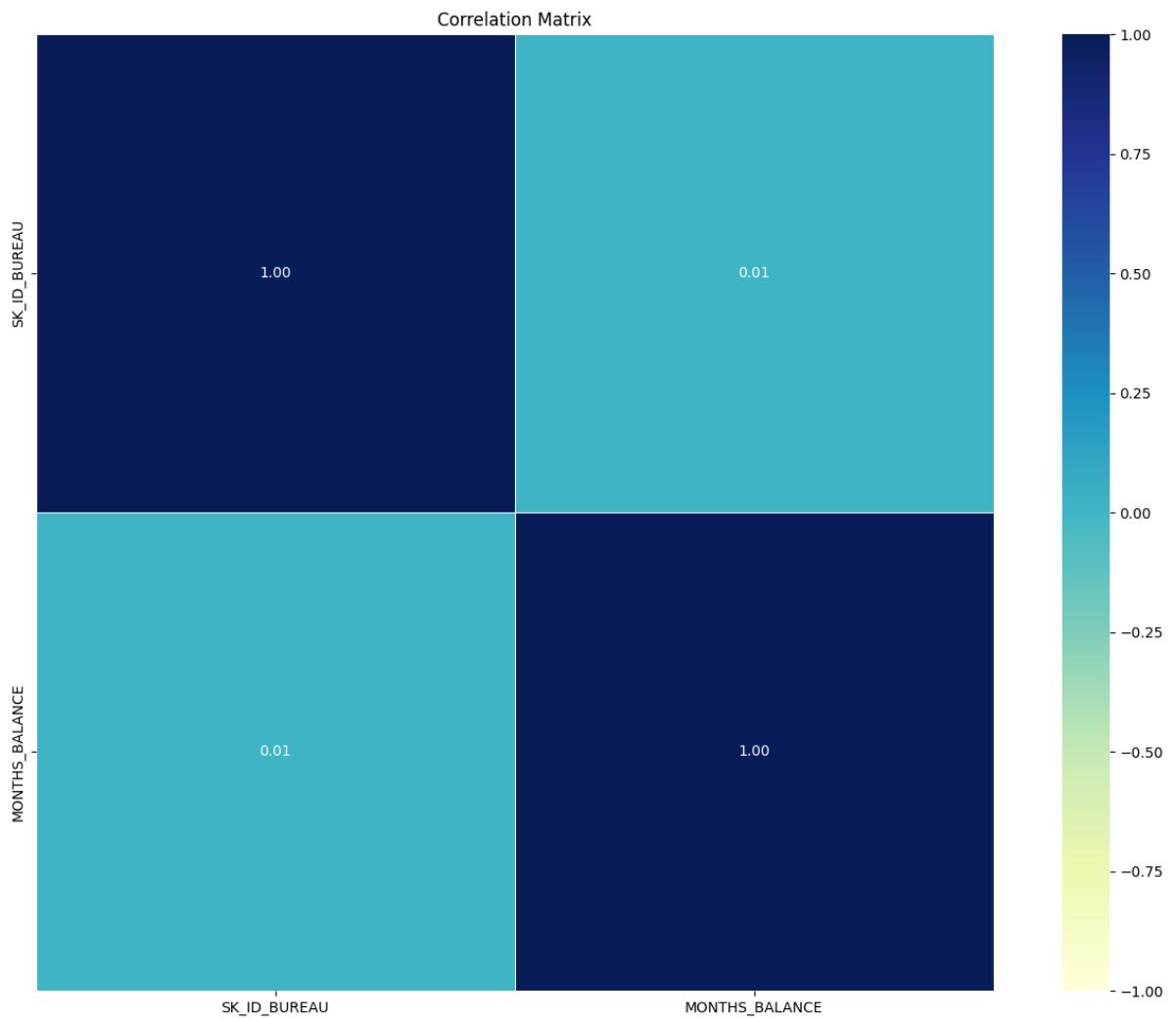
```
In [ ]: # Check the correlation between the variables
corr_matrix = datasets["bureau_balance"].corr()
sns.heatmap(corr_matrix, annot=True, cmap='YlGnBu')

# Set the figure size
plt.figure(figsize=(17,12))

# Plot the heatmap with annotations
sns.heatmap(corr_matrix, cmap='YlGnBu', annot=True, fmt='.2f', vmin=-1, vmax=1)

# Add a title
plt.title('Correlation Matrix')

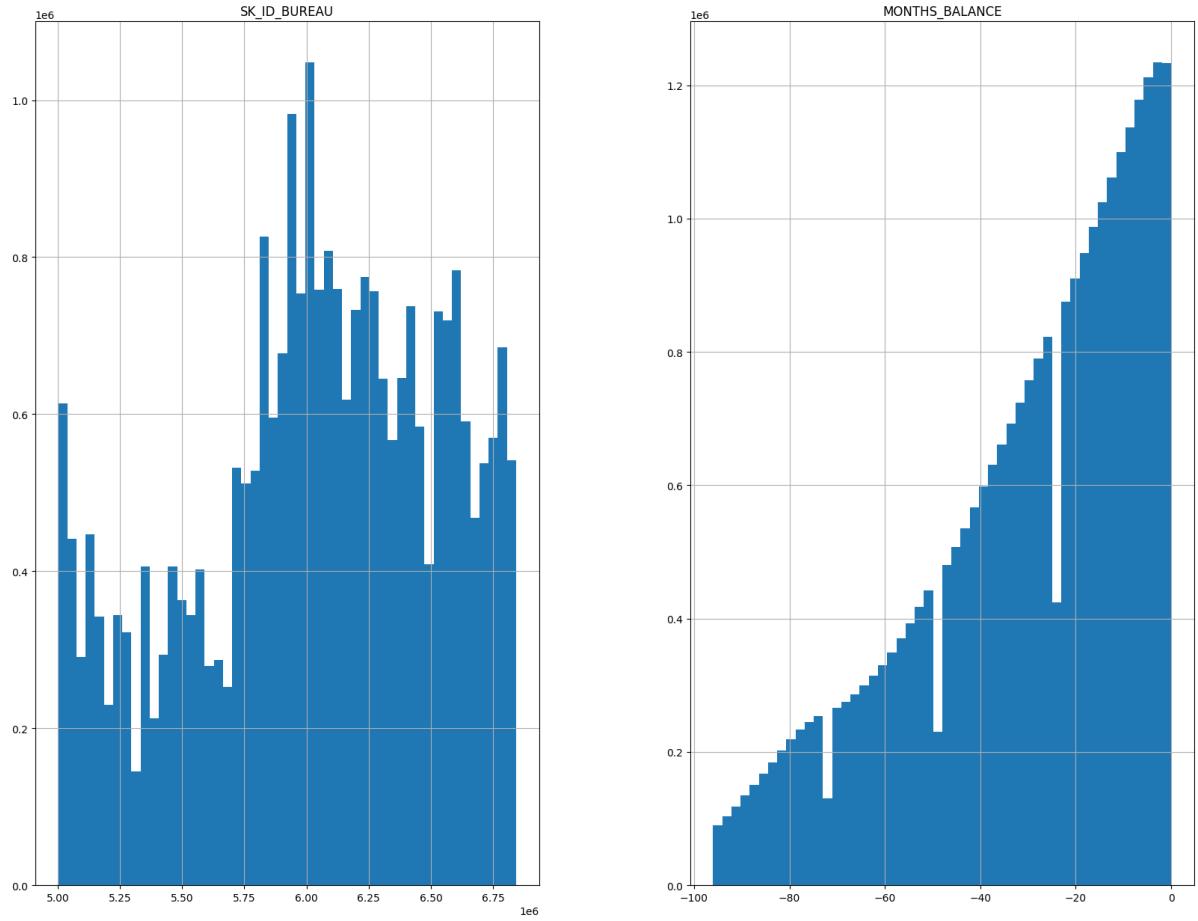
# Show the plot
plt.show()
```



- The above graph shows that there is very less correlation between the variables of the bureau balance file

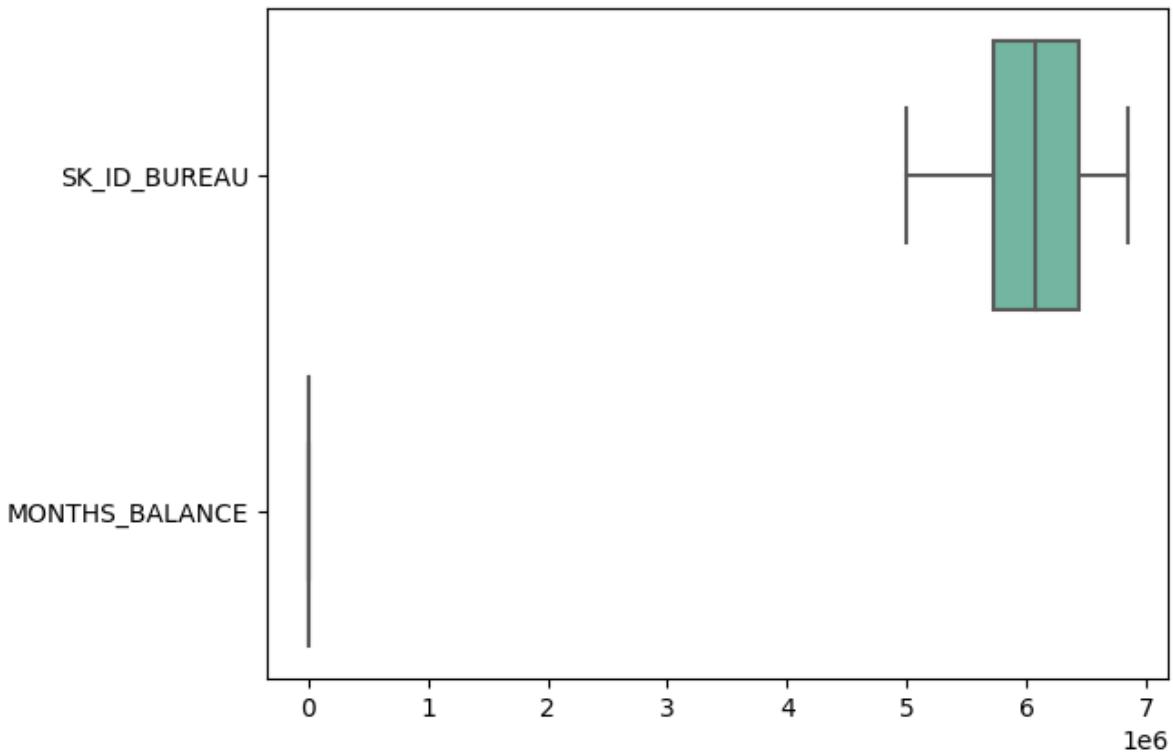
```
In [ ]: # Check the distribution of the variables
datasets["bureau_balance"].hist(bins=50, figsize=(20,15))
```

```
Out[ ]: array([ [
```



```
In [ ]: # Check for outliers
sns.boxplot(data=datasets["bureau_balance"], orient='h', palette='Set2')
```

```
Out[ ]: <Axes: >
```



- There exists outliers in the SK\_ID\_BUREAU column of the bureau balance data
- 
- 

## previous applications for the submission file

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

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

In [ ]: # print(f'There are {appsDF.shape[0]}:{} previous applications')

In [ ]: # #Find the intersection of two arrays.
# print(f'Number of train applicants with previous applications is {len(np.intersect1d(appsDF['SK_ID_CURR'], train['SK_ID_CURR']))}')

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

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

```
In [ ]:
```

```
In [ ]: # prevAppCounts[prevAppCounts >50].plot(kind='bar')
# plt.xticks(rotation=25)
# plt.show()
```

## Histogram of Number of previous applications for an ID

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

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

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

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

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

## Joining secondary tables with the primary table

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

### Joining previous\_application with application\_x

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

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

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

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

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

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

1. Preprocess each of the non-application data sets, thereby generating many new (derived) features, and then joining (aka merge) the results with the `application_train` data (the labeled dataset) and with the `application_test` data (the unlabeled submission dataset) prior to processing the data (in a train, valid, test partition) via your machine learning pipeline. [This approach is recommended for this HCDR competition. WHY?]
  - Do the joins as part of the transformation steps. [Not recommended here. WHY?]. How can this be done? Will it work?
    - This would be necessary if we had dataset wide features such as IDF (inverse document frequency) which depend on the entire subset of data as opposed to a single loan application (e.g., a feature about the relative amount applied for such as the percentile of the loan amount being applied for).

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

## Roadmap for secondary table processing

1. Transform all the secondary tables to features that can be joined into the main table the application table (labeled and unlabeled)
  - 'bureau', 'bureau\_balance', 'credit\_card\_balance', 'installments\_payments',
  - 'previous\_application', 'POS\_CASH\_balance'
- Merge the transformed secondary tables with the primary tables (i.e., the `application_train` data (the labeled dataset) and with the `application_test` data (the unlabeled submission dataset)), thereby leading to `X_train`, `y_train`, `X_valid`, etc.
- Proceed with the learning pipeline using `X_train`, `y_train`, `X_valid`, etc.

- Generate a submission file using the learnt model

```
In [ ]: # df=pd.DataFrame(datasets["bureau"])

In [ ]: # datasets["application_train_small"] = df.head(50000).copy()

# ## Joining the data for bureau and bureau balance
# datasets["bureau"] = datasets["bureau"].merge(datasets["bureau_balance"], r
# datasets["bureau"] = datasets["bureau"].groupby("SK_ID_BUREAU").mean()

# ## joining application_train_tiny with datasets_bureau
# datasets["application_train_tiny"] = datasets["application_train_tiny"].me
# # datasets["application_train_tiny"] = datasets["application_train_tiny"].
# datasets["application_train_tiny"].shape

In [ ]: # # Rename columns in each dataset
# app_train.columns = ['app_' + c.lower() for c in app_train.columns]
# app_test.columns = ['app_' + c.lower() for c in app_test.columns]
# prev_app.columns = ['prev_' + c.lower() for c in prev_app.columns]

In [ ]: # app_data.head(10)

In [ ]: # # Merge the datasets
# # app_data = app_train.append(app_test, ignore_index=True)
# merged_data = pd.merge(app_data, prev_app, left_on='app_sk_id_curr', right_
```

## feature transformer for prevApp table

### Join the labeled dataset

```
In [ ]: # ~3==3

In [ ]: # datasets.keys()

In [ ]: # features = ['AMT_ANNUITY', 'AMT_APPLICATION']
# prevApps_feature_pipeline = Pipeline([
#     ('prevApps_add_features1', prevApps_add_features1()), # add some
#     ('prevApps_add_features2', prevApps_add_features2()), # add some
#     ('prevApps_aggregator', prevAppsFeaturesAggregator()), # Aggregate
# ])

# X_train= datasets["application_train"] #primary dataset
# appsDF = datasets["previous_application"] #prev app

# merge_all_data = False

# # transform all the secondary tables
# # 'bureau', 'bureau_balance', 'credit_card_balance', 'installments_payment
```

```

# # 'previous_application', 'POS_CASH_balance'

# if merge_all_data:
#     prevApps_aggregated = prevApps_feature_pipeline.transform(appsDF)

#     #'bureau', 'bureau_balance', 'credit_card_balance', 'installments_paym
#     #'previous_application', 'POS_CASH_balance'

# # merge primary table and secondary tables using features based on meta da
# if merge_all_data:
#     # 1. Join/Merge in prevApps Data
#     X_train = X_train.merge(prevApps_aggregated, how='left', on='SK_ID_CURR')

#     # 2. Join/Merge in ..... Data
#     #X_train = X_train.merge(....._aggregated, how='left', on="SK_ID_CURR")

#     # 3. Join/Merge in .....Data
#     #dX_train = X_train.merge(...._aggregated, how='left', on="SK_ID_CURR")

#     # 4. Join/Merge in Aggregated ..... Data
#     #X_train = X_train.merge(...._aggregated, how='left', on="SK_ID_CURR")

#     # .....

```

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

```

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

    # 2. Join/Merge in ..... Data
    #X_train = X_train.merge(....._aggregated, how='left', on="SK_ID_CURR")

    # 3. Join/Merge in .....Data
    #df_labeled = df_labeled.merge(...._aggregated, how='left', on="SK_ID_CURR")

    # 4. Join/Merge in Aggregated ..... Data
    #df_labeled = df_labeled.merge(...._aggregated, how='left', on="SK_ID_CURR")

    # .....

```

```
In [ ]: # approval rate 'NFLAG_INSURED_ON_APPROVAL'
```

```

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

```

```
#         X['CharlsonIndex'] = X['CharlsonIndex'].apply(lambda x: charlson_i
#         X['LengthOfStay'] = X['LengthOfStay'].apply(lambda x: None if pd.i
#         return X
```

## Processing pipeline

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

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

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

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

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

Here is an example that is in action:

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

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

```
    handle_unknown="ignore"))
])
```

```
In [ ]: # # load data
# df = pd.read_csv('chronic_kidney_disease.csv', header="infer")
# # names=['age', 'bp', 'sg', 'al', 'su', 'rbc', 'pc', 'pcc', 'ba', 'bgr', '#
# # 'hemo', 'pcv', 'wc', 'rc', 'htn', 'dm', 'cad', 'appet', 'pe', 'ane', 'cl#
# # head of df
# df.head(10)
```

```
In [ ]: # # Categorical boolean mask
# categorical_feature_mask = df.dtypes==object
# categorical_feature_mask
```

```
In [ ]: # # filter categorical columns using mask and turn it into a list
# categorical_cols = X.columns[categorical_feature_mask].tolist()
# categorical_cols
```

```
In [ ]: # from sklearn.preprocessing import OneHotEncoder
# import pandas as pd
# categorical_feature_mask = [True, False]
# # instantiate OneHotEncoder
# enc = OneHotEncoder(categorical_features = categorical_feature_mask,sparse=False)
# # categorical_features = boolean mask for categorical columns
# # sparse = False output an array not sparse matrix
# X_train = pd.DataFrame([[['small', 1], ['small', 3], ['medium', 3], ['large', 1], ['EXTRA-large', 2]]])
# X_test = [[['small', 1.2], ['medium', 4], ['EXTRA-large', 2]]]
# print(f"X_train:{\n{X_train}}")
# print(f"enc.fit_transform(X_train):\n{enc.fit_transform(X_train)}")
# print(f"enc.transform(X_test):\n{enc.transform(X_test)}")

# print(f"enc.get_feature_names():\n{enc.get_feature_names()}")
```

```
In [ ]: # print(f"enc.categories_{enc.categories_}")

# print(f"enc.categories_{enc.categories_}")
# enc.transform([[['Female', 1], ['Male', 4]]]).toarray()

# enc.inverse_transform([[0, 1, 1, 0, 0], [0, 0, 0, 1, 0]])

# enc.get_feature_names()
```

## OHE case study: The breast cancer wisconsin dataset (classification)

```
In [ ]: # from sklearn.datasets import load_breast_cancer
# data = load_breast_cancer(return_X_y=False)
# X, y = load_breast_cancer(return_X_y=True)
# print(y[[10, 50, 85]])
# #([0, 1, 0])
# list(data.target_names)
```

```
# #['malignant', 'benign']
# X.shape
```

```
In [ ]: # data.feature_names
```

Please [this blog](#) for more details of OHE when the validation/test have previously unseen unique values.

## HCDR preprocessing

```
In [ ]: # # Split the provided training data into training and validation and test
# # The kaggle evaluation test set has no labels
# #
# from sklearn.model_selection import train_test_split

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

# selected_features = ['AMT_INCOME_TOTAL', 'AMT_CREDIT', 'DAYS_EMPLOYED', 'DA
#                       'EXT_SOURCE_2', 'EXT_SOURCE_3', 'CODE_GENDER', 'FLAG_OWN_REALTY', 'FL
#                       'NAME_EDUCATION_TYPE', 'OCCUPATION_TYPE', 'NAME_INCOME_TY
# y_train = X_train['TARGET']
# X_train = X_train[selected_features]
# X_train, X_valid, y_train, y_valid = train_test_split(X_train, y_train, te
# X_train, X_test, y_train, y_test = train_test_split(X_train, y_train, test
# X_kaggle_test= X_kaggle_test[selected_features]
# # y_test = datasets["application_test"]['TARGET']    #why no TARGET??! (hi

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

```
In [ ]: # from sklearn.base import BaseEstimator, TransformerMixin
# import re

# # Creates the following date features
# # But could do so much more with these features
# # E.g.,
# #       extract the domain address of the homepage and OneHotEncode it
# #
# # ['release_month', 'release_day', 'release_year', 'release_dayofweek', 'rele
```

```

# class prep_OCCUPATION_TYPE(BaseEstimator, TransformerMixin):
#     def __init__(self, features="OCCUPATION_TYPE"): # no *args or **kargs
#         self.features = features
#     def fit(self, X, y=None):
#         return self # nothing else to do
#     def transform(self, X):
#         df = pd.DataFrame(X, columns=self.features)
# #from IPython.core.debugger import Pdb;    pdb().set_trace()
#         df['OCCUPATION_TYPE'] = df['OCCUPATION_TYPE'].apply(lambda x: 1. if x == 'Sales' else 0.)
# #df.drop(self.features, axis=1, inplace=True)
#         return np.array(df.values) #return a Numpy Array to observe the pipeline
# 
```

```

# from sklearn.pipeline import make_pipeline
# features = ["OCCUPATION_TYPE"]
# def test_driver_prep_OCCUPATION_TYPE():
#     print(f"X_train.shape: {X_train.shape}\n")
#     print(f"X_train['name'][0:5]: \n{X_train[features][0:5]}")
#     test_pipeline = make_pipeline(prep_OCCUPATION_TYPE(features))
#     return(test_pipeline.fit_transform(X_train))

# x = test_driver_prep_OCCUPATION_TYPE()
# print(f"Test driver: \n{test_driver_prep_OCCUPATION_TYPE()[0:10, :]}")
# print(f"X_train['name'][0:10]: \n{X_train[features][0:10]}")

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

```

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

```

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

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

# # Notice handle_unknown="ignore" in OHE which ignore values from the validation set
# # do NOT occur in the training set
# cat_pipeline = Pipeline([
# 
```

```
#         ('selector', DataFrameSelector(cat_atr)),  
#         ('imputer', SimpleImputer(strategy='most_frequent')),  
#         ('imputer', SimpleImputer(strategy='constant', fill_value='missing'))  
#     ])  
  
# data_prep_pipeline = FeatureUnion(transformer_list=[  
#     ("num_pipeline", num_pipeline),  
#     ("cat_pipeline", cat_pipeline),  
# ])
```

```
In [ ]: # list(datasets["application_train"].columns)
```

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

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

```
In [ ]: # try:  
#       expLog  
# except NameError:  
#     expLog = pd.DataFrame(columns=["exp_name",  
#                                     "Train Acc",  
#                                     "Valid Acc",  
#                                     "Test Acc",  
#                                     "Train AUC",  
#                                     "Valid AUC",  
#                                     "Test AUC"  
#                                 ])
```

```
In [ ]: # %%time  
# np.random.seed(42)  
# full_pipeline_with_predictor = Pipeline([  
#     ("preparation", data_prep_pipeline),  
#     ("linear", LogisticRegression())  
# ])  
# model = full_pipeline_with_predictor.fit(X_train, y_train)
```

```
In [ ]: # from sklearn.metrics import accuracy_score  
  
# np.round(accuracy_score(y_train, model.predict(X_train)), 3)
```

## Experiment 1: Using Selected Features (Baseline Model)

## Joining application\_train.csv and other CSVs

```
In [ ]: df_app=pd.DataFrame(datasets["application_train"])
df_bureau= pd.DataFrame(datasets["bureau"])
df_bureau_balance= pd.DataFrame(datasets["bureau_balance"])
df_prev_app = pd.DataFrame(datasets["previous_application"])
df_pos = pd.DataFrame(datasets["POS_CASH_balance"])
df_inst = pd.DataFrame(datasets ["installments_payments"])
df_credit_card_balance=pd.DataFrame(datasets["credit_card_balance"])

In [ ]: df_tiny = df_app.head(50000).copy()
df_bureau = df_bureau.merge(df_bureau_balance.reset_index(), how='left', on=
df_bureau = df_bureau.groupby("SK_ID_BUREAU").mean().reset_index()

In [ ]: df_tiny = df_tiny.merge(df_bureau.reset_index(), how="left", on="SK_ID_CURR")
df_tiny.shape

Out[ ]: (245372, 140)

In [ ]: df_prev_app = df_prev_app.groupby("SK_ID_CURR").mean().reset_index()
df_tiny = df_tiny.merge(df_prev_app.reset_index(), how="left", on="SK_ID_CURR")
df_tiny.shape

Out[ ]: (245372, 160)

In [ ]: df_pos = df_pos.groupby("SK_ID_CURR").mean().reset_index()
df_tiny = df_tiny.merge(df_pos.reset_index(), how="left", on="SK_ID_CURR",
df_tiny.shape

Out[ ]: (245372, 167)

In [ ]: df_inst = df_inst.groupby("SK_ID_CURR").mean().reset_index()
df_tiny = df_tiny.merge(df_inst.reset_index(), how="left", on="SK_ID_CURR",
df_tiny.shape

Out[ ]: (245372, 175)

In [ ]: df_credit_card_balance = df_credit_card_balance.groupby("SK_ID_CURR").mean()
df_tiny = df_tiny.merge(df_credit_card_balance.reset_index(), how="left", on="SK_ID_CURR")
df_tiny.shape

Out[ ]: (245372, 191)

In [ ]: y = df_tiny["TARGET"]
x = df_tiny.drop("TARGET", axis=1)
x_train, x_valid, y_train, y_valid = train_test_split(x, y, test_size=0.15,
x_train, x_test, y_train, y_test = train_test_split(x_train, y_train, test_s
```

## Selecting Features

After performing the EDA, we have got some important columns that has the highest affect on the Target variable. It means that these columns could predict the outcome instead of using all the columns.

**EXT\_SOURCE\_1**, **EXT\_SOURCE\_2**, and **EXT\_SOURCE\_3** : These columns are moderately correlated with the target variable and also moderately correlated with each other, so they may be good features to include in a model.

**DAYS\_BIRTH** : This column is moderately negatively correlated with the target variable, indicating that younger applicants are more likely to default on their loans.

**DAYS\_EMPLOYED** : This column has a weak negative correlation with the target variable, indicating that longer-term employment may be associated with a lower risk of default.

**AMT\_CREDIT** : This column has a moderate positive correlation with the target variable, indicating that higher loan amounts may be associated with a higher risk of default.

**AMT\_GOODS\_PRICE** : This column has a moderate positive correlation with the target variable, similar to AMT\_CREDIT.

The following columns have also been selected to train the model because after merging the 'TARGET' column from application\_train.csv to different dataset files such as instalment\_payments, previous\_application, pos\_cash\_balance, bureau and bureau\_balance, and plotting the correlation matrix we inferred that these columns had correlation more than 0.3 with target column: **DAYS\_CREDIT**, **DAYS\_CREDIT\_UPDATE**, **DAYS\_ENDDATE\_FACT**, **DAYS\_ENTRY\_PAYMENT**, **DAYS\_INSTALMENT**, **DAYS\_DECISION**, **DAYS\_FIRST\_DRAWING** and **CNT\_PAYMENT**.

```
In [ ]: # extracting the cat and num features
categorical_feat = ['NAME_CONTRACT_TYPE', 'NAME_FAMILY_STATUS', 'OCCUPATION_TYPE']
numerical_features= ['AMT_INCOME_TOTAL', 'AMT_CREDIT', 'DAYS_EMPLOYED', 'DAYS_BIRTH',
'EXT_SOURCE_2', 'EXT_SOURCE_3', 'DAYS_CREDIT', 'DAYS_CREDIT_UPDATE', 'DAYS_ENDDATE_FACT', 'DAYS_ENTRY_PAYMENT', 'DAYS_INSTALMENT', 'DAYS_DECISION', 'DAYS_FIRST_DRAWING', 'CNT_PAYMENT']
```

```
In [ ]: x_train[categorical_feat].nunique().sort_values()
```

```
Out[ ]: NAME_CONTRACT_TYPE      2
NAME_FAMILY_STATUS       6
OCCUPATION_TYPE        18
ORGANIZATION_TYPE      58
dtype: int64
```

```
In [ ]: print(len(categorical_feat), categorical_feat)
print(len(numerical_features), numerical_features)

4 ['NAME_CONTRACT_TYPE', 'NAME_FAMILY_STATUS', 'OCCUPATION_TYPE', 'ORGANIZATION_TYPE']
15 ['AMT_INCOME_TOTAL', 'AMT_CREDIT', 'DAYS_EMPLOYED', 'DAYS_BIRTH', 'EXT_SOURCE_1', 'EXT_SOURCE_2', 'EXT_SOURCE_3', 'DAYS_CREDIT', 'DAYS_CREDIT_UPDATE', 'DAYS_ENDDATE_FACT', 'DAYS_DECISION', 'DAYS_ENTRY_PAYMENT', 'DAYS_INSTALMENT', 'DAYS_FIRST_DRAWING', 'CNT_PAYMENT']
```

```
In [ ]: exp_log = pd.DataFrame(columns=["Pipeline", "Parameters", "TrainAcc", "Valid  
          "Train Time(s)", "Test Time(
```

## Pipeline

- This code establishes a preparation pipeline for the HCDR dataset using scikit-learn's ColumnTransformer. Both category and numerical features are likely included in the dataset, thus the preparation pipeline must treat them separately.
- To change categorical and numerical information, the algorithm first generates two distinct pipelines. The pipeline applies one-hot encoding to categorical features before using SimpleImputer to fill in missing values with the value that occurs the most frequently. The pipeline uses StandardScaler to scale the data for numerical characteristics and SimpleImputer to replace missing values with the median value.
- The appropriate pipeline is then applied to each type of feature using the ColumnTransformer that was established before. Categorical\_trans pipeline is applied to categorical characteristics by the "cat" transformer, whereas the numerical\_trans pipeline is applied to numerical features by the "num" transformer.
- The ColumnTransformer object is added to the parameter of the transformers to establish the preprocessing pipeline. The data can then be preprocessed using the resulting pp\_pipe object before being fitted to a machine-learning model.
- Overall, by applying the proper transformations to each type of feature, this code is building a preprocessing pipeline that can handle both categorical and numerical characteristics in the HCDR dataset. The pipeline can assist in making sure that the data is appropriately prepared for modeling and can increase the precision of machine learning models developed using the dataset.

```
In [ ]: from sklearn.compose import ColumnTransformer  
# Categorical features are transformed  
categorical_trans = Pipeline(  
    steps=[  
        ('ohe', OneHotEncoder(handle_unknown='ignore')),  
        ('imputer', SimpleImputer(strategy='most_frequent'))  
    ]  
)  
# Numerical features are transformed  
numerical_trans = Pipeline(  
    steps=[  
        ('scaler', StandardScaler()),  
        ('imputer', SimpleImputer(strategy='median'))  
    ]  
)  
# Creating a preprocessing Pipeline  
pp_pipe = ColumnTransformer(  
    transformers=[  
        ('cat', categorical_trans, categorical_feat),
```

```
        ('num', numerical_trans, numerical_features)
    )
)
```

```
In [ ]: x_train = pp_pipe.fit_transform(x_train)
x_valid = pp_pipe.transform(x_valid)
x_test = pp_pipe.transform(x_test)
```

```
In [ ]: print(x_train.shape)
print(y_train.shape)

(177281, 100)
(177281,)
```

## Models

We implemented the following models to find which one gives the best performance:

1. Logistic Regression: A common model for situations involving binary classification is logistic regression. It operates by utilizing a logistic function to estimate the likelihood of the target variable (default) with respect to the input parameters (such as age, income, and loan amount). Based on the applicant's demographic and financial data, logistic regression can be applied in the HCDR dataset to forecast the likelihood of default.
2. Random Forest Classifier: Several Decision Trees are used in an ensemble model called Random Forest to lessen overfitting and boost accuracy. It operates by randomly choosing a subset of features at each split while training a series of Decision Trees on bootstrapped subsets of the data.
3. Decision Tree Classifier: A decision tree model divides the data recursively into subsets depending on the most significant attributes until a stopping requirement is satisfied. As a result, a tree-like model is produced, with each internal node standing in for a judgment call based on a feature and each leaf node for a prediction.
4. Adaboost Classifier: A popular machine learning approach for categorization issues is called Adaboost (Adaptive Boosting). It functions by fusing weak classifiers to produce a powerful classifier.

```
In [ ]: import time
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
from warnings import simplefilter
from sklearn.exceptions import ConvergenceWarning
simplefilter("ignore", category=ConvergenceWarning)
from sklearn.ensemble import AdaBoostClassifier
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
```

## Logistic\_Regression

- This code sets up a hyperparameter grid to be searched through during model selection and creates a scikit-learn pipeline for a Logistic Regression model.
- The LogisticRegression() object wrapped in an estimator with the identifier "lr" is the only step used to generate the Pipeline object. The LogisticRegression() object's hyperparameters are specified in the hyperparameter grid using the term 'lr'.
- The Logistic\_Regressionparams dictionary is used to specify the hyperparameter grid. The Logistic Regression model's "C" hyperparameter, which determines the regularization strength, is set to the value 0.01. The "penalty" hyperparameter, which has the values "l1" and "l2" in a list, regulates the kind of regularization the model employs. These variables indicate whether L1 regularization or L2 regularization should be used to fit the model.
- GridSearchCV or another hyperparameter tuning method can use the hyperparameter grid to find the ideal set of hyperparameters for the Logistic Regression model. The L1 and L2 regularization may be compared during the search thanks to this hyperparameter grid configuration, and the value of the 'C' hyperparameter regulates the regularization's strength.
- In general, this code creates a pipeline and hyperparameter grid that can be used for model selection, hyperparameter tuning, and a Logistic Regression model on the HCDR dataset.

```
In [ ]: Logistic_Regression = Pipeline(
    [('lr', LogisticRegression())])
Logistic_Regressionparams = {'lr_C' : [0.01],
                            'lr_penalty' : ['l1', 'l2']}
```

## Decision Tree

- In addition to putting up a hyperparameter grid to be searched through during model selection, the below code creates a scikit-learn Pipeline for a Decision Tree model.
- The DecisionTreeClassifier() object wrapped in an estimator with the name "dt" is the only step required to generate the Pipeline object. The DecisionTreeClassifier()

object's hyperparameters are specified in the hyperparameter grid using the term 'dt'.

- The hyperparameter grid is specified as a dictionary named Decision\_Treeparams. The maximum depth of the Decision Tree is controlled by the 'max\_depth' hyperparameter, which has the values 5 and 10 in its list of possible values. These numbers indicate that a maximum depth of 5 or 10 should be used to fit the model.
- The maximum depth hyperparameter for the Decision Tree model can be tuned using GridSearchCV or any hyperparameter tuning method with the help of the hyperparameter grid. The code enables for the performance of the Decision Tree model to be assessed at various maximum depths and regulates the model's complexity through the maximum depth by adjusting the hyperparameter grid in this manner.
- For a Decision Tree model on the HCDR dataset, this code creates a Pipeline and hyperparameter grid that can be used for hyperparameter tuning and model selection.

```
In [ ]: Decision_Tree = Pipeline(  
    [ ('dt', DecisionTreeClassifier())]  
)  
Decision_Treeparams = { 'dt__max_depth' : [5, 10]}
```

### AdaBoost classifier

- The Pipeline function of Scikit-Learn is used in the code to build an AdaBoost classifier. An ensemble classifier known as the AdaBoostClassifier combines a number of "weak" classifiers to produce a single, stronger classifier. In this instance, a decision tree classifier is the basic estimator.
- Hyperparameters that are provided to the AdaBoostClassifier during training are contained in the AdaBoost\_Classifierparams dictionary. Scikit-learn will try all possible combinations of the hyperparameters to identify the combination that yields the best performance on the dataset. These hyperparameters are supplied in a grid-search format.
- The hyperparameters in this instance are configured as follows:

ada\_\_n\_estimators: The number of weak classifiers to be included in the ensemble is specified by this hyperparameter. The value is set to 20 in this instance.

ada\_\_learning\_rate: This parameter regulates how much each weak classifier contributes to the final ensemble. Each weak classifier has a bigger effect on the result when the learning rate is increased. In this instance, 0.01 and 0.1 learning rates are attempted.

ada\_\_base\_estimator\_\_max\_depth: The base estimator's decision tree's maximum depth is controlled by this hyperparameter. A decision tree that is less complicated and not as prone to overfit the training set has less depth. In this instance, the maximum depths of 1 and 5 are tested.

- Scikit-learn will fit the AdaBoost\_Classifier pipeline using the chosen parameters on the training data. The final model is going to be used to forecast the probability of loan default on a holdout set, and the algorithm's performance may be assessed using measures.

```
In [ ]: AdaBoost_Classifier = Pipeline([ ('ada', AdaBoostClassifier(base_estimator=None)), ('rf', RandomForestClassifier(n_estimators=20, criterion='gini')), ('svc', SVC(C=1, kernel='rbf'))])
AdaBoost_Classifierparams = {
    'ada_n_estimators': [20],
    'ada_learning_rate': [0.01, 0.1],
    'ada_base_estimator_max_depth': [1,5]
}
```

## Random Forest

- This code sets up a hyperparameter grid to be searched through during model selection and creates a scikit-learn Pipeline for a Random Forest model.
- The RandomForestClassifier() object is wrapped in an estimator with the name "rf" to generate the Pipeline object in a single step. The RandomForestClassifier() object's hyperparameters are specified in the hyperparameter grid using the name 'rf'.
- The Random\_Forestparam dictionary is the one used to specify the hyperparameter grid. The 'n\_estimators' hyperparameter, which is specified as 20, regulates the number of decision trees in the Random Forest model. These numbers indicate that 20 decision trees should be used to fit the model. Each decision tree's "criterion" hyperparameter, which is set to the values "gini" and "log\_loss," regulates the quality of the split. These numbers indicate that the decision trees' node splitting criteria should be either Gini impurity or cross-entropy in order to fit the model.
- The hyperparameter grid is made to make it possible for GridSearchCV or any hyperparameter tuning technique to look for the ideal set of hyperparameters for the Random Forest model. The code allows for the comparison of the number of decision trees and the quality of the splits in the decision trees throughout the search and manages the complexity and performance of the model using these hyperparameters by configuring the hyperparameter grid in this manner.
- Overall, this code creates a Pipeline and hyperparameter grid for a Random Forest model on the HCDR dataset that can be used for hyperparameter tuning and model selection.

```
In [ ]: Random_Forest = Pipeline([('rf', RandomForestClassifier())])
Random_Forestparam = {
    'rf_n_estimators' : [20],
    'rf_criterion' : ['gini','log_loss']}
```

Four dictionaries that each represent a machine learning algorithm make up the "main" list:

- Random\_Forest
- Decision\_Tree
- Logistic\_Regression
- AdaBoost\_Classifier

There is a corresponding dictionary of hyperparameters for each algorithm:

- Random\_Forestparam
- Decision\_Treeparams
- Logistic\_Regressionparams
- AdaBoost\_Classifierparams

This code specifies a collection of machine learning models that will be tested on the HCDR dataset. Both a "pipeline" key and a "params" key are present in each dictionary in the "main" list. The machine learning algorithm to be employed is specified using the "pipeline" key, and its hyperparameters are specified using the "params" key.

The optimal model for the given dataset can be found by training and analyzing the models after they have been defined.

```
In [ ]: main = [
    {
        'pipeline' : Random_Forest,
        "params" : Random_Forestparam
    },
    {
        'pipeline' : Decision_Tree,
        "params" : Decision_Treeparams
    },
    {
        'pipeline' : Logistic_Regression,
        "params" : Logistic_Regressionparams
    },
    {
        'pipeline' : AdaBoost_Classifier,
        "params" : AdaBoost_Classifierparams
    }
]
```

```
In [ ]: from sklearn.metrics import roc_auc_score

dict1 = {}
for i in main:
    print(f"Running {i['pipeline']}")
    GS_CV = GridSearchCV(i["pipeline"], i["params"], n_jobs=-1, cv=2, verbose=0)
    GS_CV.fit(x_train, y_train)
    p = GS_CV.best_estimator_
    dict1[i['pipeline']] = GS_CV
    T = time.time()
    Tr_acc = p.score(x_train, y_train)
```

```
Tr_T = time.time() - T
Va_acc = p.score(x_valid, y_valid)
T = time.time()
Te_acc = p.score(x_test, y_test)
Te_T = time.time() - T
AUC_Train=roc_auc_score(y_train, dict1[i['pipeline']].predict_proba(x_tr
AUC_Test=roc_auc_score(y_test, dict1[i['pipeline']].predict_proba(x_test
AUC_val=roc_auc_score(y_valid, dict1[i['pipeline']].predict_proba(x_vali
exp_log.loc[len(exp_log)] =[f"Baseline {i['pipeline']} with {x_train.sha
                                         f"\n{Tr_acc*100:8.2f}%", f"\n{Va_acc
Tr_T, Te_acc,AUC_Train, AUC_val,
print(f"Completed {i['pipeline']}")

print("ROC AUC score")
print("Logistic Regression :",roc_auc_score(y_train, dict1[Logistic_Regressi
print("Decision Tree :",roc_auc_score(y_train, dict1[Decision_Tree].predict_
print("Random Forest :",roc_auc_score(y_train, dict1[Random_Forest].predict_
print("ADA Boost :",roc_auc_score(y_train, dict1[AdaBoost_Classifier].predic
Running Pipeline(steps=[('rf', RandomForestClassifier())])
Fitting 2 folds for each of 2 candidates, totalling 4 fits
Completed Pipeline(steps=[('rf', RandomForestClassifier())])
Running Pipeline(steps=[('dt', DecisionTreeClassifier())])
Fitting 2 folds for each of 2 candidates, totalling 4 fits
Completed Pipeline(steps=[('dt', DecisionTreeClassifier())])
Running Pipeline(steps=[('lr', LogisticRegression())])
Fitting 2 folds for each of 2 candidates, totalling 4 fits
Completed Pipeline(steps=[('lr', LogisticRegression())])
Running Pipeline(steps=[('ada',
                           AdaBoostClassifier(base_estimator=DecisionTreeClassifier
                           ()))])
Fitting 2 folds for each of 4 candidates, totalling 8 fits
```

```
/usr/local/lib/python3.11/site-packages/sklearn/ensemble/_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
/usr/local/lib/python3.11/site-packages/sklearn/ensemble/_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
/usr/local/lib/python3.11/site-packages/sklearn/ensemble/_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
/usr/local/lib/python3.11/site-packages/sklearn/ensemble/_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
/usr/local/lib/python3.11/site-packages/sklearn/ensemble/_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
/usr/local/lib/python3.11/site-packages/sklearn/ensemble/_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
/usr/local/lib/python3.11/site-packages/sklearn/ensemble/_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4.
    warnings.warn(
Completed Pipeline(steps=[('ada',
                           AdaBoostClassifier(base_estimator=DecisionTreeClassifier
                           ()))])
ROC AUC score
Logistic Regression : 0.7470648772667208
Decision Tree : 0.8074479725505205
Random Forest : 0.9999999478492032
ADA Boost : 0.8129134572402118
```

In [ ]: exp\_log

Out[ ]:

Pipeline

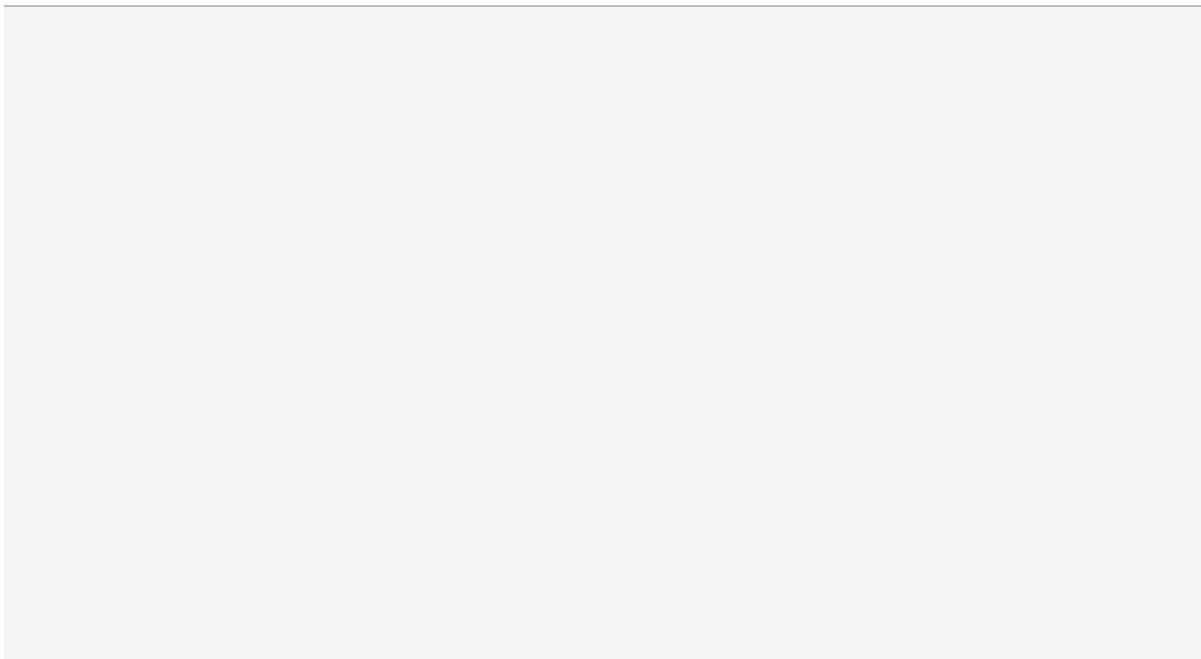
Parameter

0	Baseline Pipeline(steps=[('rf', RandomForestClassifier())]) with 100 inputs	{'rf__n_estimators': [20], 'rf__criterion': ['gini', 'log_loss']}
---	--	--

1	Baseline Pipeline(steps=[('dt', DecisionTreeClassifier())]) with 100 inputs	{'dt__max_depth': [5, 10]}
---	--	----------------------------

2	Baseline Pipeline(steps=[('lr', LogisticRegression())]) with 100 inputs	{'lr__C': [0.01], 'lr__penalty': ['l1', 'l2']}
---	--	---

3	Baseline Pipeline(steps=[('ada', AdaBoostClassifier(base_estimator=DecisionTreeClassifier()))]) with 100 inputs	{'ada__n_estimators': [20], 'ada__learning_rate': [0.01, 0.1], 'ada__base_estimator__max_depth': [1, 5]}
---	---	--



## Plotting Maximum Depth of Random Forest Scores

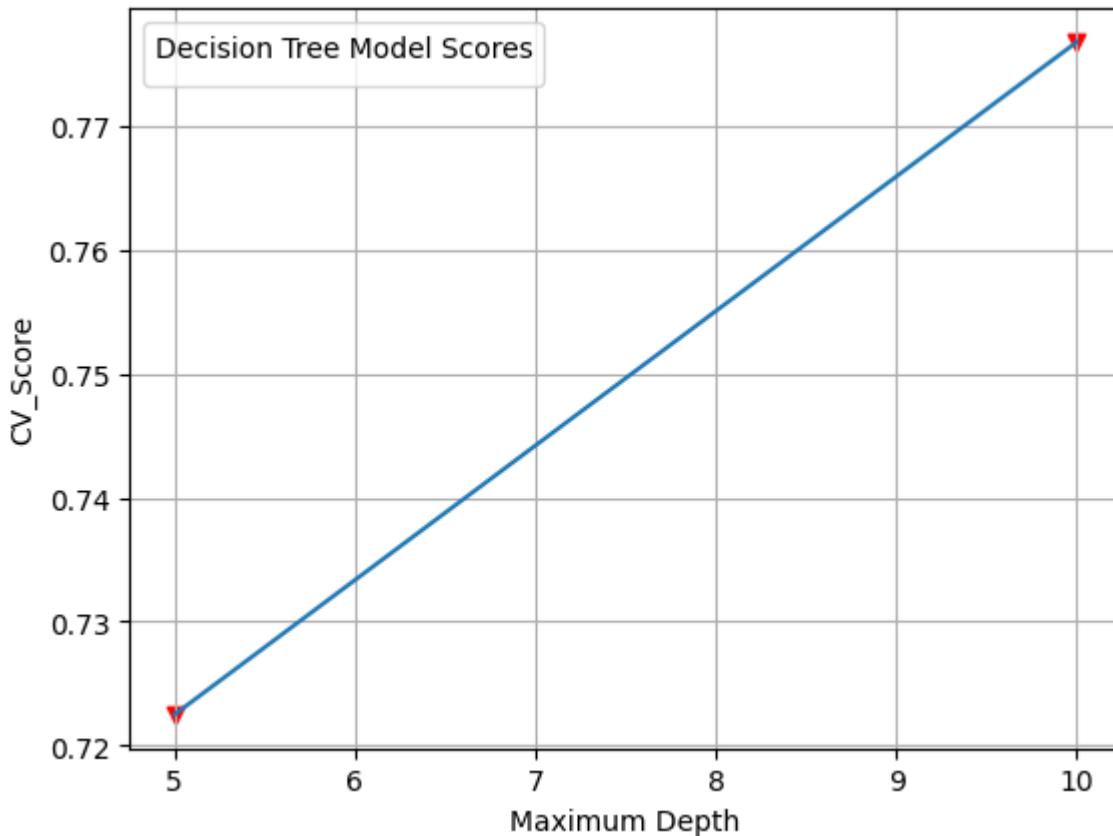
```
In [ ]: DT = dict1[Decision_Tree]
DT_analysis = pd.DataFrame(DT.cv_results_['params'])
DT_analysis['cv_score'] = DT.cv_results_['mean_test_score']
```

```
In [ ]: x1 = []
y1 = []
```

```
In [ ]: for element in Decision_Treeparams['dt__max_depth']:
    num = DT_analysis.query(f'dt__max_depth == {element}')
    y1.append(num.cv_score), x1.append(num.dt__max_depth)
```

```
In [ ]: plt.plot(x1, y1)
plt.ylabel('CV_Score')
plt.xlabel('Maximum Depth')
plt.legend(title='Decision Tree Model Scores')
plt.scatter(x=x1, y=y1, marker='v', color='r')
plt.grid()
plt.show()
```

No artists with labels found to put in legend. Note that artists whose label start with an underscore are ignored when legend() is called with no argument.



```
In [ ]: for i, j in zip(DT.cv_results_['params'], DT.cv_results_['mean_test_score']):
    print(f"depth: {i['dt__max_depth']}>3}, score: {j:.4f}")

depth: 5, score: 0.7225
depth: 10, score: 0.7768
```

## Combining the test data with the remaining CSVs

```
In [ ]: df_bureau = df_bureau.merge(df_bureau_balance.reset_index(), how='left', on="SK_ID_BUREAU")
df_bureau = df_bureau.groupby("SK_ID_BUREAU").mean().reset_index()

In [ ]: df_app_test=pd.DataFrame(datasets["application_test"])
df_app_test = df_app_test.merge(df_bureau.reset_index(), how="left", on="SK_ID_BUREAU")
df_app_test.shape

Out[ ]: (257527, 141)

In [ ]: df_prev_app = df_prev_app.groupby("SK_ID_CURR").mean().reset_index()
df_app_test= df_app_test.merge(df_prev_app.reset_index(), how="left", on="SK_ID_CURR")
df_app_test.shape

Out[ ]: (257527, 162)

In [ ]: df_pos = df_pos.groupby("SK_ID_CURR").mean().reset_index()
df_app_test = df_app_test.merge(df_pos.reset_index(), how="left", on="SK_ID_CURR")
df_app_test.shape
```

```
Out[ ]: (257527, 169)
```

```
In [ ]: df_inst = df_inst.groupby("SK_ID_CURR").mean().reset_index()
df_app_test = df_app_test.merge(df_inst.reset_index(), how="left", on="SK_ID_CURR")
df_app_test.shape
```

```
Out[ ]: (257527, 177)
```

```
In [ ]: df_credit_card_balance = df_credit_card_balance.groupby("SK_ID_CURR").mean()
df_app_test = df_app_test.merge(df_credit_card_balance.reset_index(), how="left", on="SK_ID_CURR")
df_app_test.shape
```

```
Out[ ]: (257527, 199)
```

## Data Lineage

- The data lineage for this project began with the raw data, which was initial preprocessing to treat missing values, such as through imputation techniques. The next step was Exploratory Data Analysis (EDA) to understand the underlying patterns and relationships in the data, followed by visual EDA to gain additional insights into the data.
- After that, feature selection was performed to identify the most important features for the model. This was followed by training baseline models to establish a benchmark for model performance of our base line models.
- Overall, it involved multiple steps, including preprocessing, EDA, visual EDA, feature selection, and training baseline models, with each step building upon the previous one to create a robust and reliable ML model. This workflow was to ensure that the data was clean, consistent, and appropriate for the ML model, providing transparency and accountability for all data-related activities.

## Feature Engineering

In Phase 2, we did EDA and understood more about the dataset and ran baseline models subset of dataset with features that has high correlation values with the target variable.

In Phase3, we start with feature engineering. In this section, we first grouped the respective DataFrame by the SK\_ID\_CURR column and calculated the median value for each column. This is being done to aggregate the data and summarize the history of each borrower. We then merged "DF\_Bureau" and "DF\_BureauBalance", by combining the data from the "DF\_Bureau" and "DF\_BureauBalance" databases, this method aims to produce a new dataset with summary statistics by "SK\_ID\_CURR". In machine learning models to forecast credit default risk, the generated dataset can be used as a feature. To

take care of high proportion of zeros, we created a function Count\_Percentage\_of\_Zeroes. This function's goal is to locate the dataset columns with the highest proportion of zero values. These columns can be eliminated from the dataset since they could not contain data that is helpful for estimating credit default risk. We then have function names Divide\_DataFrame\_into\_Categorical\_and\_Numerical, this function's goal is to get the data ready for additional modeling and analysis. While categorical columns need to be converted into numerical representations, numerical columns can be used directly in many machine-learning techniques. We also have a function named Correlation\_Between\_InputDF\_Target, in this function the goal column "TARGET" and the columns of the input data frame are correlated. The columns whose correlation value is smaller than the input parameter correlation's absolute value are then removed. Then the numerical columns in the dataset New\_DataFrame\_Numerical are subjected to missing value imputation.

For adding new features to the dataset, we establish a pipeline that separates and then combines the categorical and numerical elements into a single data frame. First, two pipelines are established separately for the category and numerical features. While the categorical pipeline replaces missing values with the most frequent value of each feature using SimpleImputer() and then applies one-hot encoding using OneHotEncoder, the numerical pipeline replaces missing values with the mean of each feature using SimpleImputer(). In order to specify which pipeline should be used for each set of columns, the two pipelines are then concatenated into a single ColumnTransformer() object. The two pipelines are then applied to the corresponding sets of columns in DF\_Temp1 by calling the fit\_transform() method of the Data\_Pipeline object, which results in the creation of a single numpy array. In the end, the OneHotEncoder() object's get\_feature\_names\_out() method is used to retrieve the feature names of the one-hot encoded categorical features. The resulting numpy array is then transformed into a pandas data frame by combining the numerical and categorical feature names. Finally, we create 13 new features (Feature\_1 to Feature\_13) utilizing different columns from the consolidated data frame after merging numerous data frames into one (DF\_Final). Mathematical procedures such as division, multiplication, and addition are used to calculate these additional properties. The final data frame's shape is subsequently determined by the algorithm and assigned to the variable DF\_Final. In order to calculate the correlation between the newly produced features and the goal variable "TARGET," the final line of code invokes the function Correlation\_Between\_InputDF\_Target. The function generates a correlation matrix after calculating the Pearson correlation coefficient between each feature in the input and the desired variable. To only include features with a correlation coefficient greater than or equal to 0.0, a threshold value of 0.0 is crossed. Finding the newly developed features that have the highest correlation with the objective variable and may one day serve as predictors in a machine learning model is the goal of this stage.

We had added the following new features:

```

1. DF_Final['Feature_1']=  

   DF_Final['AMT_CREDIT_SUM']/(DF_Final['DAYS_EMPLOYED']+1)  

2. DF_Final['Feature_2']=  

   DF_Final['CNT_DRAWINGS_POS_CURRENT']/(DF_Final['FLOORSMIN_AVG']+1)  

3. DF_Final['Feature_3']=  

   DF_Final['AMT_CREDIT_SUM']/(DF_Final['FLOORSMIN_AVG']+1)  

4. DF_Final['Feature_4']=DF_Final['DAYS_CREDIT']*  

   (DF_Final['DAYS_ENDDATE_FACT']+1)  

5. DF_Final['Feature_5']=DF_Final['AMT_CREDIT_SUM']/(DF_Final['AMT_CREDIT_SUM']+1)  

6. DF_Final['Feature_6']=DF_Final['AMT_CREDIT_x']/(DF_Final['AMT_CREDIT_SUM']+1)  

7. DF_Final['Feature_7']=(DF_Final['DAYS_CREDIT']+DF_Final['AMT_CREDIT_SUM'])  

8. DF_Final['Feature_8']=(DF_Final['AMT_CREDIT_x']+DF_Final['FLOORSMIN_AVG'])  

9. DF_Final['Feature_9']=  

   DF_Final['AMT_CREDIT_SUM']/(DF_Final['AMT_GOODS_PRICE_x']+1)  

10. DF_Final['Feature_10']=  

   DF_Final['AMT_CREDIT_SUM']/(DF_Final['AMT_GOODS_PRICE_x']+1)  

11. DF_Final['Feature_11']=DF_Final['DAYS_BIRTH']+(DF_Final['DAYS_EMPLOYED']+1)  

12. DF_Final['Feature_12']=  

   (DF_Final['EXT_SOURCE_1']+DF_Final['EXT_SOURCE_2']+DF_Final['EXT_SOURCE_3'])  

13. DF_Final['Feature_13']=DF_Final['CNT_INSTALMENT_MATURE_CUM']*  

   (DF_Final['CNT_DRAWINGS_POS_CURRENT']+1)

```

We also added the following RFM features:

```

1. DF_Final['RFM']=hcdr_rfm["RFM"]  

2. DF_Final['R']=hcdr_rfm["R"]  

3. DF_Final['F']=hcdr_rfm["F"]  

4. DF_Final['M']=hcdr_rfm["M"]

```

```
In [ ]: import numpy as np  
import pandas as pd
```

```
In [ ]: DF_Train = datasets["application_train"]  
DF_Test = datasets["application_test"]  
DF_Bureau = datasets["bureau"]  
DF_BureauBalance = datasets["bureau_balance"]  
DF_PosCashBalance = datasets["POS_CASH_balance"]  
DF_PreviousApplication = datasets["previous_application"]  
DF_CreditCardBalance = datasets["credit_card_balance"]  
DF_InstallmentsPayemnts = datasets["installments_payments"]
```

```
In [ ]: # import numpy as np  
# import pandas as pd  
# DF_Train= pd.read_csv("application_train.csv")  
# DF_Test = pd.read_csv("application_test.csv")
```

```
# DF_Bureau = pd.read_csv("bureau.csv")
# DF_BureauBalance = pd.read_csv("bureau_balance.csv")
# DF_PosCashBalance = pd.read_csv("POS_CASH_balance.csv")
# DF_PreviousApplication = pd.read_csv("previous_application.csv")
# DF_CreditCardBalance = pd.read_csv("credit_card_balance.csv")
# DF_InstallmentsPayemnts = pd.read_csv("installments_payments.csv")
```

## Create a sample dataset from application\_train.csv and join with other CSV's

Here the DataFrames are being manipulated using pandas functions. We are selecting only columns from the respective DataFrame that do not have an object data type, i.e., columns that are numeric. This is done to prepare the data for grouping and aggregation. Then we group the respective DataFrame by the SK\_ID\_CURR column and calculates the median value for each column. This is being done to aggregate the data and summarize the history for each borrower.

```
In [ ]: DF_InstallmentsPayemnts = DF_InstallmentsPayemnts.select_dtypes(exclude='object')
DF_InstallmentsPayemnts = DF_InstallmentsPayemnts.groupby('SK_ID_CURR')
DF_InstallmentsPayemnts = DF_InstallmentsPayemnts.median()

DF_PosCashBalance = DF_PosCashBalance.select_dtypes(exclude='object')
DF_PosCashBalance = DF_PosCashBalance.groupby('SK_ID_CURR')
DF_PosCashBalance = DF_PosCashBalance.median()

DF_PreviousApplication = DF_PreviousApplication.select_dtypes(exclude='object')
DF_PreviousApplication = DF_PreviousApplication.groupby('SK_ID_CURR')
DF_PreviousApplication = DF_PreviousApplication.median()

DF_CreditCardBalance = DF_CreditCardBalance.select_dtypes(exclude='object')
DF_CreditCardBalance = DF_CreditCardBalance.groupby('SK_ID_CURR')
DF_CreditCardBalance = DF_CreditCardBalance.median()
```

Using the shared column "SK\_ID\_BUREAU," the code merges the "DF\_Bureau" and "DF\_BureauBalance" datasets. The generated dataset is then grouped by "SK\_ID\_CURR" and "SK\_ID\_BUREAU" and duplicates are removed. The next step is to further filter the output dataset to maintain just particular columns, after which it is grouped by "SK\_ID\_CURR" and the median is determined.

By combining the data from the "DF\_Bureau" and "DF\_BureauBalance" databases, this method aims to produce a new dataset with summary statistics by "SK\_ID\_CURR". In machine learning models to forecast credit default risk, the generated dataset can be used as a feature.

```
In [ ]: Merged_BureauBalance = pd.merge(left=DF_Bureau, right=DF_BureauBalance, how='left')
Merged_BureauBalance = Merged_BureauBalance.drop_duplicates()
Merged_BureauBalance = Merged_BureauBalance.groupby(['SK_ID_CURR', 'SK_ID_BUREAU'])
Merged_BureauBalance = Merged_BureauBalance.min()
```

```
In [ ]: Updated_BureauBalance = Merged_BureauBalance[['DAYS_CREDIT_UPDATE', 'DAYS_CREDIT']]
Updated_BureauBalance = Updated_BureauBalance.reset_index()
Updated_BureauBalance = Updated_BureauBalance.groupby('SK_ID_CURR')
Updated_BureauBalance = Updated_BureauBalance.median()
Updated_BureauBalance = Updated_BureauBalance.reset_index()

In [ ]: Updated_BureauBalance = Updated_BureauBalance.select_dtypes(exclude='object')
Updated_BureauBalance.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 305811 entries, 0 to 305810
Data columns (total 7 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   SK_ID_CURR       305811 non-null   int64  
 1   SK_ID_BUREAU     305811 non-null   float64 
 2   DAYS_CREDIT_UPDATE 305811 non-null   float64 
 3   DAYS_CREDIT      305811 non-null   float64 
 4   AMT_CREDIT_SUM    305809 non-null   float64 
 5   MONTHS_BALANCE    134542 non-null   float64 
 6   DAYS_ENDDATE_FACT 268155 non-null   float64 
dtypes: float64(6), int64(1)
memory usage: 16.3 MB
```

## Treating highest proportion of zero values

DataFrame and Thres\_Percentage are the two inputs that the function Count\_Percentage\_of\_Zeroes, which is defined in the code below, takes.

With the exception of the "TARGET" column, this function determines the proportion of zeros in each column of the supplied DataFrame. When the Thres\_Percentage parameter is exceeded, a new data frame out containing the columns with the highest proportion of zeros is created. (given as a percentage).

This function's goal is to locate the dataset columns with the highest proportion of zero values. These columns can be eliminated from the dataset since they could not contain data that is helpful for estimating credit default risk.

```
In [ ]: def Count_Percentage_of_Zeroes(DataFrame,Thres_Percentage):
    out = pd.DataFrame()
    temp1 = []
    temp2 = []
    for i in DataFrame.columns:
        if i == 'TARGET':
            continue
        count = (DataFrame[i] == 0).sum()
        temp1.append(i)

        temp2.append(count/len(DataFrame[i]))
    out['Column'] = temp1
    out['Percentage'] = temp2
    Thres_Percentage = Thres_Percentage/100
```

```

        out = out[out['Percentage'] > Thres_Percentage]
    return out
UnwantedColumns = Count_Percentage_of_Zeroes(DF_Train, 85)
print(UnwantedColumns)

```

	Column	Percentage
26	FLAG_EMAIL	0.943280
33	REG_REGION_NOT_LIVE_REGION	0.984856
34	REG_REGION_NOT_WORK_REGION	0.949231
35	LIVE_REGION_NOT_WORK_REGION	0.959341
36	REG_CITY_NOT_LIVE_CITY	0.921827
91	DEF_30_CNT_SOCIAL_CIRCLE	0.882323
93	DEF_60_CNT_SOCIAL_CIRCLE	0.912881
95	FLAG_DOCUMENT_2	0.999958
97	FLAG_DOCUMENT_4	0.999919
98	FLAG_DOCUMENT_5	0.984885
99	FLAG_DOCUMENT_6	0.911945
100	FLAG_DOCUMENT_7	0.999808
101	FLAG_DOCUMENT_8	0.918624
102	FLAG_DOCUMENT_9	0.996104
103	FLAG_DOCUMENT_10	0.999977
104	FLAG_DOCUMENT_11	0.996088
105	FLAG_DOCUMENT_12	0.999993
106	FLAG_DOCUMENT_13	0.996475
107	FLAG_DOCUMENT_14	0.997064
108	FLAG_DOCUMENT_15	0.998790
109	FLAG_DOCUMENT_16	0.990072
110	FLAG_DOCUMENT_17	0.999733
111	FLAG_DOCUMENT_18	0.991870
112	FLAG_DOCUMENT_19	0.999405
113	FLAG_DOCUMENT_20	0.999493
114	FLAG_DOCUMENT_21	0.999665
115	AMT_REQ_CREDIT_BUREAU_HOUR	0.859696
116	AMT_REQ_CREDIT_BUREAU_DAY	0.860142

## Divinding main df into categorical and numerical dataframe

The function Divide\_DataFrame\_into\_Categorical\_and\_Numerical, which accepts a single argument DataFrame and divides it into two dataframes—one with only numerical columns (excluding the "TARGET" column) and the other with only categorical columns—is defined in the code below.

This function's goal is to get the data ready for additional modeling and analysis. While categorical columns need to be converted into numerical representations, numerical columns can be used directly in many machine learning techniques.

The function is defined, then the DF\_Train data frame is passed through the function to produce two new data frames, New\_DataFrame\_Numerical and New\_DataFrame\_Categorical, which include the numerical and categorical features,

respectively. The describe() function is then used to print out the numerical data frame's summary statistics.

```
In [ ]: DF_Train.drop(columns = UnwantedColumns['Column'], inplace = True)
def Divide_DataFrame_into_Categorical_and_Numerical(DataFrame):
    Numerical_DataFrame= DataFrame.select_dtypes(exclude='object')
    Numerical_DataFrame['TARGET'] = DataFrame['TARGET']
    Categorical_DataFrame= DataFrame.select_dtypes(include='object')
    return Numerical_DataFrame,Categorical_DataFrame
New_DataFrame_Numerical, New_DataFrame_Categorical = Divide_DataFrame_into_C
New_DataFrame_Numerical.describe()
```

```
Out[ ]:
```

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT
<b>count</b>	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05
<b>mean</b>	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05
<b>std</b>	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05
<b>min</b>	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04
<b>25%</b>	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05
<b>50%</b>	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05
<b>75%</b>	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05
<b>max</b>	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06

8 rows × 78 columns

---

```
In [ ]: New_DataFrame_Numerical.describe().T
```

Out[ ]:

		count	mean	std	min	max
	<b>SK_ID_CURR</b>	307511.0	278180.518577	102790.175348	100002.0	1891
	<b>TARGET</b>	307511.0	0.080729	0.272419	0.0	1.0
	<b>CNT_CHILDREN</b>	307511.0	0.417052	0.722121	0.0	5.0
	<b>AMT_INCOME_TOTAL</b>	307511.0	168797.919297	237123.146279	25650.0	1125
	<b>AMT_CREDIT</b>	307511.0	599025.999706	402490.776996	45000.0	2700
	...	...	...	...	...	...
	<b>FLAG_DOCUMENT_3</b>	307511.0	0.710023	0.453752	0.0	1.0
	<b>AMT_REQ_CREDIT_BUREAU_WEEK</b>	265992.0	0.034362	0.204685	0.0	1.0
	<b>AMT_REQ_CREDIT_BUREAU_MON</b>	265992.0	0.267395	0.916002	0.0	1.0
	<b>AMT_REQ_CREDIT_BUREAU_QRT</b>	265992.0	0.265474	0.794056	0.0	1.0
	<b>AMT_REQ_CREDIT_BUREAU_YEAR</b>	265992.0	1.899974	1.869295	0.0	1.0

78 rows × 8 columns

In [ ]: New\_DataFrame\_Categorical.describe().T

Out[ ]:

	count	unique	top	freq
<b>NAME_CONTRACT_TYPE</b>	307511	2	Cash loans	278232
<b>CODE_GENDER</b>	307511	3	F	202448
<b>FLAG_OWN_CAR</b>	307511	2	N	202924
<b>FLAG_OWN_REALTY</b>	307511	2	Y	213312
<b>NAME_TYPE_SUITE</b>	306219	7	Unaccompanied	248526
<b>NAME_INCOME_TYPE</b>	307511	8	Working	158774
<b>NAME_EDUCATION_TYPE</b>	307511	5	Secondary / secondary special	218391
<b>NAME_FAMILY_STATUS</b>	307511	6	Married	196432
<b>NAME_HOUSING_TYPE</b>	307511	6	House / apartment	272868
<b>OCCUPATION_TYPE</b>	211120	18	Laborers	55186
<b>WEEKDAY_APPR_PROCESS_START</b>	307511	7	TUESDAY	53901
<b>ORGANIZATION_TYPE</b>	307511	58	Business Entity Type 3	67992
<b>FONDKAPREMONT_MODE</b>	97216	4	reg oper account	73830
<b>HOUSETYPE_MODE</b>	153214	3	block of flats	150503
<b>WALLSMATERIAL_MODE</b>	151170	7	Panel	66040
<b>EMERGENCYSTATE_MODE</b>	161756	2	No	159428

## Correlation with respect to target variable

DataFrame and correlation are the two parameters required by the function

Correlation\_Between\_InputDF\_Target. The goal column "TARGET" and the columns of the input dataframe are correlated. The columns whose correlation value is smaller than the input parameter correlation's absolute value are then removed.

The function produces a pandas dataframe with the columns "col\_name" (the name of the column) and "correlation" in it. Only columns whose correlation value with the target is higher than the input parameter correlation's absolute value are included in the data frame that is delivered.

```
In [ ]: def Correlation_Between_InputDF_Target(DataFrame,correlation):
    temp = DataFrame.corr()['TARGET']
    temp = temp.sort_values(key=abs,ascending=False)
    temp=temp.reset_index()
    temp.columns = ['col_name','Correlation']
    out = temp[abs(temp['Correlation'])>correlation]
    return out
p= Correlation_Between_InputDF_Target(New_DataFrame_Numerical,0.00)
p
```

```
Out[ ]:
```

	col_name	Correlation
0	TARGET	1.000000
1	EXT_SOURCE_3	-0.178919
2	EXT_SOURCE_2	-0.160472
3	EXT_SOURCE_1	-0.155317
4	DAYS_BIRTH	0.078239
...	...	...
73	AMT_REQ_CREDIT_BUREAU_QRT	-0.002022
74	NONLIVINGAPARTMENTS_MODE	-0.001557
75	AMT_REQ_CREDIT_BUREAU_WEEK	0.000788
76	FLAG_MOBIL	0.000534
77	FLAG_CONT_MOBILE	0.000370

78 rows × 2 columns

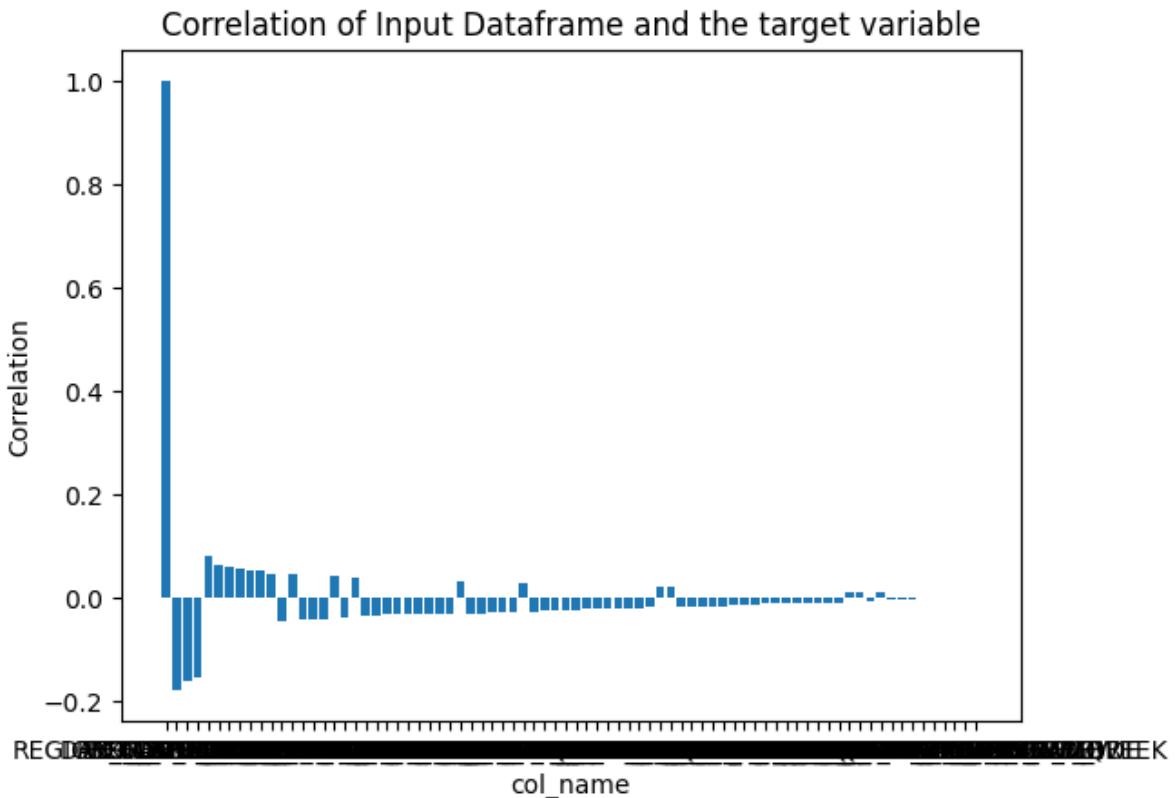
```
In [ ]: import matplotlib.pyplot as plt
import numpy as np

# Creating a new figure
plt.figure(dpi=100)
# Plotting Numpy array
plt.bar(p["col_name"],p["Correlation"])
```

```

# Adding details to the plot
plt.title('Correlation of Input Dataframe and the target variable')
plt.xlabel('col_name')
plt.ylabel('Correlation')
# Displaying the plot
plt.show()

```



In the code, the DataFrame DF\_Train is filtered to remove rows with invalid or irrelevant values for the columns 'NAME\_FAMILY\_STATUS', 'CODE\_GENDER', and 'NAME\_INCOME\_TYPE'. Specifically, rows where the 'NAME\_FAMILY\_STATUS' is 'Unknown', 'CODE\_GENDER' is 'XNA', or 'NAME\_INCOME\_TYPE' is 'Maternity leave' are removed.

Then, correlation\_target() function is called to calculate the correlation between the remaining columns in the New\_DataFrame\_Numerical DataFrame and the 'TARGET' column, and to return only the columns with a correlation coefficient greater than 0.02 or less than -0.02. The resulting DataFrame is stored in temp\_df.

```

In [ ]: DF_Train =DF_Train[DF_Train['NAME_INCOME_TYPE']!='Maternity leave']
DF_Train =DF_Train[DF_Train['NAME_FAMILY_STATUS']!='Unknown']
DF_Train =DF_Train[DF_Train['CODE_GENDER']!='XNA']
New_DataFrame = Correlation_Between_InputDF_Target(New_DataFrame_Numerical, e)
New_DataFrame.describe()

```

Out [ ]: Correlation

<b>count</b>	46.000000
<b>mean</b>	0.003511
<b>std</b>	0.159330
<b>min</b>	-0.178919
<b>25%</b>	-0.033559
<b>50%</b>	-0.026157
<b>75%</b>	0.031520
<b>max</b>	1.000000

In [ ]: New\_DataFrame

Out[ ]:

	col_name	Correlation
0	TARGET	1.000000
1	EXT_SOURCE_3	-0.178919
2	EXT_SOURCE_2	-0.160472
3	EXT_SOURCE_1	-0.155317
4	DAY_S_BIRTH	0.078239
5	REGION_RATING_CLIENT_W_CITY	0.060893
6	REGION_RATING_CLIENT	0.058899
7	DAY_S_LAST_PHONE_CHANGE	0.055218
8	DAY_S_ID_PUBLISH	0.051457
9	REG_CITY_NOT_WORK_CITY	0.050994
10	FLAG_EMP_PHONE	0.045982
11	DAY_S_EMPLOYED	-0.044932
12	FLAG_DOCUMENT_3	0.044346
13	FLOORSMAX_AVG	-0.044003
14	FLOORSMAX_MEDI	-0.043768
15	FLOORSMAX_MODE	-0.043226
16	DAY_S_REGISTRATION	0.041975
17	AMT_GOODS_PRICE	-0.039645
18	OWN_CAR_AGE	0.037612
19	REGION_POPULATION_RELATIVE	-0.037227
20	ELEVATORS_AVG	-0.034199
21	ELEVATORS_MEDI	-0.033863
22	FLOORSMIN_AVG	-0.033614
23	FLOORSMIN_MEDI	-0.033394
24	LIVINGAREA_AVG	-0.032997
25	LIVINGAREA_MEDI	-0.032739
26	FLOORSMIN_MODE	-0.032698
27	TOTALAREA_MODE	-0.032596
28	LIVE_CITY_NOT_WORK_CITY	0.032518
29	ELEVATORS_MODE	-0.032131
30	LIVINGAREA_MODE	-0.030685
31	AMT_CREDIT	-0.030369
32	APARTMENTS_AVG	-0.029498
33	APARTMENTS_MEDI	-0.029184

	col_name	Correlation
34	FLAG_WORK_PHONE	0.028524
35	APARTMENTS_MODE	-0.027284
36	LIVINGAPARTMENTS_AVG	-0.025031
37	LIVINGAPARTMENTS_MEDI	-0.024621
38	HOUR_APPR_PROCESS_START	-0.024166
39	FLAG_PHONE	-0.023806
40	LIVINGAPARTMENTS_MODE	-0.023393
41	BASEMENTAREA_AVG	-0.022746
42	YEARS_BUILD_MEDI	-0.022326
43	YEARS_BUILD_AVG	-0.022149
44	BASEMENTAREA_MEDI	-0.022081
45	YEARS_BUILD_MODE	-0.022068

In [ ]: New\_DataFrame\_Numerical.describe().T

Out[ ]:

	count	mean	std	min	max
SK_ID_CURR	307511.0	278180.518577	102790.175348	100002.0	1891
TARGET	307511.0	0.080729	0.272419	0.0	1.0
CNT_CHILDREN	307511.0	0.417052	0.722121	0.0	3.0
AMT_INCOME_TOTAL	307511.0	168797.919297	237123.146279	25650.0	1125
AMT_CREDIT	307511.0	599025.999706	402490.776996	45000.0	2700
...	...	...	...	...	...
FLAG_DOCUMENT_3	307511.0	0.710023	0.453752	0.0	1.0
AMT_REQ_CREDIT_BUREAU_WEEK	265992.0	0.034362	0.204685	0.0	1.0
AMT_REQ_CREDIT_BUREAU_MON	265992.0	0.267395	0.916002	0.0	1.0
AMT_REQ_CREDIT_BUREAU_QRT	265992.0	0.265474	0.794056	0.0	1.0
AMT_REQ_CREDIT_BUREAU_YEAR	265992.0	1.899974	1.869295	0.0	1.0

78 rows × 8 columns

---

In [ ]:

```
import matplotlib.pyplot as plt
import numpy as np

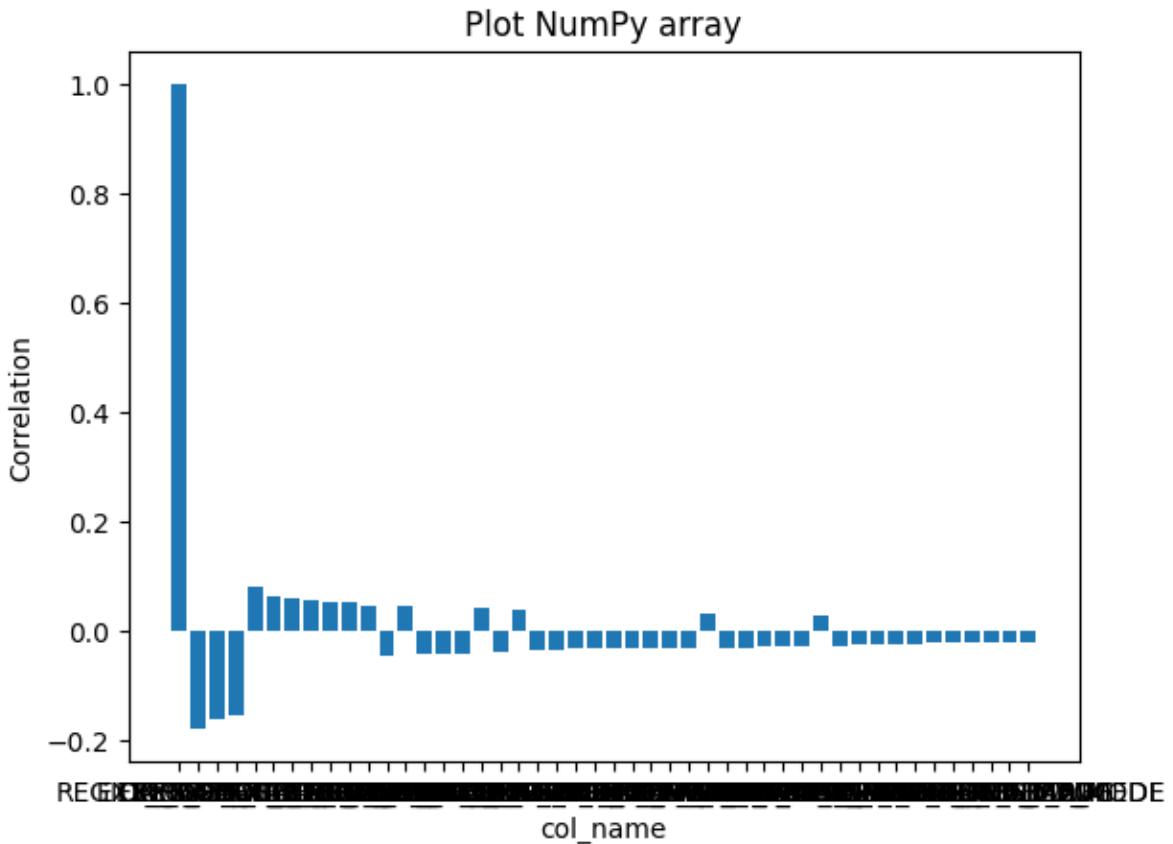
# Creating a new figure
plt.figure(dpi=100)
# Plotting Numpy array
plt.bar(New_DataFrame["col_name"], New_DataFrame["Correlation"])
```

```

# Adding details to the plot
plt.title('Plot NumPy array')
plt.xlabel('col_name')
plt.ylabel('Correlation')

# Displaying the plot
plt.show()

```



## Treating Missing Values

The numerical columns in the dataset New\_DataFrame\_Numerical are subjected to missing value imputation using the below code.

The percentage of missing values in each column of the input dataframe is determined by the helper\_function function. The dataframe and a threshold value are inputs for the Missing\_Values\_Above\_Threshold function, which outputs a dataframe with details about the columns with missing values over the threshold. The details include the name of the column, the number of unique values in the column, the median/mode value, the mean value, and the proportion of missing data.

The code then distinguishes between the columns that require imputation and the columns that contain no missing values. Additionally, it distinguishes the columns that have fewer than or equal to 10 distinct values since the mode of the column will be imputed to such columns.

The fillna method is used to impute the missing values for columns that have fewer than or equal to 10 distinct values utilizing the column's mode. The columns with imputed values and the columns with no missing values are then combined to form the final set of columns to be used in the analysis. New\_DataFrame\_Numerical\_Temp stores the initial dataframe, which is then changed to only contain the final set of columns in New\_DataFrame\_Numerical. The client's unique identifier is finally added to the dataframe at the beginning as a new column called "SK\_ID\_CURR."

```
In [ ]: def helper_function(DataFrame):
    length = len(DataFrame)
    temp4 = DataFrame.isnull()
    temp4= temp4.sum(axis=0)*100/length
    return temp4.sort_values(ascending=False)

def Missing_Values_Above_Threshold(DataFrame,threshold):
    temp5 = helper_function(DataFrame)
    temp5= temp5.reset_index()
    temp5.columns = ['index','flag']
    ref1 = []
    for i in temp5.itertuples():
        try:
            i1= DataFrame[i.index].median()
            i2= DataFrame[i.index].mean()
            i3= DataFrame[i.index].nunique()
            ref1.append([i.index,i.flag,i1,i2,i3])
        except:
            i4= DataFrame[i.index].mode()
            i5= DataFrame[i.index].nunique()
            ref1.append([i.index,i.flag,i4,'NA',i5])
    ref = ['col_name','percentage_missing','median/Mode','mean','no_of_unique']
    main_out = pd.DataFrame(ref1,columns=ref)
    return main_out[main_out['percentage_missing']>threshold]

New_DataFrame_Absent = Missing_Values_Above_Threshold(DF_Train[New_DataFrame_Absent])
temp1 = list(set(New_DataFrame.col_name).difference(set(New_DataFrame_Absent)))
temp2 = New_DataFrame_Absent[New_DataFrame_Absent['no_of_unique_values']<=10]
New_DataFrame_Absent = New_DataFrame_Absent[New_DataFrame_Absent['no_of_unique']

for i in temp2.col_name:
    temp6= New_DataFrame_Numerical[i]
    New_DataFrame_Numerical[i] = temp6.fillna(temp6.mode()[0])
for i in temp2.col_name:
    DF_Test[i] = DF_Test[i].fillna(DF_Test[i].mode()[0])

temp3 = list(temp2.col_name) + list(New_DataFrame_Absent.col_name) + temp1
New_DataFrame_Numerical_Temp = New_DataFrame_Numerical
New_DataFrame_Numerical = New_DataFrame_Numerical[temp3]
New_DataFrame_Numerical.insert(0,"SK_ID_CURR",New_DataFrame_Numerical_Temp['SK_ID_CURR'])
New_DataFrame_Numerical
```

```
Out[ ]:      SK_ID_CURR  LIVINGAPARTMENTS_MODE  LIVINGAPARTMENTS_MEDI  LIVINGAPART
0           100002          0.0220            0.0205
1           100003          0.0790            0.0787
2           100004             NaN            NaN
3           100006             NaN            NaN
4           100007             NaN            NaN
...
307506       456251          0.0882            0.1509
307507       456252          0.0220            0.0205
307508       456253          0.0918            0.0855
307509       456254             NaN            NaN
307510       456255             NaN            NaN
```

307511 rows × 47 columns

```
In [ ]: New_DataFrame_Cat_Temp = Missing_Values_Above_Threshold(New_DataFrame_Categorical)
New_DataFrame_Cat_Temp.describe()
```

```
Out[ ]:      percentage_missing  no_of_unique_values
count          6.000000          6.000000
mean          41.427841          6.833333
std           23.284402          5.845226
min           0.420148          2.000000
25%          35.358735          3.250000
50%          48.787198          5.500000
75%          50.674610          7.000000
max          68.386172          18.000000
```

```
In [ ]: Columns_to_Drop = []
Columns_to_Drop.append('FONDKAPREMONT_MODE')
Columns_to_Drop.append('WALLSMATERIAL_MODE')
Columns_to_Drop.append('OCCUPATION_TYPE')
New_DataFrame_Categorical.drop(columns = Columns_to_Drop, inplace=True)
```

```
In [ ]: from sklearn.compose import ColumnTransformer, make_column_transformer
from sklearn.model_selection import train_test_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer
```

```
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler
```

## Creating pipeline for numerical and categorical features

The below code establishes a pipeline that separates and then combines the categorical and numerical elements into a single data frame.

First, two pipelines are established separately for the category and numerical features. While the categorical pipeline replaces missing values with the most frequent value of each feature using SimpleImputer() and then applies one-hot encoding using OneHotEncoder, the numerical pipeline replaces missing values with the mean of each feature using SimpleImputer().

In order to specify which pipeline should be used for each set of columns, the two pipelines are then concatenated into a single ColumnTransformer() object.

The two pipelines are then applied to the corresponding sets of columns in DF\_Temp1 by calling the fit\_transform() method of the Data\_Pipeline object, which results in the creation of a single numpy array.

In the end, the OneHotEncoder() object's get\_feature\_names\_out() method is used to retrieve the feature names of the one-hot encoded categorical features. The resulting numpy array is then transformed into a pandas dataframe by combining the numerical and categorical feature names.

```
In [ ]: DF_Temp = list(New_DataFrame_Numerical.columns) + list(New_DataFrame_Categor
DF_Temp1 = DF_Train[DF_Temp]

Numerical_Pipeline = Pipeline(steps=[('imputer', SimpleImputer(strategy='mea
Categorical_Pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
])
Data_Pipeline = ColumnTransformer([
    ("num_pipeline", Numerical_Pipeline, New_DataFrame_Numerical.columns),
    ("cat_pipeline", Categorical_Pipeline, New_DataFrame_Categorical.columns)
])

model = Data_Pipeline.fit_transform(DF_Temp1)
```

## Categorical Features after one hot encoding

```
In [ ]: # Get feature names after one-hot encoding
ohe_feature_names = Data_Pipeline.named_transformers_['cat_pipeline'].named_
ohe_feature_names
```

```
Out[ ]: array(['NAME_CONTRACT_TYPE_Cash loans',
   'NAME_CONTRACT_TYPE_Revolving loans', 'CODE_GENDER_F',
   'CODE_GENDER_M', 'FLAG_OWN_CAR_N', 'FLAG_OWN_CAR_Y',
   'FLAG_OWN_REALTY_N', 'FLAG_OWN_REALTY_Y',
   'NAME_TYPE_SUITE_Children', 'NAME_TYPE_SUITE_Family',
   'NAME_TYPE_SUITE_Group of people', 'NAME_TYPE_SUITE_Other_A',
   'NAME_TYPE_SUITE_Other_B', 'NAME_TYPE_SUITE_Spouse, partner',
   'NAME_TYPE_SUITE_Unaccompanied', 'NAME_INCOME_TYPE_Businessman',
   'NAME_INCOME_TYPE_Commercial associate',
   'NAME_INCOME_TYPE_Pensioner', 'NAME_INCOME_TYPE_State servant',
   'NAME_INCOME_TYPE_Student', 'NAME_INCOME_TYPE_Unemployed',
   'NAME_INCOME_TYPE_Working', 'NAME_EDUCATION_TYPE_Academic degree',
   'NAME_EDUCATION_TYPE_Higher education',
   'NAME_EDUCATION_TYPE_Incomplete higher',
   'NAME_EDUCATION_TYPE_Lower secondary',
   'NAME_EDUCATION_TYPE_Secondary / secondary special',
   'NAME_FAMILY_STATUS_Civil marriage', 'NAME_FAMILY_STATUS_Married',
   'NAME_FAMILY_STATUS_Separated',
   'NAME_FAMILY_STATUS_Single / not married',
   'NAME_FAMILY_STATUS_Widow', 'NAME_HOUSING_TYPE_Co-op apartment',
   'NAME_HOUSING_TYPE_House / apartment',
   'NAME_HOUSING_TYPE_Municipal apartment',
   'NAME_HOUSING_TYPE_Office apartment',
   'NAME_HOUSING_TYPE_Rented apartment',
   'NAME_HOUSING_TYPE_With parents',
   'WEEKDAY_APPR_PROCESS_START_FRIDAY',
   'WEEKDAY_APPR_PROCESS_START_MONDAY',
   'WEEKDAY_APPR_PROCESS_START_SATURDAY',
   'WEEKDAY_APPR_PROCESS_START_SUNDAY',
   'WEEKDAY_APPR_PROCESS_START_THURSDAY',
   'WEEKDAY_APPR_PROCESS_START_TUESDAY',
   'WEEKDAY_APPR_PROCESS_START_WEDNESDAY',
   'ORGANIZATION_TYPE_Advertising', 'ORGANIZATION_TYPE_Agriculture',
   'ORGANIZATION_TYPE_Bank',
   'ORGANIZATION_TYPE_Business Entity Type 1',
   'ORGANIZATION_TYPE_Business Entity Type 2',
   'ORGANIZATION_TYPE_Business Entity Type 3',
   'ORGANIZATION_TYPE_Cleaning', 'ORGANIZATION_TYPE_Construction',
   'ORGANIZATION_TYPE_Culture', 'ORGANIZATION_TYPE_Electricity',
   'ORGANIZATION_TYPE_Emergency', 'ORGANIZATION_TYPE_Government',
   'ORGANIZATION_TYPE_Hotel', 'ORGANIZATION_TYPE_Housing',
   'ORGANIZATION_TYPE_Industry: type 1',
   'ORGANIZATION_TYPE_Industry: type 10',
   'ORGANIZATION_TYPE_Industry: type 11',
   'ORGANIZATION_TYPE_Industry: type 12',
   'ORGANIZATION_TYPE_Industry: type 13',
   'ORGANIZATION_TYPE_Industry: type 2',
   'ORGANIZATION_TYPE_Industry: type 3',
   'ORGANIZATION_TYPE_Industry: type 4',
   'ORGANIZATION_TYPE_Industry: type 5',
   'ORGANIZATION_TYPE_Industry: type 6',
   'ORGANIZATION_TYPE_Industry: type 7',
   'ORGANIZATION_TYPE_Industry: type 8',
   'ORGANIZATION_TYPE_Industry: type 9',
   'ORGANIZATION_TYPE_Insurance', 'ORGANIZATION_TYPE_Kindergarten',
   'ORGANIZATION_TYPE_Legal Services', 'ORGANIZATION_TYPE_Medicine',
```

```
'ORGANIZATION_TYPE_Military', 'ORGANIZATION_TYPE_Mobile',
'ORGANIZATION_TYPE_Other', 'ORGANIZATION_TYPE_Police',
'ORGANIZATION_TYPE_Postal', 'ORGANIZATION_TYPE_Realtor',
'ORGANIZATION_TYPE_Religion', 'ORGANIZATION_TYPE_Restaurant',
'ORGANIZATION_TYPE_School', 'ORGANIZATION_TYPE_Security',
'ORGANIZATION_TYPE_Security Ministries',
'ORGANIZATION_TYPE_Self-employed', 'ORGANIZATION_TYPE_Services',
'ORGANIZATION_TYPE_Telecom', 'ORGANIZATION_TYPE_Trade: type 1',
'ORGANIZATION_TYPE_Trade: type 2',
'ORGANIZATION_TYPE_Trade: type 3',
'ORGANIZATION_TYPE_Trade: type 4',
'ORGANIZATION_TYPE_Trade: type 5',
'ORGANIZATION_TYPE_Trade: type 6',
'ORGANIZATION_TYPE_Trade: type 7',
'ORGANIZATION_TYPE_Transport: type 1',
'ORGANIZATION_TYPE_Transport: type 2',
'ORGANIZATION_TYPE_Transport: type 3',
'ORGANIZATION_TYPE_Transport: type 4',
'ORGANIZATION_TYPE_University', 'ORGANIZATION_TYPE_XNA',
'HOUSETYPE_MODE_block of flats', 'HOUSETYPE_MODE_specific housing',
'HOUSETYPE_MODE_terraced house', 'EMERGENCYSTATE_MODE_No',
'EMERGENCYSTATE_MODE_Yes'], dtype=object)
```

```
In [ ]: c = list(New_DataFrame_Numerical.columns) + list(ohe_feature_names)
DF_Temp2 = pd.DataFrame(model, columns=c)
```

```
In [ ]: DF_Temp2
```

```
Out[ ]:
```

	SK_ID_CURR	LIVINGAPARTMENTS_MODE	LIVINGAPARTMENTS_MEDI	LIVINGAPART
0	100002.0	0.022000	0.020500	
1	100003.0	0.079000	0.078700	
2	100004.0	0.105646	0.101956	
3	100006.0	0.105646	0.101956	
4	100007.0	0.105646	0.101956	
...	...	...	...	
307495	456251.0	0.088200	0.150900	
307496	456252.0	0.022000	0.020500	
307497	456253.0	0.091800	0.085500	
307498	456254.0	0.105646	0.101956	
307499	456255.0	0.105646	0.101956	

307500 rows × 155 columns

```
In [ ]: DF_Temp3 = DF_Temp2
DF_Temp2 = pd.merge(left=DF_Temp2, right=DF_InstallmentsPayemnts, how='left')
```

```
DF_Temp2 = pd.merge(left=DF_Temp2, right=DF_PosCashBalance, how='left', left_on='SK_ID_CURR', right_on='SK_ID_CURR')
DF_Temp2 = pd.merge(left=DF_Temp2, right=DF_PreviousApplication, how='left', left_on='SK_ID_CURR', right_on='SK_ID_CURR')
DF_Temp2 = pd.merge(left=DF_Temp2, right=DF_CreditCardBalance, how='left', left_on='SK_ID_CURR', right_on='SK_ID_CURR')
DF_Temp2 = pd.merge(left=DF_Temp2, right=Updated_BureauBalance, how='left', left_on='SK_ID_CURR', right_on='SK_ID_CURR')
DF_Final = DF_Temp2
DF_Final
```

```
<ipython-input-25-00707a56931c>:5: FutureWarning: Passing 'suffixes' which
cause duplicate columns {'SK_ID_PREV_x'} in the result is deprecated and wi
ll raise a MergeError in a future version.
```

```
    DF_Temp2 = pd.merge(left=DF_Temp2, right=DF_CreditCardBalance, how='lef
t', left_on='SK_ID_CURR', right_on='SK_ID_CURR')
```

```
Out[ ]:   SK_ID_CURR  LIVINGAPARTMENTS_MODE  LIVINGAPARTMENTS_MEDI  LIVINGAPART
          0      100002.0           0.022000           0.020500
          1      100003.0           0.079000           0.078700
          2      100004.0           0.105646           0.101956
          3      100006.0           0.105646           0.101956
          4      100007.0           0.105646           0.101956
          ...
          ...
          307495  456251.0           0.088200           0.150900
          307496  456252.0           0.022000           0.020500
          307497  456253.0           0.091800           0.085500
          307498  456254.0           0.105646           0.101956
          307499  456255.0           0.105646           0.101956
```

307500 rows × 215 columns

---

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

Out [ ]:

	SK_ID_CURR	LIVINGAPARTMENTS_MODE	LIVINGAPARTMENTS_MEDI	LIVINGAPART
<b>count</b>	307500.000000	307500.000000	307500.000000	30
<b>mean</b>	278181.087798	0.105646	0.101956	
<b>std</b>	102789.822017	0.055062	0.052678	
<b>min</b>	100002.000000	0.000000	0.000000	
<b>25%</b>	189146.750000	0.105646	0.101956	
<b>50%</b>	278202.500000	0.105646	0.101956	
<b>75%</b>	367143.250000	0.105646	0.101956	
<b>max</b>	456255.000000	1.000000	1.000000	

8 rows × 215 columns

## Adding New Features

The code below creates 13 new features (Feature\_1 to Feature\_13) utilizing different columns from the consolidated dataframe after merging numerous dataframes into one (DF\_Final). Mathematical procedures such as division, multiplication, and addition are used to calculate these additional properties. The final dataframe's shape is subsequently determined by the algorithm and assigned to the variable DF\_Final.

In order to calculate the correlation between the newly produced features and the goal variable "TARGET," the final line of code invokes the function

Correlation\_Between\_InputDF\_Target. The function generates a correlation matrix after calculating the Pearson correlation coefficient between each feature in the input and the desired variable. To only include features with a correlation coefficient greater than or equal to 0.0, a threshold value of 0.0 is crossed. Finding the newly developed features that have the highest correlation with the objective variable and may one day serve as predictors in a machine learning model is the goal of this stage.

```
In [ ]: DF_Final['Feature_1']= DF_Final['AMT_CREDIT_SUM']/(DF_Final['DAYS_EMPLOYED'])
DF_Final['Feature_2']= DF_Final['CNT_DRAWINGS_POS_CURRENT']/(DF_Final['FLOOR'])
DF_Final['Feature_3']= DF_Final['AMT_CREDIT_SUM']/(DF_Final['FLOORSMIN_AVG'])
DF_Final['Feature_4']=DF_Final['DAYS_CREDIT']*(DF_Final['DAYS_ENDDATE_FACT'])
DF_Final['Feature_5']=DF_Final['AMT_CREDIT_SUM']/(DF_Final['AMT_CREDIT_SUM'])
DF_Final['Feature_6']=DF_Final['AMT_CREDIT_x']/(DF_Final['AMT_CREDIT_SUM']+1)
DF_Final['Feature_7']=(DF_Final['DAYS_CREDIT']+DF_Final['AMT_CREDIT_SUM'])
DF_Final['Feature_8']=(DF_Final['AMT_CREDIT_x']+DF_Final['FLOORSMIN_AVG'])
DF_Final['Feature_9']= DF_Final['AMT_CREDIT_SUM']/(DF_Final['AMT_GOODS_PRI'])
DF_Final['Feature_10']= DF_Final['AMT_CREDIT_SUM']/(DF_Final['AMT_GOODS_PRI'])
DF_Final['Feature_11']= DF_Final['DAYS_BIRTH']+(DF_Final['DAYS_EMPLOYED']+1)
DF_Final['Feature_12']= (DF_Final['EXT_SOURCE_1']+DF_Final['EXT_SOURCE_2']+1)
```

```
DF_Final['Feature_13']= DF_Final['CNT_INSTALMENT_MATURE_CUM']*(DF_Final['CNT_INSTALMENT_FUTURE']-DF_Final['CNT_PAYMENT'])
DF_Final.shape
```

```
Out[ ]: (307500, 228)
```

## Engineering RFM Features

```
In [ ]: import pandas as pd
import numpy as np

# Load the HCDR dataset and select relevant columns
hcdr_df = pd.read_csv("application_train.csv")
hcdr_df = hcdr_df[['SK_ID_CURR', 'AMT_CREDIT', 'AMT_ANNUITY']]

# Calculate monetary value as the average of AMT_CREDIT and AMT_ANNUITY
hcdr_df['MonetaryValue'] = hcdr_df[['AMT_CREDIT', 'AMT_ANNUITY']].mean(axis=1)

# Load the previous application dataset and select relevant columns
prev_df = pd.read_csv("previous_application.csv")
prev_df = prev_df[['SK_ID_CURR', 'DAYS_DECISION']]

# Calculate recency as the number of days since the latest previous application
prev_df = prev_df.groupby('SK_ID_CURR').max().reset_index()
prev_df['Recency'] = prev_df['DAYS_DECISION'] * -1

# Merge hcdr_df and prev_df on SK_ID_CURR
hcdr_rfm = pd.merge(hcdr_df, prev_df[['SK_ID_CURR', 'Recency']], on='SK_ID_CURR')

# Calculate frequency as the number of previous applications
hcdr_rfm['Frequency'] = prev_df.groupby('SK_ID_CURR').size().reset_index(name='Frequency')[['SK_ID_CURR']]

# Create labels for RFM score
quantiles = hcdr_rfm.quantile(q=[0.2, 0.4, 0.6, 0.8])
quantiles = quantiles.to_dict()

def r_score(x, c):
    if x <= c[0.2]:
        return 4
    elif x <= c[0.4]:
        return 3
    elif x <= c[0.6]:
        return 2
    elif x <= c[0.8]:
        return 1
    else:
        return 0

def fm_score(x, c):
    if x <= c[0.2]:
        return 0
    elif x <= c[0.4]:
        return 1
    elif x <= c[0.6]:
        return 2
    else:
        return 3
```

```

    elif x <= c[0.8]:
        return 3
    else:
        return 4

hcdr_rfm['R'] = hcdr_rfm['Recency'].apply(r_score, args=(quantiles['Recency'],
hcdr_rfm['F'] = hcdr_rfm['Frequency'].apply(fm_score, args=(quantiles['Frequency'],
hcdr_rfm['M'] = hcdr_rfm['MonetaryValue'].apply(fm_score, args=(quantiles['MonetaryValue'],

# Calculate RFM score
hcdr_rfm['RFM'] = hcdr_rfm['R']*100 + hcdr_rfm['F']*10 + hcdr_rfm['M']

# Display the first 10 rows of hcdr_rfm
hcdr_rfm.head(10)

```

Out[ ]:

	SK_ID_CURR	AMT_CREDIT	AMT_ANNUITY	MonetaryValue	Recency	Frequency	R	F	M
0	100002	406597.5	24700.5	215649.00	NaN		1.0	0	0
1	100003	1293502.5	35698.5	664600.50	746.0		1.0	1	0
2	100004	1350000.0	6750.0	70875.00	NaN		1.0	0	0
3	100006	312682.5	29686.5	171184.50	181.0		1.0	3	0
4	100007	513000.0	21865.5	267432.75	374.0		1.0	2	0
5	100008	490495.5	27517.5	259006.50	82.0		1.0	4	0
6	100009	1560726.0	41301.0	801013.50	449.0		1.0	2	0
7	100010	1530000.0	42075.0	786037.50	NaN		1.0	0	0
8	100011	1019610.0	33826.5	526718.25	1162.0		1.0	0	0
9	100012	405000.0	20250.0	212625.00	832.0		1.0	1	0

In [ ]: DF\_Final.shape

Out[ ]: (307500, 228)

In [ ]: DF\_Final['Frequency']= hcdr\_rfm["Frequency"]
DF\_Final['RFM']= hcdr\_rfm["RFM"]
DF\_Final['R']= hcdr\_rfm["R"]
DF\_Final['F']= hcdr\_rfm["F"]
DF\_Final['M']= hcdr\_rfm["M"]
DF\_Final.shape

Out[ ]: (307500, 233)

## Correlation of New Features and the target variable'

In [ ]: Output\_Correlation=Correlation\_Between\_InputDF\_Target(DF\_Final[['Feature\_1'],
Output\_Correlation

Out [ ]:

	col_name	Correlation
0	TARGET	1.000000
1	Feature_12	-0.221463
2	Feature_4	-0.054394
3	Feature_13	-0.050172
4	Feature_11	-0.043322
5	Feature_2	0.030728
6	Feature_8	-0.030390
7	Feature_7	-0.015612
8	Feature_3	-0.015356
9	Feature_9	-0.011549
10	Feature_10	-0.011549
11	Feature_1	-0.005593
12	Feature_5	-0.002558
13	Feature_6	0.000649

In [ ]:

```
DF_Final = DF_Final.rename(columns = {'AMT_CREDIT_y':'AMT_CREDIT'})
DF_Final = DF_Final.apply(lambda x: x.fillna(x.median()),axis=0)
filter1 = Correlation_Between_InputDF_Target(DF_Final,0.05)
filter1
```

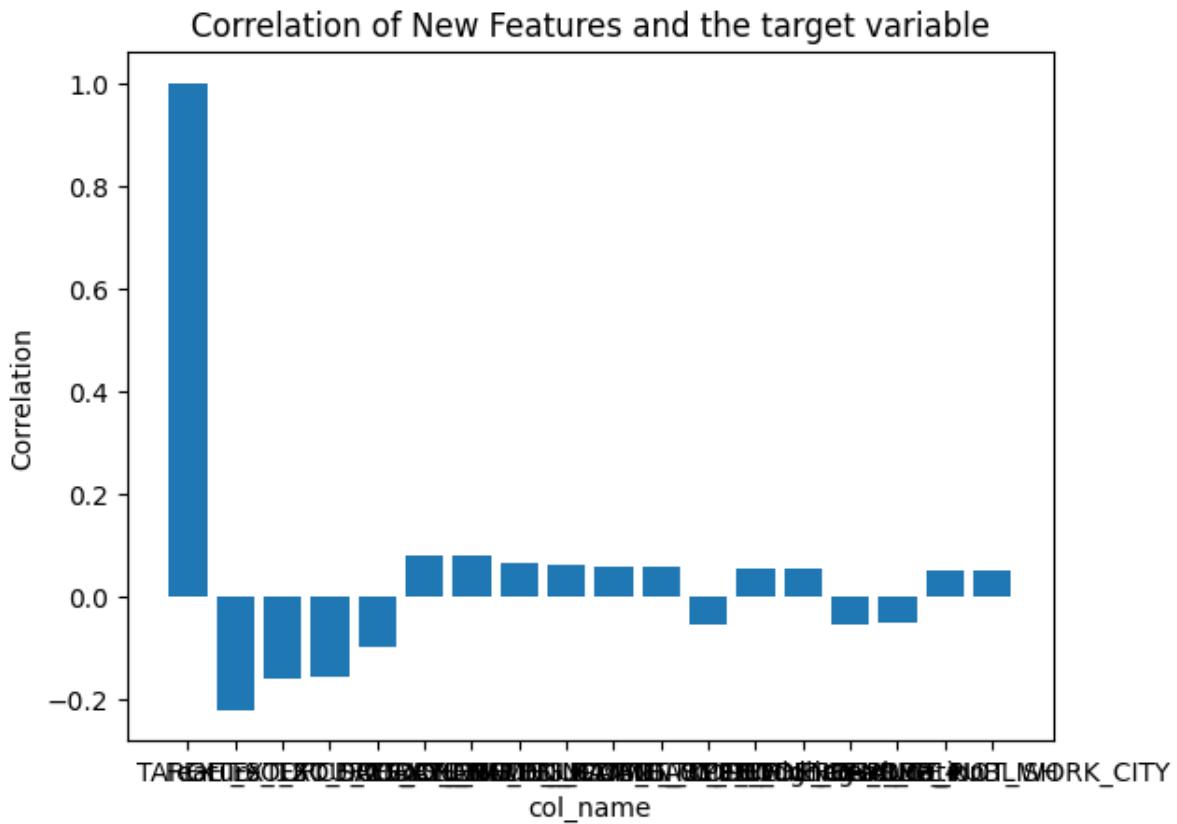
Out[ ]:

	col_name	Correlation
0	TARGET	1.000000
1	Feature_12	-0.221463
2	EXT_SOURCE_2	-0.160283
3	EXT_SOURCE_3	-0.157409
4	EXT_SOURCE_1	-0.099162
5	DAYS_CREDIT	0.079099
6	DAYS_BIRTH	0.078236
7	DAYS_CREDIT_UPDATE	0.063527
8	REGION_RATING_CLIENT_W_CITY	0.060875
9	REGION_RATING_CLIENT	0.058882
10	NAME_INCOME_TYPE_Working	0.057504
11	NAME_EDUCATION_TYPE_Higher education	-0.056578
12	DAYS_LAST_PHONE_CHANGE	0.055228
13	CODE_GENDER_M	0.054729
14	CODE_GENDER_F	-0.054729
15	Feature_4	-0.052507
16	DAYS_ID_PUBLISH	0.051455
17	REG_CITY_NOT_WORK_CITY	0.050981

In [ ]:

```
import matplotlib.pyplot as plt
import numpy as np

# Creating a new figure
plt.figure(dpi=100)
# Plotting Numpy array
plt.bar(filter1["col_name"],filter1["Correlation"])
# Adding details to the plot
plt.title('Correlation of New Features and the target variable')
plt.xlabel('col_name')
plt.ylabel('Correlation')
# Displaying the plot
plt.show()
```



```
In [ ]: filter1 = filter1['col_name'][1:]
filter1
```

```
Out[ ]: 1           Feature_12
2           EXT_SOURCE_2
3           EXT_SOURCE_3
4           EXT_SOURCE_1
5           DAYS_CREDIT
6           DAYS_BIRTH
7           DAYS_CREDIT_UPDATE
8           REGION_RATING_CLIENT_W_CITY
9           REGION_RATING_CLIENT
10          NAME_INCOME_TYPE_Working
11          NAME_EDUCATION_TYPE_Higher education
12          DAYS_LAST_PHONE_CHANGE
13          CODE_GENDER_M
14          CODE_GENDER_F
15          Feature_4
16          DAYS_ID_PUBLISH
17          REG_CITY_NOT_WORK_CITY
Name: col_name, dtype: object
```

```
In [ ]: filter1.describe()
```

```
Out[ ]: count      17
unique     17
top       Feature_12
freq      1
Name: col_name, dtype: object
```

```
In [ ]: T = DF_Final['TARGET']
DF_Temp4 = DF_Final[filter1]
X = DF_Temp4.values
y = T.values
X
```

```
Out[ ]: array([[ 4.85361340e-01,  2.62948593e-01,  1.39375780e-01, ...,
   9.77865000e+05, -2.12000000e+03,  0.00000000e+00],
   [ 1.44436873e+00,  6.22245775e-01,  5.10855639e-01, ...,
   7.47410000e+05, -2.91000000e+02,  0.00000000e+00],
   [ 1.78760234e+00,  5.55912083e-01,  7.29566691e-01, ...,
   4.60810500e+05, -2.53100000e+03,  0.00000000e+00],
   ...,
   [ 1.49860723e+00,  5.35721752e-01,  2.18859082e-01, ...,
   7.28767000e+05, -5.15000000e+03,  1.00000000e+00],
   [ 1.67730993e+00,  5.14162820e-01,  6.61023539e-01, ...,
   9.47232000e+05, -9.31000000e+02,  1.00000000e+00],
   [ 1.55695096e+00,  7.08568896e-01,  1.13922396e-01, ...,
   8.85870000e+05, -4.10000000e+02,  1.00000000e+00]])
```

## Experiment 2: Using Selected Features (Using Entire Dataset)

```
In [ ]: from sklearn.metrics import log_loss
from sklearn.metrics import accuracy_score
from sklearn.metrics import confusion_matrix
from sklearn.metrics import roc_auc_score
from sklearn.naive_bayes import GaussianNB
from sklearn.ensemble import RandomForestClassifier
import matplotlib.pyplot as plt
from sklearn.pipeline import make_pipeline
from sklearn.model_selection import GridSearchCV
from sklearn import metrics
import warnings
warnings.filterwarnings("ignore", category=DeprecationWarning)
```

```
In [ ]: df_app=pd.DataFrame(datasets["application_train"])
df_bureau= pd.DataFrame(datasets["bureau"])
df_bureau_balance= pd.DataFrame(datasets["bureau_balance"])
df_prev_app = pd.DataFrame(datasets["previous_application"])
df_pos = pd.DataFrame(datasets["POS_CASH_balance"])
df_inst = pd.DataFrame(datasets["installments_payments"])
df_credit_card_balance=pd.DataFrame(datasets["credit_card_balance"])
```

```
In [ ]: df_tiny = df_app.copy()
```

```
In [ ]: df_tiny.shape
```

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

```
In [ ]: print(df_bureau.shape)
print(df_bureau_balance.shape)

(1716428, 17)
(27299925, 3)
```

```
In [ ]: df_bureau = df_bureau.groupby("SK_ID_BUREAU").mean().reset_index()
df_bureau_balance = df_bureau_balance.groupby("SK_ID_BUREAU").mean().reset_i
```

```
In [ ]: df_bureau = df_bureau.merge(df_bureau_balance.reset_index(), how='left', on="SK_ID_BUREAU")
df_bureau = df_bureau.groupby("SK_ID_CURR").mean().reset_index()
df_bureau.shape
```

```
Out[ ]: (305811, 16)
```

```
In [ ]: df_tiny = df_tiny.merge(df_bureau.reset_index(), how="left", on="SK_ID_CURR")
df_tiny.shape
```

```
Out[ ]: (307511, 138)
```

```
In [ ]: df_prev_app = df_prev_app.groupby("SK_ID_CURR").mean().reset_index()
df_tiny = df_tiny.merge(df_prev_app.reset_index(), how="left", on="SK_ID_CURR")
df_tiny.shape
```

```
Out[ ]: (307511, 159)
```

```
In [ ]: df_pos = df_pos.groupby("SK_ID_CURR").mean().reset_index()
df_tiny = df_tiny.merge(df_pos.reset_index(), how="left", on="SK_ID_CURR", s
df_tiny.shape
```

```
Out[ ]: (307511, 166)
```

```
In [ ]: df_inst = df_inst.groupby("SK_ID_CURR").mean().reset_index()
df_tiny = df_tiny.merge(df_inst.reset_index(), how="left", on="SK_ID_CURR",
df_tiny.shape
```

```
Out[ ]: (307511, 174)
```

```
In [ ]: df_credit_card_balance = df_credit_card_balance.groupby("SK_ID_CURR").mean()
df_tiny = df_tiny.merge(df_credit_card_balance.reset_index(), how="left", or
df_tiny.shape
```

```
Out[ ]: (307511, 196)
```

```
In [ ]: y = df_tiny["TARGET"]
x = df_tiny.drop("TARGET", axis=1)
x_train, x_valid, y_train, y_valid = train_test_split(x, y, test_size=0.15,
x_train, x_test, y_train, y_test = train_test_split(x_train, y_train, test_s
```

## Selecting Features

After performing the EDA, we have got some important columns that has the highest affect on the Target variable. It means that these columns could predict the outcome

instead of using all the columns.

**EXT\_SOURCE\_1**, **EXT\_SOURCE\_2**, and **EXT\_SOURCE\_3** : These columns are moderately correlated with the target variable and also moderately correlated with each other, so they may be good features to include in a model.

**DAYS\_BIRTH** : This column is moderately negatively correlated with the target variable, indicating that younger applicants are more likely to default on their loans.

**DAYS\_EMPLOYED** : This column has a weak negative correlation with the target variable, indicating that longer-term employment may be associated with a lower risk of default.

**AMT\_CREDIT** : This column has a moderate positive correlation with the target variable, indicating that higher loan amounts may be associated with a higher risk of default.

**AMT\_GOODS\_PRICE** : This column has a moderate positive correlation with the target variable, similar to AMT\_CREDIT.

The following columns have also been selected to train the model because after merging the 'TARGET' column from application\_train.csv to different dataset files such as instalment\_payments, previous\_application, pos\_cash\_balance, bureau and bureau\_balance, and plotting the correlation matrix we inferred that these columns had correlation more than 0.3 with target column: **DAYS\_CREDIT**, **DAYS\_CREDIT\_UPDATE**, **DAYS\_ENDDATE\_FACT**, **DAYS\_ENTRY\_PAYMENT**, **DAYS\_INSTALMENT**, **DAYS\_DECISION**, **DAYS\_FIRST\_DRAWING** and **CNT\_PAYMENT**.

```
In [ ]: # extracting the cat and num features
categorical_feat = ['NAME_CONTRACT_TYPE', 'NAME_FAMILY_STATUS', 'OCCUPATION_TYPE']
numerical_features= ['AMT_INCOME_TOTAL', 'AMT_CREDIT', 'DAYS_EMPLOYED', 'DAYS_CREDIT',
'EXT_SOURCE_2', 'EXT_SOURCE_3', 'DAYS_CREDIT', 'DAYS_CREDIT_UPDATE', 'DAYS_E']
```

```
In [ ]: x_train[categorical_feat].nunique().sort_values()
```

```
Out[ ]: NAME_CONTRACT_TYPE      2
NAME_FAMILY_STATUS       6
OCCUPATION_TYPE        18
ORGANIZATION_TYPE      58
dtype: int64
```

```
In [ ]: print(len(categorical_feat), categorical_feat)
print(len(numerical_features), numerical_features)
```

```
4 ['NAME_CONTRACT_TYPE', 'NAME_FAMILY_STATUS', 'OCCUPATION_TYPE', 'ORGANIZATION_TYPE']
15 ['AMT_INCOME_TOTAL', 'AMT_CREDIT', 'DAYS_EMPLOYED', 'DAYS_BIRTH', 'EXT_SOURCE_1',
'EXT_SOURCE_2', 'EXT_SOURCE_3', 'DAYS_CREDIT', 'DAYS_CREDIT_UPDATE',
'DAYS_ENDDATE_FACT', 'DAYS_DECISION', 'DAYS_ENTRY_PAYMENT', 'DAYS_INSTALMENT',
'DAYS_FIRST_DRAWING', 'CNT_PAYMENT']
```

```
In [ ]: exp_log_full = pd.DataFrame(columns=["Pipeline", "Parameters", "TrainAcc", "
```

## Pipeline

- This code establishes a preparation pipeline for the HCDR dataset using scikit-learn's ColumnTransformer. Both category and numerical features are likely included in the dataset, thus the preparation pipeline must treat them separately.
- To change categorical and numerical information, the algorithm first generates two distinct pipelines. The pipeline applies one-hot encoding to categorical features before using SimpleImputer to fill in missing values with the value that occurs the most frequently. The pipeline uses StandardScaler to scale the data for numerical characteristics and SimpleImputer to replace missing values with the median value.
- The appropriate pipeline is then applied to each type of feature using the ColumnTransformer that was established before. Categorical\_trans pipeline is applied to categorical characteristics by the "cat" transformer, whereas the numerical\_trans pipeline is applied to numerical features by the "num" transformer.
- The ColumnTransformer object is added to the parameter of the transformers to establish the preprocessing pipeline. The data can then be preprocessed using the resulting pp\_pipe object before being fitted to a machine-learning model.
- Overall, by applying the proper transformations to each type of feature, this code is building a preprocessing pipeline that can handle both categorical and numerical characteristics in the HCDR dataset. The pipeline can assist in making sure that the data is appropriately prepared for modeling and can increase the precision of machine learning models developed using the dataset.

```
In [ ]: from sklearn.compose import ColumnTransformer
# Categorical features are transformed
categorical_trans = Pipeline(
    steps=[
        ('ohe', OneHotEncoder(handle_unknown='ignore')),
        ('imputer', SimpleImputer(strategy='most_frequent'))
    ]
)
# Numerical features are transformed
numerical_trans = Pipeline(
    steps=[
        ('scaler', StandardScaler()),
        ('imputer', SimpleImputer(strategy='median'))
    ]
)
# Creating a preprocessing Pipeline
pp_pipe = ColumnTransformer(
    transformers=[
        ('cat', categorical_trans, categorical_feat),
        ('num', numerical_trans, numerical_features)
```

```
    ]  
)
```

```
In [ ]: x_train = pp_pipe.fit_transform(x_train)  
x_valid = pp_pipe.transform(x_valid)  
x_test = pp_pipe.transform(x_test)
```

```
In [ ]: print(x_train.shape)  
print(y_train.shape)  
(222176, 100)  
(222176,)
```

## Models

We implemented the following models to find which one gives the best performance:

1. Logistic Regression: A common model for situations involving binary classification is logistic regression. It operates by utilizing a logistic function to estimate the likelihood of the target variable (default) with respect to the input parameters (such as age, income, and loan amount). Based on the applicant's demographic and financial data, logistic regression can be applied in the HCDR dataset to forecast the likelihood of default.
2. Random Forest Classifier: Several Decision Trees are used in an ensemble model called Random Forest to lessen overfitting and boost accuracy. It operates by randomly choosing a subset of features at each split while training a series of Decision Trees on bootstrapped subsets of the data.
3. Decision Tree Classifier: A decision tree model divides the data recursively into subsets depending on the most significant attributes until a stopping requirement is satisfied. As a result, a tree-like model is produced, with each internal node standing in for a judgment call based on a feature and each leaf node for a prediction.
4. Adaboost Classifier: A popular machine learning approach for categorization issues is called Adaboost (Adaptive Boosting). It functions by fusing weak classifiers to produce a powerful classifier.

```
In [ ]: import time  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.svm import SVC  
from sklearn.tree import DecisionTreeClassifier  
from warnings import simplefilter  
from sklearn.exceptions import ConvergenceWarning  
simplefilter("ignore", category=ConvergenceWarning)  
from sklearn.ensemble import AdaBoostClassifier  
from sklearn.pipeline import Pipeline  
from sklearn.preprocessing import StandardScaler
```

## Logistic\_Regression

- This code sets up a hyperparameter grid to be searched through during model selection and creates a scikit-learn pipeline for a Logistic Regression model.
- The LogisticRegression() object wrapped in an estimator with the identifier "lr" is the only step used to generate the Pipeline object. The LogisticRegression() object's hyperparameters are specified in the hyperparameter grid using the term 'lr'.
- The Logistic\_Regressionparams dictionary is used to specify the hyperparameter grid. The Logistic Regression model's "C" hyperparameter, which determines the regularization strength, is set to the value 0.01. The "penalty" hyperparameter, which has the values "l1" and "l2" in a list, regulates the kind of regularization the model employs. These variables indicate whether L1 regularization or L2 regularization should be used to fit the model.
- GridSearchCV or another hyperparameter tuning method can use the hyperparameter grid to find the ideal set of hyperparameters for the Logistic Regression model. The L1 and L2 regularization may be compared during the search thanks to this hyperparameter grid configuration, and the value of the 'C' hyperparameter regulates the regularization's strength.
- In general, this code creates a pipeline and hyperparameter grid that can be used for model selection, hyperparameter tuning, and a Logistic Regression model on the HCDR dataset.

```
In [ ]: Logistic_Regression = Pipeline(  
    [('lr', LogisticRegression())]  
)  
Logistic_Regressionparams = {'lr_C' : [0.01],  
                           'lr_penalty' : ['l1', 'l2']}  
}
```

## Decision Tree

- In addition to putting up a hyperparameter grid to be searched through during model selection, the below code creates a scikit-learn Pipeline for a Decision Tree model.
- The DecisionTreeClassifier() object wrapped in an estimator with the name "dt" is the only step required to generate the Pipeline object. The DecisionTreeClassifier() object's hyperparameters are specified in the hyperparameter grid using the term 'dt'.
- The hyperparameter grid is specified as a dictionary named Decision\_Treeparams. The maximum depth of the Decision Tree is controlled by the 'max\_depth' hyperparameter, which has the values 5 and 10 in its list of possible values. These numbers indicate that a maximum depth of 5 or 10 should be used to fit the model.
- The maximum depth hyperparameter for the Decision Tree model can be tuned using GridSearchCV or any hyperparameter tuning method with the help of the hyperparameter grid. The code enables for the performance of the Decision Tree

model to be assessed at various maximum depths and regulates the model's complexity through the maximum depth by adjusting the hyperparameter grid in this manner.

- For a Decision Tree model on the HCDR dataset, this code creates a Pipeline and hyperparameter grid that can be used for hyperparameter tuning and model selection.

```
In [ ]: Decision_Tree = Pipeline(  
    [('dt', DecisionTreeClassifier())]  
)  
Decision_Treeparams = {'dt__max_depth' : [5, 10]}
```

## AdaBoost classifier

- The Pipeline function of Scikit-Learn is used in the code to build an AdaBoost classifier. An ensemble classifier known as the AdaBoostClassifier combines a number of "weak" classifiers to produce a single, stronger classifier. In this instance, a decision tree classifier is the basic estimator.
- Hyperparameters that are provided to the AdaBoostClassifier during training are contained in the AdaBoost\_Classifierparams dictionary. Scikit-learn will try all possible combinations of the hyperparameters to identify the combination that yields the best performance on the dataset. These hyperparameters are supplied in a grid-search format.
- The hyperparameters in this instance are configured as follows:

ada\_\_n\_estimators: The number of weak classifiers to be included in the ensemble is specified by this hyperparameter. The value is set to 20 in this instance.

ada\_\_learning\_rate: This parameter regulates how much each weak classifier contributes to the final ensemble. Each weak classifier has a bigger effect on the result when the learning rate is increased. In this instance, 0.01 and 0.1 learning rates are attempted. ada\_\_base\_estimator\_\_max\_depth: The base estimator's decision tree's maximum depth is controlled by this hyperparameter. A decision tree that is less complicated and not as prone to overfit the training set has less depth. In this instance, the maximum depths of 1 and 5 are tested.

- Scikit-learn will fit the AdaBoost\_Classifier pipeline using the chosen parameters on the training data. The final model is going to be used to forecast the probability of loan default on a holdout set, and the algorithm's performance may be assessed using measures.

```
In [ ]: AdaBoost_Classifier = Pipeline([ ('ada', AdaBoostClassifier(base_estimator=  
AdaBoost_Classifierparams = {  
    'ada__n_estimators': [20],  
    'ada__learning_rate': [0.01, 0.1],  
    'ada__base_estimator__max_depth': [1,5]  
}])
```

## Random Forest

- This code sets up a hyperparameter grid to be searched through during model selection and creates a scikit-learn Pipeline for a Random Forest model.
- The RandomForestClassifier() object is wrapped in an estimator with the name "rf" to generate the Pipeline object in a single step. The RandomForestClassifier() object's hyperparameters are specified in the hyperparameter grid using the name 'rf'.
- The Random\_Forestparam dictionary is the one used to specify the hyperparameter grid. The 'n\_estimators' hyperparameter, which is specified as 20, regulates the number of decision trees in the Random Forest model. These numbers indicate that 20 decision trees should be used to fit the model. Each decision tree's "criterion" hyperparameter, which is set to the values "gini" and "log\_loss," regulates the quality of the split. These numbers indicate that the decision trees' node splitting criteria should be either Gini impurity or cross-entropy in order to fit the model.
- The hyperparameter grid is made to make it possible for GridSearchCV or any hyperparameter tuning technique to look for the ideal set of hyperparameters for the Random Forest model. The code allows for the comparison of the number of decision trees and the quality of the splits in the decision trees throughout the search and manages the complexity and performance of the model using these hyperparameters by configuring the hyperparameter grid in this manner.
- Overall, this code creates a Pipeline and hyperparameter grid for a Random Forest model on the HCDR dataset that can be used for hyperparameter tuning and model selection.

```
In [ ]: Random_Forest = Pipeline([('rf', RandomForestClassifier())])
Random_Forestparam = {
    'rf__n_estimators' : [20],
    'rf__criterion' : ['gini','log_loss']
}
```

Four dictionaries that each represent a machine learning algorithm make up the "main" list:

- Random\_Forest
- Decision\_Tree
- Logistic\_Regression
- AdaBoost\_Classifier

There is a corresponding dictionary of hyperparameters for each algorithm:

- Random\_Forestparam
- Decision\_Treeparams
- Logistic\_Regressionparams
- AdaBoost\_Classifierparams

This code specifies a collection of machine learning models that will be tested on the HCDR dataset. Both a "pipeline" key and a "params" key are present in each dictionary in the "main" list. The machine learning algorithm to be employed is specified using the "pipeline" key, and its hyperparameters are specified using the "params" key.

The optimal model for the given dataset can be found by training and analyzing the models after they have been defined.

```
In [ ]: main = [
    {
        'pipeline' : Random_Forest,
        "params" : Random_Forestparam
    },
    {
        'pipeline' : Decision_Tree,
        "params" : Decision_Treeparams
    },
    {
        'pipeline' : Logistic_Regression,
        "params" : Logistic_Regressionparams
    },
    {
        'pipeline' : AdaBoost_Classifier,
        "params" : AdaBoost_Classifierparams
    }
]
```

```
In [ ]: from sklearn.metrics import roc_auc_score

dict1 = {}
for i in main:
    print(f"Running {i['pipeline']}")
    GS_CV = GridSearchCV(i["pipeline"], i["params"], n_jobs=-1, cv=2, verbose=0)
    GS_CV.fit(x_train, y_train)
    p = GS_CV.best_estimator_
    dict1[i['pipeline']] = GS_CV
    T = time.time()
    Tr_acc = p.score(x_train, y_train)
    Tr_T = time.time() - T
    Va_acc = p.score(x_valid, y_valid)
    T = time.time()
    Te_acc = p.score(x_test, y_test)
    Te_T = time.time() - T
    AUC_Train=roc_auc_score(y_train, dict1[i['pipeline']].predict_proba(x_train))
    AUC_Test=roc_auc_score(y_test, dict1[i['pipeline']].predict_proba(x_test))
    AUC_val=roc_auc_score(y_valid, dict1[i['pipeline']].predict_proba(x_valid))
    exp_log_full.loc[len(exp_log_full)] =[f"Baseline {i['pipeline']} with {x_train.shape[1]} features", f"Tr acc: {Tr_acc*100:8.2f}%", f"Va acc: {Va_acc*100:8.2f}%", f"Te acc: {Te_acc*100:8.2f}%", f"AUC Train: {AUC_Train*100:8.2f}%", f"AUC Test: {AUC_Test*100:8.2f}%", f"AUC val: {AUC_val*100:8.2f}%", Tr_T, Te_T]
    print(f"Completed {i['pipeline']}")
```

```
print("ROC AUC score")
```



```
Completed Pipeline(steps=[('ada',
                           AdaBoostClassifier(base_estimator=DecisionTreeClassifier
                           ())])
ROC AUC score
Logistic Regression : 0.7362152598816321
Decision Tree : 0.7098149290587392
Random Forest : 0.9999990733902435
ADA Boost : 0.7479034185270778
```

```
In [ ]: exp_log_full
```

Out[ ]:

Pipeline

Parameter

0	Baseline Pipeline(steps=[('rf', RandomForestClassifier())]) with 100 inputs	{'rf__n_estimators': [20], 'rf__criterion': ['gini', 'log_loss']}
---	--	--

1	Baseline Pipeline(steps=[('dt', DecisionTreeClassifier())]) with 100 inputs	{'dt__max_depth': [5, 10]}
---	--	----------------------------

2	Baseline Pipeline(steps=[('lr', LogisticRegression())]) with 100 inputs	{'lr__C': [0.01], 'lr__penalty': ['l1', 'l2']}
---	--	--

3	Baseline Pipeline(steps=[('ada', AdaBoostClassifier(base_estimator=DecisionTreeClassifier()))]) with 100 inputs	{'ada__n_estimators': [20], 'ada__learning_rate': [0.01, 0.1], 'ada__base_estimator__max_depth': [1, 5]}
---	---	--

Pipeline	Parameter

## Combining the test data with the remaining CSVs

```
In [ ]: df_bureau.shape
```

```
Out[ ]: (305811, 16)
```

```
In [ ]: df_bureau = df_bureau.groupby("SK_ID_BUREAU").mean().reset_index()
```

```
In [ ]: df_app_test=pd.DataFrame(datasets["application_test"])
df_app_test = df_app_test.merge(df_bureau.reset_index(), how="left", on="SK_
df_app_test.shape
```

```
Out[ ]: (48777, 137)
```

```
In [ ]: df_prev_app = df_prev_app.groupby("SK_ID_CURR").mean().reset_index()
df_app_test= df_app_test.merge(df_prev_app.reset_index(), how="left", on="SK_
df_app_test.shape
```

```
Out[ ]: (48777, 158)
```

```
In [ ]: df_pos = df_pos.groupby("SK_ID_CURR").mean().reset_index()
df_app_test = df_app_test.merge(df_pos.reset_index(), how="left", on="SK_ID_CURR")
df_app_test.shape
```

```
Out[ ]: (48777, 165)
```

```
In [ ]: df_inst = df_inst.groupby("SK_ID_CURR").mean().reset_index()
df_app_test = df_app_test.merge(df_inst.reset_index(), how="left", on="SK_ID_CURR")
df_app_test.shape
```

```
Out[ ]: (48777, 173)
```

```
In [ ]: df_credit_card_balance = df_credit_card_balance.groupby("SK_ID_CURR").mean()
df_app_test = df_app_test.merge(df_credit_card_balance.reset_index(), how="left", on="SK_ID_CURR")
df_app_test.shape
```

```
Out[ ]: (48777, 195)
```

```
In [ ]: DF_Train = datasets["application_train"]
DF_Test = datasets["application_test"]
DF_Bureau = datasets["bureau"]
DF_BureauBalance = datasets["bureau_balance"]
DF_PosCashBalance = datasets["POS_CASH_balance"]
DF_PreviousApplication = datasets["previous_application"]
DF_CreditCardBalance = datasets["credit_card_balance"]
DF_InstallmentsPayemnts = datasets["installments_payments"]
```

```
In [ ]: DF_InstallmentsPayemnts = DF_InstallmentsPayemnts.select_dtypes(exclude='object')
DF_InstallmentsPayemnts = DF_InstallmentsPayemnts.groupby('SK_ID_CURR').median()
```

```
In [ ]: DF_PosCashBalance = DF_PosCashBalance.select_dtypes(exclude='object')
DF_PosCashBalance = DF_PosCashBalance.groupby('SK_ID_CURR').median().reset_index()
# DF_PosCashBalance = DF_PosCashBalance
```

```
In [ ]: DF_PreviousApplication = DF_PreviousApplication.select_dtypes(exclude='object')
DF_PreviousApplication = DF_PreviousApplication.groupby('SK_ID_CURR').median()
```

```
In [ ]: DF_CreditCardBalance = DF_CreditCardBalance.select_dtypes(exclude='object')
DF_CreditCardBalance = DF_CreditCardBalance.groupby('SK_ID_CURR').median().reset_index()
```

```
In [ ]: Merged_BureauBalance = pd.merge(left=DF_Bureau, right=DF_BureauBalance, how="left")
```

```
In [ ]: Merged_BureauBalance = Merged_BureauBalance.drop_duplicates()
```

```
In [ ]: Merged_BureauBalance.shape
```

```
Out[ ]: (25121815, 19)
```

```
In [ ]: Merged_BureauBalance = Merged_BureauBalance.groupby(['SK_ID_CURR', 'SK_ID_BUREAU'])
```

```
[CV 2/2; 1/2] START rf_criterion=gini, rf_n_estimators=2
0.....
[CV 2/2; 1/2] END rf_criterion=gini, rf_n_estimators=20;, score=0.668 total time= 1.8min
[CV 2/2; 2/2] START dt_max_depth=1
0.....
[CV 2/2; 2/2] END .....dt_max_depth=10;, score=0.696 total time= 5.2s
[CV 1/2; 2/2] START lr_C=0.01, lr_penalty=l
2.....
[CV 1/2; 2/2] END ...lr_C=0.01, lr_penalty=l2;, score=0.735 total time= 0.9s
[CV 2/2; 2/4] START ada_base_estimator_max_depth=1, ada_learning_rate=0.
1, ada_n_estimators=20
[CV 2/2; 2/4] END ada_base_estimator_max_depth=1, ada_learning_rate=0.1,
ada_n_estimators=20;, score=0.708 total time= 10.5s
[CV 2/2; 3/4] START ada_base_estimator_max_depth=5, ada_learning_rate=0.
01, ada_n_estimators=20
[CV 2/2; 3/4] END ada_base_estimator_max_depth=5, ada_learning_rate=0.0
1, ada_n_estimators=20;, score=0.714 total time= 49.6s
[CV 1/2; 1/2] START rf_criterion=gini, rf_n_estimators=2
0.....
[CV 1/2; 1/2] END rf_criterion=gini, rf_n_estimators=20;, score=0.670 total time= 1.8min
[CV 1/2; 2/2] START dt_max_depth=1
0.....
[CV 1/2; 2/2] END .....dt_max_depth=10;, score=0.694 total time= 5.2s
[CV 2/2; 2/2] START lr_C=0.01, lr_penalty=l
2.....
[CV 2/2; 2/2] END ...lr_C=0.01, lr_penalty=l2;, score=0.734 total time= 0.8s
[CV 2/2; 1/4] START ada_base_estimator_max_depth=1, ada_learning_rate=0.
01, ada_n_estimators=20
[CV 2/2; 1/4] END ada_base_estimator_max_depth=1, ada_learning_rate=0.0
1, ada_n_estimators=20;, score=0.659 total time= 10.4s
[CV 1/2; 3/4] START ada_base_estimator_max_depth=5, ada_learning_rate=0.
01, ada_n_estimators=20
[CV 1/2; 3/4] END ada_base_estimator_max_depth=5, ada_learning_rate=0.0
1, ada_n_estimators=20;, score=0.713 total time= 50.3s
[CV 2/2; 2/2] START rf_criterion=log_loss, rf_n_estimators=2
0.....
[CV 2/2; 2/2] END rf_criterion=log_loss, rf_n_estimators=20;, score=0.677 total time= 1.8min
[CV 2/2; 1/2] START dt_max_depth=
5.....
[CV 2/2; 1/2] END .....dt_max_depth=5;, score=0.706 total time= 2.5s
[CV 1/2; 1/2] START lr_C=0.01, lr_penalty=l
1.....
[CV 1/2; 1/2] END .....lr_C=0.01, lr_penalty=l1;, score=nan total time= 0.1s
[CV 1/2; 1/4] START ada_base_estimator_max_depth=1, ada_learning_rate=0.
01, ada_n_estimators=20
[CV 1/2; 1/4] END ada_base_estimator_max_depth=1, ada_learning_rate=0.0
1, ada_n_estimators=20;, score=0.662 total time= 10.8s
```

```

[CV 1/2; 4/4] START ada_base_estimator_max_depth=5, ada_learning_rate=0.
1, ada_n_estimators=20
[CV 1/2; 4/4] END ada_base_estimator_max_depth=5, ada_learning_rate=0.1,
ada_n_estimators=20;, score=0.729 total time= 50.2s
[CV 1/2; 2/2] START rf_criterion=log_loss, rf_n_estimators=2
0.....
[CV 1/2; 2/2] END rf_criterion=log_loss, rf_n_estimators=20;, score=0.676
total time= 1.7min
[CV 1/2; 1/2] START dt_max_depth=
5.....
[CV 1/2; 1/2] END .....dt_max_depth=5;, score=0.702 total time=
2.5s
[CV 2/2; 1/2] START lr_C=0.01, lr_penalty=l
1.....
[CV 2/2; 1/2] END .....lr_C=0.01, lr_penalty=l1;, score=nan total time=
0.1s
[CV 1/2; 2/4] START ada_base_estimator_max_depth=1, ada_learning_rate=0.
1, ada_n_estimators=20
[CV 1/2; 2/4] END ada_base_estimator_max_depth=1, ada_learning_rate=0.1,
ada_n_estimators=20;, score=0.706 total time= 10.9s
[CV 2/2; 4/4] START ada_base_estimator_max_depth=5, ada_learning_rate=0.
1, ada_n_estimators=20
[CV 2/2; 4/4] END ada_base_estimator_max_depth=5, ada_learning_rate=0.1,
ada_n_estimators=20;, score=0.730 total time= 50.2s

```

```
In [ ]: Updated_BureauBalance = Merged_BureauBalance[['SK_ID_CURR', 'DAYS_CREDIT_UPDA
Updated_BureauBalance = Updated_BureauBalance.reset_index()
```

```
In [ ]: Updated_BureauBalance = Updated_BureauBalance.groupby('SK_ID_CURR').median()
# Updated_BureauBalance = Updated_BureauBalance.median()
# Updated_BureauBalance = Updated_BureauBalance.reset_index()
```

```
In [ ]: Updated_BureauBalance = Updated_BureauBalance.select_dtypes(exclude='object'
Updated_BureauBalance.info()

<class 'pandas.core.frame.DataFrame'>
RangeIndex: 305811 entries, 0 to 305810
Data columns (total 7 columns):
 #   Column           Non-Null Count  Dtype  
--- 
 0   SK_ID_CURR        305811 non-null   int64  
 1   index             305811 non-null   float64 
 2   DAYS_CREDIT_UPDATE 305811 non-null   float64 
 3   DAYS_CREDIT        305811 non-null   float64 
 4   AMT_CREDIT_SUM     305809 non-null   float64 
 5   MONTHS_BALANCE    134542 non-null   float64 
 6   DAYS_ENDDATE_FACT 268155 non-null   float64 
dtypes: float64(6), int64(1)
memory usage: 16.3 MB
```

```
In [ ]: def Count_Percentage_of_Zeroes(DataFrame,Thres_Percentage):
    out = pd.DataFrame()
    temp1 = []
    temp2 =[]
    for i in DataFrame.columns:
        if i == 'TARGET':
```

```

        continue
    count = (DataFrame[i] == 0).sum()
    temp1.append(i)

    temp2.append(count/len(DataFrame[i]))
out['Column'] = temp1
out['Percentage'] = temp2
Thres_Percentage = Thres_Percentage/100
out = out[out['Percentage']>Thres_Percentage]
return out
UnwantedColumns = Count_Percentage_of_Zeroes(DF_Train,85)
print(UnwantedColumns)

```

	Column	Percentage
26	FLAG_EMAIL	0.943280
33	REG_REGION_NOT_LIVE_REGION	0.984856
34	REG_REGION_NOT_WORK_REGION	0.949231
35	LIVE_REGION_NOT_WORK_REGION	0.959341
36	REG_CITY_NOT_LIVE_CITY	0.921827
91	DEF_30_CNT_SOCIAL_CIRCLE	0.882323
93	DEF_60_CNT_SOCIAL_CIRCLE	0.912881
95	FLAG_DOCUMENT_2	0.999958
97	FLAG_DOCUMENT_4	0.999919
98	FLAG_DOCUMENT_5	0.984885
99	FLAG_DOCUMENT_6	0.911945
100	FLAG_DOCUMENT_7	0.999808
101	FLAG_DOCUMENT_8	0.918624
102	FLAG_DOCUMENT_9	0.996104
103	FLAG_DOCUMENT_10	0.999977
104	FLAG_DOCUMENT_11	0.996088
105	FLAG_DOCUMENT_12	0.999993
106	FLAG_DOCUMENT_13	0.996475
107	FLAG_DOCUMENT_14	0.997064
108	FLAG_DOCUMENT_15	0.998790
109	FLAG_DOCUMENT_16	0.990072
110	FLAG_DOCUMENT_17	0.999733
111	FLAG_DOCUMENT_18	0.991870
112	FLAG_DOCUMENT_19	0.999405
113	FLAG_DOCUMENT_20	0.999493
114	FLAG_DOCUMENT_21	0.999665
115	AMT_REQ_CREDIT_BUREAU_HOUR	0.859696
116	AMT_REQ_CREDIT_BUREAU_DAY	0.860142

```

In [ ]: DF_Train.drop(columns = UnwantedColumns['Column'], inplace = True)
def Divide_DataFrame_into_Categorical_and_Numerical(DataFrame):
    Numerical_DataFrame= DataFrame.select_dtypes(exclude='object')
    Numerical_DataFrame['TARGET'] = DataFrame['TARGET']
    Categorical_DataFrame= DF_Train.select_dtypes(include='object')
    return Numerical_DataFrame,Categorical_DataFrame
New_DataFrame_Numerical, New_DataFrame_Categorical = Divide_DataFrame_into_C
New_DataFrame_Numerical.describe()

```

Out [ ]:

	SK_ID_CURR	TARGET	CNT_CHILDREN	AMT_INCOME_TOTAL	AMT_CREDIT
<b>count</b>	307511.000000	307511.000000	307511.000000	3.075110e+05	3.075110e+05
<b>mean</b>	278180.518577	0.080729	0.417052	1.687979e+05	5.990260e+05
<b>std</b>	102790.175348	0.272419	0.722121	2.371231e+05	4.024908e+05
<b>min</b>	100002.000000	0.000000	0.000000	2.565000e+04	4.500000e+04
<b>25%</b>	189145.500000	0.000000	0.000000	1.125000e+05	2.700000e+05
<b>50%</b>	278202.000000	0.000000	0.000000	1.471500e+05	5.135310e+05
<b>75%</b>	367142.500000	0.000000	1.000000	2.025000e+05	8.086500e+05
<b>max</b>	456255.000000	1.000000	19.000000	1.170000e+08	4.050000e+06

In [ ]: New\_DataFrame\_Numerical.describe().T

Out[ ]:		count	mean	std	m
	<b>SK_ID_CURR</b>	307511.0	278180.518577	102790.175348	1.000020e+0
	<b>TARGET</b>	307511.0	0.080729	0.272419	0.000000e+0
	<b>CNT_CHILDREN</b>	307511.0	0.417052	0.722121	0.000000e+0
	<b>AMT_INCOME_TOTAL</b>	307511.0	168797.919297	237123.146279	2.565000e+0
	<b>AMT_CREDIT</b>	307511.0	599025.999706	402490.776996	4.500000e+0
	<b>AMT_ANNUITY</b>	307499.0	27108.573909	14493.737315	1.615500e+0
	<b>AMT_GOODS_PRICE</b>	307233.0	538396.207429	369446.460540	4.050000e+0
	<b>REGION_POPULATION_RELATIVE</b>	307511.0	0.020868	0.013831	2.900000e-0
	<b>DAYS_BIRTH</b>	307511.0	-16036.995067	4363.988632	-2.522900e+0
	<b>DAYS_EMPLOYED</b>	307511.0	63815.045904	141275.766519	-1.791200e+0
	<b>DAYS_REGISTRATION</b>	307511.0	-4986.120328	3522.886321	-2.467200e+0
	<b>DAYS_ID_PUBLISH</b>	307511.0	-2994.202373	1509.450419	-7.197000e+0
	<b>OWN_CAR_AGE</b>	104582.0	12.061091	11.944812	0.000000e+0
	<b>FLAG_MOBIL</b>	307511.0	0.999997	0.001803	0.000000e+0
	<b>FLAG_EMP_PHONE</b>	307511.0	0.819889	0.384280	0.000000e+0
	<b>FLAG_WORK_PHONE</b>	307511.0	0.199368	0.399526	0.000000e+0
	<b>FLAG_CONT_MOBILE</b>	307511.0	0.998133	0.043164	0.000000e+0
	<b>FLAG_PHONE</b>	307511.0	0.281066	0.449521	0.000000e+0
	<b>CNT_FAM_MEMBERS</b>	307509.0	2.152665	0.910682	1.000000e+0
	<b>REGION_RATING_CLIENT</b>	307511.0	2.052463	0.509034	1.000000e+0
	<b>REGION_RATING_CLIENT_W_CITY</b>	307511.0	2.031521	0.502737	1.000000e+0
	<b>HOUR_APPR_PROCESS_START</b>	307511.0	12.063419	3.265832	0.000000e+0
	<b>REG_CITY_NOT_WORK_CITY</b>	307511.0	0.230454	0.421124	0.000000e+0
	<b>LIVE_CITY_NOT_WORK_CITY</b>	307511.0	0.179555	0.383817	0.000000e+0
	<b>EXT_SOURCE_1</b>	134133.0	0.502130	0.211062	1.456813e-0
	<b>EXT_SOURCE_2</b>	306851.0	0.514393	0.191060	8.173617e-0
	<b>EXT_SOURCE_3</b>	246546.0	0.510853	0.194844	5.272652e-0
	<b>APARTMENTS_AVG</b>	151450.0	0.117440	0.108240	0.000000e+0
	<b>BASEMENTAREA_AVG</b>	127568.0	0.088442	0.082438	0.000000e+0
	<b>YEARS_BEGINEXPLUATATION_AVG</b>	157504.0	0.977735	0.059223	0.000000e+0
	<b>YEARS_BUILD_AVG</b>	103023.0	0.752471	0.113280	0.000000e+0
	<b>COMMONAREA_AVG</b>	92646.0	0.044621	0.076036	0.000000e+0
	<b>ELEVATORS_AVG</b>	143620.0	0.078942	0.134576	0.000000e+0
	<b>ENTRANCES_AVG</b>	152683.0	0.149725	0.100049	0.000000e+0

		count	mean	std	m
	<b>FLOORSMAX_AVG</b>	154491.0	0.226282	0.144641	0.000000e+(0)
	<b>FLOORSMIN_AVG</b>	98869.0	0.231894	0.161380	0.000000e+(0)
	<b>LANDAREA_AVG</b>	124921.0	0.066333	0.081184	0.000000e+(0)
	<b>LIVINGAPARTMENTS_AVG</b>	97312.0	0.100775	0.092576	0.000000e+(0)
	<b>LIVINGAREA_AVG</b>	153161.0	0.107399	0.110565	0.000000e+(0)
	<b>NONLIVINGAPARTMENTS_AVG</b>	93997.0	0.008809	0.047732	0.000000e+(0)
	<b>NONLIVINGAREA_AVG</b>	137829.0	0.028358	0.069523	0.000000e+(0)
	<b>APARTMENTS_MODE</b>	151450.0	0.114231	0.107936	0.000000e+(0)
	<b>BASEMENTAREA_MODE</b>	127568.0	0.087543	0.084307	0.000000e+(0)
	<b>YEARS_BEGINEXPLUATATION_MODE</b>	157504.0	0.977065	0.064575	0.000000e+(0)
	<b>YEARS_BUILD_MODE</b>	103023.0	0.759637	0.110111	0.000000e+(0)
	<b>COMMONAREA_MODE</b>	92646.0	0.042553	0.074445	0.000000e+(0)
	<b>ELEVATORS_MODE</b>	143620.0	0.074490	0.132256	0.000000e+(0)
	<b>ENTRANCES_MODE</b>	152683.0	0.145193	0.100977	0.000000e+(0)
	<b>FLOORSMAX_MODE</b>	154491.0	0.222315	0.143709	0.000000e+(0)
	<b>FLOORSMIN_MODE</b>	98869.0	0.228058	0.161160	0.000000e+(0)
	<b>LANDAREA_MODE</b>	124921.0	0.064958	0.081750	0.000000e+(0)
	<b>LIVINGAPARTMENTS_MODE</b>	97312.0	0.105645	0.097880	0.000000e+(0)
	<b>LIVINGAREA_MODE</b>	153161.0	0.105975	0.111845	0.000000e+(0)
	<b>NONLIVINGAPARTMENTS_MODE</b>	93997.0	0.008076	0.046276	0.000000e+(0)
	<b>NONLIVINGAREA_MODE</b>	137829.0	0.027022	0.070254	0.000000e+(0)
	<b>APARTMENTS_MEDI</b>	151450.0	0.117850	0.109076	0.000000e+(0)
	<b>BASEMENTAREA_MEDI</b>	127568.0	0.087955	0.082179	0.000000e+(0)
	<b>YEARS_BEGINEXPLUATATION_MEDI</b>	157504.0	0.977752	0.059897	0.000000e+(0)
	<b>YEARS_BUILD_MEDI</b>	103023.0	0.755746	0.112066	0.000000e+(0)
	<b>COMMONAREA_MEDI</b>	92646.0	0.044595	0.076144	0.000000e+(0)
	<b>ELEVATORS_MEDI</b>	143620.0	0.078078	0.134467	0.000000e+(0)
	<b>ENTRANCES_MEDI</b>	152683.0	0.149213	0.100368	0.000000e+(0)
	<b>FLOORSMAX_MEDI</b>	154491.0	0.225897	0.145067	0.000000e+(0)
	<b>FLOORSMIN_MEDI</b>	98869.0	0.231625	0.161934	0.000000e+(0)
	<b>LANDAREA_MEDI</b>	124921.0	0.067169	0.082167	0.000000e+(0)
	<b>LIVINGAPARTMENTS_MEDI</b>	97312.0	0.101954	0.093642	0.000000e+(0)
	<b>LIVINGAREA_MEDI</b>	153161.0	0.108607	0.112260	0.000000e+(0)
	<b>NONLIVINGAPARTMENTS_MEDI</b>	93997.0	0.008651	0.047415	0.000000e+(0)

		count	mean	std	m
	<b>NONLIVINGAREA_MEDI</b>	137829.0	0.028236	0.070166	0.000000e+0
	<b>TOTALAREA_MODE</b>	159080.0	0.102547	0.107462	0.000000e+0
	<b>OBS_30_CNT_SOCIAL_CIRCLE</b>	306490.0	1.422245	2.400989	0.000000e+0
	<b>OBS_60_CNT_SOCIAL_CIRCLE</b>	306490.0	1.405292	2.379803	0.000000e+0
	<b>DAYS_LAST_PHONE_CHANGE</b>	307510.0	-962.858788	826.808487	-4.292000e+0
	<b>FLAG_DOCUMENT_3</b>	307511.0	0.710023	0.453752	0.000000e+0
	<b>AMT_REQ_CREDIT_BUREAU_WEEK</b>	265992.0	0.034362	0.204685	0.000000e+0
	<b>AMT_REQ_CREDIT_BUREAU_MON</b>	265992.0	0.267395	0.916002	0.000000e+0
	<b>AMT_REQ_CREDIT_BUREAU_QRT</b>	265992.0	0.265474	0.794056	0.000000e+0

In [ ]: New\_DataFrame\_Categorical.describe().T

	count	unique	top	freq
<b>NAME_CONTRACT_TYPE</b>	307511	2	Cash loans	278232
<b>CODE_GENDER</b>	307511	3	F	202448
<b>FLAG_OWN_CAR</b>	307511	2	N	202924
<b>FLAG_OWN_REALTY</b>	307511	2	Y	213312
<b>NAME_TYPE_SUITE</b>	306219	7	Unaccompanied	248526
<b>NAME_INCOME_TYPE</b>	307511	8	Working	158774
<b>NAME_EDUCATION_TYPE</b>	307511	5	Secondary / secondary special	218391
<b>NAME_FAMILY_STATUS</b>	307511	6	Married	196432
<b>NAME_HOUSING_TYPE</b>	307511	6	House / apartment	272868
<b>OCCUPATION_TYPE</b>	211120	18	Laborers	55186
<b>WEEKDAY_APPR_PROCESS_START</b>	307511	7	TUESDAY	53901
<b>ORGANIZATION_TYPE</b>	307511	58	Business Entity Type 3	67992
<b>FONDKAPREMONT_MODE</b>	97216	4	reg oper account	73830
<b>HOUSETYPE_MODE</b>	153214	3	block of flats	150503
<b>WALLSMATERIAL_MODE</b>	151170	7	Panel	66040
<b>EMERGENCYSTATE_MODE</b>	161756	2	No	159428

```
In [ ]: def Correlation_Between_InputDF_Target(DataFrame,correlation):
    temp = DataFrame.corr()['TARGET']
    temp = temp.sort_values(key=abs,ascending=False)
    temp=temp.reset_index()
    temp.columns = ['col_name','Correlation']
    out = temp[temp['Correlation']>correlation]
    return out
```

```
p= Correlation_Between_InputDF_Target(New_DataFrame_Numerical,0.00)  
p
```

Out[ ]:

	col_name	Correlation
0	TARGET	1.000000
1	EXT_SOURCE_3	-0.178919
2	EXT_SOURCE_2	-0.160472
3	EXT_SOURCE_1	-0.155317
4	DAY_S_BIRTH	0.078239
5	REGION_RATING_CLIENT_W_CITY	0.060893
6	REGION_RATING_CLIENT	0.058899
7	DAY_S_LAST_PHONE_CHANGE	0.055218
8	DAY_S_ID_PUBLISH	0.051457
9	REG_CITY_NOT_WORK_CITY	0.050994
10	FLAG_EMP_PHONE	0.045982
11	DAY_S_EMPLOYED	-0.044932
12	FLAG_DOCUMENT_3	0.044346
13	FLOORSMAX_AVG	-0.044003
14	FLOORSMAX_MEDI	-0.043768
15	FLOORSMAX_MODE	-0.043226
16	DAY_S_REGISTRATION	0.041975
17	AMT_GOODS_PRICE	-0.039645
18	OWN_CAR_AGE	0.037612
19	REGION_POPULATION_RELATIVE	-0.037227
20	ELEVATORS_AVG	-0.034199
21	ELEVATORS_MEDI	-0.033863
22	FLOORSMIN_AVG	-0.033614
23	FLOORSMIN_MEDI	-0.033394
24	LIVINGAREA_AVG	-0.032997
25	LIVINGAREA_MEDI	-0.032739
26	FLOORSMIN_MODE	-0.032698
27	TOTALAREA_MODE	-0.032596
28	LIVE_CITY_NOT_WORK_CITY	0.032518
29	ELEVATORS_MODE	-0.032131
30	LIVINGAREA_MODE	-0.030685
31	AMT_CREDIT	-0.030369
32	APARTMENTS_AVG	-0.029498
33	APARTMENTS_MEDI	-0.029184

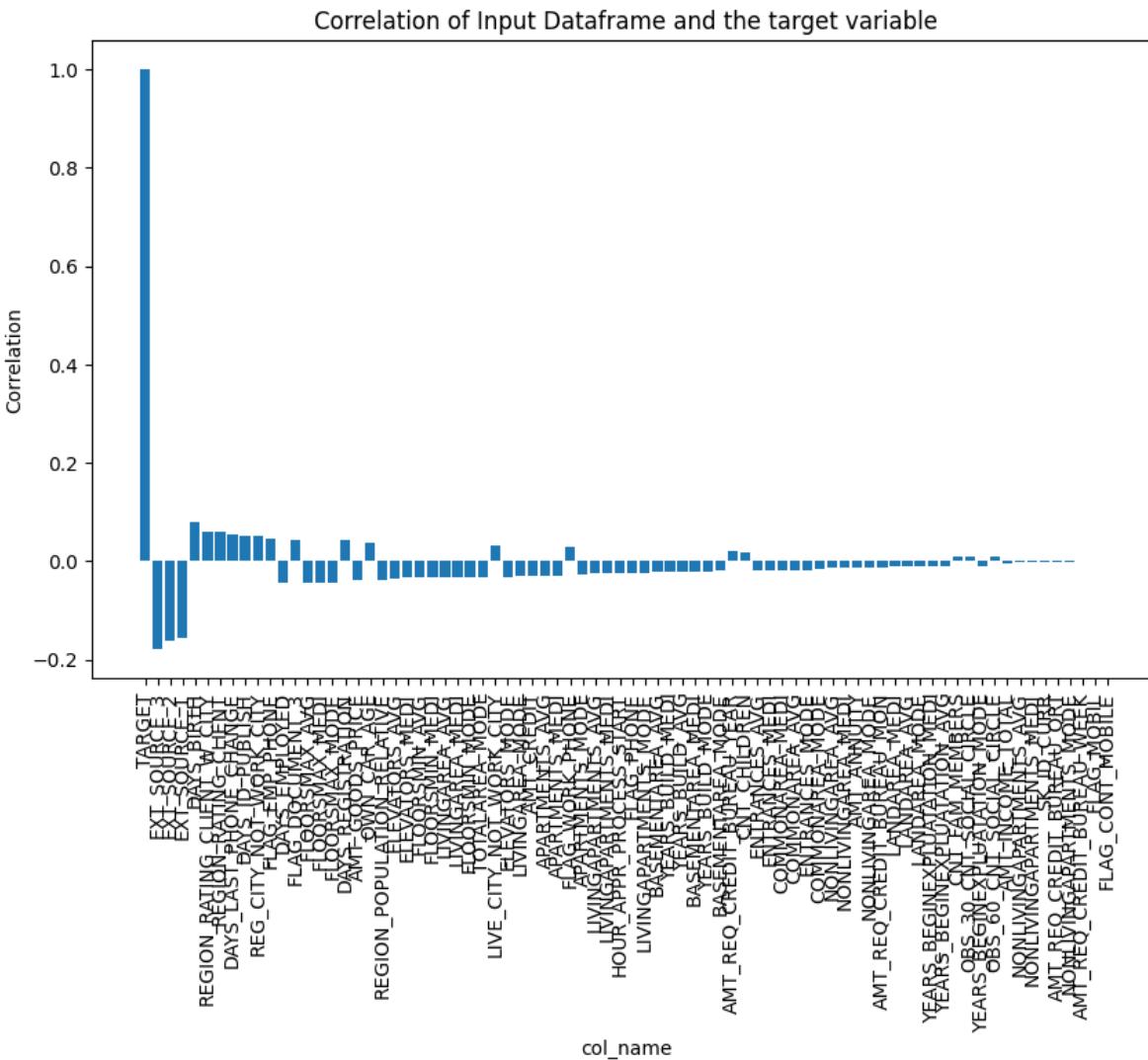
	col_name	Correlation
34	FLAG_WORK_PHONE	0.028524
35	APARTMENTS_MODE	-0.027284
36	LIVINGAPARTMENTS_AVG	-0.025031
37	LIVINGAPARTMENTS_MEDI	-0.024621
38	HOUR_APPR_PROCESS_START	-0.024166
39	FLAG_PHONE	-0.023806
40	LIVINGAPARTMENTS_MODE	-0.023393
41	BASEMENTAREA_AVG	-0.022746
42	YEARS_BUILD_MEDI	-0.022326
43	YEARS_BUILD_AVG	-0.022149
44	BASEMENTAREA_MEDI	-0.022081
45	YEARS_BUILD_MODE	-0.022068
46	BASEMENTAREA_MODE	-0.019952
47	AMT_REQ_CREDIT_BUREAU_YEAR	0.019930
48	CNT_CHILDREN	0.019187
49	ENTRANCES_AVG	-0.019172
50	ENTRANCES_MEDI	-0.019025
51	COMMONAREA_MEDI	-0.018573
52	COMMONAREA_AVG	-0.018550
53	ENTRANCES_MODE	-0.017387
54	COMMONAREA_MODE	-0.016340
55	NONLIVINGAREA_AVG	-0.013578
56	NONLIVINGAREA_MEDI	-0.013337
57	AMT_ANNUITY	-0.012817
58	NONLIVINGAREA_MODE	-0.012711
59	AMT_REQ_CREDIT_BUREAU_MON	-0.012462
60	LANDAREA_MEDI	-0.011256
61	LANDAREA_AVG	-0.010885
62	LANDAREA_MODE	-0.010174
63	YEARS_BEGINEXPLUATATION_MEDI	-0.009993
64	YEARS_BEGINEXPLUATATION_AVG	-0.009728
65	CNT_FAM_MEMBERS	0.009308
66	OBS_30_CNT_SOCIAL_CIRCLE	0.009131
67	YEARS_BEGINEXPLUATATION_MODE	-0.009036

	col_name	Correlation
68	OBS_60_CNT_SOCIAL_CIRCLE	0.009022
69	AMT_INCOME_TOTAL	-0.003982
70	NONLIVINGAPARTMENTS_AVG	-0.003176
71	NONLIVINGAPARTMENTS_MEDI	-0.002757
72	SK_ID_CURR	-0.002108
73	AMT_REQ_CREDIT_BUREAU_QRT	-0.002022
74	NONLIVINGAPARTMENTS_MODE	-0.001557
75	AMT_REQ_CREDIT_BUREAU_WEEK	0.000788
76	FLAG_MOBIL	0.000534
77	FLAG_CONT_MOBILE	0.000370

```
In [ ]: import matplotlib.pyplot as plt
import numpy as np

# Creating a new figure
plt.figure(figsize=(10, 6), dpi=100)
# Plotting Numpy array
plt.bar(p["col_name"], p["Correlation"])
# Adding details to the plot
plt.title('Correlation of Input Dataframe and the target variable')
plt.xlabel('col_name')
plt.ylabel('Correlation')

# Rotating the x-labels
plt.xticks(rotation=90)
# Displaying the plot
plt.show()
```



```
In [ ]: DF_Train =DF_Train[DF_Train['NAME_INCOME_TYPE']!='Maternity leave']
DF_Train =DF_Train[DF_Train['NAME_FAMILY_STATUS']!='Unknown']
DF_Train =DF_Train[DF_Train['CODE_GENDER']!='XNA']
New_DataFrame = Correlation_Between_InputDF_Target(New_DataFrame_Numerical,€
New_DataFrame.describe())
```

Out[ ]:

Correlation	
<b>count</b>	46.000000
<b>mean</b>	0.003511
<b>std</b>	0.159330
<b>min</b>	-0.178919
<b>25%</b>	-0.033559
<b>50%</b>	-0.026157
<b>75%</b>	0.031520
<b>max</b>	1.000000

```
In [ ]: New_DataFrame
```

Out[ ]:

	col_name	Correlation
0	TARGET	1.000000
1	EXT_SOURCE_3	-0.178919
2	EXT_SOURCE_2	-0.160472
3	EXT_SOURCE_1	-0.155317
4	DAY_S_BIRTH	0.078239
5	REGION_RATING_CLIENT_W_CITY	0.060893
6	REGION_RATING_CLIENT	0.058899
7	DAY_S_LAST_PHONE_CHANGE	0.055218
8	DAY_S_ID_PUBLISH	0.051457
9	REG_CITY_NOT_WORK_CITY	0.050994
10	FLAG_EMP_PHONE	0.045982
11	DAY_S_EMPLOYED	-0.044932
12	FLAG_DOCUMENT_3	0.044346
13	FLOORSMAX_AVG	-0.044003
14	FLOORSMAX_MEDI	-0.043768
15	FLOORSMAX_MODE	-0.043226
16	DAY_S_REGISTRATION	0.041975
17	AMT_GOODS_PRICE	-0.039645
18	OWN_CAR_AGE	0.037612
19	REGION_POPULATION_RELATIVE	-0.037227
20	ELEVATORS_AVG	-0.034199
21	ELEVATORS_MEDI	-0.033863
22	FLOORSMIN_AVG	-0.033614
23	FLOORSMIN_MEDI	-0.033394
24	LIVINGAREA_AVG	-0.032997
25	LIVINGAREA_MEDI	-0.032739
26	FLOORSMIN_MODE	-0.032698
27	TOTALAREA_MODE	-0.032596
28	LIVE_CITY_NOT_WORK_CITY	0.032518
29	ELEVATORS_MODE	-0.032131
30	LIVINGAREA_MODE	-0.030685
31	AMT_CREDIT	-0.030369
32	APARTMENTS_AVG	-0.029498
33	APARTMENTS_MEDI	-0.029184

	col_name	Correlation
34	FLAG_WORK_PHONE	0.028524
35	APARTMENTS_MODE	-0.027284
36	LIVINGAPARTMENTS_AVG	-0.025031
37	LIVINGAPARTMENTS_MEDI	-0.024621
38	HOUR_APPR_PROCESS_START	-0.024166
39	FLAG_PHONE	-0.023806
40	LIVINGAPARTMENTS_MODE	-0.023393
41	BASEMENTAREA_AVG	-0.022746
42	YEARS_BUILD_MEDI	-0.022326
43	YEARS_BUILD_AVG	-0.022149
44	BASEMENTAREA_MEDI	-0.022081
45	YEARS_BUILD_MODE	-0.022068

```
In [ ]: New_DataFrame_Numerical.describe().T
```

Out[ ]:		count	mean	std	m
	<b>SK_ID_CURR</b>	307511.0	278180.518577	102790.175348	1.000020e+0
	<b>TARGET</b>	307511.0	0.080729	0.272419	0.000000e+0
	<b>CNT_CHILDREN</b>	307511.0	0.417052	0.722121	0.000000e+0
	<b>AMT_INCOME_TOTAL</b>	307511.0	168797.919297	237123.146279	2.565000e+0
	<b>AMT_CREDIT</b>	307511.0	599025.999706	402490.776996	4.500000e+0
	<b>AMT_ANNUITY</b>	307499.0	27108.573909	14493.737315	1.615500e+0
	<b>AMT_GOODS_PRICE</b>	307233.0	538396.207429	369446.460540	4.050000e+0
	<b>REGION_POPULATION_RELATIVE</b>	307511.0	0.020868	0.013831	2.900000e-0
	<b>DAYS_BIRTH</b>	307511.0	-16036.995067	4363.988632	-2.522900e+0
	<b>DAYS_EMPLOYED</b>	307511.0	63815.045904	141275.766519	-1.791200e+0
	<b>DAYS_REGISTRATION</b>	307511.0	-4986.120328	3522.886321	-2.467200e+0
	<b>DAYS_ID_PUBLISH</b>	307511.0	-2994.202373	1509.450419	-7.197000e+0
	<b>OWN_CAR_AGE</b>	104582.0	12.061091	11.944812	0.000000e+0
	<b>FLAG_MOBIL</b>	307511.0	0.999997	0.001803	0.000000e+0
	<b>FLAG_EMP_PHONE</b>	307511.0	0.819889	0.384280	0.000000e+0
	<b>FLAG_WORK_PHONE</b>	307511.0	0.199368	0.399526	0.000000e+0
	<b>FLAG_CONT_MOBILE</b>	307511.0	0.998133	0.043164	0.000000e+0
	<b>FLAG_PHONE</b>	307511.0	0.281066	0.449521	0.000000e+0
	<b>CNT_FAM_MEMBERS</b>	307509.0	2.152665	0.910682	1.000000e+0
	<b>REGION_RATING_CLIENT</b>	307511.0	2.052463	0.509034	1.000000e+0
	<b>REGION_RATING_CLIENT_W_CITY</b>	307511.0	2.031521	0.502737	1.000000e+0
	<b>HOUR_APPR_PROCESS_START</b>	307511.0	12.063419	3.265832	0.000000e+0
	<b>REG_CITY_NOT_WORK_CITY</b>	307511.0	0.230454	0.421124	0.000000e+0
	<b>LIVE_CITY_NOT_WORK_CITY</b>	307511.0	0.179555	0.383817	0.000000e+0
	<b>EXT_SOURCE_1</b>	134133.0	0.502130	0.211062	1.456813e-0
	<b>EXT_SOURCE_2</b>	306851.0	0.514393	0.191060	8.173617e-0
	<b>EXT_SOURCE_3</b>	246546.0	0.510853	0.194844	5.272652e-0
	<b>APARTMENTS_AVG</b>	151450.0	0.117440	0.108240	0.000000e+0
	<b>BASEMENTAREA_AVG</b>	127568.0	0.088442	0.082438	0.000000e+0
	<b>YEARS_BEGINEXPLUATATION_AVG</b>	157504.0	0.977735	0.059223	0.000000e+0
	<b>YEARS_BUILD_AVG</b>	103023.0	0.752471	0.113280	0.000000e+0
	<b>COMMONAREA_AVG</b>	92646.0	0.044621	0.076036	0.000000e+0
	<b>ELEVATORS_AVG</b>	143620.0	0.078942	0.134576	0.000000e+0
	<b>ENTRANCES_AVG</b>	152683.0	0.149725	0.100049	0.000000e+0

		count	mean	std	m
	<b>FLOORSMAX_AVG</b>	154491.0	0.226282	0.144641	0.000000e+(0)
	<b>FLOORSMIN_AVG</b>	98869.0	0.231894	0.161380	0.000000e+(0)
	<b>LANDAREA_AVG</b>	124921.0	0.066333	0.081184	0.000000e+(0)
	<b>LIVINGAPARTMENTS_AVG</b>	97312.0	0.100775	0.092576	0.000000e+(0)
	<b>LIVINGAREA_AVG</b>	153161.0	0.107399	0.110565	0.000000e+(0)
	<b>NONLIVINGAPARTMENTS_AVG</b>	93997.0	0.008809	0.047732	0.000000e+(0)
	<b>NONLIVINGAREA_AVG</b>	137829.0	0.028358	0.069523	0.000000e+(0)
	<b>APARTMENTS_MODE</b>	151450.0	0.114231	0.107936	0.000000e+(0)
	<b>BASEMENTAREA_MODE</b>	127568.0	0.087543	0.084307	0.000000e+(0)
	<b>YEARS_BEGINEXPLUATATION_MODE</b>	157504.0	0.977065	0.064575	0.000000e+(0)
	<b>YEARS_BUILD_MODE</b>	103023.0	0.759637	0.110111	0.000000e+(0)
	<b>COMMONAREA_MODE</b>	92646.0	0.042553	0.074445	0.000000e+(0)
	<b>ELEVATORS_MODE</b>	143620.0	0.074490	0.132256	0.000000e+(0)
	<b>ENTRANCES_MODE</b>	152683.0	0.145193	0.100977	0.000000e+(0)
	<b>FLOORSMAX_MODE</b>	154491.0	0.222315	0.143709	0.000000e+(0)
	<b>FLOORSMIN_MODE</b>	98869.0	0.228058	0.161160	0.000000e+(0)
	<b>LANDAREA_MODE</b>	124921.0	0.064958	0.081750	0.000000e+(0)
	<b>LIVINGAPARTMENTS_MODE</b>	97312.0	0.105645	0.097880	0.000000e+(0)
	<b>LIVINGAREA_MODE</b>	153161.0	0.105975	0.111845	0.000000e+(0)
	<b>NONLIVINGAPARTMENTS_MODE</b>	93997.0	0.008076	0.046276	0.000000e+(0)
	<b>NONLIVINGAREA_MODE</b>	137829.0	0.027022	0.070254	0.000000e+(0)
	<b>APARTMENTS_MEDI</b>	151450.0	0.117850	0.109076	0.000000e+(0)
	<b>BASEMENTAREA_MEDI</b>	127568.0	0.087955	0.082179	0.000000e+(0)
	<b>YEARS_BEGINEXPLUATATION_MEDI</b>	157504.0	0.977752	0.059897	0.000000e+(0)
	<b>YEARS_BUILD_MEDI</b>	103023.0	0.755746	0.112066	0.000000e+(0)
	<b>COMMONAREA_MEDI</b>	92646.0	0.044595	0.076144	0.000000e+(0)
	<b>ELEVATORS_MEDI</b>	143620.0	0.078078	0.134467	0.000000e+(0)
	<b>ENTRANCES_MEDI</b>	152683.0	0.149213	0.100368	0.000000e+(0)
	<b>FLOORSMAX_MEDI</b>	154491.0	0.225897	0.145067	0.000000e+(0)
	<b>FLOORSMIN_MEDI</b>	98869.0	0.231625	0.161934	0.000000e+(0)
	<b>LANDAREA_MEDI</b>	124921.0	0.067169	0.082167	0.000000e+(0)
	<b>LIVINGAPARTMENTS_MEDI</b>	97312.0	0.101954	0.093642	0.000000e+(0)
	<b>LIVINGAREA_MEDI</b>	153161.0	0.108607	0.112260	0.000000e+(0)
	<b>NONLIVINGAPARTMENTS_MEDI</b>	93997.0	0.008651	0.047415	0.000000e+(0)

		count	mean	std	m
	<b>NONLIVINGAREA_MEDI</b>	137829.0	0.028236	0.070166	0.000000e+0
	<b>TOTALAREA_MODE</b>	159080.0	0.102547	0.107462	0.000000e+0
	<b>OBS_30_CNT_SOCIAL_CIRCLE</b>	306490.0	1.422245	2.400989	0.000000e+0
	<b>OBS_60_CNT_SOCIAL_CIRCLE</b>	306490.0	1.405292	2.379803	0.000000e+0
	<b>DAYS_LAST_PHONE_CHANGE</b>	307510.0	-962.858788	826.808487	-4.292000e+0
	<b>FLAG_DOCUMENT_3</b>	307511.0	0.710023	0.453752	0.000000e+0
	<b>AMT_REQ_CREDIT_BUREAU_WEEK</b>	265992.0	0.034362	0.204685	0.000000e+0
	<b>AMT_REQ_CREDIT_BUREAU_MON</b>	265992.0	0.267395	0.916002	0.000000e+0
	<b>AMT_REQ_CREDIT_BUREAU_QRT</b>	265992.0	0.265474	0.794056	0.000000e+0

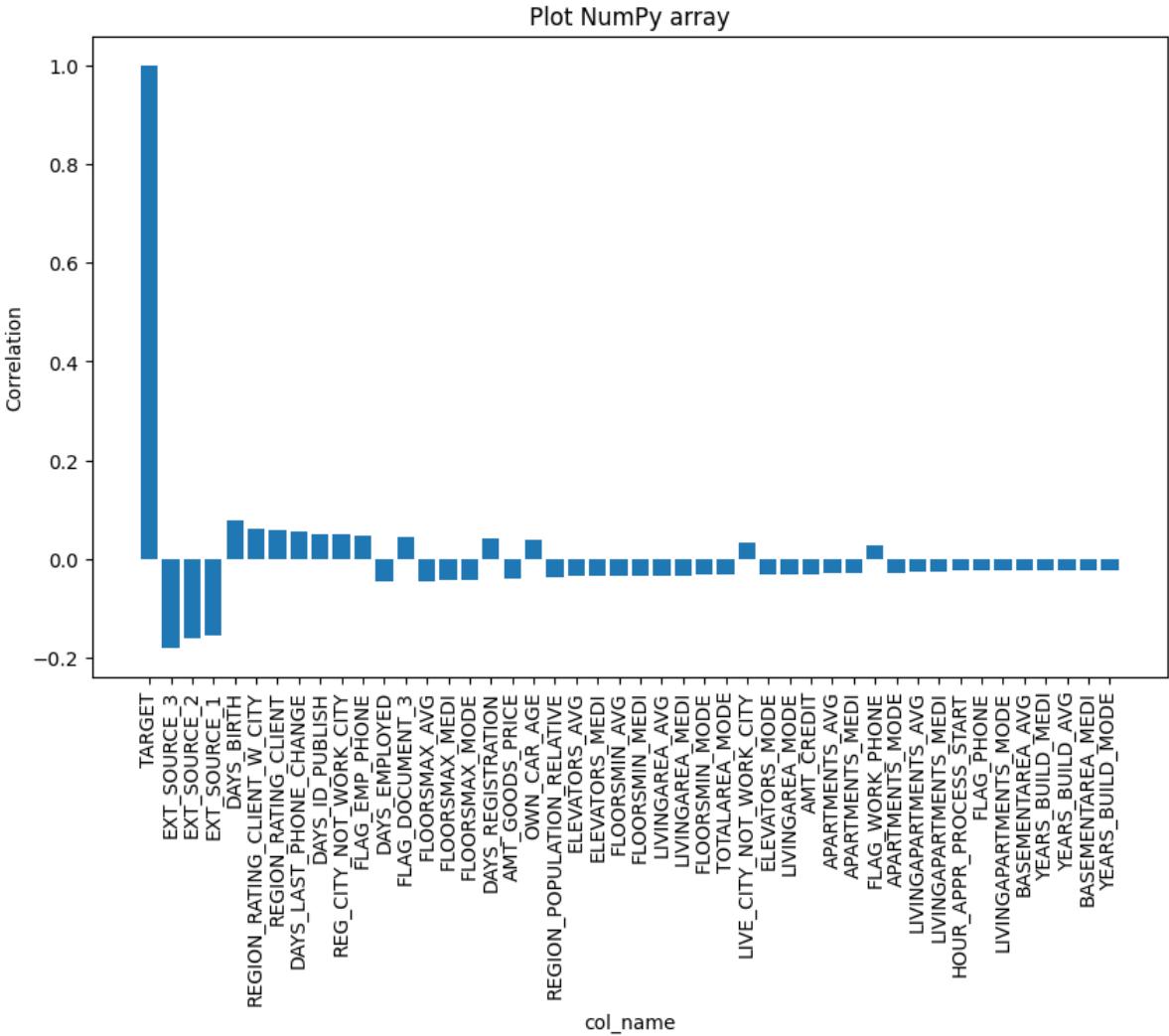
```
In [ ]: import matplotlib.pyplot as plt
import numpy as np
```

```
# Creating a new figure
plt.figure(figsize=(10, 6), dpi=100)
# Plotting Numpy array
plt.bar(New_DataFrame["col_name"], New_DataFrame["Correlation"])

# Adding details to the plot
plt.title('Plot NumPy array')
plt.xlabel('col_name')
plt.ylabel('Correlation')

# Rotating the x-labels
plt.xticks(rotation=90)

# Displaying the plot
plt.show()
```



```
In [ ]: def helper_function(DataFrame):
    length = len(DataFrame)
    temp4 = DataFrame.isnull()
    temp4= temp4.sum(axis=0)*100/length
    return temp4.sort_values(ascending=False)

def Missing_Values_Above_Threshold(DataFrame,threshold):
    temp5 = helper_function(DataFrame)
    temp5= temp5.reset_index()
    temp5.columns = ['index','flag']
    ref1 = []
    for i in temp5.itertuples():
        try:
            i1= DataFrame[i.index].median()
            i2= DataFrame[i.index].mean()
            i3= DataFrame[i.index].nunique()
            ref1.append([i.index,i.flag,i1,i2,i3])
        except:
            i4= DataFrame[i.index].mode()
            i5= DataFrame[i.index].nunique()
            ref1.append([i.index,i.flag,i4,'NA',i5])
    ref = ['col_name','percentage_missing','median/Mode','mean','no_of_unique']
    main_out = pd.DataFrame(ref1,columns=ref)
```

```

    return main_out[main_out['percentage_missing']>threshold]

New_DataFrame_Absent = Missing_Values_Above_Threshold(DF_Train[New_DataFrame
temp1 = list(set(New_DataFrame.col_name).difference(set(New_DataFrame_Absent
temp2 = New_DataFrame_Absent[New_DataFrame_Absent['no_of_unique_values']<=10
New_DataFrame_Absent = New_DataFrame_Absent[New_DataFrame_Absent['no_of_unic

for i in temp2.col_name:
    temp6= New_DataFrame_Numerical[i]
    New_DataFrame_Numerical[i] = temp6.fillna(temp6.mode()[0])
for i in temp2.col_name:
    DF_Test[i] = DF_Test[i].fillna(DF_Test[i].mode()[0])

temp3 = list(temp2.col_name) + list(New_DataFrame_Absent.col_name) + temp1
New_DataFrame_Numerical_Temp = New_DataFrame_Numerical
New_DataFrame_Numerical = New_DataFrame_Numerical[temp3]
New_DataFrame_Numerical.insert(0,"SK_ID_CURR",New_DataFrame_Numerical_Temp['
New_DataFrame_Numerical

```

In [ ]: New\_DataFrame\_Cat\_Temp = Missing\_Values\_Above\_Threshold(New\_DataFrame\_Categc  
New\_DataFrame\_Cat\_Temp.describe())

Out[ ]:

	percentage_missing	no_of_unique_values
<b>count</b>	6.000000	6.000000
<b>mean</b>	41.427841	6.833333
<b>std</b>	23.284402	5.845226
<b>min</b>	0.420148	2.000000
<b>25%</b>	35.358735	3.250000
<b>50%</b>	48.787198	5.500000
<b>75%</b>	50.674610	7.000000
<b>max</b>	68.386172	18.000000

In [ ]: Columns\_to\_Drop =[]
Columns\_to\_Drop.append('FONDKAPREMONT\_MODE')
Columns\_to\_Drop.append('WALLSMATERIAL\_MODE')
Columns\_to\_Drop.append('OCCUPATION\_TYPE')
New\_DataFrame\_Categorical.drop(columns = Columns\_to\_Drop,inplace=True)

In [ ]: from sklearn.compose import ColumnTransformer, make\_column\_transformer
from sklearn.model\_selection import train\_test\_split
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import OneHotEncoder
from sklearn.impute import SimpleImputer
from sklearn.linear\_model import LogisticRegression
from sklearn.preprocessing import StandardScaler

```
In [ ]: DF_Temp = list(New_DataFrame_Numerical.columns) + list(New_DataFrame_Categorical.columns)
DF_Temp1 = DF_Train[DF_Temp]

Numerical_Pipeline = Pipeline(steps=[('imputer', SimpleImputer(strategy='mean')),
Categorical_Pipeline = Pipeline([
    ('imputer', SimpleImputer(strategy='most_frequent')),
    ('ohe', OneHotEncoder(sparse=False, handle_unknown="ignore"))
])
Data_Pipeline = ColumnTransformer([
    ("num_pipeline", Numerical_Pipeline, New_DataFrame_Numerical.columns),
    ("cat_pipeline", Categorical_Pipeline, New_DataFrame_Categorical.columns)
])

model = Data_Pipeline.fit_transform(DF_Temp1)
```

```
/usr/local/lib/python3.11/site-packages/sklearn/preprocessing/_encoders.py:828: FutureWarning: `sparse` was renamed to `sparse_output` in version 1.2 and will be removed in 1.4. `sparse_output` is ignored unless you leave `sparse` to its default value.
  warnings.warn(
```

```
In [ ]: # Get feature names after one-hot encoding
ohe_feature_names = Data_Pipeline.named_transformers_['cat_pipeline'].named_steps['OneHotEncoder'].get_feature_names_out()
ohe_feature_names
```

```
Out[ ]: array(['NAME_CONTRACT_TYPE_Cash loans',
   'NAME_CONTRACT_TYPE_Revolving loans', 'CODE_GENDER_F',
   'CODE_GENDER_M', 'FLAG_OWN_CAR_N', 'FLAG_OWN_CAR_Y',
   'FLAG_OWN_REALTY_N', 'FLAG_OWN_REALTY_Y',
   'NAME_TYPE_SUITE_Children', 'NAME_TYPE_SUITE_Family',
   'NAME_TYPE_SUITE_Group of people', 'NAME_TYPE_SUITE_Other_A',
   'NAME_TYPE_SUITE_Other_B', 'NAME_TYPE_SUITE_Spouse, partner',
   'NAME_TYPE_SUITE_Unaccompanied', 'NAME_INCOME_TYPE_Businessman',
   'NAME_INCOME_TYPE_Commercial associate',
   'NAME_INCOME_TYPE_Pensioner', 'NAME_INCOME_TYPE_State servant',
   'NAME_INCOME_TYPE_Student', 'NAME_INCOME_TYPE_Unemployed',
   'NAME_INCOME_TYPE_Working', 'NAME_EDUCATION_TYPE_Academic degree',
   'NAME_EDUCATION_TYPE_Higher education',
   'NAME_EDUCATION_TYPE_Incomplete higher',
   'NAME_EDUCATION_TYPE_Lower secondary',
   'NAME_EDUCATION_TYPE_Secondary / secondary special',
   'NAME_FAMILY_STATUS_Civil marriage', 'NAME_FAMILY_STATUS_Married',
   'NAME_FAMILY_STATUS_Separated',
   'NAME_FAMILY_STATUS_Single / not married',
   'NAME_FAMILY_STATUS_Widow', 'NAME_HOUSING_TYPE_Co-op apartment',
   'NAME_HOUSING_TYPE_House / apartment',
   'NAME_HOUSING_TYPE_Municipal apartment',
   'NAME_HOUSING_TYPE_Office apartment',
   'NAME_HOUSING_TYPE_Rented apartment',
   'NAME_HOUSING_TYPE_With parents',
   'WEEKDAY_APPR_PROCESS_START_FRIDAY',
   'WEEKDAY_APPR_PROCESS_START_MONDAY',
   'WEEKDAY_APPR_PROCESS_START_SATURDAY',
   'WEEKDAY_APPR_PROCESS_START_SUNDAY',
   'WEEKDAY_APPR_PROCESS_START_THURSDAY',
   'WEEKDAY_APPR_PROCESS_START_TUESDAY',
   'WEEKDAY_APPR_PROCESS_START_WEDNESDAY',
   'ORGANIZATION_TYPE_Advertising', 'ORGANIZATION_TYPE_Agriculture',
   'ORGANIZATION_TYPE_Bank',
   'ORGANIZATION_TYPE_Business Entity Type 1',
   'ORGANIZATION_TYPE_Business Entity Type 2',
   'ORGANIZATION_TYPE_Business Entity Type 3',
   'ORGANIZATION_TYPE_Cleaning', 'ORGANIZATION_TYPE_Construction',
   'ORGANIZATION_TYPE_Culture', 'ORGANIZATION_TYPE_Electricity',
   'ORGANIZATION_TYPE_Emergency', 'ORGANIZATION_TYPE_Government',
   'ORGANIZATION_TYPE_Hotel', 'ORGANIZATION_TYPE_Housing',
   'ORGANIZATION_TYPE_Industry: type 1',
   'ORGANIZATION_TYPE_Industry: type 10',
   'ORGANIZATION_TYPE_Industry: type 11',
   'ORGANIZATION_TYPE_Industry: type 12',
   'ORGANIZATION_TYPE_Industry: type 13',
   'ORGANIZATION_TYPE_Industry: type 2',
   'ORGANIZATION_TYPE_Industry: type 3',
   'ORGANIZATION_TYPE_Industry: type 4',
   'ORGANIZATION_TYPE_Industry: type 5',
   'ORGANIZATION_TYPE_Industry: type 6',
   'ORGANIZATION_TYPE_Industry: type 7',
   'ORGANIZATION_TYPE_Industry: type 8',
   'ORGANIZATION_TYPE_Industry: type 9',
   'ORGANIZATION_TYPE_Insurance', 'ORGANIZATION_TYPE_Kindergarten',
   'ORGANIZATION_TYPE_Legal Services', 'ORGANIZATION_TYPE_Medicine',
```

```
'ORGANIZATION_TYPE_Military', 'ORGANIZATION_TYPE_Mobile',
'ORGANIZATION_TYPE_Other', 'ORGANIZATION_TYPE_Police',
'ORGANIZATION_TYPE_Postal', 'ORGANIZATION_TYPE_Realtor',
'ORGANIZATION_TYPE_Religion', 'ORGANIZATION_TYPE_Restaurant',
'ORGANIZATION_TYPE_School', 'ORGANIZATION_TYPE_Security',
'ORGANIZATION_TYPE_Security Ministries',
'ORGANIZATION_TYPE_Self-employed', 'ORGANIZATION_TYPE_Services',
'ORGANIZATION_TYPE_Telecom', 'ORGANIZATION_TYPE_Trade: type 1',
'ORGANIZATION_TYPE_Trade: type 2',
'ORGANIZATION_TYPE_Trade: type 3',
'ORGANIZATION_TYPE_Trade: type 4',
'ORGANIZATION_TYPE_Trade: type 5',
'ORGANIZATION_TYPE_Trade: type 6',
'ORGANIZATION_TYPE_Trade: type 7',
'ORGANIZATION_TYPE_Transport: type 1',
'ORGANIZATION_TYPE_Transport: type 2',
'ORGANIZATION_TYPE_Transport: type 3',
'ORGANIZATION_TYPE_Transport: type 4',
'ORGANIZATION_TYPE_University', 'ORGANIZATION_TYPE_XNA',
'HOUSETYPE_MODE_block of flats', 'HOUSETYPE_MODE_specific housing',
'HOUSETYPE_MODE_terraced house', 'EMERGENCYSTATE_MODE_No',
'EMERGENCYSTATE_MODE_Yes'], dtype=object)
```

```
In [ ]: c = list(New_DataFrame_Numerical.columns) + list(ohe_feature_names)
DF_Temp2 = pd.DataFrame(model, columns=c)
```

```
In [ ]: DF_Temp2 = pd.merge(left=DF_Temp2, right=DF_InstallmentsPayemnts, how='left')
```

```
In [ ]: DF_Temp2 = pd.merge(left=DF_Temp2, right=DF_PosCashBalance, how='left', left
```

```
In [ ]: DF_Temp2 = pd.merge(left=DF_Temp2, right=DF_PreviousApplication, how='left',
```

```
In [ ]: DF_Temp2 = pd.merge(left=DF_Temp2, right=DF_CreditCardBalance, how='left', l
```

```
In [ ]: DF_Temp2 = pd.merge(left=DF_Temp2, right=Updated_BureauBalance, how='left',
DF_Final = DF_Temp2
DF_Final.head(10)
```

Out [ ]:

	SK_ID_CURR	LIVINGAPARTMENTS_MODE	LIVINGAPARTMENTS_MEDI	LIVINGAPARTMENTS
0	100002.0	0.022000	0.020500	0.0
1	100003.0	0.079000	0.078700	0.0
2	100004.0	0.105646	0.101956	0.1
3	100006.0	0.105646	0.101956	0.1
4	100007.0	0.105646	0.101956	0.1
5	100008.0	0.105646	0.101956	0.1
6	100009.0	0.105646	0.101956	0.1
7	100010.0	0.105646	0.101956	0.1
8	100011.0	0.105646	0.101956	0.1
9	100012.0	0.105646	0.101956	0.1

In [ ]: DF\_Final.describe()

Out [ ]:

	SK_ID_CURR	LIVINGAPARTMENTS_MODE	LIVINGAPARTMENTS_MEDI	LIVINGAPART
<b>count</b>	307500.000000	307500.000000	307500.000000	30
<b>mean</b>	278181.087798	0.105646	0.101956	
<b>std</b>	102789.822017	0.055062	0.052678	
<b>min</b>	100002.000000	0.000000	0.000000	
<b>25%</b>	189146.750000	0.105646	0.101956	
<b>50%</b>	278202.500000	0.105646	0.101956	
<b>75%</b>	367143.250000	0.105646	0.101956	
<b>max</b>	456255.000000	1.000000	1.000000	

In [ ]: DF\_Final['Feature\_1']= DF\_Final['AMT\_CREDIT\_SUM']/(DF\_Final['DAYS\_EMPLOYED'])  
DF\_Final['Feature\_2']= DF\_Final['CNT\_DRAWINGS\_POS\_CURRENT']/(DF\_Final['FLOORSMIN\_AVG'])  
DF\_Final['Feature\_3']= DF\_Final['AMT\_CREDIT\_SUM']/(DF\_Final['FLOORSMIN\_AVG'])  
DF\_Final['Feature\_4']=DF\_Final['DAYS\_CREDIT']\*(DF\_Final['DAYS\_ENDDATE\_FACT'])  
DF\_Final['Feature\_5']=DF\_Final['AMT\_CREDIT\_SUM']/(DF\_Final['AMT\_CREDIT\_SUM'])  
DF\_Final['Feature\_6']=DF\_Final['AMT\_CREDIT\_x']/(DF\_Final['AMT\_CREDIT\_SUM']+1)  
DF\_Final['Feature\_7']=(DF\_Final['DAYS\_CREDIT']+DF\_Final['AMT\_CREDIT\_SUM'])  
DF\_Final['Feature\_8']=(DF\_Final['AMT\_CREDIT\_x']+DF\_Final['FLOORSMIN\_AVG'])  
DF\_Final['Feature\_9']= DF\_Final['AMT\_CREDIT\_SUM']/(DF\_Final['AMT\_GOODS\_PRIM'])  
DF\_Final['Feature\_10']= DF\_Final['AMT\_CREDIT\_SUM']/(DF\_Final['AMT\_GOODS\_PRIM'])  
DF\_Final['Feature\_11']= DF\_Final['DAYS\_BIRTH']+(DF\_Final['DAYS\_EMPLOYED']+1)  
DF\_Final['Feature\_12']= (DF\_Final['EXT\_SOURCE\_1']+DF\_Final['EXT\_SOURCE\_2']+DF\_Final['Feature\_13'])  
DF\_Final['Feature\_13']= DF\_Final['CNT\_INSTALMENT\_MATURE\_CUM']\*(DF\_Final['CNT\_INSTALMENT\_MATURE\_CUM'])  
DF\_Final.shape

```
Out[ ]: (307500, 228)
```

```
In [ ]: #DF_InstallmentsPayemnts
if 'PAYMENT_DATE' in DF_PosCashBalance.columns:
    print('The payment_date column exists in the installments_payments.csv file')
else:
    print('The payment_date column does not exist in the installments_payments.csv file')
```

```
The payment_date column does not exist in the installments_payments.csv file
```

```
In [ ]: DF_Train = datasets["application_train"]
DF_Test = datasets["application_test"]
DF_Bureau = datasets["bureau"]
DF_BureauBalance = datasets["bureau_balance"]
DF_PosCashBalance = datasets["POS_CASH_balance"]
DF_PreviousApplication = datasets["previous_application"]
DF_CreditCardBalance = datasets["credit_card_balance"]
DF_InstallmentsPayemnts = datasets["installments_payments"]
```

```
In [ ]: import pandas as pd
import numpy as np

# Load the HCGR dataset and select relevant columns
hcdr_df = datasets["application_train"]
hcdr_df = hcdr_df[['SK_ID_CURR', 'AMT_CREDIT', 'AMT_ANNUITY']]

# Calculate monetary value as the average of AMT_CREDIT and AMT_ANNUITY
hcdr_df['MonetaryValue'] = hcdr_df[['AMT_CREDIT', 'AMT_ANNUITY']].mean(axis=1)

# Load the previous application dataset and select relevant columns
prev_df = datasets["previous_application"]
prev_df = prev_df[['SK_ID_CURR', 'DAYS_DECISION']]

# Calculate recency as the number of days since the latest previous application
prev_df = prev_df.groupby('SK_ID_CURR').max().reset_index()
prev_df['Recency'] = prev_df['DAYS_DECISION'] * -1

# Merge hcdr_df and prev_df on SK_ID_CURR
hcdr_rfm = pd.merge(hcdr_df, prev_df[['SK_ID_CURR', 'Recency']], on='SK_ID_CURR')

# Calculate frequency as the number of previous applications
hcdr_rfm['Frequency'] = prev_df.groupby('SK_ID_CURR').size().reset_index(name='Frequency')

# Create labels for RFM score
quantiles = hcdr_rfm.quantile(q=[0.2, 0.4, 0.6, 0.8])
quantiles = quantiles.to_dict()

def r_score(x, c):
    if x <= c[0.2]:
        return 4
    elif x <= c[0.4]:
        return 3
    elif x <= c[0.6]:
        return 2
    else:
        return 1
```

```

    elif x <= c[0.8]:
        return 1
    else:
        return 0

def fm_score(x, c):
    if x <= c[0.2]:
        return 0
    elif x <= c[0.4]:
        return 1
    elif x <= c[0.6]:
        return 2
    elif x <= c[0.8]:
        return 3
    else:
        return 4

hcdr_rfm['R'] = hcdr_rfm['Recency'].apply(r_score, args=(quantiles['Recency']))
hcdr_rfm['F'] = hcdr_rfm['Frequency'].apply(fm_score, args=(quantiles['Frequency']))
hcdr_rfm['M'] = hcdr_rfm['MonetaryValue'].apply(fm_score, args=(quantiles['MonetaryValue']))

# Calculate RFM score
hcdr_rfm['RFM'] = hcdr_rfm['R']*100 + hcdr_rfm['F']*10 + hcdr_rfm['M']

# Display the first 10 rows of hcdr_rfm
hcdr_rfm.head(10)

```

Out[ ]:

	SK_ID_CURR	AMT_CREDIT	AMT_ANNUITY	MonetaryValue	Recency	Frequency	R	F	M
0	100002	406597.5	24700.5	215649.00	606.0		1	1	0
1	100003	1293502.5	35698.5	664600.50	746.0		1	0	0
2	100004	1350000.0	6750.0	70875.00	815.0		1	0	0
3	100006	312682.5	29686.5	171184.50	181.0		1	3	0
4	100007	513000.0	21865.5	267432.75	374.0		1	2	0
5	100008	490495.5	27517.5	259006.50	82.0		1	4	0
6	100009	1560726.0	41301.0	801013.50	74.0		1	4	0
7	100010	1530000.0	42075.0	786037.50	1070.0		1	0	0
8	100011	1019610.0	33826.5	526718.25	1162.0		1	0	0
9	100012	405000.0	20250.0	212625.00	107.0		1	4	0

In [ ]: DF\_Final.shape

Out[ ]: (307500, 228)

In [ ]: DF\_Final['Frequency']= hcdr\_rfm["Frequency"]
DF\_Final['RFM']= hcdr\_rfm["RFM"]
DF\_Final['R']= hcdr\_rfm["R"]
DF\_Final['F']= hcdr\_rfm["F"]

```
DF_Final['M']= hcdr_rfm["M"]
DF_Final.shape
```

```
Out[ ]: (307500, 233)
```

```
In [ ]: Output_Correlation=Correlation_Between_InputDF_Target(DF_Final[['Feature_1'],
Output_Correlation
```

```
Out[ ]:
```

	col_name	Correlation
0	TARGET	1.000000
1	Feature_12	-0.221463
2	Feature_4	-0.054394
3	Feature_13	-0.050172
4	Feature_11	-0.043322
5	Feature_2	0.030728
6	Feature_8	-0.030390
7	Feature_7	-0.015612
8	Feature_3	-0.015356
9	Feature_9	-0.011549
10	Feature_10	-0.011549
11	Feature_1	-0.005593
12	Feature_5	-0.002558
13	Feature_6	0.000649

```
In [ ]: DF_Final = DF_Final.rename(columns = {'AMT_CREDIT_y':'AMT_CREDIT'})
DF_Final = DF_Final.apply(lambda x: x.fillna(x.median()),axis=0)
filter1 = Correlation_Between_InputDF_Target(DF_Final,0.05)
filter1
```

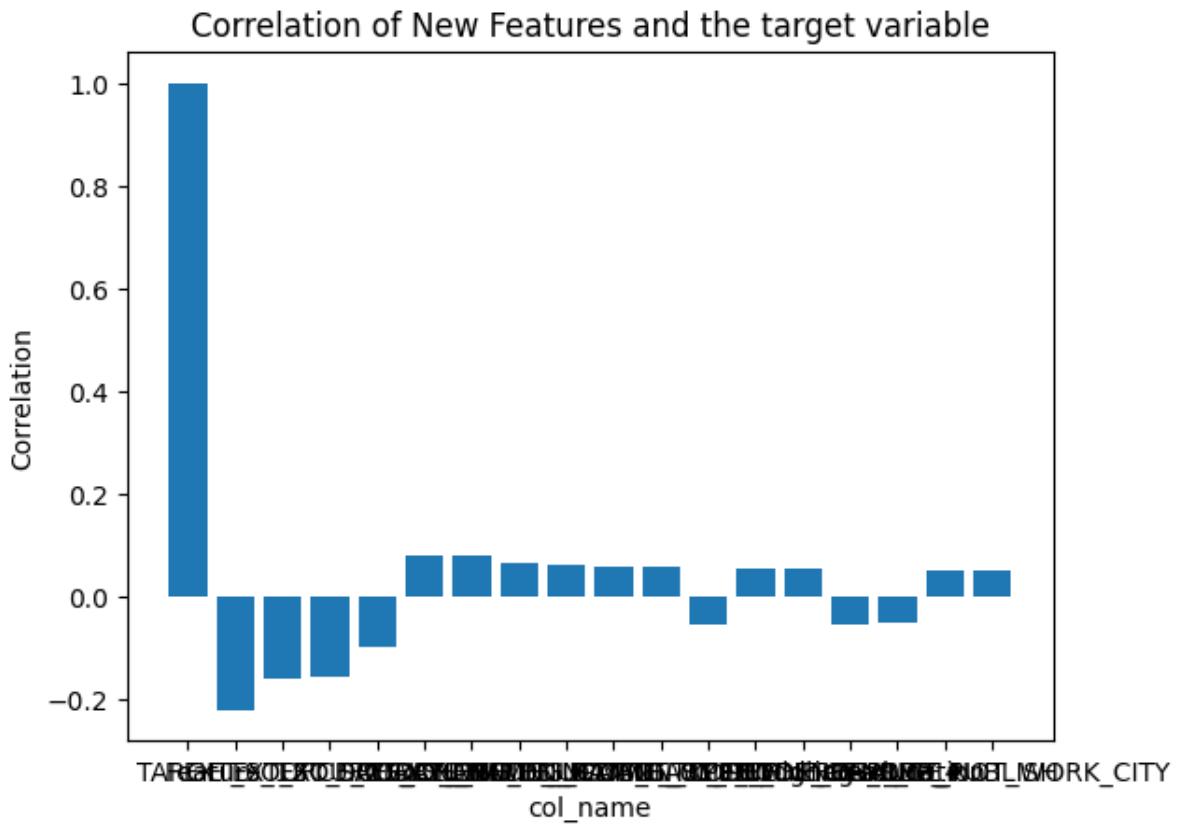
Out[ ]:

	col_name	Correlation
0	TARGET	1.000000
1	Feature_12	-0.221463
2	EXT_SOURCE_2	-0.160283
3	EXT_SOURCE_3	-0.157409
4	EXT_SOURCE_1	-0.099162
5	DAYS_CREDIT	0.079099
6	DAYS_BIRTH	0.078236
7	DAYS_CREDIT_UPDATE	0.063527
8	REGION_RATING_CLIENT_W_CITY	0.060875
9	REGION_RATING_CLIENT	0.058882
10	NAME_INCOME_TYPE_Working	0.057504
11	NAME_EDUCATION_TYPE_Higher education	-0.056578
12	DAYS_LAST_PHONE_CHANGE	0.055228
13	CODE_GENDER_M	0.054729
14	CODE_GENDER_F	-0.054729
15	Feature_4	-0.052507
16	DAYS_ID_PUBLISH	0.051455
17	REG_CITY_NOT_WORK_CITY	0.050981

In [ ]:

```
import matplotlib.pyplot as plt
import numpy as np

# Creating a new figure
plt.figure(dpi=100)
# Plotting Numpy array
plt.bar(filter1["col_name"],filter1["Correlation"])
# Adding details to the plot
plt.title('Correlation of New Features and the target variable')
plt.xlabel('col_name')
plt.ylabel('Correlation')
# Displaying the plot
plt.show()
```



```
In [ ]: filter1 = filter1['col_name'][1:]
filter1
```

```
Out[ ]: 1           Feature_12
2           EXT_SOURCE_2
3           EXT_SOURCE_3
4           EXT_SOURCE_1
5           DAYS_CREDIT
6           DAYS_BIRTH
7           DAYS_CREDIT_UPDATE
8           REGION_RATING_CLIENT_W_CITY
9           REGION_RATING_CLIENT
10          NAME_INCOME_TYPE_Working
11          NAME_EDUCATION_TYPE_Higher education
12          DAYS_LAST_PHONE_CHANGE
13          CODE_GENDER_M
14          CODE_GENDER_F
15          Feature_4
16          DAYS_ID_PUBLISH
17          REG_CITY_NOT_WORK_CITY
Name: col_name, dtype: object
```

```
In [ ]: filter1.describe()
```

```
Out[ ]: count      17
unique      17
top        Feature_12
freq       1
Name: col_name, dtype: object
```

```
In [ ]: T = DF_Final['TARGET']
DF_Temp4 = DF_Final[filter1]
X = DF_Temp4.values
y = T.values
X
```

```
Out[ ]: array([[ 4.85361340e-01,  2.62948593e-01,  1.39375780e-01, ...,
   9.77865000e+05, -2.12000000e+03,  0.00000000e+00],
   [ 1.44436873e+00,  6.22245775e-01,  5.10855639e-01, ...,
   7.47410000e+05, -2.91000000e+02,  0.00000000e+00],
   [ 1.78760234e+00,  5.55912083e-01,  7.29566691e-01, ...,
   4.60810500e+05, -2.53100000e+03,  0.00000000e+00],
   ...,
   [ 1.49860723e+00,  5.35721752e-01,  2.18859082e-01, ...,
   7.28767000e+05, -5.15000000e+03,  1.00000000e+00],
   [ 1.67730993e+00,  5.14162820e-01,  6.61023539e-01, ...,
   9.47232000e+05, -9.31000000e+02,  1.00000000e+00],
   [ 1.55695096e+00,  7.08568896e-01,  1.13922396e-01, ...,
   8.85870000e+05, -4.10000000e+02,  1.00000000e+00]])
```

---

## Experiment 3: Using Feature Engineering (Without New Features)

```
In [ ]: DF_Temp2 = DF_Temp2.apply(lambda x: x.fillna(x.median()),axis=0)
DF_Temp2= DF_Temp2.drop(['TARGET','Feature_1','Feature_2','Feature_3','Feature_4'])
X = DF_Temp2.values
y = T.values
i=0
for col in DF_Temp2.columns:
    print(i,col)
    i=i+1
```

0 SK\_ID\_CURR  
1 LIVINGAPARTMENTS\_MODE  
2 LIVINGAPARTMENTS\_MEDI  
3 LIVINGAPARTMENTS\_AVG  
4 FLOORSMIN\_MEDI  
5 FLOORSMIN\_AVG  
6 FLOORSMIN\_MODE  
7 YEARS\_BUILD\_AVG  
8 YEARS\_BUILD\_MEDI  
9 YEARS\_BUILD\_MODE  
10 OWN\_CAR\_AGE  
11 BASEMENTAREA\_MEDI  
12 BASEMENTAREA\_AVG  
13 EXT\_SOURCE\_1  
14 ELEVATORS\_AVG  
15 ELEVATORS\_MEDI  
16 ELEVATORS\_MODE  
17 APARTMENTS\_AVG  
18 APARTMENTS\_MEDI  
19 APARTMENTS\_MODE  
20 LIVINGAREA\_MEDI  
21 LIVINGAREA\_AVG  
22 LIVINGAREA\_MODE  
23 FLOORSMAX\_MODE  
24 FLOORSMAX\_MEDI  
25 FLOORSMAX\_AVG  
26 TOTALAREA\_MODE  
27 EXT\_SOURCE\_3  
28 EXT\_SOURCE\_2  
29 AMT\_GOODS\_PRICE\_x  
30 DAYS\_LAST\_PHONE\_CHANGE  
31 HOUR\_APPR\_PROCESS\_START\_x  
32 AMT\_CREDIT\_x  
33 DAYS\_ID\_PUBLISH  
34 REGION\_RATING\_CLIENT  
35 DAYS\_REGISTRATION  
36 LIVE\_CITY\_NOT\_WORK\_CITY  
37 REG\_CITY\_NOT\_WORK\_CITY  
38 FLAG\_EMP\_PHONE  
39 REGION\_POPULATION\_RELATIVE  
40 REGION\_RATING\_CLIENT\_W\_CITY  
41 DAYS\_BIRTH  
42 FLAG\_PHONE  
43 DAYS\_EMPLOYED  
44 FLAG\_WORK\_PHONE  
45 FLAG\_DOCUMENT\_3  
46 NAME\_CONTRACT\_TYPE\_Cash\_loans  
47 NAME\_CONTRACT\_TYPE\_Revolving\_loans  
48 CODE\_GENDER\_F  
49 CODE\_GENDER\_M  
50 FLAG\_OWN\_CAR\_N  
51 FLAG\_OWN\_CAR\_Y  
52 FLAG\_OWN\_REALTY\_N  
53 FLAG\_OWN\_REALTY\_Y  
54 NAME\_TYPE\_SUITE\_Children  
55 NAME\_TYPE\_SUITE\_Family

56 NAME\_TYPE\_SUITE\_Group of people  
57 NAME\_TYPE\_SUITE\_Other\_A  
58 NAME\_TYPE\_SUITE\_Other\_B  
59 NAME\_TYPE\_SUITE\_Spouse, partner  
60 NAME\_TYPE\_SUITE\_Unaccompanied  
61 NAME\_INCOME\_TYPE\_Businessman  
62 NAME\_INCOME\_TYPE\_Commercial associate  
63 NAME\_INCOME\_TYPE\_Pensioner  
64 NAME\_INCOME\_TYPE\_State servant  
65 NAME\_INCOME\_TYPE\_Student  
66 NAME\_INCOME\_TYPE\_Unemployed  
67 NAME\_INCOME\_TYPE\_Working  
68 NAME\_EDUCATION\_TYPE\_Academic degree  
69 NAME\_EDUCATION\_TYPE\_Higher education  
70 NAME\_EDUCATION\_TYPE\_Incomplete higher  
71 NAME\_EDUCATION\_TYPE\_Lower secondary  
72 NAME\_EDUCATION\_TYPE\_Secondary / secondary special  
73 NAME\_FAMILY\_STATUS\_Civil marriage  
74 NAME\_FAMILY\_STATUS\_Married  
75 NAME\_FAMILY\_STATUS\_Separated  
76 NAME\_FAMILY\_STATUS\_Single / not married  
77 NAME\_FAMILY\_STATUS\_Widow  
78 NAME\_HOUSING\_TYPE\_Co-op apartment  
79 NAME\_HOUSING\_TYPE\_House / apartment  
80 NAME\_HOUSING\_TYPE\_Municipal apartment  
81 NAME\_HOUSING\_TYPE\_Office apartment  
82 NAME\_HOUSING\_TYPE\_Rented apartment  
83 NAME\_HOUSING\_TYPE\_With parents  
84 WEEKDAY\_APPR\_PROCESS\_START\_FRIDAY  
85 WEEKDAY\_APPR\_PROCESS\_START\_MONDAY  
86 WEEKDAY\_APPR\_PROCESS\_START\_SATURDAY  
87 WEEKDAY\_APPR\_PROCESS\_START\_SUNDAY  
88 WEEKDAY\_APPR\_PROCESS\_START\_THURSDAY  
89 WEEKDAY\_APPR\_PROCESS\_START\_TUESDAY  
90 WEEKDAY\_APPR\_PROCESS\_START\_WEDNESDAY  
91 ORGANIZATION\_TYPE\_Advertising  
92 ORGANIZATION\_TYPE\_Agriculture  
93 ORGANIZATION\_TYPE\_Bank  
94 ORGANIZATION\_TYPE\_Business Entity Type 1  
95 ORGANIZATION\_TYPE\_Business Entity Type 2  
96 ORGANIZATION\_TYPE\_Business Entity Type 3  
97 ORGANIZATION\_TYPE\_Cleaning  
98 ORGANIZATION\_TYPE\_Construction  
99 ORGANIZATION\_TYPE\_Culture  
100 ORGANIZATION\_TYPE\_Electricity  
101 ORGANIZATION\_TYPE\_Emergency  
102 ORGANIZATION\_TYPE\_Government  
103 ORGANIZATION\_TYPE\_Hotel  
104 ORGANIZATION\_TYPE\_Housing  
105 ORGANIZATION\_TYPE\_Industry: type 1  
106 ORGANIZATION\_TYPE\_Industry: type 10  
107 ORGANIZATION\_TYPE\_Industry: type 11  
108 ORGANIZATION\_TYPE\_Industry: type 12  
109 ORGANIZATION\_TYPE\_Industry: type 13  
110 ORGANIZATION\_TYPE\_Industry: type 2  
111 ORGANIZATION\_TYPE\_Industry: type 3

112 ORGANIZATION\_TYPE\_Industry: type 4  
113 ORGANIZATION\_TYPE\_Industry: type 5  
114 ORGANIZATION\_TYPE\_Industry: type 6  
115 ORGANIZATION\_TYPE\_Industry: type 7  
116 ORGANIZATION\_TYPE\_Industry: type 8  
117 ORGANIZATION\_TYPE\_Industry: type 9  
118 ORGANIZATION\_TYPE\_Insurance  
119 ORGANIZATION\_TYPE\_Kindergarten  
120 ORGANIZATION\_TYPE\_Legal Services  
121 ORGANIZATION\_TYPE\_Medicine  
122 ORGANIZATION\_TYPE\_Military  
123 ORGANIZATION\_TYPE\_Mobile  
124 ORGANIZATION\_TYPE\_Other  
125 ORGANIZATION\_TYPE\_Police  
126 ORGANIZATION\_TYPE\_Postal  
127 ORGANIZATION\_TYPE\_Realtor  
128 ORGANIZATION\_TYPE\_Religion  
129 ORGANIZATION\_TYPE\_Restaurant  
130 ORGANIZATION\_TYPE\_School  
131 ORGANIZATION\_TYPE\_Security  
132 ORGANIZATION\_TYPE\_Security Ministries  
133 ORGANIZATION\_TYPE\_Self-employed  
134 ORGANIZATION\_TYPE\_Services  
135 ORGANIZATION\_TYPE\_Telecom  
136 ORGANIZATION\_TYPE\_Trade: type 1  
137 ORGANIZATION\_TYPE\_Trade: type 2  
138 ORGANIZATION\_TYPE\_Trade: type 3  
139 ORGANIZATION\_TYPE\_Trade: type 4  
140 ORGANIZATION\_TYPE\_Trade: type 5  
141 ORGANIZATION\_TYPE\_Trade: type 6  
142 ORGANIZATION\_TYPE\_Trade: type 7  
143 ORGANIZATION\_TYPE\_Transport: type 1  
144 ORGANIZATION\_TYPE\_Transport: type 2  
145 ORGANIZATION\_TYPE\_Transport: type 3  
146 ORGANIZATION\_TYPE\_Transport: type 4  
147 ORGANIZATION\_TYPE\_University  
148 ORGANIZATION\_TYPE\_XNA  
149 HOUSETYPE\_MODE\_block of flats  
150 HOUSETYPE\_MODE\_specific housing  
151 HOUSETYPE\_MODE\_terraced house  
152 EMERGENCYSTATE\_MODE\_No  
153 EMERGENCYSTATE\_MODE\_Yes  
154 SK\_ID\_PREV\_x  
155 NUM\_INSTALMENT\_VERSION  
156 NUM\_INSTALMENT\_NUMBER  
157 DAYS\_INSTALMENT  
158 DAYS\_ENTRY\_PAYMENT  
159 AMT\_INSTALMENT  
160 AMT\_PAYMENT  
161 SK\_ID\_PREV\_y  
162 MONTHS\_BALANCE\_x  
163 CNT\_INSTALMENT  
164 CNT\_INSTALMENT\_FUTURE  
165 SK\_DPD\_x  
166 SK\_DPD\_DEF\_x  
167 SK\_ID\_PREV\_x

168 AMT\_ANNUITY  
169 AMT\_APPLICATION  
170 AMT\_CREDIT\_y  
171 AMT\_DOWN\_PAYMENT  
172 AMT\_GOODS\_PRICE\_y  
173 HOUR\_APPR\_PROCESS\_START\_y  
174 NFLAG\_LAST\_APPL\_IN\_DAY  
175 RATE\_DOWN\_PAYMENT  
176 RATE\_INTEREST\_PRIMARY  
177 RATE\_INTEREST\_PRIVILEGED  
178 DAYS\_DECISION  
179 SELLERPLACE\_AREA  
180 CNT\_PAYMENT  
181 DAYS\_FIRST\_DRAWING  
182 DAYS\_FIRST\_DUE  
183 DAYS\_LAST\_DUE\_1ST\_VERSION  
184 DAYS\_LAST\_DUE  
185 DAYS\_TERMINATION  
186 NFLAG\_INSURED\_ON\_APPROVAL  
187 SK\_ID\_PREV\_y  
188 MONTHS\_BALANCE\_y  
189 AMT\_BALANCE  
190 AMT\_CREDIT\_LIMIT\_ACTUAL  
191 AMT\_DRAWINGS\_ATM\_CURRENT  
192 AMT\_DRAWINGS\_CURRENT  
193 AMT\_DRAWINGS\_OTHER\_CURRENT  
194 AMT\_DRAWINGS\_POS\_CURRENT  
195 AMT\_INST\_MIN\_REGULARITY  
196 AMT\_PAYMENT\_CURRENT  
197 AMT\_PAYMENT\_TOTAL\_CURRENT  
198 AMT\_RECEIVABLE\_PRINCIPAL  
199 AMT\_RECVABLE  
200 AMT\_TOTAL\_RECEIVABLE  
201 CNT\_DRAWINGS\_ATM\_CURRENT  
202 CNT\_DRAWINGS\_CURRENT  
203 CNT\_DRAWINGS\_OTHER\_CURRENT  
204 CNT\_DRAWINGS\_POS\_CURRENT  
205 CNT\_INSTALMENT\_MATURE\_CUM  
206 SK\_DPD\_y  
207 SK\_DPD\_DEF\_y  
208 SK\_ID\_BUREAU  
209 DAYS\_CREDIT\_UPDATE  
210 DAYS\_CREDIT  
211 AMT\_CREDIT\_SUM  
212 MONTHS\_BALANCE  
213 DAYS\_ENDDATE\_FACT  
214 Frequency  
215 RFM  
216 R  
217 F  
218 M

## Hyper Parameter Tuning and ML Modeling

The Scikit-learn GridSearchCV function is used for hyperparameter tweaking. The base\_pipeline, the collection of hyperparameters, the number of cross-validation folds (cv), etc are all inputs for the function. The optimum hyperparameter combination that delivers the maximum performance on the validation set is then found using a grid search over the hyperparameters. The reason we choose the GridSearchCV function is that it enables us to methodically search through many combinations of hyperparameters and identify the optimum combination for our particular problem, the GridSearchCV function from Scikit-learn is a well-liked technique for hyperparameter tuning. There are frequently a lot of hyperparameters that need to be set before a machine-learning model can be trained. The ideal settings for these hyperparameters will vary depending on the particular dataset and the issue at hand. By allowing you to specify a grid of hyperparameters to search through, GridSearchCV streamlines the process of hyperparameter tuning. The combination that yields the greatest cross-validated performance is then returned after a thorough search of all conceivable hyperparameter combinations in the grid. By doing this, we can determine the ideal settings for your hyperparameters without having to manually test out various pairings and gauge how well they work. This can help you develop models more quickly and with less effort.

```
In [ ]: from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.metrics import log_loss

# Splitting the dataset into train, validation, and test sets
X_train_val, X_test, y_train_val, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)

X_train, X_val, y_train, y_val = train_test_split(
    X_train_val, y_train_val, test_size=0.25, random_state=42)
experimentLog1 = pd.DataFrame(columns=["Pipeline", "Parameters", "TrainAcc",
                                         "Train Time(s)", "Test Time(s)"])
```

## Multinomial NB

```
In [ ]: from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, mean_squared_error, accuracy_score
from sklearn.metrics import roc_auc_score, roc_curve
from matplotlib import pyplot
import matplotlib.pyplot as plt
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectKBest, chi2

Train_auc_roc_score = []
```

```

Train_Accuracy_list = []
Train_MSE_list = []
Train_F1_Score_list = []
Test_Accuracy_list = []
Test_MSE_list = []
Test_F1_Score_list = []
model_list = []
Test_auc_roc_score = []
Val_Accuracy_list = []
Val_auc_roc_score = []

parameters = {
    'clf_alpha': (1, 0.1, 0.01, 0.001, 0.0001, 0.00001), 'clf_fit_prior': (True, False)
}

base_pipeline = Pipeline([
    ('scaler', MinMaxScaler()),
    ("selector", SelectKBest(chi2)),
    ("clf", MultinomialNB()) # classifier estimator we are using
])

```

```

In [ ]: import time
print(base_pipeline.get_params().keys())
grid_search_pipeline = GridSearchCV(base_pipeline, parameters, cv=5)
grid_search_pipeline.fit(X_train, y_train)
print("")

dict_keys(['memory', 'steps', 'verbose', 'scaler', 'selector', 'clf', 'scaler_clip',
           'scaler_copy', 'scaler_feature_range', 'selector_k', 'selector_score_func',
           'clf_alpha', 'clf_class_prior', 'clf_fit_prior', 'clf_force_alpha'])

```

## Feature Importance

```

In [ ]: print("\nFeature Importance")
best_model = grid_search_pipeline.best_estimator_
selector = best_model.named_steps['selector']
importance = selector.scores_
for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()

```

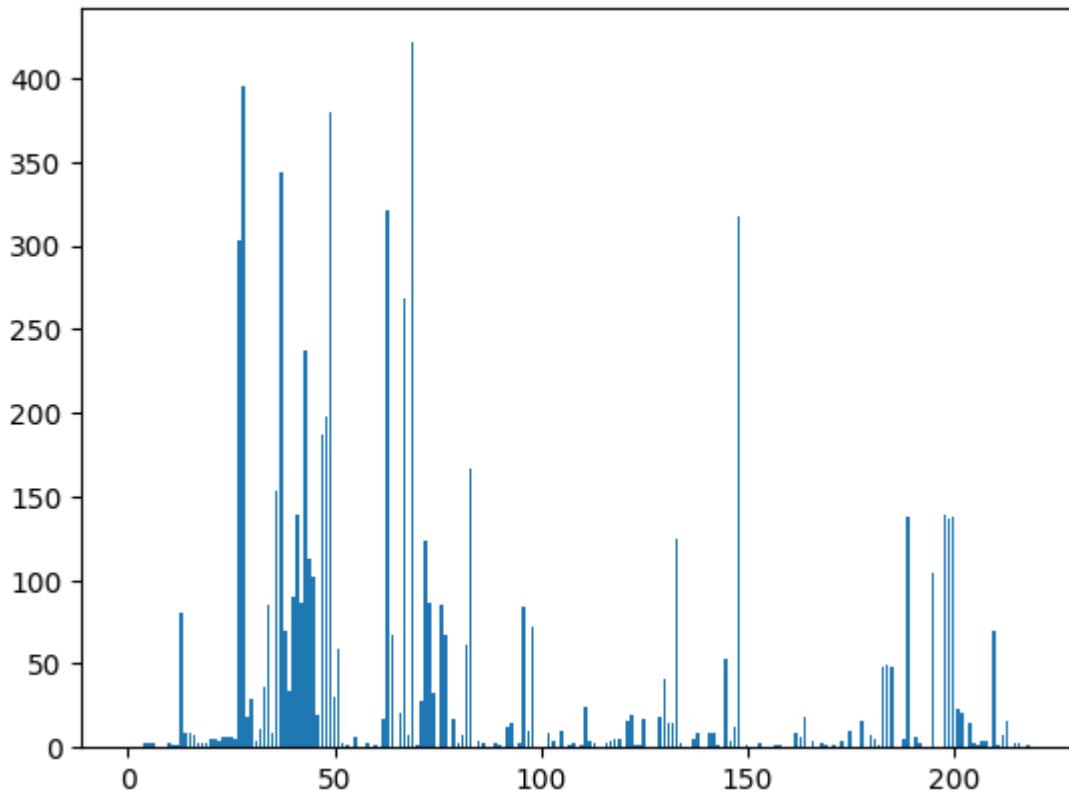
Feature Importance

Feature: 0, Score: 0.02478  
Feature: 1, Score: 0.55260  
Feature: 2, Score: 0.59274  
Feature: 3, Score: 0.61122  
Feature: 4, Score: 2.32433  
Feature: 5, Score: 2.37652  
Feature: 6, Score: 2.23136  
Feature: 7, Score: 0.14525  
Feature: 8, Score: 0.14384  
Feature: 9, Score: 0.13420  
Feature: 10, Score: 3.22707  
Feature: 11, Score: 0.98116  
Feature: 12, Score: 1.04766  
Feature: 13, Score: 79.99951  
Feature: 14, Score: 8.25148  
Feature: 15, Score: 8.19390  
Feature: 16, Score: 7.30032  
Feature: 17, Score: 2.92794  
Feature: 18, Score: 2.93798  
Feature: 19, Score: 2.54573  
Feature: 20, Score: 4.67410  
Feature: 21, Score: 4.66361  
Feature: 22, Score: 4.04883  
Feature: 23, Score: 6.54723  
Feature: 24, Score: 6.78136  
Feature: 25, Score: 6.84690  
Feature: 26, Score: 4.82648  
Feature: 27, Score: 302.74503  
Feature: 28, Score: 395.34555  
Feature: 29, Score: 17.83921  
Feature: 30, Score: 28.82445  
Feature: 31, Score: 4.34075  
Feature: 32, Score: 10.64343  
Feature: 33, Score: 36.18575  
Feature: 34, Score: 84.97622  
Feature: 35, Score: 8.92492  
Feature: 36, Score: 153.68740  
Feature: 37, Score: 343.31878  
Feature: 38, Score: 70.27735  
Feature: 39, Score: 33.34938  
Feature: 40, Score: 89.59790  
Feature: 41, Score: 138.67530  
Feature: 42, Score: 86.19473  
Feature: 43, Score: 237.08689  
Feature: 44, Score: 112.22683  
Feature: 45, Score: 102.06886  
Feature: 46, Score: 19.80500  
Feature: 47, Score: 186.66026  
Feature: 48, Score: 198.07887  
Feature: 49, Score: 379.87175  
Feature: 50, Score: 30.17973  
Feature: 51, Score: 58.67766  
Feature: 52, Score: 2.96188  
Feature: 53, Score: 1.30686  
Feature: 54, Score: 0.61812

Feature: 55, Score: 6.57692  
Feature: 56, Score: 0.28239  
Feature: 57, Score: 0.53245  
Feature: 58, Score: 2.69537  
Feature: 59, Score: 0.09215  
Feature: 60, Score: 0.96577  
Feature: 61, Score: 0.61439  
Feature: 62, Score: 17.03894  
Feature: 63, Score: 321.08423  
Feature: 64, Score: 67.30411  
Feature: 65, Score: 0.70216  
Feature: 66, Score: 20.61803  
Feature: 67, Score: 268.53286  
Feature: 68, Score: 7.25530  
Feature: 69, Score: 421.15909  
Feature: 70, Score: 1.70976  
Feature: 71, Score: 27.44226  
Feature: 72, Score: 124.06896  
Feature: 73, Score: 86.64257  
Feature: 74, Score: 32.96837  
Feature: 75, Score: 0.00728  
Feature: 76, Score: 85.12157  
Feature: 77, Score: 67.09818  
Feature: 78, Score: 0.50261  
Feature: 79, Score: 17.46049  
Feature: 80, Score: 3.09360  
Feature: 81, Score: 7.63899  
Feature: 82, Score: 61.46782  
Feature: 83, Score: 166.19381  
Feature: 84, Score: 0.08544  
Feature: 85, Score: 3.75764  
Feature: 86, Score: 2.19877  
Feature: 87, Score: 0.13774  
Feature: 88, Score: 0.06324  
Feature: 89, Score: 2.40529  
Feature: 90, Score: 1.45514  
Feature: 91, Score: 0.18018  
Feature: 92, Score: 11.80730  
Feature: 93, Score: 14.26631  
Feature: 94, Score: 0.32907  
Feature: 95, Score: 2.13323  
Feature: 96, Score: 84.15502  
Feature: 97, Score: 10.22155  
Feature: 98, Score: 72.36732  
Feature: 99, Score: 0.51523  
Feature: 100, Score: 0.06804  
Feature: 101, Score: 0.11879  
Feature: 102, Score: 9.21495  
Feature: 103, Score: 3.55076  
Feature: 104, Score: 0.12467  
Feature: 105, Score: 9.76462  
Feature: 106, Score: 0.35884  
Feature: 107, Score: 1.52971  
Feature: 108, Score: 2.95672  
Feature: 109, Score: 0.01745  
Feature: 110, Score: 1.04244

Feature: 111, Score: 23.70958  
Feature: 112, Score: 4.47310  
Feature: 113, Score: 2.79282  
Feature: 114, Score: 0.01396  
Feature: 115, Score: 0.00036  
Feature: 116, Score: 2.46084  
Feature: 117, Score: 3.96154  
Feature: 118, Score: 4.97930  
Feature: 119, Score: 5.34237  
Feature: 120, Score: 0.07568  
Feature: 121, Score: 15.47154  
Feature: 122, Score: 19.56910  
Feature: 123, Score: 1.26388  
Feature: 124, Score: 2.00610  
Feature: 125, Score: 16.76954  
Feature: 126, Score: 0.18906  
Feature: 127, Score: 0.16600  
Feature: 128, Score: 0.35670  
Feature: 129, Score: 18.37217  
Feature: 130, Score: 41.07590  
Feature: 131, Score: 14.47872  
Feature: 132, Score: 14.98150  
Feature: 133, Score: 125.01087  
Feature: 134, Score: 2.38198  
Feature: 135, Score: 0.53114  
Feature: 136, Score: 0.47043  
Feature: 137, Score: 4.82832  
Feature: 138, Score: 8.59387  
Feature: 139, Score: 0.35384  
Feature: 140, Score: 0.69356  
Feature: 141, Score: 8.43572  
Feature: 142, Score: 8.49328  
Feature: 143, Score: 1.85758  
Feature: 144, Score: 0.26471  
Feature: 145, Score: 53.25091  
Feature: 146, Score: 3.51007  
Feature: 147, Score: 12.38014  
Feature: 148, Score: 317.26537  
Feature: 149, Score: 0.00792  
Feature: 150, Score: 1.96930  
Feature: 151, Score: 0.02973  
Feature: 152, Score: 0.01719  
Feature: 153, Score: 2.25260  
Feature: 154, Score: 0.00040  
Feature: 155, Score: 0.13088  
Feature: 156, Score: 0.33594  
Feature: 157, Score: 1.15821  
Feature: 158, Score: 1.12173  
Feature: 159, Score: 0.08267  
Feature: 160, Score: 0.14903  
Feature: 161, Score: 0.00329  
Feature: 162, Score: 8.59419  
Feature: 163, Score: 6.44743  
Feature: 164, Score: 17.76019  
Feature: 165, Score: 0.10365  
Feature: 166, Score: 3.40137

Feature: 167, Score: 0.00454  
Feature: 168, Score: 2.70458  
Feature: 169, Score: 1.88175  
Feature: 170, Score: 0.84922  
Feature: 171, Score: 2.04463  
Feature: 172, Score: 0.64427  
Feature: 173, Score: 4.11532  
Feature: 174, Score: 0.00027  
Feature: 175, Score: 10.30878  
Feature: 176, Score: 0.00046  
Feature: 177, Score: 0.00001  
Feature: 178, Score: 16.44181  
Feature: 179, Score: 0.04822  
Feature: 180, Score: 7.77708  
Feature: 181, Score: 4.73869  
Feature: 182, Score: 1.29026  
Feature: 183, Score: 47.74796  
Feature: 184, Score: 49.35371  
Feature: 185, Score: 47.93896  
Feature: 186, Score: 0.20865  
Feature: 187, Score: 0.00703  
Feature: 188, Score: 5.53658  
Feature: 189, Score: 137.97564  
Feature: 190, Score: 0.01749  
Feature: 191, Score: 5.97029  
Feature: 192, Score: 3.08068  
Feature: 193, Score: 0.13461  
Feature: 194, Score: 0.10196  
Feature: 195, Score: 104.14576  
Feature: 196, Score: 0.02829  
Feature: 197, Score: 0.62914  
Feature: 198, Score: 139.06279  
Feature: 199, Score: 137.32569  
Feature: 200, Score: 137.37613  
Feature: 201, Score: 23.12037  
Feature: 202, Score: 20.50305  
Feature: 203, Score: 0.08775  
Feature: 204, Score: 14.40656  
Feature: 205, Score: 3.19486  
Feature: 206, Score: 1.04323  
Feature: 207, Score: 4.04756  
Feature: 208, Score: 4.21878  
Feature: 209, Score: 0.08497  
Feature: 210, Score: 69.89906  
Feature: 211, Score: 1.36011  
Feature: 212, Score: 7.52385  
Feature: 213, Score: 15.80405  
Feature: 214, Score: nan  
Feature: 215, Score: 2.31535  
Feature: 216, Score: 2.71028  
Feature: 217, Score: 0.00021  
Feature: 218, Score: 1.70738



## Best Params

```
In [ ]: best_params = grid_search_pipeline.best_estimator_.get_params()

print("The best parameters are: ")
for param_name in sorted(parameters.keys()):
    print("%s: %r" % (param_name, best_params[param_name]))

print("\n")
```

The best parameters are:  
clf\_alpha: 1  
clf\_fit\_prior: True

## Evaluation Metrics

```
In [ ]: # Using Validation Set
y_pred_val = grid_search_pipeline.best_estimator_.predict(X_val)
print("Val Accuracy: ", accuracy_score(y_val,y_pred_val))
Val_Accuracy = accuracy_score(y_val,y_pred_val)
y_pred_val_proba = grid_search_pipeline.best_estimator_.predict_proba(X_val)
auc_score_val = roc_auc_score(y_val, y_pred_val_proba)
print('AUC/ROC score:', auc_score_val)
Val_Accuracy_list.append(Val_Accuracy)
Val_auc_roc_score.append(auc_score_val)
print('Log loss: ', log_loss(y_val, y_pred_val))
```

```

y_pred = grid_search_pipeline.best_estimator_.predict(X_test)

y_pred_proba = grid_search_pipeline.best_estimator_.predict_proba(X_test)[:,]

# Calculate the AUC/ROC score for test set
auc_score_test = roc_auc_score(y_test, y_pred_proba)
print('AUC/ROC score:', auc_score_test)

print("Test Metrics")
print("Test Accuracy: ", accuracy_score(y_test,y_pred))
print("Test MSE: ", mean_squared_error(y_test,y_pred))
start = time.time()
Test_Accuracy = accuracy_score(y_test,y_pred)
Test_Time = time.time() - start
Test_MSE = mean_squared_error(y_test,y_pred)

print("\n")
print("Train Metrics")

y_train_pred = grid_search_pipeline.best_estimator_.predict(X_train)
print("Train Accuracy: ", accuracy_score(y_train,y_train_pred))
print("Train MSE: ", mean_squared_error(y_train,y_train_pred))
start = time.time()
Train_Accuracy = accuracy_score(y_train,y_train_pred)
Train_Time= time.time() - start
Train_MSE = mean_squared_error(y_train,y_train_pred)
y_train_proba = grid_search_pipeline.best_estimator_.predict_proba(X_train)[
auc_score_train = roc_auc_score(y_train, y_train_proba)
print("Train AUC score:", auc_score_train)

from sklearn.metrics import roc_auc_score
auc_score_val= roc_auc_score(y_val, grid_search_pipeline.predict_proba(X_val))
auc_score_test = roc_auc_score(y_test, grid_search_pipeline.predict_proba(X_)

model_list.append('NaiveBayes')
Train_auc_roc_score.append(auc_score_train)
Train_Accuracy_list.append(Train_Accuracy)
Train_MSE_list.append(Train_MSE)
Test_Accuracy_list.append(Test_Accuracy)
Test_MSE_list.append(Test_MSE)
Test_auc_roc_score.append(auc_score_test)

experimentLog1.loc[len(experimentLog1)] =[f"Baseline Multinomial NB with {X_
                                         f"Train_Accuracy*100:8.2f}%", f
                                         Train_Time, Test_Time, auc_score
experimentLog1

```

```
Val Accuracy: 0.9201626016260163
AUC/ROC score: 0.627887668076625
Log loss: 2.8776315144807354
AUC/ROC score: 0.6282303847735516
Test Metrics
Test Accuracy: 0.9182764227642276
Test MSE: 0.08172357723577235
```

```
Train Metrics
Train Accuracy: 0.9193116531165312
Train MSE: 0.08068834688346883
Train AUC score: 0.6238539588324392
```

Out[ ]:	Pipeline	Parameters	TrainAcc	ValidAcc	TestAcc	Train Time(s)	Test Time(s)	Train AUC
0	Multinomial NB with 219 inputs	{'clf__alpha': (1, 0.1, 0.01, 0.001, 0.0001, 1...}	91.93%	92.02%	91.83%	0.031847	0.009847	0.623854 0.623854

## Logistic Regression

```
In [ ]: from sklearn.linear_model import LogisticRegression
from matplotlib import pyplot
parameters = {
    'clf__solver': ['lbfgs', 'liblinear', 'newton-cg'],
    'clf__fit_intercept': [True, False]
}

base_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('clf', LogisticRegression(max_iter=1000, solver = "saga", multi_class='multinomial'))
])

print(base_pipeline.get_params().keys())
grid_search_pipeline = GridSearchCV(base_pipeline, parameters, cv=5)
grid_search_pipeline.fit(X_train, y_train)
print("")

dict_keys(['memory', 'steps', 'verbose', 'scaler', 'clf', 'scaler_copy',
    'scaler_with_mean', 'scaler_with_std', 'clf_C', 'clf_class_weight',
    'clf_dual', 'clf_fit_intercept', 'clf_intercept_scaling', 'clf_l1_ratio',
    'clf_max_iter', 'clf_multi_class', 'clf_n_jobs', 'clf_penalty',
    'clf_random_state', 'clf_solver', 'clf_tol', 'clf_verbose',
    'clf_warm_start'])
```

## Feature Importance

```
In [ ]: print("\nFeature Importance")
best_model = grid_search_pipeline.best_estimator_
```

```
importance = best_model.named_steps['clf'].coef_[0]
for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()

print("\n")
```

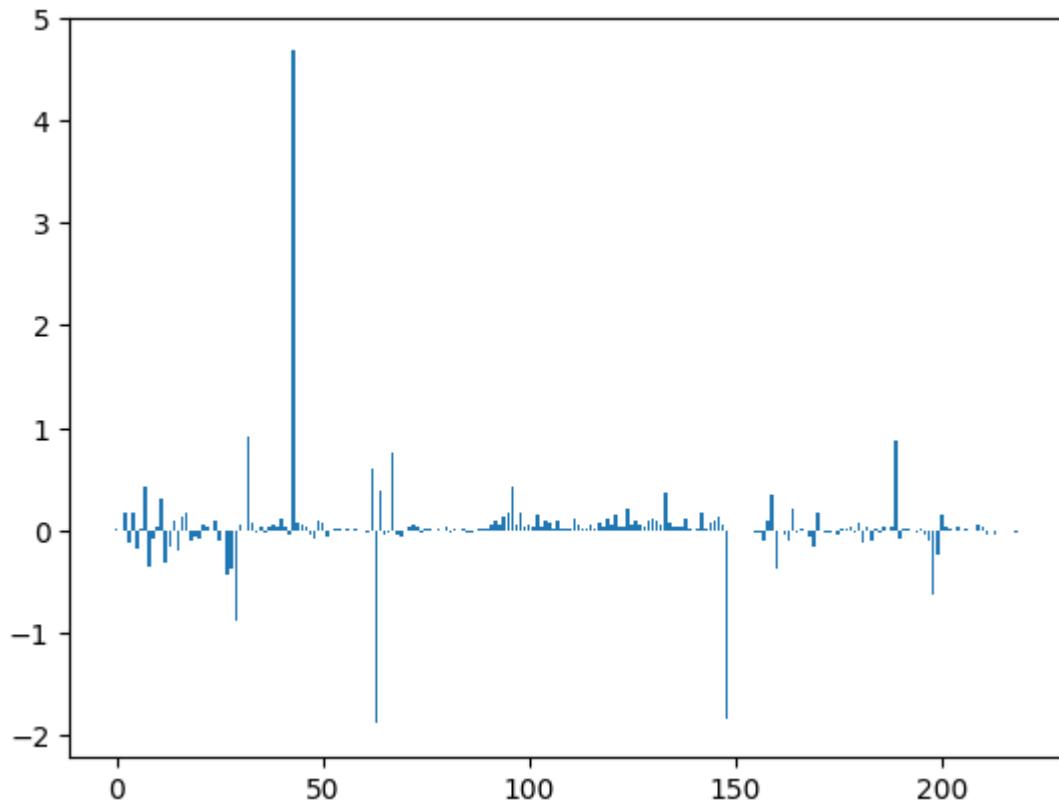
Feature Importance

Feature: 0, Score: 0.00331  
Feature: 1, Score: -0.00940  
Feature: 2, Score: 0.17297  
Feature: 3, Score: -0.13327  
Feature: 4, Score: 0.16767  
Feature: 5, Score: -0.18465  
Feature: 6, Score: 0.00443  
Feature: 7, Score: 0.43307  
Feature: 8, Score: -0.36773  
Feature: 9, Score: -0.08915  
Feature: 10, Score: 0.03494  
Feature: 11, Score: 0.30264  
Feature: 12, Score: -0.32332  
Feature: 13, Score: -0.16313  
Feature: 14, Score: 0.08574  
Feature: 15, Score: -0.19422  
Feature: 16, Score: 0.13605  
Feature: 17, Score: 0.17270  
Feature: 18, Score: -0.09733  
Feature: 19, Score: -0.07486  
Feature: 20, Score: -0.09252  
Feature: 21, Score: 0.04450  
Feature: 22, Score: 0.02693  
Feature: 23, Score: -0.01230  
Feature: 24, Score: 0.08348  
Feature: 25, Score: -0.09655  
Feature: 26, Score: -0.01403  
Feature: 27, Score: -0.43471  
Feature: 28, Score: -0.38510  
Feature: 29, Score: -0.88708  
Feature: 30, Score: 0.05728  
Feature: 31, Score: -0.00524  
Feature: 32, Score: 0.90795  
Feature: 33, Score: 0.06196  
Feature: 34, Score: -0.03201  
Feature: 35, Score: 0.04085  
Feature: 36, Score: -0.02297  
Feature: 37, Score: 0.03025  
Feature: 38, Score: 0.05434  
Feature: 39, Score: 0.02392  
Feature: 40, Score: 0.11293  
Feature: 41, Score: 0.03075  
Feature: 42, Score: -0.04466  
Feature: 43, Score: 4.67795  
Feature: 44, Score: 0.07231  
Feature: 45, Score: 0.05509  
Feature: 46, Score: 0.03705  
Feature: 47, Score: -0.03705  
Feature: 48, Score: -0.09426  
Feature: 49, Score: 0.09426  
Feature: 50, Score: 0.06196  
Feature: 51, Score: -0.06196  
Feature: 52, Score: -0.01399  
Feature: 53, Score: 0.01399  
Feature: 54, Score: 0.01062

Feature: 55, Score: 0.00116  
Feature: 56, Score: 0.00338  
Feature: 57, Score: -0.00195  
Feature: 58, Score: 0.00776  
Feature: 59, Score: -0.00957  
Feature: 60, Score: -0.00064  
Feature: 61, Score: -0.03131  
Feature: 62, Score: 0.60599  
Feature: 63, Score: -1.89373  
Feature: 64, Score: 0.37501  
Feature: 65, Score: -0.04424  
Feature: 66, Score: -0.02264  
Feature: 67, Score: 0.75762  
Feature: 68, Score: -0.05028  
Feature: 69, Score: -0.06321  
Feature: 70, Score: -0.01369  
Feature: 71, Score: 0.02874  
Feature: 72, Score: 0.06089  
Feature: 73, Score: 0.02683  
Feature: 74, Score: -0.02698  
Feature: 75, Score: 0.02168  
Feature: 76, Score: 0.00431  
Feature: 77, Score: -0.00819  
Feature: 78, Score: 0.00353  
Feature: 79, Score: -0.01025  
Feature: 80, Score: 0.02266  
Feature: 81, Score: -0.02672  
Feature: 82, Score: 0.00513  
Feature: 83, Score: 0.00270  
Feature: 84, Score: 0.00676  
Feature: 85, Score: -0.01672  
Feature: 86, Score: -0.01777  
Feature: 87, Score: -0.01580  
Feature: 88, Score: 0.00370  
Feature: 89, Score: 0.01325  
Feature: 90, Score: 0.01708  
Feature: 91, Score: 0.04382  
Feature: 92, Score: 0.09022  
Feature: 93, Score: 0.05305  
Feature: 94, Score: 0.12756  
Feature: 95, Score: 0.17400  
Feature: 96, Score: 0.42937  
Feature: 97, Score: 0.04338  
Feature: 98, Score: 0.17524  
Feature: 99, Score: 0.03663  
Feature: 100, Score: 0.05074  
Feature: 101, Score: 0.03379  
Feature: 102, Score: 0.15222  
Feature: 103, Score: 0.03452  
Feature: 104, Score: 0.08980  
Feature: 105, Score: 0.06239  
Feature: 106, Score: 0.01249  
Feature: 107, Score: 0.09173  
Feature: 108, Score: 0.02118  
Feature: 109, Score: 0.00596  
Feature: 110, Score: 0.01873

Feature: 111, Score: 0.11989  
Feature: 112, Score: 0.05607  
Feature: 113, Score: 0.01587  
Feature: 114, Score: 0.02009  
Feature: 115, Score: 0.05405  
Feature: 116, Score: 0.01692  
Feature: 117, Score: 0.06718  
Feature: 118, Score: 0.03329  
Feature: 119, Score: 0.12022  
Feature: 120, Score: 0.04392  
Feature: 121, Score: 0.15765  
Feature: 122, Score: 0.03274  
Feature: 123, Score: 0.03539  
Feature: 124, Score: 0.21279  
Feature: 125, Score: 0.04764  
Feature: 126, Score: 0.08192  
Feature: 127, Score: 0.05034  
Feature: 128, Score: 0.02380  
Feature: 129, Score: 0.08589  
Feature: 130, Score: 0.11920  
Feature: 131, Score: 0.09984  
Feature: 132, Score: 0.04367  
Feature: 133, Score: 0.36714  
Feature: 134, Score: 0.06627  
Feature: 135, Score: 0.03781  
Feature: 136, Score: 0.03294  
Feature: 137, Score: 0.03126  
Feature: 138, Score: 0.10450  
Feature: 139, Score: 0.00563  
Feature: 140, Score: 0.00172  
Feature: 141, Score: 0.01976  
Feature: 142, Score: 0.16154  
Feature: 143, Score: 0.00921  
Feature: 144, Score: 0.06537  
Feature: 145, Score: 0.09297  
Feature: 146, Score: 0.12393  
Feature: 147, Score: 0.04529  
Feature: 148, Score: -1.83780  
Feature: 149, Score: 0.00151  
Feature: 150, Score: 0.00219  
Feature: 151, Score: -0.00481  
Feature: 152, Score: 0.00183  
Feature: 153, Score: -0.00183  
Feature: 154, Score: 0.00082  
Feature: 155, Score: -0.02551  
Feature: 156, Score: -0.02393  
Feature: 157, Score: -0.10601  
Feature: 158, Score: 0.09259  
Feature: 159, Score: 0.34572  
Feature: 160, Score: -0.37218  
Feature: 161, Score: 0.00137  
Feature: 162, Score: -0.05000  
Feature: 163, Score: -0.10901  
Feature: 164, Score: 0.21590  
Feature: 165, Score: -0.01711  
Feature: 166, Score: 0.00902

Feature: 167, Score: -0.00588  
Feature: 168, Score: -0.06019  
Feature: 169, Score: -0.16680  
Feature: 170, Score: 0.17905  
Feature: 171, Score: -0.00130  
Feature: 172, Score: -0.01806  
Feature: 173, Score: -0.02906  
Feature: 174, Score: -0.00439  
Feature: 175, Score: -0.03614  
Feature: 176, Score: 0.00924  
Feature: 177, Score: 0.00567  
Feature: 178, Score: 0.04102  
Feature: 179, Score: -0.02520  
Feature: 180, Score: 0.07818  
Feature: 181, Score: -0.12960  
Feature: 182, Score: 0.03001  
Feature: 183, Score: -0.10692  
Feature: 184, Score: 0.01381  
Feature: 185, Score: -0.02319  
Feature: 186, Score: 0.02699  
Feature: 187, Score: -0.00675  
Feature: 188, Score: 0.03332  
Feature: 189, Score: 0.87997  
Feature: 190, Score: -0.08458  
Feature: 191, Score: 0.00490  
Feature: 192, Score: 0.01795  
Feature: 193, Score: -0.00056  
Feature: 194, Score: -0.02852  
Feature: 195, Score: 0.01224  
Feature: 196, Score: -0.05377  
Feature: 197, Score: -0.10760  
Feature: 198, Score: -0.63117  
Feature: 199, Score: -0.24285  
Feature: 200, Score: 0.15339  
Feature: 201, Score: 0.02897  
Feature: 202, Score: 0.00726  
Feature: 203, Score: -0.00141  
Feature: 204, Score: 0.03690  
Feature: 205, Score: -0.01489  
Feature: 206, Score: 0.01617  
Feature: 207, Score: -0.00264  
Feature: 208, Score: -0.01170  
Feature: 209, Score: 0.05747  
Feature: 210, Score: 0.02918  
Feature: 211, Score: -0.04507  
Feature: 212, Score: -0.00879  
Feature: 213, Score: -0.04225  
Feature: 214, Score: 0.00000  
Feature: 215, Score: 0.00229  
Feature: 216, Score: 0.00262  
Feature: 217, Score: -0.00176  
Feature: 218, Score: -0.01976



## Best Params

```
In [ ]: best_params = grid_search_pipeline.best_estimator_.get_params()

print("The best parameters are: ")
for param_name in sorted(parameters.keys()):
    print("%s: %r" % (param_name, best_params[param_name]))

print("\n")

The best parameters are:
clf__fit_intercept: True
clf__solver: 'lbfgs'
```

## Evaluation Metrics

```
In [ ]: # Using Validation Set
y_pred_val = grid_search_pipeline.best_estimator_.predict(X_val)
print("Val Accuracy: ", accuracy_score(y_val,y_pred_val))
Val_Accuracy = accuracy_score(y_val,y_pred_val)
y_pred_val_proba = grid_search_pipeline.best_estimator_.predict_proba(X_val)
auc_score_val = roc_auc_score(y_val, y_pred_val_proba)
print('AUC/ROC score:', auc_score_val)
Val_Accuracy_list.append(Val_Accuracy)
Val_auc_roc_score.append(auc_score_val)
print('Log loss: ', log_loss(y_val, y_pred_val))
```

```

y_pred = grid_search_pipeline.best_estimator_.predict(X_test)

y_pred_proba = grid_search_pipeline.best_estimator_.predict_proba(X_test)[:,]

# Calculate the AUC/ROC score
auc_score_test = roc_auc_score(y_test, y_pred_proba)
print('AUC/ROC score:', auc_score_test)

# Plot the ROC curve
#plot_roc_curve(grid_search_pipeline.best_estimator_, X_test, y_test)
#plt.show()

print("Test Metrics")

print("Test Accuracy: ", accuracy_score(y_test,y_pred))
print("Test MSE: ", mean_squared_error(y_test,y_pred))
Test_Accuracy = accuracy_score(y_test,y_pred)
Test_MSE = mean_squared_error(y_test,y_pred)

print("\n")
print("Train Metrics")

y_train_pred = grid_search_pipeline.best_estimator_.predict(X_train)
print("Train Accuracy: ", accuracy_score(y_train,y_train_pred))
print("Train MSE: ", mean_squared_error(y_train,y_train_pred))
start = time.time()
Train_Accuracy = accuracy_score(y_train,y_train_pred)
Train_Time= time.time() - start
Train_MSE = mean_squared_error(y_train,y_train_pred)
y_train_proba = grid_search_pipeline.best_estimator_.predict_proba(X_train)[]
auc_score_train = roc_auc_score(y_train, y_train_proba)
print("Train AUC score:", auc_score_train)

from sklearn.metrics import roc_auc_score
auc_score_val= roc_auc_score(y_val, grid_search_pipeline.predict_proba(X_val))
auc_score_test = roc_auc_score(y_test, grid_search_pipeline.predict_proba(X_)

model_list.append('LogisticRegression')

Train_auc_roc_score.append(auc_score_train)
Train_Accuracy_list.append(Train_Accuracy)
Train_MSE_list.append(Train_MSE)
Test_Accuracy_list.append(Test_Accuracy)
Test_MSE_list.append(Test_MSE)
Test_auc_roc_score.append(auc_score_test)
experimentLog1.loc[len(experimentLog1)] =[f"Baseline Logistic Regression with {Train_Accuracy*100:8.2f}%", f"Train_Time, Test_Time, auc_score"]
experimentLog1

```

```

Val Accuracy: 0.9202276422764227
AUC/ROC score: 0.7567204521464106
Log loss: 2.8752872118212807
AUC/ROC score: 0.742817707026765
Test Metrics
Test Accuracy: 0.9185365853658537
Test MSE: 0.08146341463414634

```

```

Train Metrics
Train Accuracy: 0.9193116531165312
Train MSE: 0.08068834688346883
Train AUC score: 0.7555862008922226

```

Out [ ]:	Pipeline	Parameters	TrainAcc	ValidAcc	TestAcc	Train Time(s)	Test Time(s)	Train AUC
0	Baseline Multinomial NB with 219 inputs	{'clf__alpha': (1, 0.1, 0.01, 0.001, 0.0001, 1...}	91.93%	92.02%	91.83%	0.031847	0.009847	0.623854 0.6
1	Baseline Logistic Regression with 219 inputs	{'clf__solver': ['lbfgs', 'liblinear', 'newton...']}	91.93%	92.02%	91.85%	0.030177	0.009847	0.755586 0.7

## AdaBoost Classifier

```

In [ ]: from sklearn.ensemble import AdaBoostClassifier

parameters = {"clf__n_estimators": [1, 2] }

base_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('clf', AdaBoostClassifier()) # classifier estimator you are using
])

print(base_pipeline.get_params().keys())

grid_search_pipeline = GridSearchCV(base_pipeline, parameters, cv=5)

grid_search_pipeline.fit(X_train, y_train)
print("")

dict_keys(['memory', 'steps', 'verbose', 'scaler', 'clf', 'scaler__copy',
'scaler__with_mean', 'scaler__with_std', 'clf__algorithm', 'clf__base_estimator',
'clf__estimator', 'clf__learning_rate', 'clf__n_estimators', 'clf__random_state'])

```

## Best Params

```
In [ ]: best_params = grid_search_pipeline.best_estimator_.get_params()
print("The best parameters are: ")
for param_name in sorted(parameters.keys()):
    print("%s: %r" % (param_name, best_params[param_name]))
print("\n")
```

The best parameters are:  
clf\_n\_estimators: 1

## Evaluation Metrics

```
In [ ]: # Using Validation Set
y_pred_val = grid_search_pipeline.best_estimator_.predict(X_val)
print("Val Accuracy: ", accuracy_score(y_val,y_pred_val))
Val_Accuracy = accuracy_score(y_val,y_pred_val)
y_pred_val_proba = grid_search_pipeline.best_estimator_.predict_proba(X_val)
auc_score_val = roc_auc_score(y_val, y_pred_val_proba)
print('AUC/ROC score:', auc_score_val)
Val_Accuracy_list.append(Val_Accuracy)
Val_auc_roc_score.append(auc_score_val)

y_pred = grid_search_pipeline.best_estimator_.predict(X_test)

y_pred_proba = grid_search_pipeline.best_estimator_.predict_proba(X_test) [:, 

# Calculate the AUC/ROC score
auc_score_test = roc_auc_score(y_test, y_pred_proba)
print('AUC/ROC score:', auc_score_test)

print("\n")

print("Test Metrics")

print("Test Accuracy: ", accuracy_score(y_test,y_pred))
print("Test MSE: ", mean_squared_error(y_test,y_pred))
Test_Accuracy = accuracy_score(y_test,y_pred)
Test_MSE = mean_squared_error(y_test,y_pred)

print("\n")
print("\n")
print("Train Metrics")

y_train_pred = grid_search_pipeline.best_estimator_.predict(X_train)
print("Train Accuracy: ", accuracy_score(y_train,y_train_pred))
print("Train MSE: ", mean_squared_error(y_train,y_train_pred))
start = time.time()
Train_Accuracy = accuracy_score(y_train,y_train_pred)
Train_Time= time.time() - start
Train_MSE = mean_squared_error(y_train,y_train_pred)
y_train_proba = grid_search_pipeline.best_estimator_.predict_proba(X_train) [
auc_score_train = roc_auc_score(y_train, y_train_proba)
```

```

print("Train AUC score:", auc_score_train)

from sklearn.metrics import roc_auc_score
auc_score_val = roc_auc_score(y_val, grid_search_pipeline.predict_proba(X_val))
auc_score_test = roc_auc_score(y_test, grid_search_pipeline.predict_proba(X_test))
model_list.append('AdaBoostClassifier')

Train_auc_roc_score.append(auc_score_train)
Train_Accuracy_list.append(Train_Accuracy)
Train_MSE_list.append(Train_MSE)
Test_Accuracy_list.append(Test_Accuracy)
Test_MSE_list.append(Test_MSE)
Test_auc_roc_score.append(auc_score_test)
experimentLog1.loc[len(experimentLog1)] =[f"Baseline AdaBoostClassifier with
                                         f"Train_Accuracy*100:{Train_Accuracy*100:.2f}%", f"Train_Time, Test_Time, auc_score"]
experimentLog1

```

Val Accuracy: 0.9201626016260163  
AUC/ROC score: 0.5902279734640385  
AUC/ROC score: 0.5870755670048278

Test Metrics  
Test Accuracy: 0.9182764227642276  
Test MSE: 0.08172357723577235

Train Metrics  
Train Accuracy: 0.9193116531165312  
Train MSE: 0.08068834688346883  
Train AUC score: 0.5865565426312768

Out[ ]:		Pipeline	Parameters	TrainAcc	ValidAcc	TestAcc	Train Time(s)	Test Time(s)
0	Baseline Multinomial NB with 219 inputs	{'clf__alpha': (1, 0.1, 0.01, 0.001, 0.0001, 1...}		91.93%	92.02%	91.83%	0.031847	0.009847
1	Baseline Logistic Regression with 219 inputs	{'clf__solver': ['lbfgs', 'liblinear', 'newton...']}		91.93%	92.02%	91.85%	0.030177	0.009847
2	Baseline AdaBoostClassifier with 219 inputs	{'clf__n_estimators': [1, 2]}		91.93%	92.02%	91.83%	0.031300	0.009847

# Random Forest Classifier

```
In [ ]: from sklearn.ensemble import RandomForestClassifier
parameters = {
    'rf__n_estimators':[1,10],
    'rf__min_samples_split': [2,3,4]
}
base_pipeline = Pipeline([
    ('scaler', MinMaxScaler()),
    ("rf",RandomForestClassifier()) # classifier estimator you are using
])
print(base_pipeline.get_params().keys())
grid_search_pipeline = GridSearchCV(base_pipeline, parameters, cv=5)
grid_search_pipeline.fit(X_train, y_train)
print("")

dict_keys(['memory', 'steps', 'verbose', 'scaler', 'rf', 'scaler_clip', 'scaler_copy', 'scaler_feature_range', 'rf_bootstrap', 'rf_ccp_alpha', 'rf_class_weight', 'rf_criterion', 'rf_max_depth', 'rf_max_features', 'rf_max_leaf_nodes', 'rf_max_samples', 'rf_min_impurity_decrease', 'rf_min_samples_leaf', 'rf_min_samples_split', 'rf_min_weight_fraction_leaf', 'rf_n_estimators', 'rf_n_jobs', 'rf_oob_score', 'rf_random_state', 'rf_verbose', 'rf_warm_start'])
```

## Feature Importance

```
In [ ]: print("\nFeature Importance")
best_model = grid_search_pipeline.best_estimator_
importance = best_model.named_steps['rf'].feature_importances_
for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()

print("\n")
```

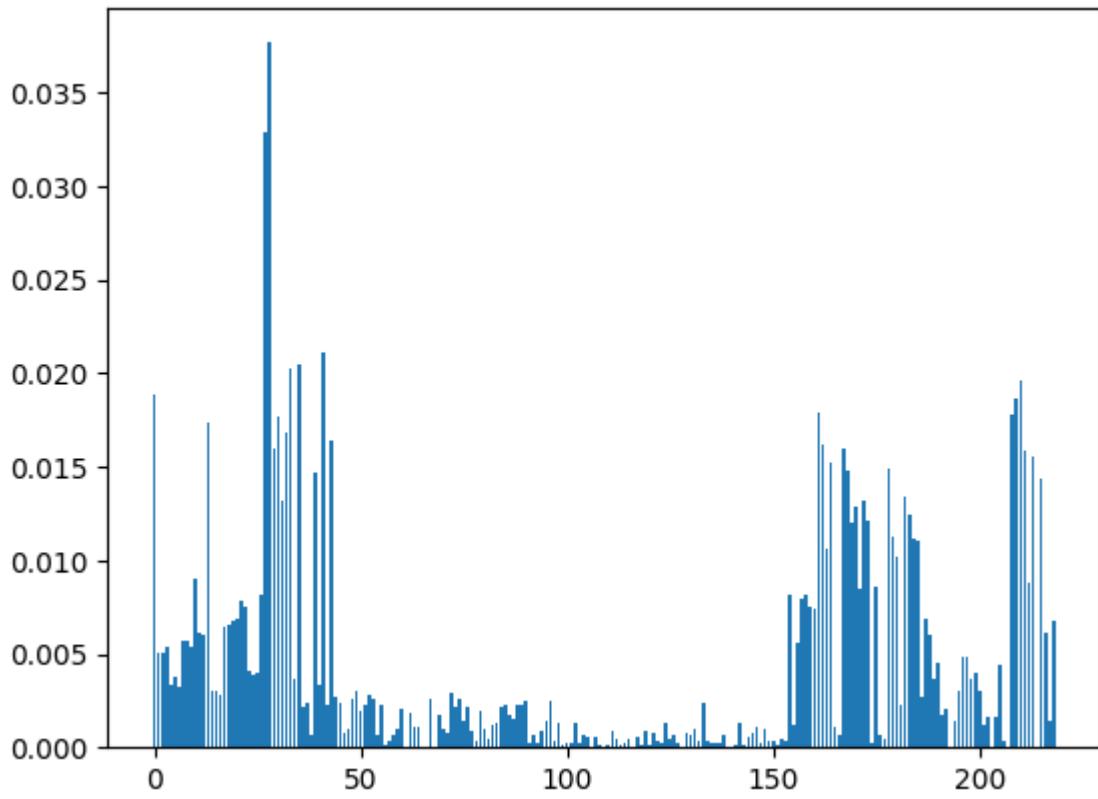
Feature Importance

Feature: 0, Score: 0.01881  
Feature: 1, Score: 0.00507  
Feature: 2, Score: 0.00509  
Feature: 3, Score: 0.00536  
Feature: 4, Score: 0.00336  
Feature: 5, Score: 0.00375  
Feature: 6, Score: 0.00323  
Feature: 7, Score: 0.00565  
Feature: 8, Score: 0.00575  
Feature: 9, Score: 0.00533  
Feature: 10, Score: 0.00898  
Feature: 11, Score: 0.00615  
Feature: 12, Score: 0.00607  
Feature: 13, Score: 0.01741  
Feature: 14, Score: 0.00299  
Feature: 15, Score: 0.00304  
Feature: 16, Score: 0.00282  
Feature: 17, Score: 0.00646  
Feature: 18, Score: 0.00659  
Feature: 19, Score: 0.00673  
Feature: 20, Score: 0.00687  
Feature: 21, Score: 0.00784  
Feature: 22, Score: 0.00754  
Feature: 23, Score: 0.00405  
Feature: 24, Score: 0.00387  
Feature: 25, Score: 0.00398  
Feature: 26, Score: 0.00819  
Feature: 27, Score: 0.03286  
Feature: 28, Score: 0.03766  
Feature: 29, Score: 0.01596  
Feature: 30, Score: 0.01769  
Feature: 31, Score: 0.01315  
Feature: 32, Score: 0.01683  
Feature: 33, Score: 0.02024  
Feature: 34, Score: 0.00371  
Feature: 35, Score: 0.02046  
Feature: 36, Score: 0.00215  
Feature: 37, Score: 0.00241  
Feature: 38, Score: 0.00065  
Feature: 39, Score: 0.01472  
Feature: 40, Score: 0.00337  
Feature: 41, Score: 0.02109  
Feature: 42, Score: 0.00231  
Feature: 43, Score: 0.01638  
Feature: 44, Score: 0.00267  
Feature: 45, Score: 0.00234  
Feature: 46, Score: 0.00083  
Feature: 47, Score: 0.00097  
Feature: 48, Score: 0.00258  
Feature: 49, Score: 0.00300  
Feature: 50, Score: 0.00200  
Feature: 51, Score: 0.00226  
Feature: 52, Score: 0.00278  
Feature: 53, Score: 0.00263  
Feature: 54, Score: 0.00070

Feature: 55, Score: 0.00228  
Feature: 56, Score: 0.00019  
Feature: 57, Score: 0.00033  
Feature: 58, Score: 0.00070  
Feature: 59, Score: 0.00098  
Feature: 60, Score: 0.00211  
Feature: 61, Score: 0.00000  
Feature: 62, Score: 0.00182  
Feature: 63, Score: 0.00106  
Feature: 64, Score: 0.00109  
Feature: 65, Score: 0.00000  
Feature: 66, Score: 0.00007  
Feature: 67, Score: 0.00261  
Feature: 68, Score: 0.00004  
Feature: 69, Score: 0.00174  
Feature: 70, Score: 0.00104  
Feature: 71, Score: 0.00078  
Feature: 72, Score: 0.00287  
Feature: 73, Score: 0.00212  
Feature: 74, Score: 0.00260  
Feature: 75, Score: 0.00144  
Feature: 76, Score: 0.00220  
Feature: 77, Score: 0.00092  
Feature: 78, Score: 0.00040  
Feature: 79, Score: 0.00194  
Feature: 80, Score: 0.00098  
Feature: 81, Score: 0.00045  
Feature: 82, Score: 0.00118  
Feature: 83, Score: 0.00135  
Feature: 84, Score: 0.00222  
Feature: 85, Score: 0.00231  
Feature: 86, Score: 0.00172  
Feature: 87, Score: 0.00152  
Feature: 88, Score: 0.00227  
Feature: 89, Score: 0.00231  
Feature: 90, Score: 0.00249  
Feature: 91, Score: 0.00024  
Feature: 92, Score: 0.00069  
Feature: 93, Score: 0.00023  
Feature: 94, Score: 0.00089  
Feature: 95, Score: 0.00138  
Feature: 96, Score: 0.00254  
Feature: 97, Score: 0.00034  
Feature: 98, Score: 0.00135  
Feature: 99, Score: 0.00016  
Feature: 100, Score: 0.00026  
Feature: 101, Score: 0.00025  
Feature: 102, Score: 0.00128  
Feature: 103, Score: 0.00030  
Feature: 104, Score: 0.00062  
Feature: 105, Score: 0.00061  
Feature: 106, Score: 0.00005  
Feature: 107, Score: 0.00060  
Feature: 108, Score: 0.00016  
Feature: 109, Score: 0.00003  
Feature: 110, Score: 0.00014

Feature: 111, Score: 0.00092  
Feature: 112, Score: 0.00043  
Feature: 113, Score: 0.00015  
Feature: 114, Score: 0.00020  
Feature: 115, Score: 0.00047  
Feature: 116, Score: 0.00003  
Feature: 117, Score: 0.00052  
Feature: 118, Score: 0.00019  
Feature: 119, Score: 0.00089  
Feature: 120, Score: 0.00017  
Feature: 121, Score: 0.00073  
Feature: 122, Score: 0.00034  
Feature: 123, Score: 0.00023  
Feature: 124, Score: 0.00132  
Feature: 125, Score: 0.00042  
Feature: 126, Score: 0.00067  
Feature: 127, Score: 0.00024  
Feature: 128, Score: 0.00005  
Feature: 129, Score: 0.00077  
Feature: 130, Score: 0.00069  
Feature: 131, Score: 0.00095  
Feature: 132, Score: 0.00033  
Feature: 133, Score: 0.00237  
Feature: 134, Score: 0.00037  
Feature: 135, Score: 0.00021  
Feature: 136, Score: 0.00025  
Feature: 137, Score: 0.00025  
Feature: 138, Score: 0.00069  
Feature: 139, Score: 0.00006  
Feature: 140, Score: 0.00002  
Feature: 141, Score: 0.00015  
Feature: 142, Score: 0.00129  
Feature: 143, Score: 0.00014  
Feature: 144, Score: 0.00057  
Feature: 145, Score: 0.00077  
Feature: 146, Score: 0.00114  
Feature: 147, Score: 0.00025  
Feature: 148, Score: 0.00103  
Feature: 149, Score: 0.00039  
Feature: 150, Score: 0.00040  
Feature: 151, Score: 0.00018  
Feature: 152, Score: 0.00043  
Feature: 153, Score: 0.00040  
Feature: 154, Score: 0.00821  
Feature: 155, Score: 0.00120  
Feature: 156, Score: 0.00558  
Feature: 157, Score: 0.00790  
Feature: 158, Score: 0.00814  
Feature: 159, Score: 0.00751  
Feature: 160, Score: 0.00743  
Feature: 161, Score: 0.01785  
Feature: 162, Score: 0.01618  
Feature: 163, Score: 0.01062  
Feature: 164, Score: 0.01520  
Feature: 165, Score: 0.00111  
Feature: 166, Score: 0.00070

Feature: 167, Score: 0.01592  
Feature: 168, Score: 0.01480  
Feature: 169, Score: 0.01198  
Feature: 170, Score: 0.01288  
Feature: 171, Score: 0.00848  
Feature: 172, Score: 0.01318  
Feature: 173, Score: 0.01212  
Feature: 174, Score: 0.00022  
Feature: 175, Score: 0.00854  
Feature: 176, Score: 0.00066  
Feature: 177, Score: 0.00049  
Feature: 178, Score: 0.01494  
Feature: 179, Score: 0.01131  
Feature: 180, Score: 0.01021  
Feature: 181, Score: 0.00229  
Feature: 182, Score: 0.01338  
Feature: 183, Score: 0.01244  
Feature: 184, Score: 0.01111  
Feature: 185, Score: 0.01105  
Feature: 186, Score: 0.00271  
Feature: 187, Score: 0.00691  
Feature: 188, Score: 0.00599  
Feature: 189, Score: 0.00362  
Feature: 190, Score: 0.00456  
Feature: 191, Score: 0.00174  
Feature: 192, Score: 0.00205  
Feature: 193, Score: 0.00007  
Feature: 194, Score: 0.00140  
Feature: 195, Score: 0.00301  
Feature: 196, Score: 0.00489  
Feature: 197, Score: 0.00487  
Feature: 198, Score: 0.00364  
Feature: 199, Score: 0.00398  
Feature: 200, Score: 0.00298  
Feature: 201, Score: 0.00126  
Feature: 202, Score: 0.00165  
Feature: 203, Score: 0.00001  
Feature: 204, Score: 0.00162  
Feature: 205, Score: 0.00440  
Feature: 206, Score: 0.00040  
Feature: 207, Score: 0.00003  
Feature: 208, Score: 0.01781  
Feature: 209, Score: 0.01860  
Feature: 210, Score: 0.01963  
Feature: 211, Score: 0.01586  
Feature: 212, Score: 0.00875  
Feature: 213, Score: 0.01557  
Feature: 214, Score: 0.00000  
Feature: 215, Score: 0.01440  
Feature: 216, Score: 0.00618  
Feature: 217, Score: 0.00143  
Feature: 218, Score: 0.00673



## Best Params

```
In [ ]: best_params = grid_search_pipeline.best_estimator_.get_params()
print("The best parameters are: ")
for param_name in sorted(parameters.keys()):
    print("%s: %r" % (param_name, best_params[param_name]))

print("\n")
```

The best parameters are:  
rf\_min\_samples\_split: 2  
rf\_n\_estimators: 10

## Evaluation Metrics

```
In [ ]: # Using Validation Set
y_pred_val = grid_search_pipeline.best_estimator_.predict(X_val)
print("Val Accuracy: ", accuracy_score(y_val,y_pred_val))
Val_Accuracy = accuracy_score(y_val,y_pred_val)
y_pred_val_proba = grid_search_pipeline.best_estimator_.predict_proba(X_val)
auc_score_val = roc_auc_score(y_val, y_pred_val_proba)
print('AUC/ROC score:', auc_score_val)
Val_Accuracy_list.append(Val_Accuracy)
Val_auc_roc_score.append(auc_score_val)
print('Log loss: ', log_loss(y_val, y_pred_val))
```

```

y_pred = grid_search_pipeline.best_estimator_.predict(X_test)

y_pred_proba = grid_search_pipeline.best_estimator_.predict_proba(X_test)[:,]

# Calculate the AUC/ROC score
auc_score_test = roc_auc_score(y_test, y_pred_proba)
print('AUC/ROC score:', auc_score_test)

# Plot the ROC curve
#plot_roc_curve(grid_search_pipeline.best_estimator_, X_test, y_test)
#plt.show()

print("\n")

print("Test Metrics")
print("Test Accuracy: ", accuracy_score(y_test,y_pred))
print("Test MSE: ", mean_squared_error(y_test,y_pred))
Test_Accuracy = accuracy_score(y_test,y_pred)
Test_MSE = mean_squared_error(y_test,y_pred)

print("\n")
print("Train Metrics")

y_train_pred = grid_search_pipeline.best_estimator_.predict(X_train)
print("Train Accuracy: ", accuracy_score(y_train,y_train_pred))
print("Train MSE: ", mean_squared_error(y_train,y_train_pred))
start = time.time()
Train_Accuracy = accuracy_score(y_train,y_train_pred)
Train_Time= time.time() - start
Train_MSE = mean_squared_error(y_train,y_train_pred)
y_train_proba = grid_search_pipeline.best_estimator_.predict_proba(X_train)[]
auc_score_train = roc_auc_score(y_train, y_train_proba)
print("Train AUC score:", auc_score_train)

from sklearn.metrics import roc_auc_score
auc_score_val= roc_auc_score(y_val, grid_search_pipeline.predict_proba(X_val))
auc_score_test = roc_auc_score(y_test, grid_search_pipeline.predict_proba(X_)

model_list.append('RandomForest')

Train_auc_roc_score.append(auc_score_train)
Train_Accuracy_list.append(Train_Accuracy)
Train_MSE_list.append(Train_MSE)
Test_Accuracy_list.append(Test_Accuracy)
Test_MSE_list.append(Test_MSE)
Test_auc_roc_score.append(auc_score_test)

experimentLog1.loc[len(experimentLog1)] =[f"Baseline Random Forest with {X_t
                                         f" {Train_Accuracy*100:8.2f}%", f
                                         Train_Time, Test_Time, auc_score
experimentLog1

```

```
Val Accuracy: 0.9196260162601626
AUC/ROC score: 0.6427599314611226
Log loss: 2.896972011421237
AUC/ROC score: 0.6328504550358041
```

```
Test Metrics
Test Accuracy: 0.9173821138211382
Test MSE: 0.08261788617886179
```

```
Train Metrics
Train Accuracy: 0.9855121951219512
Train MSE: 0.014487804878048781
Train AUC score: 0.999818485827756
```

Out[ ]:	Pipeline	Parameters	TrainAcc	ValidAcc	TestAcc	Train Time(s)	Test Time(s)
0	Baseline Multinomial NB with 219 inputs	{'clf__alpha': (1, 0.1, 0.01, 0.001, 0.0001, 1...}	91.93%	92.02%	91.83%	0.031847	0.009847
1	Baseline Logistic Regression with 219 inputs	{'clf__solver': ['lbfgs', 'liblinear', 'newton...']}	91.93%	92.02%	91.85%	0.030177	0.009847
2	Baseline AdaBoostClassifier with 219 inputs	{'clf__n_estimators': [1, 2]}	91.93%	92.02%	91.83%	0.031300	0.009847
3	Baseline Random Forest with 219 inputs	{'rf__n_estimators': [1, 10], 'rf__min_samples...}	98.55%	91.96%	91.74%	0.034335	0.009847

## KNN

```
In [ ]: from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, mean_squared_error, roc_auc_score

# Define the parameter grid to search over
param_grid = {
    'knn__n_neighbors': [3],
    'knn__weights': ['uniform'],
    'knn__p': [1, 2],
```

```

}

# Define the base pipeline with StandardScaler and KNeighborsClassifier
base_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('knn', KNeighborsClassifier())
])

# Perform GridSearchCV to find the best parameters
grid_search_pipeline = GridSearchCV(base_pipeline, param_grid)
grid_search_pipeline.fit(X_train, y_train)
print("")

```

## Best Params

```
In [ ]: # Print the best parameters found
best_params = grid_search_pipeline.best_estimator_.get_params()
print("The best parameters are: ")
for param_name in sorted(param_grid.keys()):
    print("%s: %r" % (param_name, grid_search_pipeline.best_params_[param_name]))
```

The best parameters are:

```
knn_n_neighbors: 3
knn_p: 2
knn_weights: 'uniform'
```

## Evaluation Metrics

```
In [ ]: # Use the best estimator to predict on the validation and test sets
y_pred_val = grid_search_pipeline.predict(X_val)
y_pred_val_proba = grid_search_pipeline.predict_proba(X_val)[:, 1]
auc_score_val = roc_auc_score(y_val, y_pred_val_proba)
val_accuracy = accuracy_score(y_val, y_pred_val)
print("Val Accuracy: ", val_accuracy)
print('AUC/ROC score:', auc_score_val)
print('Log loss: ', log_loss(y_val, y_pred_val))

y_pred_test = grid_search_pipeline.predict(X_test)
y_pred_test_proba = grid_search_pipeline.predict_proba(X_test)[:, 1]
auc_score_test = roc_auc_score(y_test, y_pred_test_proba)
test_accuracy = accuracy_score(y_test, y_pred_test)
print("\nTest Metrics")
print("Test Accuracy: ", test_accuracy)
print("Test MSE: ", mean_squared_error(y_test, y_pred_test))
print('AUC/ROC score:', auc_score_test)

y_pred_train = grid_search_pipeline.predict(X_train)
train_accuracy = accuracy_score(y_train, y_pred_train)
train_mse = mean_squared_error(y_train, y_pred_train)
print("\nTrain Metrics")
y_train_pred = grid_search_pipeline.best_estimator_.predict(X_train)
print("Train Accuracy: ", accuracy_score(y_train, y_train_pred))
print("Train MSE: ", mean_squared_error(y_train, y_train_pred))
```

```

start = time.time()
Train_Accuracy = accuracy_score(y_train,y_train_pred)
Train_Time= time.time() - start
Train_MSE = mean_squared_error(y_train,y_train_pred)
y_train_proba = grid_search_pipeline.best_estimator_.predict_proba(X_train)[
auc_score_train = roc_auc_score(y_train, y_train_proba)
print("Train AUC score:", auc_score_train)

from sklearn.metrics import roc_auc_score
auc_score_val= roc_auc_score(y_val, grid_search_pipeline.predict_proba(X_val))
auc_score_test = roc_auc_score(y_test, grid_search_pipeline.predict_proba(X_test))

model_list.append('KNN')

Train_auc_roc_score.append(auc_score_train)
Train_Accuracy_list.append(Train_Accuracy)
Train_MSE_list.append(Train_MSE)
Test_Accuracy_list.append(Test_Accuracy)
Test_MSE_list.append(Test_MSE)
Test_auc_roc_score.append(auc_score_test)

experimentLog1.loc[len(experimentLog1)] =[f"Baseline KNN with {X_train.shape[1]} features",
                                         f"Train Accuracy:{Train_Accuracy*100:.2f}%", f"Train Time: {Train_Time}, Test Time: {Test_Time}, auc score: {auc_score_test}"]
experimentLog1

```

Val Accuracy: 0.9045040650406504  
AUC/ROC score: 0.5516391297102932  
Log loss: 3.4420223797444716

Test Metrics  
Test Accuracy: 0.9029756097560976  
Test MSE: 0.09702439024390244  
AUC/ROC score: 0.5505512197852465

Train Metrics  
Train Accuracy: 0.9297235772357724  
Train MSE: 0.07027642276422764  
Train AUC score: 0.9391061495212746

Out[ ]:

	Pipeline	Parameters	TrainAcc	ValidAcc	TestAcc	Train Time(s)	Test Time(s)
0	Baseline Multinomial NB with 219 inputs	{'clf__alpha': (1, 0.1, 0.01, 0.001, 0.0001, 1...}	91.93%	92.02%	91.83%	0.031847	0.009847
1	Baseline Logistic Regression with 219 inputs	{'clf__solver': ['lbfgs', 'liblinear', 'newton...']}	91.93%	92.02%	91.85%	0.030177	0.009847
2	Baseline AdaBoostClassifier with 219 inputs	{'clf__n_estimators': [1, 2]}	91.93%	92.02%	91.83%	0.031300	0.009847
3	Baseline Random Forest with 219 inputs	{'rf__n_estimators': [1, 10], 'rf__min_samples...}	98.55%	91.96%	91.74%	0.034335	0.009847
4	Baseline KNN with 219 inputs	{'rf__n_estimators': [1, 10], 'rf__min_samples...}	92.97%	91.96%	91.74%	0.020069	0.009847

## Experiment 4: Using Feature Engineering (With 17 New Features)

### Hyper Parameter Tuning and ML Modeling

The Scikit-learn GridSearchCV function is used for hyperparameter tweaking. The base\_pipeline, the collection of hyperparameters, the number of cross-validation folds (cv), etc are all inputs for the function. The optimum hyperparameter combination that delivers the maximum performance on the validation set is then found using a grid search over the hyperparameters. The reason we choose the GridSearchCV function is that it enables us to methodically search through many combinations of hyperparameters and identify the optimum combination for our particular problem, the GridSearchCV function from Scikit-learn is a well-liked technique for hyperparameter tuning. There are frequently a lot of hyperparameters that need to be set before a machine-learning model can be trained. The ideal settings for these hyperparameters will vary depending on the particular dataset and the issue at hand. By allowing you to specify a grid of hyperparameters to search through, GridSearchCV streamlines the process of hyperparameter tuning. The combination that yields the greatest cross-

validated performance is then returned after a thorough search of all conceivable hyperparameter combinations in the grid. By doing this, we can determine the ideal settings for your hyperparameters without having to manually test out various pairings and gauge how well they work. This can help you develop models more quickly and with less effort.

```
In [ ]: from sklearn.model_selection import train_test_split
from sklearn.preprocessing import MinMaxScaler
from sklearn.impute import SimpleImputer
from sklearn.preprocessing import StandardScaler, OneHotEncoder
from sklearn.compose import ColumnTransformer
from sklearn.metrics import log_loss

# Splitting the dataset into train, validation, and test sets
X_train_val, X_test, y_train_val, y_test = train_test_split(
    X, y, test_size=0.2, random_state=42)

X_train, X_val, y_train, y_val = train_test_split(
    X_train_val, y_train_val, test_size=0.25, random_state=42)
```

Dataframe to log the experiments

```
In [ ]: experimentLog2 = pd.DataFrame(columns=["Pipeline", "Parameters", "TrainAcc",
                                             "Train Time(s)", "Test Time(
```

## Multinomial NB

```
In [ ]: from sklearn.model_selection import GridSearchCV
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import classification_report, mean_squared_error, accuracy_score
from sklearn.metrics import roc_auc_score, roc_curve
from matplotlib import pyplot
import matplotlib.pyplot as plt
from sklearn.pipeline import Pipeline
from sklearn.feature_selection import SelectKBest, chi2

Train_auc_roc_score = []
Train_Accuracy_list = []
Train_MSE_list = []
Train_F1_Score_list = []
Test_Accuracy_list = []
Test_MSE_list = []
Test_F1_Score_list = []
model_list = []
Test_auc_roc_score = []
Val_Accuracy_list = []
Val_auc_roc_score = []

parameters = {
    'clf_alpha': (1, 0.1, 0.01, 0.001, 0.0001, 0.00001), 'clf_fit_prior': (True,
```

```
base_pipeline = Pipeline([
    ('scaler', MinMaxScaler()),
    ('selector', SelectKBest(chi2)),
    ('clf', MultinomialNB()) # classifier estimator we are using
])
```

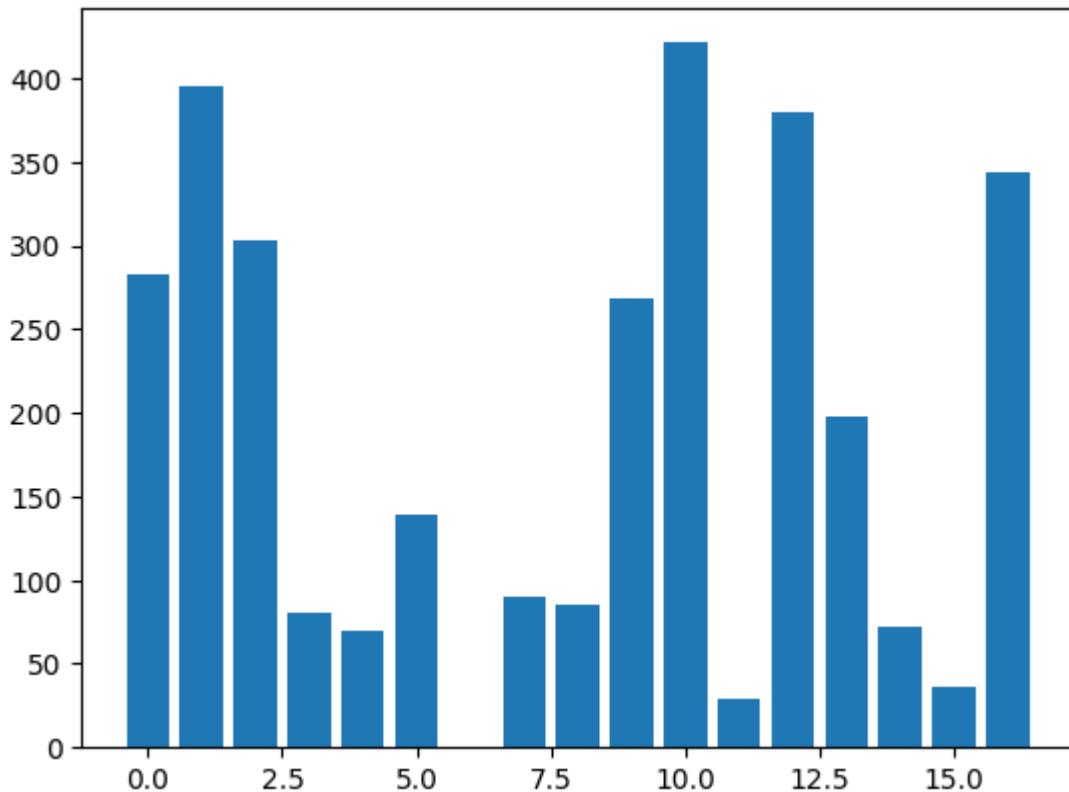
```
In [ ]: import time
print(base_pipeline.get_params().keys())
grid_search_pipeline = GridSearchCV(base_pipeline, parameters, cv=5)
grid_search_pipeline.fit(X_train, y_train)
print("")

dict_keys(['memory', 'steps', 'verbose', 'scaler', 'selector', 'clf', 'scaler_clip',
'scaler_copy', 'scaler_feature_range', 'selector_k', 'selector_score_func',
'clf_alpha', 'clf_class_prior', 'clf_fit_prior', 'clf_force_alpha'])
```

## Feature Importance

```
In [ ]: print("\nFeature Importance")
best_model = grid_search_pipeline.best_estimator_
selector = best_model.named_steps['selector']
importance = selector.scores_
for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()
```

```
Feature Importance
Feature: 0, Score: 282.45633
Feature: 1, Score: 395.34555
Feature: 2, Score: 302.74503
Feature: 3, Score: 79.99951
Feature: 4, Score: 69.89906
Feature: 5, Score: 138.67530
Feature: 6, Score: 0.08497
Feature: 7, Score: 89.59790
Feature: 8, Score: 84.97622
Feature: 9, Score: 268.53286
Feature: 10, Score: 421.15909
Feature: 11, Score: 28.82445
Feature: 12, Score: 379.87175
Feature: 13, Score: 198.07887
Feature: 14, Score: 71.79064
Feature: 15, Score: 36.18575
Feature: 16, Score: 343.31878
```



## Best Params

```
In [ ]: best_params = grid_search_pipeline.best_estimator_.get_params()

print("The best parameters are: ")
for param_name in sorted(parameters.keys()):
    print("%s: %r" % (param_name, best_params[param_name]))

print("\n")
```

The best parameters are:

```
clf_alpha: 1
clf_fit_prior: True
```

## Evaluation Metrics

```
In [ ]: # Using Validation Set
y_pred_val = grid_search_pipeline.best_estimator_.predict(X_val)
print("Val Accuracy: ", accuracy_score(y_val,y_pred_val))
Val_Accuracy = accuracy_score(y_val,y_pred_val)
y_pred_val_proba = grid_search_pipeline.best_estimator_.predict_proba(X_val)
auc_score_val = roc_auc_score(y_val, y_pred_val_proba)
print('AUC/ROC score:', auc_score_val)
Val_Accuracy_list.append(Val_Accuracy)
Val_auc_roc_score.append(auc_score_val)
print('Log loss: ', log_loss(y_val, y_pred_val))
```

```

y_pred = grid_search_pipeline.best_estimator_.predict(X_test)

y_pred_proba = grid_search_pipeline.best_estimator_.predict_proba(X_test)[:,]

# Calculate the AUC/ROC score for test set
auc_score_test = roc_auc_score(y_test, y_pred_proba)
print('AUC/ROC score:', auc_score_test)

print("Test Metrics")
print("Test Accuracy: ", accuracy_score(y_test,y_pred))
print("Test MSE: ", mean_squared_error(y_test,y_pred))
start = time.time()
Test_Accuracy = accuracy_score(y_test,y_pred)
Test_Time = time.time() - start
Test_MSE = mean_squared_error(y_test,y_pred)

print("\n")
print("Train Metrics")

y_train_pred = grid_search_pipeline.best_estimator_.predict(X_train)
print("Train Accuracy: ", accuracy_score(y_train,y_train_pred))
print("Train MSE: ", mean_squared_error(y_train,y_train_pred))
start = time.time()
Train_Accuracy = accuracy_score(y_train,y_train_pred)
Train_Time= time.time() - start
Train_MSE = mean_squared_error(y_train,y_train_pred)
y_train_proba = grid_search_pipeline.best_estimator_.predict_proba(X_train)[
auc_score_train = roc_auc_score(y_train, y_train_proba)
print("Train AUC score:", auc_score_train)

from sklearn.metrics import roc_auc_score
auc_score_val= roc_auc_score(y_val, grid_search_pipeline.predict_proba(X_val)
auc_score_test = roc_auc_score(y_test, grid_search_pipeline.predict_proba(X_)

model_list.append('NaiveBayes')
Train_auc_roc_score.append(auc_score_train)
Train_Accuracy_list.append(Train_Accuracy)
Train_MSE_list.append(Train_MSE)
Test_Accuracy_list.append(Test_Accuracy)
Test_MSE_list.append(Test_MSE)
Test_auc_roc_score.append(auc_score_test)

experimentLog2.loc[len(experimentLog2)] =[f"Baseline Multinomial NB with {X_
                                         f"Train_Accuracy*100:8.2f}%", 
                                         Train_Time, Test_Time, auc_score
experimentLog2

```

```
Val Accuracy: 0.9201626016260163
AUC/ROC score: 0.6558790478120213
Log loss: 2.8776315144807354
AUC/ROC score: 0.6591604698173175
Test Metrics
Test Accuracy: 0.9182764227642276
Test MSE: 0.08172357723577235
```

```
Train Metrics
Train Accuracy: 0.9193116531165312
Train MSE: 0.08068834688346883
Train AUC score: 0.6529611313955381
```

Out [ ]:	Pipeline	Parameters	TrainAcc	ValidAcc	TestAcc	Train Time(s)	Test Time(s)	Train AUC
0	Baseline	{'clf__alpha': 1, 0.1, 0.01, 0.001, 0.0001, 1...}						
0	Multinomial NB with 17 inputs	{'clf__alpha': 1, 0.1, 0.01, 0.001, 0.0001, 1...}	91.93%	92.02%	91.83%	0.056123	0.010725	0.652961 0.65

## Logistic Regression

```
In [ ]: from sklearn.linear_model import LogisticRegression
from matplotlib import pyplot
parameters = {
    'clf__solver': ['lbfgs', 'liblinear', 'newton-cg'],
    'clf__fit_intercept': [True, False]
}

base_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('clf', LogisticRegression(max_iter=1000, solver = "saga", multi_class='multinomial'))
])

print(base_pipeline.get_params().keys())
grid_search_pipeline = GridSearchCV(base_pipeline, parameters, cv=5)
grid_search_pipeline.fit(X_train, y_train)
print("")

dict_keys(['memory', 'steps', 'verbose', 'scaler', 'clf', 'scaler_copy',
'scaler_with_mean', 'scaler_with_std', 'clf_C', 'clf_class_weight', 'clf_dual',
'clf_fit_intercept', 'clf_intercept_scaling', 'clf_l1_ratio',
'clf_max_iter', 'clf_multi_class', 'clf_n_jobs', 'clf_penalty', 'clf_random_state',
'clf_solver', 'clf_tol', 'clf_verbose', 'clf_warm_start'])
```

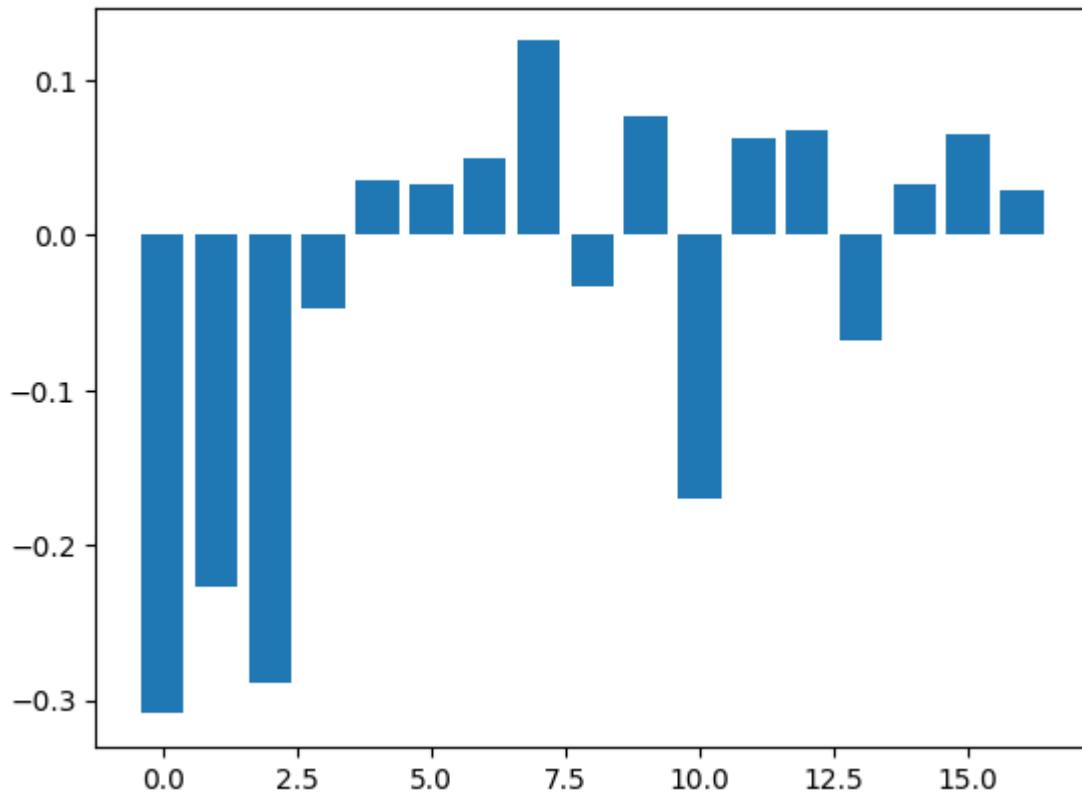
```
/usr/local/lib/python3.9/dist-packages/scipy/optimize/_linesearch.py:306: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.9/dist-packages/sklearn/utils/optimize.py:203: UserWarning: Line Search failed
  warnings.warn("Line Search failed")
/usr/local/lib/python3.9/dist-packages/scipy/optimize/_linesearch.py:306: LineSearchWarning: The line search algorithm did not converge
  warn('The line search algorithm did not converge', LineSearchWarning)
/usr/local/lib/python3.9/dist-packages/sklearn/utils/optimize.py:203: UserWarning: Line Search failed
  warnings.warn("Line Search failed")
```

## Feature Importance

```
In [ ]: print("\nFeature Importance")
best_model = grid_search_pipeline.best_estimator_
importance = best_model.named_steps['clf'].coef_[0]
for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()

print("\n")
```

```
Feature Importance
Feature: 0, Score: -0.30880
Feature: 1, Score: -0.22670
Feature: 2, Score: -0.28906
Feature: 3, Score: -0.04683
Feature: 4, Score: 0.03486
Feature: 5, Score: 0.03318
Feature: 6, Score: 0.04944
Feature: 7, Score: 0.12502
Feature: 8, Score: -0.03291
Feature: 9, Score: 0.07608
Feature: 10, Score: -0.16953
Feature: 11, Score: 0.06201
Feature: 12, Score: 0.06766
Feature: 13, Score: -0.06766
Feature: 14, Score: 0.03277
Feature: 15, Score: 0.06470
Feature: 16, Score: 0.02889
```



## Best Params

```
In [ ]: best_params = grid_search_pipeline.best_estimator_.get_params()

print("The best parameters are: ")
for param_name in sorted(parameters.keys()):
    print("%s: %r" % (param_name, best_params[param_name]))

print("\n")
```

The best parameters are:  
clf\_fit\_intercept: True  
clf\_solver: 'lbfgs'

## Evaluation Metrics

```
In [ ]: # Using Validation Set
y_pred_val = grid_search_pipeline.best_estimator_.predict(X_val)
print("Val Accuracy: ", accuracy_score(y_val,y_pred_val))
Val_Accuracy = accuracy_score(y_val,y_pred_val)
y_pred_val_proba = grid_search_pipeline.best_estimator_.predict_proba(X_val)
auc_score_val = roc_auc_score(y_val, y_pred_val_proba)
print('AUC/ROC score:', auc_score_val)
Val_Accuracy_list.append(Val_Accuracy)
Val_auc_roc_score.append(auc_score_val)
print('Log loss: ',log_loss(y_val, y_pred_val))
```

```

y_pred = grid_search_pipeline.best_estimator_.predict(X_test)

y_pred_proba = grid_search_pipeline.best_estimator_.predict_proba(X_test)[:,]

# Calculate the AUC/ROC score
auc_score_test = roc_auc_score(y_test, y_pred_proba)
print('AUC/ROC score:', auc_score_test)

# Plot the ROC curve
#plot_roc_curve(grid_search_pipeline.best_estimator_, X_test, y_test)
#plt.show()

print("Test Metrics")

print("Test Accuracy: ", accuracy_score(y_test,y_pred))
print("Test MSE: ", mean_squared_error(y_test,y_pred))
Test_Accuracy = accuracy_score(y_test,y_pred)
Test_MSE = mean_squared_error(y_test,y_pred)

print("\n")
print("Train Metrics")

y_train_pred = grid_search_pipeline.best_estimator_.predict(X_train)
print("Train Accuracy: ", accuracy_score(y_train,y_train_pred))
print("Train MSE: ", mean_squared_error(y_train,y_train_pred))
start = time.time()
Train_Accuracy = accuracy_score(y_train,y_train_pred)
Train_Time= time.time() - start
Train_MSE = mean_squared_error(y_train,y_train_pred)
y_train_proba = grid_search_pipeline.best_estimator_.predict_proba(X_train)[]
auc_score_train = roc_auc_score(y_train, y_train_proba)
print("Train AUC score:", auc_score_train)

from sklearn.metrics import roc_auc_score
auc_score_val= roc_auc_score(y_val, grid_search_pipeline.predict_proba(X_val))
auc_score_test = roc_auc_score(y_test, grid_search_pipeline.predict_proba(X_)

model_list.append('LogisticRegression')

Train_auc_roc_score.append(auc_score_train)
Train_Accuracy_list.append(Train_Accuracy)
Train_MSE_list.append(Train_MSE)
Test_Accuracy_list.append(Test_Accuracy)
Test_MSE_list.append(Test_MSE)
Test_auc_roc_score.append(auc_score_test)
experimentLog2.loc[len(experimentLog2)] =[f"Baseline Logistic Regression with {Train_Accuracy*100:8.2f}%", f"Train_Time, Test_Time, auc_score"]
experimentLog2

```

```

Val Accuracy: 0.9200813008130081
AUC/ROC score: 0.7344481709829773
Log loss: 2.880561892805054
AUC/ROC score: 0.7283819361898431
Test Metrics
Test Accuracy: 0.9181788617886179
Test MSE: 0.08182113821138211

```

```

Train Metrics
Train Accuracy: 0.919149051490515
Train MSE: 0.0808509485094851
Train AUC score: 0.7315336195277534

```

Out [ ]:	Pipeline	Parameters	TrainAcc	ValidAcc	TestAcc	Train Time(s)	Test Time(s)	Train AUC
0	Baseline Multinomial NB with 17 inputs	{'clf__alpha': (1, 0.1, 0.01, 0.001, 0.0001, 1...}	91.93%	92.02%	91.83%	0.056123	0.010725	0.652961 0.652961
1	Baseline Logistic Regression with 17 inputs	{'clf__solver': ['lbfgs', 'liblinear', 'newton...']}	91.91%	92.01%	91.82%	0.025727	0.010725	0.731534 0.731534

## AdaBoost Classifier

```

In [ ]: from sklearn.ensemble import AdaBoostClassifier

parameters = {"clf__n_estimators": [1, 2] }

base_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('clf', AdaBoostClassifier()) # classifier estimator you are using
])

print(base_pipeline.get_params().keys())

grid_search_pipeline = GridSearchCV(base_pipeline, parameters, cv=5)

grid_search_pipeline.fit(X_train, y_train)
print("")

dict_keys(['memory', 'steps', 'verbose', 'scaler', 'clf', 'scaler_copy',
'scaler_with_mean', 'scaler_with_std', 'clf_algorithm', 'clf_base_estimator',
'clf_estimator', 'clf_learning_rate', 'clf_n_estimators', 'clf_random_state'])

```

## Best Params

```
In [ ]: best_params = grid_search_pipeline.best_estimator_.get_params()
print("The best parameters are: ")
for param_name in sorted(parameters.keys()):
    print("%s: %r" % (param_name, best_params[param_name]))
print("\n")
```

The best parameters are:  
clf\_n\_estimators: 1

## Evaluation Metrics

```
In [ ]: # Using Validation Set
y_pred_val = grid_search_pipeline.best_estimator_.predict(X_val)
print("Val Accuracy: ", accuracy_score(y_val,y_pred_val))
Val_Accuracy = accuracy_score(y_val,y_pred_val)
y_pred_val_proba = grid_search_pipeline.best_estimator_.predict_proba(X_val)
auc_score_val = roc_auc_score(y_val, y_pred_val_proba)
print('AUC/ROC score:', auc_score_val)
Val_Accuracy_list.append(Val_Accuracy)
Val_auc_roc_score.append(auc_score_val)

y_pred = grid_search_pipeline.best_estimator_.predict(X_test)

y_pred_proba = grid_search_pipeline.best_estimator_.predict_proba(X_test) [:, 

# Calculate the AUC/ROC score
auc_score_test = roc_auc_score(y_test, y_pred_proba)
print('AUC/ROC score:', auc_score_test)

print("\n")

print("Test Metrics")

print("Test Accuracy: ", accuracy_score(y_test,y_pred))
print("Test MSE: ", mean_squared_error(y_test,y_pred))
Test_Accuracy = accuracy_score(y_test,y_pred)
Test_MSE = mean_squared_error(y_test,y_pred)

print("\n")
print("\n")
print("Train Metrics")

y_train_pred = grid_search_pipeline.best_estimator_.predict(X_train)
print("Train Accuracy: ", accuracy_score(y_train,y_train_pred))
print("Train MSE: ", mean_squared_error(y_train,y_train_pred))
start = time.time()
Train_Accuracy = accuracy_score(y_train,y_train_pred)
Train_Time= time.time() - start
Train_MSE = mean_squared_error(y_train,y_train_pred)
y_train_proba = grid_search_pipeline.best_estimator_.predict_proba(X_train) [
auc_score_train = roc_auc_score(y_train, y_train_proba)
```

```

print("Train AUC score:", auc_score_train)

from sklearn.metrics import roc_auc_score
auc_score_val = roc_auc_score(y_val, grid_search_pipeline.predict_proba(X_val))
auc_score_test = roc_auc_score(y_test, grid_search_pipeline.predict_proba(X_test))
model_list.append('AdaBoostClassifier')

Train_auc_roc_score.append(auc_score_train)
Train_Accuracy_list.append(Train_Accuracy)
Train_MSE_list.append(Train_MSE)
Test_Accuracy_list.append(Test_Accuracy)
Test_MSE_list.append(Test_MSE)
Test_auc_roc_score.append(auc_score_test)
experimentLog2.loc[len(experimentLog2)] =[f"Baseline AdaBoostClassifier with {Train_Accuracy*100:8.2f}%", f"Train_Time, Test_Time, auc_score"]
experimentLog2

```

Val Accuracy: 0.9201626016260163  
AUC/ROC score: 0.6449556588301388  
AUC/ROC score: 0.6393625936150891

Test Metrics  
Test Accuracy: 0.9182764227642276  
Test MSE: 0.08172357723577235

Train Metrics  
Train Accuracy: 0.9193116531165312  
Train MSE: 0.08068834688346883  
Train AUC score: 0.6458036710553374

Out[ ]:	Pipeline	Parameters	TrainAcc	ValidAcc	TestAcc	Train Time(s)	Test Time(s)
0	Baseline Multinomial NB with 17 inputs	{'clf__alpha': (1, 0.1, 0.01, 0.001, 0.0001, 1...}	91.93%	92.02%	91.83%	0.056123	0.010725
1	Baseline Logistic Regression with 17 inputs	{'clf__solver': ['lbfgs', 'liblinear', 'newton...']}	91.91%	92.01%	91.82%	0.025727	0.010725
2	Baseline AdaBoostClassifier with 17 inputs	{'clf__n_estimators': [1, 2]}	91.93%	92.02%	91.83%	0.019794	0.010725

# Random Forest Classifier

```
In [ ]: from sklearn.ensemble import RandomForestClassifier
parameters = {
    'rf__n_estimators':[1,10],
    'rf__min_samples_split': [2,3,4]
}
base_pipeline = Pipeline([
    ('scaler', MinMaxScaler()),
    ("rf",RandomForestClassifier()) # classifier estimator you are using
])
print(base_pipeline.get_params().keys())
grid_search_pipeline = GridSearchCV(base_pipeline, parameters, cv=5)
grid_search_pipeline.fit(X_train, y_train)
print("")

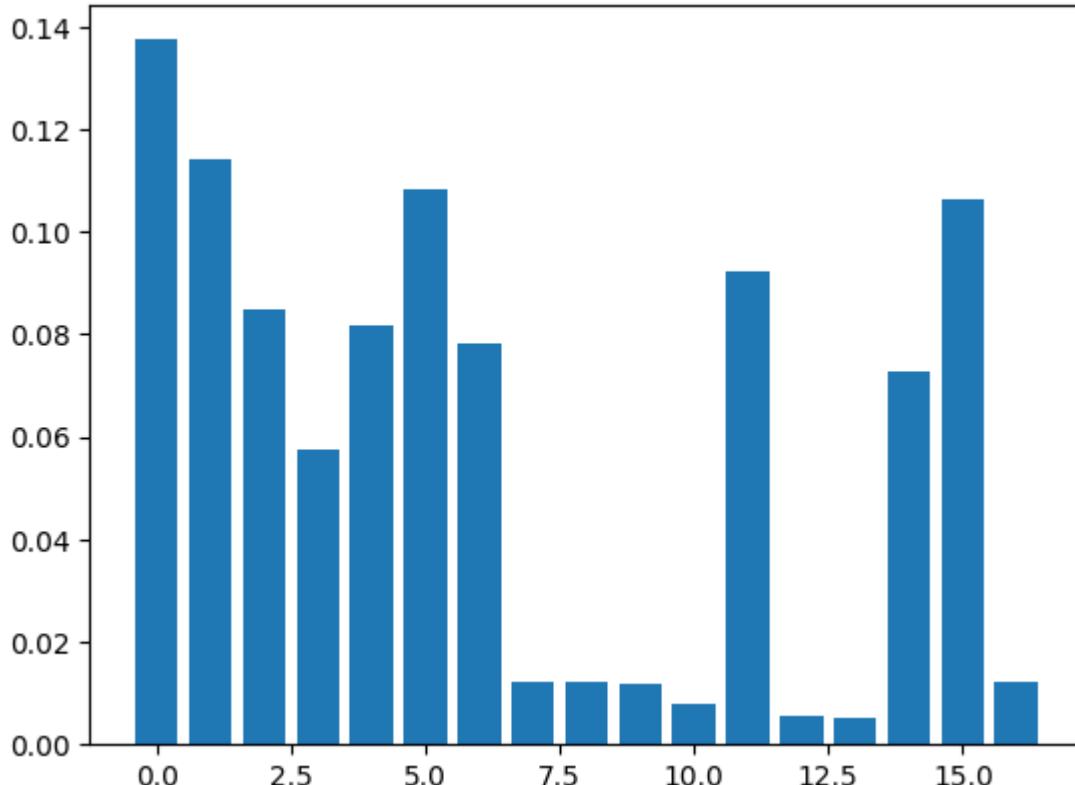
dict_keys(['memory', 'steps', 'verbose', 'scaler', 'rf', 'scaler_clip', 'scaler_copy', 'scaler_feature_range', 'rf_bootstrap', 'rf_ccp_alpha', 'rf_class_weight', 'rf_criterion', 'rf_max_depth', 'rf_max_features', 'rf_max_leaf_nodes', 'rf_max_samples', 'rf_min_impurity_decrease', 'rf_min_samples_leaf', 'rf_min_samples_split', 'rf_min_weight_fraction_leaf', 'rf_n_estimators', 'rf_n_jobs', 'rf_oob_score', 'rf_random_state', 'rf_verbose', 'rf_warm_start'])
```

## Feature Importance

```
In [ ]: print("\nFeature Importance")
best_model = grid_search_pipeline.best_estimator_
importance = best_model.named_steps['rf'].feature_importances_
for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i,v))
# plot feature importance
pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()

print("\n")
```

```
Feature Importance
Feature: 0, Score: 0.13747
Feature: 1, Score: 0.11414
Feature: 2, Score: 0.08470
Feature: 3, Score: 0.05738
Feature: 4, Score: 0.08155
Feature: 5, Score: 0.10820
Feature: 6, Score: 0.07802
Feature: 7, Score: 0.01224
Feature: 8, Score: 0.01237
Feature: 9, Score: 0.01182
Feature: 10, Score: 0.00774
Feature: 11, Score: 0.09220
Feature: 12, Score: 0.00551
Feature: 13, Score: 0.00515
Feature: 14, Score: 0.07271
Feature: 15, Score: 0.10646
Feature: 16, Score: 0.01233
```



## Best Params

```
In [ ]: best_params = grid_search_pipeline.best_estimator_.get_params()
print("The best parameters are: ")
for param_name in sorted(parameters.keys()):
    print("%s: %r" % (param_name, best_params[param_name]))

print("\n")
```

```
The best parameters are:  
rf_min_samples_split: 2  
rf_n_estimators: 10
```

## Evaluation Metrics

```
In [ ]: # Using Validation Set  
y_pred_val = grid_search_pipeline.best_estimator_.predict(X_val)  
print("Val Accuracy: ", accuracy_score(y_val,y_pred_val))  
Val_Accuracy = accuracy_score(y_val,y_pred_val)  
y_pred_val_proba = grid_search_pipeline.best_estimator_.predict_proba(X_val)  
auc_score_val = roc_auc_score(y_val, y_pred_val_proba)  
print('AUC/ROC score:', auc_score_val)  
Val_Accuracy_list.append(Val_Accuracy)  
Val_auc_roc_score.append(auc_score_val)  
print('Log loss: ',log_loss(y_val, y_pred_val))  
  
y_pred = grid_search_pipeline.best_estimator_.predict(X_test)  
  
y_pred_proba = grid_search_pipeline.best_estimator_.predict_proba(X_test)[:,  
  
# Calculate the AUC/ROC score  
auc_score_test = roc_auc_score(y_test, y_pred_proba)  
print('AUC/ROC score:', auc_score_test)  
  
# Plot the ROC curve  
#plot_roc_curve(grid_search_pipeline.best_estimator_, X_test, y_test)  
#plt.show()  
  
print("\n")  
  
print("Test Metrics")  
print("Test Accuracy: ", accuracy_score(y_test,y_pred))  
print("Test MSE: ", mean_squared_error(y_test,y_pred))  
Test_Accuracy = accuracy_score(y_test,y_pred)  
Test_MSE = mean_squared_error(y_test,y_pred)  
  
print("\n")  
print("Train Metrics")  
  
y_train_pred = grid_search_pipeline.best_estimator_.predict(X_train)  
print("Train Accuracy: ", accuracy_score(y_train,y_train_pred))  
print("Train MSE: ", mean_squared_error(y_train,y_train_pred))  
start = time.time()  
Train_Accuracy = accuracy_score(y_train,y_train_pred)  
Train_Time= time.time() - start  
Train_MSE = mean_squared_error(y_train,y_train_pred)  
y_train_proba = grid_search_pipeline.best_estimator_.predict_proba(X_train)[  
auc_score_train = roc_auc_score(y_train, y_train_proba)  
print("Train AUC score:", auc_score_train)
```

```
from sklearn.metrics import roc_auc_score
auc_score_val= roc_auc_score(y_val, grid_search_pipeline.predict_proba(X_val)
auc_score_test = roc_auc_score(y_test, grid_search_pipeline.predict_proba(X_
model_list.append('RandomForest')

Train_auc_roc_score.append(auc_score_train)
Train_Accuracy_list.append(Train_Accuracy)
Train_MSE_list.append(Train_MSE)
Test_Accuracy_list.append(Test_Accuracy)
Test_MSE_list.append(Test_MSE)
Test_auc_roc_score.append(auc_score_test)

experimentLog2.loc[len(experimentLog2)] =[f"Baseline Random Forest with {X_t
                                         f"Train_Accuracy*100:8.2f}%", f
                                         Train_Time, Test_Time, auc_score
experimentLog2
```

Val Accuracy: 0.9188130081300813  
AUC/ROC score: 0.6462053704622776  
Log loss: 2.9262757946644213  
AUC/ROC score: 0.6420933242263648

Test Metrics  
Test Accuracy: 0.9156585365853659  
Test MSE: 0.08434146341463415

Train Metrics  
Train Accuracy: 0.9856368563685637  
Train MSE: 0.014363143631436315  
Train AUC score: 0.9997181024155245

Out [ ]:

	Pipeline	Parameters	TrainAcc	ValidAcc	TestAcc	Train Time(s)	Test Time(s)
0	Baseline Multinomial NB with 17 inputs	{'clf__alpha': (1, 0.1, 0.01, 0.001, 0.0001, 1...}	91.93%	92.02%	91.83%	0.056123	0.010725
1	Baseline Logistic Regression with 17 inputs	{'clf__solver': ['lbfgs', 'liblinear', 'newton...']}	91.91%	92.01%	91.82%	0.025727	0.010725
2	Baseline AdaBoostClassifier with 17 inputs	{'clf__n_estimators': [1, 2]}	91.93%	92.02%	91.83%	0.019794	0.010725
3	Baseline Random Forest with 17 inputs	{'rf__n_estimators': [1, 10], 'rf__min_samples...}	98.56%	91.88%	91.57%	0.021411	0.010725

## KNN

In [ ]:

```

from sklearn.neighbors import KNeighborsClassifier
from sklearn.pipeline import Pipeline
from sklearn.preprocessing import StandardScaler
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score, mean_squared_error, roc_auc_score

# Define the parameter grid to search over
param_grid = {
    'knn__n_neighbors': [2],
    'knn__weights': ['uniform'],
    'knn__p': [1, 2],
}

# Define the base pipeline with StandardScaler and KNeighborsClassifier
base_pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('knn', KNeighborsClassifier())
])

# Perform GridSearchCV to find the best parameters
grid_search_pipeline = GridSearchCV(base_pipeline, param_grid)
grid_search_pipeline.fit(X_train, y_train)
print("")

```

## Best Params

```
In [ ]: # Print the best parameters found
best_params = grid_search_pipeline.best_estimator_.get_params()
print("The best parameters are: ")
for param_name in sorted(param_grid.keys()):
    print("%s: %r" % (param_name, grid_search_pipeline.best_params_[param_na
The best parameters are:
knn_n_neighbors: 2
knn_p: 2
knn_weights: 'uniform'
```

## Evaluation Metrics

```
In [ ]: # Use the best estimator to predict on the validation and test sets
y_pred_val = grid_search_pipeline.predict(X_val)
y_pred_val_proba = grid_search_pipeline.predict_proba(X_val)[:, 1]
auc_score_val = roc_auc_score(y_val, y_pred_val_proba)
val_accuracy = accuracy_score(y_val, y_pred_val)
print("Val Accuracy: ", val_accuracy)
print('AUC/ROC score:', auc_score_val)
print('Log loss: ', log_loss(y_val, y_pred_val))

y_pred_test = grid_search_pipeline.predict(X_test)
y_pred_test_proba = grid_search_pipeline.predict_proba(X_test)[:, 1]
auc_score_test = roc_auc_score(y_test, y_pred_test_proba)
test_accuracy = accuracy_score(y_test, y_pred_test)
print("\nTest Metrics")
print("Test Accuracy: ", test_accuracy)
print("Test MSE: ", mean_squared_error(y_test, y_pred_test))
print('AUC/ROC score:', auc_score_test)

y_pred_train = grid_search_pipeline.predict(X_train)
train_accuracy = accuracy_score(y_train, y_pred_train)
train_mse = mean_squared_error(y_train, y_pred_train)
print("\nTrain Metrics")
y_train_pred = grid_search_pipeline.best_estimator_.predict(X_train)
print("Train Accuracy: ", accuracy_score(y_train, y_train_pred))
print("Train MSE: ", mean_squared_error(y_train, y_train_pred))
start = time.time()
Train_Accuracy = accuracy_score(y_train, y_train_pred)
Train_Time = time.time() - start
Train_MSE = mean_squared_error(y_train, y_train_pred)
y_train_proba = grid_search_pipeline.best_estimator_.predict_proba(X_train)[
auc_score_train = roc_auc_score(y_train, y_train_proba)
print("Train AUC score:", auc_score_train)

from sklearn.metrics import roc_auc_score
auc_score_val = roc_auc_score(y_val, grid_search_pipeline.predict_proba(X_val))
auc_score_test = roc_auc_score(y_test, grid_search_pipeline.predict_proba(X_
model_list.append('KNN'))

Train_auc_roc_score.append(auc_score_train)
```

```

Train_Accuracy_list.append(Train_Accuracy)
Train_MSE_list.append(Train_MSE)
Test_Accuracy_list.append(Test_Accuracy)
Test_MSE_list.append(Test_MSE)
Test_auc_roc_score.append(auc_score_test)

experimentLog2.loc[len(experimentLog2)] =[f"Baseline KNN with {X_train.shape
                                            f" {Train_Accuracy*100:8.2f}%", f
                                            Train_Time, Test_Time, auc_score
experimentLog2

```

Val Accuracy: 0.9131869918699187  
AUC/ROC score: 0.5566684001728948  
Log loss: 3.1290579747072598

Test Metrics  
Test Accuracy: 0.9121788617886178  
Test MSE: 0.08782113821138211  
AUC/ROC score: 0.5656492708151701

Train Metrics  
Train Accuracy: 0.9305474254742547  
Train MSE: 0.06945257452574526  
Train AUC score: 0.9676837134571202

Out[ ]:

	Pipeline	Parameters	TrainAcc	ValidAcc	TestAcc	Train Time(s)	Test Time(s)
0	Baseline Multinomial NB with 17 inputs	{'clf__alpha': (1, 0.1, 0.01, 0.001, 0.0001, 1...}	91.93%	92.02%	91.83%	0.056123	0.010725
1	Baseline Logistic Regression with 17 inputs	{'clf__solver': ['lbfgs', 'liblinear', 'newton...']}	91.91%	92.01%	91.82%	0.025727	0.010725
2	Baseline AdaBoostClassifier with 17 inputs	{'clf__n_estimators': [1, 2]}	91.93%	92.02%	91.83%	0.019794	0.010725
3	Baseline Random Forest with 17 inputs	{'rf__n_estimators': [1, 10], 'rf__min_samples...}	98.56%	91.88%	91.57%	0.021411	0.010725
4	Baseline KNN with 17 inputs	{'rf__n_estimators': [1, 10], 'rf__min_samples...}	93.05%	91.88%	91.57%	0.019411	0.010725

# Ensemble Methods

```
In [ ]: from sklearn.pipeline import Pipeline
from sklearn.ensemble import RandomForestClassifier
from sklearn.ensemble import VotingClassifier
from sklearn.linear_model import LogisticRegression
from sklearn.preprocessing import StandardScaler

# Define base classifiers
clf1 = LogisticRegression(max_iter=1000, solver='saga', multi_class='ovr')
clf2 = RandomForestClassifier(n_estimators=100, random_state=42)

# Define the ensemble method
ensemble_clf = VotingClassifier(estimators=[('lr', clf1), ('rf', clf2)], vot

# Define the pipeline
pipeline = Pipeline([
    ('scaler', StandardScaler()),
    ('clf', ensemble_clf)
])

# Define the hyperparameters to tune
parameters = {
    'clf_lr_C': [0.01],
    'clf_lr_penalty': ['l1'],
    'clf_rf_max_depth': [10],
    'clf_rf_min_samples_split': [2],
    'clf_voting': ['soft']
}

# Create grid search object
grid_search = GridSearchCV(pipeline, parameters, cv=5, scoring='accuracy')

# Fit the grid search object to the data
grid_search.fit(X_train, y_train)
print("")
```

```
In [ ]: from sklearn.inspection import permutation_importance

print("\nFeature Importance")
best_model = grid_search.best_estimator_

# Compute permutation feature importance
result = permutation_importance(best_model, X_test, y_test, n_repeats=10, random_state=42)
importance = result.importances_mean

# Print feature importance
for i,v in enumerate(importance):
    print('Feature: %0d, Score: %.5f' % (i,v))

pyplot.bar([x for x in range(len(importance))], importance)
pyplot.show()
```

```
In [ ]: # Use the best estimator to predict on the validation and test sets
y_pred_val = grid_search.predict(X_val)
y_pred_val_proba = grid_search.predict_proba(X_val)[:, 1]
auc_score_val = roc_auc_score(y_val, y_pred_val_proba)
val_accuracy = accuracy_score(y_val, y_pred_val)
print("Val Accuracy: ", val_accuracy)
print('AUC/ROC score:', auc_score_val)
print('Log loss: ', log_loss(y_val, y_pred_val))

y_pred_test = grid_search.predict(X_test)
y_pred_test_proba = grid_search.predict_proba(X_test)[:, 1]
auc_score_test = roc_auc_score(y_test, y_pred_test_proba)
test_accuracy = accuracy_score(y_test, y_pred_test)
print("\nTest Metrics")
print("Test Accuracy: ", test_accuracy)
print("Test MSE: ", mean_squared_error(y_test,y_pred_test))
print('AUC/ROC score:', auc_score_test)

y_pred_train = grid_search.predict(X_train)
train_accuracy = accuracy_score(y_train, y_pred_train)
train_mse = mean_squared_error(y_train,y_pred_train)
print("\nTrain Metrics")
y_train_pred = grid_search.best_estimator_.predict(X_train)
print("Train Accuracy: ", accuracy_score(y_train,y_train_pred))
print("Train MSE: ", mean_squared_error(y_train,y_train_pred))
start = time.time()
Train_Accuracy = accuracy_score(y_train,y_train_pred)
Train_Time= time.time() - start
Train_MSE = mean_squared_error(y_train,y_train_pred)
y_train_proba = grid_search.best_estimator_.predict_proba(X_train)[:, 1]
auc_score_train = roc_auc_score(y_train, y_train_proba)
print("Train AUC score:", auc_score_train)

from sklearn.metrics import roc_auc_score
auc_score_val= roc_auc_score(y_val, grid_search_pipeline.predict_proba(X_val))
auc_score_test = roc_auc_score(y_test, grid_search_pipeline.predict_proba(X_test))

model_list.append('Ensemble')

Train_auc_roc_score.append(auc_score_train)
Train_Accuracy_list.append(Train_Accuracy)
Train_MSE_list.append(Train_MSE)
Test_Accuracy_list.append(Test_Accuracy)
Test_MSE_list.append(Test_MSE)
Test_auc_roc_score.append(auc_score_test)

experimentLog2.loc[len(experimentLog2)] =[f"Baseline Ensemble with {X_train}.
f" {Train_Accuracy*100:8.2f}%", f
Train_Time, Test_Time, auc_score

experimentLog2
```

```
Val Accuracy: 0.9202113821138211
AUC/ROC score: 0.7374699998452442
Log loss: 2.875873287486144
```

```
Test Metrics
Test Accuracy: 0.9183252032520325
Test MSE: 0.08167479674796747
AUC/ROC score: 0.7315316095228916
```

```
Train Metrics
Train Accuracy: 0.9197235772357724
Train MSE: 0.08027642276422764
Train AUC score: 0.7734654925001723
```

Out [ ]:

	Pipeline	Parameters	TrainAcc	ValidAcc	TestAcc	Train Time(s)	Test Time(s)
0	Baseline Multinomial NB with 17 inputs	{'clf__alpha': (1, 0.1, 0.01, 0.001, 0.0001, 1...}	91.93%	92.02%	91.83%	0.056123	0.010725
1	Baseline Logistic Regression with 17 inputs	{'clf__solver': ['lbfgs', 'liblinear', 'newton...']}	91.91%	92.01%	91.82%	0.025727	0.010725
2	Baseline AdaBoostClassifier with 17 inputs	{'clf__n_estimators': [1, 2]}	91.93%	92.02%	91.83%	0.019794	0.010725
3	Baseline Random Forest with 17 inputs	{'rf__n_estimators': [1, 10], 'rf__min_samples...}	98.56%	91.88%	91.57%	0.021411	0.010725
4	Baseline KNN with 17 inputs	{'rf__n_estimators': [1, 10], 'rf__min_samples...}	93.05%	91.88%	91.57%	0.019411	0.010725
5	Baseline Ensemble with 17 inputs	{'clf__lr__C': [0.01], 'clf__lr__penalty': ['l...']}	91.97%	91.88%	91.57%	0.017676	0.010725

In [ ]:

```
# Print the best parameters and the validation score
print("Best parameters found: ", grid_search.best_params_)
print("Validation accuracy: ", grid_search.best_score_)

# Evaluate the model on the test set
y_pred = grid_search.predict(X_test)
```

```

print("Test accuracy: ", accuracy_score(y_test, y_pred))
experimentLog2.loc[len(experimentLog2)] =[f"Baseline KNN with {X_train.shape[0]}{Train_Accuracy*100:8.2f}%", f"Train_Time, Test_Time, auc_score"]
experimentLog2

```

Best parameters found: {'clf\_lr\_C': 0.01, 'clf\_lr\_penalty': 'l1', 'clf\_rf\_max\_depth': 10, 'clf\_rf\_min\_samples\_split': 2, 'clf\_voting': 'soft'}

Validation accuracy: 0.919289972899729

Test accuracy: 0.9183252032520325

Out[ ]:

	Pipeline	Parameters	TrainAcc	ValidAcc	TestAcc	Train Time(s)	Test Time(s)
0	Baseline Multinomial NB with 17 inputs	{'clf_alpha': (1, 0.1, 0.01, 0.001, 0.0001, 1...}	91.93%	92.02%	91.83%	0.056123	0.010725
1	Baseline Logistic Regression with 17 inputs	{'clf_solver': ['lbfgs', 'liblinear', 'newton...']}	91.91%	92.01%	91.82%	0.025727	0.010725
2	Baseline AdaBoostClassifier with 17 inputs	{'clf_n_estimators': [1, 2]}	91.93%	92.02%	91.83%	0.019794	0.010725
3	Baseline Random Forest with 17 inputs	{'rf_n_estimators': [1, 10], 'rf_min_samples...}	98.56%	91.88%	91.57%	0.021411	0.010725
4	Baseline KNN with 17 inputs	{'rf_n_estimators': [1, 10], 'rf_min_samples...}	93.05%	91.88%	91.57%	0.019411	0.010725
5	Baseline Ensemble with 17 inputs	{'clf_lr_C': [0.01], 'clf_lr_penalty': ['l1', 'l2']}	91.97%	91.88%	91.57%	0.017676	0.010725
6	Baseline KNN with 17 inputs	{'clf_lr_C': [0.01], 'clf_lr_penalty': ['l1', 'l2']}	91.97%	91.88%	91.57%	0.017676	0.010725

## Experimental Results

### Experiment 1 Using Selected Features (Baseline Model)

In [ ]: exp\_log

Out[ ]:

Pipeline

Parameter

0	Baseline Pipeline(steps=[('rf', RandomForestClassifier())]) with 100 inputs	{'rf__n_estimators': [20], 'rf__criterion': ['gini', 'log_loss']}
---	--	--

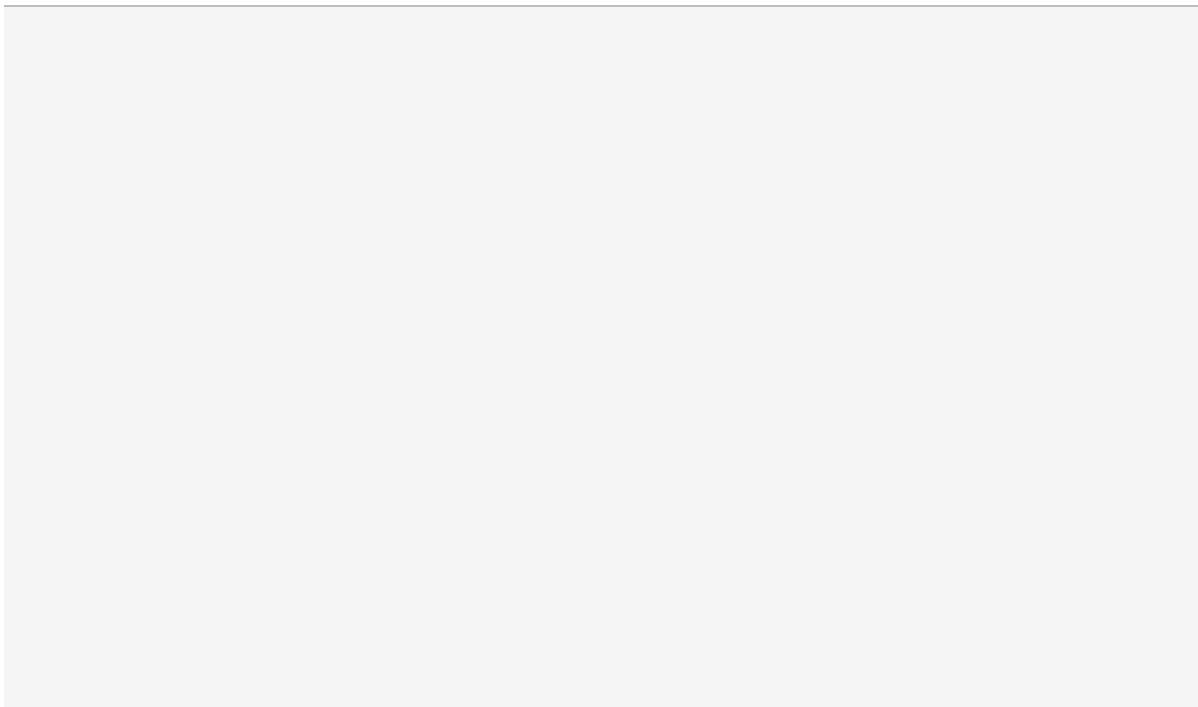
1	Baseline Pipeline(steps=[('dt', DecisionTreeClassifier())]) with 100 inputs	{'dt__max_depth': [5, 10]}
---	--	----------------------------

2	Baseline Pipeline(steps=[('lr', LogisticRegression())]) with 100 inputs	{'lr__C': [0.01], 'lr__penalty': ['l1', 'l2']}
---	--	---

3	Baseline Pipeline(steps=[('ada', AdaBoostClassifier(base_estimator=DecisionTreeClassifier()))]) with 100 inputs	{'ada__n_estimators': [20], 'ada__learning_rate': [0.01, 0.1], 'ada__base_estimator__max_depth': [1, 5]}
---	---	--

Pipeline

Parameter



```
[CV 1/2; 1/2] START rf_criterion=gini, rf_n_estimators=2
0.....
[CV 1/2; 1/2] END rf_criterion=gini, rf_n_estimators=20;, score=0.965 total time= 53.9s
[CV 1/2; 2/2] START dt_max_depth=1
0.....
[CV 1/2; 2/2] END .....dt_max_depth=10;, score=0.784 total time= 3.9s
[CV 2/2; 2/2] START lr_C=0.01, lr_penalty=l
2.....
[CV 2/2; 2/2] END ...lr_C=0.01, lr_penalty=l2;, score=0.745 total time= 0.9s
[CV 2/2; 1/4] START ada_base_estimator_max_depth=1, ada_learning_rate=0.01, ada_n_estimators=20
[CV 2/2; 1/4] END ada_base_estimator_max_depth=1, ada_learning_rate=0.01, ada_n_estimators=20;, score=0.652 total time= 8.9s
[CV 1/2; 4/4] START ada_base_estimator_max_depth=5, ada_learning_rate=0.1, ada_n_estimators=20
[CV 1/2; 4/4] END ada_base_estimator_max_depth=5, ada_learning_rate=0.1, ada_n_estimators=20;, score=0.802 total time= 34.7s
[CV 2/2; 2/2] START rf_criterion=log_loss, rf_n_estimators=2
0.....
[CV 2/2; 2/2] END rf_criterion=log_loss, rf_n_estimators=20;, score=0.966 total time= 51.1s
[CV 1/2; 1/2] START dt_max_depth=
5.....
[CV 1/2; 1/2] END .....dt_max_depth=5;, score=0.726 total time= 2.2s
[CV 1/2; 1/2] START lr_C=0.01, lr_penalty=l
1.....
[CV 1/2; 1/2] END .....lr_C=0.01, lr_penalty=l1;, score=nan total time= 0.1s
[CV 1/2; 1/4] START ada_base_estimator_max_depth=1, ada_learning_rate=0.01, ada_n_estimators=20
[CV 1/2; 1/4] END ada_base_estimator_max_depth=1, ada_learning_rate=0.01, ada_n_estimators=20;, score=0.632 total time= 8.7s
[CV 2/2; 3/4] START ada_base_estimator_max_depth=5, ada_learning_rate=0.01, ada_n_estimators=20
[CV 2/2; 3/4] END ada_base_estimator_max_depth=5, ada_learning_rate=0.01, ada_n_estimators=20;, score=0.746 total time= 35.4s
[CV 2/2; 1/2] START rf_criterion=gini, rf_n_estimators=2
0.....
[CV 2/2; 1/2] END rf_criterion=gini, rf_n_estimators=20;, score=0.966 total time= 53.7s
[CV 2/2; 1/2] START dt_max_depth=
5.....
[CV 2/2; 1/2] END .....dt_max_depth=5;, score=0.719 total time= 2.2s
[CV 2/2; 1/2] START lr_C=0.01, lr_penalty=l
1.....
[CV 2/2; 1/2] END .....lr_C=0.01, lr_penalty=l1;, score=nan total time= 0.1s
[CV 1/2; 2/4] START ada_base_estimator_max_depth=1, ada_learning_rate=0.1, ada_n_estimators=20
[CV 1/2; 2/4] END ada_base_estimator_max_depth=1, ada_learning_rate=0.1, ada_n_estimators=20;, score=0.720 total time= 8.5s
```

```
[CV 1/2; 3/4] START ada_base_estimator__max_depth=5, ada_learning_rate=0.01, ada_n_estimators=20
[CV 1/2; 3/4] END ada_base_estimator__max_depth=5, ada_learning_rate=0.01, ada_n_estimators=20;, score=0.756 total time= 35.7s
[CV 1/2; 2/2] START rf_criterion=log_loss, rf_n_estimators=20
0.....
[CV 1/2; 2/2] END rf_criterion=log_loss, rf_n_estimators=20;, score=0.962
total time= 51.2s
[CV 2/2; 2/2] START dt_max_depth=1
0.....
[CV 2/2; 2/2] END .....dt_max_depth=10;, score=0.770 total time=
3.8s
[CV 1/2; 2/2] START lr_C=0.01, lr_penalty=l2
2.....
[CV 1/2; 2/2] END ...lr_C=0.01, lr_penalty=l2;, score=0.746 total time=
0.9s
[CV 2/2; 2/4] START ada_base_estimator__max_depth=1, ada_learning_rate=0.1, ada_n_estimators=20
[CV 2/2; 2/4] END ada_base_estimator__max_depth=1, ada_learning_rate=0.1,
ada_n_estimators=20;, score=0.717 total time= 9.0s
[CV 2/2; 4/4] START ada_base_estimator__max_depth=5, ada_learning_rate=0.1, ada_n_estimators=20
.....
```

## Experiment 2: Using Selected Features (Using Entire Dataset)

```
In [ ]: exp_log_full
```

Out[ ]:

Pipeline

Parameter

0	Baseline Pipeline(steps=[('rf', RandomForestClassifier())]) with 100 inputs	{'rf__n_estimators': [20], 'rf__criterion': ['gini', 'log_loss']}
---	--	--

1	Baseline Pipeline(steps=[('dt', DecisionTreeClassifier())]) with 100 inputs	{'dt__max_depth': [5, 10]}
---	--	----------------------------

2	Baseline Pipeline(steps=[('lr', LogisticRegression())]) with 100 inputs	{'lr__C': [0.01], 'lr__penalty': ['l1', 'l2']}
---	--	--

3	Baseline Pipeline(steps=[('ada', AdaBoostClassifier(base_estimator=DecisionTreeClassifier()))]) with 100 inputs	{'ada__n_estimators': [20], 'ada__learning_rate': [0.01, 0.1], 'ada__base_estimator__max_depth': [1, 5]}
---	---	--

Pipeline	Parameter

## Experiment 3: Using Feature Engineering (Without New Features)

```
In [ ]: experimentLog1
```

Out [ ]:

	Pipeline	Parameters	TrainAcc	ValidAcc	TestAcc	Train Time(s)	Test Time(s)
0	Baseline Multinomial NB with 219 inputs	{'clf__alpha': (1, 0.1, 0.01, 0.001, 0.0001, 1...}	91.93%	92.02%	91.83%	0.031847	0.009847
1	Baseline Logistic Regression with 219 inputs	{'clf__solver': ['lbfgs', 'liblinear', 'newton...']}	91.93%	92.02%	91.85%	0.030177	0.009847
2	Baseline AdaBoostClassifier with 219 inputs	{'clf__n_estimators': [1, 2]}	91.93%	92.02%	91.83%	0.031300	0.009847
3	Baseline Random Forest with 219 inputs	{'rf__n_estimators': [1, 10], 'rf__min_samples...}	98.55%	91.96%	91.74%	0.034335	0.009847
4	Baseline KNN with 219 inputs	{'rf__n_estimators': [1, 10], 'rf__min_samples...}	92.97%	91.96%	91.74%	0.020069	0.009847

## Experiment 4 (Using Feature Engineering (With 17 New Features))

In [ ]: experimentLog2

Out[ ]:

	Pipeline	Parameters	TrainAcc	ValidAcc	TestAcc	Train Time(s)	Test Time(s)
0	Baseline Multinomial NB with 17 inputs	{'clf__alpha': (1, 0.1, 0.01, 0.001, 0.0001, 1...}	91.93%	92.02%	91.83%	0.056123	0.010725
1	Baseline Logistic Regression with 17 inputs	{'clf__solver': ['lbgfs', 'liblinear', 'newton...']}	91.91%	92.01%	91.82%	0.025727	0.010725
2	Baseline AdaBoostClassifier with 17 inputs	{'clf__n_estimators': [1, 2]}	91.93%	92.02%	91.83%	0.019794	0.010725
3	Baseline Random Forest with 17 inputs	{'rf__n_estimators': [1, 10], 'rf__min_samples...}	98.56%	91.88%	91.57%	0.021411	0.010725
4	Baseline KNN with 17 inputs	{'rf__n_estimators': [1, 10], 'rf__min_samples...}	93.05%	91.88%	91.57%	0.019411	0.010725
5	Baseline Ensemble with 17 inputs	{'clf__lr__C': [0.01], 'clf__lr__penalty': ['l...']}	91.97%	91.88%	91.57%	0.017676	0.010725
6	Baseline KNN with 17 inputs	{'clf__lr__C': [0.01], 'clf__lr__penalty': ['l...']}	91.97%	91.88%	91.57%	0.017676	0.010725

## Submission in Kaggle

In [ ]:

```
temp = pp_pipe.transform(df_app_test)
temp2 = dict1[Decision_Tree].predict_proba(temp)[:, 1]
Final_DataFrame = df_app_test[['SK_ID_CURR']]
Final_DataFrame['TARGET'] = temp2
Final_DataFrame = Final_DataFrame.groupby('SK_ID_CURR', as_index=False)
Final_DataFrame = Final_DataFrame.max()
Final_DataFrame.head()
```

Out[ ]:

	SK_ID_CURR	TARGET
0	100001	0.043521
1	100005	0.051460
2	100013	0.018695
3	100028	0.044092
4	100038	0.076037

In [ ]:

```
Final_DataFrame.columns
```

```
Out[ ]: Index(['SK_ID_CURR', 'TARGET'], dtype='object')
```

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

```
In [ ]:
```

## Kaggle report submission

<https://www.kaggle.com/competitions/home-credit-default-risk/submissions>

Submission	Private Score	Public Score	Selected
submission.csv	0.73315	0.73762	<input type="checkbox"/>
submission.csv	0.66457	0.65961	<input type="checkbox"/>
submission.csv	0.65139	0.66622	<input type="checkbox"/>

## Write-up

For this phase of the project, you will need to submit a write-up summarizing the work you did. The write-up form is available on Canvas (Modules-> Module 12.1 - Course Project - Home Credit Default Risk (HCDR)-> FP Phase 2 (HCDR) : write-up form ). It has the following sections:

## Abstract

The main problems to tackle in this phase are merging the whole dataset, extracting relevant features, creating new features, treating missing values, applying OHE and imputing methods, engineering RFM features and optimizing the model's parameters using hyperparameter tuning. The main goal of the feature is to evaluate the model's performance on the test set to check for improvement. The key experiments that we will perform are first we will consider selected features and implement 5 models, second, we performed feature engineering without adding the new features and third we performed feature engineering, and we will add the newly generated result. Our results and finding are that with feature engineering and adding newly generated features, AdaBoost and MultinomialNB models achieved 91.83% accuracy. While on the other hand, Logistic

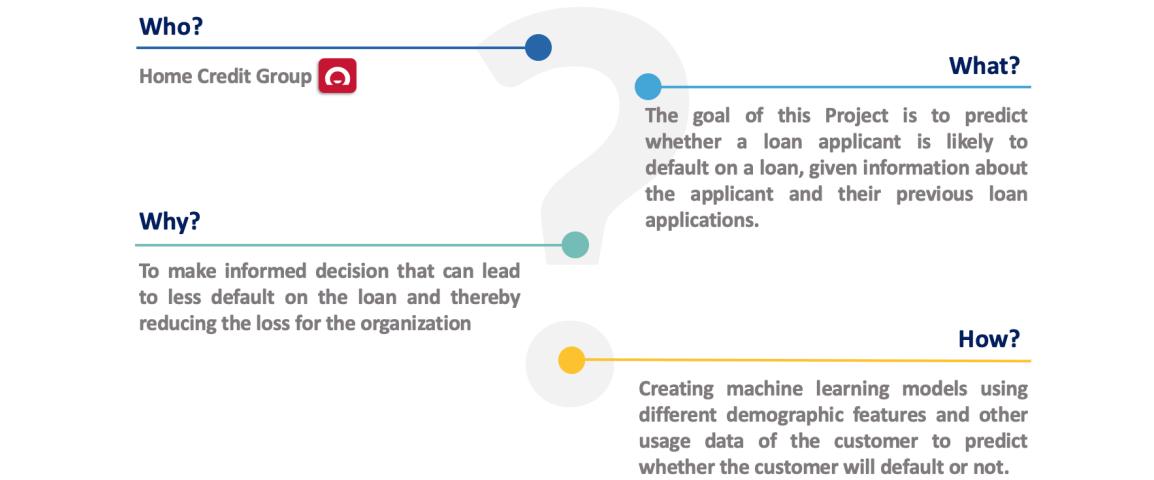
Regression achieved 91.85% accuracy with feature engineering (in this case we did not consider the newly generated features).

## Introduction

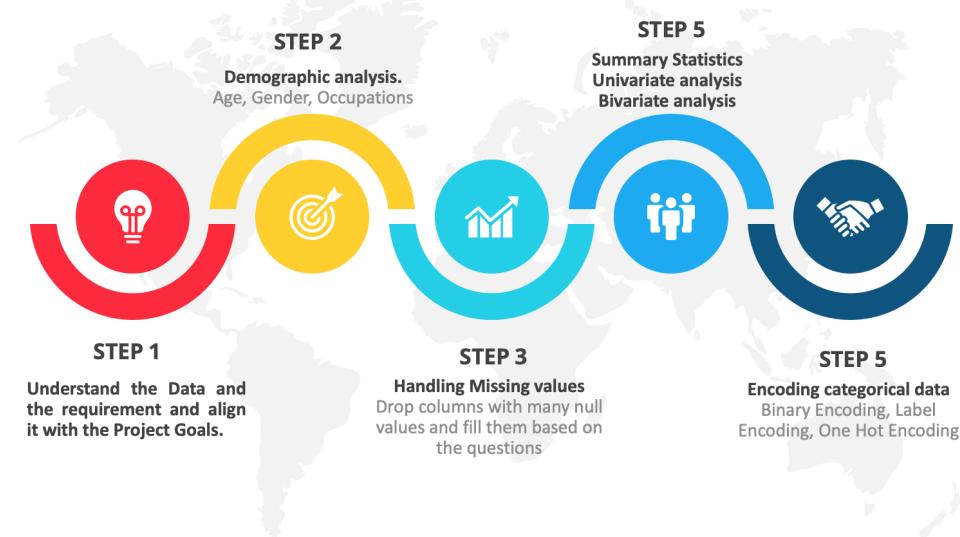
The Home Credit Default Risk (HCDR) dataset, which contains data pertaining to individuals' loan applications and repayment histories, is a widely used dataset in the field of credit risk analysis. The purpose of this research is to use this dataset to create a prediction model that is capable of accurately predicting whether a loan application would default on their loan. This is a critical responsibility for banks and lending organizations because loan defaults can result in large financial losses. Exploratory data analysis, data preprocessing, feature engineering, and machine learning model creation are all aspects of the project. The ultimate goal is to create a model that can reliably anticipate loan default risk and help banks make educated lending decisions. The project also includes difficulties like as dealing with missing data, imbalanced classes, and choosing appropriate evaluation criteria. The project's findings have the potential to give lending institutions with significant insights into their risk assessment processes, allowing them to reduce the likelihood of financial losses due to loan defaults.

## Project Description

### What are we aiming to solve?



## EDA and Data Preparation



## Data

There are 7 different sources of data:

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

card can have many rows.

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

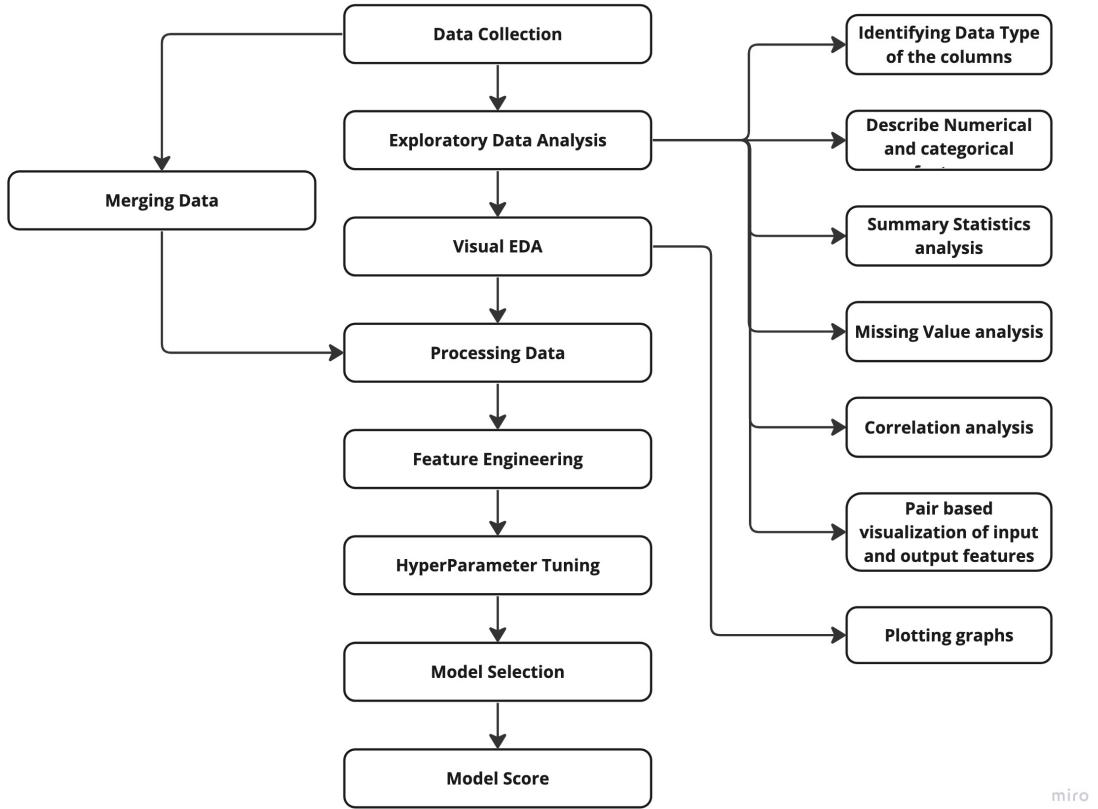
## Table sizes

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

## Task to be tackled

- Merge the whole dataset
- Extract all possible features using feature Engineering
- Clean the data
- Performing feature selection.
- Provide analysis of feature importance.
- Identify the most relevant features to the Target variable and prediction
- Create new features from the most correlated features to contribute towards the prediction
- Apply One-Hot Encoding (OHE) to categorical features
- Apply imputing methods to fix missing values in the data
- Prepare the data for feeding into the model
- Perform hyperparameter tuning for the models using K-Fold cross validation and GridSearchCV
- Evaluate the model's performance on the test set to check if accuracy has improved

## Machine Learning Pipeline



miro

**Data collection** is the first step in any machine learning pipeline. It involves gathering and compiling relevant data from various sources, which may include databases, APIs, web scraping, or other means. The quality and relevance of the data collected is crucial to the success of the entire pipeline, as it forms the basis for all subsequent steps.

In the data collection stage, it is important to identify and understand the problem to be solved and the data required to solve it. This includes defining the variables of interest, identifying potential data sources, and collecting the necessary data from these sources.

Data collection can be an iterative process, and it is often necessary to refine the data collection process as new insights are gained through the EDA (Exploratory Data Analysis) process. It is important to carefully document the data collection process, including any assumptions or limitations, as this information will be important for interpreting the results of the subsequent analysis.

During data collection, it is also important to ensure that the data is properly cleaned and preprocessed before analysis. This includes handling missing values, dealing with outliers, and transforming the data as necessary to meet the assumptions of the analysis.

Overall, data collection is a critical step in the machine learning pipeline, and it requires careful planning and attention to detail to ensure that the resulting data is of high quality and relevance to the problem being solved.

**Exploratory Data Analysis** : Exploratory Data Analysis (EDA) is a critical phase in the Machine Learning (ML) pipeline in which data scientists or ML engineers evaluate and analyze the information to understand the underlying structure, trends, and correlations.

EDA employs a number of approaches, including data visualization, summary statistics, and statistical testing, to obtain insights into the dataset's features, detect possible faults or anomalies, and influence the next steps in the ML pipeline.

To prepare the data for modeling, data scientists may undertake activities such as data cleansing, missing value imputation, outlier identification, and feature engineering during EDA. They may also evaluate the balance of the classes or target variable in the dataset and decide whether to use data balancing techniques like oversampling or undersampling.

Overall, EDA is crucial in assuring data quality and assisting in the selection of the best relevant algorithms and strategies to develop a strong and accurate ML model.

**Visual EDA** : Visual EDA, or visual Exploratory Data Analysis, is a technique that involves using visualizations to understand the structure and patterns in a dataset. Visual EDA aims to provide a quick and intuitive understanding of the data, enabling data scientists to make informed decisions on data preprocessing, feature engineering, and modeling.

Visual EDA techniques typically involve creating plots and charts that illustrate the distribution, relationship, and variability of the data. Some common visualizations used in EDA include histograms, scatter plots, box plots, heatmaps, and bar charts.

By visualizing the data, we can identify any potential issues with the dataset, such as missing or outlier values, skewed distributions, or correlations between variables. Visual EDA can also help in selecting appropriate data preprocessing techniques, such as scaling or normalization, and identifying which features may be relevant for the ML model.

Overall, visual EDA is an important step in the ML pipeline that enables data scientists to gain a deeper understanding of the data and make informed decisions about data preprocessing, feature engineering, and modeling.

**Processing Data** : Processing data refers to the steps involved in preparing and transforming raw data into a format that is suitable for analysis or machine learning (ML) models. Processing data is a crucial step in the data science workflow, as raw data may contain errors, inconsistencies, or missing values that can adversely affect the accuracy and reliability of the analysis.

The first step in processing data is to clean the data, which involves identifying and correcting any errors, removing duplicate records, and dealing with missing data. Data

cleaning may also involve transforming data into a consistent format, such as converting categorical data into numerical data or changing the date format to a standardized format.

After cleaning the data, the next step is to preprocess the data, which involves transforming the data to make it more suitable for analysis or ML models. Preprocessing may involve feature scaling, which ensures that all features are on a similar scale to avoid bias in the analysis. Feature engineering, which involves creating new features that may be more relevant to the analysis, is another important preprocessing step.

**Modeling** : Modeling refers to the process of creating a machine learning (ML) model that can make predictions or decisions based on input data. The goal of modeling is to develop a model that can generalize well to new, unseen data and accurately predict the outcome of interest.

The modeling process typically involves the following steps:

Selecting a model: Choosing an appropriate ML model that is best suited for the problem being solved. This can involve considering factors such as the type of data, the number of features, and the desired output.

Training the model: Using a training dataset to fit the model to the data. This involves estimating the model parameters using a chosen optimization algorithm that minimizes the error between the predicted and actual outcomes.

Evaluating the model: Using a validation dataset to assess the performance of the model. This involves measuring metrics such as accuracy, precision, recall, or F1-score to evaluate how well the model is performing.

Tuning the model: Adjusting the model parameters to optimize its performance on the validation dataset. This involves techniques such as grid search, random search, or Bayesian optimization to find the best combination of parameters.

Testing the model: Using a separate test dataset to evaluate the final performance of the model. This is done to ensure that the model can generalize well to new, unseen data.

Once the model has been developed and tested, it can be used for making predictions or decisions on new data. The model can also be updated or retrained periodically to ensure that it remains accurate and up-to-date.

**Feature Engineering** : Feature engineering is the process of creating new features, or variables, from existing raw data that can improve the performance of machine learning (ML) models. Feature engineering involves extracting information from the raw data and representing it in a way that is more suitable for modeling.

The goal of feature engineering is to create features that are relevant to the problem being solved and that capture the underlying patterns in the data. This process may involve domain knowledge or intuition about the data, as well as experimentation and iteration to find the most effective features.

Feature engineering involve a variety of techniques, such as:

Feature transformation: Transforming variables to fit a particular distribution or to normalize the data. Examples include log transformation, scaling, and standardization.

Feature extraction: Creating new variables by extracting information from existing variables. For example, extracting the day of the week or month from a date variable, or extracting the length of text.

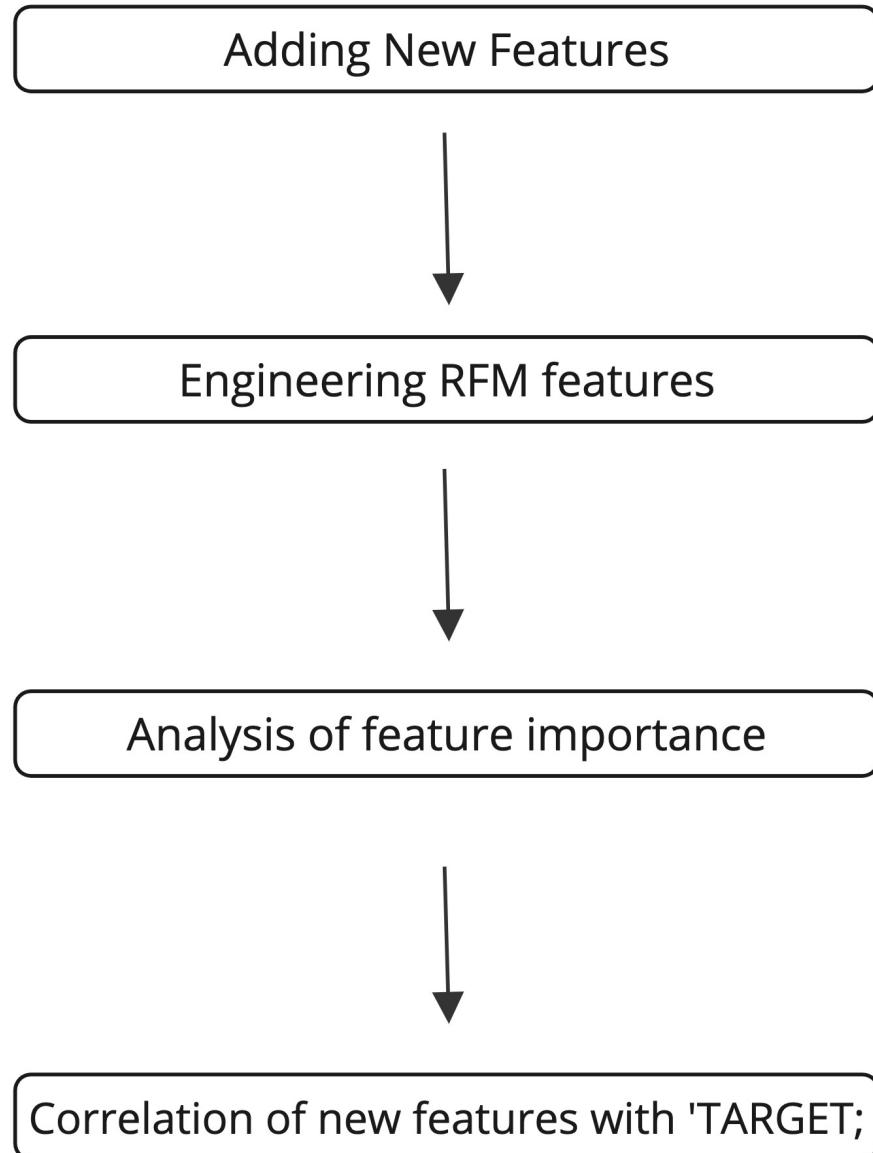
Feature aggregation: Creating new variables by combining multiple variables. For example, calculating the average or sum of a group of variables, or computing the percentage of one variable relative to another.

Feature selection: Selecting a subset of the most important variables for modeling. This can involve techniques such as correlation analysis, principal component analysis, or feature importance ranking.

Feature generation: Creating new variables through mathematical operations or other data manipulations. This can include creating interaction terms between variables, or combining categorical variables into a single feature using one-hot encoding.

Overall, feature engineering is an essential step in the ML pipeline that can significantly improve the accuracy and performance of models. It involves creating new features that capture relevant information from the data, and can require a combination of domain knowledge, creativity, and experimentation to achieve the best results.

# FEATURE ENGINEERING



miro

**Hyper Parameter Tuning :** Hyperparameter tuning refers to the process of optimizing the hyperparameters of a machine learning (ML) model to achieve better performance. Hyperparameters are parameters that are not learned during the training process but are set before training and can significantly impact the performance of the model.

Hyperparameter tuning involves searching for the optimal combination of hyperparameters that results in the best model performance. This is typically done using a validation dataset, where the performance of different hyperparameter configurations is evaluated.

Some common hyperparameters that may be tuned include:

Learning rate: A hyperparameter that determines how quickly the model learns during training.

Regularization parameters: Hyperparameters that control the level of regularization applied to the model, such as L1 or L2 regularization.

Number of layers: Hyperparameters that determine the number of layers in a neural network.

Number of hidden units: Hyperparameters that determine the number of neurons in each layer of a neural network.

Dropout rate: A hyperparameter that determines the amount of dropout regularization applied to a neural network.

Hyperparameter tuning can be done using a variety of techniques, including grid search, random search, and Bayesian optimization. Grid search involves evaluating the model performance for all possible combinations of hyperparameters within a predefined range. Random search involves randomly sampling the hyperparameter space and evaluating the performance. Bayesian optimization uses a probabilistic model to select the next set of hyperparameters to evaluate, based on the previous results.

## EDA

After performing the EDA, we have got some important columns that has the highest affect on the Target variable. It means that these columns could predict the outcome instead of using all the columns.

**EXT\_SOURCE\_1**, **EXT\_SOURCE\_2**, and **EXT\_SOURCE\_3**: These columns are moderately correlated with the target variable and also moderately correlated with each other, so they may be good features to include in a model.

**DAYS\_BIRTH**: This column is moderately negatively correlated with the target variable, indicating that younger applicants are more likely to default on their loans.

**DAYS\_EMPLOYED**: This column has a weak negative correlation with the target variable, indicating that longer-term employment may be associated with a lower risk of default.

**AMT\_CREDIT**: This column has a moderate positive correlation with the target variable, indicating that higher loan amounts may be associated with a higher risk of default.

**AMT\_GOODS\_PRICE**: This column has a moderate positive correlation with the target variable, similar to AMT\_CREDIT.

The following columns have also been selected to train the model because after merging the 'TARGET' column from application\_train.csv to different dataset files such as instalment\_payments, previous\_application, pos\_cash\_balance, bureau and bureau\_balance, and plotting the correlation matrix we inferred that these columns had correlation more than 0.3 with target column: **DAYS\_CREDIT**, **DAYS\_CREDIT\_UPDATE**, **DAYS\_ENDDATE\_FACT**, **DAYS\_ENTRY\_PAYMENT**, **DAYS\_INSTALMENT**, **DAYS\_DECISION**, **DAYS\_FIRST\_DRAWING** and **CNT\_PAYMENT**.

## Loss Function

Logistic Regression and MultinomialNB: Logarithmic Loss Function

$$L(y, \hat{y}) = - (y \log(\hat{y}) + (1 - y) \log(1 - \hat{y}))$$

Random Forest Classifier: Gini Impurity Decision Tree: Splitting Criterion used in Gini Impurity

$$L(y, \hat{y}) = \max 0, \hat{y} - y$$

AdaBoost Classifier: Exponential Loss Function

$$L(y, f(x)) = \exp(-yf(x))$$

## Models

We implemented the following models to find which one gives the best performance:

1. Logistic Regression: A common model for situations involving binary classification is logistic regression. It operates by utilizing a logistic function to estimate the likelihood of the target variable (default) with respect to the input parameters (such as age, income, and loan amount). Based on the applicant's demographic and financial data, logistic regression can be applied in the HCDR dataset to forecast the likelihood of default.
2. Random Forest Classifier: Several Decision Trees are used in an ensemble model called Random Forest to lessen overfitting and boost accuracy. It operates by randomly choosing a subset of features at each split while training a series of Decision Trees on bootstrapped subsets of the data.
3. Decision Tree Classifier: A decision tree model divides the data recursively into subsets depending on the most significant attributes until a stopping requirement is satisfied. As a result, a tree-like model is produced, with each internal node standing in for a judgment call based on a feature and each leaf node for a prediction.
4. Adaboost Classifier: A popular machine learning approach for categorization issues is called Adaboost (Adaptive Boosting). It functions by fusing weak classifiers to

produce a powerful classifier.

5. KNN: The k nearest points in the dataset to the new data point are found using the straightforward and efficient classification technique known as KNN, which then uses the majority class of those k neighbors as the projected class.
6. Multinomial NB: Based on Bayes' theorem, Multinomial NB is a probabilistic classification algorithm. Given the characteristics of the data point, each class's probability is calculated, and the class with the highest probability is chosen as the predicted class.
7. Ensemble Model: An ensemble model is a collection of machine learning models that collaborate to predict something. The objective behind ensemble models is to aggregate the results of various models to provide a prediction that is more reliable and accurate.

## Data Lineage

The data lineage for this project would begin with the raw data, which would undergo initial preprocessing to treat missing values, such as through imputation techniques. The next step would be Exploratory Data Analysis (EDA) to understand the underlying patterns and relationships in the data, followed by visual EDA to gain additional insights into the data.

After that, feature selection would be performed to identify the most important features for the model. This would be followed by training baseline models to establish a benchmark for model performance of our base line models.

Overall, it involved multiple steps, including preprocessing, EDA, visual EDA, feature selection, and training baseline models, with each step building upon the previous one to create a robust and reliable ML model. This workflow would ensure that the data is clean, consistent, and appropriate for the ML model, providing transparency and accountability for all data-related activities.

## Experimental Results.

In Phase 2, we did EDA and understood more about the dataset, and ran baseline models on the subset of the dataset with features that have high correlation values with the target variable. We implemented 4 algorithms from which the Decision Tree algorithm has the highest accuracy of 93.46%. This was our first experiment.

In Phase 3, we did three more experiments. Following are the details about the experiments we conducted:

### **1. Experiment 2: Using Selected Features (Using Entire Dataset)**

For this experiment, we considered the entire dataset. After performing the EDA in the last phase, we have got some important columns that have the highest effect on the Target variable. It means that these columns could predict the outcome instead of using all the columns. **EXT\_SOURCE\_1**, **EXT\_SOURCE\_2**, and **EXT\_SOURCE\_3**:

These columns are moderately correlated with the target variable and also moderately correlated with each other, so they may be good features to include in a model.

**DAYS\_BIRTH**: This column is moderately negatively correlated with the target variable, indicating that younger applicants are more likely to default on their loans.

**DAYS\_EMPLOYED**: This column has a weak negative correlation with the target variable, indicating that longer-term employment may be associated with a lower risk of default.

**AMT\_CREDIT**: This column has a moderate positive correlation with the target variable, indicating that higher loan amounts may be associated with a higher risk of default.

**AMT\_GOODS\_PRICE**: This column has a moderate positive correlation with the target variable, similar to AMT\_CREDIT. The following columns have also been selected to train the model because after merging the 'TARGET' column from application\_train.csv to different dataset files such as instalment\_payments, previous\_application, pos\_cash\_balance, bureau and bureau\_balance, and plotting the correlation matrix we inferred that these columns had correlation more than 0.3 with target column:

**DAY\_CREDIT**, **DAY\_CREDIT\_UPDATE**, **DAY\_ENDDATE\_FACT**,

**DAY\_ENTRY\_PAYMENT**, **DAY\_INSTALMENT**, **DAY\_DECISION**,

**DAY\_FIRST\_DRAWING** and **CNT\_PAYMENT**. We prepared a pipeline using sci-kit-learn's ColumnTransformer. Both category and numerical features are included in the dataset, thus the preparation pipeline treated them separately. The pipeline applies one-hot encoding to categorical features before using SimpleImputer to fill in missing values with the value that occurs the most frequently. The pipeline uses StandardScaler to scale the data for numerical characteristics and SimpleImputer to replace missing values with the median value. The appropriate pipeline was applied to each type of feature using the ColumnTransformer. Categorical\_trans pipeline is applied to categorical characteristics by the "cat" transformer, whereas the numerical\_trans pipeline is applied to numerical features by the "num" transformer. The ColumnTransformer object is added to the parameter of the transformers to establish the preprocessing pipeline. The data can then be preprocessed using the resulting pp\_pipe object before being fitted to a machine-learning model.

Overall, by applying the proper transformations to each type of feature, we are building a preprocessing pipeline that can handle both categorical and numerical characteristics in the HCDR dataset. The pipeline can assist in making sure that the data is appropriately prepared for modeling and can increase the precision of machine learning models developed using the dataset.

Modeling:

We implemented the following algorithms considering 100 features from the dataset:

1. Random Forest
2. Decision Tree
3. Logistic Regression
4. Adaboost Classifier

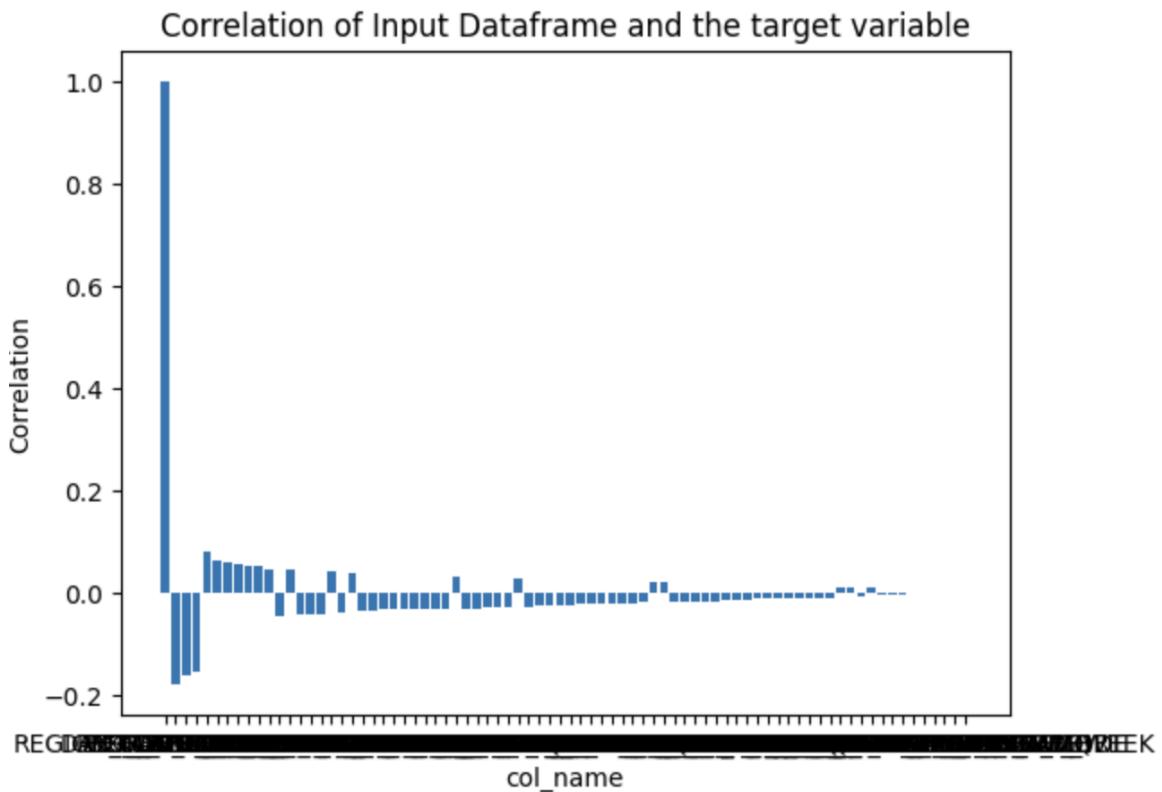
The following table will give us more information on the best params and evaluation metrics we used for all the above algorithms:

<b>Experiments</b>	<b>Pipeline</b>	<b>Parameters</b>
2	Baseline Pipeline(steps=[('rf', RandomForestClassifier())]) with 100 inputs	{'rf__n_estimators': [20, 50], 'rf__criterion': ['gini', 'entropy']}
	Baseline Pipeline(steps=[('dt', DecisionTreeClassifier())]) with 100 inputs	{'dt__max_depth': [5, 10, 15]}
	Baseline Pipeline(steps=[('lr', LogisticRegression())]) with 100 inputs	{'lr__C': [0.01], 'lr__penalty': ['l2']}
	Baseline Pipeline(steps=[('ada', AdaBoostClassifier(base_estimator=DecisionTreeClassifier()))]) with 100 inputs	{'ada__n_estimators': [50, 100], 'ada__learning_rate': [0.1, 0.5], 'ada__base_estimator_': [1, 5]}

Experiments	Pipeline	Parameters

## 2. Experiment 3: Using Feature Engineering (Without New Features) Feature Engineering:

For this experiment, we first grouped the respective DataFrame by the SK\_ID\_CURR column and calculated the median value for each column. This is being done to aggregate the data and summarize the history of each borrower. We then merged "DF\_Bureau" and "DF\_BureauBalance", by combining the data from the "DF\_Bureau" and "DF\_BureauBalance" databases, this method aims to produce a new dataset with summary statistics by "SK\_ID\_CURR". In machine learning models to forecast credit default risk, the generated dataset can be used as a feature. To take care of high proportion of zeros, we created a function Count\_Percentage\_of\_Zeroes. This function's goal is to locate the dataset columns with the highest proportion of zero values. These columns can be eliminated from the dataset since they could not contain data that is helpful for estimating credit default risk. We then have function names Divide\_DataFrame\_into\_Categorical\_and\_Numerical, this function's goal is to get the data ready for additional modeling and analysis. While categorical columns need to be converted into numerical representations, numerical columns can be used directly in many machine-learning techniques. We also have a function named Correlation\_Between\_InputDF\_Target, in this function the goal column "TARGET" and the columns of the input data frame are correlated. The columns whose correlation value is smaller than the input parameter correlation's absolute value are then removed. The following diagram is the correlation between our input data frame and target variable:



Then the numerical columns in the dataset New\_DataFrame\_Numerical are subjected to missing value imputation.

#### **Hyper Parameter Tuning:**

The Scikit-learn GridSearchCV function is used for hyperparameter tweaking. The base\_pipeline, the collection of hyperparameters, the number of cross-validation folds (cv), etc are all inputs for the function. The optimum hyperparameter combination that delivers the maximum performance on the validation set is then found using a grid search over the hyperparameters. The reason we choose the GridSearchCV function is that it enables us to methodically search through many combinations of hyperparameters and identify the optimum combination for our particular problem, the GridSearchCV function from Scikit-learn is a well-liked technique for hyperparameter tuning. There are frequently a lot of hyperparameters that need to be set before a machine-learning model can be trained. The ideal settings for these hyperparameters will vary depending on the particular dataset and the issue at hand. By allowing you to specify a grid of hyperparameters to search through, GridSearchCV streamlines the process of hyperparameter tuning. The combination that yields the greatest cross-validated performance is then returned after a thorough search of all conceivable hyperparameter combinations in the grid. By doing this, we can determine the ideal settings for your hyperparameters without having to manually test out various pairings and gauge how well they work. This can help you develop models more quickly and with less effort.

#### **Modeling:**

We considered a total of 213 features for this experiment and implemented the following algorithms:

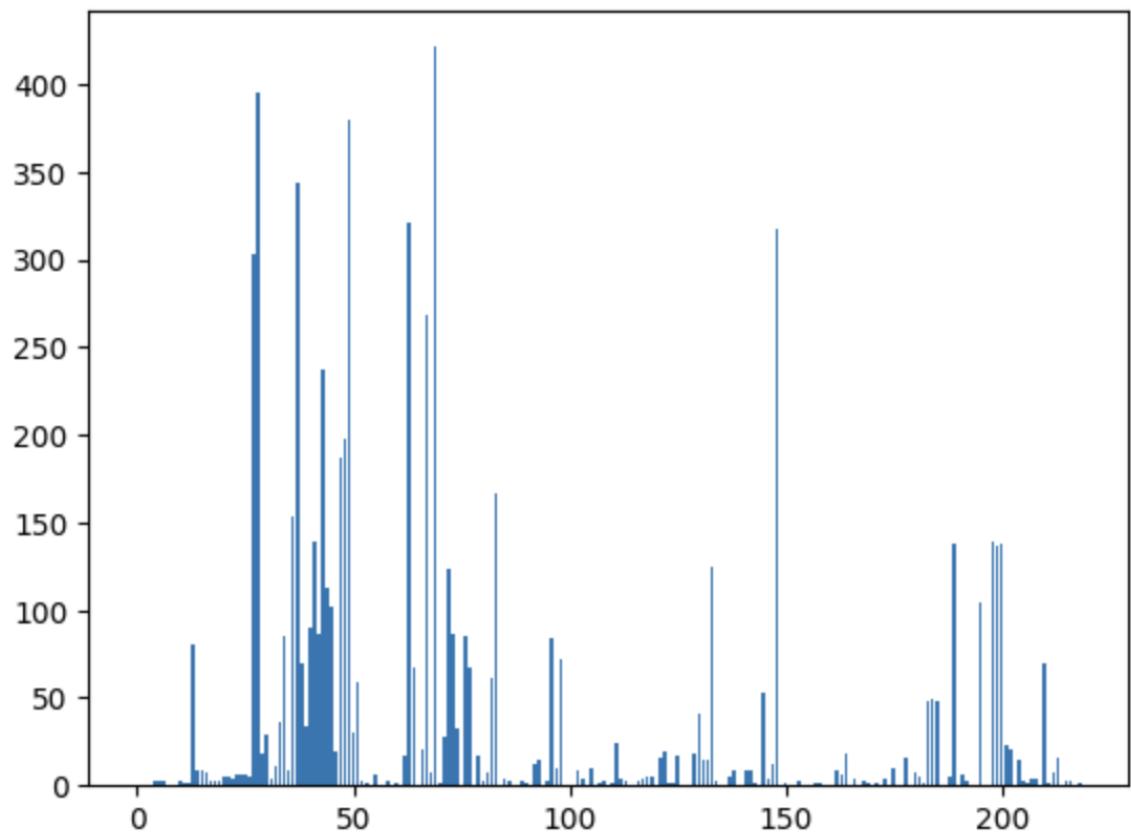
1. Multinomial NB
2. Logistic Regression
3. AdaBoost Classifier
4. Random Forest
5. KNN

The following tables will give us more information on the best params and evaluation metrics we used for all the above algorithms:

Experiments	Pipeline	Parameters	TrainAcc	ValidAcc	TestAcc	Train Time(s)
3	Baseline Multinomial NB with 219 inputs	{'clf_alpha': (1, 0.1, 0.01, 0.001, 0.0001, 1)}	91.93%	92.02%	91.83%	0.031847
	Baseline Logistic Regression with 219 inputs	{'clf_solver': ['lbfgs', 'liblinear', 'newton']}	91.93%	92.02%	91.85%	0.030177
	Baseline AdaBoostClassifier with 219 inputs	{'clf_n_estimators': [1, 2]}	91.93%	92.02%	91.83%	0.031300
	Baseline Random Forest with 219 inputs	{'rf_n_estimators': [1, 10]}	98.55%	91.96%	91.74%	0.034335
	Baseline KNN with 219 inputs	{'rf_n_estimators': [1, 10], 'rf_min_samples': 5}	92.97%	91.96%	91.74%	0.020069

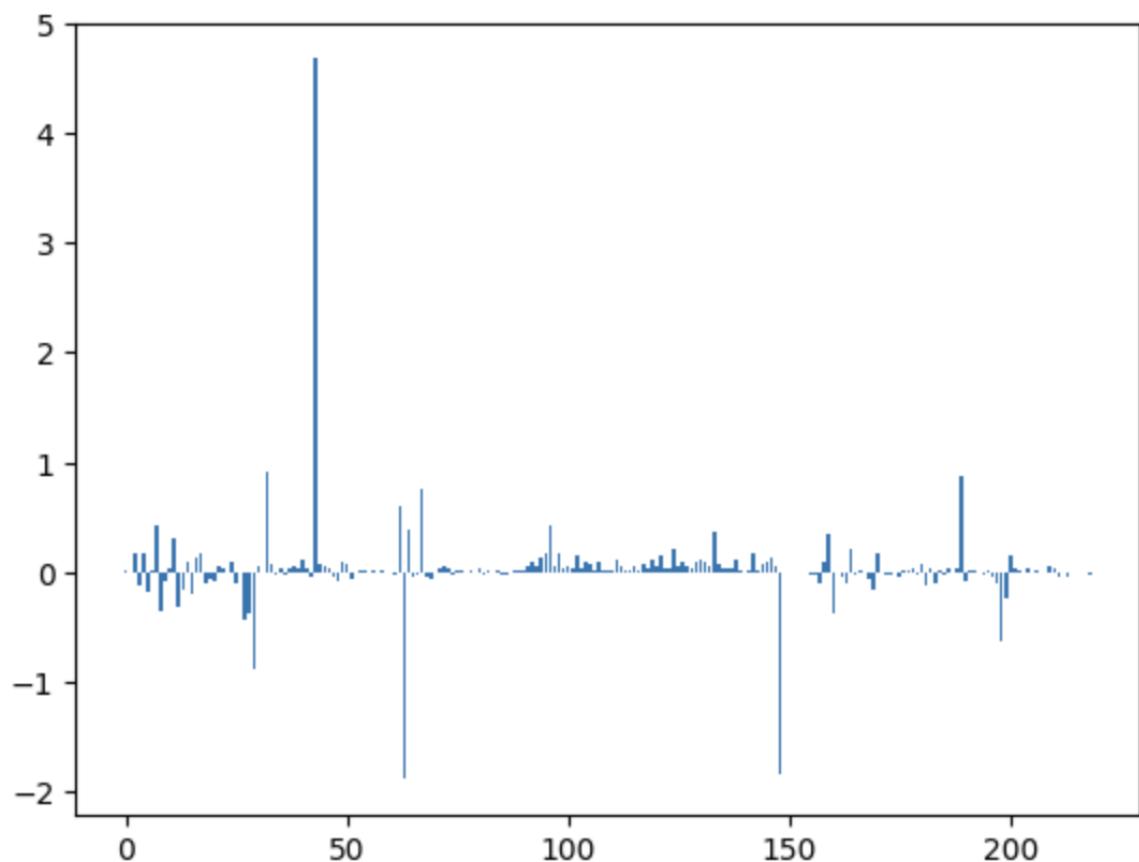
#### Feature Importance:

1. Multinomial NB

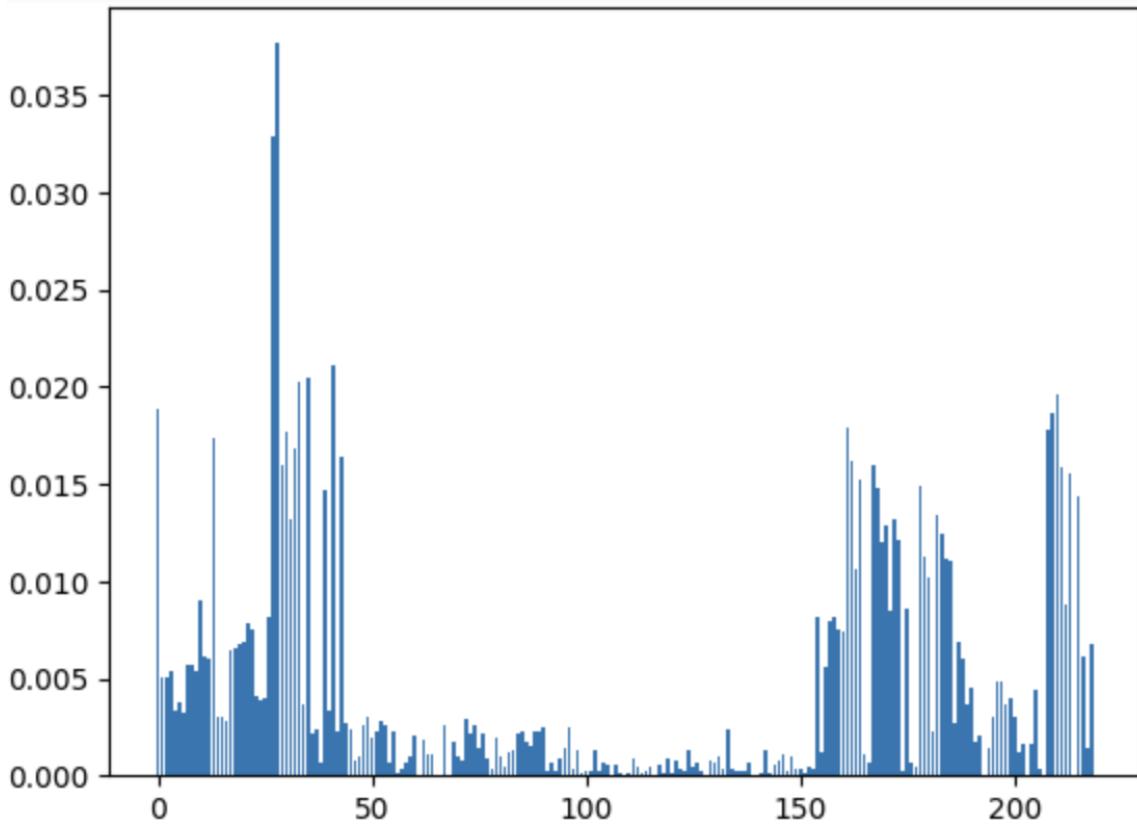


2. Logistic Regression

---



3. Random Forest



### 3. Experiment 4: Using Feature Engineering (With 17 New Features) Feature

**Engineering:** For adding new features to the dataset, we establish a pipeline that separates and then combines the categorical and numerical elements into a single data frame. First, two pipelines are established separately for the category and numerical features. While the categorical pipeline replaces missing values with the most frequent value of each feature using SimpleImputer() and then applies one-hot encoding using OneHotEncoder, the numerical pipeline replaces missing values with the mean of each feature using SimpleImputer(). In order to specify which pipeline should be used for each set of columns, the two pipelines are then concatenated into a single ColumnTransformer() object. The two pipelines are then applied to the corresponding sets of columns in DF\_Temp1 by calling the fit\_transform() method of the Data\_Pipeline object, which results in the creation of a single numpy array. In the end, the OneHotEncoder() object's get\_feature\_names\_out() method is used to retrieve the feature names of the one-hot encoded categorical features. The resulting numpy array is then transformed into a pandas data frame by combining the numerical and categorical feature names.

Finally, we create 13 new features (Feature\_1 to Feature\_13) utilizing different columns from the consolidated data frame after merging numerous data frames into one (DF\_Final). Mathematical procedures such as division, multiplication, and addition are used to calculate these additional properties. The final data frame's shape is subsequently determined by the algorithm and assigned to the variable DF\_Final. In order to calculate the correlation between the newly produced features and the goal

variable "TARGET," the final line of code invokes the function Correlation\_Between\_InputDF\_Target. The function generates a correlation matrix after calculating the Pearson correlation coefficient between each feature in the input and the desired variable. To only include features with a correlation coefficient greater than or equal to 0.0, a threshold value of 0.0 is crossed. Finding the newly developed features that have the highest correlation with the objective variable and may one day serve as predictors in a machine learning model is the goal of this stage.

We had added the following new features:

1. DF\_Final['Feature\_1']=  
DF\_Final['AMT\_CREDIT\_SUM']/(DF\_Final['DAYS\_EMPLOYED']+1)
2. DF\_Final['Feature\_2']=  
DF\_Final['CNT\_DRAWINGS\_POS\_CURRENT']/(DF\_Final['FLOORSMIN\_AVG']+1)
3. DF\_Final['Feature\_3']=  
DF\_Final['AMT\_CREDIT\_SUM']/(DF\_Final['FLOORSMIN\_AVG']+1)
4. DF\_Final['Feature\_4']=DF\_Final['DAYS\_CREDIT']\*  
(DF\_Final['DAYS\_ENDDATE\_FACT']+1)
5. DF\_Final['Feature\_5']=DF\_Final['AMT\_CREDIT\_SUM']/(DF\_Final['AMT\_CREDIT\_SUM']+1)
6. DF\_Final['Feature\_6']=DF\_Final['AMT\_CREDIT\_x']/(DF\_Final['AMT\_CREDIT\_SUM']+1)
7. DF\_Final['Feature\_7']=(DF\_Final['DAYS\_CREDIT']+DF\_Final['AMT\_CREDIT\_SUM'])
8. DF\_Final['Feature\_8']=(DF\_Final['AMT\_CREDIT\_x']+DF\_Final['FLOORSMIN\_AVG'])
9. DF\_Final['Feature\_9']=  
DF\_Final['AMT\_CREDIT\_SUM']/(DF\_Final['AMT\_GOODS\_PRICE\_x']+1)
10. DF\_Final['Feature\_10']=  
DF\_Final['AMT\_CREDIT\_SUM']/(DF\_Final['AMT\_GOODS\_PRICE\_x']+1)
11. DF\_Final['Feature\_11']= DF\_Final['DAYS\_BIRTH']+(DF\_Final['DAYS\_EMPLOYED']+1)
12. DF\_Final['Feature\_12']=  
(DF\_Final['EXT\_SOURCE\_1']+DF\_Final['EXT\_SOURCE\_2']+DF\_Final['EXT\_SOURCE\_3'])
13. DF\_Final['Feature\_13']= DF\_Final['CNT\_INSTALMENT\_MATURE\_CUM']\*  
(DF\_Final['CNT\_DRAWINGS\_POS\_CURRENT']+1)

We also added the following RFM features:

1. DF\_Final['RFM']= hcdr\_rfm["RFM"]
2. DF\_Final['R']= hcdr\_rfm["R"]
3. DF\_Final['F']= hcdr\_rfm["F"]
4. DF\_Final['M']= hcdr\_rfm["M"]

### **Impact of new features :**

We found a correlation between the new features and the target to understand the impact of the features as shown below:

	col_name	Correlation
0	TARGET	1.000000
1	Feature_12	-0.221463
2	Feature_4	-0.054394
3	Feature_13	-0.050172
4	Feature_11	-0.043322
5	Feature_2	0.030728
6	Feature_8	-0.030390
7	Feature_7	-0.015612
8	Feature_3	-0.015356
9	Feature_9	-0.011549
10	Feature_10	-0.011549
11	Feature_1	-0.005593
12	Feature_5	-0.002558
13	Feature_6	0.000649

### **Why these features were added:**

We added new features because by giving the model new information that can be helpful in forecasting the target variable, new features can increase the predictive ability of the model. We also added RFM features. Recency, Frequency, and Monetary (RFM) analysis is a popular customer segmentation technique that aids in the identification of high-value clients and the optimization of marketing plans. RFM attributes can be useful in illuminating a borrower's credit behavior in the context of the HCDR (Home Credit Default Risk) dataset and enhancing the predictive capability of the models. The terms "Recency," "Frequency," and "Monetary" relate to the interval since the borrower's most recent loan or credit transaction, respectively. "Monetary" also refers to the total amount borrowed or the credit limit. We can capture the temporal patterns and credit behavior of each borrower by including these RFM elements in the modeling process, which can be useful in determining their creditworthiness. We can make more informative features that can considerably increase the accuracy of our models in forecasting loan defaults by integrating RFM features with other pertinent factors from the HCDR dataset, like demographic data and loan characteristics.

### **Hyperparameter Tuning:**

The Scikit-learn GridSearchCV function is used for hyperparameter tweaking. The base\_pipeline, the collection of hyperparameters, the number of cross-validation folds (cv), etc are all inputs for the function. The optimum hyperparameter combination that delivers the maximum performance on the validation set is then found using a grid search over the hyperparameters. The reason we choose the GridSearchCV function is that it enables us to methodically search through many combinations of hyperparameters and identify the optimum combination for us particular problem, the GridSearchCV function from Scikit-learn is a well-liked technique for hyperparameter tuning. There are frequently a lot of hyperparameters that need to be set before a machine-learning model can be trained. The ideal settings for these hyperparameters will vary depending on the particular dataset and the issue at hand. By allowing you to specify a grid of hyperparameters to search through, GridSearchCV streamlines the process of hyperparameter tuning. The combination that yields the greatest cross-validated performance is then returned after a thorough search of all conceivable hyperparameter combinations in the grid. By doing this, we can determine the ideal settings for your hyperparameters without having to manually test out various pairings and gauge how well they work. This can help you develop models more quickly and with less effort.

### **Modeling:**

We considered a total of 17 new features for this experiment and implemented the following algorithms:

1. Multinomial NB

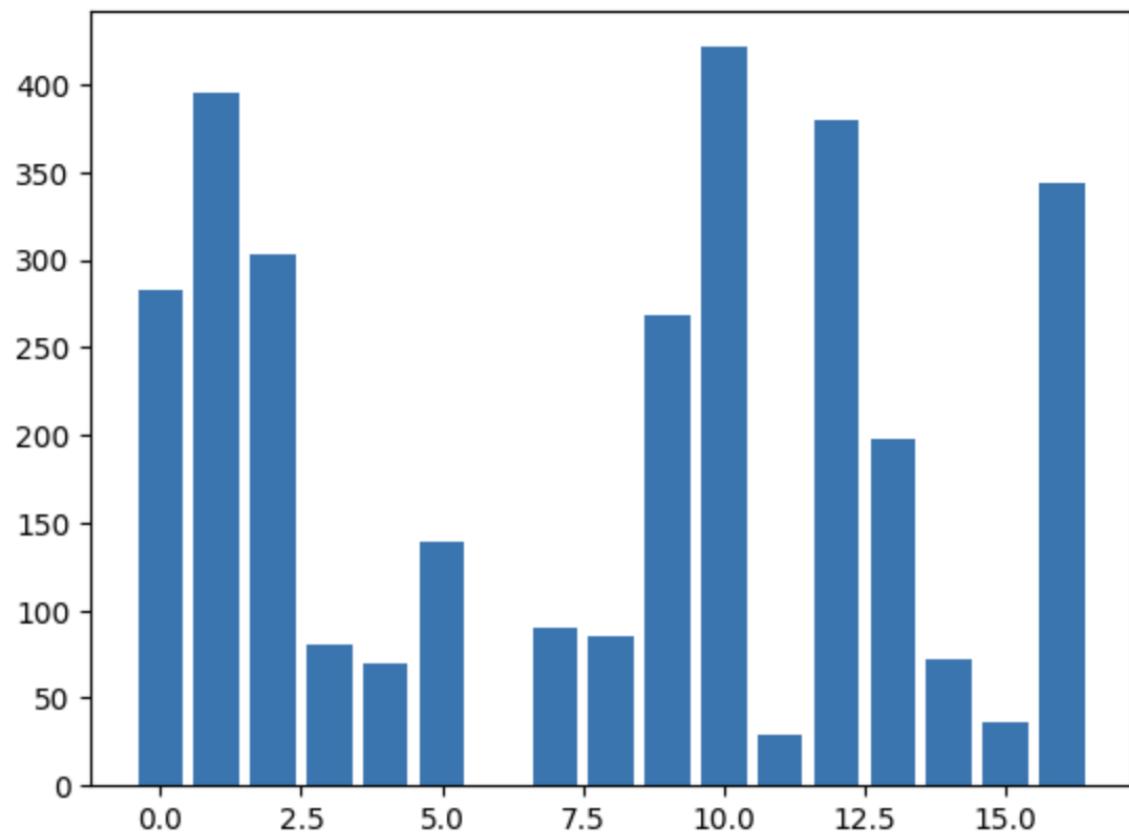
2. Logistic Regression
3. AdaBoost Classifier
4. Random Forest
5. KNN
6. Ensemble Methods

The following tables will give us more information on the best params and evaluation metrics we used for all the above algorithms:

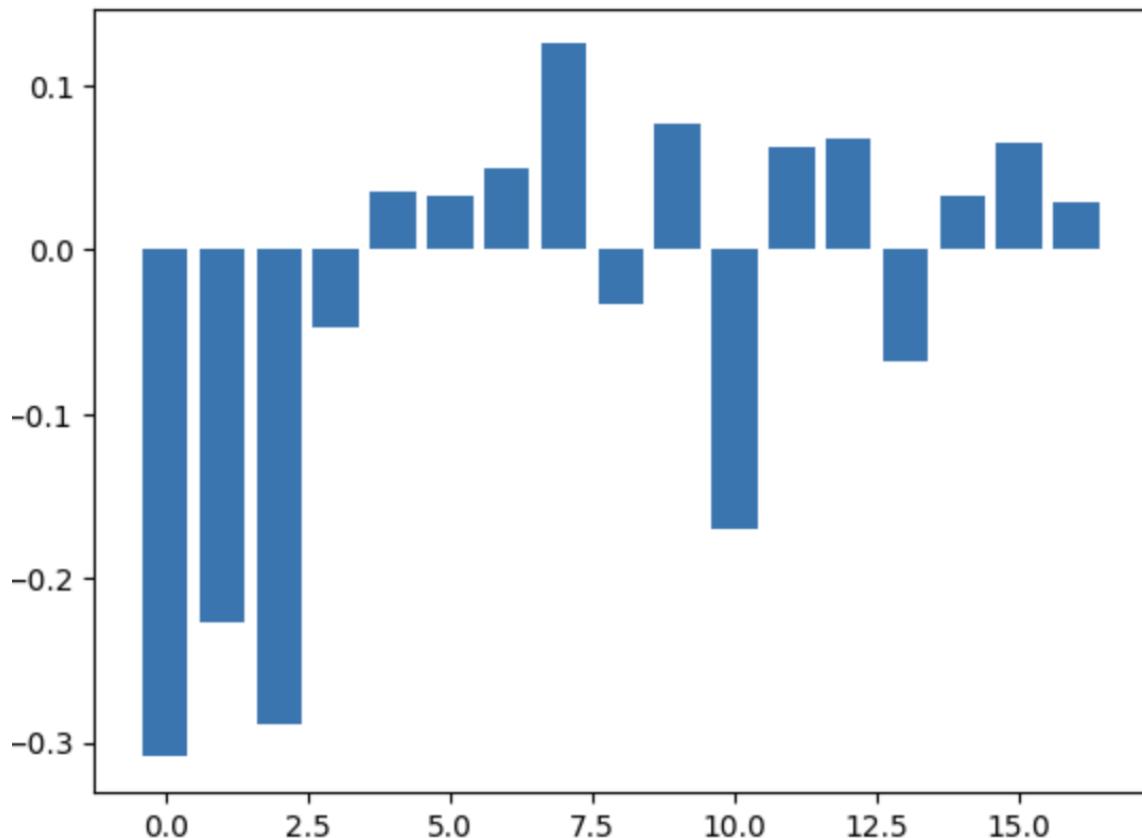
<b>Experiments</b>	<b>Pipeline</b>	<b>Parameters</b>	<b>TrainAcc</b>	<b>ValidAcc</b>	<b>TestAcc</b>	<b>Train Time(s)</b>
4	Baseline Multinomial NB with 17 inputs	{'clf_alpha': (1, 0.1, 0.01, 0.001, 0.0001, 1}	91.93%	92.02%	91.83%	0.056123
	Baseline Logistic Regression with 17 inputs	{'clf_solver': ['lbfgs', 'liblinear', 'newton']}	91.91%	92.01%	91.82%	0.025727
	Baseline AdaBoostClassifier with 17 inputs	{'clf_n_estimators': [1, 2]}	91.93%	92.02%	91.83%	0.019794
	Baseline Random Forest with 17 inputs	{'rf_n_estimators': [1, 10], 'rf_min_samples'}	98.56%	91.88%	91.57%	0.021411
	Baseline KNN with 17 inputs	{'rf_n_estimators': [1, 10], 'rf_min_samples'}	93.05%	91.88%	91.57%	0.019411
	Baseline Ensemble with 17 inputs	{'clf_lr_C': [0.01], 'clf_lr_penalty': ['l1']}	91.97%	91.88%	91.57%	0.017676

### **Feature Importance:**

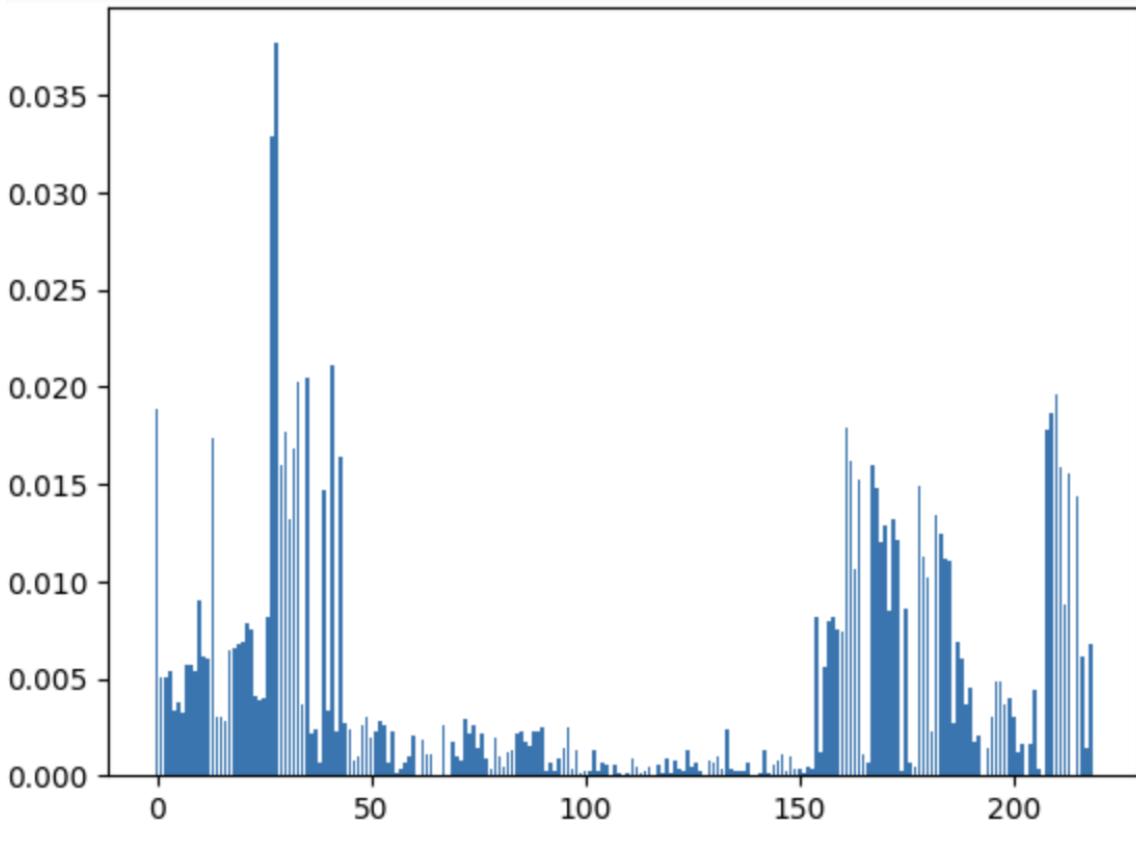
#### **1. Multinomial NB**



2. Logistic Regression



3. Random Forest



## Success/Failure analysis

Experiments	Pipeline	Parameters	TrainAcc	ValidAcc	TestAcc	Train Time(s)	Test Time(s)	Train AUC
2 Baseline Pipeline(steps=[('rf', RandomForestClassifier())]) with 100 inputs		{'rf__n_estimators': [20], 'rf__criterion': ['gini', 'log_loss']}	99.39%	91.92%	91.56%	1.503872	0.915629	0.999999

Baseline Pipeline(steps=[('dt', DecisionTreeClassifier())]) with 100 inputs {'dt\_\_max\_depth': [5, 10]} 91.98% 91.94% 91.60% 0.055064 0.916038 0.709815

Baseline Pipeline(steps=[('lr', LogisticRegression())]) with 100 inputs {'lr\_\_C': [0.01], 'lr\_\_penalty': ['l1', 'l2']} 91.97% 91.96% 91.60% 0.032885 0.916038 0.736215

Experiments	Pipeline	Parameters	TrainAcc	ValidAcc	TestAcc	Train Time(s)	Test Time(s)	Train AUC	Valid AUC	Test AUC	Best Params
3 Baseline Multinomial NB with 219 inputs		{'clf_alpha': (1, 0.1, 0.01, 0.001, 0.0001, 1)}	91.93%	92.02%	91.83%	0.031847	0.009847	0.623854	0.627888	0.628230	{'memory': None, 'steps': [('scaler', MinMaxSc
Baseline Logistic Regression with 219 inputs		{'clf_solver': ['lbfgs', 'liblinear', 'newton']}	91.93%	92.02%	91.85%	0.030177	0.009847	0.755586	0.756720	0.742818	{'memory': None, 'steps': [('scaler', Standard
Baseline AdaBoostClassifier with 219 inputs		{'clf_n_estimators': [1, 2]}	91.93%	92.02%	91.83%	0.031300	0.009847	0.586557	0.590228	0.587076	{'memory': None, 'steps': [('scaler', Standard
Baseline Random Forest with 219 inputs		{'rf_n_estimators': [1, 10]}	98.55%	91.96%	91.74%	0.034335	0.009847	0.999818	0.642760	0.632850	{'memory': None, 'steps': [('scaler', MinMaxSc
Baseline KNN with 219 inputs		{'rf_n_estimators': [1, 10], 'rf_min_samples.'	92.97%	91.96%	91.74%	0.020069	0.009847	0.939106	0.551639	0.550551	{'memory': None, 'steps': [('scaler', Standard

Experiments	Pipeline	Parameters	TrainAcc	ValidAcc	TestAcc	Train Time(s)	Test Time(s)	Train AUC	Valid AUC	Test AUC	Best Params
4	Baseline Multinomial NB with 17 inputs	{'clf_alpha': (1, 0.1, 0.01, 0.001, 0.0001, 1}	91.93%	92.02%	91.83%	0.056123	0.010725	0.652961	0.655879	0.659160	{'memory': None, 'steps': [('scaler', MinMaxSc
	Baseline Logistic Regression with 17 inputs	{'clf_solver': ['lbfgs', 'liblinear', 'newton']}	91.91%	92.01%	91.82%	0.025727	0.010725	0.731534	0.734448	0.728382	{'memory': None, 'steps': [('scaler', Standard
	Baseline AdaBoostClassifier with 17 inputs	{'clf_n_estimators': [1, 2]}	91.93%	92.02%	91.83%	0.019794	0.010725	0.645804	0.644956	0.639363	{'memory': None, 'steps': [('scaler', Standard
	Baseline Random Forest with 17 inputs	{'rf_n_estimators': [1, 10], 'rf_min_samples': [1, 2]}	98.56%	91.88%	91.57%	0.021411	0.010725	0.999718	0.646205	0.642093	{'memory': None, 'steps': [('scaler', MinMaxSc
	Baseline KNN with 17 inputs	{'rf_n_estimators': [1, 10], 'rf_min_samples': [1, 2]}	93.05%	91.88%	91.57%	0.019411	0.010725	0.967684	0.556668	0.565649	{'memory': None, 'steps': [('scaler', Standard
	Baseline Ensemble with 17 inputs	{'clf_lr_C': [0.01], 'clf_lr_penalty': ['l1']}	91.97%	91.88%	91.57%	0.017676	0.010725	0.773465	0.556668	0.565649	{'memory': None, 'steps': [('scaler', Standard

## Statistical Significance of Results.

In the previous phase, we had already provided the best baseline model that was working on subset of a dataset using all the features available in the dataset.

In the initial experiment that we performed where we selected entire dataset using only the selected features that we obtained we observed that Decision Tree and logistic regression had the best test accuracy of 91.60% and training time of 1.503872

In the second experiment, we used feature engineering to treat the missing values and used the selected features that had highest correlation with the target variable in the application\_train.csv without adding the 17 newly generated features. We observed that logistic regression performed the best with test accuracy of 91.85% and had a train time of 0.030177. The best parameter are clf\_fit\_intercept: True and clf\_solver:'lbfgs'

In the third experiment, we used feature engineering and hyper-parameter tuning and added the new feature that we generated to the dataset to obtain the best model as AdaBoost and Multinomial NB with test accuracy of 91.83% and train time of 0.056123. For Multinomial NB the best parameter are clf\_alpha: 1 and clffit\_prior: True and for AdaBoost are clf\_n\_estimator: 1.

## Gap Analysis

In the previous phase, the best baseline model accuracy that we achieved was for the model Decision Tree which was 93.46% using all the available features in the dataset.

In the initial experiment that we performed where we selected entire dataset using only the selected features that we obtained we observed that Decision Tree and logistic regression had the best test accuracy of 91.60%

In the second experiment, we used feature engineering to treat the missing values and used the selected features that had highest correlation with the target variable in the

application\_train.csv without adding the 17 newly generated features. We observed that logistic regression performed the best with test accuracy of 91.85%

In the third experiment, we used feature engineering and hyper-parameter tuning and added the new feature that we generated to the dataset to obtain the best model as AdaBoost and Multinomial NB with test accuracy of 91.83%

The best pipeline that we achieved out of all the experiments we performed was when we used feature engineering to treat the missing values and used the selected features that had highest correlation with the target variable in the application\_train.csv without adding the 17 newly generated features in which Logistic Regression Model performed with the highest accuracy of all the experiments that was 91.85%

We can see that there is a drop in the accuracy of the model which because in the experiments performed in the phase 2 we have used entire dataset instead of using subset of the data and we come to the conclusion that, after feature engineering and hyperparameter tuning the model performs best when the newly generated feature are not considered while training the model (in this case our model had the highest test accuracy).

## Kaggle Submission

We got the following scores after Kaggle Submission:

Submission	Private Score	Public Score	Status
submission.csv	0.73315	0.73762	Selected
submission.csv	0.66457	0.65961	
submission.csv	0.65139	0.66622	

## Discussion:

**Experiment 2: Using Selected Features (Using Entire Dataset)** In experiment 2, we considered the entire dataset with 100 features, we found that the model with the highest test accuracy is the Decision Tree Classifier and Logistic Regression with an accuracy of 91.60%. Validation accuracy is highest for Logistic

Regression with an accuracy of 91.96%. Train accuracy is highest for Random Forest Classifier with an accuracy of 99.39%. The highest Train AUC is for Random Forest Classifier with a score of 0.999999. Logistic Regression has the highest validation and test score of 0.736477 and 0.735735 respectively. Therefore, the model that performs the best in experiment 2 is Logistic Regression and Decision Tree Classifier. The best params used for Logistic Regression and Decision Tree Classifier are {'memory': None, 'steps': [('lr', LogisticRegression(C=0.01))], 'verbose': False, 'lr': LogisticRegression(C=0.01), 'lr\_C': 0.01, 'lrclass\_weight': None, 'lr\_dual': False, 'lrfit\_intercept': True, 'lrintercept\_scaling': 1, 'lrl1\_ratio': None, 'lrmrmax\_iter': 100, 'lrmulti\_class': 'auto', 'lrn\_jobs': None, 'lrapenalty': 'l2', 'lrrandom\_state': None, 'lrsolver': 'lbfgs', 'lrtol': 0.0001, 'lrvrbose': 0, 'lrwarm\_start': False} and {'memory': None, 'steps': [('dt', DecisionTreeClassifier(max\_depth=5))], 'verbose': False, 'dt': DecisionTreeClassifier(max\_depth=5), 'dtccp\_alpha': 0.0, 'dtclass\_weight': None, 'dtcriterion': 'gini', 'dtmax\_depth': 5, 'dtmax\_features': None, 'dtmax\_leaf\_nodes': None, 'dtmin\_impurity\_decrease': 0.0, 'dtmin\_samples\_leaf': 1, 'dtmin\_samples\_split': 2, 'dt\_min\_weight\_fraction\_leaf': 0.0} respectively.

**Experiment 3: Using Feature Engineering (Without New Features)** In experiment 3, we considered 219 features and implemented five algorithms. The algorithm that works the best is Logistic Regression with an accuracy of 91.85%. The models with the highest validation accuracy are for the AdaBoost classifier and Logistic regression of 92.02%. Train accuracy is highest for random forest with an accuracy of 98.55%. Test AUC is highest for Logistic regression with a score of 0.742. Validation AUC is highest for Logistic Regression with a score of 0.756. Train AUC is highest for KNN with a score of 0.939. Therefore, the model that performs the best in experiment 3 is Logistic Regression. The best params used for Logistic Regression are {'memory': None, 'steps': [('scaler', StandardScaler())]}.

**Experiment 4 (Using Feature Engineering (With 17 New Features))** In experiment 3, we considered 17 features and implemented 6 models. Train accuracy is highest for the random forest model with an accuracy of 98.56%. Validation accuracy is highest for the AdaBoost classifier and Multinomial NB with an accuracy of 92.02%. The test accuracy is highest for the AdaBoost classifier and Multinomial NB with an accuracy of 91.83%. Test AUC is highest for Logistic Regression with a score of 0.728. Valid AUC is highest for Logistic Regression with a score of 0.734. Train AUC is highest for KNN with a score of 0.967. Therefore, the model that performs the best in experiment 4 is the AdaBoost classifier and Multinomial NB with an accuracy of 91.83%. The best params used for AdaBoost classifier and Multinomial are {'memory': None, 'steps': [('scaler', StandardScaler())]} and {'memory': None, 'steps': [('scaler', MinMaxScaler())]}.

**Impact of new features in the model:** We can see that the added new features don't work as we expected as the accuracy of the models has decreased from 91.85% to 91.83%. However, the decrease in accuracy is very minor.

**Comparing experiments 2, 3, and 4:** After comparing all the results, we got our best performance found in experiment 3 where we did feature engineering. As part of feature engineering, we took care of high proportions of zero. Values, treated missing values, divided the main frame into a categorical and numerical data frame and took into consideration the correlation of the input data frame with respect to the target variable. The best perfuming model in experiment 3 is Logistic Regression.

## Conclusion

The project aims to provide insights into the key factors that contribute to default risk and to develop a reliable ML model that can be used to predict default risk with high accuracy. ML pipelines that incorporate custom features, in addition to the standard features, can improve the accuracy of predicting the risk of default on loans. In phase3, we merged, cleaned data, extracted relevant features, and created new ones from the most correlated features. Next, we featured data performing OHE and applied imputing methods. We performed hyperparameter tuning using K-Fold cross validation and GridSearchCV which contributed to improving the test. With feature engineering and added features, AdaBoost and MultinomialNB models achieved 91.83% accuracy. Logistic Regression achieved 91.85% accuracy with feature engineering only. Future plans include applying deep learning techniques, such as developing artificial neural networks, for improved prediction results.

## Kaggle Submission

Please provide a screenshot of your best kaggle submission.

The screenshot should show the different details of the submission and not just the score.

The screenshot shows the Kaggle interface for the 'Home Credit Default Risk' competition. The left sidebar includes links for Create, Home, Competitions, Datasets, Models, Code, Discussions, Learn, and More. Under 'Your Work', there are sections for Recently Viewed (Home Credit Default..., Deloitte Media Con...) and Recently Edited (Regularization with ..., Disease Symptom ...). The main content area displays the competition details: 'Featured Prediction Competition', 'Home Credit Default Risk', 'Can you predict how capable each applicant is of repaying a loan?', 'Prize Money \$70,000', 'Home Credit Group - 7178 teams - 5 years ago', and tabs for Overview, Data, Code, Discussion, Leaderboard, Rules, Team, Submissions, Late Submission, and ...'. A message indicates '0/2' submissions have been evaluated for final score. The 'Submissions' table lists three entries: 'submission.csv' (Complete (after deadline) - now - Group15 AML Logistic Regression), 'Private Score 0.73315', 'Public Score 0.73762', and a checkbox; 'submission.csv' (Complete (after deadline) - 7h ago - group 15 AML), 'Private Score 0.66457', 'Public Score 0.65961', and a checkbox; and 'submission.csv' (Complete (after deadline) - 7h ago - Submission from Group 15 AML), 'Private Score 0.65139', 'Public Score 0.66622', and a checkbox.

# References

Some of the material in this notebook has been adopted from [here](#)

In [ ]:

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

Read the following:

- feature engineering via Featuretools library:
  - <https://github.com/Featuretools/predict-loan-repayment/blob/master/Automated%20Loan%20Repayment.ipynb>
- <https://www.analyticsvidhya.com/blog/2018/08/guide-automated-feature-engineering-featuretools-python/>
- feature engineering paper: [https://dai.lids.mit.edu/wp-content/uploads/2017/10/DSAA\\_DSM\\_2015.pdf](https://dai.lids.mit.edu/wp-content/uploads/2017/10/DSAA_DSM_2015.pdf)
- <https://www.analyticsvidhya.com/blog/2017/08/catboost-automated-categorical-data/>