

# Bank Credit Application Default Risk Prediction

## Team Members

1. Priya Enuganti ([priyaenuganti@my.unt.edu](mailto:priyaenuganti@my.unt.edu))
2. Nayana Shrestha ([nayanashrestha@my.unt.edu](mailto:nayanashrestha@my.unt.edu))
3. Roshan Sah ([RoshanSah@my.unt.edu](mailto:RoshanSah@my.unt.edu))
4. Brayan Murillo ([brayanmurillogutierrez@my.unt.edu](mailto:brayanmurillogutierrez@my.unt.edu))

## • Introduction

Bank loan default is a famous example of how machine learning models may be used to anticipate hazardous consumers and hence reduce lender losses. Because the financial industry is heavily regulated, any model or classification of clients based on their behavior, demographics, or other factors must be disclosed to authorities to ensure fair operations.

For both companies and people, loan lending has played a significant role in their everyday lives. Due to increased competition in the financial sector and a large number of financial restraints, taking out a loan has become unavoidable. Individuals all across the world rely on loan lending for a variety of reasons, including overcoming financial restraints and achieving personal ambitions. Similarly, banks and other small to large businesses rely on loan lending for the essential purpose of managing their affairs and operating efficiently during times of financial restriction.

Even though it is profitable and helpful for both lenders and borrowers. It does, however, come with a significant risk, which is referred to as Loan risk in the context of loan lending. Credit scores are assigned to individuals by industry professionals and researchers all around the globe to assess their risk and creditworthiness.

Machine learning algorithms have been used to calculate and forecast credit risk based on an individual's historical data for many years. This project examines the current literature on models that employ Machine Learning algorithms to predict default loan.

## • Background

In the paper [2], the author addresses the risks that banks face every day with respect to bank loans, the losses that a bank faces when rejecting an applicant who might have paid the loan, and on the other hand, the losses associated with loans approved to those who failed to repay the loan.

Given the importance of the case study, the author proposes to carry out an EDA to find out which are the characteristics that stand out most in an applicant who failed to repay his or her loans.

For this purpose, the author makes use of two tables, known as current applications and previous applications, which together add up to almost two million records, where the first table has 122 columns and the second has 37.

The article also describes how to identify all those customers who failed to pay (target value) by combining some columns, (such as the date of the last scheduled payment compared to the date of the last payment received).

The article was done in the Python programming language and the main libraries used were pandas (table processing), matplotlib (graphing) and some mathematical packages (it does not make use of RDBMS or other database related programs).

The author made an analysis of the null values that each column had, he/she got rid of those that had more than 40% of their values as null. He/she also graphed the null values of the tables.

In addition, the author also made an analysis of the contribution that each variable had in the column describing whether they were payers or defaulters, this was done by correlation and heat maps.

After removing all the unwanted columns, the author applied various filters and/or techniques on the remaining columns to help with the analysis. The author grouped by range some numerical values such as applicant income, amount of credit requested, age, among others. These were delimited by intervals according to what was to be demonstrated.

In addition, the article shows imputation methods (from the pandas library) used to fill in some null values in the columns and to validate that the imputation was appropriate, the distribution was plotted after applying several types of imputation (mean, mode, ...) to make sure that it did not considerably modify the original distribution of the data. He also plots some whisker boxes to find those atypical data that could redirect the reliability of the data.

The author also plots the categorical variables, without converting them into any numerical value to represent them, to make an analysis of the proportions according to the values taken by the variables.

At the end, the author performs a univariate numerical analysis, a bivariate analysis and a comparative analysis between the previous applications and the current applications.

On the other hand, the article [3] talks about the different models that are applied in the industry to identify defaulters. The article discusses the nature of these models, their validity and reliability depending on the source of the case study.

However, the article focuses more on the linear regression machine learning model. The author also discusses how these types of models are more sensitive and specific and how they take more information than the financial statement of the customer's bank accounts.

He then discusses how an approximate logistic regression model can easily determine the probability that a customer will or will not repay a loan.

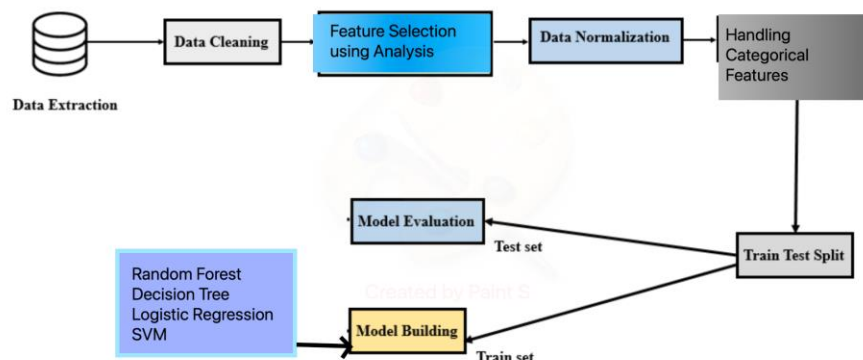
In the end, the paper suggests additional values or attributes that banks should consider about the applicant when deciding whether to approve a loan.

The article [4] makes use of the "Tobit Technique" also known as the "censored regression model" which is useful for finding the linear relationship between variables by applying censoring above or below those values that exceed thresholds.

Finally, article [5] performs an analysis of college credit, applying multilevel regression models, and describes how socioeconomic, family, unemployment, and career type are factors that influence loan rates and the number of students who default on their student loans.

## • Our Model

### Architecture Diagram with explanation



The architecture of the project is a step-by-step procedure we took to predict the loan risk analysis

### Data Extraction:

The data is collected from kaggle but in real time the data is the input from a loan applicant which will be used to decide whether to approve loan application.

### Data Cleaning:

During this process the missing values from data is handled in different ways I.e features with more than 90% of missing values will be completely discarded as they would be a noise to our final classifier. Features with few missing values are imputed with mean values.

### **Feature Selection:**

Here the most prominent features that has a high impact I.e., correlation to the outcome are filtered out. There are several methods used for this such as drawing a correlation heat map and principal component analysis(PCA) and found the top 18 significant features from 122 Features.

### **Data Normalization:**

In machine learning, normalization is a data preparation method that is commonly utilized. Normalization is the process of converting the columns in a dataset to the same scale. Machine learning does not need the normalization of every dataset. When the ranges of features are different, it is just necessary. Our dataset have fluctuated value of different column. So, to make the performance of the model better, we normalize the dataset. There are many ways to normalize the data but most widely used are Standarization scaling, minimax scaling.

### **Handling Categorical Value**

Any organized dataset would often include numerous columns containing a mix of numerical and category information. Only numbers can be understood by a machine. It is unable to comprehend the text and that holds true for Machine Learning algorithms as well. That is why, for a machine learning algorithm to grasp it, we need to transform categorical columns to numerical columns. The process of turning categories to numbers is known as categorical encoding.

In our dataset, "NAME\_HOUSING\_TYPE", "CODE\_GENDER", "NAME\_FAMILY\_STATUS", "ORGANIZATION\_TYPE", "NAME\_EDUCATION\_TYPE", "NAME\_INCOME\_TYPE", "OCCUPATION\_TYPE" have categorical values. For this we have performed the label encoding and one-hot encoding to change the categorical value into numerical values which will be shown in the implementation section.

### **Train Test Split:**

The dataset is split into training and test dataset with a 70 to 30 ratio respectively. The training dataset will be used for the model to learn whereas testing data is used for evaluating the performance of the classifier.

### **Workflow diagram with explanation**



---

## Data Cleaning

The data collected from Kaggle is structured with few missing/null values. We will use the data cleaning/imputation techniques for the missing values imputation. We may also need to remove some features which have missing values for more than 50% of the data.

## Data Analysis

Apache spark RDD/data frame/sql to do exploratory data analysis to explore business insights from the data. Cassandra is also used to filter and query the required information. Data analysis done in detail using spark's transformations, actions, aggregation API's and SQL queries.

## Visualization

The feature analysis is visualized using various plots like bar graphs, pie charts and scatter plots to give a better understanding of how the features impact the outcome. This further helped in feature selection before building a classifier.

## ML Classifier

To make a smart business decision, investors require a more comprehensive assessment of the loan data provided by financial sectors. A classifier is built using data mining techniques and Machine Learning algorithms, to predict new applicants that would default on their loans. Dataset is split for both training and testing dataset. Each of us developed a different classifier

(Logistic Regression, decision tree,SVM and Random Forest) and finalized decision tree as it gave high performance.

## Dataset

### Detailed description of Dataset

Dataset is taken from Kaggle. It is a huge amount of data with 122 features with over 280k records of loan applicants. A few of the key features are gender, work experience, credit history, Loan Amount, Family status, Marital status etc. The dataset is imbalanced I.e., its highly skewed towards repayers which is case in real scenario. It also had a few missing data in features which were addressed during pre-processing.

<https://www.kaggle.com/datasets/mishra5001/credit-card>

The purpose is to analyze and find the driving factors behind loan default, i.e. the features which are strong indicators of default. The financial services can utilize this knowledge for risk assessment for all their loan applications.

The dataset can be viewed below to have an understanding on what it looks like.

AutoSave

application\_data

HomeInsertDrawPage LayoutFormulasDataReviewViewTell me

Calibri (Body)12

Wrap Text

Merge & Center

General

Conditional Formatting

Format Painter

Cell Styles

Insert

Delete

Format

Sort & Filter

Find & Select

Analyze Data

Sensitivity

Possible Data Loss

Some features might be lost if you save this workbook in the comma-delimited (.csv) format. To preserve these features, save it in an Excel file format.

Save As...

	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V
1	SK_ID_CURR	TARGET	NAME_CONC	CODE_GEND	FLAG_OWN	FLAG_OWN_CNT_CHILD	AMT_INCOM	AMT_CREDIT	AMT_ANNU	GOOD	NAME_TYPE	NAME_INCO	NAME_EDUC	NAME_F	NAME_HOU	REGION	POI	DAYS_BIRTH	EMPLE_DAYS	REGIS_DAYS	ID_PU_OWN_CAR	
2	100002	0	Cash loans	M	N	Y	0	202500	400597.5	24700.5	351000	Uncompar	Working	Secondary / Single / not	House / apar	0.018801	9461	637	3648	-2120		
3	100003	0	Cash loans	F	N	N	0	270000	1293502.5	35698.5	1129000	Family	State servan	Higher educ	Married	House / apar	0.003541	-16765	-1188	-1186	-291	
4	100004	0	Revolving loa	M	Y	Y	0	67500	135000	6750	135000	Uncompar	Working	Secondary / Single / not	House / apar	0.010032	-19046	-225	-4260	-2531		
5	100006	0	Cash loans	F	N	Y	0	135000	312682.5	29686.5	297000	Uncompar	Working	Secondary / Civil marriag	House / apar	0.008019	-19005	-3039	-9833	-2437		
6	100007	0	Cash loans	M	N	Y	0	121500	513000	21865.5	513000	Uncompar	Working	Secondary / Single / not	House / apar	0.028663	-19932	-3038	-4311	-3458		
7	100008	0	Cash loans	M	N	Y	0	90000	490495.5	27517.5	454500	Spouse, part	State servan	Secondary / Married	House / apar	0.035792	-16941	-1588	-4970	-477		
8	100009	0	Cash loans	F	Y	Y	1	171000	1560726	41301	1395000	Uncompar	Commercial	Higher educ	Married	House / apar	0.035792	-13778	-3130	-1213	-619	
9	100010	0	Cash loans	M	Y	Y	0	360000	1530000	42075	1530000	Uncompar	State servan	Higher educ	Married	House / apar	0.003122	-18850	449	-4097	-2378	
10	100011	0	Cash loans	F	N	Y	0	112500	1019610	33826.5	913500	Children	Pensioner	Secondary / Married	House / apar	0.018634	-20059	365243	-7427	-3514		
11	100012	0	Revolving loa	M	N	Y	0	135000	405000	20250	405000	Uncompar	Working	Secondary / Single / not	House / apar	0.019689	-14469	-2019	-14437	-3992		
12	100014	0	Cash loans	F	N	Y	1	112500	602500	21177	602500	Uncompar	Working	Higher educ	Married	House / apar	0.03228	-10197	679	-4427	-738	
13	100015	0	Cash loans	F	N	Y	0	38419.155	148365	10678.5	135000	Children	Pensioner	Secondary / Married	House / apar	0.015221	-20417	365243	-5246	-2512		
14	100016	0	Cash loans	F	N	Y	0	67500	80865	5881.5	67500	Uncompar	Working	Secondary / Married	House / apar	0.031329	-13439	-2717	-311	-3227		
15	100017	0	Cash loans	M	Y	N	1	225000	518468	28966.5	697500	Uncompar	Working	Secondary / Married	House / apar	0.016612	-14086	-3028	-643	-4911		
16	100018	0	Cash loans	F	N	Y	0	189000	773680.5	32778	679500	Uncompar	Working	Secondary / Married	House / apar	0.010006	-14583	-203	-615	-2056		
17	100019	0	Cash loans	M	Y	Y	0	157500	399772	20160	247500	Family	Working	Secondary / Single / not	Rented apart	0.020713	-8728	-1157	-3494	-1368		
18	100020	0	Cash loans	M	N	N	0	108000	509602.5	26149.5	387000	Uncompar	Working	Secondary / Married	House / apar	0.018634	-12931	-1317	-6392	-3866		
19	100021	0	Revolving loa	F	N	Y	1	81000	270000	13500	270000	Uncompar	Working	Secondary / Married	House / apar	0.010966	9776	-191	-4143	-2427		
20	100022	0	Revolving loa	F	N	Y	0	112500	137500	7875	157500	Other, A	Working	Secondary / Widow	House / apar	0.04622	-17718	-7804	8751	-1259		
21	100023	0	Cash loans	F	N	Y	1	90000	544491	17563.5	454500	Uncompar	State servan	Higher educ	Single / not	House / apar	0.015221	-11348	-2038	-1021	-3964	
22	100024	0	Revolving loa	M	Y	Y	0	135000	427500	21375	427500	Uncompar	Working	Secondary / Married	House / apar	0.015221	-18252	-4286	-298	-1800		
23	100025	0	Cash loans	F	Y	Y	1	202500	1132573.5	37561.5	927000	Uncompar	Commercial	Secondary / Married	House / apar	0.025164	-18185	-1652	-2299	-2299		
24	100026	0	Cash loans	F	N	N	1	450000	497520	32521.5	450000	Uncompar	Working	Secondary / Married	Rented apart	0.020713	-11416	-4306	-114	-2518		
25	100027	0	Cash loans	F	N	Y	0	81250	239850	23850	235000	Uncompar	Pensioner	Secondary / Married	House / apar	0.006296	-24827	805243	-9012	-3684		
26	100029	0	Cash loans	M	Y	N	2	135000	247500	12703.5	247500	Uncompar	Working	Secondary / Married	House / apar	0.026392	-11286	-746	-108	-3729		
27	100030	0	Cash loans	F	N	Y	0	90000	225000	11074.5	225000	Uncompar	Working	Secondary / Married	House / apar	0.028663	-19354	-3494	-2419	-2893		
28	100031	1	Cash loans	F	N	Y	0	112500	979992	27076.5	702000	Uncompar	Working	Secondary / Widow	House / apar	0.018029	-18774	-2628	-6573	-1827		
29	100032	0	Cash loans	M	N	Y	1	112500	327024	23827.5	270000	Family	Working	Secondary / Married	House / apar	0.019101	-15948	-1234	-5782	-3153		
30	100033	0	Cash loans	M	Y	Y	0	270000	700830	57676.5	675000	Uncompar	State servan	Higher educ	Single / not	House / apar	0.04622	-9954	-1796	-4668	-2661	
31	100034	0	Revolving loa	M	N	Y	0	90000	180000	9000	180000	Uncompar	Working	Higher educ	Single / not	With parents	0.030755	-10341	-1010	-4799	-3015	
32	100035	0	Cash loans	F	N	Y	0	292500	665892	24592.5	477000	Uncompar	Commercial	Secondary / Civil marriag	House / apar	0.025164	-15280	-2668	-5266	-3787		
33	100036	0	Cash loans	F	N	Y	0	112500	512064	25033.5	360000	Family	Working	Secondary / Civil marriag	House / apar	0.008075	-11144	-1104	-7846	-2904		
34	100037	0	Cash loans	F	N	N	0	90000	199008	20893.5	180000	Uncompar	Working	Secondary / Civil marriag	House / apar	0.010032	-12974	-4404	-7123	-4464		
35	100039	0	Cash loans	M	Y	N	1	360000	733315.5	39069	679500	Uncompar	Commercial	Secondary / Married	House / apar	0.015221	-11694	-2060	-3517	-3557		
36	100040	0	Cash loans	F	N	Y	0	135000	1125000	32895	1125000	Uncompar	State servan	Higher educ	Married	House / apar	0.019689	-15997	-4585	-5735	-4067	
37	100041	0	Cash loans	F	N	N	0	112500	48409.5	48409.5	450000	Uncompar	Working	Higher educ	Married	House / apar	0.008075	-12158	-1275	-6265	-2009	
38	100043	0	Cash loans	F	N	Y	2	198000	641173.5	23157	533500	Uncompar	Commercial	Secondary / Married	House / apar	0.01885	-17199	-768	-63	-735		
39	100044	0	Cash loans	M	N	Y	0	121500	454500	15151.5	454500	Uncompar	Working	Secondary / Married	House / apar	0.030755	-21077	-1288	-5474	-4270		

application\_data

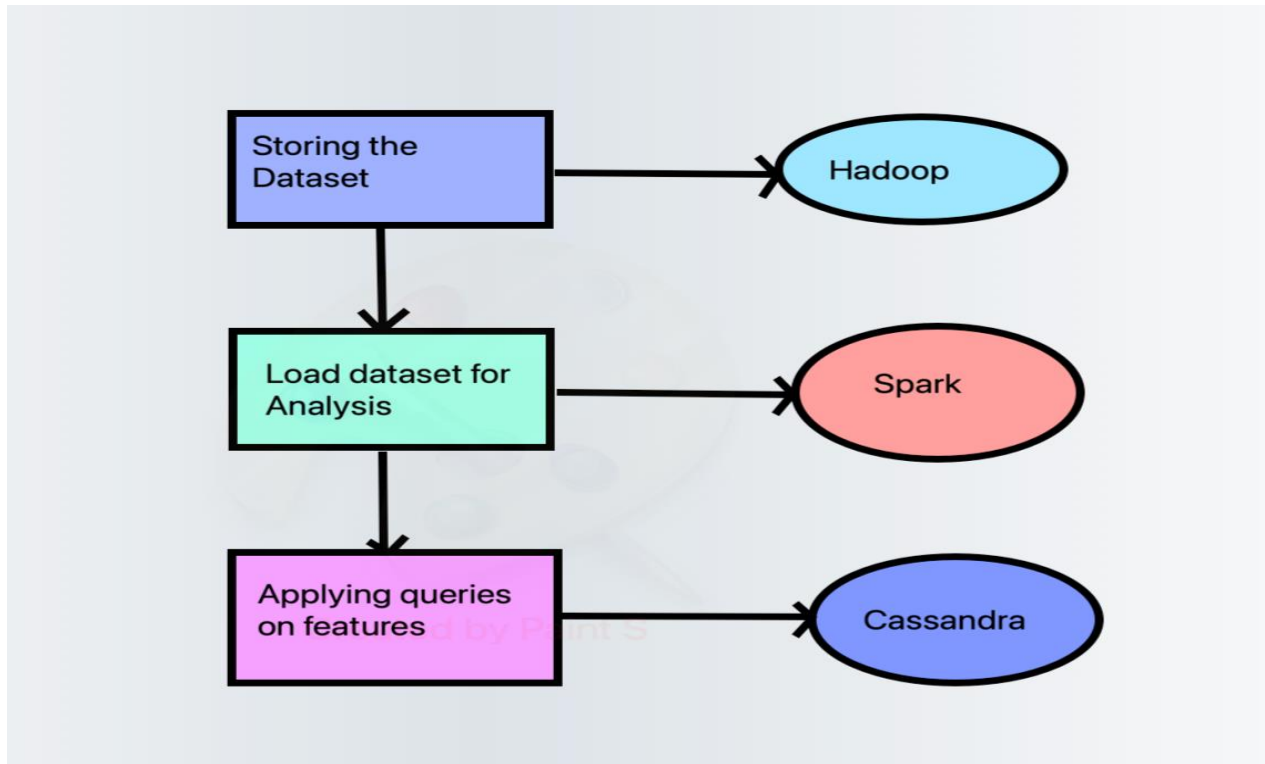
Ready

### Detail design of Features with diagram

Out of 122 features we selected 16 columns that we mostly related to the target. We use a heat map for this decision making. Then we divided the 20 columns among the four of us with five columns each to analyze them separately.

Functionality

Tools



For data analysis we have used various tools such as: Hadoop, Spark, Cassandra

Hadoop is used to store the dataset in HDFS server. This is considering the advantage of fault tolerance of hadoop which would help in not losing the data.

Spark is used to load the dataset from hadoop and perform all computations as it is relatively fast in doing so when compared to hadoop.

Cassandra is used for querying and filtering the data to perform analysis to check data imbalance and number of missing values from the data.

## **Analysis of data**

### **Data Pre-processing**

We did the statistical analysis of our dataset for each feature as shown in figure below. After the statistical analysis, we can perform this to transform the raw observation into information which we can understand and share. We can observe the mean, total number of count, minimum and maximum value of each feature of our dataset.

```
# selected variables for the demonstration
df.describe().show()
```

summary	_c0	NAME_HOUSING_TYPE	AMT_CREDIT	AMT_INCOME_TOTAL	CODE_GENDER	SK_ID_CURR	NAME_FAMILY_STATUS	CNT_CHILDREN	CNT_FAM_MEMBERS	ORGANIZATION_TYPE	DAYS_BIRTH	DAYS_EMPLOYED	AMT_GOOD_PRICE
count	307511	307511	307511	307511	307511	307511	307511	307511	307509	307511	307511	307511	307511
mean	153755.0	null	599025.9997057016	168797.9192969045	null	278180.51857657125	null	0.4170517477423572	2.152665450442101	null	-16036.995066843137	63815.04590404896	538396.0
stddev	88770.92365183546	null	402490.7769958496	237123.1462788521	null	102790.1753484231	null	0.7221213844376189	0.9106815691792911	null	4363.980631785574	141275.76651872683	369446.4
min	0	Co-op apartment	45000.0	25650.0	F	100002	Civil marriage	0	1.0	Advertising	-25229	-17912	-17912
max	307510	With parents	4050000.0	1.17E8	XNA	456255	Widow	19	20.0	XNA	-7489	365243	365243

**Figure: Dataset Description**

We did the correlation to find the degree of relationship between the features and target. From the diagram, we can clearly see that AMT\_CREDIT, SK\_ID\_CURR, CNT\_CHILDREN, CNT\_FAM\_MEMBERS and AMT\_GOOD\_PRICE are highly positively correlated with a value of 1, 0.099, 1, 1, 0.88, 0.88, 0.99 respectively. Some of the features are also negatively correlated and their correlation value are higher than 0.6 such as DAYS\_BIRTH and DAYS\_EMPLOYED. We can see these two features negatively correlated because they have negative datapoints.



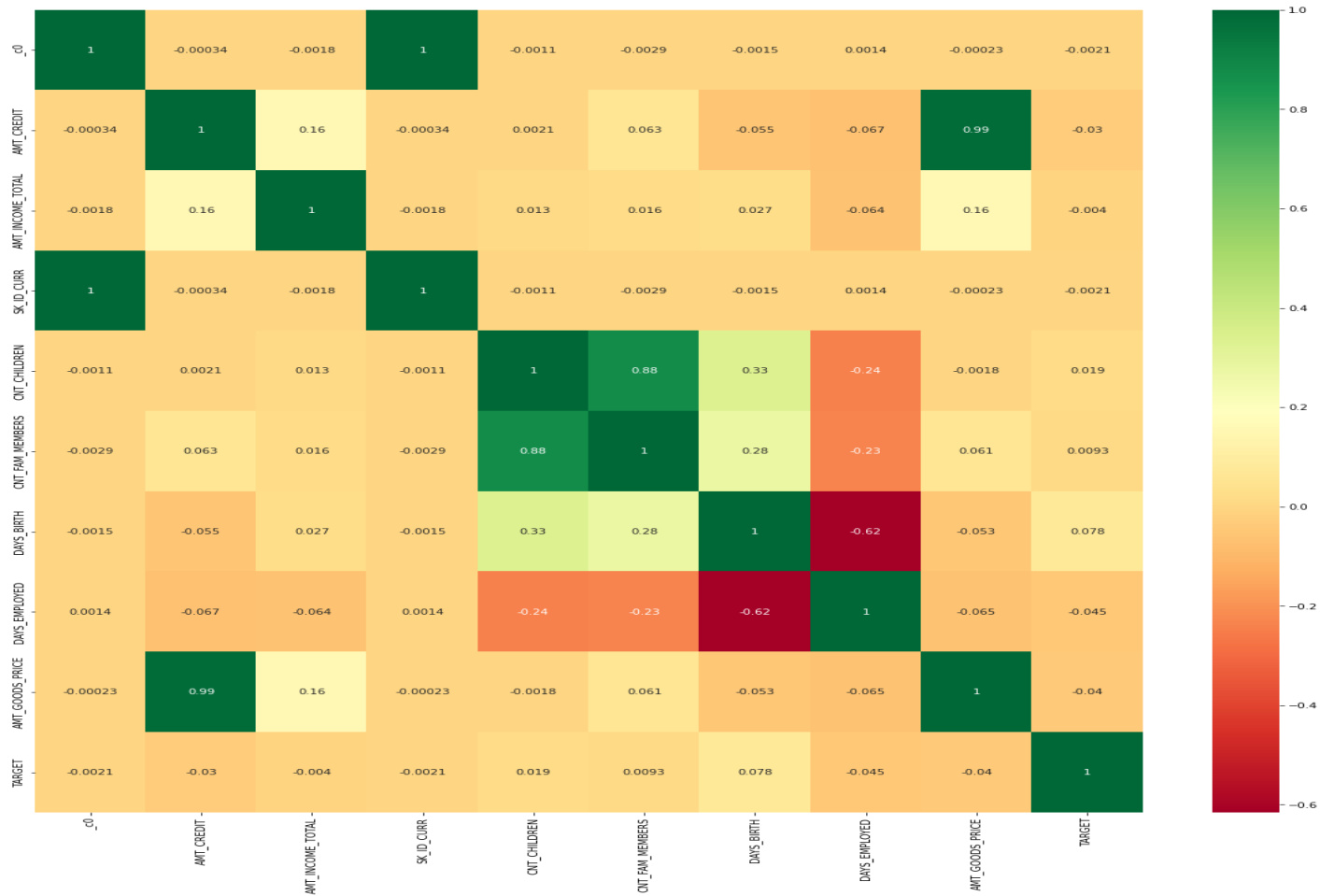


Figure: Correlation of features with target value

In the figure below, we find the null value of each column. So, we can fill the missing and null value to fit the model well. As we have more missing or Null value, our model will not perform well. Hence, we fill the missing and null value with the previous datapoint of that column using the 'bfill' function. At last, we check if there are any remaining missing or null values in our dataset.

```

## Get count of both null and missing values in pyspark

from pyspark.sql.functions import isnan, when, count, col
df.select([count(when(isnan(c) | col(c).isNull(), c)).alias(c) for c in df.columns]).show()

```

ME_HOUSING_TYPE	AMT_CREDIT	AMT_INCOME_TOTAL	CODE_GENDER	SK_ID_CURR	NAME_FAMILY_STATUS	CNT_CHILDREN	CNT_FAMILY_MEMBERS	ORGANIZATION_TYPE	DAYS_BIRTH	DAYS_EMPLOYED	AMT_GOODS_PRICE	NAME_EDUCATION_TYPE	NAME_INCOME_TYPE	OCCUPATION_TYPE	TARGET
0	0	0	0	0	0	0	2	0	0	0	278	0	0	96391	0

Figure: Finding Missing and Null Value

```
#filling the null and missing Value
df_m = df.na.fill(value='bfill',subset=["AMT_GOODS_PRICE"])
df_m.show(5)
df_m = df.na.fill("unknown",["OCCUPATION_TYPE"])
df_m.show(5)
```

_c0	NAME_HOUSING_TYPE	AMT_CREDIT	AMT_INCOME_TOTAL	CODE_GENDER	SK_ID_CURR	NAME_FAMILY_STATUS	CNT_CHILDREN	CNT_FAM_MEMBERS	ORGANIZATION_TYPE	DAYS_BIRTH	DAYS_EMPLOYED	AMT_GOODS_PRICE	NAME_EDUCATION_TYPE	NAME_INCOME_TYPE	OCCUPATION
0	House / apartment	406597.5	202500.0	M	100002	Single / not married	0	1.0	Business Entity T...	-9461	-637	351000.0	Secondary / secon...	Working	Lat
1	House / apartment	1293502.5	270000.0	F	100003	Married	0	2.0	School	-16765	-1188	1129500.0	Higher education	State servant	Core
2	House / apartment	135000.0	67500.0	M	100004	Single / not married	0	1.0	Government	-19046	-225	135000.0	Secondary / secon...	Working	Lat
3	House / apartment	312682.5	135000.0	F	100006	Civil marriage	0	2.0	Business Entity T...	-19005	-3039	297000.0	Secondary / secon...	Working	Lat
4	House / apartment	513000.0	121500.0	M	100007	Single / not married	0	1.0	Religion	-19932	-3038	513000.0	Secondary / secon...	Working	Core

only showing top 5 rows

_c0	NAME_HOUSING_TYPE	AMT_CREDIT	AMT_INCOME_TOTAL	CODE_GENDER	SK_ID_CURR	NAME_FAMILY_STATUS	CNT_CHILDREN	CNT_FAM_MEMBERS	ORGANIZATION_TYPE	DAYS_BIRTH	DAYS_EMPLOYED	AMT_GOODS_PRICE	NAME_EDUCATION_TYPE	NAME_INCOME_TYPE	OCCUPATION
0	House / apartment	406597.5	202500.0	M	100002	Single / not married	0	1.0	Business Entity T...	-9461	-637	351000.0	Secondary / secon...	Working	Lat
1	House / apartment	1293502.5	270000.0	F	100003	Married	0	2.0	School	-16765	-1188	1129500.0	Higher education	State servant	Core
2	House / apartment	135000.0	67500.0	M	100004	Single / not married	0	1.0	Government	-19046	-225	135000.0	Secondary / secon...	Working	Lat
3	House / apartment	312682.5	135000.0	F	100006	Civil marriage	0	2.0	Business Entity T...	-19005	-3039	297000.0	Secondary / secon...	Working	Lat
4	House / apartment	513000.0	121500.0	M	100007	Single / not married	0	1.0	Religion	-19932	-3038	513000.0	Secondary / secon...	Working	Core

only showing top 5 rows

Figure: Filling the Missing and Null Value

```
[ ] #checking which column have null value again to ensure we have any remaining null value
df_clean.isnull().any()
```

_c0	False
NAME_HOUSING_TYPE	False
AMT_CREDIT	False
AMT_INCOME_TOTAL	False
CODE_GENDER	False
SK_ID_CURR	False
NAME_FAMILY_STATUS	False
CNT_CHILDREN	False
CNT_FAM_MEMBERS	False
ORGANIZATION_TYPE	False
DAYS_BIRTH	False
DAYS_EMPLOYED	False
AMT_GOODS_PRICE	False
NAME_EDUCATION_TYPE	False
NAME_INCOME_TYPE	False
OCCUPATION_TYPE	False
TARGET	False
dtype:	bool

Figure: Checking the missing and Null Value

Label encoding is the process of translating labels into numeric format so that they may be read by machines. Machine learning algorithms can then better decide how those labels should be used. In supervised learning, it is a crucial pre-processing step for the structured dataset. From the figure below, we can see that the categorical value is converted into numerical values using the label encoder.

```
[ ] #Label Encoder
le = LabelEncoder()
cols = ["NAME_HOUSING_TYPE", "CODE_GENDER", "NAME_FAMILY_STATUS", "ORGANIZATION_TYPE", "NAME_EDUCATION_TYPE", "NAME_INCOME_TYPE", "OCCUPATION_TYPE"]

# Encode labels of multiple columns at once
df_pd[cols] = df_pd[cols].apply(LabelEncoder().fit_transform)

# Print head
df_pd.head()
```

NAME_HOUSING_TYPE	AMT_CREDIT	AMT_INCOME_TOTAL	CODE_GENDER	SK_ID_CURR	NAME_FAMILY_STATUS	CNT_CHILDREN	CNT_FAM_MEMBERS	ORGANIZATION_TYPE	DAYS_BIRTH	DAYS_EMPLOYED	AMT_GOODS_PRICE	NAME_EDUCATION_TYPE	NAME_INCOME_TYPE	OCCUPATION
1	406597.5	202500.0	1	100002	3	0	1.0	5	-9461	-637	351000.0	4	7	
1	1293502.5	270000.0	0	100003	1	0	2.0	39	-16765	-1188	1129500.0	1	4	
1	135000.0	67500.0	1	100004	3	0	1.0	11	-19046	-225	135000.0	4	7	
1	312682.5	135000.0	0	100006	0	0	2.0	5	-19005	-3039	297000.0	4	7	
1	513000.0	121500.0	1	100007	3	0	1.0	37	-19932	-3038	513000.0	4	7	

Figure: Label Encoder

The technique of converting data into the range [0, 1] (or any other range) or simply transforming data onto the unit sphere is known as normalization in machine learning. We normalize the data as we mentioned our dataset is not in range. So, we normalized all the numerical features to the range of 0 to 1 to make the model work well.

```
# normalize the data attributes
from sklearn import preprocessing
normalized_X = preprocessing.normalize(X)
print(normalized_X)

[[0.00000000e+00 1.71594732e-06 6.97699890e-01 ... 6.86378928e-06
 1.20116312e-05 1.37275786e-05]
 [5.74285820e-07 5.74285820e-07 7.42840144e-01 ... 5.74285820e-07
 2.29714328e-06 1.72285746e-06]
 [8.82421015e-06 4.41210508e-06 5.95634185e-01 ... 1.76484203e-05
 3.08847355e-05 3.52968406e-05]
 ...
 [2.89551295e-01 9.41605730e-07 6.38092306e-01 ... 9.41605730e-07
 6.59124011e-06 9.41605730e-06]
 [4.06879010e-01 1.32314505e-06 4.89705244e-01 ... 5.29258019e-06
 1.32314505e-06 1.05851604e-05]
 [2.76255414e-01 8.98362375e-07 6.06394603e-01 ... 8.98362375e-07
 8.98362375e-07 7.18689900e-06]]
```

Figure: Normalization of dataset

**Binary Encoding for Categorical Features with 2 classes**

```
In [34]: indexer = StringIndexer(inputCol="CODE_GENDER", outputCol="GenderClass", handleInvalid="keep")
df_loan = indexer.fit(df_loan).transform(df_loan)

In [35]: df_loan.select(['NAME_HOUSING_TYPE', 'CODE_GENDER', 'GenderClass', 'TARGET']).show(15)
```

NAME_HOUSING_TYPE	CODE_GENDER	GenderClass	TARGET
House / apartment	M	1.0	1
House / apartment	F	0.0	0
House / apartment	M	1.0	0
House / apartment	F	0.0	0
House / apartment	M	1.0	0
House / apartment	M	1.0	0
House / apartment	F	0.0	0
House / apartment	M	1.0	0
House / apartment	F	0.0	0
House / apartment	M	1.0	0
House / apartment	F	0.0	0
House / apartment	F	0.0	0
House / apartment	F	0.0	0
House / apartment	M	1.0	0
House / apartment	F	0.0	0

only showing top 15 rows

Figure: Binary Encoding Gender

Since "Gender" feature has two classes binary encoding is done which resulted in 1 for males and 0 for females.

One hot encoding is done for categorical features like "NAME\_HOUSING\_TYPE", "NAME\_FAMILY\_STATUS", "ORGANIZATION\_TYPE", "NAME\_EDUCATION\_TYPE", "NAME\_INCOME\_TYPE", "OCCUPATION\_TYPE". This adds additional columns giving 1 to the category present in that record and 0 for other classes.

## One hot Encoding categorical Features

```
In [36]: categories = df_loan.select("NAME_HOUSING_TYPE").distinct().toPandas().NAME_HOUSING_TYPE.tolist()
# print(categories)
exprs = [F.when(F.col("NAME_HOUSING_TYPE") == category, 1).otherwise(0).alias(category)
         for category in categories]

df_loan = df_loan.select("?", *exprs)

newCategories = []
for category in categories:
    df_loan = df_loan.withColumnRenamed(category, "NAME_HOUSING_TYPE_"+category)
    newCategories.append("NAME_HOUSING_TYPE_"+category)
print(newCategories)

['NAME_HOUSING_TYPE_House / apartment', 'NAME_HOUSING_TYPE_Municipal apartment', 'NAME_HOUSING_TYPE_Co-op apartment',
 'NAME_HOUSING_TYPE_Rented apartment', 'NAME_HOUSING_TYPE_Office apartment', 'NAME_HOUSING_TYPE_With parents']
```

```
In [185]: df_loan.select(['NAME_HOUSING_TYPE_House / apartment', 'NAME_HOUSING_TYPE_Rented apartment', 'NAME_HOUSING_TYPE_Municipal apartment'])
```

NAME_HOUSING_TYPE_House / apartment	NAME_HOUSING_TYPE_Rented apartment	NAME_HOUSING_TYPE_Municipal apartment
1	0	0
1	0	0
1	0	0
1	0	0
1	0	0
1	0	0
1	0	0
1	0	0
1	0	0
1	0	0
1	0	0
1	0	0
1	0	0
1	0	0
1	0	0

only showing top 15 rows

```
In [37]: categories = df_loan.select("NAME_FAMILY_STATUS").distinct().toPandas().NAME_FAMILY_STATUS.tolist()
# print(categories)
exprs = [F.when(F.col("NAME_FAMILY_STATUS") == category, 1).otherwise(0).alias(category)
         for category in categories]

df_loan = df_loan.select("?", *exprs)

newCategories = []
for category in categories:
    df_loan = df_loan.withColumnRenamed(category, "NAME_FAMILY_STATUS_"+category)
    newCategories.append("NAME_FAMILY_STATUS_"+category)
```

```
In [38]: categories = df_loan.select("ORGANIZATION_TYPE").distinct().toPandas().ORGANIZATION_TYPE.tolist()
# print(categories)
exprs = [F.when(F.col("ORGANIZATION_TYPE") == category, 1).otherwise(0).alias(category)
         for category in categories]

df_loan = df_loan.select("?", *exprs)

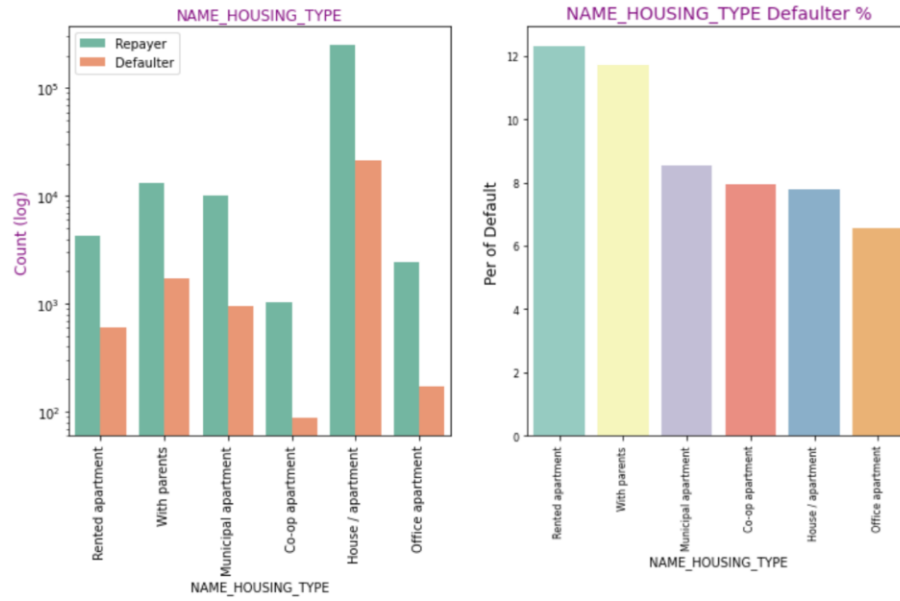
newCategories = []
for category in categories:
    df_loan = df_loan.withColumnRenamed(category, "ORGANIZATION_TYPE_"+category)
    newCategories.append("ORGANIZATION_TYPE_"+category)
```

```
In [39]: categories = df_loan.select("NAME_EDUCATION_TYPE").distinct().toPandas().NAME_EDUCATION_TYPE.tolist()
# print(categories)
exprs = [F.when(F.col("NAME_EDUCATION_TYPE") == category, 1).otherwise(0).alias(category)
         for category in categories]

df_loan = df_loan.select("?", *exprs)

newCategories = []
for category in categories:
    df_loan = df_loan.withColumnRenamed(category, "NAME_EDUCATION_TYPE_"+category)
    newCategories.append("NAME_EDUCATION_TYPE_"+category)
```

## o Graph model with explanation

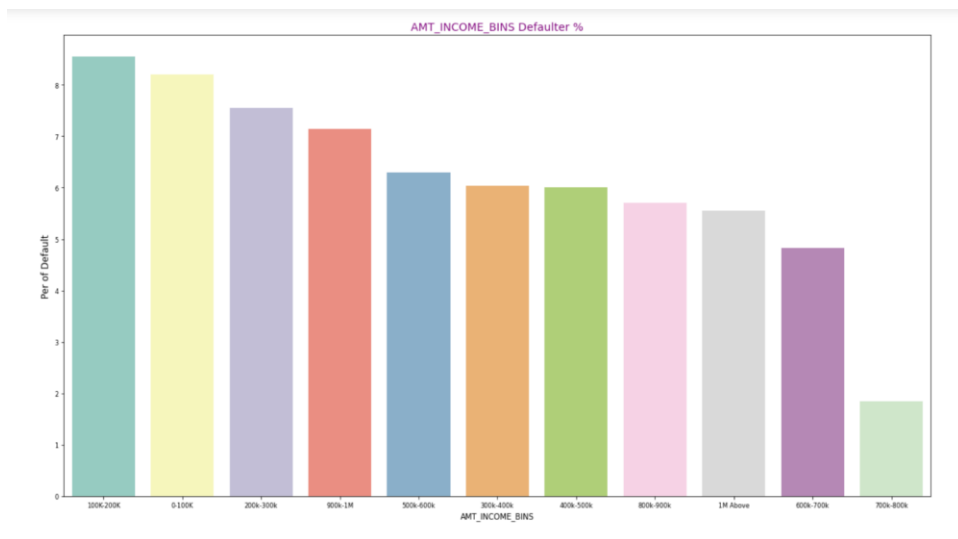
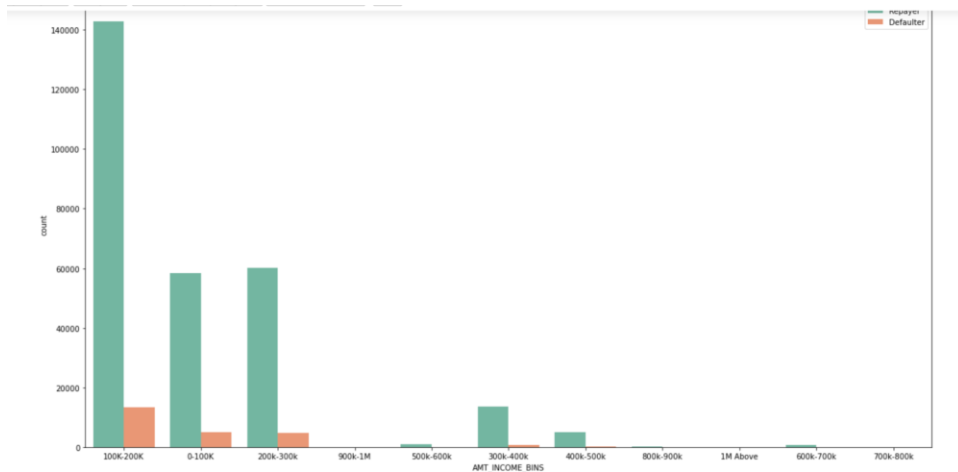


## Inferences

Majority of applicants live in House/apartment

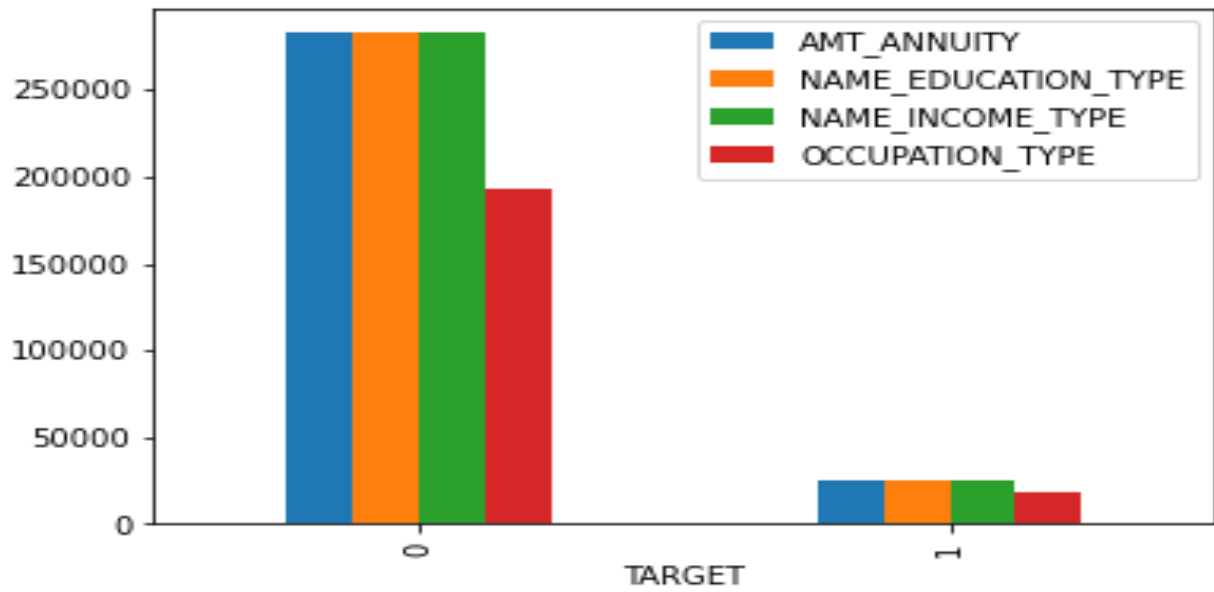
Applicants living in office apartments have the lowest default rate

Applicants living with parents (~11.5%) and living in rented apartments (>12%) are highly likely to default.

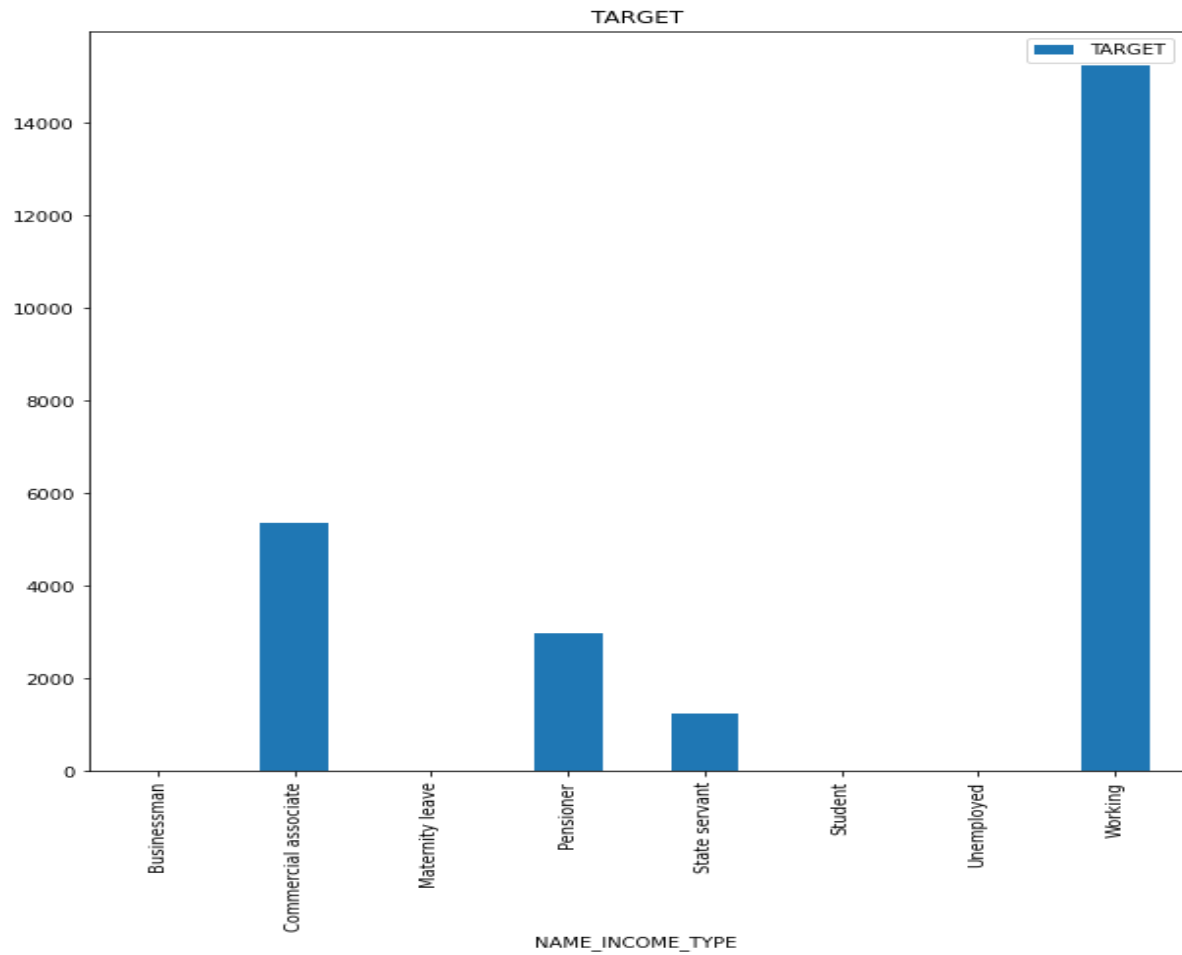


Over 90 percent applicants have an Income < 300k and have high chances of defaulting

Applicants with income > 700k are less likely to be defaulters

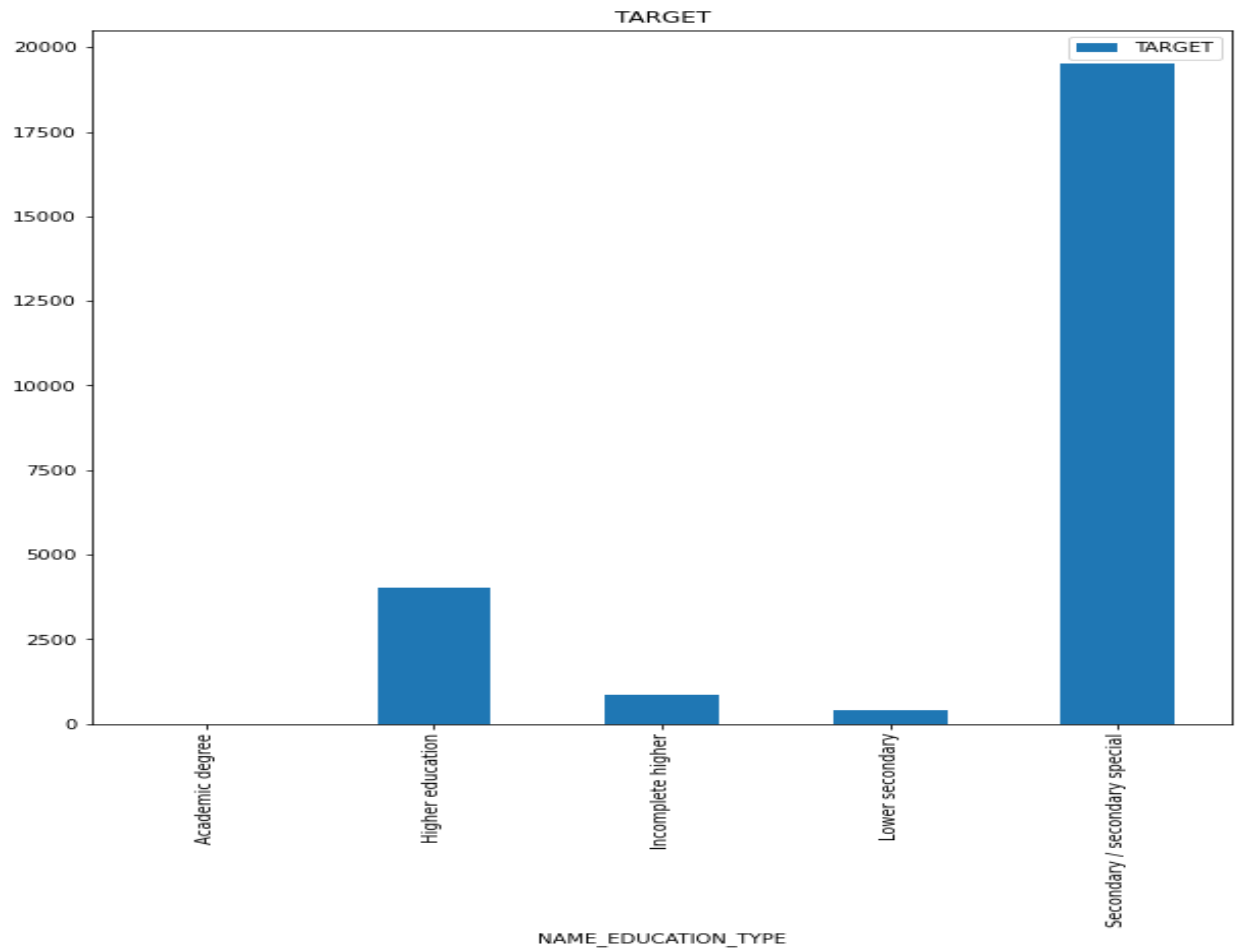


The above Bar Chart shows that amount Annuity, Name of Education Type, Name of Income Type and Occupation Type has more than 25k applicants likely to pay the loan (Payer) and a smaller number of Defaulters which means fail to pay the loan. So, we can assume that these four columns are correlated with the Target value.

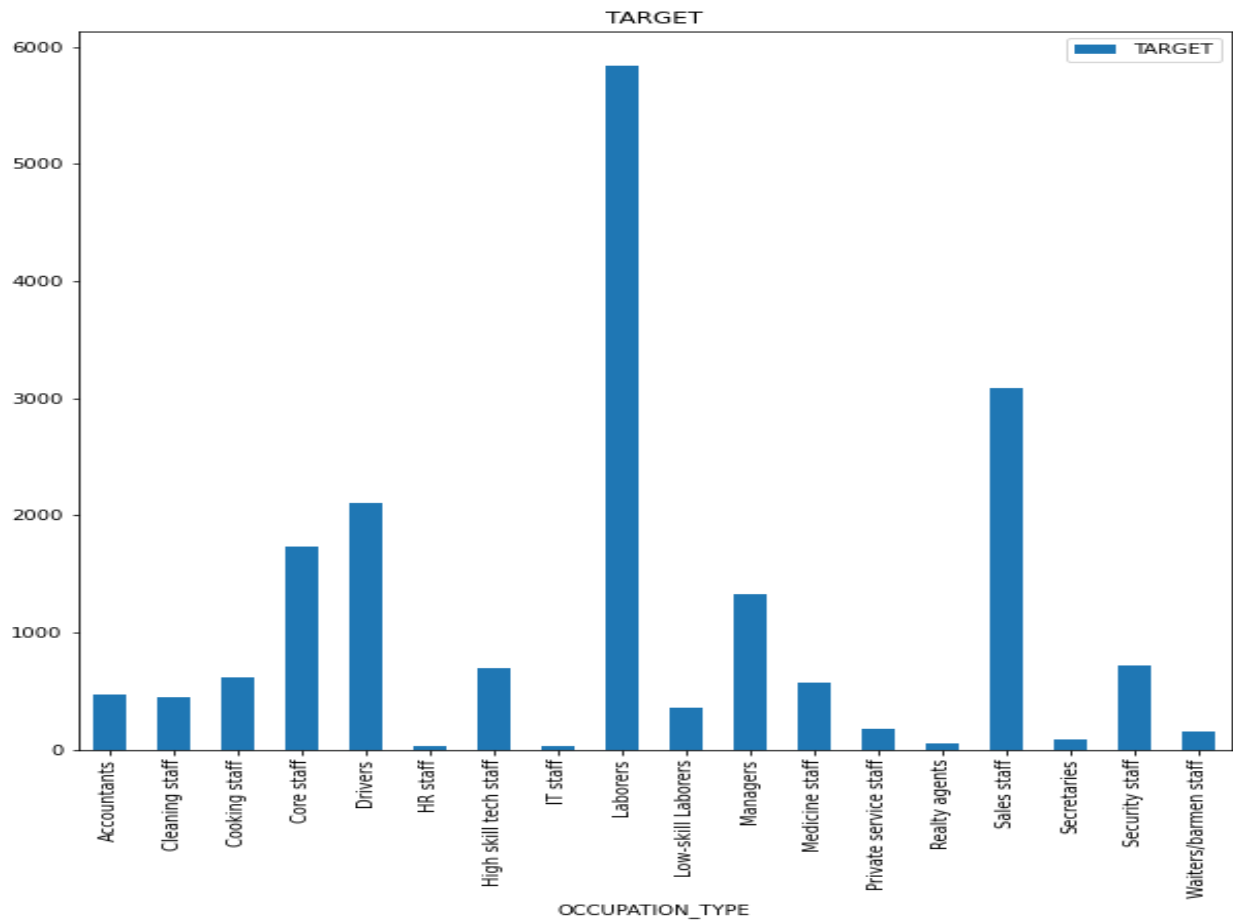


The above figure shows the number of defaulters and Repayor of various Income type name. Here, we have working more applicants.



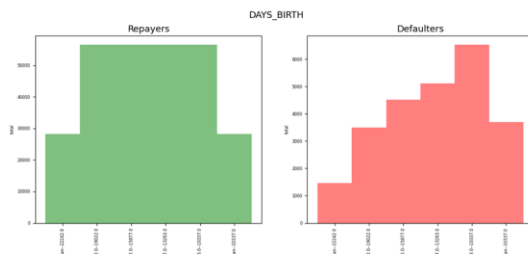
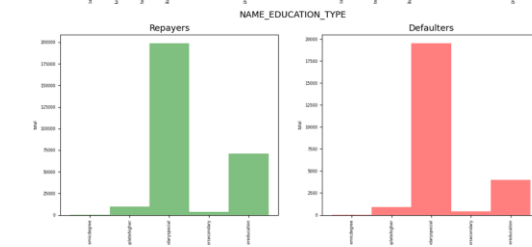
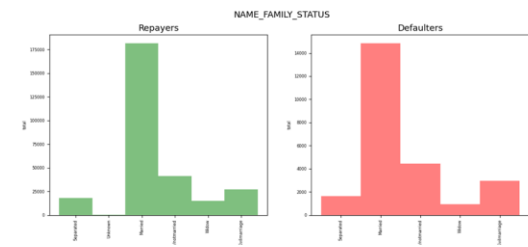
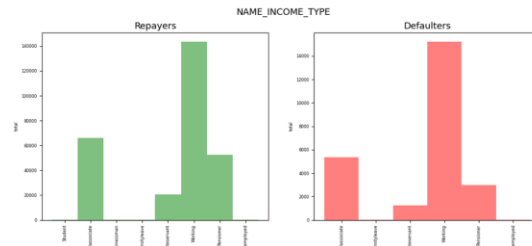
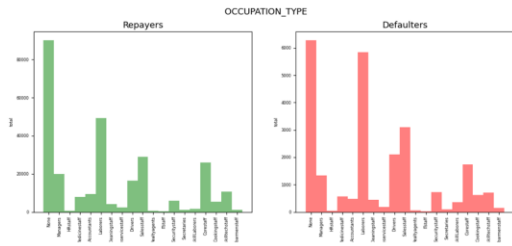
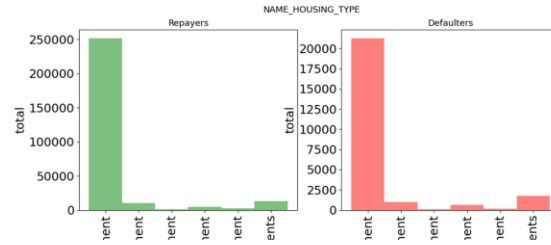
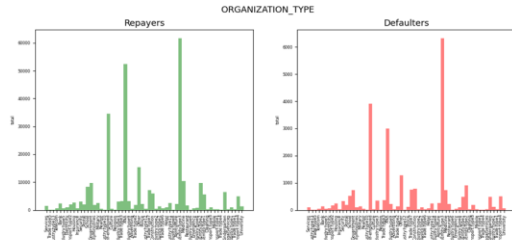


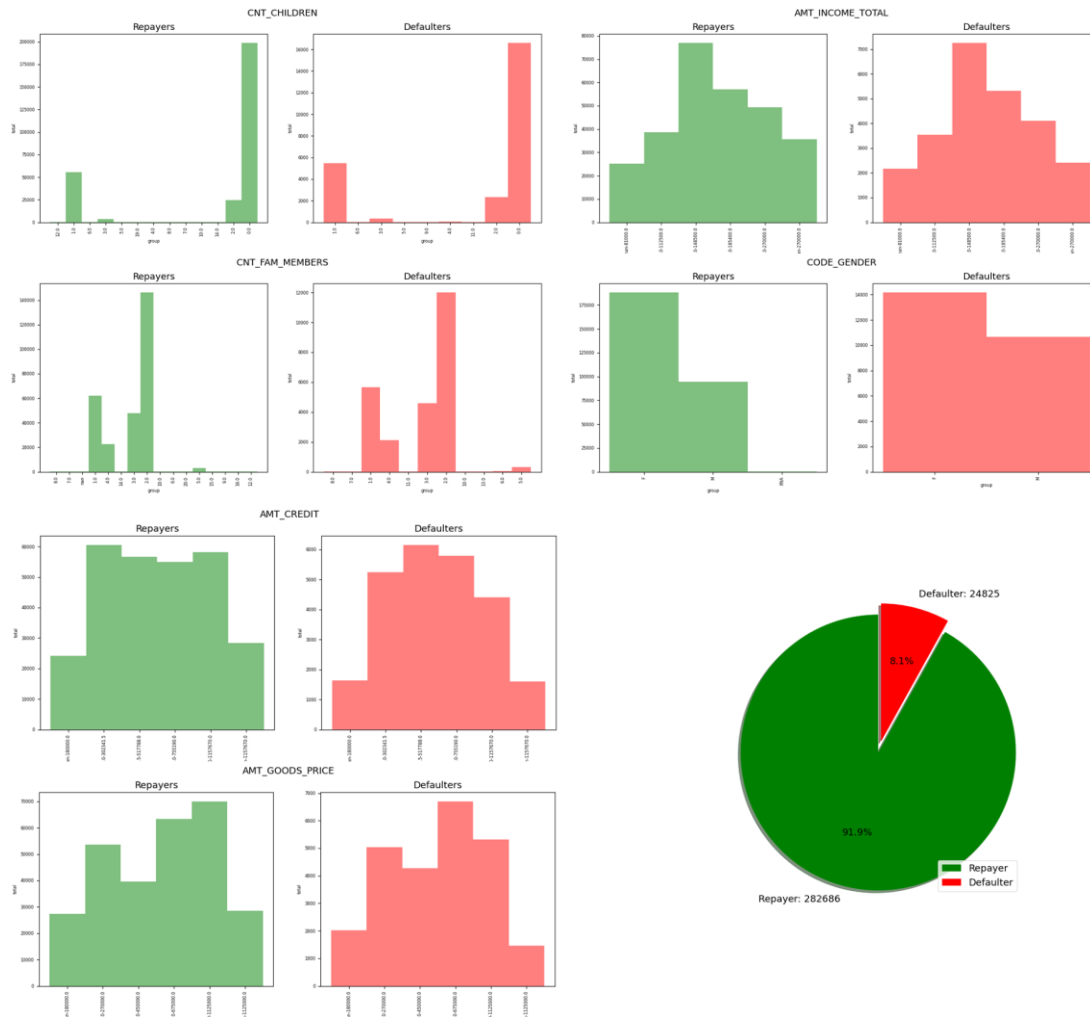
From the Above Bar chart of the NAME\_EDUCATION\_TYPE, we can see that maximum number of applicants are Secondary/secondary special types who have applied for the loan. Only few applicants have Academic degree in this column.



From the Above Bar chart of the Occupation type, we can see that maximum number of laborers have applied for the loan and IT staff and HR Staff have the lowest number of applicants

Analysis of data and pre-processing:





**Figure: bar plots for each attribute repayers vs defaulters**

- **NAME\_HOUSING\_TYPE:** the distribution is similar, most of the applicants have their own house/apartment. The house type impacts the approval since those applicants that live with their parents or rent an apartment are most likely to be defaulters since they have the highest proportion fraction for that attribute
- **AMT\_CREDIT:** applications between '300k and 755k' have a higher probability of being considered as defaulters (but this decision may be related to other attribute values)
- **CODE\_GENDER:** the ratio or proportion for males is greater than the females. so, Females are better at repaying their loans than males
- **NAME\_FAMILY\_STATUS:** applicants with status civil marriage and single/not married are leaning towards being defaulters.
- **ORGANIZATION\_TYPE:** there are several types of organizations that have bad scores greater than 10 percent, so if the application is approved or not may depends a lot on this attribute or just the sample data is not big enough to determine if the organization type is good.

- DAYS\_BIRTH: the ratios show that the oldest people have a small proportion of defaulters than younger people, which means that the oldest people are more likely to get approval.
- DAYS\_WORKING: those with more than 3 years working are more likely to repay the loan
- NAME\_EDUCATION\_TYPE: those with lower secondary are more likely to be defaulters
- NAME\_INCOME\_TYPE: the pensioners are the best group repaying the loans.

## DATASET

	NAME_HOUSING_TYPE	AMT_CREDIT	AMT_INCOME_TOTAL	CODE_GENDER	SK_ID_CURR	NAME_FAMILY_STATUS	CNT_CHILDREN	CNT_FAM_MEMBER
0	House / apartment	1019610.0	112500.000	F	100011	Married	0	2
1	House / apartment	148365.0	38419.155	F	100015	Married	0	2
2	House / apartment	239850.0	83250.000	F	100027	Married	0	2
3	House / apartment	790830.0	270000.000	M	100033	Single / not married	0	1
4	House / apartment	665892.0	292500.000	F	100035	Civil marriage	0	2
5	House / apartment	247275.0	99000.000	F	100045	Married	0	2
6	House / apartment	746280.0	108000.000	F	100050	Single / not married	0	1
7	House / apartment	661702.5	202500.000	M	100051	Civil marriage	0	2
8	House / apartment	305221.5	202500.000	F	100053	Single / not married	0	1
9	House / apartment	454500.0	76500.000	M	100060	Married	0	2
10	House / apartment	675000.0	81000.000	M	100062	Married	0	2
11	Municipal apartment	298728.0	67500.000	F	100064	Single / not married	0	1
12	House / apartment	1130760.0	324000.000	M	100073	Civil marriage	0	2

	AMT_CREDIT	AMT_INCOME_TOTAL	SK_ID_CURR	CNT_CHILDREN	CNT_FAM_MEMBERS	DAYS_BIRTH	DAYS_EMPLOYED	AMT_GOODS_PRICE	TARGET
AMT_CREDIT	1.000000	0.156870	-0.000343	0.002145	0.063160	-0.055436	-0.066838	0.986968	-0.030369
AMT_INCOME_TOTAL	0.156870	1.000000	-0.001820	0.012882	0.016342	0.027261	-0.064223	0.159610	-0.003982
SK_ID_CURR	-0.000343	-0.001820	1.000000	-0.001129	-0.002895	-0.001500	0.001366	-0.000232	-0.002108
CNT_CHILDREN	0.002145	0.012882	-0.001129	1.000000	0.679161	0.330938	-0.239818	-0.001827	0.019187
CNT_FAM_MEMBERS	0.063160	0.016342	-0.002895	0.679161	1.000000	0.278894	-0.233549	0.061185	0.009308
DAYS_BIRTH	-0.055436	0.027261	-0.001500	0.330938	0.278894	1.000000	-0.615864	-0.053442	0.078239
DAYS_EMPLOYED	-0.066838	-0.064223	0.001366	-0.239818	-0.233549	-0.615864	1.000000	-0.064842	-0.044932
AMT_GOODS_PRICE	0.986968	0.159610	-0.000232	-0.001827	0.061185	-0.053442	-0.064842	1.000000	-0.039645
TARGET	-0.030369	-0.003982	-0.002108	0.019187	0.009308	0.078239	-0.044932	-0.039645	1.000000

Figure: data frame and correlation values

For special imputation, it is necessary to count the number of null values each column has, to do so it is necessary to iterate over the columns of the dataset to run a filter transformation with a count action to get the number of null values in each column, then in order to get a fraction those numbers are divided by the size of the dataframe, and the columns with more than 10% will be grouped by the values of another column to see if they are related.

```
{'NAME_HOUSING_TYPE': 0.0,
'AMT_CREDIT': 0.0,
'AMT_INCOME_TOTAL': 0.0,
'CODE_GENDER': 0.0,
'SK_ID_CURR': 0.0,
'NAME_FAMILY_STATUS': 0.0,
'CNT_CHILDREN': 0.0,
'CNT_FAM_MEMBERS': 6.503832383231819e-06,
'ORGANIZATION_TYPE': 0.0,
'DAYS_BIRTH': 0.0,
'DAYS_EMPLOYED': 0.0,
'AMT_GOODS_PRICE': 0.0009040327012692228,
'NAME_EDUCATION_TYPE': 0.0,
'NAME_INCOME_TYPE': 0.0,
'OCCUPATION_TYPE': 0.31345545362604915,
'TARGET': 0.0}
```

**Figure: ratio of null values per column**

Here Occupation\_type is the only one column with more than 30% null values. After making several groups by the values of other columns, the NAME\_INCOME\_TYPE column with 0 null values was the best column to group OCCUPATION\_TYPE.

NAME_INCOME_TYPE count	NAME_INCOME_TYPE  count
Student  5	Student  13
Commercial associate 12297	Commercial associate  59320
Businessman  2	Businessman  8
Maternity leave  1	Maternity leave  4
State servant  3787	State servant  17916
Working 24920	Working 133854
Pensioner 55357	Pensioner  5
Unemployed  22	

$$QtyNull_i / QtyNotNull_i$$

```
{'Student': 0.38461538461538464,
'Commercial associate': 0.2072993931220499,
'Businessman': 0.25,
'Maternity leave': 0.25,
'State servant': 0.211375306988167,
'Working': 0.18617299445664678,
'Pensioner': 11071.4,
'Unemployed': 'there is nothing to compare'}
```

**Figure: attributes to make special imputation**

The above picture shows how the Pensioner and Unemployed categories from NAME\_INCOME\_TYPE have more OCCUPATION\_TYPE null values (left top table) related than not null values (right top table). At the bottom of the previous picture the fractions (left Values divided by right table values) are shown.

## Implementation

(Everyone)

### Algorithms / Pseudocode

We have used the four models to train the dataset and evaluate the performance of these models by comparing with each other based on balanced and unbalanced dataset. We also compare the performance of model based on label encoder and one-hot encoder for the categorical dataset.

### Logistic Regression Model

Under the Supervised Learning approach, one of the most prominent Machine Learning algorithms is logistic regression. It's a method for predicting a categorical dependent variable from a set of independent factors. A categorical dependent variable's output is predicted using logistic regression. As a result, the result must be of a discrete or categorical value. It can be Yes or No, 0 or 1, true or false, and so on, but instead of giving precise values like 0 and 1, it delivers probabilistic values that are somewhere between 0 and 1. The equation for the Logistic regression is given by

$$y = e^{(b_0 + b_1 \cdot x)} / (1 + e^{(b_0 + b_1 \cdot x)})$$

Where y is the predicted output, b0 is the bias or intercept term and b1 is the coefficient for the single input value (x).

The pseudocode for the logistic regression is shown below:

---

**Input:** Training data

---

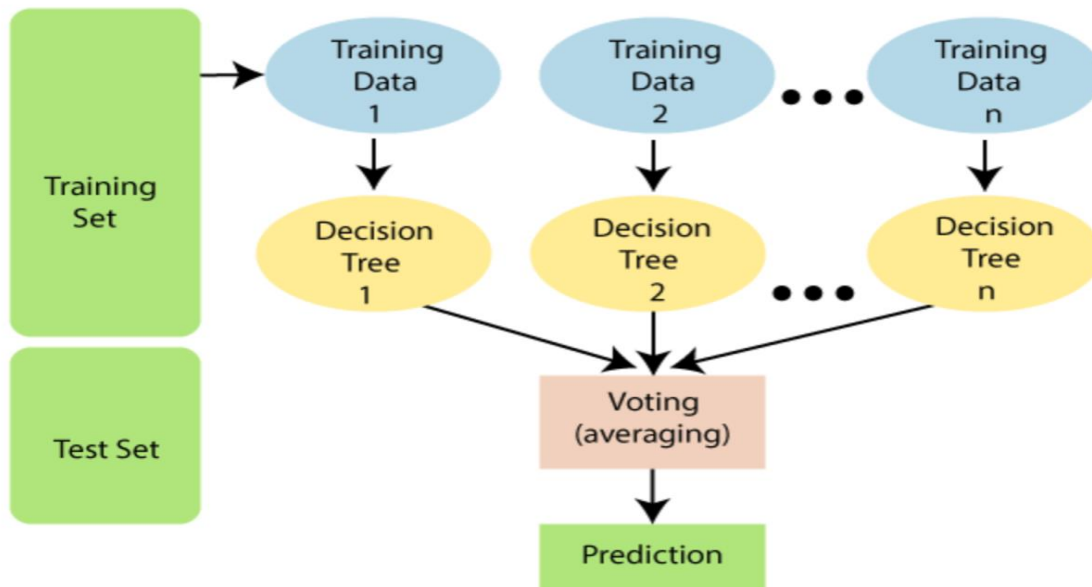
1. For  $i \leftarrow 1$  to  $k$
  2. For each training data instance  $d_i$ :
  3. Set the target value for the regression to  
$$z_i \leftarrow \frac{y_j - P(1 | d_j)}{[P(1 | d_j) \cdot (1 - P(1 | d_j))]}$$
  4. initialize the weight of instance  $d_j$  to  $P(1 | d_j) \cdot (1 - P(1 | d_j))$
  5. finalize a  $f(j)$  to the data with class value ( $z_j$ ) & weights ( $w_j$ )
- Classification Label Decision**
6. Assign (class label:1) if  $P(1 | d_j) > 0.5$ , otherwise (class label: 2)
- 

Figure: pseudocode of Logistic Regression

**Random Forest Classifier:**

A random forest is a widely used supervise learning approach. It fits a number of decision tree classifiers on various sub-samples of the dataset. Averaging is used to enhance predictive accuracy and control over-fitting. The number of sub-samples in each tree is controlled with the `max_samples` parameter.





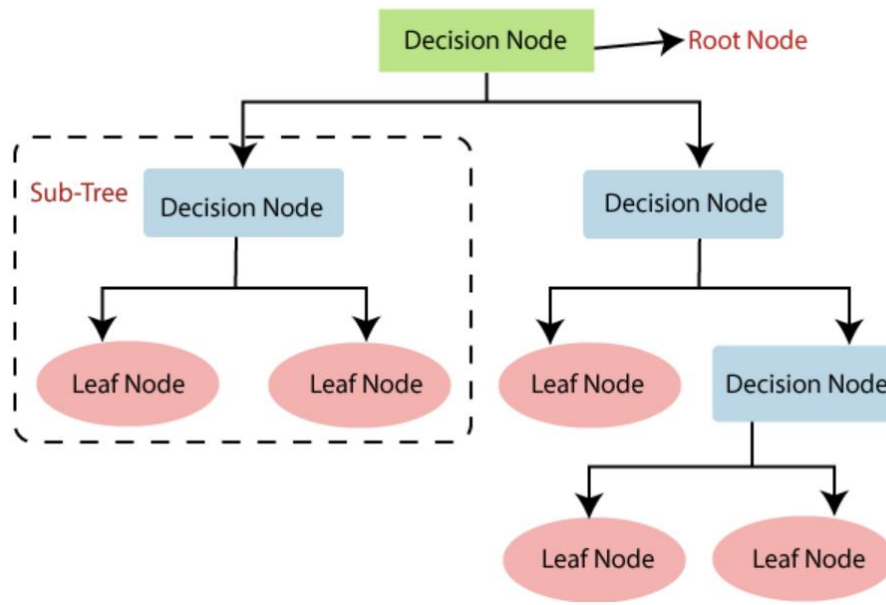
**Fig: Working of Random Forest Classifier**

**The reasons why I choose to use random forest algorithm is**

- Training time is low when compared to other algorithms.
- It gives highly accurate results, runs efficiently even for the huge datasets.
- It can also maintain accuracy even if a large percentage of data is missing.

#### Decision Tree Model

Decision tree is a type of supervised machine learning algorithm where the data is continually separated based on a certain parameter. It is used for both classification and regression problems but mostly used for classification problems. It is of a tree structure that basically has two types of nodes i.e., decision node and leaf node.



**Figure: Decision tree diagram**

Reasons for using decision tree model, is that it is easy to understand as it uses tree like structure that is easily readable.

### **Random Forest with data imputed by groups:**

In order to use more query statements, this section proposes to identify if the null values of a column are related to the null values of another column, and apply a more appropriate imputation according to the information found, this is proposed because the imputation method will replace the null values with the mean or median value of the samples, and if the null value has a meaning, replacing it can lose the consistency of the data and if in addition the amount of null values is large, replacing them will make my data unreliable. for example, I could assign a very well-paid job to someone who already said in his data that he was unemployed.

To identify the aforementioned, it is necessary to make use of several functions that spark provides and that can be associated with a statement in SQL. For example, to find the frequencies of the categories of a column it is necessary to group and count the data (group by in SQL and groupby() in PySpark), it will also be necessary to change some values of the columns (update 'colName' where 'cond' in SQL and withColumn('colName',where('cond')) in PySpark), and other cases that use different transformations and actions to obtain relevant information from the DataFrames.

So, special imputation will be performed on those columns that have more than 10% of null values, it will be checked if the value is due to a value in another column and a new category will be added, or if on the contrary it was the case of missing information, then values that depend on the median of a group with similar values (not of the whole column in general) will be imputed.

# Special Imputation

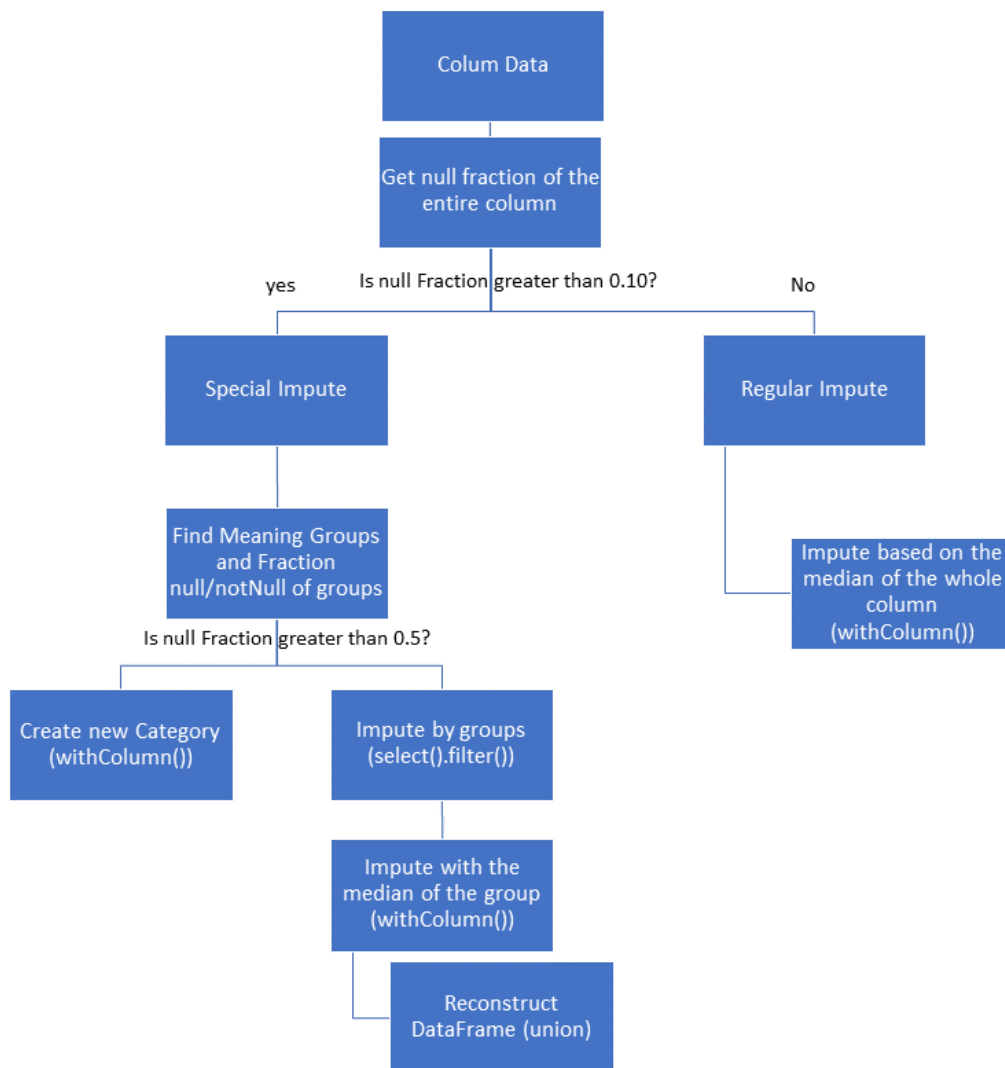


Figure: special imputation diagram

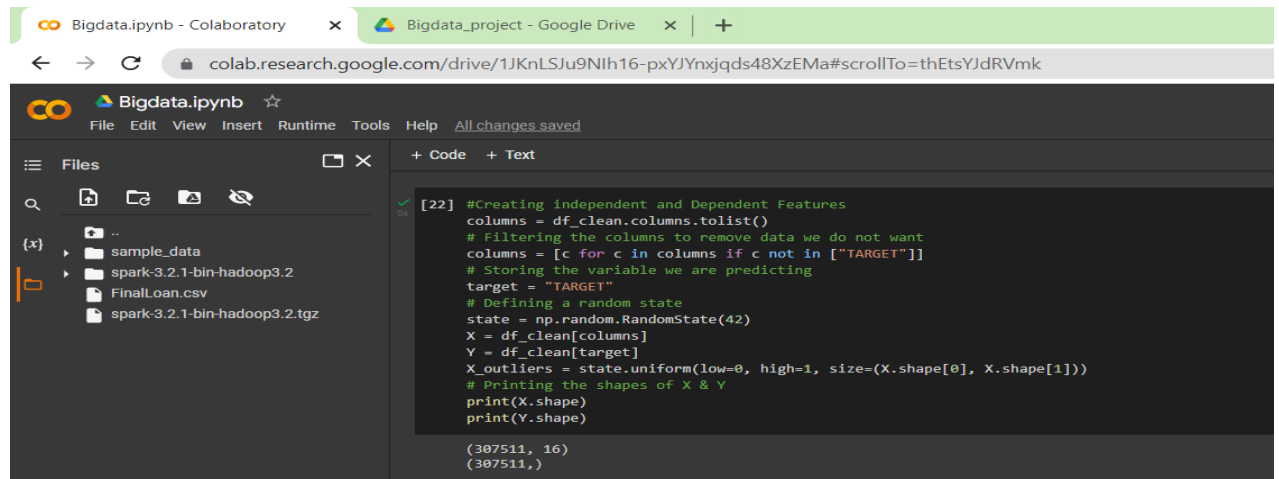
Finally, the Random Forest model is also used for prediction.

## Explanation of implementation

### Logistic Regression Model:

I count the total number of datapoint and separate into features and target value from the dataset. We have 307511 total datapoint in our dataset and then I pre-proceed the dataset and

find the null and missing value of each feature of the dataset and fill those values with the previous value of that features using the spark.



The screenshot shows a Jupyter Notebook interface with a file explorer on the left and a code editor on the right. The file explorer shows a directory structure with files like 'sample\_data', 'spark-3.2.1-bin-hadoop3.2', 'FinalLoan.csv', and 'spark-3.2.1-bin-hadoop3.2.tgz'. The code editor shows a code cell [22] with the following Python code:

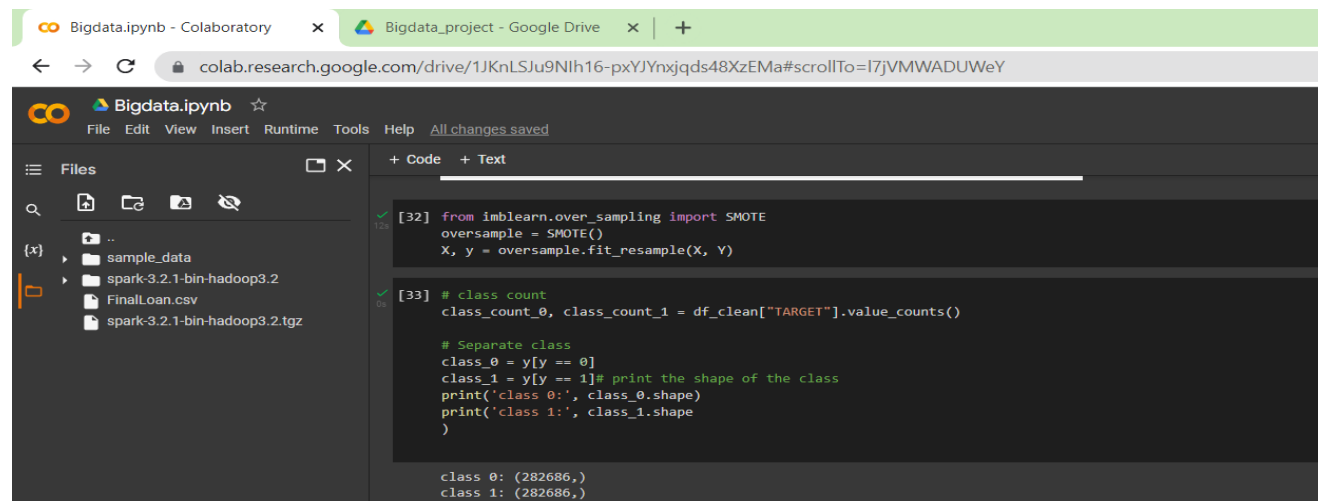
```
[22] #Creating independent and Dependent Features
columns = df_clean.columns.tolist()
# Filtering the columns to remove data we do not want
columns = [c for c in columns if c not in ["TARGET"]]
# Storing the variable we are predicting
target = "TARGET"
# Defining a random state
state = np.random.RandomState(42)
X = df_clean[columns]
Y = df_clean[target]
X_outliers = state.uniform(low=0, high=1, size=(X.shape[0], X.shape[1]))
# Printing the shapes of X & Y
print(X.shape)
print(Y.shape)
```

The output of the code cell is:

```
(307511, 16)
(307511,)
```

Figure: Features and Target count value

After that, I normalized the dataset using the pre-processing library and did the label encoder to feed the model. Firstly, I train the model with unbalanced data and then balance the dataset with **SMOTE** function and again feed the Logistic Regression Model.



The screenshot shows a Jupyter Notebook interface with a file explorer on the left and a code editor on the right. The file explorer shows a directory structure with files like 'sample\_data', 'spark-3.2.1-bin-hadoop3.2', 'FinalLoan.csv', and 'spark-3.2.1-bin-hadoop3.2.tgz'. The code editor shows two code cells. The first code cell [32] contains the following Python code:

```
[32] from imblearn.over_sampling import SMOTE
oversample = SMOTE()
X, y = oversample.fit_resample(X, y)
```

The second code cell [33] contains the following Python code:

```
[33] # class count
class_count_0, class_count_1 = df_clean["TARGET"].value_counts()

# Separate class
class_0 = y[y == 0]
class_1 = y[y == 1]# print the shape of the class
print('class 0:', class_0.shape)
print('class 1:', class_1.shape)
```

The output of the second code cell is:

```
class 0: (282686,)
class 1: (282686,)
```

Figure: SMOTEN to Balanced dataset

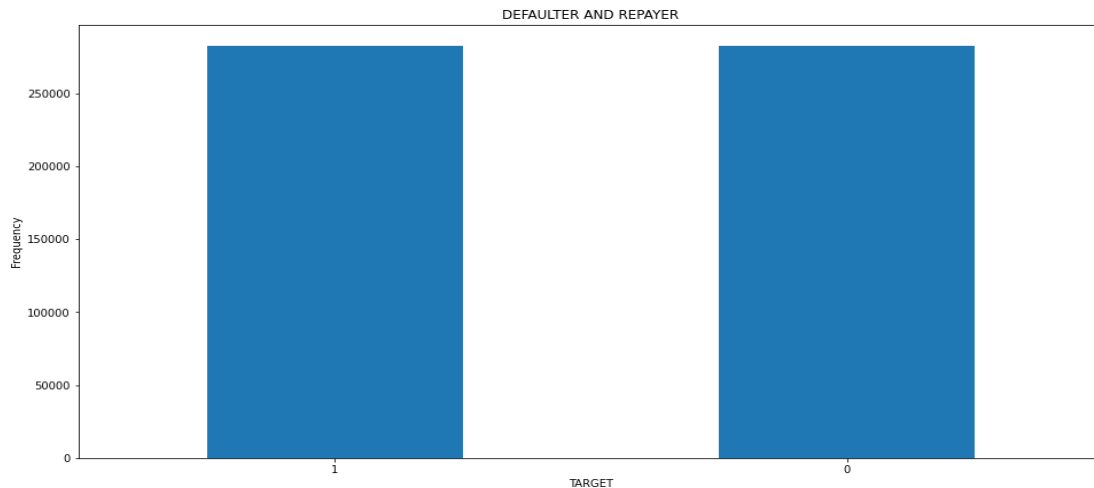


Figure:

**Balanced Dataset**

To compare the model, I use the hot-encoder for the categorical features and feed the model but this time I didn't normalize the data before feeding the Logistic Regression Model. I split the dataset into 75:25 ratio for the training and testing which is 230633 datapoint for training and 76878 datapoint for testing as shown in figure below. So, from the figure, we can see that the training and testing dataset is different from balanced and unbalanced dataset.

```
[24] #splitting the dataset for training and testing
from sklearn.model_selection import train_test_split

X_train, X_test, y_train, y_test = train_test_split(normalized_X, Y, test_size=0.25, random_state=0)
print(X_train.shape)
print(X_test.shape)

(230633, 16)
(76878, 16)
```

**Figure: Split of unbalanced dataset**

```
[35] X_train, X_test, Y_train, Y_test = train_test_split(X, y, test_size=0.25, random_state=0)
print(X_train.shape)
print(X_test.shape)

(424029, 16)
(141343, 16)
```

**Figure: Split of balanced dataset**

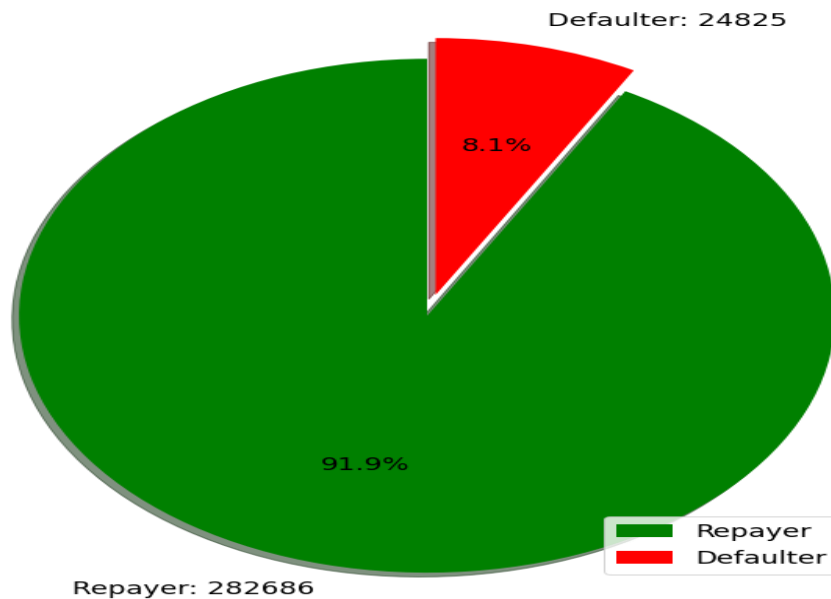
For the evaluation of the model, I use confusion matrix and binary classification to see the performance of Logistic Regression Model. The comparison and output are discussed in the results part of the report.

### Random Forest Classifier:

Post Feature analysis I have done binary encoding for categorical variables with 2 classes and one hot encoding for categorical variables with multiple classes. For the numeric features I handled the missing values by setting thresholds. If more than 90% of the data is missing, then

the features are completely discarded. For features with few missing values they are replaced with mean.

At first, I trained the random forest classifier on the whole dataset which was highly imbalanced and can be seen below.



**Figure: Repayers and Defaulters population.**

I got an accuracy of **91%** which is a completely **misleading** measure because the model was predicting everything as 0 i.e., as repayers. Since the test dataset has 91% of repayers the accuracy is 91%. This is a completely overfitted model.

I then **balanced** the dataset by downscaling the repayers. I have chosen randomly 30k records of repayers and all 24825 defaulters. I split the train and test at 70 :30 ratio. I have tuned the depth parameter i.e max\_depth=2.

## Handling Data Imbalance

```
In [ ]: df_loan.groupby('TARGET').count().show()

[Stage 116:=====> (2
+ 2) / 4]

+-----+-----+
| TARGET | count |
+-----+-----+
|      1 | 24825 |
|      0 | 282686|
+-----+-----+
```

```
In [ ]: df1=clean_loan_df.query("CLASS == 0.0").sample(n=30000)
```

```
In [ ]: df2=clean_loan_df.query("CLASS == 1.0").sample(n=24825)
```

```
In [ ]: frames=[df1,df2]
        balancedDF= pd.concat(frames)
```

With balanced data I got an **accuracy of ~95%** with the tuned random forest classifier. I got a **recall of 0.89** for defaulters and 1 for repayers. The **precision** was **0.91** for repayers and 1 for defaulters.

```
In [179]: print(classification_report(test_labels, pred))
```

	precision	recall	f1-score	support
0.0	0.91	1.00	0.96	8970
1.0	1.00	0.89	0.94	7478
accuracy			0.95	16448
macro avg	0.96	0.94	0.95	16448
weighted avg	0.95	0.95	0.95	16448

## Decision Tree model

First of all, the data was analyzed and preprocessed. All the missing values were imputed, and the categorical values were converted to numerical values. Then the data was trained using the decision tree classifier, but the model was overfitted due to the imbalanced data. Hence the data was first balanced out by cutting out the excess number of repayers. And making the data such that the number of repayers and the defaulter would not vary much.

```
[29] from sklearn.tree import DecisionTreeClassifier
      from sklearn.metrics import accuracy_score
      from sklearn.metrics import plot_confusion_matrix
      from sklearn.model_selection import train_test_split
      from sklearn.metrics import recall_score
      from sklearn.metrics import precision_score

data = clean_loandf
print(len(data))
label_names = data['CLASS']
labels = data['CLASS']
feature_names = clean_loandf.drop(['features', 'CLASS'], axis=1)
features = feature_names
train, test, train_labels, test_labels = train_test_split(
    features, labels, test_size = 0.30, random_state = 42)
# define the model
clf = DecisionTreeClassifier(max_depth=7)
clf.fit(train, train_labels)
# evaluate the model
pred=clf.predict(test)
print(accuracy_score(test_labels, pred))
plot_confusion_matrix(clf, test, test_labels)
```

### Random forest with imputation by groups.

First a data frame was created loading data from Hadoop DFS, then, I analyzed the data making bar plots for each attribute and finding ratios between Repayers and defaulters. Then a special imputation method proposed was implemented in order to keep the data consistency as much as possible.

After special imputation, the regular imputation was done for the columns with a few null values. The categorical data was converted to numerical, and a random forest model was trained and tested



## Imputing by groups

```
1 # make partitions of the dataframe based on the different values of the attributes
2 # to do so run filter transformations with condition "colName == one_attribute"
3 df_18attrPen = df_18attrI.filter('NAME_INCOME_TYPE == "Pensioner"')
4 df_18attrPen = df_18attrPen.withColumn('OCCUPATION_TYPE_i_imp',df_18attrPen.OCCUPATION_TYPE_
5 df_18attrUne = df_18attrI.filter('NAME_INCOME_TYPE == "Unemployed"')
6 df_18attrUne = df_18attrUne.withColumn('OCCUPATION_TYPE_i_imp',df_18attrUne.OCCUPATION_TYPE_
7 df_18attrStu = df_18attrI.filter('NAME_INCOME_TYPE == "Student"')
8 df_18attrCmA = df_18attrI.filter('NAME_INCOME_TYPE == "Commercial associate"')
9 df_18attrBsm = df_18attrI.filter('NAME_INCOME_TYPE == "Businessman"')
10 df_18attrMaL = df_18attrI.filter('NAME_INCOME_TYPE == "Maternity leave"')
11 df_18attrStS = df_18attrI.filter('NAME_INCOME_TYPE == "State servant"')
12 df_18attrWor = df_18attrI.filter('NAME_INCOME_TYPE == "Working"')

1 # create the Impute model that all groups will use (since the imputation will be based on t
2 # grouping to impute helps to change the median value for each group, then the median value
3 imputerOccType = Imputer(
4     inputCols=['OCCUPATION_TYPE_i'],
5     outputCols=["{}_imp".format('OCCUPATION_TYPE_i')]
6     ).setStrategy("median") #the values will be filled with the median of the column
7
8 # Add imputation cols to df
9 df_18attrStuI = imputerOccType.fit(df_18attrStu).transform(df_18attrStu)#
10 df_18attrCmAI = imputerOccType.fit(df_18attrCmA).transform(df_18attrCmA)
11 df_18attrBsmI = imputerOccType.fit(df_18attrBsm).transform(df_18attrBsm)
12 df_18attrMaLI = imputerOccType.fit(df_18attrMaL).transform(df_18attrMaL)
13 df_18attrStSI = imputerOccType.fit(df_18attrStS).transform(df_18attrStS)
14 df_18attrWorI = imputerOccType.fit(df_18attrWor).transform(df_18attrWor)

1 # build again the entire dataframe. use the union transformation to do so
2 df_18attrCreImp = df_18attrPen.union(df_18attrUne).union(df_18attrStuI).union(df_18attrCmAI)
```

- Results

### Analysis Results

- o Diagrams for results with detailed explanation

After data analysis on features the results of each feature with respect to target is

Feature	Insight
NAME_EDUCATION_TYPE	Applicants with Academic degrees have <u>less defaults</u>
NAME_INCOME_TYPE	Student and Businessmen have no defaults
REGION_RATING_CLIENT	client with RATING 1 can be given loans
ORGANIZATION_TYPE	Clients with Trade Type 4 and 5 and Industry type 8 have defaulted less than 3%
DAYS_BIRTH	People aged 50 have low chances of defaulting
DAYS_EMPLOYED	Clients worked for 40+ years defaults
AMT_INCOME_TOTAL	A person earning 700k or more is highly likely to repay
OCCUPATION_TYPE	Maximum number of applicants are Laborer and only few are IT staff and HR Staff.
NAME_EDUCATION_TYPE	Maximum number of applicants are Secondary/secondary and Only significant applicants have Academic degree.
NAME_INCOME_TYPE	Maximum number of applicants are from working background and hardly any are <u>businessman</u> , <u>Maternity leave</u> , unemployed and students
CNT_CHILDREN	Most of the applicants are Childless and tend to be <u>repayer</u>
NAME_FAMILY_STATUS	Married couples tend to be <u>repayers</u> and are the biggest group
CNT_FAMILY_MEMBERS	This attribute is related to the two attributes above
REGION_RATING_CLIENT	Applicants from region 3 are most likely to default

For Logistic regression

After the pre-processing , I train the Logistic Regression model using the clean dataset. First, I did the label encoder and trained the model where evaluation of this model is shown in figure below. After training, the model gives 92% of accuracy with F-1 score of 0.92 and 0 for defaulter and repayer respectively. I got the auc value 0.5 for the unbalanced data using the logistic regression model. In the confusion matrix, I didn't get any true negative value which shows 0 but for true positive it is 70787.

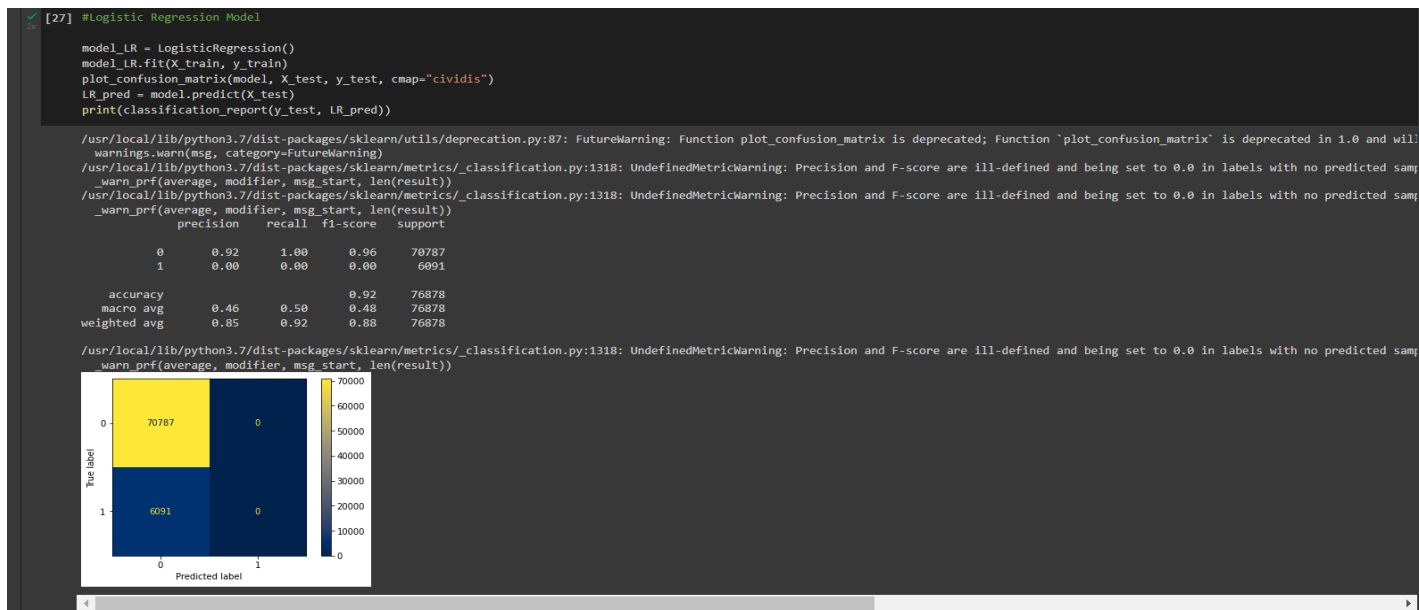


Figure: Performance of Logistic Regression on unbalanced data

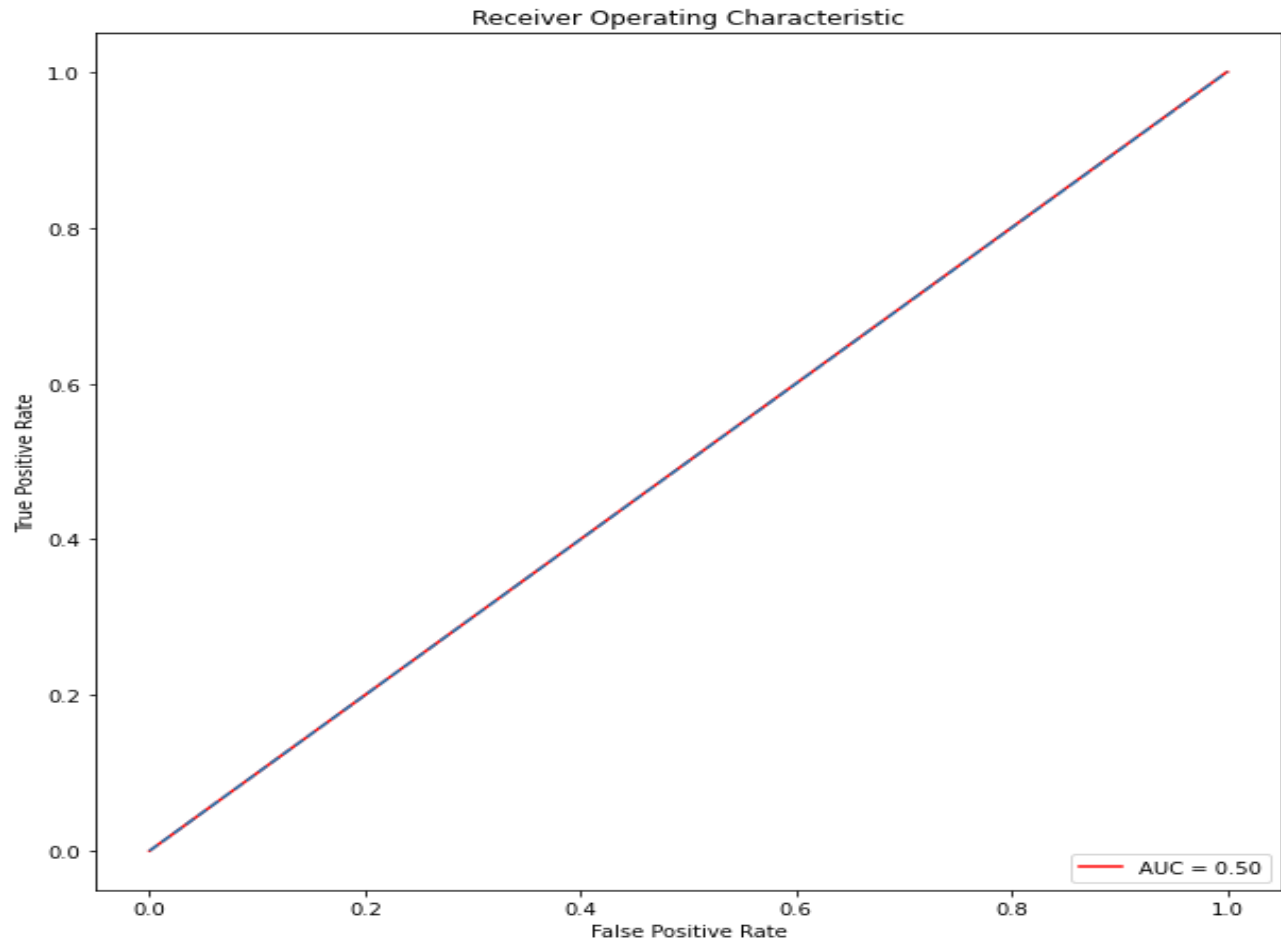


Figure: ROC curve of unbalanced dataset using Logistic Regression Model

After balancing the dataset using SMOTE function, I feed that balanced dataset to the logistic regression model and get the 54% accuracy only as shown in figure below. This means that the model is not working well when the dataset is balanced and categorical value is converted to numerical using label encoder. Here, the confusion matrix has truer negative and false positive. I get the roc value 0.54 for the balanced dataset using the logistic regression model.

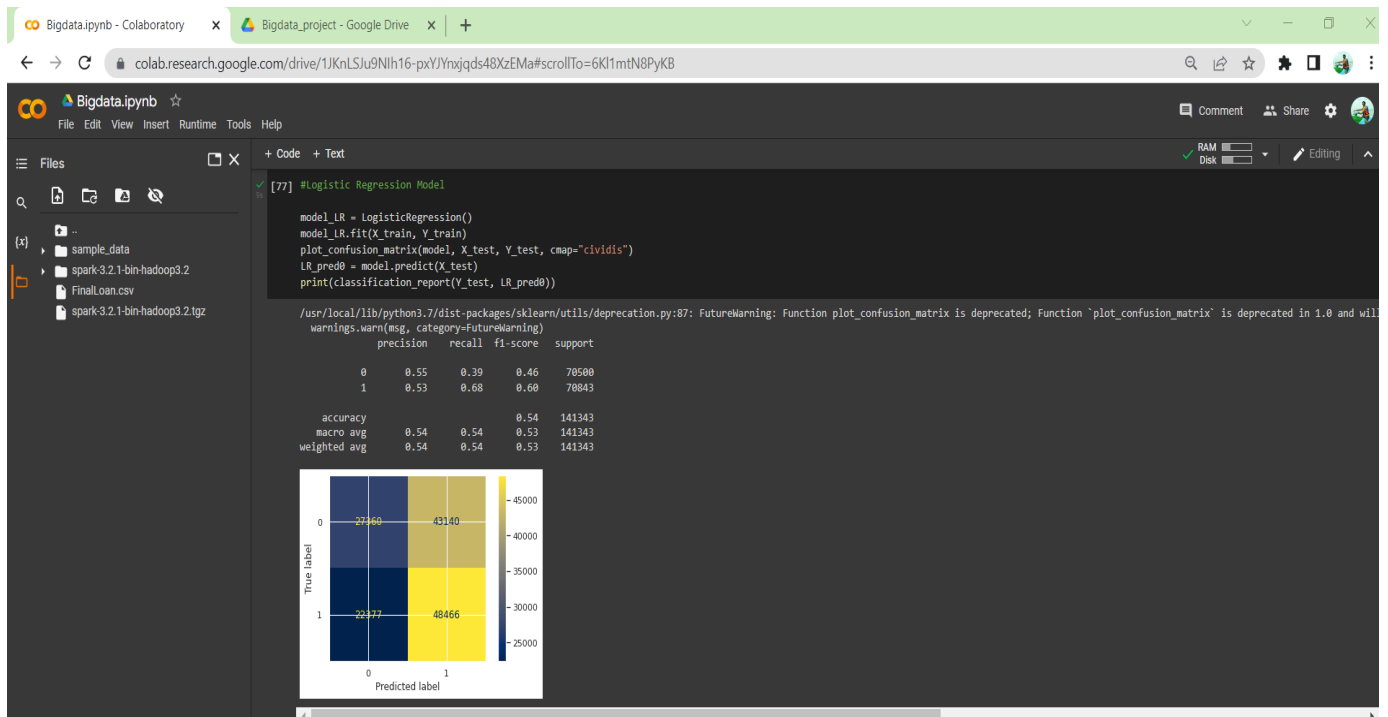


Figure: Logistic Regression model Evaluation after balancing

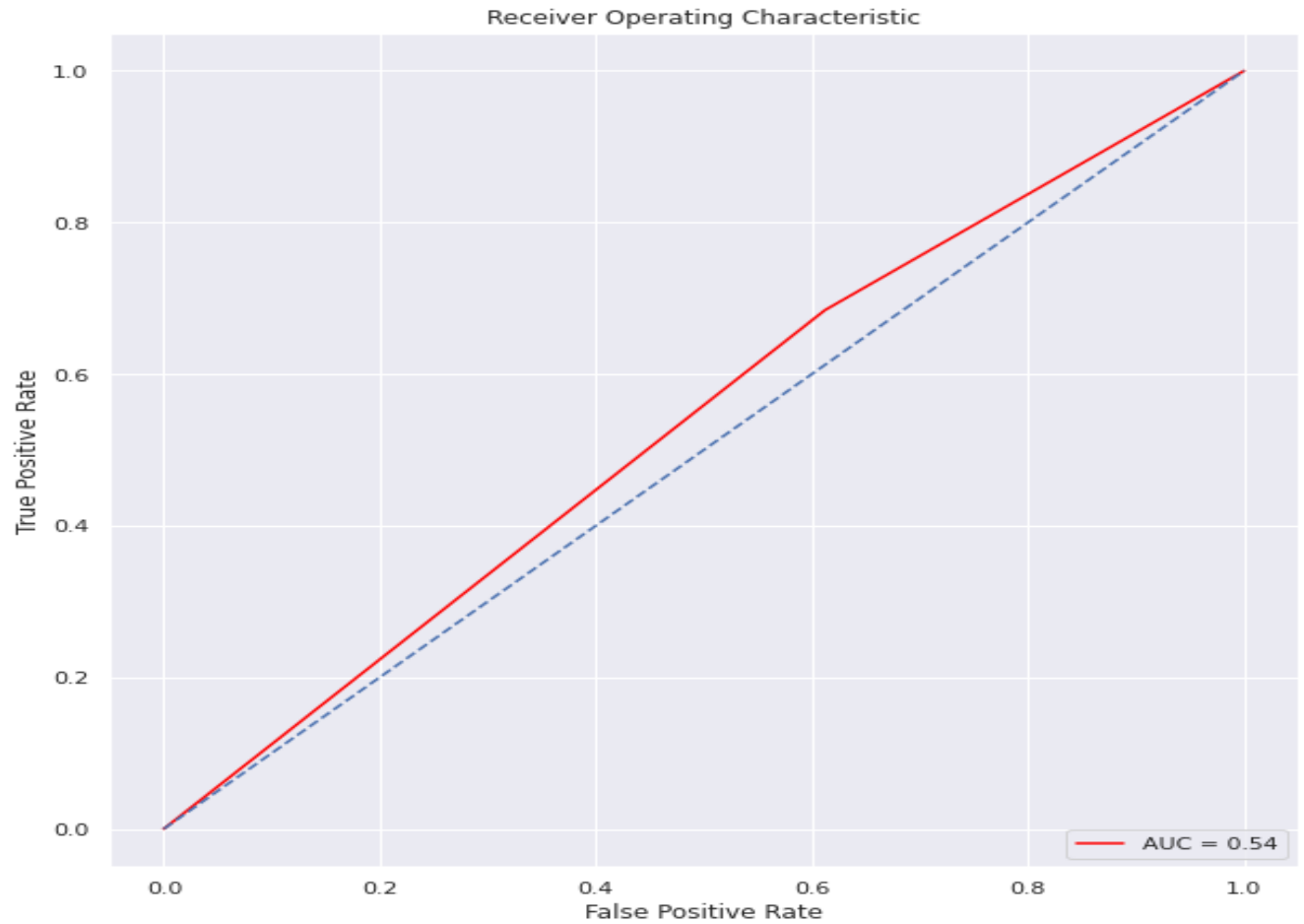


Figure: ROC curve of balanced dataset using Logistic Regression Model

Now, I use the Hot- encoder to transform the categorical value into numerical value and change into the array before feeding the model. After the training, the model gives the accuracy of 0.92 which is same as the previous in case of label encoder of unbalanced dataset and the roc curve is also same.

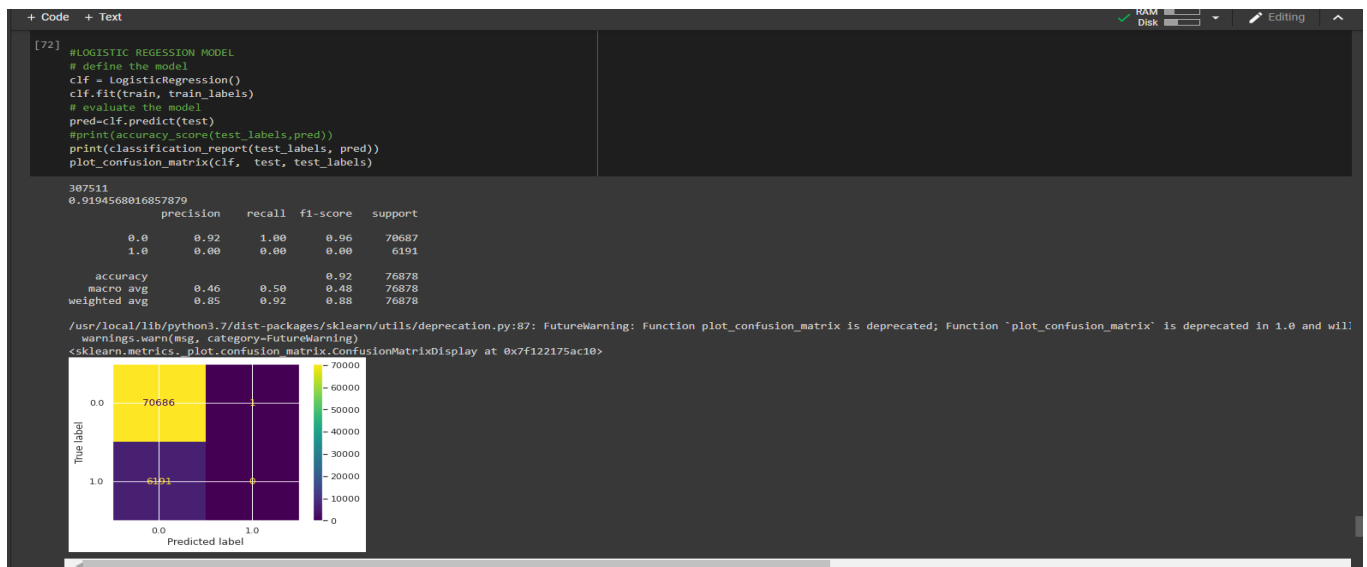


Figure: Logistic Regression Using Hot encoder of unbalanced data

For the balanced dataset, my model gives 57% accuracy which is 3% more than the previous using the label encoder in case of balanced dataset. I got the F-1 score 0.68 for 0 and 0.38 for 1. In the confusion matrix, the model predicts 6111 true positive, 1756 true negative, 4487 false positive and 1363 false negative. I get the auc value 0.55 which is more than the label encoder auc value and roc curve is shown below.

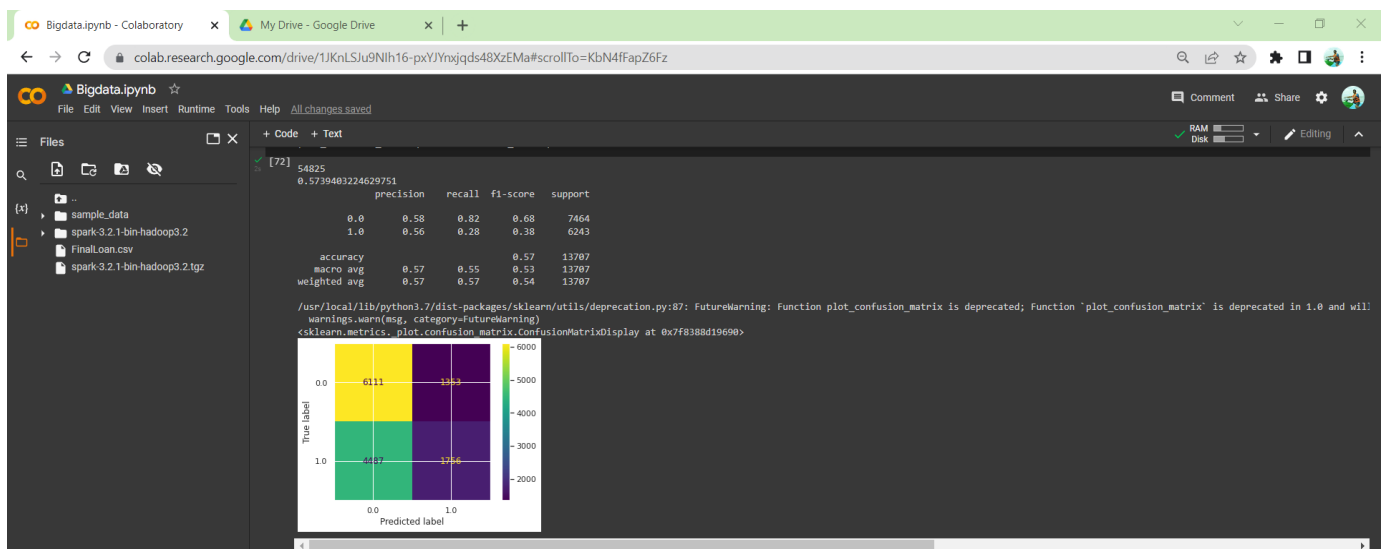


Figure: Logistic Regression Using Hot encoder of balanced data



Figure: ROC curve of balanced dataset using Logistic Regression Model and Hot-encoder

#### For Random Forest Classifier

At first, I trained the classifier on whole dataset with over 280K records where 280K are repayers and only 24k are repayers. The model was highly overfitted and predicted everything as 0 i.e repayers. We can see the precision and recall is 0 for 1 i.e defaulters. The classifier would always give outcome as repayer for any loan applicant which is undesirable for business.

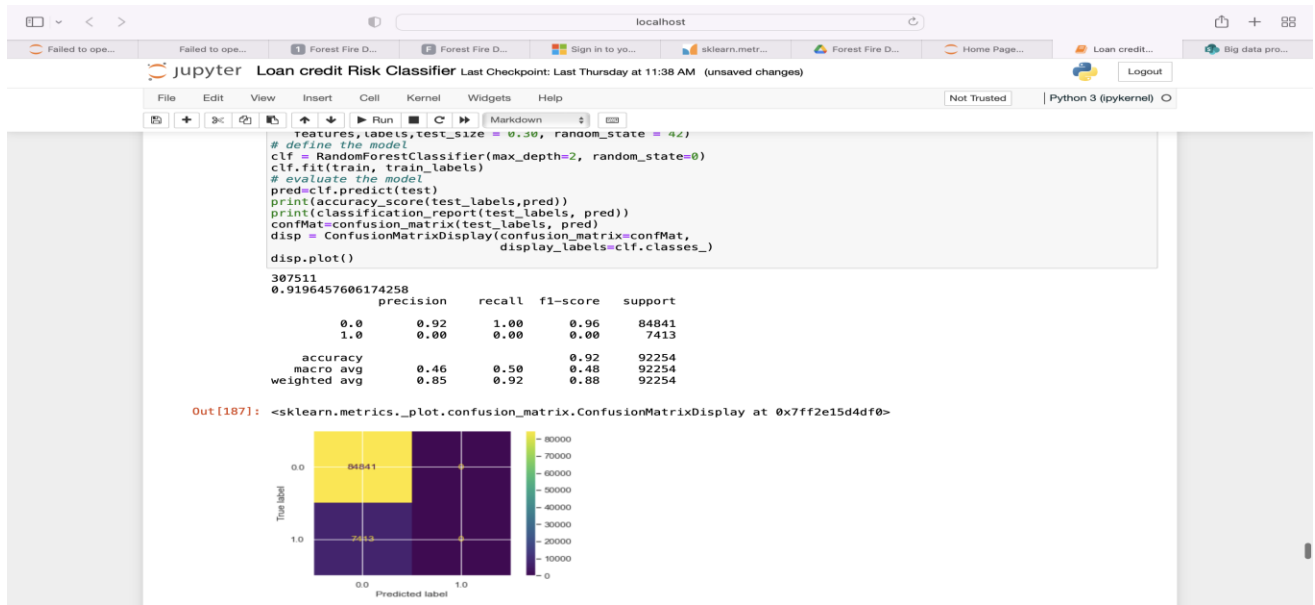


Fig Classification matrix and confusion matrix with imbalanced data

The ROC curve also says the same with 50% accuracy and shows the model built is highly undesirable.

```
In [188]: print("Roc AUC:", roc_auc_score(test_labels,pred,average='macro'))
fpr, tpr, thresholds = roc_curve(test_labels,pred)
plt.plot(fpr, tpr, label='Loan risk Pred - Random Forest')
plt.xlabel('False positive rate')
plt.ylabel('True positive rate')
plt.title('ROC curve')
plt.legend(loc='best')
plt.savefig('1.png')
plt.show()
```

Roc AUC: 0.5



Fig: ROC curve for highly imbalanced data using random forest classifier

Results after balancing

After balancing the dataset, the model is trained again and now it has an accuracy of 95% and confusion matrix shows the model built has good performance.



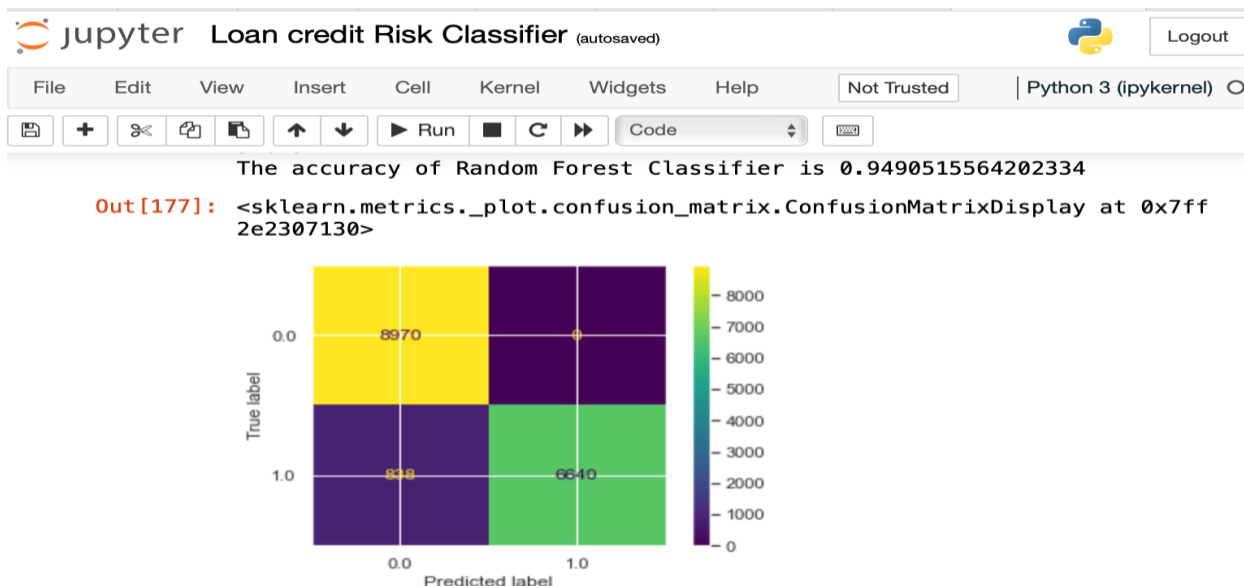


Fig : Confusion matrix with balanced data for random forest classifier

The classification report shows the recall for repayers and defaulters are 1.00 and 0.89 respectively. The precision for repayers and defaulters are 0.91 and 1.00 respectively. This shows how important it is to consider other metrics as accuracy can be misleading in highly imbalanced datasets.

```
In [179]: print(classification_report(test_labels, pred))
```

	precision	recall	f1-score	support
0.0	0.91	1.00	0.96	8970
1.0	1.00	0.89	0.94	7478
accuracy			0.95	16448
macro avg	0.96	0.94	0.95	16448
weighted avg	0.95	0.95	0.95	16448

The area under ROC curve is close to 1 which shows the performance of the classifier is good and is trustworthy to use for business purposes.

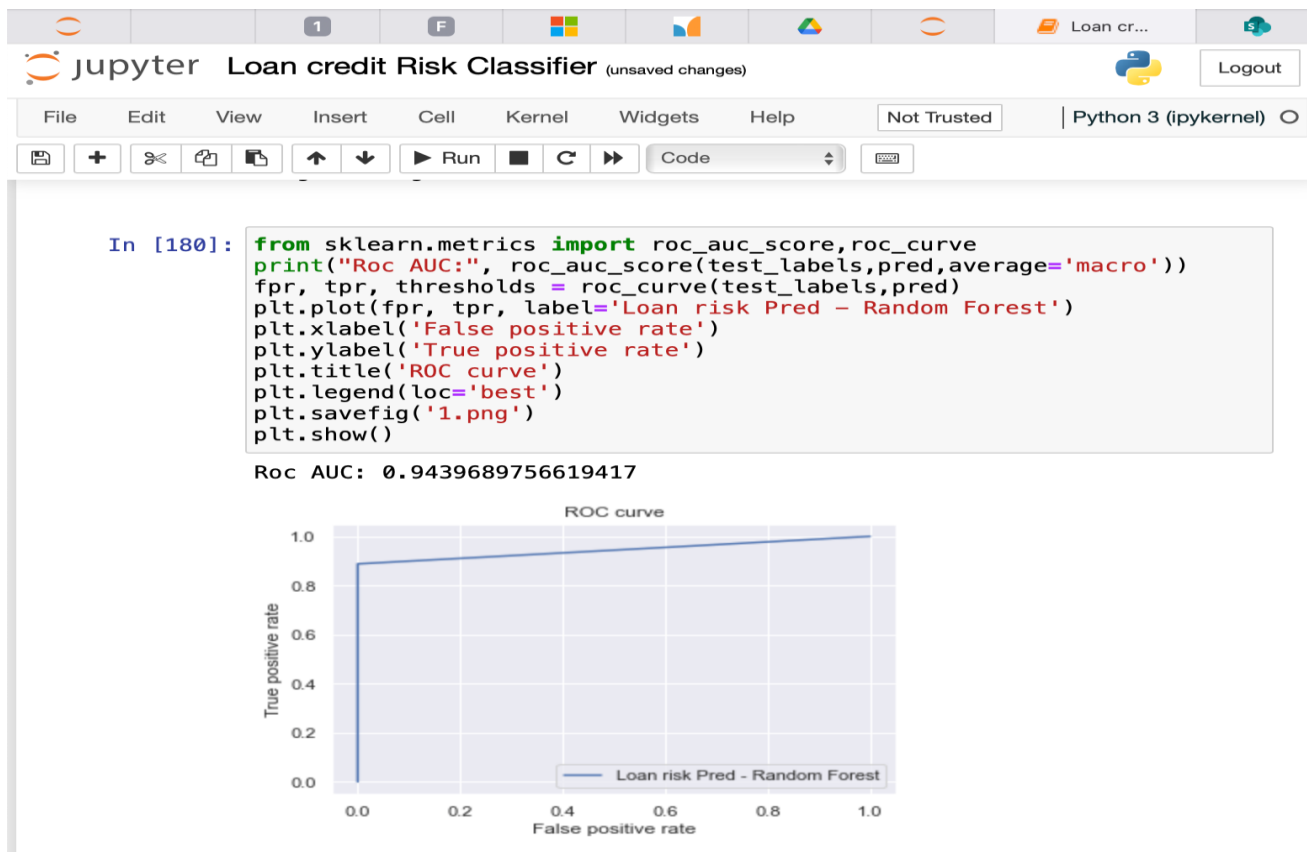
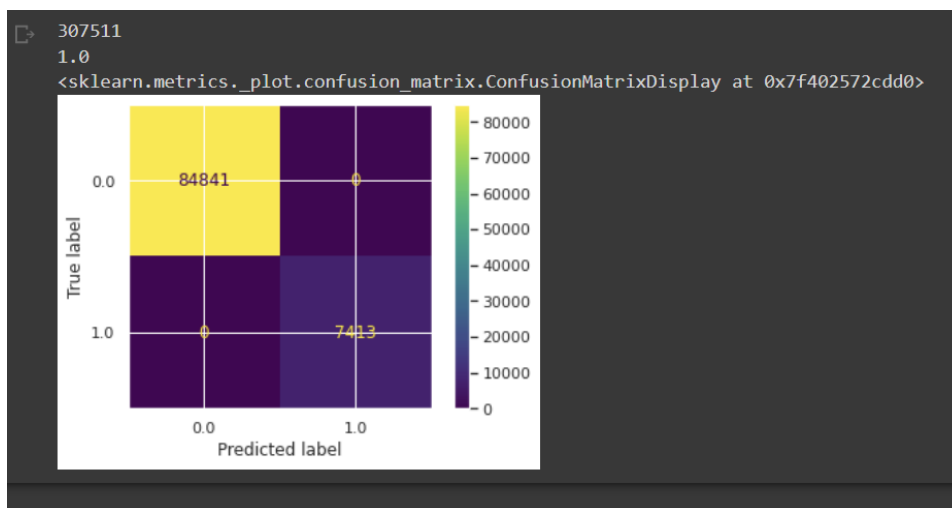


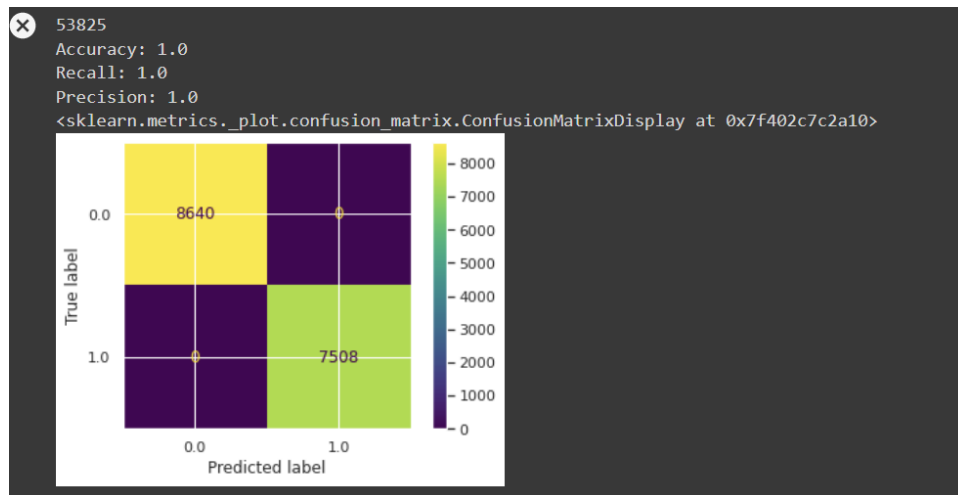
Fig: ROC curve for balanced data using random forest classifier

## Results for Decision tree

First when the unbalanced data was classified using the decision tree the model seemed to have overfitted as the accuracy was perfect with the confusion matrix as below:



Hence as mentioned before the data was balanced out and classified again with this classifier giving out the result as follows. Below is the confusion matrix,

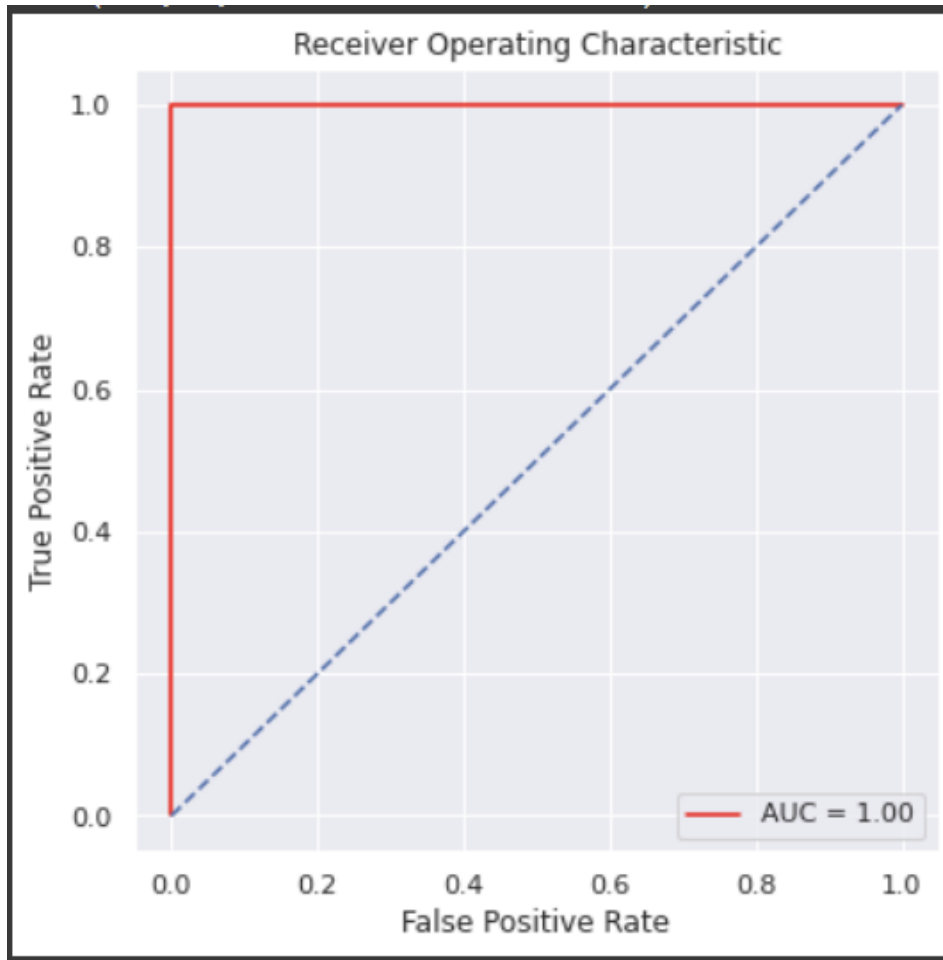


All the predictions were made perfectly using the decision tree as seen in the results.

```
getEvaluationMatrix(list(pred),list(test_labels))
```

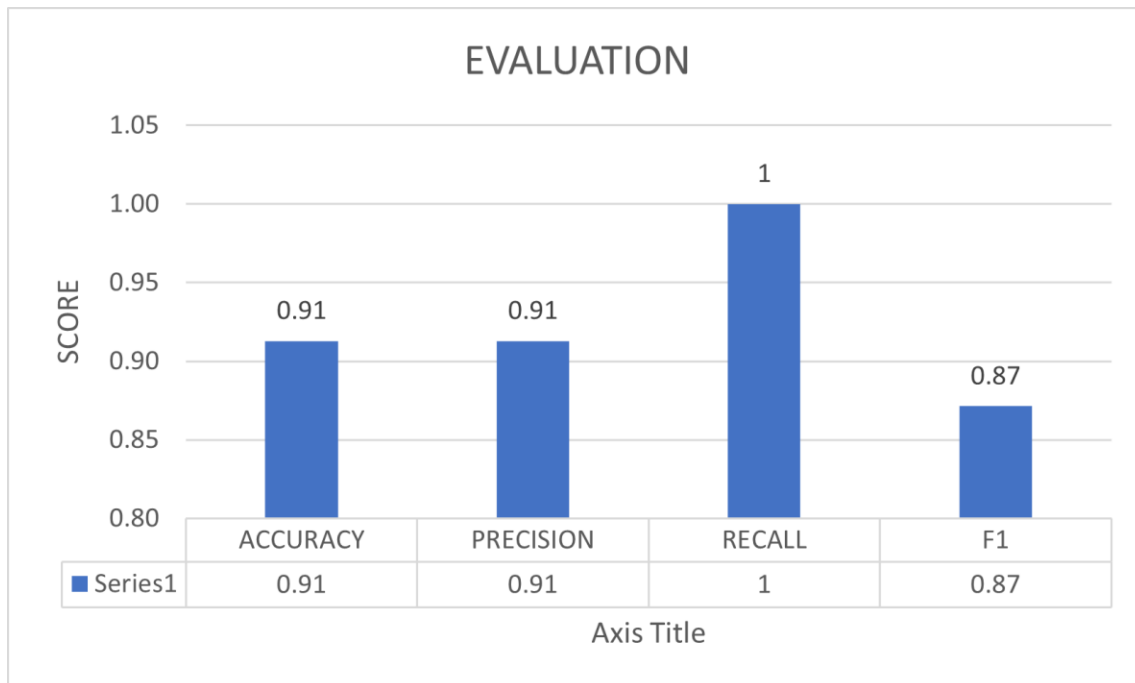
totalCount	-	16148
correctCount	-	16148
wrongCount	-	0
trueP	-	7508
trueN	-	8640
falseN	-	0
falseP	-	0
ratioWrong	-	0.0
ratioCorrect	-	1.0
Accuracy	-	1.0
Precision	-	100.0
Recall	-	100.0
F-1 Score	-	100.0
Sensitivity	-	100.0
Specificity	-	100.0

Also following is the ROC curve with area under the curve 1 which indicates the best classifier



**Results for Random Forest (special imputation):**

After running the Random Forest Classification Model, with unbalanced data and imputation by groups, the classifier evaluation parameters were the following.



considering the precision and the recall values, it can be said that all the defaulters were predicted correctly, but due to the precision value it can be said that more defaulters than expected were predicted, this means that the bank may reject some applications that were repayers.

## Project Management

### Conclusion

Of all the four classifiers trained, decision tree classifier is perfect for the dataset as it gave 100% accuracy on the test dataset of 16k+ records. Followed by random forest with 95% accuracy. So

## Implementation status report

### Work completed

Members	Feature Analysis and ML analytics	Contribution
Priya Enuganti	Data analysis, Pre-processing, Integrating spark and Hadoop. Handling missing values and categorical features. Training Random Forest Classifier on both balanced and Imbalanced dataset and compared the evaluation metrics for both the models using classification report and confusion matrix. Handled 25% of documentation	Analysis –100% as I worked on complete dataset to do analysis Analytics –100% I trained,tuned and evaluated random forest classifier Documentation -25%, the documentation was divided

		equally among the four of us.
Nayana Shrestha	Training the pre-processed data using the decision tree algorithm for both the balanced and unbalanced data and evaluating both of them using confusion matrix, accuracy, recall and precision.	Analysis – 100% Worked on complete dataset Analytics – 100% Trained data using Decision tree Documentation- 25% Completed 25% of document
Brayan Murillo	creating data frames in Spark from files in Hadoop DFS, analysis of data, pre-processed data (with special imputation) and training a random forest classifier and evaluating them and completing 25% of the document.	Analysis – 100% I Worked on complete dataset, I applied imputation by groups. Analytics – 100% I Trained with unbalanced and evaluate Random Forest classifier Documentation- 25% Completed 25% of document
Roshan Sah	Data Preprocessing, features extraction with label encoder and hot encoder to train the Logistic Regression and compare the performance of Model and Evalution is made using confusion Matric and Binary Classification.	Analysis – 100% Worked on complete dataset Analytics – 100% Trained data using Logisitic regression Documentation- 25% Completed 25% of document

#### • References/Bibliography

[1] (2022). Retrieved 1 May 2022, from [https://www.researchgate.net/figure/Pseudocode-of-logistic-regression\\_fig15\\_343120305](https://www.researchgate.net/figure/Pseudocode-of-logistic-regression_fig15_343120305)

[2] CHATTERJEE, A. (2020, August 8). *EDA : Bank Loan Default Risk Analysis*. Kaggle. Retrieved March 30, 2022, from <https://www.kaggle.com/code/amritachatterjee09/eda-bank-loan-default-risk-analysis/notebook>

[3] Greene, L. L. (1989). An Economic Analysis of Student Loan Default. *Educational Evaluation and Policy Analysis*, 11(1), 61–68. <https://doi.org/10.3102/01623737011001061>

- [4] Hillman, N. W. (2014). College on Credit: A Multilevel Analysis of Student Loan Default. *The Review of Higher Education*, 37(2), 169–195. <https://doi.org/10.1353/rhe.2014.0011>
- [5] Sheikh, M. A., Goel, A. K., & Kumar, T. (2020). An Approach for Prediction of Loan Approval using Machine Learning Algorithm. 2020 International Conference on Electronics and Sustainable Communication Systems (ICESC). <https://doi.org/10.1109/icesc48915.2020.9155614>
- [7] sklearn.linear\_model.LogisticRegression. (2022). Retrieved 1 April 2022, from [https://scikit-learn.org/stable/modules/generated/sklearn.linear\\_model.LogisticRegression.html](https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html)
- [8] Classification — Learning Apache Spark with Python documentation. (2022). Retrieved 15 March 2022, from <https://runawayhorse001.github.io/LearningApacheSpark/classification.html>
- [9] Machine Learning Decision Tree Classification Algorithm - Javatpoint. (2022). Retrieved 1 May 2022, from <https://www.javatpoint.com/machine-learning-decision-tree-classification-algorithm>
- [10] Machine Learning Random Forest Algorithm - Javatpoint. (2022). Retrieved 1 May 2022, from <https://www.javatpoint.com/machine-learning-random-forest-algorithm>

### **Submission Video Links**

Priya Enuganti :

<https://drive.google.com/file/d/1uP4FCbOQeoeAJtMcSqoC8GOK80lvLIWB/view?usp=sharing>

Nayana Shrestha:

<https://drive.google.com/file/d/1rfHWXzJHjxY LVbJV95kEh6loxxVHvzC/view?usp=sharing>

Roshan Sah:

<https://drive.google.com/drive/folders/1IUEb3LibZn PKs8AQnoyxNUDw5YDsPCk?usp=sharing>

Brayan Murillo

<https://drive.google.com/file/d/1-PmsREE-pHnFRmKT2x683WIBv2xVFwxa/view?usp=sharing>