LOAN DEFAULT PREDICTION

UNIVERSITY OF MUMBAI
DEPARTMENT OF STATISTICS
VIDYANAGARI MUMBAI - 4000098

University of Mumbai

CERTIFICATE

This is to certify that

VAISHNAVI KHAIRE
TEJAL KURLE
TANMAY PATIL
ADITYA SARODE

Have successfully completed their project "**Loan Default Prediction**" as per the course curriculum for the award of M.Sc. Statistics degree for the year 2021-2022 by University of Mumbai.

This work to the best of our knowledge and belief is original.

PROJECT MENTOR
Mr. Priyesh Tiwari.

HEAD OF THE DEPARTMENT
Dr. Vaijayanti Dixit

ACKNOWLEDGMENT

We are greatly indebted to Mr. Priyesh Tiwari who has been a tremendous mentor to us, constantly encouraging our research through his unremitting support and guidance.

We also wish to express special admiration and gratitude to Prof. Kulkarni sir for his co-operation and invaluable suggestions towards our project.

Lastly, our sincere thanks to all the faculty members, the non-teaching staff and our classmates for their support to bring out a research friendly environment.

INDEX

INTRODUCTION

Loan is the lending of money by one or more individuals or an organization. The recipient (i.e., the borrower) incurs a debt and is usually liable to pay interest on that debt until it is repaid as well as to repay the principal amount borrowed. Loans have become a solution to the financial problems of many.
On the other hand loans are the biggest source of income for banks and Non-Banking Financial sectors but it can also be the source of biggest risk for the banking and Non-Banking Financial sectors . So a solution is needed for the banks to maximize their profit and minimize the risk involved in granting a loan to a customer.

Warren Buffet famously said that he follows two rules for investing his money. The first rule says: "**Never lose money**." and the second rule adds: "**Never forget the first rule**." This rule in the context of retail banking or P2P lending means that the money should not be lent to someone who will not pay them back. There exist numerous classification algorithms, such as Logistic regression or Random Forest, for assessment of borrower's creditworthiness. These classification techniques support the decision-making process of whether to lend money to a borrower or not.

Peer-to-peer leading has opened up access to the credit market to clienteles that previously were left out by banks. Due to the fact that they were not bankable in credit scoring terms, the likelihood of default was bigger than the probability of profit for banks (Gomber, Kauffman, Poker, & Webci, 2018). Peer-to-peer lending opens up an opportunity for these individual borrowers to acquire credit. The hardest challenge for investors is to allocate their assets to the right opportunity. This problem is traditionally solved by financial intermediaries, since the investor does not have the information required to make a valid decision. Therefore, the financial intermediaries allocate the money for the small investors and gain money in this way. (Iyer. Ijaz khwaja. Luttmer & Shue ,2016).

Due to the absence of a third party, investors are exposed to different risks compared to the traditional situation. To investigate these risks, a few effects should be measured. First the determinants of loan default within this new peer-to-peer lending market should be investigated. Secondly, after this research it should be clear what determines the loan default. This research will be structured in the following parts;
we will focus on the overview of the data that is used in this research including the descriptive statistics of the variables.
In the next part the methodology of the research is stated and described. This will introduce various machine learning classifiers that are used. The results of these tests will be presented and considered thoroughly. At last, the results will form the basis for the conclusions and recommendations of this research.

The aim is to implement different statistical learning techniques to try to estimate the probability of default for each loan. Machine learning models are used to Classify a set of unseen data and statistical metrics are used to compare the results.

OBJECTIVE

1. To build a predictive Model.

2. To compute the probability/score of the defaulters.

3. Developing an understanding of the behavioural pattern of the customers before granting the loan.

METHODOLOGY

Data Collection: The source of the data is from the Kaggle repository. The dataset consists of 1007987 rows and 149 columns.

Exploratory Data Analysis:

- **Univariate:** Visualization of how a particular variable is affecting the Loan defaulter rate.
- **Bivariate:** Graphically, checking if any two variables are affecting the Loan defaulter rate.
- **Automation EDA:** In order to get an understanding of each variable, we have used DataPrep library which provides us with summary of the variable (i.e., measure of central tendencies and dispersion, Quartiles, etc.), distribution curve, Q-Q plot (which tells us whether the data is normally distributed or not), Boxplot (which was helpful for outlier detection), Pearson's correlation heat maps, etc.
- **Tableau:** we have used tableau for creating visualizations to obtain valuable insights from the data.

Data Preprocessing:

- **Missing Data Threshold:** Variables with more than 50% missing values were removed. (I.e., Around 58 variables were dropped/removed.)

- **Variance Threshold:** Any feature with only 1 category in the dataset would not provide any information to the model e.g. 'policy_code', 'hardship_type', 'deferral_term', 'hardship_length'. Feature with less variance would not be useful for the model, thus we dropped such variables.

- **Outlier Detection:** Only the extreme end outliers were removed & the rest of the outliers were retained since lot of important data can be loosed.

- **Handling Missing Values:** As there are some missing values in the dataset, so these missing values are replaced by appropriate measure of central tendencies and some missing values were dropped.

- **Feature Transformation:** Variables that were named categorical but are numeric e.g., emp_length is converted into numeric variables. New variables were created by combining 2 or 3 variables together e.g., fico is created using fico_low_range and fico_high_range. We retained the new variables and dropped the variables used to create the new variable.

- **Scaling of Feature variables:** Feature Scaling was used to standardize the independent features present in the data in a fixed range. i.e., between 0 and 1. And we used Standard Scalar for scaling.

- **Multicollinearity Detection:** Detection was done using Correlation Heatmap and Variance Inflation Factor (VIF).

- **Encoding of Categorical variables:** Encoding was used for converting categorical variables into numerical values so that it could be easily fitted to a machine learning model.

- **Train-Test Split:** For splitting the dataset we have used Stratified Shuffle Split, as we had an imbalanced dataset, so in order to keep the proportion of categories in the target variable same in both train and test dataset we have used this technique.

Modelling:

Models used were as following,

1) Logistic Regression,
2) Random Forest,
3) eXtreme Gradient Boosting.

Evaluation: Models were evaluated on the basis of metrics like ROC-AUC Score, F1 Score, and Recall Value. The model with best metrics was select for the prediction of loan defaulters.

Variables

loan_amnt	The listed amount of the loan applied for by the borrower. If at some point in time, the credit department reduces the loan amount, then it will be reflected in this value.
term	The number of payments on the loan. Values are in months and can be either 36 or 60.
instalment	The monthly payment owed by the borrower if the loan originates.
grade	LC assigned loan grade
sub_grade	LC assigned loan subgrade
emp_title	The job title supplied by the Borrower when applying for the loan.
emp_length	Employment length in years. Possible values are between 0 and 10 where 0 means less than one year and 10 means ten or more years.
home_ownership	The home ownership status provided by the borrower during registration or obtained from the credit report. Our values are: RENT, OWN, MORTGAGE, OTHER
annual_inc	The self-reported annual income provided by the borrower during registration.
verification_status	Indicates if income was verified by LC, not verified, or if the income source was verified
issue_d	The month which the loan was funded
purpose	A category provided by the borrower for the loan request.
title	The loan title provided by the borrower
addr_state	The state provided by the borrower in the loan application
dti	A ratio calculated using the borrower's total monthly debt payments on the total debt obligations, excluding mortgage and the requested LC loan, divided by the borrower's self-reported monthly income.
earliest_cr_line	The month the borrower's earliest reported credit line was opened
open_acc	The number of open credit lines in the borrower's credit file.
pub_rec	Number of derogatory public records
revol_bal	Total credit revolving balance
revol_util	Revolving line utilization rate, or the amount of credit the borrower is using relative to all available revolving credit.
total_acc	The total number of credit lines currently in the borrower's credit file
initial_list_status	The initial listing status of the loan. Possible values are – W, F
application_type	Indicates whether the loan is an individual application or a joint application with two co-borrowers
mort_acc	Number of mortgage accounts.
pub_rec_bankruptcies	Number of public record bankruptcies
fico	A FICO score is a three-digit number, typically on a 300-850 range, that tells lenders how likely a consumer is to repay borrowed money based on their credit history.

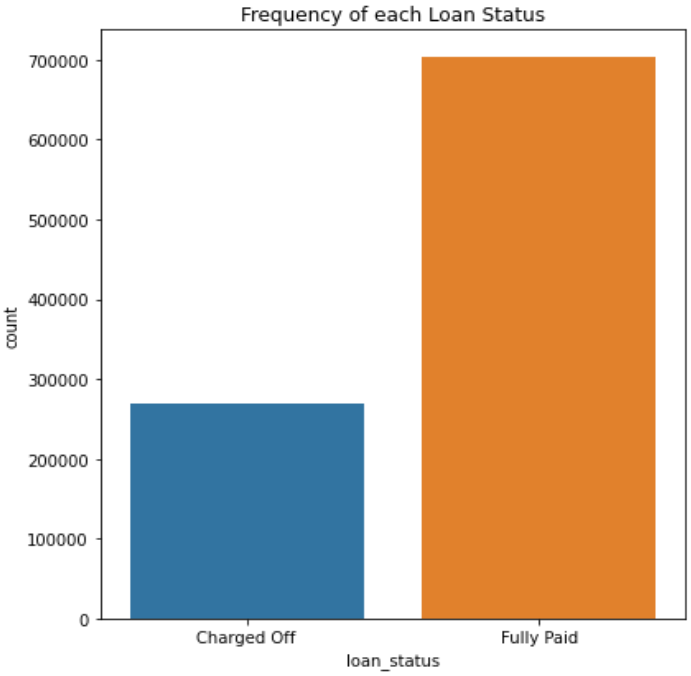
Since we don't have a clear understanding for many features, it would be unwise to include them into our analysis without truly understanding a feature. Moreover, the goal of this project is to predict whether a loan will be paid off before making the decision to lend the loan. Therefore, we would like to remove the features that were unavailable before lending a loan. So, we keep only those variables that gives information about the customer before lending them the loan.

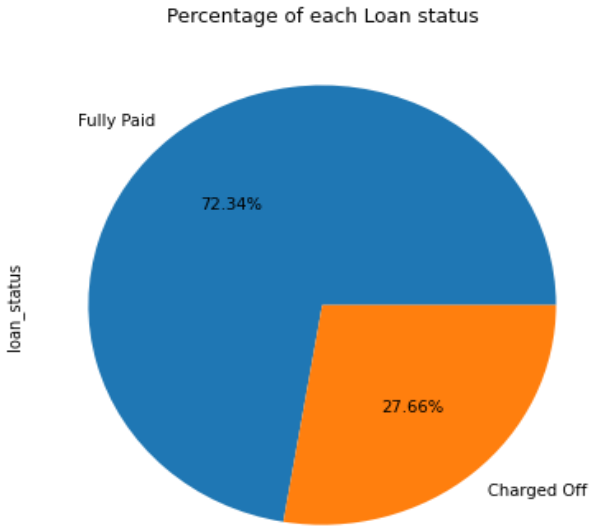
Target variable:

The target column was identified as loan_status which had the following value counts:

Fully Paid	702354
Charged Off	268559
Late (31-120 days)	21467
In Grace Period	8436
Late (16-30 days)	4349
Does not meet the credit policy. Status:Fully Paid	1988
Does not meet the credit policy. Status:Charged Off	761
Default	40

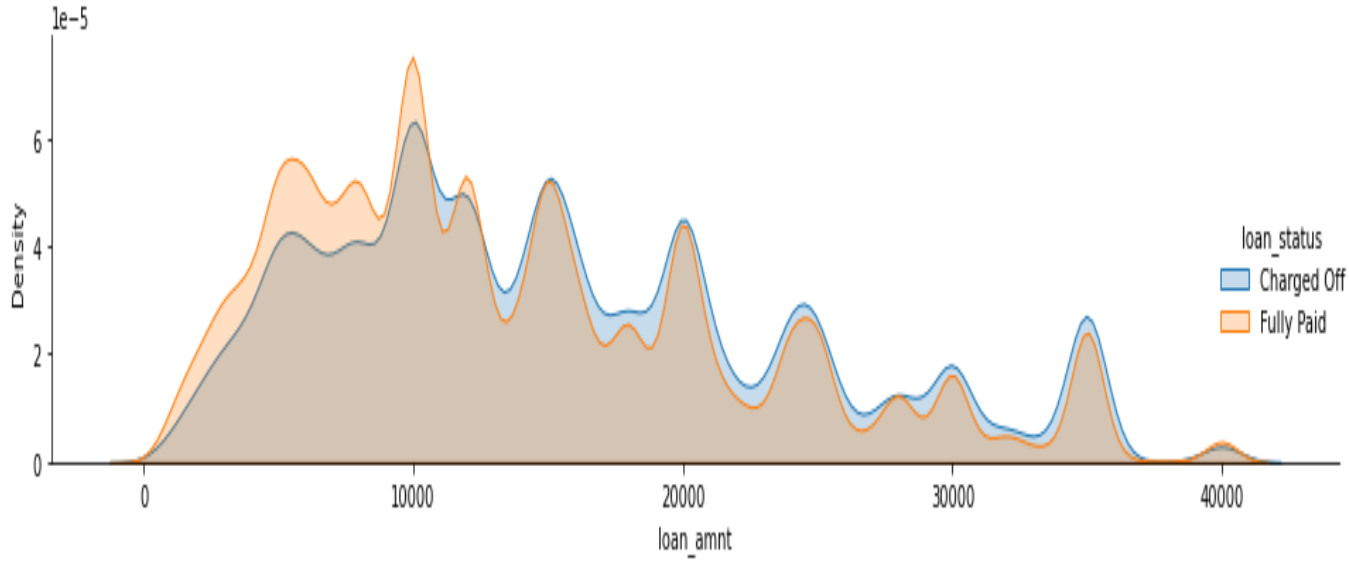
For the analysis only the Full Paid loans and the Charged Off/ Defaulted Loans have been taken into consideration.





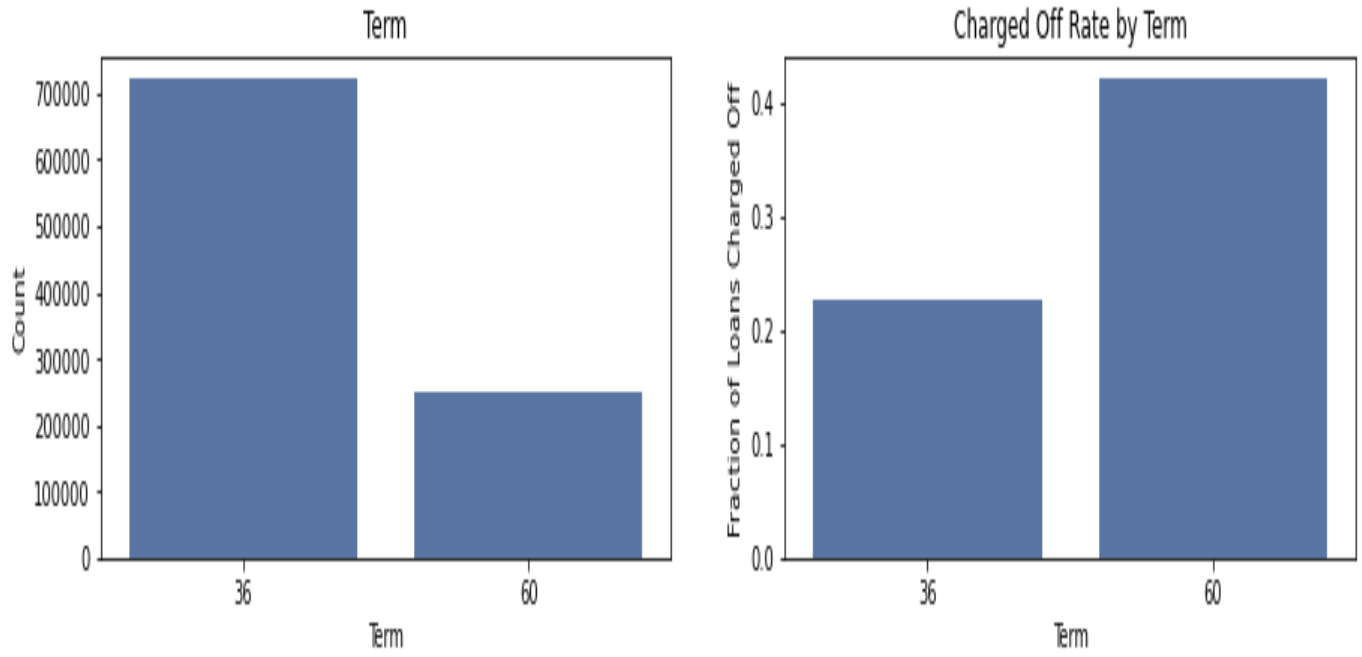
Exploratory Data Analysis:

Loan_amnt:



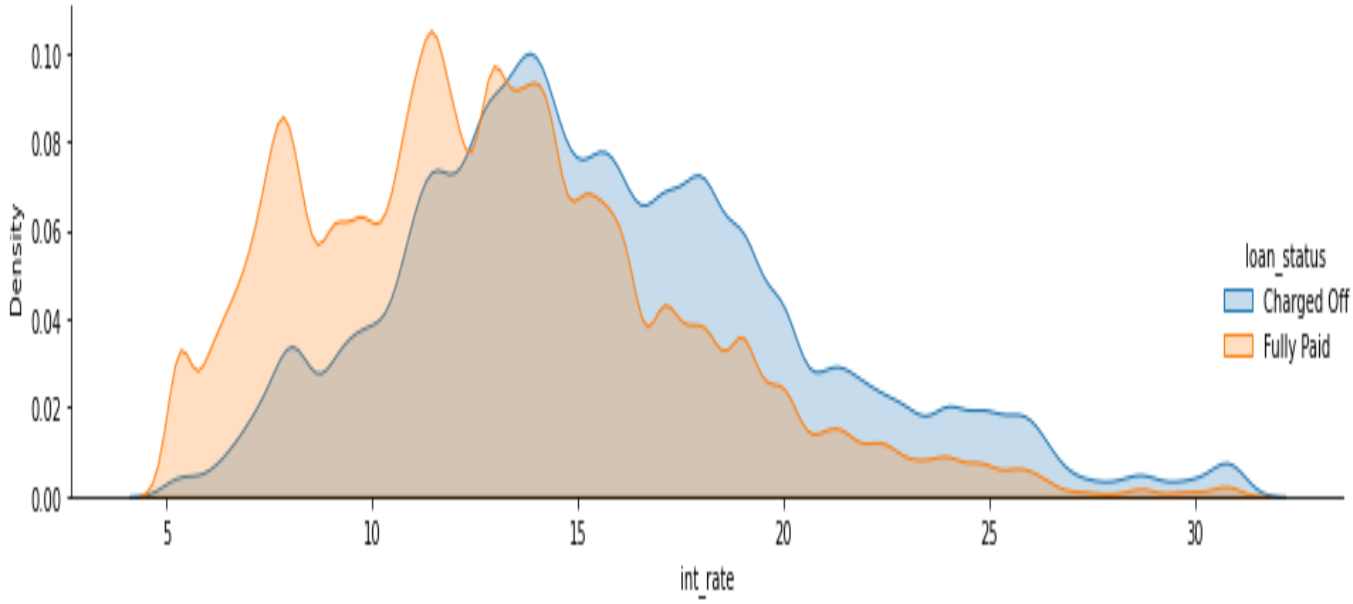
As loan amount increases there is slight tendency of customer to default.

Term:



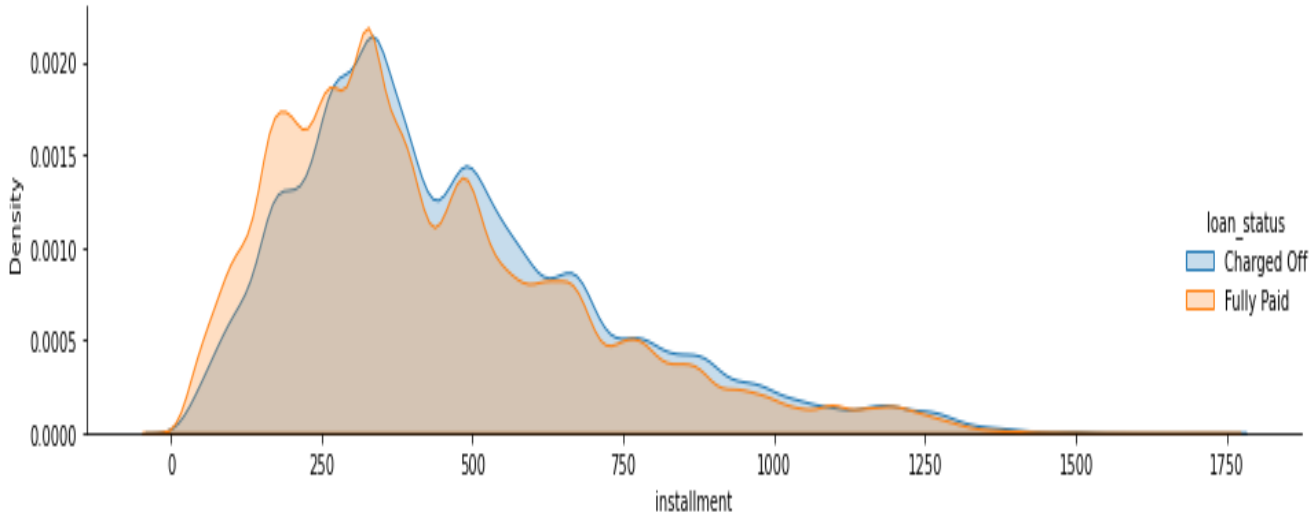
Customers with 60 months term plan tends to default more the 36 months term. In 36 months, 24% of the customers are defaulting the loan, whereas in 60 months 45% of the customers are defaulting the loans.

Interest Rate:



The higher the interest rate is, the more likely for being charged off.

Installment:

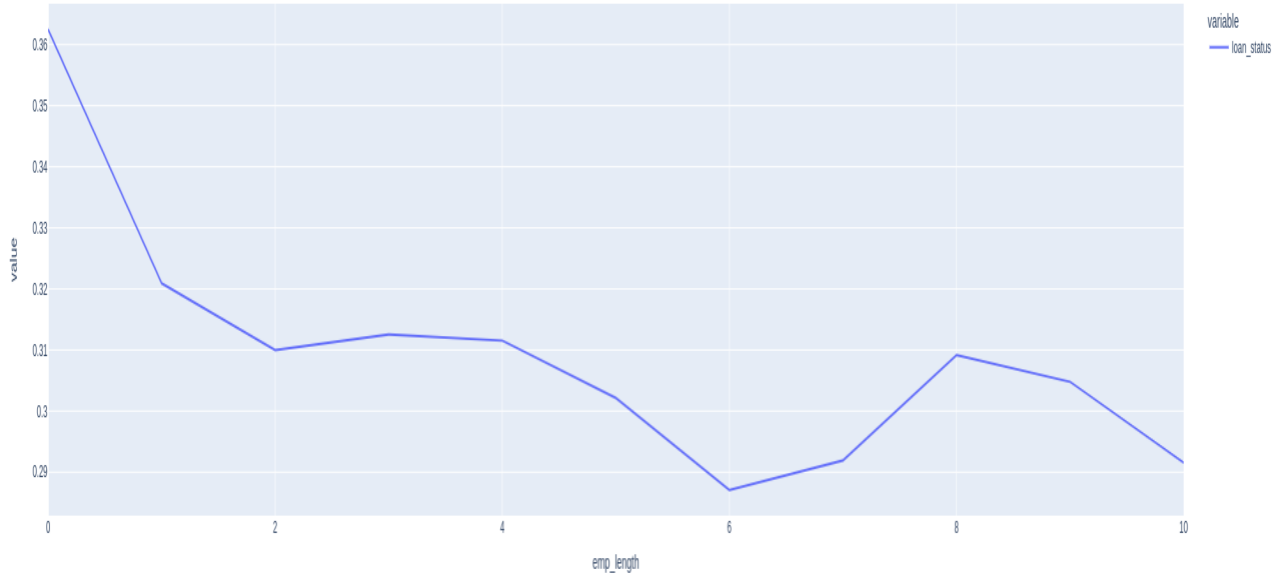


Charged-off loans tend to have higher installments but the difference of the mean values is very small.

Grade:

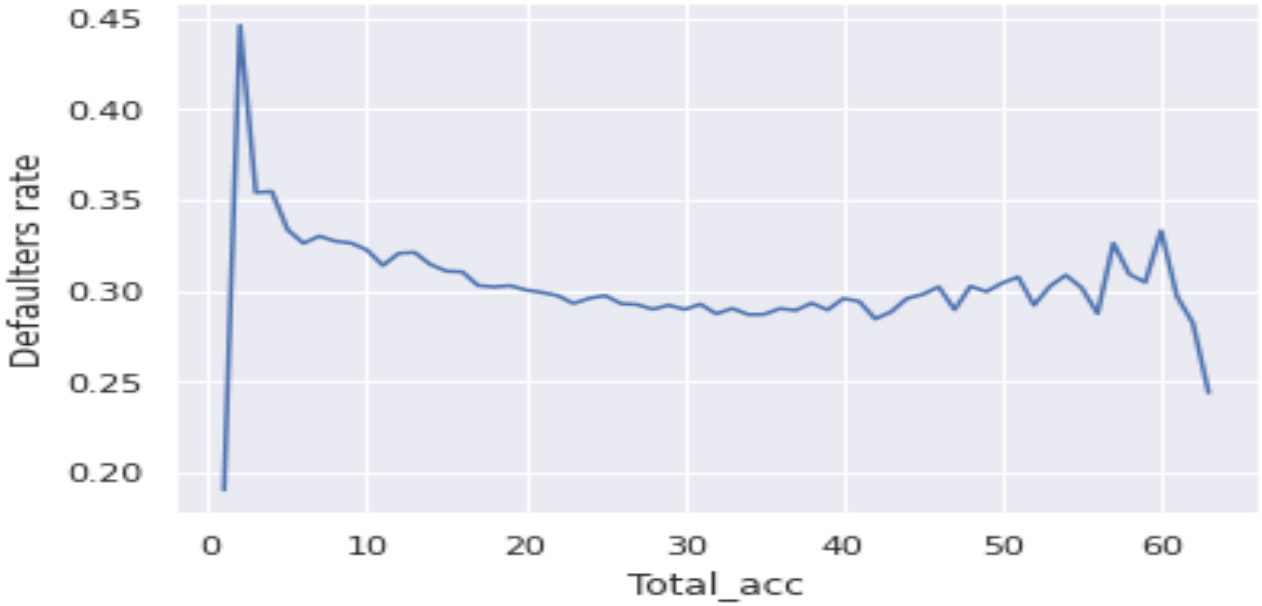
As the grade increases, the defaulters Rate also increases.

emp_length:



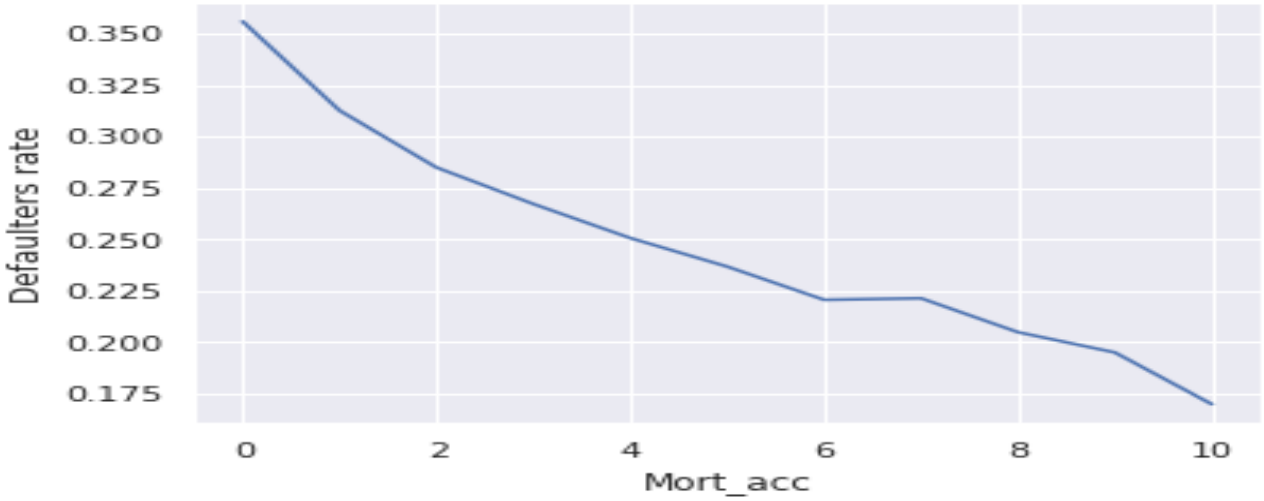
As employment length increases, the Defaulters rate decreases.

total_acc:



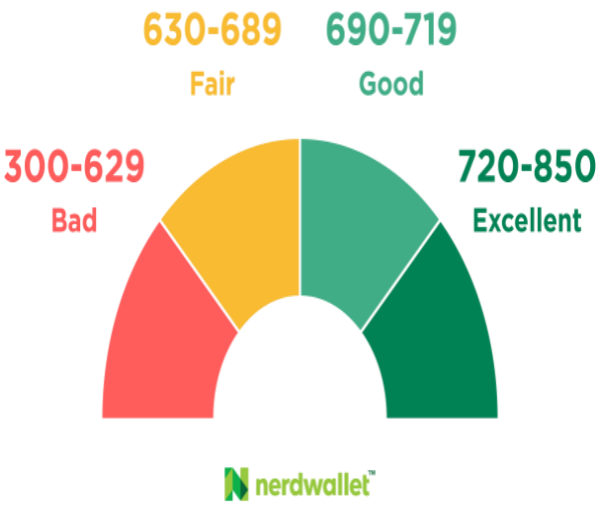
Defaulters rate is high with total_acc lying between 0-10, and is some-what having constant defaulters' rate there-after.

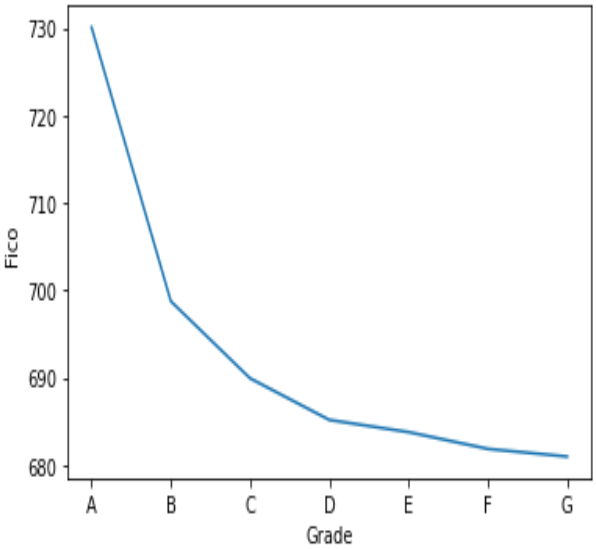
Mort_acc:



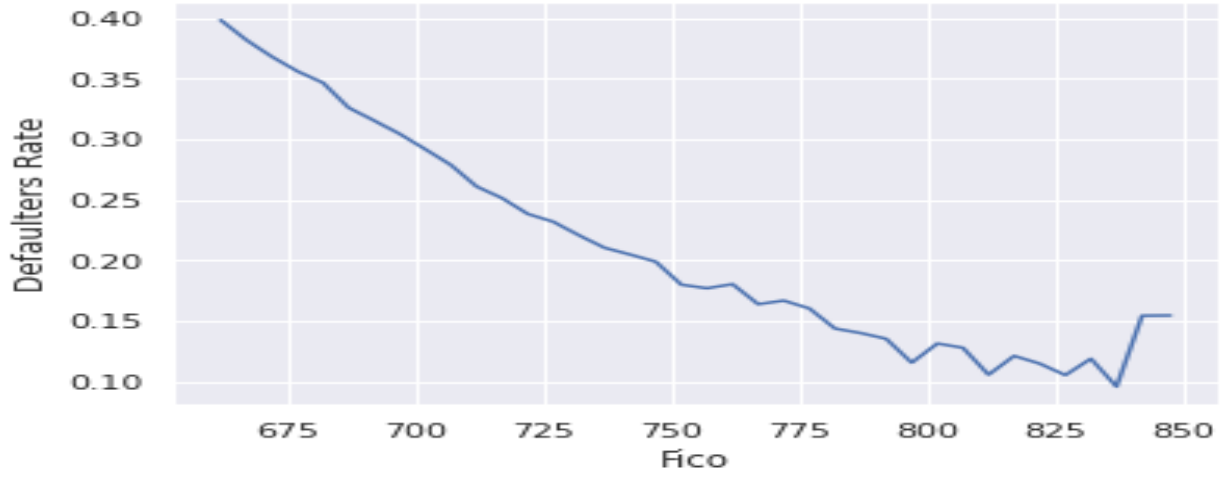
The more the Mortgage account, the less is the Defaulters Rate.

Fico:

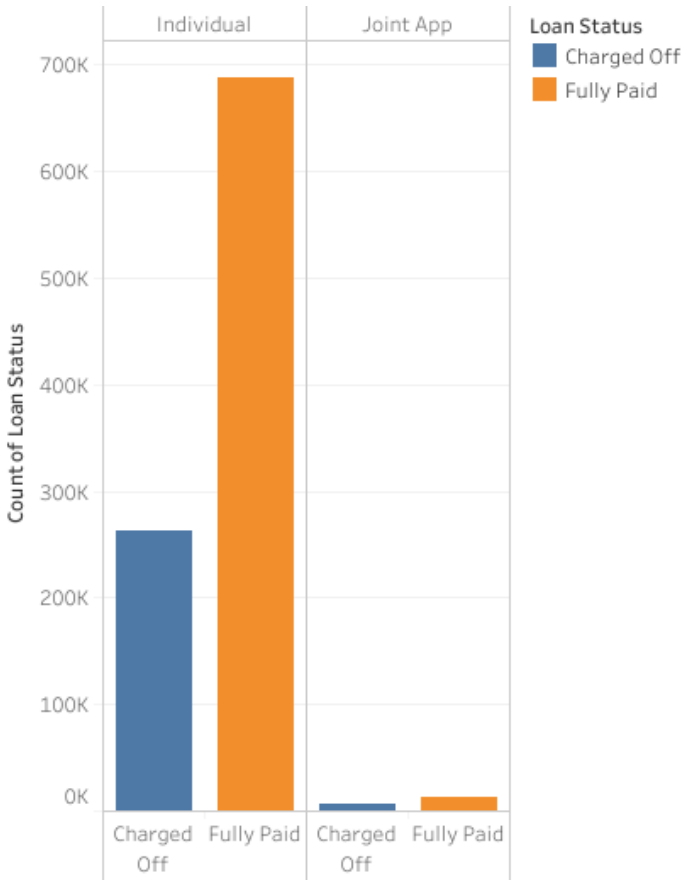




As the Fico Score decrease, the Grades increase.



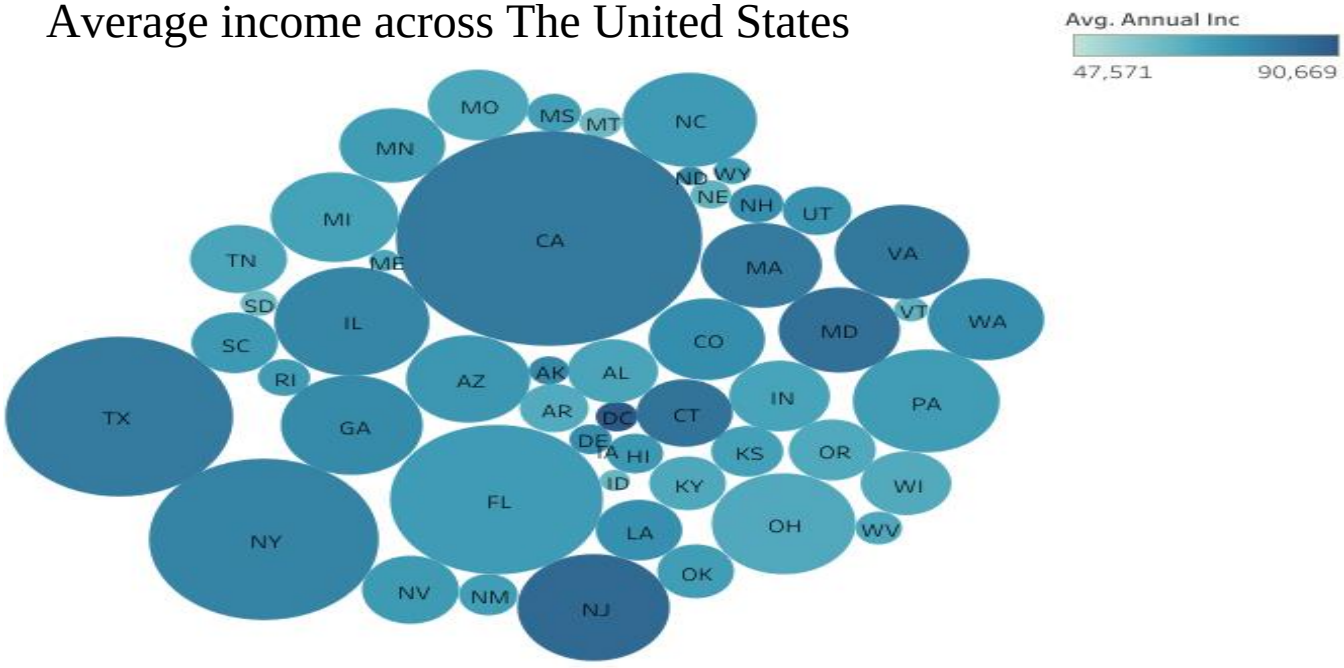
As the FICO score increases, the Defaulters Rate decreases.

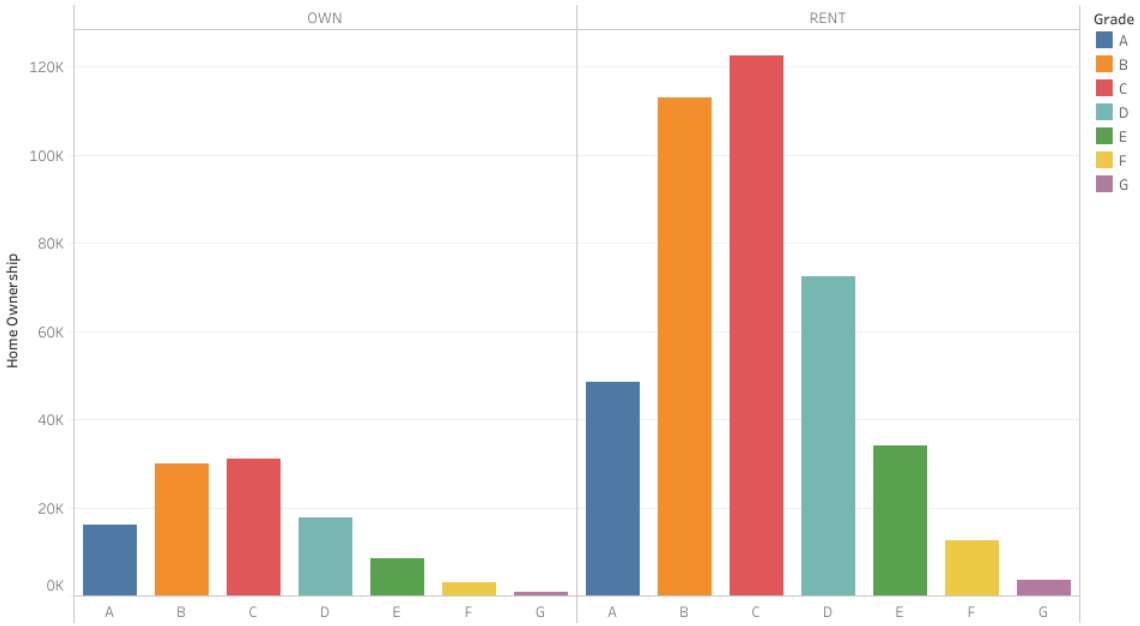


Joint applicants tend to default more than individual applicants.

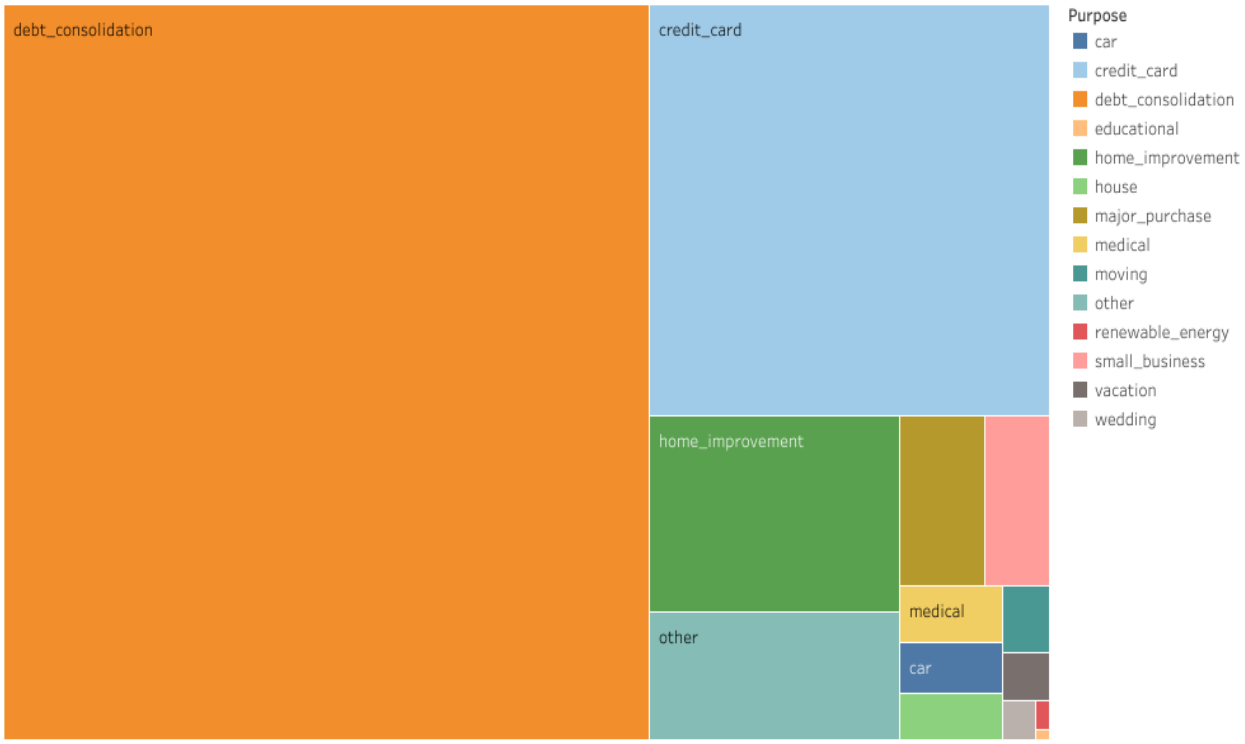
Joint applicant has a 43% chance of defaulting a loan whereas it is 30% for individual applicants.

Average income across The United States





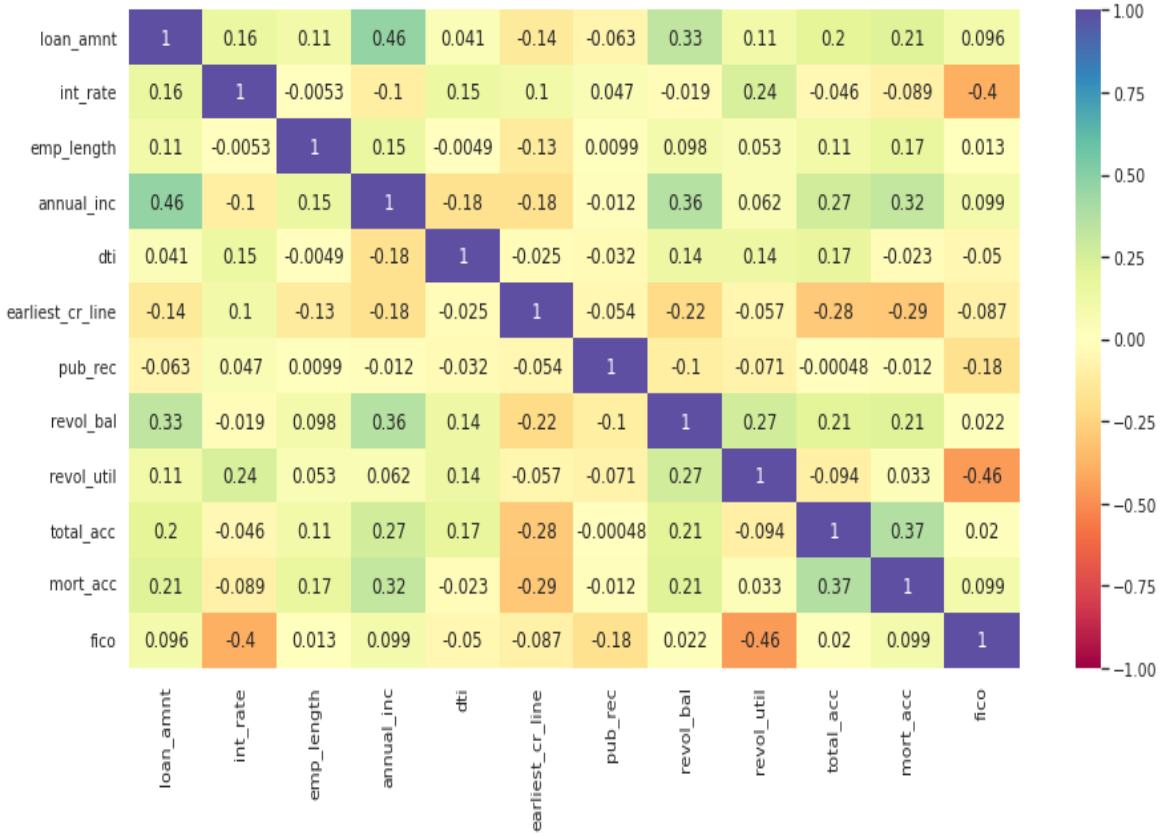
A customer living on rent tends to apply for a loan more than those owning a house.



Maximum loans issued are for debt consolidation and credit card.

Multicollinearity:

- Correlation Heatmap:



- Variance Inflation Factor: It is a measure of the amount of multicollinearity in a set of multiple regression variables.

$$VIF_i = \frac{1}{1 - R_i^2}$$

VIF > 10 means Serious Multicollinearity is present. So, we have removed variables having VIF >10.

So, the variables that were dropped are int_rate(VIF = 12.000234), earliest_cr_line(VIF = 935.114577)

Association between Categorical Variables:

- **Chi-square Test of Independence:**
  The Pearson's Chi-Square statistical hypothesis is a test for independence between categorical variables.

  H0: There is no association between two given Categorical Variables.
  H1: There is association between two given Categorical Variables.

$$X^2 = \sum \frac{(\text{Observed value} - \text{Expected value})^2}{\text{Expected value}}$$

  Decision Criterion: reject H0, if p-value < 0.05

- **Crammer's V and Phi:**
  It is a measure of association between two categorical variables, giving a value between 0 and +1 (inclusive). It is based on Pearson's chi-squared statistic and was published by Harald Cramer in 1946.

$$\phi_c = \sqrt{\frac{\chi^2}{N(k-1)}}$$

Phi and Cramer's V.	Interpretation
> 0.25	Very strong
> 0.15	Strong
> 0.10	Moderate
> 0.05	Weak
> 0	No or very weak

From Chi-square test of independence, we get that all the categorical variables are associated with each other. But Crammer's V tells us that, how much the variables are associated. 'Term' and 'grade' are very strongly associated with Crammer's V value 0.4366. 'verification_status' and 'grade' are strongly associated with Crammer's V value 0.1789. Rest variables have very weak or week association with Crammer's V and phi value ranging between 0 – 0.1. Therefore, we drop the variable 'grade'.

Performance Metrics:

- Confusion matrix: A confusion matrix is a table that is often used to describe the performance of a classification model (or "classifier") on a set of test data for which the true values are known. The confusion matrix itself is relatively simple to understand, but the related terminology can be confusing.

		Predicted	
		Negative	**Positive**
Actual	**Negative**	True Negative	False Positive
	Positive	False Negative	True Positive

- Precision: Out of all the positive predicted, what percentage is truly positive.

$$\boldsymbol{precision} = \frac{tp}{tp + fp}$$

- Recall / Sensitivity: Out of the total positive, what percentages are predicted positive. It is the same as TPR (true positive rate).

$$\boldsymbol{recall} = \frac{tp}{tp + fn}$$

- F1-Score: It is the harmonic mean of precision and recall. It takes both false positive and false negatives into account. Therefore, it performs well on an imbalanced dataset.

$$F1\ score = \frac{2}{\frac{1}{Precision} + \frac{1}{Recall}} = \frac{2 * (Precision * Recall)}{(Precision + Recall)}$$

- Specificity: Out of all the people that do not have the disease, how many got negative results.

$$\boldsymbol{specificity} = \frac{tn}{tn + fp}$$

As our Data is Imbalanced, we will focus mostly on these metrics rather than accuracy.

Modelling:

Logistic Regression

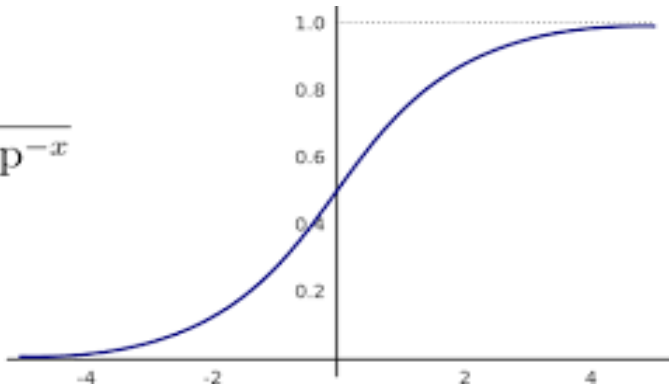
Logistic regression is a fundamental classification technique. It belongs to the group of linear classifiers and is somewhat similar to polynomial and linear regression. Logistic regression is fast and relatively uncomplicated, and it's convenient for you to interpret the results. Although it's essentially a method for binary classification, it can also be applied to multiclass problems.

$$\ln\left(\frac{\hat{p}}{(1-\hat{p})}\right) = b_0 + b_1X_1 + b_2X_2 + \ldots + b_pX_p$$

$$\hat{p} = \frac{\exp(b_0 + b_1X_1 + b_2X_2 + \ldots + b_pX_p)}{1 + \exp(b_0 + b_1X_1 + b_2X_2 + \ldots + b_pX_p)}$$

Were, b_0 is intercept of the regression line and b_i's are slopes of regression line.

$$\sigma(x) = \frac{1}{1 + \exp^{-x}}$$



- Wald Statistics: It is used to test the significance of individual coefficients in the model.

H0: $b_i = 0$; i = 1, 2...,18 v/s H1: $b_i \neq 0$; for at least one i.

$$W = b_i / SE(b_i) \sim N(0,1)$$

Decision criterion: Reject H0 if p-value < 0.05

Logit Regression Results

Dep. Variable:	loan_status	No. Observations:	752784
Model:	Logit	Df Residuals:	752767
Method:	MLE	Df Model:	16
Date:	Sun, 15 May 2022	Pseudo R-squ.:	-0.05201
Time:	09:55:33	Log-Likelihood:	-4.9005e+05
converged:	True	LL-Null:	-4.6583e+05
Covariance Type:	nonrobust	LLR p-value:	1.000

	coef	std err	z	P>|z|	[0.025	0.975]
x1	0.1614	0.003	50.715	0.000	0.155	0.168
x2	-0.0587	0.002	-23.693	0.000	-0.064	-0.054
x3	-0.0650	0.003	-20.439	0.000	-0.071	-0.059
x4	0.2697	0.004	73.182	0.000	0.262	0.277
x5	0.0168	0.003	6.630	0.000	0.012	0.022
x6	-0.0423	0.003	-14.376	0.000	-0.048	-0.037
x7	-0.0753	0.003	-24.859	0.000	-0.081	-0.069
x8	-0.0652	0.003	-22.615	0.000	-0.071	-0.060
x9	-0.1072	0.003	-35.077	0.000	-0.113	-0.101
x10	-0.3013	0.003	-100.371	0.000	-0.307	-0.295
x11	0.3464	0.003	128.074	0.000	0.341	0.352
x12	0.1039	0.003	36.665	0.000	0.098	0.110
x13	0.0668	0.003	26.421	0.000	0.062	0.072
x14	0.0957	0.002	38.768	0.000	0.091	0.101
x15	0.0024	0.002	0.989	0.323	-0.002	0.007
x16	0.0686	0.002	28.029	0.000	0.064	0.073
x17	0.0219	0.003	8.467	0.000	0.017	0.027

From the above table, as the p-value > 0.05, for the variable x15 (i.e., add_state). So, the variable x15 is insignificant, and will not provide much information from our further studies.

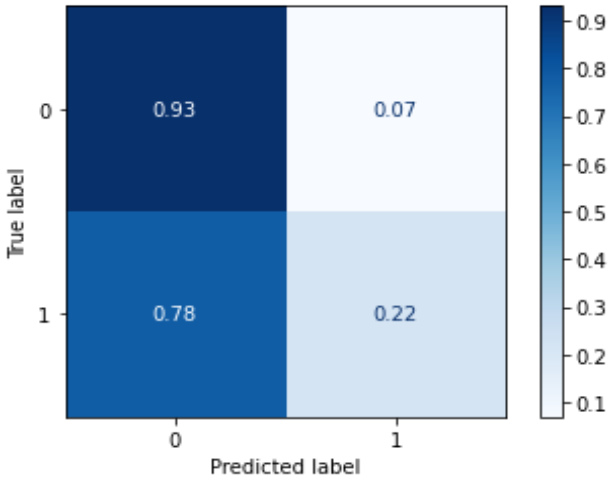
Regression Equation:

p = P(defaulter) = 1 / 1 + exp [- (-0.89379523 + 0.1614(loan_amnt) - 0.0587(emp_length) - 0.0650(annual_inc) + 0.2697(dti) + 0.0168(pub_rec) - 0.0423(revol_bal) - 0.0753(revol_util) - 0.0652(total_acc) - 0.1072(mort_acc) - 0.3013(fico) + 0.3464(term) + 0.1039(home_ownership) + 0.0668(verificaction_status) + 0.0957(purpose) + 0.0686(initial_list_status) + 0.0219(application_type))]

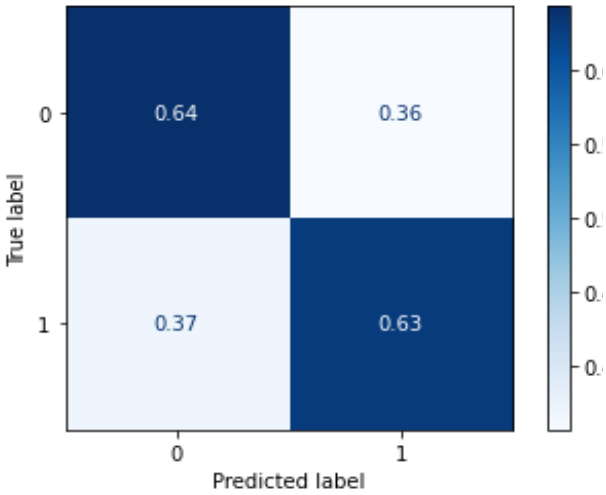
Model Evaluation:

Evaluation_metrics -->	Training_accuracy	Testing_accuracy	Precision	Specificity	Recall(Sensitivity)	f1-Score	ROC-AUC_Score
logistic	0.710217	0.710495	0.586634	0.930444	0.220089	0.320089	0.691379
logistic(Hyperparameter tunned)	0.639652	0.638469	0.441278	0.642371	0.629771	0.518938	0.691628
logistic(Oversampling)	0.638185	0.638123	0.630223	0.701104	0.567911	0.597446	0.690962

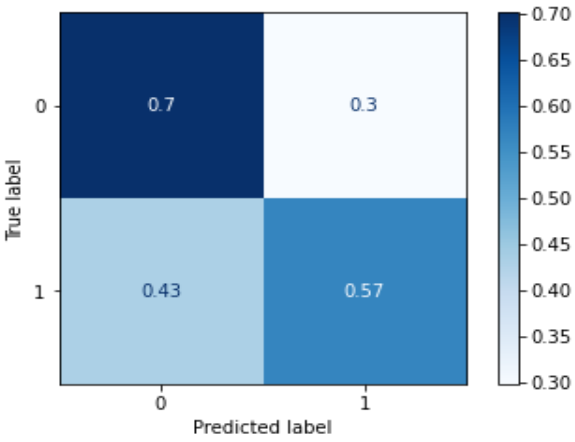
Model1: On fitting the logistic regression on the dataset, it was observed that the model does a good job in predicting the Fully paid loans with **specificity** of almost 93%. But it equally does a bad job in predicting the true positives i.e., the defaulted loans, with **sensitivity** of 22%. **F1-Score** is 32%. This model is a bad classifier as lot of defaulted loans are classified as Fully paid loans.

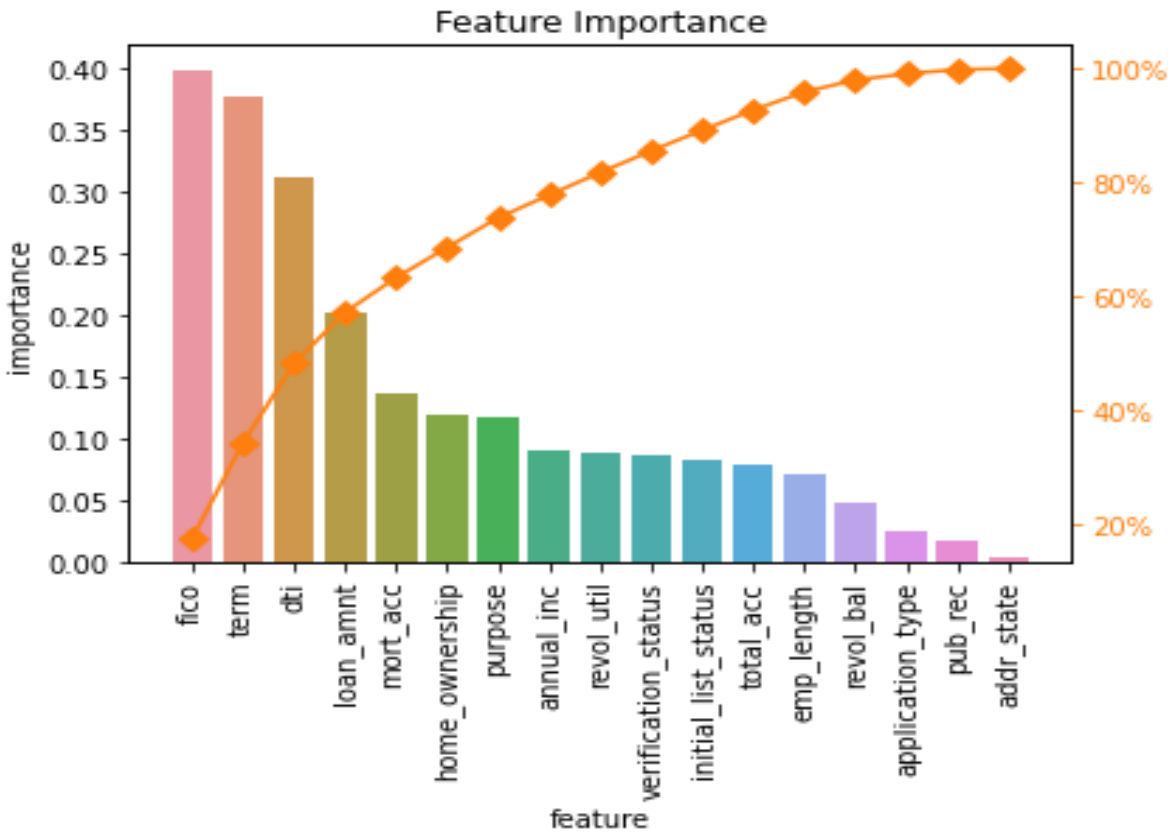


Model2: Later, we tuned the hyper parameters (i.e.,Class_weight = balanced) and on fitting it to the dataset, it was observed that the model was predicting the Fully paid loans with **specificity** of 64.24% and defaulted loan with the **sensitivity** of almost 63%. **F1-Score** is 52%. This model seems to be good as it is classifying most the defaulted loans properly.



Model3: Later, we did over-sampling, and we fitted the Model1 to the dataset, it was observed that it performed better than the Model1 with **specificity** of 70%, **sensitivity** of 57% and **F1- Score** of 60%, but did not perform better than Model2**.**



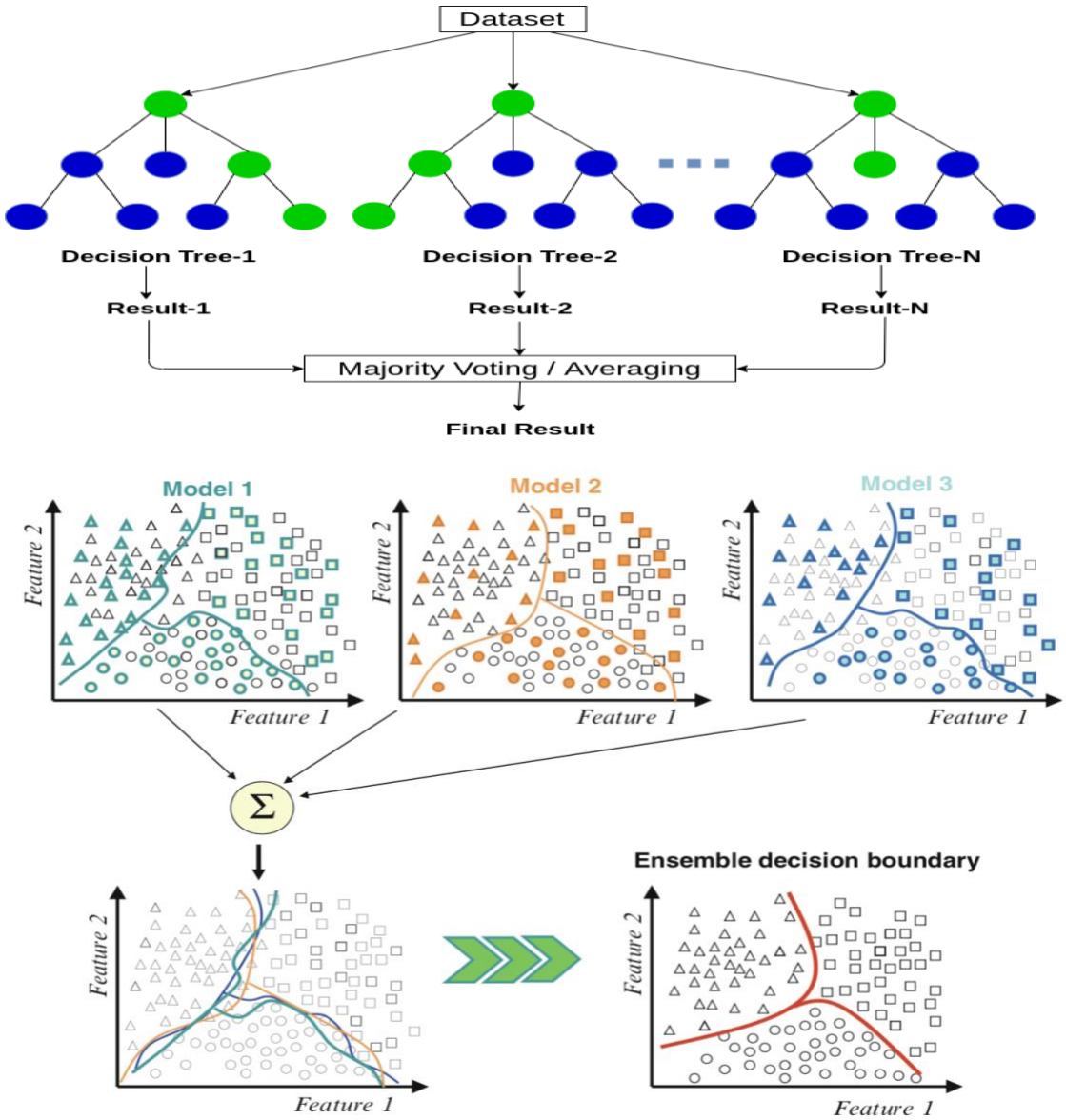


We can see that the important variables are fico, term, dti, loan_amnt, mort_acc, Home_ownership, purpose, annual_inc forms 80% of the important reasons for a customer to default a loan.

Random Forest

Random Forest is bootstrap aggregation method in which, firstly the observations and features for each base learner is chosen through bootstrap sampling method. The base learners are decision trees.

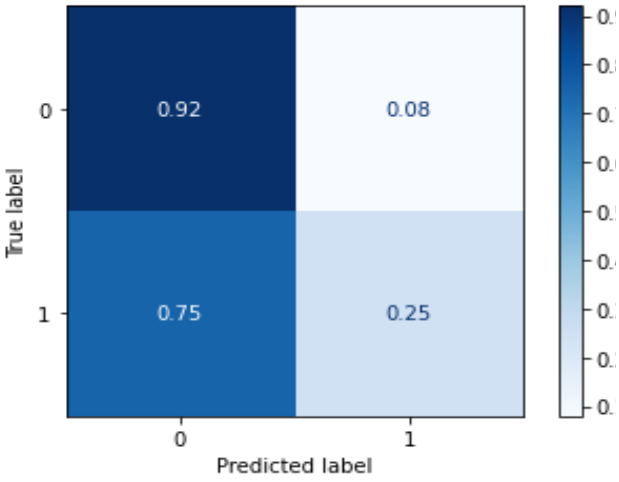
For classification task, each base learner will provide an output (if binary, then the output can be either 0 or 1), and majority voting / aggregation will be done and the final output for random forest will be selected.



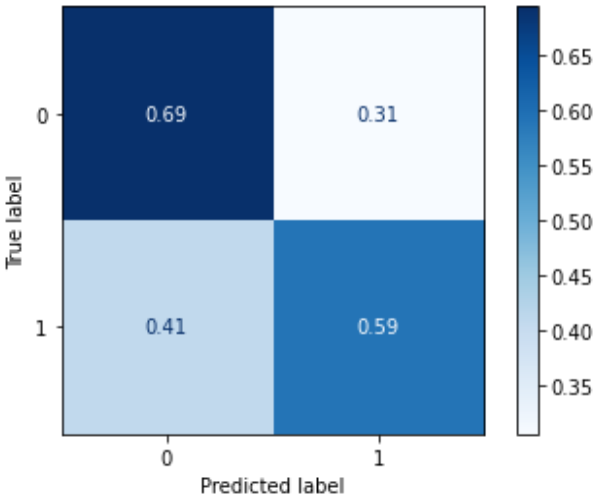
Model evaluation:

Evaluation_metrics -->	Training_accuracy	Testing_accuracy	Precision	Specificity	Recall(Sensitivity)	f1-Score	ROC-AUC_Score
RandomForest	0.999992	0.712121	0.582328	0.920069	0.248473	0.348321	0.695187
RandomForest(Hyperparameter tunned)	0.731541	0.662582	0.464759	0.69437	0.591708	0.520606	0.703043
RandomForest(Oversampling)	1.0	0.850143	0.815629	0.821043	0.882584	0.847787	0.926691

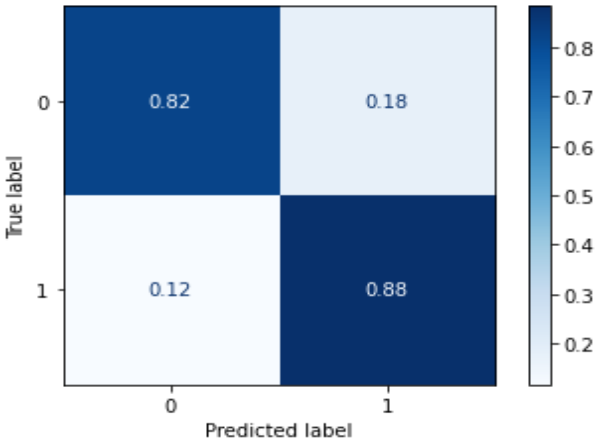
Model1: like logistic regression, on fitting the random forest on the dataset, it was observed that the model does a good job in predicting the Fully paid loans with **specificity** of 92%. But it equally does a bad job in predicting the true positives i.e., the defaulted loans, with **sensitivity** of 25%. **F1-Score** is 35%. This model is a bad classifier as lot of defaulted loans are classified as Fully paid loans.

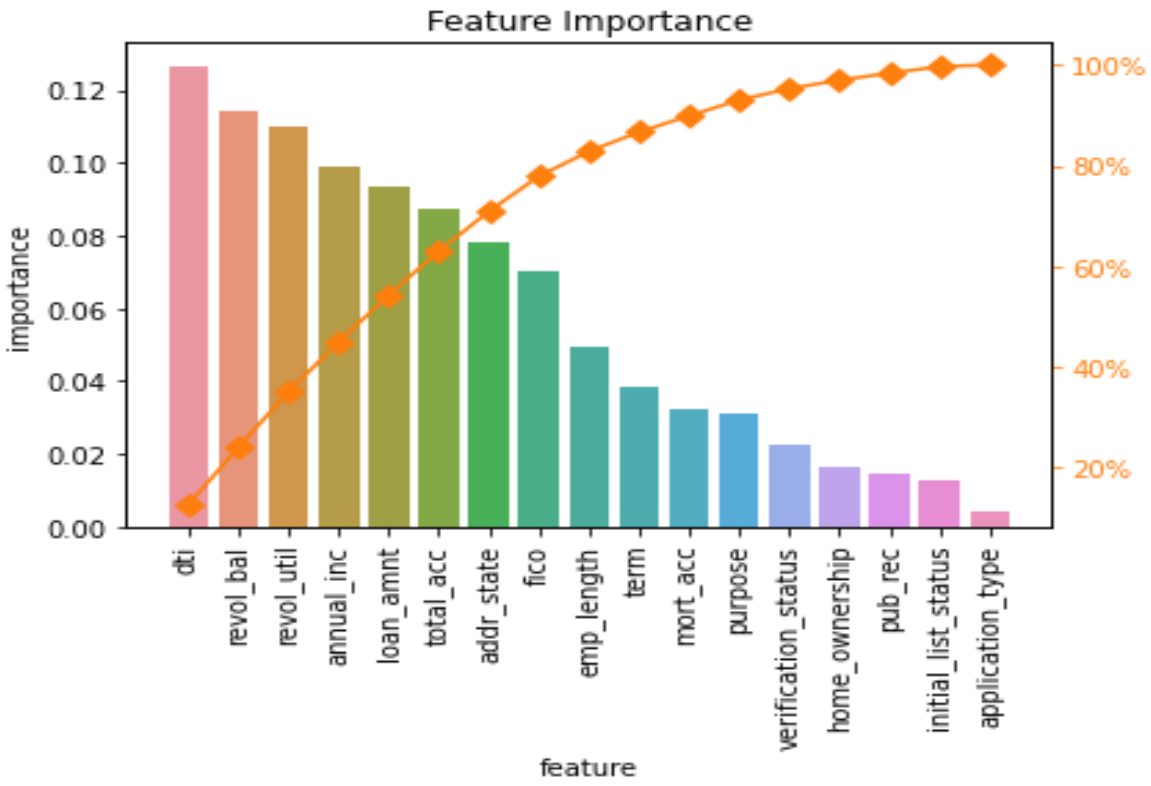


Model2: we did hyper parameter tuning (i.e.,Class_weight = balanced) and on fitted it to the dataset, it was observed that the model was predicting the Fully paid loans with **specificity** of 69% and defaulted loan with the **sensitivity** of almost 59%. **F1-Score** is 52%. This model seems to be good as it is classifying most the defaulted loans properly.



Model 3: Next, we did over-sampling, and we fitted the Model1 to the dataset, it was observed that it performed better than the Model1 with **specificity** of 82%, **sensitivity** of 88% and **F1- Score** of 84%, this model gives the best results out of these 3 models.





We can see that the important variables are dti, revol_bal, revol_util, annual income, loan amount, total_acc, fico, emp_length forms 80% of the important reasons for a customer to default a loan.

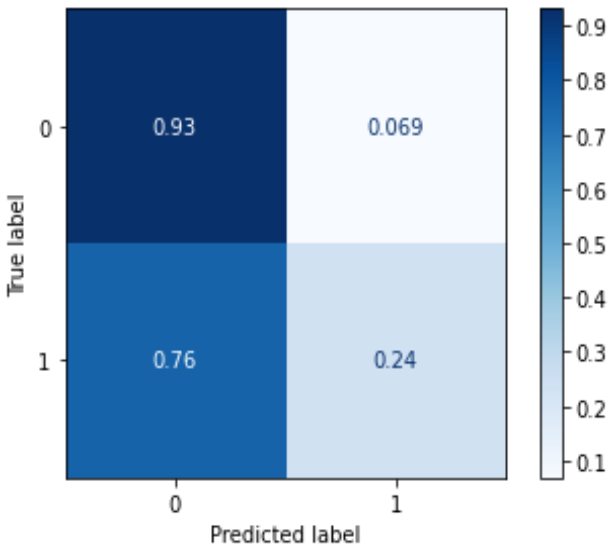
eXtreme Gradient Boosting

XGBoost is an optimized distributed gradient boosting library designed to be highly efficient, flexible and portable. It implements machine learning algorithms under the gradient Boosting framework. XGBoost provides a parallel tree boosting (also known as GBDT, GBM) that solve many data science problems in a fast and accurate way.

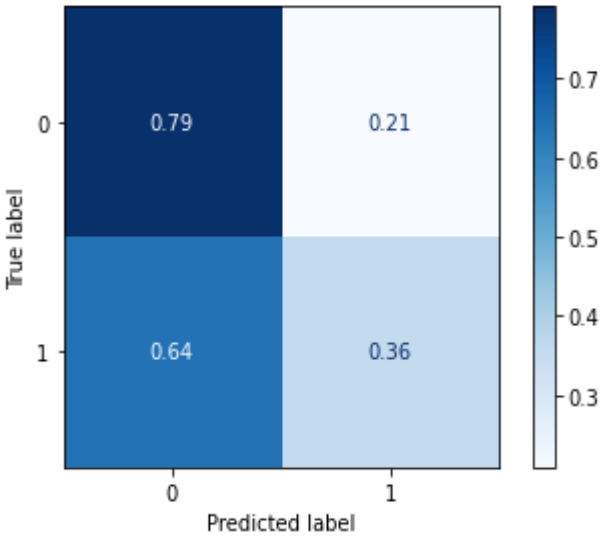
Model evaluation:

Evaluation_metrics -->	Training_accuracy	Testing_accuracy	Precision	Specificity	Recall(Sensitivity)	f1-Score	ROC-AUC_Score
XGB	0.717396	0.716637	0.600733	0.924587	0.252986	0.356035	0.709295
XGB(Hyperparameter tunned)	0.998692	0.656775	0.433986	0.791395	0.356621	0.391518	0.626418
XGB(Oversampling)	0.652483	0.650827	0.641564	0.702975	0.592691	0.61616	0.70903

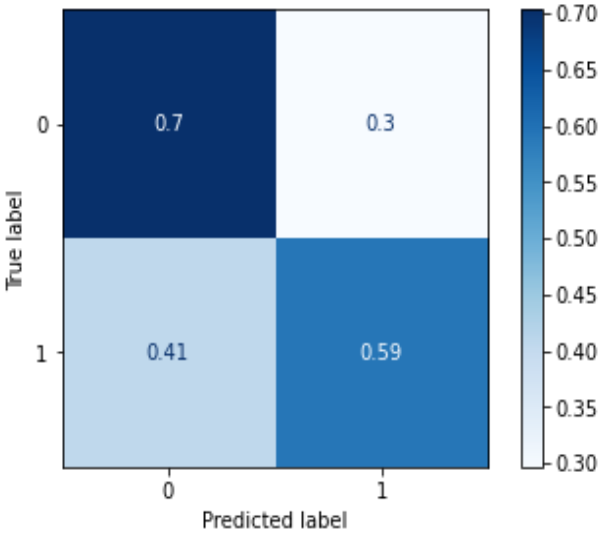
Model 1: we fitted XGBoost on the dataset, it was observed that the model does a good job in predicting the Fully paid loans with **specificity** of 93%. But it equally does a bad job in predicting the true positives i.e., the defaulted loans, with **sensitivity** of 24%. **F1-Score** is 35%. This model is a bad classifier as lot of defaulted loans are classified as Fully paid loans.

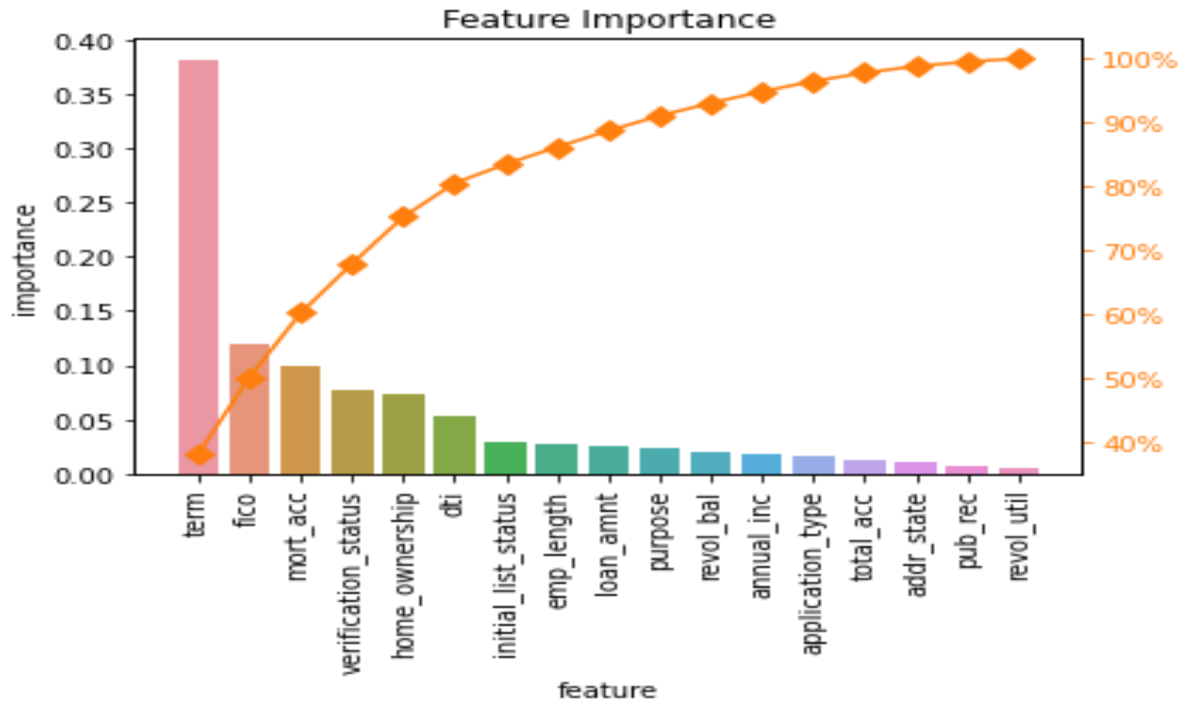


Model 2: we did hyper parameter tuning (i.e.,Class_weight= balanced) and on fitted it to the dataset, it was observed that the model was predicting the Fully paid loans with **specificity** of 79% and defaulted loan with the **sensitivity** of almost 36%. **F1-Score** is 39%. This model seems to be performing equally bad.



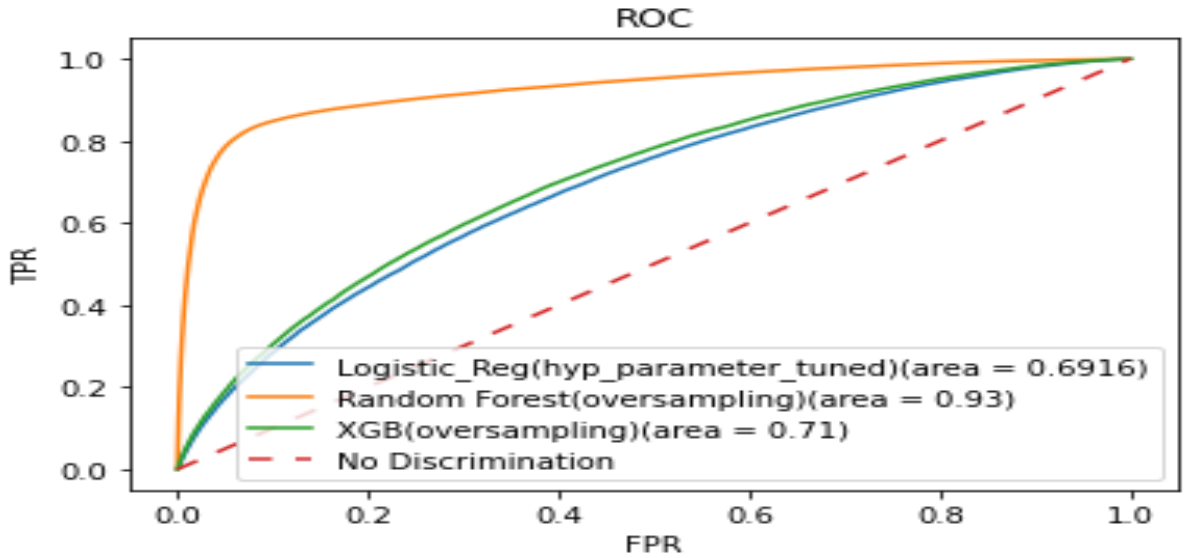
Model 3: Next, we did over-sampling, and we fitted the Model1 to the dataset, it was observed that it performed better than the Model 1 with **specificity** of 70%, **sensitivity** of 59% and **F1- Score** of 61%, this model performs slightly better than the above two models but it is not satisfactory.





We can see that the important variables are term, fico, mort_acc, verification status, home_ownership, dti forms 80% of the important reasons for a customer to default a loan.

ROC-AUC Curve



For each of the machine learning models, auc_score is plotted and Random Forest with oversampling is selected as the final model since it has the highest ROC-AUC Score.

Summary:

1. As the loan amount goes beyond $10000, there is a slight tendency of a customer to default a loan.
2. Customers having term of 60 months tends to default more than the customers with term of 36 months.
3. The tendency of a customer to default a loan increase as the interest rate goes beyond 25%.
4. Customers having less employment length are more likely to default a loan.
5. Customers having more the mortgage account less are the chance of them defaulting a loan and vice-versa.
6. Customers having high fico score, the less is the chance of them defaulting the loan and vice-versa.
7. Customers living on rent tend to apply for a loan more than those owning a house.
8. The top three purposes of most of the customers, to issue a loan is for debt consolidation, credit card and home improvement.
9. Customers with low dti are likely to manage their monthly debt payment effectively, so there is a less chance of them defaulting a loan.

❖ **Factors significantly affecting a customer to default a loan are:**

- Dti (debt-income ratio)
- Revol_bal
- Annual Income
- Loan amount
- Fico score
- Employment length
- Total_acc
- Term
- Mort_acc
- Purpose
- Home_ownership

Conclusion:

The application of machine learning techniques in the financial sector with the goal of profit maximization has seen a rising interest over the last few years. There has been increasing number of research conducted in the areas of credit scoring, risk management and bankruptcy prediction using machine learning approaches.

This research proposes machine learning as a method to improve the accuracy of loan default predictions. This better understanding of customer behaviours to improve the prediction of loan default will contribute to tremendous financial benefit in the lending sector. This research successfully explores the features of loan data that contribute to the risk of loan defaults. Exploratory data analysis shows the correlation of various features with loan default to select the most appropriate features to train the machine learning model. The train and test data set are then applied to three machine learning algorithms to determine the one with the most accurate results. Key performance metrics which include confusion matrix, accuracy, precision and recall and applied to evaluate the best machine learning technique in loan default prediction.

Limitations of the research:

Machine learning algorithms are limited to the dataset used to train and test the model. Data is governed by data protection laws making it challenging to access primary data for the purpose of research. This research was conducted using secondary open data from Kaggle.com that is available to the public. This limits the generalization of the model as it is specific towards the dataset used to train and test the model. It would be beneficial to look comprehensively at the main features that are relevant to the characteristic that drive default and can be applied. Further limitation is in reference to the variables provided in the dataset, although the dataset is open due to data protection laws some factors may not be available to the public and these may have had an impact on the predictions of the probability of default. Lastly, the research focused on the probability of default in a default state however default loans may still be recovered during the recollection process.

REFERENCES

- **Exploratory data analysis:**

1. Hands-On Exploratory Data Analysis with Python: Perform EDA techniques to understand, summarize, and investigate your data
2. Krish Naik YouTube videos on EDA and Feature Engineering. https://youtu.be/F-X82zhIfBo

- Harry Khamis (2008). Measures of Association - How to Choose? https://journals.sagepub.com/doi/pdf/10.1177/8756479308317006

- Credit scoring approaches guidelines - World Bank. (n.d.). http://pubdocs.worldbank.org/en/935891585869698451/CREDIT-SCORING- APPROACHES-GUIDELINES-FINAL-WEB.pdf

- Bhandari, M. (2020, October 19). Predict Loan Eligibility using Machine Learning Models. https://towardsdatascience.com/predict-loan-eligibility-using-machine-learning-models-7a14ef904057

- Xu, Z. (. (2021, March 10). Loan default prediction with Berka dataset. Medium. https://towardsdatascience.com/loan-default-prediction-an-end-to-end-ml-project-with- real-bank-data-part-1-1405f7aecb9e

- Aleksandrova, Y. (2021). Comparing Performance of Machine Learning Algorithms for Default Risk Prediction in Peer-to-Peer Lending. https://doi.org/10.18421/tem101-16

- Understanding Confusion Matrix .Sarang Narkhede. Medium

- **Modelling:**

1. Hands-On Machine Learning with Scikit-Learn and TensorFlow: Concepts, Tools, and Techniques to Build Intelligent Systems.
2. Krish Naik's Machine Learning Playlist. https://youtu.be/4UJelID_ICw

APPENDIX

Importing Libraries

```
import os
import math
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from matplotlib.ticker import PercentFormatter
import plotly.express as px
import seaborn as sns
from scipy import stats
from scipy.stats import chi2_contingency
from scipy.stats import chi2
import researchpy as rp
from statsmodels.stats.outliers_influence import variance_inflation_factor
from sklearn.feature_selection import VarianceThreshold
from sklearn.preprocessing import StandardScaler
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics import classification_report, roc_curve, precision_recall_curve, roc_auc_score,f1_score,
confusion_matrix , plot_confusion_matrix , accuracy_score , precision_score , recall_score
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.model_selection import StratifiedShuffleSplit
import xgboost as xgb
import warnings
warnings.filterwarnings('ignore')
%matplotlib inline

pd.options.mode.chained_assignment = None
pd.options.display.max_rows = 200
pd.options.display.max_columns = 150
```

Retrieving Dataset

```
data=pd.read_csv('/content/drive/MyDrive/Loan_data',low_memory=False)
```

Data Pre-processing and EDA

```
data = data.drop(['Unnamed: 0','member_id'],axis = 1)
data.loan_status.value_counts()
Loan_Main = data.copy()
```

```
Loan_Main['loan_status'] = np.where(Loan_Main['loan_status'].str.contains('Fully'),0,1)
Loan_Main.loan_status.value_counts()

def emp_to_num(term):
if pd.isna(term):
return None
elif term[2]=='+':
return 10
elif term[0]=='<':
return 0
else:
return int(term[0])
Loan_Main['emp_length'] = Loan_Main['emp_length'].apply(emp_to_num)

nunique_val = Loan_Main.apply(pd.Series.nunique)
single_val_cols = nunique_val[nunique_val == 1].index.tolist()
single_val_cols

Loan_Main= Loan_Main.drop(single_val_cols, axis =1)

a=Loan_Main.isnull().sum()/len(Loan_Main)*100
variables = Loan_Main.columns
len(variables)

variable = [ ]
for i in range(0,len(variables)):
if a[i]<50:
variable.append(variables[i])
drop_columns1=list(set(variables) - set(variable))
df_drop_nan=Loan_Main.drop(columns=drop_columns1)

((df_drop_nan.isnull().sum()/len(df_drop_nan))*100).sort_values(ascending=False)

final_features = ['loan_amnt', 'term', 'int_rate', 'installment', 'grade', 'sub_grade',
'emp_title', 'emp_length', 'home_ownership', 'annual_inc',
'verification_status', 'issue_d', 'loan_status', 'purpose', 'title',"addr_state",
'dti', 'earliest_cr_line', 'open_acc', 'pub_rec', 'revol_bal',
'revol_util', 'total_acc', 'initial_list_status', 'application_type',
'mort_acc', 'pub_rec_bankruptcies','fico_range_low','fico_range_high']

df = df_drop_nan[final_features]

plt.figure(figsize=(6,3),dpi=120)
df.corr()['loan_status'].sort_values().drop('loan_status').plot(kind='bar', cmap='viridis') # correlation with
loan_status for continuous features with loan_status feature dropped
```

```
plt.xticks(rotation=90);

def create_countplot(axes, x_val,order_val, title, rotation="n"):
sns.countplot(ax= axes, data=df, x=x_val, order = order_val.value_counts(dropna= False).index,hue =
"loan_status")
axes.set_title(title)
if rotation =="y":
axes.set_xticklabels(list(order_val.unique()), rotation=90)

fig, ax = plt.subplots(2,3, figsize= (20,10))
create_countplot(ax[0,0],'term', df["term"],"The number of payments on the loan (months)" )
create_countplot(ax[0,1],'grade', df["grade"],"Loan grade")
create_countplot(ax[0,2],'sub_grade', df["sub_grade"],"Loan sub_grade","y")
create_countplot(ax[1,0],'emp_length', df["emp_length"],"Borrower length of employment (years)", "y" )
create_countplot(ax[1,1],'home_ownership', df["home_ownership"],"Borrower home ownership status" )
create_countplot(ax[1,2],'verification_status', df["verification_status"],"verification_status" )
plt.tight_layout()
plt.show()

fig, ax = plt.subplots(1,3, figsize= (20,6))
create_countplot(ax[0],'purpose', df["purpose"],"Purpose of loan" ,"y")
create_countplot(ax[1],'initial_list_status', df["initial_list_status"],"Initial listing status of the loan" )
create_countplot(ax[2],'application_type', df["application_type"],"Application type" )
plt.tight_layout()
plt.show()

plt.plot(df.groupby('grade')['loan_status'].mean())
plt.xlabel("Grade")
plt.ylabel("Defaulters rate")

plt.figure(figsize=(12,5), dpi=130)
sns.distplot(x=df['annual_inc'], bins=100)

percent_of_rows_of_customers_with_annual_inc_more_than_250000 = len(df[df['annual_inc'] >
250000])*100/len(df)
percent_of_rows_of_customers_with_annual_inc_more_than_250000
df = df[df['annual_inc'] <= 250000]
plt.figure(figsize=(12,5), dpi=130)
sns.distplot(x=df['annual_inc']);
sns.displot(data=df, x='annual_inc', hue='loan_status', bins=80, height=5, aspect=3, kde=True,
palette='viridis');
df.groupby('loan_status')['annual_inc'].describe()

plt.figure(figsize=(24,4))
plt.xticks(rotation=90)
```

```
sns.countplot(data=df, x='total_acc', palette='viridis');
df = df[df['total_acc'] < 64]
plt.figure(figsize=(24,4))
plt.xticks(rotation=90)
sns.countplot(data=df, x='total_acc', palette='viridis');
plt.plot(df.groupby('total_acc')['loan_status'].mean())
plt.xlabel("Total_acc")
plt.ylabel("Defaulters rate")
plt.figure(figsize=(24,4))
plt.xticks(rotation=90)
sns.countplot(data=df, x='total_acc', hue='loan_status', palette='viridis');

plt.figure(figsize=(24,4))
plt.xticks(rotation=90)
sns.countplot(data=df, x='mort_acc', palette='viridis');
df = df[df['mort_acc'] < 11]
plt.figure(figsize=(24,4))
plt.xticks(rotation=90)
sns.countplot(data=df, x='mort_acc', palette='viridis');
plt.plot(df.groupby('mort_acc')['loan_status'].mean())
plt.xlabel("Mort_acc")
plt.ylabel("Defaulters rate")

df = df.drop(['title','emp_title'],axis =1)
df['earliest_cr_line'] = df['earliest_cr_line'].apply(lambda date: int(date[-4:]))
sns.displot(data=df, x='earliest_cr_line', hue='loan_status', bins=100, height=4, aspect=3, kde=True,
palette='viridis');
plt.plot(df.groupby('earliest_cr_line')['loan_status'].mean())
plt.xlabel("earliest_cr_line")
plt.ylabel("Defaulters rate")
df.groupby('loan_status')['earliest_cr_line'].describe()
df['revol_util'] = np.where(df['revol_util'].isnull(),df['revol_util'].median(),df['revol_util'])


df['emp_length'] = np.where(df['emp_length'].isna(),0.0,df['emp_length'])
fig = px.bar(df['emp_length'].value_counts().sort_index()/len(df))
fig.show()

fig = px.line(df.groupby('emp_length')['loan_status'].mean())
fig.show()

df['fico'] = (df['fico_range_high'] + df['fico_range_low'])/2
df = df.drop(['fico_range_high','fico_range_low'],axis = 1)
plt.figure(figsize=(24,4))
plt.xticks(rotation=90)
```

```
sns.countplot(data=df, x='fico', palette='viridis');
plt.plot(df.groupby('fico')['loan_status'].mean())
plt.xlabel("Fico")
plt.ylabel("Defaulters Rate")

plt.plot(df.groupby('grade')['fico'].mean())
plt.xlabel("Grade")
plt.ylabel("Fico")

df = df.dropna().reset_index()

df_copy = df.copy()
df = df.drop(['index'],axis =1)
target_col = pd.DataFrame(df['loan_status'])
df = df.drop(['loan_status'],axis =1)
num_col = df.select_dtypes(exclude = 'object')
num_col = num_col.drop(['pub_rec_bankruptcies','open_acc','installment'],axis =1)
plt.figure(figsize= (15,7))
sns.heatmap(num_col.corr(), vmin=1, vmax=-1, annot=True, cmap="Spectral")
```

VIF

```
vif_data2 = pd.DataFrame()
vif_data2["feature"] = num_col.columns
vif_data2["VIF"] = [variance_inflation_factor(num_col.values, i)
for i in range(len(num_col.columns))]
vif_data2

num_col = num_col.drop(['int_rate','earliest_cr_line'],axis =1)
cat_col = df.select_dtypes(include = 'object')
cat_col = cat_col.drop(['sub_grade','issue_d'],axis =1)
```

CHISQUARE TEST FOR INDEPENDENCE

```
rp.summary_cat(cat_col)
crosstab, test_results, expected = rp.crosstab(cat_col ['term'], data['grade'],
                                   test= "chi-square",
                                   expected_freqs= True,
                                   prop= "cell")
test_results
# Similarly check for each categorical variable

cat_col = cat_col.drop(['grade',],axis =1)
enc = LabelEncoder()
Cat_col = cat_col.apply(LabelEncoder().fit_transform)
feature_col = pd.concat([num_col,Cat_col],axis =1)
```

Train-Test-Split

```
sss = StratifiedShuffleSplit(n_splits=1 , test_size = 0.2 , random_state=42)
for train_index, test_index in sss.split(feature_col, target_col):
print("TRAIN:", train_index, "TEST:", test_index)
X_train, X_test = feature_col.iloc[train_index], feature_col.iloc[test_index]
y_train, y_test = target_col.iloc[train_index], target_col.iloc[test_index]

ss = StandardScaler()
X_train = ss.fit_transform(X_train)
X_test = ss.fit_transform(X_test)
```

Creating functions for modelling and Evaluation

```
def fit_score_clf(clf,X_train,y_train,X_test,y_test, name = 'clf'):
clf.fit(X_train,y_train)
y_pred = clf.predict(X_test)
clf_report = pd.DataFrame(classification_report(y_test, y_pred, output_dict=True))
clf_report['clf'] = name
return clf_report

def Confusion_matrix(clf):
y_pred = clf.predict(X_test)
plot_confusion_matrix(clf, X_test , y_test , normalize='true', cmap = 'Blues')


def rocscore(clf,X_train,y_train,X_test,y_test, name = 'clf'):
clf.fit(X_train, y_train)
ytrain_pred = clf.predict_proba(X_train)
print('Training roc-auc: {}'.format(roc_auc_score(y_train, ytrain_pred[:,1])))
ytest_pred = clf.predict_proba(X_test)
print('Test roc-auc: {}'.format(roc_auc_score(y_test, ytest_pred[:,1])))


roc_dict = {} # ROC dict for FPR and TPR (FPR = False Positive Rate & TPR = True Positive Rate)
def modify_roc_dict(clf, y_test, X_test, name ,dict_modify): # clf = Classifier
fpr, tpr, thresholds = roc_curve(y_test,
clf.predict_proba(X_test)[:,1],
pos_label=1)
ROC_AUC_SCORE = roc_auc_score(y_test,clf.predict_proba(X_test)[:,1])
clf_roc_dict = {'fpr':fpr, 'tpr': tpr, 'thresholds': thresholds ,'ROC_AUC_SCORE': ROC_AUC_SCORE}
dict_modify[name] = clf_roc_dict
```

LOGISTIC REGRESSION

```
lr = LogisticRegression(penalty='l1',
```

```
C = 1.0,
solver = 'liblinear',
max_iter = 1000,
n_jobs = 2)


lr_report = fit_score_clf(lr,X_train,y_train,X_test,y_test, name = 'lr')
lr_report Confusion_matrix(lr)
import statsmodels.api as sm
logit_model=sm.Logit(y_train,X_train)
result=logit_model.fit()
print(result.summary())
params = result.params
conf = result.conf_int()
conf['OR'] = params
conf.columns = ["Lower CI", "Upper CI", "OR"]
np.exp(conf)

A = pd.DataFrame(lr.coef_).T
importance_df = pd.DataFrame({
'feature': feature_col.columns,
'importance': abs(A[0])
}).sort_values('importance', ascending=False)

plt.title('Feature Importance')
sns.barplot(data=importance_df.head(25), x='importance', y='feature');
```

LOGISTIC REGRESSION HYPERPARAMETER TUNNED

```
lr1 = LogisticRegression(penalty='l1',
C = 1.0,
solver = 'liblinear',
max_iter = 1000,
n_jobs = 2,
class_weight = 'balanced')

lr1_report = fit_score_clf(lr1,X_train,y_train,X_test,y_test, name = 'lr1')
lr1_report
Confusion_matrix(lr1)
A = pd.DataFrame(lr1.coef_).T
importance_df = pd.DataFrame({
'feature': feature_col.columns,
'importance': abs(A[0])
}).sort_values('importance', ascending=False)
```

```
importance_df['cumpercentage'] =
importance_df['importance'].cumsum()/importance_df['importance'].sum()*100 fig, ax1 = plt.subplots()
plt.title('Feature Importance')
sns.barplot(data=importance_df.head(25), y='importance', x='feature');
ax1.set_xticklabels(importance_df['feature'], rotation=90)
ax2 = ax1.twinx()
ax2.plot(importance_df['feature'], importance_df['cumpercentage'], color="C1", marker="D", ms=7)
ax2.yaxis.set_major_formatter(PercentFormatter())
ax2.tick_params(axis="y", colors="C1")
plt.show()

modify_roc_dict(clf = lr1 ,
y_test = y_test,
X_test = X_test,
name = 'Logistic_Reg(hyp_parameter_tuned)(area = 0.6916)',
dict_modify = roc_dict)
```

RANDOM FOREST

```
RF = RandomForestClassifier(n_estimators=100,
criterion= 'gini',
n_jobs=-1,
random_state = 42)
lin_RF_report = fit_score_clf(RF,X_train,y_train,X_test,y_test, name = 'Random_Forest')
lin_RF_report
Confusion_matrix(RF)
```

RANDOM FOREST WITH HYPERPARAMETER TUNNED

```
RF1 = RandomForestClassifier(n_estimators=100,
criterion= 'gini',
n_jobs=-1,
random_state = 42,
max_depth = 15,
class_weight = 'balanced')
lin_RF1_report = fit_score_clf(RF1,X_train,y_train,X_test,y_test, name = 'Random_Forest')
lin_RF1_report
Confusion_matrix(RF1)
modify_roc_dict(clf = RF1 ,
y_test = y_test,
X_test = X_test,
name = 'Random Forest',
dict_modify = roc_dict)
```

XGB

```
xgb = xgb.XGBClassifier(n_estimators=300,
learning_rate = 0.1,
```

```
subsample=0.9,
n_jobs=-1)


xgb_report = fit_score_clf(xgb, X_train, y_train, X_test,y_test, name='xgb')
xgb_report
Confusion_matrix(xgb)
```

XGB WITH HYPERPARAMETER TUNNED

```
xgb1 = xgb.XGBClassifier(n_estimators=300,
learning_rate = 1,
subsample=0.9,
n_jobs=-1,
max_depth = 12)


xgb1_report = fit_score_clf(xgb1, X_train, y_train, X_test,y_test, name='xgb')
xgb1_report
importance_df = pd.DataFrame({
'feature': feature_col.columns,
'importance': xgb1.feature_importances_
}).sort_values('importance', ascending=False)
plt.title('Feature Importance')
sns.barplot(data=importance_df.head(21), x='importance', y='feature');
y_pred = xgb1.predict(X_test)
plot_confusion_matrix(xgb1, X_test , y_test , normalize='true', cmap = 'Blues')

modify_roc_dict(clf = xgb1 ,
y_test = y_test,
X_test = X_test,
name = 'XGB',
dict_modify = roc_dict)
```

WITH OVERSAMPLING

```
df1 = df_copy[df_copy['loan_status'] == 1]
DF = pd.concat([df_copy,df1],axis = 0)
DF = DF.dropna().reset_index()
Target_col = pd.DataFrame(DF['loan_status'])
DF = DF.drop(['index','loan_status','level_0'],axis =1)
Num_col = DF.select_dtypes(exclude = 'object')
Num_col = Num_col.drop(['pub_rec_bankruptcies','open_acc','installment','int_rate','earliest_cr_line'],axis =1)
CAT_col = DF.select_dtypes(include = 'object')
CAT_col = CAT_col.drop(['sub_grade','issue_d','grade'],axis =1)
enc = LabelEncoder()
CAT_COL = CAT_col.apply(LabelEncoder().fit_transform)
```

```
Feature_col = pd.concat([Num_col,CAT_COL],axis = 1)
sss = StratifiedShuffleSplit(n_splits=1 , test_size = 0.2 , random_state=42)
for train_index, test_index in sss.split(Feature_col, Target_col):
print("TRAIN:", train_index, "TEST:", test_index)
X_train1, X_test1 = Feature_col.iloc[train_index], Feature_col.iloc[test_index]
y_train1, y_test1 = Target_col.iloc[train_index], Target_col.iloc[test_index]

ss = StandardScaler()
X_train1 = ss.fit_transform(X_train1)
X_test1 = ss.fit_transform(X_test1)

def Confusion_matrix(clf):
y_pred = clf.predict(X_test1)
plot_confusion_matrix(clf, X_test1 , y_test1 , normalize='true', cmap = 'Blues')

roc_dict1 = {} # ROC dict for FPR and TPR (FPR = False Positive Rate & TPR = True Positive Rate)

def modify_roc_dict(clf, y_test, X_test, name ,dict_modify): # clf = Classifier
fpr, tpr, thresholds = roc_curve(y_test1,
clf.predict_proba(X_test1)[:,1],
pos_label=1)
ROC_AUC_SCORE = roc_auc_score(y_test,clf.predict_proba(X_test1)[:,1])
clf_roc_dict = {'fpr':fpr, 'tpr': tpr, 'thresholds': thresholds ,'ROC_AUC_SCORE': ROC_AUC_SCORE}
dict_modify[name] = clf_roc_dict
```

LOGISTIC REGRESSION (OVERSAMPLING)

```
lr2 = LogisticRegression(penalty='l1',
C = 1.0,
solver = 'liblinear',
max_iter = 1000,
n_jobs = 2)


lr2_report = fit_score_clf(lr2,X_train1,y_train1,X_test1,y_test1, name = 'lr2')
lr2_report
Confusion_matrix(lr2)
```

RANDOM FOREST (OVERSAMPLING)

```
RF2 = RandomForestClassifier(n_estimators=100,
criterion= 'gini',
n_jobs=-1,
random_state = 42)
lin_RF2_report = fit_score_clf(RF2,X_train1,y_train1,X_test1,y_test1, name = 'Random_Forest2')
lin_RF2_report
importance_df = pd.DataFrame({
```

```
'feature': Feature_col.columns,
'importance': RF2.feature_importances_
}).sort_values('importance', ascending=False)
importance_df['cumpercentage'] =
importance_df['importance'].cumsum()/importance_df['importance'].sum()*100
fig, ax1 = plt.subplots()
plt.title('Feature Importance')
sns.barplot(data=importance_df.head(25), y='importance', x='feature');
ax1.set_xticklabels(importance_df['feature'], rotation=90)
ax2 = ax1.twinx()
ax2.plot(importance_df['feature'], importance_df['cumpercentage'], color="C1", marker="D", ms=7)
ax2.yaxis.set_major_formatter(PercentFormatter())
ax2.tick_params(axis="y", colors="C1")
plt.show()

Confusion_matrix(RF2)
modify_roc_dict(clf = RF2,
y_test = y_test1,
X_test = X_test1,
name = 'Random Forest(oversampling)(area = 0.93)',
dict_modify = roc_dict)
```

XGB (OVERSAMPLING)

```
xgb2 = xgb.XGBClassifier(n_estimators=300,
subsample=0.9,
n_jobs=-1)
xgb2.fit(X_train1,y_train1)
y_pred = xgb2.predict(X_test1)
xgb2_report = pd.DataFrame(classification_report(y_test1, y_pred, output_dict=True))


xgb2_report
y_pred = xgb2.predict(X_test1)
plot_confusion_matrix(xgb2, X_test1 , y_test1 , normalize='true', cmap = 'Blues')

importance_df = pd.DataFrame({
'feature': Feature_col.columns,
'importance': xgb2.feature_importances_
}).sort_values('importance', ascending=False)
importance_df['cumpercentage'] =
importance_df['importance'].cumsum()/importance_df['importance'].sum()*100
fig, ax1 = plt.subplots()
plt.title('Feature Importance')
sns.barplot(data=importance_df.head(25), y='importance', x='feature');
ax1.set_xticklabels(importance_df['feature'], rotation=90)
```

```
ax2 = ax1.twinx()
ax2.plot(importance_df['feature'], importance_df['cumpercentage'], color="C1", marker="D", ms=7)
ax2.yaxis.set_major_formatter(PercentFormatter())
ax2.tick_params(axis="y", colors="C1")
plt.show()

modify_roc_dict(clf = xgb2 ,
y_test = y_test1,
X_test = X_test1,
name = 'XGB(oversampling)(area = 0.71)',
dict_modify = roc_dict)
```

COMPARISION OF PERFORMANCE METRICS

LOGISTIC REGRESSION

```
datalr = {'Evaluation_metrics -->':['Training_accuracy', 'Testing_accuracy',
'Precision','Specificity','Recall(Sensitivity)','f1-Score','ROC-AUC_Score'],
'logistic':[accuracy_score(y_train,lr.predict(X_train)),accuracy_score(y_test,lr.predict(X_test)),
precision_score(y_test,lr.predict(X_test)),
recall_score(y_test,lr.predict(X_test),pos_label=0),recall_score(y_test,lr.predict(X_test)),f1_score(y_test,lr.pre
dict(X_test)),roc_auc_score(y_test,lr.predict_proba(X_test)[:,1])],
'logistic(Hyperparameter
tunned)':[accuracy_score(y_train,lr1.predict(X_train)),accuracy_score(y_test,lr1.predict(X_test)),
precision_score(y_test,lr1.predict(X_test)),recall_score(y_test,lr1.predict(X_test),pos_label=0),recall_score(y_
test,lr1.predict(X_test)),f1_score(y_test,lr1.predict(X_test)),roc_auc_score(y_test,lr1.predict_proba(X_test)[:,1
])],
'logistic(Oversampling)':[accuracy_score(y_train1,lr2.predict(X_train1)),accuracy_score(y_test1,lr2.predict(X
_test1)), precision_score(y_test1,lr2.predict(X_test1)),recall_score(y_test1,lr2.predict(X_test1),pos_label=0)
,recall_score(y_test1,lr2.predict(X_test1)),f1_score(y_test1,lr2.predict(X_test1)),roc_auc_score(y_test1,lr2.pre
dict_proba(X_test1)[:,1])]}

summary1 = pd.DataFrame(datalr)
Summary1.T
```

RANDOM FOREST

```
datarf = {'Evaluation_metrics -->':['Training_accuracy', 'Testing_accuracy',
'Precision','Specificity','Recall(Sensitivity)','f1-Score','ROC-AUC_Score'],
'RandomForest':[accuracy_score(y_train,RF.predict(X_train)),accuracy_score(y_test,RF.predict(X_test)),
precision_score(y_test,RF.predict(X_test)),
recall_score(y_test,RF.predict(X_test),pos_label=0),recall_score(y_test,RF.predict(X_test)),f1_score(y_test,R
F.predict(X_test)),roc_auc_score(y_test,RF.predict_proba(X_test)[:,1])],
'RandomForest(Hyperparameter
tunned)':[accuracy_score(y_train,RF1.predict(X_train)),accuracy_score(y_test,RF1.predict(X_test)),
precision_score(y_test,RF1.predict(X_test)),recall_score(y_test,RF1.predict(X_test),pos_label=0),recall_score
```

```
(y_test,RF1.predict(X_test)),f1_score(y_test,RF1.predict(X_test)),roc_auc_score(y_test,RF1.predict_proba(X_
test)[:,1])],
'RandomForest(Oversampling)':[accuracy_score(y_train1,RF2.predict(X_train1)),accuracy_score(y_test1,RF2.
predict(X_test1)),
precision_score(y_test1,RF2.predict(X_test1)),recall_score(y_test1,RF2.predict(X_test1),pos_label=0)
,recall_score(y_test1,RF2.predict(X_test1)),f1_score(y_test1,RF2.predict(X_test1)),roc_auc_score(y_test1,RF
2.predict_proba(X_test1)[:,1])]}
summary2 = pd.DataFrame(datarf)
summary2.T
```

XBG

```
dataxgb = {'Evaluation_metrics -->':['Training_accuracy', 'Testing_accuracy',
'Precision','Specificity','Recall(Sensitivity)','f1-Score','ROC-AUC_Score'],
'XGB':[accuracy_score(y_train,xgb.predict(X_train)),accuracy_score(y_test,xgb.predict(X_test)),
precision_score(y_test,xgb.predict(X_test)),recall_score(y_test,xgb.predict(X_test),pos_label=0),recall_score(
y_test,xgb.predict(X_test)),f1_score(y_test,xgb.predict(X_test)),roc_auc_score(y_test,xgb.predict_proba(X_tes
t)[:,1])],
'XGB(Hyperparameter
tunned)':[accuracy_score(y_train,xgb1.predict(X_train)),accuracy_score(y_test,xgb1.predict(X_test)),
precision_score(y_test,xgb1.predict(X_test)),recall_score(y_test,xgb1.predict(X_test),pos_label=0),recall_scor
e(y_test,xgb1.predict(X_test)),f1_score(y_test,xgb1.predict(X_test)),roc_auc_score(y_test,xgb1.predict_proba(
X_test)[:,1])],
'XGB(Oversampling)':[accuracy_score(y_train1,xgb2.predict(X_train1)),accuracy_score(y_test1,xgb2.predict(
X_test1)),
precision_score(y_test1,xgb2.predict(X_test1)),recall_score(y_test1,xgb2.predict(X_test1),pos_label=0)
,recall_score(y_test1,xgb2.predict(X_test1)),f1_score(y_test1,xgb2.predict(X_test1)),roc_auc_score(y_test1,xg
b2.predict_proba(X_test1)[:,1])]}
summary3 = pd.DataFrame(dataxgb)
Summary3.T
```

ROC-AUC CURVE

```
for key in roc_dict:
clf = roc_dict[key]
plt.plot(clf['fpr'], clf['tpr'], label=key)
plt.xlabel("FPR")
plt.ylabel("TPR")
plt.title("ROC")
plt.plot([0,1], [0,1], label='No Discrimination', linestyle='-', dashes=(5, 5))
plt.legend()
plt.show()
```

GitHub link for Codes

https://github.com/adityasarode14/Loan-Default-Prediction