TUNKU ABDUL RAHMAN UNIVERSITY COLLEGE

BEYOND EDUCATION

BACS2003 ARTIFICIAL INTELLIGENCE

202306 Session, Year 2023/24

Assignment Documentation

Project Title: ***Prediction of Bank Loan Default Using Supervised Machine Learning***

Programme: ***RDS Y2S1***

Tutorial Group: ***2***

Tutor: ***Dr. Ho Chuk Fong***

Team members' data

No	Student Name	Student ID	Module In Charge
1	Ryan Kho Yuen Thian	22WMR04097	1. Exploratory Data Analysis 2. Data Preparation & Preprocessing 3. Training & Testing (XGBoost, Artificial Neural Network) 4. Deployment
2	Thong Cheng How	22WMR03154	1. Exploratory Data Analysis

			2. Data Preparation & Preprocessing **3. Training & Testing (AdaBoost, Least Discriminant Analysis)** **4. Deployment**
3	**Ong Weng Kai**	**22WMR03309**	**1. Exploratory Data Analysis** **2. Data Preparation & Preprocessing** **3. Training & Testing (SVM, Random Forest)** **4. Deployment**

1. Introduction

1.1 Problem Background

One of the major financial risks facing banks is credit risk (also known as loan default) which is the failure of borrowers to repay their loans. This is an important problem to be addressed as loan defaults can bring detrimental effects to banks. One of the repercussions is the decrease in the bank's profitability. If the bank does not earn sufficient profit to cover the costs of its operations, when a sufficient number of large loans default, it is possible that the bank can collapse.

Another effect is the diminishing of the bank's asset quality. When a loan, which is an asset, is defaulted, the asset's value decreases and turns into a liability, bringing down the bank's total asset amount. Another serious effect is the drop in the banks' overall credit rating. This is because failure of the banks to collect loan payments will cause them to lose their reputation in properly paying back the depositors and it will also be difficult for them to do business with foreign banks.

Solving the bank loan default problem not only helps banks to survive but also brings benefits to the society, such as being able to drive the economy. This is because they provide money for people to purchase home, vehicle etc and for businesses to expand their operations, purchase equipment that they need etc. Besides that, banks provide a safe place for us to keep our money.

Therefore, to minimise or eliminate the adverse effects of loan defaults, it is critical for banks to identify loan applicants who may have difficulties in repaying the loans and interests, and as such determine whether the bank loans should be approved or not. For this assignment, we will be using an Artificial Intelligence (AI) technique called Supervised Machine Learning (ML) to try to provide reasonably accurate predictions of potential loan defaulters. In order to do this, we will use a Credit Risk dataset obtained from the Kaggle website. We will use several different types of ML algorithms to learn from this dataset. Then we will compare the results from these different ML models and select the best performing model.

1.2 Objectives/Aims

- To provide a reasonably accurate method of predicting loan default using data on the loan applicant's financial history and behaviour

- To compare several Supervised Machine Learning (ML) algorithms in order to determine the best prediction model

- To provide an efficient system to make decisions on loan approvals

- To provide a user-friendly Graphical User Interface (GUI) for the loan default prediction system that can be easily and conveniently used by end users

- To help the banks to make better decisions to reduce loss.

1.3 Motivation

- To reduce the financial losses incurred by banks due to non-performing loans.

- To provide bank staff an accurate and efficient system for loan approvals.

- To help banks to make better decision making on loaning money to individuals

- To predict whether an individual is capable of returning the full loan that has been borrowed money by the bank.

- This can consequently help banks to survive and they can continue to provide money for people to purchase homes, vehicles etc and for businesses to expand their operations, purchase equipment that they need etc and provide a safe place for us to keep our money.

- This drives the economy to be more sustainable in the long run

1.4 Timeline/Milestone

Week	Start Date	End Date	Activity
1	3/7/2023	9/7/2023	Explored the five A.I. disciplines and decided to choose Supervised Machine Learning. For the problem, we decided to solve the issue of bank loan default. Researched on how Supervised Machine Learning can be used to solve this problem. Worked together to find the best dataset
2	10/7/2023	16/7/2023	Described the problem in detail in documentation using the 5W and 1H method and planned what we intend to achieve from this assignment (timeline). Studied the dataset
3	17/7/2023	23/7/2023	Learn Python coding using Jupyter Notebook and learn how to use Pandas for pre-processing.
4	24/7/2023	30/7/2023	Continue tasks from Week 3
5	31/7/2023	6/8/2023	Learn about the different Classification ML algorithms.
6	7/8/2023	13/8/2023	Continue tasks from Week 5
7	14/8/2023	20/8/2023	Learn how to use the Scikit-Learn ML library and entire ML process
8	21/8/2023	27/8/2023	Perform Exploratory Data Analysis on selected dataset.
9	28/8/2023	3/9/2023	Perform Data Cleaning and Transformation on selected dataset and split the dataset into Training, Validation
10	4/9/2023	10/9/2023	Perform ML Modelling, Train models, Evaluate models, Select best model, and test the selected Best Model

11	**11/9/2023**	**15/9/2023**	**Deployment and Documentation**

2. Research Background

2.1 Background of the applications

Provide detailed explanations of the background of the application, e.g. machine learning algorithms, chatbot development, recommender system, sentiment analytic applications, robotic processing automation applications, image processing applications, etc.

2.1.1 Overview of Machine Learning (documented by Ong Weng Kai)

Machine learning is a subfield of Artificial intelligence that focuses on using algorithms and data to imitate how humans learn and gradually improve accuracy. It gives the computer the ability to learn without explicitly being programmed and can perform complex tasks similar to how humans solve problems. The primary function of machine learning can be divided into three parts: prescriptive, predictive and descriptive.

Prescriptive means that the machine would learn the user preference to recommend the content to the user, what type of advertisement is displayed to them etc. Prescriptive is widely used on social media platforms such as Twitter and Facebook because they need to show users the content that will attract and engage them so they can gain more profit by displaying advertisements. It would ensure the advertisement would display to the right audience; by hitting the correct audience, other people are more willing to pay to this platform to attract potential customers.

The predictive model predicts the future based on the historical data. This would be used in a telecommunication company. For example, Maxis would use it to predict which customers are at risk of cancelling their services (churning). The predictive model is significant for telecommunication companies because it allows them to allocate their resources properly by giving promotions and discounts to the target customers. Customers with a high risk of churning but with high potential lifetime value would be more likely to be their target audience to this is because The company can earn revenue from customers who are loyal and are more likely to stay with the company for an extended period, so retain customers who are more likely to leave but with high potential lifetime value would increase overall profit. Doing this would increase customer loyalty, increase lifetime value, and eventually increase overall profit.

Descriptive explains what has already happened. For example, an e-commerce company like Shopee would use it to understand their customer base better so they can give advertising campaigns more effectively. The main advantage is it would increase customer experience because by segmenting customers, the company can provide more personalised advertising to each relevant group. It would increase the likelihood of conversion. A high conversion rate means it would increase the company's revenue as more people buy their product. This would also be more cost-effective as the company can allocate its marketing budget to a specific customer segment to reduce spending on less relevant audiences.

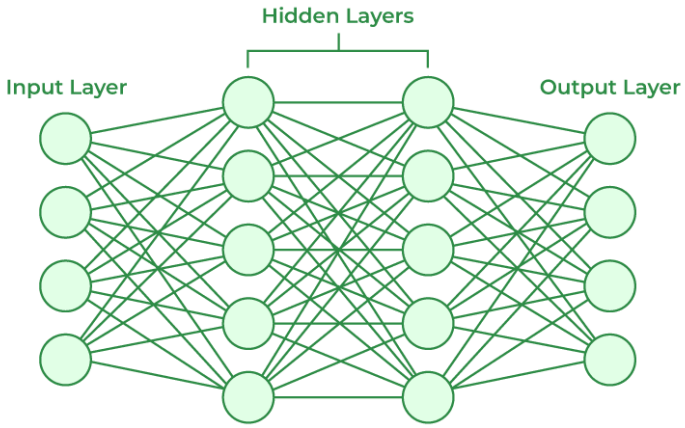
The subcategory of machine learning would be unsupervised, supervised and reinforcement. Supervised learning deals with labelled data, for example, training an algorithm to recognise and differentiate dog and cat pictures.
Unsupervised machine learning deals with unlabeled data, for example, grouping similar customers based on buying patterns without label.

Reinforcement learning means training models to make decisions by interacting with the environment. The model would receive either rewards or penalties, and the main goal is maximising cumulative compensation. This is widely used to train autonomous vehicles to tell them what is right or wrong. The application example would be penalties for hitting an obstacle and giving a reward for staying within the speed limit, following traffic rules or reaching the destination quickly so the model would know what action to take.

2.1.2 Artificial Neural Networks (documented by Ryan Kho Yuen Thian)

Overview:

Artificial Neural Networks (ANN) are supervised machine learning algorithms that are based on the human brain’s biological neural networks. Deep learning uses ANN to solve practical tasks in a variety of fields such as computer vision (image), natural language processing (text), and automatic speech recognition (audio). Below is a diagram of an Artificial Neural Network.



An ANN is made up of interconnected nodes, aka neurons, that are organised into layers. The input layer, which is the first layer of the neural network, accepts input data. The hidden layers (the processing layer) are the intermediary layer between the input and output layers where every computation is done. The last layer is the output layer, where results are produced. Flowing through the nodes is information. During training, the weights are actually adjusted by the network in order to learn from the data, allowing it to see patterns, predict, as well as to solve different tasks in artificial intelligence and machine learning. Activation functions are used in ANN to decide whether a neuron should be activated or deactivated. This is done by calculating the weighted sum and adding bias to it further. Its purpose is to introduce non-linearity into the neuron's output, making the ANN capable of learning and performing more complicated jobs. Variants of activation functions include sigmoid function, tanh function and RELU function.

For this project assignment, we will be using the Sci-Kit Learn Multi-Layer Perceptron (MLP) library to perform binary classification for predicting loan default. More specifically, we will be using the MLPClassifier. The reason for this choice is because this is our first venture into ANN and we want to start out with something which is not overly complex. Sci-Kit Learn's MLP library is not meant for large-scale applications as it does not have GPU support.

<u>Advantages:</u>

Artificial Neural Networks have the capacity to learn and model complicated and non-linear relationships. In real life, a great deal of relationships between outputs and inputs are complicated and non-linear. Another advantage is that it can generalise. This means that ANN is able to work out hidden relationships on unseen data after the algorithm learns from the initial

inputs and their relationships. Therefore, ANN can generalise and make predictions on data that are unseen. Next, it was found that ANN can model heteroskedasticity better since ANN can learn unseen relationships in the data without setting fixed relationships in the data. Heteroskedasticity refers to data that is highly volatile and has variance that is non-constant.

Disadvantages:

On the training data, ANN is susceptible to overfitting. One of the main reasons that overfitting likely occurs is due to the fact that an Artificial Neural Network's structure and size are mostly selected via trial and error. Another drawback is that it takes a long time for ANN to be trained. This can also mean that the bigger the dataset and the more parameters, the longer ANN takes to converge on an answer. Lastly, convergence on a solution or prediction is not guaranteed.

2.1.3 XGBoost (documented by Ryan Kho Yuen Thian)

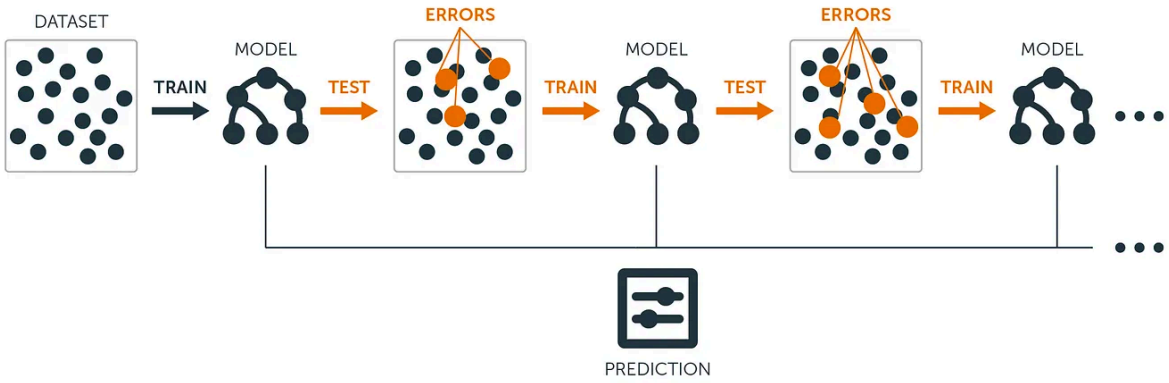
Overview:

XGBoost (Extreme Gradient Boosting) is a supervised ensemble Machine Learning (ML) algorithm which is an implementation of gradient-boosted decision trees that is built for performance and speed. Models based on XGBoost often outperform models based on other ML algorithms. XGBoost can be used for both regression and classification tasks. Since we will be using XGBoost for binary classification in this project assignment, I will describe the working of XGBoost from the classification point of view.

A gradient boosting classifier, such as XGBoost, makes use of logistic regression concepts. It uses log-odds to make a prediction, converts the log-odds to probabilities through the logistic function, and then make a classification based on a user-defined threshold. For example, let's say we want to predict whether a student will pass an exam (binary classification). The outcome "Pass" is represented by 1 and "Fail" by 0. Given the threshold 0.5, if a probability of a student passing the exam is 0.8, he/she will be classified as "Pass" (0.8 > threshold 0.5).

Decision trees are used as the weak learners by this gradient-boosted ensemble algorithm. As with other boosting algorithms, XGBoost sequentially trains the model and each weak learner attempts to rectify the weaknesses of the previous ones. To be more specific, each decision tree

is designed to fit the residuals (errors) of the predictions of the previous trees. The term "gradient boosting" means a gradient descent algorithm is used to optimise the loss function, which is to correct errors made by previous weak learners. This learning process is illustrated by the diagram below.



Let's say we are trying to predict whether students will pass or fail an exam based on the number of hours studied. Each student will have a label of 1 (Pass) or 0 (Fail). The first weak learner created will give a default prediction score of 0.5 to each student, which means each student has a 50% chance to pass the exam. Next, we calculate the residuals which is the label (1 or 0) minus the prediction i.e. (1 - 0.5 = 0.5) for label 1 and (0 - 0.5 = - 0.5) for label 0. After that, a second weak learner (decision tree) is created based on these residuals. In the decision tree, the residuals are split based on a metric called Similarity Score (formula shown below).

$$\text{Similarity score} = \frac{(\Sigma \text{ Residuals})^2}{\Sigma\,[\text{Prob}*(1-\text{Prob})] + \lambda}$$

$$\text{Gain} = \text{Sim(Left node)} + \text{Sim(Right node)} - \text{Sim(Root node)}$$

A hyperparameter (denoted by the lambda symbol) is added to the denominator of the similarity score to help to prevent overfitting. This hyperparameter is a regularization parameter which lowers the values of the similarity score, limiting the ability of the model to learn. Next, we proceed to split the node using the 'Number of Hours Studied' feature in all the possible ways. For each split, we compute the Gain. This is the sum of the similarity score of the left and right nodes minus the similarity score of the root node. The split which results in the highest Gain is used to split the root node.

At this point, before we consider the second weak learner complete, there is one more step left. In order to reduce overfitting, XGBoost does a check to decide whether or not to perform tree pruning, in which we remove the branches of the trees if they don't satisfy certain conditions. XGBoost prevents a split from occurring if the gain value computed for the node is less than the Cover value which is computed as follows:

$$\text{Cover value} = \sum^{N}[P\ (1 - P)]$$

After the split (or decision not to split), the calculation of the gain is repeated for the new node(s). A new cycle starts which recomputes new residuals from the new predictions and thereafter create a new weak learner (decision tree) to fit the new residuals and so on. With the completion of the weak learners, we can now make predictions. The combined prediction of all the weak learners is equal to the prediction of the first weak learner plus the predictions of the rest of the weak learners, each multiplied by the learning rate. The effect of multiplication by the learning rate is to slow down the convergence to a good prediction, with the intention to avoid overfitting.

<u>Advantages:</u>

XGBoost is a powerful algorithm with a combination of hardware and software optimisation methods, that produces superior results in the least amount of time with less computing resources. Besides speed, this algorithm was also created for performance on huge datasets.

There are several reasons why XGBoost has good performance. Firstly, it uses regularisation to prevent overfitting and efficiently handles missing values. In every iteration, XGBoost has a cross-validation method built into it, which removes the need to program the search in an explicit manner and to indicate the precise number of boosting iterations needed in one run.

Another reason for its performance is because it uses tree pruning, a depth-first method that brings significant improvement to computational performance. Another advantage is its customisability, which means that its broad range of hyperparameters can be tweaked for optimal performance. XGBoost also provides feature importances, which enables one to understand better about which variables are most essential in predicting. This means that XGBoost can be easily interpreted.

Disadvantages:

Especially when training huge models, this algorithm can be computationally intensive. This makes it less appropriate for systems that are limited in resources. The algorithm can also be memory intensive particularly when dealing with huge datasets. This means that XGBoost is less appropriate for limited memory systems.

2.1.4 Least Discriminant Analysis (documented by Thong Cheng How)

Overview:

Least discriminant analysis is a supervised learning algorithm that is commonly used for problems such as classification task in machine learning in general. Least Discriminant Analysis uses techniques of linear combination that allows the model to separate features/variables into different classes.

With different classes that has been separated into different classes, usually how Least Discriminant Analysis chooses a variable/feature is by selecting the data which has a lower-dimensional space that maximises the separation between the two classes. In another word, the classes are also separated based on the space that the separate the different class of data.

Apart from that, Least Discriminant Analysis assumes that data in Gaussian's distribution and the covariance matrices can be differ in class can be equal. Because in the theories of Least Discriminant Analysis found, the classes are separated by the linearity of the covariance and gaussian's distribution, Therefore making the LDA a more linearly decision boundary compared to the other models in sklearn library.

The advantages of LDA:

-LDA can work efficiently with simple and computational data

-LDA can handle numbers of feature if the number of feature is much more compared to training set.

-can handle multicollinearity in the dataset (if happens to occur during the user explore it in the pre-processing phrase)

-Depending on the scenario of the dataset, LDA has also different types to test for example using NDA (Normal Discriminant Analysis) or DFA (Discriminant Function Analysis)

The disadvantage of LDA:

-Assumes the data has a Gaussian distribution

-it assumes the data has a covariance matrix that the different class has equal value for the covariance.

-May not perform well with high dimensional datasets

-It assumes the data is linearly separable

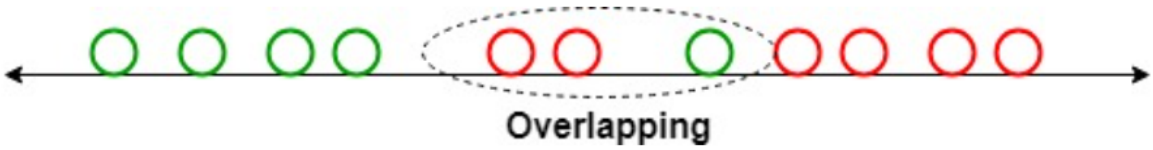
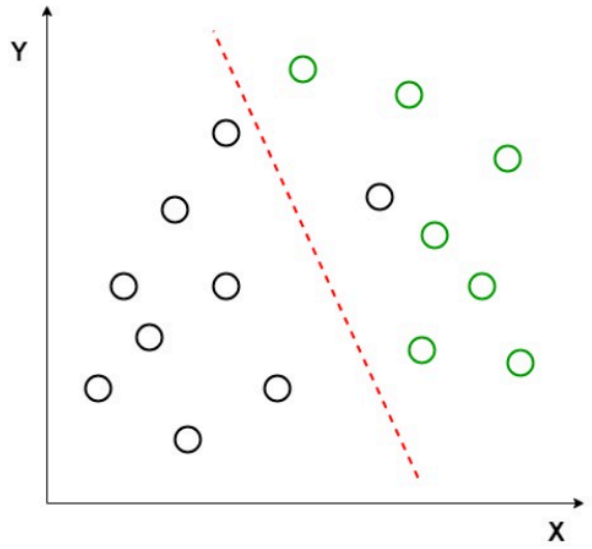
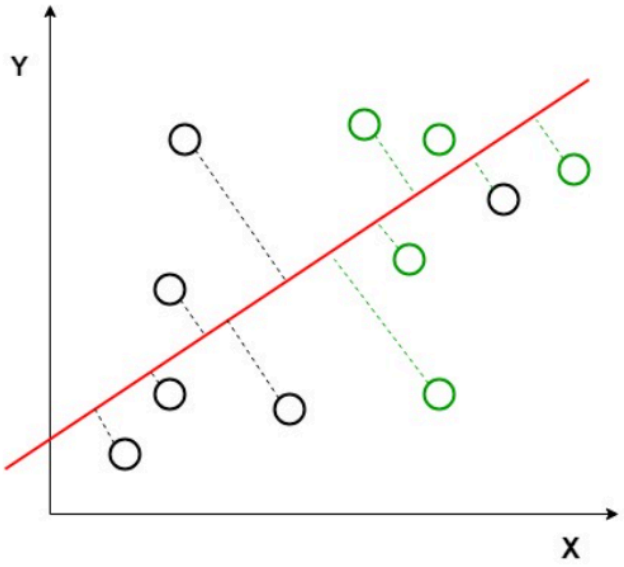


Illustration above, helps to demonstrate how linear discriminant analysis classifies two different classes of data. As above the reason why the green balls overlap with the red balls is because this illustration is only using one single feature to classify. As time goes on, the feature will increase and the green balls will slowly stop overlapping with the red balls.



As this image shows, white balls represent a class and the other green balls represent another class. As the red dotted line represents the separations of the classes by using the Linear Discriminant Analysis model to split the two classes apart. As currently the graph is a 2D graph and linear discriminant analysis splits it into 1D graph. The reason why linear discriminant analysis uses 1D graph instead of 2D graph is because linear discriminant analysis can maximise the separation of the two classes as shown in the red dotted line that represents the separation of the two different classes.

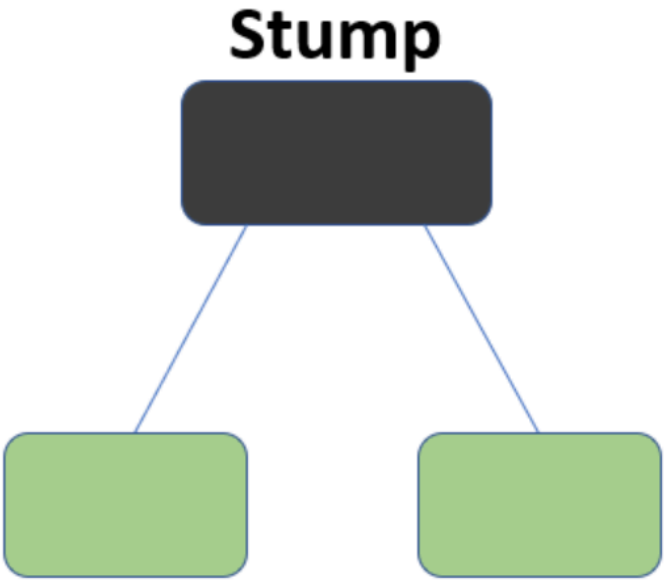


As the illustration above shows, two different criteria are used to maximise the abilities of linear discriminant analysis. The most important criteria are the maximum distance between the means of two classes and to minimise the variation within each class. A straight red line indicates the LDA model is trying to fulfil the requirements based on the criteria stated before. Not only that, a new axis has also been generated to maximise the separation between the data points of two classes as shown above which is the white balls and the green balls.

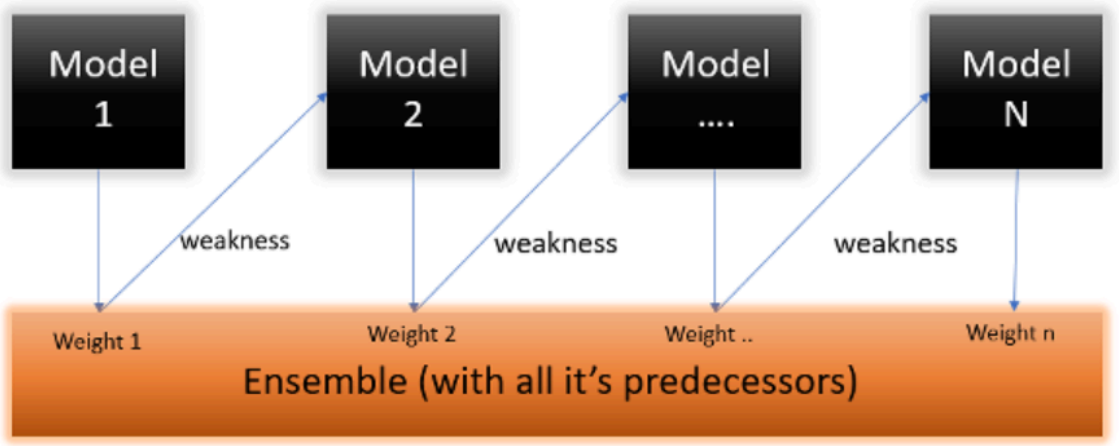
As shown above, the linear discriminant analysis has been completed in this particular illustration as stated before. As circled in the black and green balls which indicates two different classes as it is linearly separated. Another word is that LDA has successfully linearly separated two different classes and linearly split them according to the requirements in the criterion of LDA.

2.1.5 AdaBoost (documented by Thong Cheng How**)**

Overview of AdaBoost:



AdaBoost stands for adaptive boosting. AdaBoost uses techniques from machine learning such as ensemble methods for solving classification / regression problems. Models that show significantly weaker performance can be combined with adaBoost to form a strong model when combined together, so the result can be more accurate and precise. Usually, weaker models can be decision trees with one split. Because usually decision trees need more than one split to predict a better result. Where decision trees with only one split are usually called decision stumps. As the illustration shows, ada boost model gives weights to all the data points that is associated with the dataset. Ada Boost then assigns and evaluates higher weights to those data points that have been wrongly classified. As why ada boost uses this strategy is for the next stage as ada boost gives more priority to the data points that contain higher weights compared to other lower weights. As it will not stop there until all data points as its own values.



As shown in the illustration above, these models are examples from different models where those models considered to be weaker or models that are considered to be better are classified in different areas of the model. Ada boost's main objective is to use the weaker or stronger models to do prediction on which model can perform better as assumed the prediction power has been modified for ada boost to evaluate the models. From the picture above, these models are considered to be quite weak. Therefore Linear Discriminant Analysis puts weights into the models that are weak to modify and make the accuracy rate higher.

Overview:

AdaBoost takes a weak classifier that has been trained from a given dataset. This dataset is based on the weighted samples of the dataset. This weight indicates the importance of the correlation of the weighted sample in the dataset. As usually the weaker classifier will be given an equal weight for every set in the dataset. In the dataset, each variable will be given a decision stump to all the variables and also the target variables. After finding all the correlation of all variables and target variables. The weaker classifier for that variable will be added with more weights, because there is a theory saying that, if adding more weights the accuracy will likely increase over time. And the model will perform better over time

Advantage of adaboost:

AdaBoost can be used for improving the predicted result of other weak classifiers and cases of classification problems with slightly weaker results. AdaBoost also does not need many input parameters for the model to work effectively. Another thing is that adaBoost can work well with SVM too.

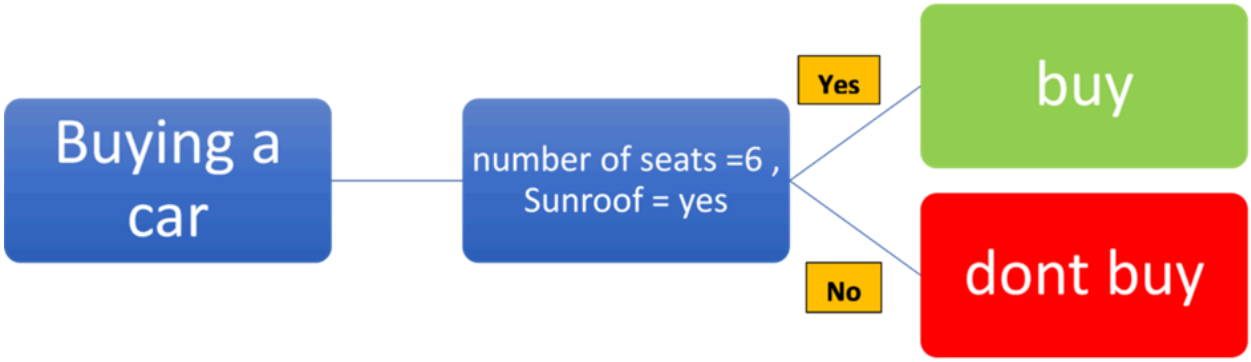
Disadvantage of adaboost:

AdaBoost only can be using progressively learning boosting techniques because with progressively learning, the boosting needs a high-quality dataset to be trained with it. Because without high quality data, boosting would take a longer time because if the data is not processed properly, it will result in a lower accuracy. Outliers and noisy data will occur because training a good dataset, needs to find all possible outliers and noisy data. Compared to XGBoost and AdaBoost, XGBoost will produce the result more efficiently because AdaBoost needs to use a weaker classifier or a classifier that produces a weaker link.

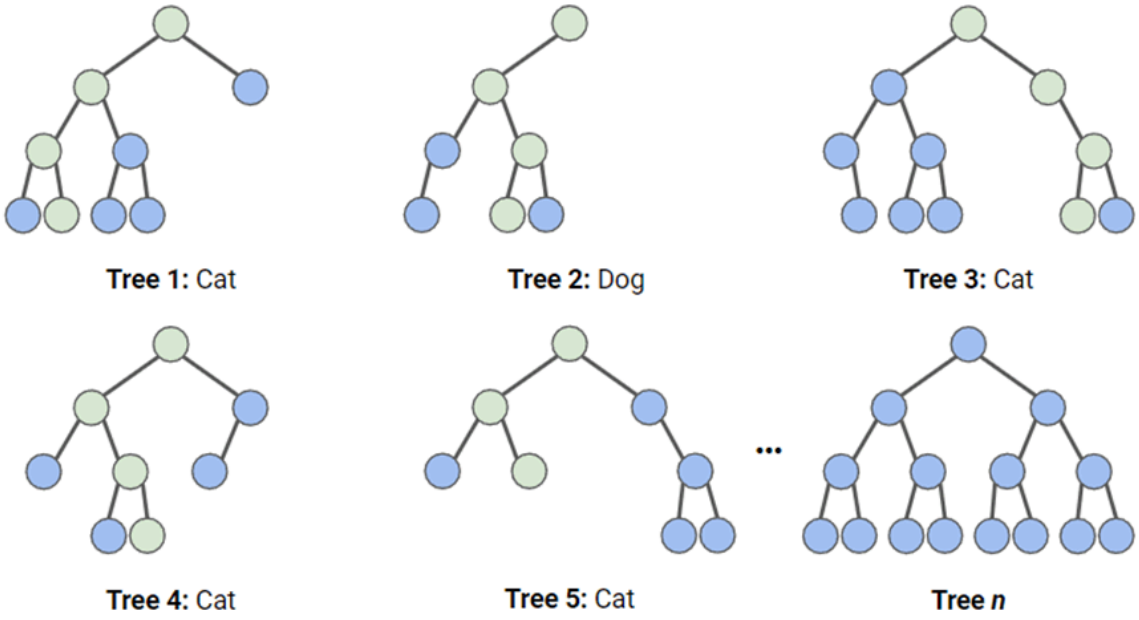
2.1.6 Random Forest (documented by Ong Weng Kai)

Overview

Random forest is a supervised machine learning algorithm widely used in classification and regression problems, and it is an ensemble method that combines predictions from another method. It can be used to handle a variety of datasets ranging from categorical variables to continuous variables. It does not search for the best prediction and does not rely on a singular decision. Instead, it assembles randomised decisions and makes the final decision based on the majority. In the real world, a forest means a combination of trees. In machine learning, it means a combination/ensemble of the decision tree. Before explaining random forests, It is necessary to understand how decision trees work. A decision tree begins with a root node, splits the data based on specific criteria, and then creates a new internal node that corresponds to the best attribute and connects it to the root node. Partition the dataset into subsets based on the values of the best attribute. And then repeat the same process. This is how a decision TREE works. a random forest model merges/ensembles multiple decision trees and combines the output of all these trees to get a stable and accurate prediction without biases. The final decision is based on averaged(Regression)/ max voted (classification), based on majority voting or averaging for classification and regression, respectively. In short, it would get the final decision from the decision tree and then count every tree; most trees vote for certain decisions. That would be the final decision made by a random forest. So, to simplify, random forest is a machine learning algorithm with multiple decision trees, and it makes predictions that do not rely on one single model (decision trees). It combines multiple decision trees, and the final decision would rely on the majority. The steps to run a random forest would be to take some training dataset, make a decision tree, repeat the process for some time and then make the decision by taking a majority vote.



Based on the diagram above. The final decision would be made based on the feature number of seats and sunproof. This decision tree would give the highest rating for the model that satisfies both criteria, the lesser if either of the parameters is satisfied, and the lowest if both are not satisfied. The final decision would be to buy or not buy. Combining the decision tree with different criteria with the final decision to buy or not buy would form a random forest. If a majority votes to buy, the final decision is to buy. The decision tree would only make the decision based on the feature selected. Only if a decision tree considers all the features that would cause an overfit model. The feature selected by the decision tree is by a method called bagging, the data selected would be random, and it can be used again (with replacement) or without replacement(kept aside). The randomness would allow finding the feature's importance; the features that influence the majority decision tree would be of maximum importance. In the classification scenario, the decision would be yes or no. Each decision tree consists of three nodes: the root node, the leaf node and the decision node. The root node is where the data is stored, the decision node is a function to decide is made, and the leaf node is the node where the final decision is made.



The diagram above shows a random forest where the prediction is either cat or dog; the first five predictions are shown. Each tree is trained with different sample of the dataset and different numbers of features. Assume the ratio remains the same, where 4/5 would predict a cat, and 1/5 would be a dog. The green circle means the tree's hypothetical path to reach its decision. The random forest would choose the cat as the final prediction as it is the most popular prediction after counting the number of predictions from the decision tree.

Advantages

1. Highly accurate
2. Can handle multiple features (complex data)
3. Can handle both regression and classification tasks
4. Easily understandable
5. Mitigate overfitting (as it is based on averaging or majority voting)
6. It works well for both categorical and continuous values.
7. Work well with missing/null values
8. Highly stable as the prediction depends on multiple trees.
9. Immune to the curse of dimensionality as it does not consider all the attributes.

Disadvantage

1. Biassed to specific features (sometimes)
2. Slow and ineffective due to large numbers of trees.
3. It is not a descriptive model, so it cannot find the description relationship of the data.
4. Decision tree might suffer from overfitting if it grow without limitation.
5. Require more computational power as it combines all the decision tree output.

6. Long training time.
7. Suffer from interpretability as it is an ensemble of decision trees, so it fails to determine each variable's significance.

2.1.7 Support Vector Machine (SVM) (documented by Ong Weng Kai)

Overview

Support vector machine (SVM) is a supervised machine learning algorithm that can be used to solve regression and classification tasks. SVM can perform exceptionally well when it requires classifying data elements into two groups to perform binary classification. This algorithm aims to separate different data points using the best possible line, often called the Decision boundary. It is referred to as a hyperplane when working in high-dimensional feature spaces. SVM would select a hyperplane with the highest margin between the distance of the hyperplane and the closest data points of each category, making it easy to distinguish. Nonlinear SMV can transform data into higher-dimensional space using mathematical tricks; this is especially useful when dealing with complex data, and a simple straight line cannot separate it.

SVM can deal with non-linearly separable data and linearly separable data. SVM can use different kernel functions such as linear kernel, radial basis function(RBF) kernel or polynomial kernel to capture the pattern in the data in a complex relationship. SVM find an optimal hyperplane in higher-dimensional space using mathematical formulation kernel space. The hyperplane can minimise the classification error and maximise the margin between data points of different classes. The kernel Function allows original feature space to the kernel space, and selecting the proper kernel function would significantly impact the algorithm's performance.

a) linear kernel
When data is linearly separable, the data to a higher dimensional space.
b) polynomial kernel
when data is non-linearly separable and is used to map the data to a higher dimensional space.
c) RBF kernel
This kernel is mainly used for classification problems.

The more powerful kernel function requires more data and computation time, and it would result in higher accuracy and complexity. Once trained, SVMs can classify new, unseen data points by

determining which side of the decision boundary they fall on. The output of the SVM is the class label associated with the side of the decision boundary.

The more powerful kernel function requires more data and computation time, and it would result in higher accuracy and complexity. Once trained, SVMs can classify new, unseen data points by determining which side of the decision boundary they fall on. The output of the SVM is the class label associated with the side of the decision boundary.

Type of SVM

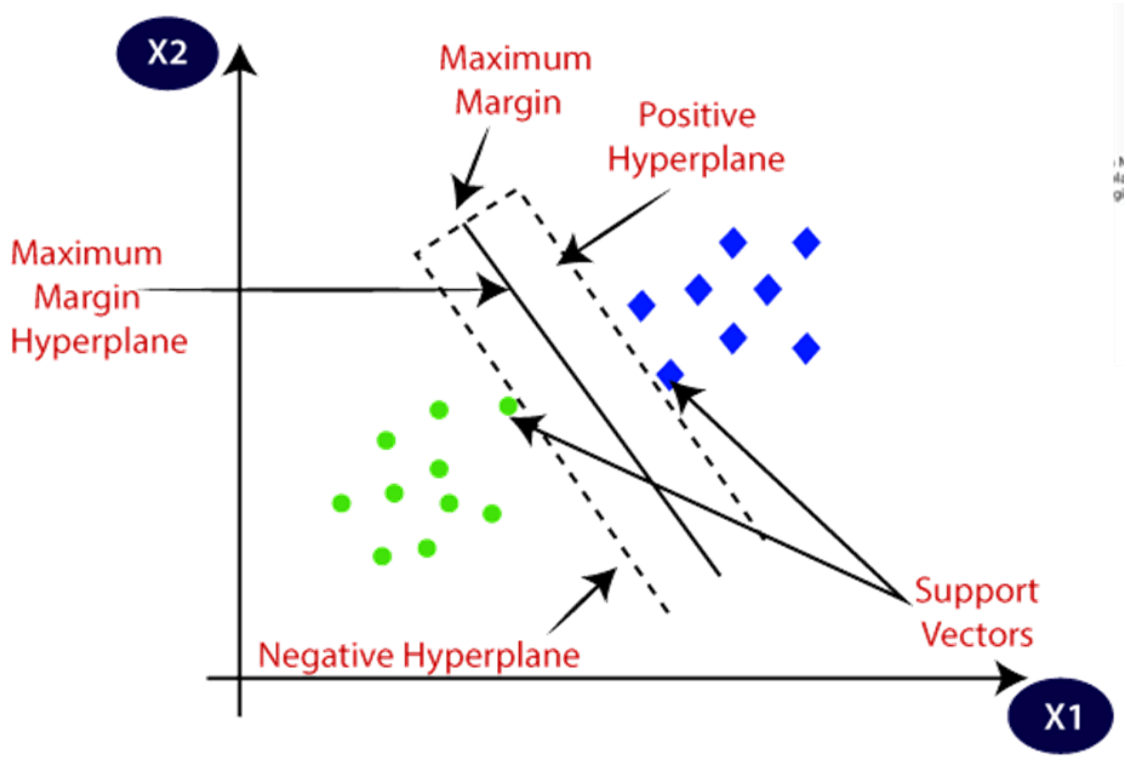
A) Linear SVM

Linear SVM is effective and computationally efficient when the data is linearly separable. Linear SVM is easy to understand and interpret as it uses a linear kernel to create a straight-line decision boundary to separate different classes.

b) Non-linear SVM

Nonlinear SVM is used when the data cannot be separated by a straight line in the input feature space. So, it uses a kernel function to map the data into higher-dimensional feature space where it can use linear decision boundaries to split the data. This type of SVM would have higher classification accuracy but with the cost of higher computation times.

The hyperplane is used to split the class by a straight line. The line that separates the two categories is the decision boundary: blue is on one side, and green is on another.

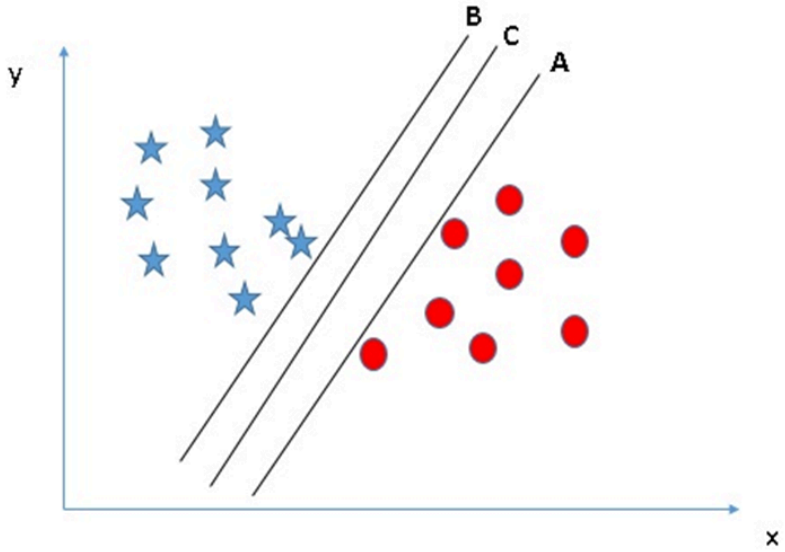


Margin - distance between support vector and hyperplane

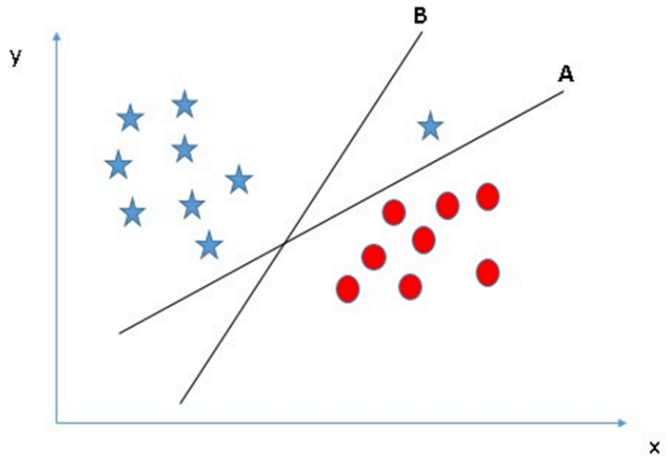
Support vector- the datapoint that are closest to the hyperplane.

How SVM Works:

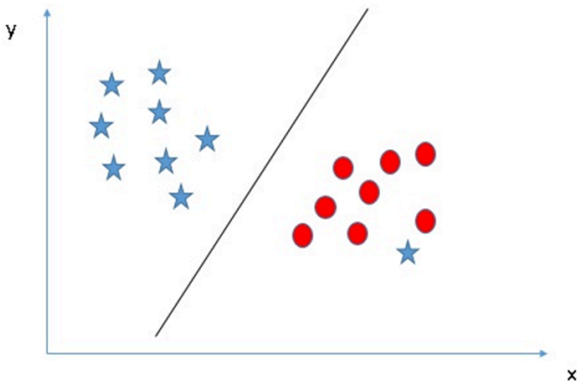
In this section, Is mainly discuss about how Hyperplane segregate the classes.



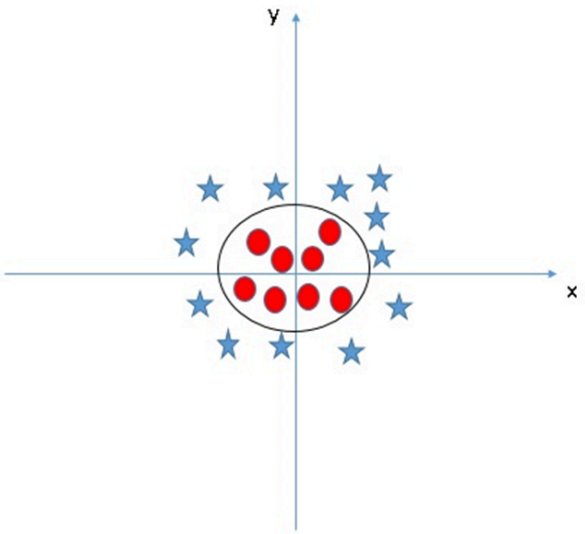
Scenario 1: B, C, and A can segregate the classes well; which is the best hyper-plane?
C would be the best hyperplane as it has the highest margin (the distance between the support vector and the hyperplane). The hyper-plane with a lower margin would be more likely to cause misclassification, and a higher margin is more robust to outliers than a lower margin.



Scenario2: A hyperplane would be preferable even if B has a higher margin than A because it would prioritise class accuracy over maximising margin. A classified all correctly, and B had a classification error.



Scenario 3: There has an outlier for the star class. However, SVM can ignore the outlier and find the hyper-plane that has the maximum margin. SVM is robust to outliers, as It is unnecessary to generalize all the data points. It only needs to generalize most of the points. They can perform well despite having an outlier.



Scenario4: When cannot have a linear hyper-plane to separate two classes, When this happened, SVM has a technique called the kernel trick where it can convert a not separable problem to a separable problem. It transforms into high dimensional space from low dimensional input space; It would do extremely complex data transformation to separate the data based on the output that has been defined.

Advantages

1. Effective in high Dimensional Spaces: It works well with data with many independent variables, even if the independent variables exceed the number of samples.
2. Versatile: SVM can adapt to different data types due to the Kernel Trick function.
3. Generalization: Maximizing the margin around the separating hyperplane. Make it easier to interpret.
4. Robustness: It is less likely to be affected by outliers and overfitting as the model is used to generalize most of the data points, not all.
5. Better speed, performance, and good accuracy with less memory consumption as it only uses a subset of training points in the decision phase.

Disadvantages of SVM:

1. High processing time: SVM does take a long time to process large datasets; it needs to scale better with the size of the dataset.
2. Selection of kernel and regularization: It is hard to choose the proper regularization parameter and kernel, and it would affect the model's performance significantly.
3. Sensitive to overlapping classes and noise: The model would underperform if the noise in the data is high and the dataset is overlapping.
4. Hard to interpret: Due to its black-box nature, the concept is relatively difficult to interpret and understand compared with another algorithm. This is essential when interpretability is important.
5. Sensitive: The model performance varies with a different type of kernel used.

2.2 Analysis of selected tool with any other relevant tools

Part 1:

		Selected tools		
Tools comparison	**Remark**	**Google Colaboratory**	**Scikit-learn**	**Jupyter notebook**
Type of licence and open source licence	**State all types of licence**	Google Workspace Licence	New BSD License and open source	3-Clause BSD License and open source
Year founded	**When is this tool being introduced?**	2017	2007	2014
Founding company	**Owner**	Larry Page, Sergey Brin	David Cournapeau	Fernando Pérez, Brian Granger
Licence Pricing	**Compare the prices if the licence is used for development and business/ commercialization**	Free	Free, machine learning library for python	Free
Supported features	**What features that it offers?**	Run python code, without needing to download, install or run anything in the file	Machine learning algorithms, data preprocessing, model selection	Create / share document that contains code, visualisation tools for data analytics

Common applications	**In what areas this tool is usually used?**	Machine learning, artificial intelligence	Machine Learning, Data Analysis	Machine learning, artificial intelligence, collaboration
Customer support	**How the customer support is given, e.g. proprietary, online community, etc.**	Call using their office number, email google customer service, visit their office	Using documentation, online community	Documentation, option in the menu or using online community
Limitations	**The drawbacks of the software**	Maximum runtime is only 12 hours, remote desktops	Not very suitable for string processing, for graph algorithms	No build in-version control, can be slow when running code

Part 2:

		Selected tools	
Tools comparison	**Remark**	**XGBoost**	**Tkinter**
Type of licence and open source licence	**State all types of licence**	Apache License 2.0	Python License
Year founded	**When is this tool being introduced?**	2014	1991
Founding company	**Owner**	Tianqi Chen & Distributed (Deep) Machine Learning Community (DMLC) group	John Ousterhout
Licence Pricing	**Compare the prices if the licence is used for levelopment and business/ commercialization**	Free	Free
Supported features	**What features that it offers?**	Machine Learning Algorithms, High Scalable (supports Distributed & Parallel Computing)	Python-based GUI toolkit
Common applications	**In what areas this tool is usually used?**	Machine Learning (regression, classification, ranking, user-defined prediction problems)	Development of Desktop User Interfaces

Customer support	**How the customer support is given, e.g. proprietary, online community, etc.**	XGBoost Community Support, Documentation	Various communities' support
Limitations	**The drawbacks of the software**	Prone to overfitting, not very good performance on unstructured and sparse data, has numerous hyperparameters that can be adjusted	Lack of advanced widgets

*The reason there is no column on comparison tool in the above 2 tables is because we used whatever tools we could find.

2.3 Justify why the selected tool is suitable

Google Colaboratory is one of the selected tools for this assignment because it allows everyone in the group to update the codes for our assigned part in a google colab notebook. This will help to know how everyone is doing and if anyone is facing problems, it is also easier to solve because everyone in the group has access to the google colab notebook. Secondly, the codes will be stored in the cloud, meaning that our assignment will be available 24 by 7. Therefore we don't need to worry about the performance of our machine slowing down due to long lines of code to be executed, lack of storage in our machines etc. Another reason we chose google colab is because the code is run through google servers. That means that the code can run more efficiently and faster because Google has excellent and high performance servers. Not only that, any device with an internet connection can access the google colab notebook to edit or run the code from any device.

Despite having a purpose that is similar to Google Colab, **Jupyter notebook** is also selected. This is because based on our experience, when more than one person tries to edit or update the code in Google Colab at the same time, it is possible for one's changes to be overridden by another person. We utilised Google Colab mostly in the modelling, evaluation and deployment parts of our assignment. On both Google Colab and Jupyter Notebook, it may take quite a long time to train and test the various models so we would not want the hours of waiting to be wasted. Also, we experienced some problems running code related to Tkinter in Google Colab as it requires us to do additional configurations. Therefore, it can be said that the Jupyter Notebook helps to prevent our progress from being interrupted by others accidentally.

Scikit-learn was selected for most of the machine learning algorithms we use because it contains many libraries for machine learning algorithms. Scikit-learn is also considered as the most used algorithm that is used by many professional machine learning engineers. Scikit-learn also contains many machine learning tools to help extract data from datasets to help analysis. As stated in the ANN section above, we chose the MLPClassifier because it is suitable as a starting out library for those who are just venturing into ANN as it is not overly complex.

XGBoost was chosen because it has a strong reputation as a robust and high performing ML framework as it has won many machine learning competitions. Therefore we want to see if it lives up its reputation by using it for our loan default classification problem.

Tkinter was chosen to develop the user interface of the data entry program which will be used by an end-user to key in data and receive a prediction from the model. This is because Tkinter is the standard GUI of Python and it is included in the Windows, Linux and macOS install of Python. Moreover, Tkinter is relatively easy to learn and implement compared to other GUI toolkits. As it is also accessible to most of the machines compatibility and software.

3. Methodology

3.1 Description of dataset

Remarks:
This dataset, which belongs to an organisation called Grant Group Funding, has 87500 records and 30 columns. Each record contains loan and customer financial history information. The link to the dataset can be found here:

https://www.kaggle.com/datasets/marcbuji/loan-default-prediction

So far, there has been no Code or Discussion posted on the Kaggle website for this dataset from other parties.

a) Viewing the dataset.

```
In [2]: #from google.colab import drive


        df = pd.read_csv('Data_Train.csv')
        df.head(10)
```

Out[2]:

	ID	Asst_Reg	GGGrade	Experience	Validation	Yearly_Income	Home_Status	Unpaid_2_years	Already_Defaulted	Designation	...	File_Status	
0	95074	421802	II	>10yrs	Vfied	633600.00	MORTGAGE	0	0	GLENDALE NISSAN	...	fully paid	Ca
1	134131	3964312	IV	7yrs	Source Verified	85483.20	RENT	0	0	Business Teacher	...	whole	
2	92602	4247560	III	<1yr	Source Verified	79200.00	RENT	0	0	driver	...	whole	
3	22591	197179	III	<1yr	Vfied	61600.00	RENT	0	0	AvePoint	...	fully paid	New
4	125060	4646684	V	2yrs	Source Verified	68053.92	RENT	0	0	Lead Tester	...	fully paid	
5	21803	1771402	II	1yrs	Source Verified	219648.00	MORTGAGE	4	0	Financial Controller	...	whole	
6	141442	6308843	III	>10yrs	Source Verified	88000.00	RENT	0	0	Chef	...	fully paid	Ca
7	94448	1468916	III	4yrs	Source Verified	NaN	RENT	2	0	General manager	...	fully paid	
8	109853	7067902	VII	<1yr	Vfied	NaN	RENT	0	0	laborer	...	whole	Ca
9	138407	2704860	II	2yrs	Not Vfied	149600.00	OWN	0	0	truck driver	...	whole	

10 rows × 30 columns

Figure 1a: Showing sample data in the dataset

```
In [2]: #from google.colab import drive


df = pd.read_csv('Data_Train.csv')
df.head(10)
```

Out[2]:

_Defaulted	Designation	...	File_Status	State	Account_Open	Total_Unpaid_CL	Duration	Unpaid_Amount	Reason	Claim_Type	Due_Fee	Default
0	GLENDALE NISSAN	...	fully paid	California	17	58598.0	3 years	31216.05	debt consolidation	I	0.0	0
0	Business Teacher	...	whole	NC	15	18924.0	5 years	11660.49	debt consolidation	I	0.0	0
0	driver	...	whole	Florida	7	15604.0	5 years	5637.87	major purchase	I	0.0	0
0	AvePoint	...	fully paid	NewJersey	9	22410.0	3 years	15607.17	major purchase	I	0.0	1
0	Lead Tester	...	fully paid	LA	10	36022.0	5 years	27472.86	debt consolidation	I	0.0	0
0	Financial Controller	...	whole	TX	11	27556.0	3 years	19134.90	debt consolidation	I	0.0	0
0	Chef	...	fully paid	California	10	53618.0	3 years	29748.87	debt consolidation	I	0.0	0
0	General manager	...	fully paid	CT	26	123338.0	3 years	17496.72	debt consolidation	I	0.0	0
0	laborer	...	whole	California	6	19256.0	5 years	225.72	other	I	0.0	0
0	truck driver	...	whole	TX	10	18592.0	3 years	NaN	other	I	0.0	0

Figure 1b: Showing sample data in the dataset

Description of each of the 30 variables:

No	Column	Description
1	ID	Unique ID
2	Asst_Reg	Value of all the assets registered under the borrowers name
3	GGGrade	Grant Group Grade
4	Experience	Total year of work experience of the borrower
5	Validation	Validation status of the borrower
6	Yearly Income	Total yearly income of the borrower
7	Home Status	Borrower living status
8	Unpaid 2 years	No. of times the Borrower has defaulted in last two years

9	Already Defaulted	Number of other loans the borrower was default
10	Designation	Designation of Borrower
11	Debt to Income	Debt to Income ratio
12	Postal Code	Postal code of borrower
13	Lend Amount	Total funded amount to borrower
14	Deprecatory Records	An entry that may be considered negative by lenders because it indicates risk and hurts your ability to qualify for credit or other services
15	Interest Charged	Interest charged on total amount
16	Usage Rate	Processing Charges on the Loan Amount
17	Inquiries	Inquiries in Last 6 Months
18	Present Balance	Current balance in the borrower account
19	Gross Collection	The gross amount payable by way of Settlement or judgement in respect of the Claims, excluding any costs
20	Sub GGGrade	Sub Grant Group Grade
21	File Status	Status of the loan file
22	State	State to which borrower belong
23	Account Open	Total number of open accounts in the name of Borrower
24	Total Unpaid CL	Unpaid dues on all the other loans
25	Duration	Duration for the amount is funded to borrower
26	Unpaid Amount	Unpaid balance on the credit card
27	Reason	Reason for loan application
28	Claim Type	Amongst all Application type what is the borrower Claim Type I - Individual Account , J - Joint Account
29	Due Fee	Charges incurred if the payment on loan amount is delayed
30	Default	Target variable indicating whether loan is defaulted or not

Part One: Exploratory Data Analysis:

```
In [7]: df.columns

Out[7]: Index(['ID', 'Asst_Reg', 'GGGrade', 'Experience', 'Validation',
               'Yearly_Income', 'Home_Status', 'Unpaid_2_years', 'Already_Defaulted',
               'Designation', 'Debt_to_Income', 'Postal_Code', 'Lend_Amount',
               'Deprecatory_Records', 'Interest_Charged', 'Usage_Rate', 'Inquiries',
               'Present_Balance', 'Gross_Collection', 'Sub_GGGrade', 'File_Status',
               'State', 'Account_Open', 'Total_Unpaid_CL', 'Duration', 'Unpaid_Amount',
               'Reason', 'Claim_Type', 'Due_Fee', 'Default'],
              dtype='object')
```

Figure 2: The names of all 30 variables

```
In [3]: df.shape

Out[3]: (87500, 30)
```

Figure 3: The dataset contains 87500 rows and 30 columns

In [6]:
```
df.describe()
```

Out[6]:

	ID	Asst_Reg	Yearly_Income	Unpaid_2_years	Already_Defaulted	Debt_to_Income	Postal_Code	Lend_Amount	Deprecatory_Records
count	87500.000000	8.750000e+04	8.192500e+04	87500.000000	87500.000000	84011.000000	86111.000000	87500.000000	87500.000000
mean	83946.253509	3.798914e+06	1.348596e+05	0.332937	0.005726	30.942660	51155.385491	25920.535680	0.202400
std	36100.102950	2.289038e+06	9.882473e+04	0.876080	0.083505	14.079813	31211.229930	14433.837213	0.64641
min	21560.000000	2.484700e+04	8.800000e+03	0.000000	0.000000	0.000000	1000.000000	1710.000000	0.000000
25%	52616.750000	1.483874e+06	8.324659e+04	0.000000	0.000000	20.428800	22900.000000	15048.000000	0.000000
50%	83900.000000	4.132010e+06	1.144000e+05	0.000000	0.000000	30.139200	47700.000000	23940.000000	0.000000
75%	115261.250000	5.952909e+06	1.601600e+05	0.000000	0.000000	40.908000	80200.000000	34200.000000	0.000000
max	146559.000000	7.351847e+06	8.264031e+06	18.000000	6.000000	639.290400	99900.000000	59850.000000	86.000000

Findings: Above, we obtain some descriptive statistics about the dataset.

Figure 4: Descriptive statistics of the dataset

In [5]:
```
df.isnull().sum()
```

Out[5]:
```
ID                      0
Asst_Reg                0
GGGrade                 0
Experience              0
Validation              0
Yearly_Income        5575
Home_Status             0
Unpaid_2_years          0
Already_Defaulted       0
Designation          1414
Debt_to_Income       3489
Postal_Code          1389
Lend_Amount             0
Deprecatory_Records     0
Interest_Charged        0
Usage_Rate              0
Inquiries               0
Present_Balance         0
Gross_Collection        0
Sub_GGGrade             0
File_Status             0
State                   0
Account_Open            0
Total_Unpaid_CL      4186
Duration                0
Unpaid_Amount        4852
Reason                  0
Claim_Type              0
Due_Fee                 0
Default                 0
dtype: int64
```

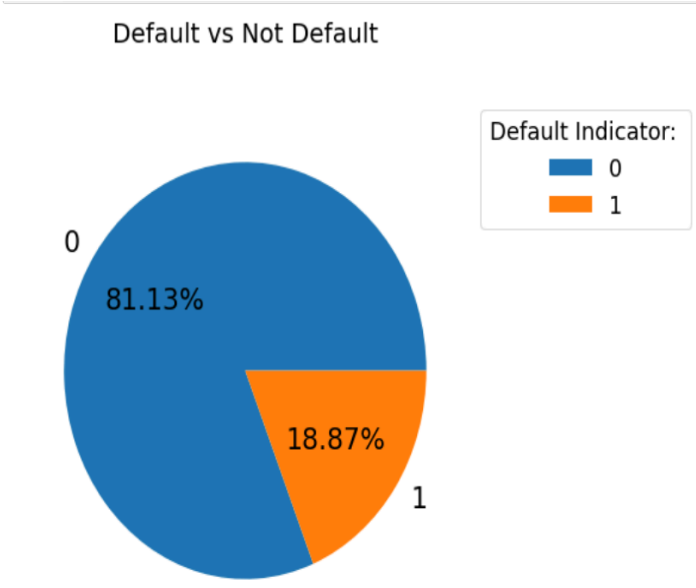
Figure 5: There are 6 columns with missing values: Yearly_Income, Designation, Debt_to_Income, Postal_Code, Total_Unpaid_CL, Unpaid_Amount

Dependent Variable:

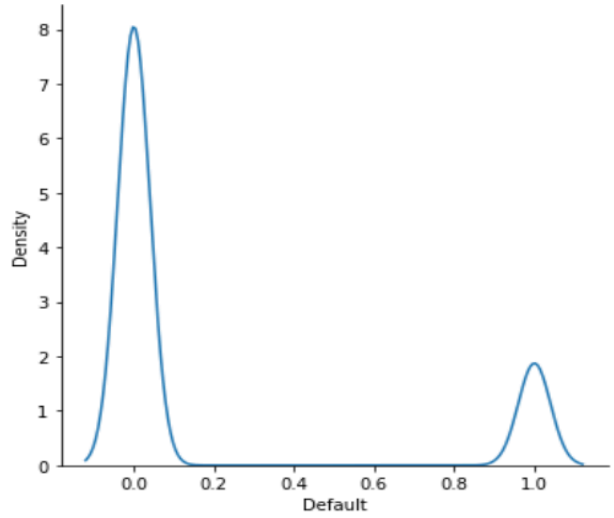
1. Default

```
In [10]: print(df['Default'].value_counts())

0    70988
1    16512
Name: Default, dtype: int64

In [11]: print(df['Default'].unique())

[0 1]
```

This is a nominal variable. 0 means "Not Default" while 1 means "Default".




```
sns.displot(data=df, x="Default", kind="kde" )
<seaborn.axisgrid.FacetGrid at 0x1ece724edc0>
```



Finding: From both diagrams, we can see that most of the loans were defaulted (1). This indicates that the dataset is imbalanced. Class imbalance can lead to overfitting, which means that it can result in a model being biased towards the class that is the majority. We will take action about this later on to make the dataset balanced.

Independent Variables:

1. ID

```
In [15]: print(df['ID'].value_counts())

133119    1
78451     1
125544    1
123497    1
129642    1
         ..
93447     1
72969     1
66826     1
68875     1
131072    1
Name: ID, Length: 87500, dtype: int64
```

```
In [16]: print(df['ID'].unique())

[ 95074 134131  92602 ... 102153 115343 104968]
```

```
In [17]: df['ID'].isna().sum()

Out[17]: 0
```

0 indicates, that there is no missing values for column unique ID

```
In [18]: print(df['ID'].isnull())

0        False
1        False
2        False
3        False
4        False
         ...
87495    False
87496    False
87497    False
87498    False
87499    False
Name: ID, Length: 87500, dtype: bool
```

```
In [19]: df['ID'].isna().sum()

Out[19]: 0
```

Findings: no missing value contain in the variable ID column

Finding: No missing values are found in this column. All the values in variable ID is unique

2. Asst_Reg (Asset Registered(

```
print(df['Asst_Reg'].value_counts())
```

```
588694     3
4772166    3
3368961    3
1090707    3
7026385    3
          ..
182899     1
7008884    1
2144863    1
2012698    1
2762946    1
Name: Asst_Reg, Length: 83966, dtype: int64
```

```
print(df['Asst_Reg'].unique())
```

```
[ 421802 3964312 4247560 ... 2624710 5966919 6637289]
```

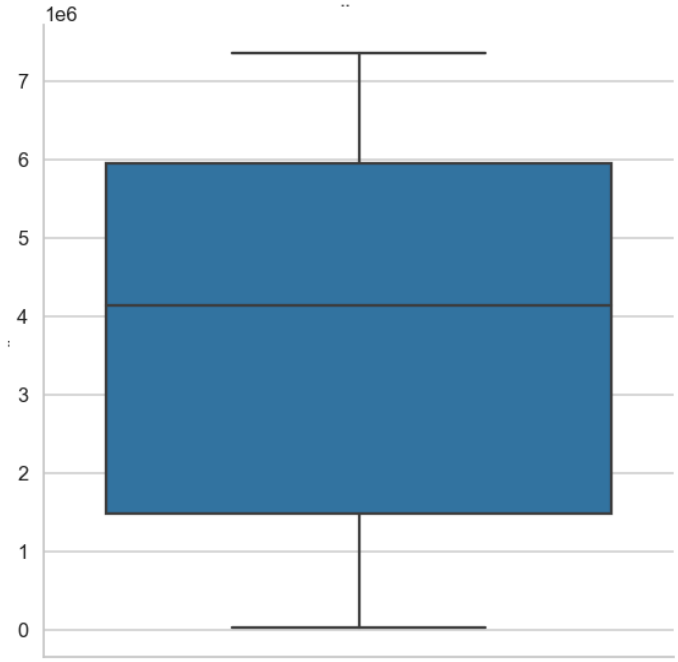
```
print(df['Asst_Reg'].isnull())
```

```
0        False
1        False
2        False
3        False
4        False
         ...
87495    False
87496    False
87497    False
87498    False
87499    False
Name: Asst_Reg, Length: 87500, dtype: bool
```

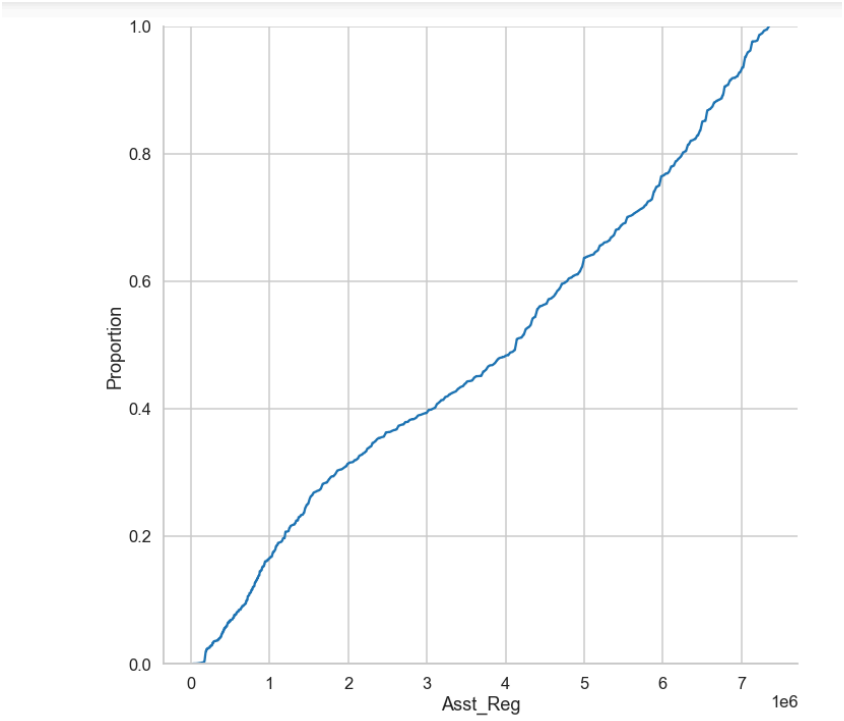
```
df['Asst_Reg'].isna().sum()
```

```
8]: 0
```

Finding: No missing values are found in this column. This shows every assist registered is unique too.



Finding: From the box-plot above, it shows that there are no outliers in this variable named Assg_Reg, which is having a 75% quartile at the 6 x 10 ^ 6 currency value and 25% quartile in between 1 and 2.



Finding: We can notice the value of assets registered increased as the proportion increased over time

3. GGGrade (Grant Group Grade)

```
In [621]: print(df['GGGrade'].value_counts())
```

```
II     24966
III    24652
I      14171
IV     13867
V       7154
VI      2192
VII      498
Name: GGGrade, dtype: int64
```

```
In [622]: print(df['GGGrade'].unique())
```

```
['II' 'IV' 'III' 'V' 'VII' 'VI' 'I']
```

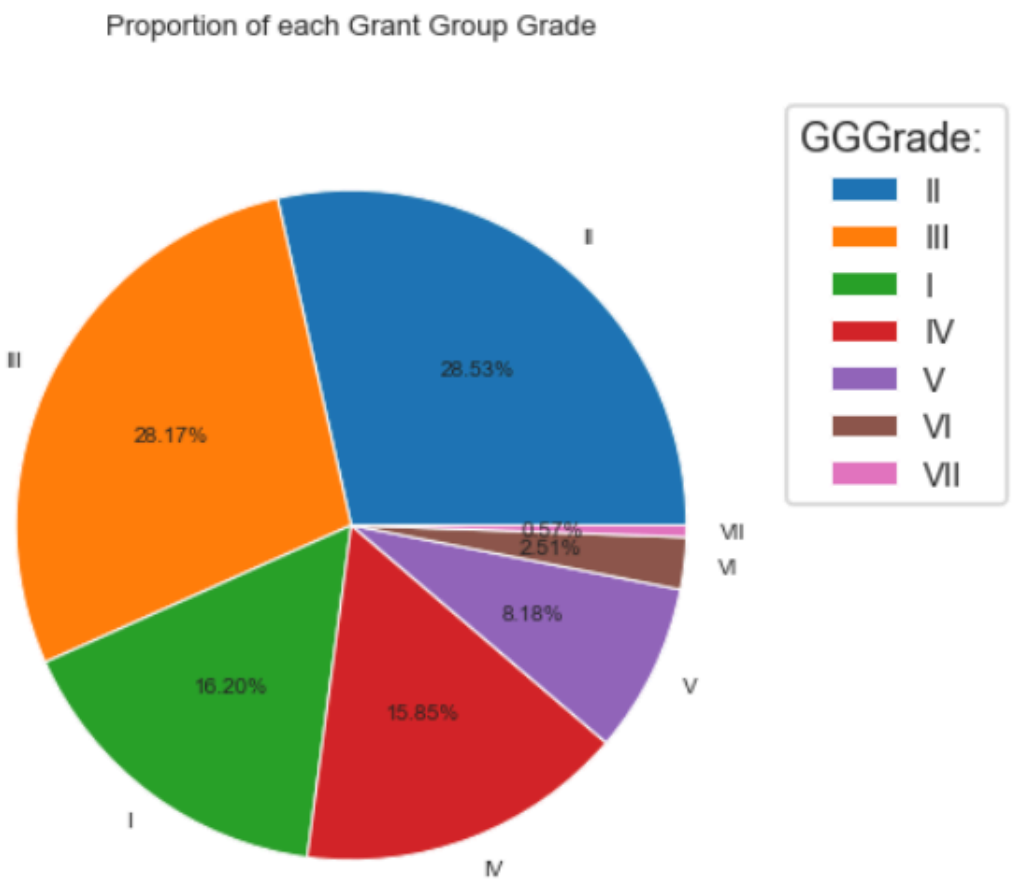
```
In [623]: print(df['GGGrade'].isnull())
```

```
0        False
1        False
2        False
3        False
4        False
         ...
87495    False
87496    False
87497    False
87498    False
87499    False
Name: GGGrade, Length: 87500, dtype: bool
```

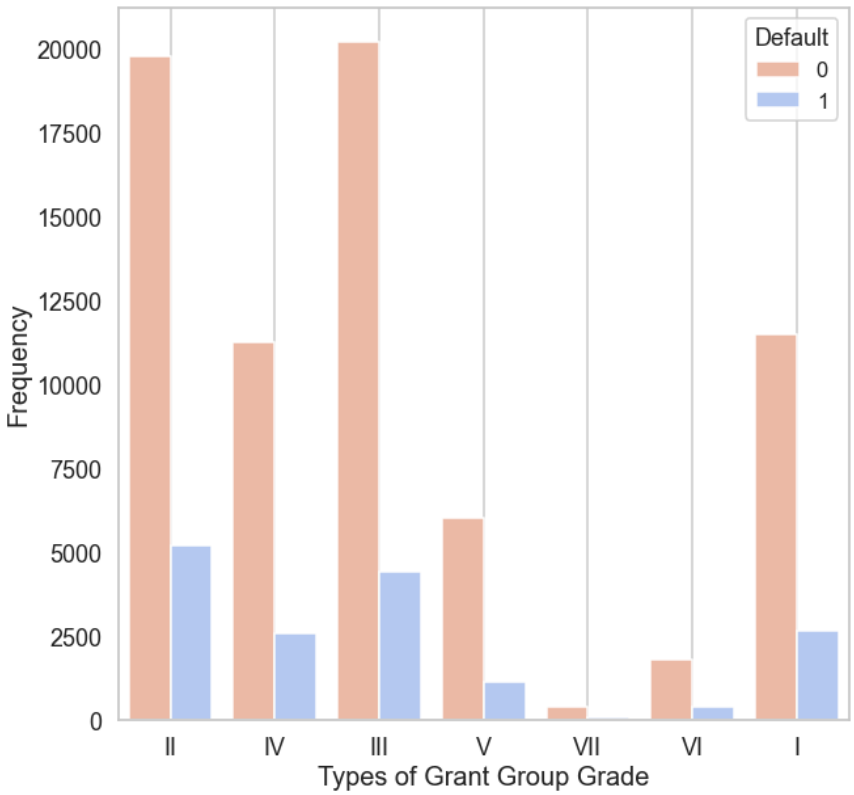
```
In [624]: df['GGGrade'].isna().sum()
```

```
Out[624]: 0
```

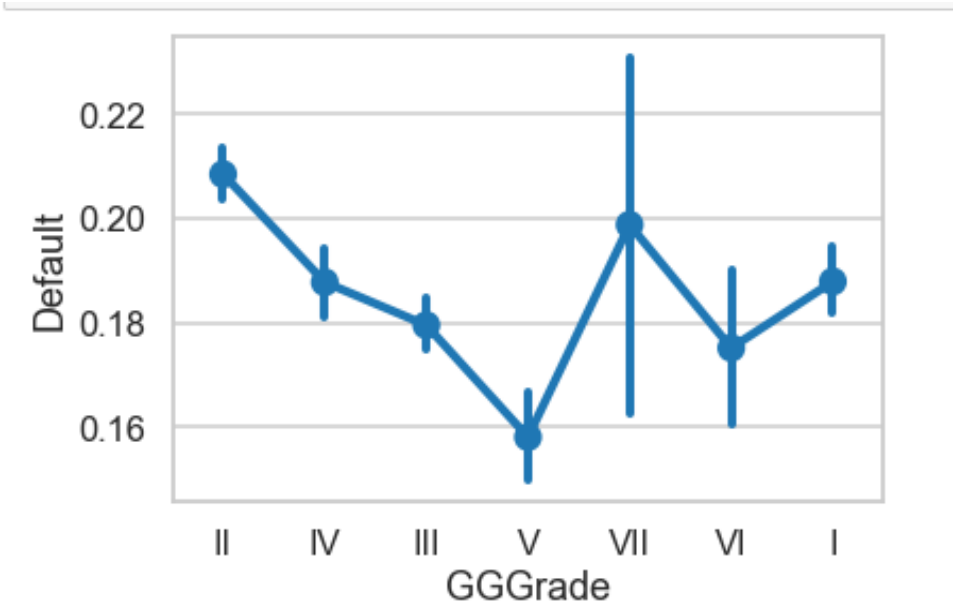
FInding: This is a categorical variable and there are no missing values found in this column.



Finding: Majority of the cases are classified as GGGrade II. GGGrade VII forms the smallest part of the dataset.



Finding: This graph shows most of the non-default cases are classified as GGGrade III. Most of the default cases are classified as GGGrade II.



4. Experience

```
print(df['Experience'].value_counts())
```

```
>10yrs    30849
2yrs       8064
3yrs       7350
<1yr       7209
1yrs       5853
5yrs       5623
4yrs       5324
7yrs       4762
8yrs       4604
6yrs       4255
9yrs       3607
Name: Experience, dtype: int64
```

```
print(df['Experience'].unique())
```

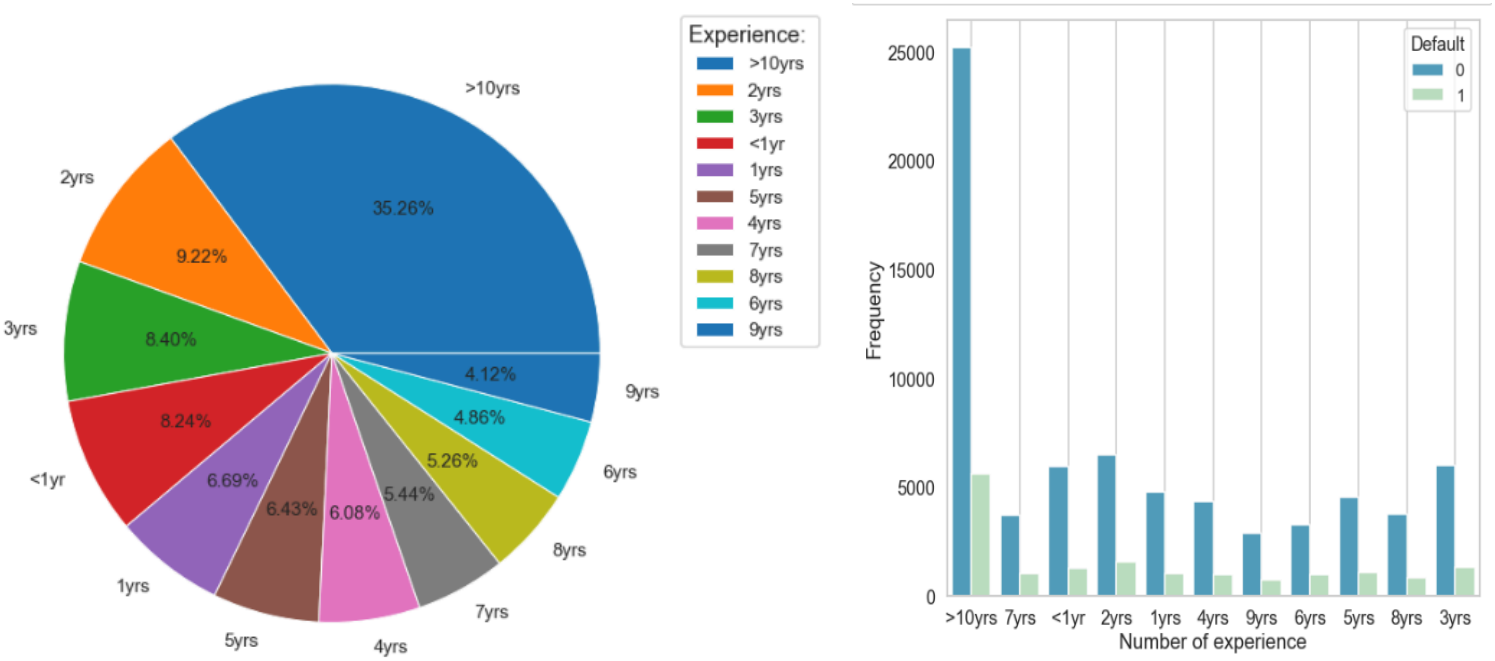
```
['>10yrs' '7yrs' '<1yr' '2yrs' '1yrs' '4yrs' '9yrs' '6yrs' '5yrs' '8yrs'
 '3yrs']
```

Finding: This is a categorical variable and the years of working experience are expressed as string rather than numeric.

```
df['GGGrade'].isna().sum()
```

```
]: 0
```

Finding: There are no missing values



Finding: According to the pie chart, majority of the loan applicants (around 35%) have more than 10 years of working experience. Very few of the loan applicants (4.12%) had 9 years of working experience. The column chart tells us that most of the defaulters have more than 10 years of working experience and most of the non-defaulters also had more than 10 years of working experience.

5. Validation

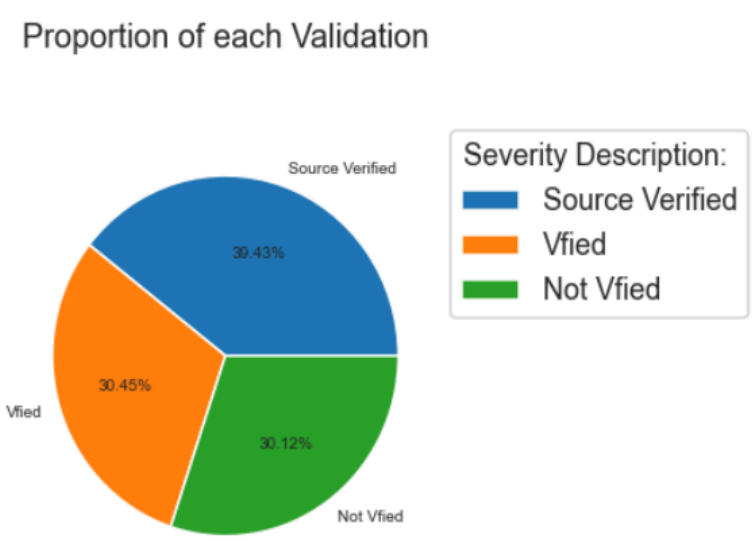
```
In [634]: print(df['Validation'].value_counts())

Source Verified    34504
Vfied              26642
Not Vfied          26354
Name: Validation, dtype: int64
```

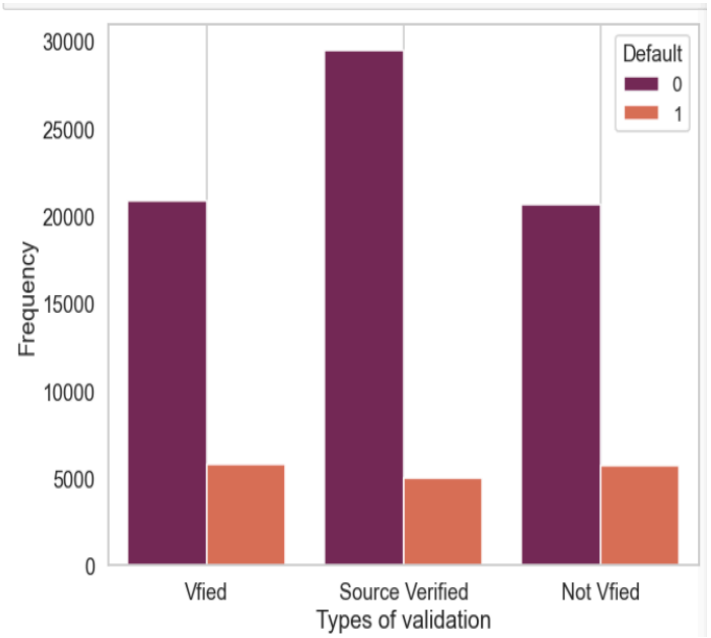
```
In [635]: print(df['Validation'].unique())

['Vfied' 'Source Verified' 'Not Vfied']
```

```
In [636]: print(df['Validation'].isnull())

0        False
1        False
2        False
3        False
4        False
         ...
87495    False
87496    False
87497    False
87498    False
87499    False
Name: Validation, Length: 87500, dtype: bool
```

```
In [637]: df['Validation'].isna().sum()

Out[637]: 0
```

Finding: This is a categorical variable with no missing values.There is some ambiguity between ‘Source Verified’ and ‘Vfied’.





Finding: Source verified forms the highest proportion among the 3 validation types. Although majority of the source verified cases are classified as non-default, this validation type has the least number of default cases.

6. Yearly Income

```
print(df['Yearly_Income'].value_counts())
```

```
105600.000    3211
88000.000     2859
114400.000    2489
123200.000    2410
70400.000     2286
              ...
197155.200       1
186945.440       1
63340.640        1
68786.432        1
81519.680        1
Name: Yearly_Income, Length: 6871, dtype: int64
```

```
print(df['Yearly_Income'].unique())
```

```
[633600.    85483.2   79200.   ... 510398.24 154191.84  90423.52]
```

```
print(df['Yearly_Income'].isnull())
```

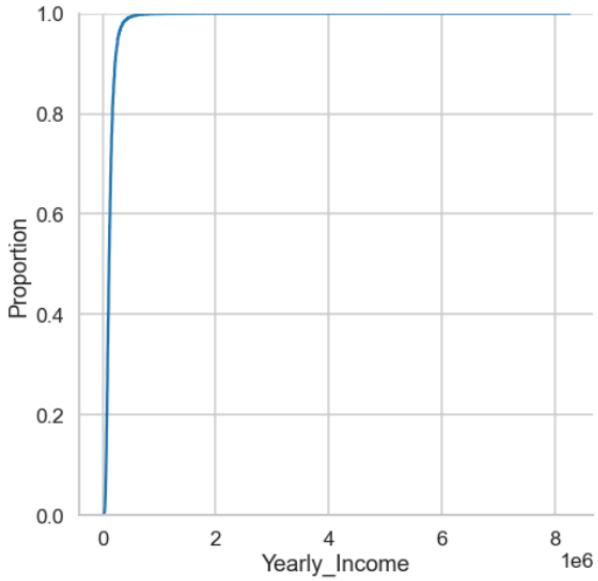
```
0        False
1        False
2        False
3        False
4        False
         ...
87495    False
87496    False
87497    False
87498    False
87499     True
Name: Yearly_Income, Length: 87500, dtype: bool
```

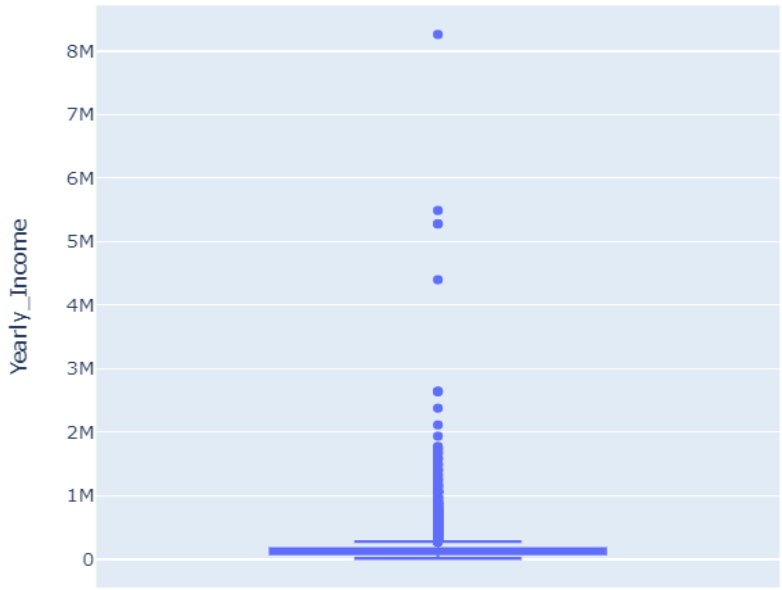
```
df['Yearly_Income'].isna().sum()
```

```
43]: 5575
```

Finding: This is a continuous variable with 5575 missing values.

644]: <seaborn.axisgrid.FacetGrid at 0x20e198ba430>





Findings: There is an exponential increase in the yearly income by the individuals as the proportion reaches to 1. It eventually becomes constant. From the box plot, we can see that there are several outliers.

7. Home Status

```
print(df['Home_Status'].value_counts())
```

```
MORTGAGE    44160
RENT        34914
OWN          8416
OTHER           6
NONE            4
Name: Home_Status, dtype: int64
```

```
print(df['Home_Status'].unique())
```

```
['MORTGAGE' 'RENT' 'OWN' 'OTHER' 'NONE']
```

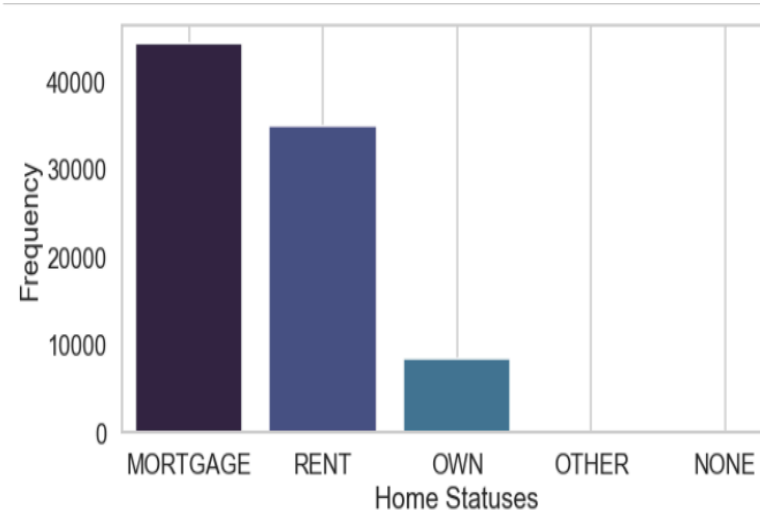
```
print(df['Home_Status'].isnull())
```

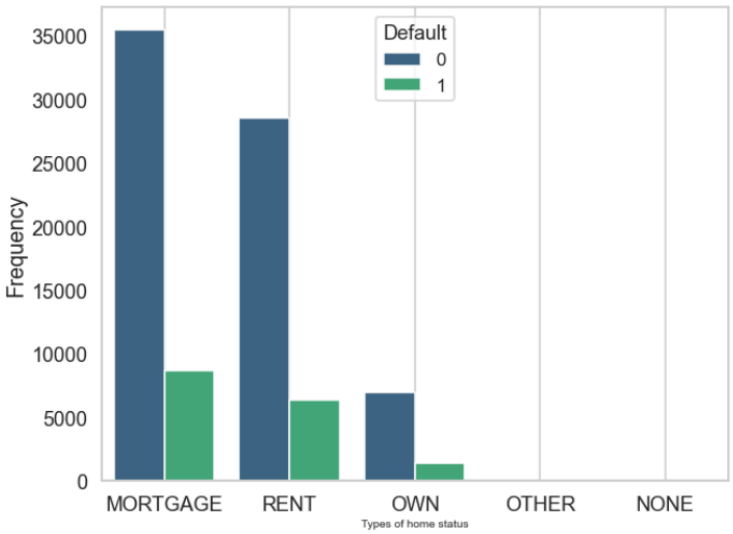
```
0        False
1        False
2        False
3        False
4        False
         ...
87495    False
87496    False
87497    False
87498    False
87499    False
Name: Home_Status, Length: 87500, dtype: bool
```

```
df['Home_Status'].isna().sum()
```

```
9]: 0
```

Findings: This is a categorical variable with no missing values.





Findings: This shows that mortgage is among the most based on individual home statuses, as rent is the second highest after mortgage and less than 10000 in frequency owns their own home. People with mortgage home status are more likely to be defaulters

9. Already Defaulted

```
print(df['Already_Defaulted'].value_counts())
```

```
0    87035
1      439
2       22
6        1
5        1
4        1
3        1
Name: Already_Defaulted, dtype: int64
```

```
print(df['Already_Defaulted'].unique())
```

```
[0 1 2 6 5 4 3]
```

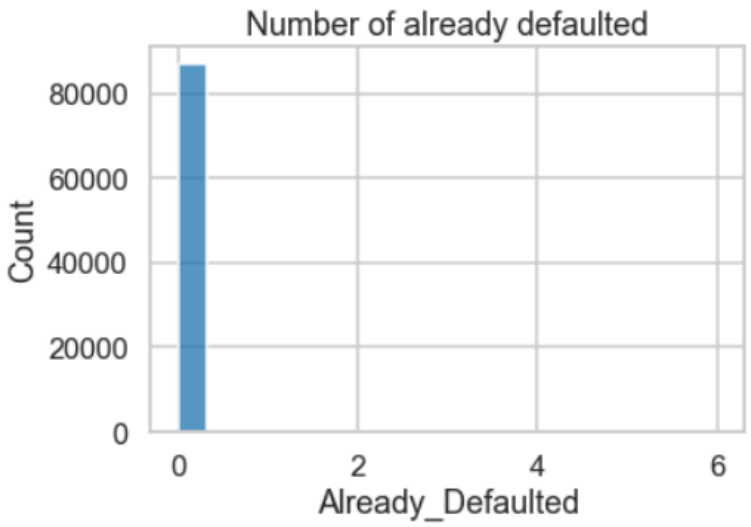
```
print(df['Already_Defaulted'].isnull())
```

```
0        False
1        False
2        False
3        False
4        False
         ...
87495    False
87496    False
87497    False
87498    False
87499    False
Name: Already_Defaulted, Length: 87500, dtype: bool
```

```
df['Already_Defaulted'].isna().sum()
```

```
0
```

Finding: This is a discrete variable with no missing values.



Findings: Majority of the individuals did not default on other loans yet, but some individuals have already defaulted 1 or 2 other loans.

10. Designation

```
print(df['Designation'].value_counts())
```

```
School Teacher                                1585
Super Lead                                    1322
Nurse                                          743
RN                                             646
Owner                                          623
                                              ...
New York University Langone Hospital             1
office manager/RENTal manager/title cler         1
builder                                          1
Associate Account Executive                      1
Rutgers University Cooperative Extension         1
Name: Designation, Length: 40348, dtype: int64
```

```
print(df['Designation'].unique())
```

```
['GLENDALE NISSAN' 'Business Teacher' 'driver' ... 'Interface Coordinator'
 'PARTS CLERK' 'Outside sales rep']
```

Finding: This is a categorical variable with many different kinds of values

```
print(df['Designation'].isnull())
```

```
0        False
1        False
2        False
3        False
4        False
         ...
87495    False
87496    False
87497    False
87498    False
87499    False
Name: Designation, Length: 87500, dtype: bool
```

```
df['Designation'].isna().sum()
```

```
'2]:  1414
```

Findings: There are 1414 missing value in Designation

11 Debt to Income

Debt to income ratio means the percentage of your gross monthly income that goes to paying your monthly debt. High debt to income is less favourable to lender, for example 15% debt to income means 15 % of your income is used to pay for debt. The higher the rate, the higher the risk to default on the loan.

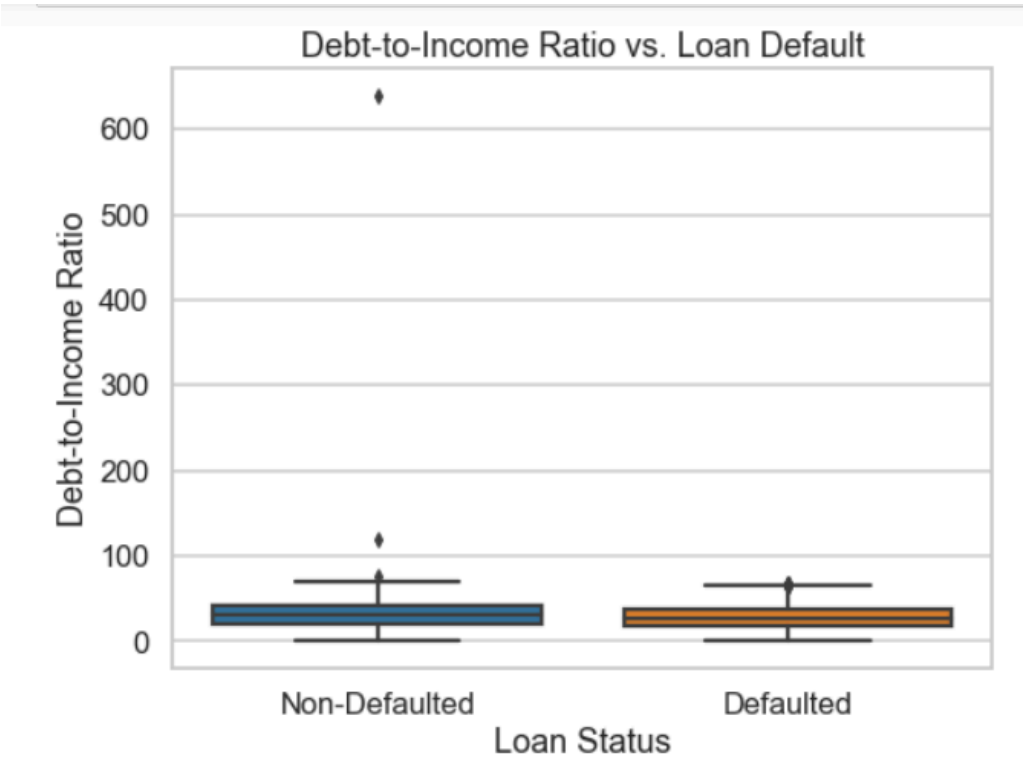
```
print(df['Debt_to_Income'].value_counts())
```

```
22.9824    61
28.2240    61
36.2880    60
24.1920    59
26.2080    59
           ..
66.6456     1
1.1256      1
1.4952      1
63.9240     1
0.9576      1
Name: Debt_to_Income, Length: 3964, dtype: int64
```

```
df['Debt_to_Income'].isna().sum()
```

```
3]:  3489
```

Finding: It is a continuous variable with missing values.




```
Average Debt-to-Income Ratio for Non-Defaulted Loans: 31.56
Average Debt-to-Income Ratio for Defaulted Loans: 28.30
```

Finding: The median value for non-default and defaulted loans does not vary; this indicates the debt to income ratio. In fact, defaulted model have slighly lower debt to income ratios. This indicates that the debt-to-income ratio alone is not a good indicator of whether a person would default their loan. The average of non-defaulted model is 31.56 which is relatively higher compared with defaulted models.

Based on the quartile it indicates that 50% of the time the debt to income ratio would rought the same for both non-default and defaulted. The boxplot is more condensed for both model.This indicate that the data is more consistent and the differences between the debt-to-income for both model is not vary.Based on the boxplot above the non-defaulted model have slighly higher debt to income ratio compare with the defaulted model.This indicate that if a person with lower debt to avearge ratio is more likely to default their loan.Lastly based on the box plot non-defaulted model have more outlier compare with defaulted model, based on the t-shape wisker the non-defaulted model have higher unusual observation which is higher than the maximum of what non-defaulted model should be.That being said the outlier for both model is resonable not too many outlier is spotted.

When using the model to make predictions both of the models in terms of debt to income ratio is approximately the same with the defaulted model with lesser variation in terms of debt-to-income ratio this indicates that the model is more consistent so it is more suitable to make predictions and more dependable.

12. Postal Code

```
print(df['Postal_Code'].value_counts())
```

```
75000.0    977
94500.0    925
11200.0    903
60600.0    842
33100.0    774
          ...
5900.0       1
9600.0       1
94200.0      1
83600.0      1
87800.0      1
Name: Postal_Code, Length: 865, dtype: int64
```

```
df['Postal_Code'].isna().sum()
```

```
81]: 1389
```

Finding: There are many possible values for this categorical variable and there are missing values.

13. Lend Amount

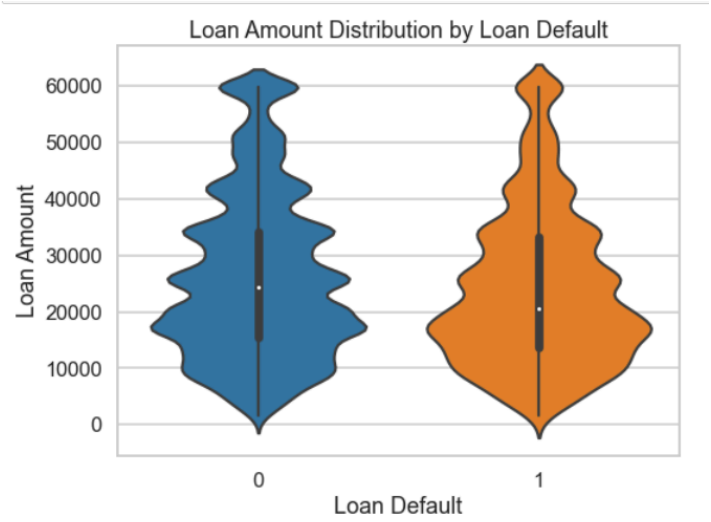
```
print(df['Lend_Amount'].value_counts())
```

```
17100.00    6117
20520.00    4945
34200.00    4878
25650.00    4836
59850.00    3832
            ...
54805.50       1
58225.50       1
53480.25       1
43220.25       1
42921.00       1
Name: Lend_Amount, Length: 1298, dtype: int64
```

```
print(df['Lend_Amount'].unique())
```

```
[42023.25 38133.   17100.   ... 56857.5  52026.75 52796.25]
```

Finding: This is a continuous variable.



Finding: 0 means not default and 1 means default. Based on the violin plot above the distribution of the data is about the same so it show that the loan amount for both default and non default is approximately the same. The density plot width for around 18,000 is the highest and around 55,000 would be the lowest for both group. The density plot width indicate frequency of each group the highest(wider) mean more people get that amount of loan. The median for the defaulted group is lower than non default group this indicates that most of the defaulted individuals would get lesser loan amount compared with non default group. As the violin plot show both groups have outlier, but the condition is not severe.

14. Deprecatory Records

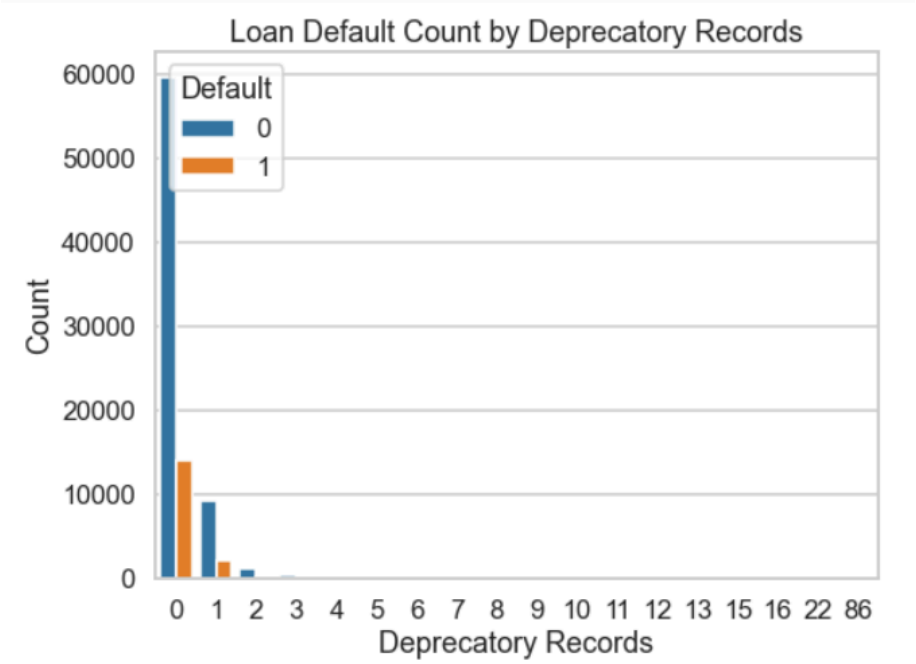
```
print(df['Deprecatory_Records'].value_counts())
```

```
0     73827
1     11354
2      1481
3       465
4       193
5        85
6        39
7        24
8        14
10        7
9         3
16        2
22        1
11        1
12        1
13        1
15        1
86        1
Name: Deprecatory_Records, dtype: int64
```

```
print(df['Deprecatory_Records'].unique())
```

```
[ 0  1  2  5  4  3  7  6  9  8 10 12 86 11 13 16 22 15]
```

Finding: This is a discrete variable.



Finding: Based on the graph, The graph is very left skewed which mean that for both default and non defaulted group they do not have any deprecatory record before. And as frequency of the deprecatory records increase frequency of individual decrease

15. Interest Charged

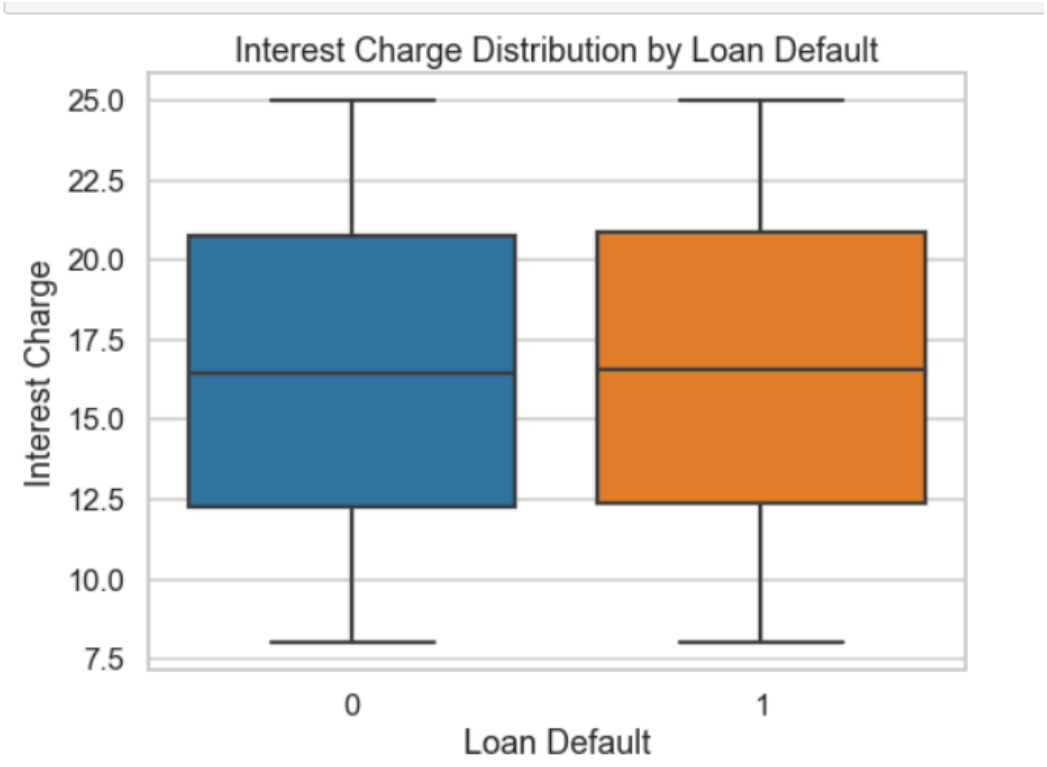
```
print(df['Interest_Charged'].value_counts())
```

```
14.56    75
15.69    74
9.53     74
21.40    73
23.65    73
         ..
19.02    34
16.20    33
9.93     31
8.00     31
25.00    25
Name: Interest_Charged, Length: 1701, dtype: int64
```

```
print(df['Interest_Charged'].unique())
```

```
[15.39  9.94 22.35 ... 24.13  9.95 22.24]
```

Finding: This is a continuous variable



Finding: This indicate that the distribution of the data is approximately the same for both default and not default group. and no outlier is spotted.

16. Usage Rate

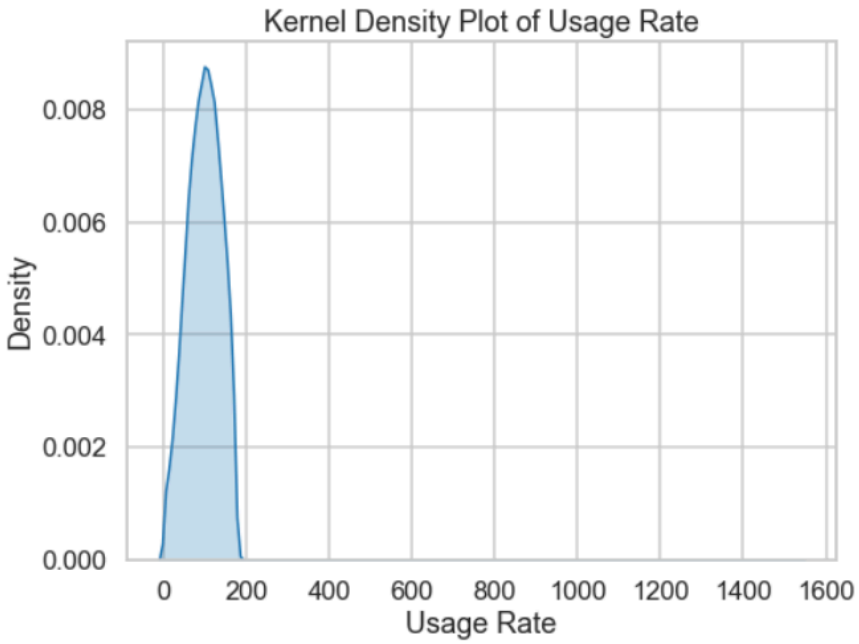
```
print(df['Usage_Rate'].value_counts())
```

```
0.000      236
113.520    210
91.160     206
99.760     205
96.320     184
          ...
179.740      1
211.560      1
183.696      1
176.300      1
210.528      1
Name: Usage_Rate, Length: 1088, dtype: int64
```

```
print(df['Usage_Rate'].unique())
```

```
[ 88.924 102.856  60.372 ... 185.76  176.3   222.74 ]
```

Finding: This is a continuous variable



Finding: The data is very left skewed this indicate that most of the respondent are receive from 0 to 200 processing fee for their loan. Since the data is very skewed thus this indicate that data cleanning is necessary to avoid prediction failure.

17. Inquiries

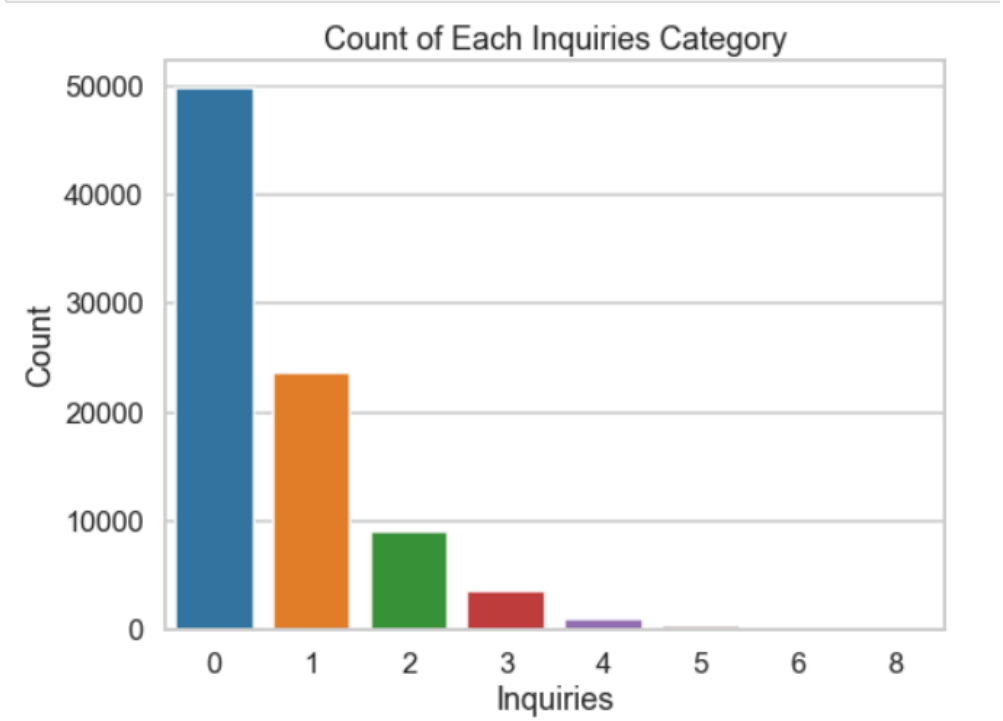
```
print(df['Inquiries'].value_counts())
```

```
0    49853
1    23601
2     9040
3     3537
4     1056
5      315
6       97
8        1
Name: Inquiries, dtype: int64
```

```
print(df['Inquiries'].unique())
```

```
[3 0 1 2 4 5 6 8]
```

Finding: This is a discrete variable.



Finding: This graph indicates that in this dataset most of the individuals do not have inquiries before. As the number of inquiries increases the frequency decreases.

18. Present Balance

```
print(df['Present_Balance'].value_counts())
```

```
0.00         14
40570.98      6
39779.40      6
22114.14      6
43530.22      6
             ..
139439.99     1
717517.17     1
91420.81      1
947796.81     1
82379.43      1
Name: Present_Balance, Length: 73819, dtype: int64
```

```
print(df['Present_Balance'].unique())
```

```
[607161.9  269234.06  22476.53 ... 102811.88  14584.11  10445.85]
```

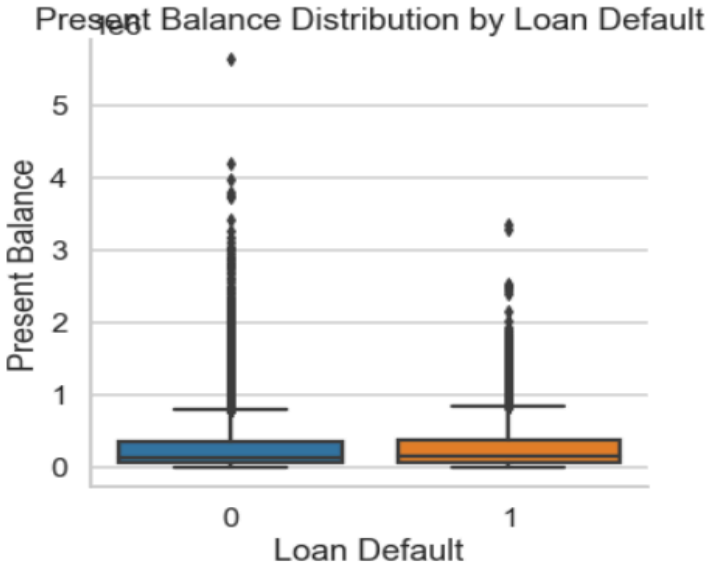
Finding: This is a continuous variable.

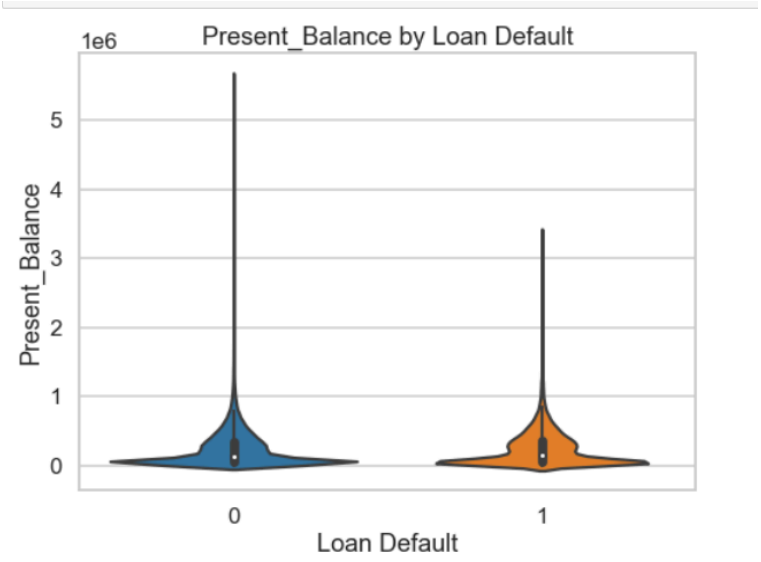
```
summary_stats = df.groupby('Default')['Present_Balance'].agg(['mean', 'median', 'std'])

# Display the summary statistics
print(summary_stats)
```

```
                  mean      median            std
Default
0       234961.128901  136044.045  255239.642408
1       242530.132637  152425.075  254185.429428
```

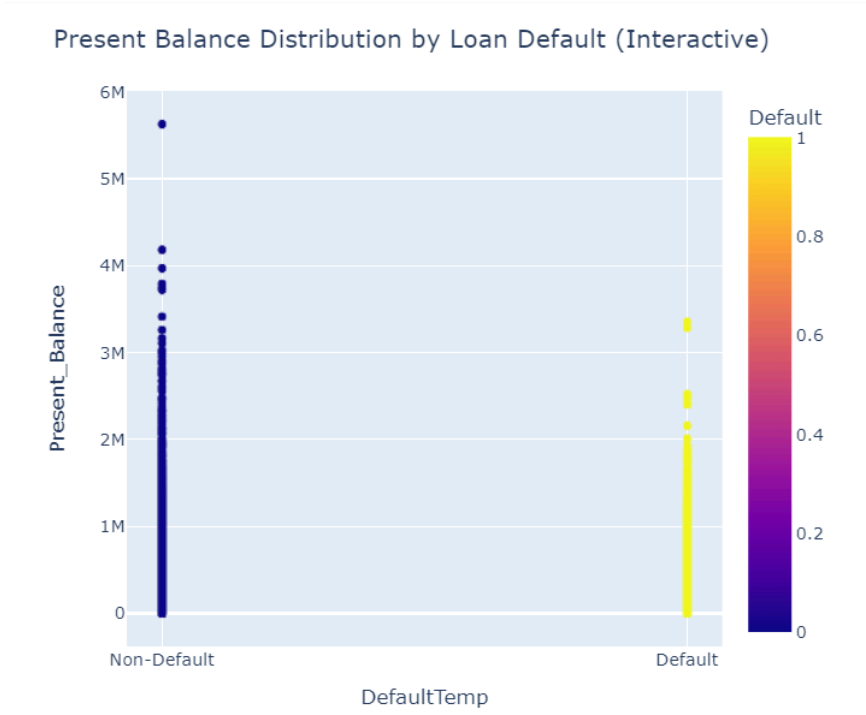
Finding: We can see some descriptive statistics, such as mean, median and standard deviation

19. Present Balance





Finding: This indicates that for both defaulted and non-default groups, most of the respondents have 0-0.2million of present balance. Data cleaning to remove outlier is necessary for both groups as the outlier is relatively prevalent in the dataset

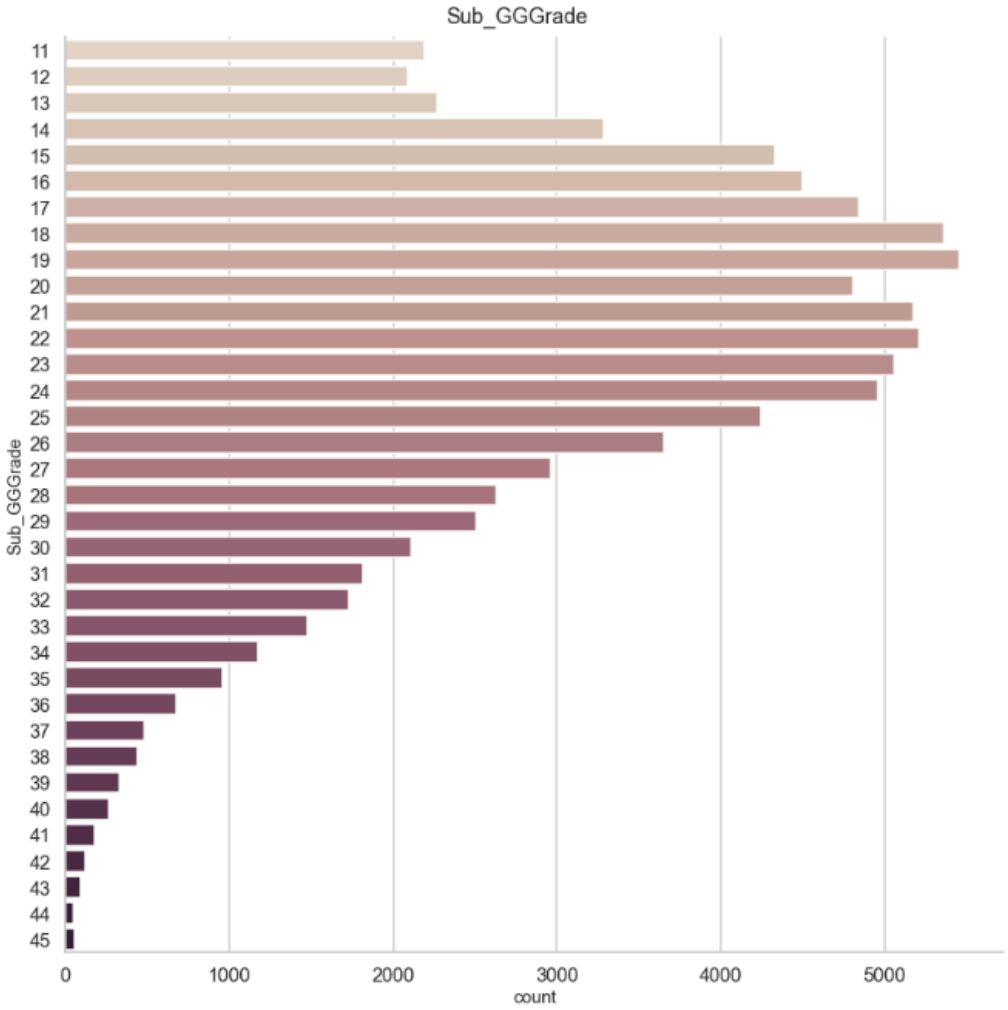


Finding: This indicates that the non-default group has the highest present balance with 5.69 million and the defaulted group highest present balance is 3.35 million.

20. Sub_GGGrade

```
print(df['Sub_GGGrade'].value_counts())
```

```
19    5454
18    5363
22    5215
21    5174
23    5063
24    4957
17    4845
20    4808
16    4496
15    4331
25    4243
26    3654
14    3290
27    2961
28    2628
29    2511
13    2271
11    2192
30    2113
12    2087
31    1818
32    1727
33    1477
34    1176
35     956
36     678
37     482
38     437
39     328
40     267
41     177
42     121
43      91
45      58
44      51
Name: Sub_GGGrade, dtype: int64
```



Finding: This indicates that most of the loans were classified as a sub grade of 19 and very few loans were categorised as sub grade. Based on the dataset most of the individual would receive a rating of 17-24

21. File Status

```
print(df['File_Status'].value_counts())
```

```
whole         46300
fully paid    41200
Name: File_Status, dtype: int64
```

```
print(df['File_Status'].unique())
```

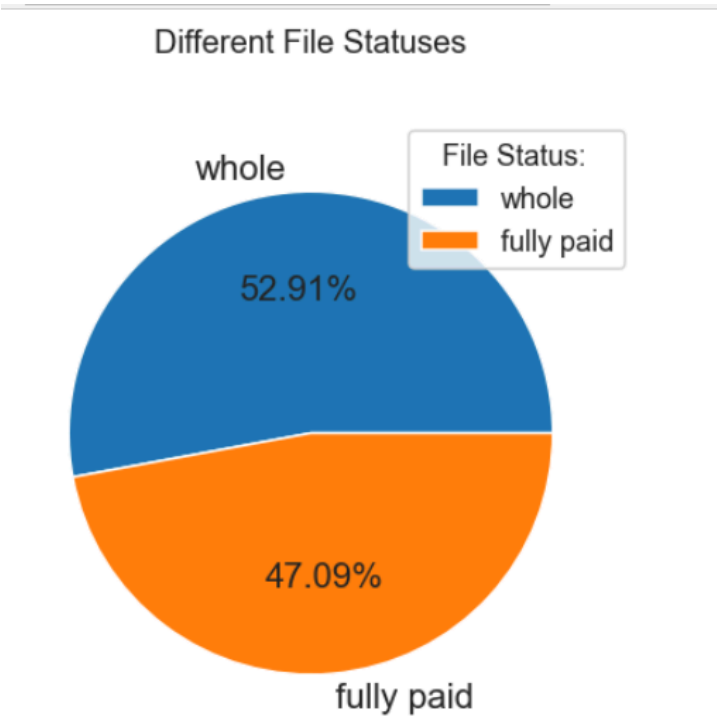
```
['fully paid' 'whole']
```

This is a nominal variable, which has 2 possible values 'fully paid' and 'whole'.

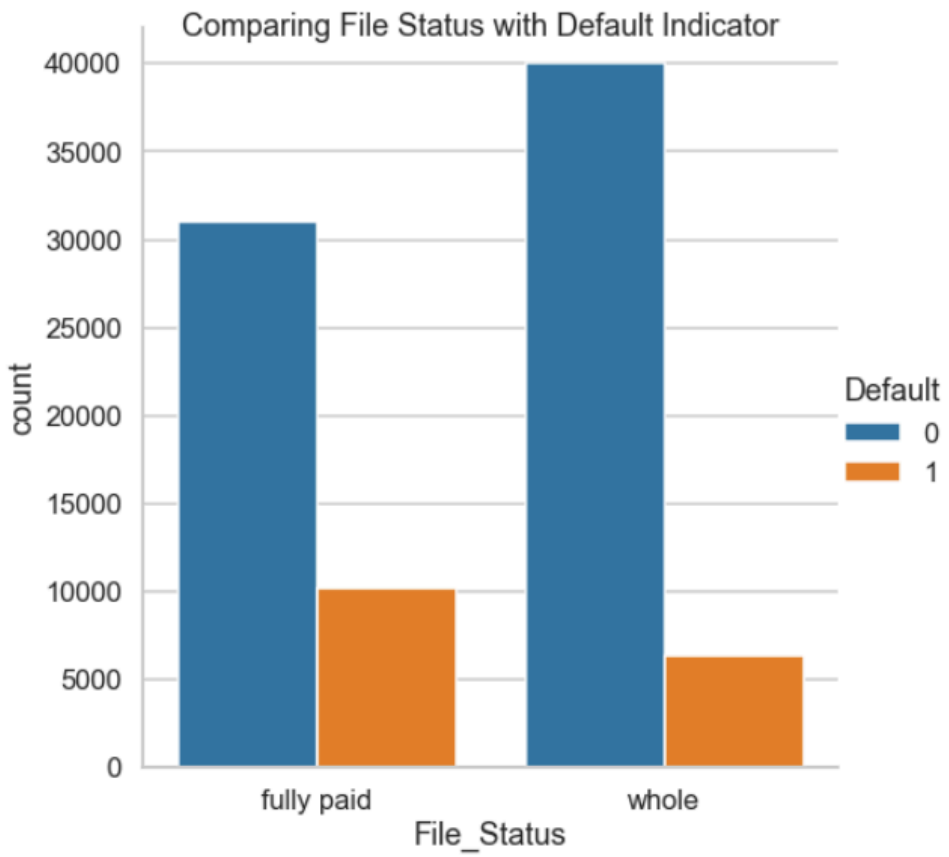
```
df['File_Status'].isna().sum()
```

```
7]: 0
```

Finding: There are no missing values in this column



Finding: Slightly more than half of the records had the 'whole' file status while slightly less than half of the records had the 'fully paid' file status .



Finding: Majority of the records in each category of file status belonged to "Not Default" cases. There were more "Not Default" cases for the "whole" file status than the "fully paid" file status. However, the "fully paid" file status had more "Default" cases than the "whole" file status.

22. State

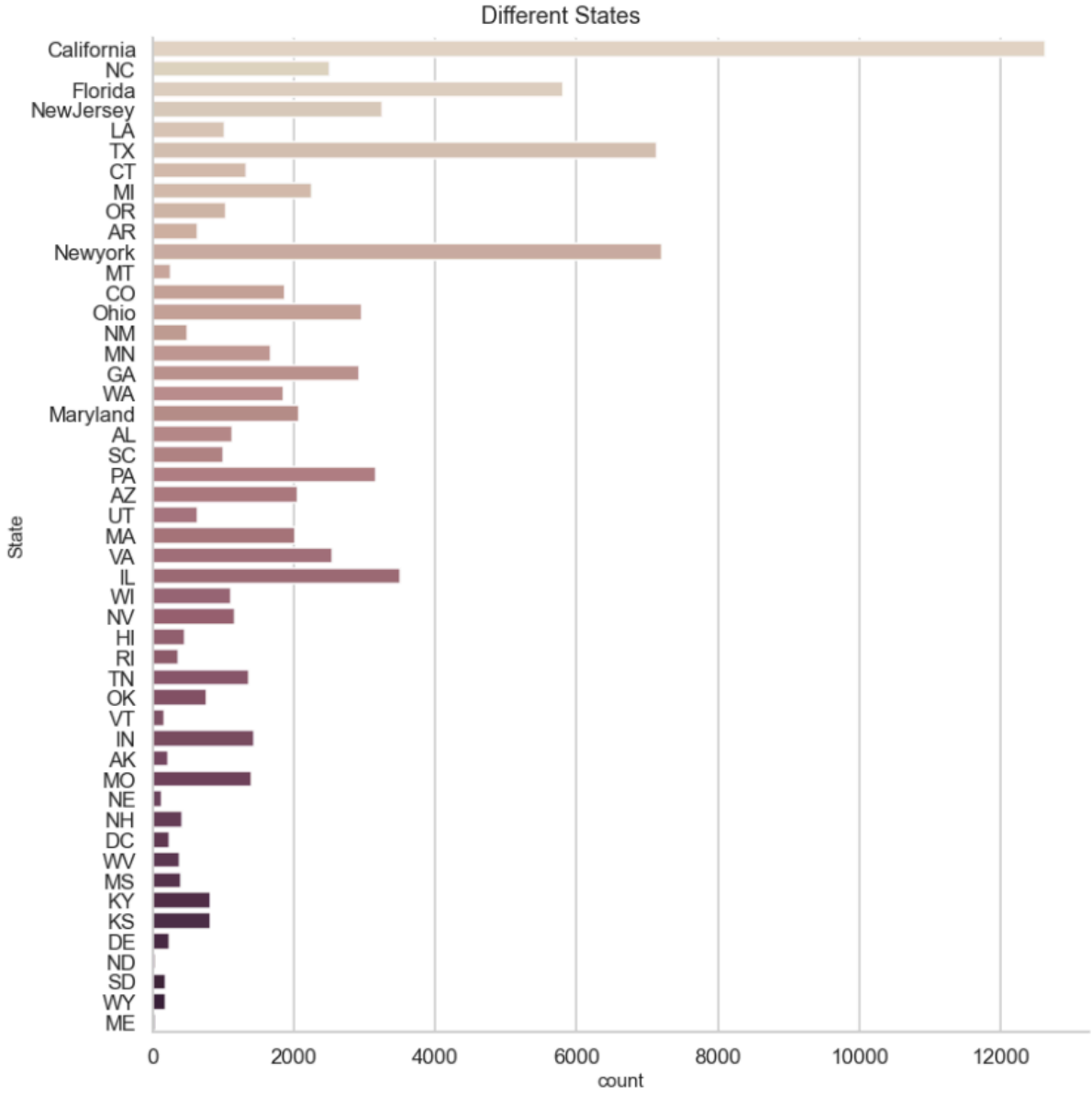
```
print(df['State'].value_counts())
```

```
California    12625
Newyork        7210
TX             7130
Florida        5813
IL             3502
NewJersey      3253
PA             3169
Ohio           2969
GA             2924
VA             2545
NC             2520
MI             2253
Maryland       2073
AZ             2058
MA             2025
CO             1882
WA             1860
MN             1673
IN             1433
MO             1410
TN             1361
CT             1325
NV             1172
AL             1140
WI             1116
OR             1050
LA             1019
SC             1005
KY              821
KS              819
OK              766
AR              650
UT              639
NM              497
HI              465
NH              427
MS              410
WV              388
RI              372
MT              259
DE              238
DC              235
AK              221
WY              186
SD              181
VT              178
NE              134
ND               50
ME               49
Name: State, dtype: int64
```

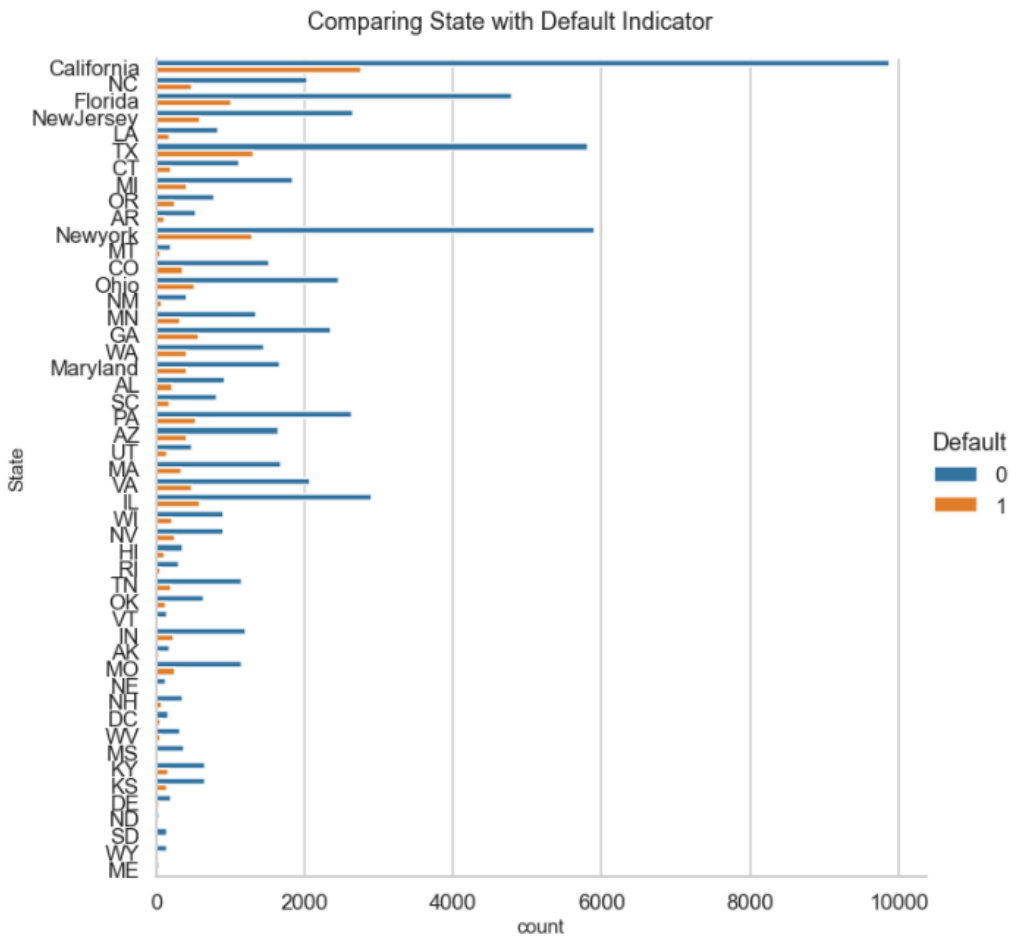
```
df['State'].isna().sum()
```

```
2]: 0
```

Finding: There are no missing values in categorical column



Finding: Majority of the borrowers came from California, followed by Newyork, TX, Florida and so on. Very few borrowers were from ND and ME.



Finding: In each state, the majority of the borrowers did not default on their loans. California has the highest number of borrowers ,who did not default on the loans, and the highest number of borrowers, who defaulted on the loans.

23. Account Open

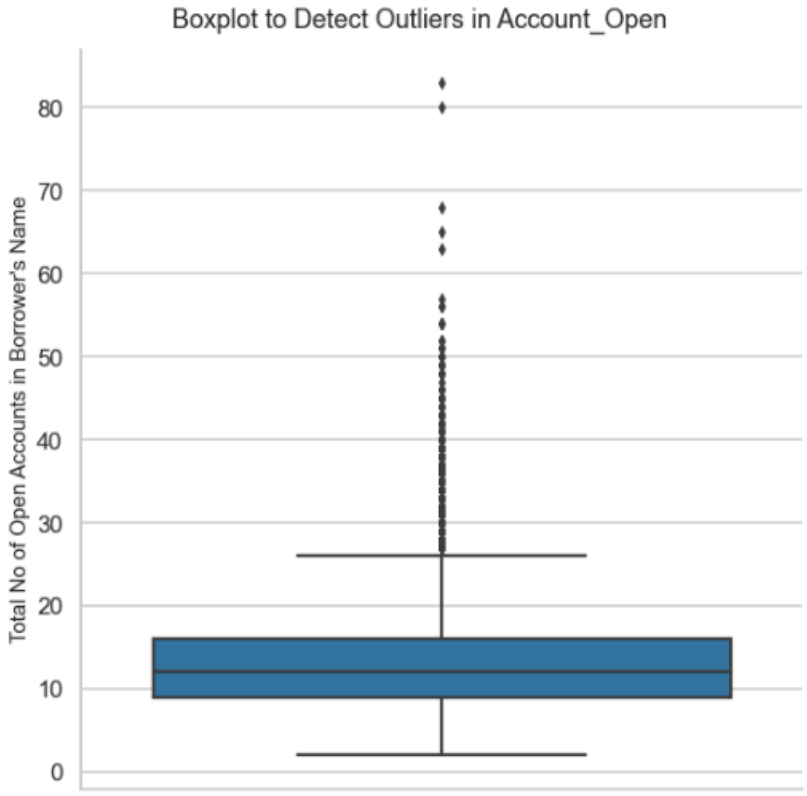
```
print(df['Account_Open'].unique())
```

```
[17 15  7  9 10 11 26  6 14  8 13 27 12 16 39 19  5 20 18 30 34 21 24  3
 29 25 23 32 28 22 44  4 68 83 31 38 33 36 37 35 43 54 48 42  2 40 63 41
 80 49 46 45 50 56 52 51 57 65 47]
```

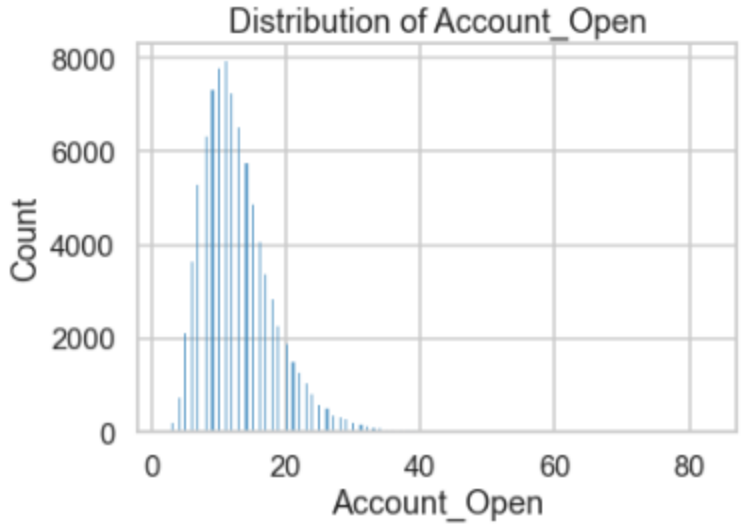
```
df['Account_Open'].isna().sum()
```

```
0
```

Finding: There are no missing values in this discrete column



Finding: There are outliers in this column, whose values are approximately above 26**.**



Finding: This column has a right-skewed distribution, indicating that there are outliers.

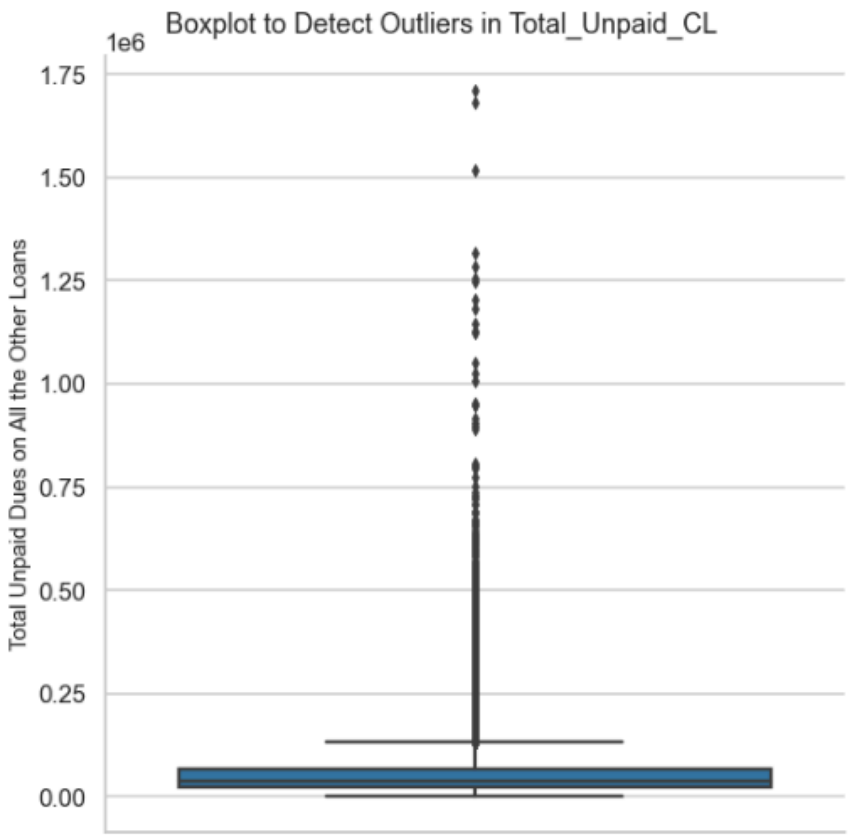
24. Total Unpaid CL

```
print(df['Total_Unpaid_CL'].value_counts())
```

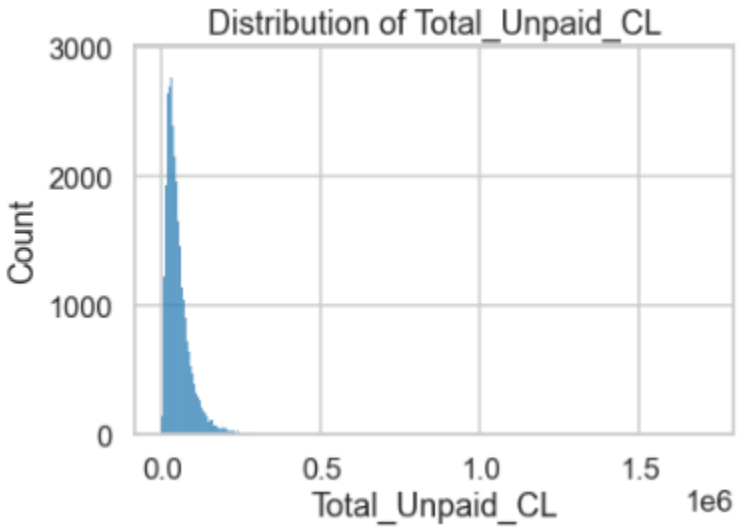
```
24070.00     288
14110.00     282
24900.00     281
19090.00     281
27390.00     271
            ...
13097.40       1
216269.78      1
117574.48      1
61454.86       1
35067.50       1
Name: Total_Unpaid_CL, Length: 4385, dtype: int64
```

```
df['Total_Unpaid_CL'].isna().sum()
```

```
4186
```



Finding: There are many outliers in this column, whose values are approximately more than 125000



Finding: This column has a right-skewed distribution, indicating that there are outliers.

25. Duration

```
print(df['Duration'].value_counts())
```

```
3 years    60061
5 years    27439
Name: Duration, dtype: int64
```

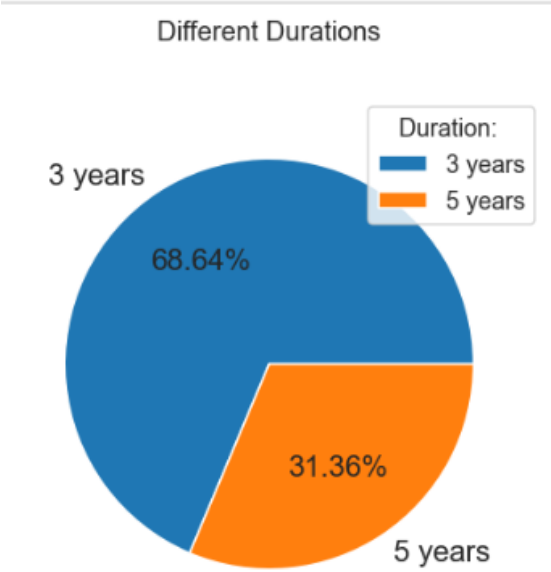
```
print(df['Duration'].unique())
```

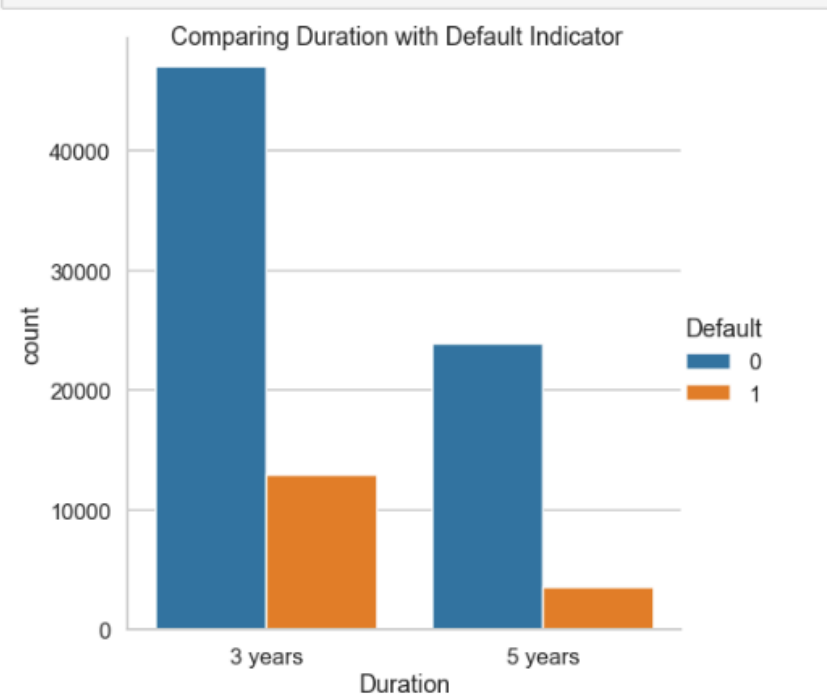
```
['3 years' '5 years']
```

```
df['Duration'].isna().sum()
```

```
0
```

Finding: This is a categorical variable with no missing values.





Finding: Most of the records (68.84%) had the duration of 3 years for the amount funded to borrower while the remaining records (31.36%) had a duration of 5 years for the amount funded to borrower. Majority of the records in each category of Duration belonged to "Not Default" cases. The number of "Not Default" cases for the "3 years" duration is approximately twice the number of records for the "5 years" duration.

26. Unpaid Amount

```
print(df['Unpaid_Amount'].value_counts())
```

```
0.00        181
11407.41     14
25915.05     13
12907.08     12
15514.83     12
            ...
47098.53      1
42873.12      1
47.88         1
68088.78      1
58694.04      1
Name: Unpaid_Amount, Length: 34679, dtype: int64
```

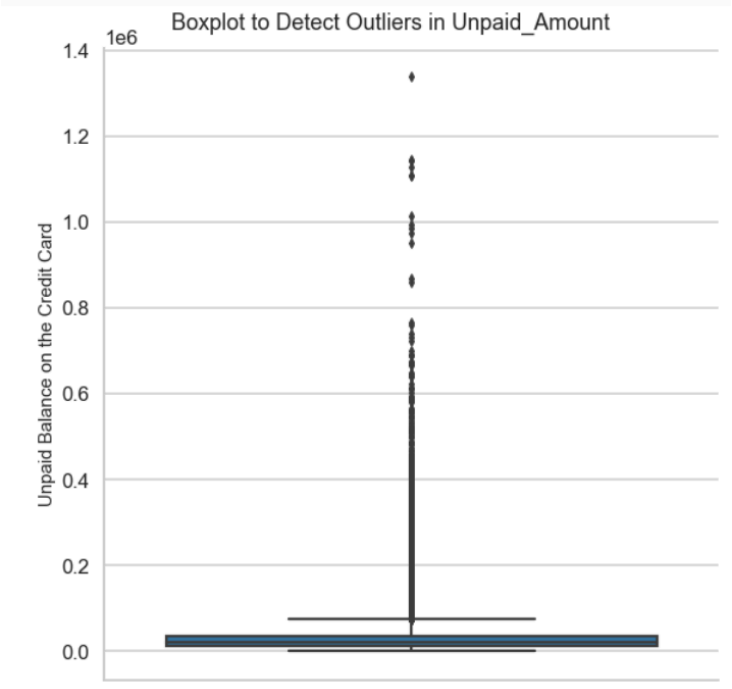
```
print(df['Unpaid_Amount'].unique())
```

```
[31216.05 11660.49  5637.87 ... 82601.55 82056.06 52676.55]
```

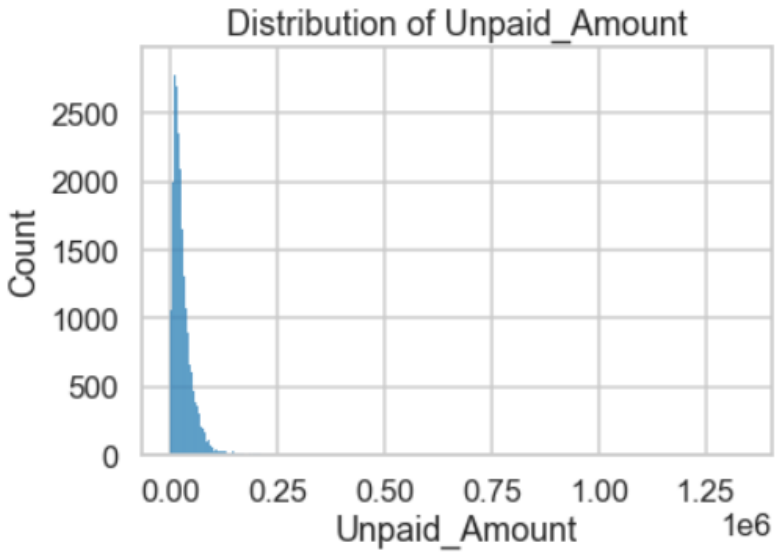
```
df['Unpaid_Amount'].isna().sum()
```

```
]: 4852
```

FInding: This is a continuous variable with missing values



Finding: There are many outliers in this column, whose values are approximately more than 50000



Finding: This column has a right-skewed distribution, indicating that there are outliers.

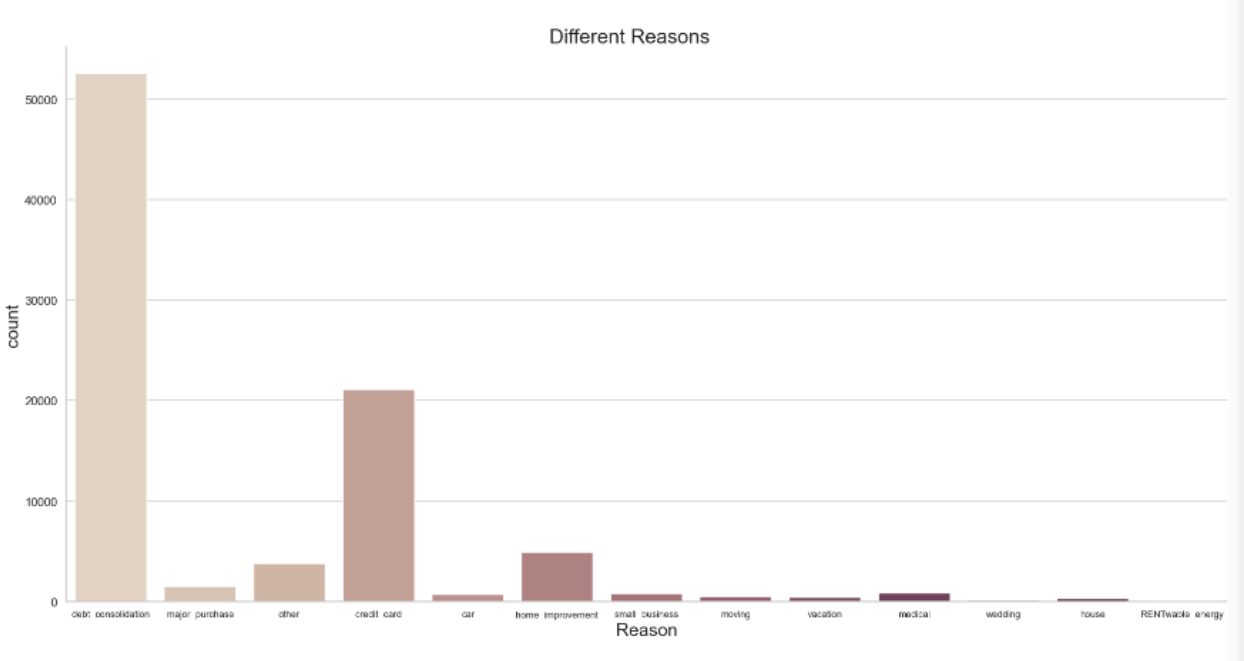
27. Reason

```
print(df['Reason'].value_counts())
```

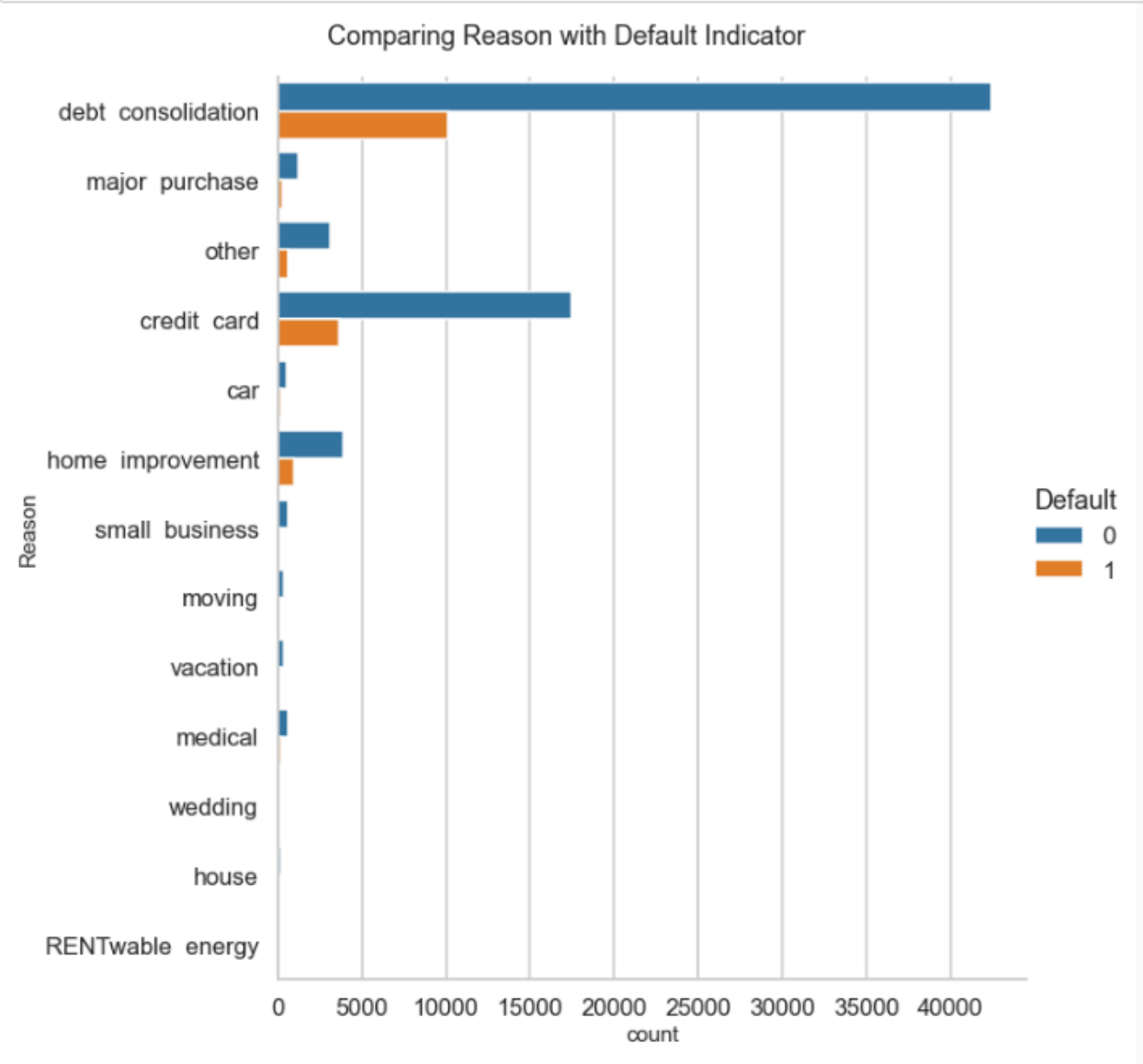
```
debt  consolidation    52559
credit  card           21102
home  improvement       4886
other                    3774
major  purchase         1512
medical                   821
small  business           751
car                       710
moving                    503
vacation                  424
house                     311
wedding                   109
RENTwable  energy          38
Name: Reason, dtype: int64
```

```
df['Reason'].isna().sum()
```

```
0
```



Finding: It is a categorical variable with no missing values. Most of the loans were applied because of debt consolidation, followed by credit card and so on. Very few loans were applied due to RENTwable energy. It can be seen that for every kind of reason, most of the loans were not defaulted. The debt consolidation category has the highest number of records for both "Not defaulted" and "Defaulted" cases. The RENTwable energy category has the least number of records for both "Not defaulted" and "Defaulted" cases.


Comparing Reason with Default Indicator
debt consolidation
major purchase
other
credit card
car
home improvement
small business
moving
vacation
medical
wedding
house
RENTwable energy
Reason
0
5000
10000
15000
20000
25000
30000
35000
40000
count
Default
0
1

28. Claim Type

```
print(df['Claim_Type'].value_counts())
```

```
I    87454
J       46
Name: Claim_Type, dtype: int64
```

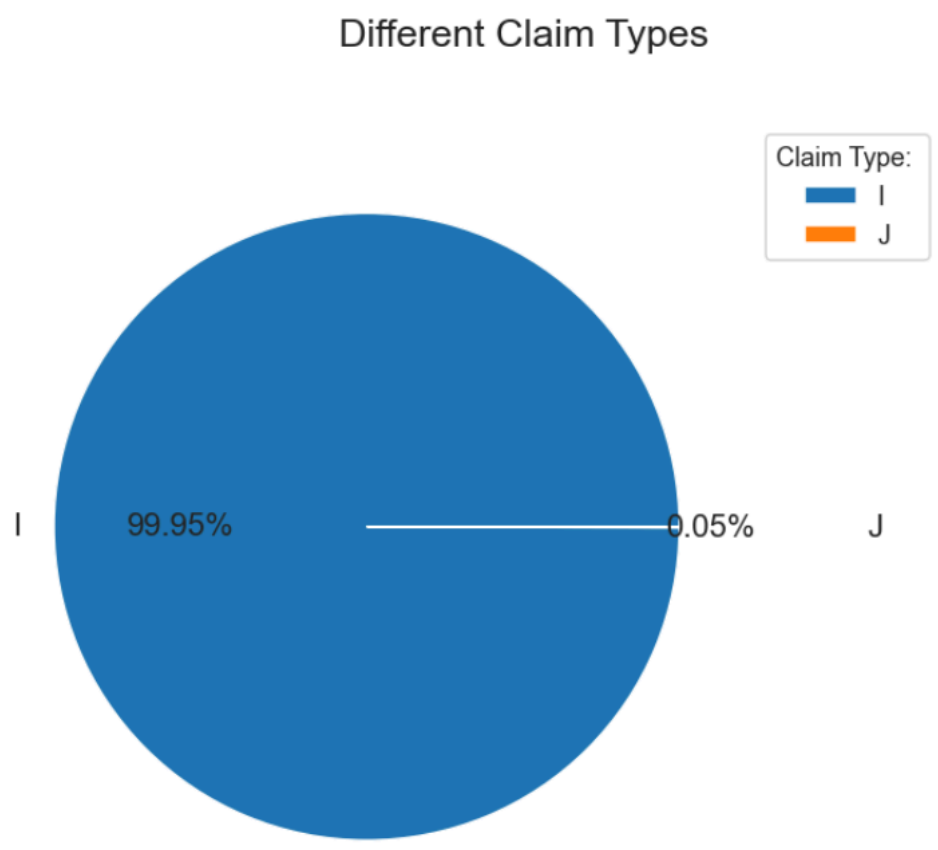
```
print(df['Claim_Type'].unique())
```

```
['I' 'J']
```

```
df['Claim_Type'].isna().sum()
```

```
]: 0
```

Finding: It is a categorical variable with no missing values



Finding: We can see that almost all of the loans are of the Individual Account claim type (99.95%). Less than 1% of the loans are of the Joint Account claim type.

29. Due Fee

```
print(df['Due_Fee'].value_counts())
```

```
0.00     86511
15.00      275
30.00       28
44.03        4
20.50        4
         ...
29.54        1
30.14        1
36.72        1
25.02        1
71.00        1
Name: Due_Fee, Length: 608, dtype: int64
```

```
df['Due_Fee'].isna().sum()
```

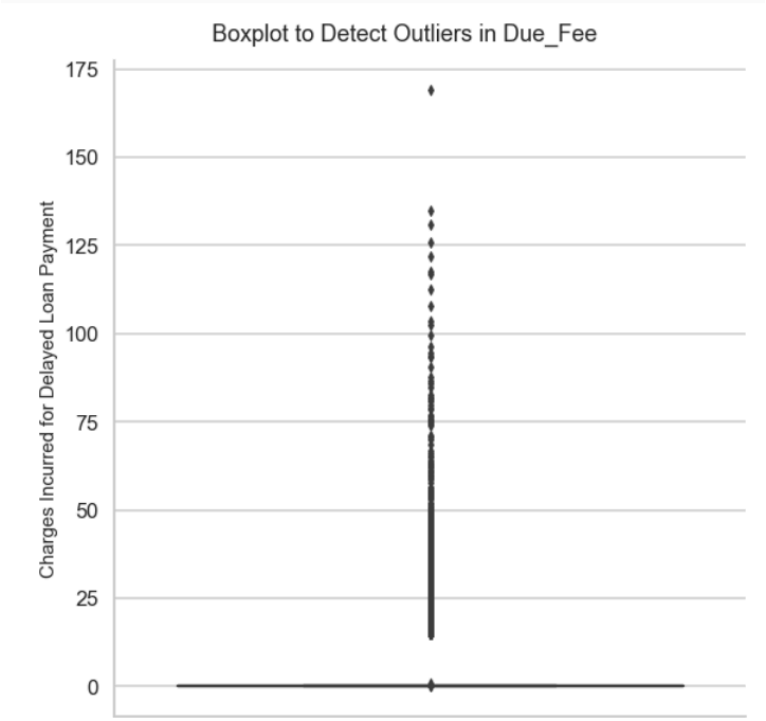
```
50]: 0
```

There are no missing values in this column.

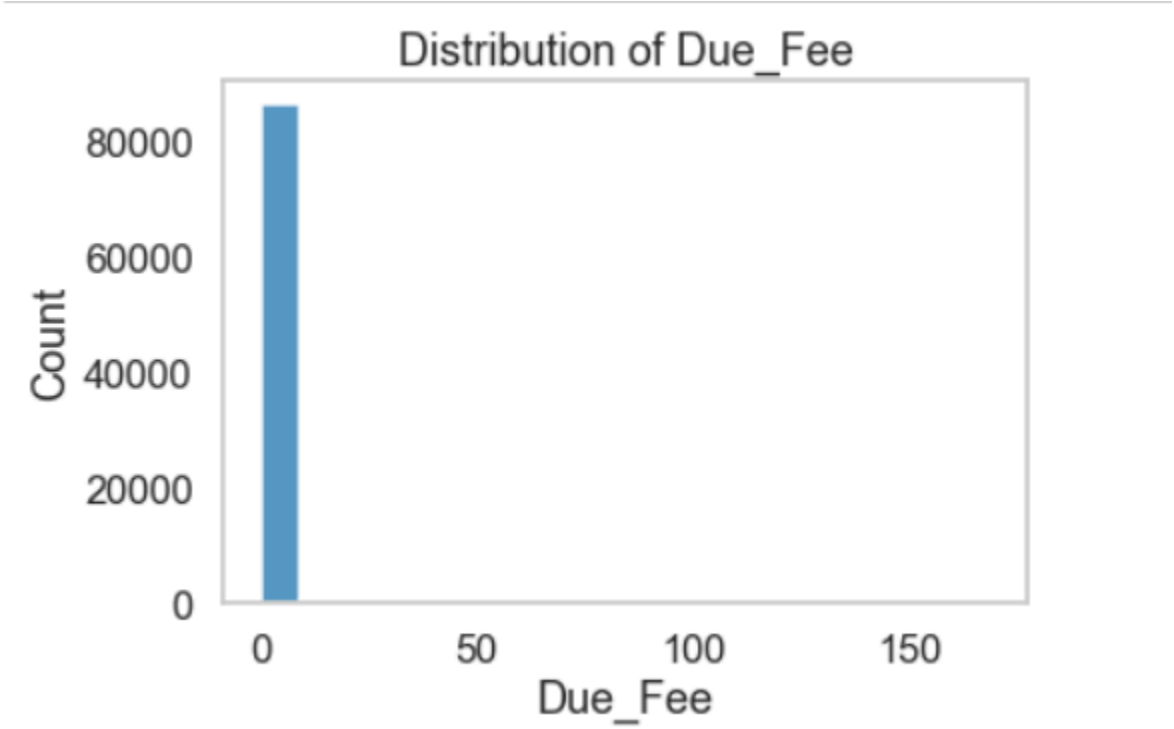
```
df['Due_Fee'].describe()
```

```
51]: count    87500.000000
     mean         0.316407
     std          3.546087
     min          0.000000
     25%          0.000000
     50%          0.000000
     75%          0.000000
     max        169.050000
     Name: Due_Fee, dtype: float64
```

Finding: This is a continuous variable with no missing values.



Finding: Although it is obvious that there are many outlier values in this column, it is not easy to interpret the boxplot. So, we will rely on the histogram below.



Finding: This column has a right-skewed distribution, indicating that there are outliers. We can see that most of the due fees lie within the 0 to 6.25 bin. Very few due fees lie within the 6.25 to 37.5 bin.

3.2 Applications of the algorithm(s)

3.2.1 Algorithms

The algorithms selected for this project are all used to make a prediction for our loan default dataset.

a) XGBoost

We ran the XGBClassifier in 2 configurations:

(1) with Default Hyperparameters

- gamma = 0 (regularization)
- learning_rate = 0.1
- max_depth = 6
- subsample = 1

(2) with Custom Hyperparameters

- gamma = 2
- learning_rate = 0.1
- max_depth = 7
- subsample = 0.7

b) Artificial neural networks

For this project assignment, we are using the Sci-Kit Learn Multi-Layer Perceptron library, more specifically, the MLPClassifier to perform binary classification for predicting loan default. We ran the MLPClassifier in 2 configurations:

(1) with Default Hyperparameters

- max_iter = 200 (Max Number of Iterations)
- alpha = 0.0001 (L2 Regularization Term)
- hidden_layer_sizes = (100) (1 hidden layer with 100 nodes)

(2) with Custom Hyperparameters

- max_iter = 500
- alpha = 0.05

- hidden_layer_sizes = (100, 100, 100) (3 hidden layers with 100 nodes each layer)

c) SVM

Random forest with Default Hyper parameter

Employed Support Vector classification from sci-kit learn
Ability to handle versatility of kernel function and high-dimensional data.
Kernel : rbf
C : 1.0
Degree : 3
Gamma :scale

d) Random forest

Random forest with Hyper parameter

Utilise Random ForestClassifier with randomizedSearchCV From Sci-kit learn Library and also Sci-kit learn library.
Configuration
n_estimators :10-200
max_features : number of features to consider when choosing best split “auto”,”sqrt” and “log2”
Max_dept : maximum dept of each tree (1-20)
Min_sample_split :2-20
Min_sample_leaf :1-20
Bootstrap : True and false

Randomized search Setup:
Iteration : 100 and 5 fold cross-validation
N_jobs =-1 (speed up computing process)
Reproducibility set to random to 42

Random forest without Default Hyper parameter

Utilise Random ForestClassifier From Sci-kit learn Library.

Reason

It is less likely to prone to overfitting. It can handle higher dimensionality and determine most significant variable among variables.
Configuration
This is ran by default hyperparameter

N_estimators : 100
Criterion: gini
Max_dept = none
Min_samples_split:2
min_samples_leaf:1

e) AdaBoost

AdaBoost with default hyper parameter

```
In [286]: from sklearn.ensemble import AdaBoostClassifier
          from sklearn.metrics import classification_report
          from sklearn.metrics import accuracy_score, precision_score, f1_score, recall_score

          adab = AdaBoostClassifier(random_state=0)
          model = adab.fit(trainFeatures, trainLabels)

          y_pred_adab = model.predict(validationFeatures)
```

Explanation: In this code, it shows the default way to fit a model using adaBoost. As shown in the picture above, the minimum random_state is 0. But the user can edit the value of the random state based on the scenario and how the user wants to control the randomness of each seed based on the estimator. Another way of using the random_state is for machine learning to do reproductivity. Apart from that, we need the ada boost classifier function that is pre-programmed in sklearn library.

AdaBoost with tuned hyper parameter

```
from sklearn.model_selection import GridSearchCV

# Defininining the parameter grid
param_grid = {
    'n_estimators': [10, 50, 100, 250, 500, 1000],
    'learning_rate': [0.001, 0.01, 0.1, 0.5, 1]
}

# Initialize AdaBoost with the random_state parameter for reproducibility
adab = AdaBoostClassifier(random_state=0)

# Initialize GridSearchCV
grid_search = GridSearchCV(adab, param_grid, cv=5, scoring='accuracy', n_jobs=-1)

# Fit the model
grid_search.fit(trainFeatures, trainLabels)

# Print the best parameters
print(grid_search.best_params_)
```

```
{'learning_rate': 0.5, 'n_estimators': 1000}
```

Explanation: In the pic shown above, this is to find the best parameter for ada boost by using grid search cv as one of the methods to find the best parameter for adaBoost. As we explain from the top, as shown we declared the param_grid which contains the **n_estimators** and **learning_rate**. Where n_estimators is the maximum number of estimators which boosting is suspended. According to sklearn the default value for sklearn is 50. If the model can fit perfectly the learning process will stop early depending on the dataset. As for the learning_rate it actually depends on the weight of the classifier that has been applied at each boosting algorithm. As the higher the learning rate, the higher is the contribution of each classifier. As for learning_rate default value is 1.0. But both n_estimator and learning_rate must be in numeric values and cannot be in negative numeric values. As the picture shows the best parameters for this model is learning rate of 0.5 and n_estimator of 1000. And inside the parameter of grid search we use the values to determine to allow grid search to test possibly combinations that best fits our model for this dataset. For more information of grid search parameter we put under the Linear Discriminant analysis tuned hyper parameter section.

```
: from sklearn.ensemble import AdaBoostClassifier

  #default value for n_estimators is 50
  #maximum value for learning_rate is 1

  adab = AdaBoostClassifier(n_estimators=1000, learning_rate=0.5, random_state=0)
  model = adab.fit(trainFeatures, trainLabels)
```

```
: y_pred_adaboost = model.predict(validationFeatures)
  print(y_pred_adaboost)
```

```
[1 0 1 ... 0 1 1]
```

Explanation: As shown above, since the grid search cv has been applied we use the parameters to find the accuracy and we could compare it with the default hyper parameters.

```
: from sklearn.metrics import ConfusionMatrixDisplay
  from sklearn.metrics import confusion_matrix

  cf = confusion_matrix(validationLabels, y_pred_adaboost, labels=model.classes_)

  disp = ConfusionMatrixDisplay(confusion_matrix=cf, display_labels=model.classes_)

  disp.plot(cmap='Blues')

  plt.grid(False)

  plt.show()
```

Explanation: As shown here, this picture is to generate the confusion matrix to allow users to understand more about the result as this confusion gives user more information about the true positive and false negative values.

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

y_prob = model.predict_proba(validationFeatures)[:,1]

fpr, tpr, threshold = roc_curve(validationLabels, y_prob)

roc_auc = auc(fpr, tpr)

plt.figure(figsize=(10, 10))
plt.plot(fpr, tpr, color='darkorange', lw=2, label=f'ROC curve (area = {roc_auc:.2f})')
plt.plot([0, 1], [0, 1], color='navy', lw=2, linestyle='--')  # Diagonal line for reference
plt.xlim([0.0, 1.0])
plt.ylim([0.0, 1.05])
plt.title('ROC Curve')
plt.legend(loc="lower right")
plt.show()
```


Explanation: This code is to plot the graph for adaBoost tuned hyper parameter while we plot the graph for a easier visualisation of the code.

f) Linear Discriminant Analysis

In [302]:
```
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis
from sklearn.model_selection import GridSearchCV

import numpy as np
shrinkage_values = np.linspace(0, 1, 20)

param_grid = {
    'solver': ['lsqr', 'eigen'],
    'shrinkage': [None, 'auto'] + list(shrinkage_values)
}

# Initialize LDA
lda = LinearDiscriminantAnalysis()

# GridSearch
grid_search = GridSearchCV(lda, param_grid, cv=5, scoring='accuracy', n_jobs=-1)
grid_search.fit(trainFeatures, trainLabels)

# Print best hyperparameters
print(grid_search.best_params_)
```

```
{'shrinkage': None, 'solver': 'lsqr'}
```

As shown here, we use grid search cv to find the best parameters. As shown we can sort the parameters by the specified parameters but then if we rerun the parameters again, it may take a long time as needs to find each specific parameter which parameters work the best.

To understand how grid search is implemented, we can use cv, scoring and n_jobs to determine the hyper parameter to determine which is the best parameters for adaBoost. CV stands for cross-validation, where 5 indicates to split between the number of parts where as typically 4 is to split into training and 1 is for testing. The number of cv can be increased depending on the scenario and requirements. Scoring is defined as the metrics to examine the predictions on the test set. As for this scenario we use accuracy, but not only that, we also can use f1, roc_auc too. N_jobs is a parameter that determines the number of processors that should be executed parallelly , where -1 indicates the number of processors available for the number of jobs.

As for inside the param_grid, solver means it specifies the solution for Linear Discriminant Analysis to find the solution. Whereas lsqr stands for Least Square solution and eigen stands for eigenvalue decomposition. Shrinkage is a technique to improve the estimation of the matrix of Linear Discriminant Analysis.

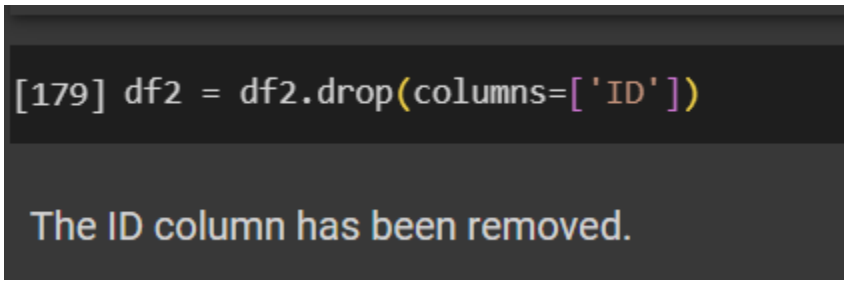
After this process, we generally plot the confusion matrix and roc_auc curve for further analysis.

3.2.2 Techniques Used

a) Dropping of Unnecessary Features

The purpose of doing this is to remove features that are not meaningful or may not/does not help in the prediction process.

i) ID



Findings: The dataset has a feature/column called ID. The values in this column are unique and are used to identify each record uniquely, which means that this key does not help predict the process. Therefore, the Team decided to drop this column as it does not bring any valuable findings to the result.

ii) Usage Rate

```
[181] df2 = df2.drop(columns=['Usage_Rate'])
```

Findings: *Usage Rate* means processing charges on the loan amount. The Team decided that this may not/does not help with the prediction process so this column was dropped.

b) Label Encoding & Replace

This dataset has some categorical columns, which are GGGrade, Experience, Validation, Home_Status, Designation, File_Status, State, Duration, Reason and Claim_Type. We have to convert these categorical columns into numerical columns in order to be able to appear in the results of the corr() function and be fitted by ML models that handle numerical data only. We will be using 2 methods, which are Label Encoding and the replace function. The replace function also helps to solve consistency problems in the data. So in the modelling and deployment parts, it will be easier to predict as all values are in numerical format.

i) Ensuring Consistency

```
[183] df2['Validation'].value_counts()

      Source Verified    34504
      Vfied              26642
      Not Vfied          26354
      Name: Validation, dtype: int64
```

Below, we have done the replacing of the "Source Verified" value with "Vfied".

```
[184] df2['Validation'] = df['Validation'].replace(['Source Verified'], 'Vfied')

[185] df2['Validation'].value_counts()

      Vfied        61146
      Not Vfied    26354
      Name: Validation, dtype: int64
```

Findings: "Source Verified" and "Vfied" should carry the same meaning so the team decided to replace the instances of the phrase “Source Verified” with "Vfied"

ii) Converting Categorical Columns Into Numerical Columns

a) GGGrade

```
[186] from sklearn import preprocessing
      labelEncoder = preprocessing.LabelEncoder()
```

a) GGGrade

```
[187] df2['GGGrade'].replace(['I', 'II','III','IV','V','VI','VII'],
                             [1,2,3,4,5,6,7], inplace=True)
```

```
[188] df2['GGGrade'].value_counts()
```

```
2    24966
3    24652
1    14171
4    13867
5     7154
6     2192
7      498
Name: GGGrade, dtype: int64
```

```
df['GGGrade'].value_counts()
```

```
II     24966
III    24652
I      14171
IV     13867
V       7154
VI      2192
VII      498
Name: GGGrade, dtype: int64
```

Findings: This dataset has some categorical columns: GGGrade, Experience, Validation, Home_Status, Designation, File_Status, State, Duration, Reason and Claim_Type. We have to convert these categorical columns into numerical columns in order to be able to appear in the results of the corr() function and be fitted by ML models that handle numerical data only. We will use two methods: Label **Encoding** and the **replace** function.

b) Experience

```
[ ] df['Experience'].value_counts()

    >10yrs    30849
    2yrs       8064
    3yrs       7350
    <1yr       7209
    1yrs       5853
    5yrs       5623
    4yrs       5324
    7yrs       4762
    8yrs       4604
    6yrs       4255
    9yrs       3607
    Name: Experience, dtype: int64

[ ] df2['Experience'].replace(['<1yr', '1yrs','2yrs','3yrs','4yrs','5yrs','6yrs','7yrs','8yrs','9yrs','>10yrs'],
                              [0,1,2,3,4,5,6,7,8,9,10], inplace=True)

[ ] df2['Experience'].value_counts()

    10    30849
    2      8064
    3      7350
    0      7209
    1      5853
    5      5623
    4      5324
    7      4762
    8      4604
    6      4255
    9      3607
    Name: Experience, dtype: int64
```

New Value	Old Value
0	<1yr
1	1yrs
2	2yrs
3	3yrs
4	4yrs
5	5yrs
6	6yrs
7	7yrs
8	8yrs
9	9yrs
10	>10yrs

Findings: For example, before encoding, “<1yr” was used to represent less than one year. Now, when it is converted to numerical format, that value is represented by 0.

```
[193] bins = [0, 2, 5, 8, 11]
      labels = ['0 - 2 years', '3 - 5 years', '6 - 8 years', '>= 9 years']
      df2 = df2.assign(experienceCat=df2.Experience)
      df2.experienceCat = pd.cut(x = df2['Experience'], labels = labels, bins = bins, include_lowest = True)
      df2.experienceCat.value_counts()

      >= 9 years     34456
      0 - 2 years    21126
      3 - 5 years    18297
      6 - 8 years    13621
      Name: experienceCat, dtype: int64
```

```
[194] df2 = df2.drop(columns=['Experience'])
      df2.info()

      <class 'pandas.core.frame.DataFrame'>
      RangeIndex: 87500 entries, 0 to 87499
      Data columns (total 28 columns):
       #   Column               Non-Null Count  Dtype
      ---  ------               --------------  -----
       0   Asst_Reg             87500 non-null  int64
       1   GGGrade              87500 non-null  int64
       2   Validation           87500 non-null  object
       3   Yearly_Income        81925 non-null  float64
       4   Home_Status          87500 non-null  object
       5   Unpaid_2_years       87500 non-null  int64
       6   Already_Defaulted    87500 non-null  int64
       7   Designation          86086 non-null  object
       8   Debt_to_Income       84011 non-null  float64
       9   Postal_Code          86111 non-null  float64
       10  Lend_Amount          87500 non-null  float64
       11  Deprecatory_Records  87500 non-null  int64
       12  Interest_Charged     87500 non-null  float64
       13  Inquiries            87500 non-null  int64
       14  Present_Balance      87500 non-null  float64
       15  Gross_Collection     87500 non-null  float64
       16  Sub_GGGrade          87500 non-null  int64
       17  File_Status          87500 non-null  object
       18  State                87500 non-null  object
       19  Account_Open         87500 non-null  int64
       20  Total_Unpaid_CL      83314 non-null  float64
       21  Duration             87500 non-null  object
       22  Unpaid_Amount        82648 non-null  float64
       23  Reason               87500 non-null  object
       24  Claim_Type           87500 non-null  object
       25  Due_Fee              87500 non-null  float64
       26  Default              87500 non-null  int64
       27  experienceCat        87500 non-null  category
      dtypes: category(1), float64(10), int64(9), object(8)
      memory usage: 18.1+ MB
```

C) Validation

```
[ ]  df2['Validation']= labelEncoder.fit_transform(df2['Validation'])
     df2['Validation'].unique()

     array([1, 0])
```

Before

Transform Vfied to '1' and Not Vfied to '0'

```
[ ]  df2['Validation'].value_counts()

     1    61146
     0    26354
     Name: Validation, dtype: int64
```

New Value	Old Value
1	Vfied
0	Not Vfied

After

Result After successfully transforming.

Findings: The values for validation has successfully transformed into numeric values for further evaluations which 1 represents: Vfied, 0 represents: Not Vfied

D) Home_Status

```
[ ]  df['Home_Status'].value_counts()

     MORTGAGE    44160
     RENT        34914
     OWN          8416
     OTHER           6
     NONE            4
     Name: Home_Status, dtype: int64

[ ]  df2['Home_Status']= labelEncoder.fit_transform(df2['Home_Status'])
     df2['Home_Status'].unique()

     array([0, 4, 3, 2, 1])
```

Before

Before Transforming Mortgage into ‘0’, Rent into ‘4’, Own into ‘3’, Other into ‘2’ and ‘None’ into 1, display the quantity for each.

```
[ ] df2['Home_Status'].value_counts()

    0    44160
    4    34914
    3     8416
    2        6
    1        4
    Name: Home_Status, dtype: int64
```

New Value	Old Value
0	MORTGAGE
1	NONE
2	OTHER
3	OWN
4	RENT

After

Findings: Check that the quantity for the newly transformed value is the same as the old value. For example, check whether the quantity of the Mortgage value is the same as the quantity for the '0' value after transformation.

E)File_Status

```
df2['File_Status']= labelEncoder.fit_transform(df2['File_Status'])
df2['File_Status'].unique()

    array([0, 1])

[ ] print(df2['File_Status'].value_counts())

    1    46300
    0    41200
    Name: File_Status, dtype: int64
```

New Value	Old Value
1	whole
0	fully paid

Findings: Transform the ‘whole’ into 1 and ‘fully paid’ into 0 and count the quantity for each.

F) Designation

```
[ ]  df2['Designation'].unique()

     array(['GLENDALE NISSAN', 'Business Teacher', 'driver', ...,
            'Interface Coordinator', 'PARTS CLERK', 'Outside sales rep'],
           dtype=object)

[ ]  df2['Designation']= df2['Designation'].astype(str)
     df2['Designation']= labelEncoder.fit_transform(df2['Designation'])
     df2['Designation'].unique()

     array([12862,  4177, 36011, ..., 15514, 21431, 21262])
```

Findings: Transform the value into its numerical equivalent.

G) State

```
[ ]  df2['State']= labelEncoder.fit_transform(df2['State'])
     df2['State'].unique()

     array([ 6, 25,  9, 31, 16, 41,  5, 19, 34,  2, 32, 23,  4, 35, 29, 20, 10,
            45, 24,  1, 38, 36,  3, 42, 17, 43, 12, 46, 30, 11, 37, 40, 33, 44,
            13,  0, 21, 27, 28,  7, 47, 22, 15, 14,  8, 26, 39, 48, 18])
```

Findings: Transform the value into its numerical equivalent.

h) Duration

```
[ ]  df2['Duration'].replace(['3 years', '5 years'],
                             [3, 5], inplace=True)

[ ]  print(df2['Duration'].value_counts())

     3    60061
     5    27439
     Name: Duration, dtype: int64
```

New Value	Old Value
3	3 years
5	5 years

Transform ‘ 3 years ‘ and ‘5 years’ into the actual numerical values: 3 and 5

I) Reason

```
[ ] df2['Reason']= labelEncoder.fit_transform(df2['Reason'])
    df2['Reason'].unique()
```

```
array([ 3,  6,  9,  2,  1,  4, 10,  8, 11,  7, 12,  5,  0])
```

```
print(df2['Reason'].value_counts())
```

```
3     52559
2     21102
4      4886
9      3774
6      1512
7       821
10      751
1       710
8       503
11      424
5       311
12      109
0        38
Name: Reason, dtype: int64
```

New Value	Old Value
0	RENTwable energy
1	car
2	credit card
3	debt consolidation
4	home improvement
5	house
6	major purchase
7	medical
8	moving
9	other
10	small business
11	vacation
12	wedding

Transform from categorical data to numerical data and display the New Value (Numerical) with the corresponding Old Value (Categorical)

J) Claim_Type

```
[ ] df2['Claim_Type']= labelEncoder.fit_transform(df2['Claim_Type'])
    df2['Claim_Type'].unique()

    array([0, 1])

[ ] print(df2['Claim_Type'].value_counts())

    0    87454
    1       46
    Name: Claim_Type, dtype: int64
```

New Value	Old Value
0	I
1	J

Transform the values for Claim_Type from categorical to numerical. The values of 'I' and 'J' are transformed into '0' and '1'.

c) Binning

Binning reduces the complexity and number of unique numbers by splitting the number into different bins. So, it treats numbers as an interval instead of making every number a unique value to make it easier to perform statistical analysis. So, instead of treating 0,1,2 as distinct numbers, it would treat 0-2 as one value to be trained by the model.

```
bins = [0, 2, 5, 8, 11]
labels = ['0 - 2 years', '3 - 5 years', '6 - 8 years', '>= 9 years']
df2 = df2.assign(experienceCat=df2.Experience)
df2.experienceCat = pd.cut(x = df2['Experience'], labels = labels, bins = bins, include_lowest = True)
df2.experienceCat.value_counts()
```

```
9]: >= 9 years     34456
    0 - 2 years    21126
    3 - 5 years    18297
    6 - 8 years    13621
    Name: experienceCat, dtype: int64
```

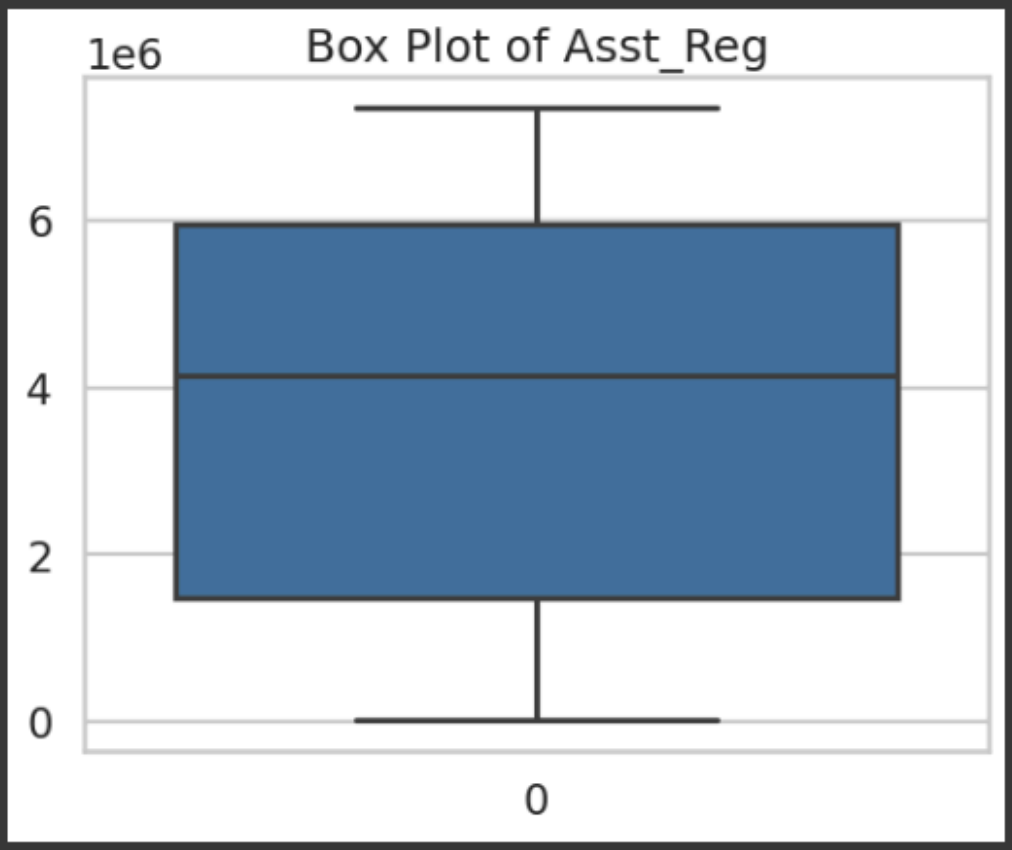
d) Outlier Detection Using Boxplot, HistPlot, DistPlot… & Treatment Using Log Transformation & Standard Deviation

As you can see later on, we have used Log transformation to remove or reduce the skewness of the numeric values.

i) Asst_Reg

a) Asst_Reg

```
sns.boxplot(df2['Asst_Reg'])
plt.title("Box Plot of Asst_Reg")
plt.show()
```



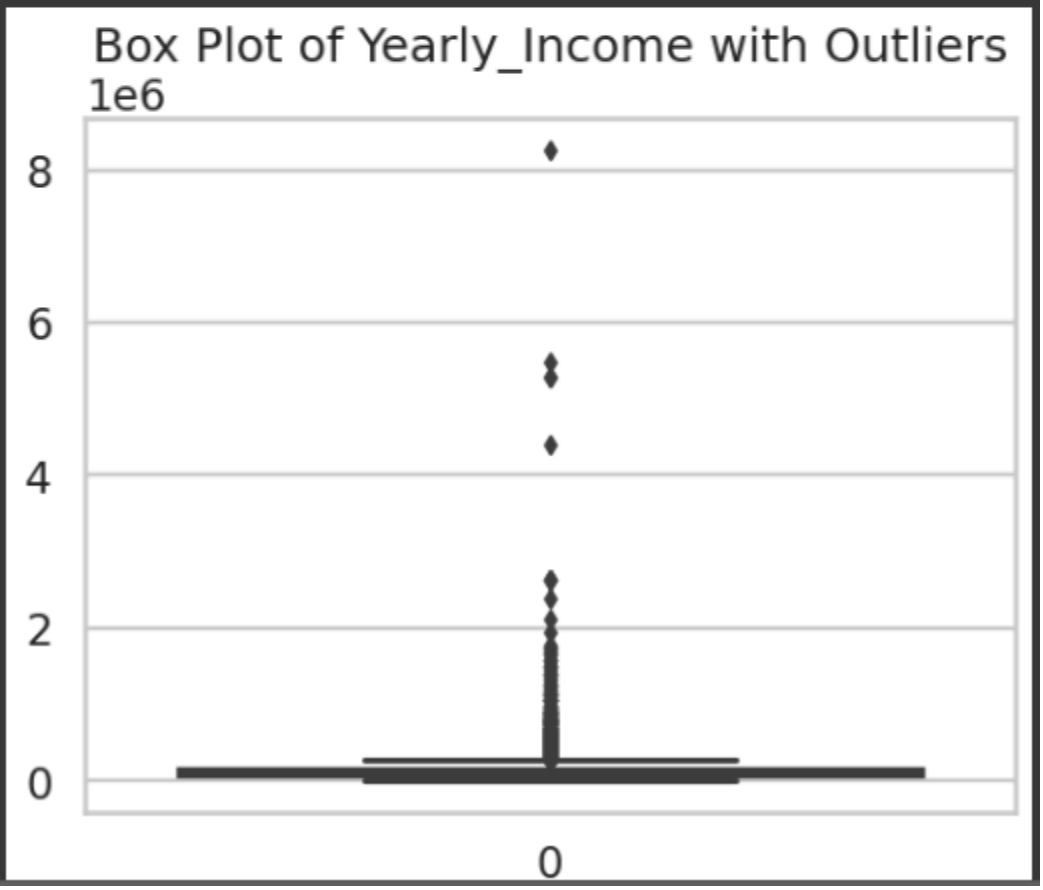
This column has no Outlier.

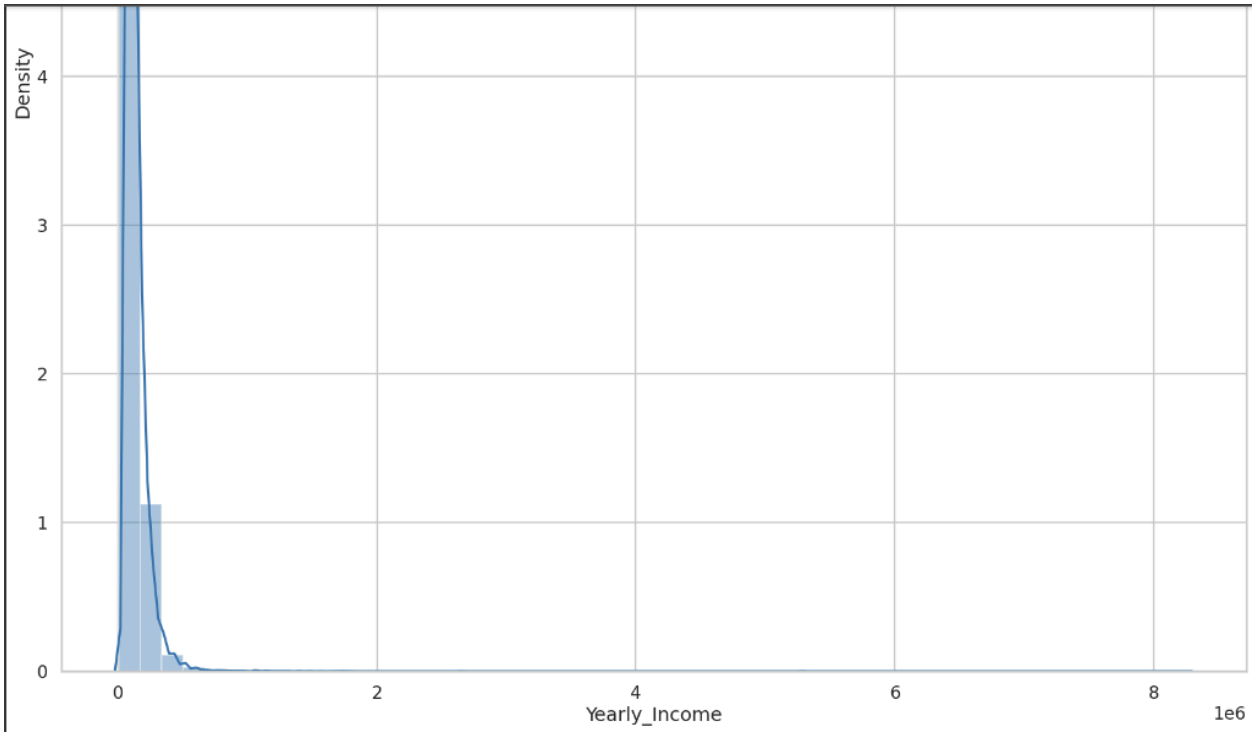
ii) Yearly_Income

```
[ ]  df2['Yearly_Income'].describe()
```

```
count    8.192500e+04
mean     1.348596e+05
std      9.882473e+04
min      8.800000e+03
25%      8.324659e+04
50%      1.144000e+05
75%      1.601600e+05
max      8.264031e+06
Name: Yearly_Income, dtype: float64
```

```
[ ]  sns.boxplot(df2['Yearly_Income'])
     plt.title("Box Plot of Yearly_Income with Outliers")
     plt.show()
```



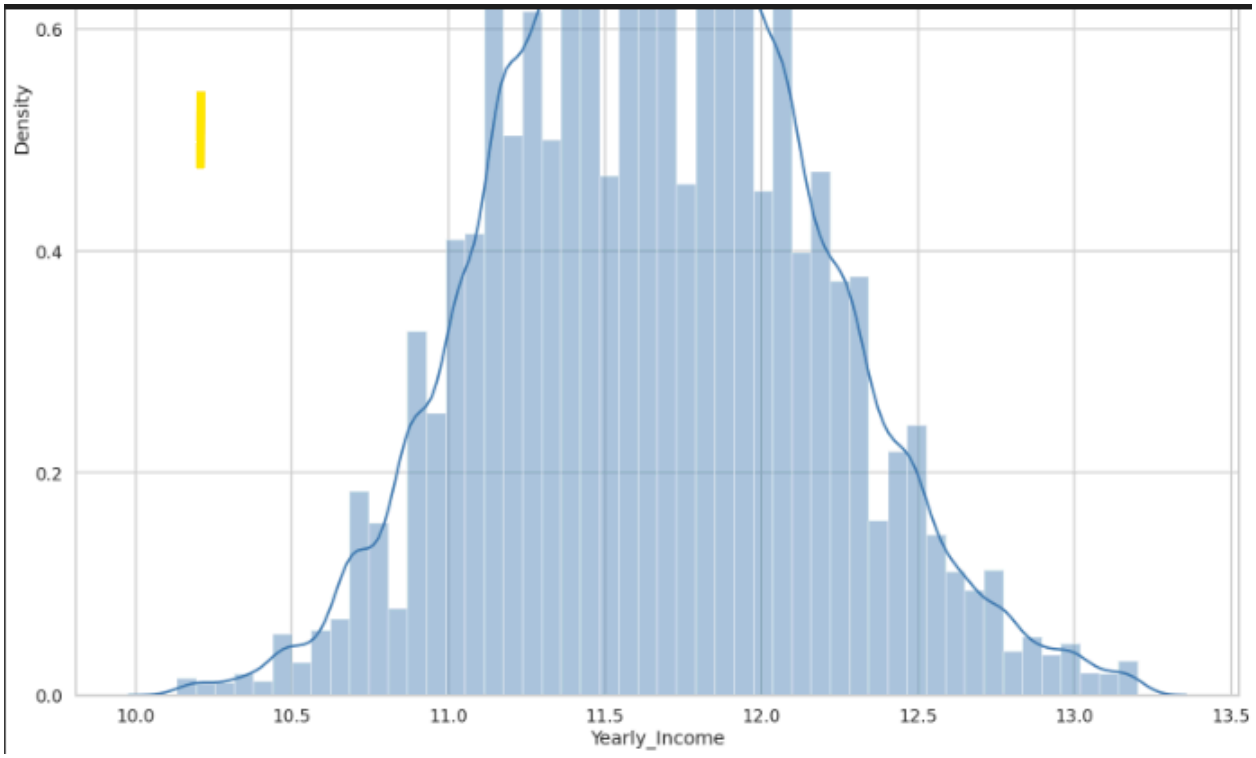


Before

This **Yearly_Income** column has numerous Outliers, as indicated on the Box Plot, and the distribution Plot indicates it is left-skewed, which means all data aggregates around 0 to 1.

```
[ ] df2['Yearly_Income'] = np.log(df2['Yearly_Income'] + 1)
    up_bd = df2['Yearly_Income'].mean() + 3 * df2['Yearly_Income'].std()
    low_bd = df2['Yearly_Income'].mean() - 3 * df2['Yearly_Income'].std()
    print("Greatest Value allowed for Yearly_Income: ", up_bd)
    print("Smallest Value allowed for Yearly_Income: ", low_bd)

    Greatest Value allowed for Yearly_Income:  13.20708299933477
    Smallest Value allowed for Yearly_Income:  10.131924427500536

[ ] df2 = df2[(df2['Yearly_Income'] < up_bd) & (df2['Yearly_Income'] > low_bd)]

[ ] df2['Yearly_Income'].describe()

    count    81397.000000
    mean        11.662687
    std          0.491026
    min         10.131969
    25%         11.328534
    50%         11.647465
    75%         11.972885
    max         13.203171
    Name: Yearly_Income, dtype: float64
```

Transform the data, identity and remove the potential Outliers using standard deviation.



After

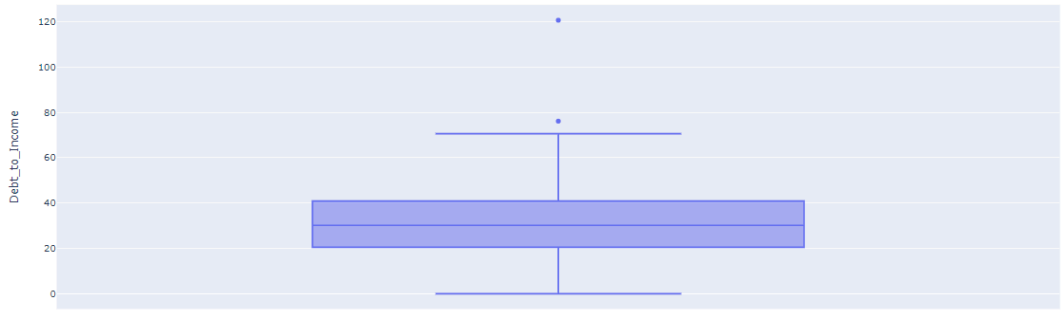
After removing the outlier, the graph becomes approximately normally distributed, which fulfils the binomial distribution concept.

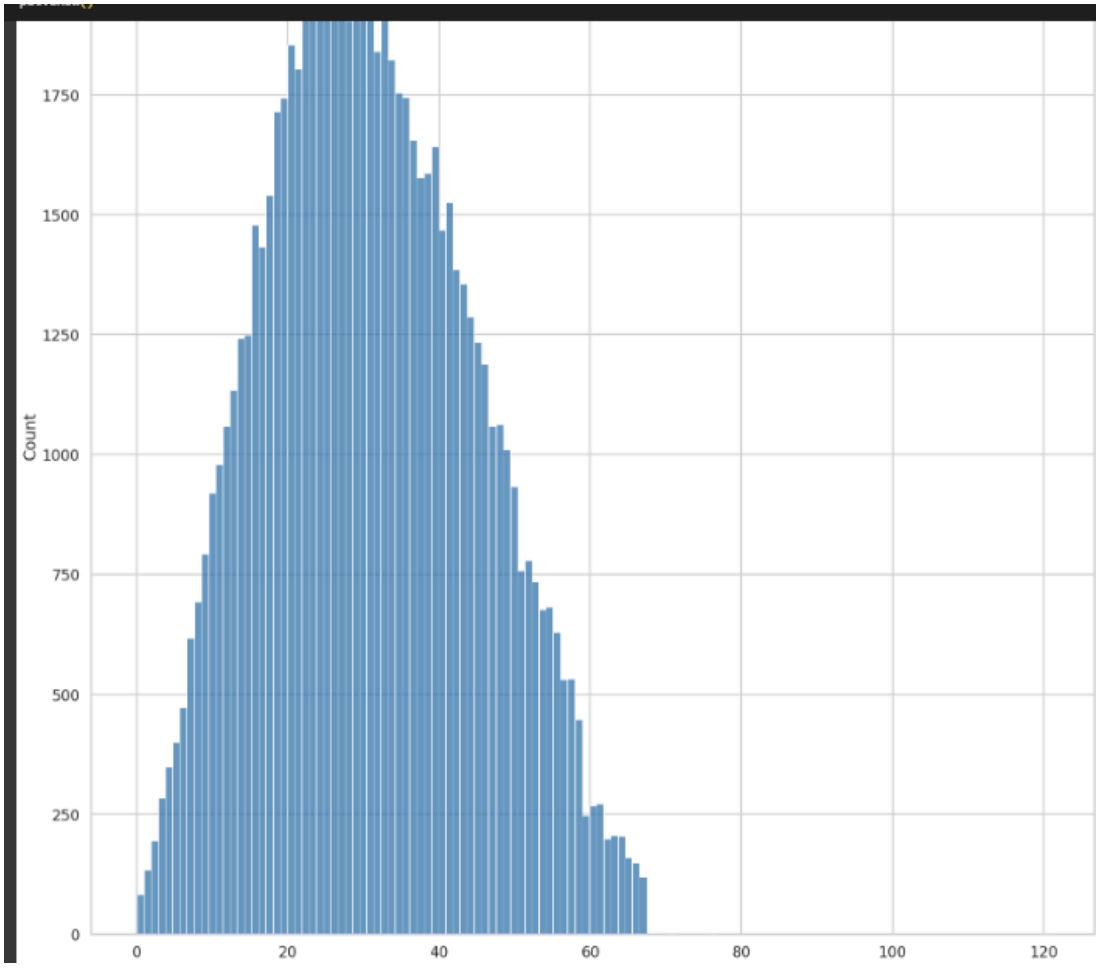
iii) Debt_to_Income

```
df2['Debt_to_Income'].describe()
```

```
count    78161.000000
mean        30.979739
std         13.888910
min          0.000000
25%         20.479200
50%         30.189600
75%         40.924800
max        120.657600
Name: Debt_to_Income, dtype: float64
```

```
import plotly.express as px
fig = px.box(df2, y="Debt_to_Income")
fig.show()
```





Before

This **Debt_to_Income** column has some outliers, as shown in the graph; this column is slightly right skewed.

```
up_bd = df2['Debt_to_Income'].mean() + 3 * df2['Debt_to_Income'].std()
low_bd = df2['Debt_to_Income'].mean() - 3 * df2['Debt_to_Income'].std()
print("Greatest Value allowed for Debt_to_Income: ", up_bd)
print("Smallest Value allowed for Debt_to_Income: ", low_bd)
```

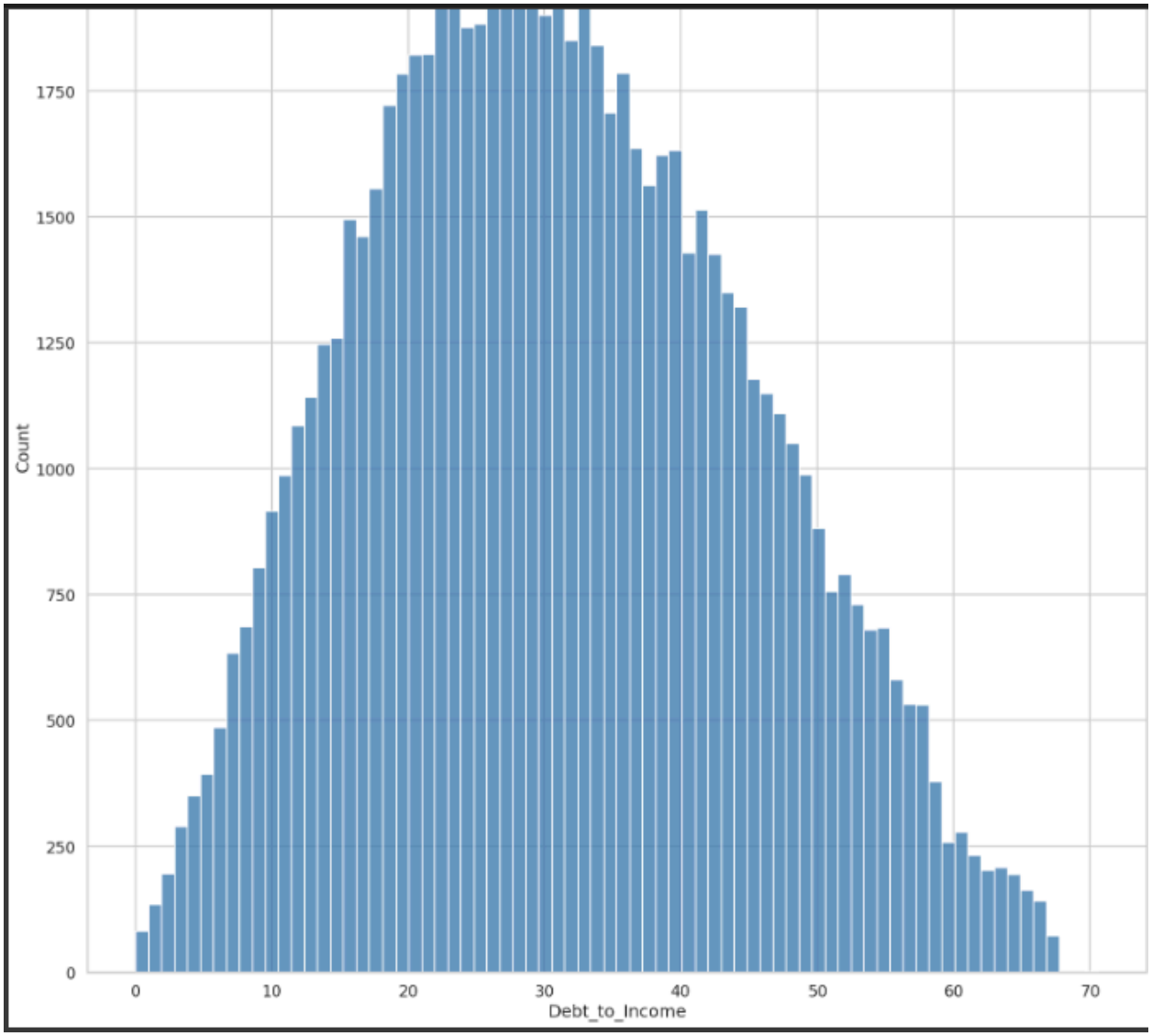
```
Greatest Value allowed for Debt_to_Income:  72.64646825482801
Smallest Value allowed for Debt_to_Income:  -10.686990110996685
```

```
df2 = df2[(df2['Debt_to_Income'] < up_bd) & (df2['Debt_to_Income'] > low_bd)]
```

```
df2['Debt_to_Income'].describe()
```

```
count    78159.000000
mean        30.978014
std         13.884443
min          0.000000
25%         20.479200
50%         30.189600
75%         40.924800
max         70.627200
Name: Debt_to_Income, dtype: float64
```

Findings: Transform the data, identity and remove the potential Outliers using standard deviation.

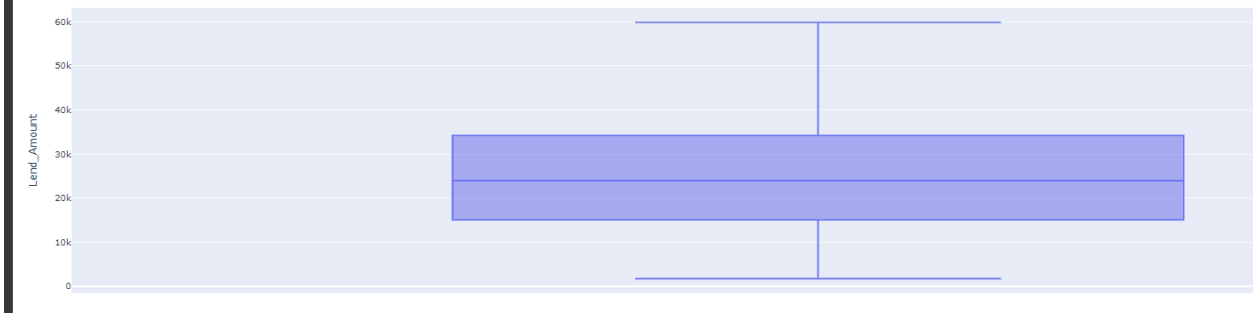


After

After removing the outlier, the **Debt_to_Income** column becomes approximately normally distributed.

iv) Lend_Amount

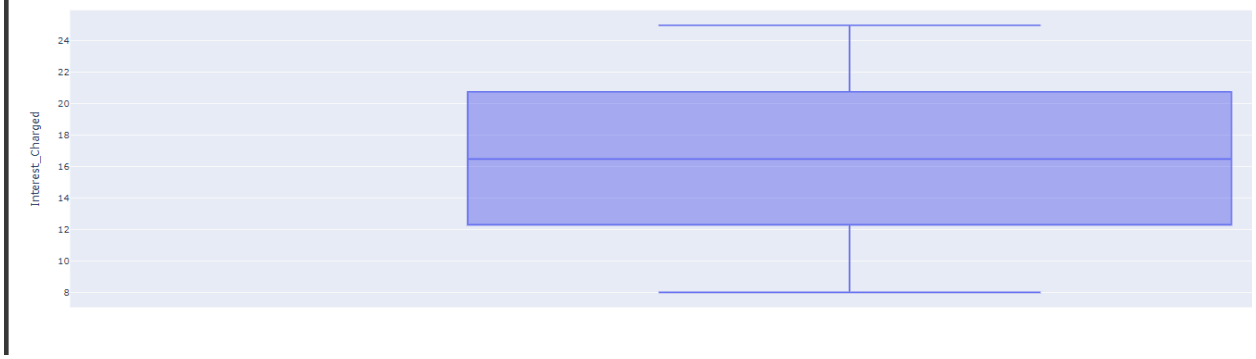
```
import plotly.express as px
fig = px.box(df2, y="Lend_Amount")
fig.show()
```



There is no outlier for the **Lend_Amount** column.

v) Interest Charge

```
import plotly.express as px
fig = px.box(df2, y="Interest_Charged")
fig.show()
```



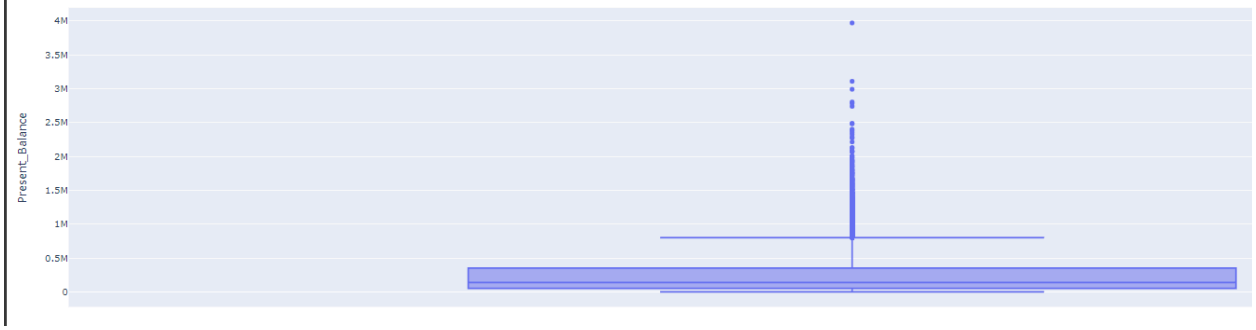
There is no outlier for the **Interest Charge** column.

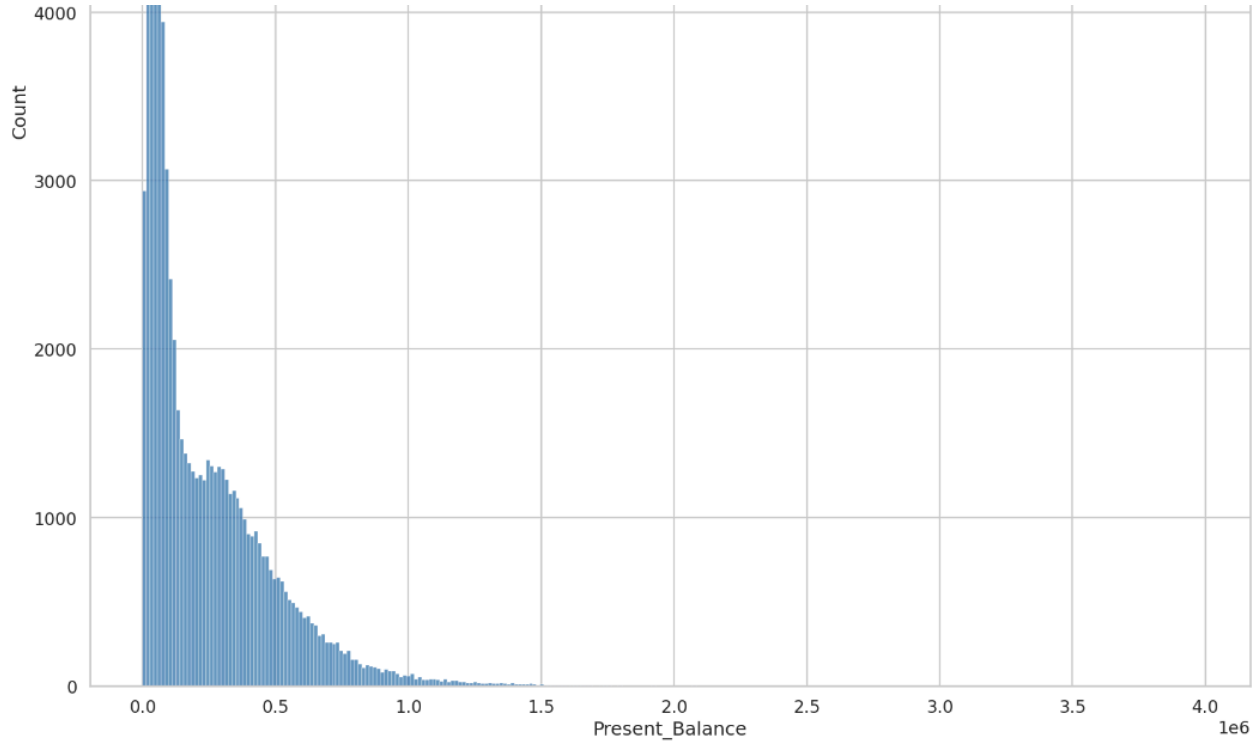
vi) Present Balance

```
df2['Present_Balance'].describe()
```

```
count    7.815900e+04
mean     2.326984e+05
std      2.424059e+05
min      0.000000e+00
25%      5.186853e+04
50%      1.379320e+05
75%      3.512311e+05
max      3.970979e+06
Name: Present_Balance, dtype: float64
```

```
import plotly.express as px
fig = px.box(df2, y="Present_Balance")
fig.show()
```





This **Present Balance** column has numerous Outliers, as indicated on the Box Plot, and the distribution Plot indicates it is right-skewed, which means all data aggregates around 0 to 0.3 present_Balance.

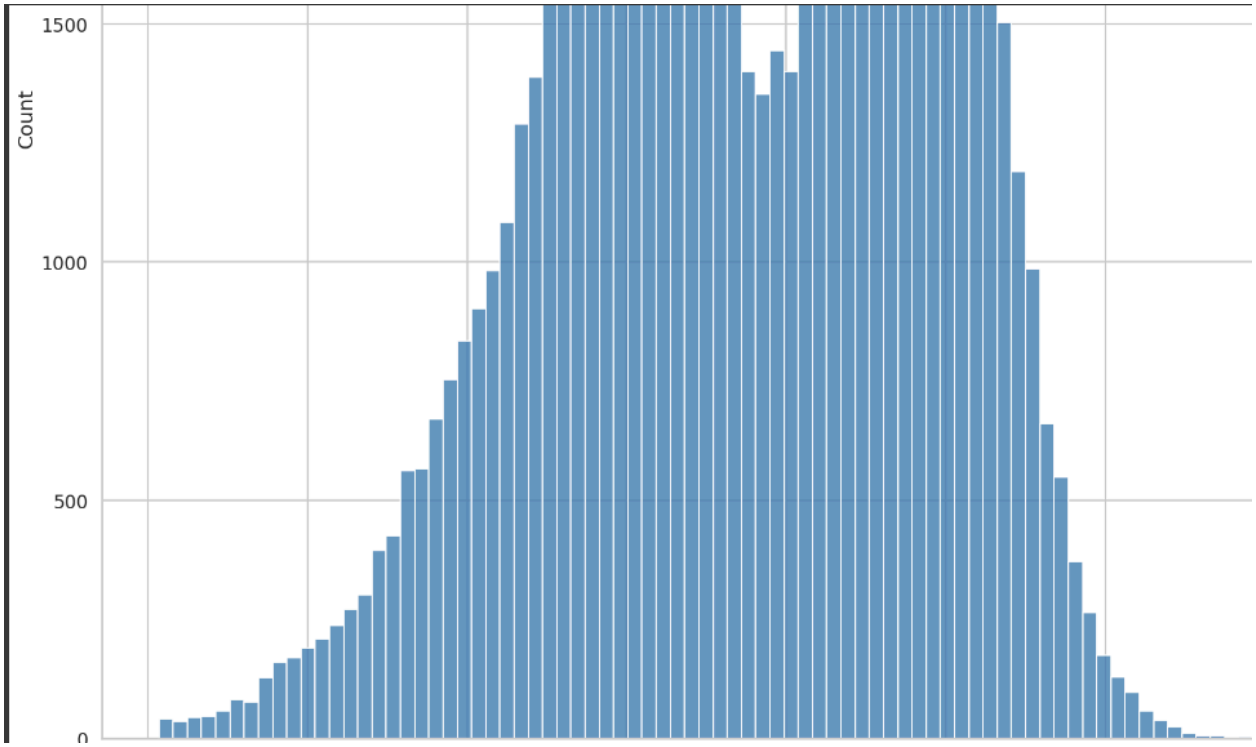
```
df2['Present_Balance'] = np.log(df2['Present_Balance'] + 1)
```

```
up_bd = df2['Present_Balance'].mean() + 3 * df2['Present_Balance'].std()
low_bd = df2['Present_Balance'].mean() - 3 * df2['Present_Balance'].std()
print("Greatest Value allowed for Present_Balance: ", up_bd)
print("Smallest Value allowed for Present_Balance: ", low_bd)
```

```
Greatest Value allowed for Present_Balance:  15.43981632061838
Smallest Value allowed for Present_Balance:  8.066463196744838
```

```
df2 = df2[(df2['Present_Balance'] < up_bd) & (df2['Present_Balance'] > low_bd)]
```

Transform the data, identity and remove the potential Outliers using standard deviation for **Present Balance** column.

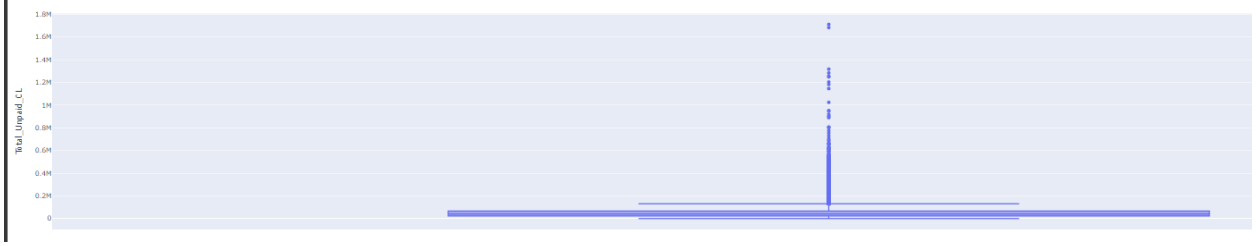


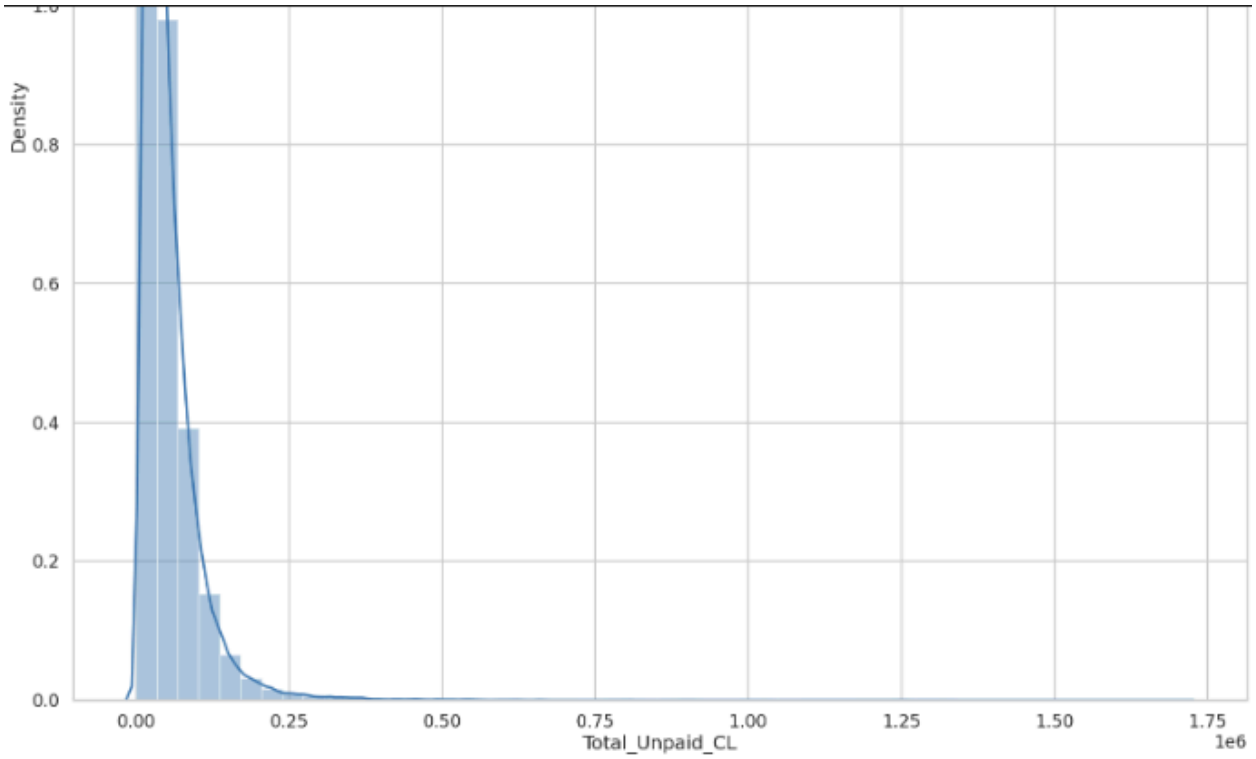
Findings: After removing the outlier, the **Present Balance** column becomes approximately normally distributed.

vii) Total Unpaid CL

```
df2['Total_Unpaid_CL'].describe()

count    7.416600e+04
mean     5.280039e+04
std      5.111036e+04
min      3.320000e+02
25%      2.340600e+04
50%      3.967400e+04
75%      6.623400e+04
max      1.710630e+06
Name: Total_Unpaid_CL, dtype: float64
```

```
import plotly.express as px
fig = px.box(df2, y="Total_Unpaid_CL")
fig.show()
```





Findings: This **Total Unpaid CL** column has numerous Outliers, as indicated on the Box Plot, and the distribution Plot indicates it is right-skewed, which means most of the data aggregate around 0 to 0.125.

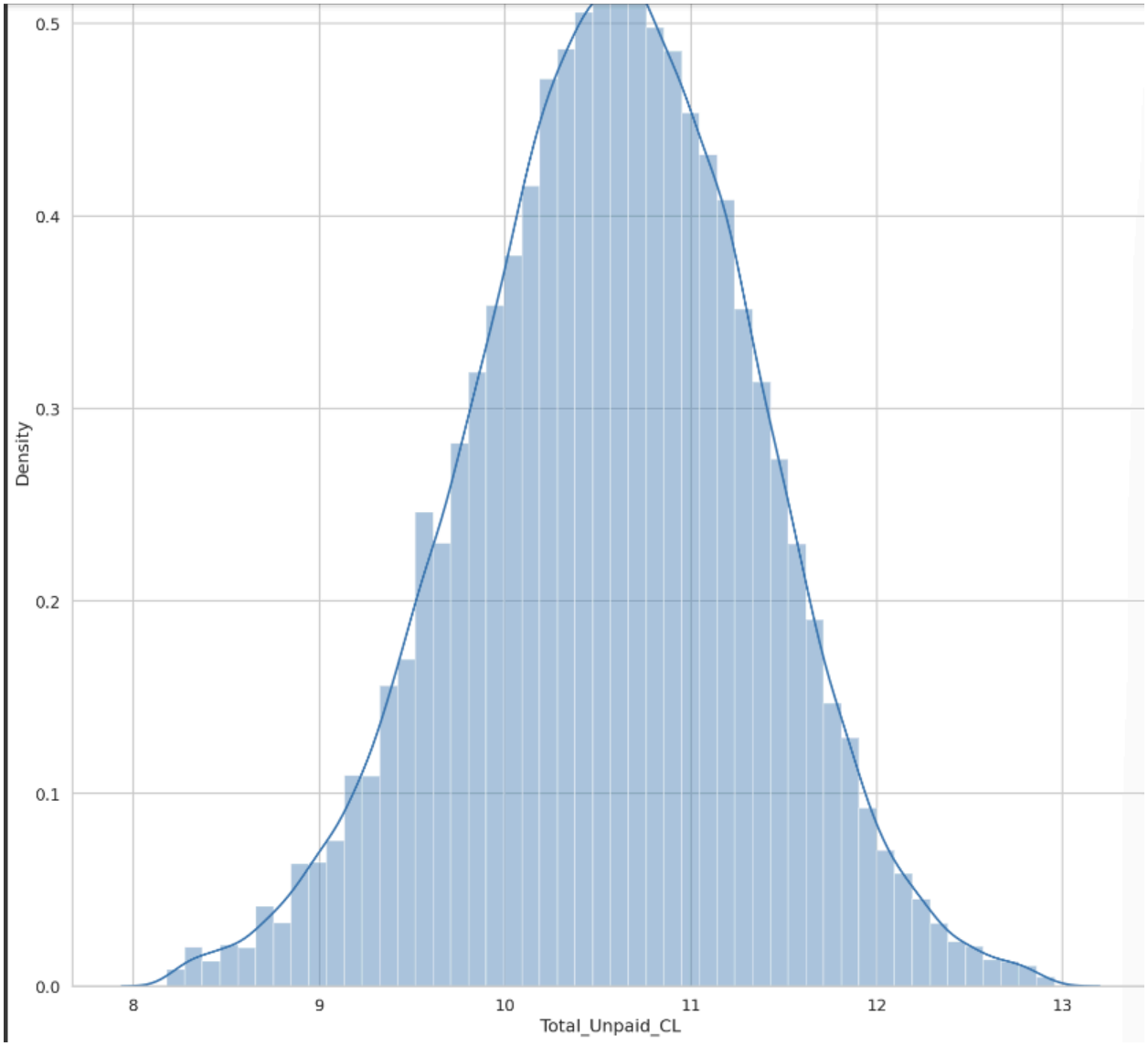
```
[ ] df2['Total_Unpaid_CL'] = np.log(df2['Total_Unpaid_CL'] + 1)
```

```
[ ] up_bd = df2['Total_Unpaid_CL'].mean() + 3 * df2['Total_Unpaid_CL'].std()
    low_bd = df2['Total_Unpaid_CL'].mean() - 3 * df2['Total_Unpaid_CL'].std()
    print("Greatest Value allowed for Total_Unpaid_CL: ", up_bd)
    print("Smallest Value allowed for Total_Unpaid_CL: ", low_bd)

    Greatest Value allowed for Total_Unpaid_CL:  12.957568042956122
    Smallest Value allowed for Total_Unpaid_CL:  8.171568206536076
```

```
[ ] df2 = df2[(df2['Total_Unpaid_CL'] < up_bd) & (df2['Total_Unpaid_CL'] > low_bd)]
```

```
[ ] df2['Total_Unpaid_CL'].describe()

    count    73672.000000
    mean        10.572839
    std          0.765899
    min          8.180321
    25%         10.067857
    50%         10.588477
    75%         11.100964
    max         12.956231
    Name: Total_Unpaid_CL, dtype: float64
```

Transform the data, identity and remove the potential Outliers using standard deviation for **Total Unpaid CL** column.



Findings: After removing the outlier, the **Total Unpaid CL** column becomes equally distributed.

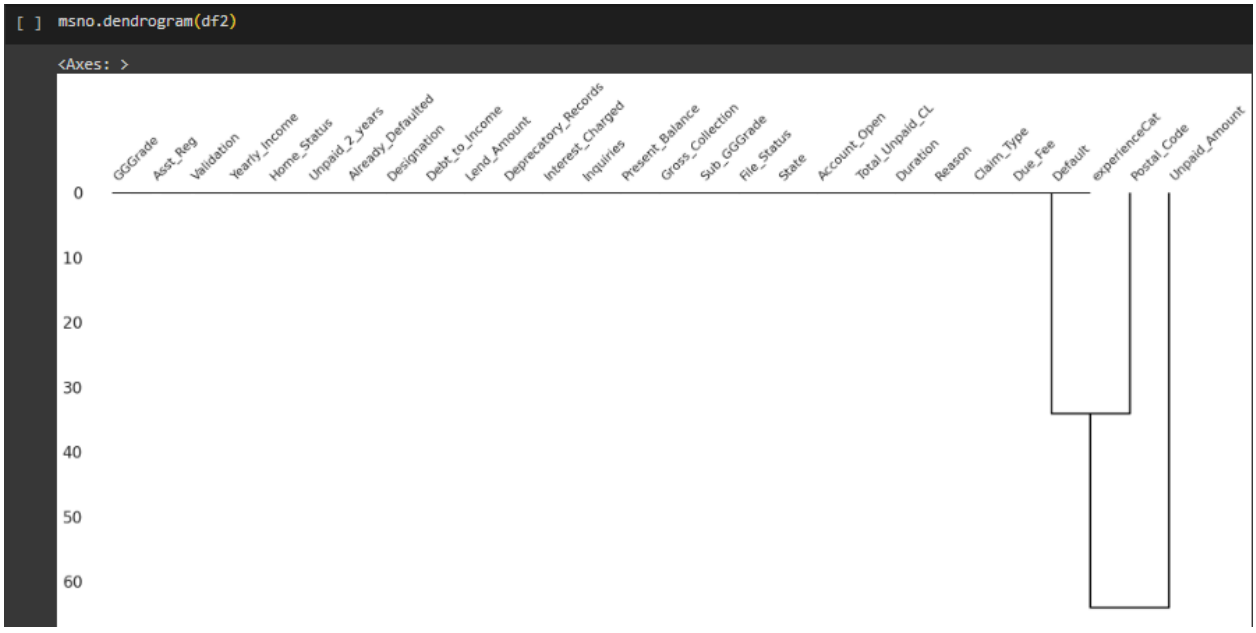
e) Impute Missing Values with Mean for Numeric Values and Mode for Categorical Values

Missing Values mean some feature has a null value (Does not have a value); an Appropriate method should be applied to these data.

```
[ ] df2.isnull().sum()

Asst_Reg                0
GGGrade                 0
Validation              0
Yearly_Income           0
Home_Status             0
Unpaid_2_years          0
Already_Defaulted       0
Designation             0
Debt_to_Income          0
Postal_Code          1160
Lend_Amount             0
Deprecatory_Records     0
Interest_Charged        0
Inquiries               0
Present_Balance         0
Gross_Collection        0
Sub_GGGrade             0
File_Status             0
State                   0
Account_Open            0
Total_Unpaid_CL         0
Duration                0
Unpaid_Amount        4062
Reason                  0
Claim_Type              0
Due_Fee                 0
Default                 0
experienceCat           0
dtype: int64
```

Postal_Code and **Unpaid_Amount** have missing values.

```
[ ] msno.dendrogram(df2)
```



Dendrogram shows the relationship between each classes that have been merged together. Hence showing missing values that contained in the data. To avoid overlapping each class, dendrogram uses a hierarchical cluster method to separate the classes.

i) PostalCode

```
[ ]  from sklearn.impute import SimpleImputer

     modeImputer = SimpleImputer(missing_values=np.NaN, strategy='most_frequent')

     df2['Postal_Code'] = modeImputer.fit_transform(df2['Postal_Code'].values.reshape(-1,1))[:,0]
```

Postal Code is a categorical variable with 1160 missing values. Since imputing missing data with mean or median values can only be done with numerical data, it is not appropriate to use either mean or median to replace missing values for this column. Therefore, mode is the most suitable as it can work with numerical and categorical columns.

ii) Unpaid Amount

```
[ ]  meanImputer = SimpleImputer(missing_values=np.NaN, strategy='mean')

     df2['Unpaid_Amount'] = meanImputer.fit_transform(df2['Unpaid_Amount'].values.reshape(-1,1))[:,0]
```

We could choose either mean or median for this variable, but we will select median. The median only uses one or two values; it is not affected by extreme outliers or non-symmetric distributions of scores. On the other hand, the mean and mode can vary in skewed distributions.

f) Changing data type of column

```
[ ] df2.info()

<class 'pandas.core.frame.DataFrame'>
Int64Index: 73672 entries, 1 to 87498
Data columns (total 28 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Asst_Reg             73672 non-null  int64
 1   GGGrade              73672 non-null  int64
 2   Validation           73672 non-null  int64
 3   Yearly_Income        73672 non-null  float64
 4   Home_Status          73672 non-null  int64
 5   Unpaid_2_years       73672 non-null  int64
 6   Already_Defaulted    73672 non-null  int64
 7   Designation          73672 non-null  int64
 8   Debt_to_Income       73672 non-null  float64
 9   Postal_Code          73672 non-null  float64
 10  Lend_Amount          73672 non-null  float64
 11  Deprecatory_Records  73672 non-null  int64
 12  Interest_Charged     73672 non-null  float64
 13  Inquiries            73672 non-null  int64
 14  Present_Balance      73672 non-null  float64
 15  Gross_Collection     73672 non-null  float64
 16  Sub_GGGrade          73672 non-null  int64
 17  File_Status          73672 non-null  int64
 18  State                73672 non-null  int64
 19  Account_Open         73672 non-null  int64
 20  Total_Unpaid_CL      73672 non-null  float64
 21  Duration             73672 non-null  int64
 22  Unpaid_Amount        73672 non-null  float64
 23  Reason               73672 non-null  int64
 24  Claim_Type           73672 non-null  int64
 25  Due_Fee              73672 non-null  float64
 26  Default              73672 non-null  int64
 27  experienceCat        73672 non-null  category
dtypes: category(1), float64(10), int64(17)
memory usage: 15.8 MB
```

As seen above, the data type of the experience cat is categorical data type.. Therefore, we have to change its data type to numeric.

Experience category

```
[ ] df2['experienceCat'].replace(['0 - 2 years', '3 - 5 years', '6 - 8 years', '>= 9 years'],
                                 [0, 1, 2, 3], inplace=True)
```

```
[ ] df2['experienceCat'] = df2['experienceCat'].astype(int)
```

```
[ ] df2['experienceCat'].value_counts()
```

```
3    29025
0    17735
1    15361
2    11551
Name: experienceCat, dtype: int64
```

```
[ ] df2['experienceCat'].value_counts()
```

```
3    29025
0    17735
1    15361
2    11551
Name: experienceCat, dtype: int64
```

New Value	Old Value
0	0 - 2 years
1	3 - 5 years
2	6 - 8 years
3	>= 9 years

Changing of experienceCat data type from categorical to numerical, where 0 represents '0-2 years', 1 represents '3-5', 2 represents '6-8' and 3 represents '>=9'.

```
[ ] df2.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 73672 entries, 1 to 87498
Data columns (total 28 columns):
 #   Column               Non-Null Count  Dtype
---  ------               --------------  -----
 0   Asst_Reg             73672 non-null  int64
 1   GGGrade              73672 non-null  int64
 2   Validation           73672 non-null  int64
 3   Yearly_Income        73672 non-null  float64
 4   Home_Status          73672 non-null  int64
 5   Unpaid_2_years       73672 non-null  int64
 6   Already_Defaulted    73672 non-null  int64
 7   Designation          73672 non-null  int64
 8   Debt_to_Income       73672 non-null  float64
 9   Postal_Code          73672 non-null  float64
 10  Lend_Amount          73672 non-null  float64
 11  Deprecatory_Records  73672 non-null  int64
 12  Interest_Charged     73672 non-null  float64
 13  Inquiries            73672 non-null  int64
 14  Present_Balance      73672 non-null  float64
 15  Gross_Collection     73672 non-null  float64
 16  Sub_GGGrade          73672 non-null  int64
 17  File_Status          73672 non-null  int64
 18  State                73672 non-null  int64
 19  Account_Open         73672 non-null  int64
 20  Total_Unpaid_CL      73672 non-null  float64
 21  Duration             73672 non-null  int64
 22  Unpaid_Amount        73672 non-null  float64
 23  Reason               73672 non-null  int64
 24  Claim_Type           73672 non-null  int64
 25  Due_Fee              73672 non-null  float64
 26  Default              73672 non-null  int64
 27  experienceCat        73672 non-null  int64
dtypes: float64(10), int64(18)
memory usage: 16.3 MB
```

The data type of experienceCat is changed from categorical to int64(int).

g) Normalisation

It is 1 of the most often used data preparation methods. It changes the values of numeric columns to use a common scale. We will be using a normalisation technique called the min-max scaling, which rescales values in the feature into the range [0,1].

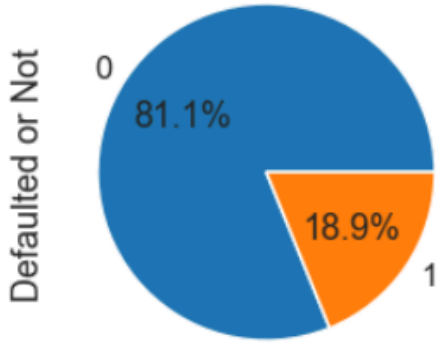
```
In [245]: df3 = df2.copy()

In [246]: from sklearn.preprocessing import MinMaxScaler

In [247]: mmScaler = MinMaxScaler()
          scaledData = mmScaler.fit_transform(df3)
```

Using min-max scaler to rescale the values in the feature into the range [0,1]. Normalisation is utilised when we do not know the data's distribution or when we know that the distribution is not normal (a bell curve). This technique is useful when the data has varying scales.

h) Balancing Dataset by Upsampling

The dataset is imbalanced as there is an uneven number of Default and Non-Default records. In this part we are handling the data that are imbalanced so that in the modelling stage the result will be more accurate for our deployment stage.

```
In [253]: df4['Default'].value_counts()

Out[253]: 0    59777
          1    13895
          Name: Default, dtype: int64

In [254]: df4.groupby('Default').size().plot(kind='pie',
                                             y = "Default",
                                             label = "Defaulted or Not",
                                             autopct='%1.1f%%')

Out[254]: <AxesSubplot:ylabel='Defaulted or Not'>
```



Findings: As shown above the variable that we want to predict is the default variable. Since there is an imbalance between the Default and Non-default records, it can result in a bias if the model or the deployment is carried out in this manner. So to make it fair we will do upsample the dataset to balance it out.

In [255]:
```
from sklearn.utils import resample
upDF = df4.copy()
defaulted = upDF[upDF["Default"] == 1]
nonDefaulted  = upDF[upDF["Default"] == 0]

print("Shape of Defaulted cases Before Upsampling:", defaulted.shape)
print("Shape of Not Default cases Before Upsampling:", nonDefaulted.shape)
```

```
Shape of Defaulted cases Before Upsampling: (13895, 28)
Shape of Not Default cases Before Upsampling: (59777, 28)
```

In [256]:
```
defaulted_upsample = resample(defaulted,
             replace=True,
             n_samples=len(nonDefaulted),
             random_state=42)

print(defaulted_upsample.shape)
```

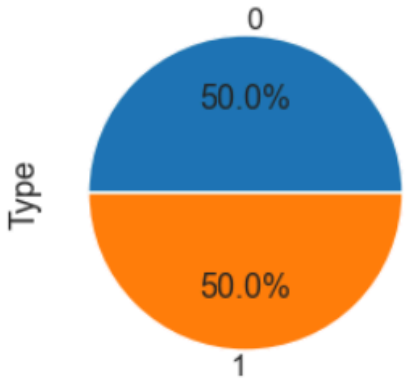
```
(59777, 28)
```

Findings: To know about the data contained inside, we have gained more information about this particular variable, In the 81.1% there are a total of around 60000 records as for the 18.9% contains records of 13895. Which shows this is very imbalance.

In [257]:
```
df4_upsampled = pd.concat([defaulted_upsample, nonDefaulted])

print(df4_upsampled["Default"].value_counts())

df4_upsampled.groupby('Default').size().plot(kind='pie',
                                        y = "Default",
                                        label = "Type",
                                        autopct='%1.1f%%')
```

```
1    59777
0    59777
Name: Default, dtype: int64
```

Out[257]: <AxesSubplot:ylabel='Type'>



As shown above, we managed to allocate the same number of records to each indicator of the Default variable.

i) Feature Importances to aid in Feature Selection

In this section, we are trying to find out which variables are important apart from what we want to predict, which is the Default variable.

2.9 Feature Selection ¶

In [260]:
```
train_data = trainFeatures.copy()
train_data['Default'] = trainLabels
train_data.head()
```

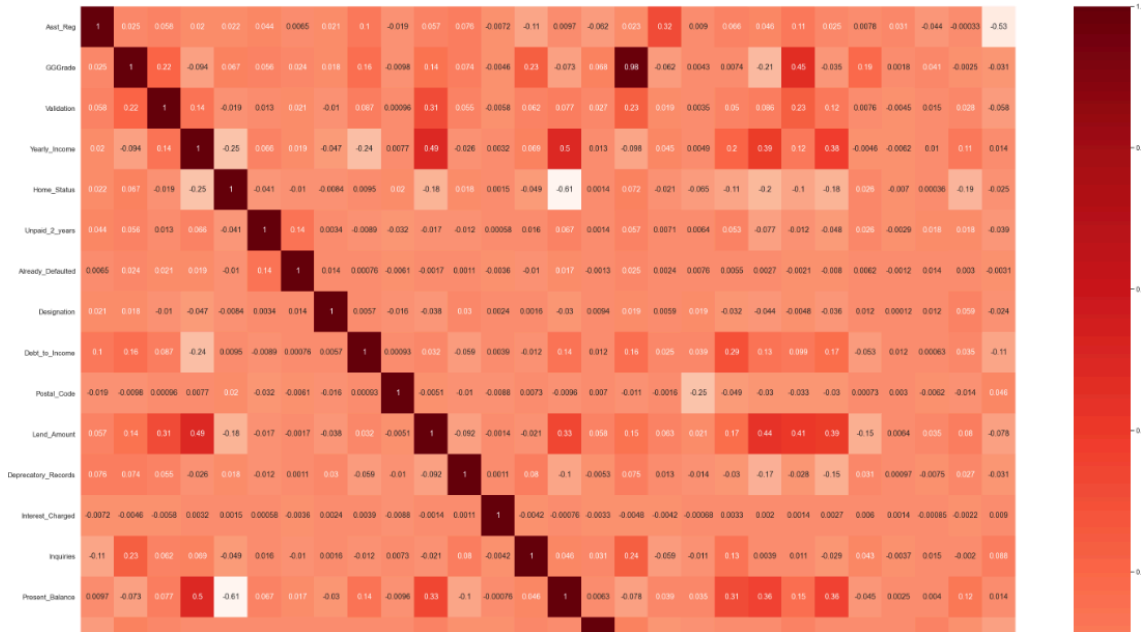
Out[260]:

	Asst_Reg	GGGrade	Validation	Yearly_Income	Home_Status	Unpaid_2_years	Already_Defaulted	Designation	Debt_to_Income	Postal_Code	...	Sta
60389	0.970092	0.166667	1.0	0.301066	0.75	0.000000	0.0	0.324427	0.703378	0.748231	...	0.2083
21497	0.020302	0.000000	0.0	0.611835	0.00	0.000000	0.0	0.808020	0.219553	0.550051	...	0.4166
16496	0.858318	0.500000	1.0	0.607629	0.00	0.000000	0.0	0.642783	0.500951	0.287159	...	0.7916
44395	0.754583	0.166667	0.0	0.301066	0.75	0.611111	0.0	0.658546	0.265699	0.618807	...	0.2500
24729	0.053225	0.500000	1.0	0.658777	1.00	0.000000	0.0	0.738227	0.097526	0.945399	...	0.1250

5 rows × 28 columns

Findings: Displaying the correlation between the variables to check the importance of each variable.

In [261]:
```
plt.figure(figsize=(50,50))
corrCoeff = train_data.corr()
sns.heatmap(corrCoeff, annot=True, cmap=plt.cm.Reds)
plt.show()
```

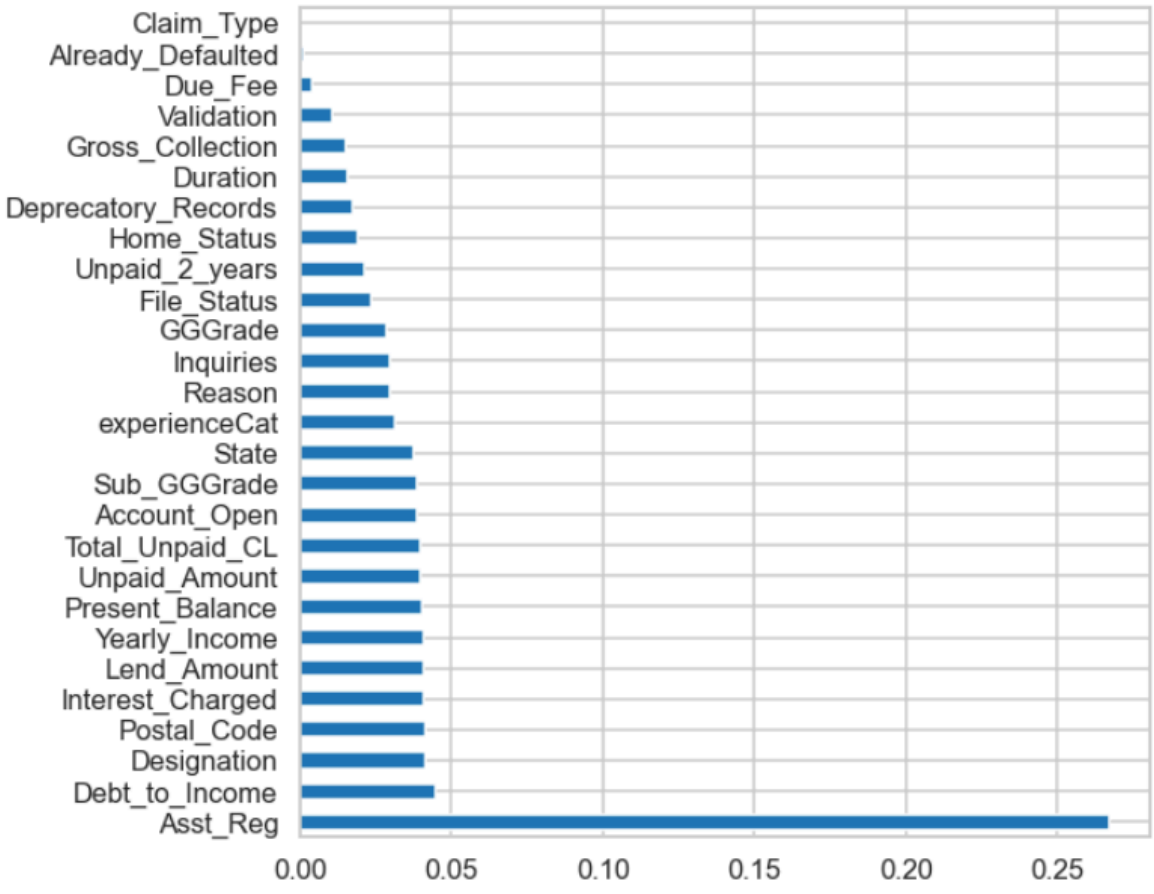


Findings: Due to limited space to capture the entire heatmap, this illustrates the relationship between each variable to determine which variables are important in predicting Default and to determine which variables are highly correlated.

In [263]:
```
X = trainFeatures
y = trainLabels
from sklearn.ensemble import ExtraTreesClassifier
import matplotlib.pyplot as plt
mdl = ExtraTreesClassifier()
mdl.fit(X,y)
print(mdl.feature_importances_)
#plot graph of feature importances for better visualization
featureImportance = pd.Series(mdl.feature_importances_, index=X.columns)
featureImportance.nlargest(28).plot(kind='barh', figsize=(9,9))
plt.show()
```

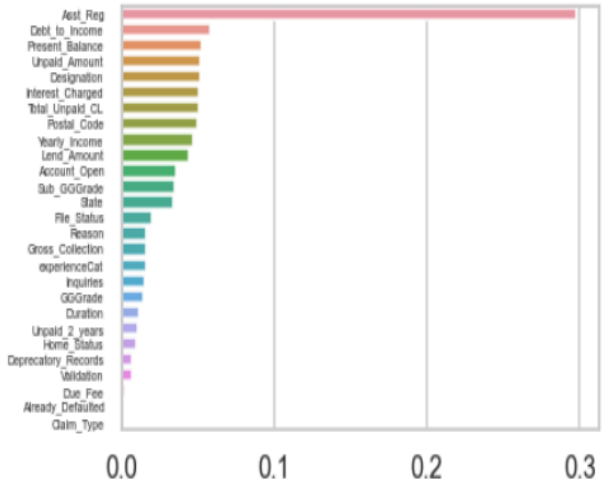
```
[2.67188224e-01 2.87853122e-02 1.05406070e-02 4.07058849e-02
 1.91638535e-02 2.11959150e-02 1.41054854e-03 4.16406033e-02
 4.50141494e-02 4.15357884e-02 4.08428683e-02 1.75514686e-02
 4.11249887e-02 2.95020205e-02 4.00451741e-02 1.51059871e-02
 3.84433456e-02 2.35316079e-02 3.77449192e-02 3.88437443e-02
 3.95127241e-02 1.58424418e-02 3.97106071e-02 2.98378642e-02
 4.54790277e-05 3.81335457e-03 3.13205186e-02]
```

Since we had plot the heat map, but the visualisation is not so user-friendly, we decided to use a few ML models to show the feature importances .



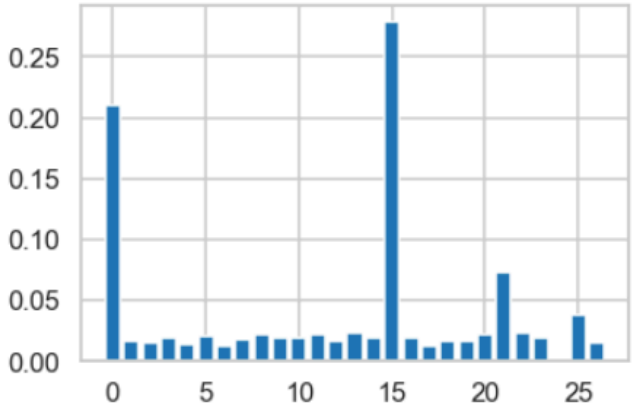
Findings: Compared with all the variables, Asst_Reg has the highest importance and claim type has the lowest importance.
Below are another 2 feature importances created using the Random Forest Classifier and the XGBClassifier.

```
from sklearn.ensemble import RandomForestClassifier

X= trainFeatures
y= trainLabels
rf = RandomForestClassifier()
rf.fit(X,y)
featureimportance = pd.Series(rf.feature_importances_, index=X.columns).sort_values(ascending=False)
sns.barplot(x=featureimportance, y=featureimportance.index)
plt.yticks(fontsize=7)
plt.show()
```



In [265]:
```
from xgboost import XGBClassifier
from matplotlib import pyplot
model = XGBClassifier()
model.fit(trainFeatures, trainLabels)
print(model.feature_importances_)
pyplot.bar(range(len(model.feature_importances_)), model.feature_importances_)
pyplot.show()
```

```
[0.21043317 0.01600799 0.01557069 0.01949524 0.01367614 0.02007148
 0.01188548 0.01731237 0.02232014 0.01885837 0.01895832 0.02255805
 0.01717215 0.02290189 0.01910659 0.27852145 0.01921534 0.01315306
 0.01626039 0.01587755 0.02232323 0.07288969 0.02381978 0.01879897
 0.         0.0375067  0.01530563]
```



j) Dropping of Columns

In [266]:
```
trainFeatures.columns
```

Out[266]:
```
Index(['Asst_Reg', 'GGGrade', 'Validation', 'Yearly_Income', 'Home_Status',
       'Unpaid_2_years', 'Already_Defaulted', 'Designation', 'Debt_to_Income',
       'Postal_Code', 'Lend_Amount', 'Deprecatory_Records', 'Interest_Charged',
       'Inquiries', 'Present_Balance', 'Gross_Collection', 'Sub_GGGrade',
       'File_Status', 'State', 'Account_Open', 'Total_Unpaid_CL', 'Duration',
       'Unpaid_Amount', 'Reason', 'Claim_Type', 'Due_Fee', 'experienceCat'],
      dtype='object')
```

In [267]:
```
trainFeatures.drop(['Sub_GGGrade'], axis=1, inplace=True)
testFeatures.drop(['Sub_GGGrade'], axis=1, inplace=True)
validationFeatures.drop(['Sub_GGGrade'], axis=1, inplace=True)

trainFeatures.drop(['Unpaid_Amount'], axis=1, inplace=True)
testFeatures.drop(['Unpaid_Amount'], axis=1, inplace=True)
validationFeatures.drop(['Unpaid_Amount'], axis=1, inplace=True)

trainFeatures.drop(['State'], axis=1, inplace=True)
testFeatures.drop(['State'], axis=1, inplace=True)
validationFeatures.drop(['State'], axis=1, inplace=True)

trainFeatures.drop(['Designation'], axis=1, inplace=True)
testFeatures.drop(['Designation'], axis=1, inplace=True)
validationFeatures.drop(['Designation'], axis=1, inplace=True)
```

We decided to drop the Sub_GGGrade column because it is highly correlated with GGGrade column. Unpaid_Amount is also dropped because it has a weak correlation with Default. The other two columns are also dropped because each of them has too many possible values and they could be referring to the same thing.

```
In [268]: trainFeatures.columns
```

```
Out[268]: Index(['Asst_Reg', 'GGGrade', 'Validation', 'Yearly_Income', 'Home_Status',
       'Unpaid_2_years', 'Already_Defaulted', 'Debt_to_Income', 'Postal_Code',
       'Lend_Amount', 'Deprecatory_Records', 'Interest_Charged', 'Inquiries',
       'Present_Balance', 'Gross_Collection', 'File_Status', 'Account_Open',
       'Total_Unpaid_CL', 'Duration', 'Reason', 'Claim_Type', 'Due_Fee',
       'experienceCat'],
      dtype='object')
```

```
In [269]: testFeatures.columns
```

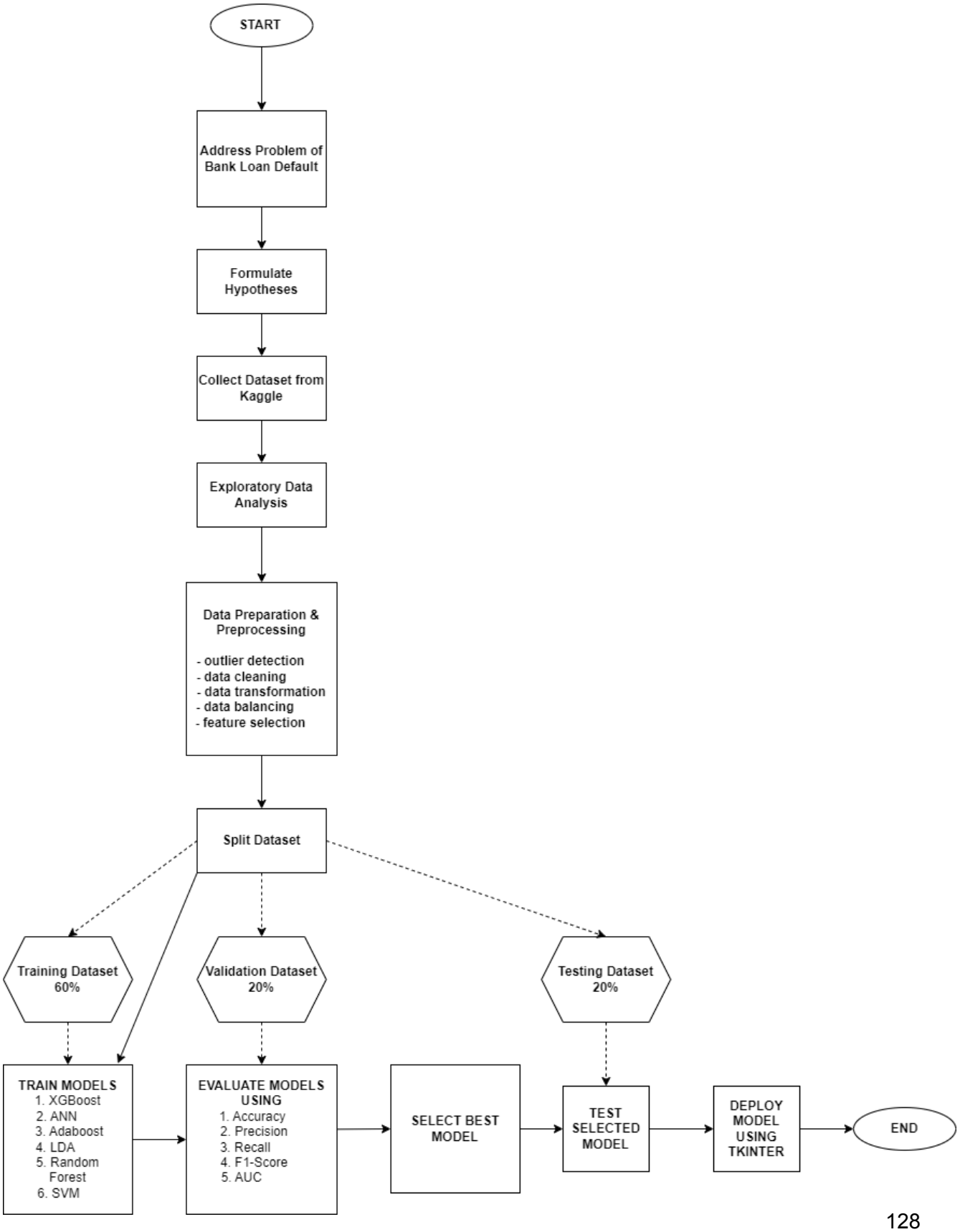
```
Out[269]: Index(['Asst_Reg', 'GGGrade', 'Validation', 'Yearly_Income', 'Home_Status',
       'Unpaid_2_years', 'Already_Defaulted', 'Debt_to_Income', 'Postal_Code',
       'Lend_Amount', 'Deprecatory_Records', 'Interest_Charged', 'Inquiries',
       'Present_Balance', 'Gross_Collection', 'File_Status', 'Account_Open',
       'Total_Unpaid_CL', 'Duration', 'Reason', 'Claim_Type', 'Due_Fee',
       'experienceCat'],
      dtype='object')
```

```
In [270]: validationFeatures.columns
```

```
Out[270]: Index(['Asst_Reg', 'GGGrade', 'Validation', 'Yearly_Income', 'Home_Status',
       'Unpaid_2_years', 'Already_Defaulted', 'Debt_to_Income', 'Postal_Code',
       'Lend_Amount', 'Deprecatory_Records', 'Interest_Charged', 'Inquiries',
       'Present_Balance', 'Gross_Collection', 'File_Status', 'Account_Open',
       'Total_Unpaid_CL', 'Duration', 'Reason', 'Claim_Type', 'Due_Fee',
       'experienceCat'],
      dtype='object')
```

These are the new variables that we are going to use to train, test and validate the models.

3.3 System flowchart/activity diagram



3.4 Proposed test plan/hypothesis

a) Test Plan:

ai) The Analysis, Preprocessing & Splitting of the Dataset

We plan to first analyse the dataset to gain a better understanding of the data and then, preprocess the dataset, which involves data cleaning, data transformation, etc. After that, we will split the dataset into 3 parts: (1) Training set 60% (2) Validation set 20% (3) Test set 20%. The training set is used to train the ML model. The validation set is used to test each different model in order to select the best performing model. The selected best model is then re-tested on the test set to re-confirm the results obtained when tested against the validation set. The models that we will be evaluating are XGBoost, Artificial Neural Network, Linear Discriminant Analysis, Adaboost, Random Forest and Support Vector Machine. Where the best performing model will be using the testing set to evaluate the model. Where we use the best model for deployment purposes.

aii) Evaluation of the Models Based on:

1. Accuracy = (TP + TN) / (TP + TN + FP + FN)
2. Recall = TP/(TP + FN)
3. Precision = TP/(TP + FP)
4. F-score = 2 x (Precision x Recall) / (Precision + Recall)
5. ROC (AUC)

aiii) Deployment of Selected Model (User Interface)

We plan to develop a desktop data entry program (written using Python Tkinter) to allow an end-user to enter a single record of loan and customer information whereupon the data will be fed into the Machine Learning model, which will return a prediction. This program also needs to be tested by us.

b) Hypotheses:

H1: XGBoost is the best performing model among all the models being evaluated

H2: There is a negative correlation between (loan) Default and the Value of all the assets registered under the borrower

H3: A model that is tuned performs better than when using default hyperparameters' values.

4. Result

4.1 Results

a) XGBoost Without Hyperparameter Tuning

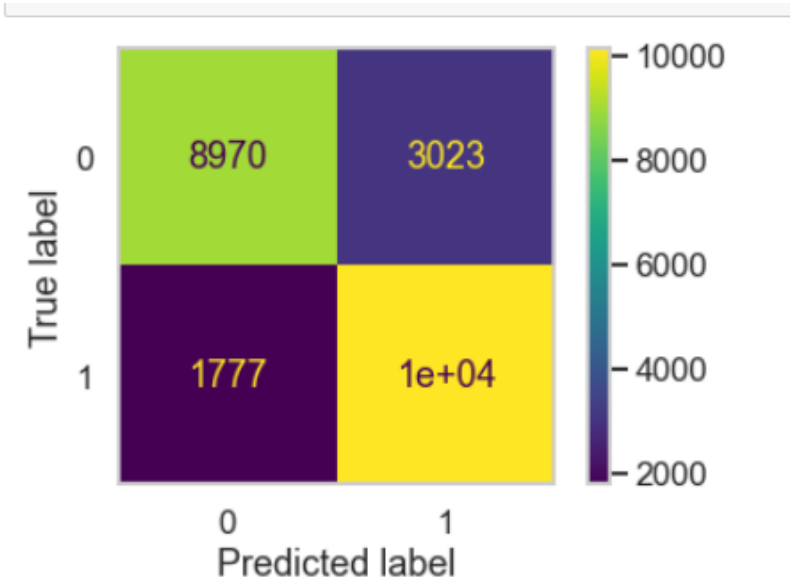
i) Results of the 4 performance metrics (Accuracy, Precision, F1-Score, Recall)

```
accuracy = accuracy_score(validationLabels, xgprediction)
print("Accuracy: %.2f%%" % (accuracy * 100.0))

precision = precision_score(validationLabels, xgprediction)
print("Precision: %.2f%%" % (precision * 100.0))

f1score = f1_score(validationLabels, xgprediction)
print("F1 score: %.2f%%" % (f1score * 100.0))

recall = recall_score(validationLabels, xgprediction)
print("Recall: %.2f%%" % (recall * 100.0))
```

```
Accuracy: 79.93%
Precision: 77.04%
F1 score: 80.86%
Recall: 85.09%
```

As shown above, the XGBoost Without Tuning model has 79.93% accuracy, 77.04% precision, 80.86% F1-score and 85.09% recall. Their scores are not too low.

ii) Results of the Confusion Matrix & Classification Report



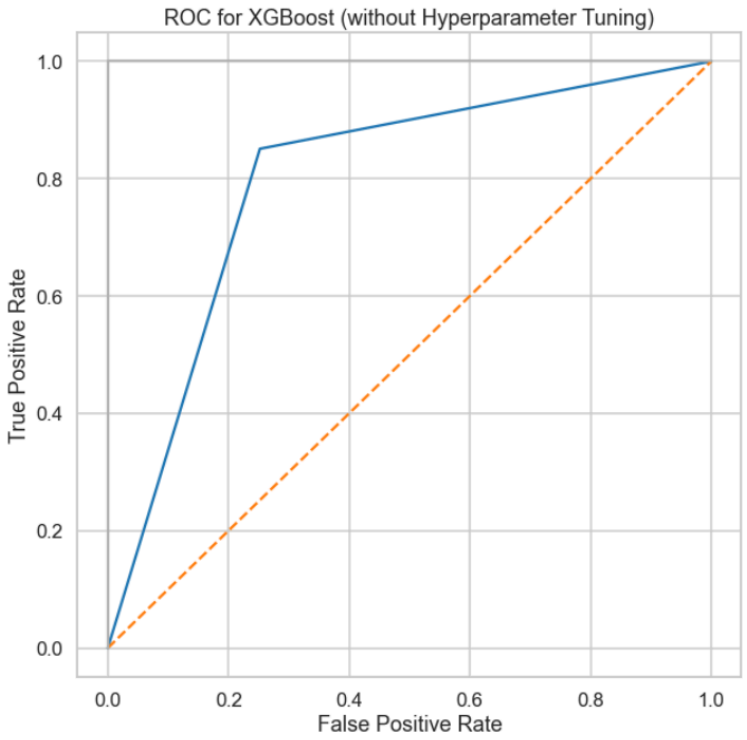
8970 Non-default cases are correctly predicted as Non-default. 1000 Defaulted cases are correctly predicted as default. 3023 Non-default cases are wrongly classified as defaulted. 1777 defaulted cases are wrongly classified as non-defaulted.

	precision	recall	f1-score	support
0	0.83	0.75	0.79	11993
1	0.77	0.85	0.81	11918
accuracy			0.80	23911
macro avg	0.80	0.80	0.80	23911
weighted avg	0.80	0.80	0.80	23911

iii) Result of ROC and AUC

```
print('roc_auc_score for XGBoost without Hyperparameter Tuning: ', roc_auc_score(validationLabels, xgprediction))
```

```
roc_auc_score for XGBoost without Hyperparameter Tuning:  0.7994170489086693
```



If we round up the AUC score to 0.8, the model would be considered as good in distinguishing between negative and positive instances.

b) XGBoost With Hyperparameter Tuning

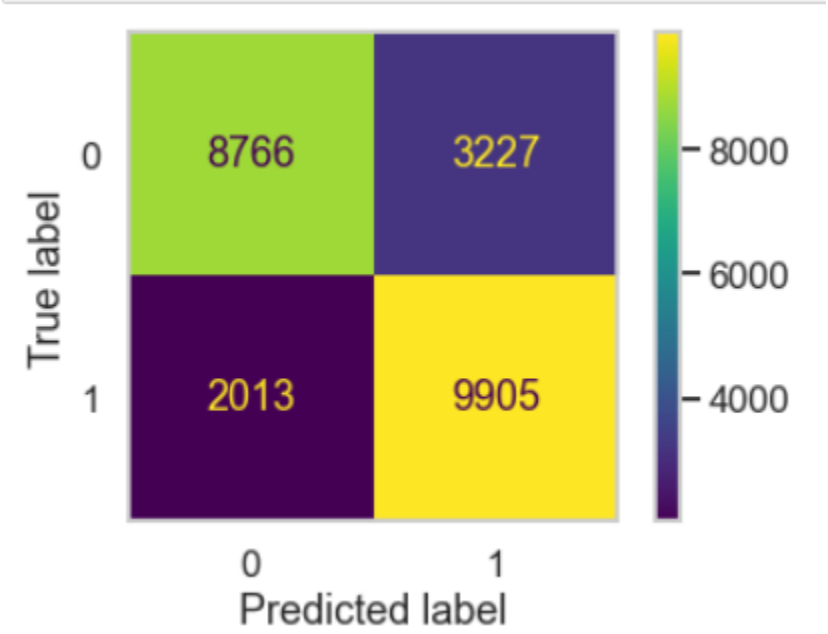
i) Results of the 4 performance metrics (Accuracy, Precision, F1-Score, Recall)

```
accuracy = accuracy_score(validationLabels, xgprediction2)
print("Accuracy: %.2f%%" % (accuracy * 100.0))

precision = precision_score(validationLabels, xgprediction2)
print("Precision: %.2f%%" % (precision * 100.0))

f1score = f1_score(validationLabels, xgprediction2)
print("F1 score: %.2f%%" % (f1score * 100.0))

recall = recall_score(validationLabels, xgprediction2)
print("Recall: %.2f%%" % (recall * 100.0))
```

```
Accuracy: 78.09%
Precision: 75.43%
F1 score: 79.08%
Recall: 83.11%
```

As shown above, the XGBoost with Tuning model has 78.09% accuracy, 75.43% precision, 79.08% F1-Score and 83.11% recall.

ii) Results of the Confusion Matrix & Classification Report



8766 non-default cases are correctly classified as non-default. 9905 defaulted cases are correctly classified as defaulted. 3227 non-default cases are wrongly classified as defaulted cases. 2013 defaulted cases are wrongly classified as non-default cases.

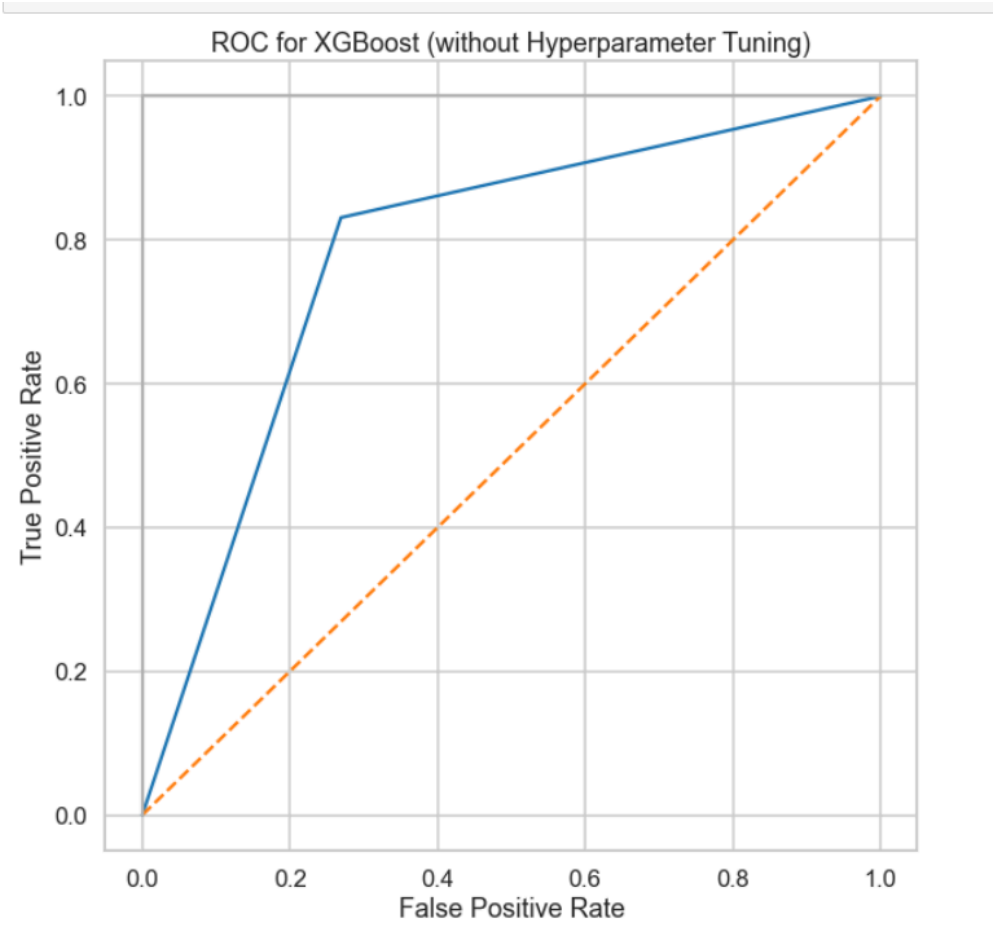
	precision	recall	f1-score	support
0	0.81	0.73	0.77	11993
1	0.75	0.83	0.79	11918
accuracy			0.78	23911
macro avg	0.78	0.78	0.78	23911
weighted avg	0.78	0.78	0.78	23911

iii) Result of ROC and AUC

```
print('roc_auc_score for XGBoost without Hyperparameter Tuning: ', roc_auc_score(validationLabels, xgprediction2))
```

```
roc_auc_score for XGBoost without Hyperparameter Tuning:  0.7810110975822768
```

Since its AUC score is below 0.8, it is considered as average in distinguishing between positive and negative instances.


ROC for XGBoost (without Hyperparameter Tuning)
True Positive Rate
False Positive Rate
0.0
0.2
0.4
0.6
0.8
1.0

c) ANN Without Hyperparameter Tuning

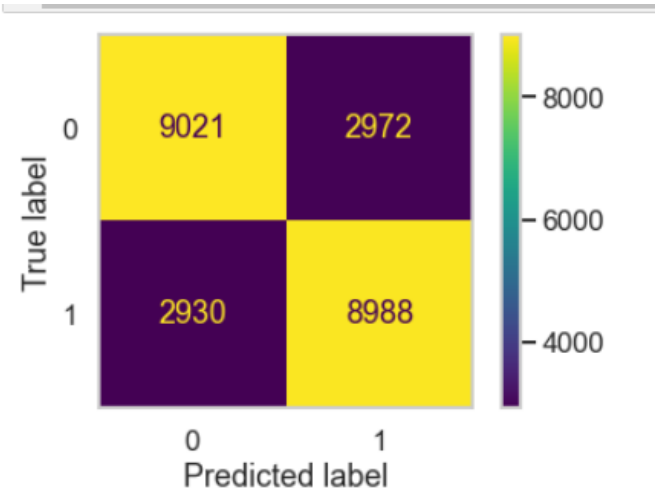
i) Results of the 4 performance metrics (Accuracy, Precision, F1-Score, Recall)

```
annAccuracy = accuracy_score(validationLabels, annPrediction)
print("Accuracy: %.2f%%" % (annAccuracy * 100.0))

annPrecision = precision_score(validationLabels, annPrediction)
print("Precision: %.2f%%" % (annPrecision * 100.0))

annF1score = f1_score(validationLabels, annPrediction)
print("F1 score: %.2f%%" % (annF1score * 100.0))

annRecall = recall_score(validationLabels, annPrediction)
print("Recall: %.2f%%" % (annRecall * 100.0))
```

```
Accuracy: 75.32%
Precision: 75.15%
F1 score: 75.28%
Recall: 75.42%
```

As shown above, the ANN model has 75.32% accuracy, 75.15% precision, 75.28% F1-score and 75.42% recall. The scores are not too low.

ii) Results of the Confusion Matrix & Classification Report



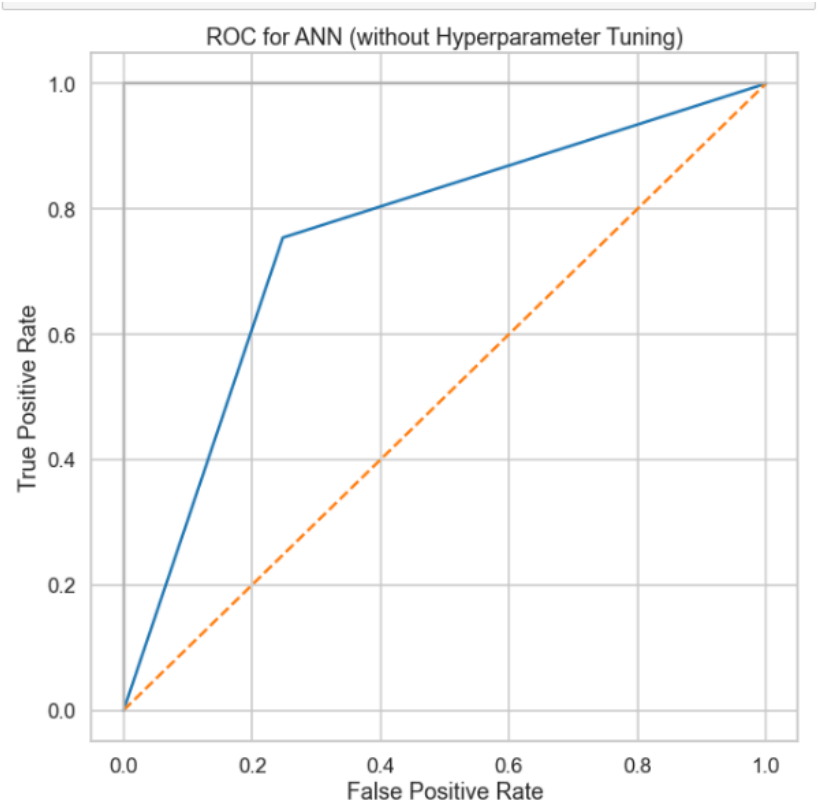
9021 non-default cases are correctly classified as non-default. 8988 defaulted cases are correctly classified as defaulted. 2972 non-default cases are wrongly classified as defaulted. 2930 default cases are wrongly classified as non-default.

	precision	recall	f1-score	support
0	0.75	0.75	0.75	11993
1	0.75	0.75	0.75	11918
accuracy			0.75	23911
macro avg	0.75	0.75	0.75	23911
weighted avg	0.75	0.75	0.75	23911

iii) Result of ROC and AUC

```
print('roc_auc_score for ANN without Hyperparameter Tuning: ', roc_auc_score(validationLabels, annPrediction))
```

```
roc_auc_score for ANN without Hyperparameter Tuning:  0.7531710791131488
```



Since the AUC score is below 0.8, it is not considered as good in distinguishing between positive and negative instances.

d) ANN With Customised Hyperparameter Values

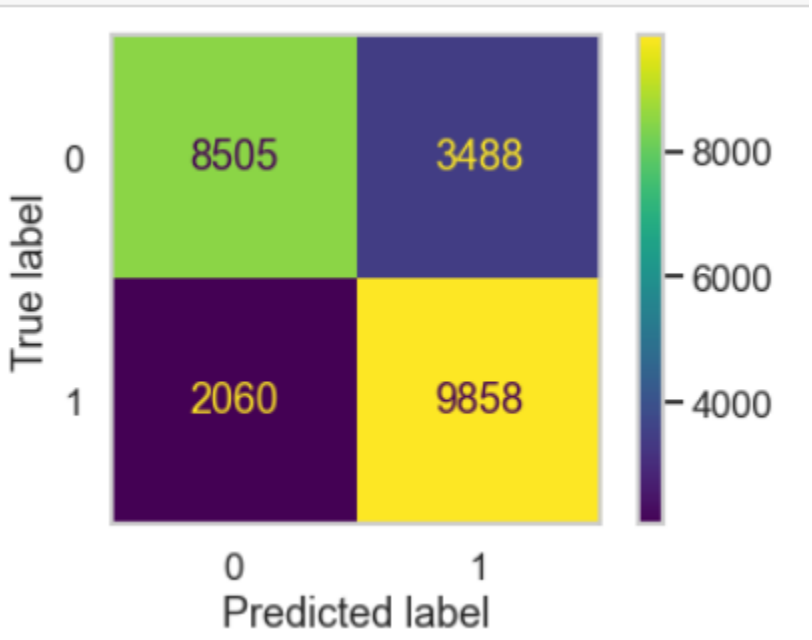
i) Results of the 4 performance metrics (Accuracy, Precision, F1-Score, Recall)

```
In [342]:
annAccuracy = accuracy_score(validationLabels, annPrediction2)
print("Accuracy: %.2f%%" % (annAccuracy * 100.0))

annPrecision = precision_score(validationLabels, annPrediction2)
print("Precision: %.2f%%" % (annPrecision * 100.0))

annF1score = f1_score(validationLabels, annPrediction2)
print("F1 score: %.2f%%" % (annF1score * 100.0))

annRecall = recall_score(validationLabels, annPrediction2)
print("Recall: %.2f%%" % (annRecall * 100.0))
```

```
Accuracy: 76.80%
Precision: 73.86%
F1 score: 78.04%
Recall: 82.72%
```

As shown above, the ANN with customised hyperparameter model has 76.80% accuracy, 73.86% precision, 78.04% F1-score and 82.72% recall.

ii) Results of the Confusion Matrix & Classification Report



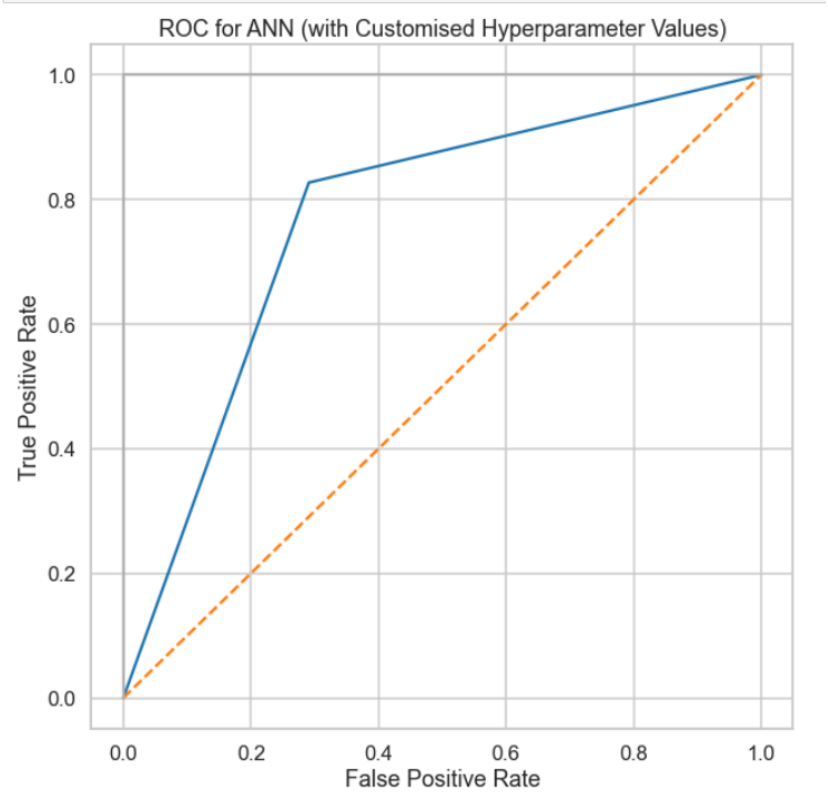
8505 non-default cases are correctly classified as non-default. 9858 default cases are correctly classified as defaulted. 3488 non-default cases are wrongly classified as defaulted. 2060 defaulted cases are wrongly classified as non-default.

	precision	recall	f1-score	support
0	0.81	0.71	0.75	11993
1	0.74	0.83	0.78	11918
accuracy			0.77	23911
macro avg	0.77	0.77	0.77	23911
weighted avg	0.77	0.77	0.77	23911

iii) Result of ROC and AUC

```
print('roc_auc_score for ANN with Customised Hyperparameter Values: ', roc_auc_score(validationLabels, annPrediction2))
```

```
roc_auc_score for ANN with Customised Hyperparameter Values:  0.7681579427793696
```



Since its AUC score is below 0.8, it is considered as average in distinguishing between positive and negative instances.

e) Adaboost Without Hyperparameter Tuning
i) Results of the 4 performance metrics (Accuracy, Precision, F1-Score, Recall)

After we had done the splitting the data into training, testing and validation parts respectively, Ada boost is used to evaluate the model by comparing the result and to find the accuracy, precision, f1-score and recall store to be able to find out which model performs the best for the dataset to help to deploy into our user interface design and also to determine whether the model fits nicely with the dataset for the assignment.

But before, we tested the model by tuning hyperparameters. We had to do the default hyperparameter to observe the relationship between the adaboost model with the dataset to determine. The results are shown below.

```
print("Accuracy:", accuracy_score(validationLabels, y_pred_adab))
print("Precision:", precision_score(validationLabels, y_pred_adab))
print("F1-score:", f1_score(validationLabels, y_pred_adab))
print("Recall:",recall_score(validationLabels,y_pred_adab))
```

```
Accuracy: 0.7482748525783113
Precision: 0.7301958947943495
F1-score: 0.7566212445918078
Recall: 0.7850310454774291
```

As the illustration above, these are the results tested for ada boost with just using the default hyperparameters. With an accuracy of 74.83%, Precision of 73.20%, F1-score of 75.66% and recall of 78.50% Through the result generated the ada boost still performs significantly above average as the result has been generated. But as compared with other models, ada boost is not considered to perform badly as some models may not work very well with some dataset and not add dataset can run certain models but depending on the scenario of the dataset. This shows that above 70% of the model can fit significantly. As this is with just the default hyper parameters.

Since the above result has been executed by finding the default hyper parameters, we now run the tuning parameter as illustrated below at section f

ii) Results of the Classification Report

	precision	recall	f1-score	support
0	0.77	0.71	0.74	11993
1	0.73	0.79	0.76	11918
accuracy			0.75	23911
macro avg	0.75	0.75	0.75	23911
weighted avg	0.75	0.75	0.75	23911

As shown here this is the result for the classification task. Where all the tests have been projected to here. As usual, we have the accuracy, precision, recall and f1-score. The support indicates the number of defaults and not yet defaulted by support based on the total as shown above. About 0.75 for weight and macro are averagely distributed for the classification report.

f) Adaboost with Hyperparameter Tuning

i) Results of the 4 performance metrics (Accuracy, Precision, F1-Score, Recall)

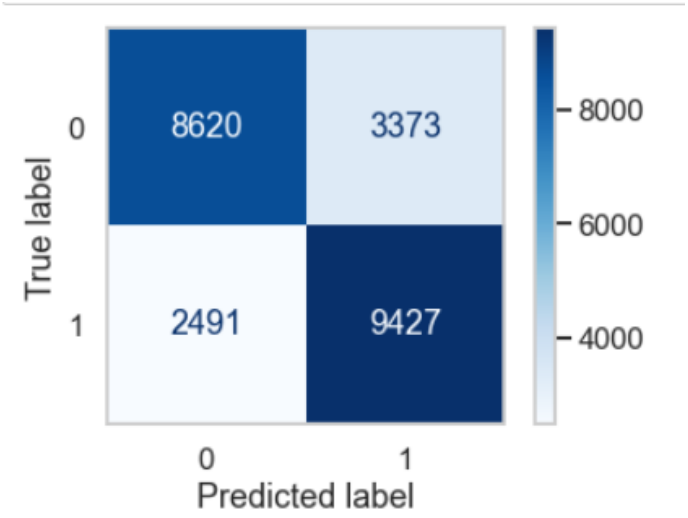
```
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report

print("Accuracy:", accuracy_score(validationLabels, y_pred_adaboost))
print("Precision:", precision_score(validationLabels, y_pred_adaboost))
print("F1-score:", f1_score(validationLabels, y_pred_adaboost))
print("Recall:",recall_score(validationLabels,y_pred_adaboost))
```

```
Accuracy: 0.7547572247082932
Precision: 0.736484375
F1-score: 0.762763977668096
Recall: 0.7909884208759859
```

As shown above the result for tuned hyper parameters is not very significant compared to default hyperparameters as we use grid search cv, to find the best parameters for the best but unfortunately the result will not vary a lot as shown above. Most of the accuracy testing gone up by 0.1, although it is not increase drastically but it increased slightly based on the result. As this still considers better as compared to the default hyper parameter tuning.

Now since the result has been plotted, we now plot the confusion matrix and the roc curve to help understand the tuned hyper parameters more better through visualisation graphs.

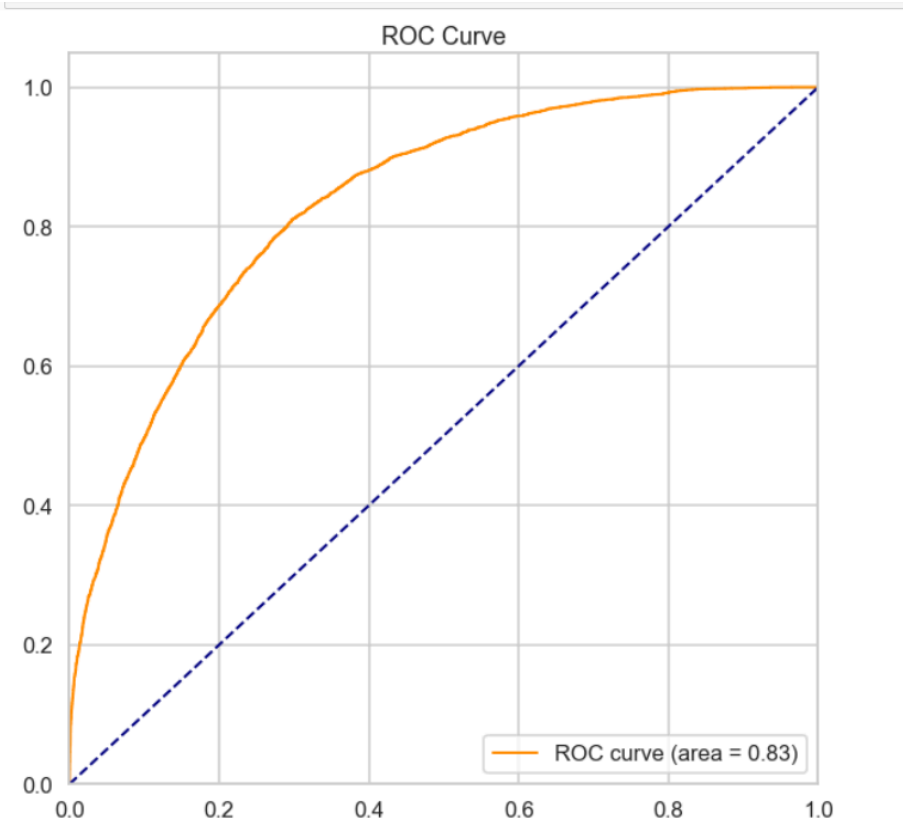
ii) Results of the Confusion Matrix & Classification Report



	precision	recall	f1-score	support
0	0.78	0.72	0.75	11993
1	0.74	0.79	0.76	11918
accuracy			0.75	23911
macro avg	0.76	0.75	0.75	23911
weighted avg	0.76	0.75	0.75	23911

iii) Result of ROC and AUC

-As shown here, this is the result generated for the confusion matrix for adaBoost that has tuned the hyper parameter.



As shown, this is the roc curve for ada boost that has tuned the hyper parameters. In general ROC curve represents the performance of a binary classifier with different sets of threshold decisions. Where the true positive will be plotted against the false positive. But generally the roc curve is labelled between 0 to 1. Where 0 is the lowest and 1 is the highest. As shown the dark blue dotted line represents the threshold value. Where in another word a higher threshold value results in a lower recall value and leads to fewer false positive cases.

g) Linear Discriminant Analysis Without Hyperparameter Tuning

i) Results of the 4 performance metrics (Accuracy, Precision, F1-Score, Recall)

```
y_pred_lda = lda.predict(validationFeatures)

print("Accuracy:", accuracy_score(validationLabels, y_pred_lda))
print("Precision:", precision_score(validationLabels, y_pred_lda))
print("F1-score:", f1_score(validationLabels, y_pred_lda))
print("Recall:",recall_score(validationLabels,y_pred_lda))
```

```
Accuracy: 0.7416670151812973
Precision: 0.7223297962977306
F1-score: 0.7512183334004592
Recall: 0.7825138446047994
```

As shown here about 74.17% is for the accuracy, 72.23% is for the precision, 75.12 is for the F1-score and recall is about 78.25%. This result is slightly higher than the adaBoost, as explained earlier.

ii) Results of the Confusion Matrix & Classification Report

	precision	recall	f1-score	support
0	0.76	0.70	0.73	11993
1	0.72	0.78	0.75	11918
accuracy			0.74	23911
macro avg	0.74	0.74	0.74	23911
weighted avg	0.74	0.74	0.74	23911

As illustrated above, this is the result for the default hyper parameter for linear discriminant analysis. The results composed have very similar result as the one with ada boost. This shows LDA has also had above average performance. Although most of the result is higher than .70 we have to try to evaluate the tuned hyper parameter to examine whether the model can be still improved.

h) Linear Discriminant Analysis With Hyperparameter Tuning

i) Results of the 4 performance metrics (Accuracy, Precision, F1-Score, Recall)

```
lda = LinearDiscriminantAnalysis(shrinkage = None, solver = 'lsqr')
lda = lda.fit(trainFeatures, trainLabels)
y_gs_lda = lda.predict(validationFeatures)

print("Accuracy:", accuracy_score(validationLabels, y_gs_lda))
print("Precision:", precision_score(validationLabels, y_gs_lda))
print("F1-score:", f1_score(validationLabels, y_gs_lda))
print("Recall:",recall_score(validationLabels,y_gs_lda))
```

```
Accuracy: 0.7416670151812973
Precision: 0.7223297962977306
F1-score: 0.7512183334004592
Recall: 0.7825138446047994
```

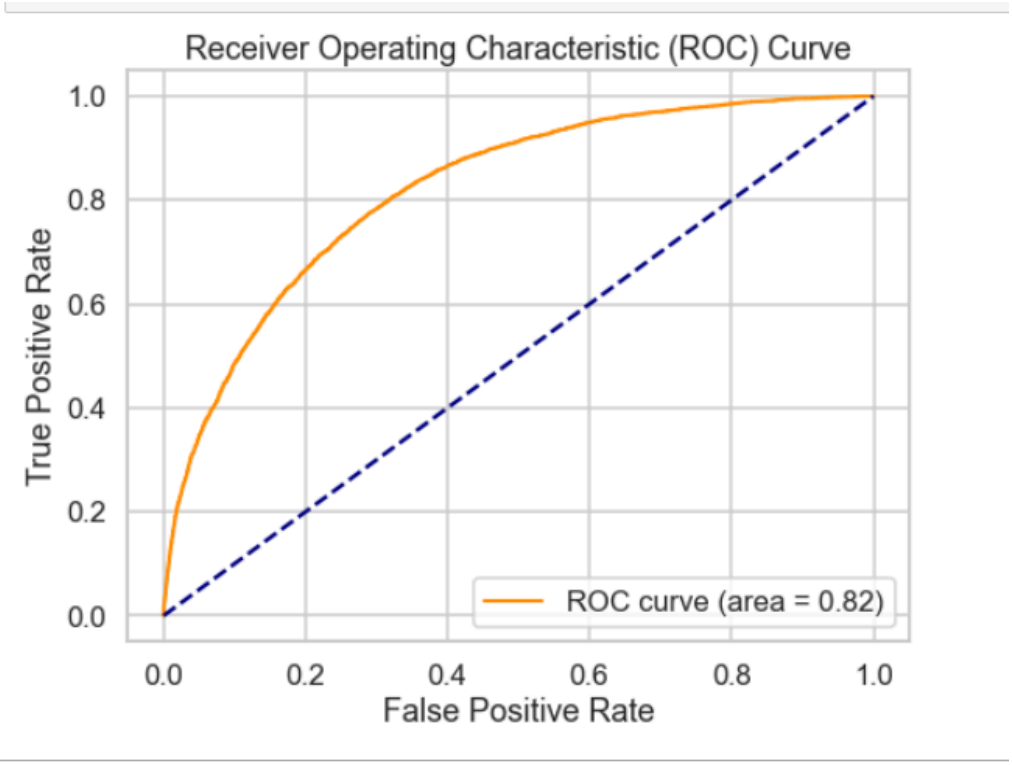
The result shown after we have tuned hyper parameter for least discriminant analysis, the result is the same with the default hyper parameter. Although we have find the best parameter for least discriminant analysis, the result is not always will improve as some of the models will also decrease and some models may have the same result as the default hyper parameter

ii) Results of the Confusion Matrix & Classification Report

	precision	recall	f1-score	support
0	0.76	0.70	0.73	11993
1	0.72	0.78	0.75	11918
accuracy			0.74	23911
macro avg	0.74	0.74	0.74	23911
weighted avg	0.74	0.74	0.74	23911

As shown this is the result generated for the accuracy readings for linear discriminant analysis result. The result improved slightly while some accuracy dropped slightly. As this is due to the parameters and the values contained inside the dataset.

iii) Result of ROC and AUC



This is the roc curve for the turned parameters for Linear Discriminant Analysis. As compared to the roc curve with adaboost, we can observe that the result is not that much of a difference in terms of the results generated by the both models. As for adaboost the roc curve reading is 0.83 as for linear discriminant analysis has a roc curve reading of 0.82. Where adaboost is slightly better of 0.01. As the dotted line increases the way, the True positive and false positive decision is still the same as it moves in a straight line all the way.

i) RandomForest Without Hyperparameter Tuning
i) Results of the 4 performance metrics (Accuracy, Precision, F1-Score, Recall)

```
accuracy = accuracy_score(validationLabels, rfModelPrediction)
precision = precision_score(validationLabels, rfModelPrediction)
fiscore = f1_score(validationLabels, rfModelPrediction)
recall = recall_score(validationLabels, rfModelPrediction)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
print("Precision: %.2f%%" % (precision * 100.0))
print("F1 Score: %.2f%%" % (fiscore * 100.0))
print("Recall: %.2f%%" % (recall * 100.0))
```

```
Accuracy: 92.05%
Precision: 88.89%
F1 Score: 92.34%
Recall: 96.06%
```

Accuracy means the number of correct predictions made by a model out of the total forecast. The accuracy of this model is 92.05%, which means that it correctly identifies 92 out of 100 default cases.

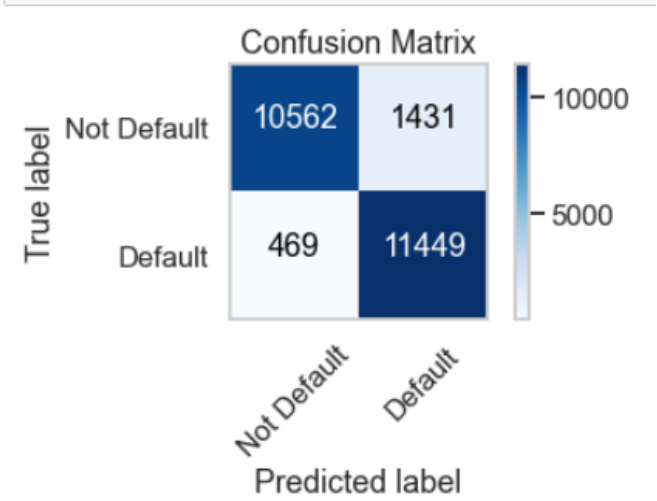
Precision calculates how many of the predicted positives are positive. In short, it is used to test the reliability of the system. The formula for Precision is Precision = True Positives/ (True Positives + False positives). True positives mean correctly identifying someone who would default on the loan. False positives mean it wrongly classifies someone as a loan, but it does not. This is used to test the reliability of the positive prediction. The Precision ranges from 0 to 1. 1 means it correctly identifies all loan defaulters and has no false positives (100%). So, a model has a Precision of 0, which wrongly classifies all non-defaulters as loan defaulters. So, this model has a relatively high precision score of 88.89%, which correctly identifies most loan defaulters. This is crucial for banks as falsely accusing customers of potential default (false positive) would negatively impact customer relationships and missed business opportunities.

Recall is used to test the true positive rate, also known as sensitivity. Recall calculates how many positive cases the model missed; it doesn't count how many issues it wrongly classifies negative (no default) as positive (default). High Recall is essential, especially if the bank wants to minimise risk at the cost of mistakenly flagging some non-defaulters. Generally, the recall score is significant for the model, as failing to identify a defaulter (false negative) would result in financial loss from the missed defaulter. In this model, the recall rate is 96.06%, which means this model can capture almost all of the defaulters, and only 3.94% of the actual defaulters remain unnoticed. The Recall is significant for the bank as the lending loan is the source of income for the bank, so if the model cannot capture the defaulter, it would result in financial loss.

The F1 score is the harmonic mean of Precision and Recall. This is used to seek the balance and trade-off between Precision and Recall, Where it can only achieve a high f1 score if both

Recall and Precision are high. A high f1 score means it can achieve an outstanding balance between Precision (Not wrongly identifying someone as a loan defaulter) and recall (getting as many loan defaulters as possible. In this model, 92.34% indicates this model achieves a good balance between Precision and Recall.

ii) Results of the Confusion Matrix & K-fold cross-validation



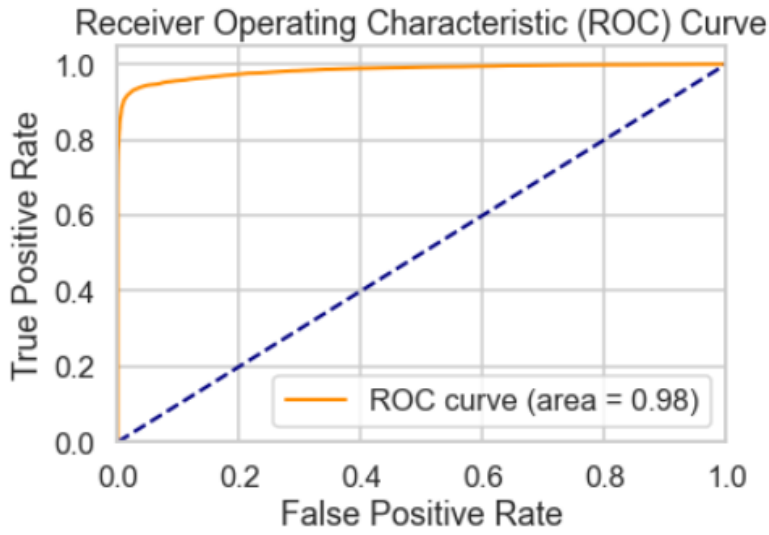
The component of Confusion can be split into four parts: True positive (TP), which is located at the top left, which means the model predicted as positive and the true label is also positive; false positive (FP), also referred to as Type I error located at the top right which is predicted as positive but the true label is negative. The False negatives(FN), also called Type II error, are predicted as positive, but in reality, it is positive. The true negative (TN) is located at the bottom right, which means the model can identify negative classes, and the true label is also negative. In this case, TN and TP are high, which indicates it can predict and identify default and not default loans. The Type I error is higher than the Type II error, meaning it is more likely to lend money to someone who ends up defaulting than to deny a loan to someone who can repay.

```
from sklearn.model_selection import cross_val_score
val_score = cross_val_score(estimator=rfModel, X = trainFeatures, y=trainLabels, cv=10)
print("Model Accuracy Score: {:.2f} %".format(val_score.mean()*100))
print("Std. Dev: {:.2f} %".format(val_score.std()*100))
```

```
Model Accuracy Score: 91.24 %
Std. Dev: 0.27 %
```

Based on K-fold cross-validation, the Model correctly predicted whether a loan would default at a 91.24% successful rate with a Standard deviation of 0.27%, which means The Model is able to perform at a relatively high accuracy with a high consistency and the accuracy score is tightly clustered around the means. This means that the Model is reliable.

iii) Result of ROC and AUC



The ROC curve approaches the top left corner of the plot, which means it can achieve a high True Positive Rate (TPR) with a low False positive rate (FPR), Which also indicates that it has a high discrimination ability, which means the classifier can distinguish positive and negative classes correctly. This suggests that the model make very few mistakes in terms of false positives and false negative. The model is performing exceptionally well as it is very close to the perfect score 1.

j) RandomForest With Hyperparameter Tuning

i) Results of the 4 performance metrics (Accuracy, Precision, F1-Score, Recall)

```
# 5. Display the results
accuracy_rs = accuracy_score(validationLabels, random_search_prediction)
print("Random Search - Accuracy: %.2f%%" % (accuracy_rs * 100.0))

precision_rs = precision_score(validationLabels, random_search_prediction)
print("Random Search - Precision: %.2f%%" % (precision_rs * 100.0))

fiscore_rs = f1_score(validationLabels, random_search_prediction)
print("Random Search - F1 Score: %.2f%%" % (fiscore_rs * 100.0))

recall_rs = recall_score(validationLabels, random_search_prediction)
print("Random Search - Recall: %.2f%%" % (recall_rs * 100.0))

print("\nBest Parameters from Random Search: ", random_search.best_params_)
```

```
Random Search - Accuracy: 86.76%
Random Search - Precision: 82.25%
Random Search - F1 Score: 87.57%
Random Search - Recall: 93.63%
```

The accuracy of this model is 86.76%, are able to identify approximately 87 out of 100 default cases. The precision shows that the Model is able to identify most loan defaulters with 82.25% rate. The recall value of 93.64 % means it able to capture 93.64% of all the defaulter in the dataset. The F1 score indicate that it achieve a great balance between precision and recall value.

k) SVM Without Hyperparameter Tuning

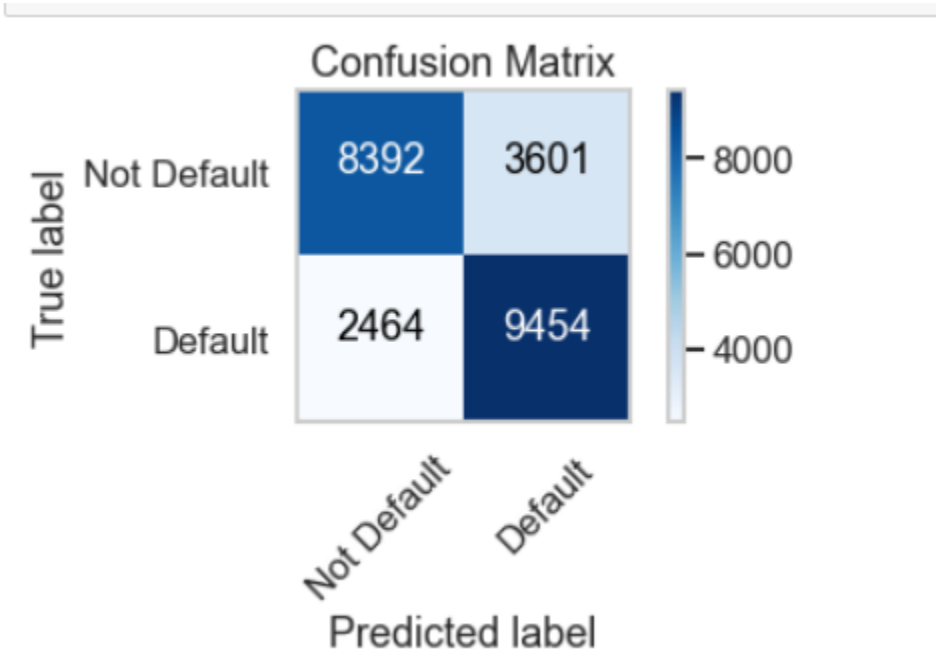
i) Results of the 4 performance metrics (Accuracy, Precision, F1-Score, Recall)

```
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score

accuracy = accuracy_score(validationLabels, svcModelPrediction)
print("Accuracy: %.2f%%" % (accuracy * 100.0))

precision = precision_score(validationLabels, svcModelPrediction)
print("Precision: %.2f%%" % (precision * 100.0))

fiscore = f1_score(validationLabels, svcModelPrediction)
print("F1 Score: %.2f%%" % (fiscore * 100.0))

recall = recall_score(validationLabels, svcModelPrediction)
print("Recall: %.2f%%" % (recall * 100.0))
```

```
Accuracy: 74.64%
Precision: 72.42%
F1 Score: 75.71%
Recall: 79.33%
```

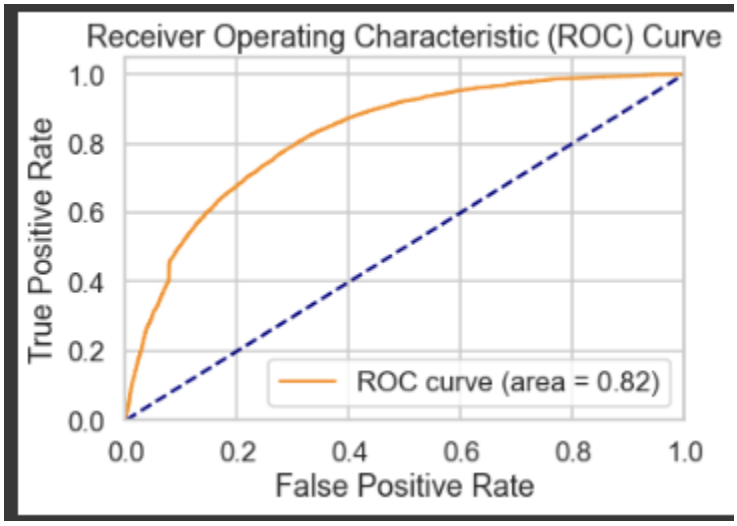
The Accuracy score of 74.64% indicates that it can predict well above average and the defaulter correctly. The precision of 72.42% suggests that this model is reliable, and the recall rate of 79.33 % indicates that it can predict most of the defaulter; the F1 score of 75.71 suggests that it can achieve a good balance between precision and recall. It does not sacrifice the precision for recall or vice versa. This model achieves an outstanding balance between these two.

ii) Results of the Confusion Matrix & Classification Report



Based on the confusion matrix, it can identify most of the default cases, 8392 cases, and identify the defaulter with 9454 cases. It predicts it as a defaulter, but in true label, it is a defaulter. Cases are 3601 and 2464 cases where it is predicted as a defaulter, but in true label, it is a defaulter.

iii) Result of ROC and AU

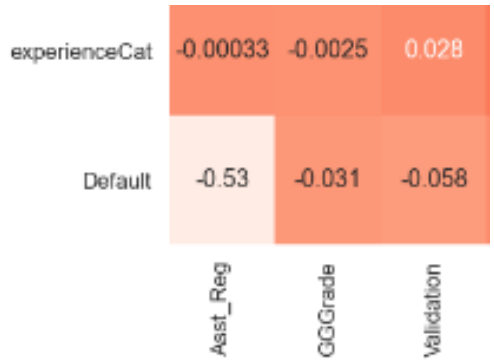


The ROC curve approaches the top left corner of the plot, which means it can achieve a high True Positive Rate (TPR) with a low False positive rate (FPR), Which also indicates that it has a high discrimination ability, which means the classifier can distinguish positive and negative classes correctly. The result is perform well above average with the ROC curve area of 0.82.

Accept or Reject Hypotheses

1. We reject the hypothesis, H1, which stated that XGBoost is the best performing model among all the models being evaluated. This is because the Random Forest model, especially the one with default hyperparameters, performed the best in terms of accuracy, precision, recall, f1-score and AUC.

2. We accept the hypothesis, H2, that there is a negative correlation between (loan) Default and the Value of all the assets registered under the borrower. This is because the correlation coefficient between them is -0.53 (The full heatmap can be found in chapter 3).

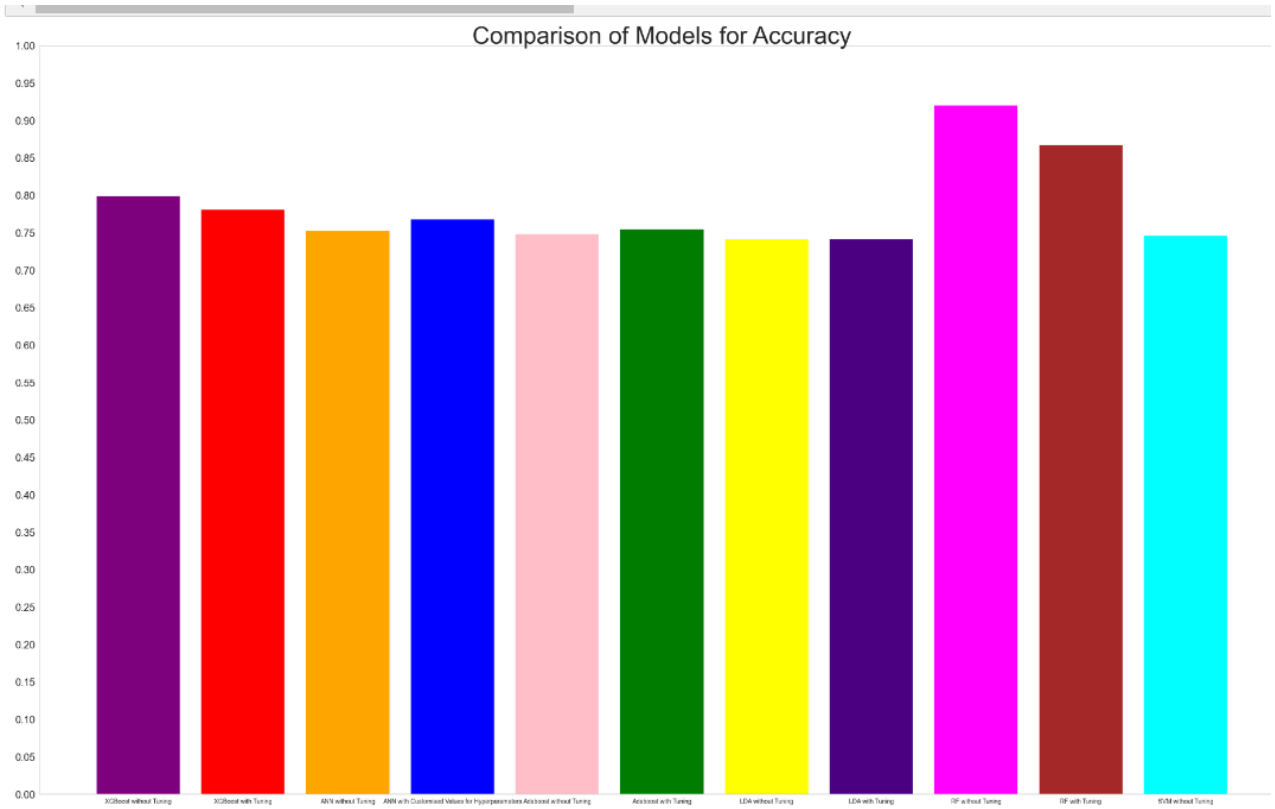


3. We reject the hypothesis, H3, which stated that a model that is tuned performs better than when using default hyperparameters' values. This is based on the results of the performance metrics for the models used in this study. For example, the "Random Forest with default hyperparameters' values" model performed better than the "Random Forest with Tuning" model.

4.2 Discussion/Interpretation

Legend:
1. XGBoost without Tuning (Purple)
2. XGBoost with Tuning (Red)
3. ANN without Tuning (Orange)
4. ANN with Custom Values for Hyperparameters (Blue)
5. Adaboost without Tuning (Pink)
6. Adaboost with custom Tuning (Green)
7. LDA without Tuning (Yellow)
8. LDA with Tuning (Indigo)
9. RF without Tuning (magenta)
10. RF with Tuning (brown)
11. SVM without Tuning (cyan)

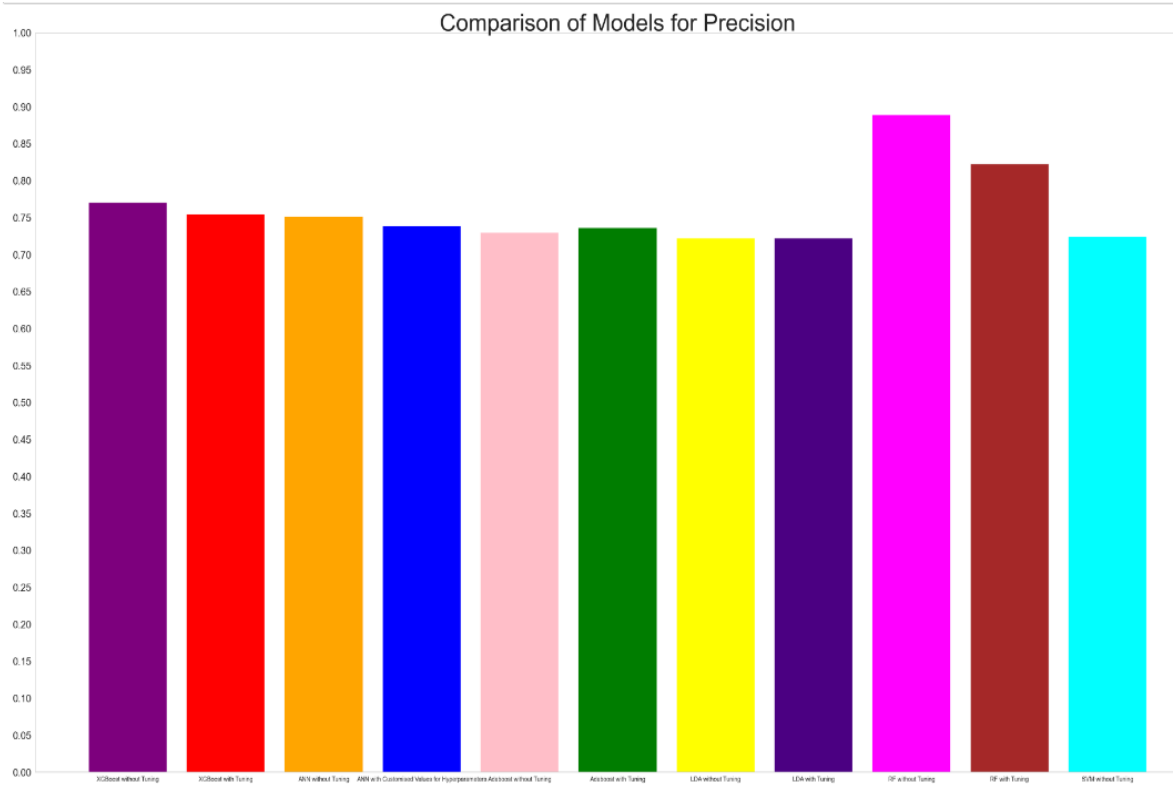
a) Comparison of Models Based on Accuracy



```
Results for Accuracy
------------------------------
XGBoost without Tuning                          : 79.93%
XGBoost with Tuning                             : 78.09%
ANN without Tuning                              : 75.32%
ANN with Customised Values for Hyperparameters: 76.80%
Adaboost without Tuning                         : 74.83%
Adaboost with Tuning                            : 75.48%
LDA without Tuning                              : 74.17%
LDA with Tuning                                 : 74.17%
Random Forest without Tuning                    : 92.05%
Random Forest with Tuning                       : 86.76%
Support Vector Machine without Tuning           : 74.64%
```

Findings: Based on the results for accuracy, we can see that the “Random Forest without Tuning” model has the highest accuracy, followed by the “Random Forest with Tuning” model and the “XGBoost without Tuning” model. Both LDA models have the lowest accuracy. Majority of the models have an accuracy that is below 80%. The “Random Forest without Tuning” model is able to correctly predict the majority of the labels.

b) Comparison of Models Based on Precision

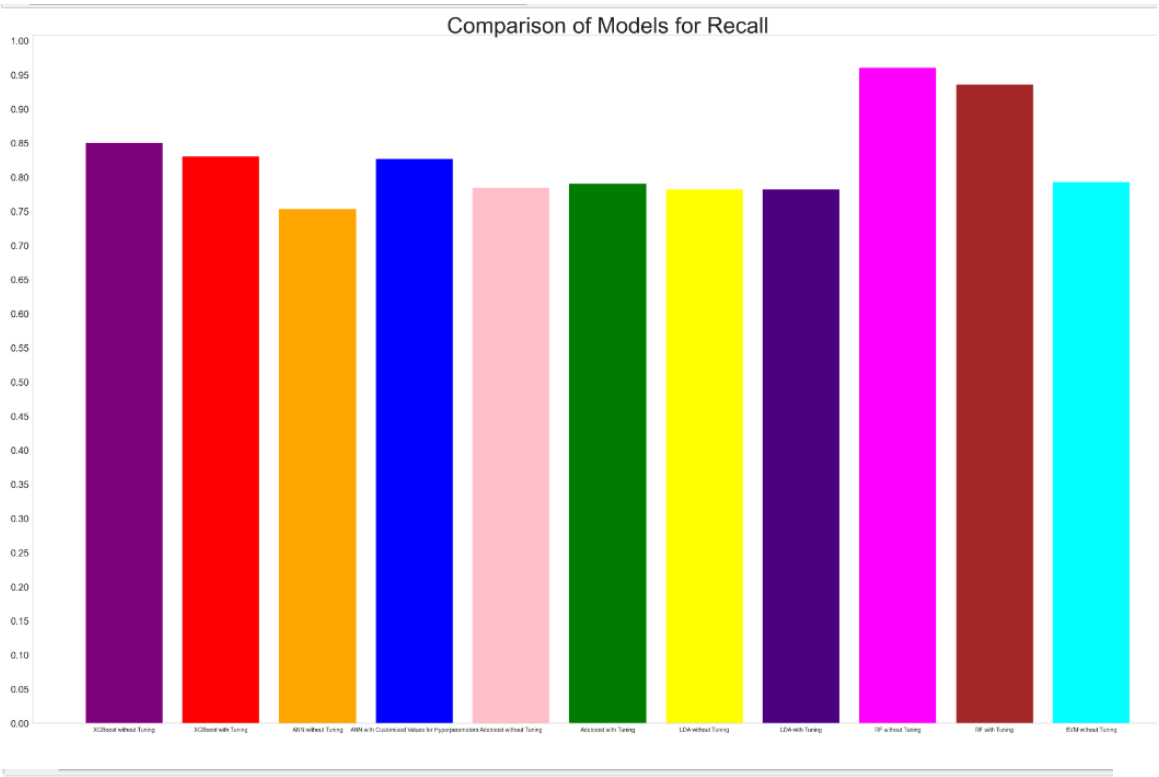



```
Results for Precision
------------------------------
XGBoost without Tuning                          : 77.04%
XGBoost with Tuning                             : 75.43%
ANN without Tuning                              : 75.15%
ANN with Customised Values for Hyperparameters: 73.86%
Adaboost without Tuning                         : 73.02%
Adaboost with Tuning                            : 73.65%
LDA without Tuning                              : 72.23%
LDA with Tuning                                 : 72.23%
Random Forest without Tuning                    : 88.89%
Random Forest with Tuning                       : 82.25%
Support Vector Machine without Tuning           : 72.42%
```

Findings: Based on the results for Precision, the “Random Forest without Tuning” model has the highest precision, followed by the “Random Forest with Tuning” model and the “XGBoost

without Tuning” model. Majority of the models have a precision that is below 80%. The “Random Forest without Tuning” model has few to no false positives and the classifier is very stern in the benchmark for categorising something as positive.

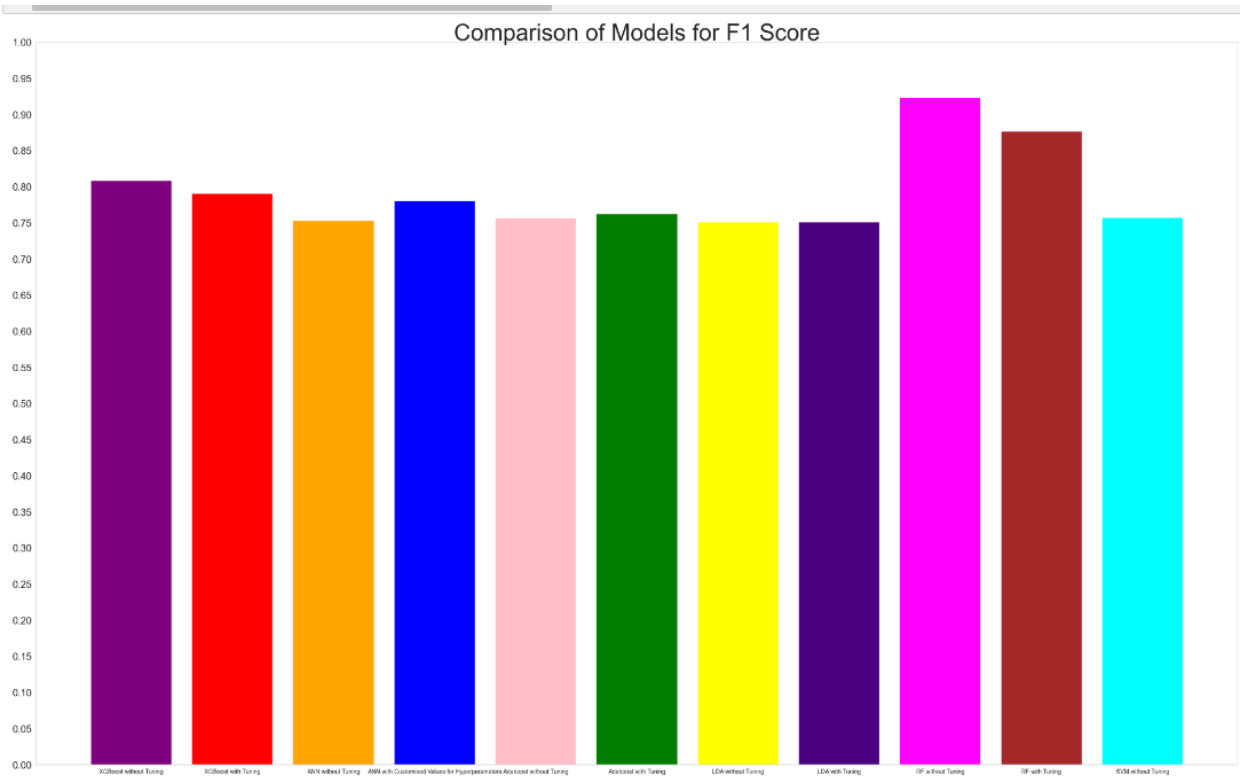
c) Comparison of Models Based on Recall




```
Results for Recall
------------------------------
XGBoost without Tuning                          : 85.09%
XGBoost with Tuning                             : 83.11%
ANN without Tuning                              : 75.42%
ANN with Customised Values for Hyperparameters: 82.72%
Adaboost without Tuning                         : 78.50%
Adaboost with Tuning                            : 79.10%
LDA without Tuning                              : 78.25%
LDA with Tuning                                 : 78.25%
Random Forest without Tuning                    : 96.06%
Random Forest with Tuning                       : 93.63%
Support Vector Machine without Tuning           : 79.33%
```

Findings: Based on the results for Recall, the “Random Forest without Tuning” model has the highest recall, followed by the “Random Forest with Tuning” model and the “XGBoost without Tuning” model. The Recall for most of the models are quite close to 80%. The “Random Forest without Tuning” model has very few false negatives and that the classifier is more liberal in the benchmark for categorising something as positive.

d) Comparison of Models Based on F1 Score




```
Results for F1 Score
------------------------------
XGBoost without Tuning                          : 80.86%
XGBoost with Tuning                             : 79.08%
ANN without Tuning                              : 75.28%
ANN with Customised Values for Hyperparameters: 78.04%
Adaboost without Tuning                         : 75.66%
Adaboost with Tuning                            : 76.28%
LDA without Tuning                              : 75.12%
LDA with Tuning                                 : 75.12%
Random Forest without Tuning                    : 92.34%
Random Forest with Tuning                       : 87.57%
Support Vector Machine without Tuning           : 75.71%
```

Findings: Based on the results for F1 Score, the “Random Forest without Tuning” model has the highest F1 Score, followed by the “Random Forest with Tuning” model and the “XGBoost without Tuning” model. Majority of the models have a F1 Score that is less than 80%. The high F1 score of the “Random Forest without Tuning” model indicates that it performs well overall and it is due to its high precision and recall.

e) Comparison of Models Based on AUC Score

Model	AUC Score
XGBoost without Tuning	79.94%
XGBoost with Tuning	78.10%
ANN without Tuning	75.32%
ANN with Customised Hyperparameters	76.82%
Adaboost without Tuning	74.84%
Adaboost with Tuning	83.00%
LDA without Tuning	74.18%
LDA with Tuning	82.00%
Random Forest without Tuning	98.00%
Random Forest with Tuning	86.82%
SVM without Tuning	82.00%

FIndings: We can see that the model with the highest AUC score is “Random Forest without Tuning”, followed by “Random Forest with Tuning” model and “Adaboost with Tuning” model. Since the “Random Forest without Tuning” model has the highest AUC score, it is the best among the models at predicting the true negatives and true positives.

Analysis of the Results:

Note: The metrics scores below are based on the Validation set. The scores for each algorithm are the highest scores for that algorithm irrespective of whether default or custom hyperparameter values are used.

Among the 6 main algorithms, Random Forest really stands out as it achieved very high Accuracy (92.05%), Recall (96.06%), F1-Score (92.34%) and Precision (88.89%).

The second highest performer in all the 4 metrics is XGBoost with Accuracy (79.93%), Recall (85.09%), F1-Score (80.86%) and Precision (77.04%).

The third highest performer is ANN with Accuracy (78.09%), Recall (82.72%), F1-Score (78.04%) and Precision (75.15%).

The gap between the 2nd and 3rd best performers (i.e. XGBoost and ANN) is not much and even less so between ANN and the other remaining 3 algorithms (AdaBoost, LDA, SVM).

The very notable result is the exceptionally high performance of Random Forest as its scores are significantly higher than the other algorithms. It is quite surprising that Random Forest can perform so much better than the rest. It could be that data issues play a part in the significantly different performances between Random Forest and the rest. For example, XGBoost is said to be sensitive to noisy data whereas Random Forest is not. However, in the data pre-processing step, we did handle both outliers and missing values. We also applied Min-Max scaling on the dataset. On top of that, we also balanced the Imbalanced target variable class. However, it is possible that there may be some “class noise” in our data e.g. existence of contradictory samples where similar samples have different classes (misclassification). Another data issue concerns the normal distribution (or lack) in the features. A few of the features in our dataset do not quite have a normal distribution, even after applying log transformation. In fact, one of the features has a bi-modal distribution. Algorithms such as ANN work better with normally distributed features because they result in faster convergence and better model performance. This goes for LDA as well.

Random Forest, on the other hand, is robust to noisy data and data which is not normally distributed. In the former case, Random Forest averages the predictions of multiple decision trees and therefore less sensitive to noise and outliers. Random Forest is non-parametric and has the ability to deal with skewed and multi-modal data.

Result of testing the Best Model on the Test Dataset:

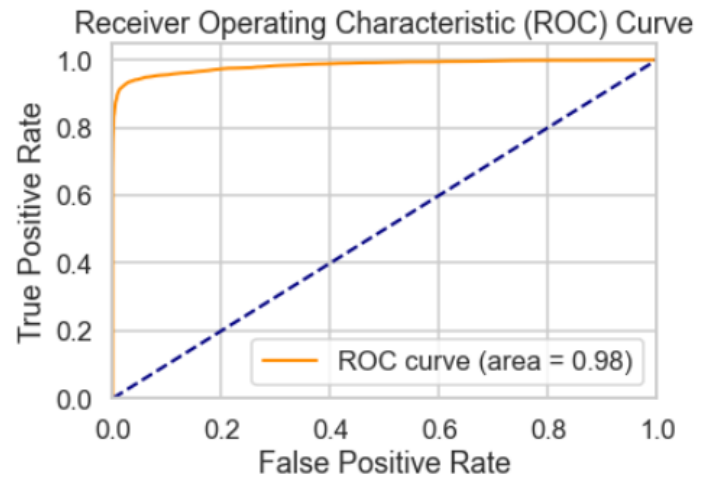
i) On Test Dataset:

```
print("Results for the Performance Metrics")
print("-----------------------------------------")
print("Precision for Random Forest: ", format(rfModel_pr * 100, ".2f"),"%")
print("Recall for Random Forest: ", format(rfModel_rc * 100, ".2f"),"%")
print("F1 Score for Random Forest: ", format(rfModel_f1 * 100, ".2f"),"%")
print("Accuracy for Random Forest: ", format(rfModel_ac * 100, ".2f"),"%")
print("AUC for Random Forest: ", format(rfModel_auc * 100, ".2f"),"%")
```

```
Results for the Performance Metrics
-----------------------------------------
Precision for Random Forest:  89.21 %
Recall for Random Forest:  96.07 %
F1 Score for Random Forest:  92.51 %
Accuracy for Random Forest:  92.14 %
AUC for Random Forest:  92.10 %
```

ii) On Validation Dataset

```
Accuracy: 92.04%
Precision: 88.92%
F1 Score: 92.32%
Recall: 95.99%
```



From the results, we can see that the model's precision, recall, accuracy and F1-Score on the test dataset are quite close to the results when the model is being used on the validation dataset. However, the AUC score (92.10%) when tested on the test set is much lower than that when tested on the validation set (98%).

5. Discussion and Conclusion

5.1 Achievements

5.1.1 Aspects of the Project we have achieved

The following were our objectives when we set out on this project:
a) To provide a reasonably accurate method of predicting loan default using data on the loan applicant's financial history and behaviour
b) To compare several Supervised Machine Learning (ML) algorithms in order to determine the best prediction model
c) To provide an efficient system to make decisions on loan approvals
d) To provide a user-friendly Graphical User Interface (GUI) for the loan default prediction system that can be easily and conveniently used by end users
e) To help the banks to make better decisions to reduce loss.

We have met the objective a) as we have managed to develop a Machine Learning Model which allows the bank or financial institution to make accurate (92%) prediction as to whether or not a loan (or loan application) will default based on the customer's financial history and current information. This ML prediction model is based on the Random Forest (RF) algorithm.

To develop the loan default predictor, we developed and compared the performances of 6 supervised ML algorithms (Random Forest, XGBoost, Artificial Neural Networks, Linear Discriminant Analysis, Support Vector Machines, AdaBoost), including trying out different hyperparameters in addition to the default parameters.

Our ML prediction model has been trained on real datasets and as a result it provides a fast prediction after loan info and applicant's financial history have been passed to it. Combined with the GUI (described next), it provides an efficient system to make decisions on loan approvals, therefore meeting objective c) above.

In addition, we created a user-friendly Graphical User Interface (GUI) using Tkinter (see Appendix), that allows the end-user to enter loan information and the loan applicant's financial history which are then fed to the Random Forest-based Loan Default Predictor Model (with Default Hyperparameters) which in turn returns a prediction of whether or not the loan will default. In short, we have met objective d) above.

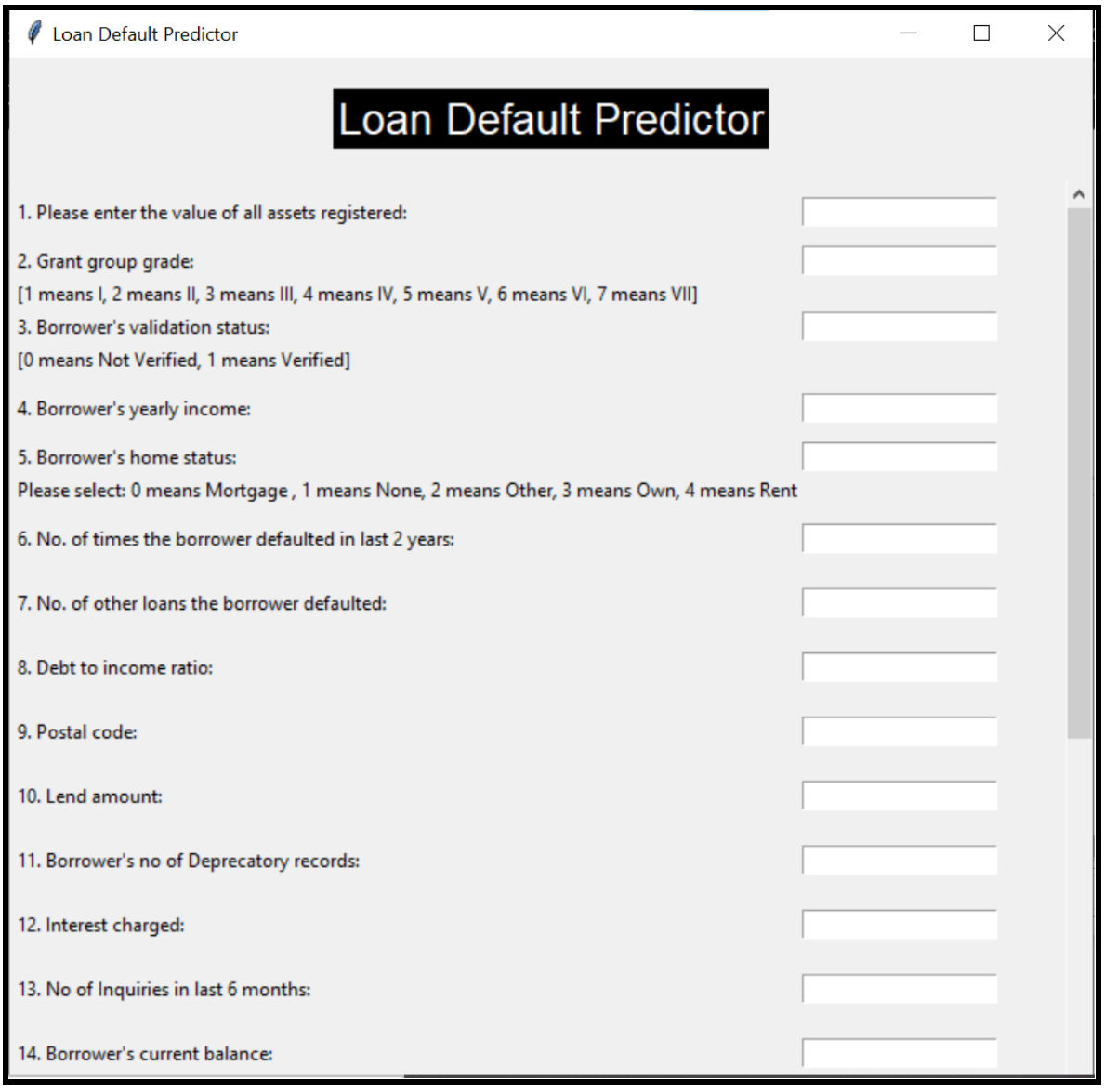
By providing a reliable and performant loan default predictor and building an efficient and user-friendly user interface for it, the bank or financial institution can make loan application decisions faster and customers can get faster replies on their loan applications. This improves not just efficiency but also improves customer service by having a faster turnaround time.

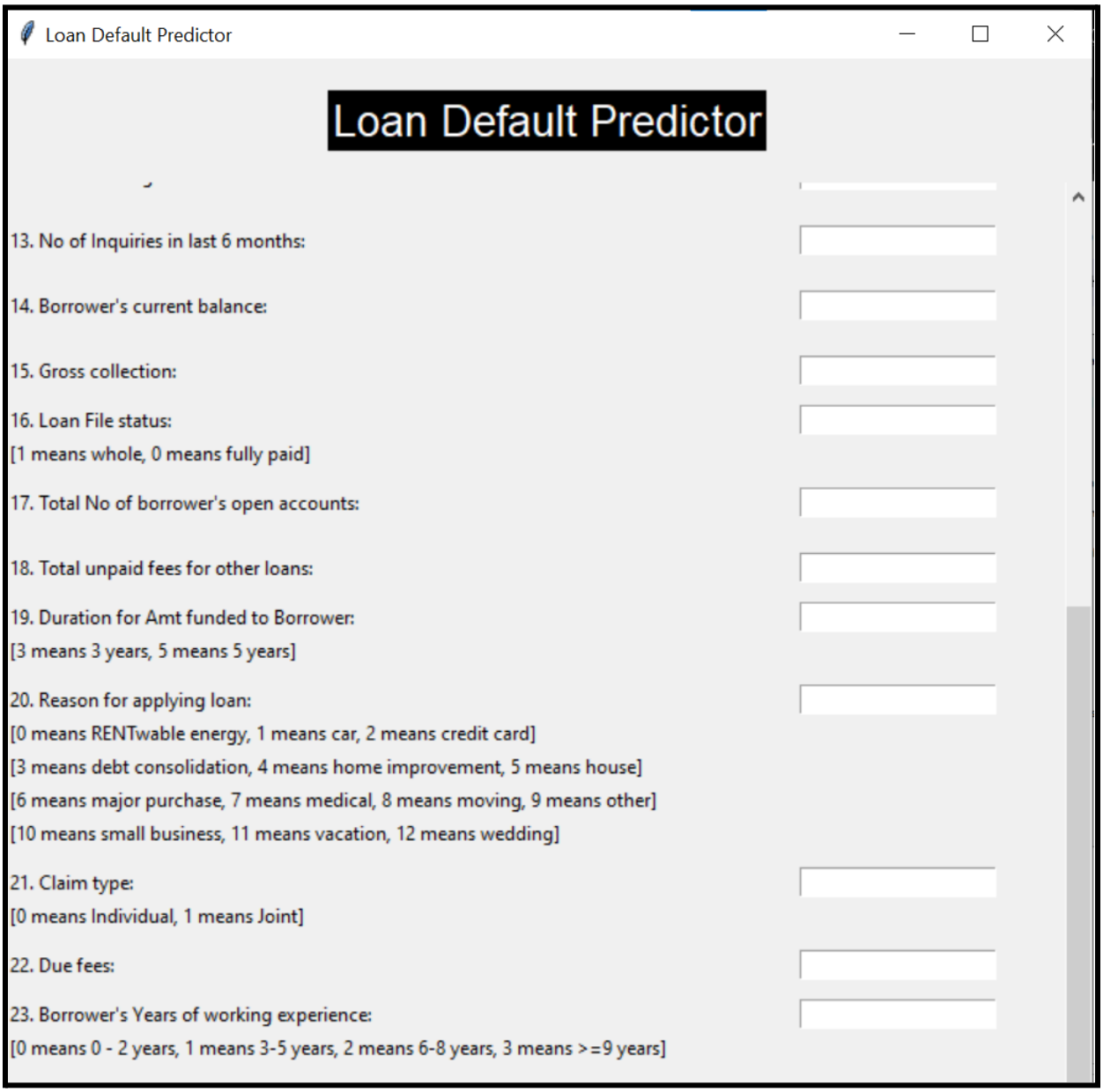
Ultimately, the bank or financial institution can reduce the number loan defaults and reduce loss, meeting objective e) above.

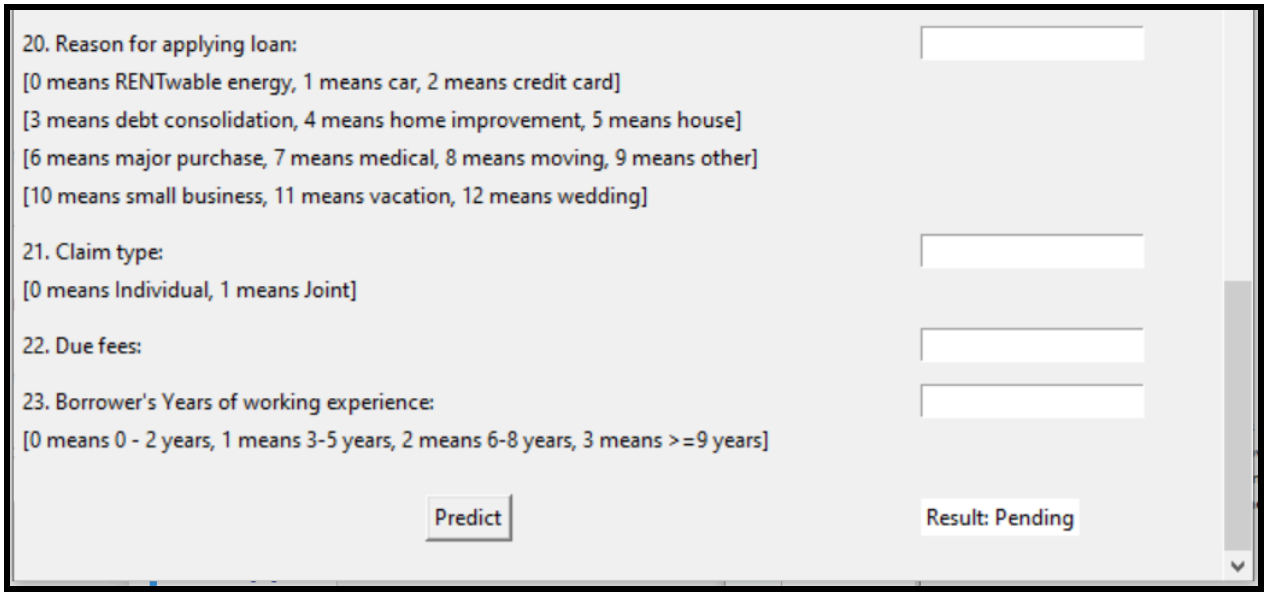
5.1.2 Significance of the Results

The results, besides telling us that Random Forest is the best performing algorithm, also indicates that ensemble algorithms perform better than other algorithms because the second best performer, XGBoost, is also an ensemble algorithm.

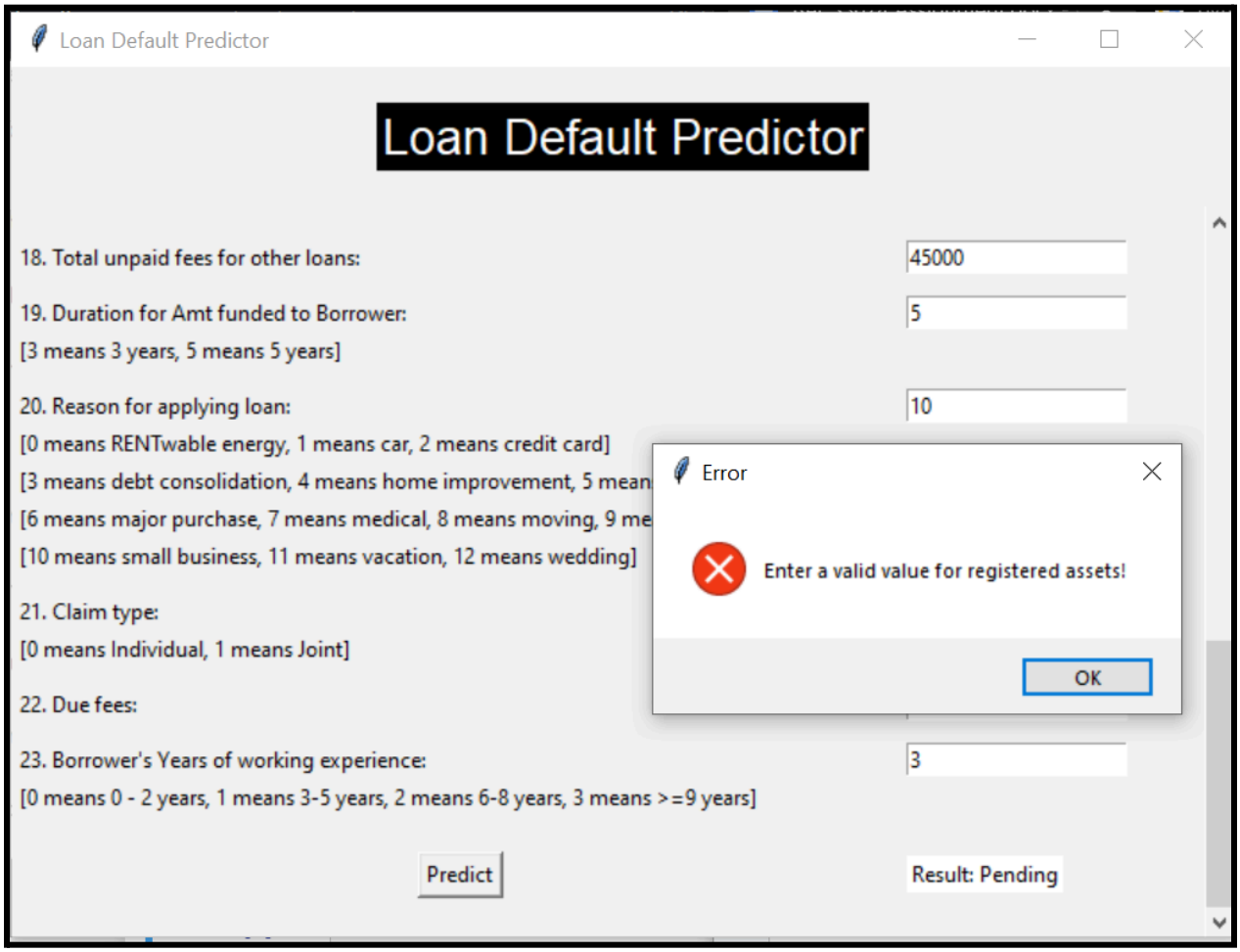
APPENDIX



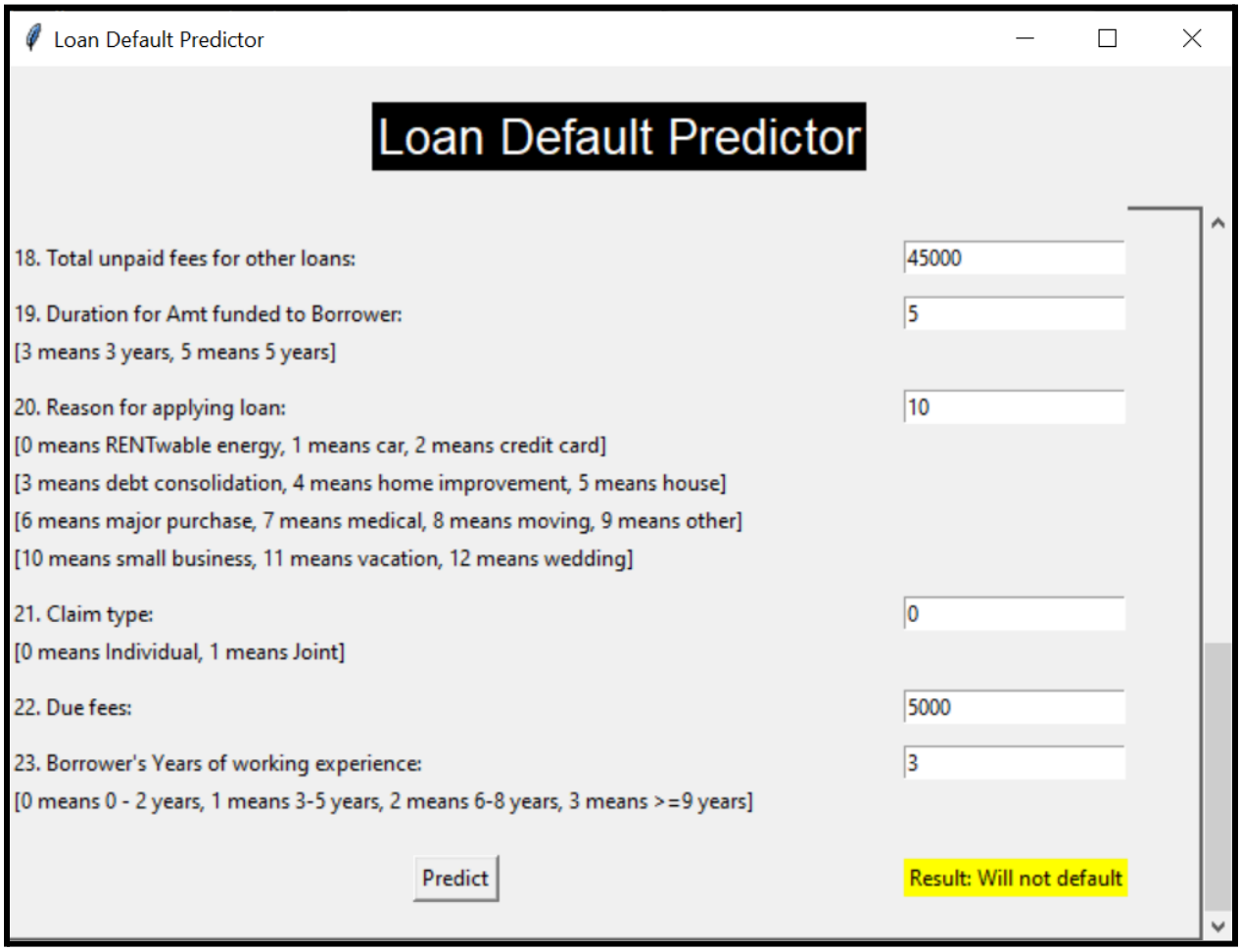

Loan Default Predictor
Loan Default Predictor
13. No of Inquiries in last 6 months:
14. Borrower's current balance:
15. Gross collection:
16. Loan File status:
[1 means whole, 0 means fully paid]
17. Total No of borrower's open accounts:
18. Total unpaid fees for other loans:
19. Duration for Amt funded to Borrower:
[3 means 3 years, 5 means 5 years]
20. Reason for applying loan:
[0 means RENTwable energy, 1 means car, 2 means credit card]
[3 means debt consolidation, 4 means home improvement, 5 means house]
[6 means major purchase, 7 means medical, 8 means moving, 9 means other]
[10 means small business, 11 means vacation, 12 means wedding]
21. Claim type:
[0 means Individual, 1 means Joint]
22. Due fees:
23. Borrower's Years of working experience:
[0 means 0 - 2 years, 1 means 3-5 years, 2 means 6-8 years, 3 means >=9 years]



If an invalid value is entered into any one of the input fields or the user forgets to enter input in at least one of the fields, the system will throw an exception and display an error message.



If the user enters valid values for all the input fields, the system will predict whether the given loan will be defaulted or not and display the prediction at the bottom right of the GUI.



5.2 Limitations and Future Works

In this project, we did not manage to use many different hyper parameter configurations when running our algorithms. The main reason for not doing this is processing time. Algorithms like ANN take considerably long to run. We did try using GridSearchCV to try different hyper parameter combinations when running our algorithms. However, they simply took so long to complete that it was impractical to proceed in this manner. That is why, besides running the algorithm with default hyper parameters, we only chose one set of custom hyper parameters to run the same algorithm.

Another reason for the long running times may be attributed to the fact that our dataset is not small. Moreover, our PC with the highest hardware specs is an Intel i7 with 16Gb of RAM. In the future, given sufficient time and access to more powerful hardware, we will be able to perform proper hyper parameter tuning for those algorithms which did not perform so well. The exception is the Random Forest algorithm which actually performed best without any tuning, i.e. using the default hyper parameters.

Another limitation is that it is possible that the dataset could only be applicable to the country, the United States of America. Based on our analysis of the State variable of the dataset, its possible values seemed to be of USA origin so there is uncertainty about whether the results obtained from the dataset could be generalised/transferable to other countries.

The dataset that we have chosen may not contain all of the variables/features needed to determine whether a bank loan will be defaulted or not. Perhaps, we could collect more data which are related to the loan and customer.

To improve the project in the future, we could try to obtain access to higher end laptops/desktop computers so that we can take advantage of GridSearchCV or we could have access to better cloud service that can provide with the performance needed to run the model at a faster speed with less amount of time. We could also try to use other supervised machine learning algorithms apart from RandomForest, XGBoost, SVM, Artificial Neural Network, Linear DIscriminant Analysis and Adaboost in order to explore potentially better models.

6. Reference & Source

Chen, J. (n.d.). Default: What it means, what happens when you default, examples. Investopedia. https://www.investopedia.com/terms/d/default2.asp

Bank Finance. Advantages and disadvantages of bank loans. (n.d.). https://www.nibusinessinfo.co.uk/content/advantages-and-disadvantages-bank-loans

The effects of loan default on commercial banks profitability: Case ... (n.d.). https://www.researchgate.net/publication/347529410_The_Effects_of_Loan_Default_on_Commercial_Banks_Profitability_Case_Study_BICEC_Limbe

Analyzing overfitting under class imbalance in neural networks for image ... (n.d.-a). https://arxiv.org/pdf/2102.10365.pdf

Nik. (2023, February 19). Seaborn catplot - categorical data visualizations in python • datagy. datagy.https://datagy.io/seaborn-catplot/#:~:text=To%20add%20a%20title%20to,spacing%20of%20our%20figure%20object.

Nik. (2023a, February 19). Seaborn catplot - categorical data visualizations in python • datagy. datagy. https://datagy.io/seaborn-catplot/

Visualizing categorical data#. Visualizing categorical data - seaborn 0.12.2 documentation. (n.d.). https://seaborn.pydata.org/tutorial/categorical.html

GeeksforGeeks. (2022, June 24). Different ways to iterate over rows in pandas dataframe. GeeksforGeeks.https://www.geeksforgeeks.org/different-ways-to-iterate-over-rows-in-pandas-dataframe/

Seaborn.histplot#. seaborn.histplot - seaborn 0.12.2 documentation. (n.d.). https://seaborn.pydata.org/generated/seaborn.histplot.html

GeeksforGeeks. (2023, April 18). Label encoding in Python. GeeksforGeeks. https://www.geeksforgeeks.org/ml-label-encoding-of-datasets-in-python/

How to read CSV to dataframe in google colab. Saturn Cloud Blog. (2023, August 25). https://saturncloud.io/blog/how-to-read-csv-to-dataframe-in-google-colab/

Olumide, S. (2023, June 8). Dataframe drop column in pandas – how to remove columns from Dataframes. freeCodeCamp.org. https://www.freecodecamp.org/news/dataframe-drop-column-in-pandas-how-to-remove-columns-from-dataframes/

Undefined. (2023, June 12). What is Binning in data mining?. Scaler Topics. https://www.scaler.com/topics/binning-in-data-mining/

Javed, H. (1969). Retrieved from https://linuxhint.com/pandas-convert-categorical-to-int/

Singh, G. (2023). Introduction to artificial neural networks. Retrieved from https://www.analyticsvidhya.com/blog/2021/09/introduction-to-artificial-neural-networks/#:~:text=Artificial%20Neural%20Networks%20(ANN)%20are,human%20brain%20Biological%20Neural%20Networks.

SheetalSharma. (2022). Artificial Neural Network (ANN) in Machine Learning. Retrieved from https://www.datasciencecentral.com/artificial-neural-network-ann-in-machine-learning/

Input layer definition - glossary. (2023). Retrieved from https://nordvpn.com/cybersecurity/glossary/input-layer/#:~:text=Input%20layer%20refers%20to%20the,function%20of%20the%20human%20brain.

Ognjanovski, G. (2020, June 7). Everything you need to know about neural networks and backpropagation - machine learning made easy... Medium. https://towardsdatascience.com/everything-you-need-to-know-about-neural-networks-and-backpropagation-machine-learning-made-easy-e5285bc2be3a

GeeksforGeeks. (2023, February 17). Activation functions in neural networks. GeeksforGeeks. https://www.geeksforgeeks.org/activation-functions-neural-networks

Morde, V. (2019, April 8). XGBoost algorithm: Long may she reign!. Medium. https://towardsdatascience.com/https-medium-com-vishalmorde-xgboost-algorithm-long-she-may-rein-edd9f99be63d

Simplilearn. (2023, June 2). What is xgboost? an introduction to XGBoost algorithm in Machine Learning: Simplilearn. Simplilearn.com. https://www.simplilearn.com/what-is-xgboost-algorithm-in-machine-learning-article#:~:text=cost%20to%20accuracy!-,What%20is%20XGBoost%20Algorithm%3F,optimize%20their%20machine%2Dlearning%20models.

GeeksforGeeks. (2023b, March 31). Gradient boosting in ML. GeeksforGeeks. https://www.geeksforgeeks.org/ml-gradient-boosting/

GeeksforGeeks. (2023a, February 6). XGBoost. GeeksforGeeks. https://www.geeksforgeeks.org/xgboost/

Rawat, S. (n.d.). Advantages and disadvantages of Neural Networks. Analytics Steps. https://www.analyticssteps.com/blogs/advantages-and-disadvantages-neural-networks

Mahanta, J. (2017, July 12). Introduction to neural networks, advantages and applications. Medium. https://towardsdatascience.com/introduction-to-neural-networks-advantages-and-applications-96851bd1a207

Ivankov, A. (2023, May 9). Advantages and disadvantages of Artificial Neural Networks. Profolus. https://www.profolus.com/topics/advantages-disadvantages-artificial-neural-networks/

GeeksforGeeks. (2020, December 11). Normalize a column in Pandas. GeeksforGeeks. https://www.geeksforgeeks.org/normalize-a-column-in-pandas/

(NNK), N. (2023, January 19). Pandas drop rows with condition. Spark By {Examples}. https://sparkbyexamples.com/pandas/pandas-drop-rows-with-condition/?expand_article=1

Brown, S. (2021, April 21). Machine Learning, explained. MIT Sloan. https://mitsloan.mit.edu/ideas-made-to-matter/machine-learning-explained

Normalization in machine learning - javatpoint. www.javatpoint.com. (n.d.). https://www.javatpoint.com/normalization-in-machine-learning

Goyal, C. (2023, August 14). Outlier Detection & Removal: How to Detect & Remove Outliers (updated 2023). Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/05/feature-engineering-how-to-detect-and-remove-outliers-with-python-code/

GeeksforGeeks. (2019, July 4). Python: Visualize missing values (nan) values using Missingno Library. GeeksforGeeks. https://www.geeksforgeeks.org/python-visualize-missing-values-nan-values-using-missingno-library/

Kumar, A. (2022, August 5). Sklearn simpleimputer example - impute missing data. Analytics Yogi. https://vitalflux.com/imputing-missing-data-sklearn-simpleimputer/

When should I use the median?. Scribbr. (2021, December 29). https://www.scribbr.com/frequently-asked-questions/when-should-i-use-the-median/#:~:text=Because%20the%20median%20only%20uses,can%20vary%20in%20skewed%20distributions.

Brownlee, J. (2020, August 20). How to calculate feature importance with python. MachineLearningMastery.com. https://machinelearningmastery.com/calculate-feature-importance-with-python/

Simplilearn. (2023, January 18). Normalization vs standardization - what's the difference?: Simplilearn. Simplilearn.com. https://www.simplilearn.com/normalization-vs-standardization-article#:~:text=When%20your%20data%20have%20different,of%20your%20data%20is%20Gaussian.

GeeksforGeeks. (2021, June 10). Selecting rows in pandas DataFrame based on conditions. GeeksforGeeks. https://www.geeksforgeeks.org/selecting-rows-in-pandas-dataframe-based-on-conditions/

Vijetha. (2023, February 8). How to change pandas plot size?. Spark By {Examples}. https://sparkbyexamples.com/pandas/change-pandas-plot-size/#:~:text=The%20figsize%20parameter%20of%20the,specify%20height%20of%20the%20plot.

How to convert numpy array to pandas DataFrame: A comprehensive guide. Saturn Cloud Blog. (2023, August 25). https://saturncloud.io/blog/how-to-convert-numpy-array-to-pandas-dataframe-a-comprehensive-guide/

3 essential ways to calculate feature importance in Python. Better Data Science. (n.d.). https://betterdatascience.com/feature-importance-python/

Rithp. (2023, January 16). Optimizing xgboost: A guide to hyperparameter tuning. Medium. https://medium.com/@rithpansanga/optimizing-xgboost-a-guide-to-hyperparameter-tuning-77b6e48e289d

tilii7. (2017, October 13). Hyperparameter grid search with XGBoost. Kaggle. https://www.kaggle.com/code/tilii7/hyperparameter-grid-search-with-xgboost

XGBoost parameters. XGBoost Parameters - xgboost 2.0.0 documentation. (n.d.). https://xgboost.readthedocs.io/en/stable/parameter.html

How to add space between two widgets placed in a grid in Tkinter. How to add space between two widgets placed in a grid in tkinter. (n.d.). https://www.tutorialspoint.com/how-to-add-space-between-two-widgets-placed-in-a-grid-in-tkinter

GeeksforGeeks. (2023, March 13). ML: Linear discriminant analysis. GeeksforGeeks. https://www.geeksforgeeks.org/ml-linear-discriminant-analysis/

Pandas: Replacing nans using median/mean of the column. w3resource. (n.d.). https://www.w3resource.com/python-exercises/pandas/missing-values/python-pandas-missing-values-exercise-14.php

Saini, A. (2023, August 4). Master the adaboost algorithm: Guide to implementing & understanding adaboost. Analytics Vidhya. https://www.analyticsvidhya.com/blog/2021/09/adaboost-algorithm-a-complete-guide-for-beginners/

How to explain the ROC AUC score and Roc Curve?. How to explain the ROC AUC score and ROC curve? (n.d.). https://www.evidentlyai.com/classification-metrics/explain-roc-curve#:~:text=ROC%20AUC%20score%20shows%20how,have%20an%20AUC%20of%200.5.

Sklearn.ensemble.adaboostclassifier. scikit. (n.d.). https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.AdaBoostClassifier.html

Dino, L. (2022, July 6). *XGBoost (classification) in Python*. Medium. https://medium.com/@24littledino/xgboost-classification-in-python-f29cc2c50a9b#:~:text=First%2520of%2520all%252C%2520XGBoost%2520can,One%252Dvs%252Dall

Richmond, S. (n.d.). *Algorithms exposed: Random Forest*. BCCVL. https://bccvl.org.au/algorithms-exposed-random-forest/#:~:text=ASSUMPTIONS,are%20ordinal%20or%20non%2Dordinal.

Gandhi, R. (2022, November 14). Support Vector Machine — Introduction to machine learning algorithms. *Medium*. https://towardsdatascience.com/support-vector-machine-introduction-to-machine-learning-algorithms-934a444fca47

What is Random Forest? | *IBM*. (n.d.).
https://www.ibm.com/topics/random-forest#:~:text=Random%20forest%20is%20a%20commonly,both%20classification%20and%20regression%20problems.

What is Random Forest? | *IBM*. (n.d.-b).
https://www.ibm.com/topics/random-forest#:~:text=Random%20forest%20is%20a%20commonly,both%20classification%20and%20regression%20problems.