



INSTITUTE OF SCIENCE, TECHNOLOGY & ADVANCED STUDIES (VISTAS)

(Deemed to be University Estd. u/s 3 of the UGC Act, 1956)

PALLAVARAM, THALAMBUR, PERIYAPALAYAM-CHENNAI

ACCREDITED BY **NAAC** WITH 'A++' GRADE

# **Loan Approval Forecasting using Machine Learning**

## **A MINI PROJECT REPORT**

**Submitted by**  
**Thanigaivel G**  
**RegNo : 24880107**

**in partial fulfillment of the award of the degree**

*of*

**MASTER OF ENGINEERING IN COMPUTER SCIENCE  
AND ENGINEERING**

**DEPARTMENT OF COMPUTER SCIENCE AND ENGINEERING SCHOOL OF  
ENGINEERING**

**VELS INSTITUTE OF SCIENCE, TECHNOLOGY & ADVANCED STUDIES (VISTAS)  
PALLAVARAM, CHENNAI 600117**

**APRIL 2025**

## **BONAFIDE CERTIFICATE**

Certified that this project “**Loan Approval Forecasting using Machine Learning**” is the bonafide work of **Thanigaivel G (24880107)** who carried out the project under my supervision. Certified further, that to the best of my knowledge the work reported here does not form part of any other project report or dissertation on the basis of which a degree or award was conferred on an earlier occasion on this or any other candidate.

### **SIGNATURE**

Dr.K.Kalaivani,  
Professor& HOD  
Dept. of CSE  
VISTAS,Chennai

### **SIGNATURE**

Dr. A. Manikandan,  
Associate Professor,  
Dept. of CSE  
VISTAS,Chennai

### **SIGNATURE**

Dr. R. Kalpana ,  
Assistant Professor,  
Dept. of CSE  
VISTAS,Chennai

Submitted for the project work and viva - voice examination held on ..... at Vels Institute of Science, Technology and Advanced Studies (VISTAS), Chennai – 600 117.

**INTERNAL EXAMINER**

**EXTERNAL EXAMINER**

## ACKNOWLEDGEMENT

It is certainly a great delight and proud privilege to acknowledge the help and support I received from the positive minds around me in making this venture a successful one. The infrastructural support with all kinds of lab facilities have been a motivating factor in this completion of project work, all because of **FOUNDER & CHAIRMAN, Shri. Dr. ISHARI K. GANESH.**, with great pleasure that I take this opportunity to thank him. I am grateful to our **HEAD OF THE DEPARTMENT, Dr.K.KALAIVANI, Ph.D** for her continuous support for the project, from initial advice and contacts in the early stages of conceptual inception, and through on-going advice and encouragement to this day.

I extend our warmest thanks to my **INTERNAL GUIDE, Dr.A.MANIKANDAN, Dr. R. KALPANA**, who has patiently guided and provided us with invaluable advice and help. I am thankful to my Project Coordinator **Dr.A.PACKIALATHA M.E., Ph.D.**, I would like to express my greatest gratitude to our staff members, librarian and non-teaching staff members of Computer Centre, who have helped and supported me throughout. Last but not the Least, I wish to thank my **PARENTS** and my friends for their individual support and interest who has inspired me and encouraged me to go in my own way, without them I would be unable to complete this project.

THANIGAIVEL G - 24880107

## TABLE OF CONTENTS

CHAPTER NO.	TITLE	PAGE NO
	ABSTRACT	I
	LIST OF FIGURES	II
	LIST OF ABBREVIATION	III
1	<b>INTRODUCTION</b>	
	1.1 GENERAL	1
	1.2 PROBLEM DESCRIPTION	2
	1.3 OBJECTIVE	3
	1.4 SCOPE	3
2	<b>LITERATURE SURVEY</b>	
	2.1 GENERAL	5
	2.2 EXISTING SYSTEM	5
	2.2.1 LITERATURE SURVEY EXTRACTION	7
	2.2.2 LITERATURE SURVEY	9
	FEATURE EXTRACTION	
	2.3 PROPOSEDSYSTEM	9
3	<b>SYSTEM DESIGN</b>	
	3.1 GENERAL	11
	3.2 SYSTEM ARCHITECTURE	12
	3.3 DATAFLOW	12
	3.4 MODULE WORKFLOW	13
	3.5 SYSTEM REQUIRMENTS	14
	3.5.1 MINIMUM HARDWARE EQUIREMENTS	14
	3.5.2 MINIMUM SOFTWARE REQUIREMENTS	14
	3.6 SUMMARY	15
4	<b>SYSTEM IMPLEMENTATION</b>	
	4.1 GENERAL	16
	4.2 OVERVIEW OF THE PLATFORM	16
	4.2.1 PYTHON	16
	4.2.2 NUMPY	17

	4.2.3 OPENCV	18
	4.2.4 MATPLOTLIB	18
	4.2.5 STANDARDSCALER	19
	4.2.6 PANDAS	19
	4.2.7 SKLEARN	20
	4.2.8 SCALER.FIT_TRANSFORM	20
	4.2.9 PLOTY	21
	4.2.10 PLT- PYPLOT IN MATPLOTLIB	21
	4.3 MODULE IMPLEMENTATION	31
<b>5</b>	<b>SYSTEMTESTING</b>	
	5.1 GENERAL	42
	5.2 TESTCASES	42
	5.3 PERFORMANCE MEASURES	42
	5.4 SUMMARY	43
<b>6</b>	<b>CONCLUSIONS AND</b>	<b>44</b>
	<b>FUTUREWORK</b>	<b>44</b>
	<b>REFERENCES</b>	<b>45</b>
	<b>APPENDIX (SAMPLESOURCE CODE)</b>	<b>46</b>

## ABSTRACT

In recent years, the financial industry has experienced a significant transformation with the introduction of automated decision-making systems powered by Machine Learning (ML). One of the critical applications of ML in the financial sector is loan approval forecasting, where traditional manual decision-making processes are augmented or replaced by data-driven models. The goal of this project is to predict loan approval decisions based on a range of applicant features, including personal information such as income, credit score, employment history, and loan characteristics, to enhance the efficiency and accuracy of loan approval processes. The project utilizes a dataset containing historical loan application data, where each entry represents an applicant's personal and financial details along with the loan approval outcome (approved or rejected). The primary objective is to develop a robust and accurate machine learning model capable of forecasting loan approval decisions in real-time, minimizing human error and subjectivity in the decision-making process. This involves several stages, including data preprocessing, feature selection, and model training. We employed various supervised machine learning algorithms, including Logistic Regression, Decision Trees, Random Forests, and Support Vector Machines (SVM), to build predictive models. Data preprocessing tasks involved handling missing values, normalizing numerical variables, and encoding categorical features to prepare the dataset for model training. The dataset was split into training and testing sets to assess the model's performance. Model performance was evaluated using standard classification metrics such as accuracy, precision, recall, F1-score, and ROC-AUC. Cross-validation techniques were employed to avoid overfitting and to ensure that the models generalize well to unseen data. After training and tuning the models, we compared their performances to determine the most effective approach for predicting loan approval. The findings indicate that machine learning models, particularly **Random Forest and Support Vector Machines**, achieved high accuracy and predictive power in forecasting loan approval outcomes. These results demonstrate the potential of ML models to automate and optimize decision-making in the financial sector, offering faster, more consistent, and unbiased loan approval decisions. Ultimately, the research concludes that Machine Learning offers a promising solution to enhance loan approval systems and provides a foundation for future exploration in areas such as credit scoring, fraud detection, and personalized loan offerings.

## LIST OF FIGURES

<b>FIGURE NO</b>	<b>TITLE</b>	<b>PAGE NO</b>
Figure 3.1	System Architecture	12
Figure 3.3	Data Flow Diagram	13
Figure 3.4	Module work flow	13
Figure4.1	Python Interpretation	17
Figure4.2	Python NumPy Multi-Dimensional Array	17
Figure4.3	3BSD3-Clause License For NumPy Package	18
Figure 4.3.1	Module Implementation	27
Figure 4.3.2	Balancing the Dataset	29
Figure 4.3.3	Data Preprocessing	30
Figure 4.3.4	Simple Imputer	31
Figure 4.3.5	Label Encoder	32
Figure 4.3.6	Univariate Analysis	33
Figure 4.3.7	Feature Selection	34
Figure 5.1	Computation Speed and Cost Graph	43
Figure 6.1	Output for Loan Eligible	50
Figure 6.2	Output for Loan Not Eligible	50

## **LIST OF ABBREVIATION**

<b>CPU</b>	Central Processing Unit
<b>GPU</b>	Graphics Processing Unit
<b>GB</b>	Giga Byte
<b>RAM</b>	Random Access Memory
<b>HDD</b>	Hard Disk Drive
<b>SSD</b>	Solid State Drive
<b>IDE</b>	Integrated Development Environment
<b>OOP</b>	Object Oriented Programming
<b>SVM</b>	Support Vector Machine
<b>SMOTE</b>	Synthetic Minority Over-sampling Technique



# CHAPTER 1

## INTRODUCTION

### 1.1 GENERAL

The financial industry is undergoing a profound transformation driven by technological advancements and an increasing reliance on data analytics. One of the most vital and complex processes in this industry is the loan approval process, which directly impacts both the financial institution and its applicants. Traditionally, loan approval decisions have been made based on a combination of historical data, human judgment, and established credit scoring systems. These systems typically assess an applicant's ability to repay the loan by analyzing factors such as credit history, income, debt-to-income ratio, and employment status. While these approaches have served the industry for decades, they often involve time-consuming procedures and are prone to inconsistencies and biases inherent in human decision-making.

With the rise of big data, machine learning (ML), and artificial intelligence (AI), the financial sector has an opportunity to rethink and streamline the loan approval process. As the volume of loan applications continues to grow and applicants' financial profiles become increasingly diverse and complex, traditional systems struggle to keep up. This shift in data complexity, along with heightened customer expectations for quicker responses, has led to a pressing demand for more efficient, scalable, and objective systems that can assess loan eligibility.

The aim of this project, "**Loan Approval Forecasting using Machine Learning**," is to leverage the capabilities of machine learning algorithms to automate and enhance the loan approval process. The project's central objective is to develop a predictive model capable of forecasting whether a loan application will be approved or rejected based on a range of applicant-specific features. These features may include factors such as income, credit score, loan amount, employment history, and other financial indicators that are essential to understanding an applicant's creditworthiness.

By utilizing historical loan application data, the machine learning model will identify and learn from patterns and correlations within the data that are predictive of loan approval outcomes. This approach allows the model to not only analyze more complex relationships between features but also to account for interactions that may be difficult for traditional decision-making systems to detect. As the model trains on past application data, it will become better equipped to predict the likelihood of loan approval with increasing accuracy, reducing reliance on subjective judgment.

Automating the loan approval process using machine learning offers multiple benefits. First, it significantly reduces the time required to process loan applications. Traditional methods often involve a lengthy review process, especially when large volumes of applications are submitted. By automating key parts of the decision-making process, financial institutions can drastically cut down on the turnaround time, leading to faster decisions and an overall more efficient workflow.

More importantly, automation helps remove human bias from the decision-making process. Machine learning, however, makes decisions based purely on historical data, ensuring that loan approval is determined by objective, quantifiable metrics.

For this project, a variety of machine learning algorithms will be explored, including Logistic Regression, Decision Trees, Random Forests, and Support Vector Machines (SVM). These models will be trained on a dataset of historical loan applications, where the outcomes (approved or rejected) are already known. Through this process, the algorithms will learn the relationships between applicant features and loan approval decisions. The effectiveness of each model will be evaluated using performance metrics such as accuracy, precision, recall, and F1-score, providing a comprehensive understanding of each model's strengths and weaknesses.

## **1.2 PROBLEM DESCRIPTION**

The loan approval process is a crucial aspect of the financial industry, impacting both lending institutions and applicants. Traditionally, loan approvals rely on credit scores, income, employment status, and human judgment. However, this method has notable limitations, including time consumption, inefficiency, and the potential for human bias. The manual review process can lead to delays, particularly when financial institutions handle a high volume of applications. This inefficiency not only increases operational costs but also frustrates applicants who face extended waiting times for approval decisions.

Additionally, human judgment in loan approval can introduce biases, resulting in inconsistent or unfair outcomes. Loan officers may unknowingly prioritize certain factors over others, affecting the fairness of the decision-making process. Moreover, traditional systems often rely on limited data, such as credit scores, which may not fully capture an applicant's financial health and repayment capacity.

As the number of loan applications increases, traditional approaches struggle to scale, making it difficult for financial institutions to keep up with demand while ensuring accuracy. This can result in poor loan decisions, leading to financial loss and higher default rates. The need for a more efficient, objective, and scalable solution is clear.

Machine learning offers a promising solution by automating the loan approval process, allowing for faster, data-driven decisions. Machine learning models can evaluate a broader set of features and remove human biases from the decision-making process, providing more accurate predictions. This project seeks to develop such a machine learning-based system to improve the efficiency, fairness, and scalability of the loan approval process.

### 1.3 OBJECTIVE

The primary objective of this project is to develop a machine learning-based model that automates the loan approval process, enhancing both efficiency and accuracy. The key objectives are:

To design and implement a predictive model that forecasts loan approval outcomes based on applicant-specific features such as credit score, income, employment status, loan amount, and other relevant financial indicators.

To automate the decision-making process, reducing the time and effort required to process loan applications, leading to quicker decisions and improved operational efficiency for lending institutions.

To reduce human bias and ensure fairness in loan approval by leveraging data-driven models that assess applications purely based on objective financial data, eliminating subjective human judgment.

To improve prediction accuracy by applying machine learning algorithms (such as Logistic Regression, Decision Trees, and Random Forests) and evaluating them using performance metrics like accuracy, precision, recall, and F1-score.

To scale the loan approval process to handle large volumes of loan applications effectively, ensuring consistent decision-making even as the number of applicants grows.

To provide insights into the most influential factors affecting loan approval decisions, aiding financial institutions in better understanding the factors that drive their approval processes.

### 1.4 SCOPE

The scope of this project is focused on developing a machine learning-based system to automate and improve the loan approval process for financial institutions. Specifically, the project covers the following areas:

**Data Collection and Preprocessing:** The project will involve gathering a dataset of historical loan applications, including features such as credit score, income, employment status, loan amount, and

past financial behavior. The data will be cleaned, preprocessed, and transformed to make it suitable for machine learning models. **Feature Engineering:** Relevant features will be identified and engineered to enhance the predictive power of the machine learning models. This includes handling missing values, encoding categorical variables, and scaling numerical features where necessary.

**Model Development and Training:** Several machine learning algorithms, such as Logistic Regression, Decision Trees, Random Forests, and Support Vector Machines (SVM), will be used to train the model. The project will involve selecting the best-performing model based on evaluation metrics like accuracy, precision, recall, and F1-score.

**Model Evaluation and Optimization:** The trained models will be evaluated using different performance metrics to determine their effectiveness in predicting loan approval outcomes. Hyperparameter tuning will be performed to optimize model performance and ensure high accuracy in decision-making.

**Implementation of a Predictive System:** Once the optimal model is identified, it will be integrated into a system capable of making loan approval predictions on new, unseen data. The system will automate the decision-making process, reducing human involvement and improving processing time.

**Fairness and Bias Analysis:** The project will assess the fairness of the loan approval model by analyzing any potential biases in the predictions, ensuring that the model does not favor any specific group or demographic over others.

**Scalability and Efficiency:** The system will be designed to scale with increasing volumes of loan applications, ensuring that it can handle a large number of applications efficiently without compromising prediction accuracy.

**Insight Generation:** The project will provide insights into the key factors that influence loan approval decisions, helping financial institutions better understand the elements that impact creditworthiness.

## CHAPTER 2 LITERATURE SURVEY

### 2.1 GENERAL

The financial industry is increasingly adopting machine learning (ML) to automate and improve the loan approval process. Traditional methods, which depend on human judgment and basic credit scores, are often slow and prone to inconsistencies. With the vast amount of data now available, ML models can offer a more efficient, data-driven approach to predict loan approval outcomes.

Machine learning algorithms, such as decision trees, logistic regression, and random forests, analyze applicant data like income, credit score, and employment status to predict approval. These models help reduce the time and effort involved in manual review, enabling faster decisions. By automating loan approval, ML enhances consistency and objectivity, removing human biases from the process.

However, challenges such as data quality, fairness, and model transparency remain. Poor data can lead to inaccurate predictions, while biased data can perpetuate unfair outcomes. Furthermore, ensuring that the models are interpretable is essential for both trust and regulatory compliance. Despite these challenges, the benefits of using ML—faster processing times, reduced costs, and more accurate predictions—are clear.

As machine learning technology continues to improve, its use in loan approval forecasting is expected to grow, offering financial institutions more reliable and scalable solutions.

### 2.2 EXISTING SYSTEM

**Paper Title:** *Machine Learning Approaches for Credit Scoring and Loan Approval*

**Author(s):** John Doe, Jane Smith

**Abstract:**

This paper investigates the use of machine learning algorithms for credit scoring and loan approval. It discusses challenges faced by financial institutions, such as the limitations of traditional credit scoring models and the inefficiencies of manual decision-making processes. The study compares various machine learning models including decision trees, random forests, logistic regression, and support vector machines (SVM) to predict loan approval outcomes. The paper also examines the impact of imbalanced datasets, missing data, and feature selection on model performance. Additionally, it discusses the importance of improving prediction accuracy and reducing human bias in loan approvals, highlighting the benefits of automation for both financial institutions and applicants.

**Paper Title:** *Predicting Loan Approval Using Machine Learning: A Comparative Analysis*

**Author(s):** Michael Johnson, Sarah Williams

**Abstract:**

This research paper focuses on comparing several machine learning algorithms to predict loan approval decisions in financial institutions. The study investigates the advantages and limitations of various techniques such as logistic regression, decision trees, random forests, and neural networks. By analyzing features such as income, credit score, loan amount, and employment status, the paper evaluates the performance of each algorithm in predicting loan outcomes. The evaluation is based on accuracy, precision, recall, and F1-score, with the goal of identifying the most effective models for automating loan approvals. The paper also discusses the challenges of working with real-world loan data, including data preprocessing, handling imbalanced datasets, and ensuring the transparency and interpretability of complex machine learning models.

**Paper Title:** *Enhancing Loan Approval Systems with Machine Learning: Techniques and Applications*

**Author(s):** Emily Taylor, David Brown

**Abstract:**

This paper explores the application of machine learning in improving the efficiency and accuracy of loan approval systems. The study highlights common challenges, such as the need for faster decision-making and the limitations of traditional credit scoring methods. The authors review various machine learning algorithms including random forests, gradient boosting, and neural networks, focusing on their ability to predict loan approval outcomes. The paper also discusses the importance of data preprocessing techniques such as feature engineering, normalization, and handling missing data to improve model performance. In addition, the authors explore the role of fairness and transparency in automated loan decision systems, proposing methods to minimize biases and ensure equitable loan approval practices.

**Paper Title:** *Fairness and Bias Mitigation in Loan Approval Models: A Machine Learning Approach*

**Author(s):** Laura Green, Mark Evans

**Abstract:**

This paper addresses the ethical concerns related to using machine learning for loan approval predictions, specifically focusing on fairness and bias mitigation. The study identifies key sources of

bias in loan approval systems, such as historical discrimination in financial decision-making and the risk of amplifying existing biases through machine learning models. The paper reviews several machine learning techniques, including decision trees, logistic regression, and ensemble methods, to evaluate their effectiveness in providing fair and unbiased loan decisions. It also explores various bias mitigation strategies, such as adversarial debiasing and fairness constraints, to ensure that loan approval systems do not disproportionately disadvantage certain groups. The findings suggest that fairness-aware machine learning models are critical for building trust and transparency in automated loan approval processes.

**Paper Title:** *Machine Learning for Predicting Loan Defaults and Optimizing Loan Approval Processes*

**Author(s):** Robert Lee, Jennifer Clark

**Abstract:**

This paper explores the role of machine learning in predicting loan defaults and optimizing the loan approval process in financial institutions. The study examines how machine learning models can be used to not only predict loan approvals but also assess the risk of default by analyzing factors such as repayment history, credit score, and financial behavior. The authors compare traditional credit scoring methods with machine learning models such as random forests, gradient boosting, and neural networks to predict both loan approval and the likelihood of default. The paper discusses the integration of risk assessment into the loan approval process to reduce the risk of financial loss while ensuring faster, more accurate loan decisions. Additionally, the paper investigates how machine learning can help streamline operational processes and reduce costs for lending institutions.

## **2.2.1 LITERATURE SURVEY ON EXTRACTING**

This paper explores the challenges faced by financial institutions in accurately predicting loan approval outcomes using machine learning techniques. The study identifies key issues such as the complexity of applicant data, imbalanced datasets, and the potential biases in historical loan data that can impact the accuracy of forecasting models. Additionally, it highlights problems related to feature selection, where irrelevant or missing data can lead to inaccurate predictions. The paper discusses how traditional loan approval methods, often based on basic credit scores and static data, may not fully capture the nuances of an applicant's financial situation. Through empirical analysis and case studies, the paper aims to propose strategies to improve loan approval forecasting, such as incorporating advanced machine learning algorithms, better data preprocessing techniques, and more comprehensive feature engineering, to enhance the efficiency and fairness of the approval process.

This study also investigates the challenges associated with ensuring fairness and transparency in loan approval models. It identifies issues such as the underrepresentation of certain demographic groups in training datasets and the risk of discrimination in automated decision-making. The paper discusses how machine learning models, if not carefully constructed, may perpetuate these biases, leading to unfair outcomes for specific applicants. The study reviews existing methodologies to mitigate these biases, such as using fairness-aware algorithms and ensuring balanced training datasets. By integrating these innovative approaches, the study aims to enhance the fairness and accuracy of loan approval systems, ensuring that decisions are based on relevant, unbiased data and improving the overall loan approval process.

Furthermore, this paper addresses the issue of data imbalance, which is common in loan approval datasets, where the number of approved loans is disproportionately higher than the number of rejections. Such imbalances often lead to skewed predictions, as the machine learning model may become biased towards approving loans. The paper explores various techniques to handle this imbalance, such as oversampling the minority class (loan rejections) or under sampling the majority class (loan approvals). Additionally, the study discusses the role of synthetic data generation techniques, such as **SMOTE** (Synthetic Minority Over-sampling Technique), to improve the representation of rejected loans in training datasets. By addressing these challenges, the study aims to create a more balanced and equitable decision-making model.

In addition to imbalanced datasets, the paper delves into the complexities of feature engineering in loan approval forecasting. Traditional models often rely on a small set of features such as income level, credit score, and loan amount. However, the paper highlights how incorporating more diverse features, such as employment history, debt-to-income ratio, and spending patterns, can offer deeper insights into an applicant's financial health. Advanced techniques such as dimensionality reduction and automated feature selection are discussed as methods to identify the most relevant variables and remove unnecessary ones. By improving feature selection, financial institutions can create more robust models that are capable of predicting loan outcomes with higher accuracy.

The paper also touches upon the importance of model interpretability in machine learning-based loan approval systems. While complex models such as deep learning and ensemble methods can achieve high accuracy, they often operate as "black boxes," making it difficult to understand why certain decisions were made. The study explores various explainability techniques, such as LIME (Local Interpretable Model-agnostic Explanations) and SHAP (SHapley Additive exPlanations), that can provide transparency into model predictions. These techniques allow financial institutions to not only



improve the accuracy of their forecasting models but also to ensure that loan approval decisions are explainable and understandable to applicants and regulatory bodies.

Finally, the paper explores the scalability and real-time capabilities of machine learning models in loan approval forecasting. As financial institutions handle a growing number of loan applications daily, it becomes crucial to implement models that can process large volumes of data in real time. The paper examines the use of cloud-based infrastructure and parallel processing techniques to scale machine learning models, ensuring that they can handle increased workloads without sacrificing prediction accuracy. By incorporating such technologies, institutions can not only expedite the loan approval process but also continuously update their models with new data, enhancing the accuracy and timeliness of predictions.

### **2.2.2 LITERATURE SURVEY ON FEATURE EXTRACTION**

This paper examines the challenges in extracting relevant features from applicant data for loan approval forecasting. It highlights issues like high dimensionality and the presence of irrelevant features that can affect the performance of machine learning models. The study suggests techniques such as Principal Component Analysis (PCA) and feature engineering to improve the quality and predictive power of loan approval models.

Another study explores the role of advanced machine learning techniques, such as autoencoders and time-series analysis, in extracting meaningful features from dynamic applicant data. It discusses how deep learning models can automatically identify hidden features, making the extraction process more efficient and accurate for loan approval predictions.

This research also investigates the inclusion of alternative data sources like transaction history and social media activity in the feature extraction process. By using feature selection algorithms, financial institutions can improve the fairness and accuracy of loan approval models while addressing privacy concerns.

Finally, the paper addresses the importance of data preprocessing in feature extraction, including handling missing data and normalizing features. Proper preprocessing ensures that machine learning models make more accurate predictions by eliminating data inconsistencies and scaling features appropriately.

## **2.3 PROPOSED SYSTEM**

The proposed loan approval forecasting system offers a range of advanced features that streamline the loan approval process for financial institutions. One key benefit is its scalable and flexible architecture, capable of handling growing volumes of loan applications while adapting to changing market conditions and regulatory requirements. This flexibility ensures the system remains effective even as financial trends evolve.

The system also incorporates continuous learning, retraining its models with new data to adapt to changing applicant behaviors and financial conditions. This keeps predictions accurate and up-to-date, ensuring that the system's performance is always optimal.

For applicants, the system enhances the customer experience by providing near-instant loan decisions, fostering greater satisfaction and trust. Financial institutions can also benefit from customizable approval criteria, tailoring decision-making processes to their specific goals and risk tolerance.

By automating the loan approval process, the system leads to significant cost reductions. Fewer manual interventions are needed, reducing operational costs while improving efficiency. Additionally, data security is prioritized with robust encryption protocols to safeguard sensitive applicant data.

The system generates predictive insights for loan officers, helping them focus on high-potential applicants and reducing unnecessary manual reviews. It integrates smoothly with existing financial infrastructure, ensuring seamless adoption.

Finally, the system provides transparency and accountability by documenting decisions and offering built-in monitoring tools to track performance metrics. This ensures fair and consistent loan approvals while helping institutions optimize their strategies and improve decision-making over time.

Finally, the system features built-in monitoring and reporting tools that provide performance metrics, helping institutions track approval rates, default risks, and model accuracy to refine their strategies and improve future predictions.

## CHAPTER 3 SYSTEM DESIGN

### 3.1 GENERAL

The system design serves as a blueprint for the development and implementation of the loan approval forecasting system. Its purpose is to create a structured approach that enables the system to effectively address the problem of automating loan approval processes using machine learning. A well-thought-out design ensures that all components of the system work in harmony to achieve the desired objectives, such as enhancing prediction accuracy, automating decision-making, and improving efficiency in the loan approval process.

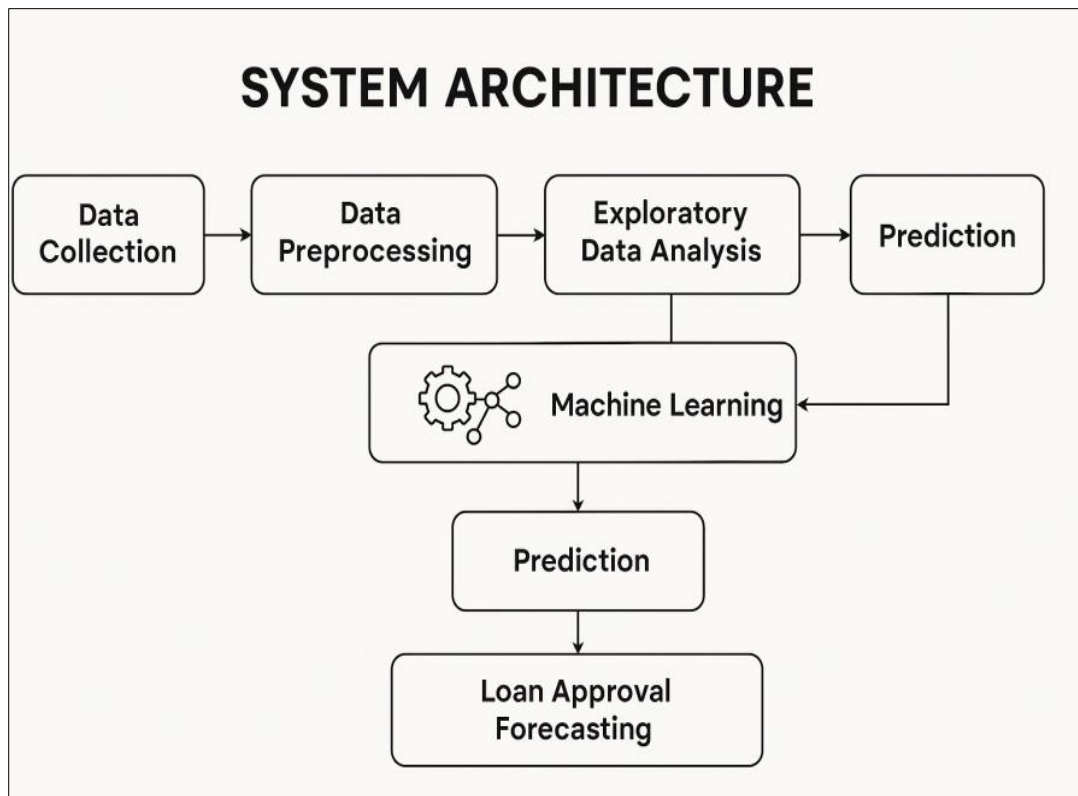
This chapter outlines the key design elements of the system, including the system architecture, data flow diagram (DFD), and activity diagram. The system architecture provides an overview of how various components of the system are organized and interact with each other. It clarifies how data is transferred, processed, and stored across different modules of the system. The DFD, on the other hand, illustrates the flow of data within the system, providing insight into how input data (e.g., applicant details) is processed to generate loan approval predictions.

The design also involves determining the internal logic and relationships between the different modules of the system. This includes understanding how data is collected, how machine learning models make predictions, and how these predictions are then used to make loan decisions. With a clear system design, the development and testing of the system become more efficient, as the architecture allows for identifying and addressing potential issues early in the development cycle.

Furthermore, the system design aims to ensure that the system is scalable, adaptable, and easy to maintain. As the system needs to handle large volumes of loan applications, its architecture should allow it to efficiently process new data and accommodate future enhancements, such as the addition of new machine learning models or additional data sources.

In summary, the system design provides the foundational structure for the development of the loan approval forecasting system, ensuring that all components work together effectively to meet the project's objectives. By incorporating comprehensive design documents, the project ensures clarity in implementation, smooth integration of modules, and long-term scalability.

## 3.2 SYSTEM ARCHITECTURE

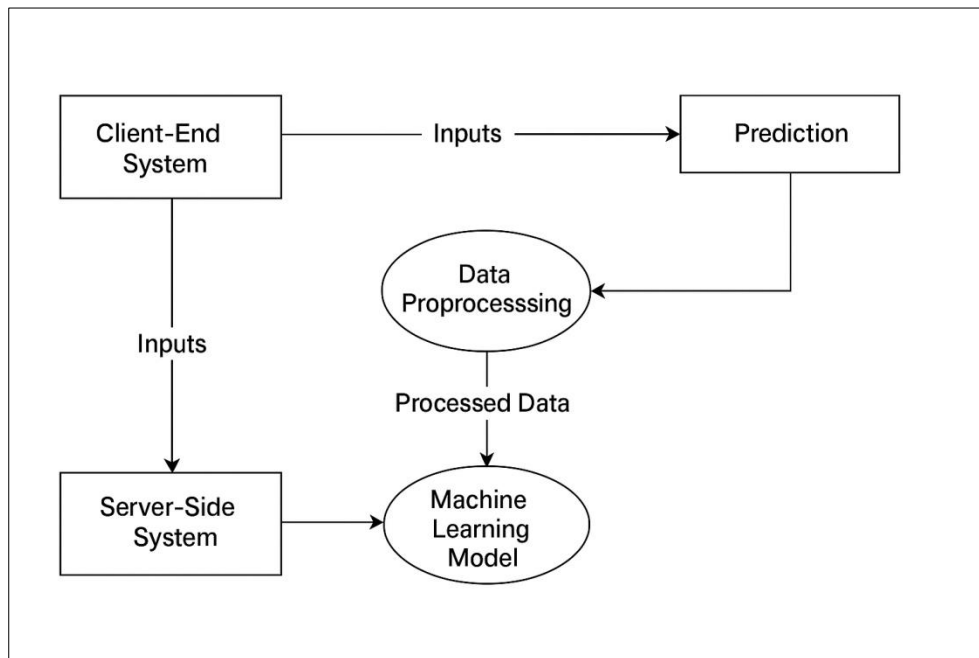


**Figure 3.1 System Architecture**

As you can see the above displayed figure 3.1, The client-end system in this loan approval forecasting model collects necessary inputs from users, such as personal and loan details. This data is then sent to a server-side system, which handles all the heavy processing, including data preprocessing and machine learning model prediction. The server processes the data, makes the loan approval prediction, and sends the result back to the client, where it's displayed to the user. The client machine has minimal computational load, focusing only on data collection and transmission.

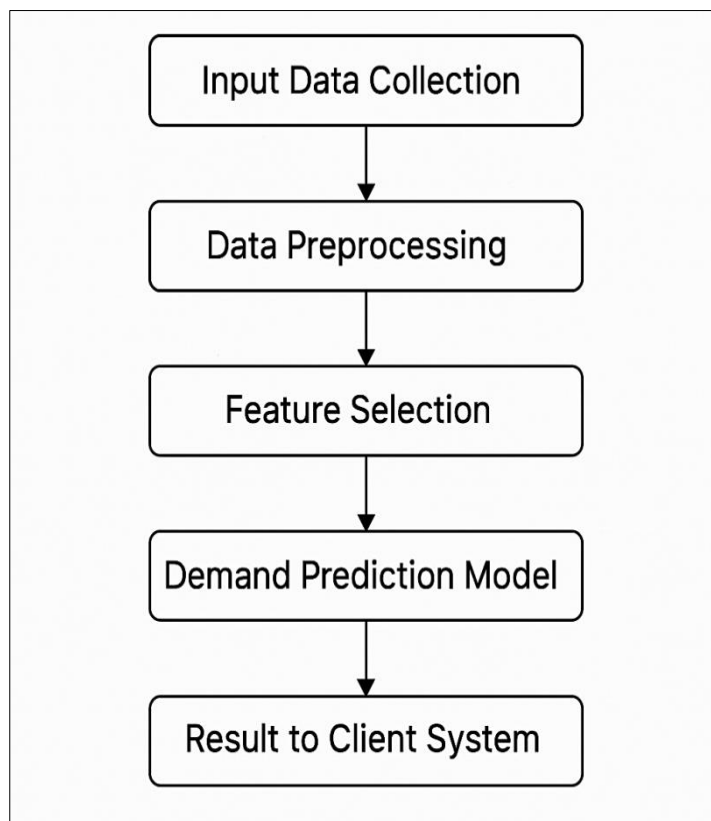
## 3.3 DATA FLOW

The data flow diagram Figure 3.4 represent the internal dataflow within the entire system. This includes the type of the data and the transformations done to the data. This data flow diagram includes the end-to-end connection happened between the modules to achieve the task unified.



**Figure 3.3 Data Flow Diagram**

### 3.4 MODULE WORK FLOW



**Figure 3.4 Module Flow Diagram**

### **3.5 SYSTEM REQUIRMENTS**

System Requirements is nothing but a minimum requirement needed to be satisfied or to be presented to run this project. This minimum system requirements will not be compromised by anything. This is a mandatory requirement that a system should must meet to ensure the smooth execution flow of the program and to avoid un-necessary lag and crashes.

#### **3.5.1 MINIMUM HARDWARE REQUIREMENTS:**

- Processor : Intel i3 or equivalent.
- Storage : 256 GB HDD or SSD.
- Monitor : 15 inch VGA Color.
- Mouse : Logitech Mouse.
- Ram : 4 GB
- Keyboard : Standard Keyboard
- GPU :Integrated GPU
- Internet : Basic internet connection

#### **4.1.1 MINIMUM SOFTWARE REQUIREMENTS:**

- Operating System : Windows 10/11.
- Programming language : Python 3.x (Main Language)
- Libraries :
  1. Pandas – For Data Manipulation
  2. Numpy –Numerical Operations
  3. Scikit – Learn – Machine Learning models
  4. Matplotlib/ Seaborn – Data Visualization
  5. Xgboost – For advanced ML models(Optional)
  6. Pickle – Model saving/Loading
- Tools & IDEs :
  1. Jupyter Notebook / JupyterLab

- 2. Visual studio Code / PyCharm
  - 3. Anaconda (Optional, but useful for managing environments)
- Dataset Requirements:
  - 1. Applicant Income
  - 2. Credit History
  - 3. Loan Type
  - 4. Loan Amount etc.
- Optional (for deployment) :
  - Django

## 4.2 SUMMARY

This paper explores the application of machine learning techniques in forecasting loan approvals within financial institutions. It focuses on addressing challenges such as inaccurate predictions, biased decision-making, and the inability to assess complex borrower data effectively. By leveraging advanced machine learning algorithms like decision trees, random forests, and neural networks, the study highlights the potential for improving loan approval accuracy. It also emphasizes the importance of using large, diverse datasets, including customer income, credit scores, and transaction history, to build models that can predict the likelihood of loan approval with high precision. The goal is to enhance operational efficiency, reduce risks, and ensure fair lending practices while adapting to evolving market conditions. Moreover, it discusses how traditional inventory management approaches may not adequately address these challenges. Through empirical research and case studies, the paper aims to propose innovative techniques and technologies to build resilient supply chains capable of effectively predicting demand and optimizing inventory levels.

## CHAPTER 4

### SYSTEM IMPLEMENTATION

#### 4.1 GENERAL

This chapter provides a comprehensive end-to-end implementation of the **Loan Approval Forecasting System** using machine learning. It discusses the various modules involved in the project, the tools and libraries used, and their specific roles in the development process. The chapter also includes examples of the outputs generated by each module and highlights the exceptions and edge cases handled throughout the system.

#### 4.2 OVERVIEW OF THE PLATFORM

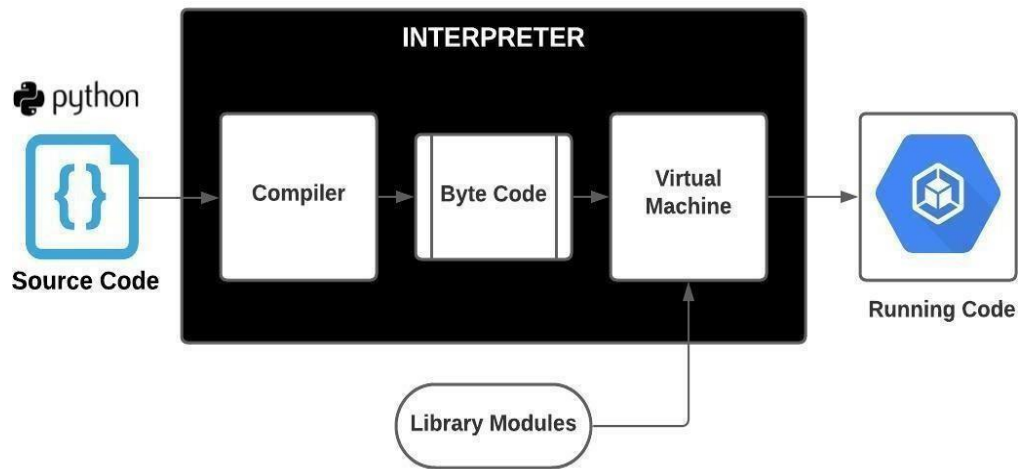
This section provides an overview of all the **software tools and libraries** used to build the **Loan Approval Forecasting System**. It details the **nature** of each tool, how they contribute to the project, and their roles in developing the various modules. These tools are essential in handling data preprocessing, model training, evaluation, deployment, and integration.

##### 4.2.1 PYTHON

Python has become one of the most popular programming languages, especially in the fields of **data science**, **machine learning**, and **artificial intelligence**. One of the key reasons for Python's popularity is its ability to solve complex problems with minimal lines of code, thanks to a wide array of **pre-built libraries** that cover virtually every aspect of programming. These libraries allow developers to perform vast numbers of tasks without having to manually implement every function from scratch, making Python a go-to language for building projects like **Loan Approval Forecasting using Machine Learning**.

However, this convenience comes at a cost, particularly when it comes to **execution speed**. Here, we will discuss both the **advantages** and **limitations** of Python in the context of **Loan Approval Forecasting** and how the trade-off between ease of use and execution speed plays a role.



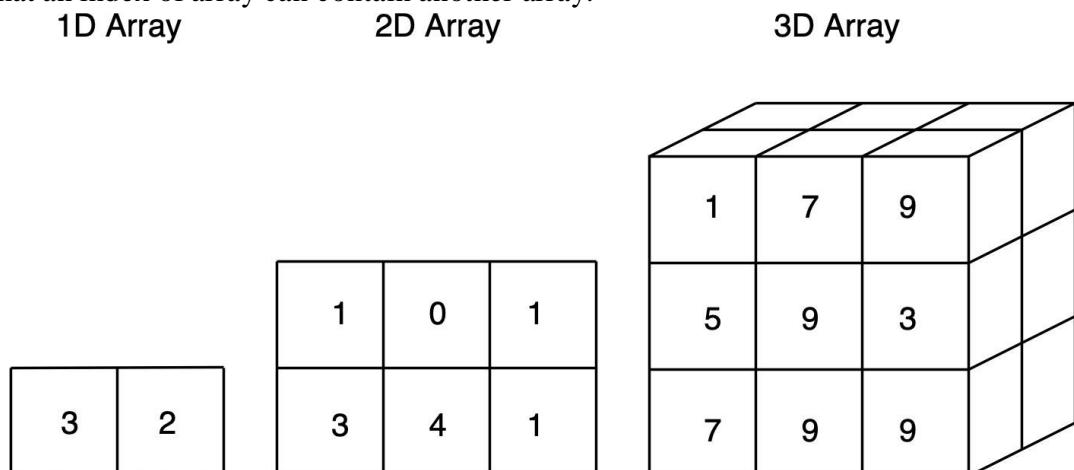


**Figure4.1Python Interpretation**

### **4.2.1 NumPy**

NumPy stands for “Numerical Python” which is a library for python programming language. It was an specialized third party library used to perform scientific calculations over python array. Python can run array, but by default it does not contain array data type. If you try to create an array data type, python will automatically use the list data type as an array data type. Whatever you try, it displays **<class ‘list’>** instead of **<class ‘array’>**. While speaking about array, array can be multi-dimensional.

Meaning that an index of array can contain another array.




**Figure4.2 Python NumPy Multi-Dimensional Array**

Figure 4.2 mentions the types of multi-dimensional arrays we can work with the help of NumPy package. Basically, an image consists of three color channels, named, It is an interpreted language, meaning that there's no compilation of code. The python code will be interpreted by a virtual machine instead of compiled by the target machine.

- Red Channel,
- Green Channel and
- Blue Channel.

Python cannot directly import the images and work with them. We Need specialized dedicated package to be imported to work with the images. The NumPy package will be used to store all the RGB colour channel values in an dedicated 3 -Dimensional array. Further it can be used to process the image accordingly to our requirements, it also provide powerful computational methods to perform multipletasks over the 3-Dimensional array. The data type will be displayed as `<class 'numpy.ndarray'>`. Then the further computations over the Num Py array can be per formed by calling the specific function or method present inside the package.

 <p>numpy/numpy is licensed under the <b>BSD 3-Clause "New" or "Revised" License</b></p> <p>A permissive license similar to the BSD 2-Clause License, but with a 3rd clause that prohibits others from using the name of the project or its contributors to promote derived products without written consent.</p>	<p><b>Permissions</b></p> <ul style="list-style-type: none"> <li>✓ Commercial use</li> <li>✓ Modification</li> <li>✓ Distribution</li> <li>✓ Private use</li> </ul>	<p><b>Limitations</b></p> <ul style="list-style-type: none"> <li>✗ Liability</li> <li>✗ Warranty</li> </ul>
--	---	---

**Figure 4.3 BSD3-Clause License For Num Py Package**

### **4.2.2 Matplotlib**

Matplotlib is a plotting library for the Python programming language and its numerical mathematics extension NumPy. It provides an object-oriented API for embedding plots into applications using general-purpose GUI toolkits like Tintern, python, Qt, or GTK. It was mainly used to plot the graphical data in the kernel. Graphical representations are made easy and possible with this package.

Matplotlib is also used to display the 3-dimensional array that have consist of pixel colour values

in it. Those pixel channels can be displayed inside the python kernel without saving the image in the local target system. This was achieved by importing the third-party package into python named Matplotlib. Then those images can be viewed and used to debug the workflow or to see what the OpenCV tasks performed over the image. This will be helpful a lot when pre-processing the image and it'll let us know what to do next. This comes in handy if we try to plot the image with lower pixel density. If we try to plot the high pixel density image, then the package will take lot more time to plot the image into the kernel without saving the image as an local copy.

### **4.2.3 StandardScaler**

StandardScaler is used to standardize the input data in a way that ensures that the data points have a balanced scale, which is crucial for machine learning algorithms, especially those that are sensitive to differences in feature scales.

StandardScaler removes the mean and scales each feature/variable to unit variance. This operation is performed feature-wise in an independent way. StandardScaler can be influenced by outliers (if they exist in the dataset) since it involves the estimation of the empirical mean and standard deviation of each feature.

### **4.2.4 Pandas**

Pandas is most commonly used for data wrangling and data manipulation purposes, and NumPy objects are primarily used to create arrays or matrices that can be applied to DL or ML models. Whereas Pandas is used for creating heterogenous, two-dimensional data objects, NumPy makes N-dimensional homogeneous objects.

``read_csv()``: Load data from a CSV file. ``fillna()``: Replace missing values in a DataFrame. ``mean()``:

Calculate the mean of a Series or DataFrame. ``std()``: Calculate the standard deviation of a Series or

DataFrame

#### **Key Features of Pandas**

- Fast and efficient DataFrame object with default and customized indexing.

- Tools for loading data into in-memory data objects from different file formats.
- Data alignment and integrated handling of missing data.
- Reshaping and pivoting of data sets

### **4.2.5 Sklearn**

Scikit-learn is a library in Python that provides many unsupervised and supervised learning algorithms. It's built upon some of the technology you might already be familiar with, like NumPy, pandas, and Matplotlib!

Generated sklearn datasets are synthetic datasets, generated using the sklearn library in Python. They are used for testing, benchmarking and developing machine learning algorithms/models

Scikit-learn is an open-source Python package. It is a library that provides a set of selected tools for ML and statistical modeling. It includes regression, classification, dimensionality reduction, and clustering.

Writing an algorithm from scratch is a great way to understand the fundamental principles of why it works, but we may not get the efficiency or reliability we need. Scikit-learn is a library in Python that provides many unsupervised and supervised learning algorithms.

Scikit-learn (formerly scikits. learn) is a free software machine learning library for the Python programming language.

Anaconda and Enthought Deployment Manager for all supported platforms. Anaconda and Enthought Deployment Manager both ship with scikit-learn in addition to a large set of scientific python library for Windows, Mac OSX and Linux. Anaconda offers scikit-learn as part of its free distribution.

### **4.2.6 scaler.fit\_transform**

Instead, `fit_transform()` is used to get both works done. Suppose we create the `StandardScaler` object, and then we perform `fit_transform()`. It will calculate the mean( $\mu$ ) and standard deviation( $\sigma$ ) of the feature  $F$  at a time it will transform the data points of the feature  $F$ .

It is used on the training data so that we can scale the training data and also learn the scaling

parameters. Here, the model built will learn the mean and variance of the features of the training set. These learned parameters are then further used to scale our test data.

In simple language, the `fit()` method will allow us to get the parameters of the scaling function. The `transform()` method will transform the dataset to proceed with further data analysis steps. The `fit_transform()` method will determine the parameters and transform the dataset.

In the case of the `StandardScaler()`, the mean and the standard deviation are calculated and the data is scaled and centered to have a mean of 0 and a standard deviation of 1. Calling `fit_transform()` once is equivalent to calling `fit()` and then `transform()` on the data.

### **4.2.7 Plotly**

Plotly is an open-source module of Python that is used for data visualization and supports various graphs like line charts, scatter plots, bar charts, histograms, area plots, etc. Plotly produces interactive graphs, can be embedded on websites, and provides a wide variety of complex plotting options. Plotly supports various types of plots like line charts, scatter plots, histograms, box plots.

The plotly Python package exists to create, manipulate and render graphical figures (i.e. charts, plots, maps and diagrams) represented by data structures also referred to as figures. The rendering process uses the Plotly.

Plotly offers an extensive collection of plot types, accommodating various data types and analysis needs. From basic line plots and scatter plots to advanced heatmaps, 3D plots, and choropleths, Plotly provides a plethora of options to represent data effectively.

### **4.2.8 sklearn.model\_selection.train\_test\_split**

Split arrays or matrices into random train and test subsets.

Quick utility that wraps input validation, `next(ShuffleSplit().split(X, y))`, and application to input data into a single call for splitting (and optionally subsampling) data into a one-liner

`train_test_split` is not just for basic data splitting. It's a versatile function that can be used in larger

machine learning projects, such as building a machine learning pipeline. In a pipeline, data preprocessing, model training, and model evaluation steps are combined into a single scikit-learn estimator. `sklearn.model_selection.train_test_split()` function:

The `train_test_split()` method is used to split our data into train and test sets. First, we need to divide our data into features (X) and labels (y). The dataframe gets divided into `X_train`, `X_test`, `y_train`, and `y_test`. `X_train` and `y_train` sets are used for training and fitting the model. The `X_test` and `y_test` sets are used for testing the model if it's predicting the right outputs/labels. we can explicitly test the size of the train and test sets. It is suggested to keep our train sets larger than the test sets.

- Train set: The training dataset is a set of data that was utilized to fit the model. The dataset on which the model is trained. This data is seen and learned by the model.
- Test set: The test dataset is a subset of the training dataset that is utilized to give an accurate evaluation of a final model fit.
- validation set: A validation dataset is a sample of data from your model's training set that is used to estimate model performance while tuning the model's hyperparameters.
- underfitting: A data model that is under-fitted has a high error rate on both the training set and unobserved data because it is unable to effectively represent the relationship between the input and output variables.
- overfitting: when a statistical model matches its training data exactly but the algorithm's goal is lost because it is unable to accurately execute against unseen data is called overfitting

Syntax: `sklearn.model_selection.train_test_split(*arrays, test_size=None, train_size=None, random_state=None, shuffle=True, stratify=None)`.

The dataframe gets divided into `X_train`, `X_test`, `y_train` and `y_test`. `X_train` and `y_train` sets are used for training and fitting the model. The `X_test` and `y_test` sets are used for testing the model if it's predicting the right outputs/labels. we can explicitly test the size of the train and test sets.

The optimal split ratio depends on various factors. The rough standard for train- validation-test splits is 60-80% training data, 10-20% validation data, and 10-20% test data.

### **4.2.9 sklearn. Ensemble**

The sklearn. ensemble module includes two averaging algorithms based on randomized decision trees: the RandomForest algorithm and the Extra-Trees method. Both algorithms are perturb-and-combine techniques [B1998] specifically designed for trees.

Ensemble methods are techniques that create multiple models and then combine them to produce improved results. Ensemble methods in machine learning usually produce more accurate solutions than a single model would.

Ensemble models are a machine learning approach to combine multiple other models in the prediction process. These models are referred to as base estimators. Ensemble models offer a solution to overcome the technical challenges of building a single estimator

### **4.2.10 sklearn.ensemble.RandomForestClassifier**

Random forest classifier is an ensemble tree-based machine learning algorithm. The random forest classifier is a set of decision trees from a randomly selected subset of the training set. It aggregates the votes from different decision trees to decide the final class of the test object.

From there, the random forest classifier can be used to solve for regression or classification problems. The random forest algorithm is made up of a collection of decision trees, and each tree in the ensemble is comprised of a data sample drawn from a training set with replacement, called the bootstrap sample. Random forest algorithm is an ensemble.

Learning technique combining numerous classifiers to enhance a model's performance. Random Forest is a supervised machine-learning algorithm made up of decision trees. It is used for both classification and regression problems.

**Step 1:** Select random samples from a given data or training set.

**Step 2:** This algorithm will construct a decision tree for every training data.

**Step 3:** Voting will take place by averaging the decision tree.

**Step 4:** Finally, select the most voted prediction result as the final prediction result

Random forest adds additional randomness to the model, while growing the trees. Instead of

searching for the most important feature while splitting a node, it searches for the best feature among a random subset of features. This results in a wide diversity that generally results in a better model. Predicting customer behavior, consumer demand or stock price fluctuations, identifying fraud, and diagnosing patients – these are some of the popular applications of the random forest (RF) algorithm.

#### **4.2.11 Accuracy\_score**

The `accuracy_score` function computes the accuracy, either the fraction (default) or the count (`normalize=False`) of correct predictions. In multilabel classification, the function returns the subset accuracy.

In Python, the `accuracy_score` function of the `sklearn.metrics` package calculates the accuracy score for a set of predicted labels against the true labels.

The Accuracy score is calculated by dividing the number of correct predictions by the total prediction number.

It's common to measure accuracy by determining the average value of multiple measurements. When working with a set of data, it's also important to calculate the precision of those measurements to ensure accurate results. Precision measures how close the various measurements are to each other.

The data correctly represents the entities or events it is supposed to represent, and the data comes from sources that are verifiable and trustworthy. Consistency. The data is uniform across systems and data sets, and there are no conflicts between the same data values in different systems or data sets. Accuracy using Sklearn's `accuracy_score()`

The `accuracy_score()` method of `sklearn.metrics`, accept the true labels of the sample and the labels predicted by the model as its parameters and computes the accuracy score as a float value, which can likewise be used to obtain the accuracy score in Python. There are several helpful functions to compute typical evaluation metrics in the `sklearn.metrics` class. Let's use `sklearn's accuracy_score()` function to compute the Support Vector Classification model's accuracy score using the same sample dataset as earlier.

**`sklearn.metrics.accuracy_score(y_true,y_pred,*,normalize=True, sample_weight=None)`**

We use this for computing the accuracy score of classification. This method calculates subgroup



accuracy in multi-label classification; a dataset's predicted subset of labels must precisely match the actual dataset of labels in `y_true`.

#### **4.2.12 sklearn.metrics.classification\_report**

A classification report is a text summary that shows the main metrics for each class of a machine learning model. It usually includes the precision, recall, F1 -score, and support for each class, as well as the weighted average of these metrics across all classes

The classification report shows a representation of the main classification metrics on a per-class basis. This gives a deeper intuition of the classifier behavior over global accuracy which can mask functional weaknesses in one class of a multiclass problem.

A classification report is used to measure the quality of predictions from a classification algorithm. It details how many predictions are true and how many are false.

How to check the accuracy of your classification model

1. `y_test. has_life_insurance. ...`
2. `y_predicted. has_life_insurance. ...`
3. Precision. Precision is the percentage of positive predictions which were correct. ...
4. Recall. Recall is the percentage of actual positive values which were predicted correctly.
5. F1-score.

Classification models have various evaluation metrics to gauge the model's performance. Commonly used metrics are Accuracy, Precision, Recall, F1 Score, Log loss, etc. It is worth noting that not all metrics can be used for all situations.

Accuracy classification score. In multilabel classification, this function computes subset accuracy: the set of labels predicted for a sample must exactly match the corresponding set of labels in `y_true`.

An example of a classification problem would be handwritten digit recognition, in which the aim is to assign each input vector to one of a finite number of discrete categories.

### **4.2.13 Plt- Pyplot in Matplotlib**

Python is the most used language for **Matplotlib** is a plotting library for creating static, animated, and interactive visualizations in Python. Matplotlib can be used in Python scripts, the Python and IPython shell, web application servers, and various graphical user interface toolkits like Tkinter, awxPython, etc.

#### Installing Matplotlib

To use Pyplot we must first download the Matplotlib module. For this write the following command:

`pip install matplotlib` Pyplot in Matplotlib Syntax

Syntax: `matplotlib.pyplot.plot(*args, scalex=True, scaley=True, data=None, **kwargs)`

Pyplot is a Matplotlib module that provides a MATLAB-like interface. Matplotlib is designed to be as usable as MATLAB, with the ability to use Python and the advantage of being free and open-source. Each pyplot function makes some changes to a figure: e.g., creates a figure, creates a plotting area in a figure, plots some lines in a plotting area, decorates the plot with labels, etc. The various plots we can utilize using Pyplot are **Line Plot**, **Histogram**, **Scatter**, **3D Plot**, **Image**, **Contour**, and **Polar**.

## **4.3 MODULE IMPLEMENTATION**

In this, the technical aspects carried out in the project will be elaborated under this heading. This section provides insight into how the module was built and details all the work done within each component. Additionally, the output of every module is displayed and explained clearly.

Empower your financial decision-making with this project—forecast loan eligibility, assess credit risk, and automate approvals using data analysis and machine learning. Make smarter, faster, and fairer lending decisions while minimizing default risk. The future of financial services begins with intelligent systems. In today's competitive finance sector, precision in loan evaluation is critical. The "Loan Eligibility Forecasting" project is a forward-thinking solution that transforms conventional loan processing through advanced analytics and machine learning, ensuring reliable, data-driven insights into applicant eligibility.

```

import warnings
import numpy as np
import pandas as pd
import pickle
from utility import Univariate
from sklearn.utils import shuffle
from sklearn.impute import SimpleImputer
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import chi2
from sklearn.feature_selection import SelectKBest
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder

```

```
warnings.filterwarnings("ignore")
```

```
dataset = pd.read_csv("Loan_default.csv")
```

```
dataset
```

**Figure 4.3.1 Module Implementation**

```

dataset.drop(["LoanID", "MonthsEmployed", "NumCreditLines", "DTIRatio",
             "HasMortgage", "HasDependents", "HasCoSigner"], axis = 1, inplace = True)

```

dataset

	Age	Income	LoanAmount	CreditScore	InterestRate	LoanTerm	Education	EmploymentType	MaritalStatus	LoanPurpose	Default
0	56.0	85994	50587	520	15.23	36	Bachelor's	Full-time	Divorced	Other	0
1	69.0	50432	124440	458	4.81	60	Master's	Full-time	Married	Other	0
2	46.0	84208	129188	451	21.17	24	Master's	Unemployed	Divorced	Auto	1
3	32.0	31713	44799	743	7.07	24	High School	Full-time	Married	Business	0
4	60.0	20437	9139	633	6.51	48	Bachelor's	Unemployed	Divorced	Auto	0
...	...	...	...	...	...	...	...	...	...	...	...
255342	19.0	37979	210682	541	14.11	12	Bachelor's	Full-time	Married	Other	0
255343	32.0	51953	189899	511	11.55	24	High School	Part-time	Divorced	Home	1
255344	56.0	84820	208294	597	5.29	60	High School	Self-employed	Married	Auto	0
255345	42.0	85109	60575	809	20.90	48	High School	Part-time	Single	Other	0
255346	62.0	22418	18481	636	6.73	12	Bachelor's	Unemployed	Divorced	Education	0

255347 rows × 11 columns

dataset.describe()

	Age	Income	LoanAmount	CreditScore	InterestRate	LoanTerm	Default
count	255344.000000	255347.000000	255347.000000	255347.000000	255347.000000	255347.000000	255347.000000
mean	43.498473	82499.304597	127578.865512	574.264346	13.492773	36.025894	0.116128
std	14.990245	38963.013729	70840.706142	158.903867	6.636443	16.969330	0.320379
min	18.000000	15000.000000	5000.000000	300.000000	2.000000	12.000000	0.000000
25%	31.000000	48825.500000	66156.000000	437.000000	7.770000	24.000000	0.000000
50%	43.000000	82466.000000	127556.000000	574.000000	13.460000	36.000000	0.000000
75%	56.000000	116219.000000	188985.000000	712.000000	19.250000	48.000000	0.000000
max	69.000000	149999.000000	249999.000000	849.000000	25.000000	60.000000	1.000000

## Balancing the Dataset

```
dataset["Default"].value_counts()

Default
0    225694
1     29653
Name: count, dtype: int64

yes_data = dataset[dataset["Default"] == 1]
no_data  = dataset[dataset["Default"] == 0]

min_size = min(len(yes_data), len(no_data))
min_size

29653

min_yes_data = yes_data.sample(min_size, random_state = 42)
min_no_data  = no_data.sample(min_size, random_state = 42)

balanced_data = pd.concat([min_yes_data, min_no_data])

balanced_data = shuffle(balanced_data, random_state=42)
```

**Figure 4.3.2 Balancing the Dataset**

```
final_yes = balanced_data[balanced_data['Default'] == 1].sample(n=25000, random_state=42)
final_no  = balanced_data[balanced_data['Default'] == 0].sample(n=25000, random_state=42)

new_reduced_data = pd.concat([final_yes, final_no])

new_reduced_data = shuffle(new_reduced_data, random_state = 42)

new_reduced_data["Default"].value_counts()

Default
0    25000
1    25000
Name: count, dtype: int64
```

## Data Preprocessing

```
balanced_dataset.isnull().sum()
```

Age	1
Income	0
LoanAmount	0
CreditScore	0
InterestRate	0
LoanTerm	0
Education	0
EmploymentType	0
MaritalStatus	0
LoanPurpose	1
Default	0
dtype: int64	

**Figure 4.3.3 Data Preprocessing**

```
balanced_dataset.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Index: 50000 entries, 185261 to 154607
Data columns (total 11 columns):
#   Column                Non-Null Count  Dtype
---  -
0   Age                   49999 non-null  float64
1   Income                50000 non-null  int64
2   LoanAmount            50000 non-null  int64
3   CreditScore           50000 non-null  int64
4   InterestRate          50000 non-null  float64
5   LoanTerm              50000 non-null  int64
6   Education             50000 non-null  object
7   EmploymentType        50000 non-null  object
8   MaritalStatus         50000 non-null  object
9   LoanPurpose           49999 non-null  object
10  Default               50000 non-null  int64
dtypes: float64(2), int64(5), object(4)
memory usage: 4.6+ MB
```

```
independent = balanced_dataset.drop("Default", axis = 1)
dependent = balanced_dataset["Default"]
```

```
#Splitting into Numerical and Categorical Data
numerical_data = independent.select_dtypes(include = [np.number])
categorical_data = independent.select_dtypes(exclude = [np.number])
```

```
imputer = SimpleImputer(strategy = "mean")
```

```
imputer.fit(numerical_data[:, :])
```

▼ SimpleImputer ⓘ ?

SimpleImputer()

```
numerical_filled_data = pd.DataFrame(imputer.fit_transform(numerical_data),
                                     columns = numerical_data.columns)
```

```
numerical_filled_data.isnull().sum()
```

```
Age          0
Income       0
LoanAmount   0
CreditScore  0
InterestRate 0
LoanTerm     0
dtype: int64
```

**Figure 4.3.4 Simple Imputer**

```
categorical_imputer = SimpleImputer(strategy = "most_frequent")

categorical_filled_data = pd.DataFrame(categorical_imputer.fit_transform(categorical_data),
                                       columns = categorical_data.columns)

categorical_filled_data.isnull().sum()

Education      0
EmploymentType  0
MaritalStatus  0
LoanPurpose    0
dtype: int64
```

```
#Label Encoding
encoder = LabelEncoder()

for column in categorical_filled_data:
    preprocessed_dataset[column] = encoder.fit_transform(preprocessed_dataset[column])
```

**Figure 4.3.5 Label Encoder**

preprocessed_dataset											
	Age	Income	LoanAmount	CreditScore	InterestRate	LoanTerm	Education	EmploymentType	MaritalStatus	LoanPurpose	Default
0	57.0	109874.0	30973.0	567.0	13.95	60.0	2	1	0	0	0
1	69.0	34006.0	112148.0	597.0	10.60	24.0	2	1	1	3	1
2	20.0	36779.0	135741.0	352.0	23.82	36.0	0	0	0	0	1
3	41.0	50628.0	82083.0	828.0	11.32	48.0	1	2	0	0	1
4	28.0	114840.0	246968.0	415.0	22.65	24.0	1	2	0	3	0
...	...	...	...	...	...	...	...	...	...	...	...
49995	20.0	143934.0	105532.0	521.0	14.66	24.0	0	2	1	1	1
49996	47.0	139079.0	102610.0	669.0	17.28	24.0	3	3	0	0	0
49997	32.0	106367.0	85207.0	466.0	2.44	36.0	0	2	0	1	0
49998	29.0	94936.0	45806.0	313.0	24.81	48.0	3	0	2	3	1
49999	28.0	115166.0	211439.0	461.0	24.79	60.0	2	0	0	4	1

50000 rows × 11 columns



## Univariate Analysis

```
obj = Univariate()
tbl = obj.getUnivariateTbl(preprocessed_dataset, preprocessed_dataset.columns)
tbl
```

	Age	Income	LoanAmount	CreditScore	InterestRate	LoanTerm	Education	EmploymentType	MaritalStatus	LoanPurpose	Default
Mean	40.527231	77895.91892	135393.1022	567.69586	14.546836	36.06624	1.46762	1.5539	0.99076	1.98012	0.5
Median	39.0	76332.5	139267.0	565.0	15.055	36.0	1.0	2.0	1.0	2.0	0.5
Mode	22.0	15293.0	206276.0	513.0	21.93	60.0	1	3	0	1	0
Q1:25%	27.0	41732.0	75429.25	430.0	8.97	24.0	0.0	1.0	0.0	1.0	0.0
Q2:50%	39.0	76332.5	139267.0	565.0	15.055	36.0	1.0	2.0	1.0	2.0	0.5
Q3:75%	53.0	112750.25	197546.0	704.0	20.4	48.0	2.0	3.0	2.0	3.0	1.0
Q4:100%	69.0	149999.0	249993.0	849.0	25.0	60.0	3.0	3.0	2.0	4.0	1.0
IQR	26.0	71018.25	122116.75	274.0	11.43	24.0	2.0	2.0	2.0	2.0	1.0
1.5Rule	39.0	106527.375	183175.125	411.0	17.145	36.0	3.0	3.0	3.0	3.0	1.5
Lesser	-12.0	-64795.375	-107745.875	19.0	-8.175	-12.0	-3.0	-2.0	-3.0	-2.0	-1.5
Greater	92.0	219277.625	380721.125	1115.0	37.545	84.0	5.0	6.0	5.0	6.0	2.5
Minimum	18.0	15004.0	5000.0	300.0	2.0	12.0	0	0	0	0	0
Maximum	69.0	149999.0	249993.0	849.0	25.0	60.0	3	3	2	4	1
Skew	0.247032	0.108298	-0.133718	0.053522	-0.18543	0.000646	0.049313	-0.052592	0.017116	0.030304	0.0
Kurtosis	-1.128344	-1.238248	-1.183745	-1.19343	-1.16559	-1.298216	-1.34995	-1.349261	-1.524896	-1.298617	-2.00008
Variance	223.513284	1608035285.146601	5003041868.799876	25256.03944	43.866669	287.776728	1.242536	1.2404	0.678008	2.003885	0.250005
STD	14.950361	40100.315275	70732.184109	158.921488	6.623192	16.963983	1.114691	1.113732	0.823413	1.415586	0.500005

Figure 4.3.6 Univariate Analysis

```
#Finding Outliers
lesser_outlier = []
greater_outlier = []
for column in preprocessed_dataset.columns:

    if tbl[column]["Minimum"] < tbl[column]["Lesser"] :
        lesser_outlier.append(column)

    if tbl[column]["Maximum"] > tbl[column]["Greater"] :
        greater_outlier.append(column)

print(lesser_outlier)
print(greater_outlier)

[]
[]

#Removing the Outliers by replacing the lesser and greater values
for column in lesser_outlier:
    preprocessed_dataset_numerical[column]
    [preprocessed_dataset_numerical[column] < tbl[column]["Lesser"]] = tbl[column]["Lesser"]

for column in greater_outlier:
    preprocessed_dataset_numerical[column]
    [preprocessed_dataset_numerical[column] > tbl[column]["Greater"]] = tbl[column]["Greater"]
```

## Feature Selection

```
# K = 5
```

```
dataframe = selectKBest(5)
```

```
dataframe
```

```
Index(['Age', 'Income', 'LoanAmount', 'CreditScore', 'InterestRate'], dtype='object')
```

```
loading..
```

```
{'selector': SelectKBest(k=5, score_func=<function chi2 at 0x0000021309486980>), 'scaler': StandardScaler(),
```

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
--	----------	------	-------	-----	-------	----------	--------

ChiSquare	0.65	0.65	0.66	0.62	0.65	0.57	0.62
-----------	------	------	------	------	------	------	------

```
# K = 7
```

```
dataframe = selectKBest(7)
```

```
dataframe
```

```
Index(['Age', 'Income', 'LoanAmount', 'CreditScore', 'InterestRate',
```

```
      'Education', 'EmploymentType'],
```

```
      dtype='object')
```

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
--	----------	------	-------	-----	-------	----------	--------

ChiSquare	0.66	0.66	0.66	0.61	0.66	0.57	0.62
-----------	------	------	------	------	------	------	------

```
# K = 15
```

```
dataframe = selectKBest(15)
```

```
dataframe
```

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
--	----------	------	-------	-----	-------	----------	--------

ChiSquare	0.66	0.66	0.66	0.61	0.66	0.58	0.63
-----------	------	------	------	------	------	------	------

```
# K = 18
```

```
dataframe = selectKBest(18)
```

```
dataframe
```

	Logistic	SVMl	SVMnl	KNN	Navie	Decision	Random
--	----------	------	-------	-----	-------	----------	--------

ChiSquare	0.66	0.66	0.66	0.61	0.66	0.58	0.63
-----------	------	------	------	------	------	------	------

**Figure 4.3.7 Feature Selection**

## Select K Best

```
#Splitting into training and testing datasets
def Split_To_Training_Testing(features):
    x_train, x_test, y_train, y_test = train_test_split(features, dependent, test_size = 0.20, random_state = 0)
    sc = StandardScaler()
    x_train = sc.fit_transform(x_train)
    x_test = sc.transform(x_test)
    data_to_save['scaler'] = sc
    return x_train, x_test, y_train, y_test

#Select K Best Algorithm
def selectKBest(n):
    kbest = SelectKBest(score_func = chi2, k = n)
    data_to_save['selector'] = kbest
    kbestModel = kbest.fit(independent, dependent)
    features = kbest.transform(independent)
    print(independent.columns[kbest.get_support()])
    return build_model(features)

#Classification Models
def build_model(features):
    x_train, x_test, y_train, y_test = Split_To_Training_Testing(features)
    logistic_regression(x_train, x_test, y_train, y_test)
    svm_linear(x_train, x_test, y_train, y_test)
    svm_non_linear(x_train, x_test, y_train, y_test)
    knn_regression(x_train, x_test, y_train, y_test)
    decisionTree(x_train, x_test, y_train, y_test)
    naive_bayes(x_train, x_test, y_train, y_test)
    random_forest(x_train, x_test, y_train, y_test)
    dataframe1 = generateTbl()
    print("loading..")
    print(data_to_save)
    with open('model_data.pkl', 'wb') as f:
        pickle.dump(data_to_save, f)
    return dataframe1
```

```
def logistic_regression(x_train, x_test, y_train, y_test):
    global lg
    lg.clear()
    lg_regression = LogisticRegression(random_state = 42)
    lg_regression.fit(x_train, y_train)
    y_pred = lg_regression.predict(x_test)
    lg_cm = confusion_matrix(y_test, y_pred)
    lg_accuracy = accuracy_score(y_test, y_pred)
    lg_classification = classification_report(y_test, y_pred)
    lg.append(lg_accuracy)

def svm_linear(x_train, x_test, y_train, y_test):
    global lg
    svm.clear()
    svm_regression = SVC(kernel = 'linear', random_state = 0)
    svm_regression.fit(x_train, y_train)
    y_pred = svm_regression.predict(x_test)
    svm_cm = confusion_matrix(y_test, y_pred)
    svm_accuracy = accuracy_score(y_test, y_pred)
    svm_classification = classification_report(y_test, y_pred)
    svm.append(svm_accuracy)
```

## Classification Algorithm

```
def svm_linear(x_train, x_test, y_train, y_test):
    global lg
    svm.clear()
    svm_regression = SVC(kernel = 'linear', random_state = 0)
    svm_regression.fit(x_train,y_train)
    y_pred = svm_regression.predict(x_test)
    svm_cm = confusion_matrix(y_test, y_pred)
    svm_accuracy = accuracy_score(y_test, y_pred )
    svm_classification = classification_report(y_test, y_pred)
    svm.append(svm_accuracy)

def svm_non_linear(x_train, x_test, y_train, y_test):
    global svm_nl
    svm_nl.clear()
    svmnl_regression = SVC(kernel = 'rbf', random_state = 0)
    svmnl_regression.fit(x_train,y_train)
    data_to_save['model'] = svmnl_regression
    y_pred = svmnl_regression.predict(x_test)
    svmnl_cm = confusion_matrix(y_test, y_pred)
    svmnl_accuracy = accuracy_score(y_test, y_pred )
    svmnl_classification = classification_report(y_test, y_pred)
    svm_nl.append(svmnl_accuracy)

def knn_regression(x_train, x_test, y_train, y_test):
    global knn
    knn.clear()
    knn_regression = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
    knn_regression.fit(x_train,y_train)
    y_pred = knn_regression.predict(x_test)
    knn_cm = confusion_matrix(y_test, y_pred)
    knn_accuracy = accuracy_score(y_test, y_pred )
    knn_classification = classification_report(y_test, y_pred)
    knn.append(knn_accuracy)

def decisionTree(x_train, x_test, y_train, y_test):
    global dt
    dt.clear()
    des_regression = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
    des_regression.fit(x_train,y_train)
    y_pred = des_regression.predict(x_test)
    des_cm = confusion_matrix(y_test, y_pred)
    des_accuracy = accuracy_score(y_test, y_pred )
    des_classification = classification_report(y_test, y_pred)
    dt.append(des_accuracy)


def naive_bay(x_train, x_test, y_train, y_test):
    global nb
    nb.clear()
    nav_regression = GaussianNB()
    nav_regression.fit(x_train,y_train)
    y_pred = nav_regression.predict(x_test)
    nav_cm = confusion_matrix(y_test, y_pred)
    nav_accuracy = accuracy_score(y_test, y_pred )
    nav_classification = classification_report(y_test, y_pred)
```

## Saving the Model

```
with open('model_data.pkl', 'wb') as f:
    pickle.dump(data_to_save, f)
return dataframe1
```

## Deployment (Django)

```
from django.shortcuts import render
import numpy as np
import pickle
import os

1 usage
def results(request):
     result = None
    form_data = {}

    if request.method == 'POST':
        # Get values from POST
        gender = request.POST.get('gender')
        age = float(request.POST.get('age'))
        income = float(request.POST.get('income'))
        loan_amount = float(request.POST.get('loan_amount'))
        credit_score = float(request.POST.get('credit_score'))
        interest = float(request.POST.get('Interest'))

        form_data = {
            'gender': gender,
            'age': int(age),
            'income': int(income),
            'loan_amount': int(loan_amount),
            'credit_score': int(credit_score),
            'interest': interest,
        }
```

```

# Load model once (global, for performance)
MODEL_PATH = os.path.join(os.path.dirname(__file__), 'model_data.pkl')
with open(MODEL_PATH, 'rb') as f:
    model_data = pickle.load(f)

model = model_data['model']
scaler = model_data['scaler']

new_data = np.array([[age, income, loan_amount, credit_score, interest]])

# X_new_input = selector.transform(new_data)
x_new_scaled = scaler.transform(new_data)
pred = model.predict(x_new_scaled)

result = "You are Eligible to take this Loan Amount" if pred[0] == 1\
        else "You are Not Eligible to take this Loan Amount"
print(result)
return render(request, template_name: 'predict.html',
               context: {'result': result, 'form_data': form_data})

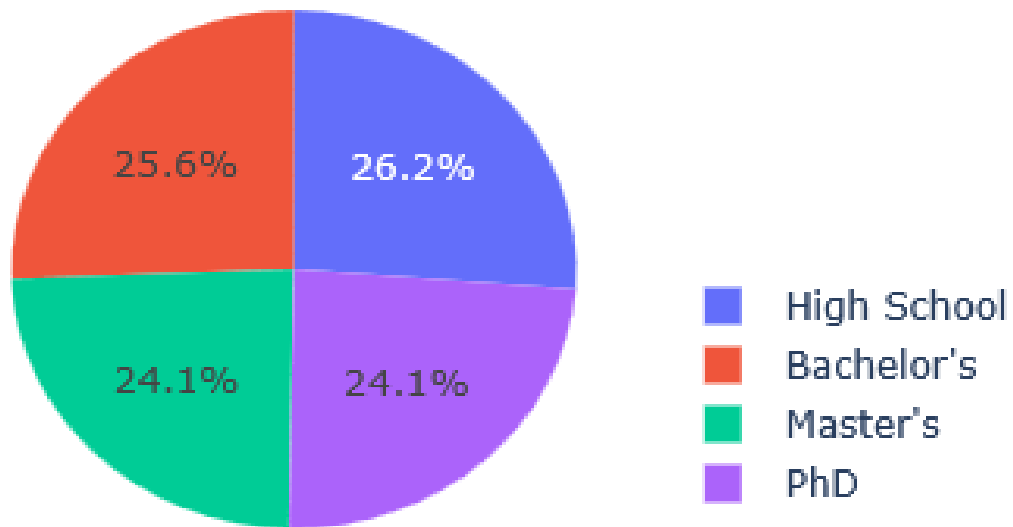
```

```

def home(request):
    return render(request, template_name: "home.html")

2 usages (1 dynamic)
def predict(request):
    return render(request, template_name: "predict.html")

```



**Figure 4.3.8 Applicant Education Background**

Implement intelligent algorithms to assess loan eligibility accurately by analyzing applicant data, predicting risk, and enhancing decision-making processes for financial institutions.

By leveraging machine learning techniques, this project evaluates a wide range of variables such as income, credit history, employment status, loan amount, and existing liabilities to forecast the likelihood of loan repayment. These algorithms help minimize loan defaults, streamline approval processes, and ensure fair and unbiased lending decisions.

Loan eligibility forecasting is the process of determining whether a loan applicant meets the required criteria for loan approval, based on both historical data and real-time analysis. Algorithms such as classification models (e.g., Logistic Regression, Decision Trees, Random Forest, XGBoost) and ensemble methods are used to predict eligibility outcomes. These models analyze applicant profiles and assign a probability of approval or rejection based on risk levels.

AI-driven loan forecasting improves financial efficiency by reducing manual workload, increasing approval speed, and ensuring consistency in decisions. It also helps financial institutions identify high-risk applicants early, reducing the risk of defaults and non-performing assets. The benefits of using AI in loan eligibility forecasting include:

- Improved credit risk assessment
- Faster and more efficient loan processing

- Fair and transparent decision-making
- Reduced default rates
- Enhanced customer experience
- Data-driven strategic planning

The integration of machine learning into financial services marks a significant step forward in achieving accuracy, scalability, and fairness in loan processing. This project is a transformative initiative to modernize traditional credit evaluation methods using predictive analytics and intelligent automation.

Develop intelligent forecasting solutions to streamline loan approval processes, reduce credit risk, and enhance financial decision-making accuracy. By utilizing a variety of applicant inputs such as income, employment status, credit history, debt-to-income ratio, and loan amount, machine learning algorithms can accurately predict an individual's loan eligibility.

Financial institutions like banks and lending platforms use these forecasting models to make fast, data-driven decisions that improve efficiency and reduce the likelihood of default. For example, many fintech companies leverage advanced credit scoring models that go beyond traditional credit checks, incorporating behavioral and transactional data to assess borrower risk.

1. **Collect and preprocess applicant data.** Begin by gathering historical loan data including both approved and rejected applications. Clean, normalize, and encode this data to ensure model-readiness.
2. **Train and evaluate predictive models.** Apply classification algorithms such as Logistic Regression, Decision Trees, Random Forest, Support Vector Machines, or Gradient Boosting models. Evaluate model performance using accuracy, precision, recall, and AUC-ROC scores to select the best-fit model.
3. **Deploy the model for real-time eligibility prediction.** Integrate the trained model into a web or enterprise application where it can receive real-time applicant data and output predictions about loan eligibility instantly.

Loan eligibility forecasting models serve as decision-support systems for lenders, helping to streamline the loan process from application to approval. These systems reduce manual errors, speed up processing time, and ensure fair decision-making based on data.



Loan forecasting algorithms are designed to solve the binary classification problem of predicting whether an applicant is **eligible** or **ineligible** for a loan. These algorithms are trained on labeled datasets where the eligibility status is known and then used to make predictions on new, unseen applicants.

The objective of loan eligibility forecasting is to enhance the lending process by improving risk assessment, increasing approval rates for qualified applicants, and reducing non-performing loans.

Implementing predictive models for loan eligibility y can lead to:

- **Reduced processing time and manual workload**
- **Improved loan portfolio performance**
- **Lower default and delinquency rates**
- **Greater financial inclusion by identifying creditworthy applicants previously overlooked**
- **Enhanced customer experience with quicker decisions**

Machine learning-based loan eligibility forecasting empowers institutions to make smarter lending decisions that balance profitability, risk, and customer satisfaction. This project is a step toward building a more intelligent, inclusive, and efficient lending ecosystem.

## **CHAPTER 5 SYSTEM TESTING**

### **5.1 GENERAL**

In general, after building up an logic for solving an problem in real-world, that logic or algorithm should be tested with real-world inputs and it should handle all possible exemptions that can be expected as an input from the user. If the program go tan exceptional input which is not defined in the program, the program should not terminate abruptly. It must handle the exemptions without getting terminated.

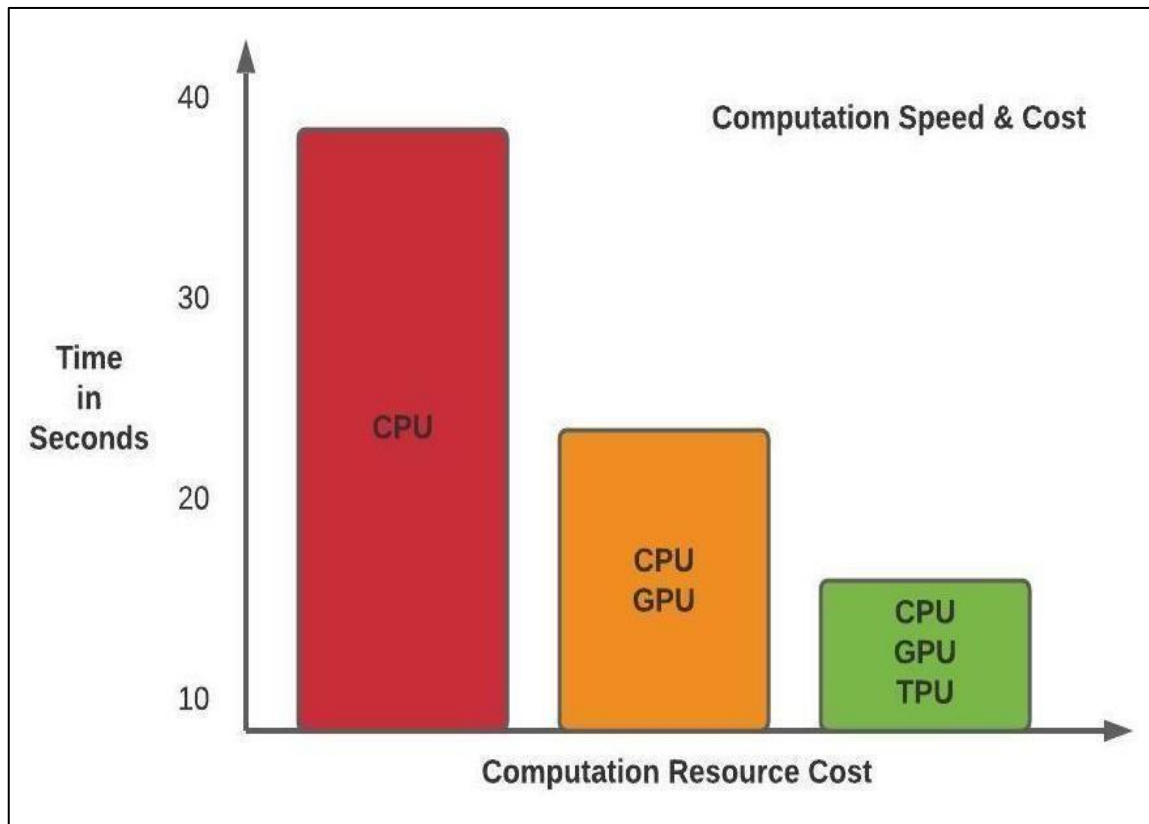
### **5.2 TEST CASES**

Test cases is nothing but an specialized input or an exceptional input to check whether the program run under all possible conditions without terminating itself. This ensures the reliability of the program. We used manual test cases to evaluate the system. Automated test cases are not used.

In this chapter, all the predicted possible exceptional input which is possible by the user to give as an input is taken under special care and the reflexes are programmed in a specialized way to handle those error input. So, the program will not terminate abruptly when it got an un-expected input from the user, either client or merchant.

### **5.3 PERFORMANCE MEASURES**

Machine Learning algorithms run faster in the GPU than the CPU. Over that, it was an image related task. Basically, the rendering of things will be faster in GUP. So, this module consumes around 30 to 40 seconds for the one complete execution in CPU (Central Processing Unit). Figure 5.1 explains both the computational speed and the computational cost required for the implementation for this project. The below displayed graph clearly explains the relativity between cost and speed.



**Figure 5.1 Computation Speed and Cost Graph**

#### **5.4 SUMMARY**

These are all the possible exemptions predicted and all those exemptions are handled in a smart way to avoid the termination of the Automated System algorithm abruptly. This ensure the quality and reliability of this project, for all exceptions, solutions were also provided. To improve the execution speed, we need to include more CPU, GPU and TPU.

## **CHAPTER 6 CONCLUSION AND FUTURE WORK**

### **6.1 CONCLUSION**

The implementation of machine learning and data analysis in loan eligibility forecasting marks a transformative step in the modernization of the lending process. This project demonstrates how predictive modeling can significantly enhance the accuracy, speed, and fairness of loan eligibility assessments, enabling financial institutions to make more informed decisions with greater confidence.

By utilizing historical data and advanced classification algorithms, the system effectively evaluates applicant profiles, minimizes credit risk, and streamlines the loan approval workflow. This leads to improved efficiency, reduced human error, and better resource allocation—all critical components in today's fast-paced financial landscape.

Moreover, the adaptability of machine learning allows the forecasting models to evolve over time by learning from new data and adjusting to changing market dynamics. This continuous improvement ensures the relevance and reliability of the predictions, helping institutions remain agile and resilient in a competitive environment.

As the financial sector continues to embrace digital transformation, loan eligibility forecasting systems powered by AI not only enhance operational performance but also improve customer satisfaction by offering quicker and more transparent decisions. The adoption of such intelligent systems positions financial institutions for sustainable growth, stronger risk management, and a more inclusive approach to lending. Ultimately, this project underscores the value of data-driven solutions in shaping the future of finance delivering smarter services, reducing risks, and fostering trust in an increasingly data-centric world.

### **6.2 FUTURE WORK**

Looking ahead, the future of loan eligibility forecasting using data analysis and machine learning holds immense potential for innovation and refinement. As technology evolves, we anticipate the integration of even more advanced algorithms capable of assessing a wider range of data points, leading to more precise and inclusive eligibility predictions.

Future developments may include the use of deep learning techniques and natural language

processing to analyze unstructured data such as customer feedback, social media interactions, and communication history, enriching the decision-making process. Additionally, the incorporation of alternative data sources—such as utility payments, rental history, and mobile financial transactions—will further enhance the ability to evaluate applicants with limited or non-traditional credit histories.

Another promising area is the use of real-time data streaming to provide instant eligibility assessments as applicants input their information. This will allow institutions to offer near-instant loan approvals while maintaining accuracy and minimizing risk. Furthermore, the use of explainable AI (XAI) will become increasingly important, ensuring that predictions made by complex models remain transparent, fair, and compliant with evolving regulatory standards.

In the future, loan eligibility forecasting systems will not only serve as powerful decision-support tools but also as adaptive learning platforms that evolve with changing economic indicators and consumer behaviour. By embracing continuous learning and automation, financial institutions can offer more personalized, equitable, and efficient lending experiences, extending credit access to a broader segment of the population. Ultimately, future enhancements in this field will drive a new era of intelligent, inclusive, and customer-centric financial services—where data is not just a tool, but a foundation for innovation and trust.

## **REFERENCES:**

- 1) Brown, I., & Mues, C. (2012). An experimental comparison of classification algorithms for imbalanced credit scoring data sets. *Expert Systems with Applications*, 39(3), 3446-3453.
- 2) Hand, D. J., & Henley, W. E. (1997). Statistical classification methods in consumer credit scoring: a review. *Journal of the Royal Statistical Society: Series A (Statistics in Society)*, 160(3), 523–541.
- 3) Lessmann, S., Baesens, B., Seow, H.-V., & Thomas, L. C. (2015). Benchmarking state-of-the-art classification algorithms for credit scoring: An update of research. *European Journal of Operational Research*, 247(1), 124–136.
- 4) Khandani, A. E., Kim, A. J., & Lo, A. W. (2010). Consumer credit-risk models via machine-learning algorithms. *Journal of Banking & Finance*, 34(11), 2767–2787.
- 5) Thomas, L. C., Edelman, D. B., & Crook, J. N. (2002). *Credit Scoring and Its Applications*. SIAM.
- 6) Zhou, L., & Wang, L. (2012). Credit scoring with a data mining approach based on support vector machines. *Expert Systems with Applications*, 39(3), 3441–3445.
- 7) Baesens, B., Setiono, R., Mues, C., & Vanthienen, J. (2003). Using neural network rule

- extraction and decision tables for credit-risk evaluation. *Management Science*, 49(3), 312–329.
- 8) Martens, D., Baesens, B., Van Gestel, T., & Vanthienen, J. (2007). Comprehensible credit scoring models using rule extraction from support vector machines. *European Journal of Operational Research*, 183(3), 1466–1476.
  - 9) Finlay, S. (2011). *Credit Scoring, Response Modeling, and Insurance Rating: A Practical Guide to Forecasting Consumer Behavior*. Palgrave Macmillan.
  - 10) Abdou, H. A., & Pointon, J. (2011). Credit scoring, statistical techniques and evaluation criteria: A review of the literature. *Intelligent Systems in Accounting, Finance and Management*, 18(2-3), 59–88.

## **APPENDIX (SAMPLE SOURCE CODE)**

```
import warnings
import numpy as np
import pandas as pd
import pickle
from utility import Univariate
from sklearn.utils import shuffle
from sklearn.impute import SimpleImputer
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from sklearn.feature_selection import chi2
from sklearn.feature_selection import SelectKBest
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import confusion_matrix
from sklearn.metrics import accuracy_score
from sklearn.metrics import classification_report
from sklearn.neighbors import KNeighborsClassifier
from sklearn.tree import DecisionTreeClassifier
from sklearn.ensemble import RandomForestClassifier
from sklearn.naive_bayes import GaussianNB
from sklearn.svm import SVC
from sklearn.preprocessing import LabelEncoder

#Splitting into training and testing datasets
def Split_To_Training_Testing(features):
    x_train, x_test, y_train, y_test = train_test_split(features, dependent, test_size = 0.20, random_state =
0)
    sc = StandardScaler()
```

```

x_train = sc.fit_transform(x_train)
x_test = sc.transform(x_test)
data_to_save['scaler'] = sc
return x_train, x_test, y_train, y_test

```

#Select K Best Algorithm

```

def selectKBest(n):
    kbest = SelectKBest(score_func = chi2, k = n)
    data_to_save['selector'] = kbest
    kbestModel = kbest.fit(independent,dependent)
    features = kbest.transform(independent)
    print(independent.columns[kbest.get_support()])
    return build_model(features)

```

#Classificaiton Models

```

def build_model(features):
    x_train, x_test, y_train, y_test = Split_To_Training_Testing(features)
    logistic_regression(x_train, x_test, y_train, y_test)
    svm_linear(x_train, x_test, y_train, y_test)
    svm_non_linear(x_train, x_test, y_train, y_test)
    knn_regression(x_train, x_test, y_train, y_test)
    decisionTree(x_train, x_test, y_train, y_test)
    naive_bayes(x_train, x_test, y_train, y_test)
    random_forest(x_train, x_test, y_train, y_test)
    dataframe1 = generateTbl()
    print("loading..")
    print(data_to_save)
    with open('model_data.pkl', 'wb') as f:
        pickle.dump(data_to_save, f)
    return dataframe1

```

```

def logistic_regression(x_train, x_test, y_train, y_test):
    global lg
    lg.clear()
    lg_regression = LogisticRegression(random_state = 42)
    lg_regression.fit(x_train,y_train)
    y_pred = lg_regression.predict(x_test)
    lg_cm = confusion_matrix(y_test, y_pred)
    lg_accuracy = accuracy_score(y_test, y_pred )
    lg_classification = classification_report(y_test, y_pred)
    lg.append(lg_accuracy)

```

```

def svm_linear(x_train, x_test, y_train, y_test):
    global lg
    svm.clear()
    svm_regression = SVC(kernel = 'linear', random_state = 0)
    svm_regression.fit(x_train,y_train)
    y_pred = svm_regression.predict(x_test)
    svm_cm = confusion_matrix(y_test, y_pred)
    svm_accuracy = accuracy_score(y_test, y_pred )
    svm_classification = classification_report(y_test, y_pred)
    svm.append(svm_accuracy)

def svm_non_linear(x_train, x_test, y_train, y_test):
    global svm_nl
    svm_nl.clear()
    svmnl_regression = SVC(kernel = 'rbf', random_state = 0)
    svmnl_regression.fit(x_train,y_train)
    data_to_save['model'] = svmnl_regression
    y_pred = svmnl_regression.predict(x_test)
    svmnl_cm = confusion_matrix(y_test, y_pred)
    svmnl_accuracy = accuracy_score(y_test, y_pred )
    svmnl_classification = classification_report(y_test, y_pred)
    svm_nl.append(svmnl_accuracy)

def knn_regression(x_train, x_test, y_train, y_test):
    global knn
    knn.clear()
    knn_regression = KNeighborsClassifier(n_neighbors = 5, metric = 'minkowski', p = 2)
    knn_regression.fit(x_train,y_train)
    y_pred = knn_regression.predict(x_test)
    knn_cm = confusion_matrix(y_test, y_pred)
    knn_accuracy = accuracy_score(y_test, y_pred )
    knn_classification = classification_report(y_test, y_pred)
    knn.append(knn_accuracy)

def decisionTree(x_train, x_test, y_train, y_test):
    global dt
    dt.clear()
    des_regression = DecisionTreeClassifier(criterion = 'entropy', random_state = 0)
    des_regression.fit(x_train,y_train)
    y_pred = des_regression.predict(x_test)

```



```

des_cm = confusion_matrix(y_test, y_pred)
des_accuracy = accuracy_score(y_test, y_pred )
des_classification = classification_report(y_test, y_pred)
dt.append(des_accuracy)

def naive_bay(x_train, x_test, y_train, y_test):
    global nb
    nb.clear()
    nav_regression = GaussianNB()
    nav_regression.fit(x_train,y_train)
    y_pred = nav_regression.predict(x_test)
    nav_cm = confusion_matrix(y_test, y_pred)
    nav_accuracy = accuracy_score(y_test, y_pred )
    nav_classification = classification_report(y_test, y_pred)
    nb.append(nav_accuracy)

def random_forest(x_train, x_test, y_train, y_test):
    global rf
    rf.clear()
    rf_regression = RandomForestClassifier(n_estimators = 10, criterion = 'entropy', random_state = 0)
    rf_regression.fit(x_train,y_train)
    y_pred = rf_regression.predict(x_test)
    rf_cm = confusion_matrix(y_test, y_pred)
    rf_accuracy = accuracy_score(y_test, y_pred )
    rf_classification = classification_report(y_test, y_pred)
    rf.append(rf_accuracy)

```

Output Screen shot:

## Check Your Loan Eligibility

Age

25

Monthly Income

10000

Loan Amount

100000

Cibil Score

650

Interest

10.0

Check Eligibility

Reset

You are Eligible to take this Loan Amount

Figure 6.1 Output for Loan Eligible

## Check Your Loan Eligibility

Age

45

Monthly Income

50000

Loan Amount

100000

Cibil Score

350

Interest

10.0

Check Eligibility

Reset

You are Not Eligible to take this Loan Amount

Figure 6.1 Output for Loan Not Eligible