

# 1. INTRODUCTION

## 1.1. ABSTRACT

A **Bank Loan Eligibility Prediction** project aims to automate the loan approval process by analyzing applicant data using machine learning.

In today's fast-paced financial industry, banks and lending institutions require efficient and accurate methods to assess loan eligibility. Traditional loan approval processes involve manual verification of an applicant's income, credit history, and other financial factors, which can be time-consuming and prone to human error. To streamline this process, a Bank Loan Eligibility Prediction Web Application can provide a quick and automated solution for evaluating loan applications.

This web application leverages machine learning algorithms to analyze applicant data, including income, employment status, credit score, and existing debts, to determine the likelihood of loan approval. By integrating predictive modeling with a user-friendly web interface, the application allows applicants to check their eligibility in real time while assisting financial institutions in making data-driven lending decisions.

The system aims to enhance efficiency, reduce processing time, and ensure a transparent and fair evaluation process. This project will focus on developing a responsive and interactive web application that provides instant loan eligibility predictions, benefiting both lenders and borrowers alike.

## 2. SYSTEM STUDY

A Bank Loan Eligibility Prediction Using Machine Learning as Web Application aims to automate the loan approval process by analyzing applicant data using machine learning. This system study explores the existing challenges, proposed solutions, and technical feasibility of the project.

### 2.1 EXISTING SYSTEM

The traditional loan approval process used by banks and financial institutions involves a manual evaluation of an applicant's financial credentials to determine their eligibility for a loan. This process is typically carried out by loan officers who assess various factors such as income level, employment status, credit history, outstanding debts, and repayment capacity before making a decision. While this method has been in place for years, it comes with several limitations and inefficiencies.

#### 2.1.1. Manual Verification Process

The current system relies on human verification of documents, including:

- **Income proof** (salary slips, tax returns, business income statements)
- **Employment details** (employment certificate, job stability, industry risk)
- **Credit history** (credit score reports from agencies like CIBIL, Experian, Equifax)
- **Collateral assessment** (for secured loans like home loans or auto loans)

This process is time-consuming and requires significant effort from both the applicant and the bank.

### **2.1.2. Challenges in the Existing System**

#### **a) Time-Consuming Process**

- The traditional method involves multiple rounds of document submission, verification, and evaluation, which may take several days or even weeks.
- Any discrepancies or missing documents can further delay the process.

#### **b) Human Errors and Subjectivity**

- Loan officers may have biases or inconsistencies in their decisions.
- The same applicant might be evaluated differently by different officers, leading to inconsistent loan approvals.
- Human errors in data entry or verification can lead to incorrect rejections or approvals.

#### **c) High Operational Costs**

- Banks need to maintain a large workforce to handle document processing and verification.
- Additional resources are required for fraud detection, customer verification, and manual credit scoring.
- Paper-based documentation leads to higher administrative costs.

#### **d) Lack of Real-Time Decision Making**

- Customers must wait for several days to receive loan approval or rejection.
- In emergencies, such as medical expenses or urgent business funding, delayed approvals can cause inconvenience.
- Banks lose potential customers due to long processing times.

#### **e) Limited Access to Data Analytics**

- The existing system does not leverage predictive analytics or machine learning to enhance decision-making.

- Loan approvals are based on predefined rules, rather than dynamic data-driven insights.
- Traditional methods fail to detect hidden patterns of risk or fraudulent applications.

#### **f) Inflexibility in Changing Market Conditions**

- Manual loan approval processes are slow in adapting to economic changes, such as interest rate fluctuations, inflation, or recession.
- New borrowers, such as gig workers and freelancers, may not fit traditional evaluation criteria, leading to rejections even if they have stable income streams.

## **2.2. PROPOSED SYSTEM**

The proposed system is a web-based application that leverages machine learning algorithms to automate and optimize the loan eligibility prediction process. Unlike the traditional manual method, this system uses data-driven analysis to evaluate an applicant's creditworthiness instantly. By integrating a user-friendly interface, real-time data processing, **and** automated decision-making, the proposed system enhances efficiency, reduces human errors, and improves the overall loan approval process.

### **2.2.1. Key Objectives of the Proposed System**

- Automate loan eligibility assessment to minimize manual efforts.
- Improve accuracy and consistency in loan approval decisions.
- Reduce loan processing time for both applicants and banks.
- Provide real-time eligibility predictions based on financial and credit data.
- Enhance fraud detection through machine learning models.
- Offer a user-friendly interface for applicants to check their loan eligibility online.
- Ensure secure handling of sensitive financial data through encryption and authentication.

## **2.2.2 Features of the Proposed System**

### **a) Automated Loan Eligibility Prediction**

The system will use machine learning models to predict loan approval based on:

- Applicant's income level
- Credit history & score
- Employment type & stability
- Loan amount requested
- Other financial parameters

The algorithm will analyze these factors and generate an instant eligibility score, helping applicants understand their chances of approval.

### **b) User-Friendly Web Application**

- The system will have an interactive and responsive UI.
- The application will display real-time results on eligibility status.
- Applicants will receive insights on how to improve their eligibility if they do not qualify.

### **c) Machine Learning-Based Decision Making**

- The system will use supervised learning algorithm in XGBoost Classifier
- These models will be trained on historical loan data to improve prediction accuracy.

### **d) Real-Time Processing & Fast Results**

- Unlike traditional manual systems, this application will process user inputs in real-time.
- It will instantly display loan eligibility results to the user, reducing waiting time.
- Loan officers can make quicker lending decisions, improving efficiency.

### **3. SYSTEM REQUIREMENTS**

The system requirement deals with hardware requirement and software requirement to install and run this web application.

#### **3.1 HARDWARE REQUIREMENTS**

Processor	: Intel(R) Core(TM) i3-2120 CPU@3.30GHz
RAM	: 4GB
Hard disk	: 500GB
Monitor	: HP

#### **3.2 SOFTWARE REQUIREMENTS**

Operating System	: Windows10 (64 bit)
Frontend	: Python
Backend	: CSV file
Libraries	: Flask, Scikit-learn, pickle
IDE	: VS code / Jupyter Notebook
Documentation & Presentation	: Microsoft word 2007

## 4. SOFTWARE DESCRIPTION

The Loan Approval Eligibility Prediction System is a machine learning-based web application designed to predict whether a loan should be approved based on user input or uploaded data. This project using various software technology.

### Development Environment

- **Model Development:** JupyterLab using Python
- **Web Application:** Visual Studio Code (VS Code)

### 4.1. JupyterLab

JupyterLab is a next-generation web-based user interface. It provides a robust platform for interactive computing that supports notebooks, code, and data. Designed as an extension of the Jupyter Notebook environment, JupyterLab offers a more flexible and powerful interface that allows users to work with multiple documents and activities side by side.

JupyterLab is particularly popular among data scientists, machine learning engineers, and researchers for its ability to combine code execution, rich text, and visualizations in one place. This integrated workflow simplifies data analysis, model development, and documentation.

#### 4.1.1. Key Features of JupyterLab

- **Multiple Document Interface:** JupyterLab allows users to open multiple notebooks, terminals, text editors, and data file viewers simultaneously in tabs or split screens.
- **Interactive Development:** Users can write and execute Python code, view the output, and visualize data using various plotting libraries.
- **Rich Text and Documentation:** With built-in support for Markdown and LaTeX, users can write comprehensive explanations alongside their code.
- **Support for Extensions:** JupyterLab supports a wide array of extensions that enhance productivity, such as Git integration, code formatters, and variable inspectors.

- **File Browser and Terminal Access:** JupyterLab includes a powerful file browser and terminal, allowing users to manage files and execute shell commands directly.

#### 4.1.2.. Advantages of Using JupyterLab

- **User-Friendly Interface:** Its graphical interface is intuitive and customizable, making it accessible to both beginners and experienced developers.
- **Real-Time Feedback:** Users can instantly see the results of their code, which accelerates the development and debugging process.
- **Cross-Language Support:** Though commonly used with Python, JupyterLab also supports R, Julia, and other languages via kernel support.
- **Collaboration and Sharing:** Notebooks created in JupyterLab can be shared as HTML or CSV documents or through version control systems like Git.

#### 4.1.3. Role in Machine Learning

JupyterLab has become a central tool in machine learning and data science due to its ability to handle the entire data pipeline in a single interface. From data ingestion and preprocessing to model training and evaluation, all tasks can be performed within a Jupyter notebook. Its visual feedback and support for inline charts make exploratory data analysis (EDA) intuitive and effective.

Machine learning libraries like Scikit-learn, TensorFlow, Keras, and XGBoost integrate seamlessly with JupyterLab. This allows users to train models, test predictions, and visualize results all in one workflow. Moreover, model performance metrics can be tracked and compared easily.

## 4.2. Visual Studio Code

Visual Studio Code, commonly referred to as VS Code, is a lightweight but powerful source code editor developed by Microsoft. It is available for Windows, macOS, and Linux and supports development in a wide array of programming languages such as Python, JavaScript, C++, Java, and more.



VS Code is free and open-source, making it accessible to developers worldwide. It is especially popular among web developers, software engineers, and data scientists due to its ease of use, performance, and vast extension ecosystem.

#### 4.2.1 Features of VS Code

- **Syntax Highlighting and IntelliSense:** VS Code offers intelligent code completion and real-time syntax suggestions, improving productivity and reducing errors.
- **Built-in Terminal:** A powerful terminal allows developers to run shell commands and scripts directly within the editor.
- **Extensions Marketplace:** Thousands of extensions are available to enhance functionality, including support for additional languages, debuggers, linters, themes, and more.
- **Source Control Integration:** Native Git integration lets developers commit, push, and pull code directly from the editor, streamlining version control workflows.
- **Debugger:** VS Code includes a robust debugger for inspecting and troubleshooting code execution across many languages.
- **Customization:** Users can tailor the interface, keyboard shortcuts, and settings to match their workflow preferences.

#### 4.2.2. VS Code in Web Development

VS Code is widely used for front-end and back-end web development. It supports HTML, CSS, JavaScript, TypeScript, and frameworks like React, Angular, and Vue.js out of the box. Features like Emmet for HTML autocompletion and live server extensions help in developing and previewing responsive web pages.

For back-end development, VS Code supports Node.js, Python, PHP, and more. Developers can set up servers, manage databases, and build APIs directly from the editor. Extensions such as Prettier, ESLint, and REST Client enhance the development experience.

#### 4.2.3. VS Code for Python Development

The Python extension in VS Code provides a complete environment for writing Python code. It includes support for:

- Running and debugging Python scripts
- Working with Jupyter Notebooks
- Environment and dependency management with virtualenv or conda
- Code linting and formatting
- Testing and profiling

VS Code makes it easy to switch between script files and notebooks, view variable states, and visualize data outputs. This flexibility makes it an excellent tool for both Python programming and data science.

## 4.3. Python

Python is a high-level, interpreted programming language known for its simplicity, readability, and versatility. Developed in the late 1980s by Guido van Rossum, Python has become one of the most widely used languages in various fields such as web development, data science, machine learning, automation, and more.

Its straightforward syntax and vast ecosystem of libraries make it accessible for beginners while remaining powerful enough for experts. Python supports multiple programming paradigms, including procedural, object-oriented, and functional programming.

### 4.3.1. Key Features of Python

- **Easy to Learn and Use:** Python's syntax is clean and easy to understand, making it an excellent choice for newcomers.
- **Extensive Standard Library:** Python comes with a rich set of libraries that support various tasks, from file handling to networking.
- **Large Community Support:** A vibrant community contributes to the continuous growth of libraries, tools, and learning resources.
- **Cross-Platform Compatibility:** Python runs on all major operating systems, including Windows, macOS, and Linux.
- **Integration Capabilities:** Python integrates well with other languages and tools, making it suitable for building hybrid applications.

### 4.3.2. Python for Machine Learning Model Development

Python is the go-to language for machine learning due to its wide range of libraries and frameworks. Developers and data scientists rely on Python tools to build, train, evaluate, and deploy models efficiently.

Popular Python libraries for machine learning include:

- **Scikit-learn:** Used for implementing basic ML algorithms like decision trees, SVMs, and regression.
- **Pandas and NumPy:** Essential for data manipulation and numerical operations.
- **XGBoost:** A powerful gradient boosting library known for its high performance..

Python also offers serialization tools such as pickle and joblib that allow trained models to be saved and reused in applications or services. It supports data visualization through libraries like Matplotlib and Seaborn, which are important for analyzing model performance.

### 4.3.3 Python for Web Application Development

Python is also widely used in web development, thanks to its powerful and flexible frameworks. It allows developers to build anything from simple websites to complex web services.

Some key Python frameworks for web development include:

- **Flask:** A lightweight framework perfect for building small to medium-sized web applications and APIs.
- **Django:** A high-level framework that follows the "batteries-included" philosophy, providing everything needed to build robust web applications.
- **FastAPI:** A modern, high-performance web framework for building APIs with Python based on standard type hints.

Python makes it easy to handle backend logic, connect to databases, manage user sessions, and integrate with front-end technologies. Web applications can be enhanced using templating engines such as Jinja2, which helps generate dynamic HTML content.

Python's support for RESTful API development also makes it ideal for creating services that connect front-end interfaces to back-end databases and machine learning models.

## 4.4. HTML

HTML, or HyperText Markup Language, is the standard markup language used to create and structure content on the web. It provides the basic building blocks of web pages by defining elements such as headings, paragraphs, links, images, tables, and forms. HTML is not a programming language; rather, it is a markup language that tells a web browser how to display content.

### 4.4.1. Advantages of HTML

- **Simplicity:** HTML is easy to learn and use, making it beginner-friendly.
- **Platform Independent:** Works across all platforms and browsers.
- **Free and Open:** HTML is not tied to any vendor or platform and is open for everyone to use.
- **Integrates Well:** HTML integrates seamlessly with CSS and JavaScript for enhanced styling and interactivity.
- **SEO-Friendly:** Proper use of HTML tags helps improve website visibility on search engines.
- **Multimedia Support:** HTML5 allows embedding of audio and video directly into web pages without plugins.

### 4.4.2. Features of HTML

The evolution of HTML continues to align with the growing demands of modern web applications. Key trends shaping the future include:

- **Progressive Web Apps (PWAs):** HTML5 is a foundation for PWAs, which offer app-like experiences directly in the browser.
- **Accessibility Improvements:** Ongoing development is enhancing support for assistive technologies.
- **Web Components:** HTML now supports reusable custom elements, making code modular and maintainable.

- **Native Support for APIs:** HTML5 works with modern web APIs for geolocation, offline storage, notifications, and more.
- **Integration with AI and IoT:** HTML will continue playing a role in interfaces for intelligent and interconnected devices.

## 4.5. CSS

CSS, or Cascading Style Sheets, is a style sheet language used to control the presentation and design of web pages written in HTML. While HTML provides the structure of the content, CSS defines how that content looks—such as its layout, colors, fonts, and spacing. CSS helps make websites more visually appealing and user-friendly.

### 4.5.1. Advantages of CSS

- **Separation of Concerns:** Keeps content (HTML) and design (CSS) separate for easier maintenance.
- **Reusability:** Styles can be reused across multiple pages using external stylesheets.
- **Consistency:** Ensures a uniform look and feel throughout the website.
- **Improved Load Times:** Cleaner code and caching of external stylesheets lead to faster performance.
- **Accessibility and Responsiveness:** Enables responsive design through media queries and flexible layouts.

### 4.5.2. The Features of CSS

CSS continues to evolve to meet modern web design requirements. The future includes:

- **CSS Grid and Flexbox:** Advanced layout models that simplify complex designs.
- **Custom Properties (CSS Variables):** Allow for more flexible and dynamic styling.
- **Subgrid and Container Queries:** Give finer control over nested and component-based layouts.
- **Integration with JavaScript Frameworks:** CSS is increasingly integrated with React, Vue, and Angular using CSS-in-JS libraries.
- **Enhanced Performance and Features:** Browsers continue to optimize CSS rendering and introduce new styling capabilities.

## 4.6. JavaScript

JavaScript is a high-level, interpreted programming language primarily used to create dynamic and interactive effects within web browsers. It allows developers to implement complex features on web pages, such as interactive forms, real-time updates, animations, and client-side validations. JavaScript runs in the browser without the need for server-side processing, making it essential for modern web development.

### 4.6.1. Core Features of JavaScript

- **Client-Side Scripting:** JavaScript runs in the user's browser, enabling interactive behavior on web pages.
- **Event Handling:** Detects and responds to user actions like clicks, hovers, and form submissions.
- **DOM Manipulation:** Allows developers to dynamically change the structure and content of HTML documents.
- **AJAX Requests:** Facilitates communication with the server without refreshing the page.
- **Compatibility:** Supported by all major browsers without any need for additional plugins.

### 4.6.2. Advantages of JavaScript

- **Interactivity:** Enhances user engagement by allowing real-time updates and dynamic content.
- **Speed:** JavaScript runs immediately in the browser, improving response times and performance.
- **Versatility:** Can be used on both the front-end (with frameworks like React or Vue) and the back-end (with Node.js).
- **Rich Ecosystem:** Offers a wide range of libraries and frameworks that speed up development.
- **Community Support:** Large and active community with extensive resources and tutorials.

## **5. PROBLEM DESCRIPTION**

In the modern banking sector, evaluating the eligibility of applicants for loans is a critical process. Traditionally, this evaluation has been manual, time-consuming, and susceptible to human biases or inconsistencies. With the increase in the number of applicants and the volume of data, it has become essential to automate the loan eligibility process using advanced technologies like machine learning.

The core problem is to accurately classify loan applications as either Approval or Rejected for approval based on various applicant attributes such as:

- Number of dependents
- Education level
- Employment status
- Annual income
- Loan amount
- Loan term
- CIBIL score
- Owned assets

### **5.1. About page**

The about page display the overview of the Loan Eligibility Prediction System, a smart and efficient solution designed to help banks, financial institutions, and individual users make informed decisions about loan approvals.

This application leverages the power of Machine Learning, built using Python and the XGBoost algorithm.

### **5.2. Individual Prediction in Loan Eligibility Systems**

#### **5.2.1. Overview**

Individual Prediction refers to the capability of a loan eligibility prediction system to assess a single applicant's data in real-time and determine whether they are eligible for a loan.

This feature is particularly useful for front-end banking staff, financial advisors, or directly for customers via a web interface. With the help of machine learning models, such predictions can be made instantly and with a high degree of accuracy.

### 5.2.2. Purpose and Importance

In traditional banking workflows, evaluating a loan applicant involves collecting various personal and financial details, and then manually analyzing the eligibility based on pre-defined rules. This process is often time-consuming and prone to errors or inconsistencies due to human intervention.

With Individual Prediction, the process becomes:

- **Faster** – Results are available in seconds.
- **More accurate** – Machine learning reduces subjectivity.
- **User-friendly** – Interactive forms and dashboards make it easy to use for non-technical users.

### 5.2.3. Input Features

The prediction model typically uses the following applicant information:

- **Number of Dependents:** Indicates financial responsibility.
- **Education Level:** Reflects the applicant's qualification, which can influence employability and earning potential.
- **Self-Employed Status:** Differentiates between salaried and self-employed individuals, which affects income stability.
- **Annual Income:** Determines the repayment capacity of the applicant.
- **Loan Amount Requested:** Represents the financial support needed.
- **Loan Duration:** Indicates the period over which the applicant plans to repay the loan. This directly affects monthly installment amounts and the total interest.
- **CIBIL Score:** A measure of the applicant's credit history and risk level.
- **Assets:** Indicates existing financial strength and loan security.



#### 5.2.4. Workflow

1. **User Input Interface:** A web form allows users to input values for each feature (e.g., dropdowns for education level, number inputs for income, loan amount, and loan duration).
2. **Data Preprocessing:** Before feeding data into the model, it is preprocessed. This may include scaling numerical data using techniques like StandardScaler.
3. **Model Prediction:** The preprocessed data is passed into the trained machine learning model (e.g., XGBoost), which returns a prediction – either Approval or Rejected.
4. **Output Display:** The result is displayed to the user in a clear and intuitive format, possibly with additional insights (e.g., "High probability of approval", "Improve credit score for better chances").

### 5.3. File Prediction (Bulk Loan Application Classification)

#### 5.3.1. Overview

File Prediction, also known as File Classification, is a key feature of modern loan eligibility systems that allows banks and financial institutions to process multiple loan applications at once. Instead of entering applicant details individually, users can upload a file—typically in xls or CSV format—containing the data of multiple applicants. The system then processes the entire dataset using a machine learning model and classifies each application as Approval or Rejected for a loan.

This approach is highly efficient for organizations that handle large volumes of loan applications, enabling them to automate and streamline their loan screening process.

#### 5.3.2. Purpose

In real-world banking operations, loan officers and processing teams often receive hundreds or even thousands of applications daily. Manually analyzing each application is not only time-consuming but also prone to human error.

Bulk file prediction solves this problem by:

- Reducing processing time from hours to minutes.

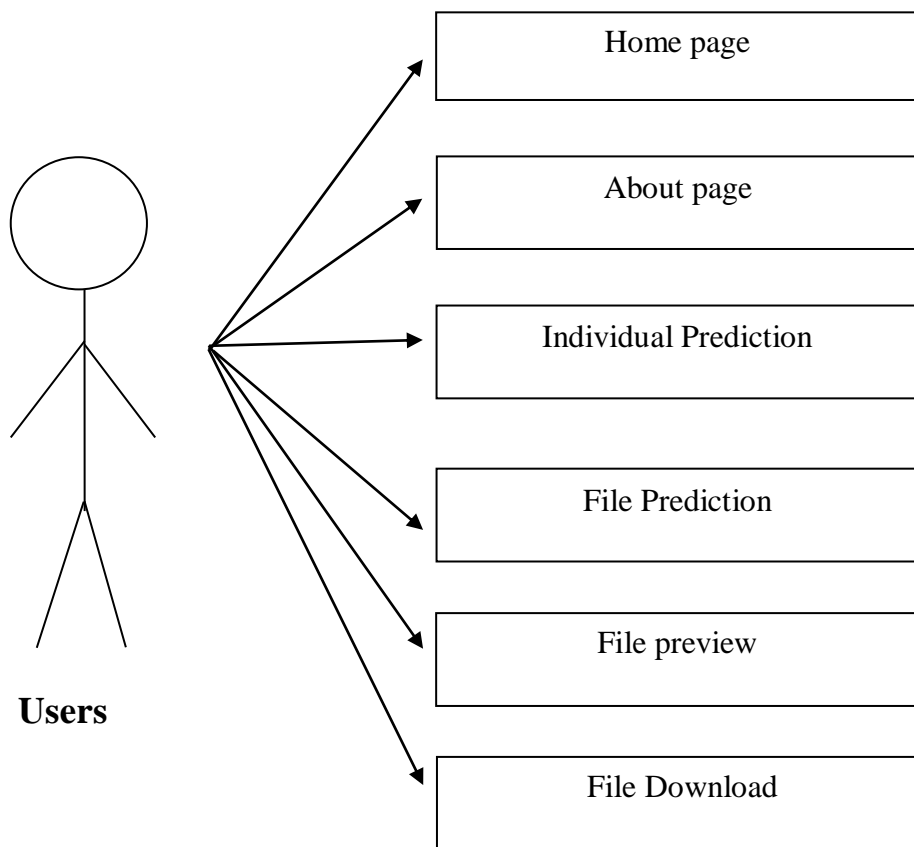
- Ensuring consistent and objective decision-making.
- Allowing easy file upload and download through a web-based interface.
- Improving operational efficiency within loan departments.

### 5.3.3. Processing Workflow

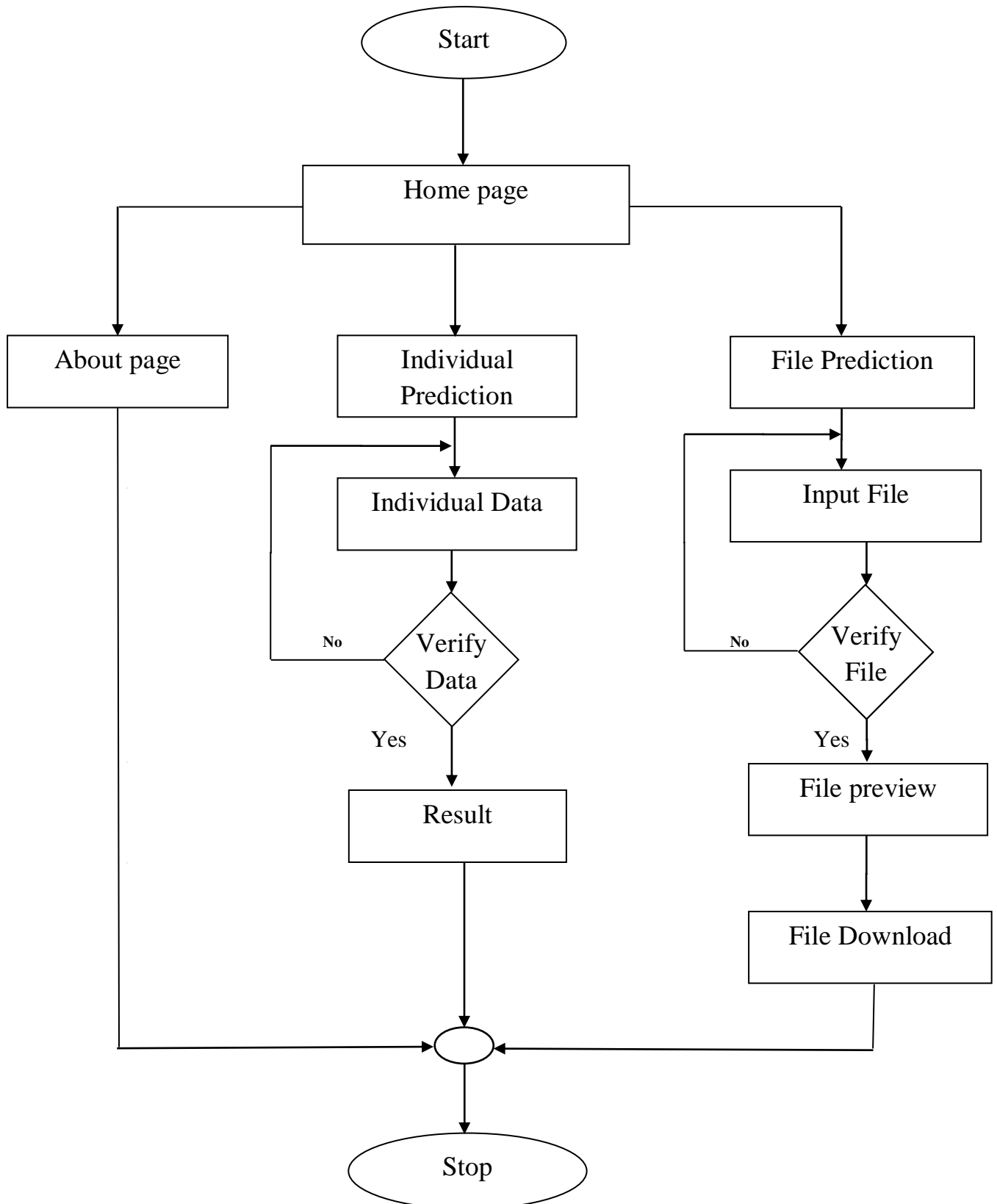
1. **File Upload:** Users upload a CSV file through the web application.
2. **Data Validation:** The system checks for missing values, incorrect formats, and ensures all required fields are present.
3. **Preprocessing :**
  - Numerical features (e.g., income, CIBIL score, loan amount) are scaled using tools like StandardScaler.
  - Data is prepared without encoding if the model was trained with raw or preprocessed categorical values.
4. **Model Prediction :**
  - The XGBoost model processes the dataset.
  - For each entry, the model returns a prediction: Approval or Rejected.
5. **Output Generation:**
  - A new CSV file is created with an additional column labeled prediction.
  - This file is displayed in a preview on the web interface.
6. **Download Option:** Users can download the classified file for record-keeping, reporting, or further analysis.

## 6. SYSTEM DESIGN

### 6.1 USE CASE DIAGRAM



## 6.2. ACTIVITY DIAGRAM



### 6.3. CSV File

Feature Name	Description	Data Type
No_of_Dependents	Number of dependents of the applicant	Integer
Education	Applicant's education level(Graduate=1/Not Graduate=0)	Binary
Self_Employed	Whether the applicant is self-employed	Binary
Income_per_Annum	Annual income of the applicant	Float
Loan_Amount	Requested loan amount	Float
Loan_Term	Loan repayment duration (in months)	Integer
CIBIL_Score	Credit score of the applicant	Integer
Assets_Value	Value of the applicant's assets	Float
Loan_Status	Target variable (1 = Approved, 0 = Rejected)	Binary

## **7. TESTING AND IMPLEMENTATION**

### **7.1. TESTING**

Testing is a critical phase in the development of any software application, ensuring that all features function correctly, the model behaves as expected, and the user interface provides a smooth and error-free experience. For the Loan Eligibility Prediction System, both functional testing and model performance evaluation were carried out to verify the system's accuracy and reliability.

#### **7.1.1. Functional Testing**

Functional testing focused on ensuring each component of the application works as intended. The main areas tested include:

##### **a. Individual Prediction**

- Form inputs accept valid user data such as income, loan amount, CIBIL score, etc.
- Predict button triggers model evaluation correctly.
- Results display eligibility status instantly.

##### **b. File Upload and Bulk Prediction**

- Only .csv files are accepted.
- File preview loads correctly showing data rows and columns.
- Prediction results are displayed with proper classification.
- Output file is generated with prediction column and available for download.
- Uploading incorrect file types or empty files triggers warnings.

##### **c. UI/UX Testing**

- All buttons, forms, and messages function across different browsers.
- Page responsiveness and layout tested on different screen sizes.
- Validation feedback is clear and user-friendly.

### 7.1.2. Model Testing and Evaluation

To ensure high-quality predictions, the XGBoost model was evaluated using various metrics:

#### a. Accuracy

The model achieved high accuracy during testing with the dataset, successfully predicting eligible and non-eligible applicants based on the input features.

	precision	recall	f1-score	support
0	0.98	0.98	0.98	315
1	0.99	0.99	0.99	539
accuracy			0.99	854
macro avg	0.99	0.99	0.99	854
weighted avg	0.99	0.99	0.99	854

#### b. Performance Testing

- Model loading time was optimized using Pickle serialization.
- Bulk file prediction was tested with increasing data sizes (up to 1000+ entries).
- File download time remained under a few seconds, even with large output files.

### 7.1.3. Unit testing

Unit testing is a level of software testing where individual or components of software are tested. The purpose is to validate that each unit of software platforms as designed. A unit is a smallest testable part of any software it usually has one or a few inputs and usually a single output.

### 7.1.4. Integration testing

Integration testing is a level of software testing where individual units are combined and tested as a group. The purpose of this level of testing is to expose faults in the interaction between integrated units. Test drivers and stubs are used to assist in integration testing. The

main objective of integrating is to reduce risks and finding defects of the software components.

### **7.1.5. System testing**

System testing is a level of testing that validates the complete and fully integrated software product. The purpose of system is to evaluate the end to end system specifications. Usually, the software is only one element large computer based system. System testing tests is not only the design, but also the behaviour and even they believed expectations of the customers.

### **7.1.5. Validation testing**

At the culmination of integration testing, software is completely assembled as a package, interfacing error have been uncovered and corrected and finals series of software tests begin-validation test begins. Validation testing can be defined in many ways.

## **7.2. IMPLEMENTATION PLAN**

The Loan Eligibility Prediction System was developed with a structured and phased implementation strategy. This plan outlines the key steps taken from requirement gathering through to deployment, ensuring a clear roadmap for both development and integration.

### **7.2.1. Requirement Analysis**

In this initial phase, the primary focus was to understand the business goals, technical needs, and functional requirements of the application.

#### **Objectives**

- Identify the target users (e.g., loan officers, banks, applicants).
- Define input features: no\_of\_dependent, education, self\_employed, annum\_income, loan\_amount, loan\_term, cibil\_score, assets.
- Specify output: Binary prediction of Approval or reject
- Determine project scope: Individual predictions and File-based bulk predictions.



### 7.2.2. Dataset Collection and Preparation

A synthetic or real-world dataset was gathered and preprocessed to make it suitable for model training.

#### Steps Involved

- Load dataset using Python (e.g., CSV format).
- Handle missing values and outliers.
- Normalize numerical features using StandardScaler.
- Split data into training and testing sets (typically 80/20).
- Ensure feature consistency across individual and file-based inputs.

### 7.2.3. Model Development in JupyterLab

The machine learning model was developed using XGBoost, chosen for its performance and accuracy in classification tasks.

#### Tasks Completed

- Model training using the processed dataset in JupyterLab.
- Model tuning with hyperparameter optimization.
- Model evaluation using accuracy, confusion matrix, and cross-validation.
- Save the final model using pickle for later use in the web application.

### 7.2.4. Web Application Design and Development

Using **HTML**, **CSS**, **JavaScript**, and **Python**, the user interface was built to support real-time interaction.

#### Key Functionalities

- Form for Individual Prediction with input validation.
- File upload system to preview and process batch applications.
- Display and download classified results in tabular format.
- Responsive and styled UI for better user experience.

### **7.2.5. Backend Integration**

The trained model was integrated into the backend using Python scripts.

#### **Actions Taken**

- Load pickled model and scaler during runtime.
- Process individual and batch inputs through the same pipeline.
- Generate and return prediction results in real time.
- Handle exceptions and input errors gracefully.

### **7.2.6. Testing and Validation**

Both unit and integration testing were performed to ensure model and web app reliability.

#### **Included**

- Testing for valid and invalid inputs.
- Stress testing with large input files.
- Model accuracy and performance checks.
- Functional and UI testing by sample users.

### **7.1.7. Deployment and Usage**

The application was finalized and prepared for real-world use or local deployment.

#### **Options Considered**

- Local deployment using Python Flask.
- Distribution via ZIP or GitHub for local execution in VS Code.
- Possibility for future deployment to cloud platforms (e.g., Heroku, Render).

## 8. SAMPLE SOURCE CODE

### Data Pre-processing And Machine learning

#### xgboost.ipynb

```
import pandas as pd
data1=pd.read_csv('loan_approval_dataset.csv')
data1.drop(columns=['loan_id'],inplace=True)
data1.columns=data1.columns.str.strip()
data1['Assets']= data1.residential_assets_value + data1.commercial_assets_value+
data1.luxury_assets_value + data1.bank_asset_value
data1.drop(columns
=['residential_assets_value','commercial_assets_value','luxury_assets_value','bank_asset_valu
e'],inplace=True)
data1.drop(data1[(data1['loan_status']=='
Approved')&(data1['cibil_score']<=600)].index,inplace=True)
data1.drop(data1[(data1['loan_status']=='
Approved')&(data1['loan_term']<=6)].index,inplace=True)
def clean_data(st):
st=st.strip()
return st

data1.education =data1.education.apply(clean_data)

data1.education = data1['education'].replace(['Graduate','Not Graduate'],[1,0])

data1.self_employed = data1.self_employed.apply(clean_data)

data1.self_employed = data1['self_employed'].replace(['No','Yes'],[0,1])

data1.loan_status = data1.loan_status.apply(clean_data)
```

```

data1.loan_status = data1.loan_status.replace(['Rejected','Approved'],[0,1])
print(data1.dtypes)
no_of_dependents    int64
education           int64
self_employed       int64
income_annum        int64
loan_amount         int64
loan_term           int64
cibil_score         int64
loan_status         int64
Assets              int64

from sklearn.model_selection import train_test_split
input_data1 = data1.drop(columns=['loan_status'])
output_data1 = data1['loan_status']
x_train,x_test,y_train,y_test=train_test_split(input_data1,output_data1,test_size=0.2)
x_train.shape,x_test.shape,y_train.shape,y_test.shape
((2451, 8), (613, 8), (2451,), (613,))

from sklearn.preprocessing import StandardScaler
scaler = StandardScaler()
x_train_scaled=scaler.fit_transform(x_train)
x_test_scaled=scaler.transform(x_test)

from xgboost import XGBClassifier
model = XGBClassifier()

model.fit(x_train_scaled,y_train)

```



XGBClassifier



```

XGBClassifier(base_score=None, booster=None, callbacks=None,
               colsample_bylevel=None, colsample_bynode=None,
               colsample_bytree=None, device=None, early_stopping_rounds=None,
               enable_categorical=False, eval_metric=None, feature_types=None,

```

```

        gamma=None, grow_policy=None, importance_type=None,
interaction_constraints=None, learning_rate=None, max_bin=None,
max_cat_threshold=None, max_cat_to_onehot=None,
max_delta_step=None, max_depth=None, max_leaves=None,
min_child_weight=None, missing=nan, monotone_constraints=None,
multi_strategy=None, n_estimators=None, n_jobs=None,
num_parallel_tree=None, random_state=None, ...)

```

```
accuracy =model.score(x_train_scaled,y_train)
```

```
print(f"Model Accuracy: {accuracy:.2f}")
```

```
Model Accuracy: 1.00
```

```
accuracy =model.score(x_test_scaled,y_test)
```

```
print(f"Model Accuracy: {accuracy:.2f}")
```

```
Model Accuracy: 0.99
```

```
fromsklearn.metricsimportclassification_report,confusion_matrix,ConfusionMatrixDisplay
```

```
Y_pred=model.predict(x_test_scaled)
```

```
report =classification_report(y_test,Y_pred,zero_division=0)
```

```
print(report)
```

```

precision  recall  f1-score  support

0         1.00    0.99    0.99    332
1         0.99    1.00    0.99    281

accuracy                0.99    613
macro avg    0.99    0.99    0.99    613
weighted avg    0.99    0.99    0.99    613

```

```
pred_data=
```

```
pd.DataFrame([[ '2','0','0','9600000','29900000','12','778','50700000']],columns=['no_of_dependents','education','self_employed','income_annum','loan_amount','loan_term','cibil_score','Assets'])
```

```
pred_data=scaler.transform(pred_data)
```

```
model.predict(pred_data)
```

```
array([1])
import pickle as pk
pk.dump(model, open('model.plk','wb'))
pk.dump(scaler, open('scaler.plk','wb'))
```

## Web Application

### App.py

```
from flask import Flask, render_template, request, send_file, jsonify

import pandas as pd

import pickle as pk

import os

# Initialize Flask app

app = Flask(__name__, template_folder="templates", static_folder="static")

# Define and create necessary folders

UPLOAD_FOLDER = 'uploads'

os.makedirs(UPLOAD_FOLDER, exist_ok=True)

app.config['UPLOAD_FOLDER'] = UPLOAD_FOLDER

PREDICTIONS_FOLDER = os.path.join(app.instance_path, 'predictions')

os.makedirs(PREDICTIONS_FOLDER, exist_ok=True)

app.config['PREDICTIONS_FOLDER'] = PREDICTIONS_FOLDER # Fix for KeyError

# Load trained model and scaler

try:

    model = pk.load(open("model.plk", "rb"))
```

```

    scaler = pk.load(open("scaler.pkl", "rb"))

except FileNotFoundError:

print("Error: Model or scaler file not found. Ensure 'model.pkl' and 'scaler.pkl' exist.")

    model, scaler = None, None

# Home Page

@app.route("/")

def home():

    return render_template("home.html")

@app.route("/about")

def about():

    return render_template("about.html")

# Single Loan Prediction Page

@app.route("/single", methods=["GET"])

def single():

    return render_template("single.html", form_data={ })

# Single Loan Prediction Processing

@app.route("/predict_single", methods=["GET"])

def predict_single():

form_data = request.args

    if "clear" in form_data:

        return render_template("single.html", result=None, suggestion=None, form_data={ })

    if not form_data:

```

```

        return render_template("single.html", result=None, form_data={ })

    try:

        # Extract input values

no_of_dep = int(form_data.get("no_of_dep", 0))

        education = form_data.get("education", "Graduated")

self_employed = form_data.get("self_employed", "No")

income_annum = float(form_data.get("income_annum", 0))

loan_amount = float(form_data.get("loan_amount", 0))

loan_term = int(form_data.get("loan_term", 0))

cibil_score = int(form_data.get("cibil_score", 0))

        assets = float(form_data.get("assets", 0))

        # Encode categorical values

education_encoded = 0 if education == "Graduated" else 1

self_employed_encoded = 0 if self_employed == "No" else 1

        # Prepare input data

pred_data = pd.DataFrame([[no_of_dep, education_encoded, self_employed_encoded,
income_annum,

loan_amount, loan_term, cibil_score, assets]],

                           columns=["no_of_dependents", "education", "self_employed",
"income_annum", "loan_amount", "loan_term", "cibil_score", "Assets"])

        # Scale input data

pred_data = scaler.transform(pred_data)

        # Make Prediction

```



```

prediction = model.predict(pred_data)

# Loan Suggestion if rejected

def suggest_loan(income_annum, cibil_score, loan_amount, assets):

    if cibil_score >= 750:

max_loan = income_annum * 1.5

    elif cibil_score >= 650:

max_loan = income_annum * 1.3

    elif cibil_score >= 550:

max_loan = income_annum * 1.2

    elif cibil_score >= 470:

max_loan = income_annum * 0.6

    else:

        return "Your CIBIL score is too low for loan approval."

    if max_loan >= assets:

        return "Your asset value is too low for the requested loan amount."

    elif max_loan < loan_amount:

        return f"Loan is rejected. You may qualify for a loan up to: {max_loan:.2f}"

    else:

        return "Try increasing cibil score, asset, and loan duration for better approval chances."

# Generate Result

if prediction[0] == 1:

    result = "✔️ Loan is Approved!"

```

```

        suggestion = None

    else:

        result = "✗Loan is Rejected!"

        suggestion = suggest_loan(income_annum, cibil_score, loan_amount, assets)

    except Exception as e:

        result = f"Error: {str(e)}"

        suggestion = None

    return render_template("single.html", result=result, suggestion=suggestion,
form_data=form_data)

# File-based Loan Prediction

@app.route("/index")

def index():

    return render_template("index.html")

# Process uploaded file

def process_file(file_path, file_ext, scale=False, for_preview=False):

    if file_ext == "csv":

        df = pd.read_csv(file_path)

    elif file_ext in ["xls", "xlsx"]:

        df = pd.read_excel(file_path)

    else:

        return None

required_columns = ["name", "no_of_dependents", "education", "self_employed",
"income_annum",

```

```

        "loan_amount", "loan_term", "cibil_score", "Assets"]

    if not all(col in df.columns for col in required_columns):

        return None

    df_original = df.copy()

    if for_preview:

        return df_original

    # Convert categorical data

    df["education"] = df["education"].map({"Graduated": 0, "Not Graduated": 1}).fillna(0)

    df["self_employed"] = df["self_employed"].map({"No": 0, "Yes": 1}).fillna(0)

    # Fill missing numerical values

    df = df.fillna(0)

    numerical_cols = ["no_of_dependents", "education", "self_employed", "income_annum",

        "loan_amount", "loan_term", "cibil_score", "Assets"]

    df_numeric = df[numerical_cols].copy()

    if scale:

        df_numeric = df_numeric.astype(float)

        df_numeric[numerical_cols] = scaler.transform(df_numeric[numerical_cols])

    df_numeric.insert(0, "name", df["name"])

    return df_numeric, df_original

# Predict loan status

def predict_loan_status(df_scaled, df_original):

```

```

try:

    predictions = model.predict(df_scaled.drop(columns=["name"]))

df_original["loan_status"] = ["Approved" if p == 1 else "Rejected" for p in predictions]

except Exception as e:

print("Prediction error:", e)

    return None

    return df_original

# File Preview

@app.route('/preview', methods=['POST'])

def preview():

    if 'file' not in request.files:

        return jsonify({"error": "No file part"})

    file = request.files['file']

    if file.filename == "":

        return jsonify({"error": "No selected file"})

    file_ext = file.filename.split('.')[-1].lower()

    if file_ext not in ["csv", "xls", "xlsx"]:

        return jsonify({"error": "Unsupported file format. Please upload a CSV or Excel file."})

    file_path = os.path.join(app.config['UPLOAD_FOLDER'], file.filename)

    file.save(file_path)

    original_df = process_file(file_path, file_ext, for_preview=True)

```

```

if original_df is None:

    return jsonify({"error": "Failed to process file. Ensure correct format and columns."})

return jsonify({"preview": original_df.to_dict(orient='records')})

# Predict loan status

@app.route('/predict', methods=['POST'])

def predict():

    if 'file' not in request.files:

        return jsonify({"error": "No file part"})

    file = request.files['file']

    if file.filename == "":

        return jsonify({"error": "No selected file"})

    file_ext = file.filename.split('.')[-1].lower()

    if file_ext not in ["csv", "xls", "xlsx"]:

        return jsonify({"error": "Unsupported file format."})

    file_path = os.path.join(app.config['UPLOAD_FOLDER'], file.filename)

    file.save(file_path)

    df_scaled, df_original = process_file(file_path, file_ext, scale=True)

    if df_scaled is None or df_original is None:

        return jsonify({"error": "Failed to process file."})

    df_with_predictions = predict_loan_status(df_scaled, df_original)

    prediction_file = os.path.join(app.config['PREDICTIONS_FOLDER'], "predictions.csv")

```

```

df_with_predictions.to_csv(prediction_file, index=False)

    return jsonify({"predictions": df_with_predictions.to_dict(orient='records')})

# Download predictions

@app.route('/download', methods=['GET'])

def download():

    return send_file(os.path.join(app.config['PREDICTIONS_FOLDER'], "predictions.csv"),
as_attachment=True)

if __name__ == '__main__':

app.run(debug=True)

```

## home.html

```

<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta http-equiv="X-UA-Compatible" content="IE=edge">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>HV PROJECT</title>

<link rel="stylesheet" href="{{ url_for('static', filename='stylehome.css') }}">

</head>

<body>

<div class="bgAnimation"></div>

```

```

<div class="background-boxes">

<script>

    for (let i = 0; i< 600; i++) {

document.write('<div></div>');

    }

</script>

</div>

<div class="container">

<nav>

<h1><span>HV </span>PROJECT</h1>

<ul>

<li><a href="/">Home</a></li>

<li><a href="/about">About</a></li>

</ul>

</nav>

<section>

<div class="textBox">

<h1><span>BANK LOAN</span> ELIGIBILITY PREDICTION</h1>

<p>USING MACHINE LEARNING </p>

<a      href="/single"><button      class="homeBtn"      style="--i:#fff;">Single
Prediction</button></a>

<a      href="/index"><button      class="homeBtn"      style="--i:#00bfff;">File
Prediction</button></a>

```

```
</div>

</section>

</div>

</body>

</html>
```

## Single.html

```
<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Single Loan Prediction</title>

<link rel="stylesheet" href="{{ url_for('static', filename='singlestyle.css') }}">

</head>

<body>

<div class="container">

<h1>Loan Eligibility Prediction</h1>

<form action="/predict_single" method="GET" id="loanForm">

<h3><label for="no_of_dep">Number of Dependents:</label>

<input type="number" id="no_of_dep" name="no_of_dep" min="0" required value="{{
form_data.get('no_of_dep', '') }}">
```



```

<label for="education">Education:</label>

<select id="education" name="education" required>

<option value="Graduated" {% if form_data.get('education') == 'Graduated' %}selected{%
endif %}>Graduated</option>

<option value="Not Graduated" {% if form_data.get('education') == 'Not Graduated'
%}selected{% endif %}>Not Graduated</option>

</select>

<label for="self_employed">Self Employed:</label>

<select id="self_employed" name="self_employed" required>

<option value="No" {% if form_data.get('self_employed') == 'No' %}selected{% endif
%}>No</option>

<option value="Yes" {% if form_data.get('self_employed') == 'Yes' %}selected{% endif
%}>Yes</option>

</select>

<label for="income_annum">Annual Income:</label>

<input type="number" id="income_annum" name="income_annum" min="1000" required
value="{{ form_data.get('income_annum', '') }}">

<label for="loan_amount">Loan Amount:</label>

<input type="number" id="loan_amount" name="loan_amount" min="500" required
value="{{ form_data.get('loan_amount', '') }}">

<label for="loan_term">Loan Duration (months):</label>

<input type="number" id="loan_term" name="loan_term" min="1" required value="{{
form_data.get('loan_term', '') }}">

<label for="cibil_score">CIBIL Score:</label>

```

```
<input type="number" id="cibil_score" name="cibil_score" min="300" max="900" required
value="{{ form_data.get('cibil_score', '') }}">
```

```
<label for="assets">Assets Value:</label>
```

```
<input type="number" id="assets" name="assets" min="0" required value="{{
form_data.get('assets', '') }}">
```

```
</h3>
```

```
<div class="button-container">
```

```
<button type="submit">Predict</button>
```

```
<button type="submit" name="clear" value="true">Clear</button>
```

```
</div>
```

```
</form>
```

```
{% if result %}
```

```
<div class="result-box {% if 'Approved' in result %}approved{% else %}rejected{% endif
%}">
```

```
<p>{{ result }}</p>
```

```
</div>
```

```
{% if suggestion %}
```

```
<p class="suggestion"><strong>Suggestion:</strong>{{ suggestion }}</p>
```

```
{% endif %}
```

```
{% endif %}
```

```
</div>
```

```
</body>
```

```
</html>
```

## Batch.html

```
<!DOCTYPE html>

<html lang="en">

<head>

<meta charset="UTF-8">

<meta name="viewport" content="width=device-width, initial-scale=1.0">

<title>Loan Prediction App</title>

<link rel="stylesheet" href="{{ url_for('static', filename='style.css') }}">

</head>

<body>

<div class="container">

<h2>Loan Eligibility Prediction for File </h2>

<form id="upload-form" enctype="multipart/form-data">

<input type="file" id="file-input" name="file" accept=".csv, .xls, .xlsx">

<button type="button" onclick="previewFile()">Preview</button>

<button type="button" onclick="predictFile()">Predict</button>

</form>

<div id="preview-container" style="display: none;">

<h3>File Preview</h3>

<div class="table-wrapper">
```

```

<table id="preview-table">

<thead>

<tr>

<th>name</th>

<th>no_of_dependents</th>

<th>education</th>

<th>self_employed</th>

<th>income_annum</th>

<th>loan_amount</th>

<th>loan_term</th>

<th>cibil_score</th>

<th>Assets</th>

</tr>

</thead>

<tbody></tbody>

</table>

</div>

</div>

<div id="prediction-container" style="display: none;">

<h3>Predicted Results</h3>

<div class="table-wrapper">

```

```

<table id="prediction-table">

<thead>

<tr>

<th>name</th>

<th>no_of_dependents</th>

<th>education</th>

<th>self_employed</th>

<th>income_annum</th>

<th>loan_amount</th>

<th>loan_term</th>

<th>cibil_score</th>

<th>Assets</th>

<th>loan_status</th>

</tr>

</thead>

<tbody></tbody>

</table>

</div>

<button id="download-btn" onclick="downloadFile()">Download Predictions</button>

</div>

</div>

<script>

```

```

function previewFile() {

    let formData = new FormData(document.getElementById('upload-form'));

    fetch('/preview', { method: 'POST', body: formData })

    .then(response => response.json())

    .then(data => {

        if (data.preview) {

            populateTable(data.preview, 'preview-table');

            document.getElementById("preview-container").style.display = "block";

        } else {

            alert("Error: " + data.error);

        }

    })

    .catch(error => console.error('Error:', error));

}

function predictFile() {

    let formData = new FormData(document.getElementById('upload-form'));

    fetch('/predict', { method: 'POST', body: formData })

    .then(response => response.json())

    .then(data => {

        if (data.predictions) {

            populateTable(data.predictions, 'prediction-table');

            document.getElementById("prediction-container").style.display = "block";

```

```

        if (data.download_url) {

document.getElementById("download-btn").style.display = "block";

        }

        } else {

alert("Error: " + data.error);

        }

    })

.catch(error => console.error('Error:', error));

}

function populateTable(data, tableId) {

    let table = document.getElementById(tableId).querySelector("tbody");

table.innerHTML = "";

    let orderedColumns = ["name", "no_of_dependents", "education", "self_employed",
"income_annum",

        "loan_amount", "loan_term", "cibil_score", "Assets"];

    if (tableId === "prediction-table") {

orderedColumns.push("loan_status");

    }

data.forEach(row => {

    let tr = document.createElement("tr");

orderedColumns.forEach(col => {

        let td = document.createElement("td");

td.textContent = row[col] !== undefined && row[col] !== "" ? row[col] : "N/A";

```

```

tr.appendChild(td);

    });

table.appendChild(tr);

    });

}

function downloadFile() {

window.location.href = "/download";

}

</script>

</body>

</html>

```

## **Stylehome.css**

```

*{

    margin: 0;

    padding: 0;

    box-sizing: border-box;

}

body{

color: #fff;

    background: #111;

    font-family: 'Poppins',sans-serif;

    overflow: hidden;

```



```

}

.container{

    width: 100%;

    min-height: 100vh;

    backdrop-filter: blur(1px);

    pointer-events: none;

}

.container nav{

    width: 100%;

    display: flex;

    justify-content: space-between;

    box-sizing: border-box;

    padding: 0 40px;

    height: 100px;

    align-items: center;

}

.container nav h1{

    font-size: 2.6em;

    color: #fff;

    position: relative;

    cursor: pointer;

    pointer-events: all;

```

```
    letter-spacing: 4px;

}

.container nav h1 span{

color: #00bfff;

}

.container nav ul{

    display: flex;

}

.container nav ulli{

    position: relative;

    list-style: none;

    font-size: 1.5em;

    font-weight: 400;

    padding: 12px 18px;

    cursor: pointer;

    pointer-events: all;

    overflow: hidden;

}

.container nav ulli::after{

    content: ";

    position: absolute;
```

```
    bottom: 5px;

    width: 0%;

    height: 3px;

    left: 50%;

    background: #00bfff;

    transform: translateX(-50%);

    transition: .3s;

}

.container nav li:hover{

color: #00bfff;

}

.container nav li:hover::after{

    width: 80%;

}

.container section{

    display: flex;

    justify-content: center;

    align-items: center;

    height: calc(100vh - 100px);

    text-align: center;

    pointer-events: none;

}
```

```
.textBox h1{  
    font-size: 3.4em;  
}  
  
.textBox h1 span{  
    color: #00bfff;  
}  
  
.textBoxp{  
    font-size: 1.2em;  
}  
  
.textBox .homeBtn{  
    font-family: 'Poppins';  
    padding: 8px 18px;  
    margin: 16px 12px;  
    font-size: 26px;  
    background-color: transparent;  
    color: var(--i);  
    outline: none;  
    border: 3px solid var(--i);  
    border-radius: 4px;  
    cursor: pointer;  
    pointer-events: all;  
    transition: .3s;
```

```

    font-weight: 600;

}

.textBox .homeBtn:hover{

    background-color: var(--i);

color: #000;

}

.bgAnimation {

    position: absolute;

    top: 0;

    left: 0;

    width: 100%;

    height: 100vh;

    background: linear-gradient(to bottom, rgba(248, 3, 3, 0), rgba(0, 191, 255));

    animation: fadeBackground 6s linear infinite;

    z-index: -1;

}

@keyframes fadeBackground {

    0% { transform: translateY(-100%); opacity: 0.3; }

    50% { transform: translateY(0); opacity: 1; }

    100% { transform: translateY(100%); opacity: 0.3; }

}

.background-boxes {

```

```

position: absolute;

top: 0;

left: 0;

width: 100%;

height: 100%;

display: grid;

grid-template-columns: repeat(31, 50px);

grid-template-rows: repeat(auto-fill, 50px);

z-index: 0;

}

.background-boxes div {

width: 47px;

height: 47px;

background: #000000;

border-radius: 10px;

transition: background 1.3s ease;

}

.background-boxes div:hover {

background: #00bfff;

transition: 0s;

}

a{

```

```
text-decoration: none;

color: #fff;

}
```

## **Singlestyle.css**

```
body {

    display: flex;

    background-image: url('bank.jpg');

background-repeat:no-repeat;

    justify-content: center;

    background-size: cover;

    align-items: center;

    min-height: 100vh;

    /* background: #000;*/

    position: relative;

}

/* Loan Form Container */

.container {

    position: relative;

    background: rgba(255, 255, 255, 0.9); /* Semi-transparent white */

    padding: 20px;

    border-radius: 10px;

    box-shadow: 0px 0px 15px rgba(255, 255, 255, 0.2);

}
```

```

width: 500px;

height: 850px;

z-index: 1; /* Keeps content above animation */

text-align: center;
}

h2 {

color: #000;

}

form {

text-align: left;

margin-top: 10px;

}

input, select, button {

width: 100%;

padding: 8px;

margin: 10px 0;

border: 1px solid #ccc;

border-radius: 5px;

}

button {

background-color: #007bff;

color: white;

```



```

    cursor: pointer;

    border: none;
}

button:hover {

    background-color: #0056b3;
}

.clear-btn {

    background-color: #f44336;
}

.clear-btn:hover {

    background-color: #d32f2f;
}

/* Result Box */

.result-box {

    margin-top: 20px;

    padding: 15px;

    border-radius: 5px;

    font-size: 18px;

    font-weight: bold;
}

.approved {

    background-color: #c8e6c9;

```

```
color: green;

}

.rejected {

    background-color: #ffcdd2;

color: red;

}

.suggestion {

background-color:white;

    font-size: 30px;

color: #050505;

}
```

## **Batchstyle.css**

```
body {

    font-family: Arial, sans-serif;

    background-color: #f4f4f4;

    text-align: center;

    padding: 20px;

}

.container {

    max-width: 900px;

    margin: auto;
```

```

background: rgb(255, 255, 255);

padding: 20px;

border-radius: 8px;

box-shadow: 0px 0px 10px rgba(0, 0, 0, 0.1);

}

h2, h3 {

color: #333;

}

input[type="file"] {

margin-bottom: 10px;

}

button {

padding: 10px 15px;

margin: 5px;

border: none;

cursor: pointer;

background-color: #007BFF;

color: white;

border-radius: 5px;

font-size: 16px;

transition: background 0.3s;

}

```

```

button:hover {

    background-color: #0056b3;

}

/* Scrollable Table */

.table-wrapper {

    max-width: 100%;

    max-height: 400px; /* Added vertical scroll */

    overflow-x: auto;

    overflow-y: auto;

    border: 1px solid #ccc;

    margin-top: 10px;

    background: white;

}

table {

    width: 100%;

    border-collapse: collapse;

    min-width: 900px; /* Ensures horizontal scrolling */

}

th, td {

    border: 1px solid #ddd;

    padding: 8px;

    text-align: left;

```

```
white-space: nowrap; /* Prevents text wrapping */
```

```
}
```

```
th {
```

```
    background-color: #007BFF;
```

```
    color: white;
```

```
}
```

```
/* Responsive Design */
```

```
@media screen and (max-width: 768px) {
```

```
    .container {
```

```
        width: 90%;
```

```
        padding: 10px;
```

```
    }
```

```
    table {
```

```
        min-width: 100%;
```

```
    }
```

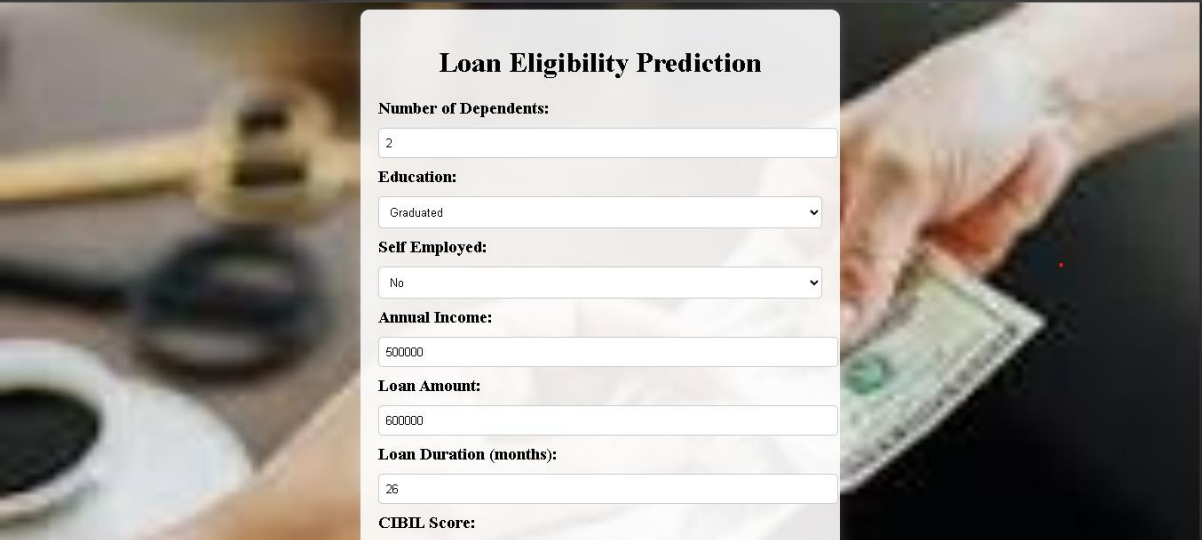
```
}
```

## 9. SAMPLE SCREENSHOTS

### Home page



## Individual Prediction Page



### Loan Eligibility Prediction

**Number of Dependents:**

**Education:**

**Self Employed:**

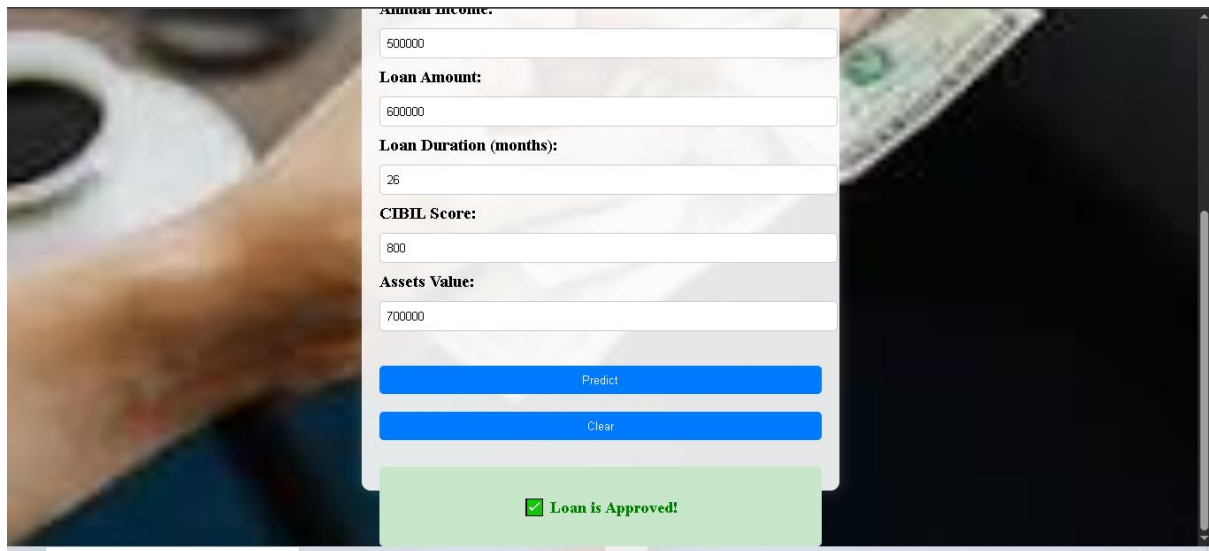
**Annual Income:**

**Loan Amount:**

**Loan Duration (months):**

**CIBIL Score:**

## Approved Page



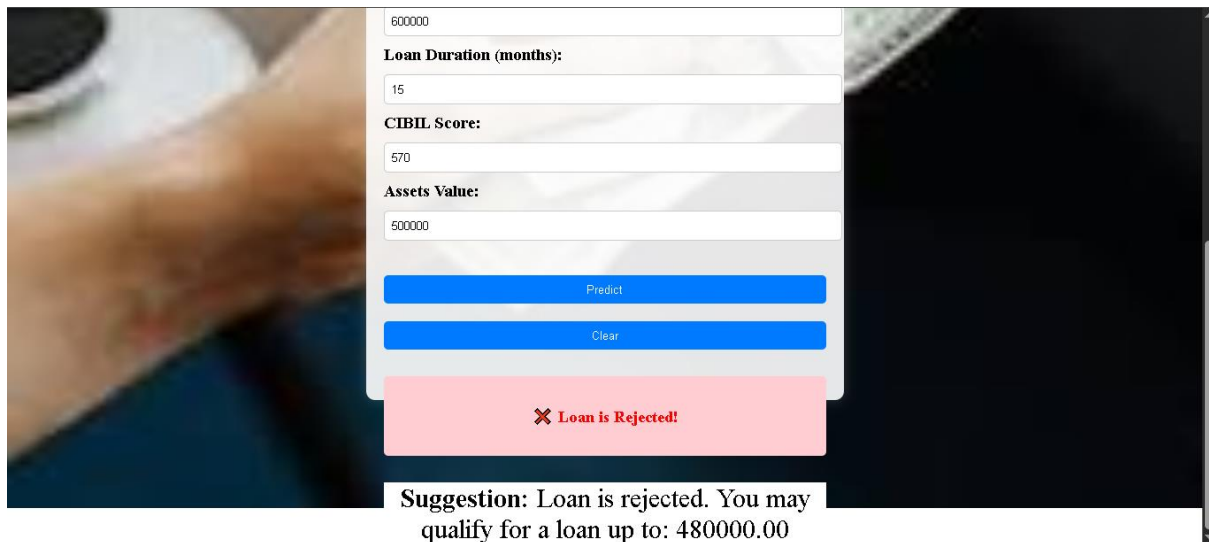
The screenshot shows a web application interface for loan approval. The background is a blurred image of a person's arm and a cup. The application form is centered and contains the following fields and buttons:

- Annual Income:** Input field with the value 500000.
- Loan Amount:** Input field with the value 600000.
- Loan Duration (months):** Input field with the value 26.
- CIBIL Score:** Input field with the value 800.
- Assets Value:** Input field with the value 700000.
- Predict** button (blue).
- Clear** button (blue).
- Loan is Approved!** message (green box with a checkmark icon).

The bottom of the screen shows a Windows taskbar with several application icons and a system clock displaying 10:47.



## Rejected Page



600000

**Loan Duration (months):**

15

**CIBIL Score:**

570

**Assets Value:**

500000

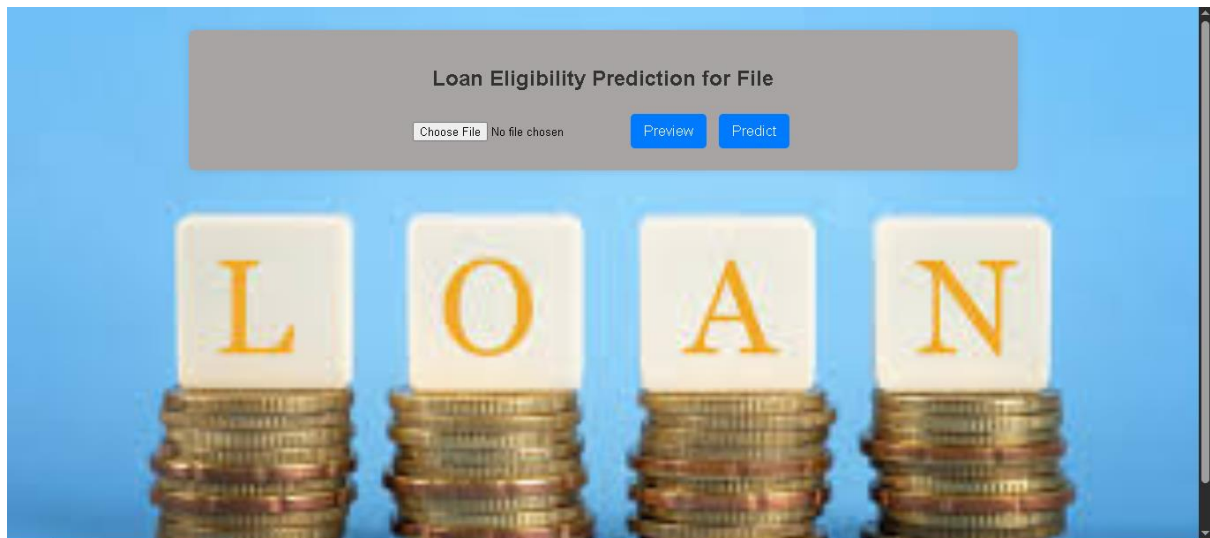
Predict

Clear

✖ Loan is Rejected!

**Suggestion:** Loan is rejected. You may qualify for a loan up to: 480000.00

## File Prediction Page



## File Preview Page

### Loan Eligibility Prediction for File

Choose File

pre3.csv

Preview

Predict

#### File Preview

name	no_of_dependents	education	self_employed	income_annum	loan_amount	loan_term	cibil
kali	2	Graduated	No	9600000	29900000	12	778
hari	0	Not Graduated	Yes	4100000	12200000	8	417
arul	3	Graduated	No	9100000	29700000	20	506
balaji	3	Graduated	No	8200000	30700000	8	467
ganesh	5	Not Graduated	Yes	9800000	24200000	20	382
chelpa	0	Graduated	Yes	4800000	13500000	10	319
venkatesh	2	Graduated	Yes	5700000	15000000	20	382
shrini	0	Graduated	Yes	800000	2200000	20	782
thiyagu	5	Not Graduated	No	1100000	4300000	10	388
kali	2	Graduated	No	9600000	29900000	12	778

## File Download Page

thiyagu	5	Not Graduated	No	1100000	4300000	10	388
Predicted Results							
education	self_employed	income_annum	loan_amount	loan_term	cibil_score	Assets	loan_status
Graduated	No	9600000	29900000	12	778	50700000	Approved
Not Graduated	Yes	4100000	12200000	8	417	17000000	Rejected
Graduated	No	9100000	29700000	20	506	57700000	Rejected
Graduated	No	8200000	30700000	8	467	52700000	Rejected
Not Graduated	Yes	9800000	24200000	20	382	55000000	Rejected
Graduated	Yes	4800000	13500000	10	319	33900000	Rejected
Graduated	Yes	5700000	15000000	20	382	36700000	Rejected
Graduated	Yes	800000	2200000	20	782	5500000	Approved
Not Graduated	No	1100000	4300000	10	388	9500000	Rejected
Graduated	No	9600000	29900000	12	778	50700000	Approved
Download Predictions							

## 10. CONCLUSION

The development of the **Loan Eligibility Prediction System** marks a significant step in the application of machine learning and web technologies to solve practical financial challenges. By leveraging the power of Python, XGBoost, and user-centric web design, the system provides a seamless and automated approach for evaluating loan eligibility. It supports both individual and bulk predictions, making it versatile and adaptable for use by individuals, banks, and lending institutions.

This project successfully covered the complete machine learning pipeline—from data preprocessing and model training to deployment and user interaction—within a structured implementation plan. The use of JupyterLab for development ensured a clear, testable, and iterative model-building process, while Visual Studio Code and front-end tools helped create a responsive and functional web interface.

Through the inclusion of loan duration, income, CIBIL score, and other key financial indicators, the system provides accurate predictions that aid decision-makers in minimizing risks and maximizing efficiency. By saving and integrating the model using pickle, the backend remains lightweight, scalable, and easy to maintain.

This project not only demonstrates technical proficiency but also showcases how AI can be used responsibly and effectively to support business operations and improve user experience.

## FUTURE ENHANCEMENTS

Although the system is fully functional, there are several areas where further improvements and additional features can elevate its effectiveness and usability.

### Enhanced Data Validation and Error Handling

- Implement more robust front-end and back-end validation to catch missing or invalid inputs.
- Provide real-time error messages and suggestions to users for correction.

## **2. Database Integration**

- Store user inputs and prediction results in a database (e.g., SQLite, MySQL).
- Enable historical tracking, search, and reporting features for admin users.

## **3. Cloud Deployment**

- Host the web application on platforms like Heroku, Render, or AWS.
- Enable global accessibility, ensuring the app can be used from any device, anywhere.

## **4. User Authentication System**

- Introduce login functionality for different user roles (e.g., admins, bank officers, customers).
- Secure access to sensitive data and enable personalization of user experience.

## **5. Model Comparison and Updates**

- Add the ability to switch between different machine learning models (e.g., Random Forest, LightGBM).
- Implement automated model retraining based on new incoming data.

## **6. Analytics Dashboard**

- Integrate charts and visual summaries (using Chart.js or Plotly) to show trends in loan approvals, common denial reasons, etc.
- Allow admin users to gain insight from real-time data.

## **7. Support for More File Formats**

- Enable support for Excel (.xlsx) and JSON file formats in addition to CSV.
- Improve flexibility for enterprise users and varied data sources.

## BIBLIOGRAPHY

1. XGBoost: A Scalable Tree Boosting System Chen, T., & Guestrin, C. (2016). *XGBoost: A Scalable Tree Boosting System*. In Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining. <https://xgboost.readthedocs.io>
2. Python Programming Language Python Software Foundation. (2023). *Python Language Documentation*. <https://www.python.org/doc/>
3. JupyterLab Documentation Project Jupyter. (2023). *JupyterLab: The Next Generation Interface for Project Jupyter*. <https://jupyterlab.readthedocs.io>
4. HTML5 and CSS3 Specifications World Wide Web Consortium (W3C). (2023). *HTML & CSS Standards and Specifications*. <https://www.w3.org/TR/html52/>
5. Pandas: Python Data Analysis Library The pandas development team. (2023). *Pandas Documentation*. <https://pandas.pydata.or>
6. Loan Dataset and Financial Risk Factors Kaggle. (2022). *Loan Prediction Dataset*. <https://www.kaggle.com>
7. **Jain, P., & Singh, R.** (2019). *Machine learning techniques for financial prediction and risk management*. Springer. This book provides an in-depth look at machine learning algorithms and their application in financial prediction, including credit risk assessment and loan eligibility prediction.
8. **Sharma, A., & Verma, S.** (2021). *Data Science and Machine Learning in Banking and Finance*. Wiley. A comprehensive guide to the application of data science and machine learning techniques in the banking and finance industry, including loan eligibility predictions, credit scoring, and fraud detection.
9. **Raj, M., & Patel, N.** (2020). *Artificial Intelligence in Financial Services: Theory and Applications*. Springer 2020. A focused exploration of artificial intelligence and machine learning in the financial services sector, including case studies on loan prediction systems, credit risk analysis, and fraud detection.
10. **Ghosh, S., & Sanyal, A.** (2022). *Machine Learning for Financial Services: Applications and Insights*. Elsevier. This book covers practical applications of machine learning in the financial services sector, with dedicated sections on credit scoring, loan eligibility prediction, and customer risk assessment.