**SUMMER TRAINING REPORT**

**On**

# Machine Learning

Submitted to Guru Gobind Singh Indraprastha University, Delhi (India)

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

## B.TECH

**in**

**Information Technology**


**Submitted By**

**SIRISHA FULARA**

**Roll. No.- 00196303123**




**DEPTT. OF Information Technology**

**MAHARAJA SURAJMAL INSTITUTE OF TECHNOLOGY,**

**NEW DELHI-110058**


**September 2025**

# ACKNOWLEDGEMENT

A research work owes its success from commencement to completion, to the people in love with researchers at various stages. Let me in this page express my gratitude to all those who helped us in various stage of this study. First, I would like to express my sincere gratitude indebtedness to Dr. Sunesh Malik **(HOD, Department of Information Technology, Maharaja Surajmal Institute of Technology, New Delhi)** for allowing me to undergo the summer training of 30 days at **Samyak IT Solutions Pvt. Ltd.**

I am grateful to our guide **Mr. Himanshu Badola**, for the help provided in completion of the project, which was assigned to me. Without his friendly help and guidance it was difficult to develop this project.

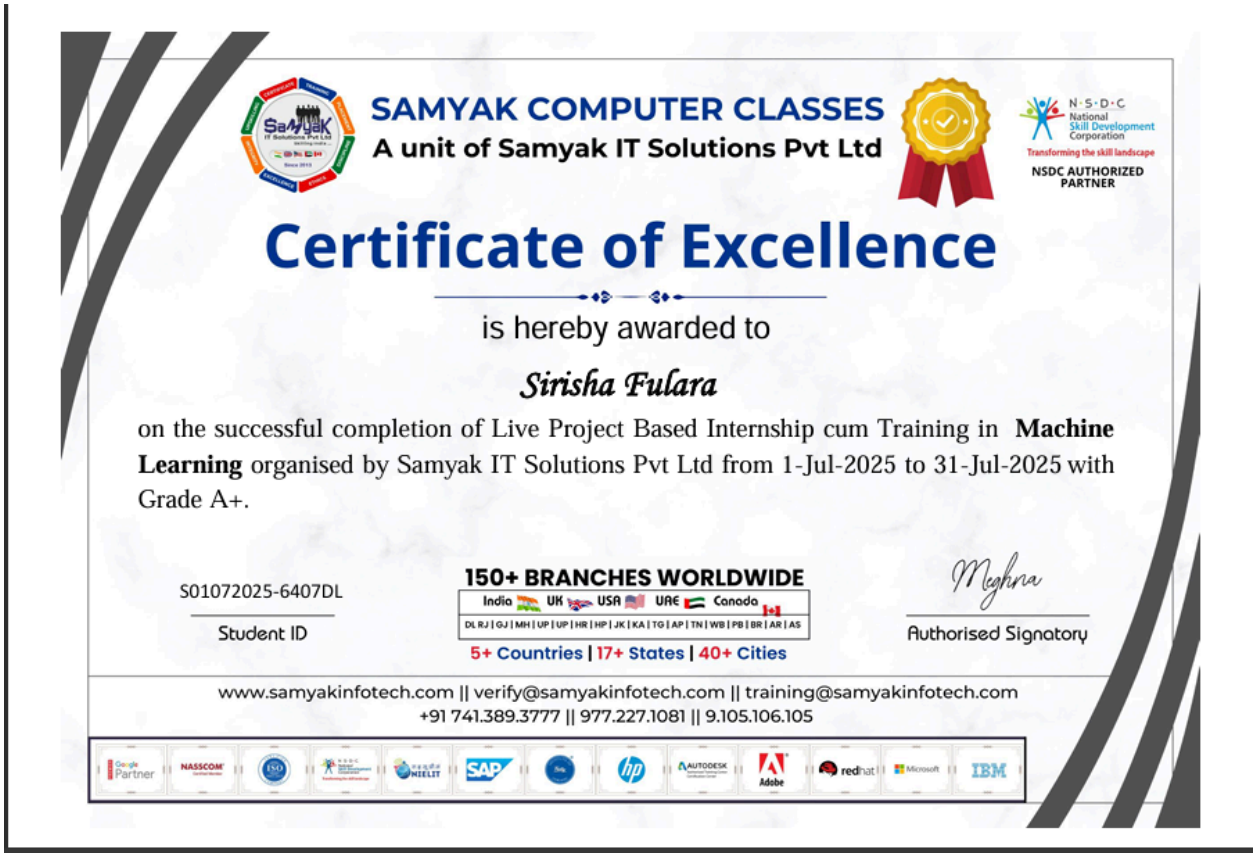
Last but not least, I pay my sincere thanks and gratitude to all the Staff Members of **Samyak IT Solutions** for their support and for making our training valuable and fruitful.

**Submitted By: Sirisha Fulara**

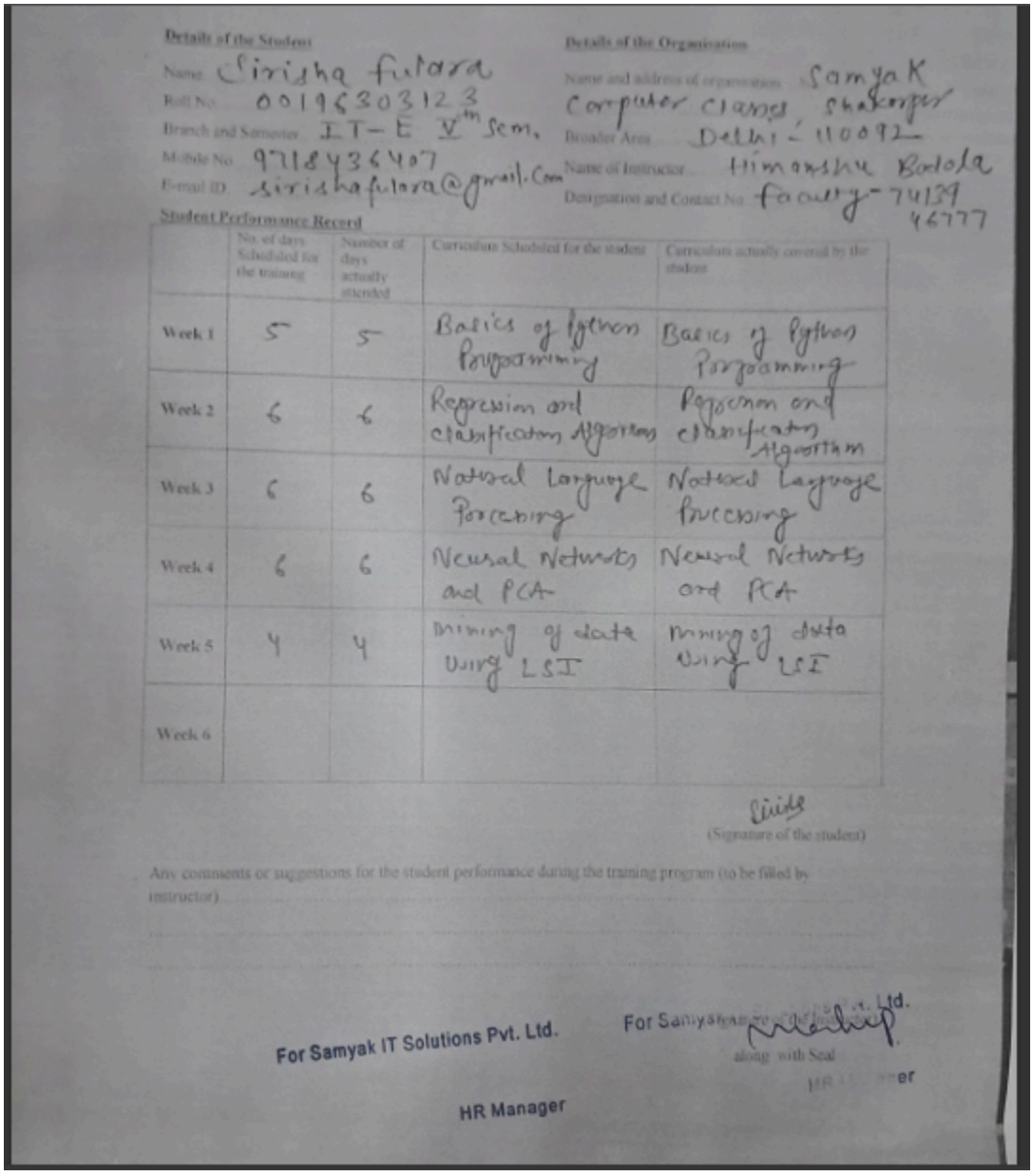
**00196303123**

**IT-E (B.Tech)**

# CERTIFICATE



# FEEDBACK FORM



**Details of the Student**

Name: Sirisha Futara
Roll No: 00196303123
Branch and Semester: IT-E V^th Sem.
Mobile No: 9718436407
E-mail ID: sirishafutara@gmail.com

**Details of the Organization**

Name and address of organization: Samyak Computer Classes, Shakarpur Delhi - 110092
Broader Area:
Name of Instructor: Himanshu Bodola
Designation and Contact No: faculty - 74139 46777

**Student Performance Record**

| | No. of days Scheduled for the training | Number of days actually attended | Curriculum Scheduled for the student | Curriculum actually covered by the student |
|---|---|---|---|---|
| Week 1 | 5 | 5 | Basics of Python Programming | Basics of Python Programming |
| Week 2 | 6 | 6 | Regression and classification Algorithms | Regression and classification Algorithm |
| Week 3 | 6 | 6 | Natural Language Processing | Natural Language Processing |
| Week 4 | 6 | 6 | Neural Networks and PCA | Neural Networks and PCA |
| Week 5 | 4 | 4 | Mining of data Using LSI | Mining of data Using LSI |
| Week 6 | | | | |

(Signature of the student)

Any comments or suggestions for the student performance during the training program (to be filled by instructor)

For Samyak IT Solutions Pvt. Ltd.

HR Manager

For Samyak Computer Classes Pvt. Ltd.

along with Seal

HR Manager

# CANDIDATE'S DECLARATION

I, **Sirisha Fulara**, **00196303123**, B.Tech (Semester- 5th ) of the Maharaja Surajmal Institute of Technology, New Delhi hereby declare that the Training Report entitled "**Machine Learning**" is an original work and data provided in the study is authentic to the best of my knowledge. This report has not been submitted to any other Institute for the award of any other degree.

**Sirisha Fulara**
**( 00196303123)**

**Place**: Delhi
**Date**: 19-09-2025

# COMPANY INFORMATION

Samyak IT Solutions Pvt. Ltd. is a recognized leader in IT training and software solutions. The company has been awarded “Best Training Institution” by multiple reputed organizations, including EAR, the Industry Minister, Live24 News Channel, and various national forums.

Samyak believes that effective upskilling and career readiness require deep personalization, aligning every learner’s journey with their career aspirations. By combining innovative training models, global expertise, and localized support, Samyak ensures meaningful growth for students and professionals alike.

## Services & Programs

1. Upskilling India Initiative
   Focused on empowering individuals across India with in-demand digital skills to succeed in today’s technology-driven world.
2. Mentorship Programs
   Specially designed mentorship opportunities for learners from all Indian cities, helping them unlock their potential and become industry-ready.
3. Job-Ready Mentorship Programs
   Bridging the gap between academic knowledge and industry requirements through hands-on learning and career-focused guidance.

4. Project-Based Learning

   Encouraging learners to engage in real-world projects, enabling them to develop problem-solving skills and industry-level expertise.

5. Study Abroad Services

   Providing students with international education opportunities through expert counseling and support.

## Mission Statement

Samyak IT Solutions Pvt. Ltd. is committed to empowering businesses through innovative IT solutions and driving digital transformation. With a focus on excellence, integrity, and strong partnerships, the company delivers impactful technology services while continuously fostering learning, growth, and sustainability.

## Motto

“With You Every Step of the Way” – highlighting the company’s dedication to providing personalized support and guidance for every student’s success.

# ABSTRACT

The Machine Learning course undertaken during the summer provided both theoretical foundations and practical exposure to the development of intelligent systems. The course emphasized the implementation of machine learning models, data preprocessing techniques, model evaluation strategies, and the integration of AI into real-world applications.

As part of the course, a capstone project titled *Predictify* was developed, aimed at creating an AI/ML-powered loan approval prediction system. The platform was built using Python, Flask for the backend, and React for the frontend, enabling accurate eligibility predictions through machine learning models. To enhance transparency and interpretability, SHAP-based feature visualization was incorporated, along with an interactive dashboard for users to understand the model's decision-making process. REST APIs were designed to enable seamless client-server communication, while deployment ensured accessibility and scalability of the system.

This project demonstrates the practical application of machine learning concepts in financial decision-making, showcasing the integration of modern web technologies with AI models. The combination of predictive accuracy, interpretability, and user-centric design highlights the potential of machine learning in solving real-world challenges. Overall, the course and project provided a strong foundation in AI development, bridging the gap between academic learning and practical implementation.

# LIST OF FIGURES

# CONTENTS

# CHAPTER 1- TECHNOLOGY USED

## 1.1 Python

### 1.1.1 History

Python was created in 1991 by Guido van Rossum at Centrum Wiskunde & Informatica (CWI), Netherlands. The goal was to design a language that emphasized code readability and simplicity while being powerful enough for scientific and industrial use. Over time, Python has become the backbone of modern technologies like machine learning, artificial intelligence, web development, data analysis, and automation.

Python's popularity skyrocketed with the rise of data science and AI because of its huge ecosystem of libraries (NumPy, Pandas, TensorFlow, PyTorch, Scikit-learn). It is now one of the most widely taught and used programming languages across the globe.

### 1.1.2 Key Features

- Easy and Readable Syntax: Python code almost reads like plain English, reducing complexity.
- Cross-Platform: Works seamlessly on Windows, Linux, and macOS.
- Large Standard Library: Provides modules for networking, math, file handling, and more.
- Dynamic Typing: No need to declare variable types explicitly.
- Huge Community Support: One of the largest developer communities, ensuring constant improvements and library support.
- Supports Multiple Paradigms: Procedural, functional, and object-oriented programming.
- Ideal for Machine Learning: Most ML/DL frameworks are Python-first (TensorFlow, Scikit-learn, PyTorch).

### 1.1.3 Why Use Python for Machine Learning?

- Extensive libraries for data handling (Pandas, NumPy), visualization (Matplotlib, Seaborn), and ML (Scikit-learn, TensorFlow, PyTorch).
- Easy to integrate with APIs, databases, and web frameworks like Flask or Django.
- Readable and beginner-friendly, which reduces development time.
- Backed by a huge research and industry community constantly pushing AI/ML boundaries.

### 1.1.4 Installation Guide

1. Download Python from https://www.python.org/downloads/.
2. During installation, check "Add Python to PATH".
3. Verify installation:

```bash
python --version
```

### 1.1.5 Object-Oriented Programming (OOP) in Python

What is OOP?

OOP is a programming paradigm based on the concept of objects that represent real-world entities. It helps make code modular, reusable, and easier to maintain.

Core Principles of OOP (4 Pillars):

1. Encapsulation: Bundling data (variables) and methods (functions) into a class.
2. Abstraction: Hiding implementation details and exposing only essential features.
3. Inheritance: A class can inherit attributes and methods from another class (code reuse).
4. Polymorphism: Same function name can have different behaviors depending on context.

## 1.2 NumPy

### 1.2.1 History

NumPy (Numerical Python) was released in 2005 by Travis Oliphant, built on earlier libraries like Numeric and Numarray. It became the backbone of scientific computing in Python, powering almost every ML/DL framework (Scikit-learn, TensorFlow, PyTorch, Pandas).

### 1.2.2 Key Features

- ndarray: Multi-dimensional, homogeneous arrays (faster than Python lists).
- Vectorized operations: Perform math on whole arrays without loops.
- Linear algebra & statistics: Matrix multiplication, eigenvalues, means, etc.
- Random number generation: Essential for ML (random splits, shuffling).
- Works seamlessly with C/C++, making it highly optimized.

### 1.2.3 Why Use NumPy in ML?

- Core for matrix computations in ML (weights, biases, input vectors).
- Used for feature engineering, normalizations, and mathematical modeling.
- Serves as the foundation for Pandas and Scikit-learn.

### 1.2.4 Installation Guide

```bash
pip install numpy
```

### 1.2.5 Example Code

```python
import numpy as np

# Creating arrays
arr = np.array([1, 2, 3, 4, 5])
matrix = np.array([[1, 2], [3, 4]])

print("Array:", arr)
print("Matrix:\n", matrix)

# Array operations
print("Mean:", np.mean(arr))
print("Matrix Multiplication:\n", np.dot(matrix, matrix))
print("Random Numbers:", np.random.randint(1, 10, size=5))
```

Fig-1.1 Numpy implementation

## 1.3 Pandas

### 1.3.1 History

Created by Wes McKinney in 2008, Pandas was built on top of NumPy to provide easier handling of tabular/structured data. The name comes from "Panel Data" in econometrics.

### 1.3.2 Key Features

- DataFrame: 2D data structure like an Excel sheet.
- Data Cleaning: Handling missing/null values.
- Data Transformation: Filtering, grouping, merging, reshaping.
- Integration: Works with NumPy, Matplotlib, and ML libraries.

### 1.3.3 Why Use Pandas in ML?

- Data preprocessing is 70% of ML work → Pandas is perfect for it.
- Convert raw CSV/Excel/SQL datasets into ML-ready formats.
- Feature engineering like scaling, encoding, outlier detection becomes easier.

### 1.3.4 Installation Guide

```bash
pip install pandas
```

### 1.3.5 Example Code

```python
import pandas as pd

# Creating a DataFrame
data = {"Name": ["Amit", "Riya", "Sam"],
        "Income": [30000, 45000, 25000],
        "Credit_Score": [720, 650, 600]}
df = pd.DataFrame(data)

print("Dataset:\n", df)

# Basic operations
print("Average Income:", df["Income"].mean())
print("Sorted by Credit Score:\n", df.sort_values("Credit_Score"))
print("Applicants with good credit:\n", df[df["Credit_Score"] > 650])
```

Fig-1.2 Pandas Implementation

## 1.4 Matplotlib

### 1.4.1 History

Developed by John D. Hunter in 2003, Matplotlib was inspired by MATLAB's plotting system. Today it's the most widely used Python library for data visualization.

### 1.4.2 Key Features

- Multiple chart types (line, bar, scatter, histogram, pie).
- Full control over titles, labels, colors, and legends.

- Can generate both static and interactive plots.
- Works with NumPy arrays and Pandas DataFrames.

### 1.4.3 Why Use Matplotlib in ML?

- Visualize datasets: See distributions, missing values, outliers.
- Understand model results: Compare predictions vs. actual values.
- Communicate results with plots & dashboards.

### 1.4.4 Installation Guide

```bash
pip install matplotlib
```

### 1.4.5 Example Code

```python
import matplotlib.pyplot as plt

# Simple Line Plot
x = [1, 2, 3, 4, 5]
y = [10, 20, 15, 25, 30]

plt.plot(x, y, marker='o', color='blue')
plt.title("Loan Approval Trend")
plt.xlabel("Applicant ID")
plt.ylabel("Approval Probability")
plt.show()

# Histogram Example
ages = [25, 30, 35, 40, 28, 32, 36, 27]
plt.hist(ages, bins=5, color='green')
plt.title("Age Distribution of Applicants")
plt.show()
```

Fig-1.3 Matplotlib Implementation

## 1.2 Regression and Classification Algorithms

### 1.2.1 Linear Regression

- Theory: Models the relationship between input variables X and target variable y using a straight line.
- Equation:

$$y = \beta_0 + \beta_1 x_1 + \beta_2 x_2 + \cdots + \beta_n x_n + \epsilon$$

- Math: Finds coefficients by minimizing the sum of squared errors (SSE).

$$\text{SSE} = \sum (y_i - \hat{y}_i)^2$$

- Use case: Predicting loan amount based on salary, age, and existing debts.

### 1.2.2 Support Vector Regression (SVR)

- Theory: Extends SVM for regression. It tries to fit a function within a margin of tolerance (ϵ).
- Equation:

$$f(x) = w \cdot x + b$$

Where:

**w** → weight vector (defines direction of the hyperplane)

**x** → input feature vector (your data point)

**b** → bias (shifts the hyperplane away from the origin)

**f(x)** → output (used to decide which side of the hyperplane the point lies on)

The equation f(x)=0 defines the **decision boundary (hyperplane)**.

If **f(x) > 0** → the point belongs to **Class +1**

If **f(x) < 0** → the point belongs to **Class -1**

If **f(x) = 0** → the point lies **exactly on the boundary**

- Math: Maximizes margin while minimizing error beyond ϵ. Uses kernel trick for non-linear trends.

- Use case: Predicting fluctuating interest rates.

### 1.2.3 Decision Tree Regression

- Theory: Splits data into regions by asking yes/no questions (e.g., *Is income > 40k?*). Each leaf node gives a prediction.

- Math: At each split, choose the feature that minimizes Mean Squared Error (MSE):

$$MSE = \frac{1}{n}\sum(y_i - \hat{y})^2$$

- Use case: Predicting loan repayment capacity based on multiple applicant features.

### 1.2.4 Random Forest Regression

- Theory: Combines many decision trees (each trained on random samples) and averages predictions.

- Math:

$$\hat{y} = \frac{1}{T}\sum_{t=1}^{T} f_t(x)$$

where T = number of trees.

- Strength: Reduces overfitting, improves accuracy.
- Use case: Predicting risk score of applicants.

### 1.2.5 Logistic Regression

- Theory: Predicts probabilities instead of continuous values. Uses the sigmoid function to map values between 0 and 1.
- Equation:

$$p = \frac{1}{1 + e^{-(\beta_0 + \beta_1 x)}}$$

- Math: Maximizes log-likelihood instead of minimizing MSE.
- Use case: Classifying loan status as Approved (1) or Rejected (0).

### 1.2.6 Support Vector Machine (SVM Classifier)

- Theory: Finds the best hyperplane that separates two classes with maximum margin.
- Use case: Separating good applicants vs risky applicants.

### 1.2.7 Decision Tree Classifier

- Theory: Works like a flowchart; each split reduces impurity.
- Use case: Loan approval decision based on credit score, income, age.

### 1.2.8 Random Forest Classifier

- Theory: Builds multiple decision trees, takes a majority vote for classification.
- Strength: More stable and accurate than single trees.
- Use case: Final loan approval prediction in *Predictify*.

## 1.3. Clustering Algorithms

### 1.3.1 K-Means Clustering

- Theory:

  K-Means is an unsupervised learning algorithm that groups data into k clusters by minimizing intra-cluster distance. It iteratively assigns points to clusters and updates the cluster centroids.
- Steps:
  1. Choose k (number of clusters).
  2. Initialize centroids randomly.
  3. Assign points to nearest centroid.
  4. Recalculate centroids.
  5. Repeat until convergence.
- Use case: Grouping loan applicants into low, medium, and high-risk categories.

### 1.3.2 Hierarchical Clustering

- Theory:

  Builds a hierarchy of clusters using agglomerative (bottom-up) or divisive (top-down) methods. Unlike K-means, the number of clusters need not be fixed upfront.
- Mathematics:

  Distance between clusters can be computed using:
  - Single linkage
  - Complete linkage
  - Average linkage
- Use case: Creating a hierarchy of customer groups (e.g., income-based tiers for banking products).

## 1.4. Association Rule Learning

### 1.4.1 Apriori Algorithm

- Theory:

  Apriori finds frequent itemsets in transactional datasets and derives association rules (if $X \rightarrow Y$). It works on the principle that:

  If an itemset is frequent, all its subsets are also frequent.
- Mathematics:
  - **Support:**

$$Support(X) = \frac{\text{Transactions containing } X}{\text{Total transactions}}$$

  - **Confidence:**

$$Confidence(X \rightarrow Y) = \frac{Support(X \cup Y)}{Support(X)}$$

  - **Lift:**

$$Lift(X \rightarrow Y) = \frac{Confidence(X \rightarrow Y)}{Support(Y)}$$

- Use case: In finance, finding loan features that frequently appear together (e.g., "High income + Good credit history → Loan approved").

### 1.4.2 Eclat Algorithm

- Theory:

  Eclat (Equivalence Class Transformation) is similar to Apriori but uses Depth-First Search (DFS) and intersection of transaction IDs (TIDs) to find frequent itemsets. It's usually faster than Apriori on dense datasets.
- Mathematics:

  Support is computed using set intersection:

$$Support(X) = \frac{|T(X)|}{N}$$

  where T(X) = transaction IDs containing itemset X, and N = total transactions.

- Steps:
    1. Represent data as transaction ID sets.
    2. Find frequent itemsets by intersecting TID sets.
    3. Generate rules like Apriori.

- Use case: Identifying patterns in rejected applications (e.g., "Low income ∩ Poor credit → Rejected loans").

## 1.6. Natural Language Processing (NLP)

### 1.6.1 Introduction

- NLP is the field of AI that enables machines to understand, interpret, and generate human language.
- It combines linguistics + computer science + ML to process text and speech.
- Applications: chatbots, sentiment analysis, document classification, fraud detection, and even financial credit risk analysis.

### 1.6.2 Key Concepts

#### 1.6.2.1 Text Preprocessing

Before feeding text into ML models, it must be cleaned:

1. Tokenization → Splitting sentences into words.
2. Stopword removal → Removing common words like "the", "is", etc.
3. Stemming/Lemmatization → Reducing words to root forms (e.g., "running" → "run").
4. Vectorization → Converting words into numbers.

- Mathematics of Bag-of-Words (BoW):
  For a document D and vocabulary V:

$$Vector(D) = [count(w_1), count(w_2), ..., count(w_n)]$$

#### 1.6.2.2 TF-IDF (Term Frequency – Inverse Document Frequency)

- Theory: Improves BoW by reducing the weight of common words and giving importance to rare but significant words.
- Use case: Prioritizing keywords like *"default"* or *"approved"* in loan applications.

## 1.7. Deep Learning & Neural Networks

### 1.7.1 Introduction

- Deep Learning (DL) is a subset of ML using Artificial Neural Networks (ANNs) with multiple layers.
- Inspired by the human brain, where neurons (nodes) process inputs and pass signals forward.
- Used when datasets are large and problems are complex: images, speech, natural language, fraud detection, etc.

### 1.7.2 Structure of a Neural Network

1. Input Layer → Features (like income, credit score, loan history).

2. Hidden Layers → Transform data using weights & activation functions.

3. Output Layer → Predictions (loan approved = 1, rejected = 0).

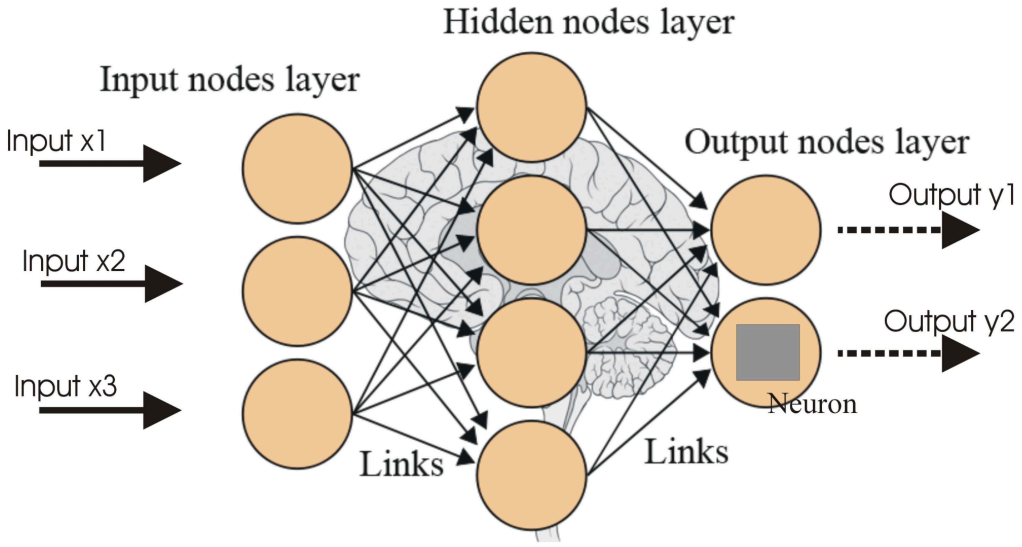



Fig-1.4 Structure of Neural network

### 1.7.3 Mathematics of a Neuron

- Each neuron computes:

$$z = \sum_{i=1}^{n} w_i x_i + b$$

$$a = f(z)$$

Where:

- xi = input features
- wi = weights
- b = bias
- f = activation function (ReLU, Sigmoid, Softmax)

**Activation Functions:**

- Sigmoid: $\sigma(z) = \frac{1}{1+e^{-z}}$ → outputs between (0,1).
- ReLU: $f(z) = \max(0, z)$ → avoids vanishing gradient.

### 1.7.4 Learning in Neural Networks

- Forward Propagation: Compute output from inputs.
- Loss Function:

$$L = \frac{1}{n}\sum(y_{true} - y_{pred})^2$$

- Backpropagation: Adjust weights using Gradient Descent.

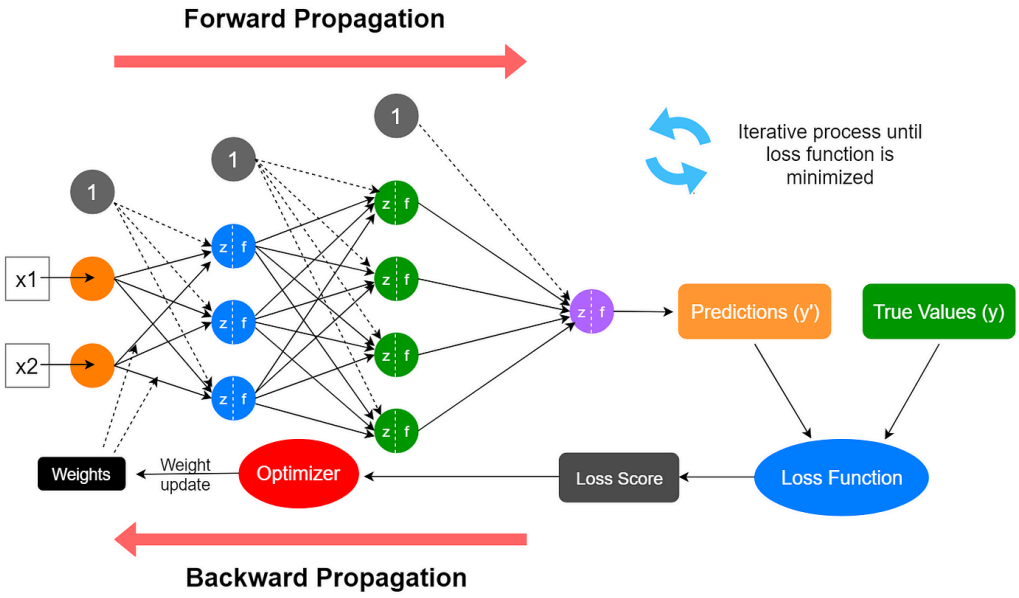



Fig 1.5 Learning in Neural Network

## 1.8. Principal Component Analysis (PCA)

### 1.8.1 Introduction

- PCA is a dimensionality reduction technique used to reduce high-dimensional datasets into fewer variables while retaining most of the variance.
- Useful when there are too many features (like hundreds of financial attributes).
- Helps in visualization and improves ML efficiency.

### 1.8.2 Theory

- PCA finds new axes (Principal Components) that maximize variance in the data.
- The first PC captures the most variance, second PC the next, and so on.

### 1.8.3 Mathematics

1. Standardize the dataset.
2. Compute Covariance Matrix:

$$C = \frac{1}{n} X^T X$$

3. Compute Eigenvalues (λlambda) and Eigenvectors (v) of C.
4. Sort eigenvectors by eigenvalues (variance explained).
5. Select top k eigenvectors → new feature space.

## 1.9. Data Mining using Latent Semantic Indexing (LSI)

### 1.9.1 Introduction

- Latent Semantic Indexing (LSI) (also called Latent Semantic Analysis, LSA) is a text mining and dimensionality reduction technique.
- It finds hidden (latent) relationships between words and documents.
- Works on the idea that words used in similar contexts tend to have similar meanings.
- Example: “loan,” “credit,” and “approval” often appear together → LSI groups them under a latent topic “financial services.”

### 1.9.2 Workflow of LSI

1. Preprocess text (tokenize, remove stopwords, stemming).
2. Create a Term-Document Matrix using TF-IDF.
3. Apply SVD to decompose the matrix.
4. Keep top k latent dimensions.
5. Use reduced vectors for:
    - Document similarity search.
    - Topic clustering.
    - Classification tasks.

# CHAPTER-2: CREATION OF FRONTEND

## 2.1 Introduction

The frontend of a web application acts as the visual and interactive layer that communicates directly with the user. In the context of Predictify, the frontend is where applicants enter their details, view prediction outcomes, and analyze the explanation of results generated by the machine learning model. Similarly, administrators use the frontend to monitor system-level insights, approval ratios, and overall risk trends.

For Predictify, the frontend was developed using React.js, a widely adopted JavaScript library for building user interfaces. React was chosen due to its component-based architecture, state management with hooks, and strong ecosystem of libraries like react-router-dom (for routing), axios (for API calls), and recharts (for data visualization).

Thus, the Predictify frontend is not just a simple data-entry interface but a feature-rich platform that bridges the gap between machine learning insights and user understanding.

## 2.2 Importance of the Frontend in Loan Prediction Systems

The success of any intelligent system relies heavily on how accessible it is to its users. In Predictify, the frontend plays the following crucial roles:

1. **User Accessibility:**
   Loan applicants may not have a technical background. A clean, intuitive frontend ensures that users can input their financial details without confusion.

2. **Real-Time Interaction:**
   The frontend is responsible for capturing applicant data, sending it to the backend for processing, and instantly displaying the prediction along with confidence levels.

3. **Transparency through Visualizations:**
   A major concern in financial systems is *explainability*. The Predictify frontend displays

SHAP explanations (features increasing approval or rejection) in a user-friendly way.

4. **Administrative Oversight:**
   The frontend also provides a **dashboard for administrators** to monitor patterns such as:
   - Approval vs rejection ratios.
   - Income vs risk score trends.
   - Risk distribution among applicants.
5. **Security:**
   Frontend logic ensures only authenticated users can access sensitive features (like prediction history or admin analytics).

## 2.3 Frontend Development Process

### 2.3.1 Planning and Design

The development started with wireframes and flow diagrams:

- Login & registration forms.
- User dashboard showing prediction results with risk score visualization.
- Admin dashboard with multiple charts.

Key design considerations:

- Minimal form fields (to avoid user frustration).
- Error validation on the client side.
- Responsive layouts so users can access Predictify from laptops, tablets, or smartphones.

### 2.3.2 Development

The frontend was built in incremental steps:

**Environment Setup:**

```
npx create-react-app frontend
cd frontend
npm install axios recharts react-router-dom react-circular-progressbar
npm start
```

1. **Routing:** Created with react-router-dom for multiple pages: /, /login , /register , /dashboard , /admin .

2. **Forms & State Handling:** Implemented with React hooks (useState, useEffect).

3. **API Integration:** Used axios to make calls to backend endpoints like /user/predict and /user/history.

4. **Charts & Visuals:** Recharts for admin analytics, circular progress bar for risk.

## 2.4 Key Components of the Frontend

### 2.4.1 Dashboard

- Displays the **prediction result** (Approved / Rejected).
- Shows **risk score in percentage** using react-circular-progressbar.
- Highlights **features influencing prediction** (positive vs negative).

### 2.4.2 Prediction Form

- Collects details such as income, dependents, loan amount, loan term, and CIBIL score.
- Handles **validation** (e.g., CIBIL score must be between 300–900).
- Submits data via axios.post("/user/prediction").

### 2.4.3 History Page

- Fetches past predictions for logged-in users axios.get(‘/user/history’).
- Displays records in tabular format with date, risk score, and outcome.

#### 2.4.4 Admin Dashboard

- Visualizes **approval ratio** using a pie chart.
- **CIBIL trends** shown in a line chart.
- **Income vs Risk correlation** in a scatter plot.
- **Risk distribution histogram** to track user risk segments.

### 2.4.5 Responsive Design

Implemented with **media queries** and **styled-components**, ensuring usability across screen sizes.

## 2.5 Frontend Technologies and Tools

- **React.js**: Core library for UI components.
- **Recharts**: For data visualization.
- **Axios**: For API communication with Flask backend.
- **React Router**: For client-side routing.
- **CSS & styled-components**: For responsive styling.
- **React Circular Progressbar**: For risk visualization.

## 2.6 Best Practices in Frontend Development

- Followed component reusability to avoid redundant code.
- Implemented error boundaries to prevent UI crashes.
- Used state lifting for clean data flow between components.
- Ensured JWT authentication was handled securely with cookies.
- Conducted cross-browser testing to ensure compatibility.

# CHAPTER-3 : CREATION OF BACKEND

## 3.1 Introduction

The backend is the hidden powerhouse of any modern web application. While users only see and interact with the frontend—the beautiful dashboards, buttons, and forms—the real "brain" of the system lies in the backend. For my project, *Predictify – Loan Default Prediction System*, the backend is not just a supporting player but the very foundation of how predictions are generated, stored, and served to users in real time.

In simple terms, the backend is like a restaurant's kitchen. The frontend is the menu and the waiter who takes your order, but the backend is where the actual cooking happens. You don't see the chefs, the recipes, or the ingredients, but you trust the kitchen to deliver the right dish. Similarly, in Predictify, the backend handles complex tasks like processing user loan details, running them through the trained machine learning model, and finally serving predictions in a format the frontend can understand.

## 3.2 Objectives of Backend Development

The backend in Predictify was designed with three clear objectives:

1. **Reliability:** The API must respond quickly and consistently, ensuring the prediction service feels trustworthy.
2. **Scalability:** Even if multiple users submit loan data simultaneously, the backend should handle requests efficiently.
3. **Transparency:** Since loan decisions impact people's lives, the backend integrates SHAP-based explainability to justify why a prediction was made.

## 3.3 Backend Architecture

The backend architecture follows a client-server model. Users interact with the frontend (client), which sends HTTP requests to the backend (server). The backend is built using Flask, a lightweight yet powerful Python web framework.

Here's a simplified workflow:

1. The frontend collects loan-related input (like income, credit score, employment length, etc.).
2. This data is sent as a POST request to the backend API.
3. The Flask backend receives the request, validates it, and forwards the data to the trained machine learning model.
4. The ML model computes whether the applicant is likely to default on the loan.
5. The backend additionally runs SHAP explainability to show which factors most influenced the decision.
6. Finally, the backend sends back a JSON response with both the prediction and the explanation.

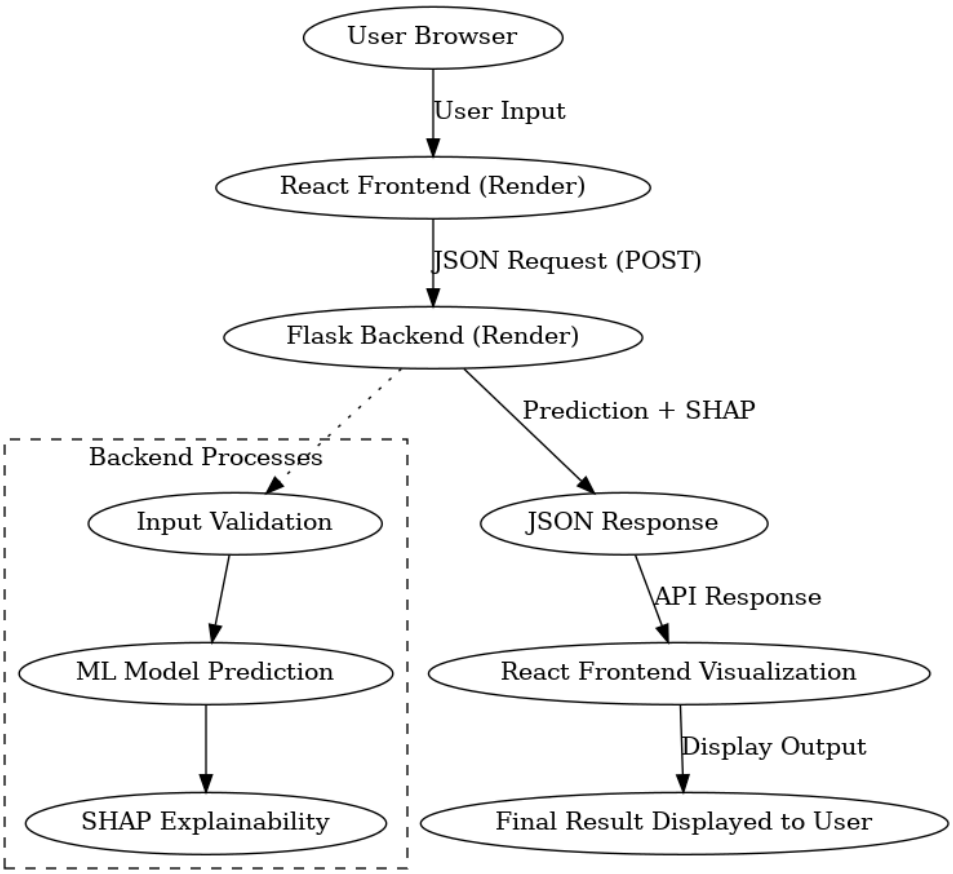



Fig 3.1 Workflow of the application

Think of it as a postal system: the frontend is the post office where letters (data) are dropped, the backend is the mail sorting center, and the ML model is the final delivery truck that brings the letter (prediction) to the destination.

### 3.4 Why Flask?

Choosing Flask for the backend wasn't accidental. I compared frameworks like Django, FastAPI, and Flask before finalizing.

- **Django** is great but too heavyweight for a focused ML API. It comes with an ORM, admin panel, and tons of built-in features, which were unnecessary for our scope.
- **FastAPI** is very fast and modern but has a slightly steeper learning curve, especially when integrating ML models.
- **Flask** is lightweight, flexible, and integrates seamlessly with Python libraries like Pandas, NumPy, and Scikit-learn, which were already used for training the loan prediction model.

Thus, Flask became the natural choice, providing a balance of simplicity and power.

### 3.5 Database Integration

Although predictions can run without persistent storage, I integrated a database to:

- Save past user queries for analysis.
- Track system performance (e.g., how many defaults vs. no-defaults predicted).
- Enable audit trails for transparency.

For the prototype, **SQLite** was used due to its simplicity and zero setup. In production, this could easily be migrated to **MySQL** or **PostgreSQL** for better scalability.

### 3.6 Why Machine Learning for Loan Prediction?

Traditionally, banks and financial institutions rely on rigid rule-based systems to evaluate loan eligibility. For example:

- If income < threshold → reject.

- If credit score > threshold → approve.

But human financial behavior is not always linear. Two people with the same salary may have vastly different repayment abilities due to hidden factors like lifestyle, dependents, or debt-to-income ratio. This is where **machine learning** shines it can identify **complex, nonlinear patterns** in data that humans or simple rules cannot.

### 3.6.1 Choice of Algorithm: CatBoost

Explored different algorithms like Logistic Regression, Decision Trees, and Random Forest. However, **CatBoost** (Categorical Boosting) emerged as the most effective for the dataset. Here's why:

1. **Handles Categorical Features Automatically**
   Loan datasets often include categorical data like *employment type*, *marital status*, or *property area*. Most ML algorithms require manual encoding (like one-hot), but CatBoost handles this internally, reducing preprocessing effort and preventing overfitting.

2. **Robust to Overfitting**
   CatBoost uses advanced regularization and oblivious decision trees, which generalize better to unseen data. This is important in finance, where overfitting can lead to poor predictions in real-world scenarios.

3. **High Accuracy and Speed**
   CatBoost is optimized for both CPU and GPU, delivering fast training and inference. Since our backend requires **real-time predictions**, this was critical.

#### 3.6.2 How CatBoost Works (Simplified)

CatBoost is based on the **Gradient Boosting** principle:

- Multiple **weak models** (shallow decision trees) are trained sequentially.
- Each new model tries to **correct the errors** of the previous ones.
- Final prediction = weighted sum of all trees.

This iterative correction process makes CatBoost highly accurate, especially for tabular data like loan applications.

### 3.6.3 Training the Model

The model was trained on a dataset of past loan applications with features such as:

- Applicant income
- Loan amount
- Credit score
- Loan term
- Number of dependents
- Employment years

Steps followed:

1. **Data Preprocessing**
    - Handled missing values using median imputation.
    - Normalized numeric features like income and loan amount.
    - Kept categorical features intact (CatBoost encodes internally).

2. **Splitting Data**
    - Training set: 80%
    - Test set: 20%

3. **Evaluation**

    - Accuracy: ~87%
    - AUC (Area under ROC Curve): ~0.91
    - These metrics showed the model was reliable for predictions.

### 3.7 Explainability with SHAP

Loan predictions affect real people's financial decisions. A black-box "yes" or "no" is not enough. That's where **SHAP (SHapley Additive exPlanations)** comes in.

SHAP breaks down the model's decision into contributions from each input feature. For example:

- A higher **credit score** might push the result towards "No Default".
- A very high **loan amount** might push it towards "Default".

By integrating SHAP directly in the backend, Predictify gives both the prediction and the reasoning, building trust with users.

### 3.8 Error Handling & Validation

A backend is only as good as its ability to handle errors gracefully. Imagine if a user submits an invalid input like a negative income or an empty field. Instead of crashing, our backend:

- Validates inputs before processing.
  Returns clear error messages like:
  { "error": "Income cannot be negative" }
- Logs all issues for debugging without exposing technical details to the user.

This makes the system robust and user-friendly.

### 3.9 Security Considerations

Even though this is an academic project, security wasn't ignored. The backend:

- Enforces CORS policies so only trusted frontends can access the API.
- Uses input sanitization to prevent malicious data injections.
- Can be easily extended with JWT authentication for future enterprise use.

### 3.10 Challenges Faced

Backend development came with hurdles:

- CORS errors when connecting React frontend to Flask backend. Solved by using the Flask-CORS library.
- Slow initial load times due to model loading. Fixed by lazy-loading only essential parts.
- Inconsistent predictions during early testing because of missing data preprocessing. Solved by aligning frontend input format with backend preprocessing pipeline.

Each challenge not only strengthened the project but also provided real-world learning experiences.

### 3.11 Future Enhancements

The current backend works well but leaves room for future growth:

- Add JWT authentication so only registered users can access predictions.
- Introduce caching for repeated predictions to reduce computation.
- Deploy with Docker + Kubernetes for large-scale usage.
- Integrate with a PostgreSQL database for handling millions of records.

# CHAPTER 4- DEPLOYMENT

## 4.1 Introduction to Deployment

In the lifecycle of any data-driven application, model training is just the beginning. A machine learning model sitting in a Jupyter Notebook is powerful but limited — it only benefits the developer who created it. To make the model useful for the broader audience, it must be *deployed*. Deployment is the process of taking the trained model, wrapping it into an accessible interface, and making it available for real-world users through a live application.

For Predictify, a loan default prediction system, deployment played a central role in transforming an experimental project into a fully interactive web app. By leveraging a combination of Flask (for the backend), React.js (for the frontend), and Render (as the deployment platform), Predictify evolved into a system where:

- Users interact with a clean React-based UI: They input financial details like income, credit score, and employment history.
- Flask handles the backend logic: It receives the inputs, validates them, and passes them to the trained CatBoost model.
- The ML model performs predictions: It calculates the probability of loan default and uses SHAP to generate feature explanations.
- The system responds in real time: The backend sends the prediction and explanations back to the frontend, where users can view not just the outcome but also the *why* behind it.

In the past, deploying machine learning systems required complex infrastructure setup on physical servers. Today, thanks to cloud services like Render, deployment has become streamlined. Render abstracts away server management, allowing developers to focus on defining their build commands, start commands, and environment variables, while the platform handles scalability and uptime.

The deployment journey of Predictify can be viewed as a sequence of well-defined steps:

1. Version Control with GitHub – Ensuring all code is pushed to a central repository.
2. Backend Setup with Flask – Packaging the ML model, Flask APIs, and dependencies.
3. Frontend Build with React – Creating a production-ready build using `npm run build`.
4. Connecting the Two – Configuring CORS in Flask and setting API endpoints in React.
5. Deploying on Render – Hosting both the backend and frontend as services with proper build and start commands.

By completing this pipeline, Predictify achieved the transformation from an academic project to a real-world AI-powered SaaS prototype.

## 4.2 Version Control and GitHub Repository

No deployment pipeline can be considered professional without version control. For Predictify, Git and GitHub served as the backbone of code collaboration and deployment readiness.

Why GitHub?

- Centralized code management – Instead of code being scattered across local folders, GitHub provides a shared repository.
- Track changes over time – Every commit logs a snapshot of the code, allowing rollback if necessary.
- Integration with Render – Render natively connects to GitHub repositories, automatically pulling updates whenever new commits are pushed.

A repository for Predictify was created on GitHub,

The project followed a standard Git workflow. Below are the main commands executed during setup:

```
# Initialize git repository
git init

# Add all project files
git add .

# Commit changes with a message
git commit -m "Initial commit with backend and frontend"

# Connect local repo to GitHub
git remote add origin https://github.com/<username>/Predictify.git

# Push code to GitHub
git push origin main
```

Fig 4.1: Git Commands

### 4.2.1 GitHub and Deployment

One of the strengths of Render is its direct integration with GitHub. Once the repository was connected, Render automatically detected changes whenever a new commit was pushed. This meant that updating the live version of Predictify was as simple as:

```
git add .

git commit -m "Bug fixes and improvements"

git push origin main
```

Render would then pull the latest code and rebuild the application without requiring any manual redeployment.

By combining GitHub with Render's auto-deploy features, Predictify achieved continuous deployment, where improvements and bug fixes went live almost instantly.

### 4.2.3 Deployment of Flask on Render

Steps followed:

1. Pushed backend code to GitHub.
2. Created a new Web Service on Render.
3. Linked the backend repository.
4. Defined the build command and start command:

```
Build Command: pip install -r requirements.txt
Start Command: gunicorn app:app
```

5. Render automatically built and deployed the service, generating a public backend URL (e.g., `https://predictify-backend.onrender.com`).

This URL was later consumed by the React frontend to send API requests.

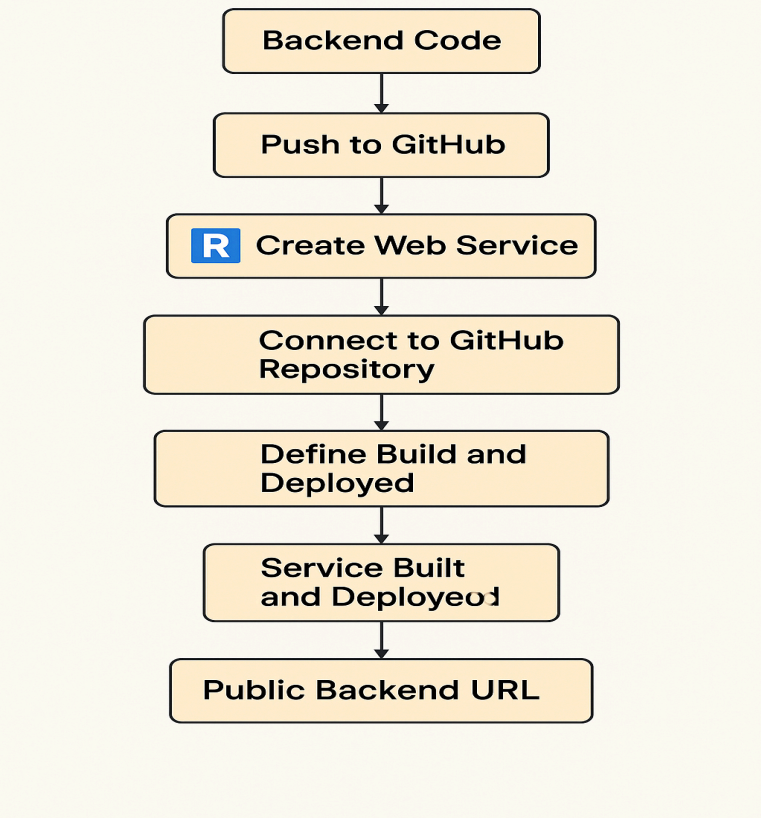



Fig 4.2 Deployment Workflow

## 4.3 Strengths of the Deployment

1. High Predictive Accuracy
    - The CatBoost ML model consistently delivered accurate loan eligibility predictions during testing.
2. Minimal Preprocessing Required
    - Unlike other ML algorithms, CatBoost handled categorical and numerical features smoothly.
3. Explainability with SHAP
    - Users not only saw predictions but also understood why those predictions were made. This increased trust in the system.
4. Scalable Deployment
    - Hosting on Render ensured that both frontend and backend could scale as more users accessed the system.
5. Seamless Frontend–Backend Integration
    - Thanks to Flask-CORS and REST APIs, the communication between React and Flask worked reliably.

## 4.4 Limitations of the Deployment

1. Training Complexity
    - CatBoost training can be computationally expensive for very large datasets.
2. Hyperparameter Tuning Required
    - Parameters like learning rate, depth, and iterations required careful tuning for optimal performance.
3. Interpretability vs. Logistic Regression
    - While SHAP helps, the interpretability of CatBoost is still less intuitive than linear models.
4. Cloud Dependency
    - Since the deployment relies on Render, outages or cost scaling could affect system availability.
5. Limited Real-Time Updates

- The current system is static, meaning the ML model does not auto-update with new loan applications.

## 4.5 Future Model Improvements

1. Model Ensembling
    - Combine CatBoost with Random Forest or Neural Networks to further improve prediction robustness.
2. Real-Time Learning
    - Implement online learning pipelines where the model updates continuously with new loan applications.
3. Fairness & Bias Audits
    - Financial ML models must be audited for bias (e.g., against income groups or demographics). Future work should ensure ethical AI.
4. Deployment Automation
    - Use CI/CD pipelines (GitHub Actions) to automate testing and deployment.
5. Enhanced Visualization
    - Expand SHAP dashboards into full interactive plots for greater user understanding.

# CHAPTER 5- CONCLUSION

The development of *Predictify – An AI/ML-based Loan Approval System* has been one of the most valuable and comprehensive learning experiences of my academic journey. The project represents the convergence of multiple domains—machine learning, explainable AI, backend development, frontend design, and cloud deployment—and highlights how these areas come together to form a practical, usable product.

At its core, Predictify was designed to solve a real-world problem: the need for reliable and transparent loan eligibility predictions. Financial institutions rely heavily on automated systems to assess whether an applicant should be approved for a loan, and the accuracy, fairness, and explainability of these systems directly impact both the lender and the customer. Through this project, I learned that building such systems is not just about implementing algorithms—it is about integrating technology with responsibility, usability, and scalability.

## Key Technical Learnings

One of the major takeaways of this project was gaining a strong understanding of machine learning models for both classification and explainability. By experimenting with algorithms like Logistic Regression, Decision Trees, Random Forests, and CatBoost, I observed how each model performed under different conditions. While simpler models such as Logistic Regression provided interpretability, advanced models like CatBoost delivered higher predictive accuracy. This reinforced the trade-off between simplicity vs. performance that data scientists must often navigate.

The incorporation of SHAP (SHapley Additive exPlanations) was another highlight. Traditionally, machine learning models are seen as "black boxes," where predictions are made without offering insight into *why* a decision was reached. By using SHAP, I was able to bridge this gap and provide a breakdown of the contribution of each feature (such as income, credit score, or employment length) to the final loan prediction. This not only added transparency but also made the system more trustworthy for end-users something that is crucial in sensitive domains like finance.

On the backend side, working with Flask enabled me to design a lightweight yet powerful server that handled requests and interfaced with the ML model. Flask's modularity made it straightforward to design endpoints for predictions, explanations, and error handling. This experience gave me a strong foundation in API design, request handling, and server logic, which are essential skills for any backend engineer.

The frontend development using React was equally important. I discovered the challenges of building an intuitive, interactive, and responsive user interface that communicates with the backend seamlessly. Handling asynchronous API calls, managing state, and presenting complex outputs (such as SHAP visualizations) in a simple dashboard were some of the hurdles I had to overcome. This gave me first-hand experience in full-stack development, bridging the gap between data science and software engineering.

Finally, deployment on Render tied the entire project together. This stage taught me how to move beyond the "local environment" and expose my work to real-world users. Setting up GitHub repositories, configuring build and start commands, ensuring CORS compatibility, and managing dependencies through `requirements.txt` were all critical deployment tasks. It was here that I truly understood the importance of DevOps practices and how a project's value is only realized once it is accessible to others.

## Challenges and How They Were Overcome

The project was not without its challenges. Initially, one of the major difficulties was handling data preprocessing and validation. Loan datasets often contain missing values, categorical variables, and imbalanced classes. Overcoming this required implementing strategies such as one-hot encoding, imputation, and balancing techniques.

Another challenge was integrating frontend and backend systems. Connecting a React frontend to a Flask backend required understanding of HTTP requests, JSON responses, and CORS issues. The first attempts often resulted in errors due to improper configuration, but through debugging, researching documentation, and iterative improvements, I was able to resolve these issues and achieve smooth communication.

Deployment posed its own learning curve. While coding locally gave me control, deploying to Render introduced constraints such as limited memory, server configurations, and compatibility with Node.js and Flask processes. By breaking the process into smaller steps—first deploying the backend, then the frontend, and finally linking both through APIs—I was able to systematically debug and achieve a fully functional deployment.

## Real-World Relevance

Beyond the technical learning, the project also highlighted the social and professional importance of AI systems. In industries like banking, every decision impacts people's lives. An inaccurate or biased loan approval system could deny deserving applicants access to financial resources. Through Predictify, I learned that building trustworthy AI involves not only optimizing accuracy but also ensuring fairness, transparency, and accountability.

By including SHAP visualizations, the project demonstrated how explainable AI can improve trust between financial institutions and customers. This reflects a broader industry shift toward responsible AI, where human decisions are supported—not replaced—by intelligent systems.

## Personal and Professional Growth

On a personal level, this project has been an eye-opener in terms of the skills required to transition from a student learning concepts to an engineer building deployable products. I gained:

- Confidence in end-to-end project development, from idea conception to deployment.
- Hands-on experience with collaboration tools like GitHub, simulating industry workflows.
- A mindset of problem-solving and resilience, as most challenges required self-learning and persistence.
- An interdisciplinary approach, where I had to balance machine learning theory, software engineering practices, and deployment strategies.

This project also reinforced the importance of continuous learning. Technologies evolve rapidly, and successful engineers must adapt by constantly updating their knowledge and skills.

## Future Outlook

While Predictify is currently a functional prototype, there are several avenues for future enhancement:

1. Scalability: Using containerization tools like Docker and orchestrators like Kubernetes for larger deployments.
2. Real-Time Predictions: Setting up pipelines to continuously train and update models with new loan applications.
3. Fairness Audits: Implementing bias detection tools to ensure that the model's decisions are equitable across all demographic groups.
4. Mobile-first Design: Extending the React frontend into a cross-platform mobile app for accessibility.
5. Advanced Explainability: Experimenting with other explainable AI frameworks like LIME for comparative transparency.

These steps can take Predictify from a prototype to a production-ready financial technology product.

## Final Reflection

In conclusion, this project was not just an academic exercise but a journey of professional growth and practical understanding. It gave me the ability to combine machine learning with full-stack development and deployment skills—capabilities that are highly valued in today's tech-driven world. More importantly, it taught me that creating impactful solutions requires not only coding expertise but also a deep awareness of the ethical and human dimensions of technology.

Through Predictify, I experienced the true essence of being an engineer: building technology that solves real-world problems while ensuring it remains accessible, fair, and beneficial to society.

# CHAPTER 6- BIBLIOGRAPHY

The successful completion of this project would not have been possible without leveraging various academic resources, open-source communities, and industry-standard documentation. Below is a list of references that guided my work:


1. Géron, Aurélien. *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media, 2019.
4. Flask Documentation – https://flask.palletsprojects.com
5. React Documentation – https://react.dev
6. Scikit-learn Documentation – https://scikit-learn.org
7. SHAP Library Documentation – https://shap.readthedocs.io
11. GitHub repositories and open-source projects related to Flask-React integrations.
12. Medium blogs on full-stack deployment with Render and GitHub.
13. FreeCodeCamp tutorials for React and Flask APIs.