

# Loan Prediction Model

**Jyotiska Saha(MST03-0056)**

August 20, 2024

Submitted to Meta Scifor Technologies



Meta  
Scifor Technologies

**Script. Sculpt. Socialize**

**UNDER GUIDIANCE OF  
Urooj Khan**

## **Abstract**

This report details the development and evaluation of a machine learning model designed to predict loan approval status. The primary objective of this project was to construct a predictive model capable of determining whether a loan application would be approved based on a set of features related to the applicant and their financial history. The report encompasses the entire process, from data preprocessing and exploratory data analysis to model selection and hyperparameter tuning.

Data preprocessing involved handling missing values, capping outliers, and encoding categorical features to prepare the data for analysis. Exploratory data analysis was conducted to understand the underlying patterns in the data through various visualizations.

Multiple machine learning algorithms were employed. Each model was optimized using hyperparameter tuning with GridSearch to enhance performance. The models were rigorously evaluated using accuracy metrics to assess their effectiveness in predicting loan status.

In addition to model development, the project included deploying the predictive model using Streamlit. This deployment provides an interactive web application where users can input their data to check the status of their loan application. The Streamlit app also offers functionalities for visualizing model training results and diagrams, facilitating a deeper understanding of the model's performance and the data it operates on. This deployment enhances user experience by making the model accessible and understandable through a user-friendly interface.

# Contents

<b>1</b>	<b>Introduction</b>	<b>3</b>
<b>2</b>	<b>Data Description</b>	<b>4</b>
<b>3</b>	<b>Classification Algorithms</b>	<b>5</b>
3.1	Logistic Regression . . . . .	5
3.1.1	Sigmoid Function . . . . .	5
3.1.2	Cost Function . . . . .	5
3.1.3	Log Loss . . . . .	6
3.2	Linear Support Vector Classification (SVC) . . . . .	6
3.2.1	Hard Margin vs. Soft Margin . . . . .	6
3.2.2	Hyperparameter $C$ . . . . .	6
3.2.3	Algorithmic Details . . . . .	7
3.3	Naive Bayes Classifier (NBC) . . . . .	7
3.3.1	Bayes' Theorem . . . . .	7
3.3.2	Naive Bayes Classification . . . . .	7
3.3.3	Gaussian Naive Bayes . . . . .	7
3.4	Decision Tree Classifier . . . . .	8
3.5	Random Forest Classifier . . . . .	8
3.6	AdaBoost Classifier . . . . .	8
3.7	Gradient Boosting Classifier . . . . .	9
3.8	K-Nearest Neighbors (KNN) Classifier . . . . .	9
<b>4</b>	<b>Data Preprocessing</b>	<b>10</b>
4.1	Handling Missing Values . . . . .	10
4.2	Outlier Detection . . . . .	10
4.3	Categorical Encoding . . . . .	10
<b>5</b>	<b>Exploratory Data Analysis</b>	<b>11</b>
5.1	Visualization . . . . .	11
<b>6</b>	<b>Modeling</b>	<b>13</b>
6.1	Feature Scaling . . . . .	13
6.2	Model Selection . . . . .	13
6.3	Hyperparameter Tuning . . . . .	13
6.4	Best Models . . . . .	14

<b>7</b>	<b>Results</b>	<b>15</b>
7.1	Model Performance . . . . .	15
7.1.1	Logistic Regression . . . . .	15
7.1.2	Linear SVC . . . . .	15
7.1.3	Naive Bayes Classifier (NBC) . . . . .	15
7.1.4	Decision Tree Classifier . . . . .	15
7.1.5	Random Forest Classifier . . . . .	16
7.1.6	AdaBoost Classifier . . . . .	16
7.1.7	Gradient Boosting Classifier . . . . .	16
7.1.8	K-Nearest Neighbors (KNN) Classifier . . . . .	16
<b>8</b>	<b>Conclusion</b>	<b>17</b>

# Chapter 1

## Introduction

The objective of this project is to develop a machine learning model that predicts the status of loan applications. By leveraging various features related to the applicants and their financial histories, the model aims to provide accurate predictions regarding loan approval.

Accurate loan prediction is critical for financial institutions, as it enhances decision-making processes by providing a data-driven basis for approving or rejecting loan applications. This model seeks to automate and improve the efficiency of loan assessments, minimizing manual reviews and reducing the potential for human error.

The project includes key phases such as data preprocessing, exploratory data analysis, model selection, and hyperparameter tuning. Data preprocessing involves handling missing values, outlier capping, and encoding categorical features. Exploratory data analysis uses visualizations to uncover patterns in the data. Multiple machine learning algorithms are evaluated and optimized to identify the best-performing model.

Additionally, the model is deployed using Streamlit, which provides an interactive web application for users. This application allows users to input their own data, check the status of their loan applications, and visualize various aspects of model performance and training results, making the tool both practical and user-friendly.

# Chapter 2

## Data Description

The dataset used in this project includes both numeric and categorical features:

- Categorical Features:  
Loan\_ID, Gender, Married, Dependents, Education, Self\_Employed, Property\_Area, Credit\_History
- Numeric Features:  
ApplicantIncome, CoapplicantIncome, LoanAmount, Loan\_Amount\_Term













No.	Attribute	Description	Data type
1	LoanID	 Unique Loan ID	Categorical
2	Gender	 Gender (Male/Female)	Categorical
3	Married	 Married (Y/N)	Categorical
4	Dependents	 Dependents(0/1/2/3+)	Categorical
5	Education	 Education (Graduate/Not Graduate)	Categorical
6	SelfEmployed	 Self-employed (Y/N)	Categorical
7	ApplicantIncome	 Applicant's Income	Numeric/Integer
8	CoapplicantIncome	 Coapplicant's Income	Numeric/float
9	LoanAmount	 Loan Amount (in thousands)	Numeric/float
10	LoanAmountTerm	 Loan Term (in months)	Numeric/float
11	CreditHistory	 Credit History(0/1)	Categorical
12	PropertyArea	 Property Area (Urban/Semi-Urban/Rural)	Categorical
13	LoanStatus	 Loan Status (Approved/Not Approved)	Categorical

Figure 2.1: Overview of dataset features

# Chapter 3

## Classification Algorithms

This is an overview of the classification algorithms used in this project. Each algorithm was employed to predict loan approval status, and their respective characteristics and functionalities are detailed below.

### 3.1 Logistic Regression

Logistic Regression is a widely used statistical model for binary and multi-class classification problems. It estimates the probability of a given outcome based on one or more predictor variables. The core idea is to model the relationship between the dependent variable (e.g., loan approval status) and one or more independent variables (e.g., applicant income, credit history) using a logistic function.

#### 3.1.1 Sigmoid Function

At the heart of Logistic Regression is the sigmoid function, which transforms the output of a linear equation into a probability value between 0 and 1. The sigmoid function is defined as:

$$\sigma(z) = \frac{1}{1 + e^{-z}} \quad (3.1)$$

where  $z$  is the linear combination of the input features. This function ensures that the output of the logistic regression model can be interpreted as a probability.

#### 3.1.2 Cost Function

Logistic Regression uses a cost function to evaluate the performance of the model. The cost function measures the difference between the predicted probabilities and the actual outcomes. For binary classification, the cost function is defined as:

$$J(\theta) = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log(h_{\theta}(x^{(i)})) + (1 - y^{(i)}) \log(1 - h_{\theta}(x^{(i)}))] \quad (3.2)$$

where  $h_{\theta}(x)$  represents the hypothesis function (i.e., the sigmoid function applied to the linear combination of features),  $y^{(i)}$  is the actual label, and  $m$  is the number of training examples. This cost function is also known as log loss or binary cross-entropy.

### 3.1.3 Log Loss

Log loss, or logistic loss, quantifies the accuracy of a classification model's predictions. It is used to compute the cost function in Logistic Regression. Log loss is defined as:

$$\text{Log Loss} = -\frac{1}{m} \sum_{i=1}^m [y^{(i)} \log(p^{(i)}) + (1 - y^{(i)}) \log(1 - p^{(i)})] \quad (3.3)$$

where  $p^{(i)}$  is the predicted probability for class 1. Lower log loss values indicate better model performance, as they reflect closer agreement between predicted probabilities and actual outcomes.

Logistic Regression is particularly useful for cases where the relationship between the dependent and independent variables is linear. It provides a probabilistic interpretation of predictions, making it suitable for binary classification tasks such as predicting loan approval status. The model's parameters are typically estimated using maximum likelihood estimation, and optimization algorithms like gradient descent are used to minimize the cost function.

## 3.2 Linear Support Vector Classification (SVC)

Linear Support Vector Classification (SVC) aims to find the optimal hyperplane that maximizes the margin between two classes. The margin is the distance between the hyperplane and the closest data points from each class, known as support vectors.

### 3.2.1 Hard Margin vs. Soft Margin

#### Hard Margin

For perfectly separable data, a hard margin approach finds a hyperplane that separates the classes without any misclassification. It assumes noise-free data, which is rare in real-world scenarios.

#### Soft Margin

When data is noisy or not perfectly separable, the soft margin approach allows some misclassification by introducing a penalty for errors. The hyperparameter  $C$  controls the trade-off between maximizing the margin and minimizing classification errors.

### 3.2.2 Hyperparameter $C$

- **High  $C$ :** Focuses on minimizing training errors, which may lead to a smaller margin and potential overfitting.
- **Low  $C$ :** Allows more misclassifications to achieve a larger margin, which may lead to underfitting.



### 3.2.3 Algorithmic Details

Linear SVC constructs a hyperplane to separate classes. For linearly separable data, it maximizes the margin. For non-linearly separable data, kernel functions can be used, though this is not the focus of linear SVC.

Linear SVC is effective for high-dimensional data and linearly separable classes. Tuning the hyperparameter  $C$  through cross-validation is essential for optimizing model performance.

## 3.3 Naive Bayes Classifier (NBC)

The Naive Bayes Classifier is based on Bayes' Theorem and assumes independence among features given the class label. This classifier calculates the probability of a data point belonging to a particular class and assigns the class with the highest probability. Despite its simplicity, Naive Bayes is effective and efficient, particularly with large datasets.

### 3.3.1 Bayes' Theorem

Bayes' Theorem is used to update the probability of a class based on the observed features:

$$P(C_k | \mathbf{x}) = \frac{P(C_k) \cdot P(\mathbf{x} | C_k)}{P(\mathbf{x})} \quad (3.4)$$

where  $P(C_k | \mathbf{x})$  is the posterior probability of class  $C_k$  given features  $\mathbf{x}$ .

### 3.3.2 Naive Bayes Classification

The likelihood  $P(\mathbf{x} | C_k)$  is computed as:

$$P(\mathbf{x} | C_k) = \prod_{i=1}^n P(x_i | C_k) \quad (3.5)$$

where  $x_i$  represents the individual features.

### 3.3.3 Gaussian Naive Bayes

For features assumed to follow a Gaussian distribution, the likelihood is modeled as:

$$P(x_i | C_k) = \frac{1}{\sqrt{2\pi\sigma_k^2}} \exp\left(-\frac{(x_i - \mu_k)^2}{2\sigma_k^2}\right) \quad (3.6)$$

where  $\mu_k$  is the mean and  $\sigma_k^2$  is the variance of the feature for class  $C_k$ .

Naive Bayes is particularly useful for classification tasks where feature independence can be assumed and is well-suited for large datasets.

### 3.4 Decision Tree Classifier

The Decision Tree Classifier builds a tree-like model by recursively splitting the data based on feature values to create subsets that are as homogeneous as possible. Each internal node of the tree represents a decision based on a feature, each branch represents an outcome of that decision, and each leaf node represents a class label. Decision Trees are intuitive and easy to interpret, as they visually map out decision paths. They handle both numerical and categorical data effectively. However, Decision Trees can easily overfit, particularly if they are too deep, capturing noise in the training data rather than the underlying patterns.

To mitigate overfitting, techniques such as pruning (reducing the size of the tree) and setting constraints like maximum depth or minimum samples per leaf are often applied. Decision Trees are also used as base learners in ensemble methods like Random Forests.

### 3.5 Random Forest Classifier

Random Forest is an ensemble learning method that aggregates the predictions of multiple decision trees to improve classification performance. It constructs a "forest" of decision trees, each trained on a different subset of the training data (via bootstrapping) and using a random subset of features for splitting at each node. This randomness helps ensure that the individual trees in the forest are diverse, which enhances the overall model's robustness and generalization.

By averaging the predictions (for regression) or using majority voting (for classification) from all the trees, Random Forest reduces the risk of overfitting and improves accuracy compared to a single decision tree. It is effective in handling large datasets, high-dimensional spaces, and provides feature importance scores, which can be useful for understanding the contribution of different features to the model.

Despite its advantages, Random Forest can be computationally intensive and less interpretable compared to individual decision trees.

### 3.6 AdaBoost Classifier

AdaBoost (Adaptive Boosting) is an ensemble method that improves the performance of weak classifiers by focusing on the errors of previous classifiers. The algorithm works as follows:

- Train a weak classifier on the original dataset.
- Increase the weights of misclassified instances so that subsequent classifiers focus more on these difficult cases.
- Combine the weak classifiers into a strong classifier through a weighted majority vote, where each classifier's vote is weighted by its accuracy.

AdaBoost enhances the predictive performance of weak models by iteratively correcting errors and adjusting weights.

## 3.7 Gradient Boosting Classifier

Gradient Boosting is an ensemble technique that builds models sequentially to correct errors made by previous models. The process involves:

- Train a base model (e.g., decision tree) on the data.
- Compute the residual errors (differences between actual and predicted values).
- Train a new model to predict these residual errors.
- Add the new model to the ensemble and update predictions.
- Repeat the process to iteratively improve model performance.

Gradient Boosting improves accuracy by focusing on the residuals of previous models, making it effective for complex datasets.

## 3.8 K-Nearest Neighbors (KNN) Classifier

The K-Nearest Neighbors (KNN) Classifier classifies a data point based on the majority class of its  $k$  nearest neighbors in the feature space. The algorithm follows these steps:

- Choose the number  $k$  of nearest neighbors.
- Compute the distance between the data point and all other points in the training set.
- Identify the  $k$  closest neighbors based on the distance metric (e.g., Euclidean distance).
- Assign the class label based on the majority class among these  $k$  neighbors.

KNN is simple and intuitive, suitable for smaller datasets, and handles multi-class classification effectively.

Each of these algorithms was applied to the loan prediction problem to determine the most effective approach. The performance of each model was evaluated and compared to select the best classifier for predicting loan approval status.

# Chapter 4

## Data Preprocessing

### 4.1 Handling Missing Values

The dataset initially contained missing values in both categorical and numerical columns. These missing values were handled using imputation. For categorical columns, the most frequent value was used for imputation, ensuring that the most common category filled in the gaps. For numerical columns, the median value of each feature was used for imputation. Although the mean could have been used, the median was preferred because it is less sensitive to outliers, which can distort the mean.

### 4.2 Outlier Detection

Outliers in the numeric columns were identified and capped to reduce their influence on the model. The  $1.5 \times \text{IQR}$  (Interquartile Range) method was employed to detect and cap outliers, ensuring that extreme values did not negatively affect the model's performance.

### 4.3 Categorical Encoding

Categorical features were transformed into numerical format using Label Encoding. This method assigns a unique integer to each category, making it suitable for algorithms that require numeric input.

# Chapter 5

## Exploratory Data Analysis

### 5.1 Visualization

The following visualizations were created to understand the data better:

- Bar plots for categorical features.
- Histograms and swarm plots for numeric features.
- Correlation coefficient matrix for numeric features.

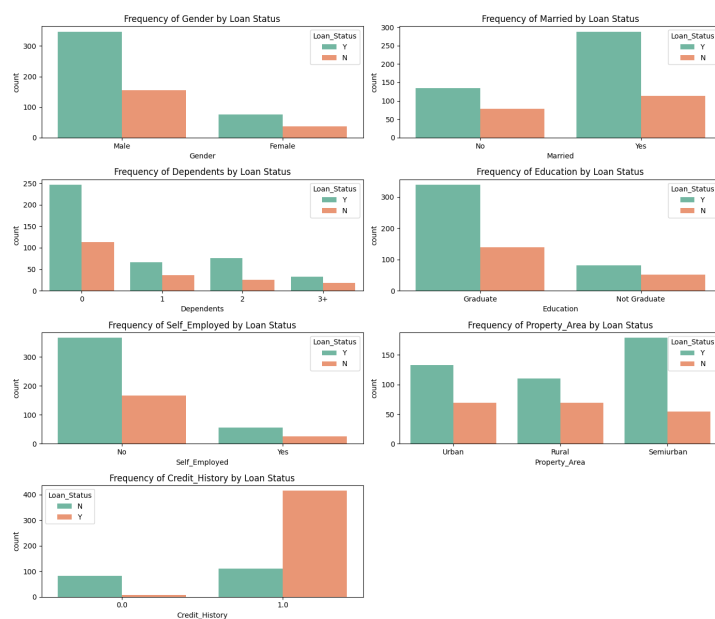


Figure 5.1: Bar plot of categorical features

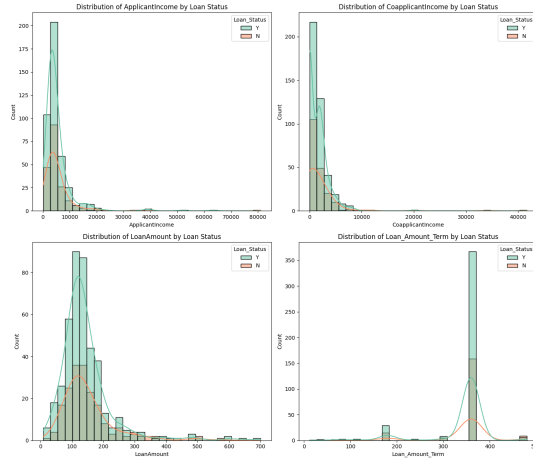


Figure 5.2: Histogram of numeric features

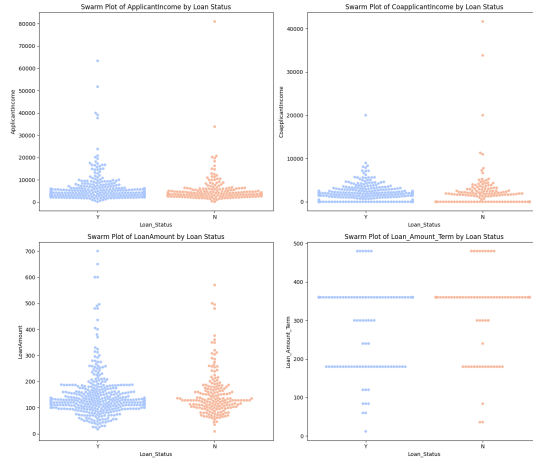


Figure 5.3: Swarm plot of numeric features

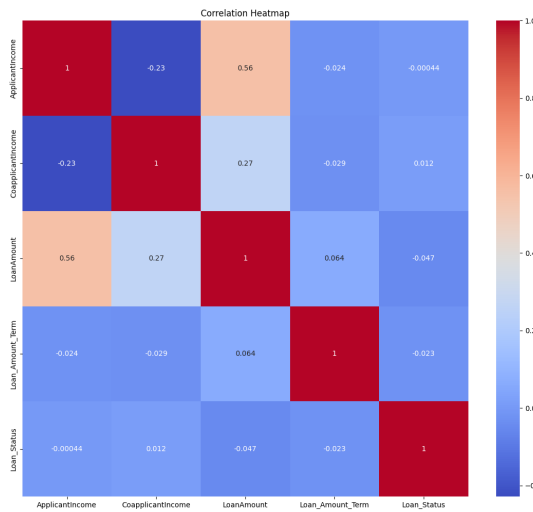


Figure 5.4: Correlation coefficient matrix

# Chapter 6

## Modeling

### 6.1 Feature Scaling

The numeric features were scaled using `StandardScaler` to ensure that all features contribute equally to the model.

### 6.2 Model Selection

Several classification algorithms were applied to the data:

- Logistic Regression
- Linear SVC
- Naive Bayes Classifier (NBC)
- Decision Tree Classifier
- Random Forest Classifier
- AdaBoost Classifier
- Gradient Boosting Classifier
- K-Nearest Neighbors (KNN) Classifier

### 6.3 Hyperparameter Tuning

To enhance the accuracy and reduce overfitting of the models, hyperparameter tuning was performed using `GridSearch`. This process allowed for the optimization of each model's performance by selecting the best combination of hyperparameters. All models, except for the Naive Bayes Classifier (NBC), underwent hyperparameter tuning, as NBC does not require hyperparameters in the same way that other models do.

## 6.4 Best Models

The following models were fine-tuned using GridSearch, and their best parameters were identified:

- **Logistic Regression:**
  - `C=0.1`
  - `max_iter=100`
  - `solver='lbfgs'`
- **Linear SVC:**
  - `C=0.1`
- **Decision Tree Classifier:**
  - `criterion='entropy'`
  - `max_depth=10`
  - `min_samples_split=2`
- **Random Forest Classifier:**
  - `max_depth=None`
  - `min_samples_split=5`
  - `n_estimators=200`
- **AdaBoost Classifier:**
  - `learning_rate=0.01`
  - `n_estimators=50`
- **Gradient Boosting Classifier:**
  - `learning_rate=0.01`
  - `max_depth=3`
  - `n_estimators=100`
- **K-Nearest Neighbors (KNN) Classifier:**
  - `metric='manhattan'`
  - `n_neighbors=7`
  - `weights='uniform'`



# Chapter 7

## Results

### 7.1 Model Performance

After training and tuning the models, their performances were evaluated using training accuracy, test accuracy, and confusion matrices on the testing dataset. Below are the results for the different models:

#### 7.1.1 Logistic Regression

- Accuracy: Training = 0.798, Test = 0.854

- Confusion Matrix:

$$\begin{bmatrix} 21 & 17 \\ 1 & 84 \end{bmatrix}$$

#### 7.1.2 Linear SVC

- Accuracy: Training = 0.798, Test = 0.854

- Confusion Matrix:

$$\begin{bmatrix} 21 & 17 \\ 1 & 84 \end{bmatrix}$$

#### 7.1.3 Naive Bayes Classifier (NBC)

- Accuracy: Training = 0.796, Test = 0.846

- Confusion Matrix:

$$\begin{bmatrix} 21 & 17 \\ 2 & 83 \end{bmatrix}$$

#### 7.1.4 Decision Tree Classifier

- Accuracy: Training = 0.945, Test = 0.732

- Confusion Matrix:

$$\begin{bmatrix} 24 & 14 \\ 19 & 66 \end{bmatrix}$$

### 7.1.5 Random Forest Classifier

- Accuracy: Training = 0.949, Test = 0.837
- Confusion Matrix:

$$\begin{bmatrix} 23 & 15 \\ 5 & 80 \end{bmatrix}$$

### 7.1.6 AdaBoost Classifier

- Accuracy: Training = 0.798, Test = 0.854
- Confusion Matrix:

$$\begin{bmatrix} 21 & 17 \\ 1 & 84 \end{bmatrix}$$

### 7.1.7 Gradient Boosting Classifier

- Accuracy: Training = 0.804, Test = 0.854
- Confusion Matrix:

$$\begin{bmatrix} 21 & 17 \\ 1 & 84 \end{bmatrix}$$

### 7.1.8 K-Nearest Neighbors (KNN) Classifier

- Accuracy: Training = 0.790, Test = 0.732
- Confusion Matrix:

$$\begin{bmatrix} 12 & 26 \\ 7 & 78 \end{bmatrix}$$

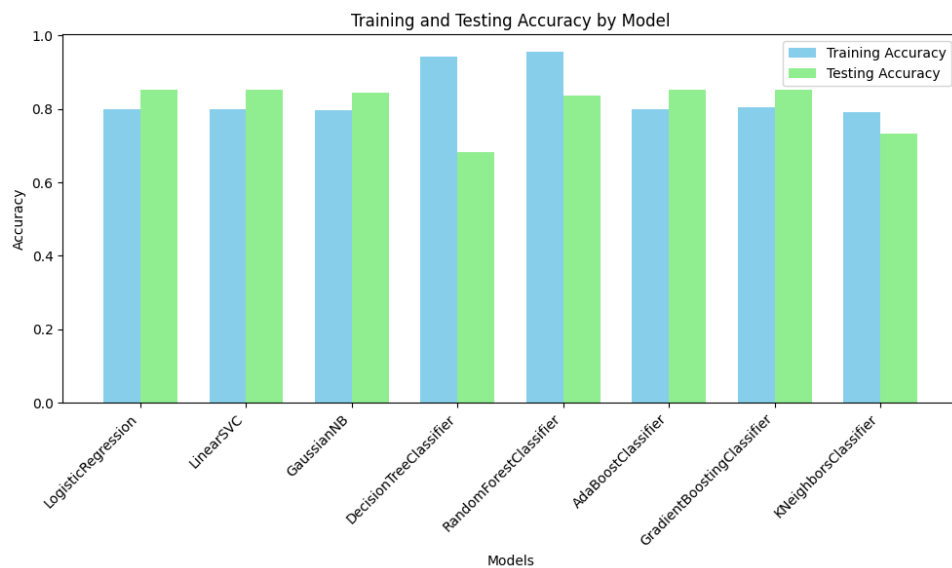


Figure 7.1: Swarm plot of numeric features

# Chapter 8

## Conclusion

In this project, various machine learning algorithms were applied to the loan prediction problem. After preprocessing and exploratory analysis, models were trained and evaluated. The final results show the effectiveness of different models and their ability to predict loan status with varying degrees of accuracy.

The most suitable models were deployed using Streamlit to create an interactive and user-friendly web application for loan prediction. This deployment allows users to input data and receive loan status predictions in real-time. The insights gained from this project can be used to improve loan application processes and decision-making in financial institutions.

Overall, this project demonstrates the potential of machine learning in automating and enhancing financial decision-making.

# Bibliography

- [1] Data source: *Loan Prediction Problem Dataset*. Available at: <https://www.kaggle.com/datasets/altruistdelhite04/loan-prediction-problem-dataset>
- [2] Aurélien Géron. (2019). *Hands-On Machine Learning with Scikit-Learn, Keras, and TensorFlow*. O'Reilly Media.
- [3] Streamlit App: *loan eligibility status prediction app link*. <https://loan-eligibility-status-prediction.streamlit.app/>