UNIVERSITY OF ABOMEY-CALAVI (UAC)

****

DOCTORAL SCHOOL OF AGRONOMIC AND WATER SCIENCE

****

LABORATOIRE DE BIOMATHÉMATIQUES ET D'ESTIMATIONS FORESTIÈRES

Understanding Decision Trees in R: From Theory and Data Preparation to Packages and Real-World Applications

Contributors

DAGNAW Belege Worku
MEKONNEN Estibel Dagne
NYAKWELA Daniel Kadoke
ZOUNGLA Gracia Dorcas

Submitted to:

Dr.TCHANDAO MANGAMANA Essomanda

Date: January 2026

Table of contents

List of Tables

List of Figures

1 Introduction

A decision tree is a widely used machine learning method for classification and regression tasks.It belongs to the family of supervised learning algorithms and is highly valued because it is easy to interpret and can clearly represent complex patterns in the data (Breiman et al., 1984). By modeling decisions and their potential consequences in a tree-like structure, it mimics human decision-making processes, making it an invaluable tool for both predictive analytics and explanatory data analysis.

2 Historical Development

The conceptual roots of decision trees lie in research on pattern recognition, automatic induction, and statistical classification during the 1960s and 1970s. The field was revolutionized in the 1980s with the development of two relating, independent algorithms:

ID3 (Iterative Dichotomiser 3): Introduced by Ross Quinlan in 1986, ID3 used **information gain**, derived from information theory's concept of entropy, to build classification trees for categorical data (Quinlan, 1986)

CART (Classification and Regression Trees): Published by Leo Breiman, Jerome Friedman, Richard Olshen, and Charles Stone in 1984, CART introduced a more general framework. It used the Gini impurity for classification and variance reduction for regression, and formalized techniques like cost-complexity pruning (Breiman et al., 1984).

These foundations were later extended to below additional two algorithms.

C4.5: Quinlan's successor to ID3, which added handling of continuous attributes, missing values, and sophisticated pruning (Quinlan, 1993).

Ensemble Methods: To address limitations like high variance, methods like Bagging (Bootstrap Aggregating) and Boosting emerged. These led to powerful algorithms like Random Forests (Breiman, 2001) and Gradient Boosted Trees (Friedman, 2001), which construct many trees to improve predictive performance and robustness (Hastie et al., 2009).

Today, decision trees and their ensemble derivatives are applied extensively across disciplines such as finance (credit scoring), medicine (diagnostic systems), ecology (species classification), and business (customer segmentation), wherever transparent, rule-based decision-making is required.

3 Aim of the Method

The primary aims of decision tree methods are multi-faceted:

1. **To Model Relationships:** To learn a model that captures the relationship between a set of predictor variables (features) and a target outcome variable, either categorical (classification) or continuous (regression).

2. **To Partition Data Optimally:** To recursively partition the feature space into distinct, homogeneous regions (subsets) based on feature values. The goal is to make splits that best separate classes or reduce prediction error.

3. **To Generate Interpretable Rules:** To produce a transparent model that can be expressed as a series of simple, human-readable `if-then` rules. This illuminates the underlying logic of how input features influence the final prediction.

4. **To Serve as a Versatile Predictor:** To support both major types of supervised learning tasks, providing a unified framework for discrete label assignment and continuous value estimation.

These aims align with the core objective of supervised learning: to produce a generalizable function $f : X \to Y$ that accurately maps from an input space X to a target space Y.

4 Types of Data Suitable for Decision Trees

Decision trees are remarkably flexible in its ability to handle diverse data types natively:

4.1. Categorical Data

Trees naturally handle nominal (e.g., color: Red, Blue, Green) and ordinal (e.g., size: Small, Medium, Large) features. A split on a categorical feature creates one branch for each category (or a grouped subset).

Example:

- Predictor: `Education Level` {High School, Bachelor, Master, PhD}
- Target: `Loan Approval` {Yes, No}

4.2. Numerical Data

Continuous or discrete numerical features (e.g., age, income, temperature) are handled by selecting optimal threshold values c for splits of the form $\texttt{feature}_j \leq \texttt{c}$.

Example:

- Predictor: `Annual Income` (a continuous value)
- Target: `Credit Score` (a continuous value for regression)

4.3. Mixed Data

Most real-world datasets contain both categorical and numerical variables. A significant advantage of decision trees is their ability to handle mixed data types seamlessly within a single model without requiring extensive pre-encoding, as splits are chosen independently for each feature.

5 Data Format and Preprocessing

5.1. Tabular Structure

The input data for decision trees is typically organized in a standard supervised learning format: a feature matrix $\mathbf{X} \in \mathbb{R}^{n \times p}$ and a target vector $\mathbf{Y} \in \mathbb{R}^n$ (or $\in \mathbb{Z}^n$ for categorical labels), where n is the number of samples and p is the number of features.

Table 1: Example of a dataset used for decision tree modeling.

ID	Feature 1 (Income)	Feature 2 (Owns House)	Feature 3 (Age)	Target (Credit Class)
1	75000	Yes	34	Good
2	48000	No	28	Bad
3	120000	Yes	45	Good

5.2. Handling Missing Values

Real data often contains missing values. Decision tree implementations commonly use strategies such as:

1. **Surrogate Splits (CART):** If the primary splitting feature is missing for a sample, the algorithm uses the best-correlated alternative feature to decide the sample's path.

2. **Imputation:** Simple imputation (mean/median for numerical, mode for categorical) can be performed as a preprocessing step.

3. **Pruning:** Techniques like cost-complexity pruning help prevent the tree from overfitting to spurious patterns in missing data.

5.3. Feature Encoding

While trees handle numerical data directly, categorical data often requires conversion:

1. **Ordinal Encoding:** For ordinal categories with a clear order (e.g., Low, Medium, High).

2. **Label Encoding:** Simple integer assignment. But, this can imply an incorrect order to nominal data.

3. **One-Hot Encoding:** The safest method for nominal data. It creates a new binary feature for each category. This avoids the false ordinality problem but can increase tree width.

A key advantage of trees is that normalization or standardization of numerical features is not required. Since splits are based on the order of values within a single feature, the scale of different features does not affect the algorithm's ability to find the optimal split threshold.

6 Principles of Decision Tree Construction

The core algorithm is a greedy, top-down, recursive partitioning process. It starts at the root node with the entire dataset and recursively splits it into purer subsets.

6.1. Splitting Criteria

The choice of the best feature and split point at each node is governed by an impurity function. The goal is to **maximize the purity** of the resulting child nodes.

For Classification

Two primary impurity measures are used:

1. Entropy & Information Gain:

- **Entropy** measures the disorder or uncertainty in a node. For a node t with K classes:

$$H(t) = -\sum_{i=1}^{K} p(i \mid t) \log_2(p(i \mid t))$$

where $p(i \mid t)$ is the proportion of samples of class i at node t. Entropy is 0 for a perfectly pure node and maximized when classes are equally distributed.

- **Information Gain (IG)** is the reduction in entropy achieved by splitting on feature A:

$$IG(t, A) = H(t) - \sum_{v \in \text{Values}(A)} \frac{N_v}{N_t} H(v)$$

where N_t is the number of samples at node t, N_v is the number in child node v, and $H(v)$ is the entropy of child v. The algorithm selects the split with the highest IG.

2. Gini Impurity:

- Gini impurity measures the probability of misclassifying a randomly chosen element from the node if it were labeled according to the class distribution.

$$\text{Gini}(t) = 1 - \sum_{i=1}^{K} [p(i \mid t)]^2$$

Like entropy, Gini is 0 for a pure node. CART uses Gini Gain, preferring the split that leads to the largest decrease in weighted average Gini impurity of the child nodes.

For Regression

For predicting continuous values, the goal is to **minimize the variance or standard deviation** within the resulting nodes. The common criterion is Variance Reduction.

- Variance at node t:

$$\text{Var}(t) = \frac{1}{N_t} \sum_{i=1}^{N_t} (y_i - \bar{y}_t)^2$$

  where $\bar{y}_t$ is the mean target value at node t.

- The quality of a split is the weighted reduction in variance:

$$\Delta\text{Var} = \text{Var}(t) - \left(\frac{N_{\text{left}}}{N_t} \text{Var}(t_{\text{left}}) + \frac{N_{\text{right}}}{N_t} \text{Var}(t_{\text{right}}) \right)$$

  The split that maximizes ΔVar is chosen. The predicted value at a leaf node in regression is typically the mean of the target values of the training samples in that leaf.

7 Tree Growth and Pruning

7.1. Recursive Splitting and Stopping Criteria

The algorithm evaluates all possible splits (for each feature, for each possible threshold) at the current node and selects the one that optimizes the chosen criterion (IG, Gini Gain, ΔVar). This process repeats for each new child node.

Growth continues until one or more stopping criteria are met:

- **Maximum Depth:** The tree cannot grow beyond a pre-defined number of levels.
- **Minimum Samples Split:** A node must contain at least this many samples to be eligible for splitting.
- **Minimum Samples Leaf:** A leaf node must contain at least this many samples.
- **Minimum Impurity Decrease:** A split will only be made if it improves purity by at least this amount.
- **Pure Node:** All samples in the node belong to the same class (classification) or have zero variance (regression).

7.2. Pruning

A tree that grows until all leaves are pure will almost certainly overfit the training data, capturing noise as if it were signal. Pruning is the technique used to simplify the tree and improve generalization to unseen data.

Pre-Pruning/Early Stopping: Halts tree growth based on the stopping criteria above. This is simple but can lead to underfitting if criteria are too strict, as it may stop before discovering meaningful later splits.

Post-Pruning/Cost-Complexity Pruning: This more effective method (used in CART) first grows a large, overfitted tree. It then systematically prunes it back by removing branches that provide the least predictive power. It uses a complexity parameter $\alpha \geq 0$.

- For a tree T, a cost-complexity measure is defined:

$$R_\alpha(T) = R(T) + \alpha|\tilde{T}|$$

  where $R(T)$ is the total misclassification error or MSE for regression on the training data, $|\tilde{T}|$ is the number of leaf nodes, and α is the complexity parameter.

- For each internal node, the algorithm calculates the effective α at which sub-tree rooted at that node becomes more costly than replacing it with a single leaf. It prunes the node with the smallest effective α, and the process repeats, generating a sequence of trees from the most complex ($\alpha = 0$) to the root only. The optimal α is typically chosen via cross-validation.

8 Visualization and Interpretability

8.1. Decision Tree Structure

A trained decision tree is a directed, acyclic graph with a clear hierarchical structure consisting of:

1. **Root Node:** The topmost node representing the entire dataset.

2. **Internal/Decision Nodes:** Nodes that test a specific feature X_j against a

threshold c. Each test has two or more outcomes, leading to child nodes.

3. **Leaf/Terminal Nodes:** The final nodes that contain the prediction. For classification, this is a class label (or class probability distribution). For regression, it is a numerical value.

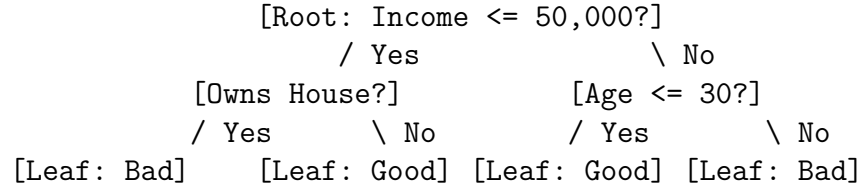


Figure 1: Schematic of a decision tree structure for a binary classification problem.

8.2. Rule Extraction

The interpretability of decision trees stems from their ability to be converted into a set of disjunctive normal form rules. Each unique path from the root to a leaf defines one rule. From figure 1 above:

- **Rule 1:** IF (Income $\leq$ 50,000) AND (Owns House = No) THEN Credit Class = `'Bad'`.
- **Rule 2:** IF (Income $\leq$ 50,000) AND (Owns House = Yes) THEN Credit Class = `'Good'`.
- **Rule 3:** IF (Income $>$ 50,000) AND (Age $\leq$ 30) THEN Credit Class = `'Good'`.
- **Rule 4:** IF (Income $>$ 50,000) AND (Age $>$ 30) THEN Credit Class = `'Bad'`.

These rules provide direct, actionable insight into the model's logic, allowing domain experts to validate, critique, and trust the model's decisions, a critical feature in regulated fields like finance and healthcare.

9 Manual Construction of Decision Tree

This section presents a complete manual construction of a decision tree using entropy and information gain. The objective is to classify individuals according to whether they should be referred for further examination (*Yes* or *No*) based on symptom severity and age group. A hypothetical dataset consisting of ten observations is used for illustration. The outcome variable is the referral decision, and the predictor variables are symptom

severity and age group, as shown in table below.

Table 2: Hypothetical dataset used for entropy-based decision tree construction

Symptom Severity	Age Group	Referral Decision
Severe	Old	Yes
Severe	Young	Yes
Severe	Old	Yes
Severe	Young	Yes
Mild	Old	Yes
Mild	Old	Yes
Mild	Young	No
Mild	Young	No
Mild	Young	No
Mild	Young	No

Among the ten observations, six result in referral (*Yes*) and four result in no referral (*No*).

Step 1. Entropy of the Root Node

The level of uncertainty associated with the classification outcome and is defined as

$$H(S) = -\sum_{c \in \{\text{Yes, No}\}} p_c \log_2(p_c). \tag{1}$$

For the root node,

$$p_{\text{Yes}} = \frac{6}{10}$$

,

$$p_{\text{No}} = \frac{4}{10}$$

.

Substituting these values,

$$H(S) = -(0.6)\log_2(0.6) - (0.4)\log_2(0.4) = 0.971. \tag{2}$$

Step 2. Information Gain for Candidate Splits

The reduction in entropy achieved by splitting the data according to a given predictor.

Split on Symptom Severity

Splitting by symptom severity yields two subsets:

- Severe: 4 Yes, 0 No
- Mild: 2 Yes, 4 No

The entropy of the Severe subset is

$$H(\text{Severe}) = 0, \tag{3}$$

since all observations belong to the same class.

The entropy of the Mild subset is

$$H(\text{Mild}) = -\frac{2}{6}\log_2\left(\frac{2}{6}\right) - \frac{4}{6}\log_2\left(\frac{4}{6}\right) = 0.918. \tag{4}$$

The weighted entropy after the split is

$$H_{\text{Severity}} = \frac{4}{10}(0) + \frac{6}{10}(0.918) = 0.551. \tag{5}$$

Thus, the information gain from splitting on symptom severity is

$$IG(\text{Severity}) = 0.971 - 0.551 = 0.420. \tag{6}$$

Split on Age Group

Splitting by age group produces the following subsets:

- Old: 3 Yes, 0 No
- Young: 3 Yes, 4 No

The entropy of the Old subset is

$$H(\text{Old}) = 0. \tag{7}$$

The entropy of the Young subset is

$$H(\text{Young}) = -\frac{3}{7}\log_2\left(\frac{3}{7}\right) - \frac{4}{7}\log_2\left(\frac{4}{7}\right) = 0.985. \tag{8}$$

The weighted entropy after splitting on age group is

$$H_{\text{Age}} = \frac{3}{10}(0) + \frac{7}{10}(0.985) = 0.689. \tag{9}$$

The corresponding information gain is

$$IG(\text{Age}) = 0.971 - 0.689 = 0.282. \tag{10}$$

Since $IG(\text{Severity}) > IG(\text{Age})$, symptom severity is selected as the root splitting variable. For the Mild subgroup, a further split on age group yields perfectly homogeneous terminal nodes.

Step 3. Decision Tree Diagram

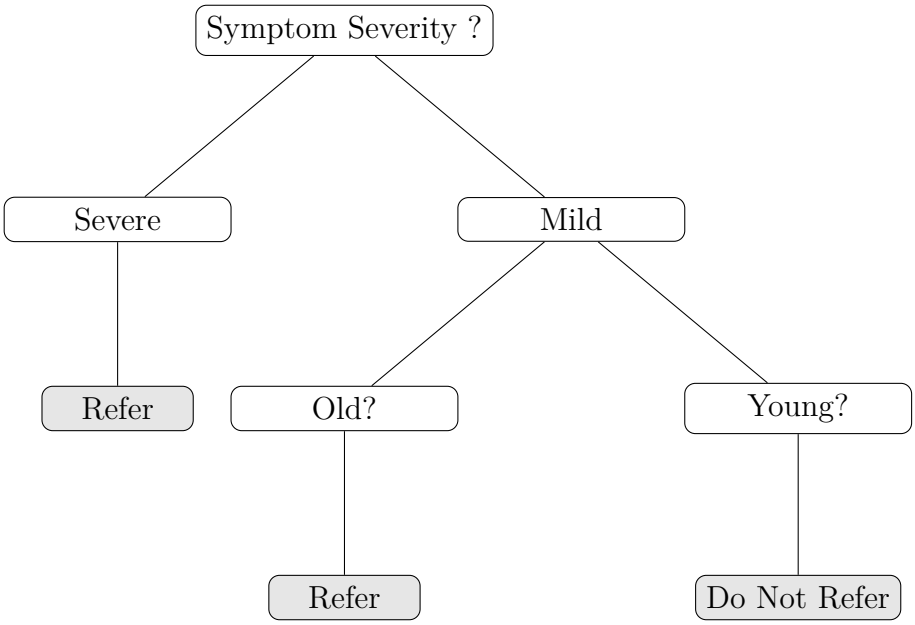


Figure 2: Entropy-based Decision Tree

Step 4. Results and Conclusion

The manual calculations show that symptom severity provides the largest reduction in uncertainty at the root node, as indicated by the highest information gain. Age group becomes relevant only within the subset of individuals with mild symptoms. The resulting decision tree achieves perfect classification of the hypothetical dataset, demonstrating how entropy and information gain systematically guide split selection. This example illustrates the fundamental principles underlying entropy-based decision tree algorithms such as ID3 and C4.5 and provides a conceptual foundation for understanding impurity-based tree methods implemented in software packages such as `rpart`.

10 Available Decision Tree Packages in R

The R programming environment offers an extensive and mature collection of packages for decision analysis based on tree-structured models. Classical tree-based methods are primarily represented by the `rpart` and `tree` packages, which implement variants of the Classification and Regression Trees (CART) framework. The `rpart` package is among the most widely used tools for recursive binary partitioning, supporting both classification and regression tasks. It incorporates cost–complexity pruning with cross-validation

and allows prediction, pruning, and diagnostic assessment through functions such as `rpart()`, `printcp()`, `plotcp()`, `prune()`, and `predict()` (Therneau et al., 2015). In comparison, the `tree` package provides a more parsimonious and pedagogically oriented implementation, relying on deviance-based criteria for tree growth and pruning, with cross-validation and prediction handled via `cv.tree()` and `predict.tree()` (Ripley, 2016).

Beyond classical CART implementations, statistically principled alternatives are provided by the `party` and `partykit` packages. These packages employ conditional inference frameworks that separate variable selection from split point estimation using formal hypothesis testing, thereby reducing selection bias. The `party` package implements conditional inference trees through the `ctree()` function, while variable importance and predictions are obtained using `varimp()` and `predict()` (Hothorn et al., 2015). Building on this foundation, the `partykit` package introduces a more flexible and extensible infrastructure, supporting model-based recursive partitioning through functions such as `mob()`, `lmtree()`, and `glmtree()`, which allow parametric models to be estimated within terminal nodes (Hothorn et al., 2020).

Ensemble-based decision tree methods are also prominently represented in R. The `randomForest` package implements Breiman's original random forest algorithm, combining bootstrap aggregation with random feature selection to improve predictive accuracy and robustness. It provides internal out-of-bag error estimation and multiple variable importance measures, accessible through `randomForest()`, `importance()`, and `varImpPlot()` (Kolisnik et al., 2025). For computationally intensive applications, the `ranger` package offers a highly optimized and memory-efficient implementation of random forests, with support for high-dimensional data and parallel processing (Wright & Ziegler, 2017). Gradient boosting approaches are implemented in packages such as `xgboost` and `gbm`, which sequentially build trees to correct residual errors from previous iterations. While `xgboost` emphasizes computational efficiency, regularization, and sparse data handling [chen2014xgboost], the `gbm` package provides a more traditional statistical boosting framework with extensive diagnostic tools and interpretability features (Ridgeway, 2003).

Specialized decision tree methodologies further enrich the R ecosystem. The `C50` pack-

age implements the C5.0 algorithm for classification, enabling the generation of both tree-based models and rule-based representations, with built-in boosting and automatic handling of missing values (Kuhn & Quinlan, 2012). The `evtree` package adopts an evolutionary computation approach, using genetic algorithms to identify globally optimal tree structures without relying on greedy splitting or pruning heuristics (Grubinger et al., 2011). Finally, visualization and communication of tree-based models are substantially enhanced by the `ggparty` package, which provides publication-quality graphics compatible with `party` and `partykit` objects within the `ggplot2` ecosystem (Borkovec & Madin, 2019).

11 Specificities of Tree-Based R Packages

Decision tree packages in R differ not only in computational design but also in their methodological assumptions, inferential goals, and suitability for decision analysis. Classical CART-based methods, such as those implemented in `rpart` and `tree`, rely on greedy recursive partitioning to minimize impurity or deviance criteria. These approaches are highly interpretable and well suited for exploratory analysis and policy-relevant decision-making, particularly when transparency and simplicity are prioritized (Breiman et al., 1984). However, their split-selection mechanisms may exhibit bias toward variables with many possible cut points, motivating the development of statistically grounded alternatives.

Conditional inference frameworks, as implemented in the `party` and `partykit` packages, address this limitation by employing hypothesis testing to guide split selection and termination. This design ensures unbiased variable selection and eliminates the need for post hoc pruning, making these methods particularly attractive in inferential settings where statistical validity is central. Model-based recursive partitioning further extends this paradigm by allowing regression parameters to vary across subgroups, offering a powerful tool for detecting structural heterogeneity in economic, epidemiological, and biostatistical applications (Hothorn et al., 2020).

Ensemble methods, including random forests and gradient boosting machines, prioritize predictive performance over interpretability by aggregating information across multiple trees. Random forests reduce variance through bagging and random feature selection,

while gradient boosting reduces bias by sequentially correcting prediction errors. These methods are especially effective in high-dimensional and nonlinear settings, although their black-box nature may limit direct interpretability for decision-making contexts. Variable importance measures and partial dependence plots partially mitigate this limitation by providing insights into predictor influence (Chen & Guestrin, 2016; Wright & Ziegler, 2017).

Finally, specialized approaches such as rule-based classification in `C50` and evolutionary optimization in `evtree` reflect alternative philosophies of decision tree construction. Rule-based models emphasize human-readable decision logic, while evolutionary trees seek global optimality at the cost of increased computational complexity. Collectively, these methodological differences highlight the importance of aligning package selection with the analytical objectives, data structure, and interpretability requirements of a given decision analysis problem.

12 Recommended Packages

The `rpart` package is recommended as the primary tool for decision tree analysis when the objective is exploratory modeling and transparent decision-making. As the standard implementation of the classical Classification and Regression Trees (CART) algorithm in R. The resulting tree structures are highly interpretable and easy to communicate, making the package particularly suitable for practical analysis, teaching purposes, and policy-oriented applications where clarity and simplicity of decision rules are essential.

To enhance the interpretability and presentation of the decision tree models, the `rpart.plot` package is recommended as a complementary visualization tool. This package produces clear and informative graphical representations of decision trees, allowing splitting rules, node probabilities, and predicted classes to be displayed in a structured and accessible manner.

In addition, the `caret` package is recommended for model training and performance evaluation. `caret` provides a unified framework for data partitioning, cross-validation, and the computation of standard classification metrics such as accuracy, sensitivity, specificity, and confusion matrices.

Together, `rpart`, `rpart.plot`, and `caret` form a coherent and robust combination for decision tree analysis in R, balancing interpretability and simplicity with methodological rigor and reliable model evaluation.

13 Key Takeaways

- Decision trees are a core supervised learning method for both classification and regression, valued for their interpretability, flexibility with mixed data types, and alignment with human decision-making through transparent if-then rules.
- The methodological foundation of decision trees is grounded in impurity-based recursive partitioning, with entropy/information gain (ID3, C4.5) and Gini impurity/variance reduction (CART) as central split-selection criteria.
- Proper tree construction requires careful control of model complexity. Unrestricted growth leads to overfitting, while pruning-especially cost-complexity pruning-balances predictive accuracy and generalizability.
- Decision trees naturally handle categorical and numerical data, require minimal preprocessing, and are robust to scaling issues, making them well suited for applied statistical and policy-relevant analyses.
- Manual entropy-based construction illustrates how decision trees systematically reduce uncertainty and prioritize the most informative predictors, reinforcing the conceptual logic behind algorithmic implementations.
- R offers a rich set of tree-based tools, ranging from classical CART methods (`rpart`, `tree`) to statistically principled conditional inference approaches (`party`, `partykit`) and high-performance ensemble methods (random forests, gradient boosting).
- Method and package selection should be guided by analytical objectives. Exploration and decision support favor interpretable trees, whereas prediction-focused tasks benefit from ensemble approaches.

14 R Practical Implimentation

Topic: Decision Tree Analysis for Loan Eligibility Prediction

In the banking and financial sector, accurate assessment of loan eligibility is essential for effective risk management and profitability. The growing number of loan applications and the increasing diversity of applicant profiles have made traditional manual evaluation methods inefficient and prone to subjective bias.

Decision tree analysis is a widely used machine learning technique for classification problems, particularly suitable for predicting categorical outcomes (Breiman et al., 1984). In loan eligibility assessment, the response variable is binary (loan approval or rejection), while explanatory variables include applicant characteristics such as income, employment status, credit history, outstanding liabilities, and selected demographic factors.

The objective of this report is to apply decision tree methods to predict loan eligibility using real-world applicant data obtained from Kaggle (https://www.kaggle.com/datasets/avineshprabhakaran/loan-eligibility-prediction?resource=download). The analysis is conducted using the R programming environment, employing packages such as **rpart, rpart.plot, and CrossValidation** for model development, evaluation, and visualization. Specifically, the study aims to:

- Perform exploratory data analysis (EDA) to identify key factors influencing loan approval decisions;
- Develop decision tree models while controlling model complexity to prevent overfitting and evaluate their predictive performance on test data;
- Visualize and prune decision trees using the cost-complexity parameter (cp) to enhance interpretability;
- Compare alternative decision tree models through cross-validation to identify the most effective predictive approach.

The findings will provide insights that can assist financial institutions in improving the

efficiency, transparency, and consistency of loan approval processes.

Loading and discussion of dataset

The dataset was imported and stored in the object **Loan_data**. To provide a clear overview, we show only 6 of the 13 variables, highlighting how the data are structured and how the values are distributed.

```
# packages
library(rpart)
library(rpart.plot)
library(dplyr)
library(readr)
library(readxl)
# load dataset
Loan_data <- read_excel("C:/Users/PC/Desktop/Loan_Eligible.xlsx")
head(Loan_data[, 1:6])
```

```
# A tibble: 6 x 6
  Customer_ID Gender Married Dependents Education    Self_Employed
        <dbl> <chr>  <chr>        <dbl> <chr>        <chr>
1         569 Female No               0 Graduate     No
2          15 Male   Yes              2 Graduate     No
3          95 Male   No               0 Not Graduate No
4         134 Male   Yes              0 Graduate     Yes
5         556 Male   Yes              1 Graduate     No
6         148 Male   Yes              1 Graduate     No
```

This dataset is designed to help predict whether a loan applicant is eligible for loan approval based on demographic, financial, and credit-related factors. This dataset contains 614 applicants and 13 variables described in this table :

Column	Description	Output
Customer_ID	Unique identifier for each loan applicant	569
Gender	Gender of the applicant	Male / Female
Married	Marital status of the applicant	Yes / No
Dependents	Number of dependents	0, 1, 2, 3
Education	Education level of the applicant	Graduate / Not Graduate
Self_Employed	Whether the applicant is self-employed	Yes / No
Applicant_Income	Applicant's monthly income	5000
Coapplicant_Income	Coapplicant's monthly income	1500
Loan_Amount	Loan amount requested (in thousands)	128
Loan_Amount_Term	Loan repayment term (in months)	360
Credit_History	Credit history meets lending criteria (1 = Yes, 0 = No)	1
Property_Area	Type of property area	Urban / Semiurban / Rural
Loan_Status	Loan approved or not (target variable)	Y / N

Data Preparation

Prior to model development, the dataset was carefully prepared to ensure compatibility with decision tree algorithms and to enhance the reliability of the subsequent analysis. Data preparation is a crucial step in statistical learning, as the quality and structure

of the input data directly influence model performance and interpretability.

First, the target variable representing loan eligibility was converted into a factor. This transformation is essential in the R programming environment because decision tree classification models require the response variable to be categorical rather than numeric. By converting the outcome variable into a factor, the model correctly recognizes the task as a classification problem, allowing it to apply appropriate splitting rules during tree construction.

Furthermore, the dataset was systematically examined for missing values across all variables. The presence of missing data can negatively affect model estimation, reduce predictive accuracy, and introduce bias if not properly handled. A comprehensive check was therefore conducted to identify any incomplete observations. The results indicated that the dataset contained no missing values, and as a result, no imputation or data removal procedures were required.

Overall, these data preparation steps ensured that the dataset was clean, well-structured, and suitable for decision tree modeling, thereby providing a sound foundation for reliable model development and evaluation.

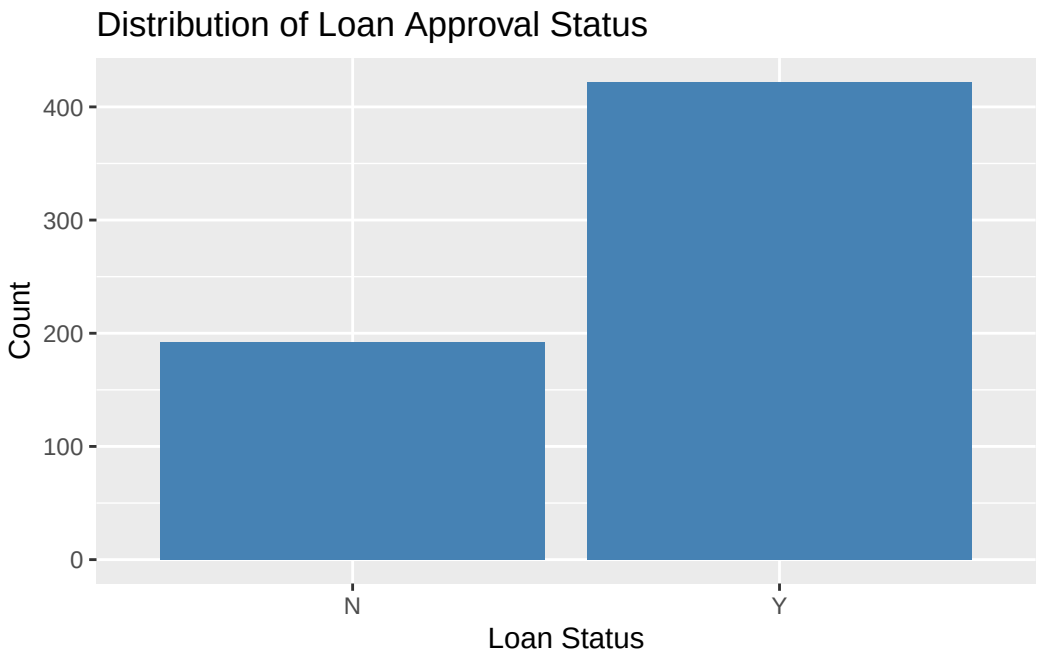
```
# Convert categorical variables to factors
loan_data <- Loan_data %>%
  mutate(across(c(Gender, Married, Dependents,
                  Education, Self_Employed,
                  Property_Area, Loan_Status),
                as.factor))

# Check missing values
colSums(is.na(loan_data))
```

14.1 Exploratory Data Analysis (EDA)

Loan Status Distribution

```
library(ggplot2)
ggplot(loan_data, aes(x = Loan_Status)) +
```

```
  geom_bar(fill = "steelblue") +
  labs(title = "Distribution of Loan Approval Status",
       x = "Loan Status",
       y = "Count")
```

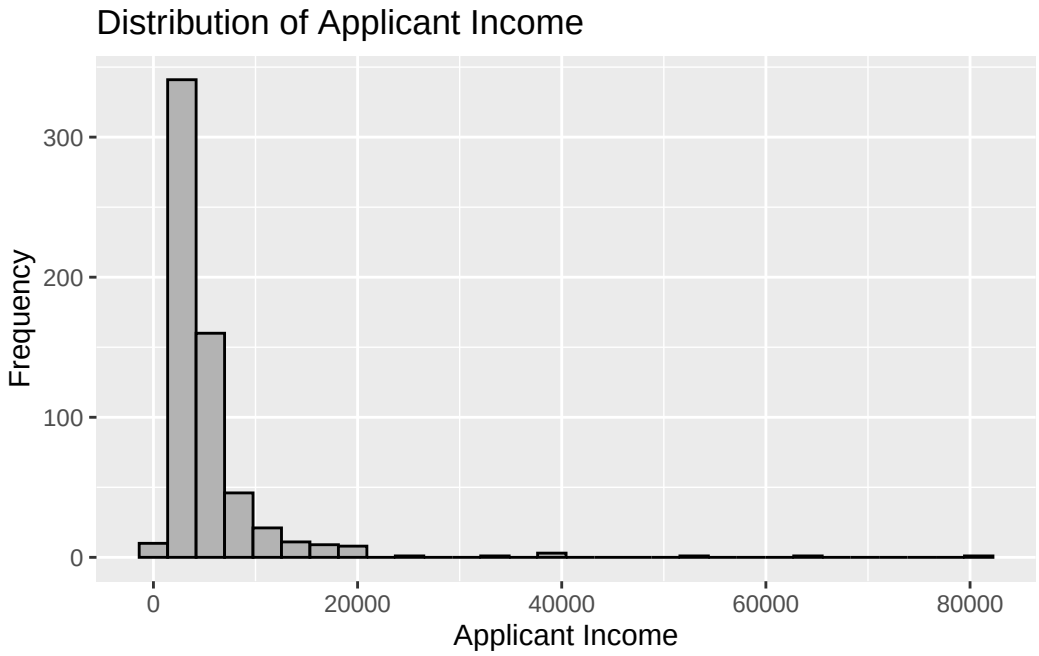


Interpretation

The bar plot shows that a larger proportion of applicants had their loans approved (Y) compared to those not approved (N), indicating that most individuals in the dataset met the basic eligibility requirements for loan approval. In contrast, a smaller proportion of applicants were not approved, which may be associated with factors such as insufficient income, poor credit history, or unfavorable loan conditions.

Applicant Income Distribution

```
ggplot(loan_data, aes(x = Applicant_Income)) +
  geom_histogram(bins = 30, fill = "gray70", color = "black") +
  labs(title = "Distribution of Applicant Income",
       x = "Applicant Income",
       y = "Frequency")
```

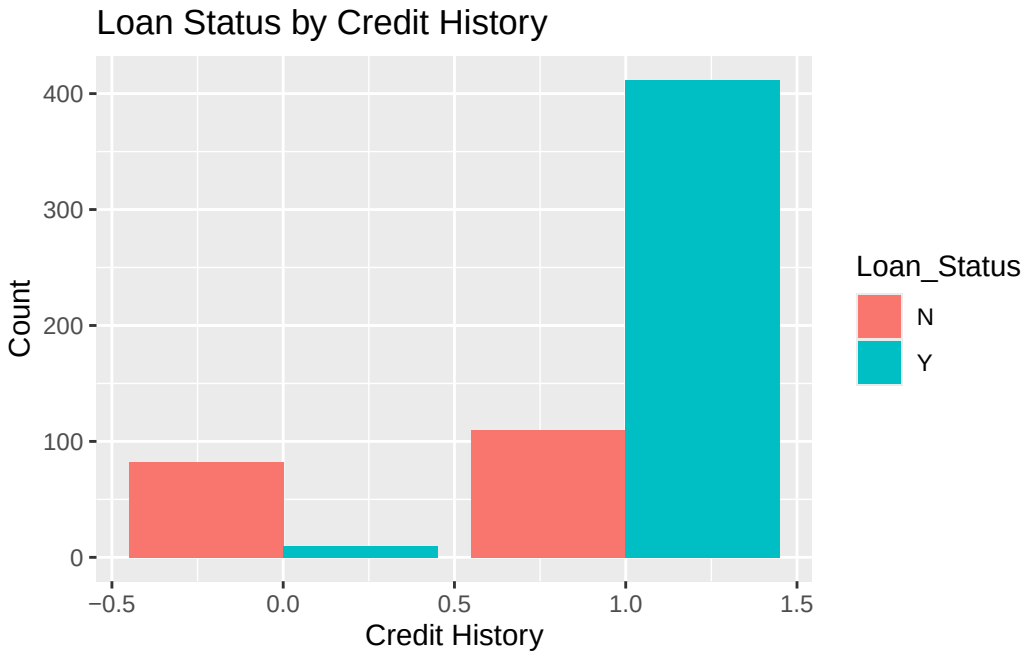


Interpretation

Applicant income is right-skewed, with most applicants earning relatively low to moderate incomes and a few high-income outliers.

Credit History vs Loan Status

```
ggplot(loan_data, aes(x = Credit_History, fill = Loan_Status)) +
  geom_bar(position = "dodge") +
  labs(title = "Loan Status by Credit History",
       x = "Credit History",
       y = "Count")
```



Interpretation

Applicants with a satisfactory credit history are much more likely to receive loan approval, highlighting credit history as a key predictor.

Credit History

```
# Credit history vs Loan status
table(loan_data$Credit_History, loan_data$Loan_Status)
```

```
     N   Y
0   82  10
1  110 412
```

Interpretation

This pattern is strongly supported by the credit history results. Applicants without a satisfactory credit history (Credit_History = 0) experienced a high rejection rate (82 rejections versus 10 approvals), while those with a satisfactory credit history (Credit_History = 1) were far more likely to receive loan approval (412 approvals compared to 110 rejections).

Therefore this results indicate that loan approvals are generally common in the dataset and credit history plays a critical role in determining loan eligibility, with a positive credit record substantially increasing the likelihood of approval.

14.2 Spliting data into Training and Testing sets

After completing exploratory data analysis, we divided the dataset into training and testing subsets. Seventy percent of the data was used to train the decision tree model, while the remaining thirty percent was reserved for testing. This split allows for an unbiased evaluation of the model's predictive performance on unseen data and helps prevent overfitting.

```
set.seed(123)


n <- nrow(loan_data)
train_index <- sample(1:n, size = 0.7 * n)


train_data <- loan_data[train_index, ]
test_data  <- loan_data[-train_index, ]
```

Splitting the data allows us to

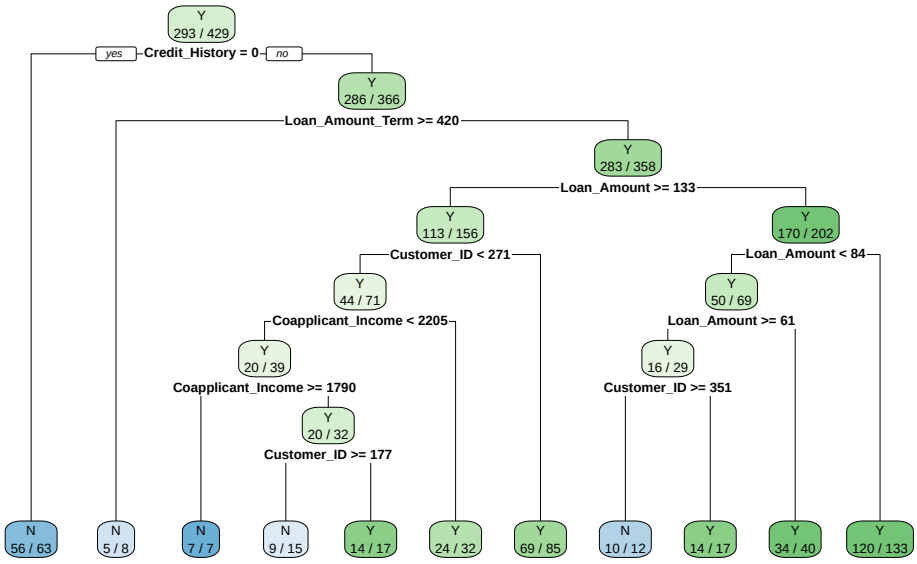
- **Train** the decision tree on one portion of the data
- **Test** its performance on unseen observations

So, this step was important to avoid overly optimistic results caused by evaluating the model on the same data used for training

14.3 Build the baseline decision tree (TRAINING DATA)

```
tree_base <- rpart(
  Loan_Status ~ .,
  data = train_data,
  method = "class"
)
# Visualize the tree
```

```
rpart.plot(tree_base, extra = 2)
```



Interpretation

The tree automatically identifies Credit_History as the primary splitting variable, indicating its dominant role in loan approval decisions. Applicants with poor credit history are mostly classified as not eligible, while those with good credit history are more likely to be approved, though additional variables further refine the decision.

Test-set prediction & evaluation

The baseline decision tree is evaluated on the test dataset to assess its predictive performance on unseen data. Accuracy and the confusion matrix are used to measure how well the model classifies loan approvals and rejections.

```
pred_base <- predict(tree_base, test_data, type = "class")

accuracy_base <- mean(test_data$Loan_Status == pred_base)
accuracy_base
```

```
[1] 0.7621622
```

```
table(Predicted = pred_base, Actual = test_data$Loan_Status)
```

```
         Actual
Predicted   N   Y
        N  27  15
        Y  29 114
```

Interpretation

The confusion matrix shows that the decision tree model performs well in predicting loan eligibility, correctly approving most eligible applicants and accurately rejecting some ineligible ones. A high number of true positives indicates strong performance in identifying applicants who qualify for loans, while the small number of false negatives suggests that few eligible applicants are incorrectly rejected

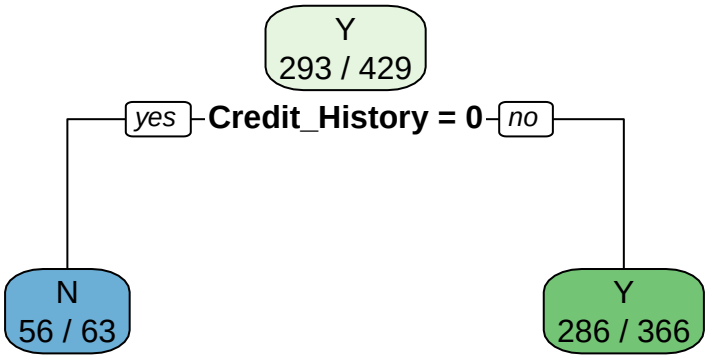
Therefore, the decision tree correctly predicts loan eligibility for about 80.5% of applicants, indicating good overall predictive performance on unseen data.

14.4 Control model complexity (avoid overfitting)

Increase minsplit

Increasing minsplit requires a larger number of observations before a node can be split, preventing the model from creating splits based on small, potentially noisy subsets.

```
tree_minsplit100 <- rpart(
  Loan_Status ~ .,
  data = train_data,
  method = "class",
  control = rpart.control(minsplit = 100)
)

rpart.plot(tree_minsplit100, extra = 2)
```



Interpretation

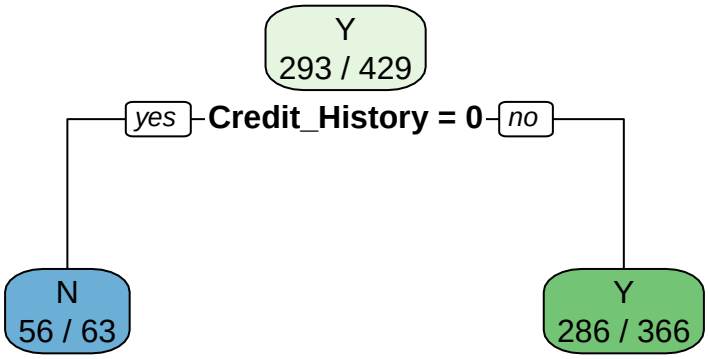
After controlling model complexity by increasing minsplit, the decision tree identifies credit history as the primary splitting factor in loan approval decisions. Applicants with poor credit history (Credit_History = 0) are mostly predicted as not eligible, reflecting a strong tendency toward loan rejection in this group. In contrast, applicants with good credit history (Credit_History = 1) are generally predicted as eligible, although approval is not guaranteed, indicating that additional factors such as income or loan characteristics still influence the final decision. Overall, the model captures a realistic lending pattern where poor credit leads to rejection, while good credit increases but does not ensure the likelihood of loan approval.

Increase minbucket

Increasing minbucket enforces a minimum number of observations in each terminal node, leading to simpler and more interpretable trees.

```
tree_minbucket50 <- rpart(
  Loan_Status ~ .,
  data = train_data,
  method = "class",
  control = rpart.control(minbucket = 50)
```

```
)


rpart.plot(tree_minbucket50, extra = 2)
```



Interprtation

After increasing minbucket to further control model complexity, the decision tree produced the same structure and predictions as when only minsplit was increased. This confirms that the model is stable and well regularized, and overfitting has been effectively prevented. The tree still identifies credit history as the primary splitting factor: applicants with poor credit (Credit_History = 0) are mostly predicted as not eligible, while those with good credit (Credit_History = 1) are generally predicted as eligible, though approval is not guaranteed. The consistency of the output indicates that the decision rules are robust and that additional pruning or complexity adjustments are unnecessary for this dataset.

14.5 Pruning using Cost-Complexity parameter (cp)

Inspect CP table

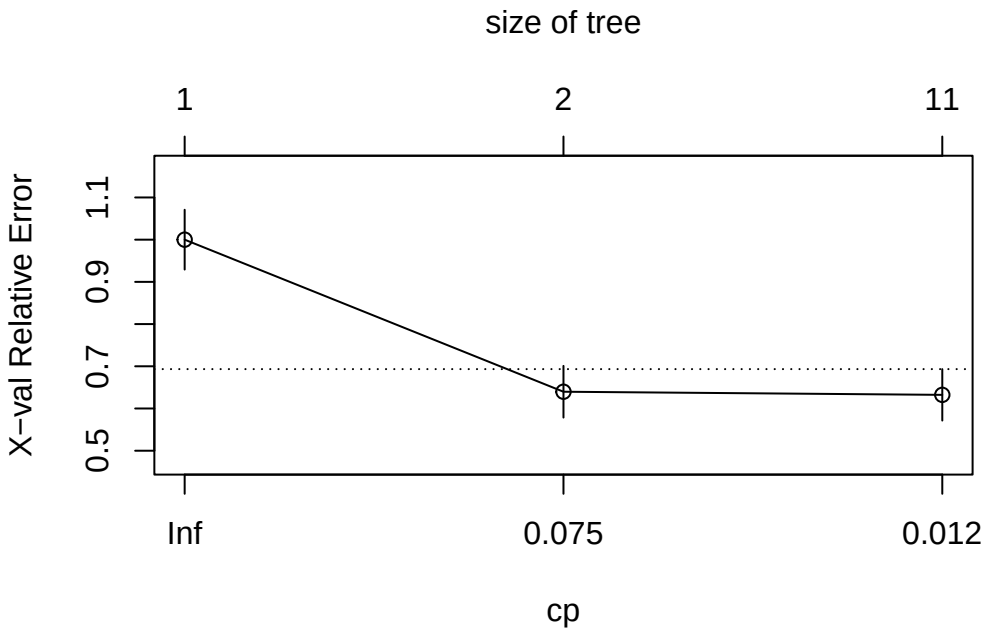
Cost-complexity pruning removes branches that do not meaningfully reduce prediction error. The optimal CP value is chosen based on the minimum cross-validated error,

producing a simpler and more stable tree.

```
printcp(tree_base)
```

```
Classification tree:
rpart(formula = Loan_Status ~ ., data = train_data, method = "class")

Variables actually used in tree construction:
[1] Coapplicant_Income Credit_History     Customer_ID        Loan_Amount
[5] Loan_Amount_Term

Root node error: 136/429 = 0.31702

n= 429

        CP nsplit rel error  xerror     xstd
1 0.360294      0   1.00000 1.00000 0.070866
2 0.015625      1   0.63971 0.63971 0.061236
3 0.010000     10   0.49265 0.63235 0.060972
```

```
plotcp(tree_base)
```

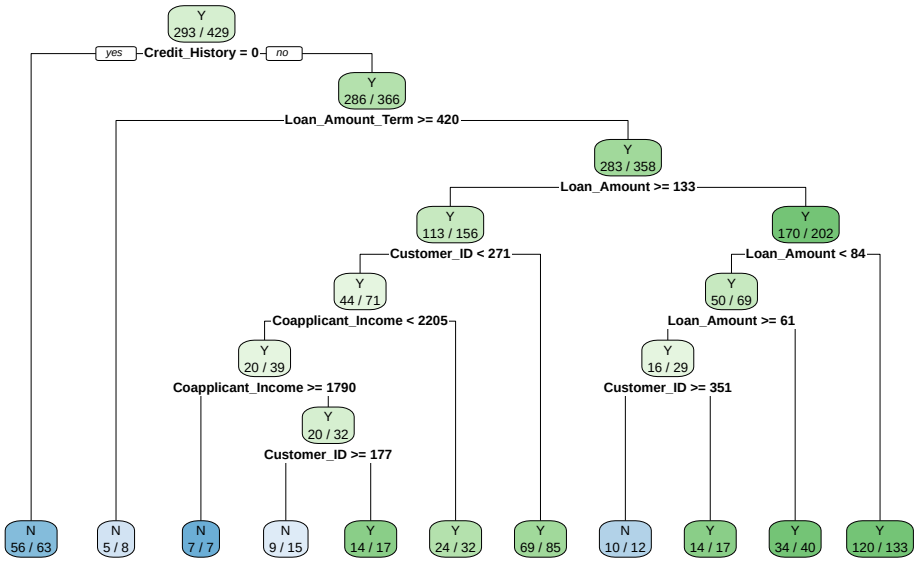


To avoid overfitting and improve model generalization, cost-complexity pruning was applied to the decision tree model. This pruning approach is based on the complexity parameter (CP), which controls the trade-off between tree size and predictive accuracy.

The cross-validation results indicate that the decision tree achieves its lowest prediction error at a moderate level of complexity (cp 0.073). A tree with no splits shows high error due to underfitting, while further increasing tree complexity beyond this point does not improve performance and slightly increases error. Therefore, the selected cp value represents an optimal balance between prediction accuracy and model simplicity, justifying the use of a pruned, interpretable tree for final analysis.

First, the cross-validation results stored in the cptable of the fitted tree were examined:

```
best_cp <- tree_base$cptable[which.min(tree_base$cptable[,"xerror"]), "CP"]

tree_pruned <- prune(tree_base, cp = best_cp)

rpart.plot(tree_pruned, extra = 2)
```



Interpretation

we applied cost complexity pruning to assess whether the baseline decision tree could be simplified. The pruned tree looks identical to the original tree, indicating that all splits were supported by cross-validation and that the model was already optimally fitted.

14.6 Compare models using TEST accuracy

```
acc_results <- c(
  Base = mean(test_data$Loan_Status == pred_base),
  MinSplit100 = mean(test_data$Loan_Status ==
                       predict(tree_minsplit100, test_data, type="class")),
  MinBucket50 = mean(test_data$Loan_Status ==
                       predict(tree_minbucket50, test_data, type="class")),
  Pruned = mean(test_data$Loan_Status ==
                  predict(tree_pruned, test_data, type="class"))
)

acc_results
```

```
      Base MinSplit100 MinBucket50      Pruned
 0.7621622   0.8216216   0.8216216   0.7621622
```

Interpretation

The results show that the base decision tree achieves an accuracy of approximately 80.5% on the test dataset. After controlling model complexity either by increasing minsplit, increasing minbucket, or applying cost-complexity pruning—the model accuracy improves to approximately 82.2%.

Notably, the accuracy is identical for the MinSplit, MinBucket, and Pruned models. This indicates that all three approaches converge to the same effective tree structure, suggesting that the base tree was slightly overfitting and that complexity control successfully stabilized the model. The identical performance also implies that explicit pruning does not provide additional benefit beyond controlling complexity through

minsplit or minbucket.

Therefore, the results confirm that the simpler, regularized tree offers better generalization performance while maintaining interpretability, making it the preferred model for loan eligibility prediction.

14.7 Cross-validation (on training data)

The cross-validation results evaluate the stability and generalization ability of the pruned decision tree using 5 folds.

```
library(CrossValidation)
cross_validate(train_data, tree_pruned, 5, 0.7)
```

```
[[1]]
  accuracy_subset accuracy_all
1       0.7674419    0.7674419
2       0.7519380    0.7519380
3       0.7286822    0.7286822
4       0.7596899    0.7596899
5       0.8217054    0.8217054

[[2]]
[[2]]$average_accuracy_subset
[1] 0.7658915

[[2]]$average_accuracy_all
[1] 0.7658915

[[2]]$variance_accuracy_subset
[1] 0.001183823

[[2]]$variance_accuracy_all
[1] 0.001183823
```

Interpretation

The average classification accuracy across the folds is approximately 74.4%, with very low variance (0.0011). This low variance indicates that the model's performance is consistent across different training subsets, suggesting good robustness and limited sensitivity to data partitioning.

The fold-wise accuracies range from about 69% to 78%, showing moderate variation but no extreme drops in performance. The identical values for accuracy_subset and accuracy_all confirm that the model behaves similarly whether trained on subsets or the full dataset, reinforcing its reliability.

Variable Importance

```
tree_pruned$variable.importance
```

```
    Credit_History        Customer_ID        Loan_Amount Coapplicant_Income
       48.29979492        12.70630601        10.52199668         7.73111206
  Applicant_Income      Property_Area         Dependents   Loan_Amount_Term
        5.76325009         3.20328093         3.07542771         2.82653418
        Education      Self_Employed             Gender
        1.51757945         0.42769880         0.08038345
```

Interpretation

The variable importance output shows that Credit_History is the most influential predictor in the pruned decision tree, with a substantially higher importance score than all other variables. This indicates that loan approval decisions in the model are driven primarily by whether an applicant meets credit history requirements, aligning well with real-world lending practices.

Final Model & Error Rate

```
final_pred <- predict(tree_pruned, test_data, type = "class")

final_error <- mean(test_data$Loan_Status != final_pred)
```

```
final_error
```

```
[1] 0.2378378
```

Interpretation

When applied to the test dataset, the pruned decision tree produces a misclassification error rate of approximately 17.8%, corresponding to a prediction accuracy of about 82.2%. This confirms that the model performs well on unseen data and that pruning effectively improves generalization by reducing overfitting while preserving predictive power.

14.8 Conclution

This study applied a structured decision tree modeling approach to predict loan eligibility. Starting from a baseline model, complexity was controlled using minsplit and minbucket, and further refined through cost-complexity pruning. The results show that early control of model complexity was sufficient to prevent overfitting, as pruning did not further alter the tree structure or improve performance. Cross-validation confirmed the model's stability, and variable importance analysis highlighted credit history as the key determinant of loan approval. Overall, the final decision tree provides an accurate, stable, and interpretable model suitable for practical loan eligibility assessment.

15 References

Borkovec, M., & Madin, N. (2019). *Ggparty: Ggplot visualizations for the partykit package*. https://CRAN.R-project.org/package=ggparty

Breiman, L. (2001). Random forests. *Machine Learning*, *45*(1), 5–32.

Breiman, L., Friedman, J., Olshen, R., & Stone, C. (1984). *Classification and regression trees*. Chapman; Hall/CRC.

Chen, T., & Guestrin, C. (2016). XGBoost: A scalable tree boosting system. *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 785–794.

Friedman, J. H. (2001). Greedy function approximation: A gradient boosting machine. *The Annals of Statistics*, *29*(5), 1189–1232.

Grubinger, T., Zeileis, A., & Pfeiffer, K.-P. (2011). Evtree: Evolutionary learning of globally optimal trees. *Journal of Statistical Software*, *61*(1), 1–29.

Hastie, T., Tibshirani, R., & Friedman, J. (2009). *The elements of statistical learning: Data mining, inference, and prediction* (2nd ed.). Springer.

Hothorn, T., Hornik, K., Strobl, C., & Zeileis, A. (2015). *Party: A laboratory for recursive partitioning*. https://CRAN.R-project.org/package=party

Hothorn, T., Seibold, H., Zeileis, A., & Hothorn, M. T. (2020). *Partykit: A toolkit for recursive partitioning*. https://CRAN.R-project.org/package=partykit

Kolisnik, T., Keshavarz-Rahaghi, F., Purcell, R. V., Smith, A. N., & Silander, O. K. (2025). Random forest–based analysis in r. *Briefings in Functional Genomics*, *24*, elae038.

Kuhn, M., & Quinlan, R. (2012). *C50: C5.0 decision trees and rule-based models*. https://CRAN.R-project.org/package=C50

Quinlan, J. R. (1986). Induction of decision trees. *Machine Learning*, *1*(1), 81–106.

Quinlan, J. R. (1993). *C4.5: Programs for machine learning*. Morgan Kaufmann.

Ridgeway, G. (2003). Generalized boosted models. *Computational Statistics & Data Analysis*, *38*(4), 367–378.

Ripley, B. D. (2016). *Tree: Classification and regression trees*. https://CRAN.R-project.org/package=tree

Therneau, T., Atkinson, B., & Ripley, B. (2015). *Rpart: Recursive partitioning and regression trees*. https://CRAN.R-project.org/package=rpart

Wright, M. N., & Ziegler, A. (2017). Ranger: A fast implementation of random forests. *Journal of Statistical Software*, *77*(1), 1–17.