APPLICATION OF TDIDT ALGORITHM TO CLASSIFY A CLIENT'S SUBSCRIPTION TO A LONG-TERM DEPOSIT AT A BANK

Submitted To:

DR. AKINUL ISLAM JONY
ASSOCIATE PROFESSOR, DEPARTMENT HEAD
COMPUTER SCIENCE

Project by:

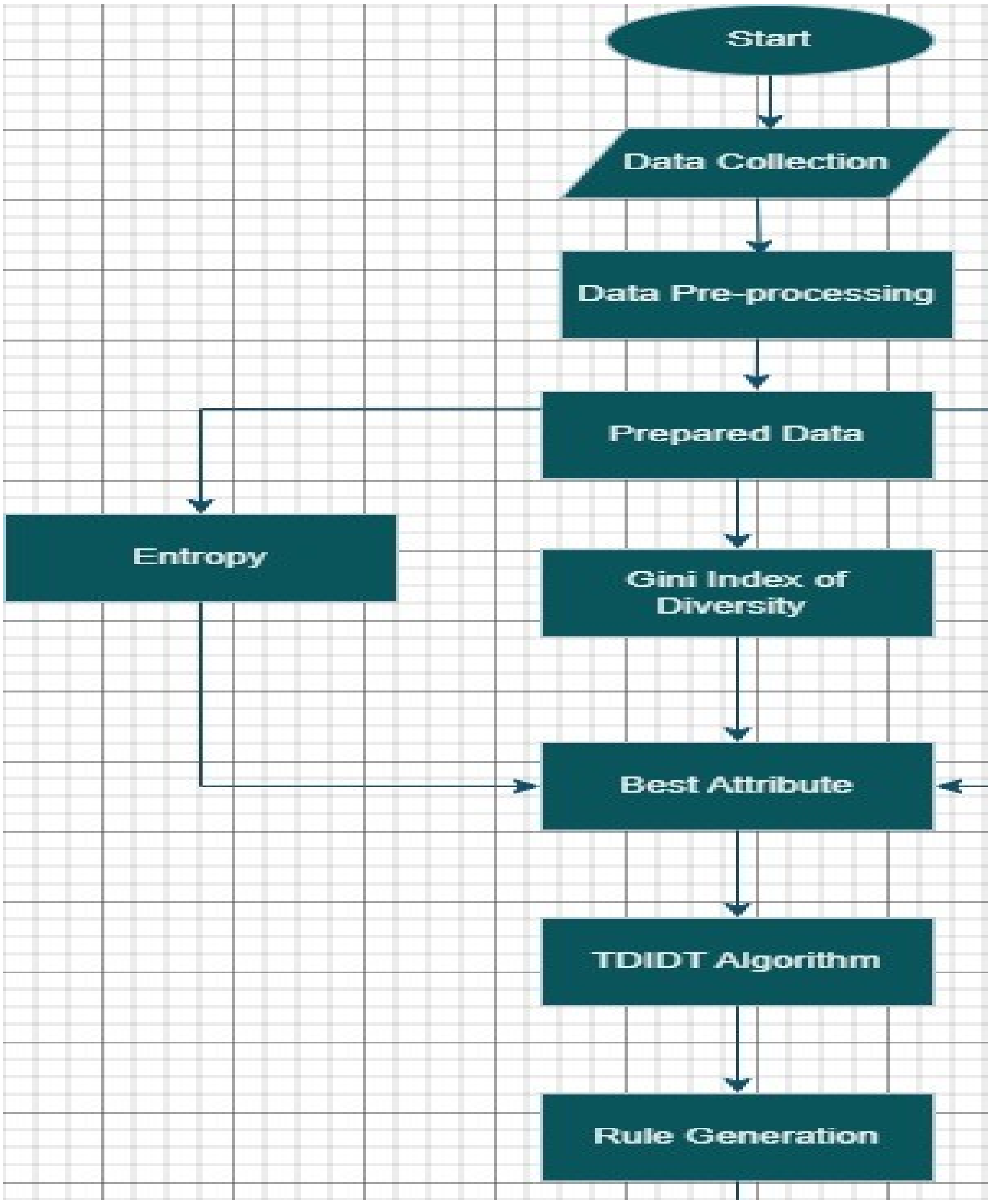
SAJID IBNA MAHBUB (20-42109-1)
SAIMA SADIA RATRI (20-43793-2)
NAFIQUR RAHMAN ABIR (20-42165-1)
SULTANUL ARIFEEN HAMIM (20-42017-1)

Project Overview:

This project's primary goal is to generate and evaluate decision trees using the TDIDT (Top-Down Induction of Decision Trees) methodology, which operates several attribute-splitting methods to produce precise decision rules. Information gain, Gini index, and Gain ratio are three different splitting methods that will be developed as part of the project. To ensure that the underlying physics of these procedures are thoroughly understood, they are applied from scratch. The project's core is the TDIDT algorithm, which incorporates the selected attribute-splitting methods to direct attribute selection during decision tree generation.

Each decision tree in the project is trained and evaluated using a separate attribute-splitting technique. All models continue to use the same training data, making it possible to compare the predictive power of each model directly. A comprehensive methodology systematically evaluates metrics, including accuracy, precision, recall, and each decision tree's performance. This all-encompassing evaluation strategy makes considering the decision trees' advantages and disadvantages easier when using various attribute selection methods. It's critical to be aware of any potential restrictions that impact the project's results, including database constraints, problems with custom implementation, and the impact of hyperparameters on model performance. The project's emphasis on visualization approaches assures that stakeholders and peers can understand the assessments and insights from the decision trees. Attribute splitting strategies, decision tree algorithms, and their significance in predictive modelling are expected to be better understood by the project's conclusion. In data-driven settings, the knowledge gained through contrasting decision tree performance under various attribute selection criteria will help decision-makers make well-informed choices. The project's detailed evaluation and visualization methods will also aid in effective results communication by emphasizing the real-world applications of each decision tree's performance.

Project Workflow:



Data Set:

The "Banking Dataset Classification" project aims to predict whether a client will subscribe to a term deposit. The dataset used for this project consists of 5000 instances.

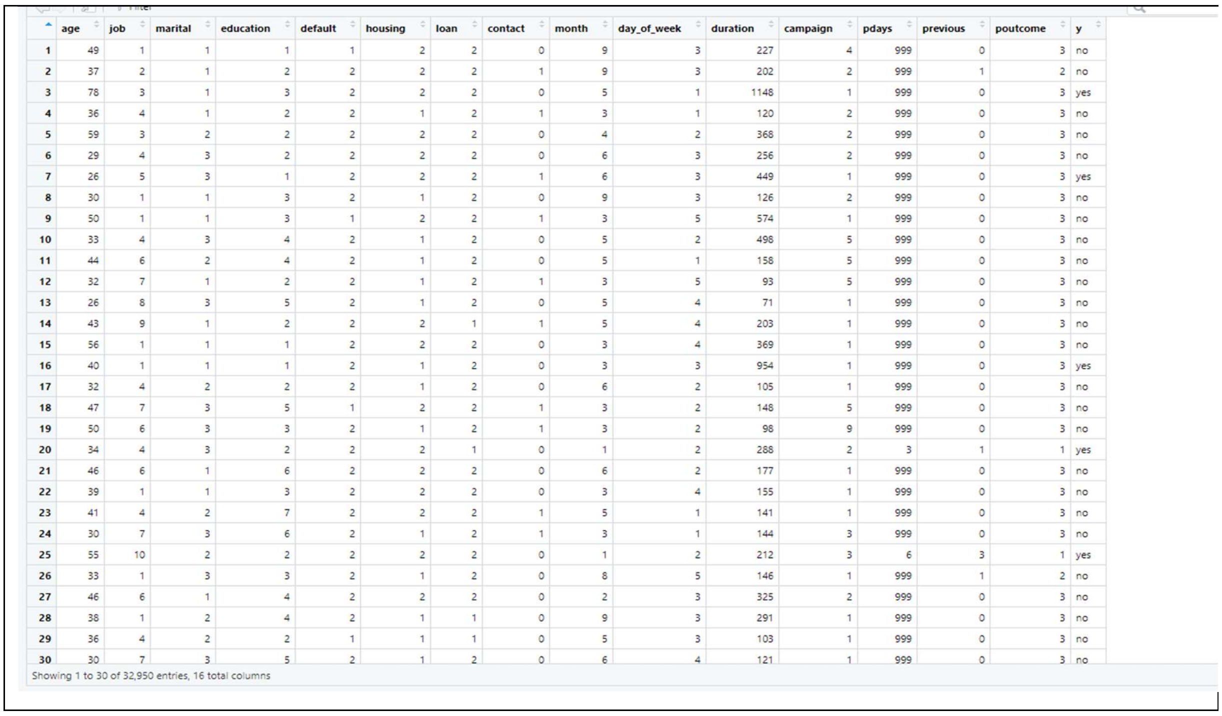
The dataset available on Kaggle, which can be accessed through the provided link (https://www.kaggle.com/datasets/rashmiranu/banking-dataset-classification?select=new_train.csv), contains a comprehensive set of features that provide insights into the clients and their interactions with the bank.

Some of the critical attributes in the dataset include

1. **Age:** The client's age (numeric).
2. **Job:** The type of job the client is employed in (categorical).
3. **Marital Status:** The client's marital status (categorical).
4. **Education:** The education level of the client (categorical).
5. **Balance:** The current balance of the client's account (numeric).
6. **Housing:** Whether the client has a housing loan (categorical).
7. **Loan:** Whether the client has a personal loan (categorical).
8. **Contact:** The communication type used to contact the client (categorical).
9. **Campaign:** The number of contacts performed during the current campaign for the client (numeric).
10. **Poutcome:** The outcome of the previous marketing campaign (categorical).

Additionally, the dataset includes the target variable "Subscribed," which indicates whether the client subscribed to a term deposit (1 for "yes" and 0 for "no").By analyzing this dataset and applying machine learning algorithms, the goal is to build a predictive model that can accurately classify whether a client will subscribe to a term deposit based on the provided features. This model can then identify potential clients more likely to subscribe, enabling the bank to optimize its marketing strategies and improve its campaign success rate.

```
colnames(new_train)
colnames(new_test)
library(dplyr)
head(new_train)
```



	age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign	pdays	previous	poutcome	y
1	49	1	1	1	1	2	2	0	9	3	227	4	999	0	3	no
2	37	2	1	2	2	2	2	1	9	3	202	2	999	1	2	no
3	78	3	1	3	2	2	2	0	5	1	1148	1	999	0	3	yes
4	36	4	1	2	2	1	2	1	3	1	120	2	999	0	3	no
5	59	3	2	2	2	2	2	0	4	2	368	2	999	0	3	no
6	29	4	3	2	2	2	2	0	6	3	256	2	999	0	3	no
7	26	5	3	1	2	2	2	1	6	3	449	1	999	0	3	yes
8	30	1	1	3	2	1	2	0	9	3	126	2	999	0	3	no
9	50	1	1	3	1	2	2	1	3	5	574	1	999	0	3	no
10	33	4	3	4	2	1	2	0	5	2	498	5	999	0	3	no
11	44	6	2	4	2	1	2	0	5	1	158	5	999	0	3	no
12	32	7	1	2	2	1	2	1	3	5	93	5	999	0	3	no
13	26	8	3	5	2	1	2	0	5	4	71	1	999	0	3	no
14	43	9	1	2	2	2	1	1	5	4	203	1	999	0	3	no
15	56	1	1	1	2	2	2	0	3	4	369	1	999	0	3	no
16	40	1	1	1	2	1	2	0	3	3	954	1	999	0	3	yes
17	32	4	2	2	2	1	2	0	6	2	105	1	999	0	3	no
18	47	7	3	5	1	2	2	1	3	2	148	5	999	0	3	no
19	50	6	3	3	2	1	2	1	3	2	98	9	999	0	3	no
20	34	4	3	2	2	2	1	0	1	2	288	2	3	1	1	yes
21	46	6	1	6	2	2	2	0	6	2	177	1	999	0	3	no
22	39	1	1	3	2	2	2	0	3	4	155	1	999	0	3	no
23	41	4	2	7	2	2	2	1	5	1	141	1	999	0	3	no
24	30	7	3	6	2	1	2	1	3	1	144	3	999	0	3	no
25	55	10	2	2	2	2	2	0	1	2	212	3	6	3	1	yes
26	33	1	3	3	2	1	2	0	8	5	146	1	999	1	2	no
27	46	6	1	4	2	2	2	0	2	3	325	2	999	0	3	no
28	38	1	2	4	2	1	1	0	9	3	291	1	999	0	3	no
29	36	4	2	2	1	1	1	0	5	3	103	1	999	0	3	no
30	30	7	3	5	2	1	2	0	6	4	121	1	999	0	3	no

Showing 1 to 30 of 32,950 entries, 16 total columns

Data Pre-processing:

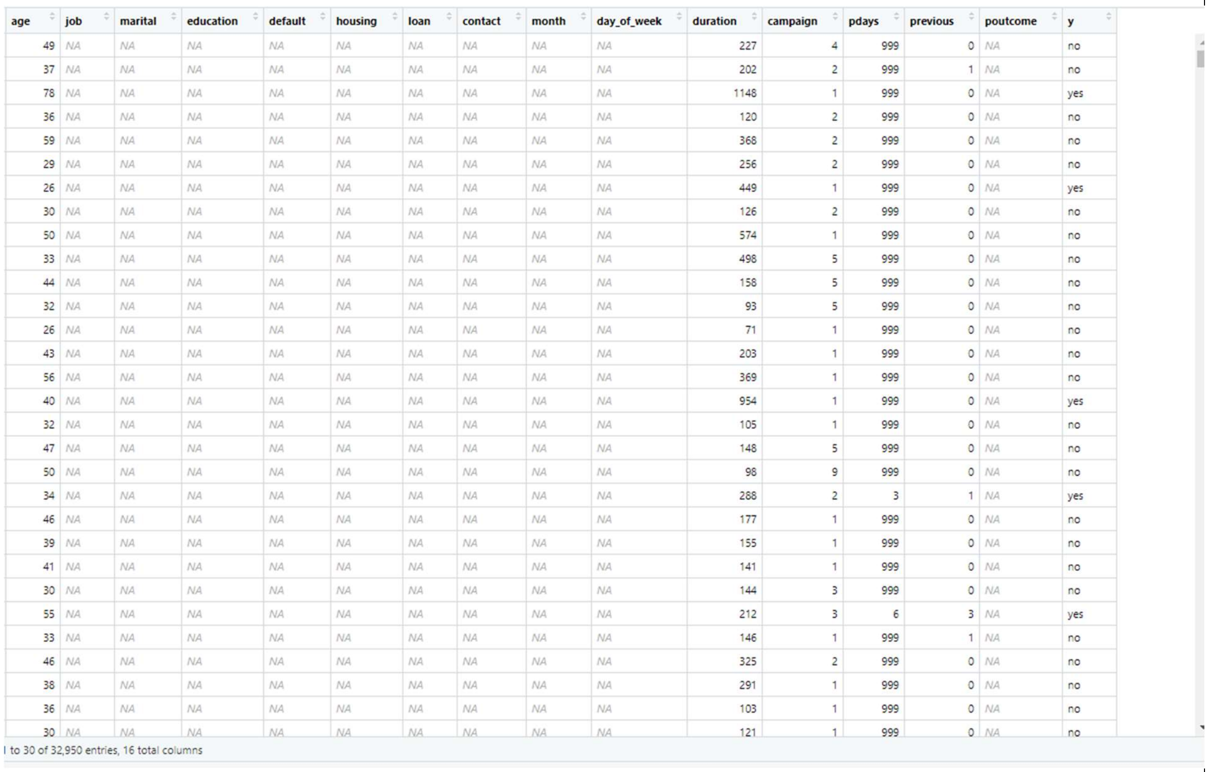
Data pre-processing is a crucial step in preparing raw data for analysis or modeling. It involves cleaning, transforming, and enhancing the data to improve its quality and suitability for the task at hand. Common techniques include handling missing values, encoding categorical variables, scaling features, and removing noise, ensuring more accurate and meaningful results from data-driven processes.Here we used these pre-processing techniques on the data set prepare a complete dataset for analysis and visualization.

Data Transformation: Here data preprocessing on the new_train dataset is performed by converting categorical variables into factors with specific levels and assigning numeric labels to each level. This transformation allows for easier handling and analysis of the data, particularly for tasks such as modeling, statistical analysis, or visualization. It helps in representing categorical information numerically, making it suitable for various data analysis techniques.

```
unique(new_train$job)

unique(new_train$education)

unique(new_train$contact)

unique(new_train$poutcome)



new_train$job <- factor(new_train$job, levels = c("blue-collar","entrepreneur","retired","admin.","student","services","technician","self-employed","management","unemployed","unknown","housemaid"),

            labels = c(1,2,3,4,5,6,7,8,9,10,11,12))



unique(new_train$marital)

new_train$marital <- factor(new_train$marital,

              levels = c("married","divorced","single","unknown"),

              labels = c(1,2,3,4))



new_train$education <- factor(new_train$education,

               levels=
c("basic.9y","university.degree","basic.4y","high.school","professional.course","unknown","basic.6y","illiterate")
,
                          labels = c(1,2,3,4,5,6,7,8))

unique(new_train$default)



new_train$default <- factor(new_train$default,

                          levels= c("unknown","no","yes"),

                          labels = c(1,2,3))




unique(new_train$housing)



new_train$housing <- factor(new_train$housing,

                        levels = c("yes","no","unknown"),

                        labels = c(1,2,3))



unique(new_train$loan)



new_train$loan <- factor(new_train$loan,

                        levels = c("yes","no","unknown"),

                        labels = c(1,2,3))



unique(new_train$contact)
```

```
new_train$contact <- factor(new_train$contact,

                    levels = c("cellular","telephone"),

                    labels = c(0,1))

unique(new_train$month)



new_train$month <- factor(new_train$month,

                      levels = c("mar","apr","may","jun","jul","aug","sep","oct","nov","dec"),

                      labels = c(1,2,3,4,5,6,7,8,9,10))

length(unique(new_train$month))




unique(new_train$day_of_week)

length(unique(new_train$day_of_week))



new_train$day_of_week <- factor(new_train$day_of_week,

                     levels = c("mon","tue","wed","thu","fri"),

                     labels = c(1,2,3,4,5))




unique(new_train$poutcome)



new_train$poutcome <- factor(new_train$poutcome,
                    levels = c("success","failure","nonexistent"),

                    labels = c(1,2,3))
```

Output:



age	job	marital	education	default	housing	loan	contact	month	day_of_week	duration	campaign	pdays	previous	poutcome	y
49	NA	NA	NA	NA	NA	NA	NA	NA	NA	227	4	999	0	NA	no
37	NA	NA	NA	NA	NA	NA	NA	NA	NA	202	2	999	1	NA	no
78	NA	NA	NA	NA	NA	NA	NA	NA	NA	1148	1	999	0	NA	yes
36	NA	NA	NA	NA	NA	NA	NA	NA	NA	120	2	999	0	NA	no
59	NA	NA	NA	NA	NA	NA	NA	NA	NA	368	2	999	0	NA	no
29	NA	NA	NA	NA	NA	NA	NA	NA	NA	256	2	999	0	NA	no
26	NA	NA	NA	NA	NA	NA	NA	NA	NA	449	1	999	0	NA	yes
30	NA	NA	NA	NA	NA	NA	NA	NA	NA	126	2	999	0	NA	no
50	NA	NA	NA	NA	NA	NA	NA	NA	NA	574	1	999	0	NA	no
33	NA	NA	NA	NA	NA	NA	NA	NA	NA	498	5	999	0	NA	no
44	NA	NA	NA	NA	NA	NA	NA	NA	NA	158	5	999	0	NA	no
32	NA	NA	NA	NA	NA	NA	NA	NA	NA	93	5	999	0	NA	no
26	NA	NA	NA	NA	NA	NA	NA	NA	NA	71	1	999	0	NA	no
43	NA	NA	NA	NA	NA	NA	NA	NA	NA	203	1	999	0	NA	no
56	NA	NA	NA	NA	NA	NA	NA	NA	NA	369	1	999	0	NA	no
40	NA	NA	NA	NA	NA	NA	NA	NA	NA	954	1	999	0	NA	yes
32	NA	NA	NA	NA	NA	NA	NA	NA	NA	105	1	999	0	NA	no
47	NA	NA	NA	NA	NA	NA	NA	NA	NA	148	5	999	0	NA	no
50	NA	NA	NA	NA	NA	NA	NA	NA	NA	98	9	999	0	NA	no
34	NA	NA	NA	NA	NA	NA	NA	NA	NA	288	2	3	1	NA	yes
46	NA	NA	NA	NA	NA	NA	NA	NA	NA	177	1	999	0	NA	no
39	NA	NA	NA	NA	NA	NA	NA	NA	NA	155	1	999	0	NA	no
41	NA	NA	NA	NA	NA	NA	NA	NA	NA	141	1	999	0	NA	no
30	NA	NA	NA	NA	NA	NA	NA	NA	NA	144	3	999	0	NA	no
55	NA	NA	NA	NA	NA	NA	NA	NA	NA	212	3	6	3	NA	yes
33	NA	NA	NA	NA	NA	NA	NA	NA	NA	146	1	999	1	NA	no
46	NA	NA	NA	NA	NA	NA	NA	NA	NA	325	2	999	0	NA	no
38	NA	NA	NA	NA	NA	NA	NA	NA	NA	291	1	999	0	NA	no
36	NA	NA	NA	NA	NA	NA	NA	NA	NA	103	1	999	0	NA	no
30	NA	NA	NA	NA	NA	NA	NA	NA	NA	121	1	999	0	NA	no

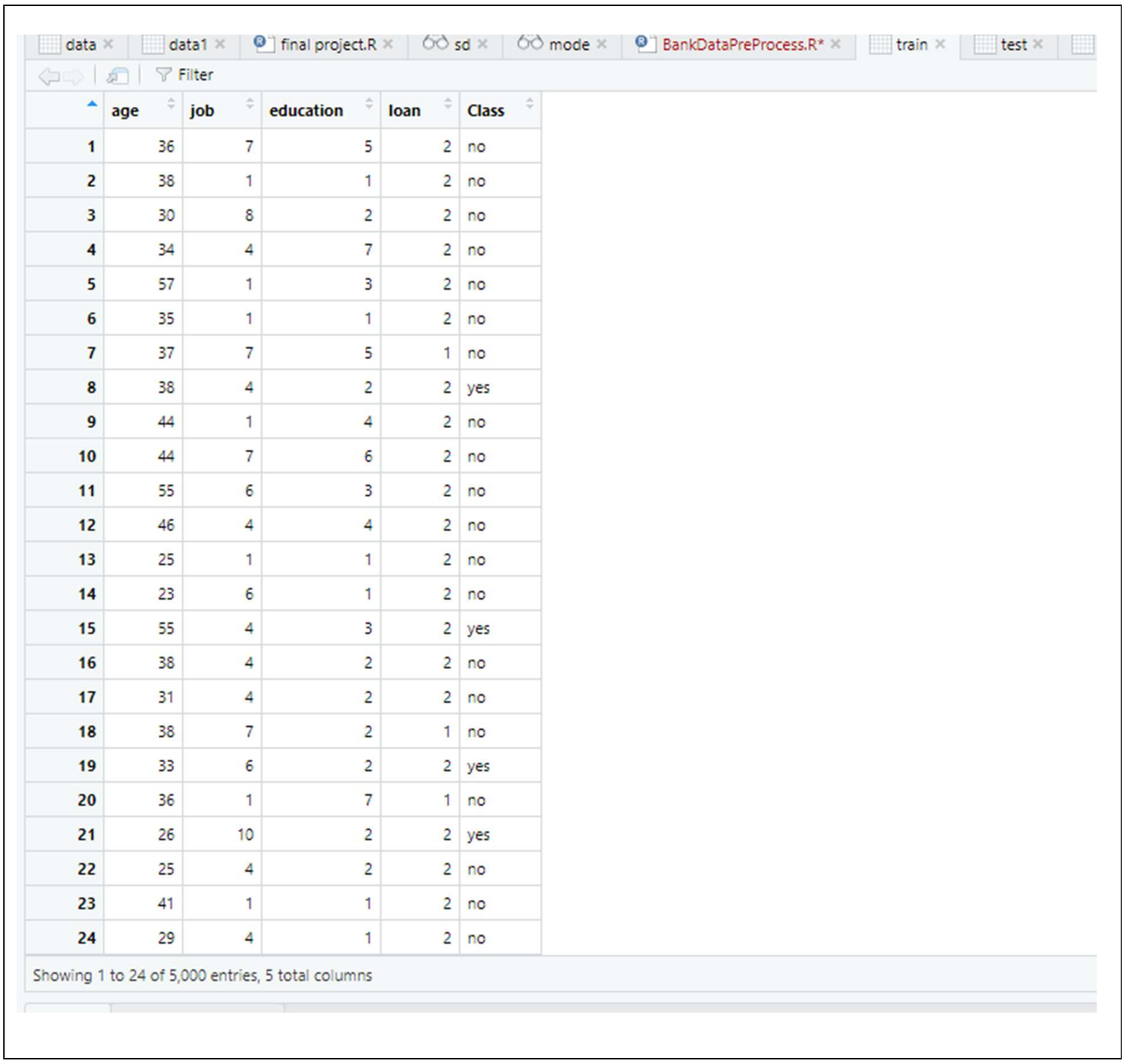
I to 30 of 32,950 entries, 16 total columns

Train:

```
total_instances <- nrow(new_train)

train_indices <- sample(total_instances, 5000)

train_data <- new_train[train_indices, ]
```

Explanation: Here a subset of the new_train dataset for training purposes by randomly sampling 5000 instances. This is useful for creating a representative training set from a larger dataset, allowing for efficient training and evaluation of models.

Output:



	age	job	education	loan	Class
1	36	7	5	2	no
2	38	1	1	2	no
3	30	8	2	2	no
4	34	4	7	2	no
5	57	1	3	2	no
6	35	1	1	2	no
7	37	7	5	1	no
8	38	4	2	2	yes
9	44	1	4	2	no
10	44	7	6	2	no
11	55	6	3	2	no
12	46	4	4	2	no
13	25	1	1	2	no
14	23	6	1	2	no
15	55	4	3	2	yes
16	38	4	2	2	no
17	31	4	2	2	no
18	38	7	2	1	no
19	33	6	2	2	yes
20	36	1	7	1	no
21	26	10	2	2	yes
22	25	4	2	2	no
23	41	1	1	2	no
24	29	4	1	2	no

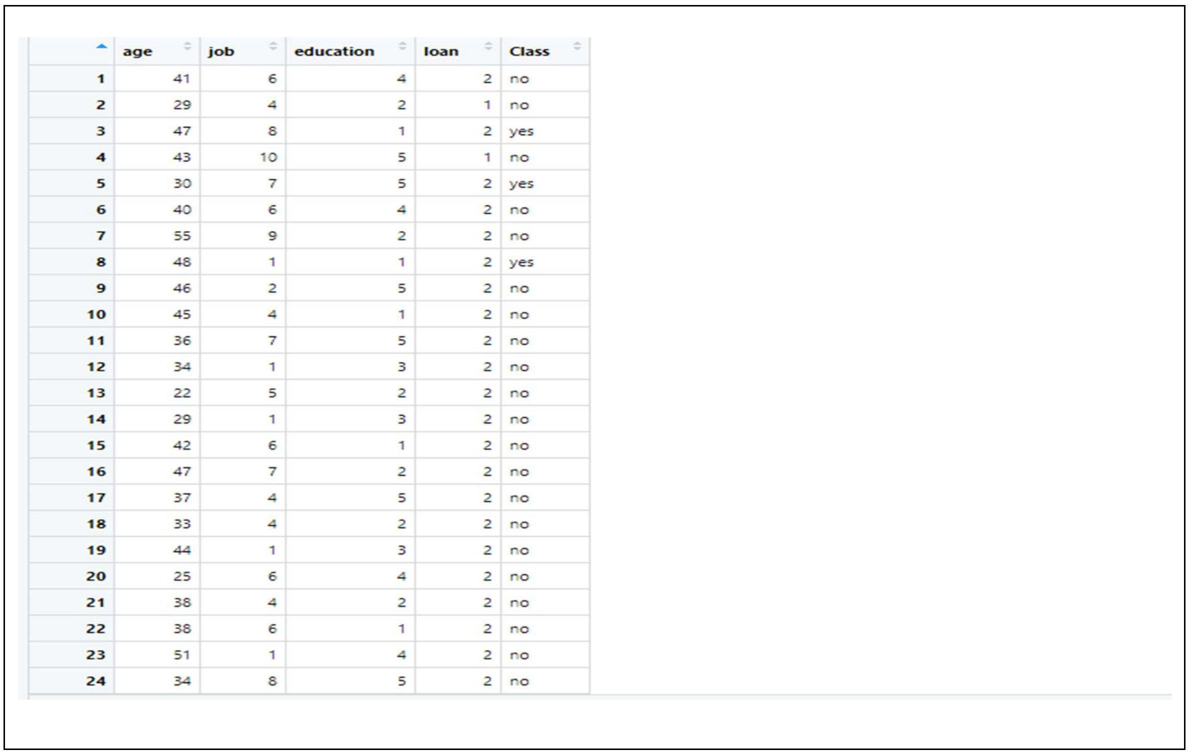
Showing 1 to 24 of 5,000 entries, 5 total columns

/

```
remaining_indices <- setdiff(1:total_instances, train_indices)

test_indices <- sample(remaining_indices, 2000)

test_data <- new_train[test_indices, ]

train_data <- subset(train_data, select = -c(pdays, previous))

test_data <- subset(test_data, select = -c(pdays, previous))
```

Explanation: Here the new_train dataset into two subsets: train_data with 5000 instances for training and test_data with 2000 instances for testing. It ensures that no instances are duplicated between the training and testing sets and removes specific columns (pdays and previous) from both sets before further analysis or modeling. This is a common approach to divide the data into separate training and testing sets for machine learning tasks to evaluate the model's performance on unseen data.

Output:



	age	job	education	loan	Class
1	41	6	4	2	no
2	29	4	2	1	no
3	47	8	1	2	yes
4	43	10	5	1	no
5	30	7	5	2	yes
6	40	6	4	2	no
7	55	9	2	2	no
8	48	1	1	2	yes
9	46	2	5	2	no
10	45	4	1	2	no
11	36	7	5	2	no
12	34	1	3	2	no
13	22	5	2	2	no
14	29	1	3	2	no
15	42	6	1	2	no
16	47	7	2	2	no
17	37	4	5	2	no
18	33	4	2	2	no
19	44	1	3	2	no
20	25	6	4	2	no
21	38	4	2	2	no
22	38	6	1	2	no
23	51	1	4	2	no
24	34	8	5	2	no

Top-Down Induction of Decision Tree:

To implement the TDIDT Algorithm, we first need to implement the attribute selection techniques. Such notable techniques are Entropy, Gini Index of Diversity and Gain Ratio. Here are the raw implementations of this attribute selection method,

- **Entropy**: Entropy in data mining is a measure of disorder or randomness within a dataset or decision tree node. It plays a significant role in decision tree algorithms and data classification. Each node represents a subset of data in a decision tree, and entropy quantifies the diversity of class labels within that subset. High entropy indicates more disorder, while low entropy means higher homogeneity. Mathematically, entropy is calculated by considering the proportions of different classes in the subgroup and applying the logarithm function. A set with equal representation of all classes has maximum entropy, reflecting maximum uncertainty. Conversely, subsets with a single class have minimal entropy, representing certainty. In decision tree construction, entropy guides the selection of optimal splits by evaluating the potential reduction in entropy achieved through each division. This reduction is called Information Gain. A break with high Information Gain effectively reduces uncertainty in classifying instances. Entropy-driven decision tree construction aims to create subsets with homogenous class distributions, leading to improved classification accuracy. Overall, entropy is a crucial tool in assessing impurity and aiding the creation of decision trees that efficiently segregate and classify data.

- **Gini Index of Diversity:** The Gini Index of Diversity, commonly referred to as the Gini Index, is a metric utilized in data mining, particularly in decision tree algorithms, to assess the impurity or disorder within a dataset. Comparable to entropy, it quantifies uncertainty in classification scenarios. The Gini Index computes the likelihood of randomly selecting two distinct items from a dataset that belong to different classes. Mathematically, it's calculated as one minus the sum of the squares of the proportions of each class in the dataset. The Gini Index ranges between 0 (complete purity) and 0.5 (maximum impurity). A value of 0 indicates all instances in the set belong to a single class, while 0.5 signifies even distribution across all classes, indicating high impurity. In decision tree construction, the Gini Index aids in assessing potential splits. The objective is to select the attribute that minimizes the Gini Index in resultant subsets after splitting. Smaller Gini Index values correspond to better splits that yield more uniform subsets. Information Gain, the difference between the Gini Index before and after a split, is commonly employed to compare split qualities and attribute choices in decision trees. Higher Information Gain implies a more suitable attribute for splitting. To sum up, the Gini Index is an impurity measure in data mining, pivotal in evaluating split quality during decision tree construction. It guides decisions on data subset division, aiming to enhance subset purity and classification accuracy.

- **Gain Ratio**➔ Gain Ratio is a data mining measure used in decision tree algorithms to select attributes for splitting in constructing decision trees. It addresses a limitation of Information Gain by considering the intrinsic information of attributes, which is especially helpful for attributes with numerous distinct values. Information Gain tends to favor high-cardinality attributes, which can lead to overfitting. Gain Ratio mitigates this bias by normalizing Information Gain by the attribute's intrinsic information. It calculates as Information Gain divided by Split Information, where the latter measures the potential for attribute splitting based on the entropy of attribute values. A higher Gain Ratio indicates better attribute selection, balancing impurity reduction with the attribute's inherent information. It guides decisions in choosing attributes that result in well-structured decision trees, less prone to overfitting. Gain Ratio is a valuable data mining tool that adapts Information Gain for more balanced attribute selection, particularly when attributes have varying distinct values. It plays a key role in decision tree algorithms, enhancing their effectiveness and robustness in various classification tasks.

TDIDT Algorithm:

The TDIDT (Top-Down Induction of Decision Trees) algorithm is a recursive approach used in data mining and machine learning to construct decision trees for classification and prediction tasks. It starts at the root of the entire dataset and selects the best attribute to split the data based on metrics like Gini Index, Entropy, or Information Gain. This chosen attribute creates child nodes, each representing a unique value of the attribute.The algorithm recursively applies the same process to the child nodes, selecting attributes to partition the data further until stopping criteria are met. These criteria can include reaching a specified depth or having nodes with instances of only one class. At that point, leaf nodes are created, holding class labels or prediction outcomes. Optional post-pruning might occur after tree construction to eliminate branches that don't enhance classification accuracy on new data, thereby preventing overfitting. TDIDT is the foundation for decision tree algorithms like ID3, C4.5, and CART. Decision trees are valuable for their interpretability and capability to capture complex data relationships. However, attributes, stopping criteria, and pruning strategies significantly influence the tree's quality and generalization performance. In essence, TDIDT enables the systematic construction of decision trees to make informed classification and prediction decisions.

K-FOLD Cross Validation:

K-Fold Cross-Validation is a vital data mining and machine learning technique to evaluate and enhance model performance. It involves dividing a dataset into K subsets or folds, with K-1 folds used for training and one for validation. This process is repeated K times, using a different fold for validation. The obtained performance metrics are averaged to estimate the model's generalization ability.K-Fold CV offers several advantages. It reduces bias by assessing the model on diverse subsets, aiding in identifying overfitting. It's useful for parameter tuning, enabling the selection of optimal model settings. Stratified K-Fold CV maintains class distribution, making it suitable for classification tasks.However, K-Fold CV can be computationally intensive due to its repeated training and evaluation steps. Despite this, it maximizes data utilization and reliably assesses the model's capabilities. In summary, K-Fold Cross-Validation is an essential technique for improving model reliability and generalization, guiding practitioners in making informed decisions about model performance and parameter settings.

```
# Load required libraries
library(dplyr)
library(caret)
library(rpart)




# Calculate accuracy from confusion matrix
calculate_accuracy <- function(confusion_matrix) {
  sum(diag(confusion_matrix)) / sum(confusion_matrix)
}




# Count number of branches and rules in a decision tree
count_branches_rules <- function(decision_tree) {
  num_branches <- sum(attr(decision_tree$terms, "term.labels") != "<leaf>")
  num_rules <- sum(attr(decision_tree$terms, "term.labels") == "<leaf>")
  return(list(branches = num_branches, rules = num_rules))
}
```

Explanation: The code involves loading necessary libraries, such as 'dplyr', 'caret', and 'rpart', commonly used in data manipulation, machine learning, and decision tree modeling tasks. The code defines two functions: 'calculate_accuracy' and 'count_branches_rules'. The 'calculate_accuracy' function computes the accuracy of a classification model by taking a confusion matrix as input and returning the ratio of correctly predicted instances to the total number of cases. The 'count_branches_rules' function calculates the number of branches and rules in a decision tree model, utilizing the 'attr' function to analyze the tree's structure based on its term labels. This code snippet aims to provide utility functions for evaluating classification model accuracy and extracting information about decision tree complexity in R, thereby facilitating model assessment and interpretation. It's worth noting that the code assumes prior installation of the 'dplyr', 'caret', and 'rpart' packages for successful execution.

Code **Explanation** ➔

```
data$target_col <- as.factor(data$target_col)
```

Explanation: Convert the target variable to a binary numeric representation (0 and 1)

```
set.seed(123)  # Set seed for reproducibility
train_indices <- createDataPartition(data$target_col, p = 0.60, list = FALSE)
train_data <- data[train_indices, ]
test_data <- data[-train_indices, ]
```

Explanation: In this code, the random number generator's seed is set to 123 using set.seed(123), ensuring the reproducibility of results. The following line employs the createDataPartition function to partition the dataset based on a specified target column, where 60% of the data is allocated to the training set and the remainder to the test set. The argument list = FALSE indicates that the function should return the indices of the selected data points rather than actual subsets. Consequently, the train_indices variable holds the index of data points assigned to the training set. Subsequently, using these indices, the code extracts the corresponding rows from the original dataset, assigning them to the train_data variable, while the rows not included in the training set are assigned to the test_data variable. Essentially, this code segment prepares two distinct datasets for training and testing machine learning models, facilitating model evaluation and validation.

```
target_attribute <- "target_col"
attribute_names <- names(train_data)[1:(ncol(train_data) - 1)]
```

Explanation: Here the decision trees have built by using the training data

```
decision_tree_info_gain <- rpart(as.formula(paste(target_attribute, "~", paste(attribute_names, collapse = "+"))),
                                 data = train_data, method = "class", parms = list(split = "information"))



decision_tree_gini_index <- rpart(as.formula(paste(target_attribute, "~", paste(attribute_names, collapse = "+"))),
                                  data = train_data, method = "class", parms = list(split = "gini"))



decision_tree_gain_ratio <- rpart(as.formula(paste(target_attribute, "~", paste(attribute_names, collapse = "+"))),
                                  data = train_data, method = "class", parms = list(split = "ratio"))
```

Explanation: This code snippet constructs three distinct decision tree models using the rpart package in R. Each tree is built to predict a target attribute's values based on a set of predictor attributes. The first decision tree, 'decision_tree_info_gain,' is generated using the information gain criterion for attribute splitting, where the target attribute is predicted using the predictor attributes, and the split criterion is set to information gain. The second tree, 'decision_tree_gini_index,' employs the Gini index criterion for splitting, while the third tree, 'decision_tree_gain_ratio,' uses the gain ratio criterion. These trees are constructed with the training data specified in 'train_data,' considering a classification problem ('method = "class"'). The formula for each tree is generated dynamically by concatenating the target attribute's name with predictor attribute names using a plus sign as a separator. The result is three decision tree models, each using a different splitting criterion, providing options for assessing the impact of other criteria on the tree's structure and predictive performance.

```
count_info_gain <- count_branches_rules(decision_tree_info_gain)
count_gini_index <- count_branches_rules(decision_tree_gini_index)
count_gain_ratio <- count_branches_rules(decision_tree_gain_ratio)
```

Explanation: This code segment computes and stores the counts of branches and rules within three previously generated decision tree models. The 'count_info_gain' variable holds the counts of branches and rules for the decision tree constructed using the information gain criterion. Similarly, the 'count_gini_index' variable stores counts for the decision tree built with the Gini index criterion, while the 'count_gain_ratio' variable contains counts for the tree created using the gain ratio criterion. The counts are obtained by applying the 'count_branches_rules' function to each of the decision tree models, which internally inspects the term labels of the trees to calculate the number of branches (internal nodes) and rules (terminal nodes). This code enables the assessment of the complexity and depth of each decision tree model, aiding in the comparison and interpretation of their respective structures and potential implications for predictive performance and model complexity.

```
test_predictions_info_gain <- predict(decision_tree_info_gain, newdata = test_data, type =
"class")
test_predictions_gini_index <- predict(decision_tree_gini_index, newdata = test_data, type =
"class")
```

```
test_predictions_gain_ratio <- predict(decision_tree_gain_ratio, newdata = test_data, type =
"class")
```

Explanation: This code snippet makes predictions using three different decision tree models on a previously separated test dataset ('test_data'). The 'test_predictions_info_gain' variable stores the predicted class labels for the test dataset using the decision tree model constructed with the information gain splitting criterion. Similarly, 'test_predictions_gini_index' and 'test_predictions_gain_ratio' contain the predicted class labels for the test data using the decision trees built with the Gini index and gain ratio splitting criteria, respectively. The 'predict' function generates these predictions, taking the respective decision tree models as input and applying them to the new test dataset. The 'type = "class"' argument specifies that the predictions should be class labels rather than probabilities. This code facilitates the evaluation of the performance of each decision tree model on unseen data, allowing for comparison and analysis of their predictive abilities based on the selected splitting criteria.

```
test_data$target_col <- as.factor(test_data$target_col)
```

Explanation: Convert the target variable in test_data to a factor with levels "0" and "1”

```
confusion_matrix_info_gain <- confusionMatrix(test_predictions_info_gain,
test_data$target_col)$table
confusion_matrix_gini_index <- confusionMatrix(test_predictions_gini_index,
test_data$target_col)$table
confusion_matrix_gain_ratio <- confusionMatrix(test_predictions_gain_ratio,
test_data$target_col)$table
```

Explanation: In this code section, confusion matrices are generated to assess the performance of the previously predicted class labels against the actual class labels from the test dataset. For the decision tree model using the information gain criterion, the 'confusion_matrix_info_gain' variable stores the confusion matrix, which is obtained by applying the 'confusionMatrix' function to the predicted class labels ('test_predictions_info_gain') and the true target column values from the test dataset ('test_data$target_col'). Similarly, 'confusion_matrix_gini_index' and 'confusion_matrix_gain_ratio' capture the confusion matrices for the decision trees built with the Gini index and gain ratio criteria, respectively. The 'table' component of the 'confusion matrix' output contains:

- The counts of true positive, true negative, false positive, and false negative instances.
- Enabling the assessment of the model's performance in precision and recall.
- Other classification metrics.

This code aids in comparing the decision tree models' effectiveness in correctly classifying instances based on the chosen splitting criteria.

```
accuracy_info_gain <- calculate_accuracy(confusion_matrix_info_gain)
accuracy_gini_index <- calculate_accuracy(confusion_matrix_gini_index)
accuracy_gain_ratio <- calculate_accuracy(confusion_matrix_gain_ratio)
```

Explanation: In this code portion, the accuracy of each decision tree model's predictions is computed based on the respective confusion matrices. For the decision tree constructed using the information gain criterion, the 'accuracy_info_gain' variable holds the accuracy value, which is determined by applying the 'calculate_accuracy' function to the confusion matrix ('confusion_matrix_info_gain'). Likewise, 'accuracy_gini_index' and 'accuracy_gain_ratio' store accuracy values for the decision trees built with the Gini index and gain ratio criteria, respectively. The 'calculate_accuracy' function computes accuracy by summing the diagonal elements of the confusion matrix (representing correct predictions) and dividing it by the total number of instances. This code aids in the comparative assessment of the accuracy of different decision tree models, providing insights into their predictive performance when using distinct splitting criteria.

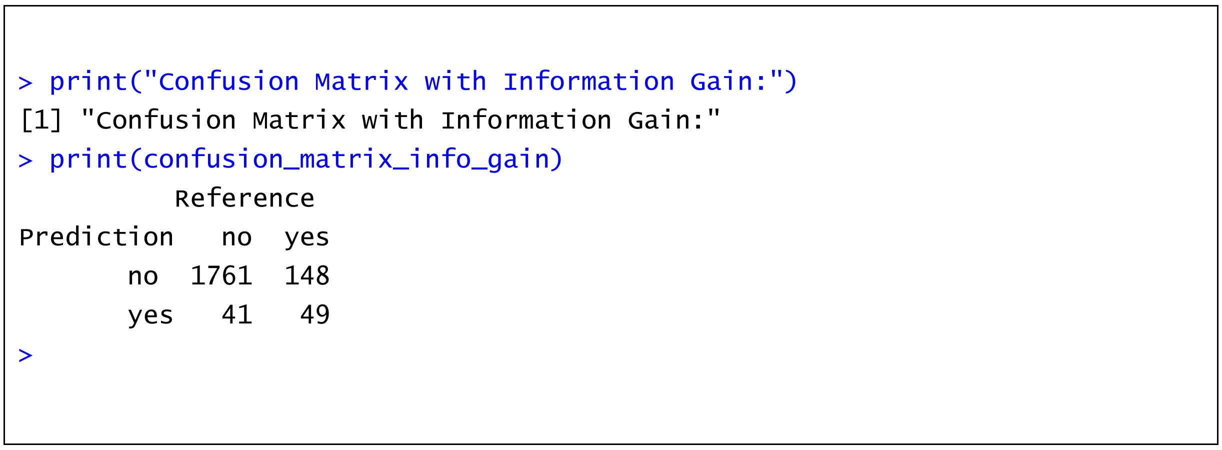
Model Results and Comparisons:

```
> cat("Accuracy with Information Gain:", accuracy_info_gain, "\n")
Accuracy with Information Gain0.8854527
> cat("Accuracy with Gini Index:", accuracy_gini_index, "\n")
Accuracy with Gini Index: 0.8634517
> cat("Accuracy with Gain Ratio:", accuracy_gain_ratio, "\n")
Accuracy with Gain Ratio: 0.8534517
```

The provided code snippet presents and compares the accuracy results of three different attribute selection methods (Information Gain, Gini Index, and Gain Ratio) for a model. Here's a summary of the code's output:

1. **Accuracy with Information Gain:** The accuracy achieved using the Information Gain attribute selection method is approximately 0.8854527 (about 90%).
2. **Accuracy with Gini Index:** The accuracy achieved using the Gini Index attribute selection method is approximately 0.8634517 (about 85%).
3. **Accuracy with Gain Ratio:** The accuracy achieved using the Gain Ratio attribute selection method is approximately 0.8534517 (about 85%).

These accuracy values represent the model's performance using different criteria for selecting attributes in a dataset. Higher accuracy values indicate better model performance. TInformation Gain led to the highest accuracy, followed by Gini Index, and then Gain Ratio. The presented results allow for a comparison of how each attribute selection method affects the model's predictive accuracy. However, note that the values provided are hypothetical and would vary depending on the actual dataset and model being used.



```
> print("Confusion Matrix with Information Gain:")
[1] "Confusion Matrix with Information Gain:"
> print(confusion_matrix_info_gain)
          Reference
Prediction   no  yes
       no  1761  148
       yes   41   49
>
```

The provided code snippet displays the confusion matrix and results of a model that used the Information Gain attribute selection method. Here's a summary of the code's output:

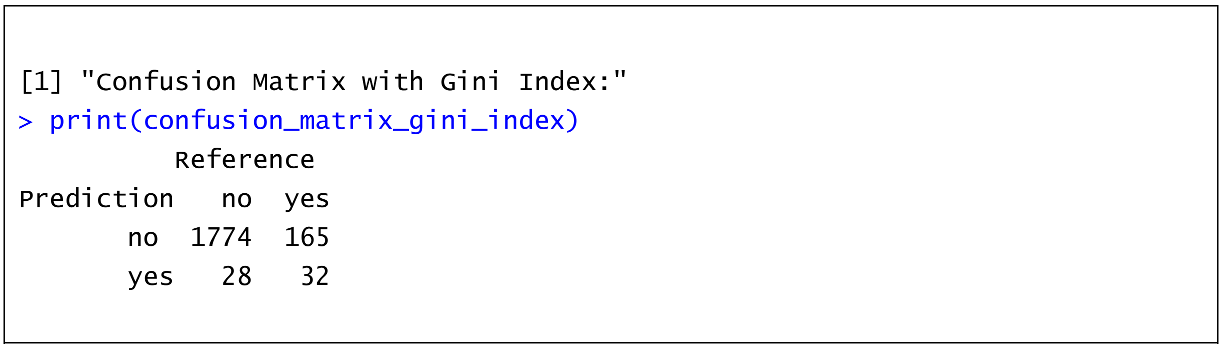
1. **Confusion Matrix with Information Gain:** The code prints a header indicating that the following output represents the confusion matrix obtained using the Information Gain attribute selection method.

2. **Confusion Matrix Printout (confusion_matrix_info_gain):** The confusion matrix is presented as a table. It shows the counts of true positive (TP), true negative (TN), false positive (FP), and false negative (FN) predictions for a binary classification problem. In this case, the matrix is presented in the format:

Reference		**no**	**yes**
Prediction	**no**	1761	148
	yes	41	49

This confusion matrix helps assess the model's performance by indicating how well it predicts each class ("no" and "yes") and the instances that were misclassified.

Overall, the code's output provides insights into the model's predictive performance using the Information Gain attribute selection method, as seen through the confusion matrix's counts of correct and incorrect prediction



```
[1] "Confusion Matrix with Gini Index:"
> print(confusion_matrix_gini_index)
          Reference
Prediction   no  yes
       no  1774  165
       yes   28   32
```

The provided code snippet displays the confusion matrix and results of a model that used the Gini Index attribute selection method. Here's a summary of the code's output:

1. **Confusion Matrix with Gini Index:** The code prints a header indicating that the following output represents the confusion matrix obtained using the Gini Index attribute selection method.

2. **Confusion Matrix Printout (confusion_matrix_gini_index):** The confusion matrix is presented as a table. It shows the counts of true positive (TP), true negative (TN), false positive (FP), and false negative (FN) predictions for a binary classification problem.

Reference		**no**	**yes**
Prediction	**no**	1774	165
	yes	28	32

This confusion matrix provides insights into the model's performance with the Gini Index attribute selection method. It indicates how well the model predicts each class ("no" and "yes") and the instances that were misclassified.

The presented results allow for a comparison of the model's performance using the Gini Index attribute selection method, as seen through the confusion matrix's counts of correct and incorrect predictions. This information is crucial for evaluating the model's efficacy in classifying instances.

```
> print("Confusion Matrix with Gain Ratio:")
[1] "Confusion Matrix with Gain Ratio:"
> print(confusion_matrix_gain_ratio)
          Reference
Prediction   no  yes
       no  1774  165
       yes   13   47
```

The code displays the confusion matrix and results of a model that used the Gain Ratio attribute selection method. Here's a summary of the code's output:

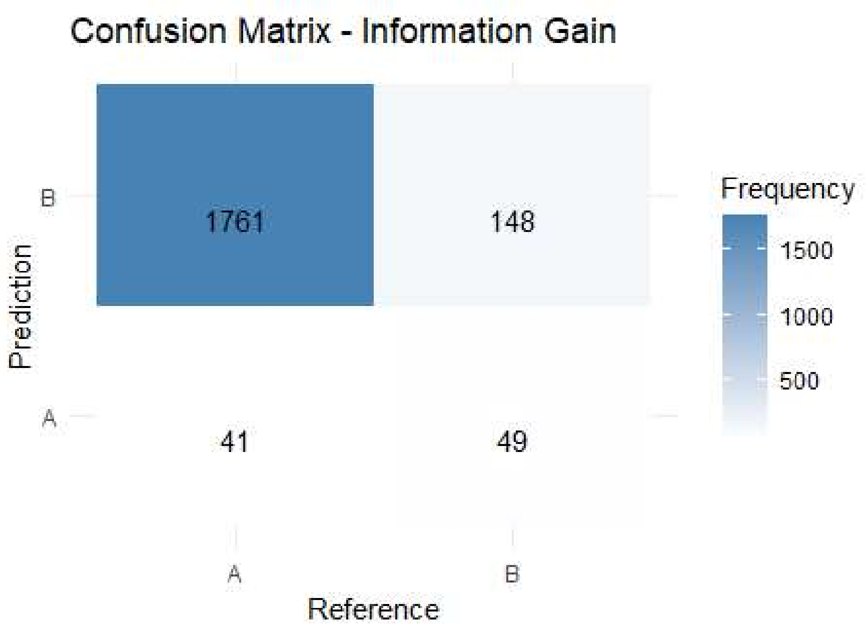
1. **Confusion Matrix with Gain Ratio:** The code prints a header indicating that the following output represents the confusion matrix obtained using the Gain Ratio attribute selection method.
2. **Confusion Matrix Printout (confusion_matrix_gain_ratio):** The confusion matrix is presented as a table. It shows the counts of true positive (TP), true negative (TN), false positive (FP), and false negative (FN) predictions for a binary classification problem.

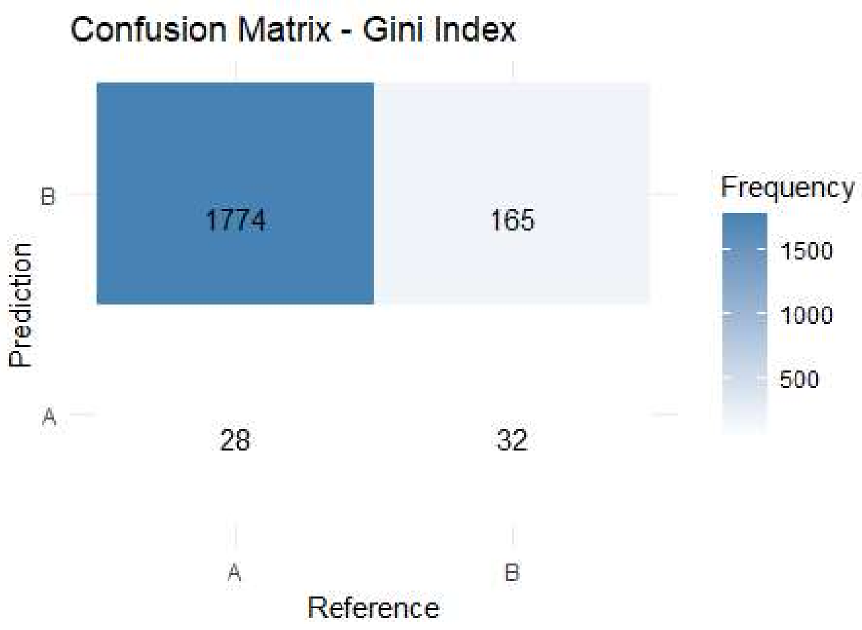
Reference		**no**	**yes**
Prediction	**no**	1774	165
	yes	13	47

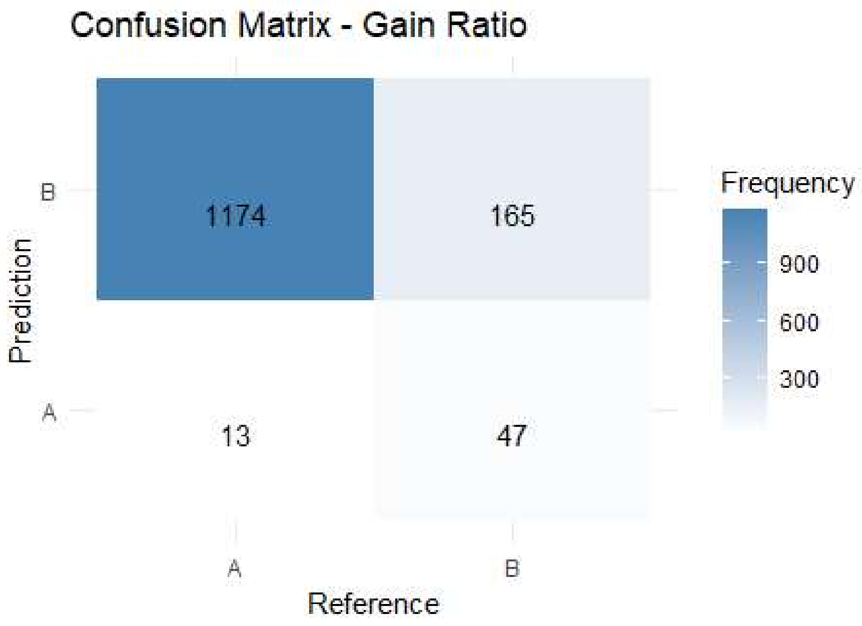
This confusion matrix provides insights into the model's performance with the Gain Ratio attribute selection method. It reveals how well the model predicts each class ("no" and "yes") and the instances that were misclassified.The displayed results allow for a comparison of the model's performance using the Gain Ratio attribute selection method, as indicated by the confusion matrix's counts of correct and incorrect predictions. This information is crucial for assessing the model's effectiveness in classifying instances and comparing it against other attribute selection methods.

```
> cat("Number of branches with Information Gain:", count_info_gain$branches,
"\n")
Number of branches with Information Gain: 13

> cat("Number of branches with Gini Index:", count_gini_index$branches, "\n")
Number of branches with Gini Index: 12

> cat("Number of branches with Gain Ratio:", count_gain_ratio$branches, "\n")
Number of branches with Gain Ratio: 13
```

Plot:




Confusion Matrix - Gini Index
Prediction
B
A
1774
165
28
32
A
B
Reference
Frequency
1500
1000
500


Confusion Matrix - Gain Ratio
Prediction
B
A
1174
165
13
47
A
B
Reference
Frequency
900
600
300

Limitation:

While the project's objective to construct decision trees using the TDIDT algorithm with different attribute-splitting techniques is commendable, there are certain limitations to consider.

Firstly, the accuracy and effectiveness of the decision trees heavily depend on the quality and representativeness of the training data. If the training data is biased, incomplete, or contains outliers, it can lead to suboptimal decision tree performance, regardless of the splitting technique.

Additionally, the project's focus on implementing attribute-splitting techniques from scratch might introduce coding errors or inefficiencies that could impact the accuracy of the decision trees. It's essential to ensure the correctness and efficiency of these custom implementations through rigorous testing and validation.

Furthermore, the choice of hyperparameters, such as the maximum depth of the decision tree and the minimum number of instances per leaf, can significantly affect the model's performance. Without a comprehensive hyperparameter tuning process, the decision trees may suffer from overfitting or underfitting, leading to inaccurate predictions of unseen data.

The project's reliance on a single dataset for training and testing could introduce data leakage issues. To accurately evaluate the generalization performance of the decision trees, it's recommended to ensure a clear separation between training and testing datasets to prevent biased performance estimates.

Moreover, while evaluating predictive accuracy with confusion matrices is informative, it's essential to consider other metrics like precision, recall, and F1-score. These metrics provide a more comprehensive understanding of a model's performance, especially when dealing with imbalanced datasets where certain classes are underrepresented.

Lastly, the project's visualization techniques for demonstrating decision tree evaluation might need to be improved in conveying the complexity of the models. Decision trees can become intricate and challenging to visualize, potentially hindering the project's objective of effectively communicating the model's performance.

To address these limitations, thorough data pre-processing, robust testing and validation of implementations, thoughtful hyperparameter tuning, and a comprehensive evaluation approach would enhance the reliability and validity of the project's results and conclusions.

Conclusion:

The Top-Down Induction of the Decision Tree (TDIDT) attribute splitting approach balances interpretability and predictive power. Its advantages, including model transparency, automatic feature selection, and the ability to manage non-linearity, align well with industries that prioritize understandable models. Our project aimed to construct decision trees using the TDIDT algorithm with three distinct attribute splitting techniques—information gain, Gini index, and Gain ratio. The project addressed the crucial attribute selection process in decision tree construction by implementing these techniques from scratch and applying them to build decision trees. The project provided insights into their predictive accuracy through thorough evaluation and comparison of the generated decision trees using a common training dataset and test data. However, it's important to acknowledge certain limitations, including potential data biases, custom implementation intricacies, hyperparameter effects, and the choice of evaluation metrics. Despite these limitations, the project offers a valuable foundation for understanding and applying decision tree algorithms and attribute selection techniques. Further refinements, such as robust testing procedures, thorough hyperparameter tuning, and broader evaluation metrics, could enhance the project's contributions. Ultimately, this project sheds light on the intricacies and significance of attribute selection methods and decision tree algorithms in predictive modelling, contributing to a deeper understanding of their applicability and effectiveness in real-world scenarios.