Package 'MuTATE'

August 21, 2026

Title Multi-Target Automated Tree Engine (MuTATE)

Version 0.1.0

Description Recursively partitions datasets on binary splits across multiple targets having different dependent variable types, including categorical, continuous, count, and survival outcomes. This overcomes single-target limitations of traditional decision trees while retaining model interpretability. See Ayton and Trevino (2023) <doi:10.1093/bioinformatics/btad507> and Ayton et al. (2025) <doi:10.1038/s44401-025-00025-4> for details.

License GPL (>= 3)

URL https://github.com/SarahAyton/MuTATE

BugReports https://github.com/SarahAyton/MuTATE/issues

Encoding UTF-8

Language en-US

Depends R (>= 3.5)

Imports reshape2, matrixStats, survival, stringr, irr, Metrics, plotrix, scales, stats, caret, dplyr, igraph, ggraph, grDevices

Suggests testthat (>= 3.0.0), knitr, rmarkdown

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

LazyData true

VignetteBuilder knitr

NeedsCompilation no

Author Sarah Ayton [aut, cre, cph] (ORCID: <https://orcid.org/0000-0001-5247-9912>)

Maintainer Sarah Ayton <sarahgabrielleayton@gmail.com>

Repository CRAN

Date/Publication 2026-08-21 15:40:09 UTC

Contents

`CVsplitEval` *Evaluate the performance of binary classifiers*

Description

A function that evaluates the performance of binary classifiers based on data split criteria

Usage

```
CVsplitEval(splitvar, X, targets, data)
```

Arguments

`splitvar`	A character string specifying the split criteria (e.g. "age < 60").
`X`	A dataframe with predictor variables.
`targets`	A data frame containing target variables.
`data`	A data frame containing the predictor and response variables.

Details

The function takes four inputs - `splitvar`, `X`, `targets` and `data` - and outputs a list with measures of the performance of binary classifiers. It first splits the data into two groups based on the specified split criteria, and then fits binary classifiers to the resulting data. The performance of the classifiers is then evaluated using various measures, such as accuracy and kappa, and is stored in the output list.

The function first splits the data into two groups based on the split criteria specified in `splitvar`. The binary classifiers are then fit to the resulting data, and their performance is evaluated using various measures such as accuracy, kappa, and root mean squared error. The results are stored in a list and returned as the output of the function.

Note that the function uses the `glm` function in R to fit the binary classifiers, and the accuracy, kappa, and root mean squared error are calculated using the `accuracy`, `kappa2`, and `rmse` functions, respectively.

Value

A list with the performance metrics for each target variable

See Also

Other relevant R functions include `glm`, `accuracy`, `kappa2`, and `rmse`.

Examples

```
data(mutate_example)
features <- c("age", "sex", "biomarker")
outcomes <- c("response", "tumor_size", "ae_count", "OS_definition_time_status")
outcome_defs <- c("Cat", "Cont", "Count", "Surv")
targets <- list(Definitions = outcome_defs, Z = mutate_example[, outcomes])

result <- CVsplitEval("age < 60", mutate_example[, features], targets, mutate_example)
```

CV_Tune *CV_Tune: Cross-Validated Hyperparameter Tuning for MTPart Models*

Description

This function performs hyperparameter tuning for MTPart models using k-fold cross-validation.

Usage

```
CV_Tune(features, outcomes, outcome_defs, data,
        continuous, quantseq, wt, reuse, kfolds, Y, seedno,
        drange, noderange, splitmin_div, method, alpharange,
        igrange, psplitrange, pdepthrange, cp_val)
```

Arguments

`features`	A character vector of feature names.
`outcomes`	A character vector of outcome names.
`outcome_defs`	A list of outcome definitions.
`data`	A data frame containing the input data.
`continuous`	A string indicating the method for handling continuous outcomes. Default is "quantile".
`quantseq`	A numeric vector specifying the quantile sequence for quantile-based methods. Default is seq(0,1,0.25).
`wt`	A numeric vector of weights for the samples. Default is NULL.
`reuse`	A logical value indicating whether to reuse previous MTPart model fits. Default is FALSE.

kfolds	An integer specifying the number of folds for k-fold cross-validation. Default is 10.
Y	A character string specifying the outcome variable for partitioning.
seedno	An integer specifying the seed for random number generation. Default is 1234.
drange	A numeric vector specifying the range of values for the 'depth' parameter. Default is seq(2, 6, by=2).
noderange	A numeric vector specifying the range of values for the 'nodesize' parameter. Default is seq(10, 30, by=10).
splitmin_div	A numeric vector specifying the range of values for the 'splitmin_div' parameter. Default is seq(2, 4, by=1).
method	A character vector specifying the method for selecting the best split. Default is c("avgIG","maxIG","mostIG","avgPVal","minPVal","mostPVal","splitError").
alpharange	A numeric vector specifying the range of values for the 'alpha' parameter. Default is seq(0.05, 0.15, by=0.05).
igrange	A numeric vector specifying the range of values for the 'IGcutoff' parameter. Default is seq(0.85, 0.95, by=0.05).
psplitrange	A numeric vector specifying the range of values for the 'psplit' parameter. Default is seq(1, 2, by=1).
pdepthrange	A numeric vector specifying the range of values for the 'pdepth' parameter. Default is seq(1, 2, by=1).
cp_val	A numeric vector specifying the range of values for the 'CP' parameter. Default is c(seq(-1, 0, by=0.5)).

Value

A data frame containing the results of hyperparameter tuning for the MTPart model.

Examples

```
data(mutate_example)
features <- c("age", "sex", "biomarker")
outcomes <- c("response", "tumor_size", "ae_count", "OS_definition_time_status")
outcome_defs <- c("Cat", "Cont", "Count", "Surv")

# A minimal single-point grid, just for illustration - a real tuning run
# would search over wider ranges for drange/noderange/method/etc.
cv_results <- CV_Tune(features, outcomes, outcome_defs, mutate_example,
                      kfolds = 2, Y = "response",
                      drange = 2, noderange = 30, splitmin_div = 2,
                      method = "avgIG", alpharange = 0.05, igrange = 0.95,
                      psplitrange = 1, pdepthrange = 1, cp_val = 0)
```

MTPart *Construct multi-target decision trees*

Description

Evaluates multiple outcome variables of different types for a set of features.

Usage

```
MTPart(
  features,
  outcomes,
  outcome_defs,
  data,
  continuous = "quantile",
  quantseq = seq(0, 1, 0.25),
  wt = NULL,
  evalmethod = "avgIG",
  alpha = 0.05,
  IGcutoff = 0.95,
  depth = 4,
  nodesize = 20,
  cp = -1,
  reuse = FALSE,
  parallelpart = FALSE,
  parallelsplit = NA,
  paralleldepth = NA,
  splitmin = floor(nodesize/2)
)
```

Arguments

features	Vector containing feature names to be evaluated.
outcomes	Data frame object of all outcomes being evaluated. Used only in Cat_Eval,Cont_Eval, Count_Eval, Surv_Eval.
outcome_defs	Vector containing a list of outcome types ("Cat", "Cont", "Count", "Surv").
data	The full dataset to be evaluated.
continuous	How to handle partition evaluation of continuous features, in which all potential split candidates may be examined ("all") or split candidates are derived from quantiles ("quantile").
quantseq	A user specified sequence ranging from 0 to 1 from which quantiles may be specified (ex: seq(0,1,0.05)). If continuous and quantseq are not specified the function will default to the "all" setting for continuous variables with fewer than 100 unique values. If there are more than 100 unique values, the function will use the "quantile" option with the default setting of seq(0,1,0.05).

`wt` A numeric vector of the outcome variable weights to be applied in multi-target evaluations. Default weights are set to `NULL`, indicating equal target variable importance.

`evalmethod` The method used to rank candidate splits across multiple targets. One of:

- `"avgIG"` (default): rank by the average standardized information gain across all targets.
- `"maxIG"`: rank by the maximum standardized information gain achieved on any single target.
- `"mostIG"`: rank by the number of targets whose standardized information gain exceeds the `IGcutoff` quantile.
- `"avgPVal"`: rank by the average p-value across targets significant at the `alpha` level.
- `"minPVal"`: rank by a weighted combination of the minimum and count of significant (`alpha`-level) target p-values.
- `"mostPVal"`: rank by the number of targets significant at the `alpha` level.
- `"splitError"`: rank by lowest average standardized information gain; used together with the parallel look-ahead search (`parallelpart = TRUE`).

`alpha` A numeric scalar specifying the significance level used to determine which targets are treated as significant when `evalmethod` is a p-value-based method (`"avgPVal"`, `"minPVal"`, or `"mostPVal"`): a target's split p-value must be less than or equal to `alpha` to count as significant. The default value for `alpha` is set to `0.05`.

`IGcutoff` A numeric scalar (between 0 and 1) specifying the quantile threshold used to decide whether a split's standardized information gain for a given target counts as a "meaningful" gain when `evalmethod = "mostIG"`. Splits with information gain at or above the `IGcutoff` quantile (computed across candidate splits for that target) count toward the total number of targets with meaningful gain. The default value for `IGcutoff` is `0.95`.

`depth` An integer scalar specifying the maximum depth of the tree (default is `6`)

`nodesize` An integer scalar specifying the minimum number of samples per node (default is `20`)

`cp` A numeric scalar specifying the complexity parameter for pruning (default is `0.02`)

`reuse` A logical scalar specifying whether to reuse split variables at higher levels of the tree (default is `FALSE`)

`parallelpart` A logical scalar specifying whether to use parallel computing for the partitioning step (default is `TRUE`)

`parallelsplit` An integer scalar specifying the number of cores to use for parallel computing (default is `2`)

`paralleldepth` An integer scalar specifying the minimum depth at which to start using parallel computing (default is `2`)

`splitmin` An integer scalar specifying the minimum sample size `n` required for a node to be partitioned. This is passed internally from `MTPart` and is used to assess feasibility of feature binarization for evaluation (i.e., a binarized feature must result in partitions with sufficient observations in both child nodes). The default value is of `splitmin` is set to half of the `nodesize`.

Value

A list with two elements:

- `partitions`: a data frame of parent/child node ID relationships describing the tree structure.
- `tree_nodes`: a list of per-node details (split variable/threshold, sample size, target summaries, error metrics).

Pass this object to `MTTest` to score new data, `MTPrune` to prune it, `MTPartSummary` or `MTSummary` to summarize it, or `PlotTree` to visualize it.

Examples

```
data(mutate_example)
features <- c("age", "sex", "biomarker")
outcomes <- c("response", "tumor_size", "ae_count", "OS_definition_time_status")
outcome_defs <- c("Cat", "Cont", "Count", "Surv")

# A 150-row subset keeps this example fast; MTPart's split search scales
# with the number of split candidates evaluated per node.
tree <- MTPart(features, outcomes, outcome_defs, mutate_example[1:150, ],
               depth = 2, nodesize = 30)
str(tree$partitions)
```

`MTPartSummary` *MTPartSummary Function*

Description

Extracts summary statistics from a decision tree.

Usage

```
MTPartSummary(tree)
```

Arguments

`tree`	A list representing a decision tree generated by a tree-based algorithm in R.

Value

A matrix containing summary statistics including the number of splits, number of leaves, complexity parameter (CP), average relative error, total relative error, cross-validated error (Xerror), standard deviation of cross-validated error (Xstd), and evaluation metric (eval).

Examples

```
data(mutate_example)
features <- c("age", "sex", "biomarker")
outcomes <- c("response", "tumor_size", "ae_count", "OS_definition_time_status")
outcome_defs <- c("Cat", "Cont", "Count", "Surv")

tree <- MTPart(features, outcomes, outcome_defs, mutate_example[1:150, ],
               depth = 2, nodesize = 30)
summ <- MTPartSummary(tree)
summ$summary_table
```

`MTPrune` *MTPrune: Prune a Decision Tree Using Minimal Cost-Complexity Pruning*

Description

Prune a MTPart decision tree object using minimal cost-complexity pruning based on the specified complexity parameter (CP) value.

Usage

```
MTPrune(tree, cp = 0.02)
```

Arguments

`tree`	A decision tree object generated by a tree-building algorithm, such as `rpart` or `ctree`.
`cp`	A numeric value specifying the complexity parameter (CP) threshold for pruning. Default is 0.02.

Value

A list with two elements:

- `partitions`: A data frame representing the pruned parent-child relationships and tree nodes.
- `tree_nodes`: A list of pruned decision tree objects, one for each node in the original tree.

Examples

```
data(mutate_example)
features <- c("age", "sex", "biomarker")
outcomes <- c("response", "tumor_size", "ae_count", "OS_definition_time_status")
outcome_defs <- c("Cat", "Cont", "Count", "Surv")

tree <- MTPart(features, outcomes, outcome_defs, mutate_example[1:150, ],
               depth = 2, nodesize = 30)
pruned <- MTPrune(tree, cp = 0.02)
```

`MTSummary` *Summarize multiple outcome variables*

Description

This function takes in a data frame of outcomes and a target table with definitions for each target, and outputs summary statistics for each target. It currently supports continuous, categorical, survival, and count outcome variable types.

Usage

```
MTSummary(targets, data)
```

Arguments

`targets`	A data frame of targets with two columns, Definitions and Z. Definitions are defined as character vectors indicating the type of variable: "Cont" for continuous, "Cat" for categorical, "Surv" for survival, and "Count" for count data. Z is a matrix of outcomes with each row representing a patient and each column representing a target.
`data`	A data frame of patient-level data with columns matching the names in targets.

Details

The `MTSummary()` function computes a summary of the outcomes in a dataset for multiple types of variables, including categorical, survival and count variables. The targets argument is a list containing a vector of outcome definitions and a matrix or data frame of outcomes. The data argument is a matrix or data frame with the data to be summarized.

For continuous targets, the function computes the mean, median, inter-quartile range, and MSE.

For categorical targets, the function computes the counts and proportions of each category, the most frequent category and its proportion, and the expected loss (i.e., proportion of observations that do not belong to the most frequent category).

For survival targets, the function computes the number of missing values, the number of events, the estimated rate of the event, and the deviance.

For count targets, the function computes the number of missing values, the total count, the estimated rate of the count, and the deviance.

Value

A list of lists with each element corresponding to an outcome in targets.

Examples

```
data(mutate_example)
outcomes <- c("response", "tumor_size", "ae_count", "OS_definition_time_status")
outcome_defs <- c("Cat", "Cont", "Count", "Surv")
targets <- list(Definitions = outcome_defs, Z = mutate_example[, outcomes])
summ <- MTSummary(targets, mutate_example)
```

MTTest *MTTest: Test Multi-Target Tree*

Description

A function to perform tree-based evaluation on test data using a trained tree model.

Usage

```
MTTest(tree, features, outcomes, outcome_defs, data_test, wt = NULL)
```

Arguments

tree	A list containing the trained tree model.
features	A character vector specifying the names of the features used in the tree model.
outcomes	A character vector specifying the names of the outcomes used in the tree model.
outcome_defs	A list containing definitions of the outcomes used in the tree model.
data_test	A data frame containing the test data.
wt	An optional numeric vector of weights for the test data.

Value

A list of evaluation results for the test data.

Examples

```
data(mutate_example)
features <- c("age", "sex", "biomarker")
outcomes <- c("response", "tumor_size", "ae_count", "OS_definition_time_status")
outcome_defs <- c("Cat", "Cont", "Count", "Surv")

train <- mutate_example[1:150, ]
tree <- MTPart(features, outcomes, outcome_defs, train,
               depth = 2, nodesize = 30)
test_result <- MTTest(tree, features, outcomes, outcome_defs, train)
```

`MultiEval` *Evaluate features across multiple outcomes and outcome types*

Description

Evaluates multiple outcome variables of different types for a set of features. `MultiEval` evaluates multiple outcomes of varying types across all features. `Cat_Eval` performs evaluation of a categorical outcome across all features. `Cont_Eval` performs evaluation of a continuous outcome across all features. `Count_Eval` performs evaluation of an event rate or count outcome across all features. `Surv_Eval` performs evaluation of a time-to-event or survival outcome across all features

Usage

```
MultiEval(X, Ztype, data, continuous = "quantile",
          quantseq = seq(0,1,0.05), wt=NULL,
          evalmethod = "avgIG", alpha = 0.05,
          IGcutoff = 0.95, splitmin)

Cat_Eval(X, m, Z, k, data, continuous,
         quantseq = seq(0,1,0.05), splitmin=10)

Cont_Eval(X, m, Z, k, data, continuous,
          quantseq = seq(0,1,0.05), splitmin=10)

Count_Eval(X, m, Z, k, data, continuous,
           quantseq = seq(0,1,0.05), splitmin=10)

Surv_Eval(X, m, Z, k, data, continuous,
          quantseq = seq(0,1,0.05), splitmin=10)
```

Arguments

`X`	Data frame object of all features to be evaluated.
`Ztype`	Vector containing a list of outcome types ("Cat", "Cont", "Count", "Surv") and a data frame object of all outcomes being evaluated.
`data`	The full dataset to be evaluated.
`continuous`	How to handle partition evaluation of continuous features, in which all potential split candidates may be examined ("all") or split candidates are derived from quantiles ("quantile").
`quantseq`	A user specified sequence ranging from 0 to 1 from which quantiles may be specified (ex: seq(0,1,0.05)). If `continuous` and `quantseq` are not specified the function will default to the "all" setting for continuous variables with fewer than 100 unique values. If there are more than 100 unique values, the function will use the "quantile" option with the default setting of seq(0,1,0.05).
`wt`	The outcome variable weights to be applied in multi-target evaluations. Default weights are set to `NULL`, indicating equal target variable importance.

`evalmethod` The method used to rank candidate splits across multiple targets. One of:

- `"avgIG"` (default): rank by the average standardized information gain across all targets.
- `"maxIG"`: rank by the maximum standardized information gain achieved on any single target.
- `"mostIG"`: rank by the number of targets whose standardized information gain exceeds the `IGcutoff` quantile.
- `"avgPVal"`: rank by the average p-value across targets significant at the `alpha` level.
- `"minPVal"`: rank by a weighted combination of the minimum and count of significant (`alpha`-level) target p-values.
- `"mostPVal"`: rank by the number of targets significant at the `alpha` level.
- `"splitError"`: rank by lowest average standardized information gain; used together with the parallel look-ahead search.

`alpha` The significance level used to determine which targets are treated as significant when `evalmethod` is a p-value-based method (`"avgPVal"`, `"minPVal"`, or `"mostPVal"`). The default value for `alpha` is set to 0.05.

`IGcutoff` The quantile threshold (between 0 and 1) used to decide whether a split's standardized information gain for a given target counts as a "meaningful" gain when `evalmethod = "mostIG"`. The default value for `IGcutoff` is 0.95.

`splitmin` The minimum sample size `n` required for a node to be partitioned. This is passed internally from `MTPart` and is used to assess feasibility of feature binarization for evaluation (i.e., a binarized feature must result in partitions with sufficient observations in both child nodes). The default value is of `splitmin` is set to 10.

`m` Specific feature to evaluate. Used only in `Cat_Eval`, `Cont_Eval`,`Count_Eval`, `Surv_Eval`.

`Z` Data frame object of all outcomes being evaluated. Used only in `Cat_Eval`,`Cont_Eval`, `Count_Eval`, `Surv_Eval`.

`k` Specific outcome to evaluate. Used only in `Cat_Eval`, `Cont_Eval`, `Count_Eval`,`Surv_Eval`.

Details

The following function requires inputs including all features to be evaluated (`X`), which specific feature to evaluate (`m`), all outcomes being evaluated (`Z`), the specific outcome to evaluate (`k`), and the data (`data`). In addition, there are options for the user to specify how to handle partition evaluation of continuous features, in which all potential split candidates may be examined ("all") or split candidates are derived from quantiles ("quantile"); the option `quantseq` takes a user specified sequence ranging from 0 to 1 from which quantiles may be specified (ex: seq(0,1,0.05)). If `continuous` and `quantseq` are not manually specified the function will default to the "all" setting for continuous variables with fewer than 100 unique values. If there are more than 100 unique values, the function will use the "quantile" option with the default setting of seq(0,1,0.05).

The function outputs a matrix including information about the outcome (Zk), outcome variable type (Ztype), the feature (Xm), feature type (Xtype), the split threshold being evaluated (Split), evaluation method (Eval), and the entropy of a partition using that split threshold (SplitEntropy), information gain relative to the root node (Inf_Gain) and the proportion of information gain (Inf_GainProp),

the significance of association between the split threshold and outcome (P_Value), relative ranking of split thresholds based on how much their respective partitions reduced entropy (Rank), and the standard normalized proportion of information gain associated with each threshold.

All evaluation functions operate internally within the multi-target evaluation function in which all outcomes and all features are assessed at a node. Therefore, m and k options are specified by a loop from the user supplied inputs: X, Z, and data. Again, continuous and quantseq are optional arguments initialized by the user.

Value

The `MultiEval` function returns a vector including of the evaluation metrics for each possible split of each feature in the dataset.

Examples

```
data(mutate_example)
features <- c("age", "sex", "biomarker")
outcomes <- c("response", "tumor_size", "ae_count", "OS_definition_time_status")
outcome_defs <- c("Cat", "Cont", "Count", "Surv")
Ztype <- list(Definitions = outcome_defs, Z = mutate_example[, outcomes])

eval_result <- MultiEval(X = mutate_example[, features], Ztype = Ztype,
                         data = mutate_example)

# Single-feature/single-outcome evaluators used internally by MultiEval:
Cat_Eval(X = mutate_example[, features], m = 1,
         Z = mutate_example[, outcomes], k = 1,
         data = mutate_example, continuous = "quantile")
```

---

mutate_example *Synthetic example dataset for MuTATE*

Description

A small simulated dataset used in function examples and tests. It is purely illustrative (not derived from real patient data) and is designed to exercise all four outcome types supported by `MTPart`: categorical, continuous, count, and survival.

Usage

```
mutate_example
```

Format

A data frame with 200 rows and 9 variables:

age Continuous feature: simulated patient age.

sex Categorical feature: `"F"` or `"M"`.

biomarker Continuous feature: simulated biomarker level.

response Categorical outcome: `"responder"` or `"non_responder"`.

tumor_size Continuous outcome: simulated tumor size.

ae_count Count outcome: simulated adverse event count.

time Survival outcome: time-to-event.

status Survival outcome: event indicator (1 = event, 0 = censored).

OS_definition_time_status Placeholder column required by `MTPart`'s survival-outcome naming convention: the outcome name passed via `outcomes` must itself be a column in `data`, and its 3rd and 4th underscore-separated tokens (`time`, `status`) must match the actual time/event column names.

Examples

```
data(mutate_example)
head(mutate_example)
```

`PlotTree` *PlotTree: Plot Decision Tree*

Description

Plot a multi-target decision tree.

Usage

```
PlotTree(tree)
```

Arguments

`tree`	A decision tree object.

Value

A plot of the multi-target decision tree (invisible; called for its plotting side effect).

Examples

```
data(mutate_example)
features <- c("age", "sex", "biomarker")
outcomes <- c("response", "tumor_size", "ae_count", "OS_definition_time_status")
outcome_defs <- c("Cat", "Cont", "Count", "Surv")

tree <- MTPart(features, outcomes, outcome_defs, mutate_example[1:150, ],
               depth = 2, nodesize = 30)
PlotTree(tree)
```

SplitPrep *Prepare data for splitting in a decision tree*

Description

SplitPrep() is a function that prepares data frames for use in tree-based models by identifying and removing candidate predictor variables that are unsuitable for splitting at a given node. This function takes in three data frames, Xdf, df, and data_splt and a character string parentsplit. It returns a list of two data frames: splitX_df, which contains the predictor variables that can be used for splitting at a given node, and split_data_df, which contains the response variables and any other variables that should be included in the data frame for further analysis.

Usage

```
SplitPrep(Xdf, df, data_splt, parentsplit)
```

Arguments

Xdf	A data frame containing the predictor variables.
df	A data frame containing the response variable.
data_splt	The data used for the split.
parentsplit	The parent split.

Details

SplitPrep() takes in two data frames, one containing the predictor variables and another containing the response variable. It identifies which predictor variables to use for splitting in the decision tree and returns a list containing two data frames: the modified predictor variable data frame and the modified response variable data frame.

The function first drops any unused factor levels in both input data frames (Xdf and df) using the droplevels() function. It then loops through the predictor variables in Xdf to determine which ones should be kept for splitting at the current node.

For each predictor variable, the function first checks if it was used for splitting at the parent node (identified by the parentsplit argument). If so, it is added to a list of variables to drop (droplist).

If the predictor variable is categorical (i.e., a factor variable), the function checks if it has less than 2 levels or a prevalence of less than 5% in the current node. If either of these conditions is true, the variable is added to droplist.

If the predictor variable is continuous, the function checks if it has only one observed value. If so, it is added to droplist.

Finally, if droplist exists, the function removes the corresponding columns from splitX_df and renames the remaining columns to match the original column names. The function then returns a list containing splitX_df and split_data_df.

This function is used to pre-process data before fitting tree-based models to ensure that only suitable predictor variables are used for splitting.

Value

A list containing two data frames: the modified predictor variable data frame and the modified response variable data frame.

Examples

```
data(mutate_example)
features <- c("age", "sex", "biomarker")
prep <- SplitPrep(Xdf = mutate_example[, features], df = mutate_example,
                  data_splt = mutate_example, parentsplit = "skip")
```

Index