Introduction to Multi-Target Automated Tree Engine (MuTATE)

The goal of MuTATE is to create automated, explainable, and comprehensive models across multiple dependent variables of interest.

It provides a collection of functions to recursively partition data on binary splits across multiple targets having different dependent variable types. This overcomes single-target limitations of traditional decision trees, without loosing model interpretability, and while handling continuous, categorical, count, and survival outcome variables. This suite of functions also includes a number of parameters for model customization, dependent variable weights, parameter tuning functions, and visualization tools.

Installation

You can install the development version of MuTATE from GitHub with:

```
# install.packages("devtools")
devtools::install_github("SarahAyton/MuTATE")
#>
#> -- R CMD build -----------------------------------------------------------------
#>      checking for file '/private/var/folders/b3/0jl6y5ld4_1_xvc9ywmwbmxc0000gn/T/RtmplglPtQ/remotes1
#>   -  preparing 'MuTATE':
#>      checking DESCRIPTION meta-information ...  v  checking DESCRIPTION meta-information
#>   -  checking for LF line-endings in source and make files and shell scripts
#>   -  checking for empty or unneeded directories
#>   -  building 'MuTATE_0.0.0.9000.tar.gz'
#>
#>
```

Once MuTATE is installed, it can be easily loaded:

```
library(MuTATE)
```

Set up the data

First, we can generate a data set that includes demographic and molecular predictors, as well as multiple dependent variables. Here, we explore age, gender, ethnicity, state, and biomarker predictors, and response to treatment, disease progression, and survival outcomes. We also include missing values to demonstrate how data with missing values can be prepared for modeling.

```
# Load required libraries
library(dplyr)
library(tidyr)

# Set seed for reproducibility
set.seed(123)

# Number of observations
n <- 100

```

```r
# Generate predictor variables
age <- rnorm(n, mean = 50, sd = 10)
gender <- sample(c("M", "F"), n, replace = TRUE)
ethnicity <- sample(c("Asian", "Caucasian", "African American", "Hispanic"), n, replace = TRUE)
state <- sample(c("CA", "NY", "TX", "FL", "IL"), n, replace = TRUE)
biomarker1 <- sample(c("Wild Type", "Mutant"), n, replace = TRUE)
biomarker2 <- sample(c("Wild Type", "Mutant"), n, replace = TRUE)
biomarker3 <- sample(c("Wild Type", "Mutant"), n, replace = TRUE)
biomarker4 <- sample(c("Wild Type", "Mutant"), n, replace = TRUE)
biomarker5 <- sample(c("Wild Type", "Mutant"), n, replace = TRUE)

# Generate dependent variables
response <- sample(c("Yes", "No"), n, replace = TRUE)
progression <- sample(c(0, 1), n, replace = TRUE)
timetoprogression <- rnorm(n, mean = 100, sd = 20)
vitalstatus <- sample(c(0, 1), n, replace = TRUE)
overallsurvivaltime <- rnorm(n, mean = 500, sd = 100)

# Create a data frame
df <- data.frame(age, gender, ethnicity, state, biomarker1, biomarker2, biomarker3,
                 biomarker4, biomarker5, response, progression, timetoprogression,
                 vitalstatus, overallsurvivaltime)

# Randomly set 10% of values to NA
na_pct <- 0.1
for (col in colnames(df)) {
  n_na <- round(n * na_pct)
  rows_to_replace <- sample(1:n, n_na, replace = FALSE)
  df[rows_to_replace, col] <- NA
}

# Print first few rows of the data set
head(df)
#>        age gender ethnicity state biomarker1 biomarker2 biomarker3 biomarker4
#> 1 44.39524      F Caucasian    TX     Mutant     Mutant     Mutant     Mutant
#> 2 47.69823      F Caucasian    CA     Mutant  Wild Type  Wild Type     Mutant
#> 3 65.58708      F     Asian    IL  Wild Type     Mutant     Mutant  Wild Type
#> 4 50.70508   <NA> Caucasian    CA     Mutant       <NA>     Mutant  Wild Type
#> 5 51.29288      F      <NA>    FL  Wild Type  Wild Type     Mutant     Mutant
#> 6 67.15065      F     Asian    NY     Mutant  Wild Type       <NA>     Mutant
#>   biomarker5 response progression timetoprogression vitalstatus
#> 1     Mutant     <NA>           1          65.46984           0
#> 2  Wild Type      Yes           1         112.08050           0
#> 3  Wild Type      Yes           1         100.53754           0
#> 4  Wild Type     <NA>          NA          39.06067           1
#> 5     Mutant       No           1         122.15055           1
#> 6  Wild Type       No           0          86.16650           1
#>   overallsurvivaltime
#> 1            481.2817
#> 2            697.8184
#> 3            364.9854
#> 4            473.5479
#> 5                  NA
#> 6            443.0814
```

Variable formatting and missing values

Now that we have our data set, we can begin to think about which variables may serve as predictors (features) or dependent variables(outcomes) in our model. This will vary depending on the research question and depends on causality and biological plausibility. Since we pre-defined our predictors and outcomes when generating the data, we can define these as follows.

```
# Variable formatting
# Convert character variables to factors
char_vars <- sapply(df, is.character)
df[, char_vars] <- lapply(df[, char_vars], factor)

# Convert integer variables to numeric
int_vars <- sapply(df, is.integer)
df[, int_vars] <- lapply(df[, int_vars], as.numeric)

# Convert 0 or 1 variables to binary
bin_vars <- sapply(df, function(x) all(x %in% c(0, 1)))
df[, bin_vars] <- lapply(df[, bin_vars], as.logical)

# Prepare survival outcomes
library(survival)

df <- df[complete.cases(df[, c("timetoprogression", "progression",
                               "vitalstatus", "overallsurvivaltime")]), ]
temp  <- coxph(Surv(timetoprogression, progression) ~ 1, data = df)
df$exp_tt_timetoprogression_progression <- predict(temp, type = 'expected')
temp  <- coxph(Surv(overallsurvivaltime, vitalstatus) ~ 1, data = df)
df$exp_tt_overallsurvivaltime_vitalstatus <- predict(temp, type = 'expected')

# Feature and outcome designation
features <- c(# Demographics
              "age","gender","ethnicity","state",
              # Molecular Biomarkers
              "biomarker1", "biomarker2", "biomarker3", "biomarker4", "biomarker5"
)

outcomes <- c('response', 'exp_tt_timetoprogression_progression',
              'exp_tt_overallsurvivaltime_vitalstatus')

surv     <- c("timetoprogression", "progression", "vitalstatus", "overallsurvivaltime")

# Keep variables of interest
df <- df[,c(features, outcomes, surv)]

# Print first few rows of the data set
head(df)
#>         age gender        ethnicity state biomarker1 biomarker2 biomarker3
#> 1  44.39524      F        Caucasian    TX     Mutant     Mutant     Mutant
#> 2  47.69823      F        Caucasian    CA     Mutant  Wild Type  Wild Type
#> 3  65.58708      F            Asian    IL  Wild Type     Mutant     Mutant
#> 6  67.15065      F            Asian    NY     Mutant  Wild Type       <NA>
#> 7  54.60916      F        Caucasian    NY     Mutant  Wild Type  Wild Type
#> 10 45.54338      F African American    FL       <NA>     Mutant     Mutant
#>    biomarker4 biomarker5 response exp_tt_timetoprogression_progression
#> 1      Mutant     Mutant     <NA>                           0.01562500
#> 2      Mutant  Wild Type      Yes                           0.65966104
#> 3   Wild Type  Wild Type      Yes                           0.34668349
#> 6      Mutant  Wild Type       No                           0.06742264
#> 7   Wild Type     Mutant      Yes                           0.47293665
#> 10  Wild Type  Wild Type       No                           0.01562500
#>    exp_tt_overallsurvivaltime_vitalstatus timetoprogression progression
#> 1                              0.30472984          65.46984           1
#> 2                              1.76683554         112.08050           1
#> 3                              0.06383662         100.53754           1
#> 6                              0.14286778          86.16650           0
#> 7                              2.26683554         103.91419           0
#> 10                             0.08201844          69.80611           0
#>    vitalstatus overallsurvivaltime
#> 1            0            481.2817
#> 2            0            697.8184
#> 3            0            364.9854
#> 6            1            443.0814
#> 7            0            733.2719
#> 10           0            403.3801
```

Now that all variables have been formatted and we have decided which variables are predictors and which are dependent, we can prepare the overall dataset for analysis. Here, we will filter out any variables missing observations in > 50% of patients (varprop = 0.5), and patients missing data in > 50% of the remaining variables (nprop = 0.5). Of course, there are other methods, such as multiple imputation, that can be used to address missing data which may be considered as well.

```r
# Function to clean data
clean_data <- function(data, varprop = 0.5, nlevel = 10, nprop = 0.5) {
  # Remove variables with more than varprop (50%) missing data
  propmiss <- apply(data, 2, function(x) sum(is.na(x))/length(x))
  droplist <- sapply(propmiss, function(x) if (x >= varprop) TRUE else NA)
  droplist <- which(droplist)
  data1 <- if (all(!is.na(droplist)) && length(droplist)>0) data[, -droplist] else data
  rm(droplist)
  # Remove variables with only one level
  levcount <- sapply(data1, function(x) length(unique(x)))
  droplist <- sapply(levcount, function(x) if (x <= 1) TRUE else NA)
  droplist <- which(droplist)
  data2 <- if (all(!is.na(droplist)) && length(droplist)>0) data1[, -droplist] else data1
  rm(droplist)
  # Categorize variables with fewer than nlevel (10) levels as factors
  factor_cols <- which(which(sapply(data2, class) != "factor") %in% which(levcount <= nlevel))
  data3 <- data2
  data3[, factor_cols] <- lapply(data3[, factor_cols], factor)
  # Remove rows with more than nprop (50%) missing data
  propmiss <- apply(data3, 1, function(x) sum(is.na(x))/length(x))
  droplist <- sapply(propmiss, function(x) if (x > nprop) TRUE else NA)
  droplist <- which(droplist)
  data4 <- if (all(!is.na(droplist)) && length(droplist)>0) data3[-droplist, ] else data3
  return(data4)
}

# Clean the data
df_clean <- clean_data(df, varprop = 0.5, nlevel = 10, nprop = 0.5)

# Remove any variables that were dropped in data cleaning
features <- features[(features %in% colnames(df_clean))]
outcomes <- outcomes[(outcomes %in% colnames(df_clean))]
surv     <- surv[(surv %in% colnames(df_clean))]

# Set outcome variable types for final outcome set
outcome_defs <- c("Cat","Surv","Surv")
```

Note that no patients or variables were dropped in this case.

Partition testing & training sets

Next, we partition our data into model development and validation sets using the {caret} package and a 60% / 40% data split, balancing vital status in both sets.

```
library(caret)

# Set seed for reproducibility
set.seed(123)

# Select one of your outcomes to be balanced in both sets
trainobs <- createDataPartition(df_clean$vitalstatus, p = .6, list = FALSE, times = 1)
x_train <- df_clean[ trainobs,]
x_test  <- df_clean[-trainobs,]

# Development dataset
data_train <- x_train[complete.cases(x_train[,c(outcomes)]), c(outcomes,surv,features)]
rownames(data_train) <- NULL

# Validation dataset
data_test <- x_test[complete.cases(x_test[,c(outcomes)]), c(outcomes,surv,features)]
rownames(data_test) <- NULL

df_set <- rbind(data_train, data_test)
rownames(df_set) <- NULL
```

Parameter tuning

We can jump right into modeling our data set with the outcomes and features we have specified. However, there are a number of parameters that can be determined by the user to improve model performance. To help guide parameter value selection, we use the CV_Tune function to perform k-fold cross-validated tuning with grid search on parameters of interest.

```
df_tune <- CV_Tune(features, outcomes, outcome_defs, data_train,
                   kfolds=10, Y="vitalstatus", seedno = 1234,
                   drange    = c(seq(2, 4, by=1)),
                   noderange = c(seq(5, 15, by=5)),
                   splitmin_div  = c(seq(1, 3, by=1)),
                   method = c("avgIG","maxIG","mostIG","avgPVal",
                              "minPVal","mostPVal","splitError"),
                   alpharange  = c(seq(0.05, 0.05, by=0.05)),
                   igrange   = c(seq(0.95, 0.95, by=0.05)),
                   psplitrange = c(seq(1, 1, by=1)),
                   pdepthrange = c(seq(1, 1, by=1)),
                   cp_val = c(seq(-1, -0.5, by=0.25)) )
```

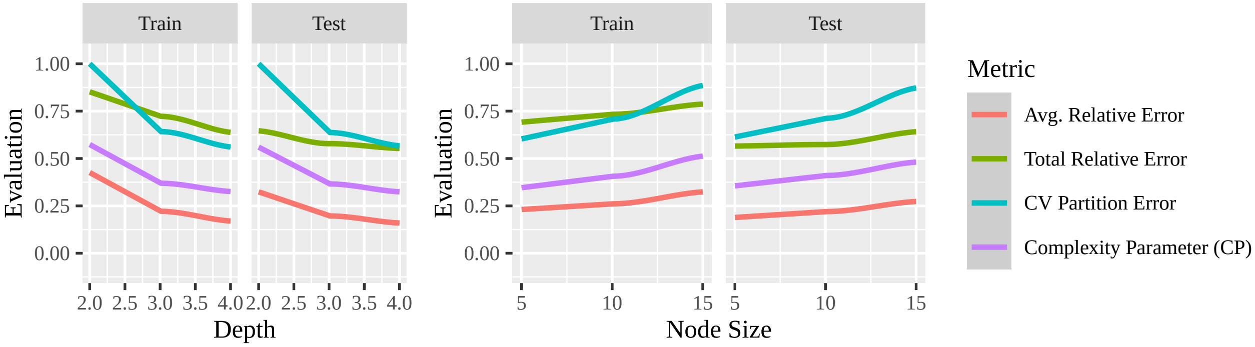
If you explore these outputs, df_tune contains two components: df_tune[[1]] provides the performance estimates averaged across all folds for each set of parameter values, and df_tune[[2]] provides the performance for each fold in each set of parameter values. Here, we will explore df_tune[[2]] which provides more information about the range of evaluations observed.

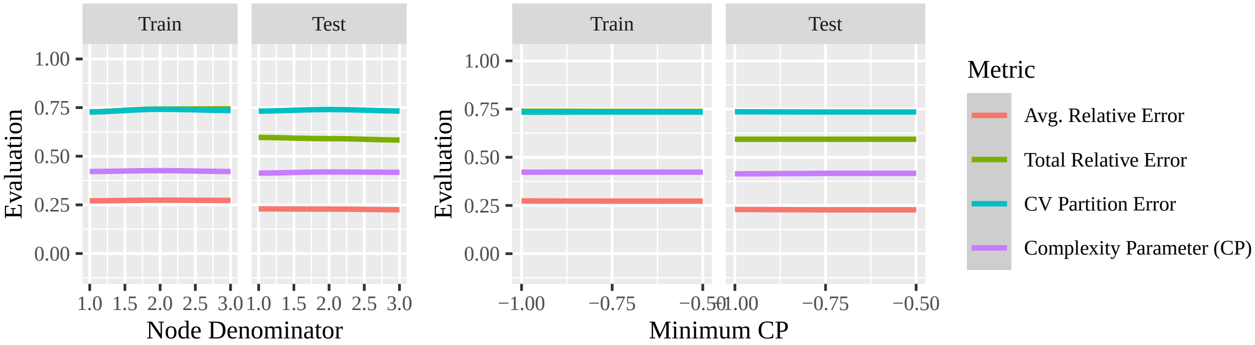
Let's take a look at the tuning results.

```
# Convert the data to long format so that each fold is on its own row
df_tune_all <- gather(df_tune[[2]], fold, result, X1:X10, factor_key=TRUE)

# Indicate which evaluations refer to the train or test subset of each fold
df_tune_all$Set <- as.factor(ifelse(grepl("test", df_tune_all$EvalName), "Test", "Train"))
df_tune_all$result <- as.numeric(unlist(df_tune_all$result))

# Drop "_train" and "_test" from evaluation names
df_tune_all$EvalName <- sub("_train|_test", "", df_tune_all$EvalName)

# Create an object "plot" with only the evaluation metrics of interest
plot  <- droplevels(df_tune_all[!(df_tune_all$EvalName %in%
                                    c("N", "RunTime", "nsplit", "leaves", "Xerror", "Xstd")),
                                c(1:ncol(df_tune_all))])

# Set evaluation name labels
plot$EvalName <- factor(plot$EvalName,
                        levels = c("AvgRelError", "TotRelError", "Eval", "CP"),
                        labels = c("Avg. Relative Error", "Total Relative Error",
                                   "CV Partition Error", "Complexity Parameter (CP)"))

# Set fold subset labels
plot$Set <- factor(plot$Set, levels = c("Train", "Test"))

# Take a look at the evaluation data set
head(plot)
#>    depth nodesize splitmin_div method alpha IGcutoff psplit pdepth CP
#> 3      2        5            1  avgIG  0.05     0.95      1      1 -1
#> 4      2        5            1  avgIG  0.05     0.95      1      1 -1
#> 5      2        5            1  avgIG  0.05     0.95      1      1 -1
#> 8      2        5            1  avgIG  0.05     0.95      1      1 -1
#> 13     2        5            1  avgIG  0.05     0.95      1      1 -1
#> 14     2        5            1  avgIG  0.05     0.95      1      1 -1
#>                       EvalName fold      result   Set
#> 3  Complexity Parameter (CP)   X1 0.600449327 Train
#> 4        Avg. Relative Error   X1 0.399550673 Train
#> 5       Total Relative Error   X1 0.799101346 Train
#> 8         CV Partition Error   X1 1.000000000 Train
#> 13 Complexity Parameter (CP)   X1 0.993901113  Test
#> 14       Avg. Relative Error   X1 0.003043633  Test
```

Training			Paritioning Method							
Metric	Evaluation	Total	Avg. IG	Max. IG	Most IG	Avg. P-val	Min. P-val	Most P-val	Split Error (Lookahead)	
Avg. Leaf Error (Relative to Root)	Mean (SD)	0.3 (0.1)	0.3 (0.1)	0.3 (0.1)	0.3 (0.1)	0.3 (0.1)	0.3 (0.1)	0.3 (0.1)	0.3 (0.1)	<0
Total Leaf Error (Relative to Root)	Mean (SD)	0.7 (0.1)	0.7 (0.1)	0.7 (0.1)	0.7 (0.1)	0.8 (0.1)	0.8 (0.1)	0.8 (0.1)	0.7 (0.1)	<0
Cross-Validated Partition Error	Mean (SD)	0.7 (0.3)	0.7 (0.3)	0.7 (0.3)	0.7 (0.3)	0.8 (0.2)	0.8 (0.2)	0.8 (0.2)	0.7 (0.3)	<0
Complexity Parameter (CP)	Mean (SD)	0.4 (0.2)	0.4 (0.2)	0.4 (0.2)	0.4 (0.2)	0.4 (0.1)	0.4 (0.1)	0.4 (0.1)	0.4 (0.2)	<0

We can now generate diagnostic plots of parameter tuning results. First, lets examine the model size parameters.





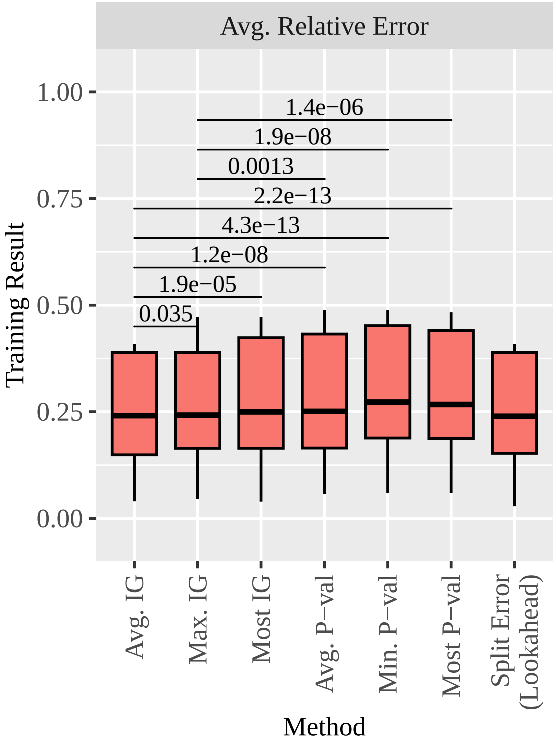
We can see a number of evaluation metrics presented: average relative error in the leaves ("Avg. Relative Error"), the total relative error across all leaves ("Total Relative Error"), cross-validated partition error calculated at the time of partitioning ("CV Partition Error" provides the cross-validated relative error from all internal node partitions), and the complexity parameter ("Complexity Parameter (CP)") which measures model improvement as partitions are added while applying a complexity penalty. Note the total and average relative error align in the training results for the node denominator and minimum CP tuning.

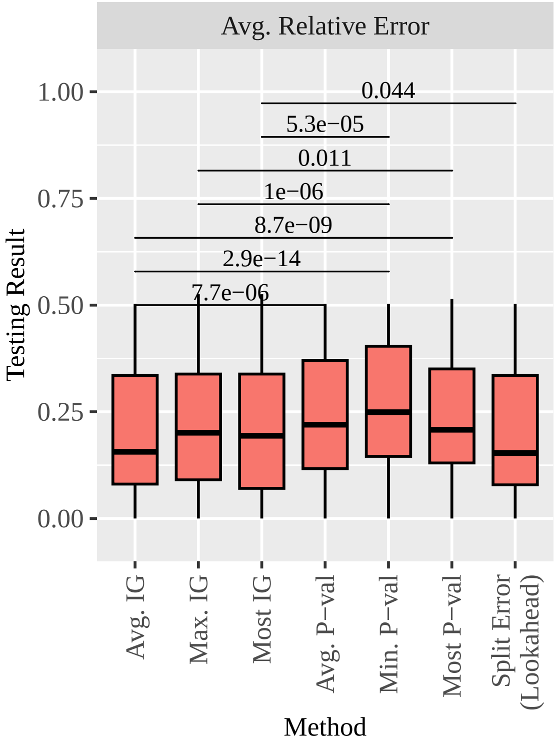
Based on these tuning results, we may consider a model depth = 4, minimum node side = 10, node denominator = 2 (minimum leaf size = 10/2), and CP = -0.50.

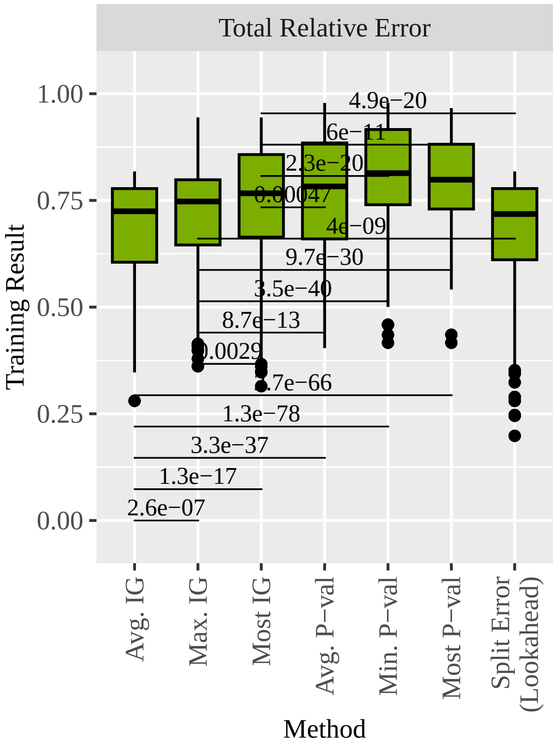
We can also look at the results from tuning performed on the partitioning method.

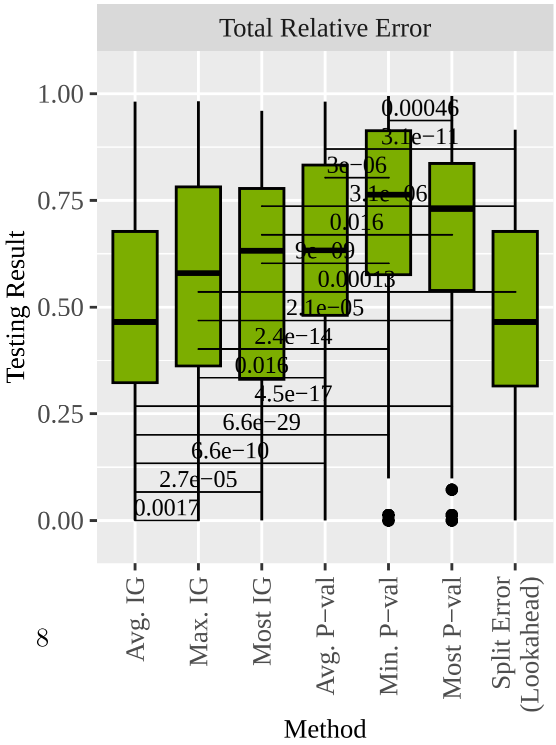
And take a closer look at pairwise comparisons between partitioning methods.

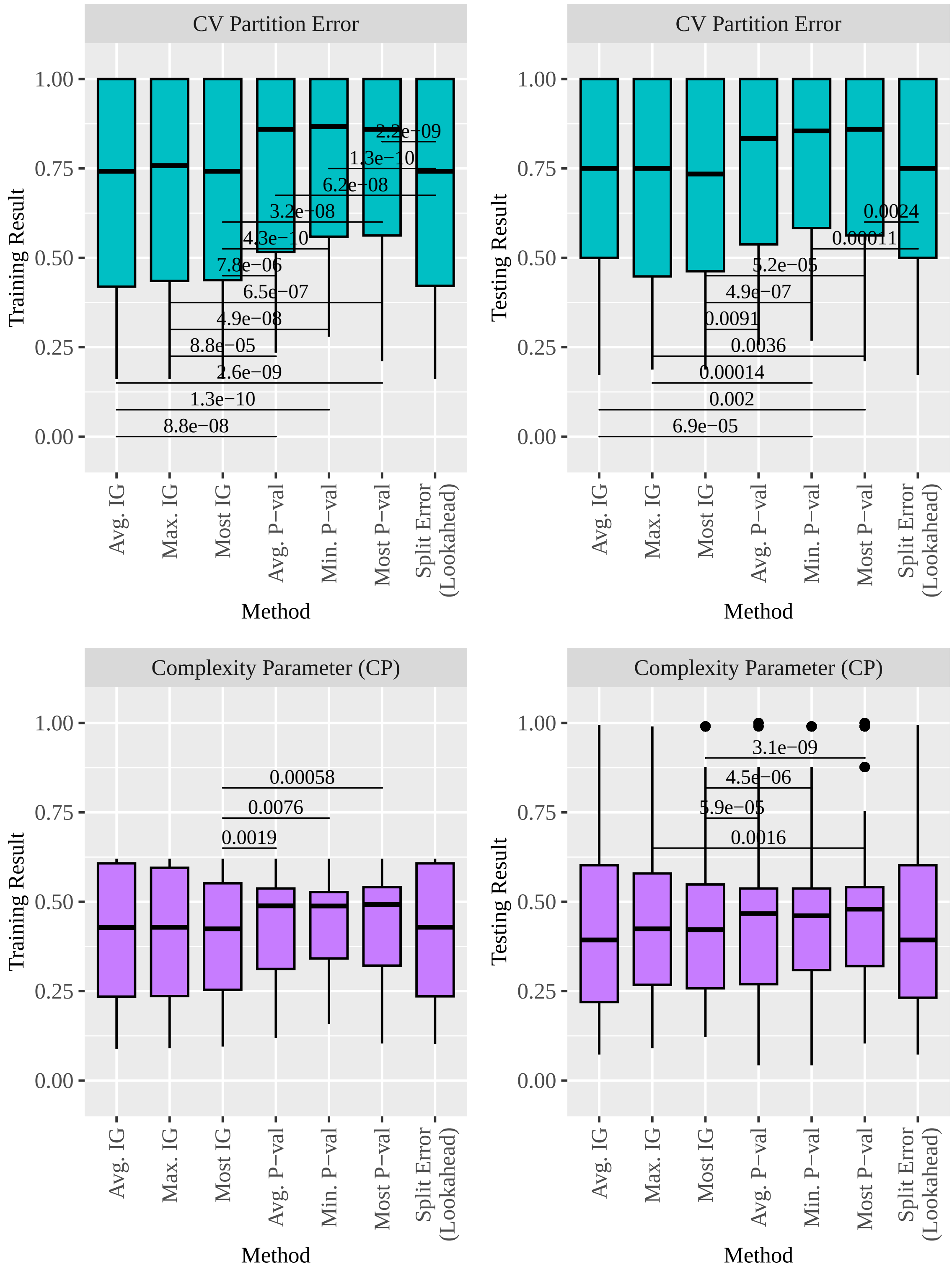
Testing			Paritioning Method							
Metric	Evaluation	Total	Avg. IG	Max. IG	Most IG	Avg. P-val	Min. P-val	Most P-val	Split Error (Lookahead)	
Avg. Leaf Error (Relative to Root)	Mean (SD)	0.2 (0.2)	0.2 (0.1)	0.2 (0.2)	0.2 (0.2)	0.2 (0.1)	0.3 (0.1)	0.2 (0.1)	0.2 (0.1)	<0
Total Leaf Error (Relative to Root)	Mean (SD)	0.6 (0.3)	0.6 (0.3)	0.6 (0.3)	0.6 (0.3)	0.7 (0.3)	0.7 (0.3)	0.7 (0.3)	0.6 (0.3)	<0
Cross-Validated Partition Error	Mean (SD)	0.7 (0.3)	0.7 (0.3)	0.7 (0.3)	0.7 (0.3)	0.8 (0.2)	0.8 (0.2)	0.8 (0.2)	0.7 (0.3)	<0
Complexity Parameter (CP)	Mean (SD)	0.4 (0.2)	0.4 (0.3)	0.4 (0.2)	0.4 (0.2)	0.4 (0.2)	0.4 (0.2)	0.4 (0.2)	0.4 (0.3)	<0











From the parameter tuning, we see that both Avg. IG and SplitError partitioning perform the best across metrics, although Avg. IG performs slightly better in the average and total relative error results. Based on this we will select Avg. IG as our partitioning method in the trained model.

Building the Model

Training the model

Now that we have performed parameter tuning, we can apply the selected parameter values when we construct our decision tree model.

```
df_model_train <- MTPart(features, outcomes, outcome_defs, data_train,
                         evalmethod = "avgIG", alpha = 0.05, IGcutoff = 0.95,
                         depth=4, nodesize=10, cp=-0.5, splitmin=floor(nodesize/2))

attributes(df_model_train)
#> $names
#> [1] "partitions" "tree_nodes"

# We can see the partitions object
head(df_model_train$partitions)
#>    parent child
#> 1    <NA>     1
#> NA      1     2
#> 3       1     3
#> 4       2   4 *
#> 5       2   5 *
#> 6       3     6

# Let's take a look at the tree node object
attributes(df_model_train$tree_nodes[[1]])
#> $names
#>  [1] "NodeID"   "SplitVar" "Var"      "Thresh"   "N"        "Pnode"
#>  [7] "Targets"  "relerr"   "PartVar"  "CP"       "CVeval"   "SXerror"
#> [13] "SXstd"    "Eval"
```

The “partitions” object is a table of partitions presenting the parent and child node IDs. Note that * is used to indicate a terminal or leaf node.

The “tree_nodes” object includes information on the node name, the parent partition, variable name, and threshold that gave rise to that node (for the root node these values are “Root”), the sample size of the node, proportion of the sample represented, node relative error, the partition variable selected for any future partitions, among other evaluation metrics. We can also see there is an “Eval” object.

```
head(df_model_train$tree_nodes[[1]]$Eval)
#>    Xm                  Split exp_tt_overallsurvivaltime_vitalstatus
#> 1 age age < 43.7496073215074                             -0.5736613
#> 2 age age < 45.0896883394346                             -0.5640063
#> 3 age age < 46.1952899898762                             -0.5220496
#> 4 age age < 47.0492851700773                             -0.4450880
#> 5 age  age < 47.920827219804                             -0.3941051
#> 6 age age < 48.6110863756096                             -0.2814336
#>   exp_tt_timetoprogression_progression  response       AvgIG     MaxIG MostIG
#> 1                           -0.7031601 0.5497598 -0.24235388 0.5497598      0
#> 2                           -0.6887785        NA -0.62639240        NA     NA
#> 3                           -0.6597246        NA -0.59088708        NA     NA
#> 4                           -0.6340058        NA -0.53954690        NA     NA
#> 5                           -0.6842230        NA -0.53916405        NA     NA
#> 6                           -0.7007151 0.8036323 -0.05950543 0.8036323      0
#>     AvgPVal   MinPVal MostPVal AvgMostPVal InvMinPVal WtMinPval Rank
#> 1 0.5957470 0.2912421        0   0.5497598  0.4502402        0   13
#> 2 0.7344712        NA       NA         NaN        NaN       NA   28
#> 3 0.7227020        NA       NA         NaN        NaN       NA   24
#> 4 0.7052452        NA       NA         NaN        NaN       NA   22
#> 5 0.7051132        NA       NA         NaN        NaN       NA   21
#> 6 0.5237252 0.2108047        0   0.8036323  0.1963677        0   10
```

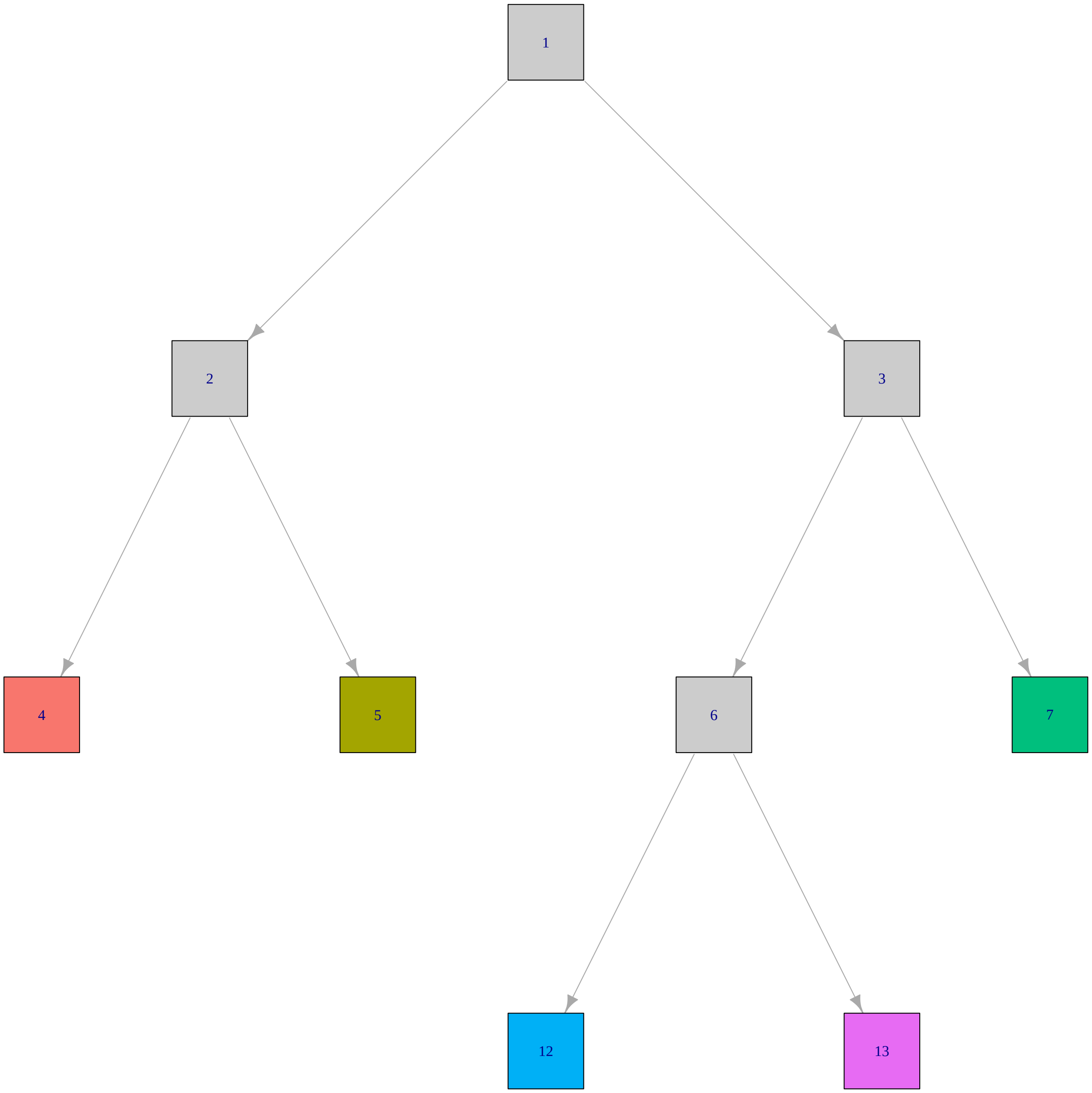
This includes the evaluation performed on all potential partitions across outcomes, including the performance on each outcome, and the overall metric (defined by the partitioning method).

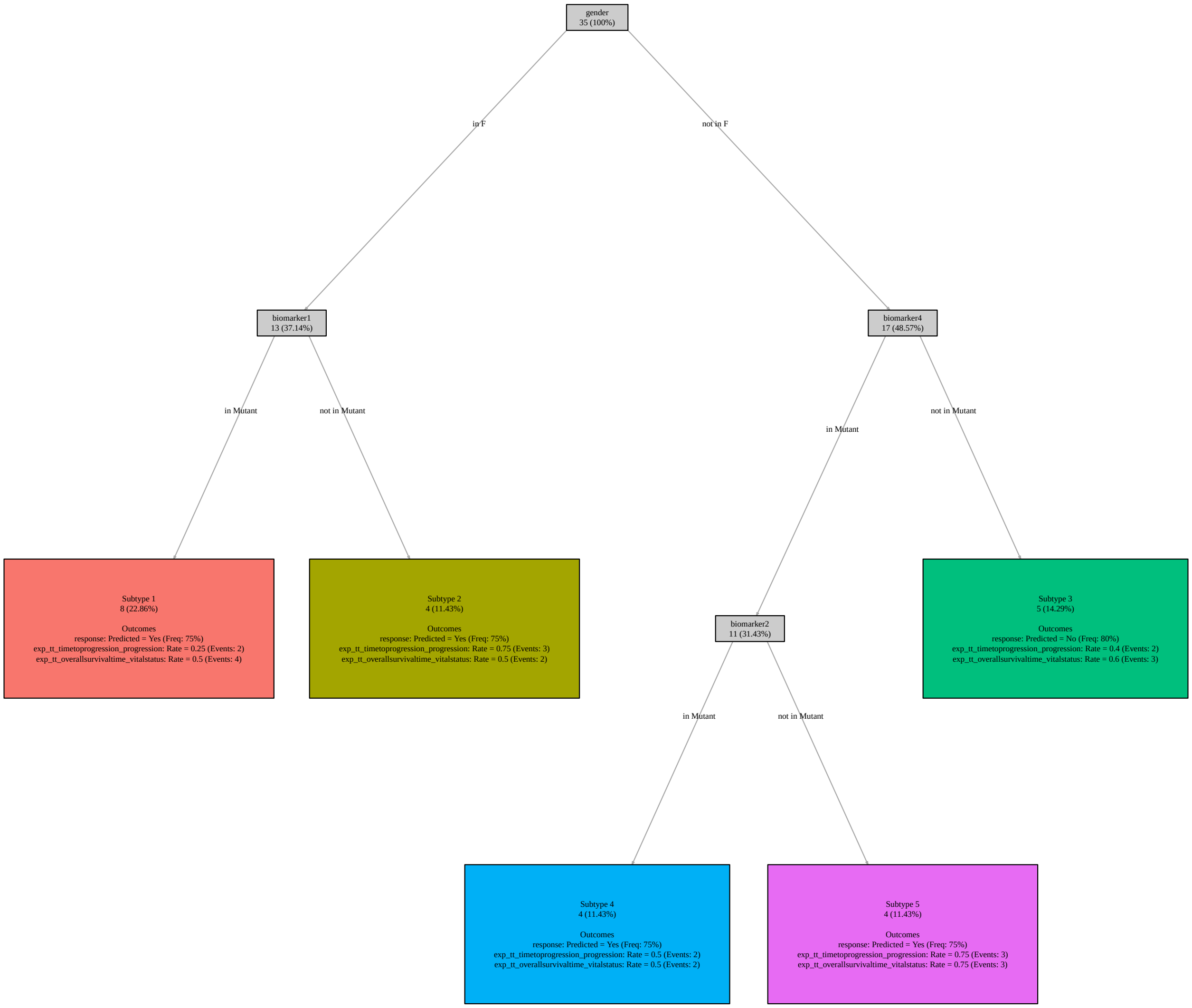
We can also obtain a summary of the model using the following.

```
df_train_summary <- MTPartSummary(df_model_train)
df_train_summary$node_data
#>   parent node                Partition  N     Pnode        CP    relerr SXerror
#> 1      0    1                     Root 35 1.0000000 0.5943635 1.0000000       1
#> 2      1    2              gender in F 13 0.3714286 0.5815473 1.0297887       1
#> 3      1    3          gender not in F 17 0.4857143 0.5036874 0.8827824       1
#> 4      2    4     biomarker1 in Mutant  8 0.2285714        NA 1.0540504      NA
#> 5      2    5 biomarker1 not in Mutant  4 0.1142857        NA 0.8054684      NA
#> 6      3    6     biomarker4 in Mutant 11 0.3142857 0.4786249 0.7499811       1
#> 7      3    7 biomarker4 not in Mutant  5 0.1428571        NA 0.9278875      NA
#> 8      6   12     biomarker2 in Mutant  4 0.1142857        NA 0.9307168      NA
#> 9      6   13 biomarker2 not in Mutant  4 0.1142857        NA 0.5617421      NA
#>   nsplit leaves
#> 1      0      1
#> 2      1      2
#> 3      1      2
#> 4      2      3
#> 5      2      3
#> 6      3      4
#> 7      3      4
#> 8      4      5
#> 9      4      5
df_train_summary$summary_table
#>   nsplit leaves        CP AvgRelError TotRelError    Xerror Xstd      Eval
#> 1      0      1 0.5943635   1.0000000   1.0000000 1.0000000    0 1.0000000
#> 2      1      2 0.5943635   0.4056365   0.8112730 1.0000000    0 1.0000000
#> 3      2      3 0.4051834   0.2539198   0.7617594 0.6857143    0 0.6857143
#> 4      3      4 0.2303257   0.1753108   0.7012430 0.4285714    0 0.4285714
#> 5      4      5 0.2036921   0.1272203   0.6361014 0.3904762    0 0.3904762
```

And we can create a simple plot of the tree to better visualize the model.

```
PlotTree(df_model_train)
```


1
2
3
4
5
6
7
12
13


gender
35 (100%)
in F
not in F
biomarker1
13 (37.14%)
biomarker4
17 (48.57%)
in Mutant
not in Mutant
in Mutant
not in Mutant
Subtype 1
8 (22.86%)
Outcomes
response: Predicted = Yes (Freq: 75%)
exp_tt_timetoprogression_progression: Rate = 0.25 (Events: 2)
exp_tt_overallsurvivaltime_vitalstatus: Rate = 0.5 (Events: 4)
Subtype 2
4 (11.43%)
Outcomes
response: Predicted = Yes (Freq: 75%)
exp_tt_timetoprogression_progression: Rate = 0.75 (Events: 3)
exp_tt_overallsurvivaltime_vitalstatus: Rate = 0.5 (Events: 2)
biomarker2
11 (31.43%)
Subtype 3
5 (14.29%)
Outcomes
response: Predicted = No (Freq: 80%)
exp_tt_timetoprogression_progression: Rate = 0.4 (Events: 2)
exp_tt_overallsurvivaltime_vitalstatus: Rate = 0.6 (Events: 3)
in Mutant
not in Mutant
Subtype 4
4 (11.43%)
Outcomes
response: Predicted = Yes (Freq: 75%)
exp_tt_timetoprogression_progression: Rate = 0.5 (Events: 2)
exp_tt_overallsurvivaltime_vitalstatus: Rate = 0.5 (Events: 2)
Subtype 5
4 (11.43%)
Outcomes
response: Predicted = Yes (Freq: 75%)
exp_tt_timetoprogression_progression: Rate = 0.75 (Events: 3)
exp_tt_overallsurvivaltime_vitalstatus: Rate = 0.75 (Events: 3)

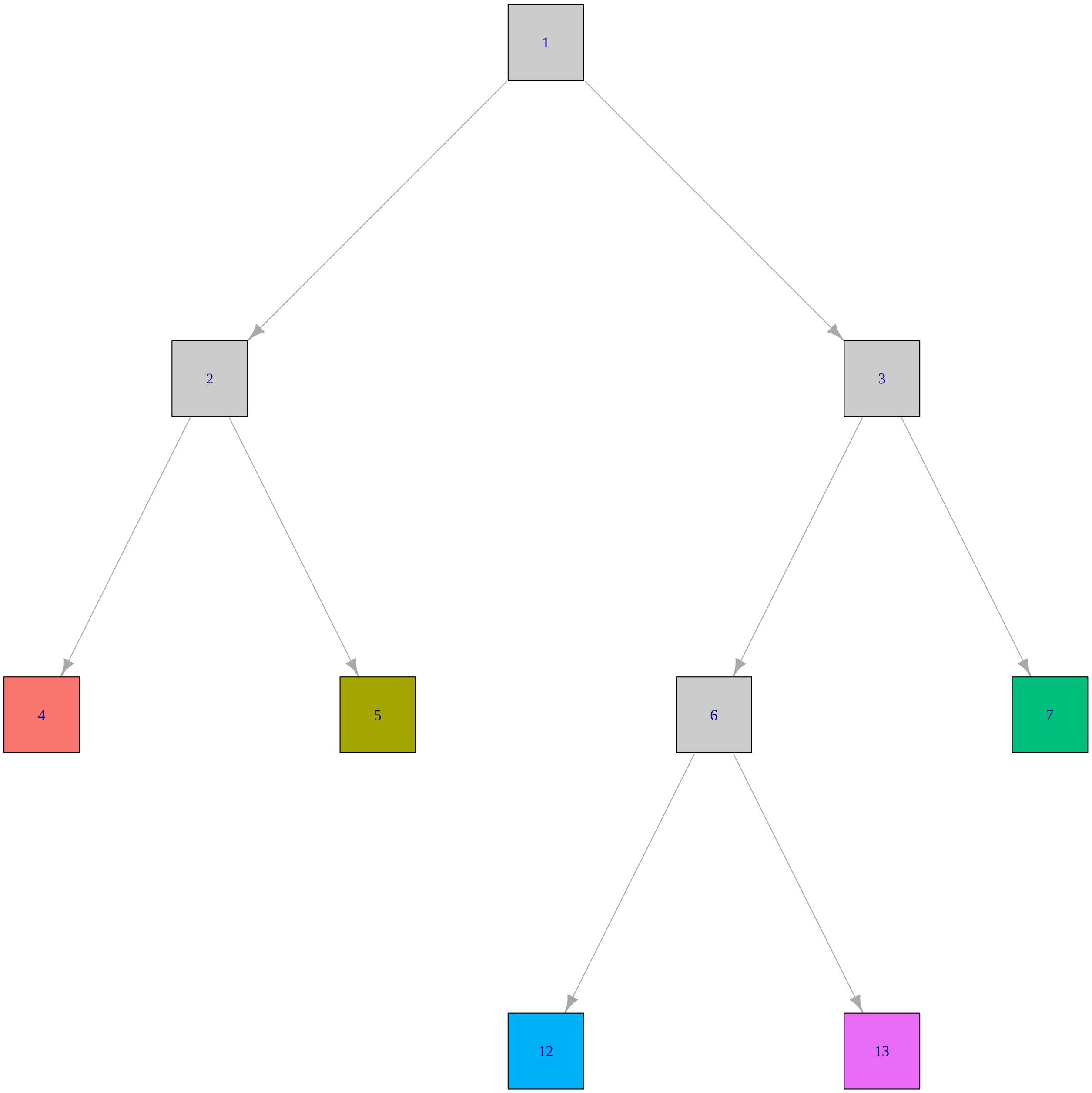
We have just generated a node ID tree, which gives a structural overview, and an annotated tree which includes the variables used for partitioning, the partitioning threshold, subtypes (leaf nodes), as well as the esimated values for outcomes in each subtype.

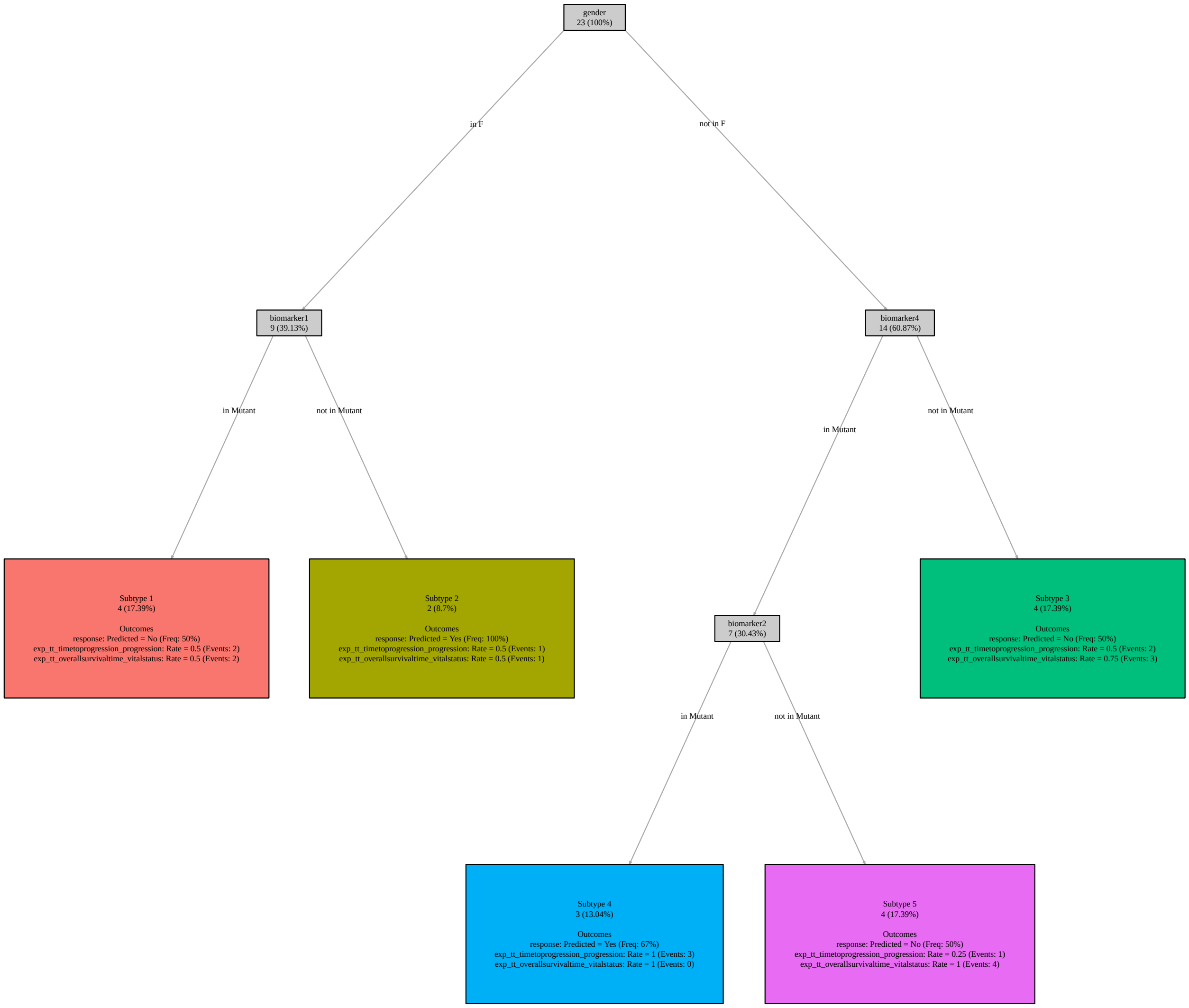
We can now apply this model to our test data, and take a look at how the model performs.

```
df_model_test <- MTTest(df_model_train, features, outcomes, outcome_defs, data_test)

df_test_summary <- MTPartSummary(df_model_test)
df_test_summary$node_data
#>   parent node                Partition  N      Pnode           CP    relerr
#> 1      0    1                     Root 23 1.00000000 -0.004251044 1.0000000
#> 2      1    2              gender in F  9 0.39130435  0.205984830 1.0604536
#> 3      1    3          gender not in F 14 0.60869565  0.069269904 0.9480485
#> 4      2    4     biomarker1 in Mutant  4 0.17391304           NA 1.0468251
#> 5      2    5 biomarker1 not in Mutant  2 0.08695652           NA 0.6621125
#> 6      3    6     biomarker4 in Mutant  7 0.30434783  0.330019624 0.8872324
#> 7      3    7 biomarker4 not in Mutant  4 0.17391304           NA 0.8703248
#> 8      6   12     biomarker2 in Mutant  3 0.13043478           NA 0.2619092
#> 9      6   13 biomarker2 not in Mutant  4 0.17391304           NA 0.8525163
#>   SXerror nsplit leaves
#> 1       1      0      1
#> 2       1      1      2
#> 3       1      1      2
#> 4      NA      2      3
#> 5      NA      2      3
#> 6       1      3      4
#> 7      NA      3      4
#> 8      NA      4      5
#> 9      NA      4      5
df_test_summary$summary_table
#>   nsplit leaves           CP AvgRelError TotRelError    Xerror Xstd      Eval
#> 1      0      1 -0.004251044   1.0000000   1.0000000 1.0000000    0 1.0000000
#> 2      1      2 -0.004251044   0.4960165   0.9920331 1.0000000    0 1.0000000
#> 3      2      3  0.038175858   0.2722348   0.8167045 0.6956522    0 0.6956522
#> 4      3      4  0.061383524   0.1652549   0.6610196 0.5000000    0 0.5000000
#> 5      4      5  0.074402601   0.1146836   0.5734181 0.4347826    0 0.4347826
```

Lastly, we can make a plot of the test model.

```
PlotTree(df_model_test)
```


1
2
3
4
5
6
7
12
13


gender
23 (100%)
in F
not in F
biomarker1
9 (39.13%)
biomarker4
14 (60.87%)
in Mutant
not in Mutant
in Mutant
not in Mutant
Subtype 1
4 (17.39%)
Outcomes
response: Predicted = No (Freq: 50%)
exp_tt_timetoprogression_progression: Rate = 0.5 (Events: 2)
exp_tt_overallsurvivaltime_vitalstatus: Rate = 0.5 (Events: 2)
Subtype 2
2 (8.7%)
Outcomes
response: Predicted = Yes (Freq: 100%)
exp_tt_timetoprogression_progression: Rate = 0.5 (Events: 1)
exp_tt_overallsurvivaltime_vitalstatus: Rate = 0.5 (Events: 1)
biomarker2
7 (30.43%)
Subtype 3
4 (17.39%)
Outcomes
response: Predicted = No (Freq: 50%)
exp_tt_timetoprogression_progression: Rate = 0.5 (Events: 2)
exp_tt_overallsurvivaltime_vitalstatus: Rate = 0.75 (Events: 3)
in Mutant
not in Mutant
Subtype 4
3 (13.04%)
Outcomes
response: Predicted = Yes (Freq: 67%)
exp_tt_timetoprogression_progression: Rate = 1 (Events: 3)
exp_tt_overallsurvivaltime_vitalstatus: Rate = 1 (Events: 0)
Subtype 5
4 (17.39%)
Outcomes
response: Predicted = No (Freq: 50%)
exp_tt_timetoprogression_progression: Rate = 0.25 (Events: 1)
exp_tt_overallsurvivaltime_vitalstatus: Rate = 1 (Events: 4)

MuTATE in LGG

The following analysis uses the example of lower-grade glioma (LGG) published by The Cancer Genome Atlas in 2015 (Cancer Genome Atlas Research Network, Brat DJ, Verhaak RG, et al. Comprehensive, Integrative Genomic Analysis of Diffuse Lower-Grade Gliomas. N Engl J Med. 2015;372(26):2481-2498. doi:10.1056/NEJMoa1402121).

We can download and install the published data from the GDC.

```
# URL of the dataset
url <- "https://api.gdc.cancer.gov/data/25c7d869-78b7-428f-bc99-b647bc3f8f81"

# File path to save the downloaded dataset
file_path <- "lgg_set.xlsx"

# Download the dataset
downloader::download(url, destfile = file_path, mode = "wb")

# Find the dataset sheet names
sheet_names <- readxl::excel_sheets(file_path)
```

```
sheet_names
#> [1] "a - Clusters"     "b - Genetics"     "c - Availability" "d - Clinical"
#> [5] "e - Clinical Key"
```

We can take a look at the data in this excel file. Here, we will build a model using genetic characteristics and clinical endpoints for the LGG cohort. We will keep data from both the clinical and genetic sheets and prepare data for modeling.

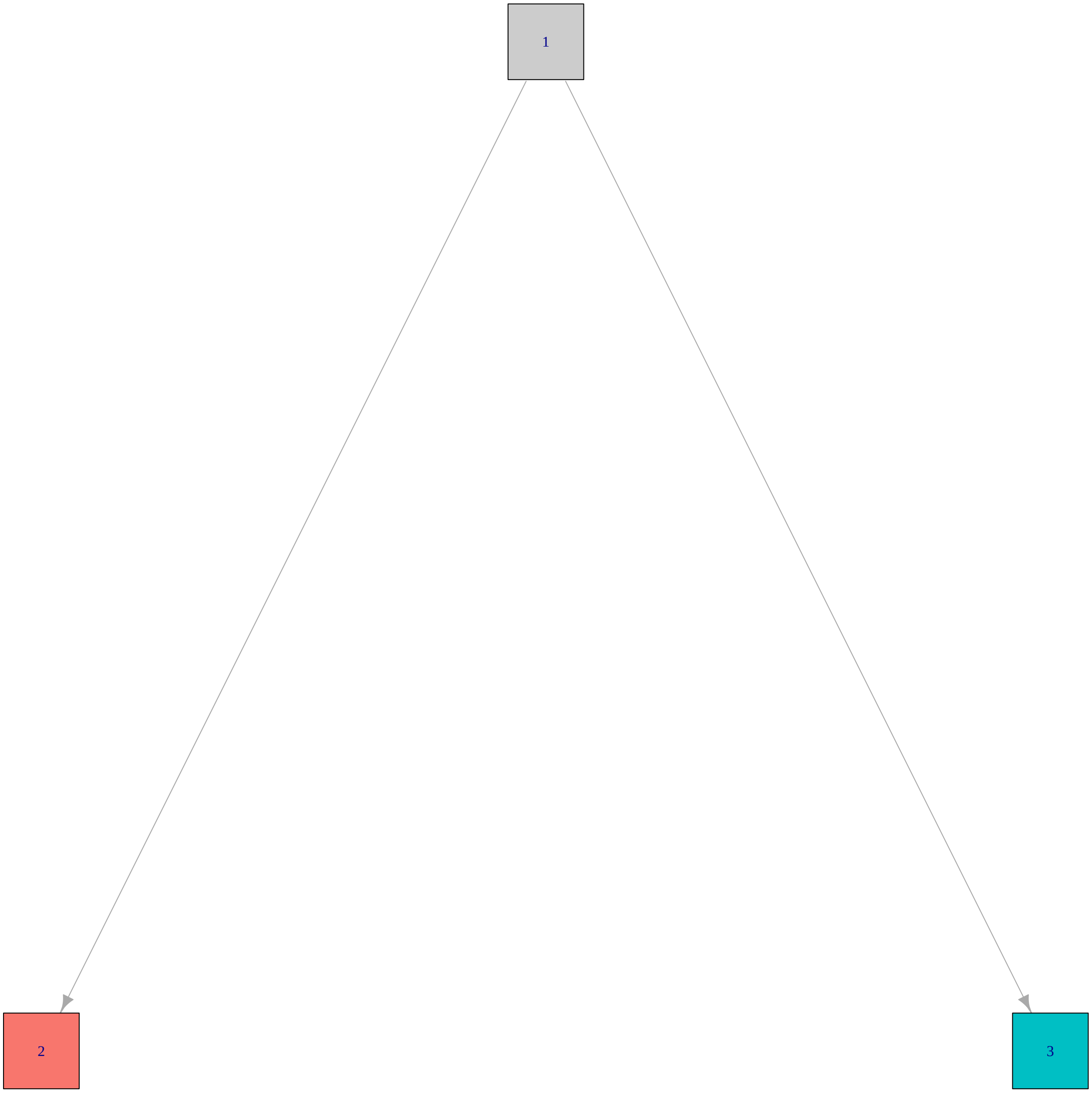
```
# Load the clinical and genetic data
clinical_data <- readxl::read_excel(file_path, sheet = sheet_names[[4]])
genetic_data  <- readxl::read_excel(file_path, sheet = sheet_names[[2]])

lgg_data <- merge(clinical_data[,c("Tumor", "death01", "daystolastordeath",
                                   "new_tumor01", "days_to_new_tumor",
                                   "ProgFreeSurvEvent01", "ProgFreeSurvTime_days")],
                  genetic_data[! names(genetic_data) %in%
                                 c("IDH/1p19q Subtype", "1p_19q_co_del_status")])

# Recode NA
lgg_data[lgg_data == "NA"] <- NA

# Convert character variables to factors
char_vars <- sapply(lgg_data, is.character)
lgg_data[, char_vars] <- lapply(lgg_data[, char_vars], factor)

# Convert integer variables to numeric
int_vars <- sapply(lgg_data, is.integer)
lgg_data[, int_vars] <- lapply(lgg_data[, int_vars], as.numeric)

# Convert 0 or 1 variables to binary
bin_vars <- sapply(lgg_data, function(x) all(x %in% c(0, 1)))
lgg_data[, bin_vars] <- lapply(lgg_data[, bin_vars], as.logical)

# Clean variable names
names(lgg_data) <- gsub("_","",names(lgg_data))
names(lgg_data) <- gsub(" ","",names(lgg_data))

# Prepare survival outcomes
lgg_data <- (lgg_data[complete.cases(lgg_data[, c("death01", "daystolastordeath",
                                   "newtumor01", "daystonewtumor",
                                   "ProgFreeSurvEvent01", "ProgFreeSurvTimedays")]), ])
temp  <- coxph(Surv(as.numeric(daystolastordeath), as.numeric(death01)) ~ 1, data = lgg_data)
lgg_data$exp_tt_daystolastordeath_death01 <- predict(temp, type = 'expected')
temp  <- coxph(Surv(as.numeric(daystonewtumor), as.numeric(newtumor01)) ~ 1, data = lgg_data)
lgg_data$exp_tt_daystonewtumor_newtumor01 <- predict(temp, type = 'expected')
temp  <- coxph(Surv(as.numeric(daystonewtumor), as.numeric(newtumor01)) ~ 1, data = lgg_data)
lgg_data$exp_tt_ProgFreeSurvTimedays_ProgFreeSurvEvent01 <- predict(temp, type = 'expected')

# Feature and outcome designation
features <- names(lgg_data[! names(lgg_data) %in%
                             c('Tumor', 'TERTpromotermutationtargetedsequencing',
                               'death01', 'daystolastordeath', 'newtumor01',
                               'daystonewtumor', 'ProgFreeSurvEvent01',
                               'ProgFreeSurvTimedays', 'exp_tt_daystolastordeath_death01',
                               'exp_tt_daystonewtumor_newtumor01',
                               'exp_tt_ProgFreeSurvTimedays_ProgFreeSurvEvent01')])

outcomes <- c('exp_tt_daystolastordeath_death01',
              'exp_tt_daystonewtumor_newtumor01',
              'exp_tt_ProgFreeSurvTimedays_ProgFreeSurvEvent01')

surv     <- c('death01', 'daystolastordeath',
              'newtumor01', 'daystonewtumor',
              'ProgFreeSurvEvent01', 'ProgFreeSurvTimedays')

lgg_data[, surv] <- lapply(lgg_data[, surv], as.numeric)

# Keep variables of interest
lgg_data <- lgg_data[,c(features, outcomes, surv)]

# Clean the data
lgg_clean <- clean_data(lgg_data, varprop = 0.5, nlevel = 10, nprop = 0.5)

# Remove any variables that were dropped in data cleaning
features <- features[(features %in% colnames(lgg_clean))]
outcomes <- outcomes[(outcomes %in% colnames(lgg_clean))]
surv     <- surv[(surv %in% colnames(lgg_clean))]

# Set outcome variable types for final outcome set
outcome_defs <- c("Surv","Surv","Surv")

# Print first few rows of the data set
head(lgg_clean)
#>   ARID1A                               ATRX              BCOR BRAF
#> 1     wt                    Frame_Shift_Del                wt   wt
#> 2     wt                                 wt                wt   wt
#> 3     wt                    Frame_Shift_Del Missense_Mutation   wt
#> 4     wt Frame_Shift_Del, Nonsense_Mutation                wt   wt
#> 6     wt                                 wt                wt   wt
#> 7     wt                                 wt                wt   wt
#>               CDKN2A CIC EGFR             FUBP1              IDH1 IDH2 NF1
#> 1                 wt  wt   wt                wt Missense_Mutation   wt  wt
#> 2                 wt  wt   wt                wt                wt   wt  wt
#> 3                 wt  wt   wt                wt Missense_Mutation   wt  wt
#> 4 Missense_Mutation  wt   wt                wt Missense_Mutation   wt  wt
#> 6                 wt  wt   wt Nonsense_Mutation Missense_Mutation   wt  wt
#> 7                 wt  wt   wt                wt Missense_Mutation   wt  wt
#>                       NOTCH1            PDGFRA PIK3CA            PIK3R1
#> 1                         wt                wt     wt                wt
#> 2                         wt Missense_Mutation     wt                wt
#> 3                         wt                wt     wt Missense_Mutation
#> 4                         wt                wt     wt                wt
#> 6 In_Frame_Del, Splice_Site                wt     wt                wt
#> 7                         wt                wt     wt                wt
#>               PLCG1 PTEN PTPN11 RB1           SMARCA4 TCF12              TP53
#> 1                wt   wt     wt  wt                wt    wt Missense_Mutation
#> 2 Missense_Mutation   wt     wt  wt                wt    wt                wt
#> 3                wt   wt     wt  wt                wt    wt Missense_Mutation
#> 4                wt   wt     wt  wt                wt    wt Missense_Mutation
#> 6                wt   wt     wt  wt                wt    wt                wt
#> 7                wt   wt     wt  wt Missense_Mutation    wt Missense_Mutation
#>   ZBTB20 ZCCHC12 exp_tt_daystolastordeath_death01
#> 1     wt      wt                      0.026232709
#> 2     wt      wt                      0.111781110
#> 3     wt      wt                      0.017098575
#> 4     wt      wt                      0.258884101
#> 6     wt      wt                      0.071711780
#> 7     wt      wt                      0.008247458
#>   exp_tt_daystonewtumor_newtumor01
#> 1                       0.07855266
#> 2                       0.75374323
#> 3                       0.03762626
#> 4                       0.46271971
#> 6                       0.13148001
#> 7                       0.05078432
#>   exp_tt_ProgFreeSurvTimedays_ProgFreeSurvEvent01 death01 daystolastordeath
#> 1                                      0.07855266       1                37
#> 2                                      0.75374323       2                91
#> 3                                      0.03762626       2                29
#> 4                                      0.46271971       1               193
#> 6                                      0.13148001       1                69
#> 7                                      0.05078432       1                21
#>   newtumor01 daystonewtumor ProgFreeSurvEvent01 ProgFreeSurvTimedays
#> 1          1             43                   1                   43
#> 2          2            220                   2                  220
#> 3          2             20                   2                   20
#> 4          1            182                   1                  182
#> 6          1             68                   1                   68
#> 7          1             27                   1                   27
```

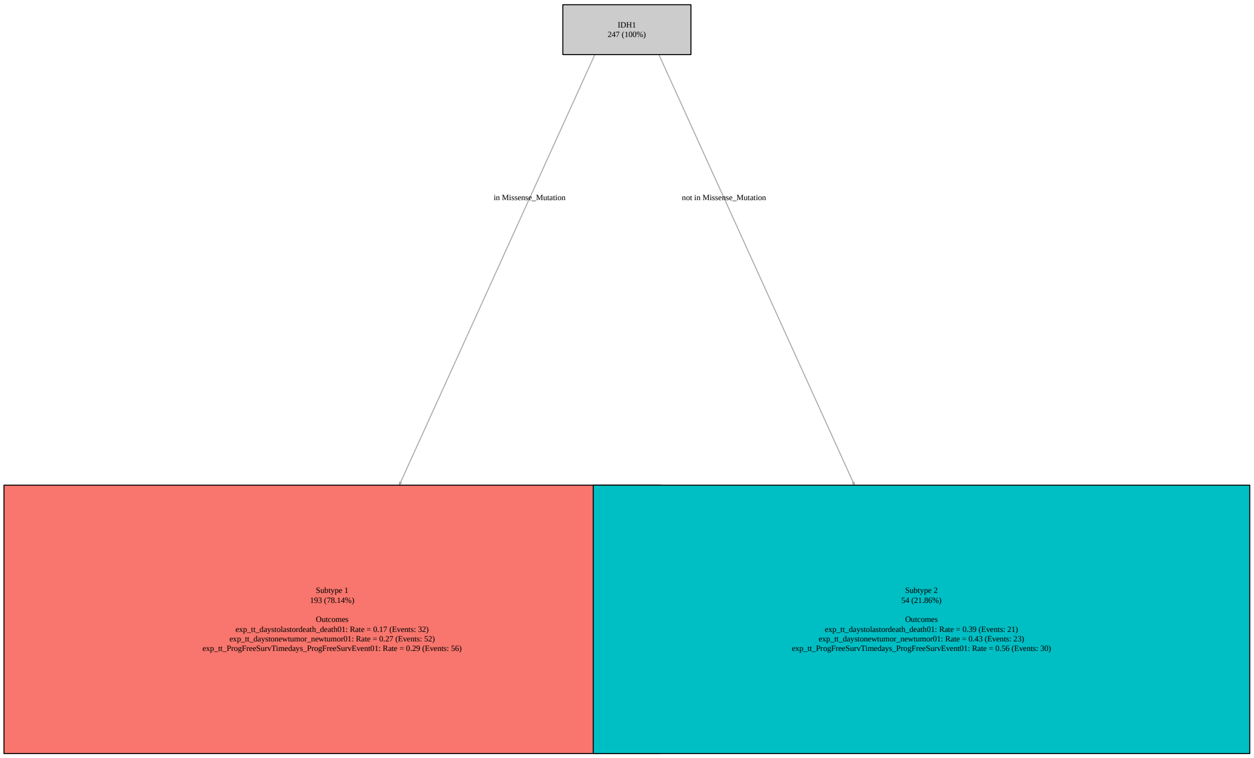
We can now construct a model for our LGG data.

```
lgg_model <- MTPart(features, outcomes, outcome_defs, lgg_clean,
                    evalmethod = "avgIG", alpha = 0.05, IGcutoff = 0.95,
                    depth=4, nodesize=5, cp=-2,
                    splitmin=floor(nodesize/1))

lgg_model_summary <- MTPartSummary(lgg_model)
lgg_model_summary$node_data
#>   parent node                    Partition   N     Pnode        CP    relerr
#> 1      0    1                         Root 247 1.0000000 0.5111636 1.0000000
#> 2      1    2     IDH1 in Missense_Mutation 193 0.7813765        NA 1.0839898
#> 3      1    3 IDH1 not in Missense_Mutation  54 0.2186235        NA 0.5976878
#>   SXerror nsplit leaves
#> 1       1      0      1
#> 2      NA      1      2
#> 3      NA      1      2
lgg_model_summary$summary_table
#>   nsplit leaves        CP AvgRelError TotRelError Xerror Xstd Eval
#> 1      0      1 0.5111636   1.0000000   1.0000000      1    0    1
#> 2      1      2 0.5111636   0.4888364   0.9776727      1    0    1
```

Lastly, we can make a plot of the model.

```
PlotTree(lgg_model)
```




IDH1
247 (100%)
in Missense_Mutation
not in Missense_Mutation
Subtype 1
193 (78.14%)
Outcomes
exp_tt_daystolastordeath_death01: Rate = 0.17 (Events: 32)
exp_tt_daystonewtumor_newtumor01: Rate = 0.27 (Events: 52)
exp_tt_ProgFreeSurvTimedays_ProgFreeSurvEvent01: Rate = 0.29 (Events: 56)
Subtype 2
54 (21.86%)
Outcomes
exp_tt_daystolastordeath_death01: Rate = 0.39 (Events: 21)
exp_tt_daystonewtumor_newtumor01: Rate = 0.43 (Events: 23)
exp_tt_ProgFreeSurvTimedays_ProgFreeSurvEvent01: Rate = 0.56 (Events: 30)