

# Iris Data

Alan n. Inglis

2022-08-24

## Load libraries:

```
library(bartMachine) # for model
library(dbarts) # for model
library(bartMan) # for visualizations
library(ggplot2) # for visualizations
```

## Read in and setup data:

```
# load data
data(iris)
iris2 = iris[1:100,]
iris2$Species <- factor(iris2$Species)
iris2$Species <- ifelse(iris2$Species == "setosa", 0, 1)
```

## Build models

```
# bartMachine
set.seed(100)
bm <- bartMachine(X = iris2[,1:4],
                  y = iris2[,5],
                  num_burn_in = 250,
                  num_trees = 20,
                  seed = 100)

## bartMachine initializing with 20 trees...
## bartMachine vars checked...
## bartMachine java init...
## bartMachine factors created...
## bartMachine before preprocess...
## bartMachine after preprocess... 5 total features...
## bartMachine sigsq estimated...
## bartMachine training data finalized...
## Now building bartMachine for regression...
## evaluating in sample data...done

# dbarts
set.seed(100)
dB <- bart(x.train = iris2[,1:4],
```

```

        y.train = iris2[,5],
        ntree = 20,
        keeptrees = TRUE,
        nskip = 250,
        ndpost = 1000
    )

##
## Running BART with binary y
##
## number of trees: 20
## number of chains: 1, number of threads 1
## tree thinning rate: 1
## Prior:
## k prior fixed to 2.000000
## power and base for tree prior: 2.000000 0.950000
## use quantiles for rule cut points: false
## proposal probabilities: birth/death 0.50, swap 0.10, change 0.40; birth 0.50
## data:
## number of training observations: 100
## number of test observations: 0
## number of explanatory variables: 4
##
## Cutoff rules c in  $x \leq c$  vs  $x > c$ 
## Number of cutoffs: (var: number of possible c):
## (1: 100) (2: 100) (3: 100) (4: 100)
## offsets:
## reg : 0.00 0.00 0.00 0.00 0.00
## Running mcmc loop:
## iteration: 100 (of 1000)
## iteration: 200 (of 1000)
## iteration: 300 (of 1000)
## iteration: 400 (of 1000)
## iteration: 500 (of 1000)
## iteration: 600 (of 1000)
## iteration: 700 (of 1000)
## iteration: 800 (of 1000)
## iteration: 900 (of 1000)
## iteration: 1000 (of 1000)
## total seconds in loop: 0.112283
##
## Tree sizes, last iteration:
## [1] 2 3 2 2 2 2 2 2 2 3 2 2 2 4 2 2 2 2
## 2 2
##
## Variable Usage, last iteration (var:count):
## (1: 2) (2: 6) (3: 5) (4: 11)
## DONE BART

```

Create dataframe of trees

```
bmDF <- extractTreeData(model = bm, data = iris2)
dbDF <- extractTreeData(model = dB, data = iris2)
```

## Visualisations for Iris example:

Figure 2:

```
vimpPlot(bmDF, plotType = 'pointGrad', metric = 'median')
```

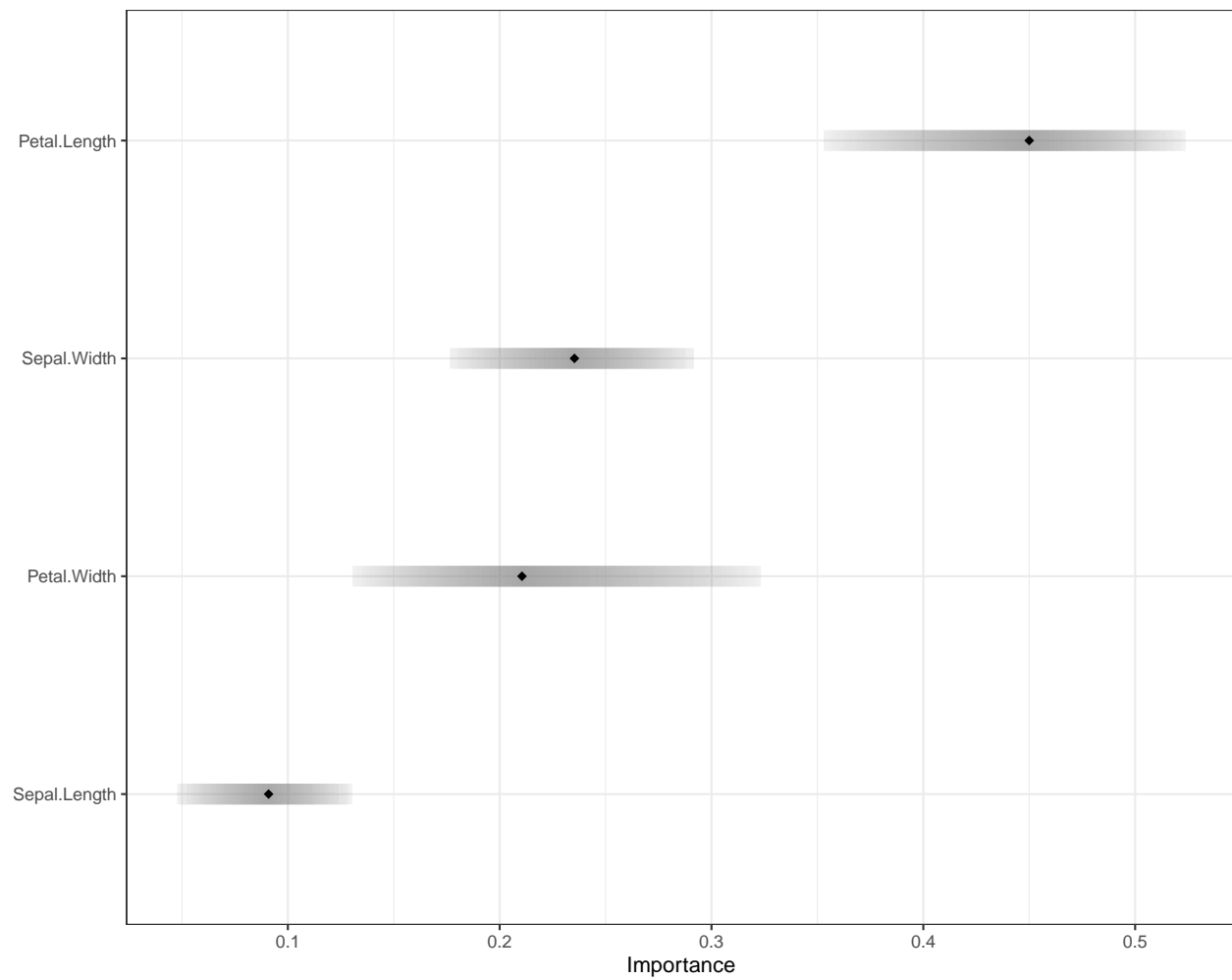


Figure 3:

```
# Set the colors (Need to automate this!!!)
colors <- scales::colour_ramp(
  colors = c(blue = '#FFFFCC', red = '#800026')
)((0:7)/7)

newCols <- RColorBrewer::brewer.pal(9, 'GnBu')
```

```

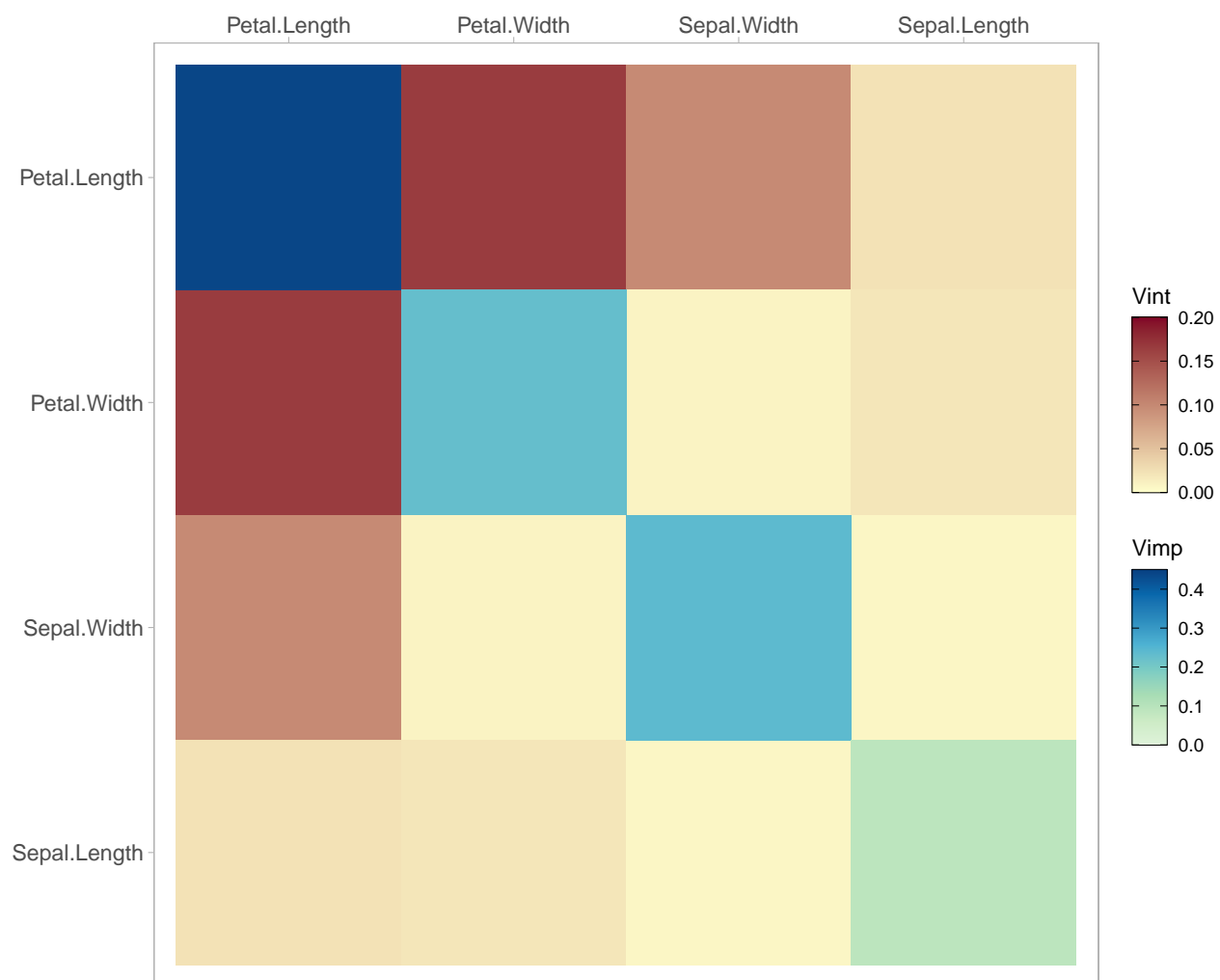
colors2 <- newCols[-1]

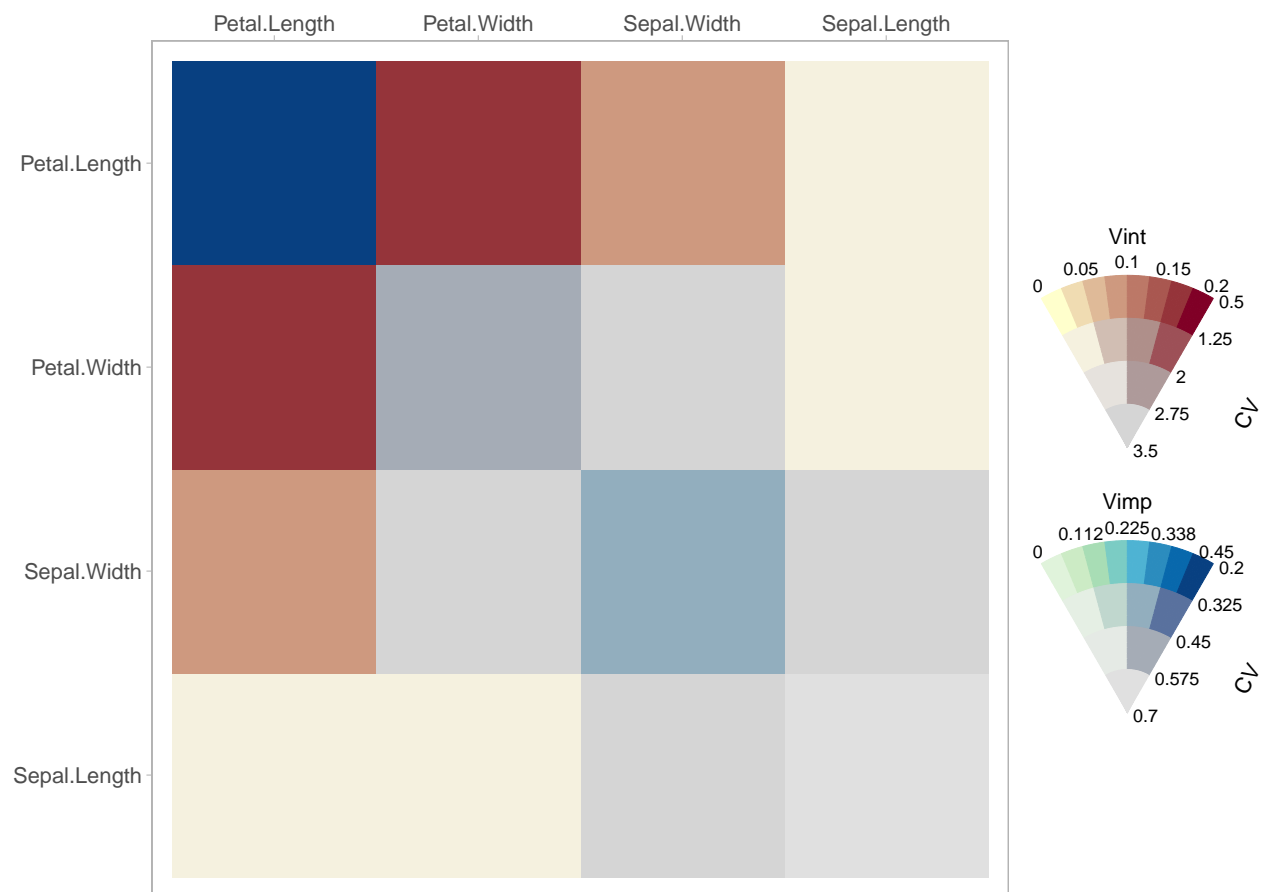
# create vivi matrices
myMat <- viviBartMatrix(bmDF,
                        type = 'vsup',
                        metric = 'propMean',
                        metricError = "CV")

# plot vivi-heatmap
vivid::viviHeatmap(myMat$actualMatrix,
                  impLims = c(0, 0.45),
                  intPal = colors,
                  impPal = colors2)

# plot vsup-heatmap
viviBartPlot(myMat,
             impLims = c(0,0.45),
             intPal = colors,
             impPal = colors2,
             max_desat = 1,
             pow_desat = 0.6,
             max_light = 0.6,
             pow_light = 1,
             label = 'CV')+
  theme(axis.text.x = element_text(hjust = 0.5))

```





**Figure 4:**

```
# plot all trees
plotAllTrees(bmDF, treeNo = 20)
```

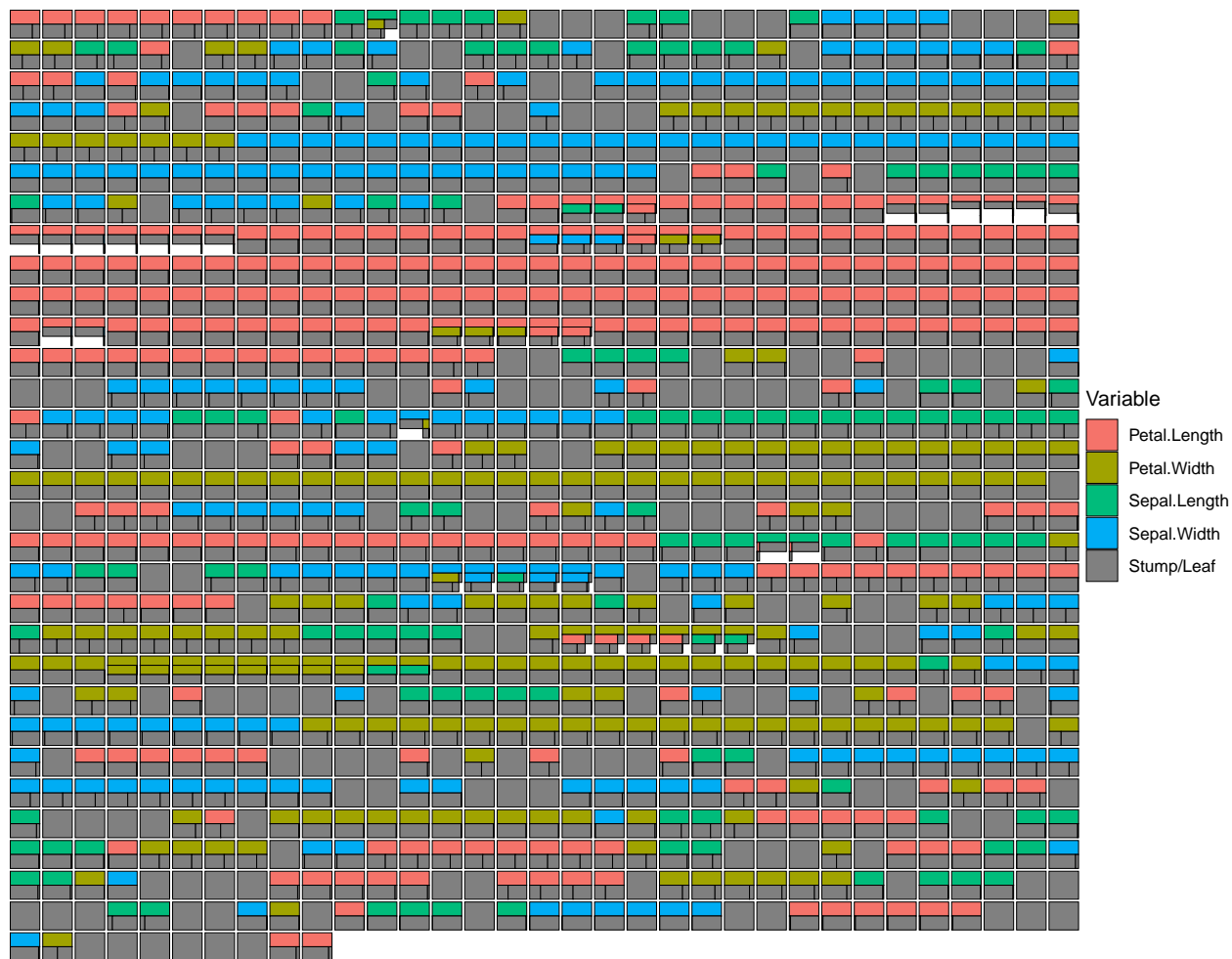


figure 5:

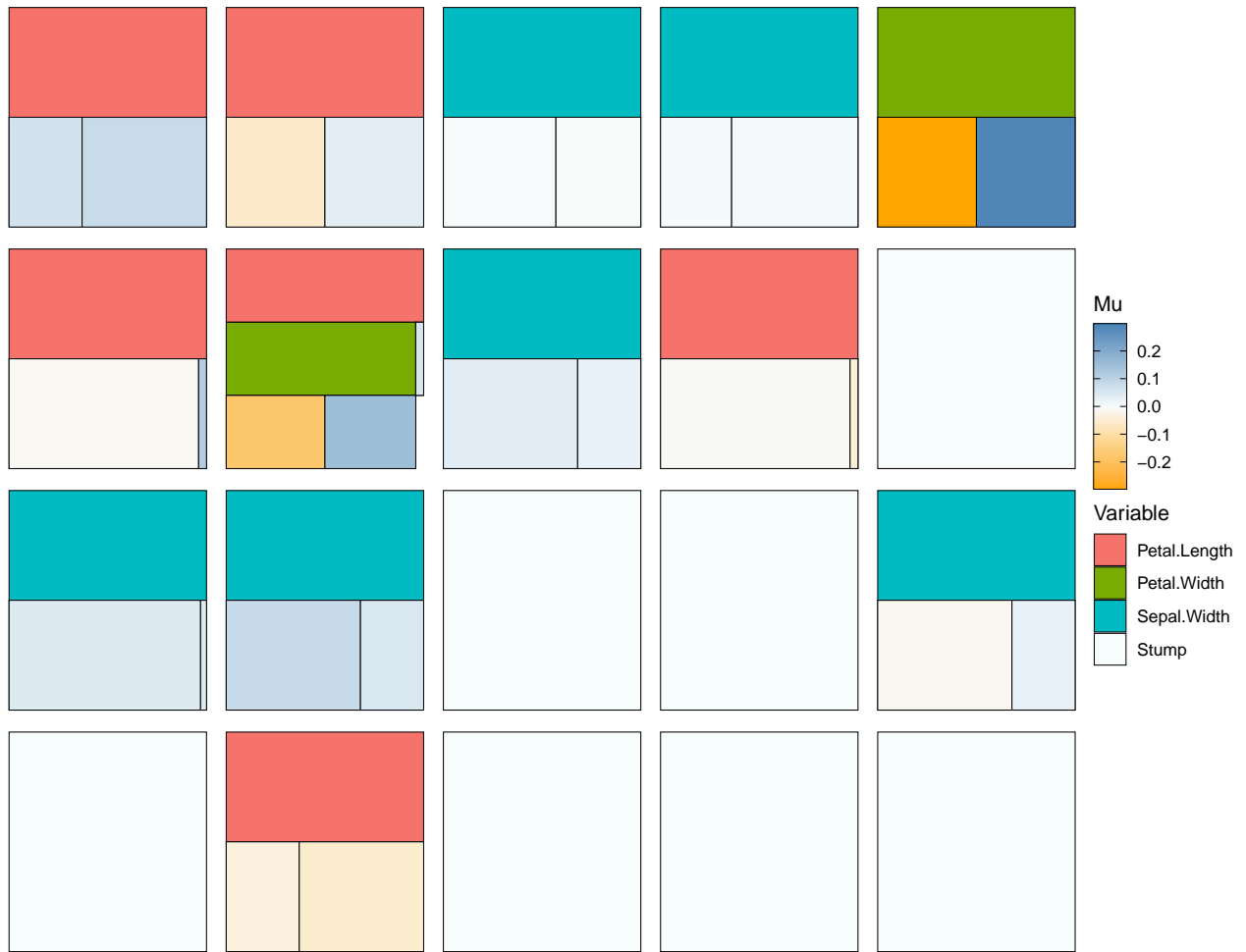
```
# finding iteration with lowest residual sd
bmPost <- bartMachine::bart_machine_get_posterior(bm, iris2[,1:4])

resid = NULL
for(i in 1:1000){
  resid[[i]] <- iris2[,5] - bmPost$y_hat_posterior_samples[,i]
}

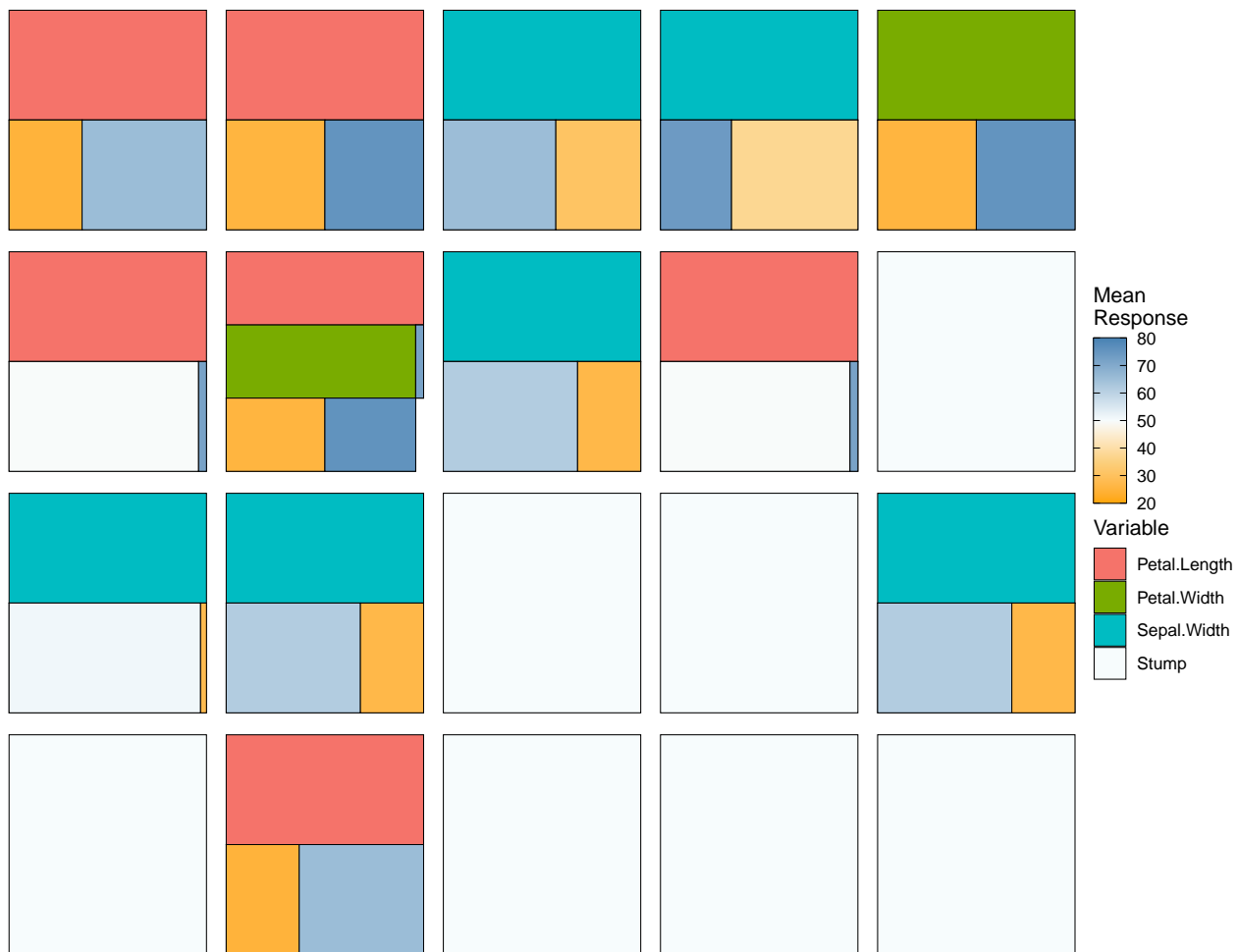
finalRes <- lapply(resid, sd)
lowestRes <- which.min(finalRes)

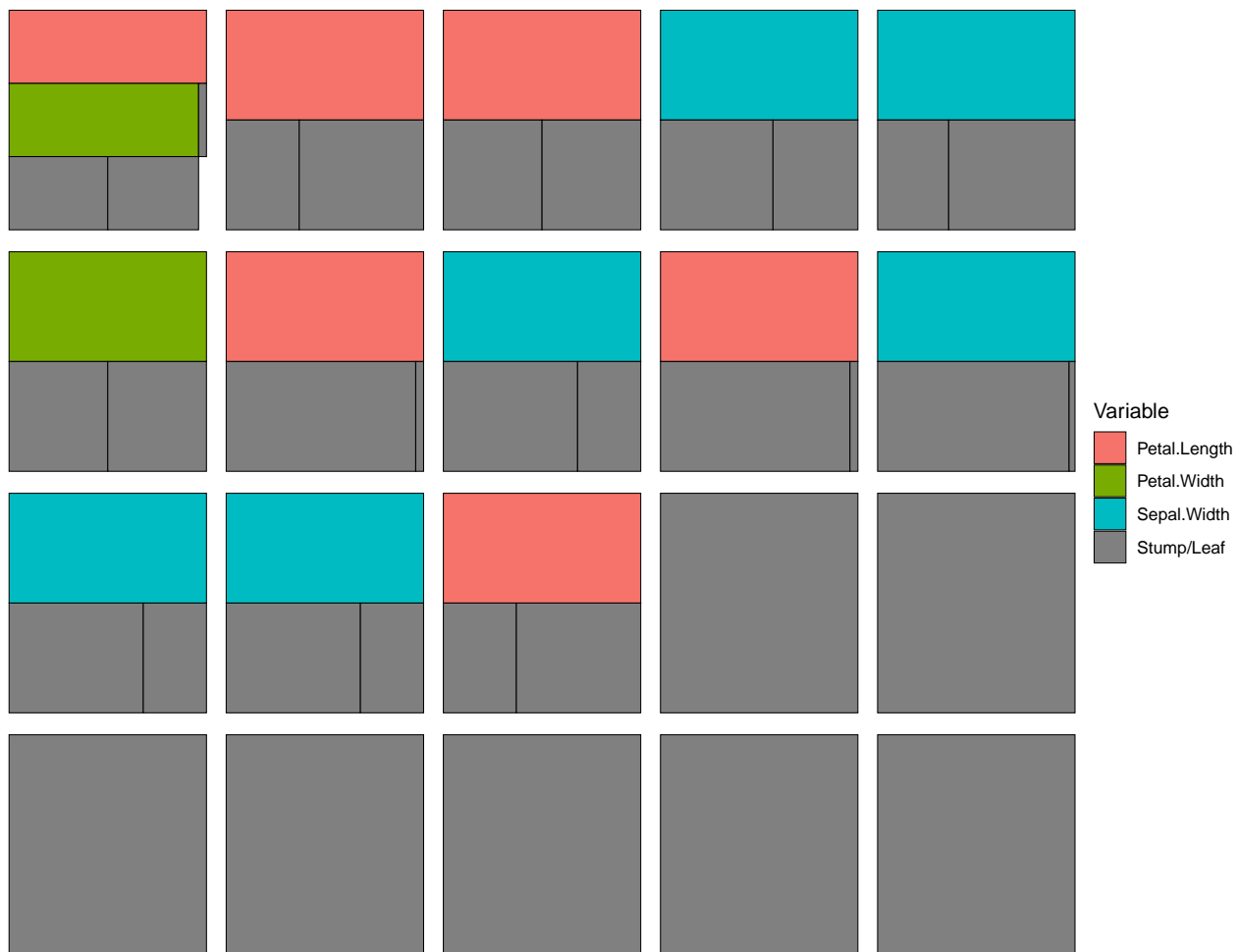
# select palette (need to automate this!!)
mypalette <- rev(colorRampPalette(c('steelblue', '#f7fcfd', 'orange'))(5))

# plot trees from selected iteration
plotAllTrees(bmDF, iter = lowestRes, sizeNode = T, fillBy = 'mu', pal = mypalette)
plotAllTrees(bmDF, iter = lowestRes, sizeNode = T, fillBy = 'response', pal = mypalette)
plotAllTrees(bmDF, iter = lowestRes, cluster = "depth")
plotAllTrees(bmDF, iter = lowestRes, cluster = "var")
```









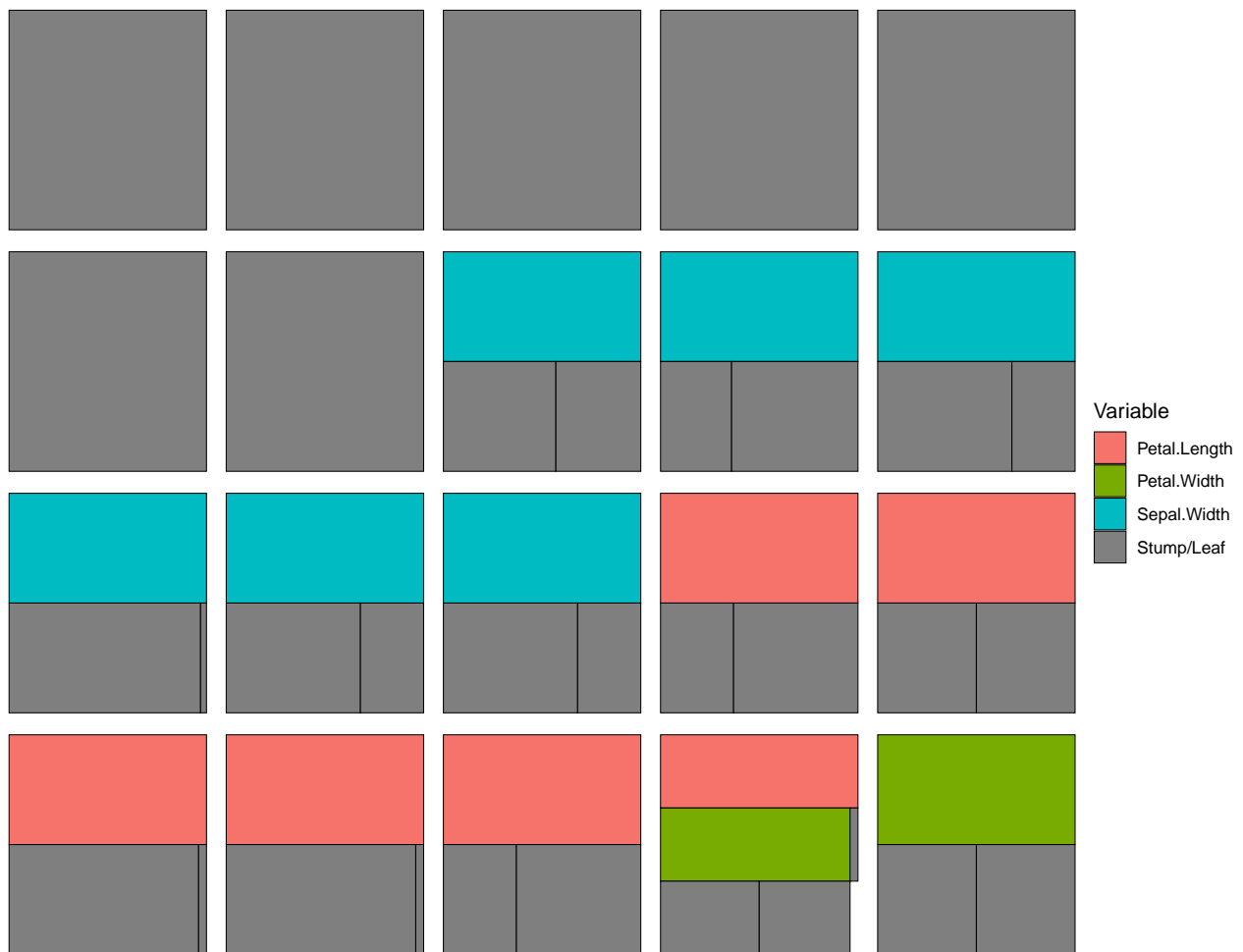


Figure 6:

```
# tree barplot
treeBarPlot(treeData = bmDF, topTrees = 10)
```

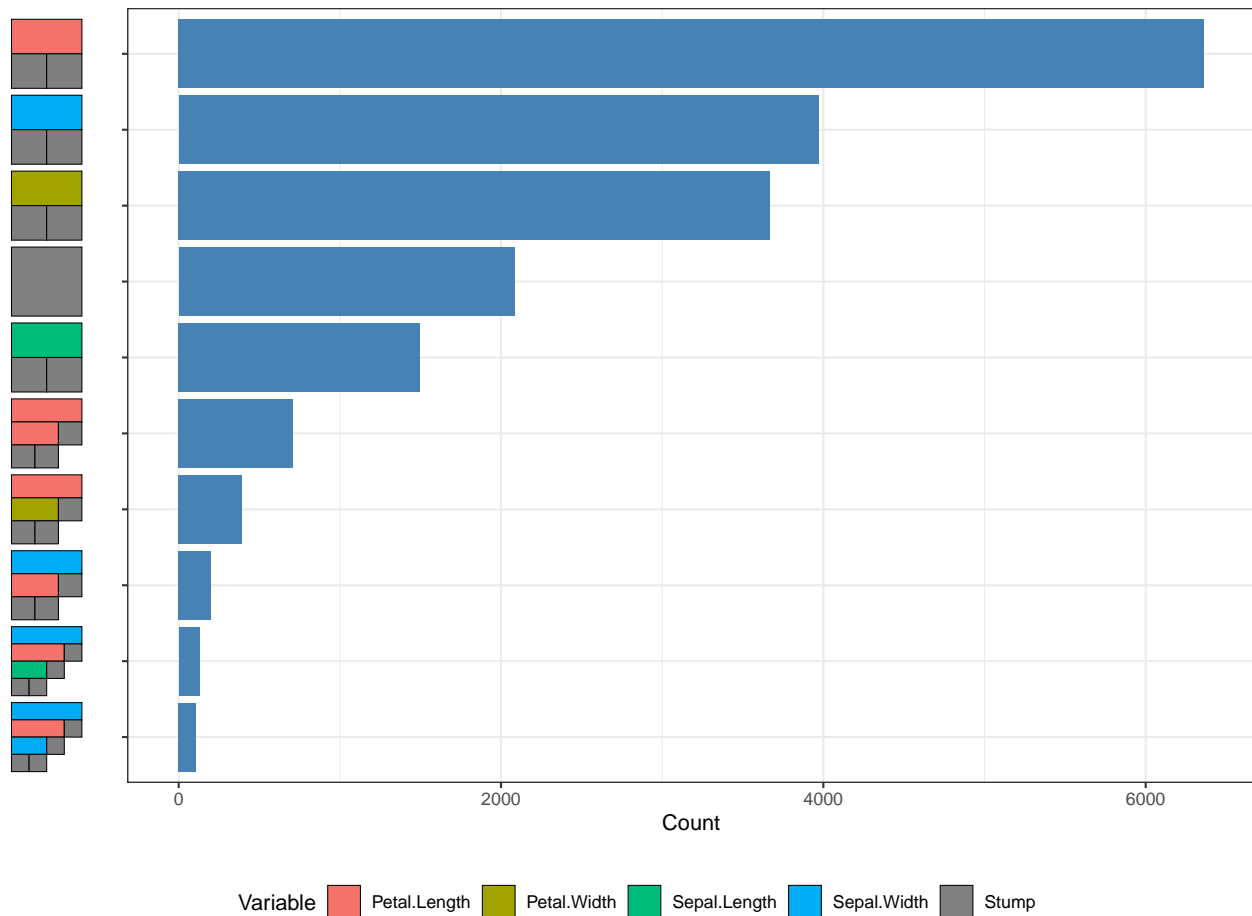
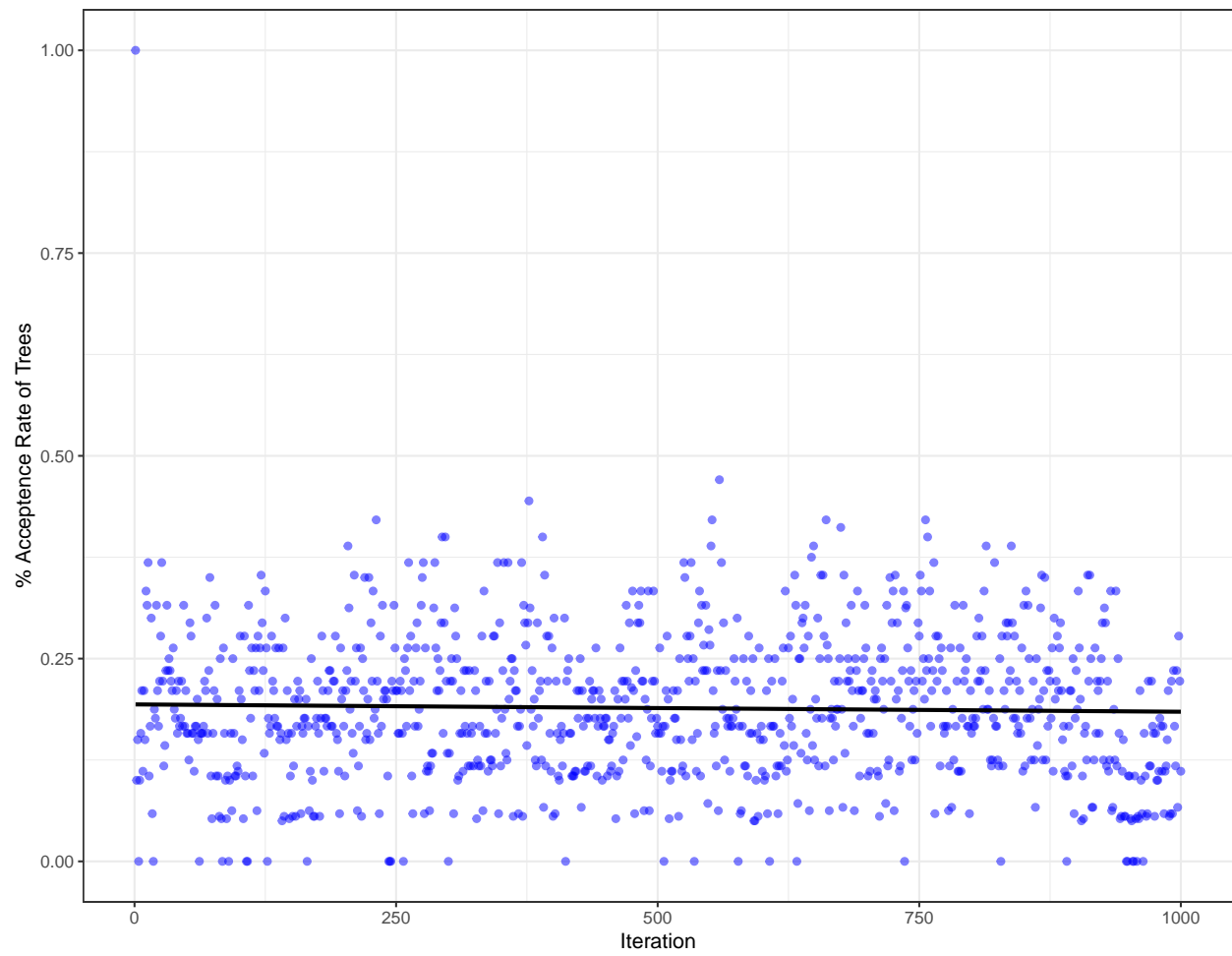


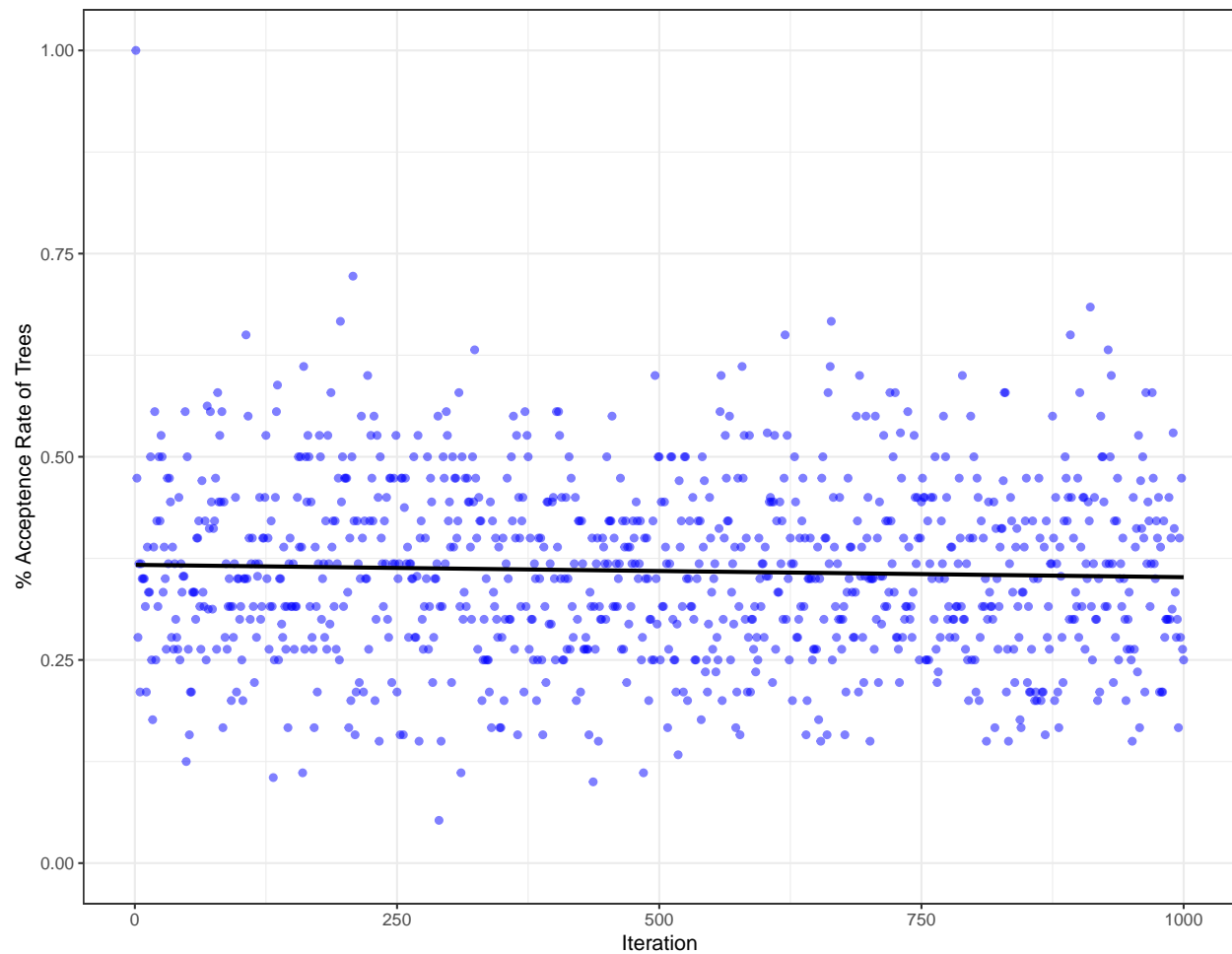
Figure 7:

```
# first get target proximity matrix
bmProx <- proximityMatrix(bmDF, iris2, reorder = T, normalize = T, iter = 736)
# plot MDS
mdsBart(treeData = bmDF, data = iris2, target = bmProx,
        plotType = 'interactive', showGroup = F)
```

Figure 8:

```
# acceptance rate (setting limits to be equal)
acceptRate(treeData = bmDF) + ylim(c(0, 1))
acceptRate(treeData = dbDF) + ylim(c(0, 1))
```

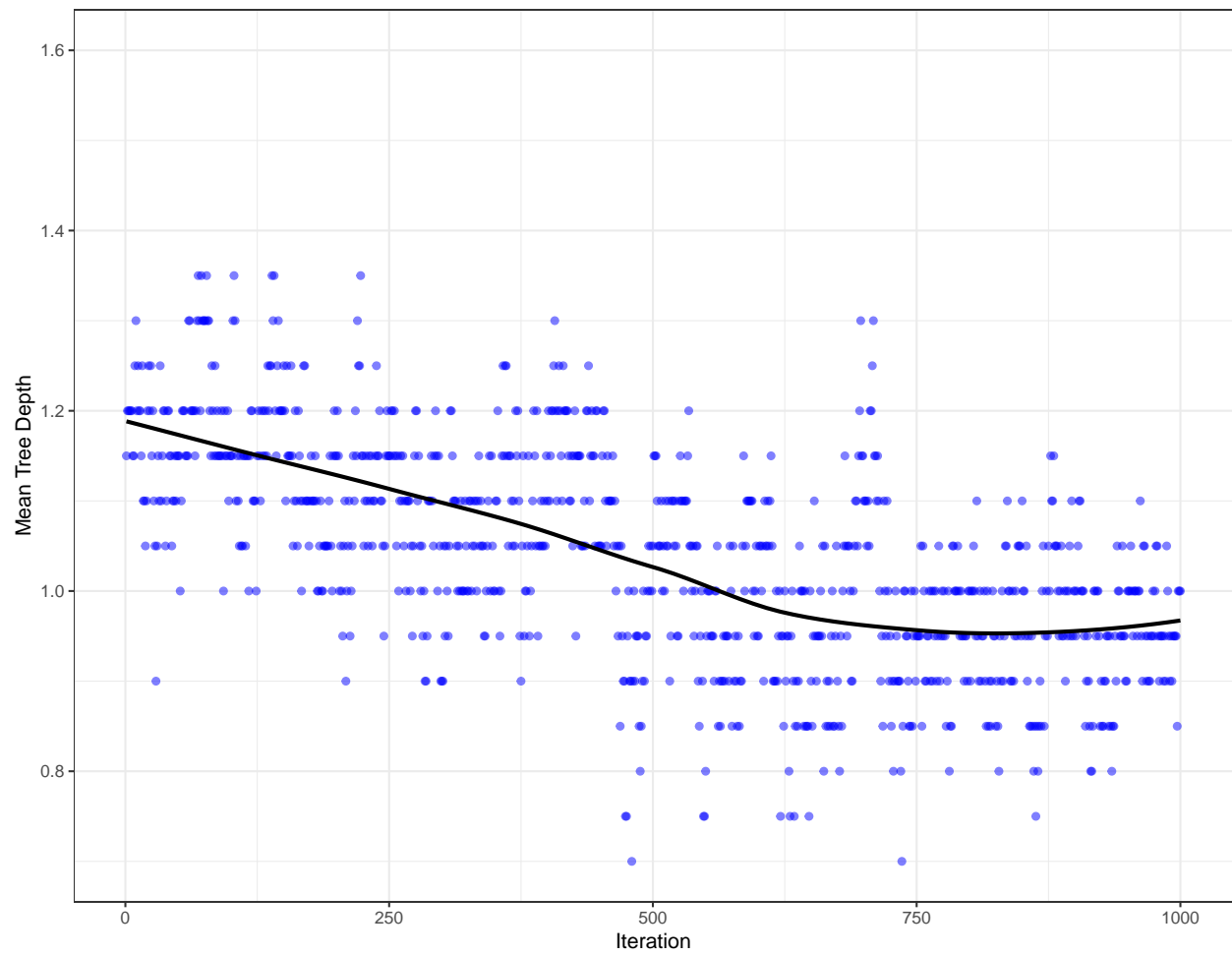


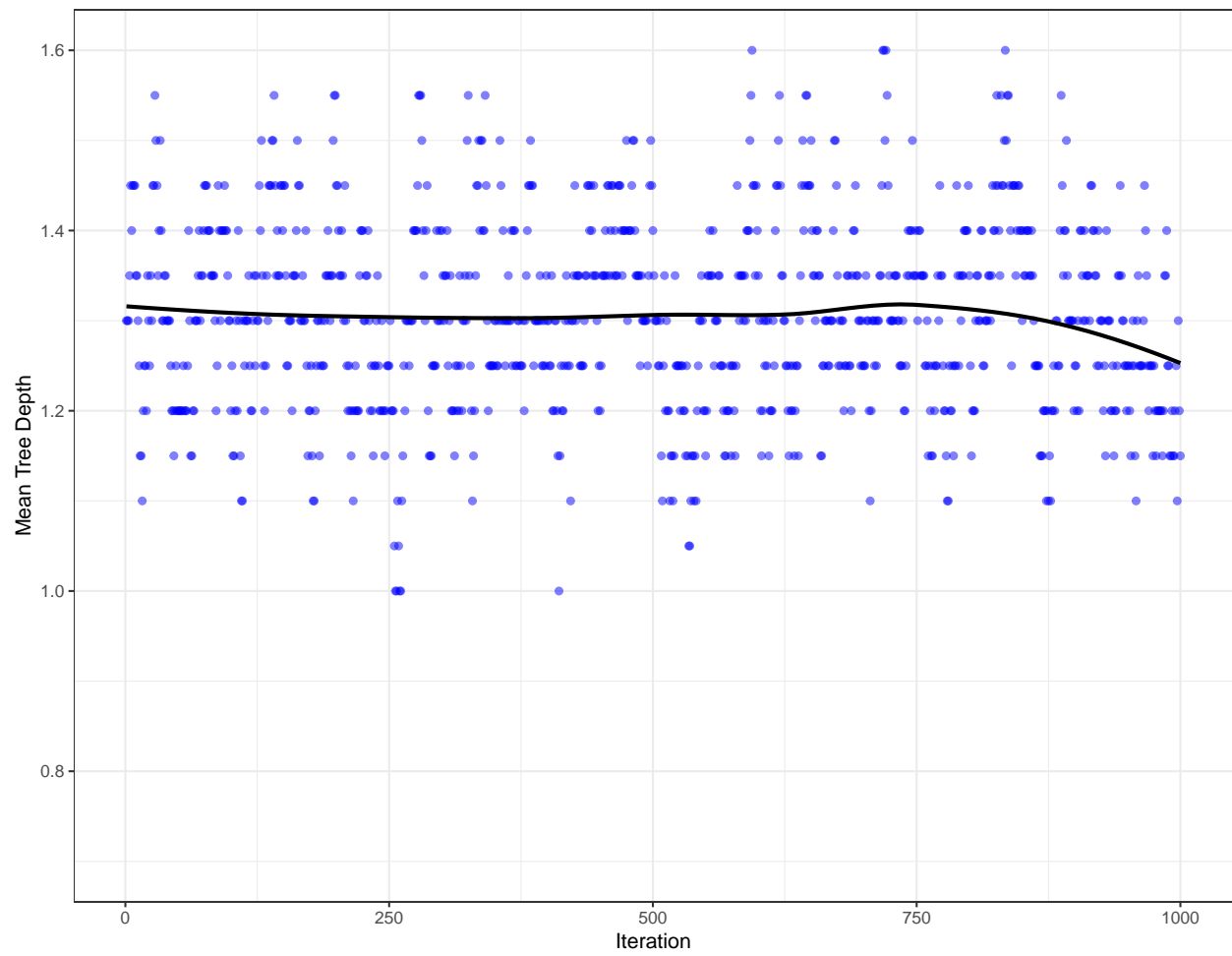


**Figure 9:**

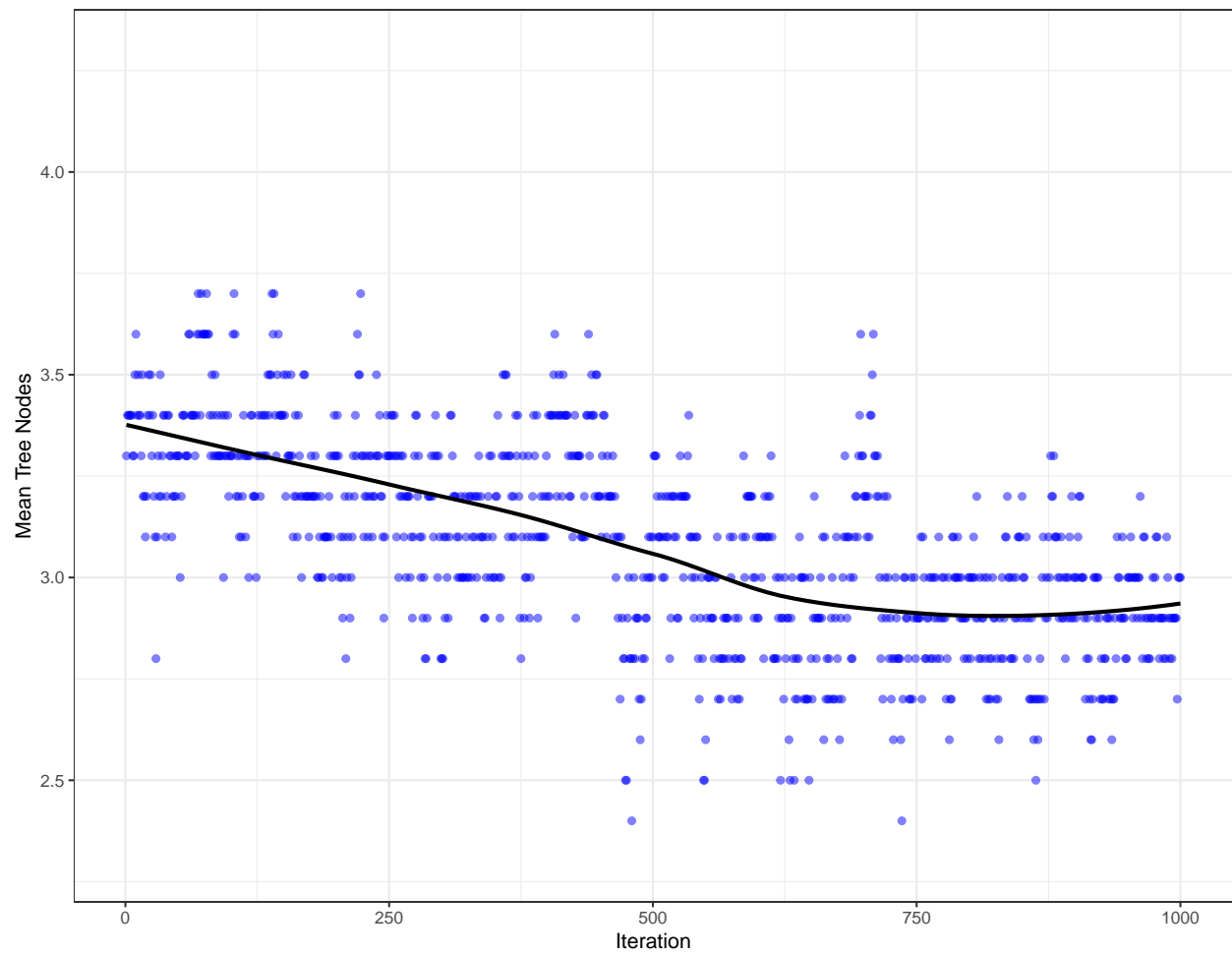
```
# tree depth per iteration (setting limits to be equal)
treeDepth(bmDF) + ylim(c(0.7, 1.6))
treeDepth(dbDF) + ylim(c(0.7, 1.6))

# tree nodes per iteration (setting limits to be equal)
treeNodes(bmDF) + ylim(c(2.3, 4.3))
treeNodes(dbDF) + ylim(c(2.3, 4.3))
```









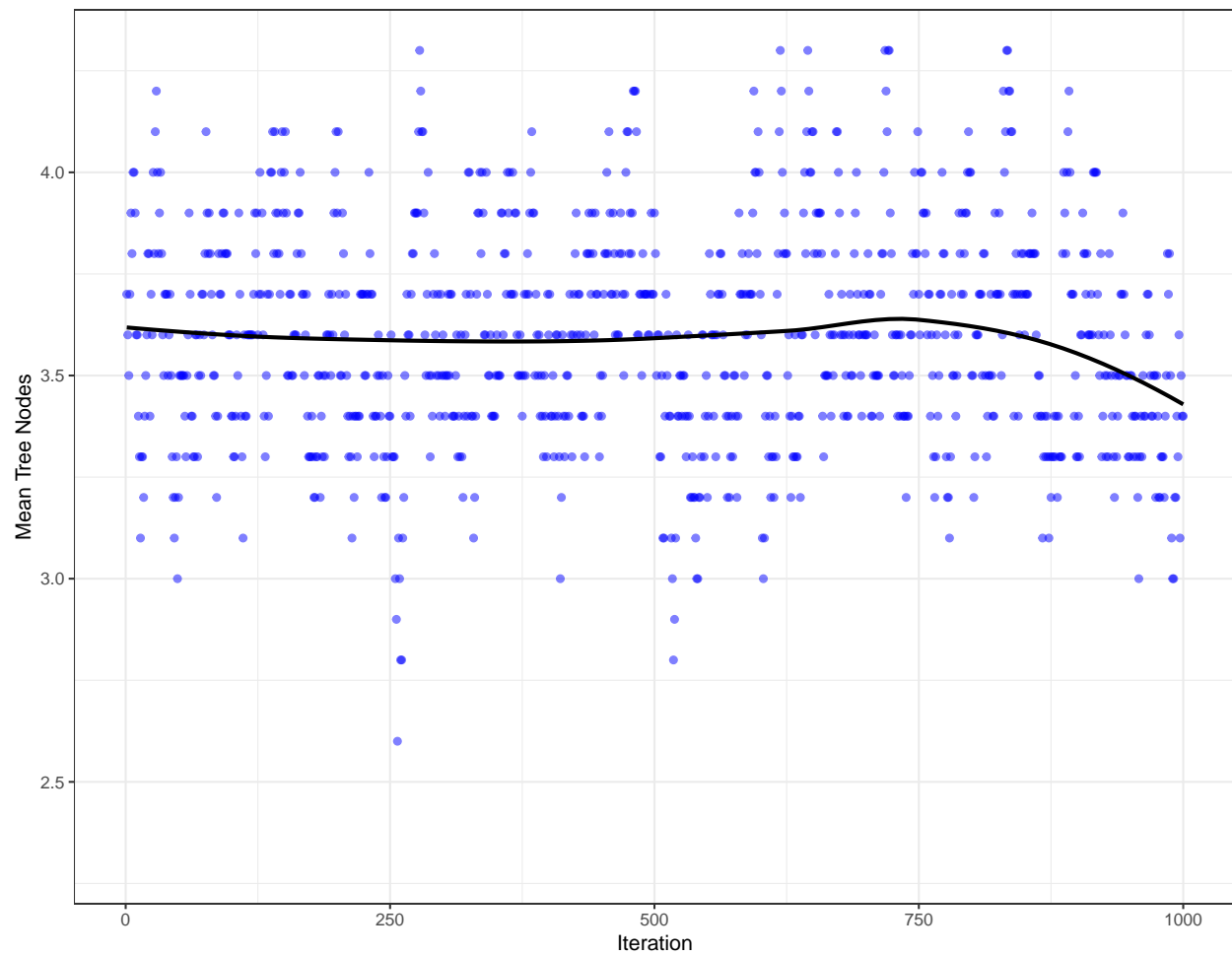


Figure 10:

```
# split density (Need to change both2)
splitDensity(treeData = bmDF, data = iris2, display = 'both2')
splitDensity(dbDF, data = iris2, display = 'both2')
```

