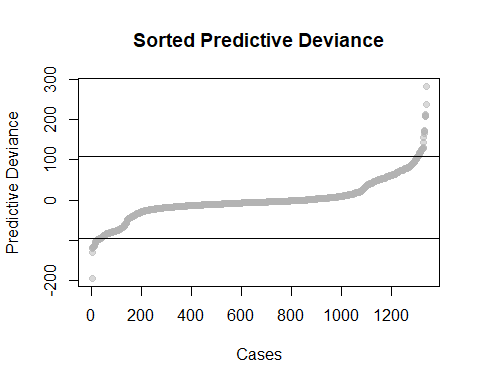
# This is an implementation of predictive and predictive-deviance oriented segmentation and is intended to be executed in RStudio  
  
# Data Preparation  
set.seed(3498348)  
insurance <- read.csv("data/insurance.csv")  
insurance <- insurance[sample(nrow(insurance)),]  
insurance <- insurance[sample(nrow(insurance)),]  
insurance$sex <- as.factor(insurance$sex)  
insurance$smoker <- as.factor(insurance$smoker)  
insurance$region <- as.factor(insurance$region)  
  
# Find predictive deviants using LOOCV  
pd\_fold <- function(i, nfold, model, outcome, dataset, fitted\_scores){  
 folds <- cut(1:nrow(dataset), breaks=nfold, labels = FALSE)  
 test\_indices <- which(folds == i, arr.ind = TRUE)  
 test <- dataset[test\_indices, ]  
 train <- dataset[-test\_indices, ]  
 trained <- lm(model, data = train)  
 predictions <- predict(trained, test)  
  
 fitted\_scores[test\_indices] - predictions # PD  
}  
  
  
kfold\_pd <- function(model, outcome, dataset, nfold=NULL) {  
 if (is.null(nfold)) {  
 nfold <- nrow(dataset)}  
  
 fitted\_model <- lm(model, data = dataset)  
 hetero<-lmtest::bptest(fitted\_model)  
 fitted\_scores <- predict(fitted\_model, dataset)  
 unlist(sapply(1:nfold, pd\_fold, nfold=nfold,   
 model = model, outcome=outcome, dataset = dataset,   
 fitted\_scores = fitted\_scores))  
}  
  
pd\_loocv <- kfold\_pd(charges ~ ., outcome='charges', dataset = insurance)  
  
# PD Plot  
sorted\_pd <- sort(pd\_loocv)  
pd\_95 <- quantile(sorted\_pd, probs = c(0.025, 0.975))  
plot(sorted\_pd,col=rgb(0.7, 0.7, 0.7, 0.5), pch=19,xlab = "Cases", ylab = "Predictive Deviance",main = "Sorted Predictive Deviance")  
abline(h=pd\_95)



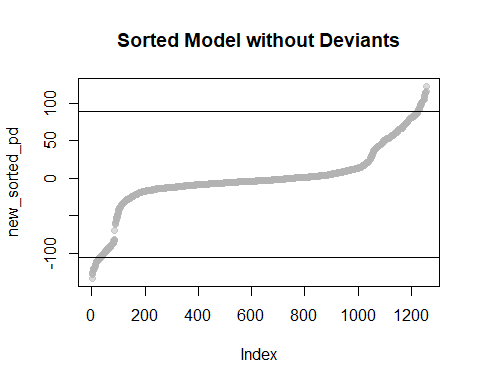
# Deviance Tree  
library(rpart)  
library(rattle)

## Loading required package: tibble

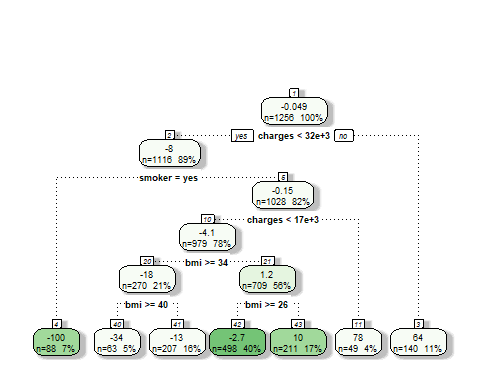
## Loading required package: bitops

## Rattle: A free graphical interface for data science with R.  
## Version 5.5.1 Copyright (c) 2006-2021 Togaware Pty Ltd.  
## Type 'rattle()' to shake, rattle, and roll your data.

library(rpart.plot)  
library(data.tree)  
  
# Find deviant nodes  
insurance\_rpart <- rpart(pd\_loocv~., data = insurance, minsplit = 2, minbucket = 1, cp = 0)  
  
frame\_deviants <- (insurance\_rpart$frame$yval <= pd\_95["2.5%"]) | (insurance\_rpart$frame$yval >= pd\_95["97.5%"])  
all\_deviants <- insurance\_rpart$frame[frame\_deviants, ]  
deviant\_nodes <- all\_deviants[all\_deviants$var != "<leaf>", ]  
head(deviant\_nodes)  
  
# Use semcoa package to find deviant groups  
library(semcoa)  
library(purrr)  
insurance\_pd <- cbind(insurance, PD = pd\_loocv)  
predictions <- list(pd\_data = insurance\_pd, PD = pd\_loocv)  
  
dtree <- deviance\_tree(predictions)  
grouped\_deviants <- unname(unlist(dtree$deviant\_groups))  
unique\_deviants <- dtree$unique\_deviants  
all\_deviants <- c(grouped\_deviants, unique\_deviants)  
  
# Find total number of cases for a set of groups  
no\_deviant\_group\_cases <- function(groups,original\_rpart\_dataset,deviance\_tree\_object){  
  
 groups<-toupper(groups)  
 no\_by\_group<-sapply(groups, function(x){group\_nodes\_index<-deviance\_tree\_object$deviant\_groups[x]  
 group\_cases<-sapply(group\_nodes\_index,function(i) original\_rpart\_dataset$frame[i, ]$n)  
 reduce(group\_cases, sum)})   
   
 print(no\_by\_group)  
 cat("Total number of cases for this set of groups is:", reduce(no\_by\_group,sum))  
}  
  
no\_deviant\_group\_cases(c("A", "B", "C", "D", "E"),insurance\_rpart, dtree)  
  
  
# Model less deviants  
new\_insurance <- insurance[-all\_deviants, ]  
new\_pd\_loocv <- kfold\_pd(charges ~ ., outcome = 'charges', dataset = new\_insurance)  
  
new\_sorted\_pd <- sort(new\_pd\_loocv)  
new\_pd\_95 <- quantile(new\_sorted\_pd, probs = c(0.025, 0.975))  
plot(new\_sorted\_pd, col = rgb(0.7, 0.7, 0.7, 0.5), pch = 19,main = "Sorted Model without Deviants")  
abline(h = new\_pd\_95)



library(rpart)  
new\_insurance\_rpart <- rpart(new\_pd\_loocv~., data = new\_insurance, minsplit = 2, minbucket = 1,cp = 0)  
node\_leaves <- semcoa:::tree\_node\_cases(new\_insurance\_rpart) # WARNING: long-running  
  
# Get average PD of all cases under a node  
node\_pd <- function(node\_name, node\_leaves, cases\_pd){  
 mean(na.omit(new\_pd\_loocv[as.character(na.omit(node\_leaves[[node\_name]]))]))  
}  
  
# Find the number of cases that belong to a node  
cases\_per\_node<-function(x){  
 z<-c()  
 number\_of\_cases<-sapply(1:length(x),function(y){  
 z[[y]]<<-data.frame("NodeIndex" = match(x[y], node\_leaves), "NodeName" = names(x[y]), "TotalNodeCases" = length(x[[y]]))} )  
 return(z)  
}  
  
cases\_per\_node(c(node\_leaves[900], node\_leaves[1000]))  
  
# Run regression on cases(subset) of a node  
library(broom)  
subsets\_models <- sapply(1:length(node\_leaves), function(x){  
 if(length(node\_leaves[[x]]) > 6){  
 node\_subset\_elements <- insurance[node\_leaves[[x]],]  
 node\_subset\_elements\_number <- sapply(lapply(node\_subset\_elements, unique), length)  
 node\_model <- lm(charges~., data = node\_subset\_elements[, node\_subset\_elements\_number>1])  
 model\_results <- data.frame(names(node\_leaves[x]), t(node\_model$coefficients), summary(node\_model)$r.squared)  
 print(model\_results)  
}})  
  
node\_lm\_results <- data.table::rbindlist(subsets\_models, fill = TRUE)  
colnames(node\_lm\_results)[1] <- "nodename"  
colnames(node\_lm\_results)[2] <- "intercept"  
colnames(node\_lm\_results)[11]<- "r-squared"  
write.csv(node\_lm\_results,"node\_lm\_results.csv", row.names = FALSE)  
  
# Pruned insurance tree with PD  
simple\_new\_insurance\_rpart <- rpart(new\_pd\_loocv~., data = new\_insurance)  
fancyRpartPlot(simple\_new\_insurance\_rpart, caption= NULL)



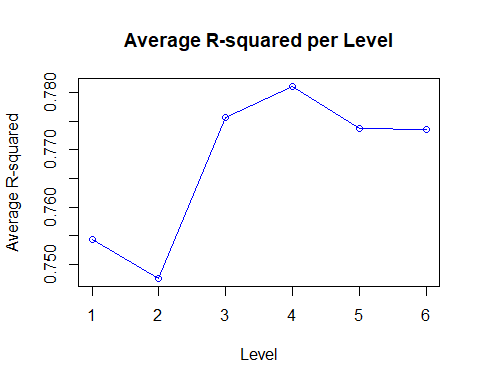
simple\_node\_leaves <- semcoa:::tree\_node\_cases(simple\_new\_insurance\_rpart)  
  
library(data.tree)  
simple\_new\_datatree <- as.Node(simple\_new\_insurance\_rpart)  
  
# Find the nodes that belong to a level  
library(dplyr)

##   
## Attaching package: 'dplyr'

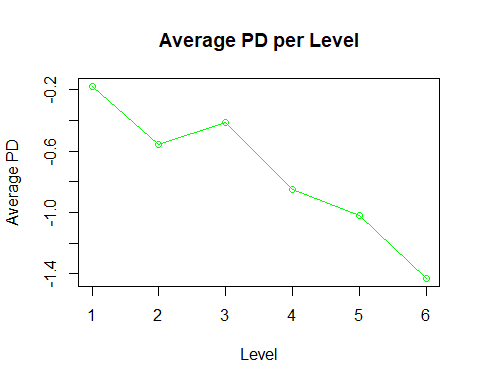
## The following objects are masked from 'package:stats':  
##   
## filter, lag

## The following objects are masked from 'package:base':  
##   
## intersect, setdiff, setequal, union

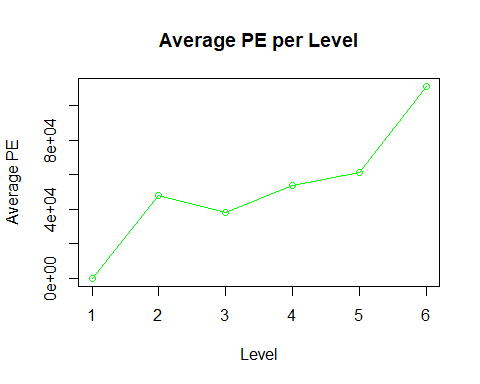
level\_combinations <- function(data\_tree\_object){   
 library(stringr)  
 return\_level<-c()  
   
 # Convert the data.tree object into a dataframe and sort the dataframe  
 to\_dataframe <- ToDataFrameTree(data\_tree\_object,'name', 'level', 'rpart.id', 'isLeaf', children = function(x){sapply(x$children, function(child)child$rpart.id)})  
 sorted\_levels <- to\_dataframe[order(to\_dataframe$level), ]  
   
 total\_level\_children <- list()  
 leaves <- NULL  
   
 # Loop through levels of a tree  
 for(i in 1:(max(sorted\_levels$level) - 1)){  
   
 # Current level's children  
 level\_children <- na.omit(sorted\_levels[sorted\_levels$level == i, ]$children)  
 level\_children <- str\_trim(unlist(strsplit(level\_children,split = ",")))  
   
 # Array of all children in a tree  
 total\_level\_children <- append(total\_level\_children,list(level\_children))  
   
   
 # Ensure that there is a previous level  
 if(length(total\_level\_children) >= 1)  
 {  
 # Loop through previous level  
 sapply(total\_level\_children[i-1],function(x){  
   
 # Loop through children in previous level  
 sapply(x, function(y){  
   
 # Loop through sorted levels to get rpart.id of leaves  
 for (j in 1:nrow(sorted\_levels)){  
 if(sorted\_levels[j, ]$rpart.id == y && sorted\_levels[j, ]$isLeaf == TRUE){  
 leaves <<- c(leaves, sorted\_levels[j, ]$rpart.id)}  
 }  
 })})}   
   
 return\_level[[i+1]] <- noquote(c(level\_children, leaves))  
 return\_level[[1]] <- sorted\_levels[1, ]$rpart.id  
 }  
   
 return(return\_level)  
}  
  
all\_levels <- level\_combinations(simple\_new\_datatree)  
  
# Find average r-squared per level  
levels\_rsquared <- sapply(all\_levels, function(x){ sapply(x, function(y){node\_lm\_results[node\_lm\_results$nodename == y, ]$`r-squared`})} )  
average\_levels\_rsquared <- sapply(levels\_rsquared, function(z)(reduce(z, sum))/length(z))  
plot(average\_levels\_rsquared, type = "o", xlab = "Level", ylab = "Average R-squared", col = "blue", main = "Average R-squared per Level")



# Find average PD per level  
levels\_pd <- sapply(all\_levels, function(x){ sapply(x, function(y) {node\_pd(y, node\_leaves, new\_pd\_loocv)})} )  
average\_levels\_pd <- sapply(levels\_pd, mean)  
plot(average\_levels\_pd, type = "o", xlab = "Level", ylab = "Average PD", col = "green",main = "Average PD per Level")  
abline(h = 0)



# Find out-of-sample predictive error per node  
node\_pe <- function(node\_name, node\_leaves, cases\_pe){  
 mean(na.omit(new\_pe\_loocv[as.character(na.omit(node\_leaves[[node\_name]]))]))  
}  
  
# Find predictive error using LOOCV  
pe\_fold <- function(i, nfold, model, outcome, dataset){  
 folds <- cut(1:nrow(dataset), breaks=nfold, labels = FALSE)  
 test\_indices <- which(folds == i, arr.ind = TRUE)  
 test <- dataset[test\_indices, ]  
 train <- dataset[-test\_indices, ]  
 trained <- lm(model, data = train)  
 predicted\_outcome <- predict(trained, test)  
  
 pe <- test[, outcome] - predicted\_outcome  
}  
  
kfold\_pe <- function(model, outcome, dataset, nfold = NULL) {  
 if (is.null(nfold)) {  
 nfold <- nrow(dataset)  
 }  
 unlist(sapply(1:nfold, pe\_fold, nfold = nfold,   
 model = model, outcome =outcome, dataset = dataset))  
}  
  
new\_pe\_loocv <- kfold\_pe(charges ~ ., outcome = 'charges', dataset = new\_insurance)  
  
levels\_pe <- sapply(all\_levels, function(x){ sapply(x, function(y) {node\_pe(y, node\_leaves, new\_pe\_loocv)})} )  
average\_levels\_pe <- sapply(levels\_pe, function (x) mean(x\*x))  
  
plot(average\_levels\_pe, type = "o", xlab = "Level", ylab = "Average PE", col = "green", main="Average PE per Level")



# Run the model on predictive segments  
PD\_segments\_model <- function(node\_names,dataset,model){  
 node\_cases <- sapply(node\_names, function (element)lm(model, data = dataset[node\_leaves[[element]], ]))  
 node\_cases  
}  
  
PD\_segments\_model(c("3", "4", "5"), dataset = insurance, model = charges~.)  
  
lm(charges~.,data = insurance)  
lm(charges~.,data = insurance[node\_leaves[["3"]], ])  
lm(charges~.,data = insurance[node\_leaves[["4"]], ])  
lm(charges~.,data = insurance[node\_leaves[["5"]], ])