Image3C\_ProcessFCS

# Process FCS files

This script and process removes highly correlated features, removes outlier samples, transforms fluorescent intensity parameters, and normalizes and aligns DNA intensity data

#defining some functions required  
  
#scale & center matrices of exprs in each flow frame in the set  
scaleSet <- function(fs){  
 fsApply(fs, FUN = function(fr){  
 mat <- exprs(fr)  
 mat <- scale(mat) #scale draq5 intensity parm  
 exprs(fr) <- mat  
 exprs(fr)  
   
 fr  
 })  
}  
  
#scale & center DNA intensity parm  
#takes flowset and channel number  
scaleDNA <- function(fs, ch){  
 fsApply(fs, FUN = function(fr, y){  
 mat <- exprs(fr)  
 mat[,ch] <- scale(mat[,ch]) #scale draq5 intensity parm  
 exprs(fr) <- mat  
 exprs(fr)  
   
 return(fr)  
 })  
}  
  
#function to return list of feature means for each sample in a flowset  
extractMeans <- function(fs){  
   
 fsApply(fs, FUN = function(fr){  
 myrownames <- colnames(fs[[1]])  
   
 frname <- fr@description$GUID  
 frname <- strsplit(frname, "\\.") #string split on "."  
 frname <- frname[[1]][[1]] #get out 1st element from strsplit output  
 mat1 <- exprs(fr)  
 means <- colMeans(mat1)  
 output1 <- data.frame(cbind(frname, means))  
 output1$feature <- rownames(output1)  
 rownames(output1) <- seq(length=nrow(output1))  
 colnames(output1) <- c("Sample","Means","Feature")  
 return(output1)  
 })  
}  
  
  
#function from DaMiRseq package to remove features with high correlation to one another  
DaMiR.FReduct <- function(data,  
 th.corr=0.85,  
 type=c("spearman","pearson")){  
   
 # check arguments  
 if (missing(data)) stop("'data' argument must be provided")  
 if (missing(type)){  
 type <- type[1]  
 }  
   
 # check the type of argument  
 if(!(is.numeric(th.corr)))  
 stop("'th.corr' must be numeric")  
 if(!(is.data.frame(data)))  
 stop("'data' must be a data.frame")  
   
 # check the presence of NA or Inf  
 if (any(is.na(data)))  
 stop("NA values are not allowed in the 'data' matrix")  
 if (any(is.infinite(as.matrix(data))))  
 stop("Inf values are not allowed in the 'data' matrix")  
   
 # specific checks  
 if (th.corr >1 | th.corr < 0)  
 stop("'th.corr must be between 0 and 1")  
 if (all((as.matrix(data) %%1) == 0))  
 warning("It seems that you are using raw counts!  
 This function works with normalized data")  
   
 features<-dim(data)[2]  
   
 # remove redundancy  
 if(type == "spearman"){  
 cormatrix <- abs(rcorr(as.matrix(data), type='spearman')$r)  
 } else if (type == "pearson"){  
 cormatrix <- abs(rcorr(as.matrix(data), type='pearson')$r)  
 } else {  
 stop("Please set 'spearman or 'pearson' as correlation type.")  
 }  
   
 index\_geneHighCorr<- findCorrelation(cormatrix, cutoff = th.corr)  
 data\_reduced<-data[, -index\_geneHighCorr, drop=FALSE]  
   
 cat(features-dim(data\_reduced)[2],  
 "Highly correlated features have been discarded for classification.",  
 "\n",  
 dim(data\_reduced)[2],  
 "Features remained.",  
 "\n")  
 return(data\_reduced)  
}  
  
#drop specified colnames  
dropSet <- function(fs, toremove){  
 fsApply(fs, FUN = function(fr){  
 exprs(fr) <- exprs(fr)[,-which(colnames(exprs(fr)) %in% toremove)]  
 fr  
 })  
}

## Process FCS files obtained from IDEAS

Let’s load all the packages that we need:

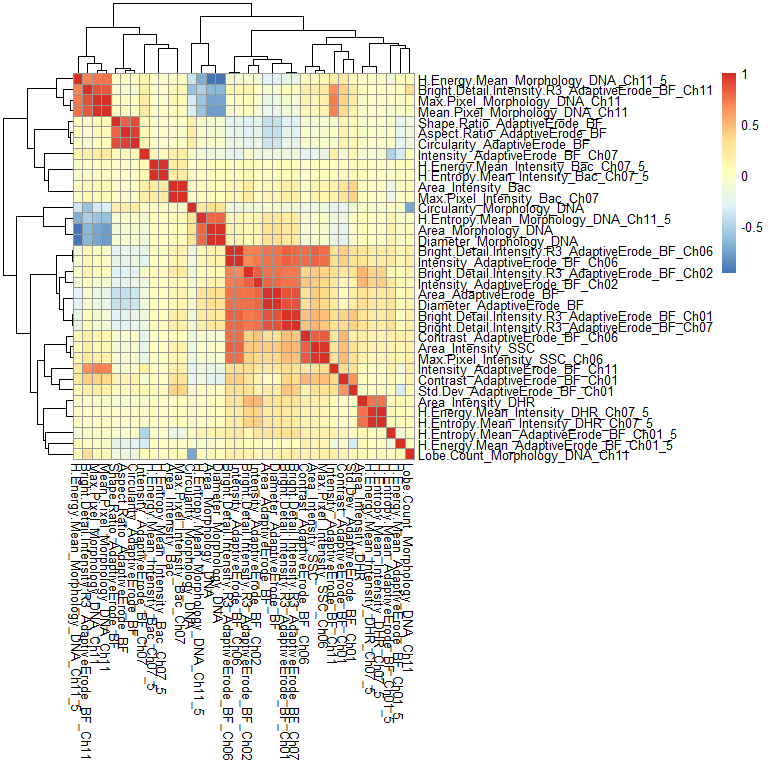
library(flowCore)  
library(flowStats)  
library(ggcyto)  
library(ggridges)  
library(stringr)  
library(Hmisc)  
library(caret)  
library(pheatmap)  
library(reshape2)  
library(data.table)  
library(RColorBrewer)  
library(knitr)  
library(png)

Now we need to read the files, pull out some data such as headers and channels and then we plot histograms for all the parameters

#set your working directory containing a copy of all .fcs files  
#setwd("your folder path here")  
setwd("~/R\_projects/Image3C Markdown/Process FCS")  
  
#read all fcs files in current dir to flowset  
myflowset <- read.flowSet(pattern = ".fcs", path = ".", alter.names = TRUE, transformation = FALSE, emptyValue = FALSE)   
#get all channel names  
mycolnames <- as.character(colnames(exprs(myflowset[[1]])))   
#plot histograms for all parameters from a sample file  
#autoplot(myflowset[[1]])

Next we generate clustering heatmap of feature correlation from a frame in the set. For this step a fully stained and representative sample should be used (not a single color control)

#pull out a single flow frame  
myframe <- myflowset[[1]]   
#extract matrix  
mat <- exprs(myframe)   
#get correlation matrix  
cor\_mat <- cor(mat, method = "spearman")   
#plot it using clustering heatmaps  
pheatmap(cor\_mat)



## Remove redundant features based on correlation

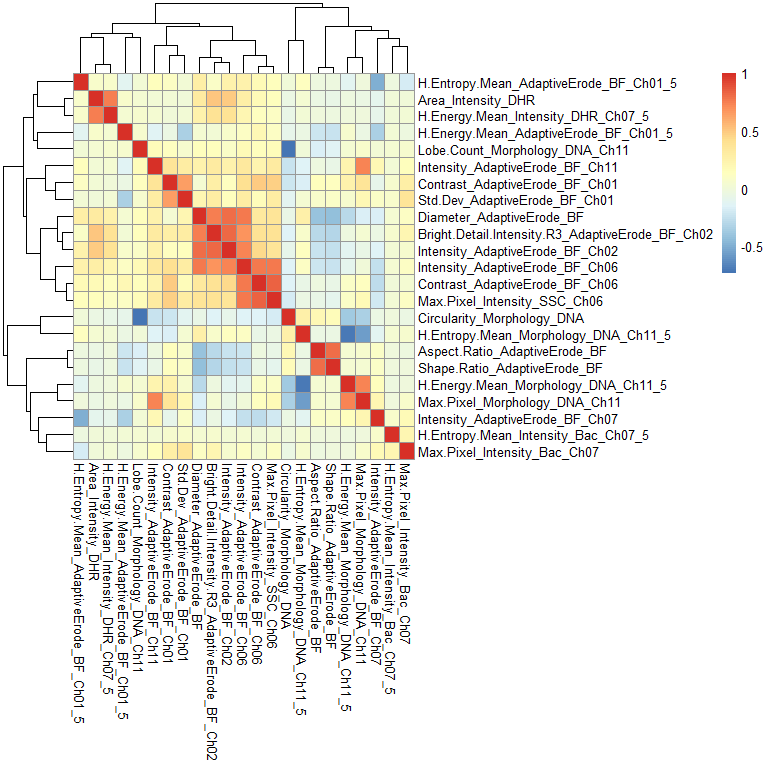
#convert data from first sample to data frame object  
myfrm <- data.frame(exprs(myframe))   
#removes features with Cor values > 0.85  
frmTrim <- DaMiR.FReduct(myfrm, th.corr = 0.85)

## 13 Highly correlated features have been discarded for classification.   
## 23 Features remained.

#get names of remaining features  
colstrimmed <- colnames(frmTrim)   
#set operation to find difference with total  
toremove <- setdiff(mycolnames, colstrimmed)   
  
myflowset <- dropSet(myflowset, toremove)

Now we generate new clustering heatmap again to check how it looks after removal

#pull out a single flow frame  
myframe <- myflowset[[1]]   
#extract matrix  
mat <- exprs(myframe)   
#get correlation matrix  
cor\_mat <- cor(mat, method = "spearman")   
#plot it using clustering heatmaps  
pheatmap(cor\_mat)



Now calculate sample correlations by feature means

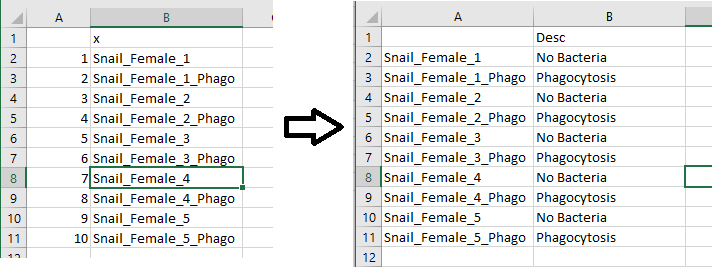
#get a list of features means, one list item per file  
myMeans <- extractMeans(myflowset)   
#combine to one big table  
myMeans <- rbindlist(myMeans)   
#recast to wide form  
myMeans <- dcast(myMeans, Sample ~ Feature, value.var = "Means")   
#rename rownames  
rownames(myMeans) <- myMeans$Sample   
#remove redundant col  
myMeans[,1] <- NULL   
#convert all to numeric  
myMeans[] <- lapply(myMeans, function(x) {  
 as.numeric(as.character(x))  
})  
  
#convert to matrix  
meansMat <- as.matrix(myMeans)   
rownames(meansMat) <- rownames(myMeans)  
#convert to matrix  
meansMat[!is.finite(meansMat)]

## numeric(0)

#save table, will manipulate to produce an annotation dataframe, see below  
write.csv(rownames(meansMat), file="NamesToGroup.csv")

### **Manual step - edit “NamesToGroup.csv” to match this format**

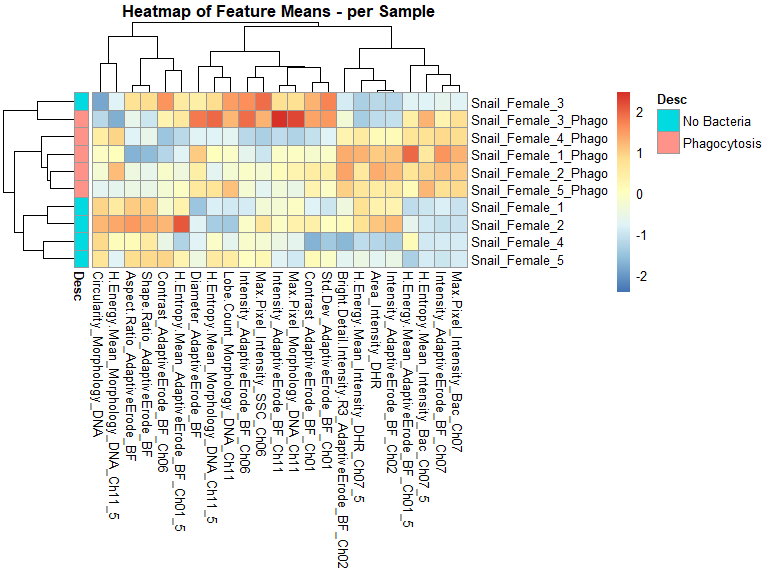
Use excel or your favorite spreadsheet application to alter the structure of NamesToGroup.csv as shown below and save the file as “RowLabels.csv”.



rowAnnot <- read.csv(file="RowLabels.csv")  
#put rownames from means table  
rownames(rowAnnot) <- rownames(meansMat)   
#remove redundant column  
rowAnnot[,1] <- NULL   
#check that sample names match, should return NULL  
setdiff(myflowset@phenoData@data$names,rowAnnot$X)

## NULL

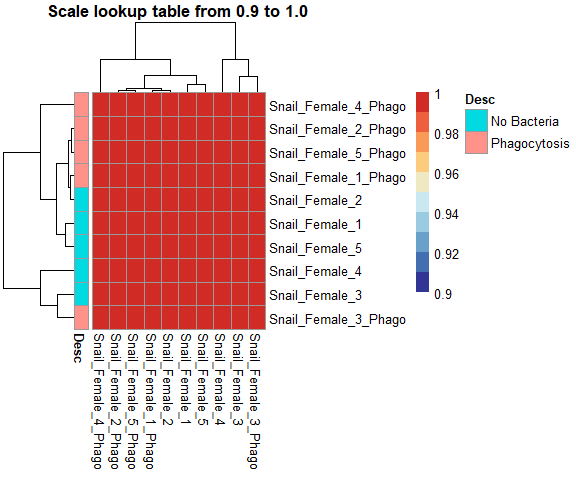
rowAnnotBak <- rowAnnot  
#make clustering heatmap of feature means per sample  
pheatmap(meansMat, scale = "column", annotation\_row = rowAnnot, main = "Heatmap of Feature Means - per Sample")



## Plot heatmap of correlation among samples, by feature means

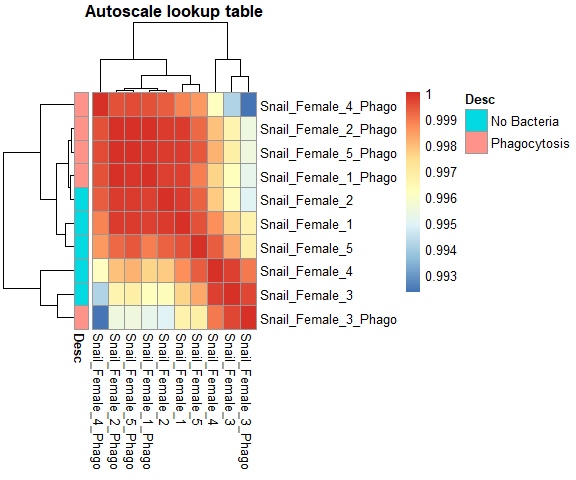
make correlation heatmap with color range for 0.9 to 1

breaksList <- seq(0.9,1, by = 0.01)  
#heatmap of correlation   
pheatmap(cor(t(meansMat)), annotation\_row = rowAnnot,  
 color = colorRampPalette(rev(brewer.pal(n = 10, name = "RdYlBu")))(length(breaksList)),  
 breaks = breaksList, main = "Scale lookup table from 0.9 to 1.0")



make correlation heatmap with default settings for color scale

pheatmap(cor(t(meansMat)), main = "Autoscale lookup table", annotation\_row = rowAnnot)

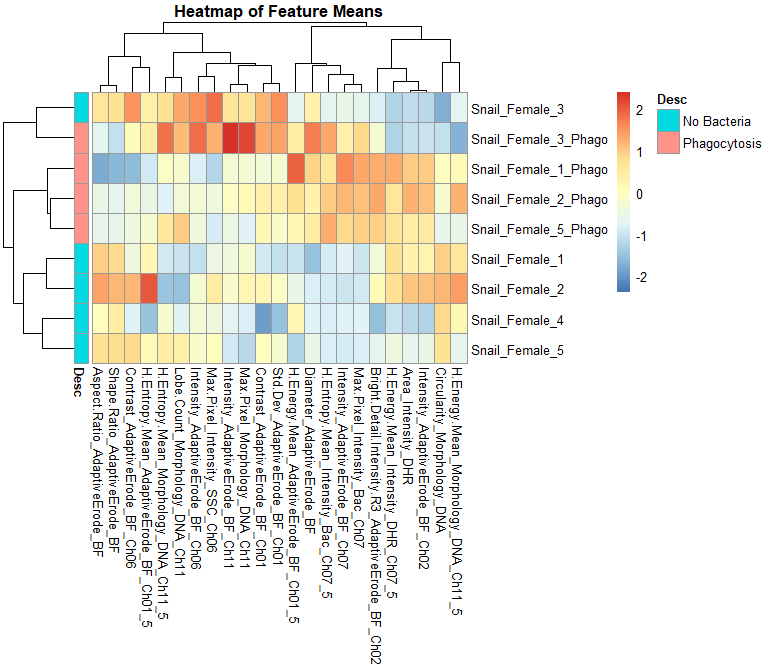


## Remove outlier samples

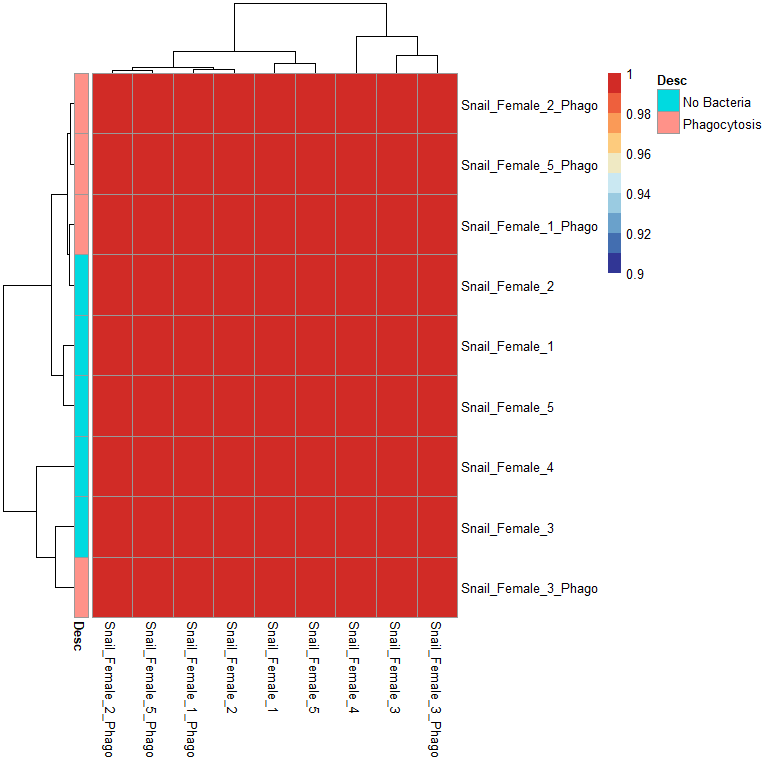
Can skip if correlation is high among all or only low between different treatment groups

meansMatBak <- meansMat  
#correlation on transposted means matrix  
meansCor <- cor(t(meansMat))   
#list of samples with mean cor >0.85  
tokeep <- findCorrelation(meansCor, cutoff = 0.85, names = FALSE)   
tokeep <- sort(tokeep)  
#same but return names not index  
tokeepnames <- findCorrelation(meansCor, cutoff = 0.85, names = TRUE)   
tokeepnames <- sort(tokeepnames)  
#subset flowset to remove low cor samples  
myflowsetbak <- myflowset  
myflowset <- myflowset[c(tokeep)]   
#remove dropped samples from annotation data  
rowAnnot <- rowAnnot[tokeepnames,]   
rowAnnot <- as.data.frame(rowAnnot)  
colnames(rowAnnot) <- "Desc"  
rownames(rowAnnot) <- myflowset@phenoData@data$name  
  
#sample correlations by feature means, do again without outliers  
#get a list of features means, one list item per file  
myMeans <- extractMeans(myflowset)   
#combine to one big table  
myMeans <- rbindlist(myMeans)   
#recast to wide form  
myMeans <- dcast(myMeans, Sample ~ Feature, value.var = "Means")   
#rename rownames  
rownames(myMeans) <- myMeans$Sample   
#remove redundant col  
myMeans[,1] <- NULL   
#convert all to numeric  
myMeans[] <- lapply(myMeans, function(x) {  
 as.numeric(as.character(x))  
})  
  
#convert to matrix  
meansMat <- as.matrix(myMeans)   
rownames(meansMat) <- rownames(myMeans)  
rownames(rowAnnot) <- rownames(meansMat)  
#check for infinite values#  
#meansMat[!is.finite(meansMat)]   
#check that sample names match, should return NULL#  
#setdiff(myflowset@phenoData@data$names,rownames(rowAnnot))

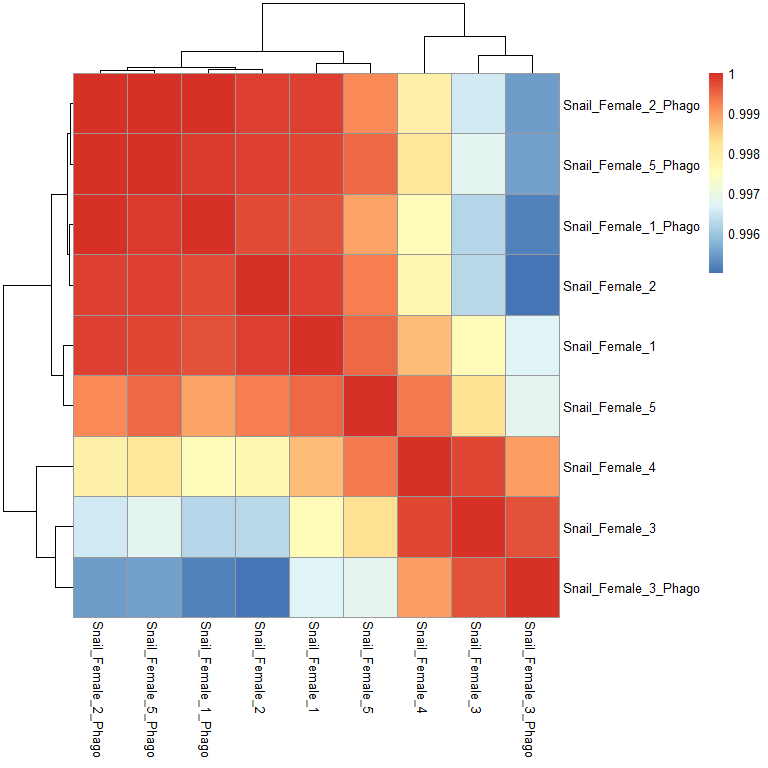
#make clustering heatmap of feature means per sample  
pheatmap(meansMat, scale = "column", annotation\_row = rowAnnot, main = "Heatmap of Feature Means")



breaksList <- seq(0.9,1, by = 0.01)  
#heatmap of correlation  
pheatmap(cor(t(meansMat)), annotation\_row = rowAnnot,  
 color = colorRampPalette(rev(brewer.pal(n = 10, name = "RdYlBu")))(length(breaksList)),  
 breaks = breaksList)



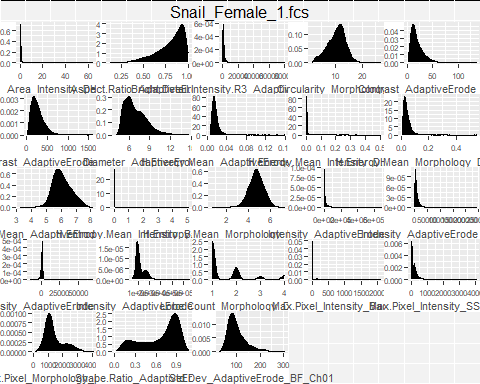
pheatmap(cor(t(meansMat)), method = "spearman")



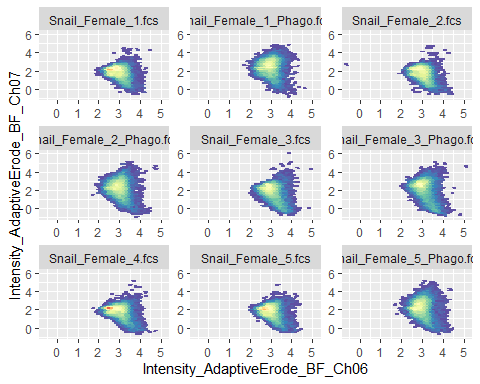
## Transform fluorescent parameters to allow gaussNorm normalization

From “mycolnames” below, identify the index column number of any fluorescence intensity parameters. Make note of these column numbers. On line 319 you’ll put those column numbers into a variable used to specify which data gets logicle transformed.

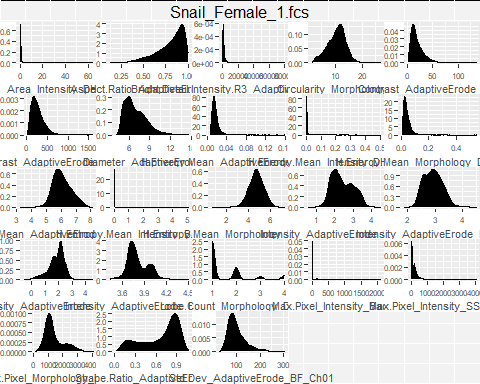
#get new list of parameter names  
mycolnames <- colnames(myflowset[[1]])   
#plot all histograms  
autoplot(myflowset[[1]])



#which parameters to transform, usually want to do all 'intensity' parameters for fluorescence channels  
ChnlsToTrans <- mycolnames[c(14:17)]   
#estimate logicle transform from data  
translist <- estimateLogicle(myflowset[[1]], ChnlsToTrans)   
#apply logicle transform to flowset  
myflowsetTrans <- transform(myflowset, translist)   
#have a look at some data  
autoplot(myflowsetTrans, mycolnames[15], mycolnames[16]) + geom\_hex(bins=40)

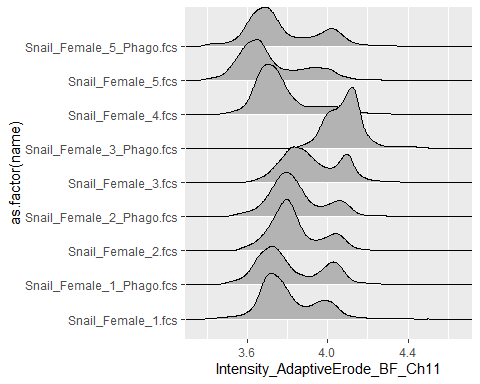


#plot histograms again after transforming data  
autoplot(myflowsetTrans[[1]])



#plot stacked histograms   
p <- ggcyto(myflowsetTrans, aes(x = 'Intensity\_AdaptiveErode\_BF\_Ch11'))  
p + geom\_density\_ridges(aes(y = as.factor(name))) + facet\_null()

## Picking joint bandwidth of 0.0199



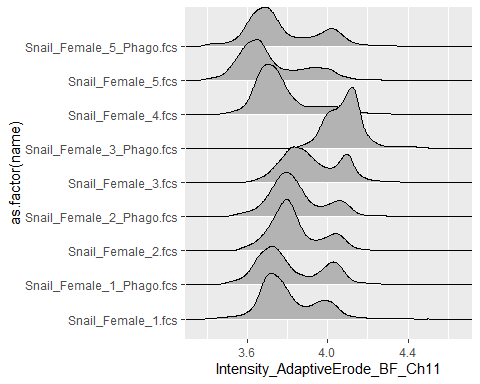
## Scale and center flow frames in set for DNA intensity cols

gaussNorm function works better on scaled data.

**For this step you must specify the channel number to use in the code below**

mycolnames <- colnames(myflowset[[1]])  
  
#Check DNA content histograms before scaling data  
p <- ggcyto(myflowsetTrans, aes(x = 'Intensity\_AdaptiveErode\_BF\_Ch11'))  
p + geom\_density\_ridges(aes(y = as.factor(name))) + facet\_null() #+ xlim(c(-3,3))

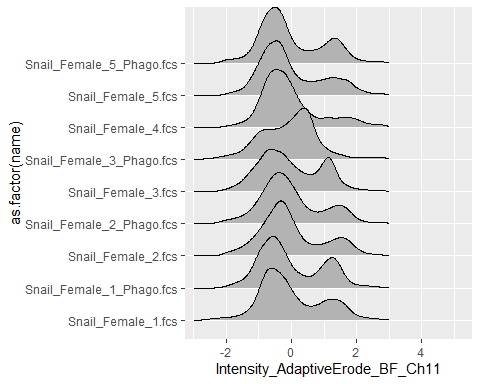
## Picking joint bandwidth of 0.0199



#Make sure scaleDNA function  
#pass in transformed flowset and DNA intensity column number  
flowsetScaled <- scaleDNA(myflowsetTrans, 17)   
  
#In this case, flowframe 12 has abnormal draq5 staining, uncomment to remove it  
# tokeep <- myflowset@phenoData@data$name  
# tokeep <- tokeep[c(1:11,13:20)]  
  
#Remove that sample (keep all but index 12 in list of names)  
# flowsetScaled <- flowsetScaled[c(tokeep)]   
  
#plot stacked histograms of transformed, original data  
  
# p <- ggcyto(flowsetScaled, aes(x = 'Intensity\_AdaptiveErode\_BF\_Ch02'))  
# p + geom\_density\_ridges(aes(y = as.factor(name))) + facet\_null()  
#   
# p2 <- ggcyto(flowsetScaled, aes(x = 'Intensity\_AdaptiveErode\_BF\_Ch07'))  
# p2 + geom\_density\_ridges(aes(y = as.factor(name))) + facet\_null()  
  
p3 <- ggcyto(flowsetScaled, aes(x = 'Intensity\_AdaptiveErode\_BF\_Ch11'))  
p3 + geom\_density\_ridges(aes(y = as.factor(name))) + facet\_null() + xlim(c(-3,3))

## Picking joint bandwidth of 0.132

## Warning: Removed 608 rows containing non-finite values (stat\_density\_ridges).



## Align DNA content or other intensity peaks

Use gaussNorm function from flowStats to align DNA (or other) peaks. Use with caution on data that’s not DNA intensity since it’s possible you will just normalize out your signal. However, with DNA content data, if the samples are of the same ploidy, we can safely assume the 2N and 4N (or other) intensity peaks should be aligned when overlaid in an intensity histogram

### gaussNorm function args

**peak.density.thr** - The peaks with density value less than “peak.density.thr times maximum peak density” are discarded.

**peak.distance.thr** - The sequences of peaks that are located closer than “peak.distance.thr times range of data” are identified. Then for each sequence only one peak (the one with the highest intensity value) is used as a landmark. In other words no two landmarks are located closer than “peak.distance.thr times range of data” to each other.

**max.lms** - A numeric vector of the maximum number of base landmarks to be used for normalizing each channel. If it has only one value that will be used as the maximum number of base landmarks for all the channels.

**Here again you must specify which channel number(s) to run this on**

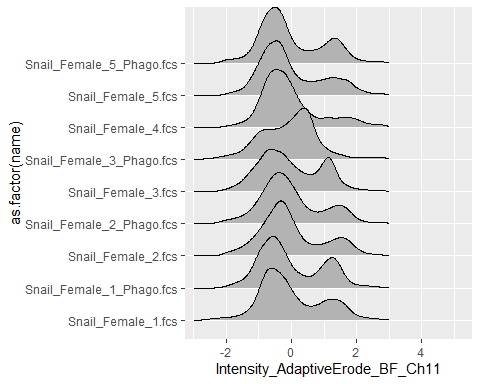
#use gaussNorm function from flowStats to normalize each channel across files  
maxlms <- c(1)  
#Using intensity for ch11 (col 12) here only  
ChnlsToNorm <- mycolnames[18]   
#normalize DNA parameters  
normResult <- gaussNorm(flowsetScaled, channel.names = ChnlsToNorm,  
 max.lms = maxlms,  
 peak.distance.thr = 0.1,  
 peak.density.thr = 0.1)

##   
## Adjusting the distance between landmarks  
## .........

#pull out flowset from returned object  
myflowsetNorm <- normResult$flowset   
  
p2 <- ggcyto(myflowsetNorm, aes(x = 'Intensity\_AdaptiveErode\_BF\_Ch11'))  
p2 + geom\_density\_ridges(aes(y = as.factor(name))) + facet\_null() + xlim(c(-3,3))

## Picking joint bandwidth of 0.132

## Warning: Removed 608 rows containing non-finite values (stat\_density\_ridges).



Optional step: Can do again with different settings on another set of channels if needed. For example, in the previous step do DNA content params with maxlmx of 2. Then, for antibody staining with different data shapes, use another set and repeat the gaussNorm step on another 1 or more channels, again specified in the code.

# maxlms <- c(2)  
# #Intensity DHR  
# ChnlsToNorm <- mycolnames[c(16)]   
# #normalize DNA parameters  
# normResult <- gaussNorm(myflowsetNorm, channel.names = ChnlsToNorm,  
# max.lms = maxlms,  
# peak.distance.thr = 0.2,  
# peak.density.thr = 0.05)   
#   
# #pull out flowset from returned object  
# myflowsetNorm1 <- normResult$flowset   
#   
#   
# p3 <- ggcyto(myflowsetNorm1, aes(x = 'Intensity\_AdaptiveErode\_BF\_Ch02'))  
# p3 + geom\_density\_ridges(aes(y = as.factor(name))) + facet\_null()  
#   
# ##and one more time  
#   
# maxlms <- c(2)  
# #Intensity draq5, set using the assumption that most cells are 2n, ie, EDTA hasn't induced arrest at 4n, but should confirm.  
# ChnlsToNorm <- mycolnames[c(18)]   
# #normalize DNA parameters  
# normResult <- gaussNorm(myflowsetNorm, channel.names = ChnlsToNorm,  
# max.lms = maxlms,  
# peak.distance.thr = 0.3,  
# peak.density.thr = 0.1)   
#   
# #pull out flowset from returned object  
# myflowsetNorm2 <- normResult$flowset   
#   
# p3 <- ggcyto(myflowsetNorm2, aes(x = 'Intensity\_AdaptiveErode\_BF\_Ch11'))  
# p3 + geom\_density\_ridges(aes(y = as.factor(name))) + facet\_null()

## Save new fcs files

These FCS files will be used for clustering in Vortex (x-shift)

#set up file names  
dir <- getwd()  
#get file names from flowset  
mynames <- myflowsetNorm@phenoData@data$name   
#string split on "."  
mynames <- strsplit(mynames, "\\.")   
#get out 1st element from strsplit output  
mynames <- sapply(mynames, function(x) strsplit(x, ":")[[1]][1])   
#paste in new suffixes  
mynames <- paste(mynames, "\_processed.fcs", sep = "")   
  
#save new fcs files  
#make sure you point to the correct flowSet here!  
write.flowSet(myflowsetNorm, dir, filename = mynames)

## [1] "C:/Users/andre/OneDrive/Documents/R\_projects/Image3C Markdown/Process FCS"