Supplementary file R script spectral workflow

H. den Braanker

25-8-2021

Introduction

This script belongs to paper …. We will walk through the described pipeline with two different spectral flow cytometry datasets. Case A will use the dataset described in the paper and available at: . Case B will use part of the files of a spectral flow cytometry dataset .. Files to download from this dataset: … If there are difficulties adapting the script to your own dataset, please do not hestitate to contact us: h.denbraanker@erasmusmc.nl or github.

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")

BiocManager::install("flowCore")
BiocManager::install("flowViz")
BiocManager::install("flowVS")
BiocManager::install("flowAI")
BiocManager::install("flowAI")
BiocManager::install("PeacoQC")
BiocManager::install("CATALYST")
BiocManager::install("SingleCellExperiment")

if(!requireNamespace("devtools", quietly=TRUE))
  install.packages("devtools")

devtools::install_github('saeyslab/CytoNorm')
install.packages("uwot")
install.packages("knitr")
install.packages("xlsx")

set.seed(123)
library(flowCore)
library(flowViz)
library(flowVS)
library(flowAI)
library(PeacoQC)
library(CATALYST)
library(CytoNorm)
library(SingleCellExperiment)
library(uwot)
library(knitr)
library(xlsx)
```

Case A: Spectral flow cytometry dataset A

For case A, we uploaded our FCS files to Flow Repository: FR-FCM-Z4KT.

Manual quality control and pregating of spectral flow cytometry data

Before proceeding to automated analyses, several manual gating steps are required for quality control and cleaning of the data. Manually exclude doublets and dead cells. After, gate your population of interest and export it als FCS files. Save these FCS files in a new folder *FCS files* in your working directory.

Importing spectral flowcytometry data

The exported FCS 3.1 files can be stored in a folder *FCS files* and subsequently imported into the R environment with the FlowCore package. We will apply transformation of the data later, so transformation=FALSE. Furthermore, to prevent truncation of the data, truncate_max_range=FALSE.

```
fcs.dir<- file.path(getwd(), "FCS files")

fcs_data <- read.flowSet(path=fcs.dir, pattern="*.fcs", transformation = FALSE, truncate_max_ra
nge = FALSE) #fcs_data will be a FlowSet object
```

Construct a data frame of your panel:

```
fcs_colname <- colnames(fcs_data)
marker_class <- rep("none", ncol(fcs_data[[1]]))
marker_state <- 36
marker_class[marker_state] <- "state" #markers that indicate "state" of a cell, such as PDL1 mar
ker, or use it to indicate markers you won't use for clustering
marker_type <- c(8:35,38)
marker_class[marker_type] <- "type" # markers that indicate surface markers, such as CD3, CD
4, or markers that you do want to use for clustering
marker_class <- factor(marker_class, levels=c("type", "state", "none"))
antigen <- pData(parameters(fcs_data[[1]]))$desc

panel <- data.frame(fcs_colname, antigen, marker_class, row.names = NULL)
write.xlsx(panel, file="panel_A.xlsx", sheetName="Panel_A")
```

fcs_colname	antigen	marker_class
FSC-A	NA	none
FSC-H	NA	none
SSC-A	NA	none
SSC-B-A	NA	none
SSC-B-H	NA	none
SSC-H	NA	none
FJComp-AF-A	NA	none
FJComp-APC-A	CCR10	type

FJComp-APC-Fire 750-A	ICOS	type
FJComp-APC-Fire 810-A	CD27	type
FJComp-APC-R700-A	LAG-3	type
FJComp-BB515-A	CD25	type
FJComp-BB700-A	CD49b	type
FJComp-BUV395-A	CD8	type
FJComp-BUV496-A	CD4	type
FJComp-BUV563-A	CD161	type
FJComp-BUV615-A	CD14	type
FJComp-BUV661-A	CD69	type
FJComp-BUV737-A	TCRgd	type
FJComp-BUV805-A	CD56	type
FJComp-BV421-A	CCR6	type
FJComp-BV480-A	CCR4	type
FJComp-BV510-A	CD45RA	type
FJComp-BV570-A	HLA-Dr	type
FJComp-BV605-A	Ki-67	type
FJComp-BV650-A	CXCR3	type
FJComp-BV711-A	CD45RO	type
FJComp-BV750-A	CXCR5	type
FJComp-BV785-A	CCR7	type
FJComp-PE-A	FOXP3	type
FJComp-PE-Cy5-A	GITR	type
FJComp-PE-Cy7-A	PD-1	type
FJComp-PE-Dazzle594-A	TIGIT	type
FJComp-PerCP-A	CD19	type
FJComp-PerCP-eFluor 710-A	CD127	type
FJComp-Spark Blue 550-A	CD3	state
FJComp-Zombie UV-A	Zombie UV	none
FJComp-eFluor 660-A	CTLA-4	type
Time	NA	none

Transforming spectral flow cytometry data

First, determine which markers you want to transform. You only have to transform the channels that you used for your experiment.

```
markerstotransform <- panel$fcs_colname[c(8:36,38)]
```

Before calculating cofactors with the FlowVS package, we will downsample our data. Including more cells in finding the optimum cofactor will come with a computational cost. Bartlett's statistics (Y-axis) are computed from density peaks after data is transformed by different cofactors (X-axis). An optimum cofactor is obtained where Bartlett's statistics is minimum (indicated by red circles). The estParamFlowVs function will show you the plots were it based its values on. It is advised to export your cofactor data as an csv or excel file, this is for reproducibility purposes.

Transforming your data with the FlowVS package

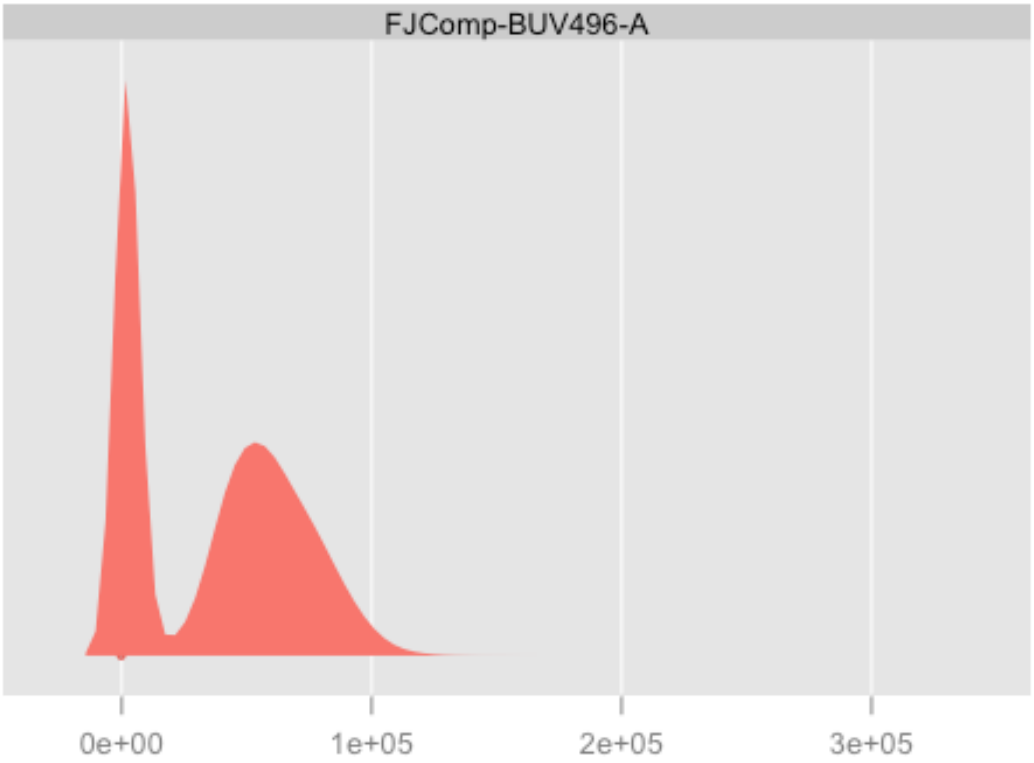
```
Downsampling_FlowSet <- function(x, samplesize , replace=TRUE, prob=NULL){
  if(missing(samplesize))
    samplesize <- min(flowCore::fsApply(x,nrow))
  flowCore::fsApply(x, function(ff){
    i <- sample(nrow(ff), size = samplesize, replace=replace, prob)
    ff[i,]
  })
}

fcs_data_small <- Downsampling_FlowSet(x=fcs_data, samplesize = 2000) #samplesize is the n
umber of cells included, you can include more cells.

cofactors <- estParamFlowVS(fcs_data_small, channels=markerstotransform)
cofactordata <- data.frame(markerstotransform, cofactors)
write.csv(x=cofactordata, file="cofactordata.csv") #csv file
write.xlsx(x=cofactordata, file="cofactordata.xlsx", sheet="cofactordata_A")

fcs_transform <- transFlowVS(fcs_data, channels = markerstotransform, cofactors)
filenames <- sampleNames(fcs_data)
sampleNames(fcs_transform) <- filenames
```

Transforming your data with a fixed cofactor

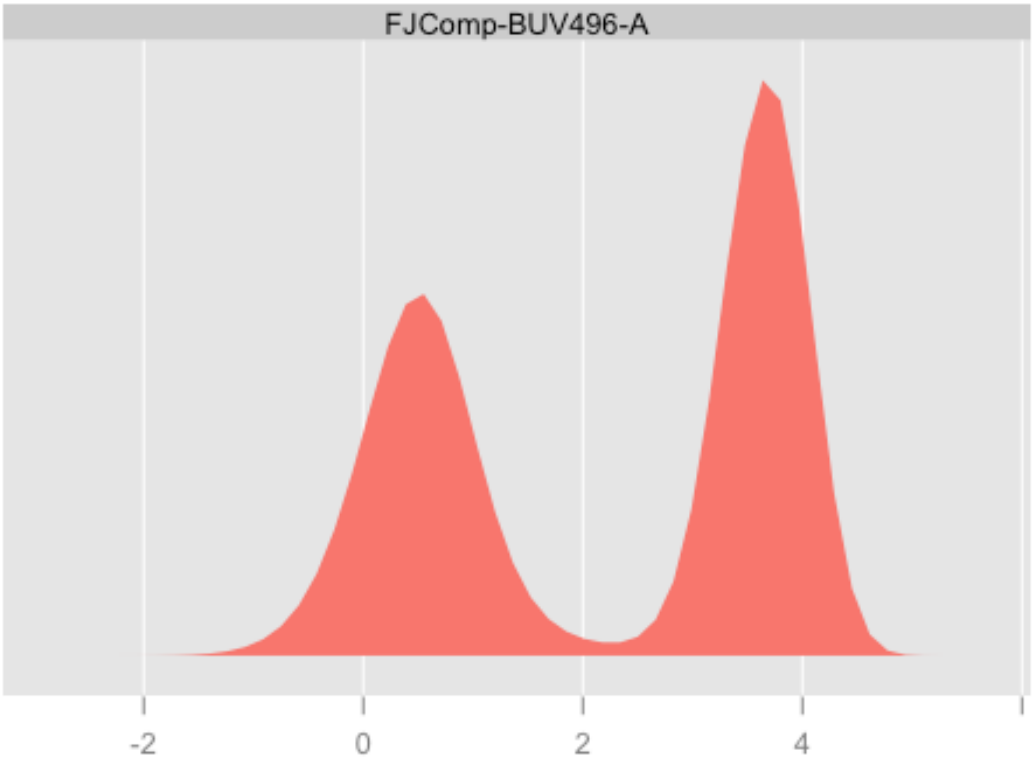
```
cofactor <- 3000
l <- length(markerstotransform)
cofactors<- rep(cofactor, l)

fcs_transform <- transFlowVS(fcs_data, channels = markerstotransform, cofactors)
filenames <- sampleNames(fcs_data)
sampleNames(fcs_transform) <- filenames
```

To evaluate the data transformation, you can visualize density plots of markers with the FlowViz package.

```
densityplot(~`FJComp-BUV496-A`, fcs_data[[1]]) #density plot before transformation, you can re
place `FJComp-BUV496-A` by . to view all markers.
```




```
densityplot(~`FJComp-BUV496-A`, fcs_transform[[1]]) # density plot after transformation
```



Automatic quality control of flow cytometry data

Either flowAI or peacoQC package can be used to clean flow cytometry data. For Case A we demonstrate FlowAI, for Case B peacoQC.

No pre-gated Time gate:

```
fcs_transform <- flow_auto_qc(fcs_transform)
```

Pre-gated Time gate:

```
fcs_transform <- flow_auto_qc(fcs_transform, remove_from = "FS_FM")

outdir <- file.path(getwd(), "Transformed FCS files")
filenames <- paste("tf",fcs_data@phenoData@data$name)
write.flowSet(fcs_transform, outdir = outdir, filename = filenames) #create a new directory with tr
ansformed FCS files
fcs.dir<- file.path(getwd(), "Transformed FCS files")
fcs_transform <-read.flowSet(path=fcs.dir, pattern="*.fcs", transformation = FALSE, truncate_ma
x_range = FALSE)
```

Batch effects

The next step is to correct batch effects, we will use the Cytonorm package to align our different files from different batches. We measured the same samples on different days (technical replicates)

```
fcs.dir<- file.path(getwd(), "Transformed FCS files")

files <- list.files(fcs.dir, pattern = "fcs$")
train_files <- file.path(getwd(),"Transformed FCS files", list.files(fcs.dir, pattern="REU271"))
validation_files <- file.path(getwd(), "Transformed FCS files", list.files(fcs.dir, pattern="REU272"))
fsom <- prepareFlowSOM(train_files, colsToUse = markerstotransform, transformList = NULL, FlowSOM.params = list(xdim=10,ydim=10, nClus=20, scale=FALSE))
```

To check if clustering is appropriate:

```
cvs <- testCV(fsom,cluster_values = c(5,10,15))
```

If the clusters are impacted by batch effects, CV values of >1.5/2 will occur, than you can also choose to put FlowSOM.params to NULL and skip clustering.

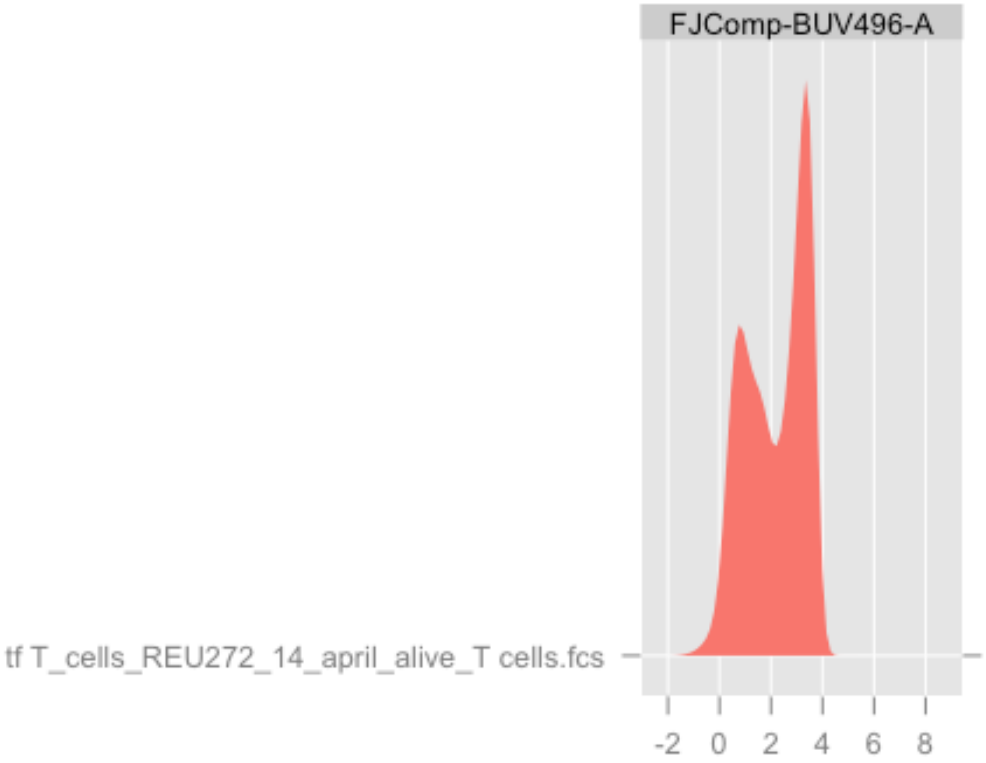
Next, load a metadata file which includes at least a sample_id and a column defining the batches. You can include a column with filenames.

```
md <-read.csv(file="md.csv", header=TRUE, sep=";")
```

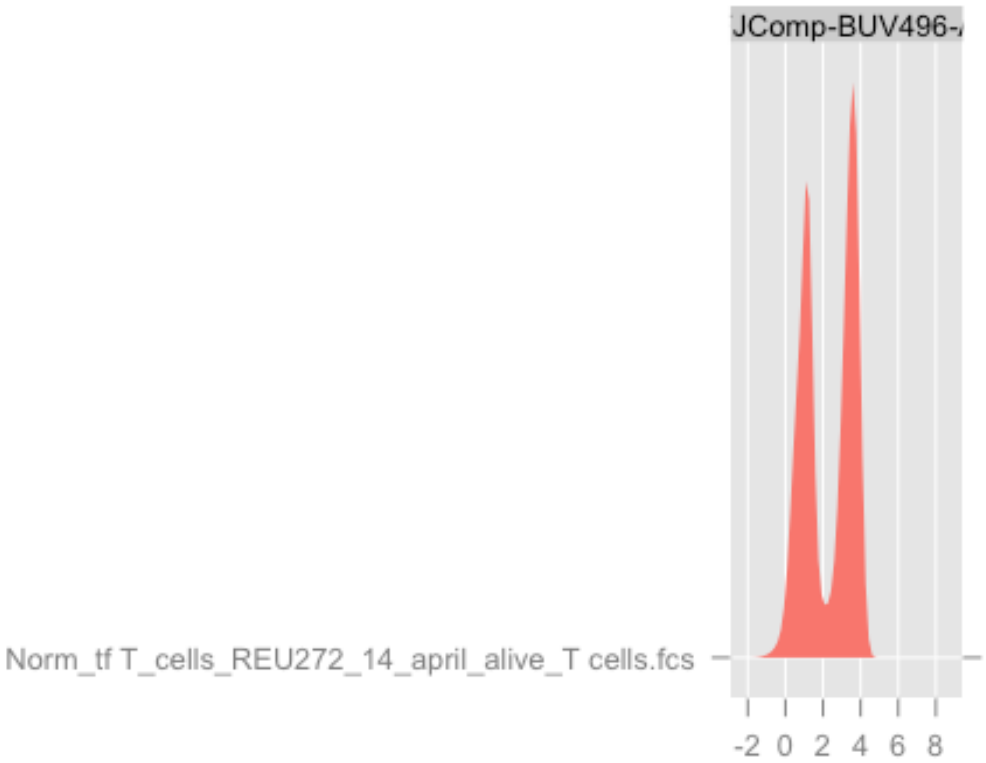
Sample_ID	Group_ID	batch
REU267	HC	A
REU268	HC	A
REU269	HC	A
REU270	HC	A
REU271_12juli	HC	B
REU271_13april	HC	C
REU271_14april	HC	D
REU271_7april	HC	E
REU271_9april	HC	F
REU271	HC	A
REU272_12juli	HC	B
REU272_13april	HC	C
REU272_14april	HC	D
REU272_7april	HC	E
REU272_9april	HC	F
REU272	HC	A

```
fcs.dir<- file.path(getwd(), "Transformed FCS files")
file_names <- list.files(fcs.dir)
file_name <- fsApply(fcs_transform, identifier)
```

```
file_name %in% file_names #check if the order of files in the directory and order of files in FlowSet object are matching

md <- data.frame(file_name, md, row.names=NULL)

labels <- c("B", "C", "D", "E", "F", "A")


model <- CytoNorm.train(files = train_files,
                        labels = labels,
                        channels = markerstotransform,
                        transformList = NULL,
                        FlowSOM.params = list(nCells = 6000,
                                              xdim = 10,
                                              ydim = 10,
                                              nClus = 5,
                                              scale = FALSE),
                        normMethod.train = QuantileNorm.train,
                        normParams = list(nQ = 101,
                                          goal = "mean"),
                        seed = 1,
                        verbose = TRUE)

CytoNorm.normalize(model = model,
                   files = validation_files,
                   labels = labels,
                   transformList = NULL,
                   transformList.reverse = NULL,
                   normMethod.normalize = QuantileNorm.normalize,
                   outputDir = "Normalized",
                   prefix = "Norm_",
                   clean = TRUE,
                   verbose = TRUE)



fcs.dir<- file.path(getwd(), "Normalized")
fcs_norm <- read.flowSet(path=fcs.dir, pattern="*.fcs", transformation = FALSE, truncate_max_range = FALSE)


densityplot(~`FJComp-BUV496-A`, fcs_transform[13])#before normalization
```




```
densityplot(~`FJComp-BUV496-A`, fcs_norm[3])#after normalization
```



At this point, you can take the FCS files from the Transformed files directory or Normalized FCS files directory and load these files into Cytosplore. More information about Cytosplore is available at: [https://www.cytosplore.org/]. You can skip arcsinh transformation for the data in Cytosplore.

Subsampling

You can either use our Downsampling_FlowSet function to randomly select n cells per sample or you can split your data into a small training set and a larger test set.

```
Downsampling_FlowSet <- function(x, samplesize , replace=TRUE, prob=NULL){
  if(missing(samplesize))
    samplesize <- min(flowCore::fsApply(x,nrow))
  flowCore::fsApply(x, function(ff){
    i <- sample(nrow(ff), size = samplesize, replace=replace, prob)
    ff[i,]
  })
}

fcs_transform <- fcs_transform[c(1:4,10,16)] #samples from batch A
md <- md[c(1:4,10,16),] # samples from batch A


Subsampling_FlowSet <- function(x, fraction, md){
```

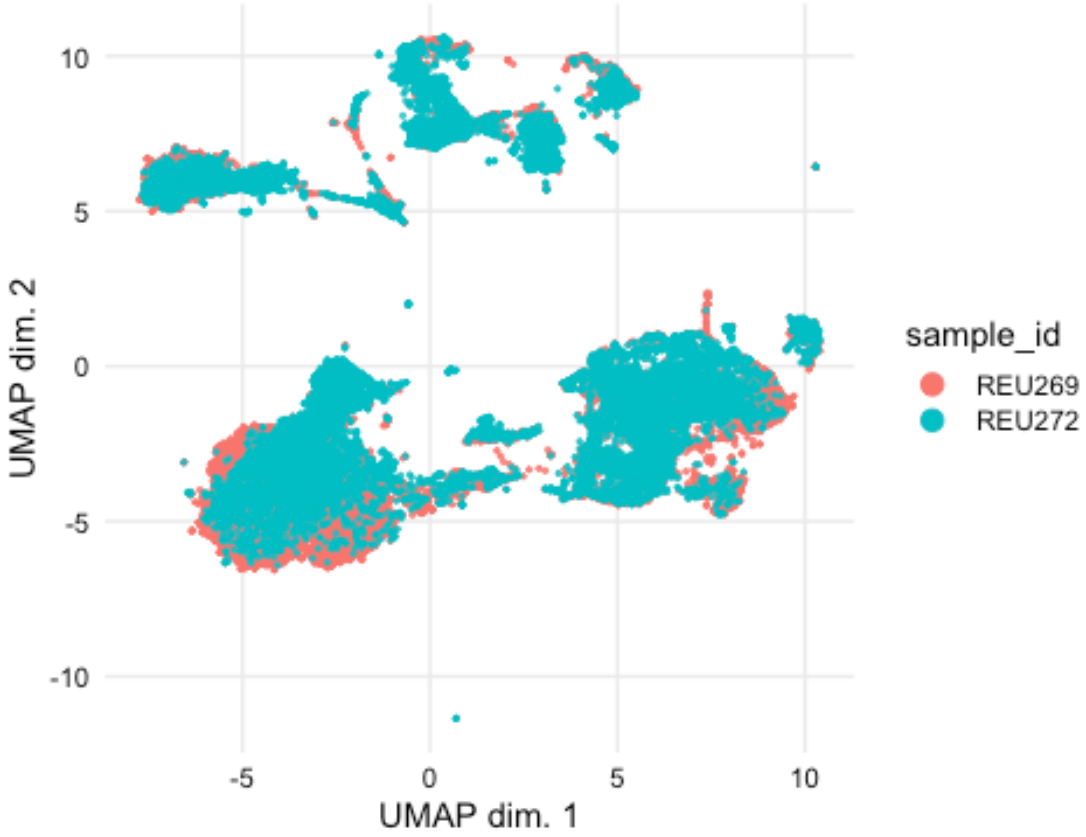
```
  b <- round(fraction*length(x), digits=0)

  listq <- sample(x=length(x), b, replace=FALSE)
  listq <- sort(listq)

  fcs_train<-x[c(listq)]

  md_train <- md[c(listq),]


fcs_train <<- fcs_train
md_train <<- md_train

  listy <- 1:length(x)

  listz <- subset(listy, !(listy %in% listq))

  listz <- sort(listz)

  fcs_test <- x[c(listz)]

  md_test <- md[c(listz),]

fcs_test <<- fcs_test
md_test <<- md_test

  }

Subsampling_FlowSet(fcs_transform,0.25, md=md) #test and train set are created
```

Exploring data with dimensionality reduction technique UMAP

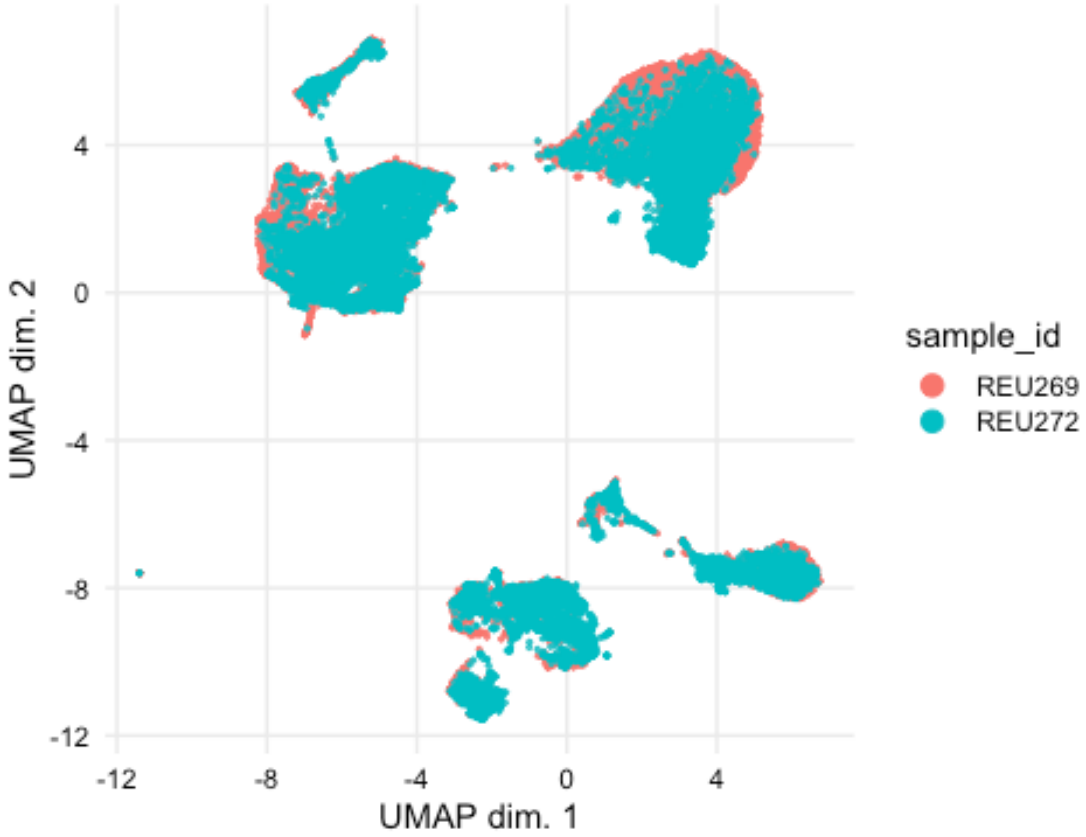
First, we can explore our data with an UMAP. UMAP will show the different populations present in the data and you can plot median expression of markers in the different clusters.

```
fcs_train <- Downsampling_FlowSet(fcs_train, samplesize =20000) # you can still downsample y
our training set if needed, but you can also include all or more cells

sce_train <- prepData(fcs_train, md=md_train, panel= panel, FACS = TRUE, transform=FALSE,
md_cols =list(file="file_name", id="Sample_ID", factors=c("Group_ID", "batch")))
assayNames(sce_train)[1] <- "exprs"

exprs_train <- assay(sce_train, "exprs")
exprs_train <- t(exprs_train)
exprs_train <- exprs_train[,c(marker_type)] #markers you want to use for clustering, you can als
o use marker_state or marker_type

set.seed(1234)
```

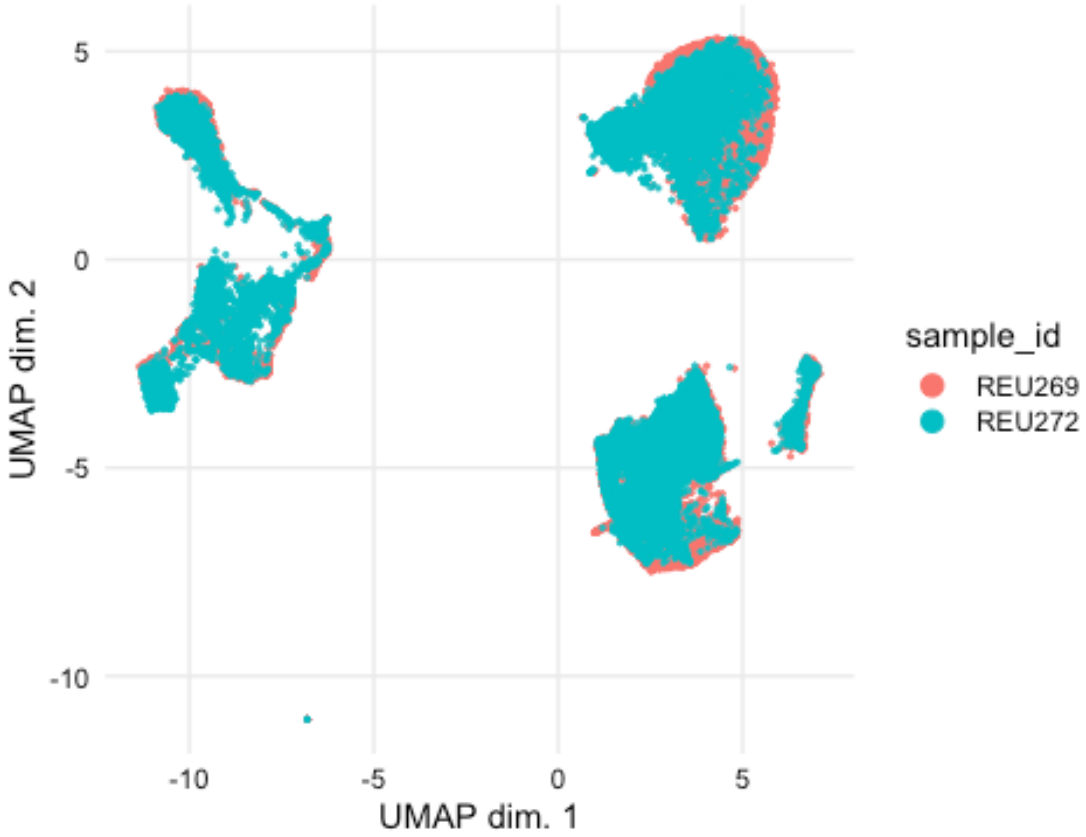
```
umap_train <- umap(exprs_train, n_neighbors=5)
reducedDim(sce_train, "UMAP")<- umap_train
plotDR(sce_train, "UMAP", color_by="sample_id")
```



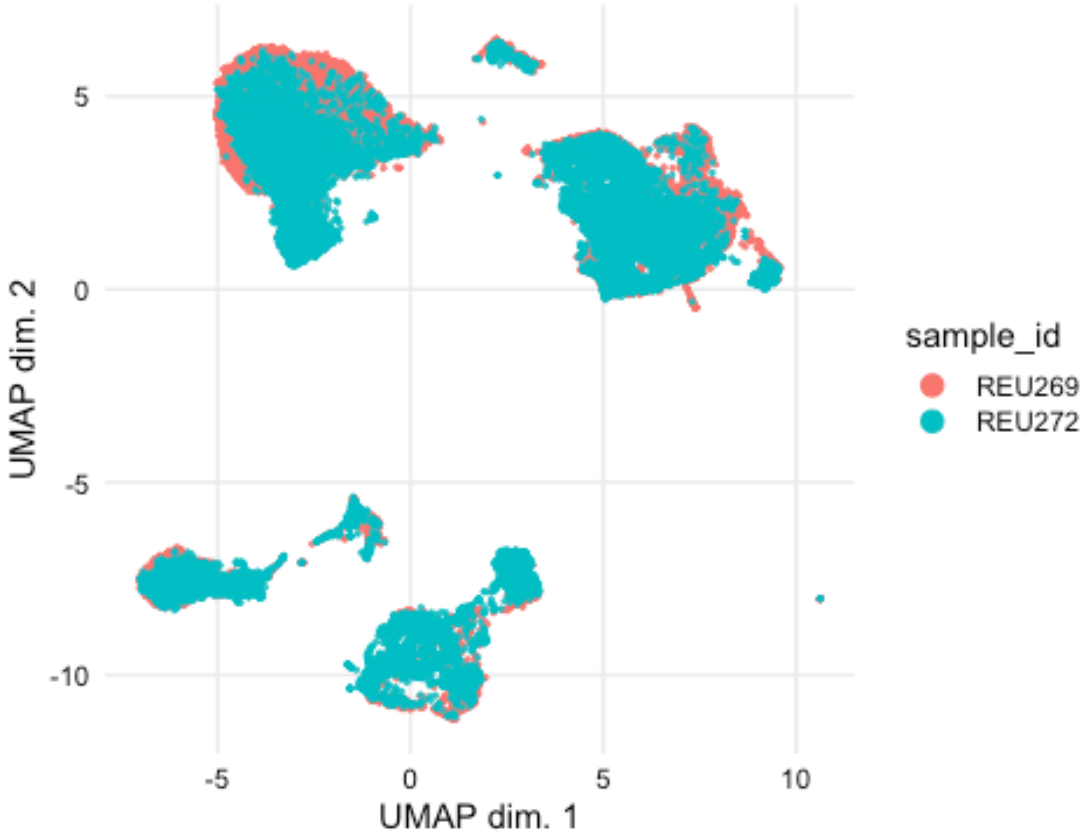

```
umap_train <- umap(exprs_train, n_neighbors=15)
reducedDim(sce_train, "UMAP")<- umap_train
plotDR(sce_train, "UMAP", color_by="sample_id")
```



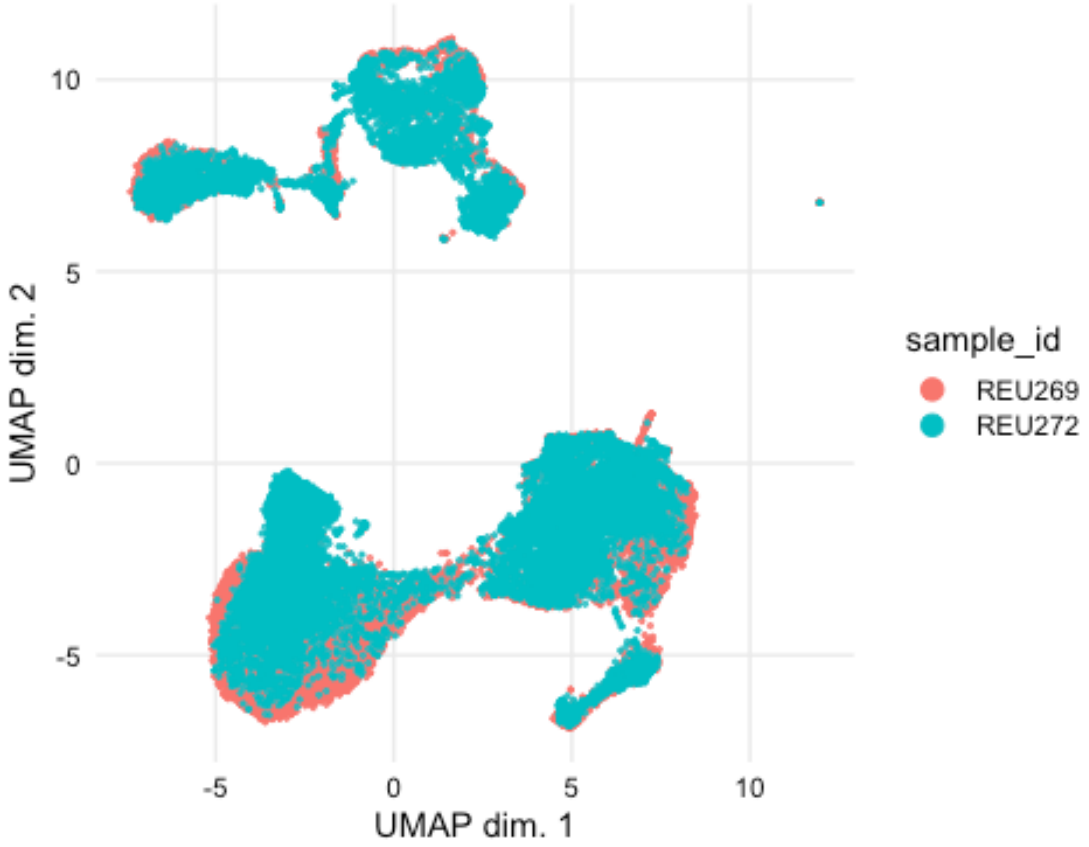

```
umap_train <- umap(exprs_train, n_neighbors=50)
reducedDim(sce_train, "UMAP")<- umap_train
plotDR(sce_train, "UMAP", color_by="sample_id")
```



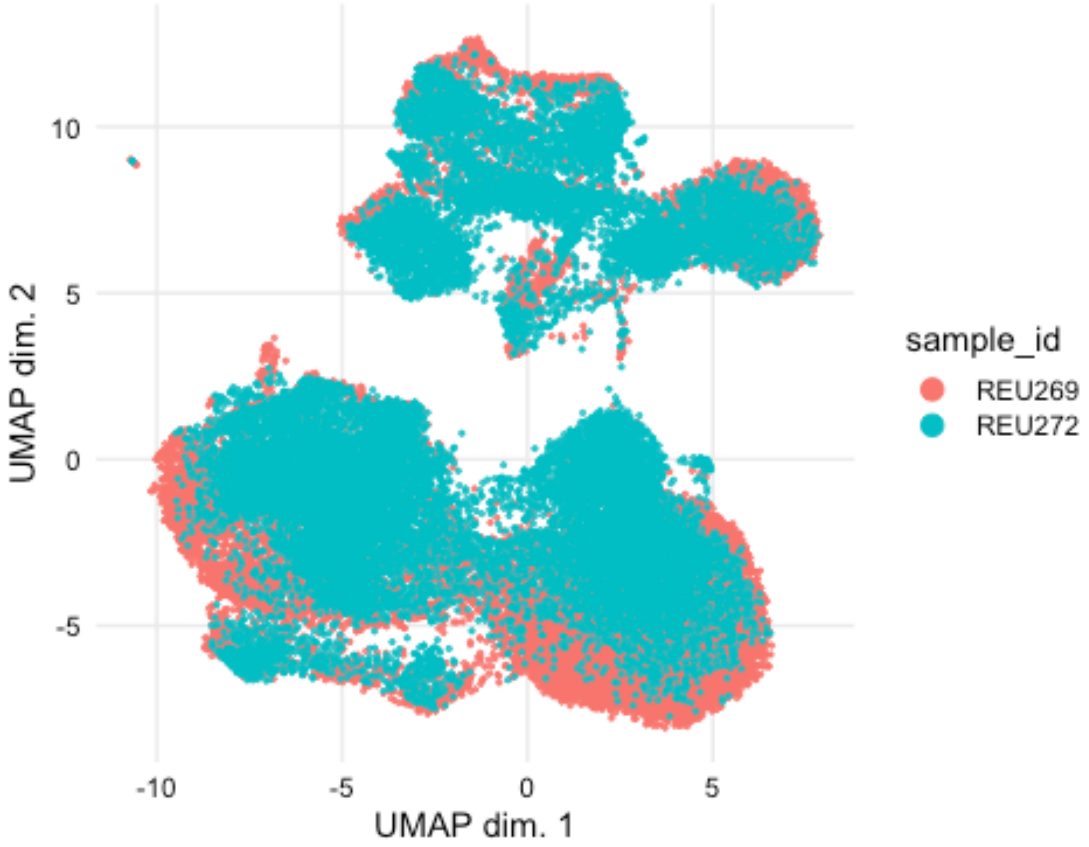

```
#if you chose the right n_neighbors, you can also test min_dist
umap_train <- umap(exprs_train, n_neighbors=15, min_dist = 0.01)
reducedDim(sce_train, "UMAP")<- umap_train
plotDR(sce_train, "UMAP", color_by="sample_id")
```



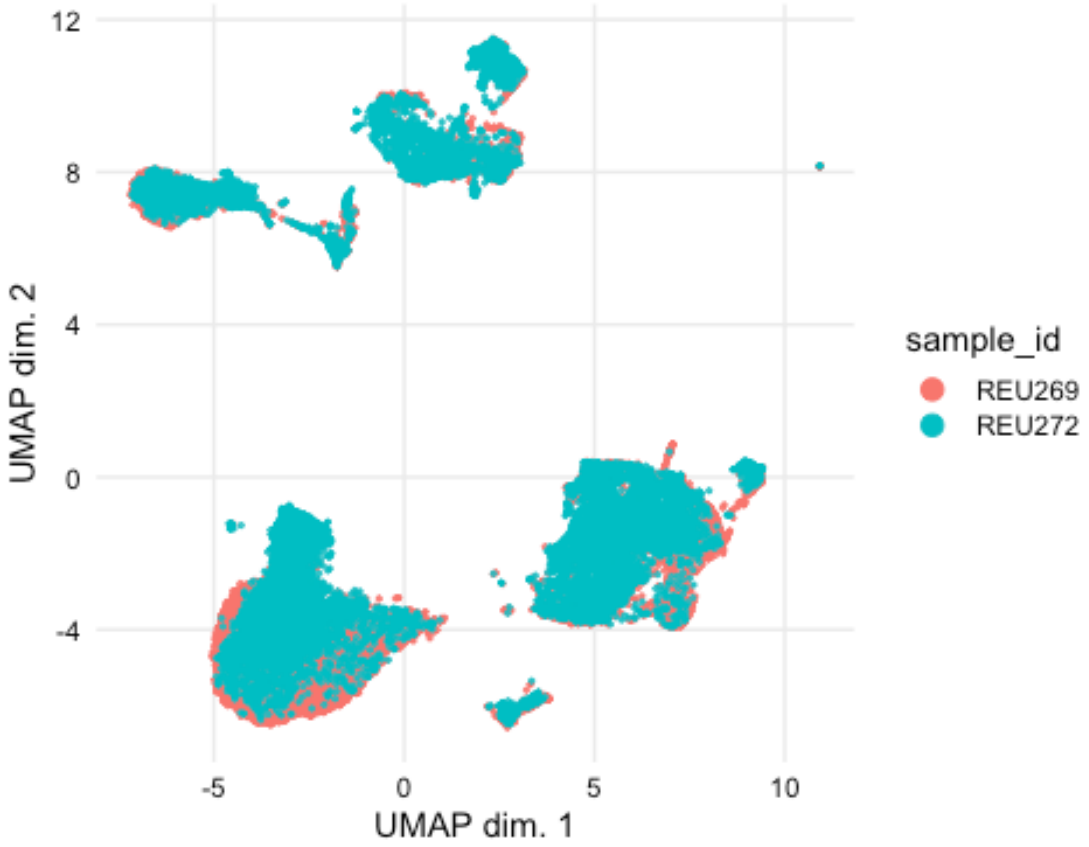

```
umap_train <- umap(exprs_train, n_neighbors=15, min_dist = 0.1)
reducedDim(sce_train, "UMAP")<- umap_train
plotDR(sce_train, "UMAP", color_by="sample_id")
```



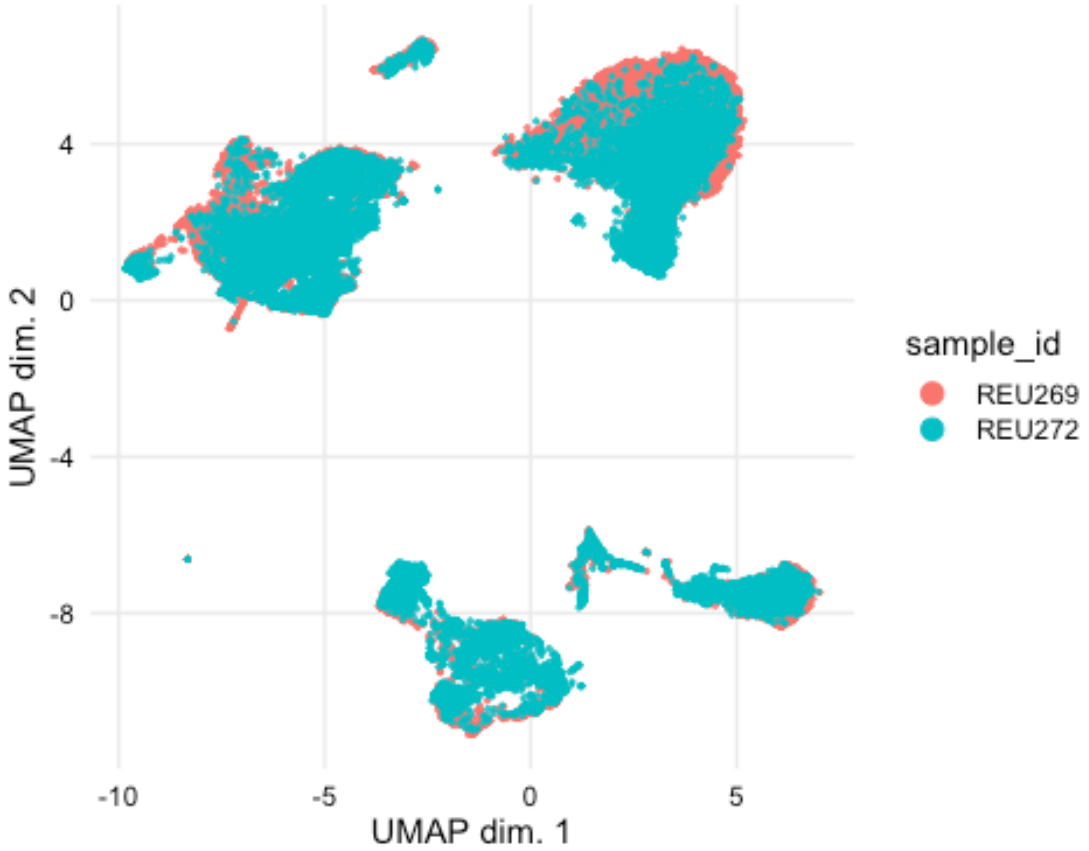

```
umap_train <- umap(exprs_train, n_neighbors=15, min_dist = 0.5)
reducedDim(sce_train, "UMAP")<- umap_train
plotDR(sce_train, "UMAP", color_by="sample_id")
```



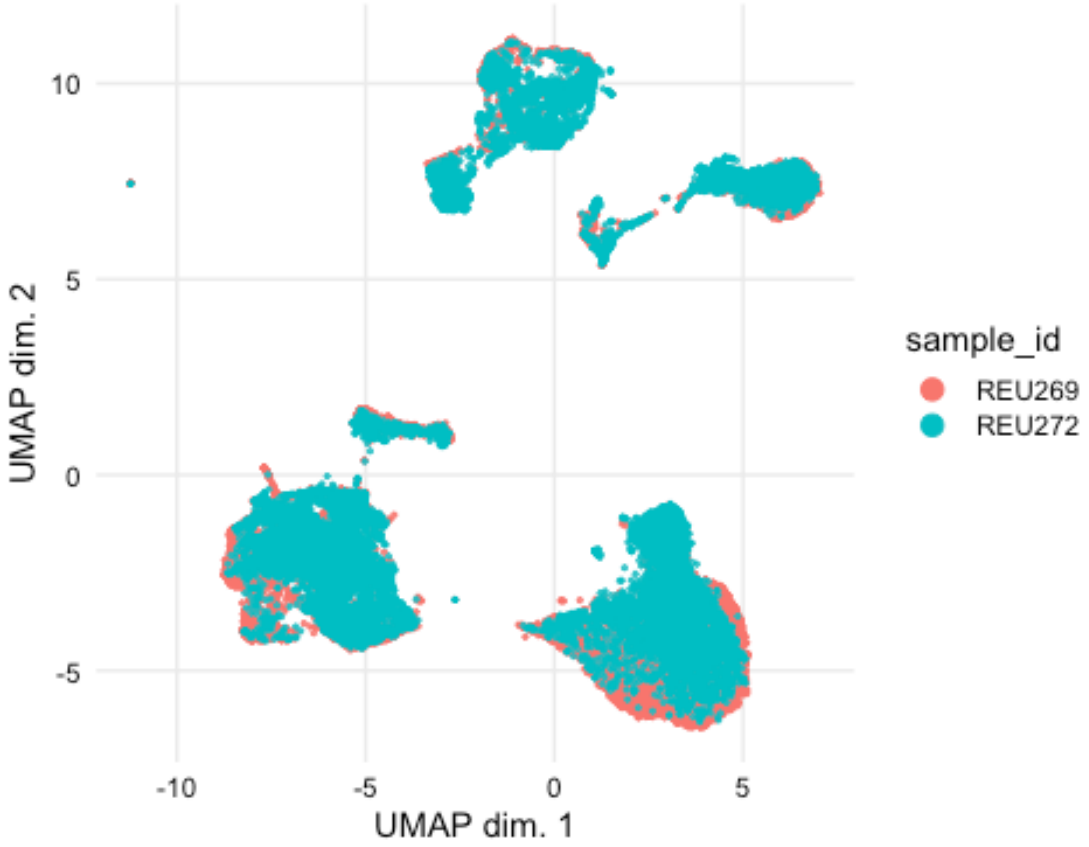

```
# choose the optimal parameters, to assess robustness of the umap you can vary the seed

set.seed(1234)
umap_train <- umap(exprs_train, n_neighbors=15, min_dist = 0.01)
reducedDim(sce_train, "UMAP")<- umap_train
plotDR(sce_train, "UMAP", color_by="sample_id")
```



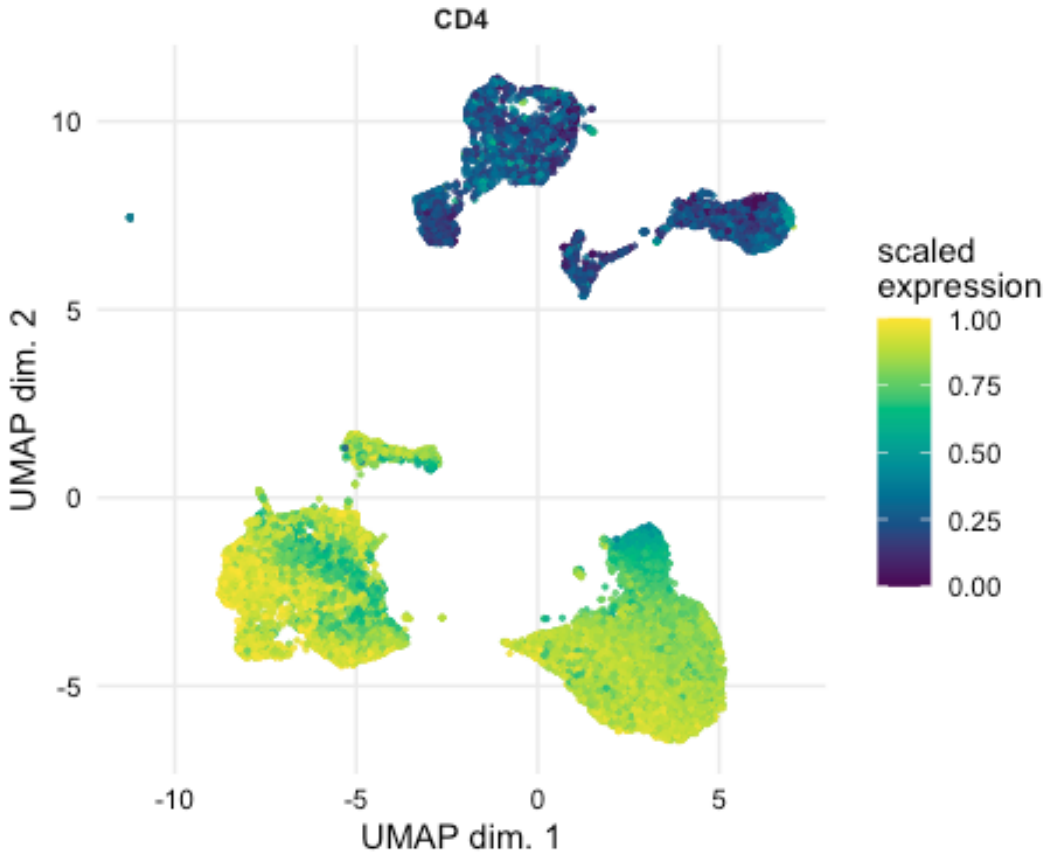

```
set.seed(7460)
umap_train <- umap(exprs_train, n_neighbors=15, min_dist = 0.01)
reducedDim(sce_train, "UMAP")<- umap_train
plotDR(sce_train, "UMAP", color_by="sample_id")
```



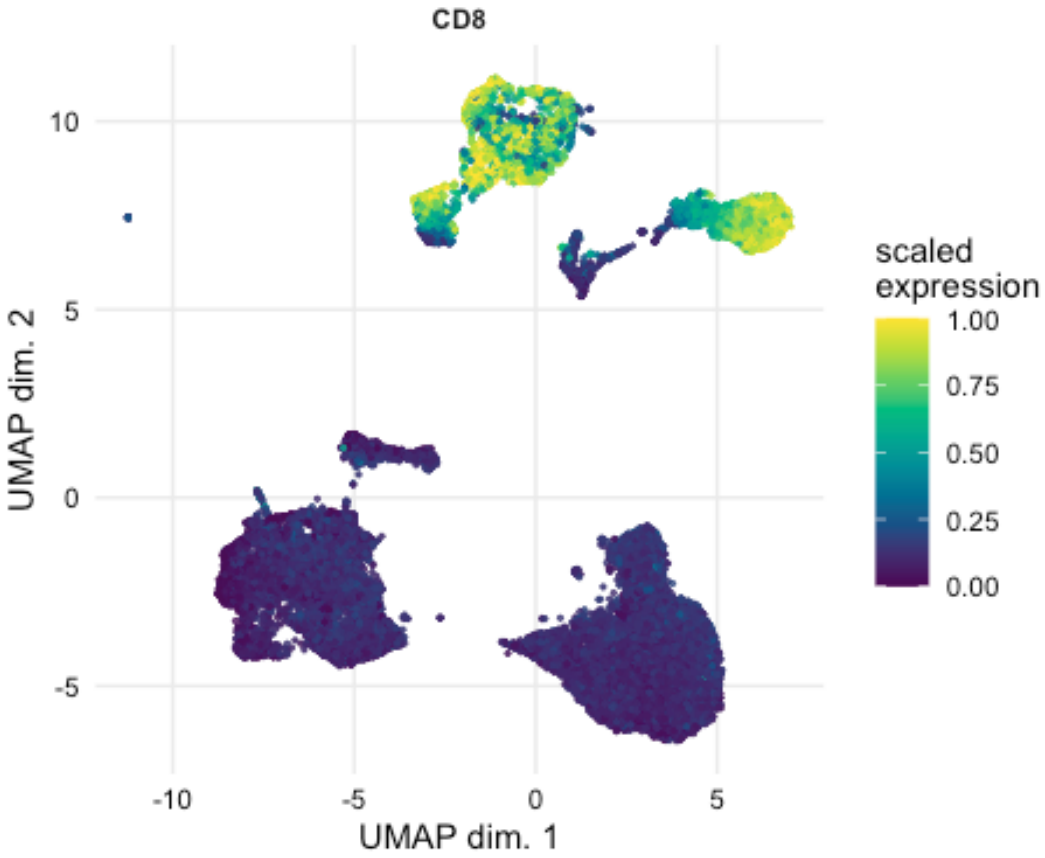

```
set.seed(5024)
umap_train <- umap(exprs_train, n_neighbors=15, min_dist = 0.01, ret_model = TRUE)
reducedDim(sce_train, "UMAP")<- umap_train$embedding
plotDR(sce_train, "UMAP", color_by="sample_id")
```




```
plotDR(sce_train, "UMAP", color_by="CD4")
```




```
plotDR(sce_train, "UMAP", color_by="CD8")
```



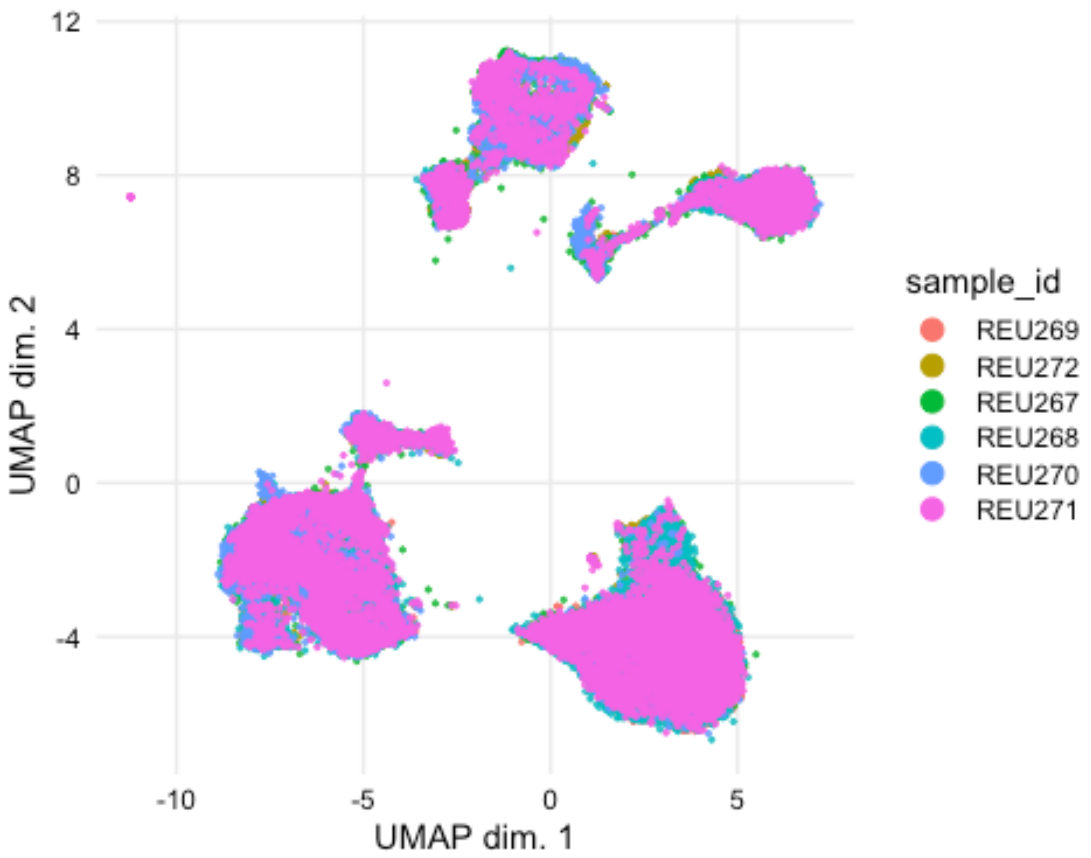
Now you can add the other samples to the UMAP

```
fcs_test <- Downsampling_FlowSet(fcs_test, samplesize =20000)

sce_test <-prepData(fcs_test, md=md_test, panel= panel, FACS = TRUE, transform=FALSE, m
d_cols =list(file="file_name", id="Sample_ID", factors=c("Group_ID", "batch")))
assayNames(sce_test)[1] <- "exprs"

exprs_test <- assay(sce_test, "exprs")
exprs_test <- t(exprs_test)
exprs_test <- exprs_test[,c(marker_type)]

umap_test <- umap_transform(exprs_test, umap_train)
```

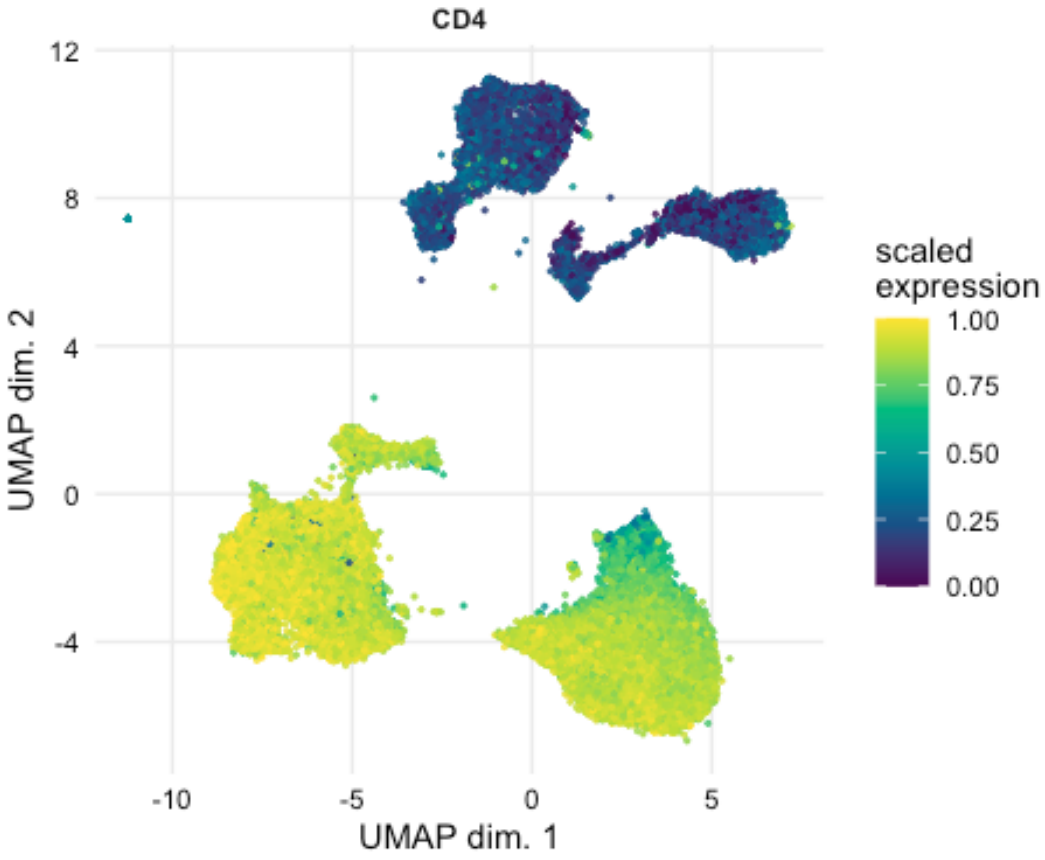
The other samples can now be embedded:

```
reducedDim(sce_train, "UMAP")<- NULL

umap_total <- rbind(umap_train$embedding, umap_test)

sce_total <- cbind(sce_train, sce_test)

reducedDim(sce_total, "UMAP") <- umap_total
```

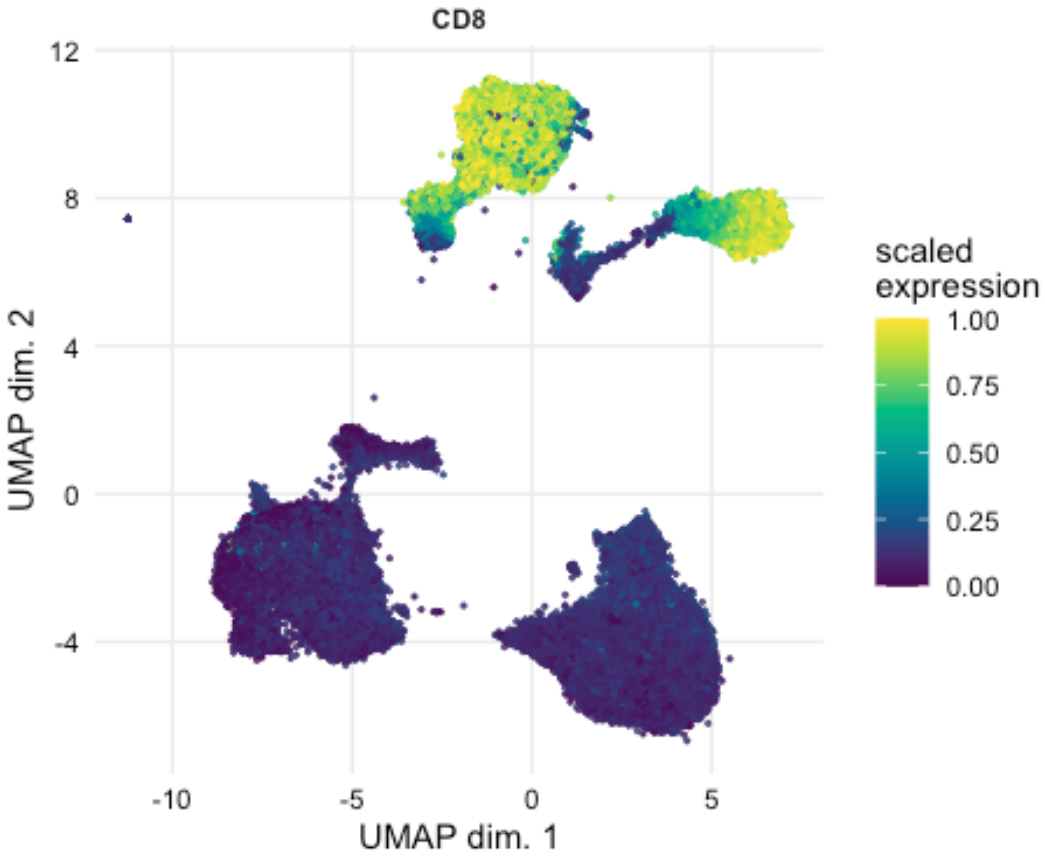
```
plotDR(sce_total, "UMAP", color_by = "sample_id")
```




```
plotDR(sce_total, "UMAP", color_by = "CD4")
```




```
plotDR(sce_total, "UMAP", color_by = "CD8")
```



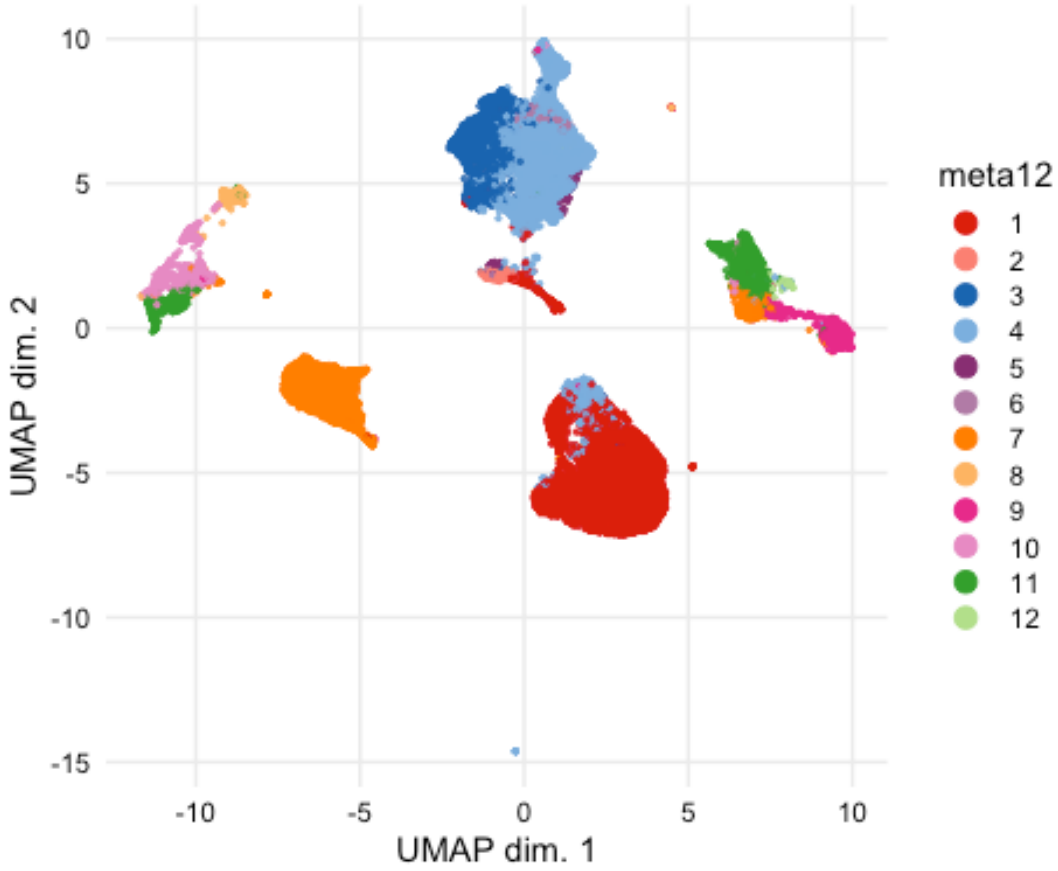
Clustering data

The CATALYST package provides functions to first cluster flow cytometry data with FlowSOM clustering and subsequently perform an UMAP or tSNE with the metacluster labels. Advantage of FlowSOM clustering is the speed of the algorithm and you don't need to downsample or split your dataset if the exploratory phase is over.

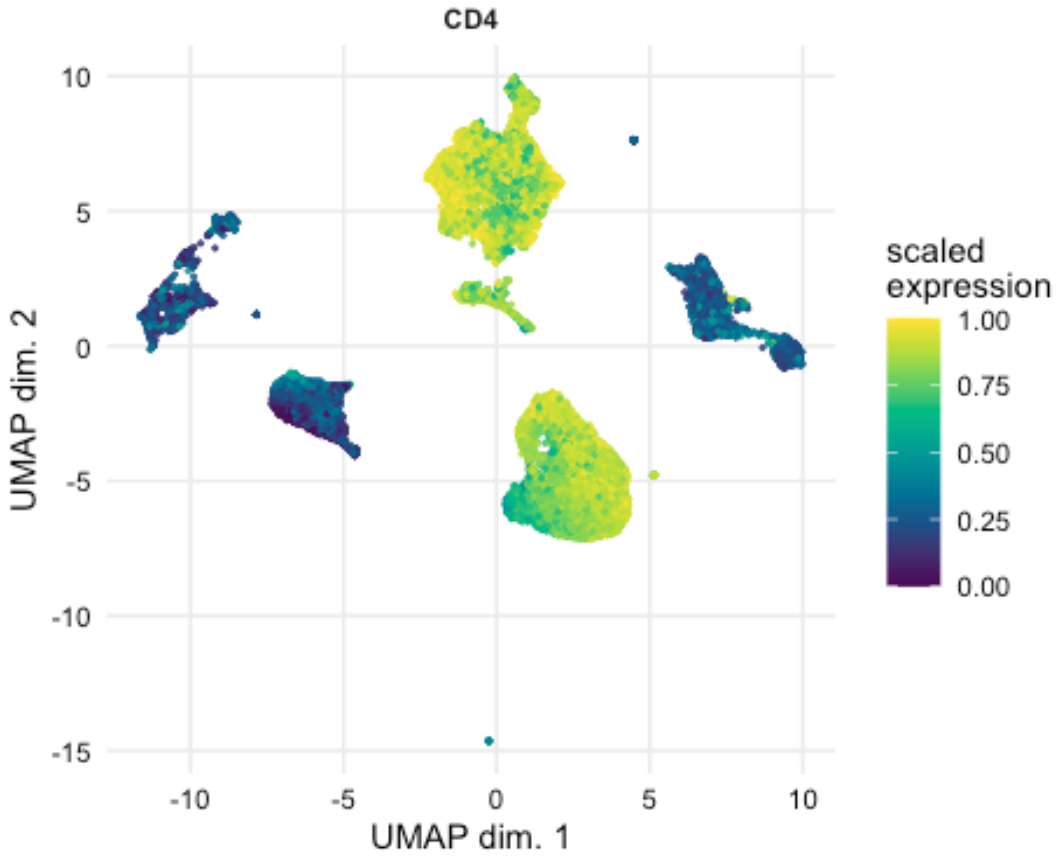
```
set.seed(5024)

sce<- prepData(fcs_transform, md=md, panel= panel, FACS = TRUE, transform=FALSE, md_co
ls =list(file="file_name", id="Sample_ID", factors=c("Group_ID", "batch")))

assayNames(sce)[1] <- "exprs"
sce <- cluster(sce, features="type", maxK=12, seed=5024)
```

With maxK you specify the number of clusters. The number of clusters to choose can be difficult. First, you need to ask yourself how many clusters would you expect in your data.You can also use the UMAP of the earlier steps to guide you in choice for number of clusters. Plotting median expression of markers in that UMAP can help to see the number of populations you would expect. Vary the number of clusters to find what best fits your data and is biological relevant.

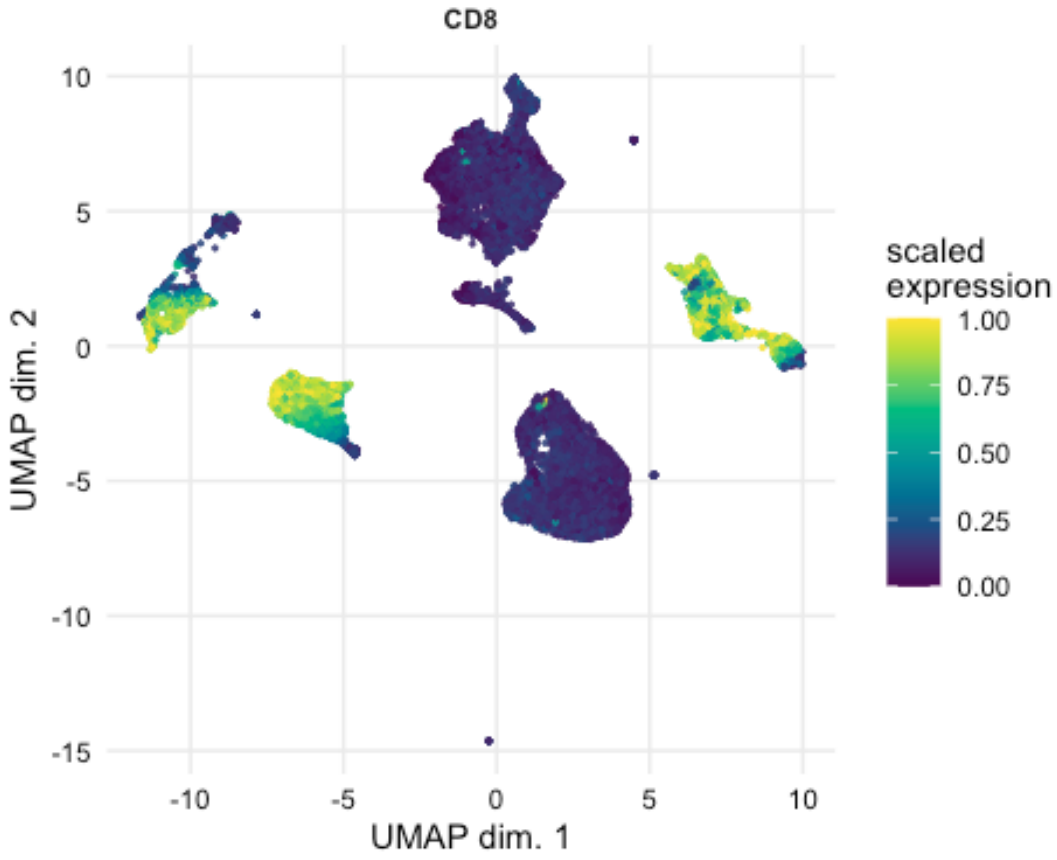
```
sce <- runDR(sce, "UMAP", cells = 6000, features = "type")
plotDR(sce, "UMAP", color_by="meta12")
```



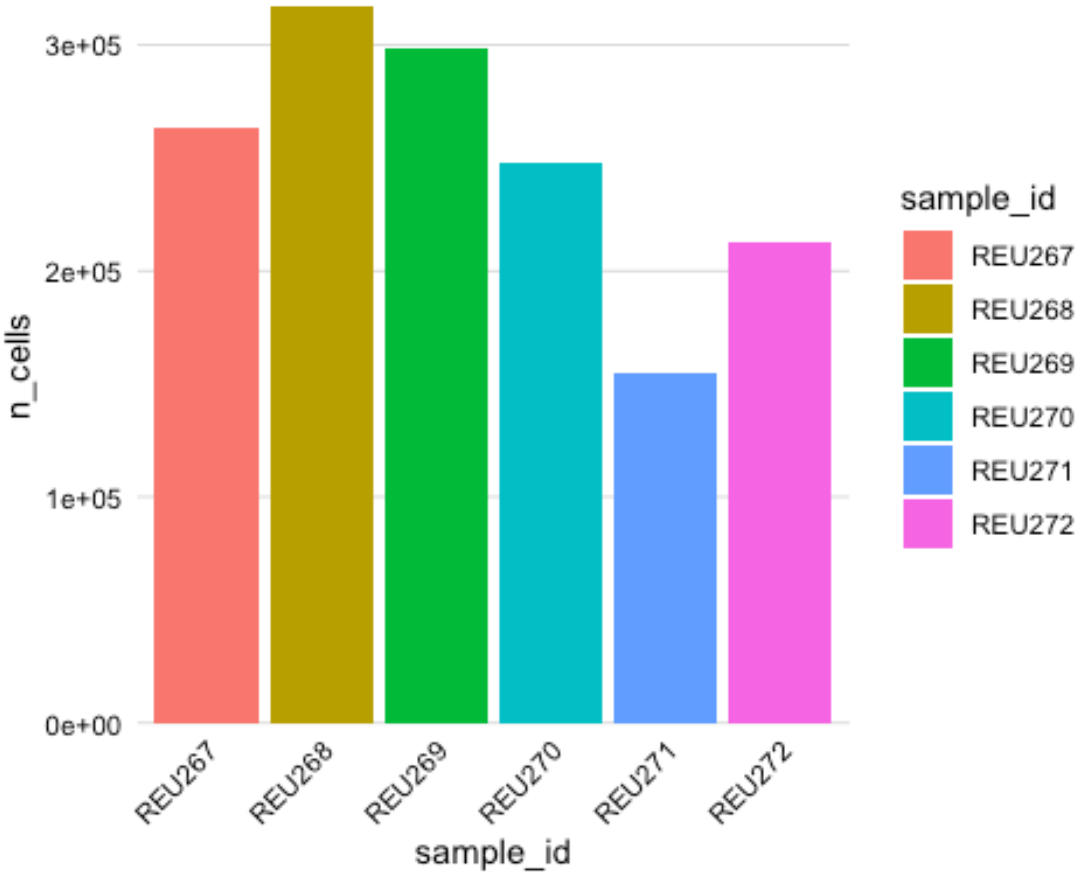

```
plotDR(sce, "UMAP", color_by="CD4")
```



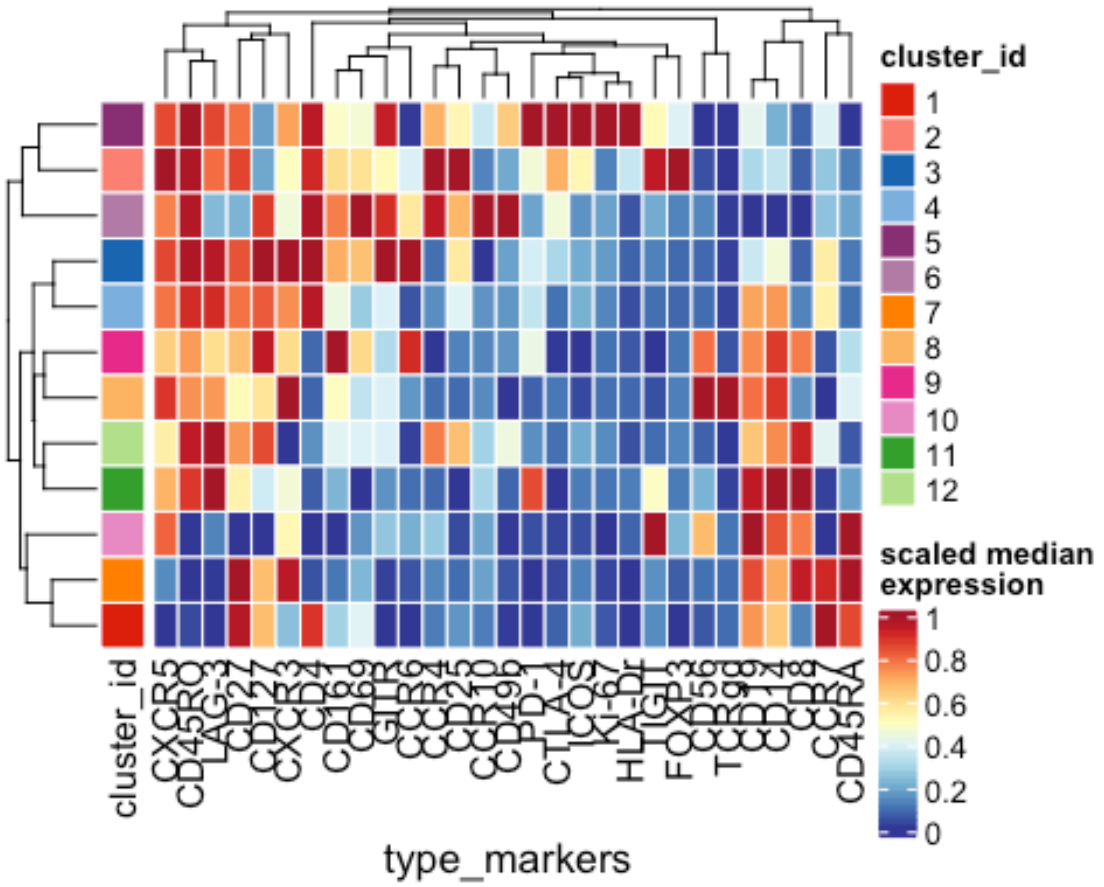

```
plotDR(sce, "UMAP", color_by="CD8")
```



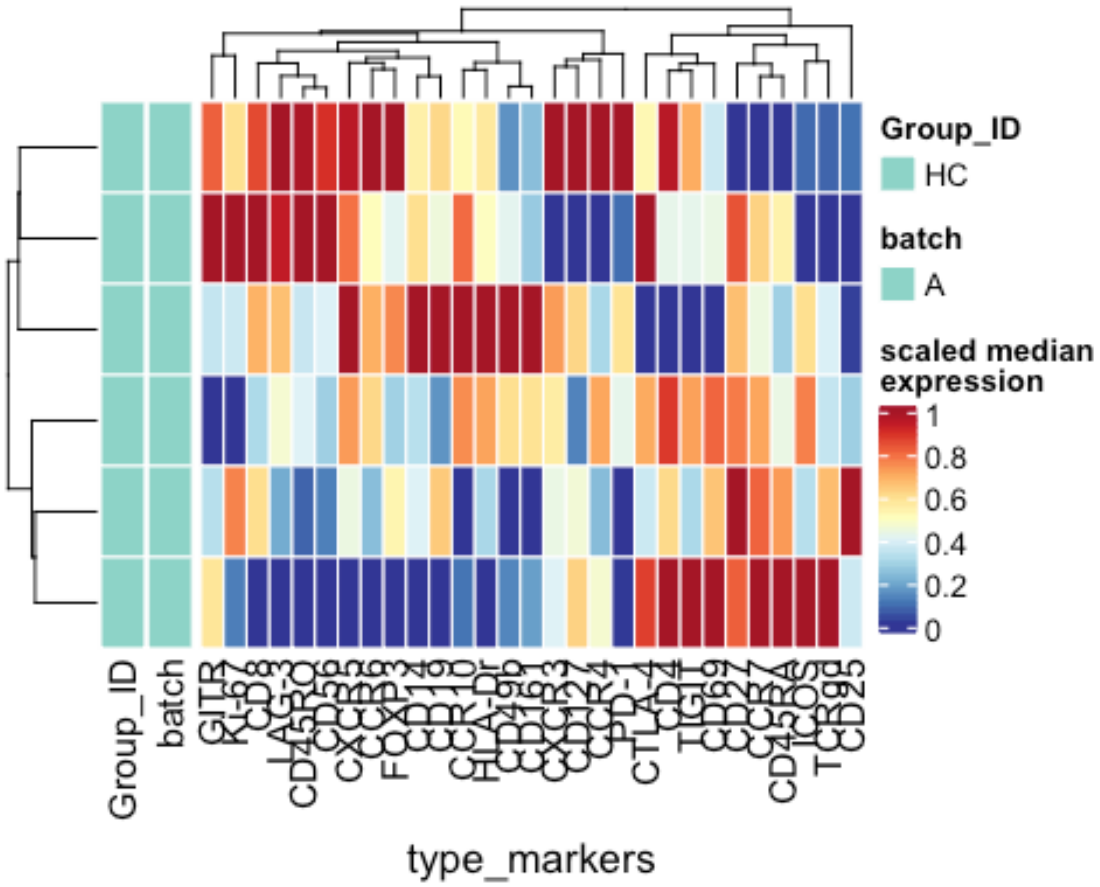

```
#Plot the number of cells per sample
Cell_numbers <-plotCounts(sce, prop=FALSE, group_by = "sample_id")#change prop to TRUE t
o see frequencies
print(Cell_numbers)
```



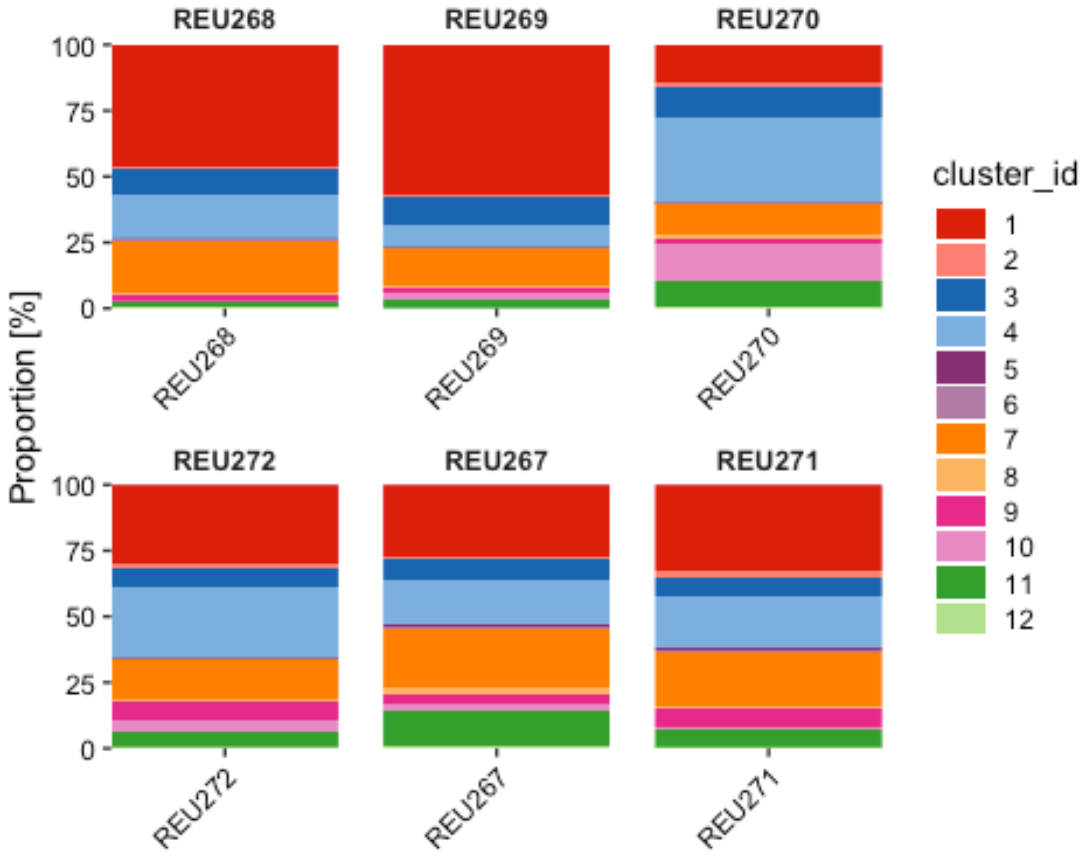

```
Cell_numbers_data <- Cell_numbers[["data"]] #dataframe of number of cells per sample, could b
e useful if you want to export dataframe and use it to make graphs in other programs, such as G
raphpad Prism

#Heatmap of the median expression per marker per metacluster or sample, more information ca
n be found in https://bioconductor.org/packages/release/bioc/html/CATALYST.html

plotExprHeatmap(sce, features = "type", by="cluster_id",k="meta12",scale = "last", q = 0, perc=T
RUE,bars = FALSE)
```




```
plotExprHeatmap(sce, features = "type", by="sample_id",k="meta12",scale = "last", q = 0, perc=
TRUE,bars = FALSE) #this plot can also be used to check for batch effects
```




```
Cell_freq_clusters <- plotAbundances(sce, k = "meta12", group_by = "sample_id")
print(Cell_freq_clusters)
```




```
Cell_freq_clusters_data <- Cell_freq_clusters[["data"]]
write.xlsx(x=Cell_freq_clusters_data, file="Cellclusterfrequencies.xlsx")
```

Case B – Covid human PBMC 36-color spectral flow cytometry

For Case B we used files from https://flowrepository.org/id/FR-FCM-Z3WR, the healthy control files: 062CD8.fcs · 22CBD6.fcs · 52BA23.fcs · 655A91.fcs · 6FD678.fcs · 77DA77.fcs · 94D44E.fcs · B29A26.fcs · F5E7EF.fcs Files were first pre-gated on living single CD3+CD19-T cells in FlowJo.

```
fcs.dir<- file.path(getwd(), "FCS files 2")

fcs_data <- read.flowSet(path=fcs.dir, pattern="*.fcs", transformation = FALSE, truncate_max_range = FALSE)

fcs_colname <- colnames(fcs_data)
marker_class <- rep("none", ncol(fcs_data[[1]]))
marker_state <- c(13,14,16,19,22,36,38,40)
marker_class[marker_state] <- "state"
marker_type <- c(7:12,17,18,20,21,23:35,37,39,41,42)
marker_class[marker_type] <- "type"
marker_class <- factor(marker_class, levels=c("type", "state", "none"))
```

```
antigen <- pData(parameters(fcs_data[[1]]))$desc

panel_B <- data.frame(fcs_colname, antigen, marker_class, row.names = NULL)
write.xlsx(panel_B, file="panel_B.xlsx", sheetName="Panel_B")

markerstotransform <- panel_B$fcs_colname[c(7:29,31:42)]
```

Transforming your data with the FlowVS package

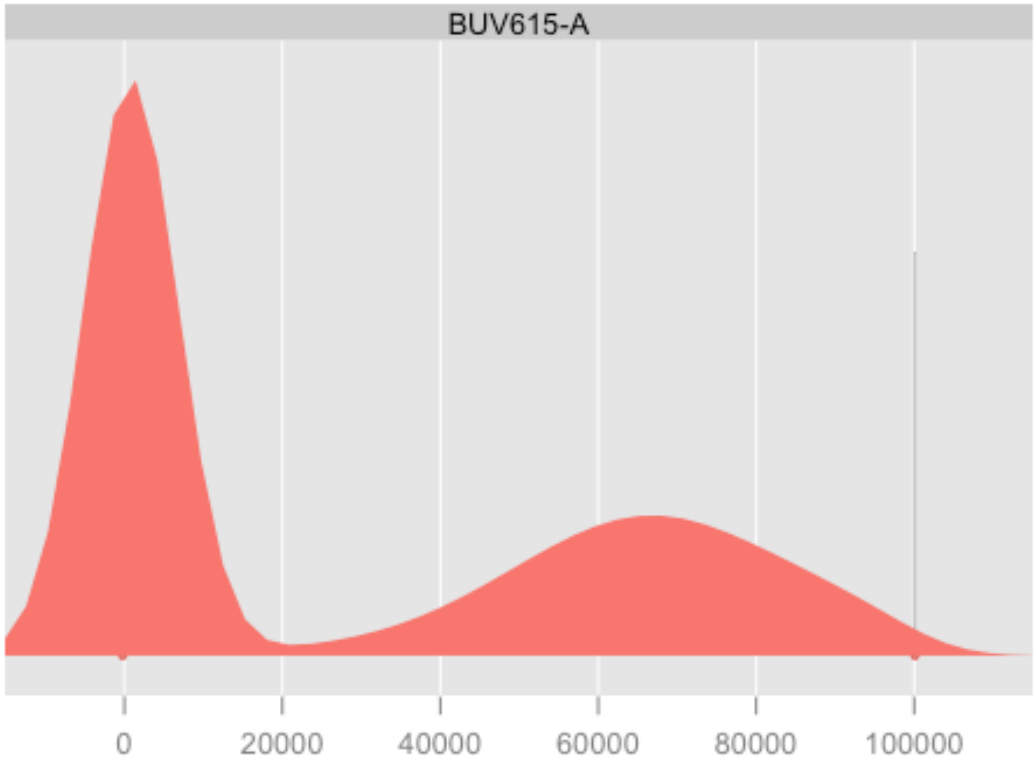
```
fcs_data_small <- Downsampling_FlowSet(x=fcs_data, samplesize = 2000) #samplesize is the n
umber of cells included, you can include more cells.

cofactors <- estParamFlowVS(fcs_data_small, channels=markerstotransform)
cofactordata <- data.frame(markerstotransform, cofactors)
write.csv(x=cofactordata, file="cofactordata_B.csv") #csv file
write.xlsx(x=cofactordata, file="cofactordata_B.xlsx", sheet="cofactordata_B")

fcs_transform <- transFlowVS(fcs_data, channels = markerstotransform, cofactors)
```

Transforming your data with a fixed cofactor

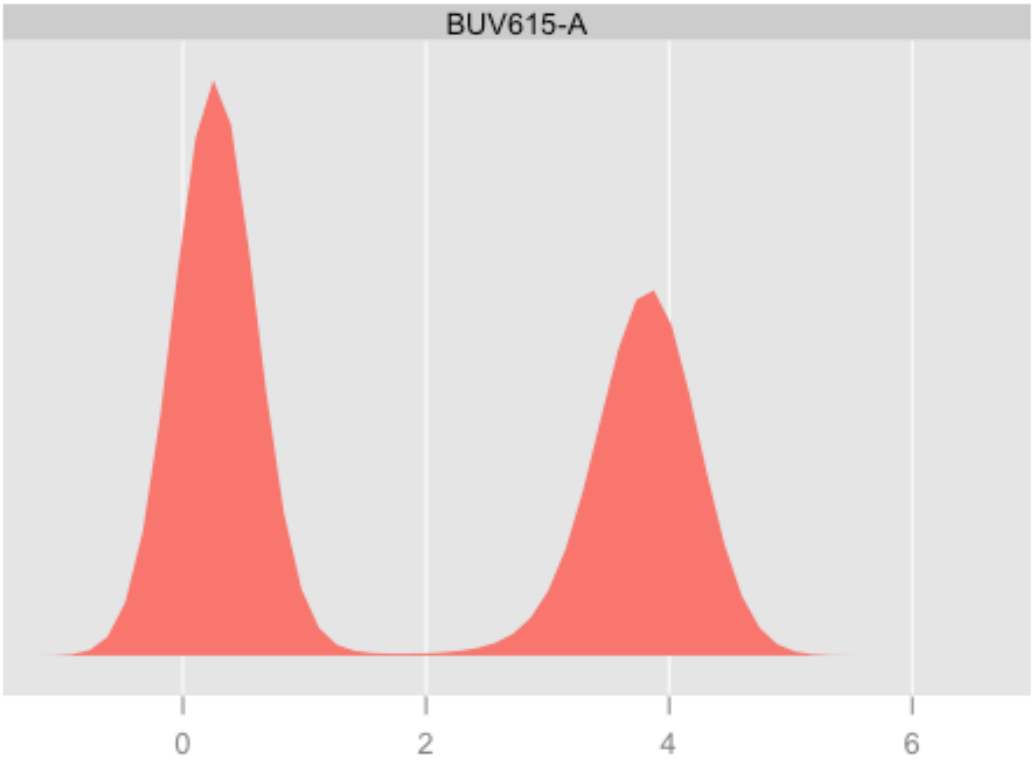
```
cofactor <- 3000
l <- length(markerstotransform)
cofactors<- rep(cofactor, l)

fcs_transform <- transFlowVS(fcs_data, channels = markerstotransform, cofactors)
filenames <- sampleNames(fcs_data)
sampleNames(fcs_transform) <- filenames

outdir <- file.path(getwd(), "Transformed FCS files 2")
filenames <- paste("TF_",fcs_data@phenoData@data$name)
write.flowSet(fcs_transform, outdir = outdir, filename = filenames)

fcs.dir <- file.path(getwd(), "Transformed FCS files 2")
fcs_transform <- read.flowSet(path=fcs.dir, pattern="*.fcs", transformation = FALSE, truncate_m
ax_range = FALSE)
```

To evaluate the data transformation, you can visualize density plots of markers with the FlowViz package.

```
densityplot(~`BUV615-A`, fcs_data[[1]]) #density plot before transformation, you can replace `BU
V615-A` by . to view all markers.
```




```
densityplot(~`BUV615-A`, fcs_transform[[1]]) # density plot after transformation
```



Automatic quality control of flow cytometry data

Either flowAI or peacoQC package can be used to clean flow cytometry data. For Case B we demonstrate peacoQC. You can run the peacoQC quality control first on 1 file to optimize the parameters. You will find the plot in the output directory and can check it to see what the algorithm removes. The first attempt could be not strict enough, because nothing is removed. The higher value for MAD, the less strict the algorithm is. In the last example with MAD of 2, the algorithm is very strict and removes almost 90% of the cells.

```
ff <- fcs_transform[[1]]

peacoqc_res <- PeacoQC(ff=ff, channels=markerstotransform, determine_good_cells = "all", save_fcs = FALSE, plot=TRUE, output_directory = "PeacoQCresults", IT_limit = 0.65, MAD=8)

## Starting quality control analysis for TF_ export_062CD8_CD3+ T cells.fcs

## Calculating peaks

## IT analysis removed 0% of the measurements

## MAD analysis removed 0% of the measurements

## The algorithm removed 0% of the measurements

## Plotting the results
```

```
peacoqc_res <- PeacoQC(ff=ff, channels=markerstotransform, determine_good_cells = "all", save_fcs = FALSE, plot=TRUE, output_directory = "PeacoQCresults", IT_limit = 0.55, MAD=5)

## Starting quality control analysis for TF_ export_062CD8_CD3+ T cells.fcs

## Calculating peaks

## IT analysis removed 0% of the measurements

## MAD analysis removed 0.26% of the measurements

## The algorithm removed 0.26% of the measurements

## Plotting the results

ff <- fcs_transform[[2]]
peacoqc_res <- PeacoQC(ff=ff, channels=markerstotransform, determine_good_cells = "all", save_fcs = FALSE, plot=TRUE, output_directory = "PeacoQCresults", IT_limit = 0.55, MAD=2)

## Starting quality control analysis for TF_ export_22CBD6_CD3+ T cells.fcs

## Calculating peaks

## IT analysis removed 0% of the measurements

## MAD analysis removed 87.96% of the measurements

## The algorithm removed 89.79% of the measurements

## Warning in PeacoQC(ff = ff, channels = markerstotransform, determine_good_cells
## = "all", : More than 70% was removed from file

## Plotting the results
```

After choosing the right parameters, you can apply the algorithm to all samples.

```
for(i in 1:9){
  ff <-fcs_transform[[i]]

  channels=markerstotransform

  peacoqc_res <- PeacoQC(ff, channels, determine_good_cells = "all", IT_limit=0.55, MAD=5, save_fcs = TRUE, plot=TRUE, output_directory = "PeacoQCresults")
}

fcs.dir <- file.path(getwd(),"PeacoQCresults/PeacoQC_results/fcs_files")

fcs_transform <- read.flowSet(path=fcs.dir, transformation=FALSE, truncate_max_range = FALSE) #construct new flowset from the cleaned files
```

Batch effects

In publically available datasets you might not always have access to control/technical replicate samples and you can not use Cytonorm to correct for batch effects. However,

you can make an overview of the dates FCS files were measured and give a batch label to your files, to be able to see the influences of batches in downstream analysis.

```
file_name <- fsApply(fcs_transform, identifier)
sample_id <- keyword(fcs_data, "GUID")# if a Sample ID was added to the original file you could also extract Sample ID or for example TUBE NAME or construct a column with new sample id's yourself
md_2 <- data.frame(file_name, sample_id, row.names=NULL)

batch <- keyword(fcs_data, "$DATE")
batch_label <- as.factor(batch)
levels(batch_label) <- c("A", "B")

md_2 <- data.frame(md_2, batch, batch_label, row.names=NULL)

colnames(md_2) <- c("file_name","sample_id", "batch","batch_label")
kable(md_2)
```

file_name	sample_id	batch	batch_label
TF_ export_062CD8_CD3+ T cells_QC.fcs	export_062CD8_CD3+ T cells.fcs	02-Jul-2020	A
TF_ export_22CBD6_CD3+ T cells_QC.fcs	export_22CBD6_CD3+ T cells.fcs	02-Jul-2020	A
TF_ export_52BA23_CD3+ T cells_QC.fcs	export_52BA23_CD3+ T cells.fcs	02-Jul-2020	A
TF_ export_655A91_CD3+ T cells_QC.fcs	export_655A91_CD3+ T cells.fcs	02-Jul-2020	A
TF_ export_6FD678_CD3+ T cells_QC.fcs	export_6FD678_CD3+ T cells.fcs	16-Sep-2020	B
TF_ export_77DA77_CD3+ T cells_QC.fcs	export_77DA77_CD3+ T cells.fcs	02-Jul-2020	A
TF_ export_94D44E_CD3+ T cells_QC.fcs	export_94D44E_CD3+ T cells.fcs	02-Jul-2020	A
TF_ export_B29A26_CD3+ T cells_QC.fcs	export_B29A26_CD3+ T cells.fcs	16-Sep-2020	B
TF_ export_F5E7EF_CD3+ T cells_QC.fcs	export_F5E7EF_CD3+ T cells.fcs	16-Sep-2020	B

Subsampling

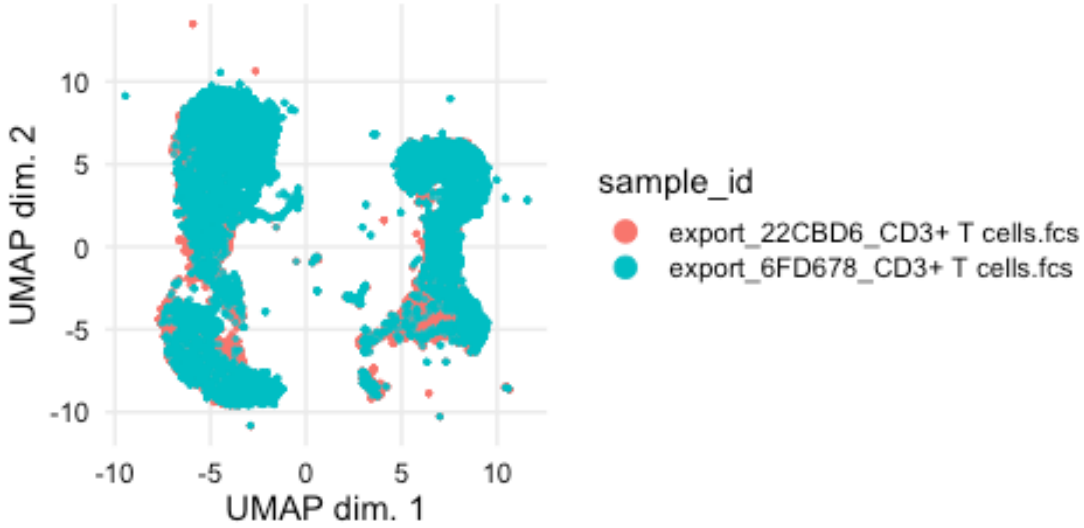
Before starting further data analysis, files can be downsampled or subsampled.

```
sce<- prepData(fcs_transform, md=md_2, panel= panel_B, FACS = TRUE, transform=FALSE, md_cols =list(file="file_name", id="sample_id", factors=c("batch", "batch_label"))) # you can specify the columns in your md file with md_cols

Subsampling_FlowSet(fcs_transform,0.25, md=md_2) #test and train set are created
```

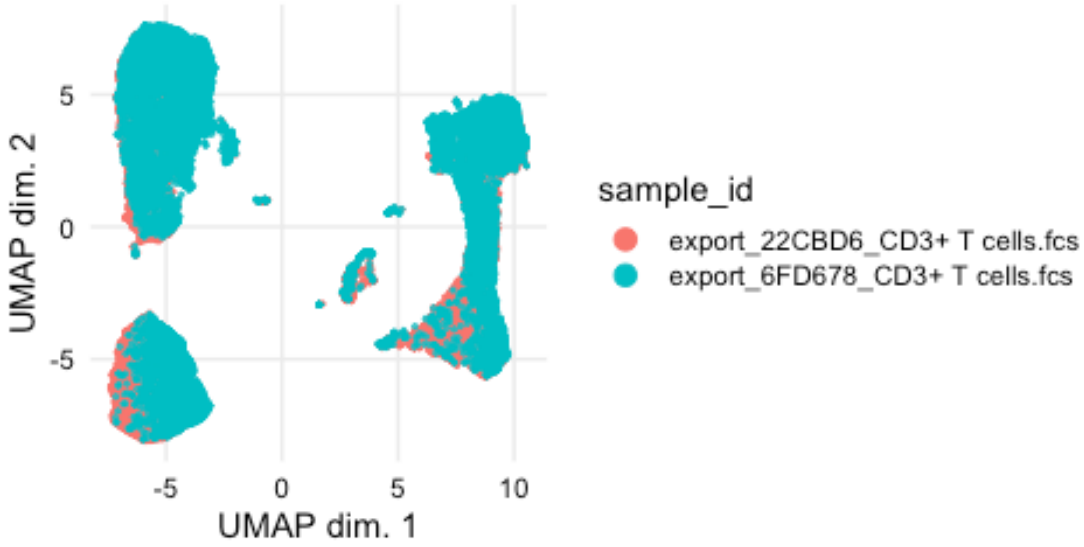
Exploring data with dimensionality reduction technique UMAP

First, we can explore our data with an UMAP. UMAP will show the different populations present in the data and you can plot median expression of markers in the different clusters.

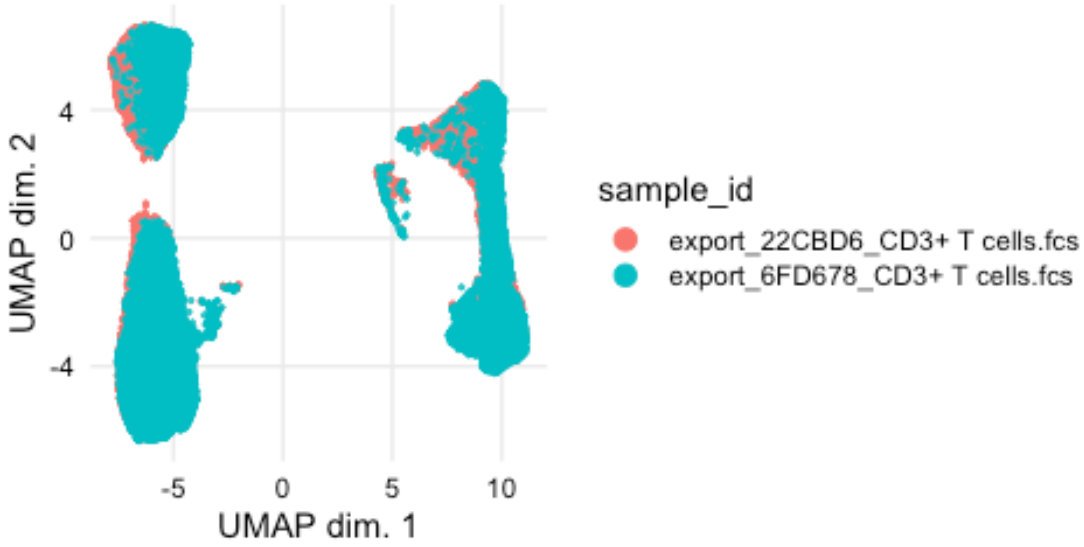
```
fcs_train <- Downsampling_FlowSet(fcs_train, samplesize =20000) # you can still downsample y
our training set if needed, but you can include more cells

sce_train <- prepData(fcs_train, md=md_train, panel= panel_B, FACS = TRUE, transform=FALS
E, md_cols =list(file="file_name", id="sample_id", factors=c("batch", "batch_label")))
assayNames(sce_train)[1] <- "exprs"

exprs_train <- assay(sce_train, "exprs")
exprs_train <- t(exprs_train)
exprs_train <- exprs_train[,c(marker_state)] #markers you want to use for clustering

umap_train <- umap(exprs_train, n_neighbors=5)
reducedDim(sce_train, "UMAP")<- umap_train
plotDR(sce_train, "UMAP", color_by="sample_id")
```



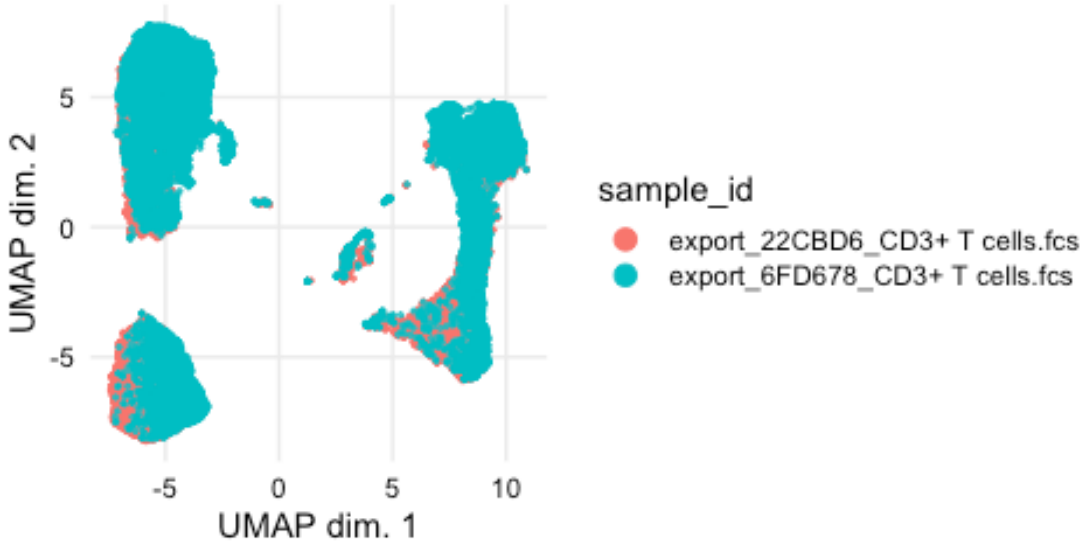

```
umap_train <- umap(exprs_train, n_neighbors=15)
reducedDim(sce_train, "UMAP")<- umap_train
plotDR(sce_train, "UMAP", color_by="sample_id")
```



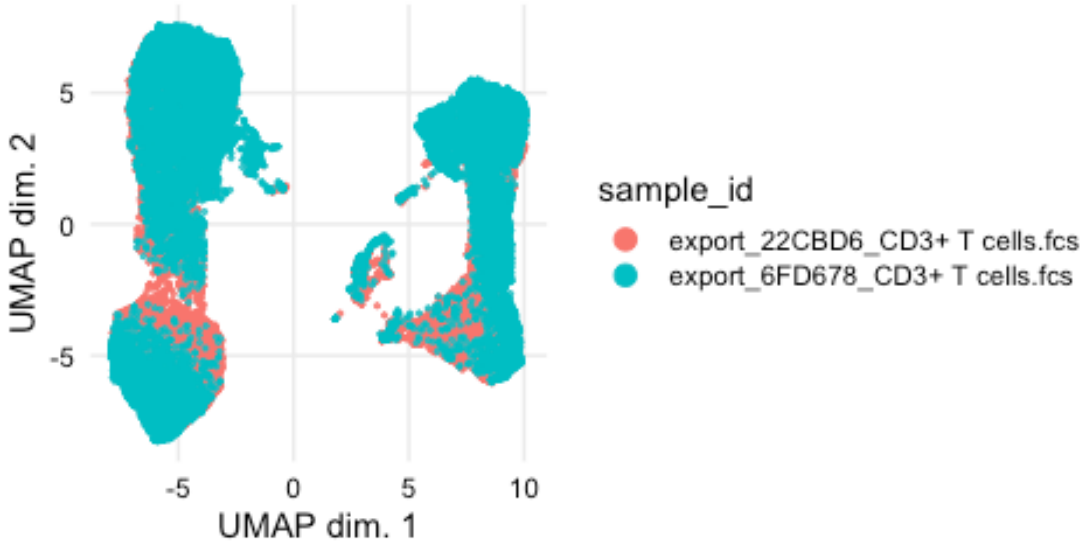

```
umap_train <- umap(exprs_train, n_neighbors=50)
reducedDim(sce_train, "UMAP")<- umap_train
plotDR(sce_train, "UMAP", color_by="sample_id")
```



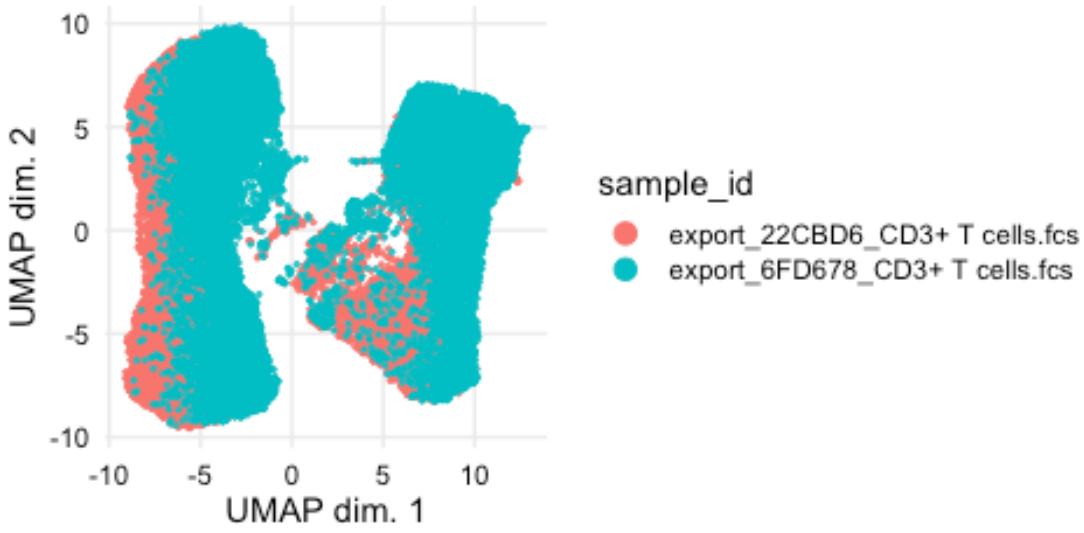

```
#if you chose the right n_neighbors, you can also test min_dist
umap_train <- umap(exprs_train, n_neighbors=15, min_dist = 0.01)
reducedDim(sce_train, "UMAP")<- umap_train
plotDR(sce_train, "UMAP", color_by="sample_id")
```



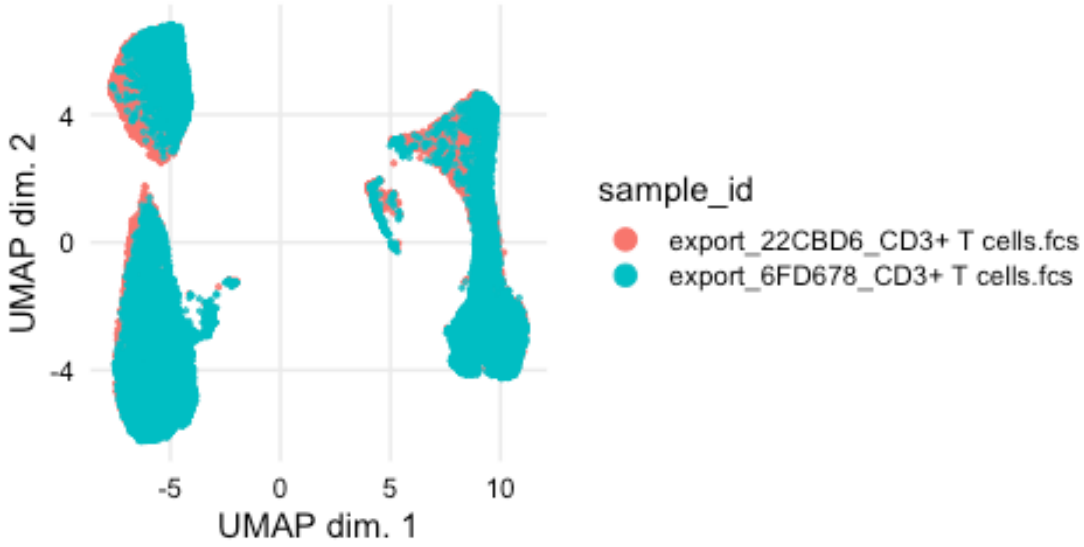

```
umap_train <- umap(exprs_train, n_neighbors=15, min_dist = 0.1)
reducedDim(sce_train, "UMAP")<- umap_train
plotDR(sce_train, "UMAP", color_by="sample_id")
```



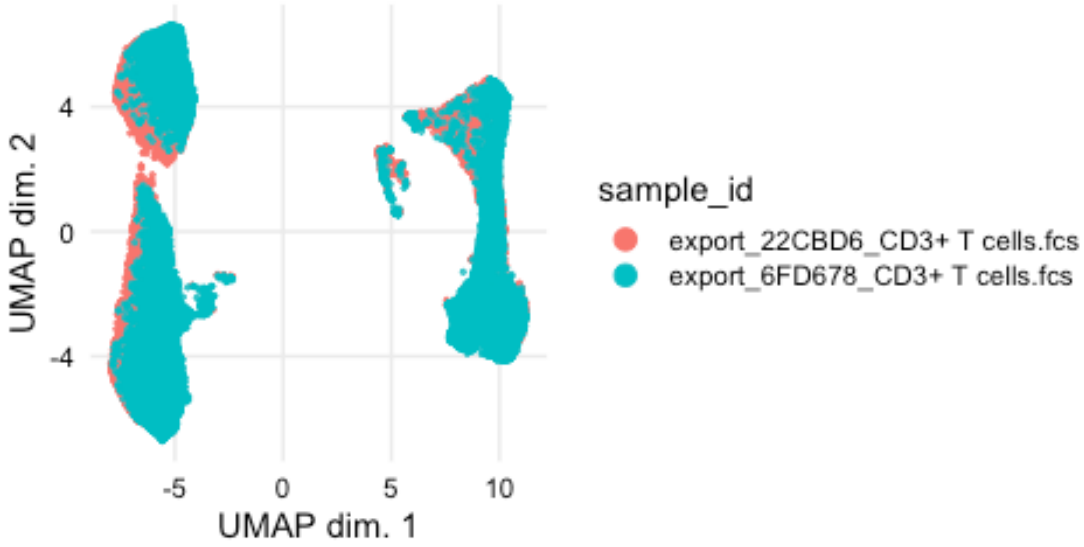

```
umap_train <- umap(exprs_train, n_neighbors=15, min_dist = 0.5)
reducedDim(sce_train, "UMAP")<- umap_train
plotDR(sce_train, "UMAP", color_by="sample_id")
```



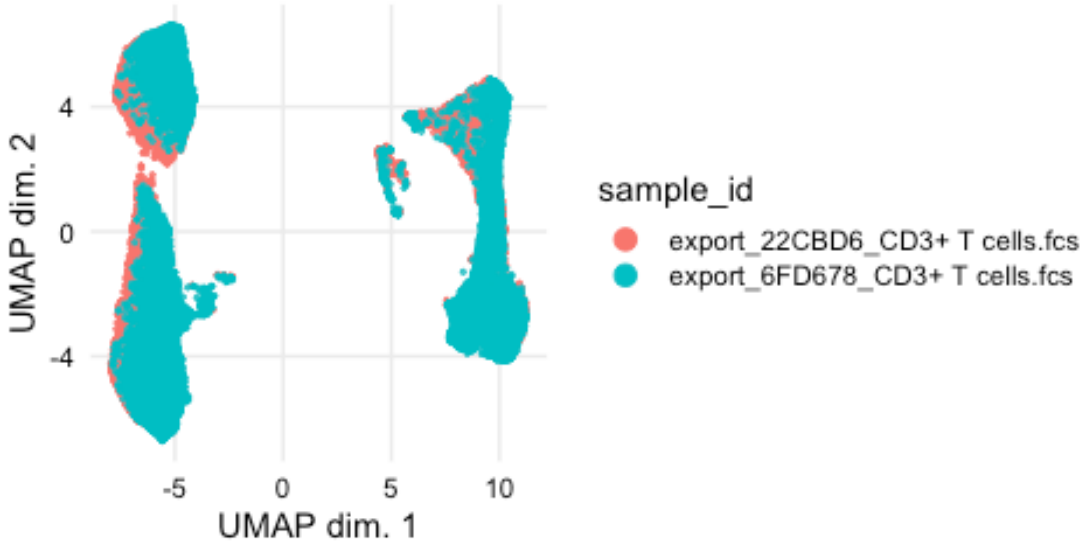

```
# choose the optimal parameters, to assess robustness of the umap you can vary the seed

set.seed(1234)
umap_train <- umap(exprs_train, n_neighbors=50, min_dist = 0.01)
reducedDim(sce_train, "UMAP")<- umap_train
plotDR(sce_train, "UMAP", color_by="sample_id")
```



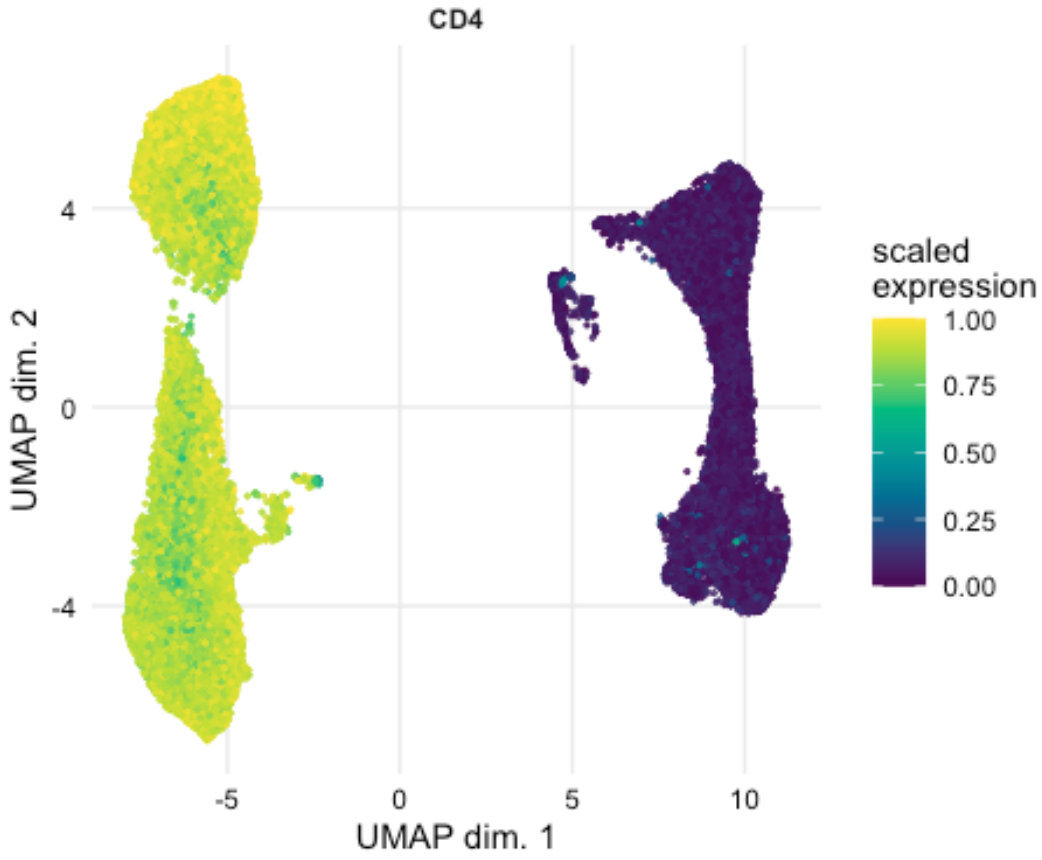

```
set.seed(7460)
umap_train <- umap(exprs_train, n_neighbors=50, min_dist = 0.01, ret_model = TRUE)
reducedDim(sce_train, "UMAP")<- umap_train$embedding
plotDR(sce_train, "UMAP", color_by="sample_id")
```



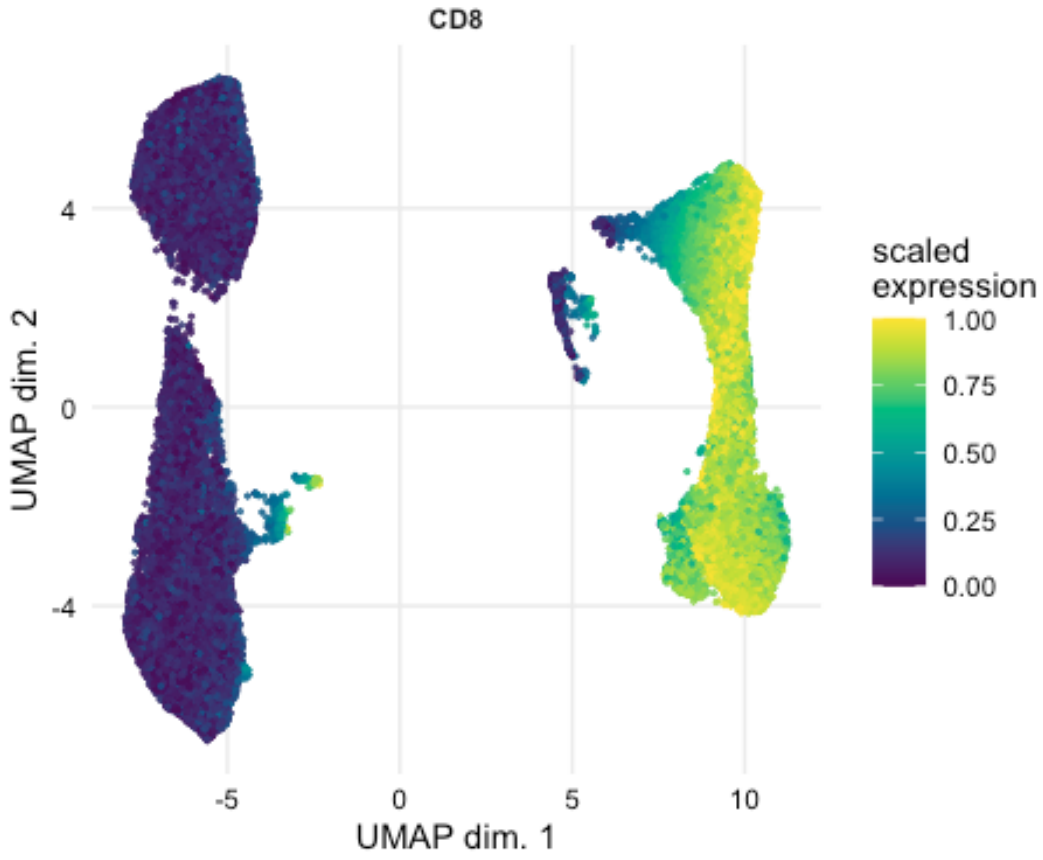

```
plotDR(sce_train, "UMAP", color_by="sample_id")
```



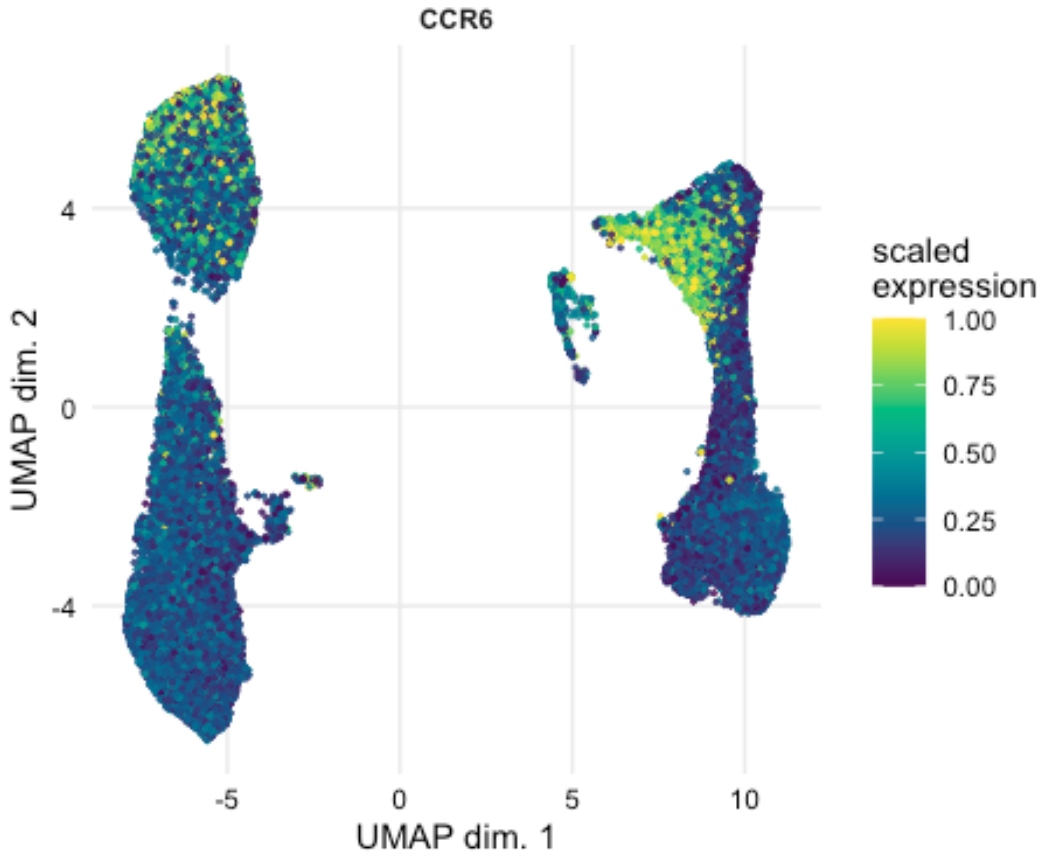

```
plotDR(sce_train, "UMAP", color_by="CD4")
```




```
plotDR(sce_train, "UMAP", color_by="CD8")
```




```
plotDR(sce_train, "UMAP", color_by="CCR6")# Now you can also plot your type markers on the UMAP
```



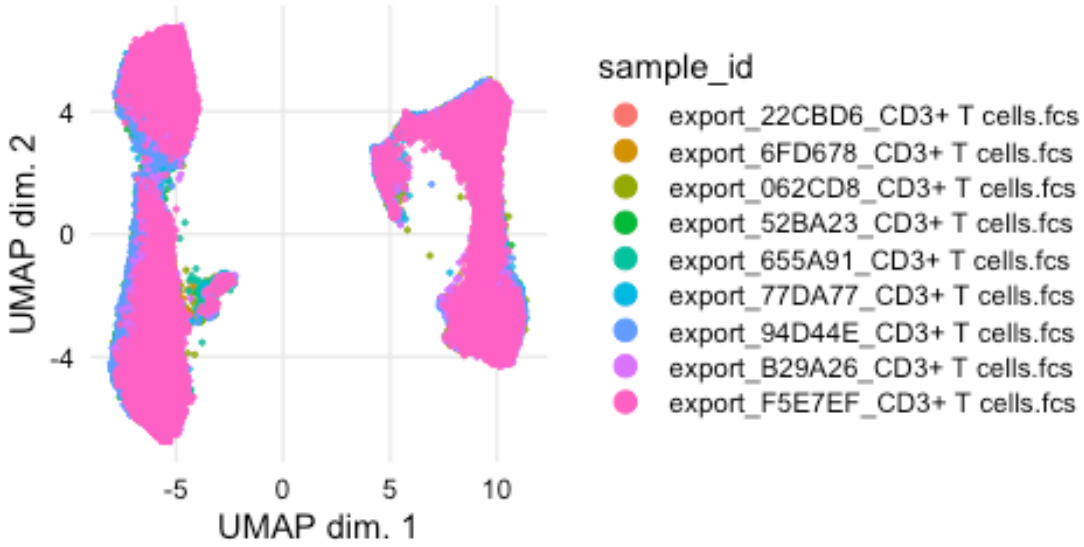
Now you can add the other samples to the UMAP

```
fcs_test <- Downsampling_FlowSet(fcs_test, samplesize =20000)

sce_test <- prepData(fcs_test, md=md_test, panel= panel_B, FACS = TRUE, transform=FALSE,
md_cols =list(file="file_name", id="sample_id", factors=c("batch", "batch_label")))
assayNames(sce_test)[1] <- "exprs"

exprs_test <- assay(sce_test, "exprs")
exprs_test <- t(exprs_test)
exprs_test <- exprs_test[,c(marker_state)]

umap_test <- umap_transform(exprs_test, umap_train)
```

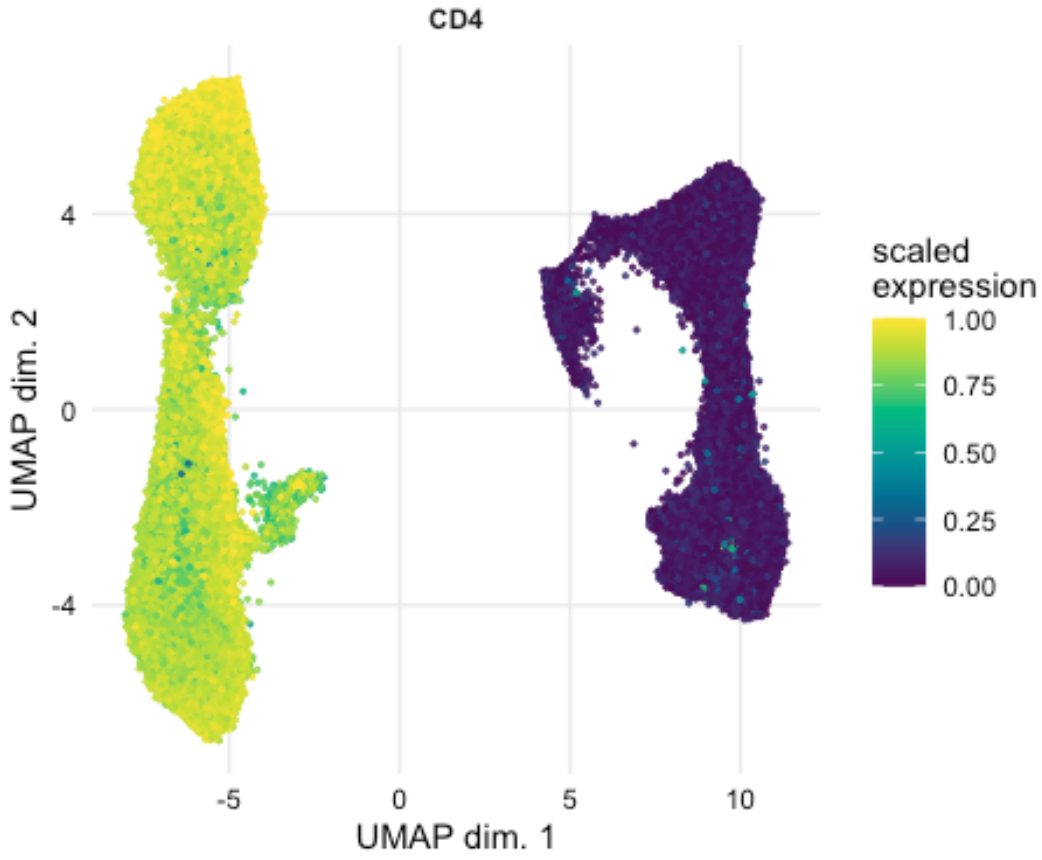
The other samples can now be embedded:

```
reducedDim(sce_train, "UMAP")<- NULL

umap_total <- rbind(umap_train$embedding, umap_test)

sce_total <- cbind(sce_train, sce_test)

reducedDim(sce_total, "UMAP") <- umap_total
```

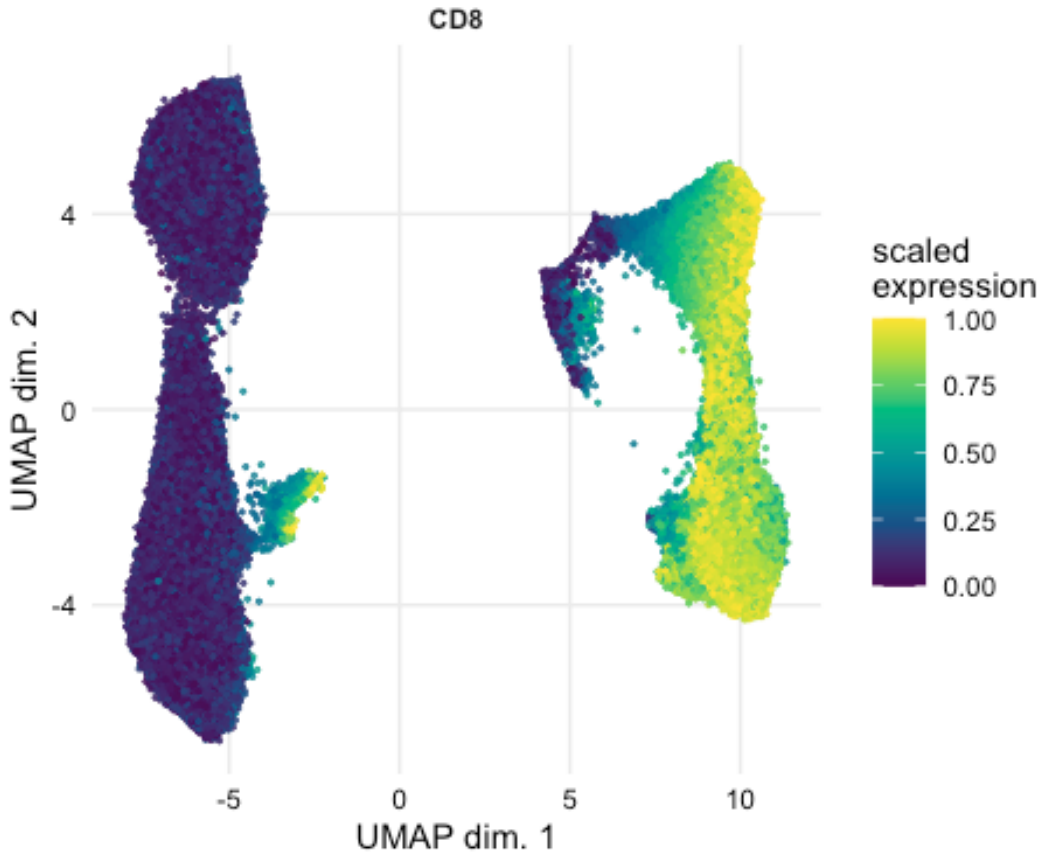
```
plotDR(sce_total, "UMAP", color_by = "sample_id")
```




```
plotDR(sce_total, "UMAP", color_by = "CD4")
```




```
plotDR(sce_total, "UMAP", color_by = "CD8")
```



###

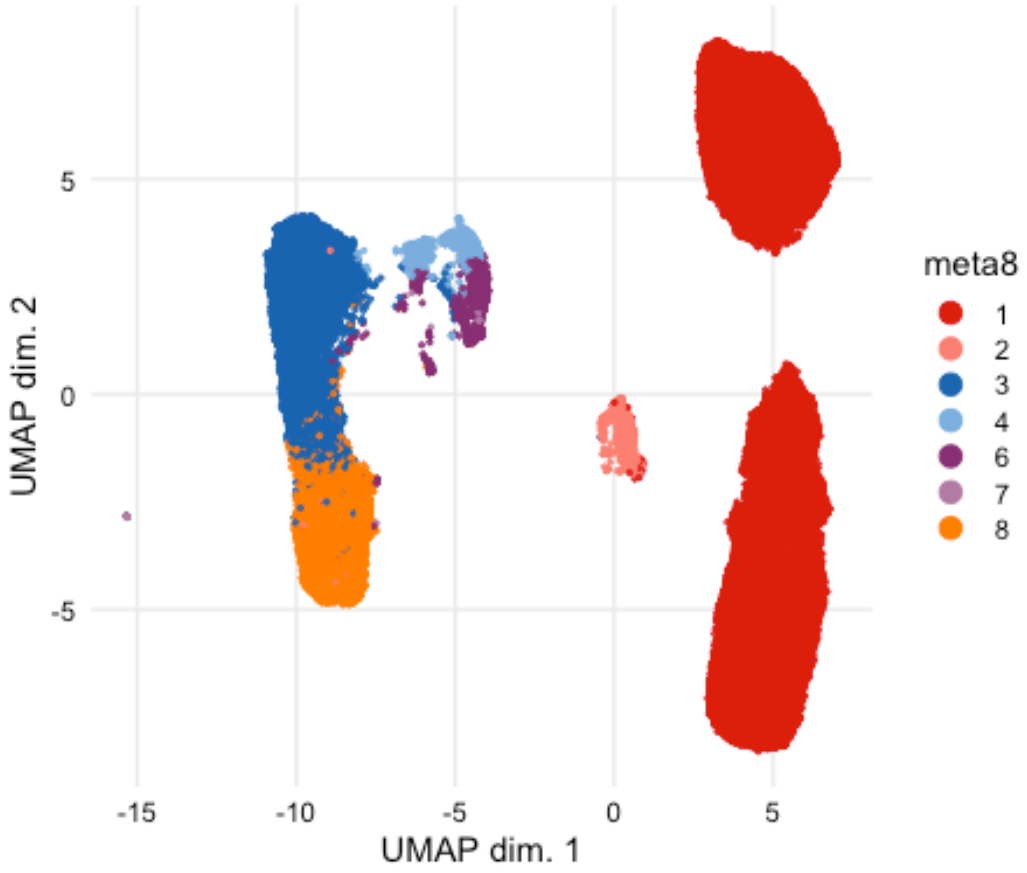
Clustering data

The CATALYST package provides functions to first cluster flow cytometry data with FlowSOM clustering and subsequently perform an UMAP or tSNE with the metacluster labels. Advantage of FlowSOM clustering is the speed of the algorithm and you don't need to downsample or split your dataset if the exploratory phase is over.

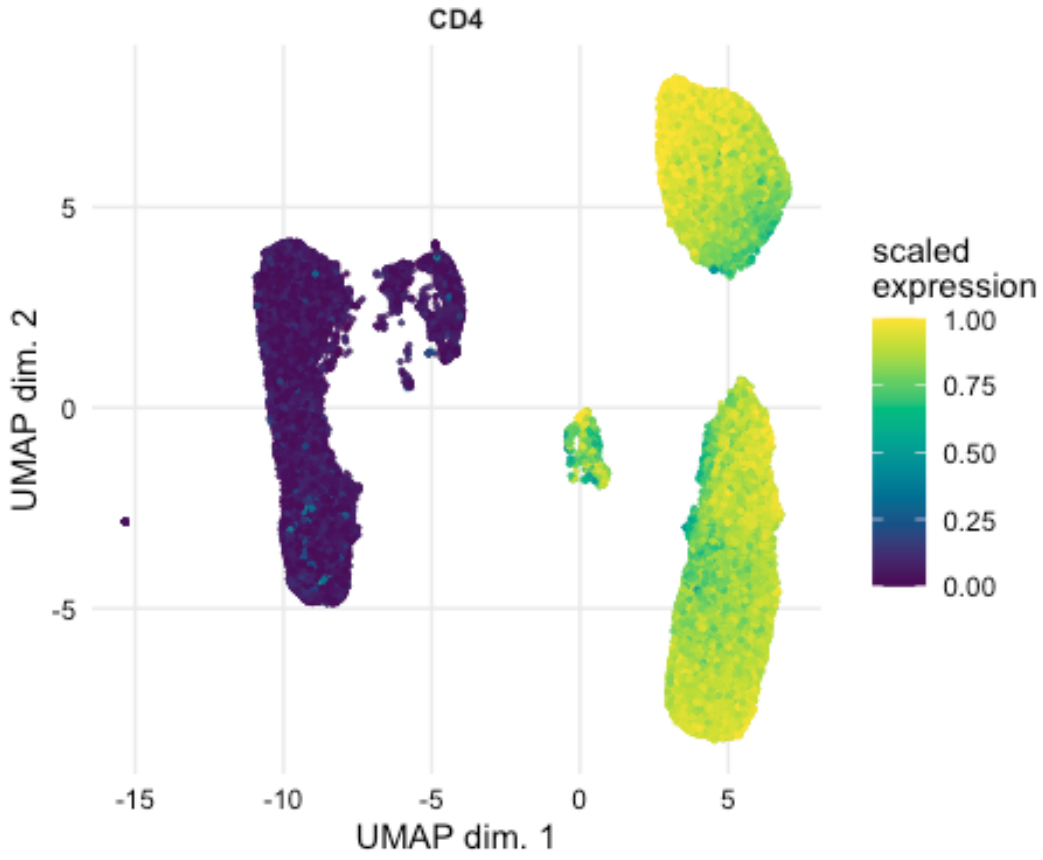
```
set.seed(7460)

sce<- prepData(fcs_transform, md=md_2, panel= panel_B, FACS = TRUE, transform=FALSE,
md_cols =list(file="file_name", id="sample_id", factors=c("batch", "batch_label")))

assayNames(sce)[1] <- "exprs"
sce <- cluster(sce, features="state", maxK=8, seed=7460)
```

With maxK you specify the number of clusters. The number of clusters to choose can be difficult. First, you need to ask yourself how many clusters would you expect in your data.You can also use the UMAP of the earlier steps to guide you in choice for number of clusters. Plotting median expression of markers in that UMAP can help to see the number of populations you would expect. Vary the number of clusters to find what best fits your data and is biological relevant.

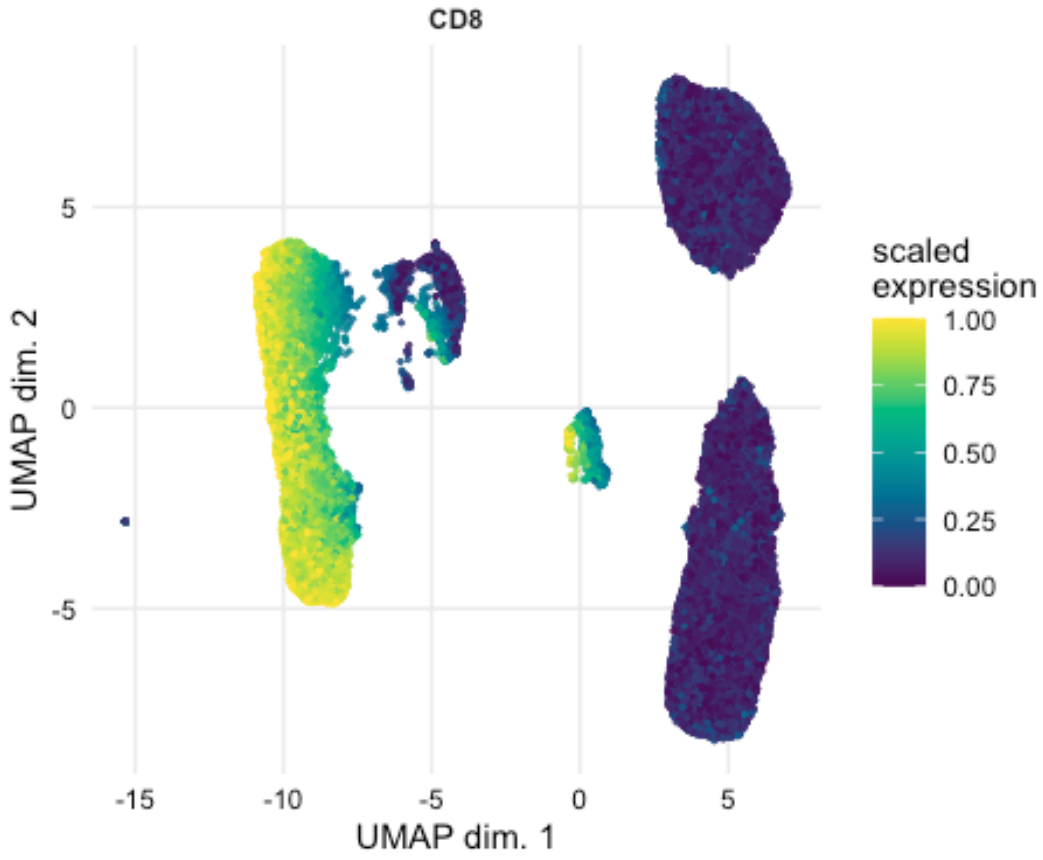
```
sce <- runDR(sce, "UMAP", cells = 6000, features = "state")
plotDR(sce, "UMAP", color_by="meta8")
```



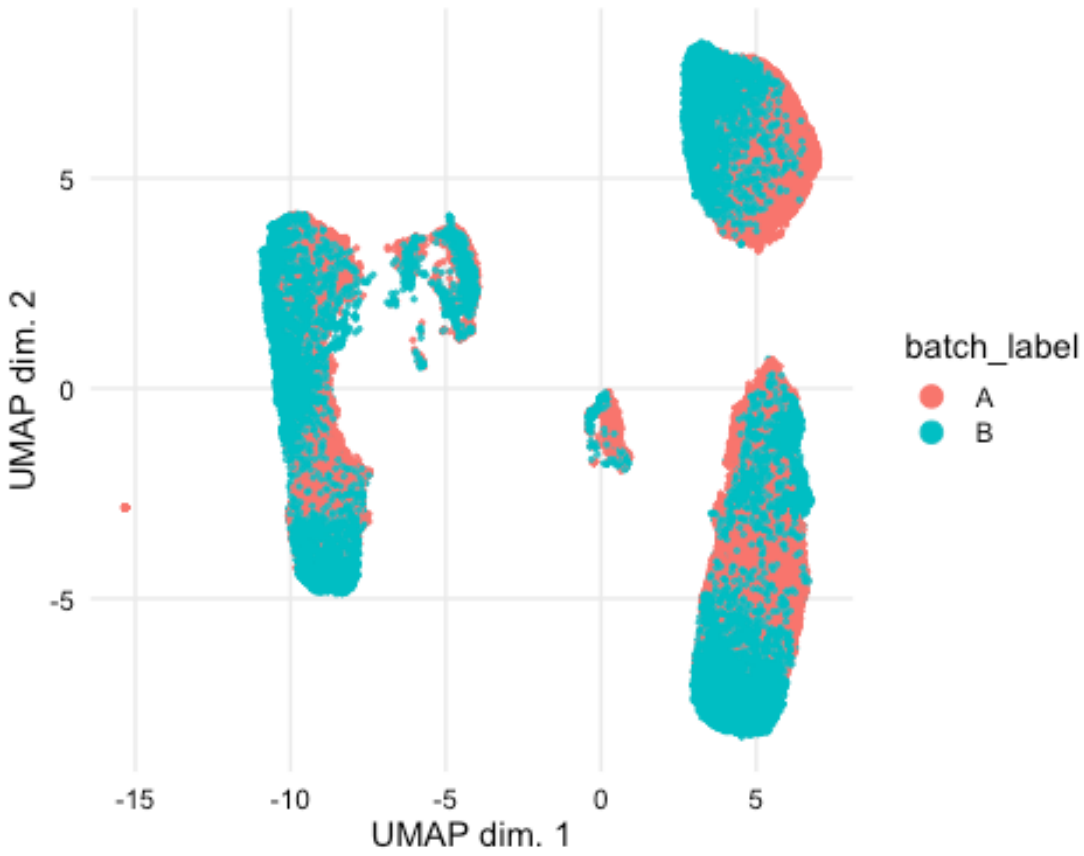

```
plotDR(sce, "UMAP", color_by="CD4")
```



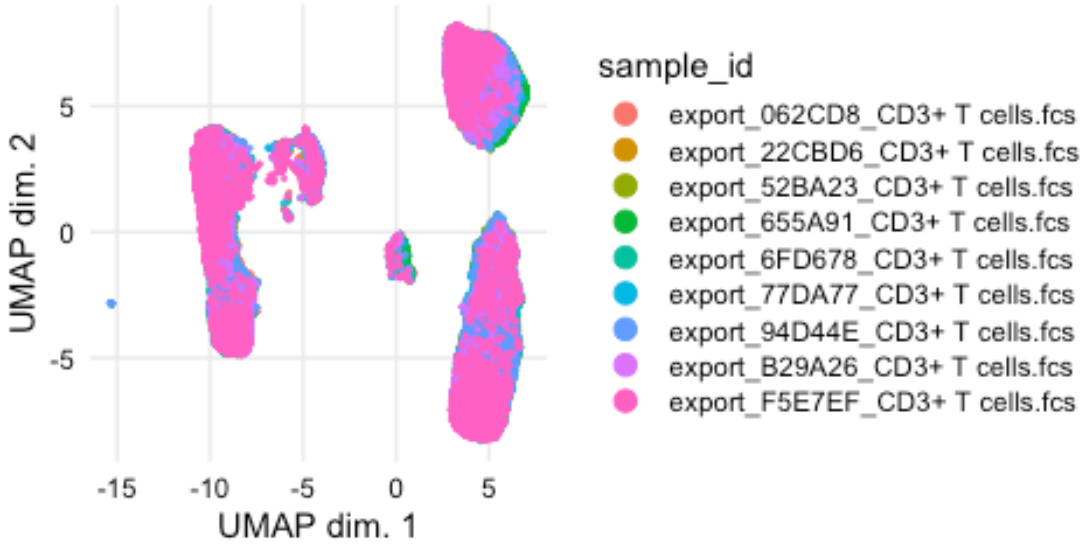

```
plotDR(sce, "UMAP", color_by="CD8")
```



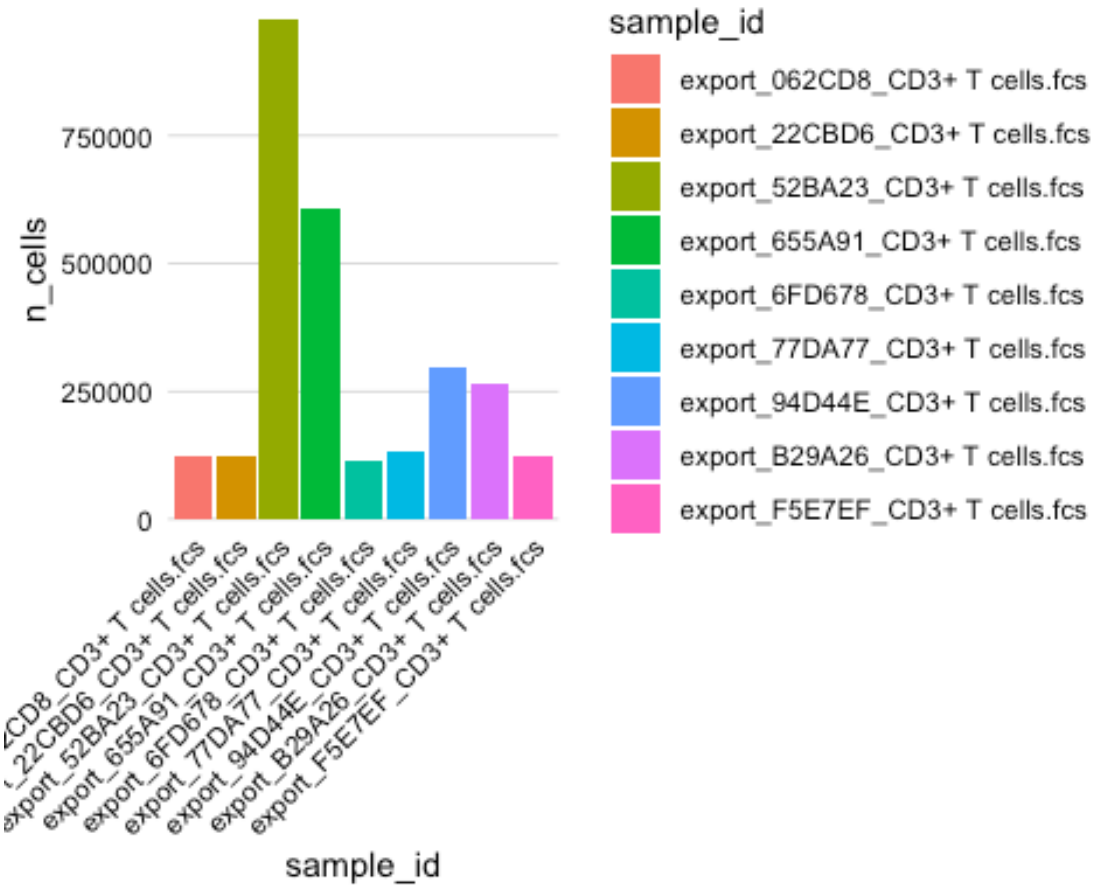

```
plotDR(sce, "UMAP", color_by="batch_label") #to check how batches are divided over the clusters
```



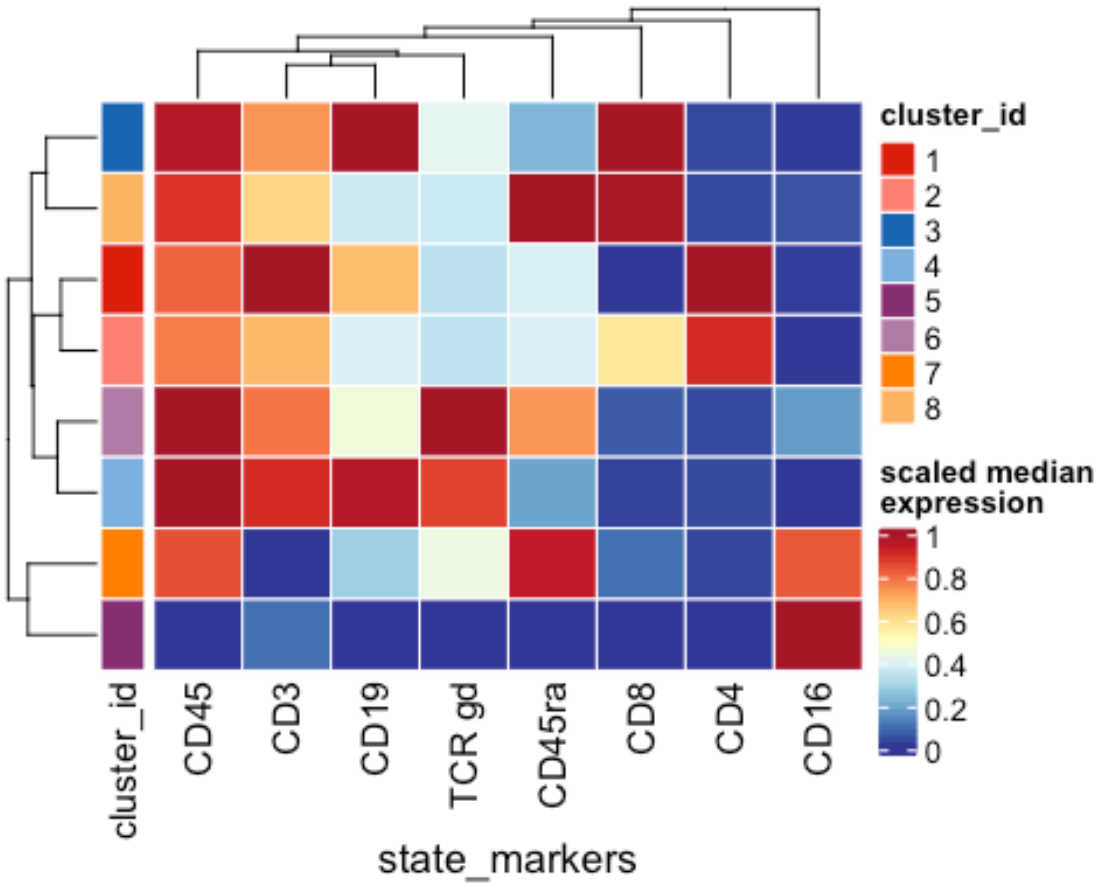

```
plotDR(sce, "UMAP", color_by="sample_id")
```



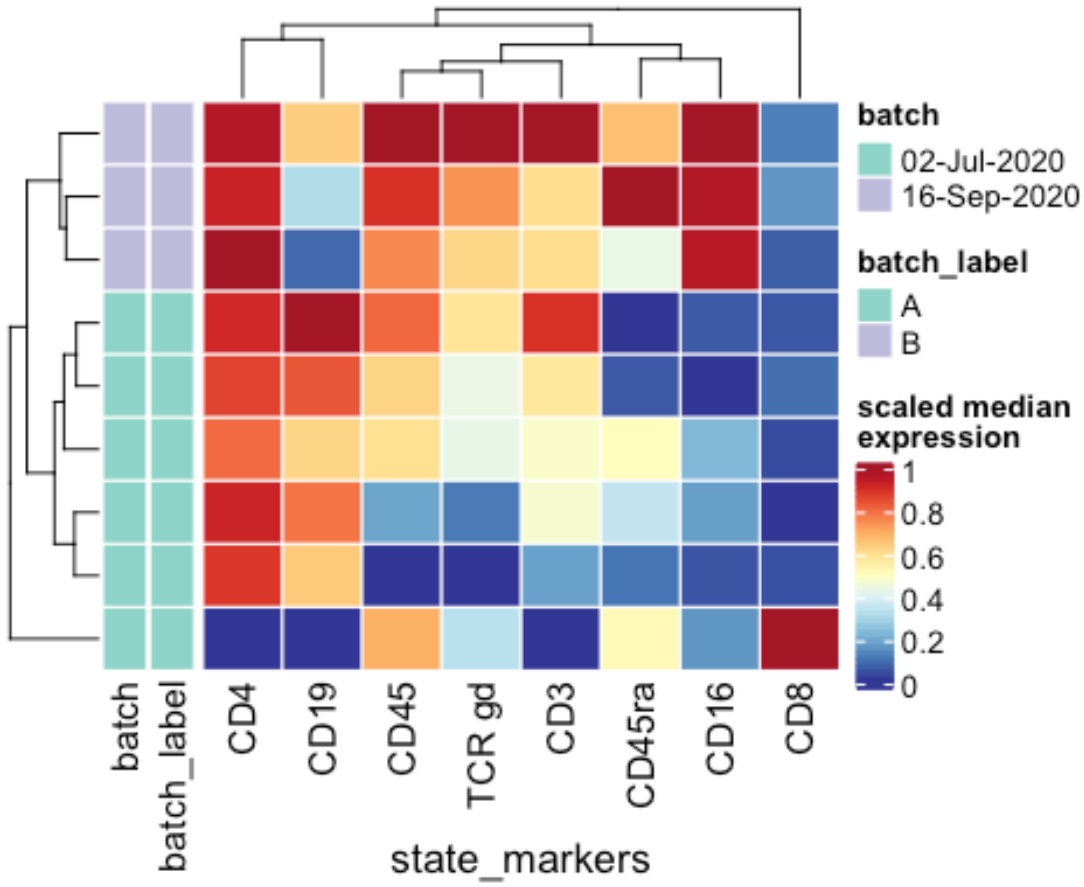

```
#Plot the number of cells per sample
Cell_numbers <-plotCounts(sce, prop=FALSE, group_by = "sample_id")#change prop to TRUE t
o see frequencies
print(Cell_numbers)
```




```
Cell_numbers_data <- Cell_numbers[["data"]] #dataframe of number of cells per sample, could b
e useful if you want to export dataframe and use it to make graphs in other programs, such as G
raphpad Prism

#Heatmap of the median expression per marker per metacluster or sample, more information ca
n be found in https://bioconductor.org/packages/release/bioc/html/CATALYST.html

plotExprHeatmap(sce, features = "state", by="cluster_id",k="meta8",scale = "last", q = 0, perc=T
RUE,bars = FALSE)
```




```
plotExprHeatmap(sce, features = "state", by="sample_id",k="meta8",scale = "last", q = 0, perc=TRUE,bars = FALSE) #this plot can also be used to check for batch effects, the two batches clearly cluster differently with hierarchical clustering
```




```
Cell_freq_clusters <- plotAbundances(sce, k = "meta8", group_by = "sample_id")
print(Cell_freq_clusters)

Cell_freq_clusters_data <- Cell_freq_clusters[["data"]]
write.xlsx(x=Cell_freq_clusters_data, file="Cellclusterfrequencies.xlsx")

sessionInfo()

## R version 4.1.0 (2021-05-18)
## Platform: x86_64-apple-darwin17.0 (64-bit)
## Running under: macOS Mojave 10.14.6
##
## Matrix products: default
## BLAS:   /Library/Frameworks/R.framework/Versions/4.1/Resources/lib/libRblas.dylib
## LAPACK: /Library/Frameworks/R.framework/Versions/4.1/Resources/lib/libRlapack.dylib
##
## locale:
## [1] nl_NL.UTF-8/nl_NL.UTF-8/nl_NL.UTF-8/C/nl_NL.UTF-8/nl_NL.UTF-8
##
## attached base packages:
## [1] parallel  stats4    stats     graphics  grDevices utils     datasets
## [8] methods   base
##
## other attached packages:
```

```
##  [1] xlsx_0.6.5                 knitr_1.33
##  [3] uwot_0.1.10                Matrix_1.3-4
##  [5] CytoNorm_0.0.7             CATALYST_1.16.2
##  [7] SingleCellExperiment_1.14.1 SummarizedExperiment_1.22.0
##  [9] Biobase_2.52.0             GenomicRanges_1.44.0
## [11] GenomeInfoDb_1.28.1        IRanges_2.26.0
## [13] S4Vectors_0.30.0           BiocGenerics_0.38.0
## [15] MatrixGenerics_1.4.0       matrixStats_0.60.0
## [17] PeacoQC_1.2.0              flowAI_1.22.0
## [19] flowVS_1.24.0              flowStats_4.4.0
## [21] flowViz_1.56.0             lattice_0.20-44
## [23] flowCore_2.4.0
##
## loaded via a namespace (and not attached):
##   [1] utf8_1.2.2                ks_1.13.2
##   [3] tidyselect_1.1.1          BiocParallel_1.26.1
##   [5] grid_4.1.0                Rtsne_0.15
##   [7] rainbow_3.6               aws.signature_0.6.0
##   [9] ScaledMatrix_1.0.0        munsell_0.5.0
##  [11] codetools_0.2-18          colorspace_2.0-2
##  [13] highr_0.9                 robustbase_0.93-8
##  [15] ggsignif_0.6.2            rJava_1.0-5
##  [17] labeling_0.4.2            GenomeInfoDbData_1.2.6
##  [19] polyclip_1.10-0           farver_2.1.0
##  [21] pheatmap_1.0.12           flowWorkspace_4.4.0
##  [23] changepoint_2.2.2         vctrs_0.3.8
##  [25] generics_0.1.0            TH.data_1.0-10
##  [27] xfun_0.24                 R6_2.5.0
##  [29] doParallel_1.0.16         ggbeeswarm_0.6.0
##  [31] clue_0.3-59               rsvd_1.0.5
##  [33] bitops_1.0-7              DelayedArray_0.18.0
##  [35] scales_1.1.1              multcomp_1.4-17
##  [37] beeswarm_0.4.0            gtable_0.3.0
##  [39] beachmat_2.8.0            Cairo_1.5-12.2
##  [41] RProtoBufLib_2.4.0        sandwich_3.0-1
##  [43] rlang_0.4.11              GlobalOptions_0.1.2
##  [45] splines_4.1.0             rstatix_0.7.0
##  [47] hexbin_1.28.2             broom_0.7.9
##  [49] yaml_2.2.1                reshape2_1.4.4
##  [51] abind_1.4-5               backports_1.2.1
##  [53] IDPmisc_1.1.20            RBGL_1.68.0
##  [55] tools_4.1.0               ggplot2_3.3.5
##  [57] ellipsis_0.3.2            RColorBrewer_1.1-2
##  [59] ggridges_0.5.3            Rcpp_1.0.7
##  [61] plyr_1.8.6                sparseMatrixStats_1.4.0
##  [63] base64enc_0.1-3           zlibbioc_1.38.0
##  [65] purrr_0.3.4               RCurl_1.98-1.3
##  [67] FlowSOM_2.0.0             ggpubr_0.4.0
##  [69] viridis_0.6.1             GetoptLong_1.0.5
##  [71] cowplot_1.1.1             zoo_1.8-9
##  [73] haven_2.4.1               ggrepel_0.9.1
```

```
##  [75] cluster_2.1.2              colorRamps_2.3
##  [77] fda_5.1.9                  magrittr_2.0.1
##  [79] RSpectra_0.16-0            ncdfFlow_2.38.0
##  [81] data.table_1.14.0          hdrcde_3.4
##  [83] scattermore_0.7            openxlsx_4.2.4
##  [85] circlize_0.4.13            mvtnorm_1.1-2
##  [87] ggnewscale_0.4.5           xlsxjars_0.6.1
##  [89] hms_1.1.0                  evaluate_0.14
##  [91] XML_3.99-0.6               rio_0.5.27
##  [93] jpeg_0.1-9                 mclust_5.4.7
##  [95] readxl_1.3.1               gridExtra_2.3
##  [97] shape_1.4.6                ggcyto_1.20.0
##  [99] compiler_4.1.0             scater_1.20.1
## [101] tibble_3.1.3               KernSmooth_2.23-20
## [103] crayon_1.4.1               ggpointdensity_0.1.0
## [105] fds_1.8                    htmltools_0.5.1.1
## [107] pcaPP_1.9-74               tidyr_1.1.3
## [109] rrcov_1.5-5                RcppParallel_5.1.4
## [111] aws.s3_0.3.21              tweenr_1.0.2
## [113] ComplexHeatmap_2.8.0       MASS_7.3-54
## [115] car_3.0-11                 igraph_1.2.6
## [117] forcats_0.5.1              pkgconfig_2.0.3
## [119] foreign_0.8-81             scuttle_1.2.0
## [121] xml2_1.3.2                 foreach_1.5.1
## [123] vipor_0.4.5                XVector_0.32.0
## [125] drc_3.0-1                  stringr_1.4.0
## [127] digest_0.6.27              RcppAnnoy_0.0.19
## [129] pracma_2.3.3               ConsensusClusterPlus_1.56.0
## [131] graph_1.70.0               rmarkdown_2.9
## [133] cellranger_1.1.0           DelayedMatrixStats_1.14.0
## [135] curl_4.3.2                 gtools_3.9.2
## [137] rjson_0.2.20               lifecycle_1.0.0
## [139] jsonlite_1.7.2             BiocNeighbors_1.10.0
## [141] carData_3.0-4              viridisLite_0.4.0
## [143] fansi_0.5.0                pillar_1.6.2
## [145] httr_1.4.2                 plotrix_3.8-1
## [147] DEoptimR_1.0-9             survival_3.2-11
## [149] glue_1.4.2                 zip_2.2.0
## [151] png_0.1-7                  iterators_1.0.13
## [153] Rgraphviz_2.36.0           ggforce_0.3.3
## [155] stringi_1.7.3              nnls_1.4
## [157] BiocSingular_1.8.1         CytoML_2.4.0
## [159] latticeExtra_0.6-29        dplyr_1.0.7
## [161] cytolib_2.4.0              irlba_2.3.3
```