Working with Cytometry Data in R

Mass Cytometry Course 2019

Benjamin Reisman
MD/PhD Candidate
Vanderbilt University

March 22, 2019

The goal of this talk is to learn how to get started with cytometry in R

In R, things that look hard are easy, but things that look easy are (a little) hard.

- Demonstrate why you might want to work in R
- Overcoming the biggest obstacles to working in R
  - Getting data into R
  - Tidying data into the right format for analysis
- Introduce some advanced analysis techniques
- Provide examples and links to resources for learning more.

How to following along:

- Slides: https://bjreisman.github.io/London2019/cytometryinr.html
- Rmarkdown: https://github.com/bjreisman
  - bjreisman.github.io
    - London2019

Why Use R?

"R is a free software environment for statistical computing and graphics"

Compared to Commercial Flow Cytometry Software, R has the following advantages:

- Reproducible (Data + Code = Figures)
- Flexible (This presentation was created in R!)
- Nice Graphics (ggplot2, rgl)
- Great for analysis pipelines and frequently used workflows
- Newest analysis techniques
- Free!

There are many ways to represent data in R, here are two:

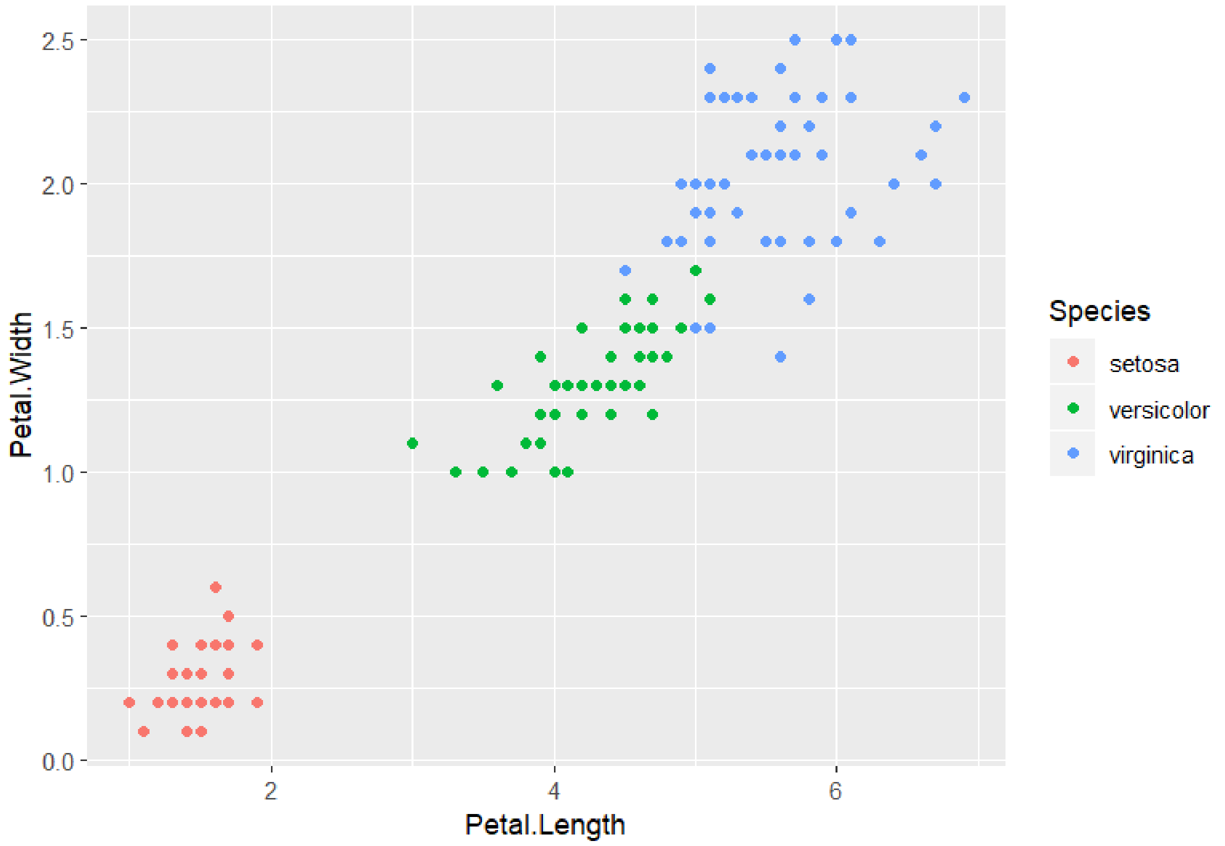
- Matrix: An $n * m$ array of items, all of the single class
- Data Frame: An $n * m$ array of items, but each column can be a different class

Example: The iris dataset: measurements of 50 flowers of 3 species of iris

```
head(iris)
```

```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         0.2  setosa
## 5          5.0         3.6          1.4         0.2  setosa
## 6          5.4         3.9          1.7         0.4  setosa
```

```
library(ggplot2)
ggplot(iris, aes(x= Petal.Length, y = Petal.Width, col = Species)) +
  geom_point()
```



Representing Data in R: Data Frames

Data Frame: An $n * m$ array of items, but each column can be a different class

```
class(iris)
```

```
## [1] "data.frame"
```

```
str(iris)
```

```
## 'data.frame':    150 obs. of  5 variables:
##  $ Sepal.Length: num  5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  $ Sepal.Width : num  3.5 3 3.2 3.1 3.6 3.9 3.4 3.4 2.9 3.1 ...
##  $ Petal.Length: num  1.4 1.4 1.3 1.5 1.4 1.7 1.4 1.5 1.4 1.5 ...
##  $ Petal.Width : num  0.2 0.2 0.2 0.2 0.2 0.4 0.3 0.2 0.2 0.1 ...
##  $ Species     : Factor w/ 3 levels "setosa","versicolor",..: 1 1 1 1 1 1 1 1 1 1 ...
```

Representing Data in R: Matricies

- Matrix: An $n * m$ array of items, all of the single class

```r
#only the numeric columns, 1:4
iris_matrix <- as.matrix(iris[1:4])
head(iris_matrix)
```

```
##      Sepal.Length Sepal.Width Petal.Length Petal.Width
## [1,]          5.1         3.5          1.4         0.2
## [2,]          4.9         3.0          1.4         0.2
## [3,]          4.7         3.2          1.3         0.2
## [4,]          4.6         3.1          1.5         0.2
## [5,]          5.0         3.6          1.4         0.2
## [6,]          5.4         3.9          1.7         0.4
```

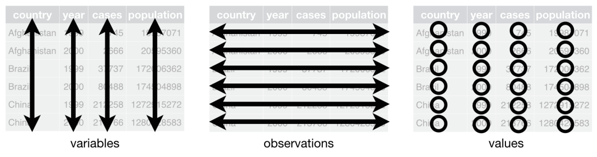
```r
str(iris_matrix)
```

```
##  num [1:150, 1:4] 5.1 4.9 4.7 4.6 5 5.4 4.6 5 4.4 4.9 ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:4] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width"
```

Defining Tidy Data

To work with data in R, it's best to have 'tidy data,' which meets the following criteria:

1. Each variable ['feature'] must have its own column.
2. Each observation ['cell'] must have its own row.
3. Each value must have its own cell ['entry'].



. . . but cytometry data is not usually tidy.

For more information, see: Wickham, Hadley. "Tidy data." *Journal of Statistical Software* 59.10 (2014): 1-23.

Representing Flow Cytometry Data in R

A number of specialized classes have been developed to represent high dimensional bioinformatics data:

- Biocondutor:
  - `SummarizedExperiment` - created to represent genetic data (RNAseq, microarray, etc. . .)
- `flowcore` (RGlab)
  - `FlowFrame` - Representation of an FCS file in R
  - `FlowSet` - Container for multiple FlowFrames + Metadata
- `flowWorkspace` (RGlab)
  - `GatingSet`- A FlowSet + associated gating hierarchy

Representing Flow Cytometry Data in R

A cytometry experiment may include:

- FCS files
- Compensations (FACS)
- Transformations
- Panels
- Gates + Populations
- Metadata

. . . but those aren't neatly represented in R:

Traditional Object	FlowCore Object	R Equivalent
FCS File	FlowFrame	Matrix
Bunch of FCS File	FlowSet	List of matrices + pData
Gated Experiment	Gatingset	-

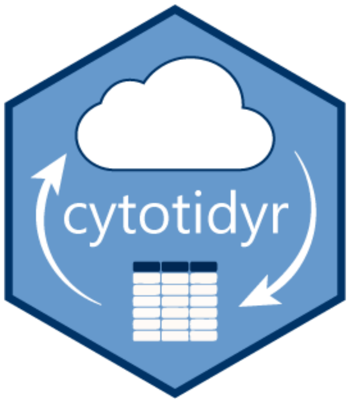
None of these are a data.frame, the most flexible datatype in R

Cytotidyr Helps Bridge the gap between Cytobank and R

Avaliable on github

- Import experiment from cytobank via CytobankAPI
  - `fetchCytobankExperiment`
- Import experiment from cytobank as exported ACS file
  - `parseCytobankExperiment`
  - alternatively, `cytoML::cytobankExperiment`
- Convert a flowSet to a dataframe w/ pData
  - `as.data.frame.flowFrame`

```
#install.packages("devtools")
devtools::install_github("bjreisman/cytotidyr")
```



It's easy to get flow cytometry data into R with the right tools

First we'll need to load a few packages. . .

```
library(CytobankAPI) #connects to cytobank
library(flowWorkspace)#loads flowcore, flowWorkspace
library(CytoML) #Used to read in gating files
library(cytotidyr) #for importing cytobank experiments, and tidying
```

```r
library(dplyr) #for manipulating data
library(tidyr) #for rearranging data from wide to long
library(ggplot2)
```

and find our files…

```r
fcs_paths <- list.files(pattern = ".fcs", recursive = T)
print(fcs_paths)
```

```
## [1] "KCL Guys Data/20180321-01 Group 1 Helios B Post-viSNE_Ungated.fcs.density.fcs.cluster.fcs"
## [2] "KCL Guys Data/20180321-02 Group 2 Helios A Post-viSNE_Ungated.fcs.density.fcs.cluster.fcs"
## [3] "KCL Guys Data/20180321-03 Group 3 Helios A Post-viSNE_Ungated.fcs.density.fcs.cluster.fcs"
## [4] "KCL Guys Data/20180321-04 Group 4 Helios B Post-viSNE_Ungated.fcs.density.fcs.cluster.fcs"
```

Cytotidyr and CytobankAPI can be used to work between Cytobank and R

Using `CytobankAPI` and `Cytotidyr` we'll read in our experiment information from cytobank. This includes:

- gates
- transformations
- panels
- sample tags.

```r
token <- "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJqdGkiOiJhNzE5YzU0MTU0OGM0ZDEzMzI3NjE4MGQzYmM0ZGJmMyIs
cyto_session <- authenticate("vanderbilt", auth_token = token)
experiment.id <- 29958
exp_info <- fetchCytobankExperiment(cyto_session, experiment.id)
```

Reading in the Data

First we'll read in the data as a flowSet

```r
myflowset <- flowCore::read.flowSet(fcs_paths)
```

Then we'll convert it to a gatingSet

```r
mygatingset <- flowWorkspace::GatingSet(myflowset)
```

```
## ....done!
```

Cytometry Preprocessing (Transformations, Gates, Panels) can be done in R

Next we'll:

- rescale the data using the defined asinh transformation for the appropriate channels:
- rename the channels according to our panel
- apply gates to the gatingset
- convert the data back to a flowset

```r
mygatingset <- flowWorkspace::transform(mygatingset, exp_info$transforms)
markernames(mygatingset) <- exp_info$panels$`Panel 1`
CytoML::gating(exp_info$gates, mygatingset)
```

```
## non-beads
```

```
## eventlength_width
## center_width

## residual_width

## DNA

## viable

## B Cells

## T Cells

## CD4s

## CD8s

## ....done!
```

```
mygatingset <- tagFlowSet(mygatingset, exp_info$sampletags)
myflowset_preprocessed <- flowWorkspace::getData(mygatingset, "viable")
```

Cytotidyr allows us to convert the flowset to a tidy data.frame

In order to work with our data using R, we'll need to convert it to a data frame, using the `as.data.frame` function from cytotidyr

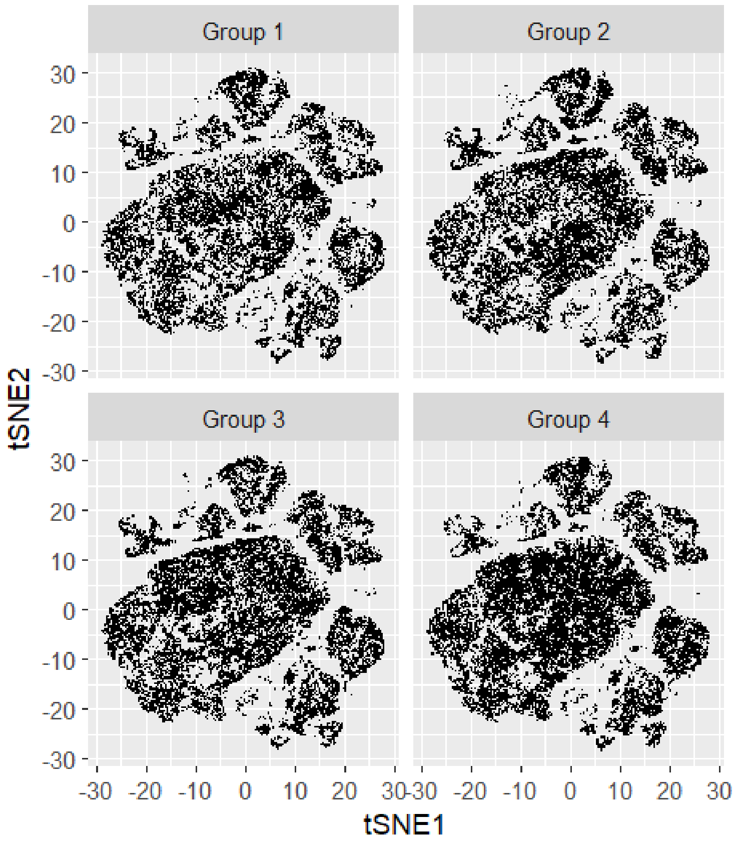
```
mydataframe <- as.data.frame(myflowset_preprocessed, use_longnames = T)
str(mydataframe)
```

```
## 'data.frame':    57684 obs. of  79 variables:
##  $ Time                : num  86.5 96.9 143.5 444.2 544 ...
##  $ Event_length        : num  15 15 15 22 17 19 17 15 19 16 ...
##  $ 89Y_CD45 (v)        : num  3.96 4.38 4.39 3.74 4.21 ...
##  $ 102Pd               : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ 103Rh_Viability (v): num  0 0 0 0 0.464 ...
##  $ 104Pd               : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ 105Pd               : num  0 0 0 0 0 ...
##  $ 106Pd               : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ 108Pd               : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ 110Pd               : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ 113In               : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ 114Cd               : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ 115In               : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ 120Sn               : num  0 0 0.014 0 0 ...
##  $ 127I                : num  0 0 0 0 0 ...
##  $ 131Xe               : num  0 0 0 0 0 ...
##  $ 133Cs               : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ 138Ba               : num  1.2 1.72 1.66 1.84 1.85 ...
##  $ 139La               : num  0 0 0 0 0 ...
##  $ 140Ce_EQ_Beads (v) : num  0.238 0 0 0 0 ...
##  $ 141Pr_CCR6 (v)     : num  0 0 0 0 0 ...
##  $ 142Nd               : num  0.268 0 0.376 0 0 ...
##  $ 143Nd               : num  0.766 0 0 0 0 ...
##  $ 144Nd               : num  0.4031 0.3158 0.0934 0 0.1101 ...
##  $ 145Nd_CD4 (v)      : num  0.65661 0.00594 0.87242 3.19221 3.99948 ...
##  $ 146Nd_CD8 (v)      : num  5.77 2.32 5.62 1.26 0 ...
##  $ 147Sm_CD20 (v)     : num  0 0 0.0314 0 0 ...
##  $ 148Nd_CD16 (v)     : num  0.1981 3.3858 0.4623 0.0712 0 ...
##  $ 149Sm_CCR4 (v)     : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ 150Nd              : num  0 0 0 0 0 ...
##  $ 151Eu              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ 152Sm              : num  0 0 0.0263 0.1204 0.2434 ...
##  $ 153Eu              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ 154Sm_CD3 (v)      : num  4.84 0 5.26 4.47 5.04 ...
##  $ 155Gd_CD45RA (v)   : num  3.299 3.014 4.374 0 0.841 ...
##  $ 156Gd              : num  0 0 0 0 0.115 ...
##  $ 157Gd              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ 158Gd              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ 159Tb_CCR7 (v)     : num  2.34 0 3.22 2.28 3.32 ...
##  $ 160Gd_CD14 (v)     : num  0 0 0.0252 0 0 ...
##  $ 161Dy              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ 162Dy              : num  0.844 0 0.889 0 0 ...
##  $ 163Dy_CXCR3 (v)    : num  0.0603 0 3.2913 0 0.9788 ...
##  $ 164Dy              : num  0 0.415 0 0 0 ...
##  $ 165Ho_CD45RO (v)   : num  0 0 0 0 0 ...
##  $ 166Er              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ 167Er_CD27 (v)     : num  3.09 0 3.72 2.56 3.22 ...
##  $ 168Er              : num  0.0415 0 0.2607 0 0.373 ...
##  $ 169Tm_CD25 (v)     : num  0 0 0 1.845 0.677 ...
##  $ 170Er              : num  0 0 0 0.0508 0 ...
##  $ 171Yb              : num  0 0 0 0 0 ...
##  $ 172Yb_CD57 (v)     : num  0 0 0 0 0 ...
##  $ 173Yb              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ 174Yb_HLA-DR (v)   : num  0 0.314 0.51 0 0 ...
##  $ 175Lu              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ 176Yb_CD127 (v)    : num  1.81 0 2.26 0.7 1.96 ...
##  $ 177Hf              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ 190BCKG            : num  0 0 0.152 0 0 ...
##  $ 191Ir_DNA (v)      : num  6.06 6.31 6.33 6.21 6.15 ...
##  $ 193Ir              : num  6.67 6.89 6.93 6.73 6.74 ...
##  $ 194Pt              : num  1.418 1.301 0.744 0.742 0.499 ...
##  $ 195Pt              : num  0 0 0 0 0.419 ...
##  $ 196Pt              : num  0 0.53 0 0 0 ...
##  $ 198Pt              : num  0 0.143 0 0 0 ...
##  $ 207Pb              : num  0 0 0 1.14 0 ...
##  $ 208Pb              : num  0.212 0 0 0.591 0 ...
##  $ 209Bi              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Center             : num  5.37 5.48 5.32 5.84 5.46 ...
##  $ Offset             : num  3.53 3.74 3.54 3.73 3.61 ...
##  $ Width              : num  2.71 2.93 2.73 3.29 2.93 ...
##  $ Residual           : num  3.23 3.17 3.36 3.32 3.4 ...
##  $ tSNE1              : num  24.4 15.83 19.45 -7.96 -13.08 ...
##  $ tSNE2              : num  -7.749 16.926 0.364 -5.673 -3.741 ...
##  $ density            : num  7.63 6.83 6.57 8.48 8.47 ...
##  $ cluster            : num  45 57 53 52 67 5 128 185 45 5 ...
##  $ FCS Filename       : chr  "20180321-04 Group 4 Helios B Post-viSNE_Ungated.fcs.density.fcs.cluste
##  $ Individuals        : Factor w/ 4 levels "Group 1","Group 2",..: 4 4 4 4 4 4 4 4 4 4 ...
##  $ Plate              : Factor w/ 1 level "Plate 1": 1 1 1 1 1 1 1 1 1 1 ...
##  $ FCS.File.Category  : Factor w/ 1 level "Experiment Files": 1 1 1 1 1 1 1 1 1 1 ...
```

Making Cytometry Figures in R (1)

One thing we may want to do is reproduce the same t-SNE figure we made on cytobank:

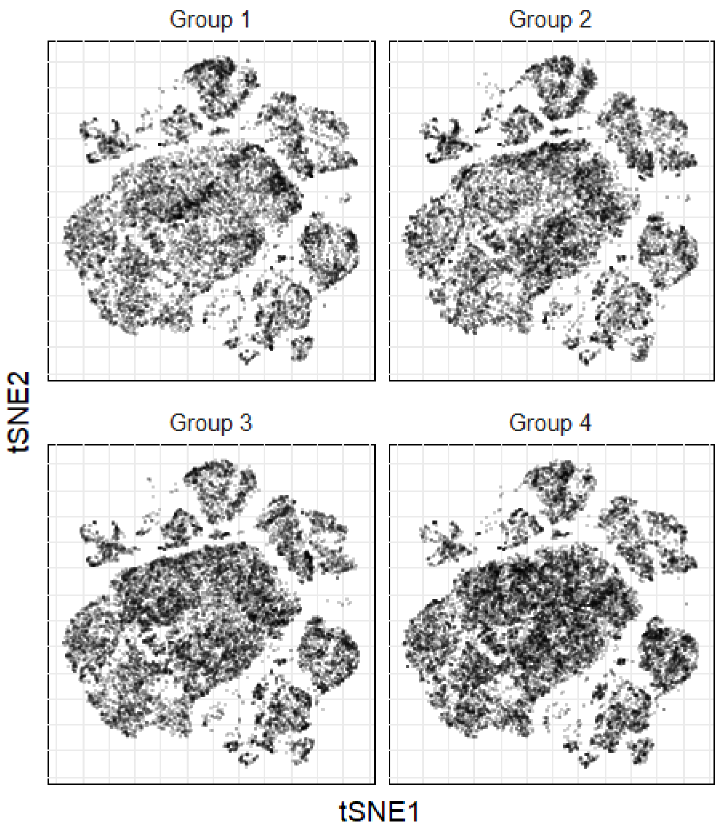
```r
ggplot(mydataframe, aes(x = tSNE1, y = tSNE2)) +
  geom_point(shape = ".") +
  coord_fixed() +
  facet_wrap(~Individuals)
```



Making Cytometry Figures in R (2)

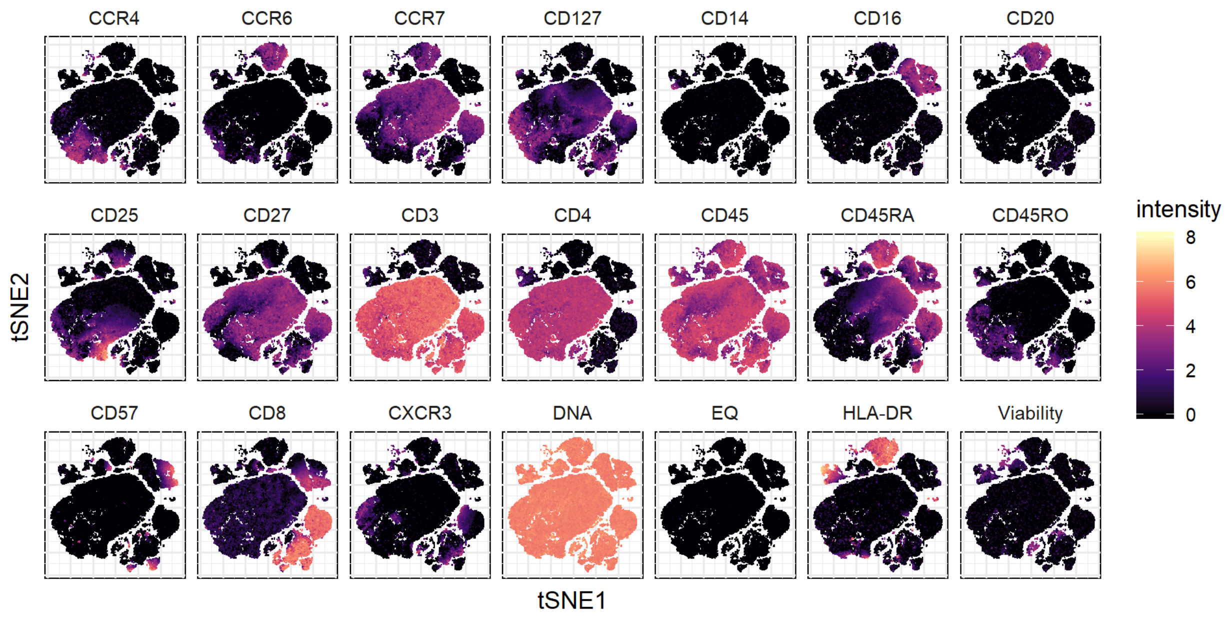
We can also customize our plots in ways that are not easy to do in cytobank:

```r
ggplot(mydataframe, aes(x = tSNE1, y = tSNE2)) +
  geom_point(shape = 16, alpha = 0.2, size = 0.2) +
  coord_fixed() +
  facet_wrap(~Individuals) +
  theme_minimal() +
  theme(axis.text = element_blank(),
        panel.background = element_rect(color = "black", fill = NA))
```



Making Cytometry Figures in R (3)

We may also want to plot multiple channels in the same plot with faceting



- Two differences between this plot compared and the last plot:
  - Marker intensity is mapped to color
  - Markers are faceted across multiple subplots
- In our current [wide] data.frame, intensity is spread across multiple columns
- The plot we want to make requires a [long] data.frame with a single column for intensity + a new column for markers
- We'll need to 'tidy' the data to the right format for our plot.

```
dim((mydataframe))
```

```
## [1] 57684    79
```

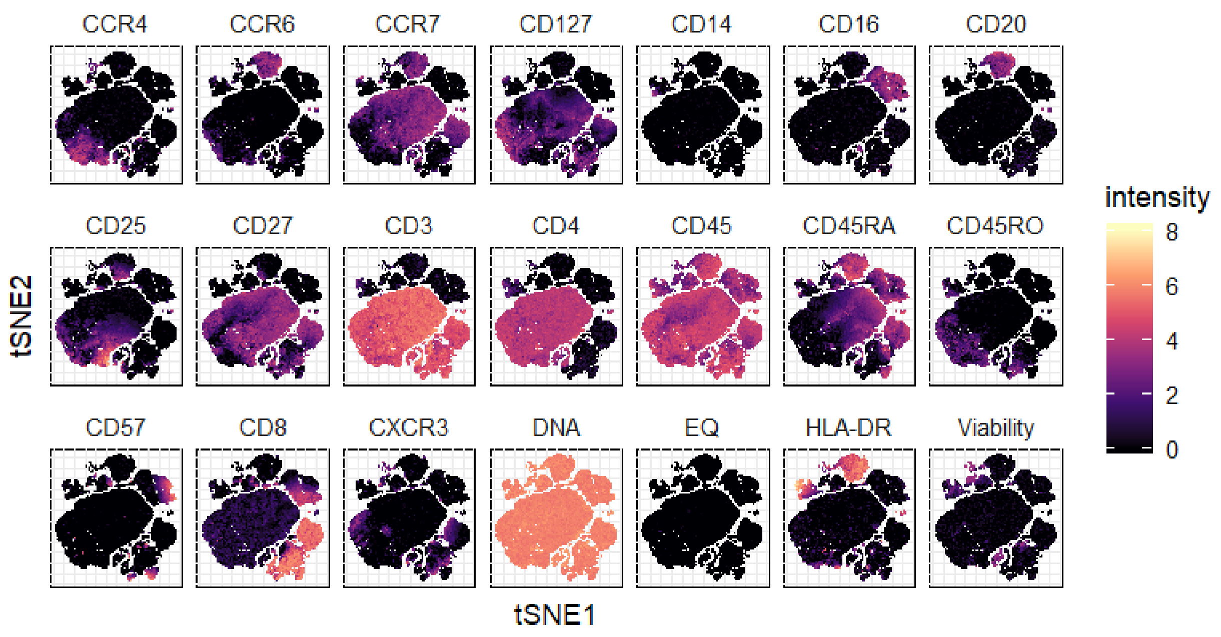
```
element2 <- function(x){unlist(lapply(strsplit(x, split = "_|\ "),"[[", 2))}
mydataframe.long <- mydataframe %>%
  as_tibble() %>%
  gather(marker, intensity, contains("(V)")) %>% # <- this is the key step
  separate(marker, c("channel", "marker", "drop"), sep= "_|\ ") %>%
  as_tibble()
dim((mydataframe.long))
```

```
## [1] 1211364      62
```

Making Cytometry Figures in R (4)

Then we'll make our plot:

```r
mydataframe.long %>%
  ggplot(aes(x = tSNE1, y = tSNE2, col = intensity)) +
  geom_point(shape = ".") +
  scale_colour_viridis_c(option = "A") +
  coord_fixed() +
  facet_wrap(~marker, nrow = 3) +
  theme_minimal() +
  theme(axis.text = element_blank(),
        panel.background = element_rect(color = "black", fill = NA))
```



Applying alternative dimensionality reduction techniques (1)

One of the advantages of R is that we're not limited to the dimensionality reduction techniques that are included in commercial packages.

- Ex: Uniform Manifold Approximation and Projection (UMAP)
  - McInnes L. et al. arXiv, 2018
  - Becht, E, et al., Nature Biotechnology 2018
- Advantages of UMAP vs. t-SNE
  - Faster (minutes vs. hours)
  - Scalable ($o(n)$ vs. $o(n * log(n))$)
  - Preserves local+global structure
  - Other nice features (embedding new points, supervised learning, etc…)

Applying alternative dimensionality reduction techniques (2)

First we'll need to create a separate matrix containing the columns we want to be included in the dimensionality reduction.

```
mymatrix <- mydataframe %>%
  select(contains("(V)")) %>%
  as.matrix()
```

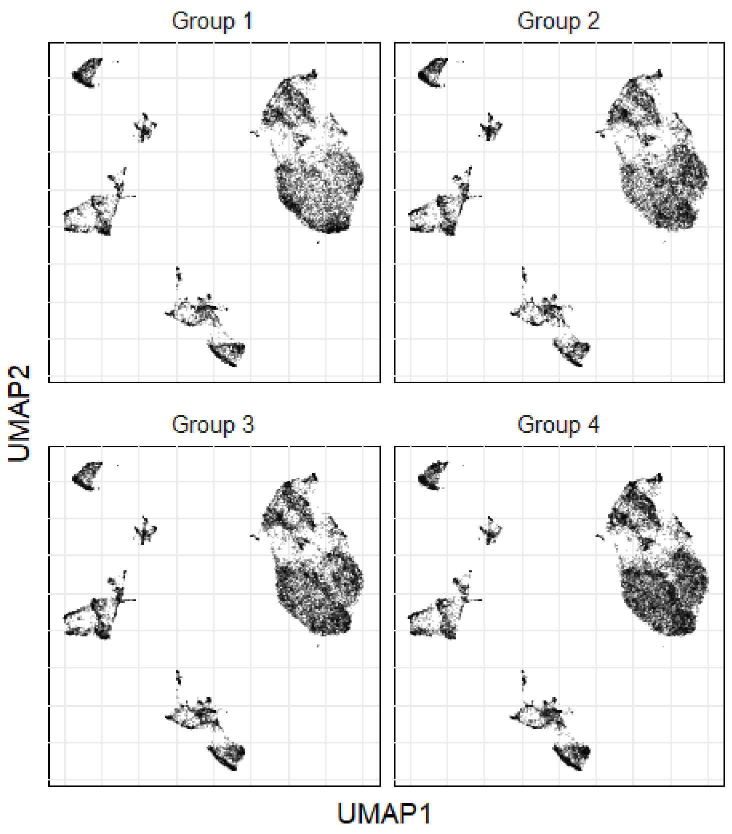
Then we'll run it through the uwot implementation of UMAP

```
#install.packages("devtools")
#devtools::install_github("jlmelville/uwot")
library(uwot)
myumap <- umap(mymatrix, init = "PCA")
str(myumap)
```

```
##  num [1:57684, 1:2] -1.783 -10.074 -0.473 4.165 3.367 ...
##  - attr(*, "scaled:center")= num [1:2] 0.0466 -0.143
```

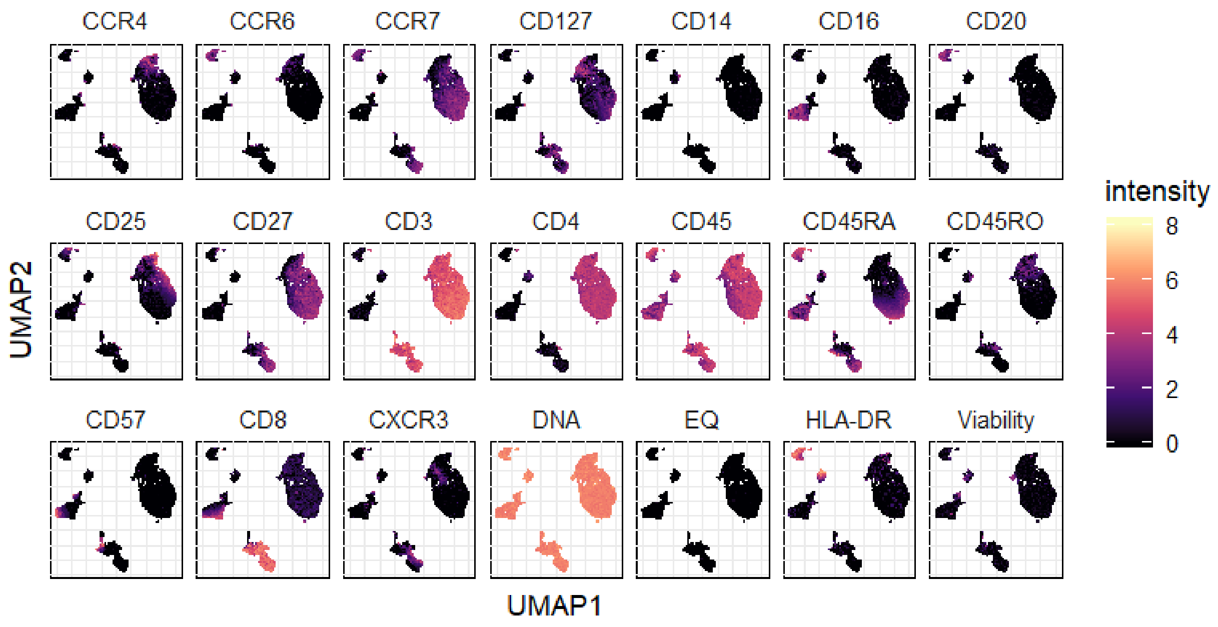
Applying alternative dimensionality reduction techniques (3)

Next, we'll rejoin the two new UMAP columns to our original dataframe, and make our plot:



Applying alternative dimensionality reduction techniques (4)

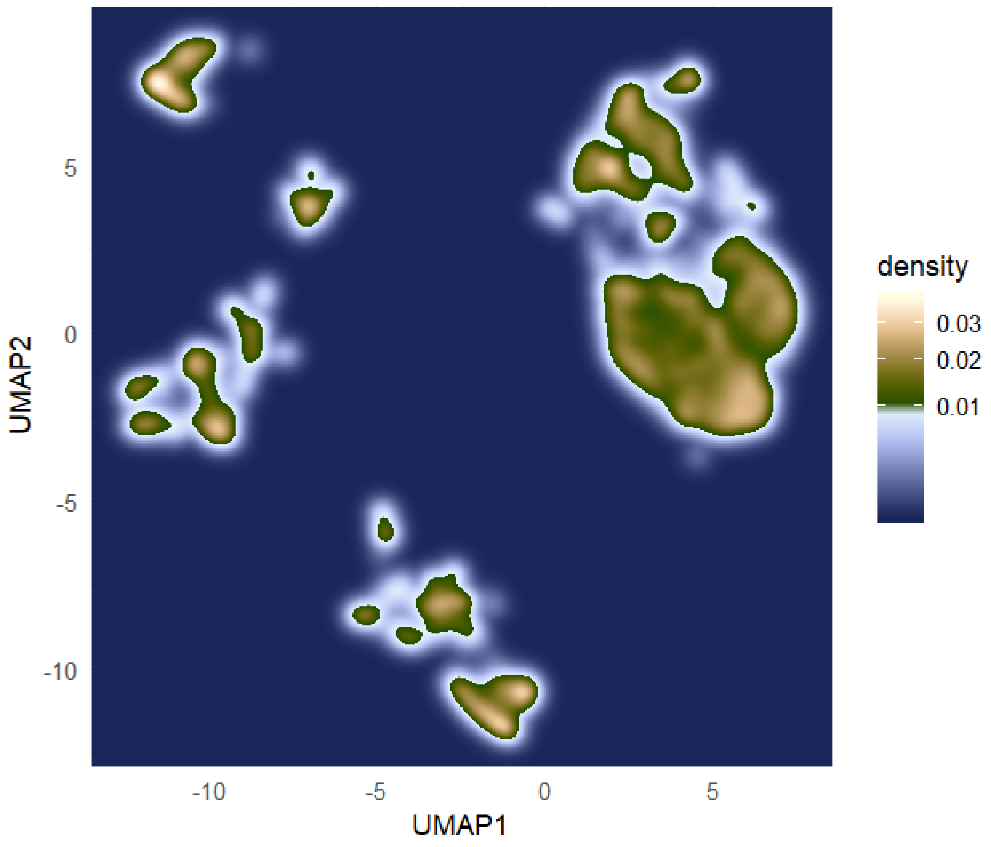
```r
mydataframe %>%
  bind_cols(as.data.frame(myumap)) %>%
  as_tibble() %>%
  gather(marker, intensity, contains("(V)")) %>% # <- this is the key step
  separate(marker, c("channel", "marker", "drop"), sep= "_|\ ") %>%
  ggplot(aes(x = UMAP1, y = UMAP2, col = intensity)) +
  geom_point(shape = ".") +
  scale_colour_viridis_c(option = "A") +
  coord_fixed() +
  facet_wrap(~marker, nrow = 3) +
  theme_minimal() +
  theme(axis.text = element_blank(),
        panel.background = element_rect(color = "black", fill = NA))
```



Applying alternative dimensionality reduction techniques (5)

We can also plot our data as a map:

```r
library(scico)
axis.max <- apply(myumap, 2, max) + 1
axis.min <- apply(myumap, 2, min) - 1

mydataframe %>%
  bind_cols(as.data.frame(myumap)) %>%
  ggplot(aes(x=UMAP1, y = UMAP2)) +
  stat_density_2d(h = c(1, 1),
                  n = 1024,
                  geom = "raster",
                  contour = F,
                  aes(fill = stat(density))) +
  scale_fill_scico(palette = "oleron", name = "density", trans = "sqrt") +
  scale_x_continuous(expand = c(0,0), limits = c(axis.min[1], axis.max[1])) +
  scale_y_continuous(expand = c(0,0), limits = c(axis.min[2], axis.max[2])) +
  theme_minimal() +
  coord_fixed()
```



Clustering in R

We can also apply a clustering algorithm to our dimensionality reduced data.

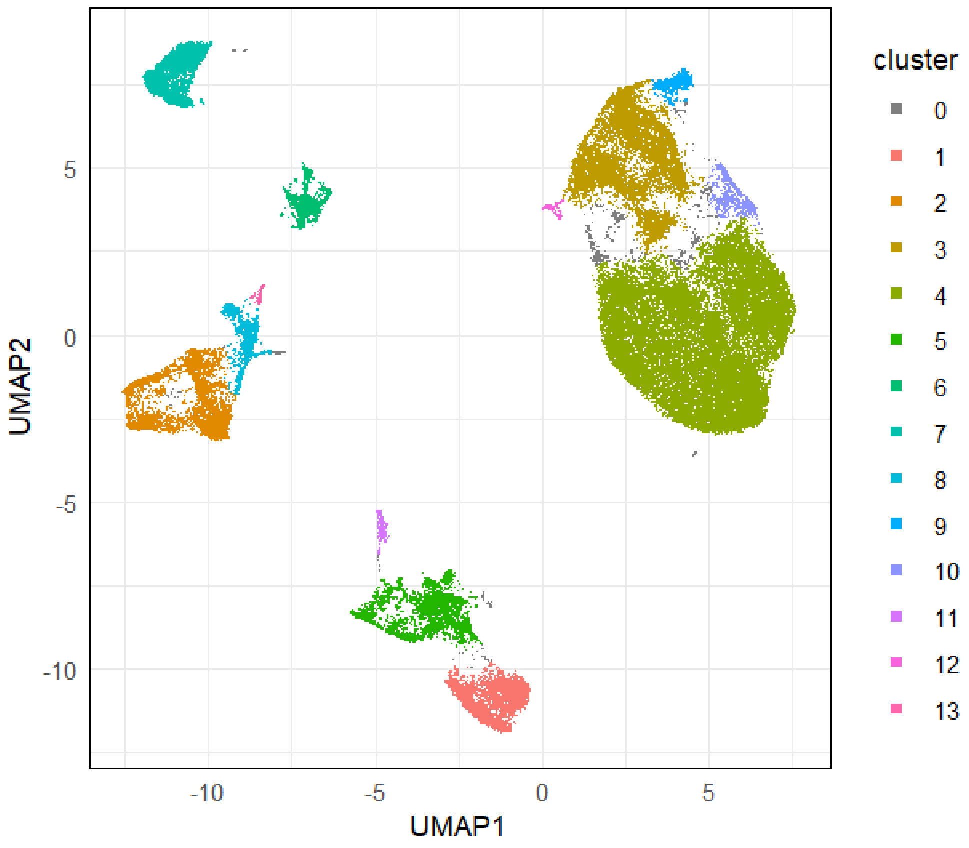
- Density-based spatial clustering of applications with noise (DBSCAN)
    - No need to specify number of clusters
    - Few parameters to tune (eps and minPts)
    - Fast + salable

```r
#install.packages("dbscan")
library(dbscan)
library(scales)
```

```r
mydbscan <- dbscan(myumap, eps = 0.35, minPts = 150)
mydbscan
```

```
## DBSCAN clustering for 57684 objects.
## Parameters: eps = 0.35, minPts = 150
## The clustering contains 13 cluster(s) and 600 noise points.
##
##     0     1     2     3     4     5     6     7     8     9    10    11
##   600  4268  5111  9200 24249  4325  1770  3994  1500   933   788   541
##    12    13
##   220   185
##
## Available fields: cluster, eps, minPts
```

```r
#this finds the number of clusters and manually defines the palette
#such that the outlier cluster is "grey50"
nclust <- max(unique(mydbscan$cluster))
mypalette <- c("grey50", hue_pal()(nclust))

mydataframe %>%
  bind_cols(as.data.frame(myumap)) %>%
  mutate(cluster = as.factor(mydbscan$cluster)) %>%
  ggplot(aes(x=UMAP1, y = UMAP2, col = cluster)) +
  geom_point(shape = ".") +
  scale_colour_manual(guide = guide_legend(override.aes = list(shape = 15)),
                      values = mypalette) +
  coord_fixed() +
  theme_minimal() +
  theme(panel.border = element_rect(color = 'black', fill = NA))
```

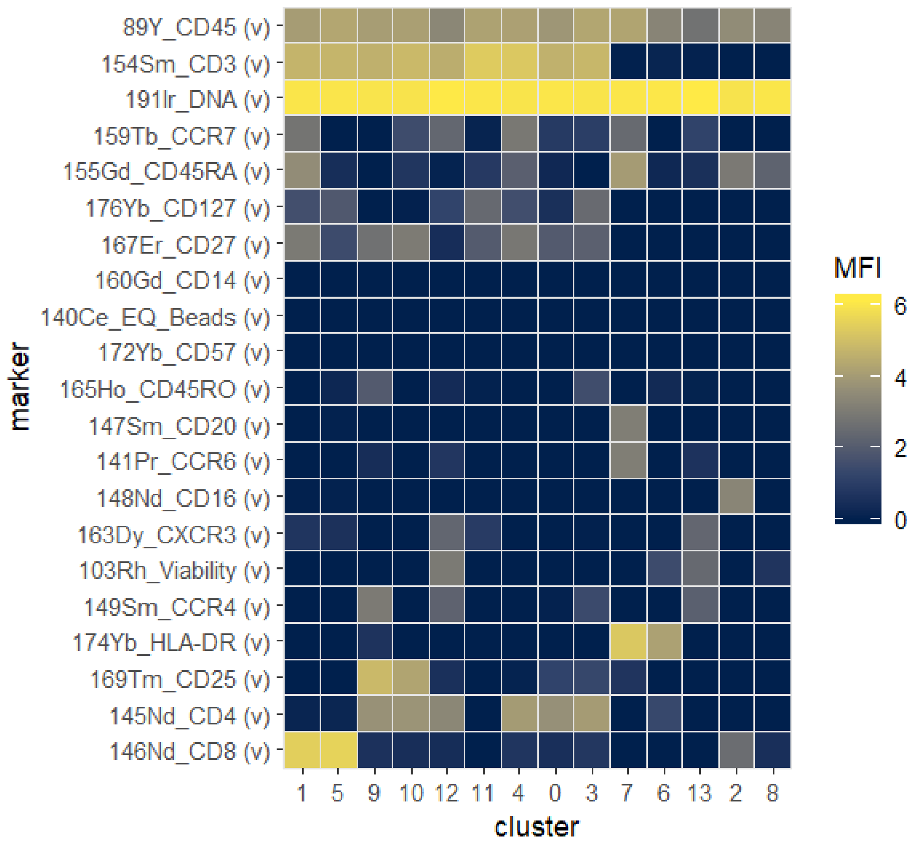


Understanding our clusters

We have clusters, but how can we understand what makes them distinct?

- Marker Enrichment Modeling (MEM)
- Heatmaps (see example below)

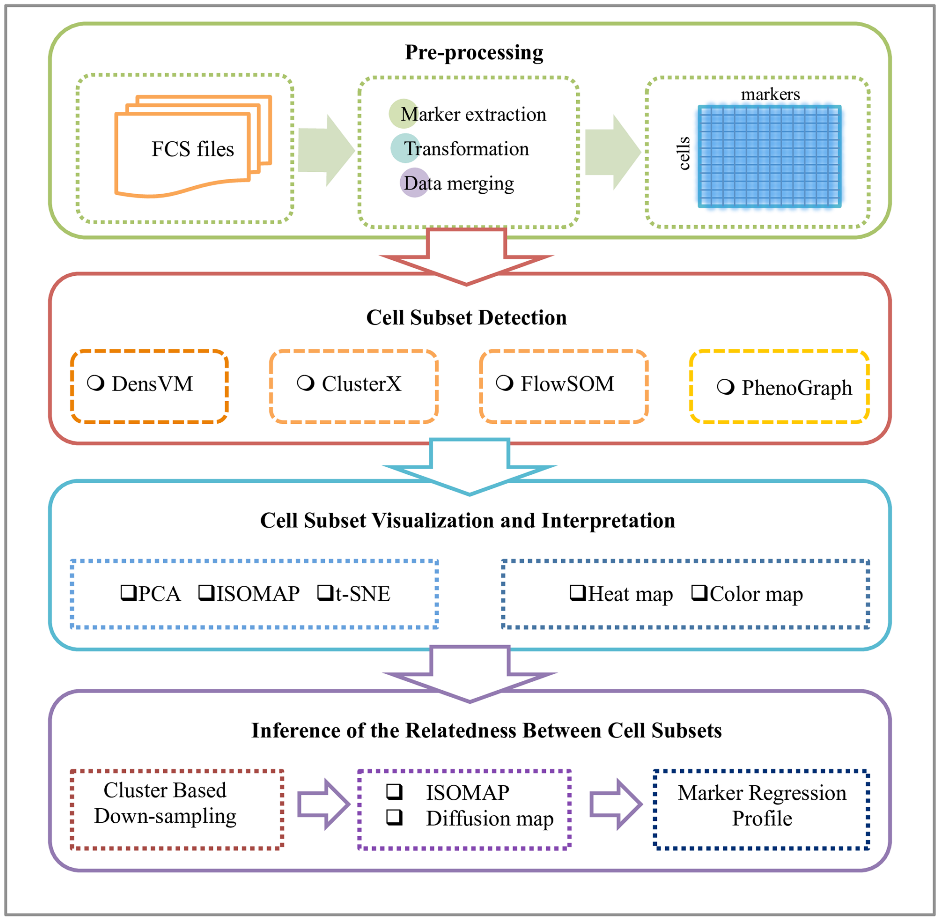
```
library(tibble)
library(scico)
library(seriation)

myheatmap <- mydataframe %>%
  bind_cols(as.data.frame(myumap)) %>%
  mutate(cluster = as.factor(mydbscan$cluster)) %>%
  gather(marker, intensity, contains("(V)")) %>% # <- this is the key step
  group_by(cluster, marker) %>%
  summarise(MFI = median(intensity)) %>%
  select(marker, MFI, cluster) %>%
  spread(marker, MFI)

myheatmap.mat <- myheatmap %>%
  ungroup() %>%
  column_to_rownames("cluster") %>%
  as.matrix()

matrix.dist.row <- dist((myheatmap.mat))
matrix.dist.col <- dist(t(myheatmap.mat))
row.order <- seriation::get_order(seriate(matrix.dist.row, method = "HC"))
col.order <- seriation::get_order(seriate(matrix.dist.col, method = "HC"))

myheatmap %>%
  ungroup() %>%
  gather(marker, MFI, -cluster) %>%
  mutate(marker = factor(marker, levels = colnames(myheatmap.mat)[(col.order)])) %>%
  mutate(cluster = factor(cluster, levels = rownames(myheatmap.mat)[(row.order)])) %>%
  ggplot(aes(x=cluster, y = marker,fill = MFI)) +
  geom_tile(colour = "grey90", size = 0.5) +
  scale_fill_viridis_c(option = "E") +
  scale_x_discrete(expand = c(0,0)) +
  scale_y_discrete(expand = c(0,0)) +
  coord_fixed()
```



What else is out there (1)

CyTOFkit

- Chen, H, et al., *PLOS Computational Biology* 2016
- Integrated pipeline for analyzing cytometry data in R
- Not currently maintained…



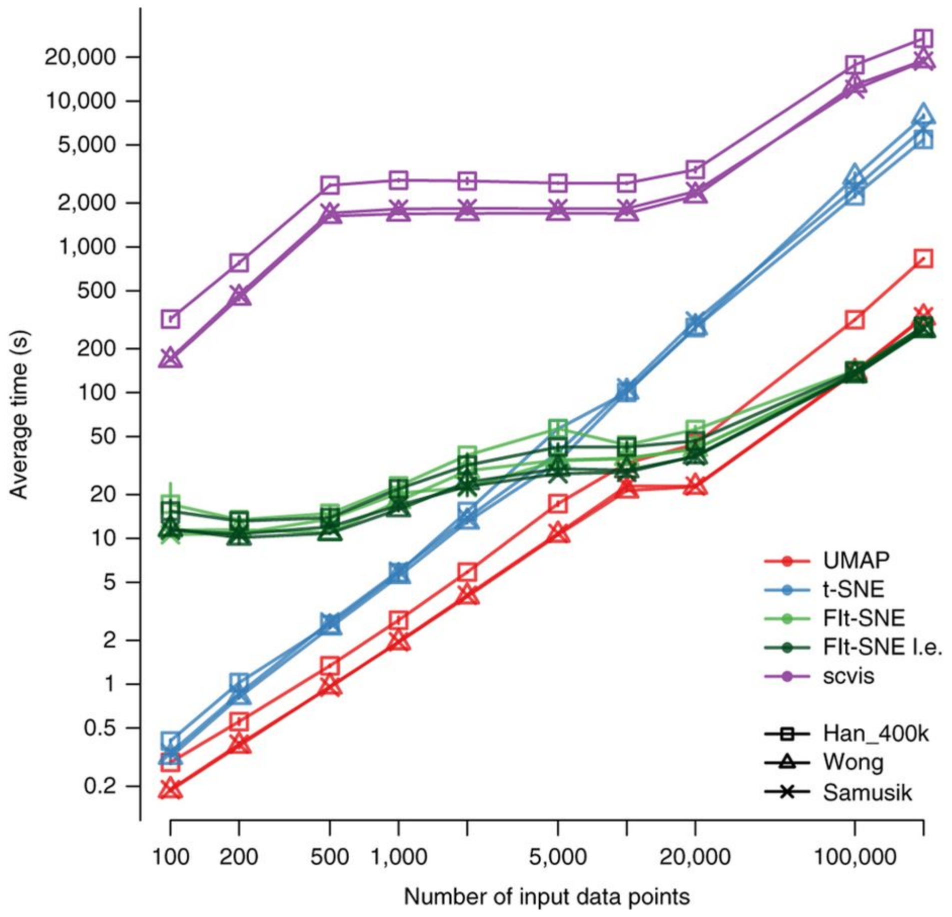
CytoRSuite

- Set of interactive tools that integrate with flowWorkspace
- *Interactive gating*, compensations, panels, etc…

What else is out there (2)

FIt-SNE

- Fast interpolation tSNE
- Linderman, GC, et al., Nature Methods 2019
- Faster than BH-tSNE with over 5K points and scales as $f(n)$ vs. $f(n * log(n))$



- (Becht, E, et al., Nature Biotechnology 2018)

Resources for Learning More

- Datacamp
  - Intro course free
  - Advanced courses $25 per month
  - 2 months free with microsoft visual studio dev essentials
- R for Data Science - Hadley Wickham
- Github (London 2019)
  - All slides from this presentation in rmarkdown format.
- RGLab Github
  - See vignettes for FlowCore, FlowWorkspace, CytoML

Acknowledgements

- Kings College London
  - Susanne Heck, PhD
- Irish Lab
  - Jonathan Irish, PhD
  - Sierra Barrone
  - Todd Bartkowiak, PhD
  - Madeline J. Hayes

  - Caroline. E Roe
- Ferrell Lab
  - P. Brent Ferrell, MD
  - Katie Ivy

- Vanderbilt Laboratory for Biosynthetic Studies
  - Brian Bachmann, PhD
- Thesis Committee
  - Jeffery Rathmell, PhD
  - Brian Bachmann, PhD
  - Lawrence Marnett, PhD
  - Vito Quaranta, MD
  - Jonathan Irish, PhD
- Funding
  - F30CA236131
  - R01CA226833
  - R01GM092218