Working with Cytometry Data in R

Mass Cytometry Course 2019

Benjamin Reisman

March 22, 2019

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)
- Great for analysis pipelines and frequently used workflows
- Newest analysis techniques
- Free!

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.

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)
```

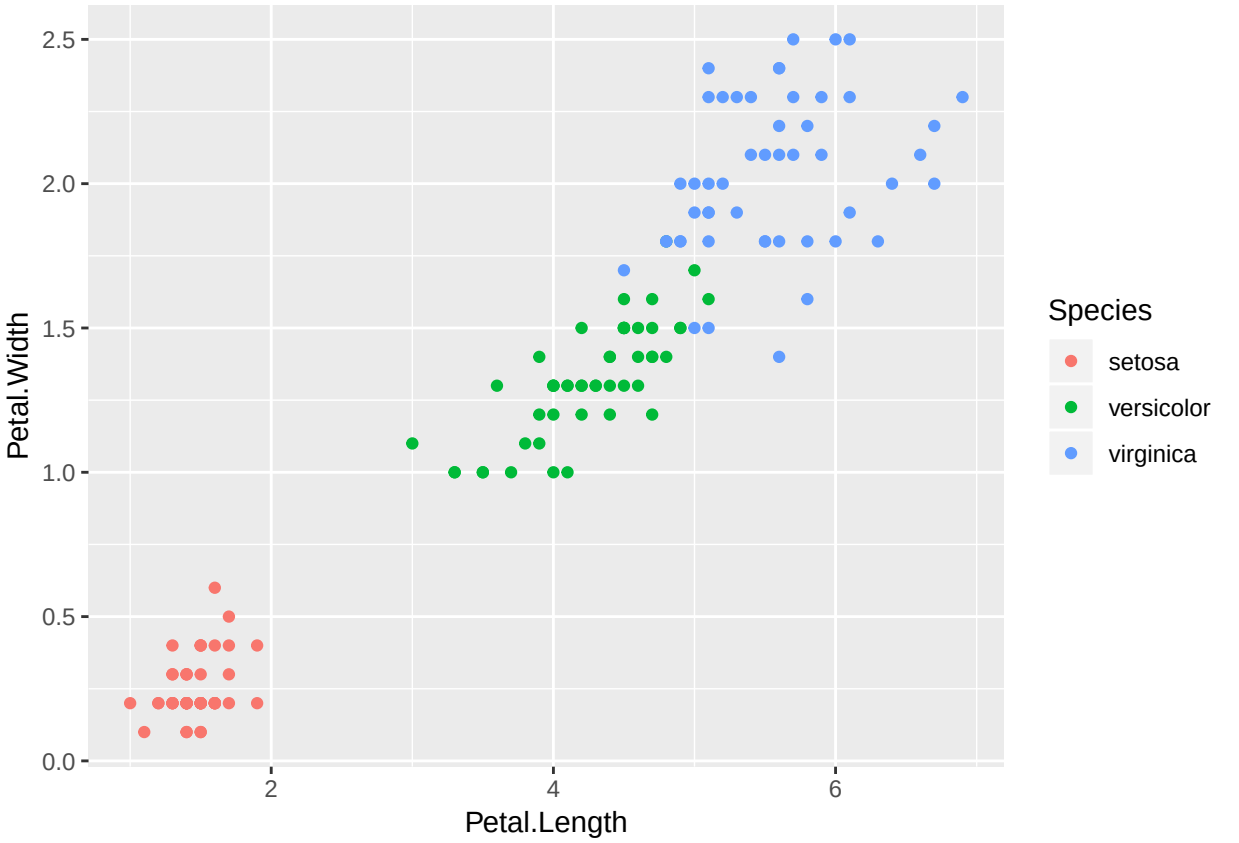
```
##   Sepal.Length Sepal.Width Petal.Length Petal.Width Species
## 1          5.1         3.5          1.4         0.2  setosa
## 2          4.9         3.0          1.4         0.2  setosa
## 3          4.7         3.2          1.3         0.2  setosa
## 4          4.6         3.1          1.5         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)
```

```
## Registered S3 methods overwritten by 'ggplot2':
##   method         from
##   [.quosures     rlang
##   c.quosures     rlang
```

```
##   print.quosures rlang
```

```
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 (1)

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

```
iris_matrix <- as.matrix(iris)
head(iris_matrix)
```

```
##      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"
```

```
str(iris_matrix)
```

```
##  chr [1:150, 1:5] "5.1" "4.9" "4.7" "4.6" "5.0" "5.4" "4.6" "5.0" ...
##  - attr(*, "dimnames")=List of 2
##   ..$ : NULL
##   ..$ : chr [1:5] "Sepal.Length" "Sepal.Width" "Petal.Length" "Petal.Width" ...
```

that doesn't look right...

Representing Data in R: Matricies (2)

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

```
#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
```

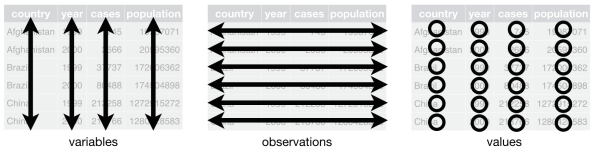
```
str(iris_matrix)
```

```
##  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	-

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

and find our files...

```
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.

```
token <- "eyJ0eXAiOiJKV1QiLCJhbGciOiJIUzI1NiJ9.eyJqdGkiOiIwNTY2MjlhYjc0YzA2ZWE3NjkOYzQ2NWM2YWM3MWIwMyIs
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

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

Then we'll convert it to a gatingSet

```
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 approraite channels:
- rename the channels according to our panel
- apply gates to the gatingset
- convert the data back to a flowset

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

```
## B Cells
## T Cells
## CD4s
## CD8s
## ....done!
```

```
myflowset_preprocessed <- flowWorkspace::getData(mygatingset)
```

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) %>%
  mutate(`FCS Filename` = basename(`FCS Filename`)) %>%
  mutate(Group = substr(`FCS Filename`, 13, 19))
str(mydataframe)
```

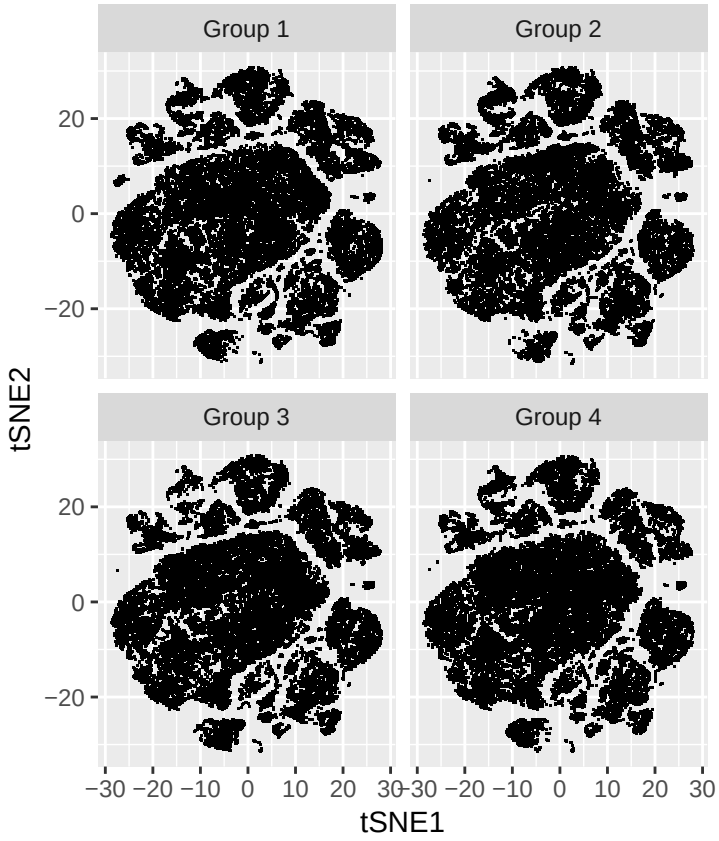
```
## 'data.frame':    100000 obs. of  77 variables:
##  $ Time                : num  30.8 86.5 96.9 115.2 143.5 ...
##  $ Event_length        : num  28 15 15 15 15 14 23 14 22 17 ...
##  $ 89Y_CD45 (v)        : num  5.09 3.96 4.38 4.25 4.39 ...
##  $ 102Pd               : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ 103Rh_Viability (v): num  1.62 0 0 0 0 ...
##  $ 104Pd               : num  0.015 0 0 0 0 ...
##  $ 105Pd               : num  0 0 0 0.182 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 0 0.014 ...
##  $ 127I                : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ 131Xe               : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ 133Cs               : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ 138Ba               : num  3.19 1.2 1.72 2.48 1.66 ...
##  $ 139La               : num  0 0 0 0 0 ...
##  $ 140Ce_EQ_Beads (v) : num  0 0.238 0 0 0 ...
##  $ 141Pr_CCR6 (v)      : num  0.345 0 0 0 0 ...
##  $ 142Nd               : num  0 0.268 0 0.188 0.376 ...
##  $ 143Nd               : num  0 0.766 0 0 0 ...
##  $ 144Nd               : num  0.0076 0.4031 0.3158 0.1317 0.0934 ...
##  $ 145Nd_CD4 (v)       : num  4.23749 0.65661 0.00594 3.82307 0.87242 ...
##  $ 146Nd_CD8 (v)       : num  2.964 5.773 2.319 0.535 5.619 ...
##  $ 147Sm_CD20 (v)      : num  0.1926 0 0 0 0.0314 ...
##  $ 148Nd_CD16 (v)      : num  0.0783 0.1981 3.3858 0 0.4623 ...
##  $ 149Sm_CCR4 (v)      : num  0 0 0 0.0259 0 ...
##  $ 150Nd               : num  0 0 0 0 0 ...
##  $ 151Eu               : num  0 0 0 0 0 ...
##  $ 152Sm               : num  1.4267 0 0 0.2121 0.0263 ...
##  $ 153Eu               : num  0 0 0 0 0 ...
##  $ 154Sm_CD3 (v)       : num  5.87 4.84 0 5.21 5.26 ...
##  $ 155Gd_CD45RA (v)    : num  3.23 3.3 3.01 3.37 4.37 ...
##  $ 156Gd               : num  0.00352 0 0 0 0 ...
##  $ 157Gd               : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ 158Gd               : num  0 0 0 0 0 ...
##  $ 159Tb_CCR7 (v)      : num  3.51 2.34 0 3.03 3.22 ...
##  $ 160Gd_CD14 (v)      : num  0 0 0 0 0.0252 ...
```

```
##  $ 161Dy              : num  0.303 0 0 0 0 ...
##  $ 162Dy              : num  0.0254 0.8437 0 0 0.8895 ...
##  $ 163Dy_CXCR3 (v)    : num  0 0.0603 0 0 3.2913 ...
##  $ 164Dy              : num  0 0 0.415 0 0 ...
##  $ 165Ho_CD45RO (v)   : num  0 0 0 0 0 ...
##  $ 166Er              : num  0.567 0 0 0 0 ...
##  $ 167Er_CD27 (v)     : num  3.15 3.09 0 2.9 3.72 ...
##  $ 168Er              : num  0.0626 0.0415 0 0 0.2607 ...
##  $ 169Tm_CD25 (v)     : num  0 0 0 0 0 ...
##  $ 170Er              : num  0 0 0 0.495 0 ...
##  $ 171Yb              : num  0 0 0 0 0 ...
##  $ 172Yb_CD57 (v)     : num  0 0 0 0 0 ...
##  $ 173Yb              : num  0 0 0 0 0 ...
##  $ 174Yb_HLA-DR (v)   : num  0 0 0.314 0 0.51 ...
##  $ 175Lu              : num  0 0 0 0.118 0 ...
##  $ 176Yb_CD127 (v)    : num  2.32 1.81 0 1.64 2.26 ...
##  $ 177Hf              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ 190BCKG            : num  0 0 0 0 0.152 ...
##  $ 191Ir_DNA (v)      : num  7.1 6.06 6.31 5.31 6.33 ...
##  $ 193Ir              : num  7.58 6.67 6.89 5.99 6.93 ...
##  $ 194Pt              : num  1.799 1.418 1.301 0.254 0.744 ...
##  $ 195Pt              : num  0 0 0 0.119 0 ...
##  $ 196Pt              : num  0 0 0.53 0 0 ...
##  $ 198Pt              : num  0.214 0 0.143 0 0 ...
##  $ 207Pb              : num  0 0 0 0 0 ...
##  $ 208Pb              : num  0.274 0.212 0 0.169 0 ...
##  $ 209Bi              : num  0 0 0 0 0 0 0 0 0 0 ...
##  $ Center             : num  6.18 5.37 5.48 5.48 5.32 ...
##  $ Offset             : num  4.45 3.53 3.74 3.66 3.54 ...
##  $ Width              : num  4.15 2.71 2.93 2.88 2.73 ...
##  $ Residual           : num  4.58 3.23 3.17 3.63 3.36 ...
##  $ tSNE1              : num  11.85 24.4 15.83 9.79 19.45 ...
##  $ tSNE2              : num  -5.23 -7.749 16.926 7.176 0.364 ...
##  $ density            : num  7.97 7.63 6.83 8.44 6.57 ...
##  $ cluster            : num  25 45 57 25 53 62 9 53 52 5 ...
##  $ FCS Filename       : chr  "20180321-04 Group 4 Helios B Post-viSNE_Ungated.fcs.density.fcs.cluste
##  $ Group              : chr  "Group 4" "Group 4" "Group 4" "Group 4" ...
```

Making Cytometry Figures in R (1)

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

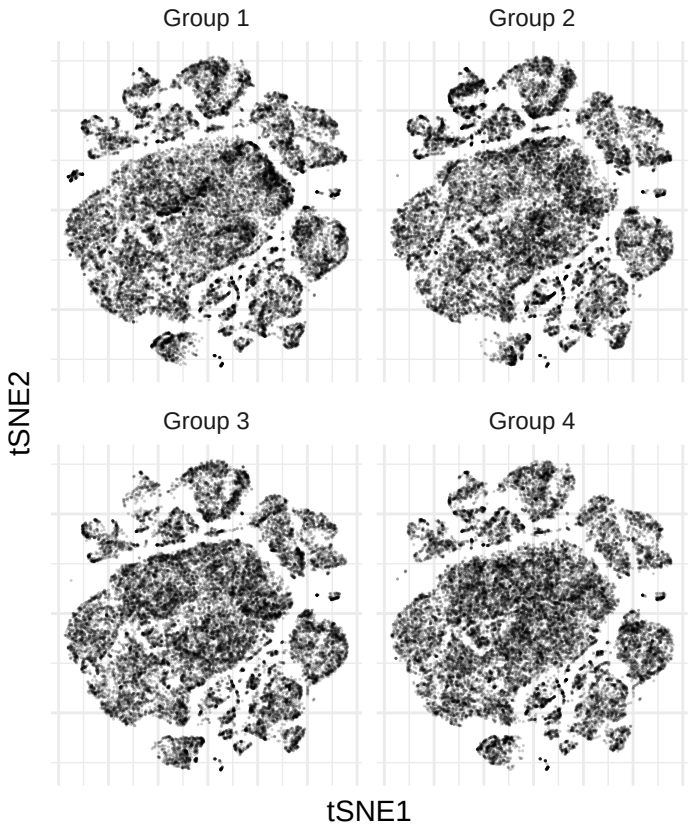
```
ggplot(mydataframe, aes(x = tSNE1, y = tSNE2)) +
  geom_point(shape = ".") +
  coord_fixed() +
  facet_wrap(~Group)
```



Making Cytometry Figures in R (2)

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

```
ggplot(mydataframe, aes(x = tSNE1, y = tSNE2)) +
  geom_point(shape = 16, alpha = 0.2, size = 0.2) +
  coord_fixed() +
  facet_wrap(~Group) +
  scale_color_viridis_c(option = "A") +
  theme_minimal() +
  theme(axis.text = element_blank())
```



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 facetted across mutliple 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] 100000     77
```

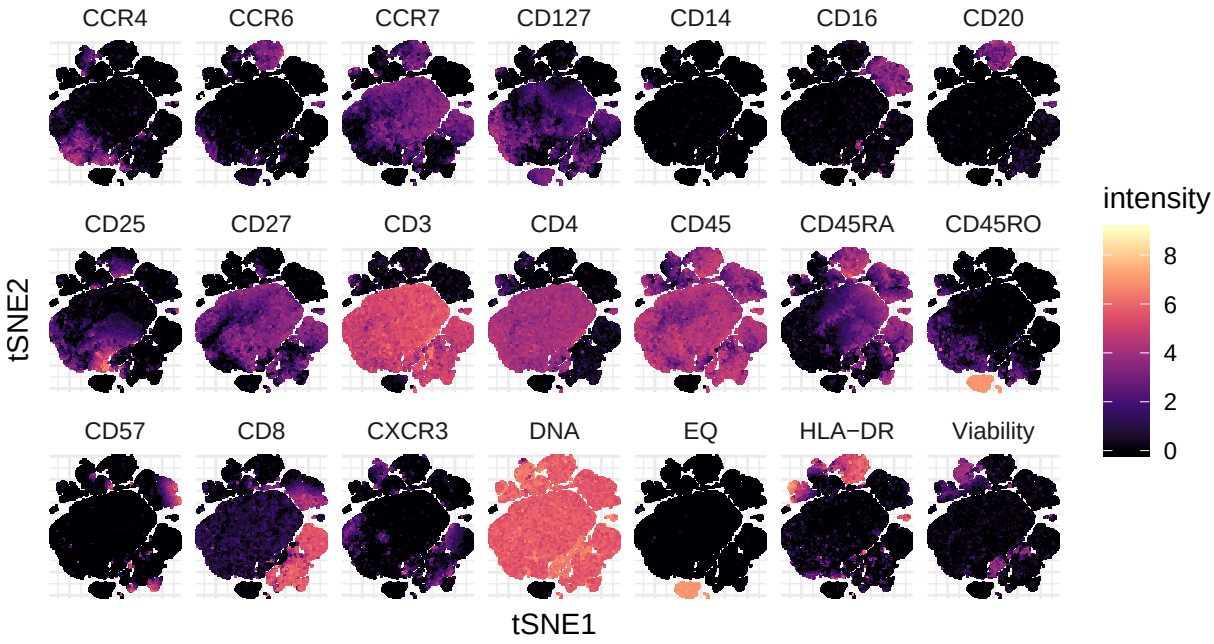
```
element2 <- function(x){unlist(lapply(strsplit(x, split = "_|\ "),"[[", 2))}
mydataframe.long <- mydataframe %>%
  as_tibble() %>%
  gather(marker, intensity, contains("(V)")) %>% # <- this is the key step
  separate(marker, c("channel", "marker", "drop"), sep= "_|\ ") %>%
  as_tibble()
dim((mydataframe.long))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

```
## [1] 2100000      60
```

Making Cytometry Figures in R (4)

Then we'll make our plot:

```
mydataframe.long %>%
  ggplot(aes(x = tSNE1, y = tSNE2, col = intensity)) +
  geom_point(shape = ".") +
  scale_x_continuous(expand = c(0,0)) +
  scale_y_continuous(expand = c(0,0)) +
  scale_color_viridis_c(option = "A") +
  coord_fixed() +
  facet_wrap(~marker, nrow = 3) +
  theme_minimal() +
  theme(axis.text = element_blank())
```



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. tSNE
  - Faster (minutes vs. hours)
  - Scalable ($f(n)$ vs. $f(n * log(n))$)

- 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 seperate 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)
```

```
## Loading required package: Matrix
```

```
##
## Attaching package: 'Matrix'
```

```
## The following object is masked from 'package:tidyr':
##
##     expand
```

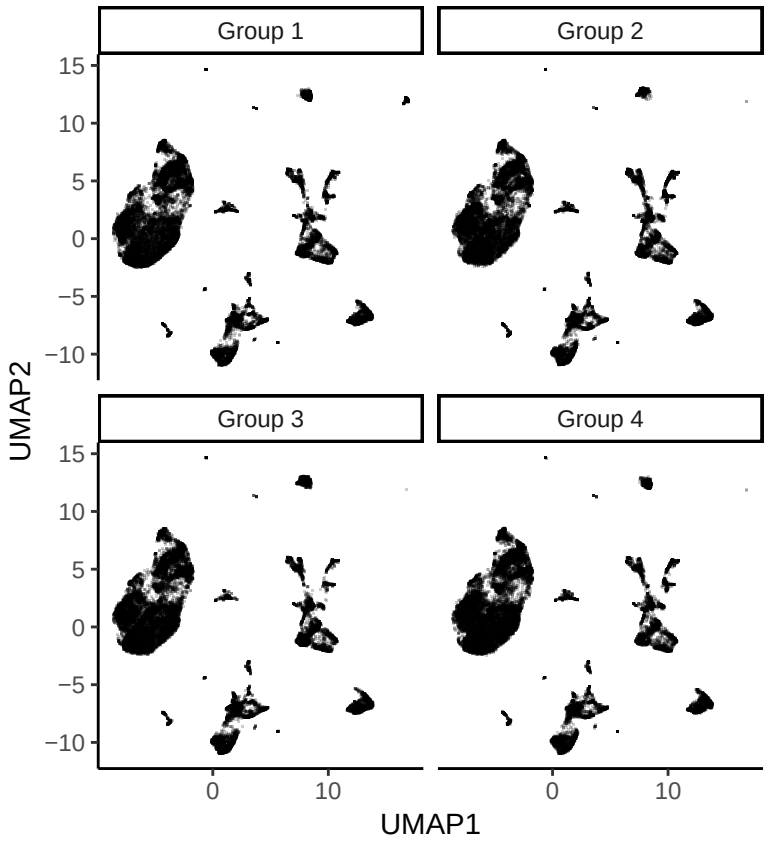
```
## The following object is masked from 'package:flowCore':
##
##     %&%
```

```
myumap <- umap(mymatrix, init = "PCA")
str(myumap)
```

```
##  num [1:100000, 1:2] -7.818 1.314 8.236 -6.362 -0.035 ...
##  - attr(*, "scaled:center")= num [1:2] 0.0173 -0.0835
```

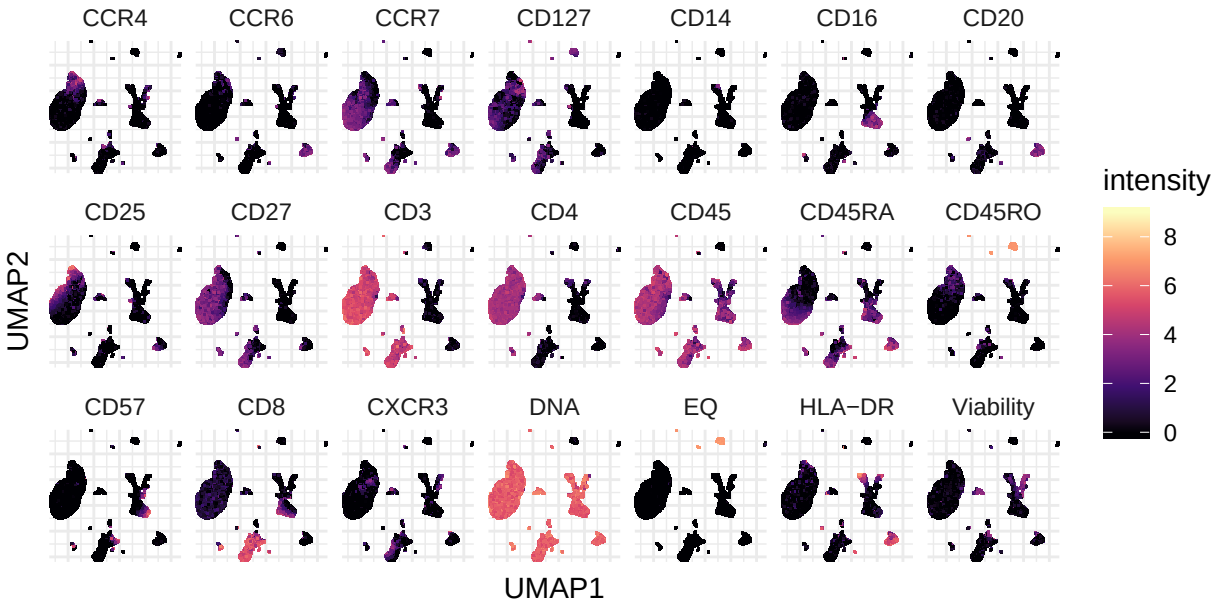
Applying alternative dimensionality reduction techniques (3)

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



Applying alternative dimensionality reduction techniques (4)

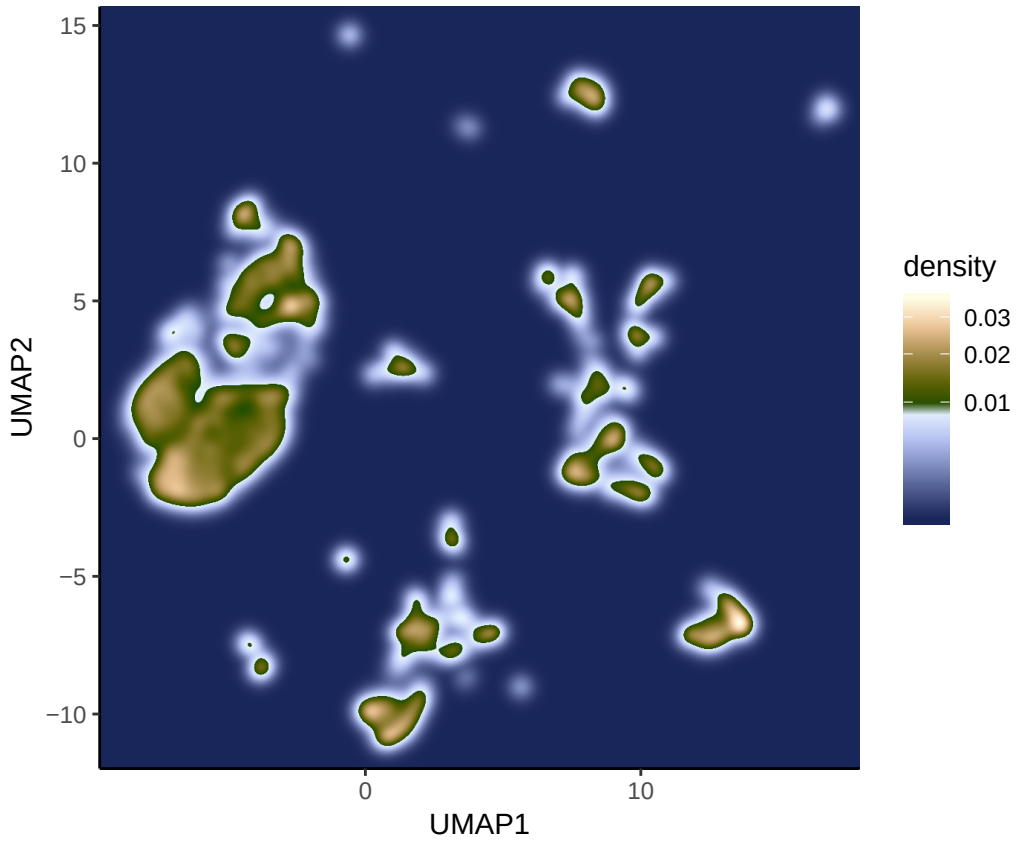
```
mydataframe %>%
  bind_cols(as.data.frame(myumap)) %>%
  as_tibble() %>%
  gather(marker, intensity, contains("(V)")) %>% # <- this is the key step
  separate(marker, c("channel", "marker", "drop"), sep= "_|\ ") %>%
  ggplot(aes(x = UMAP1, y = UMAP2, col = intensity)) +
  geom_point(shape = ".") +
  scale_x_continuous(expand = c(0,0)) +
  scale_y_continuous(expand = c(0,0)) +
  scale_color_viridis_c(option = "A") +
  coord_fixed() +
  facet_wrap(~marker, nrow = 3) +
  theme_minimal() +
  theme(axis.text = element_blank())
```



Applying alternative dimensionality reduction techniques (5)

We can also plot our data as a map:

```
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_classic() +
  coord_fixed()
```



Clustering in R

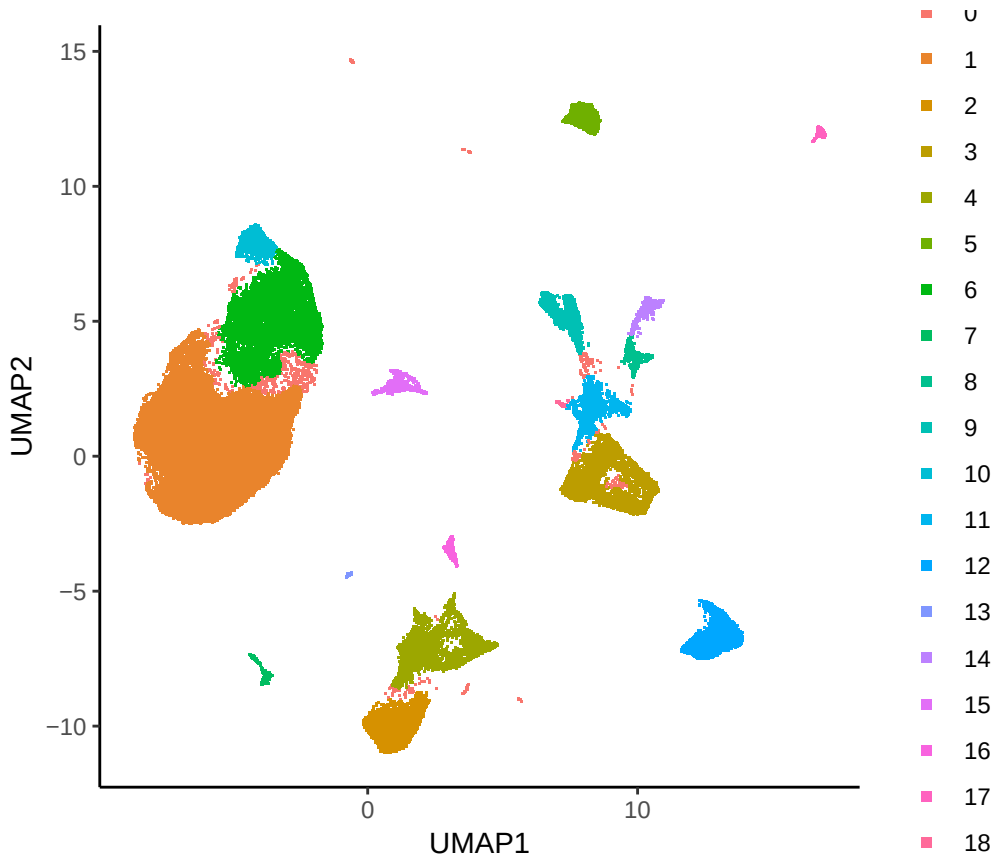
We can also apply a clustering algorithm to our dimensionality reduced data.

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

```
#install.packages("dbscan")
library(dbscan)
mydbscan <- dbscan(myumap, eps = 0.3, minPts = 150)
mydbscan
```

```
## DBSCAN clustering for 100000 objects.
## Parameters: eps = 0.3, minPts = 150
## The clustering contains 18 cluster(s) and 1256 noise points.
##
##     0     1     2     3     4     5     6     7     8     9    10    11
##  1256 37836  6534  8182  7486  2559 14659  1098  1224  2818  2111  2567
##    12    13    14    15    16    17    18
##  6484   384  1728  1724   867   349   134
##
## Available fields: cluster, eps, minPts
```

```
mydataframe %>%
  bind_cols(as.data.frame(myumap)) %>%
  mutate(cluster = as.factor(mydbscan$cluster)) %>%
 # dplyr::filter(cluster == 1:3) %>%
  ggplot(aes(x=UMAP1, y = UMAP2, col = cluster)) +
  geom_point(shape = ".") +
  scale_color_discrete(guide = guide_legend(override.aes = list(shape = 15))) +
  coord_fixed() +
  theme_classic()
```



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)
```

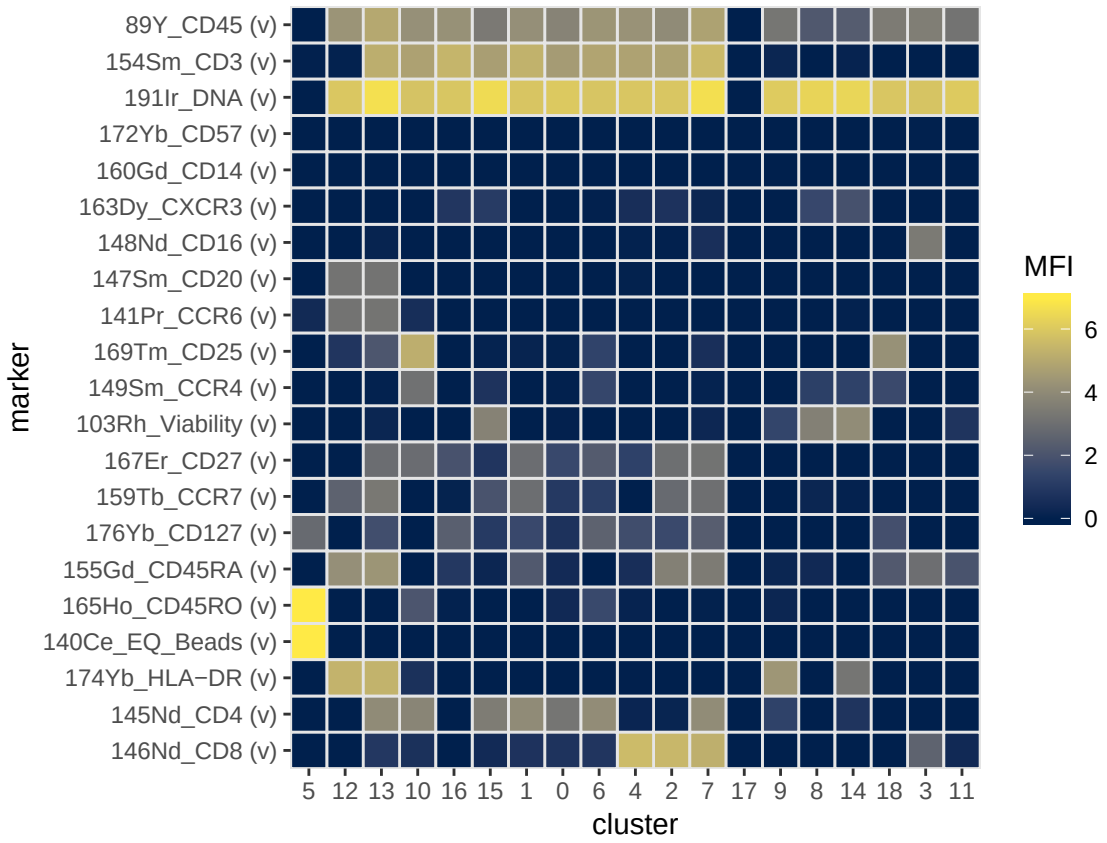
```
##
## Attaching package: 'tibble'
```

```
## The following object is masked from 'package:flowCore':
##
##     view
```

```r
library(scico)
library(seriation)
```

```
## Registered S3 method overwritten by 'seriation':
##   method         from
##   reorder.hclust gclus
```

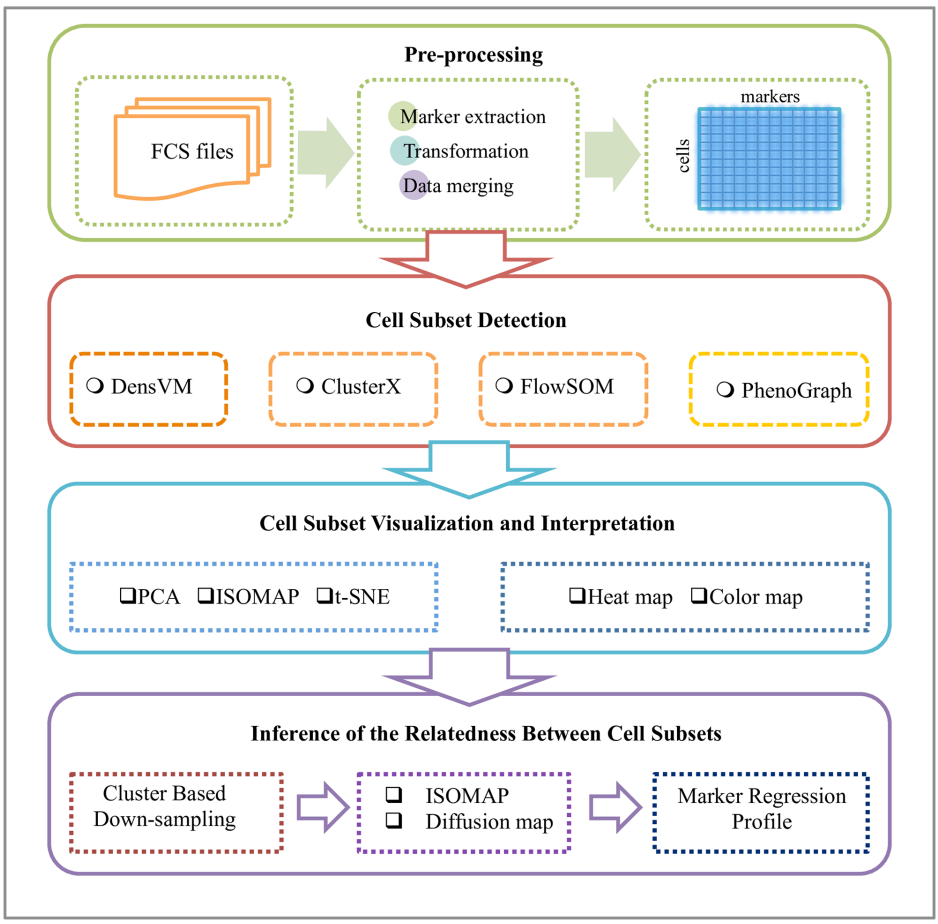
```r
myheatmap <- mydataframe %>%
  bind_cols(as.data.frame(myumap)) %>%
  mutate(cluster = as.factor(mydbscan$cluster)) %>%
  gather(marker, intensity, contains("(V)")) %>% # <- this is the key step
  group_by(cluster, marker) %>%
  summarise(MFI = median(intensity)) %>%
  select(marker, MFI, cluster)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



- Not currently maintained...

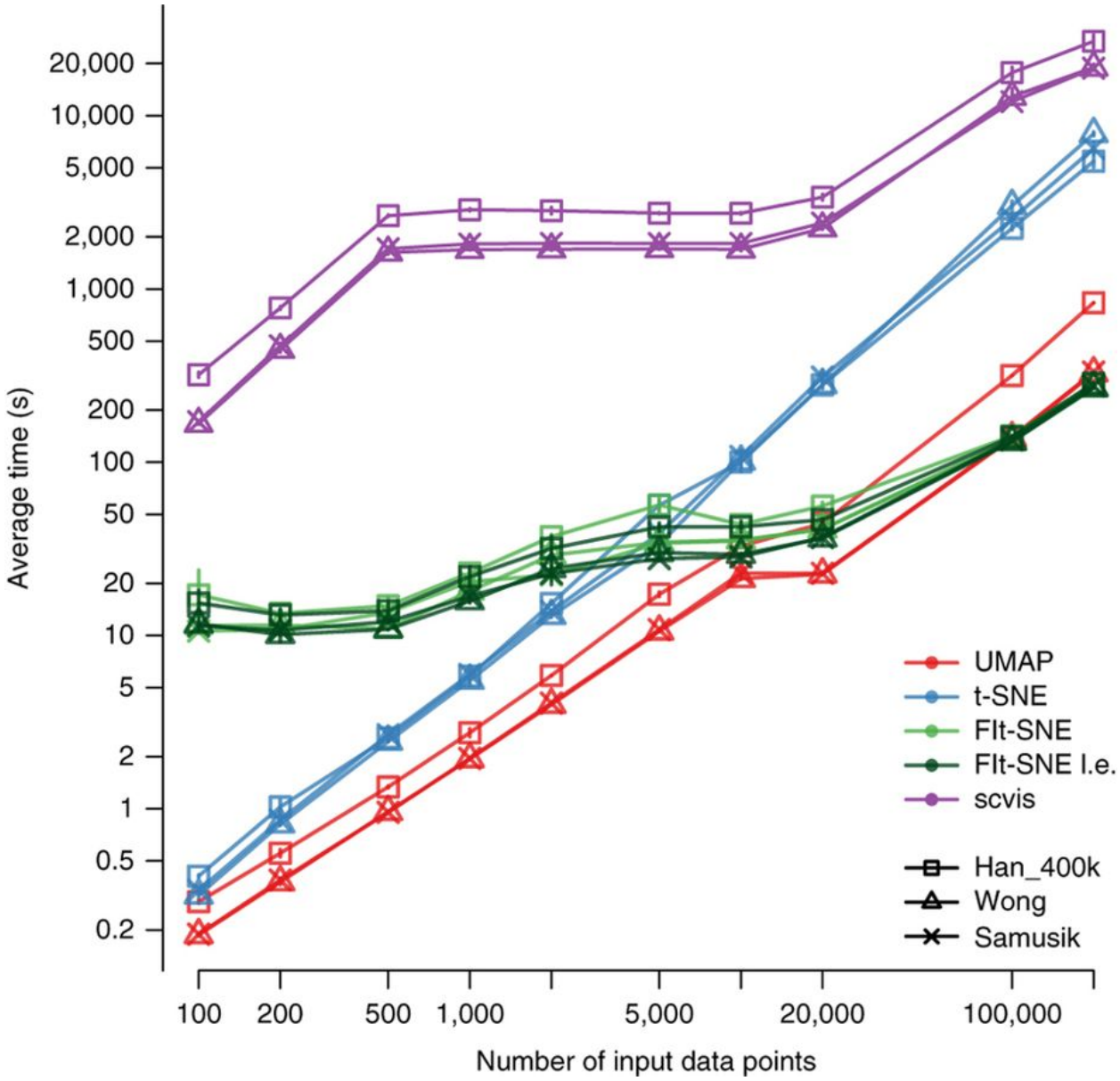
CytoRSuite

- 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. $f(n * log(n))$



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 markdown format.
- RGLab Github
  - See vignettes for FlowCore, FlowWorkspace, CytoML