Protocol: Flow Cytometry Cell Analysis

Amy Fox

Last edited: 2021-09-3

Note that descriptions are directly copied from the paper:“Cyto-Feature Engineering: A Pipeline for Flow Cytometry Analysis to Uncover Immune Populations and Associations with Disease”

# Overview

Flow cytometers can now analyze up to 50 parameters (antigens, size, granularity, cytokines, transcription factors, etc.) per cell and millions of cells per sample1. Conventional flow cytometry data analysis uses manual gating of cells on 2D plots to distinguish populations 1–2 dimensions at a time; this makes it both subjective and time consuming (up to 15 hours per experiment)2. Better methods are therefore critically needed to take full advantage of this powerful technology. Researchers have responded with open-source tools, including tools for automated gating to remove user input bias (e.g., openCyto) and tools to identify and cluster cell populations concurrently using all parameters (e.g., FlowSOM, t-SNE)3,4,5. While powerful advances, these new tools lack a straightforward way to integrate data from important technical controls or to compare resulting cell populations with other experimental measurements. Work is ongoing across several research groups to extend existing open-source tools to address some of these gaps. CytoCompare and cytofast, for example, focus on data analysis after clustering6,7. However, few tools exist that allow users to incorporate the many flow cytometry controls required for good data acquisition and analysis, and the output from the available clustering tools are often difficult for immunologists to interpret.

We have developed an end-to-end method for analyzing flow cytometry data that aims to address these limitations. For flow cytometry data, a parameter often represents a biologically binary phenomenon—that a marker is present or missing on a cell. While variation exists in the flow cytometry measurements for each parameter within cells in each binary group, that within-group variation is often uninformative noise. Our pipeline leverages this underlying biology—it uses feature engineering to create binary features for whether each cell has a positive or negative value for each marker. It does this using either external thresholds identified based on Fluorescence Minus One controls (FMOs) or the availability to separate the data based on clear population separation. The pipeline therefore identifies cell populations based on positive/negative combinations of each flow cytometry marker, a description that is readily interpretable by immunologists.

In four main steps, the pipeline: (1) cleans the data for live, single cells; (2) feature engineers the data based on FMO cutoffs or population separation; (3) analyzes the flow cytometry samples for all populations present in the sample and filters to populations above a population size threshold; (4) visualizes resulting populations through heatmaps of cell phenotypes and time series plots within experimental groups. Furthermore, it allows the use of statistical testing to identify cell populations associated with other experimental measurements (e.g., disease burden as measured through colony forming units) and novel populations induced by any experimental or clinical condition. All steps in the pipeline are modular, allowing each to be modified or replaced depending on the research question and features of the experimental data. As a case study, we illustrate the pipeline on a study involving Mycobacterium bovis Bacillus Calmette-Guérin (BCG)-vaccinated or control (Phosphate buffered saline (PBS)-injected) C57BL/6 mice infected with Mycobacterium tuberculosis (M. tuberculosis).

**Check** Add in Figure 1 here

Descriptions of the analysis method can be found here: <https://www.nature.com/articles/s41598-020-64516-0>

# Laboratory description

The laboratory procedure used for staining the cells, ensuring proper controls, and acquiring the data can be found here: <https://currentprotocols.onlinelibrary.wiley.com/doi/10.1002/cpcy.74>

# Method Limitations

While this pipeline has many advantages, there are several limitations to consider.

This analysis pipeline relies on high quality flow cytometry methodology, and/or FMO samples, as well as, strong panel design. Spillover from other channels can greatly impact the analysis, so researchers must ensure that the controls are prepared correctly.

# Data description

Flow cytometers use a standardized file format for outputting data, the .fcs file, which includes cell measurements, metadata describing data collection, and the Median Fluorescent Intensities (MFIs) of fluorescently-conjugated antibodies or fluorescent probes. Typically, a different .fcs file is created for each sample.

# Reading data into R

Multiple .fcs files generated from an experiment can be read into R and manipulated as an “ncdfFlowSet” object. Our pipeline begins by reading experimental data into an “ncdfFlowSet” object.

We start by identifying the folder where all of our FMO files are saved.

# Identify the file names of all 20 FCS flow cytometry experiments to read in.   
FMO\_fcsFiles <- list.files("../inst/extdata/FMOs", full = TRUE)  
FMO\_fcsFiles

## [1] "../inst/extdata/FMOs/CD103.fcs" "../inst/extdata/FMOs/CD122.fcs"  
## [3] "../inst/extdata/FMOs/CD153.fcs" "../inst/extdata/FMOs/CD27.fcs"   
## [5] "../inst/extdata/FMOs/CD28.fcs" "../inst/extdata/FMOs/CD3.fcs"   
## [7] "../inst/extdata/FMOs/CD4.fcs" "../inst/extdata/FMOs/CD44.fcs"   
## [9] "../inst/extdata/FMOs/CD62L.fcs" "../inst/extdata/FMOs/CD69.fcs"   
## [11] "../inst/extdata/FMOs/CD8.fcs" "../inst/extdata/FMOs/CTLA4.fcs"  
## [13] "../inst/extdata/FMOs/FoxP3.fcs" "../inst/extdata/FMOs/IFNG.fcs"   
## [15] "../inst/extdata/FMOs/IL\_10.fcs" "../inst/extdata/FMOs/IL\_17.fcs"  
## [17] "../inst/extdata/FMOs/KLRG1.fcs" "../inst/extdata/FMOs/PD1.fcs"   
## [19] "../inst/extdata/FMOs/Sca1.fcs" "../inst/extdata/FMOs/TNF.fcs"

We then reference the location of the saved files and read in the files creating an “ncdfFlowSet” object. Note that this may take a minute or so to run.

The best practice for acquiring samples means naming marker channels the exact same each time an experiment is performed. However, if the samples contain multiple aliases for the same marker, we can use the channel\_alias argument in the read.ncdfFlowSet function to standardize the naming. For example, in some samples, one marker channel is labeled “Zombie Nir-A”, whereas in other samples, the same marker channel is labeled “Zombie\_NIR-A” or “Zombie NIR-A.” We must standardize the naming of the marker channels here or they will not be recognized as the same channel later in the analysis. Note that letter case matters.

library(ncdfFlow)  
  
ncfs\_FMO <- read.ncdfFlowSet(FMO\_fcsFiles,   
 channel\_alias = data.frame(alias = c("Zombie Nir-A"),  
 channels = c("Zombie\_NIR-A,   
 Zombie NIR-A,   
 Zombie Nir-A")))

Here is a description of the ‘ncdfFlowSet’ object.

## An ncdfFlowSet with 20 samples.  
## NCDF file : /var/folders/56/2lqw28ls6b1bnx36fgm2qfcc0000gn/T//Rtmp9R2rNp/ncfs82082822f373.nc   
## An object of class 'AnnotatedDataFrame'  
## rowNames: CD103.fcs CD122.fcs ... TNF.fcs (20 total)  
## varLabels: name  
## varMetadata: labelDescription  
##   
## column names:  
## Time, SSC-H, SSC-A, FSC-H, FSC-A, BV421-H, Pacific Blue-H, BV480-H, BV510-H, BV570-H, BV605-H, BV650-H, BV711-H, BV785-H, BB515-H, Alexa Fluor 532-H, PE-H, PE-Dazzle594-H, PE-Cy5-H, PE-Cy5.5-H, PerCP-eFluor 710-H, PE-Cy7-H, APC-H, APC-R700-H, APC-Fire 750-H, AF-H, BV421-A, Pacific Blue-A, BV480-A, BV510-A, BV570-A, BV605-A, BV650-A, BV711-A, BV785-A, BB515-A, Alexa Fluor 532-A, PE-A, PE-Dazzle594-A, PE-Cy5-A, PE-Cy5.5-A, PerCP-eFluor 710-A, PE-Cy7-A, APC-A, APC-R700-A, APC-Fire 750-A, Zombie Nir-A, AF-A

The resulting ‘ncdfFlowset’ object contains row names with the individual samples and column names with the markers/parameters used in the flow cytometer. This ‘ncdfFlowset’ can be indexed in a few ways.

The following indexing pulls out the data for the first sample.

ncfs\_FMO[1]

## An ncdfFlowSet with 1 samples.  
## NCDF file : /var/folders/56/2lqw28ls6b1bnx36fgm2qfcc0000gn/T//Rtmp9R2rNp/ncfs82082822f373.nc   
## An object of class 'AnnotatedDataFrame'  
## rowNames: CD103.fcs  
## varLabels: name  
## varMetadata: labelDescription  
##   
## column names:  
## Time, SSC-H, SSC-A, FSC-H, FSC-A, BV421-H, Pacific Blue-H, BV480-H, BV510-H, BV570-H, BV605-H, BV650-H, BV711-H, BV785-H, BB515-H, Alexa Fluor 532-H, PE-H, PE-Dazzle594-H, PE-Cy5-H, PE-Cy5.5-H, PerCP-eFluor 710-H, PE-Cy7-H, APC-H, APC-R700-H, APC-Fire 750-H, AF-H, BV421-A, Pacific Blue-A, BV480-A, BV510-A, BV570-A, BV605-A, BV650-A, BV711-A, BV785-A, BB515-A, Alexa Fluor 532-A, PE-A, PE-Dazzle594-A, PE-Cy5-A, PE-Cy5.5-A, PerCP-eFluor 710-A, PE-Cy7-A, APC-A, APC-R700-A, APC-Fire 750-A, Zombie Nir-A, AF-A

# Data cleaning and initial gating

After reading in our data, we then want to perform some typical data cleaning steps normally performed on every flow cytometry sample. A typical gating strategy will first gate on “singlets” or single cells, then “lymphocytes” to remove debris and then only the live cells.

The openCyto package provides infrastructure for the use of reproducible algorithms to gate cells based on marker density. However, it alone is unable to control for instances where clumps of cells pass through the flow cytometer lasers, producing erroneous results and subsequently skewing the data. To address this phenomenon, the “singletGate” function from the flowStats package is used to remove doublet or larger cell clumps. The pipeline then funnels the data through the “mindensity” function, selecting for leukocytes via a threshold filter that distinguishes between populations based on cell density3. Finally, a “mindensity” gate is used with a live/dead stain (Zombie NIR), to filter the data to only live cells.

The openCyto package utilizes .csv to develop gating strategies. Information on developing a .csv with different gating strategies can be found here: <http://opencyto.org/articles/HowToWriteCSVTemplate.html>

Briefly, the “alias” column is what you will call each of the gating populations, for example, when gating on singlets, I’ll probably want to give the alias of “singlets”. The “pop” column takes in either a “+” or “-”. When you want to take the positive cells, or the cells on the right side (or within) a the gate, you add a “+” here. The “parent” column lists the name of the cells that you want to gate. For example, the first parent will be “root” because because we’re gating on all of the available cells. If we gate our root cells to look at our siglets, then our next gate will use “singlets” as the parent. The “dims” is the name of the flow cytometry marker name listed in the data that you want to gate on. For example, when gating on singlets, we look at SSC-A and SSC-H, so our dims will be “SSC-A,SSC-H” when gating on CD4 cells, we will write “CD4.” Note that the “dims” name must match exactly do the colunn names in your data. Finally include the name of the gating method that you want to use. Different gating options can be found here: <http://opencyto.org/articles/HowToAutoGating.html>.

**Check: Include info on why we use wider\_gate and how I chose the gating\_arg**

# Identify the file with the gating strategy   
ws <- list.files("../inst/extdata/",   
 pattern = "gating\_strategy.csv",   
 full = TRUE)

We can view the initial gating strategy with the fread function. Here is what the data looks like once it has been read in.

library(data.table)  
# Read in the template  
dtTemplate <- fread(ws)  
  
dtTemplate

## alias pop parent dims gating\_method gating\_args  
## 1: singlets + root FSC-A,FSC-H singletGate wider\_gate=TRUE,maxit=20  
## 2: lymph + singlets FSC-A mindensity gate\_range=c(4e5,8e5)  
## 3: live - lymph Zombie Nir-A mindensity gate\_range=c(1e4,1e5)  
## collapseDataForGating groupBy preprocessing\_method preprocessing\_args  
## 1: NA NA NA NA  
## 2: NA NA NA NA  
## 3: NA NA NA NA

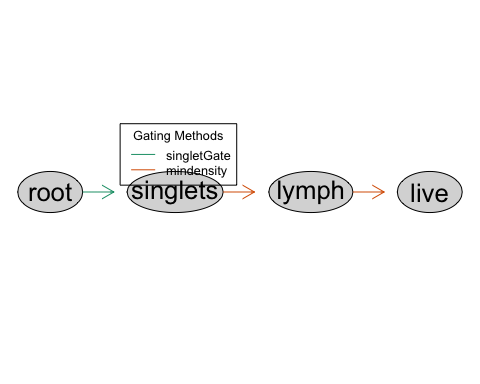
In this dataframe, each row in the .csv represents a different gating step.

We then read in the gating strategy to a gatingTemplate object.

library(openCyto)  
initial\_gate <- gatingTemplate(ws)

The gating strategy can then be viewed with the plot function.

plot(initial\_gate)

 This shows that we will take all of the cells (root), gate on singlets using a singletGate, then gate the lymphocytes using a mindensity gate, and finally gate on live cells using a mindensity gate.

We first have to convert our ‘ncdfFlowset’ object to a ‘GatingSet’ object, so we can apply the initial gating to this data.

library(ggcyto)  
gs\_FMO <- GatingSet(ncfs\_FMO)

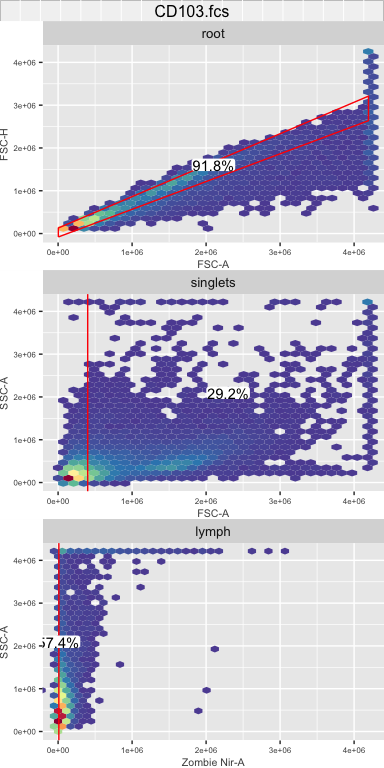
We then apply the initial gating to filter the data to only measurements on live lymphocyte cells. This may take a minute

gt\_gating(initial\_gate, gs\_FMO)

The results can then be plotted with autoplot using indexing of the gs\_FMO object. For example, to plot the gating for the first sample, run:

autoplot(gs\_FMO[[1]])

## Coordinate system already present. Adding new coordinate system, which will replace the existing one.  
## Coordinate system already present. Adding new coordinate system, which will replace the existing one.  
## Coordinate system already present. Adding new coordinate system, which will replace the existing one.



The data is next converted from a “flowSet” object into a dataframe object that complies with the “tidy data” standards, allowing further pipeline steps to draw on the powerful suite of “tidyverse” tools in R.

We pull out the data from the ‘live’ node of the gating set (the last node in the initial gating strategy) using the gs\_pop\_get\_data function. The output is a cytoset object, so we then convert the cytoset to a flowSet which can then be converted into a tidy dataframe using the tidy\_flow\_set function from the cytotypr package.

library(dplyr)  
library(cytotypr)  
FMO\_gated\_data <- gs\_pop\_get\_data(gs\_FMO, "live") %>%   
 cytoset\_to\_flowSet() %>%  
 tidy\_flow\_set()

We can then view the FMO tidy data.

head(FMO\_gated\_data)

## # A tibble: 6 x 49  
## filename Time `SSC-H` `SSC-A` `FSC-H` `FSC-A` `BV421-H` `Pacific Blue-H`  
## <chr> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl> <dbl>  
## 1 CD103.fcs 130 105692 126259. 354442 645303. -58.8 295.  
## 2 CD103.fcs 137 270246 315833. 1094791 1587586. -6837. 181.  
## 3 CD103.fcs 139 326197 377253. 1312044 1802283. -6375. 2438.  
## 4 CD103.fcs 149 74523 85250. 295947 557216. 285. 542.  
## 5 CD103.fcs 159 103797 110015. 293302 485668. -419. 250.  
## 6 CD103.fcs 215 188622 219861. 355468 550662. -589. 1265.  
## # … with 41 more variables: BV480-H <dbl>, BV510-H <dbl>, BV570-H <dbl>,  
## # BV605-H <dbl>, BV650-H <dbl>, BV711-H <dbl>, BV785-H <dbl>, BB515-H <dbl>,  
## # Alexa Fluor 532-H <dbl>, PE-H <dbl>, PE-Dazzle594-H <dbl>, PE-Cy5-H <dbl>,  
## # PE-Cy5.5-H <dbl>, PerCP-eFluor 710-H <dbl>, PE-Cy7-H <dbl>, APC-H <dbl>,  
## # APC-R700-H <dbl>, APC-Fire 750-H <dbl>, AF-H <dbl>, BV421-A <dbl>,  
## # Pacific Blue-A <dbl>, BV480-A <dbl>, BV510-A <dbl>, BV570-A <dbl>,  
## # BV605-A <dbl>, BV650-A <dbl>, BV711-A <dbl>, BV785-A <dbl>, BB515-A <dbl>,  
## # Alexa Fluor 532-A <dbl>, PE-A <dbl>, PE-Dazzle594-A <dbl>, PE-Cy5-A <dbl>,  
## # PE-Cy5.5-A <dbl>, PerCP-eFluor 710-A <dbl>, PE-Cy7-A <dbl>, APC-A <dbl>,  
## # APC-R700-A <dbl>, APC-Fire 750-A <dbl>, Zombie Nir-A <dbl>, AF-A <dbl>

Each of the rows represents a different cell with the various marker channels as the columns and the corresponding MFI values for each cell and marker channel.

# Prepare FMOs for feature engineering

FMOs are often used in manual gating to control for data spread and spillover events, which are common during flow cytometry data collection12. Take for example a panel consisting of 10 markers with different fluorophores. When excited, each of those 10 markers fluoresce at different intensities along the light spectrum. However, while they have different spectrums, tails of these spectrums can overlap. This overlap can lead to noise within a parameter’s measurements, and in extreme cases, to the detection of false positives/negatives in the presence or absence of a marker. FMOs are created experimentally; by running parallel samples where each sample has just one marker removed from the overall panel, all cells are guaranteed to be truly negative on that marker. With FMOs, we can therefore identify a threshold for the maximum parameter values possible for true negative marker signal on cells to determine marker presence in fully stained samples12. Incorporation of FMOs greatly reduces the subjectivity of manual gating and helps support unbiased analysis of flow cytometry data. Despite the importance of FMOs for accurate analysis, limited flow cytometry computational tools exist that incorporate them into unsupervised analysis13.

Note that when you plot the FMOs here, you should see all of the FMOs that you want to use. If you don’t see all of them, ensure that all ofyour filenames and column names for each of the markers matches exactly. For example, if the filename says “CD103\_f” but the corresponding column name for that marker is “CD103”, you need to either change the filename or column name so that they are exactly the same.

library(stringr)  
# note that here the filename and the column marker names need to match exactly  
df\_FMO\_gated\_data <- FMO\_gated\_data %>%  
 select(ends\_with("-A"), -`FSC-A`, `SSC-A`, filename) %>%  
 dplyr::rename(`FoxP3` = "APC-A",  
 `CD44` = "APC-Fire 750-A",  
 `CD103` = "APC-R700-A",   
 `CD3` = "Alexa Fluor 532-A",  
 `Sca1` = "BB515-A",  
 `IL\_10` = "BV421-A",  
 `CD4` = "BV480-A",  
 `CD69` = "BV510-A",  
 `CD8` = "BV570-A",   
 `CTLA4` = "BV605-A",  
 `CD27` = "BV650-A",  
 `CD153` = "BV711-A",  
 `KLRG1` = "BV785-A",  
 `IL\_17` = "PE-A",  
 `CD122` = "PE-Cy5-A",  
 `IFN` = "PE-Cy7-A",   
 `CD62L` = "PE-Dazzle594-A",  
 `TNF` = "Pacific Blue-A",   
 `CD28` = "PE-Cy5.5-A",  
 `PD1` = "PerCP-eFluor 710-A") %>%  
 na.omit()%>%  
 dplyr::filter(`SSC-A` != max(`SSC-A`)) %>%  
 mutate(filename = str\_replace(filename, ".fcs", "")) %>%  
 mutate(filename = str\_replace(filename, "IFNG", "IFN"))  
   
FMO\_filtered\_data <- filter\_FMO(df\_FMO\_gated\_data)

Our pipeline processes the data from FMOs to include in further analysis. Traditionally, FMOs have been manually gated to identify the upper threshold of a parameter’s value for negative cells. In our pipeline, we instead automate this analysis of the FMOs, measuring the threshold as the 99th percentile of the parameter values in each FMO (Fig. 2). Noise can originate from very small particles or debris that pass through the flow cytometer. In an ideal world, a 100% threshold could be used, but in reality, the 99% threshold is used to account for this random noise.

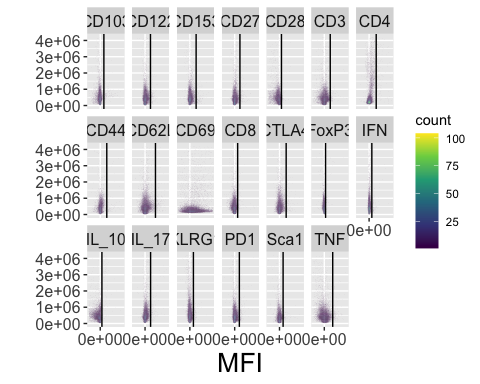
The 99th percentile values are then saved in this add\_quantile object and subsequently funneled into feature engineering of binary features (negative/positive) for each marker in the experimental data.

Here we create a dataframe of the cutoffs for each of the FMOs

add\_quantile <- get\_99(FMO\_filtered\_data)

We can can assess this 99% threshold with the FMO plots and check that each of the FMOs has properly been read in in this plot. If there is a missing FMO, the most likely cause is that the filename and the marker channel column name are not the exact same.

plot\_FMOs(FMO\_filtered\_data, add\_quantile)



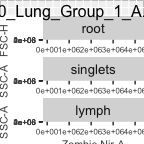
The 18 FMOs are shown with the individual marker MFI expression on the x-axis and Side Scatter (SSC-A) on the y-axis. The black vertical line indicates the 99th percentile threshold for identifying positive versus negative cells (i.e., 99% of the data is located to the left of the line in each plot). These thresholds are used on the subsequent samples to feature engineer new parameters on whether a cell positively or negatively expresses each marker.

**check: add in information about adjusting the thresholds if need be**

# Read in and gate the sample data

Up until this point, we have only worked with our control FMO data. We must now read in our samples and subsequently gate them and convert to tidy data using the same methodology as described for the FMOs.

fcsFiles <- list.files("../inst/extdata/Tcell\_samples",   
 pattern = ".fcs", full = TRUE)  
  
ncfs <- read.ncdfFlowSet(fcsFiles,   
 channel\_alias = data.frame(alias = c("Zombie Nir-A"),   
 channels = c("Zombie\_NIR-A,   
 Zombie NIR-A,   
 Zombie Nir-A")))   
gs <- GatingSet(ncfs)  
  
gt\_gating(initial\_gate, gs)  
  
autoplot(gs[[1]])



df\_all\_gated <- gs\_pop\_get\_data(gs, "live") %>%   
 cytoset\_to\_flowSet() %>%  
 tidy\_flow\_set()

# Feature engineer the data

Before feature engineering the data, we first want to do a little bit of data clean up. For this experiment, we had different timepoints and groups. We used regular expressions from the stringr package to extract information about the timepoints and groups from the filename to clean up the names a bit.

We then removed duplicated rows, for example, we have the -A (area) and -H (height) parameters for each marker channel. We can remove all of the -H channels and use just the -H. We then want to rename all of our marker channels with the name of the marker that we used rather than the fluorescent channel. This will make it easier to interpret the results later on. Finally, we remove any SSC-A parameters that are at the upper limit of detection for the flow cytometer.

library(tidyr)  
clean\_df\_all\_gated <- df\_all\_gated %>%  
 mutate(Timepoint = str\_extract(filename, "D[0-9]\*")) %>%  
 mutate(Group = str\_extract(filename, "Group[:punct:][0-9][:punct:][A-Z]"),  
 Group = str\_replace(Group, "Group\_", "")) %>%  
 unite(filename, c("Timepoint", "Group")) %>%  
 select(ends\_with("-A"), -`FSC-A`, filename) %>%  
 dplyr::rename(`FoxP3` = "APC-A",  
 `CD44` = "APC-Fire 750-A",  
 `CD103` = "APC-R700-A",   
 `CD3` = "Alexa Fluor 532-A",  
 `Sca1` = "BB515-A",  
 `IL\_10` = "BV421-A",  
 `CD4` = "BV480-A",  
 `CD69` = "BV510-A",  
 `CD8` = "BV570-A",   
 `CTLA4` = "BV605-A",  
 `CD27` = "BV650-A",  
 `CD153` = "BV711-A",  
 `KLRG1` = "BV785-A",  
 `IL\_17` = "PE-A",  
 `CD122` = "PE-Cy5-A",  
 `IFN` = "PE-Cy7-A",   
 `CD62L` = "PE-Dazzle594-A",  
 `TNF` = "Pacific Blue-A",   
 `CD28` = "PE-Cy5.5-A",  
 `PD1` = "PerCP-eFluor 710-A") %>%  
 dplyr::filter(`SSC-A` != max(`SSC-A`))

Features are measurements in a dataset, such as the MFIs used in flow cytometry. Feature engineering is a machine learning technique that uses the original features in a dataset, possibly with the integration of external knowledge or data, to create new features that make the data easier to understand14,15.

For flow cytometry, FMOs can add information about the possible range of expression measurements for cells that are truly negative for a marker. The threshold identified by FMOs can be used to create new binary features that capture whether the expression of each marker is positive or negative for the cell, thus, simplifying overly redundant, continuous MFI data with noise resulting from spillover. In the pipeline, we feature engineer each parameter using the fe function. This function utilizes the thresholds identified from the FMOs, so that positive expression on cells (values above the FMO cutoff) equal 1 and negative cells equal 0.

For each cell in the experimental data, the cell phenotype is then identified based on the set of marker expressions (0’s and 1’s) of each population. Eighteen markers were used to elucidate memory T cell populations including markers for terminal differentiation and exhaustion in the M. tuberculosis case study. The pipeline identifies all cell populations (i.e., combinations of negative and positive marker expression values) for which at least one sample includes at least one cell.

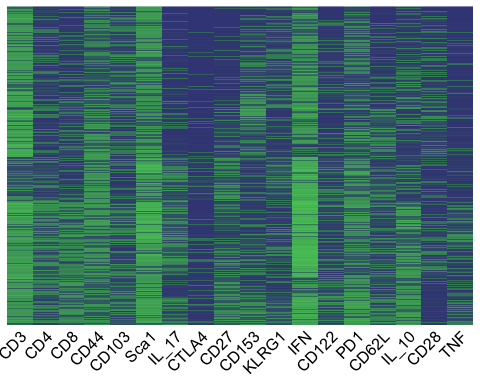
The “count\_calc” function at the end calculates the cell counts and percentage of cells in each sample for each population. The dataframe that is input into this function should only contain the markers that you’re interesting in looking at, and should remove SSC-A, FSC-A, etc. Note: Any column that doesn’t have a 0 or 1 must be removed here (Except experimental/group names). In this analysis, we also removed FoxP3 and CD69 because there was too much spread in the FMO parameters to utilize feature engineering.

all\_fe <- clean\_df\_all\_gated %>%  
 mutate(CD3 = fe(add\_quantile, CD3, "CD3"),  
 CD4 = fe(add\_quantile, CD4, "CD4"),  
 CD8 = fe(add\_quantile, CD8, "CD8"),  
 CD44 = fe(add\_quantile, CD44, "CD44"),  
 CD103 = fe(add\_quantile, CD103, "CD103"),  
 Sca1 = fe(add\_quantile, Sca1, "Sca1"),  
 IL\_17 = fe(add\_quantile,IL\_17, "IL\_17"),  
 CD69 = fe(add\_quantile,CD69, "CD69"),  
 CTLA4 = fe(add\_quantile,CTLA4, "CTLA4"),  
 CD27 = fe(add\_quantile,CD27, "CD27"),  
 CD153 = fe(add\_quantile,CD153, "CD153"),  
 KLRG1 = fe(add\_quantile,KLRG1, "KLRG1"),  
 IFN = fe(add\_quantile,IFN, "IFN"),  
 FoxP3 = fe(add\_quantile,FoxP3, "FoxP3"),  
 CD122 = fe(add\_quantile,CD122, "CD122"),  
 PD1 = fe(add\_quantile,PD1, "PD1"),  
 CD62L = fe(add\_quantile,CD62L, "CD62L"),  
 IL\_10 = fe(add\_quantile,IL\_10, "IL\_10"),  
 CD28 = fe(add\_quantile,CD28, "CD28"),  
 TNF = fe(add\_quantile,TNF, "TNF")) %>%  
 select(-`Zombie Nir-A`, -`AF-A`, -`SSC-A`, -FoxP3, -CD69) %>% #   
 count\_calc()

# Exploring the data

We first want to view all of the different cell phenotypes within the data. For the plot, we first make a vector order\_of\_markers of the order of the flow markers that we want on our x-axis. This step is not necessary, but can be helpful in organizing the flow markers according to importance. We then filter all of our data to only the flow phenotypes with filter\_for\_total\_pheno and finally plot the results with heatmap\_all\_pheno.

# this is the order of markers that we want for all of our plots  
order\_of\_markers <- c("CD3", "CD4", "CD8", "CD44", "CD103", "Sca1", "IL\_17","CTLA4",  
 "CD27", "CD153", "KLRG1", "IFN", "CD122", "PD1", "CD62L",  
 "IL\_10", "CD28","TNF")  
  
# to view all of the possible combinations  
total\_phenotypes <- filter\_for\_total\_pheno(all\_fe, marker\_order = order\_of\_markers)  
  
heatmap\_all\_pheno(total\_phenotypes)



This plot identifies all phenotypes in the samples. Each row represents a unique cell phenotype, where green indicates positive expression and blue indicates negative expression of each marker.

We can also quantify the total number of populations identified here

nrow(total\_phenotypes)

## [1] 10912

A total of 10,912 cell populations were identified in the samples for this study. As this number of populations is still very large, the data can be filtered to look at a smaller subset of the populations. In this case, we are specifically looking for CD3+ T cells that may mediate protection against M. tuberculosis infection. Immunologically, a protective population is unlikely to be present only in extremely small numbers. Therefore, in the filtering step of our pipeline we chose to filter to CD3+ T cell populations with population percentages greater than 0.5% in at least one sample. This analysis filtered the cells to look specifically at larger populations, but an alternative filter could be used to look at rare populations that compose <0.1% of the sample, for example.

After identifying all phenotypes, we can filter the data to see only the populations that we’re interested in, for example, CD3+ cells that constitute >0.5% of total live leukocytes in a sample.

sample\_populations <- all\_fe %>%  
 dplyr::filter(CD3 == 1 & percentage > 1) %>%  
 filter\_pops()

The filtering step has reduced the data to 14 populations.

## # A tibble: 14 x 19  
## IL\_10 TNF CD4 CD8 CTLA4 CD27 CD153 KLRG1 Sca1 CD3 IL\_17 CD62L CD122  
## <fct> <fct> <fct> <fct> <fct> <fct> <fct> <fct> <fct> <fct> <fct> <fct> <fct>  
## 1 0 0 0 0 0 0 0 0 1 1 0 0 0   
## 2 0 0 0 0 0 0 0 0 1 1 0 0 0   
## 3 0 0 0 0 0 0 1 0 1 1 0 0 0   
## 4 1 0 0 0 0 0 1 0 0 1 0 0 0   
## 5 0 0 0 0 0 0 1 0 1 1 0 0 0   
## 6 1 0 0 0 0 0 0 0 1 1 0 0 0   
## 7 1 0 0 0 0 0 0 0 0 1 0 0 0   
## 8 0 0 0 0 0 0 0 0 1 1 0 1 0   
## 9 0 0 0 0 0 0 0 0 1 1 0 0 0   
## 10 0 0 0 0 0 0 0 0 1 1 0 1 0   
## 11 0 0 0 0 0 0 0 0 1 1 0 0 0   
## 12 0 0 1 0 0 0 0 0 1 1 0 1 0   
## 13 0 0 1 0 0 0 0 0 1 1 0 0 0   
## 14 0 0 1 0 0 0 0 0 1 1 0 0 0   
## # … with 6 more variables: CD28 <fct>, PD1 <fct>, IFN <fct>, CD103 <fct>,  
## # CD44 <fct>, population <chr>

To calculate the number of cells in each sample with each of the above described phenoyptes, we use the identified\_pop\_per function.

sample\_populations\_all\_groups <- identified\_pop\_perc(sample\_populations, all\_fe, marker\_vector = order\_of\_markers)

## Joining, by = "population"

## Joining, by = c("filename", "IL\_10", "TNF", "CD4", "CD8", "CTLA4", "CD27", "CD153", "KLRG1", "Sca1", "CD3", "IL\_17", "CD62L", "CD122", "CD28", "PD1", "IFN", "CD103", "CD44")

We can view the first few rows of this data

## # A tibble: 6 x 3  
## population filename percentage  
## <chr> <chr> <dbl>  
## 1 Pop1 D30\_1\_A 2.70  
## 2 Pop1 D30\_1\_B 1.96  
## 3 Pop1 D30\_1\_C 1.19  
## 4 Pop1 D30\_2\_A 2.48  
## 5 Pop1 D30\_2\_B 4.02  
## 6 Pop1 D30\_2\_C 2.41

We may then want to visualize these flow populations.

library(pheatmap)  
library(tibble)  
  
simple\_pop\_df <- sample\_populations %>%  
 column\_to\_rownames("population")   
  
simple\_pop\_df %>%  
 dplyr::select(all\_of(order\_of\_markers)) %>%  
 mutate\_all(~convert\_factor\_numeric(.)) %>%  
 pheatmap::pheatmap(cluster\_rows = FALSE, cluster\_cols = FALSE,  
 labels\_row = rownames(simple\_pop\_df),  
 cellwidth = 15, cellheight = 15, angle\_col = 45,   
 color = c("#3F4788FF", "#56C667FF"), cutree\_rows = 2, legend = FALSE)

