Cytotypr Example: Flow Cytometry Cell Analysis

Amy Fox

## 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).

The goal of cytotypr is to identify flow cytometry cell populations efficiently using either Fluorescent Minus One controls (FMOs) or distinct population differences.

An in-depth description of the pipeline can be found in our paper [Cyto-Feature Engineering: A Pipeline for Flow Cytometry Analysis to Uncover Immune Populations and Associations with Disease](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.

## Installation

You can install the development version of cytotypr from [GitHub](https://github.com/) with:

if (!require("devtools")) install.packages("devtools")  
library(devtools)  
devtools::install\_github("aef1004/cytotypr")

There are a few additional packages that will need to be downloaded from CRAN:

if (!require("data.table")) install.packages("data.table")  
if (!require("dplyr")) install.packages("dplyr")  
if (!require("stringr")) install.packages("stringr")  
if (!require("scales")) install.packages("scales")  
if (!require("tidyr")) install.packages("tidyr")  
if (!require("superheat")) install.packages("superheat")  
if (!require("tibble")) install.packages("tibble")  
if (!require("pheatmap")) install.packages("pheatmap")  
if (!require("purrr")) install.packages("purrr")  
if (!require("broom")) install.packages("broom")

Other packages will need to be downloaded from bioconductor:

if (!requireNamespace("BiocManager", quietly = TRUE))  
 install.packages("BiocManager")  
  
if (!require("openCyto")) BiocManager::install("openCyto")  
if (!require("ncdfFlow")) BiocManager::install("ncdfFlow")  
if (!require("flowWorkspace")) BiocManager::install("flowWorkspace")  
if (!require("ggcyto")) BiocManager::install("ggcyto")

## Example

This is an full example which shows you how to obtain basic results and plots for flow cytometry data using FMOs. It starts by reading in the flow cytometry files, performing initial gating, and then performing the feature engineering and plotting the results.

Let’s load the necessary packages first.

library(cytotypr)

library(data.table)  
library(openCyto)  
library(ncdfFlow)  
library(flowWorkspace)  
library(dplyr)  
library(ggcyto)  
library(stringr)   
library(scales)  
library(tidyr)  
library(superheat)  
library(tibble)  
library(pheatmap)  
library(purrr)  
library(broom)

## 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.

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 resulting ‘ncdfFlowset’ object contains row names with the individual samples and column names with the markers/parameters used in the flow cytometer.

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.

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")))   
  
ncfs\_FMO  
#> An ncdfFlowSet with 20 samples.  
#> NCDF file : /var/folders/1d/nqs2fr2x23jfvdvb0y7cpr780000gp/T//Rtmpo3925f/ncfs1362456e8429b.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/1d/nqs2fr2x23jfvdvb0y7cpr780000gp/T//Rtmpo3925f/ncfs1362456e8429b.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>.

ws <- list.files("../inst/extdata",   
 pattern = "gating\_strategy.csv",   
 full = TRUE)  
ws  
#> [1] "../inst/extdata/gating\_strategy.csv"

We can view the initial gating strategy with the fread function. Here is what the data looks like once it has been read in. In this dataframe, each row in the .csv represents a different gating step.

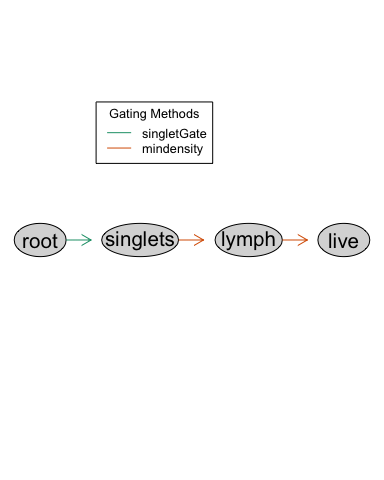
dtTemplate <- fread(ws)

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

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

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

initial\_gate <- gatingTemplate(ws)   
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.

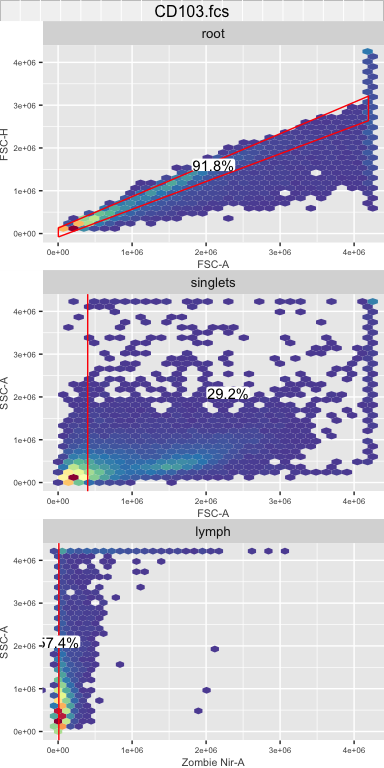
gs\_FMO <- GatingSet(ncfs\_FMO)  
gs\_FMO  
#> A GatingSet with 20 samples

Apply the initial gating to this data, to filter to only measurements on live lymphocyte cells. This may take a minute.

gt\_gating(initial\_gate, gs\_FMO)  
#> Gating for 'singlets'  
#> done!  
#> done.  
#> Gating for 'lymph'  
#> done!  
#> done.  
#> Gating for 'live'  
#> done!  
#> done.  
#> finished.

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.



## Convert data to “tidy data” format

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.

flowset\_FMO\_gated\_data <- gs\_pop\_get\_data(gs\_FMO, "live") %>%   
 cytoset\_to\_flowSet()

Apply the tidy\_flow\_set function to the ‘flowSet’ of gated FMO data to output a dataframe:

FMO\_gated\_data <- cytotypr::tidy\_flow\_set(flowset\_FMO\_gated\_data)

| filename | 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 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| CD103.fcs | 130 | 105,692 | 126,259.05 | 354,442 | 645,302.8 | -58.82495 | 294.6164 | -40.31233 | -1,501.9475 | -307.17886 | -10.97310 | -752.500793 | 78.63774 | -153.1160 | 1,033.5922 | -96.62029 | -40.02351 | -458.52356 | -295.87311 | 248.6988 | -479.8657 | 342.1824 | 165.03152 | -1.868633 | 175.06619 | 4,311.0996 | 804.2952 | -1,017.009 | 702.2532 | -4,217.877 | 188.0142 | -782.0732 | -983.7964 | -204.8062 | -225.62231 | 788.6584 | -699.6510 | 409.2456 | -729.2343 | -611.83710 | 124.0040 | -506.76807 | 789.7211 | 367.73785 | -242.7162 | 467.5756 | -12.22687 | 7,554.504 |
| CD103.fcs | 137 | 270,246 | 315,832.59 | 1,094,791 | 1,587,586.2 | -6,837.34766 | 180.9482 | 79.40491 | 952.2031 | -92.36914 | 250.72998 | 55.794922 | 936.36108 | 435.6587 | 125,317.9453 | 1,806.54541 | 3,185.22974 | 16,026.02734 | 284.83398 | -1,685.5421 | 4,191.6738 | 2,444.7927 | 2,849.67969 | -881.323975 | 22,939.68359 | 4,541.6484 | -8,557.2910 | -1,479.963 | 1,474.3097 | -3,915.632 | 171.2060 | -441.2710 | -562.6148 | 462.0623 | 569.74152 | 158,435.0156 | 1,651.1387 | 4,058.4014 | 20,026.4551 | -62.79362 | -1,609.4821 | 4,614.68311 | 3,183.6023 | 3,543.70410 | -855.6641 | 25,910.0078 | 101.11426 | 10,125.732 |
| CD103.fcs | 139 | 326,197 | 377,253.09 | 1,312,044 | 1,802,282.8 | -6,374.92285 | 2,437.9941 | -506.95679 | 6,024.4043 | 2,229.04907 | 52.42407 | 3,274.004395 | 1,000.97998 | 2,294.8848 | 142,746.4219 | 8,337.65234 | 3,257.13135 | 1,393.64014 | 1,707.43140 | 1,064.2981 | 1,056.0020 | 2,444.2676 | 1,183.84058 | 238.299652 | 15,421.60547 | -396.8066 | -10,934.6836 | 5,068.193 | -353.6312 | 11,241.875 | 3,370.5745 | 1,096.6729 | 4,036.4143 | 2,128.2139 | 3,485.33276 | 185,164.7500 | 11,073.0146 | 4,908.0371 | 4,053.7759 | 3,138.69507 | 907.2073 | 1,810.45337 | 4,592.5015 | 1,929.81848 | -734.6292 | 16,330.8379 | 51.66699 | -5,220.429 |
| CD103.fcs | 149 | 74,523 | 85,249.77 | 295,947 | 557,215.6 | 285.22537 | 542.3814 | 433.58603 | 224.1798 | 575.37128 | -42.57005 | 208.589935 | -241.81682 | 295.8936 | 408.3742 | -200.23586 | 665.90863 | -19.97466 | 72.43642 | -270.8827 | 356.8658 | 517.7744 | 556.95996 | -102.955399 | 51.51341 | 1,149.2915 | 402.7555 | -485.470 | 816.5948 | -2,637.060 | 744.1569 | -997.5383 | -186.4995 | -571.1957 | 60.79646 | 117.0377 | -740.8775 | 1,289.4952 | -148.6792 | -80.05383 | -693.9209 | 393.22012 | 1,377.3735 | 1,128.69739 | -191.9201 | -184.3577 | 146.24973 | 4,861.812 |
| CD103.fcs | 159 | 103,797 | 110,014.62 | 293,302 | 485,668.1 | -418.97879 | 250.4637 | 415.19464 | -126.7826 | -135.09859 | 106.53076 | -414.527130 | 2,192.60229 | -207.8182 | 880.5284 | -126.54059 | 491.77899 | 246.80374 | -451.05267 | 717.8876 | -552.2409 | 407.2867 | 580.07471 | -361.193268 | 269.77777 | 2,699.4885 | 307.5597 | -1,762.643 | 1,206.5282 | -5,485.723 | -140.7110 | -827.4388 | -810.7323 | 2,005.8848 | -449.09607 | 1,155.9196 | -925.6257 | 1,386.3148 | -902.6064 | -985.83081 | 254.3483 | 67.77438 | 1,280.7662 | 559.62573 | -839.1990 | 190.0231 | 557.78906 | 8,479.710 |
| CD103.fcs | 215 | 188,622 | 219,860.84 | 355,468 | 550,661.6 | -588.66052 | 1,265.1603 | -252.03519 | 453.2561 | -229.57965 | 500.05099 | 9.031342 | 422.34613 | -375.6987 | 12,304.8438 | -428.52786 | 1,185.76636 | -556.21375 | 159.39111 | -1,480.7100 | 2,798.2871 | 854.3411 | -16.06521 | -128.229553 | 638.52069 | 4,192.6396 | -119.9828 | -1,614.540 | 615.7618 | -6,470.897 | -676.4194 | -434.2599 | -675.4381 | -498.7168 | -940.79431 | 15,964.6777 | -1,940.1804 | 1,724.7844 | -2,114.6499 | -10.31729 | -2,219.9092 | 3,784.66797 | 1,916.4684 | 54.84417 | -369.9601 | 1,246.4841 | 648.56824 | 11,035.379 |

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 collection. 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 samples. 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 analysis.

In this step, we start by cleaning up the data by renaming the columns to the names of the markers rather than the fluors.

We also must ensure that all of the FMO filenames EXACTLY match the names of the column markers.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. That is what we are doing with the “IFN” column here, to ensure that the filename which says “IFN” is not labeled as “IFNG” and rather “IFN”

# 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"))

Filter the data for each FMO file to only contain information on the marker for that FMO

FMO\_filtered\_data <- filter\_FMO(df\_FMO\_gated\_data)

A snapshot of the data is shown here:

| SSC-A | filename | marker | MFI |
| --- | --- | --- | --- |
| 546,873.0 | IL\_10 | IL\_10 | -2,678.8357 |
| 255,673.0 | IL\_10 | IL\_10 | -1,537.1150 |
| 172,118.5 | IL\_10 | IL\_10 | -399.8506 |
| 4,116,310.2 | IL\_10 | IL\_10 | 4,161.5693 |
| 307,568.6 | IL\_10 | IL\_10 | -216.4338 |
| 639,882.9 | IL\_10 | IL\_10 | -2,994.9785 |

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.

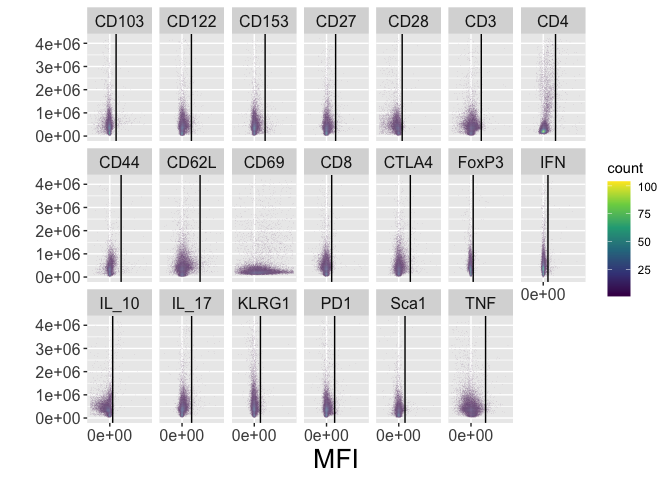
Here we create a dataframe of the cutoffs for each of the FMOs

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

| filename | quantile\_99 |
| --- | --- |
| CD103 | 3,201.716 |
| CD122 | 4,767.429 |
| CD153 | 5,505.384 |
| CD27 | 4,625.619 |
| CD28 | 1,739.658 |
| CD3 | 5,226.024 |
| CD4 | 6,243.004 |
| CD44 | 5,752.598 |
| CD62L | 9,162.103 |
| CD69 | 26,168.073 |
| CD8 | 2,621.496 |
| CTLA4 | 5,876.495 |
| FoxP3 | 1,113.967 |
| IFN | 2,397.020 |
| IL\_10 | 1,475.166 |
| IL\_17 | 4,871.126 |
| KLRG1 | 3,121.561 |
| PD1 | 4,120.256 |
| Sca1 | 3,178.948 |
| TNF | 7,404.389 |

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. 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 of your filenames and column names for each of the markers matches exactly as described above

plot\_FMOs(FMO\_filtered\_data, add\_quantile)  
#> Warning: Removed 25721 rows containing missing values (geom\_vline).



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.

## Reading in and gating the sample data

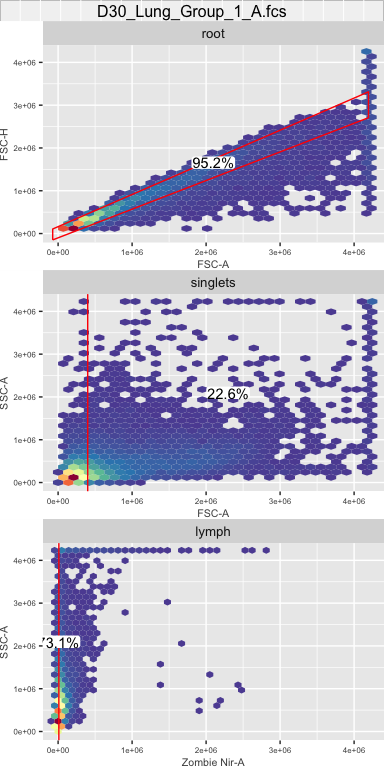
Up until this point, we have only worked with our control FMO data. We must now read in our samples and subsequently gate them and convert to tidy data using the same methodology as described for the FMOs.

Looking at the different timepoints for this study: D30, D60, and D90, some of the timepoints have the live-dead stain named slightly differently. Some samples are labeled “Zombie NIR-A”, others are labeled as “Zombie Nir-A.” This small difference in capitalization means that they will not be recognized as the same marker, so we add in a argument for “channel\_alias” which can standardize the names.

fcsFiles <- list.files("../inst/extdata/Tcell\_samples",   
 pattern = ".fcs", full = TRUE)  
  
# ncdfFlowset object contains row names with the individual samples and column names with the markers/parameters used in the flow cytometer  
ncfs <- read.ncdfFlowSet(fcsFiles, channel\_alias = data.frame(alias = c("Zombie Nir-A"), channels = c("Zombie\_NIR-A, Zombie NIR-A, Zombie Nir-A")))   
  
# apply gating set  
gs <- GatingSet(ncfs)  
  
# gate the samples  
gt\_gating(initial\_gate, gs)

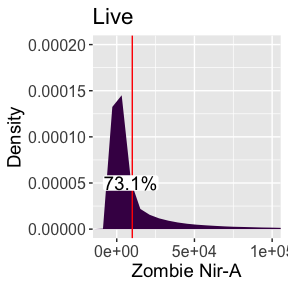
View and plot the gating

autoplot(gs[[1]])



An additional way to visualize some gates utilizes the ggcyto package

ggcyto(gs[[1]], aes(x = `Zombie Nir-A`)) +   
 geom\_area(stat = "density", fill = "#440154FF") +  
 geom\_gate("live") +  
 ggcyto\_par\_set(limits = list(y = c(0, 2e-4), x = c(-10000, 1e5))) +  
 xlab("Zombie NIR-A") +  
 ylab("Density") +  
 ggtitle("Live") +  
 theme\_gray() +  
 geom\_stats(size = 5, adjust = 0.45, position = position\_nudge(x = 10000)) +  
 labs\_cyto("both") +   
 theme(strip.text = element\_blank(),  
 axis.text = element\_text(size = 12),  
 axis.title = element\_text(size = 14),  
 title = element\_text(size = 14))+  
 scale\_x\_continuous(n.breaks = 3)  
#> Coordinate system already present. Adding new coordinate system, which will replace the existing one.

 Pull out the gated data

Want to change gated\_flowset to flowset\_gated\_data

# Pull out the gated data   
flowset\_gated\_data <- gs\_pop\_get\_data(gs, "live") %>%   
 cytoset\_to\_flowSet()   
  
# tidy the flowset and convert to a dataframe  
df\_all\_gated <- cytotypr::tidy\_flow\_set(flowset\_gated\_data)   
  
# check that samples were read in  
unique(df\_all\_gated$filename)  
#> [1] "D30\_Lung\_Group\_1\_A.fcs" "D30\_Lung\_Group\_1\_B.fcs" "D30\_Lung\_Group\_1\_C.fcs"  
#> [4] "D30\_Lung\_Group\_2\_A.fcs" "D30\_Lung\_Group\_2\_B.fcs" "D30\_Lung\_Group\_2\_C.fcs"  
#> [7] "D60\_Lung\_Group\_1\_A.fcs" "D60\_Lung\_Group\_1\_B.fcs" "D60\_Lung\_Group\_1\_C.fcs"  
#> [10] "D60\_Lung\_Group\_2\_A.fcs" "D60\_Lung\_Group\_2\_B.fcs" "D60\_Lung\_Group\_2\_C.fcs"  
#> [13] "D90\_Lung\_Group\_1\_A.fcs" "D90\_Lung\_Group\_1\_B.fcs" "D90\_Lung\_Group\_1\_C.fcs"  
#> [16] "D90\_Lung\_Group\_2\_A.fcs" "D90\_Lung\_Group\_2\_B.fcs" "D90\_Lung\_Group\_2\_C.fcs"

A snapshot of df\_all\_gated is shown below

| filename | Time | SSC-H | SSC-A | FSC-H | FSC-A | 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 |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| D30\_Lung\_Group\_1\_A.fcs | 241 | 108,112 | 111,048.1 | 223,098 | 463,057.5 | -1,140.6060 | 2,131.4758 | 329.1169 | 1,531.28564 | 727.83228 | 50.72234 | 300.7797 | 759.1093 | -234.0465 | 26,042.1094 | 1,961.6572 | 1,247.2238 | 415.80133 | 355.54547 | -646.3031 | 517.135925 | 1,381.105 | 953.0960 | 499.4547 | 291.7349 | 818.21313 | 2,490.905 |
| D30\_Lung\_Group\_1\_A.fcs | 243 | 127,457 | 132,772.8 | 230,903 | 483,070.1 | 389.1002 | 1,403.9232 | 1,319.2124 | 2,789.33008 | 89.82018 | 611.91895 | -120.9506 | 1,427.7250 | -324.2108 | 692.3067 | 886.9367 | 1,970.3325 | 82.14673 | 477.78696 | -664.5067 | 357.202850 | 1,680.539 | 1,455.7920 | -571.9879 | 682.2295 | -283.33997 | 2,143.309 |
| D30\_Lung\_Group\_1\_A.fcs | 251 | 336,441 | 389,882.5 | 615,609 | 990,053.9 | 569.1309 | 2,624.9126 | 875.4982 | 3,596.64404 | 790.45508 | -159.70485 | 578.4697 | 2,466.1636 | 422.9450 | 11,738.2842 | 2,501.9185 | 1,837.1056 | 63.04644 | 1,138.63171 | -744.3511 | 2,669.518555 | 2,586.496 | 1,897.6735 | -535.3110 | 763.1842 | -75.29114 | 4,663.604 |
| D30\_Lung\_Group\_1\_A.fcs | 255 | 140,812 | 141,888.7 | 206,304 | 441,986.2 | 938.0854 | 558.6160 | 236.2867 | 86.31412 | -491.85269 | -192.89258 | -587.6507 | 186.7440 | -110.2828 | -124.3056 | -258.1862 | 1,636.0659 | -98.25468 | -354.39343 | 317.7125 | 7.010204 | 1,079.478 | 1,260.6174 | -715.3037 | 336.3422 | -265.17773 | 4,283.675 |
| D30\_Lung\_Group\_1\_A.fcs | 257 | 119,122 | 117,973.1 | 280,611 | 500,140.0 | 159.2994 | -849.1758 | 1,347.4342 | -5,956.10498 | -127.54216 | -1,526.02075 | -953.0167 | 341.4515 | -984.2551 | -347.2966 | -877.5103 | 299.1937 | -1,201.81702 | -600.08221 | -629.2517 | -198.639099 | 1,527.334 | 816.6185 | -472.5444 | 262.4094 | 1,825.01587 | 13,980.912 |
| D30\_Lung\_Group\_1\_A.fcs | 258 | 133,957 | 161,027.5 | 1,496,930 | 1,995,630.0 | -276.4789 | 2,810.3467 | 159.7702 | 1,448.49402 | -653.72766 | 764.91193 | -547.5991 | 900.9947 | -401.0734 | 5,599.5991 | -581.7596 | 1,896.4182 | -122.92078 | -24.76933 | 313.5924 | -605.264404 | 1,749.789 | 1,203.1456 | 902.0891 | 525.9171 | 690.95746 | 5,910.558 |

## 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.

cleaned\_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") %>% # clean up the column names  
 dplyr::filter(`SSC-A` != max(`SSC-A`))

| SSC-A | IL\_10 | TNF | CD4 | CD69 | CD8 | CTLA4 | CD27 | CD153 | KLRG1 | Sca1 | CD3 | IL\_17 | CD62L | CD122 | CD28 | PD1 | IFN | FoxP3 | CD103 | CD44 | Zombie Nir-A | AF-A | filename |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 111,048.1 | -1,140.6060 | 2,131.4758 | 329.1169 | 1,531.28564 | 727.83228 | 50.72234 | 300.7797 | 759.1093 | -234.0465 | 26,042.1094 | 1,961.6572 | 1,247.2238 | 415.80133 | 355.54547 | -646.3031 | 517.135925 | 1,381.105 | 953.0960 | 499.4547 | 291.7349 | 818.21313 | 2,490.905 | D30\_1\_A |
| 132,772.8 | 389.1002 | 1,403.9232 | 1,319.2124 | 2,789.33008 | 89.82018 | 611.91895 | -120.9506 | 1,427.7250 | -324.2108 | 692.3067 | 886.9367 | 1,970.3325 | 82.14673 | 477.78696 | -664.5067 | 357.202850 | 1,680.539 | 1,455.7920 | -571.9879 | 682.2295 | -283.33997 | 2,143.309 | D30\_1\_A |
| 389,882.5 | 569.1309 | 2,624.9126 | 875.4982 | 3,596.64404 | 790.45508 | -159.70485 | 578.4697 | 2,466.1636 | 422.9450 | 11,738.2842 | 2,501.9185 | 1,837.1056 | 63.04644 | 1,138.63171 | -744.3511 | 2,669.518555 | 2,586.496 | 1,897.6735 | -535.3110 | 763.1842 | -75.29114 | 4,663.604 | D30\_1\_A |
| 141,888.7 | 938.0854 | 558.6160 | 236.2867 | 86.31412 | -491.85269 | -192.89258 | -587.6507 | 186.7440 | -110.2828 | -124.3056 | -258.1862 | 1,636.0659 | -98.25468 | -354.39343 | 317.7125 | 7.010204 | 1,079.478 | 1,260.6174 | -715.3037 | 336.3422 | -265.17773 | 4,283.675 | D30\_1\_A |
| 117,973.1 | 159.2994 | -849.1758 | 1,347.4342 | -5,956.10498 | -127.54216 | -1,526.02075 | -953.0167 | 341.4515 | -984.2551 | -347.2966 | -877.5103 | 299.1937 | -1,201.81702 | -600.08221 | -629.2517 | -198.639099 | 1,527.334 | 816.6185 | -472.5444 | 262.4094 | 1,825.01587 | 13,980.912 | D30\_1\_A |
| 161,027.5 | -276.4789 | 2,810.3467 | 159.7702 | 1,448.49402 | -653.72766 | 764.91193 | -547.5991 | 900.9947 | -401.0734 | 5,599.5991 | -581.7596 | 1,896.4182 | -122.92078 | -24.76933 | 313.5924 | -605.264404 | 1,749.789 | 1,203.1456 | 902.0891 | 525.9171 | 690.95746 | 5,910.558 | D30\_1\_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 understand.

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.

We feature engineer the data with the “fe” function which pulls in the 99% threshold from the “add\_quantile” dataframe.

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.

In the resulting dataframe, ‘all\_fe’, each row is a different cell phenotype (defined by a combination of positive (1) and negative (0) expression of each marker). The “cell\_no” column indicates the number of cells with the corresponding phenotype in each of the samples (“filename”). The “percentage” of cells with each phenotype is calculated by dividing the “cell\_no” column by the “total\_count\_by\_file” column which indicates the total number of live single leukocytes in each sample.

all\_fe <- cleaned\_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")) %>% # feature engineer the data  
 select(-`Zombie Nir-A`, -`AF-A`, -`SSC-A`, -FoxP3, -CD69) %>% # any column that doesn't have a 0 or 1 must be removed here (Except experimental/group names)  
 count\_calc()

| IL\_10 | TNF | CD4 | CD8 | CTLA4 | CD27 | CD153 | KLRG1 | Sca1 | CD3 | IL\_17 | CD62L | CD122 | CD28 | PD1 | IFN | CD103 | CD44 | filename | cell\_no | total\_count\_by\_file | percentage |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | D30\_1\_A | 15,045 | 76,657 | 19.626388 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | D30\_1\_A | 29,786 | 76,657 | 38.856204 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | D30\_1\_A | 913 | 76,657 | 1.191020 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | D30\_1\_A | 1,885 | 76,657 | 2.459006 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | D30\_1\_A | 2,072 | 76,657 | 2.702950 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | D30\_1\_A | 78 | 76,657 | 0.101752 |

### Optional Check with Flowjo

If you want to check the results from cytotypr with results from manual gating in Flowjo, the function, check\_flowjo can be used which calculates the percentage of cells in each sample that express a specific marker (in this case, CD3)

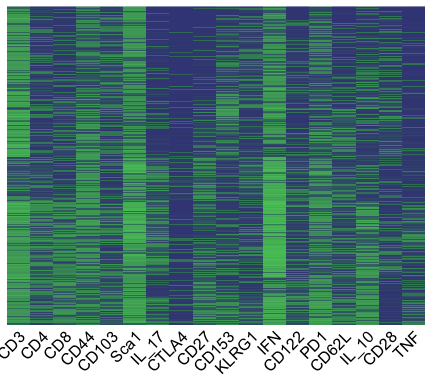
CD3\_flowjo <- check\_flowjo(all\_fe, `CD3`)

| CD3 | filename | sum\_percentage |
| --- | --- | --- |
| 1 | D30\_1\_A | 17.06563 |
| 1 | D30\_1\_B | 20.55821 |
| 1 | D30\_1\_C | 16.80377 |
| 1 | D30\_2\_A | 31.51011 |
| 1 | D30\_2\_B | 40.09170 |
| 1 | D30\_2\_C | 37.67492 |
| 1 | D60\_1\_A | 24.01919 |
| 1 | D60\_1\_B | 23.30415 |
| 1 | D60\_1\_C | 20.45215 |
| 1 | D60\_2\_A | 18.75385 |
| 1 | D60\_2\_B | 26.77780 |
| 1 | D60\_2\_C | 20.78296 |
| 1 | D90\_1\_A | 36.02344 |
| 1 | D90\_1\_B | 21.17572 |
| 1 | D90\_1\_C | 31.54755 |
| 1 | D90\_2\_A | 36.60608 |
| 1 | D90\_2\_B | 34.73928 |
| 1 | D90\_2\_C | 34.13437 |

## Visualizations: All cell phenotypes present in the samples

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 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.

# 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)



We can also quantify the total number of populations that were identified in all of the samples

nrow(total\_phenotypes)   
#> [1] 10912

## Visualizations: Filtered cell phenotypes of interest in the samples

After identifying all phenotypes, we can filter the data to see the ones that we’re interested in, for example, CD3+ cells that constitute > 1% of total live leukocytes in a 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.

# view the specific cell phenotypes we're interested in  
sample\_populations <- all\_fe %>%  
 dplyr::filter(CD3 == 1 & percentage > 1) %>%  
 filter\_pops()

The filtering step has reduced the data to 14 populations

| IL\_10 | TNF | CD4 | CD8 | CTLA4 | CD27 | CD153 | KLRG1 | Sca1 | CD3 | IL\_17 | CD62L | CD122 | CD28 | PD1 | IFN | CD103 | CD44 | population |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | Pop1 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | Pop2 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | Pop3 |
| 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | Pop4 |
| 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | Pop5 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | Pop6 |
| 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | Pop7 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 1 | Pop8 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | Pop9 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | Pop10 |
| 0 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 0 | 0 | 1 | Pop11 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | 0 | 0 | 0 | 1 | 0 | 0 | Pop12 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 1 | Pop13 |
| 0 | 0 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 1 | 0 | 0 | 0 | 0 | 0 | 1 | 0 | 1 | Pop14 |

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)

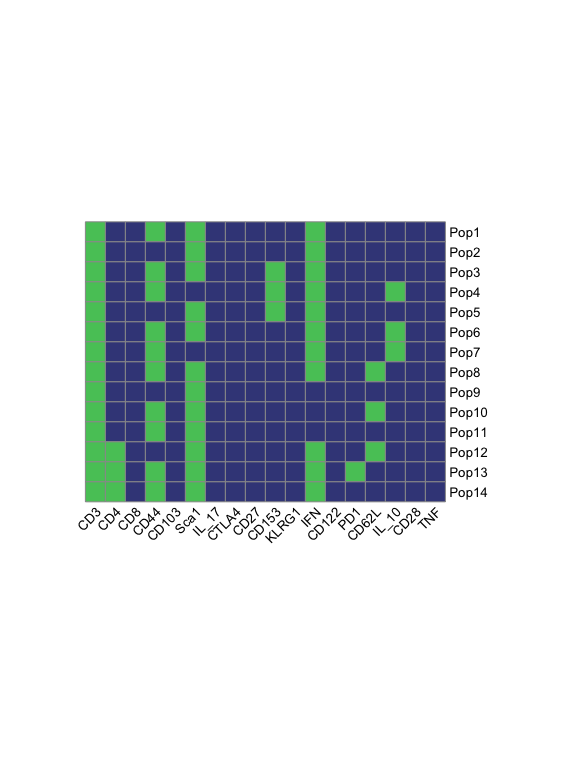
We can view the first few rows of the data

head(sample\_populations\_all\_groups) %>%  
 flextable()

| population | filename | percentage |
| --- | --- | --- |
| Pop1 | D30\_1\_A | 2.702950 |
| Pop1 | D30\_1\_B | 1.957861 |
| Pop1 | D30\_1\_C | 1.185667 |
| Pop1 | D30\_2\_A | 2.483240 |
| Pop1 | D30\_2\_B | 4.023175 |
| Pop1 | D30\_2\_C | 2.407824 |

Finally, we can visualize these flow populations. The heatmaps show the CD3+ phenotypes that constitute greater than 1% of the live leukocytes in at least one sample. Green indicates positive expression, and blue indicates negative expression of all markers used for analysis in the flow cytometry experiment on the x-axis.

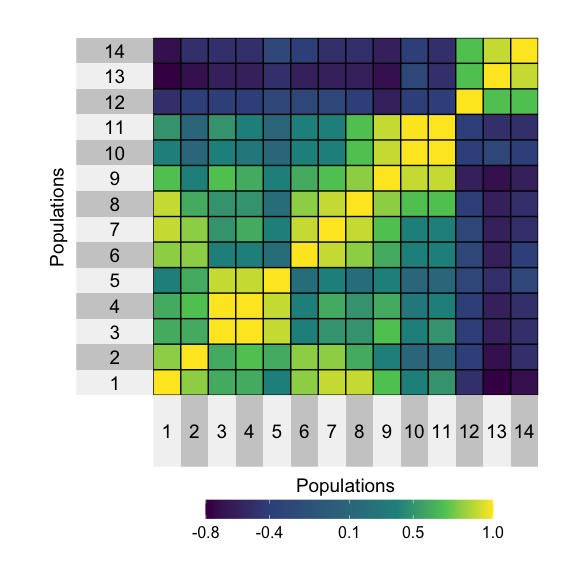
simple\_pop\_df <- sample\_populations %>%  
 column\_to\_rownames("population")   
  
simple\_pop\_df %>%  
 dplyr::select(all\_of(order\_of\_markers)) %>%  
 mutate\_all(~convert\_factor\_numeric(.)) %>%  
 pheatmap::pheatmap(cluster\_rows = FALSE, cluster\_cols = FALSE,  
 labels\_row = rownames(simple\_pop\_df),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



## Visualizations: Correlation plot

The pipeline then visualizes a correlation matrix comparing the percentage of cells in each of the populations. This allows users to explore associations between cell populations. Yellow indicates high positive correlation and purple is high negative correlation.

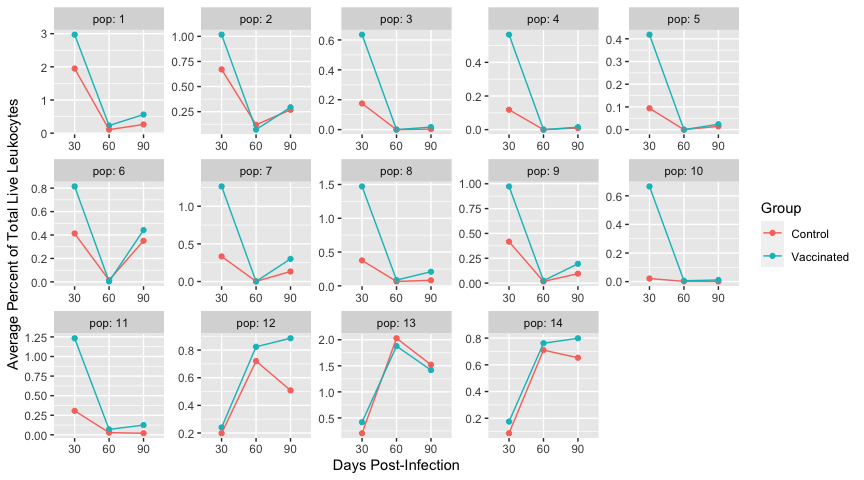
corr <- calc\_corr(sample\_populations\_all\_groups)  
  
superheat(corr, row.title = "Populations", column.title = "Populations")



## Visualizations: Time Series Plot

For this study, the percentage of cells in each population at each timepoint is plotted to compare the dynamic changes in populations over time and between groups. Each small plot shows the time series of a single cell population identified in the pipeline at the three measured timepoints post-infection. Separate lines are shown for vaccinated versus control mice. Each point represents average cell populations across all mouse replicates (4–5 per time point and vaccination status).

# take the data for filtered populations and rename so that plots are pretty  
  
pops\_for\_plots\_average <- sample\_populations\_all\_groups %>%  
 tidyr::separate(filename, into = c("Timepoint", "Group", "Number"),  
 sep = "\_") %>%  
 dplyr::group\_by(population, Timepoint, Group) %>%  
 dplyr::summarise(average\_percent = mean(percentage)) %>%  
 dplyr::mutate(Group = str\_replace(Group, "1", "Control"),  
 Group = str\_replace(Group, "2", "Vaccinated")) %>%  
 dplyr::mutate(Timepoint = str\_extract(Timepoint, "[0-9].+")) %>%  
 dplyr::mutate(pop = as.numeric(str\_extract(population, "[:digit:]+")))   
#> `summarise()` has grouped output by 'population', 'Timepoint'. You can override using the `.groups` argument.  
  
  
ggplot(pops\_for\_plots\_average, aes(x = Timepoint, y = average\_percent,   
 group = Group, color = Group)) +  
 scale\_fill\_identity() +  
 geom\_point() +  
 geom\_line() +  
 facet\_wrap("pop", scales = "free", ncol = 5, labeller = label\_both) +  
 xlab("Days Post-Infection") +  
 ylab("Average Percent of Total Live Leukocytes") +  
 theme\_gray()



## Visualizations: Correlation with other experimental data

At this stage, the pipeline allows the integration of cell population measurements with other data from the experiment, such as lesion scores or gene expression. In the M. tuberculosis study, bacterial burden (expressed as log10 transformed Colony Forming Units (CFUs)) is a measurement of the number of bacteria found in the lung. These CFU measurements were found to vary between experimental groups in the case study data, with significantly higher bacterial burden at days 30, 60, and 90 post-infection in the control group compared to the vaccinated group. It is of interest to investigate if certain cell populations identified through the pipeline, are associated with this measurement of bacterial burden, as this might help to identify cell populations possibly indicative of the host’s response to infection with or without vaccination.

We start by reading in the CFU data which is saved in an excel file and accessing the flow data (“sample\_populations\_all\_groups”) and joining the two datasets.

CFUs <- readxl::read\_xlsx("../inst/extdata/CFU\_data.xlsx") %>%  
 dplyr::filter(Organ == "Lung") %>%  
 dplyr::filter(Group == "1" | Group == "2")  
  
# clean the flow data and prepare to join with CFU  
pops\_for\_CFUs <- sample\_populations\_all\_groups %>%  
 separate(filename, into = c("Timepoint", "Group", "Number"), sep = "\_") %>%  
 dplyr::mutate(Timepoint = str\_replace(Timepoint, "D", "")) %>%  
 dplyr::mutate(Timepoint = as.numeric(Timepoint),  
 Group = as.numeric(Group))   
  
# join together the CFU data and the population data  
pops\_CFUs <- inner\_join(pops\_for\_CFUs, CFUs, by = c("Group", "Number", "Timepoint")) %>%  
 dplyr::mutate(pop = as.numeric(str\_extract(population, "[:digit:]+"))) %>%  
 select(-population)

Here we group and nest the data by the different populations and perform linear regressions on the data

fitted\_models <- pops\_CFUs %>%  
 group\_by(pop) %>%  
 nest() %>%  
 dplyr::mutate(model = map(data, ~lm(percentage ~ CFU, data = .)),  
 summary\_model = map(model, tidy)) %>%  
 unnest(summary\_model) %>%  
 dplyr::filter(term == "CFU") %>%  
 dplyr::select(pop, estimate, model) %>%  
 dplyr::mutate(tidy\_model = map(model, broom::glance)) %>%  
 unnest(tidy\_model) %>%  
 dplyr::select(pop, adj.r.squared, p.value, estimate) %>%  
 ungroup()

Because we’re performing many comparisons, we apply a multiple comparisons correction, in this case “BH” or Benjamini & Hochberg to adjust our p-values.

values\_for\_plot <- fitted\_models %>%  
 mutate(p.val.adj = p.adjust(p.value, method = "BH",   
 n = length(fitted\_models$p.value)))

| pop | adj.r.squared | p.value | estimate | p.val.adj |
| --- | --- | --- | --- | --- |
| 1 | 0.62425074 | 0.0000580774 | -0.8500766 | 0.0008130836 |
| 10 | 0.11349525 | 0.0936919370 | -0.1385980 | 0.1008990091 |
| 11 | 0.22031437 | 0.0284012031 | -0.2905024 | 0.0331347369 |
| 12 | 0.03528601 | 0.2210314633 | 0.1028211 | 0.2210314633 |
| 13 | 0.49936519 | 0.0006272798 | 0.4885339 | 0.0043909586 |
| 14 | 0.24628380 | 0.0209638249 | 0.1723344 | 0.0293493549 |
| 2 | 0.39578209 | 0.0030679223 | -0.2574319 | 0.0071584855 |
| 3 | 0.45633346 | 0.0012535998 | -0.1807536 | 0.0048251858 |
| 4 | 0.41830094 | 0.0022207775 | -0.1537443 | 0.0062181771 |
| 5 | 0.26277516 | 0.0172227558 | -0.1213729 | 0.0267909535 |
| 6 | 0.23316246 | 0.0244607406 | -0.1574129 | 0.0311318517 |
| 7 | 0.36211183 | 0.0048803506 | -0.2889993 | 0.0097607012 |
| 8 | 0.34649944 | 0.0060101232 | -0.3247976 | 0.0105177156 |
| 9 | 0.45016942 | 0.0013786245 | -0.2432635 | 0.0048251858 |

First we calculate where we want to place our labels on our ggplot. We calculate where the label will fall on the x-axis for each population by finding the range of the x-values and dividing by 2 to find the center. For the location on the y-axis, we place the label as the smallest y-axis value if the relationship is positive, and the largest y-axis value is the relationship is negative.

r\_labeling <- left\_join(pops\_CFUs, values\_for\_plot, by = "pop") %>%  
 group\_by(pop) %>%  
 mutate(average\_percentage = mean(percentage)) %>%  
 mutate(x\_axis\_label = (max(percentage) - min(percentage))/2,  
 y\_axis\_label = ifelse(estimate <= 0, max(CFU), min(CFU))) %>%  
 select(pop, average\_percentage, adj.r.squared, p.val.adj,   
 x\_axis\_label, y\_axis\_label) %>%  
 unique()

| pop | average\_percentage | adj.r.squared | p.val.adj | x\_axis\_label | y\_axis\_label |
| --- | --- | --- | --- | --- | --- |
| 1 | 1.01343934 | 0.62425074 | 0.0008130836 | 1.9705894 | 6.993161 |
| 10 | 0.11795938 | 0.11349525 | 0.1008990091 | 0.8084388 | 6.993161 |
| 11 | 0.29700240 | 0.22031437 | 0.0331347369 | 1.3531122 | 6.993161 |
| 12 | 0.56287672 | 0.03528601 | 0.2210314633 | 0.6184283 | 2.788875 |
| 13 | 1.24504447 | 0.49936519 | 0.0043909586 | 0.9460252 | 2.788875 |
| 14 | 0.53099300 | 0.24628380 | 0.0293493549 | 0.5957591 | 2.788875 |
| 2 | 0.40692674 | 0.39578209 | 0.0071584855 | 0.7178250 | 6.993161 |
| 3 | 0.13849667 | 0.45633346 | 0.0048251858 | 0.5638512 | 6.993161 |
| 4 | 0.11795553 | 0.41830094 | 0.0062181771 | 0.5081218 | 6.993161 |
| 5 | 0.09174315 | 0.26277516 | 0.0267909535 | 0.5253323 | 6.993161 |
| 6 | 0.33980750 | 0.23316246 | 0.0311318517 | 0.6498108 | 6.993161 |
| 7 | 0.33900185 | 0.36211183 | 0.0097607012 | 1.0334355 | 6.993161 |
| 8 | 0.38151764 | 0.34649944 | 0.0105177156 | 1.0542350 | 6.993161 |
| 9 | 0.28673882 | 0.45016942 | 0.0048251858 | 0.7658168 | 6.993161 |

We then plot the data where each small plot shows the association between a specific cell population and bacterial burden across all samples for the experiment. The x-axis in each small plot gives the percentage of cells in a population, with each point providing the measurement from a single mouse. The y-axis gives the log10 M. tuberculosis CFUs for that mouse. Estimates of how well the linear models fit the data between cell population sizes and log10 CFUs are given on each plot (“r2”).

ggplot(pops\_CFUs) +  
 scale\_fill\_identity() +  
 geom\_point(aes(percentage, CFU), color = "black") +  
 geom\_smooth(aes(x = percentage, y = CFU), method = "lm", se = FALSE, color = "#3F4788FF") +  
 geom\_text(data = r\_labeling, aes(x = x\_axis\_label, y = y\_axis\_label,   
 label = paste("r^2 == ",  
 round(adj.r.squared, 2))),  
 parse = TRUE) +  
 facet\_wrap(~pop, scales = "free\_x", ncol = 5, labeller = label\_both) +  
 xlab("Percentage of Cells") +  
 ylab("log10 CFU") +  
 ggtitle("Population and CFU Linear regression")   
#> `geom\_smooth()` using formula 'y ~ x'

