CytoAnalyze

Ramy Gadalla

This vignette demonstrates the processing steps of cytometry data using the workflow in CytoAnalyze package.

```
library(CytoAnalyze)
```

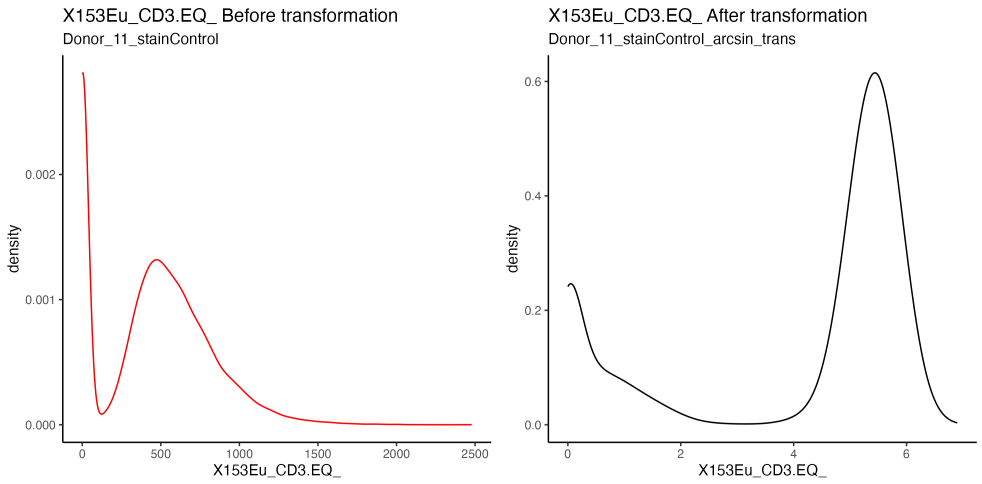
For the demonstration of the functions of CytoAnalyze package, data from *Gadalla et al* 2019 is used here.

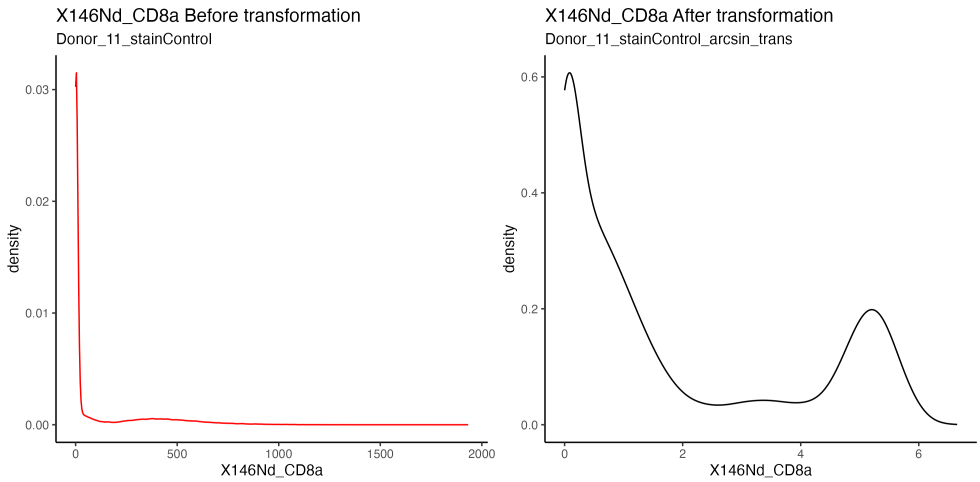
Arcsin transformation

First, we start with the arcsin transformation of the data, using arcsin_trans() function.

```
#arcsin_trans(input_folder = input_folder <- "~/Data/clean_fcs",
#              cofactor = 5,
#              #value that controls the extent of the linear region of the scale.
 #             output_folder = "~/Data/CytoAnalyze_results",
              channel.plot = TRUE
#              #If True, export histograms for visual inspection of the transformation.
#               )
```

Transformed FCS files are exported to the specified output location, with histograms of one randomly selected file to show plot all the channels before and after transformation. Here only CD3 and CD28 are shown before and after transformation. You can always go back and try different cofactor value of arcsin transformation if transformation does not result in satisfactory resolution.



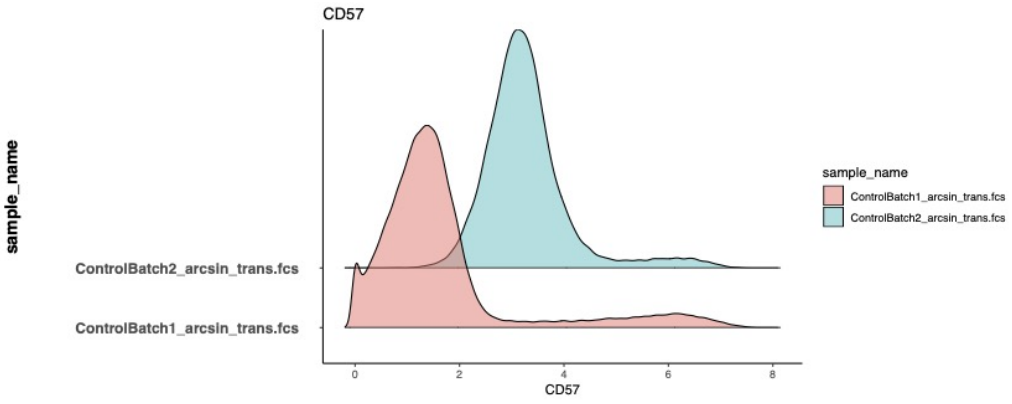


Staining quality check

Typically, if samples were acquired over multiple batches of samples, it is a good practice to check the extent of batch variation. plot_stainQC() plot channels of all markers in each batch control samples for visual inspection.

```
#plot_stainQC(input_folder = "~/Data/CytoAnalyze_results/ArcsinTrans", #arcsin transformed files
#              pattern = "ControlBatch", # keyword in the control file names
#              output_folder = "~/Data/CytoAnalyze_results")
```

Stacked histograms of all channels in the data are plotted and exported to the specified location. We are showing here only CD57 and Helios.



The histograms show variation between batch controls in this case. This will need to be corrected by batch correction tools.

Generation of metadata tables

Next, metadata tables need to be created for the experiments in the dataset and the panel of markers used. This is achieved by metadata_tables() function, which reads in the data and outputs two tables to be filled in and curated by the analyst. The first table is experiment_info.csv which contains metadata about the experiments, the groups, the sample ID..ec The analyst is encouraged to add as much data as possible to this table. Second table is marker_info.csv, which includes metadata about the markers used in the panel. Mandatory fields in the table that have to be filled in by the analyst are channel_name, marker_name. clean_name, marker_class, channel_type. Additional columns could be added to this table as needed.

```
#metadata_tables(input_folder = "~/Data/CytoAnalyze_results/ArcsinTrans",
#                 output_folder = "~/Data/CytoAnalyze_results/RDS",
#                 clean_name = TRUE
 #                )
```

The analyst must fill in these tables according to metadata for each experiment. Here, we will read the tables to show an example. In general, it is a good practice to to have any NA values.

```
# marker_info table
#marker_info <- read.csv("~/Data/CytoAnalyze_results/metadata_tables/marker_info.csv")
#head(marker_info, 10)
```

Extra fields/column can be added, but it is crucial to keep the names and the order of the first 5 columns unchanged. “marker_class” values can be either “none”, “state” or “type”. “channel_type” values can be either “biological” or “non_biolgoical”

```
#experiment_info <- read.csv("~/Data/CytoAnalyze_results/metadata_tables/experiment_info.csv")
#head(experiment_info, 8)
```

First column of experiment_info will contain the fcs filenames (Don’t change the order of the rows!). The analyst must add the rest of the metadata if available.

Down-sampling

Next step would be down-sampling, if needed. CytoAnalyze, adapts the density-dependent down-sampling approach of SPADE algorithm. It aims to preserve rare populations from getting eliminated from the sample by random down-sampling. The local density value for each cell is calculated, and the downsampling thresholds are provided in terms of density percentile.

```
#DDDsample(input_folder = "~/Data/CytoAnalyze_results/ArcsinTrans" ,
#          output_folder = "~/Data/CytoAnalyze_results",
#          marker_info = "~/Data/CytoAnalyze_results/metadata_tables/marker_info.csv",
#          experiment_info = "~/Data/CytoAnalyze_results/metadata_tables/experiment_info.csv",
#          density_to_exclude = 0.01,    # default
#          density_to_preserve = NULL,
##           percent_events_keep = NULL,   # default
#          number_events_keep = 10000
 #         )
```

If argument "number_events_keep" is used to down-sample fixed number of cells from each file, the function will skip the files that have cell count below the value provided in the argument and down-sample files that have count above the value provided.

Batch Correction

Batch correction is done using 'cyComboine' and will be demonstrated in the automated workflow section. It is recommended to check whether batch effect exists. To do so and define the markers that need correction, the detect_batch_effect() from cyCombine can be used.

Creating SummarizedExperiment object

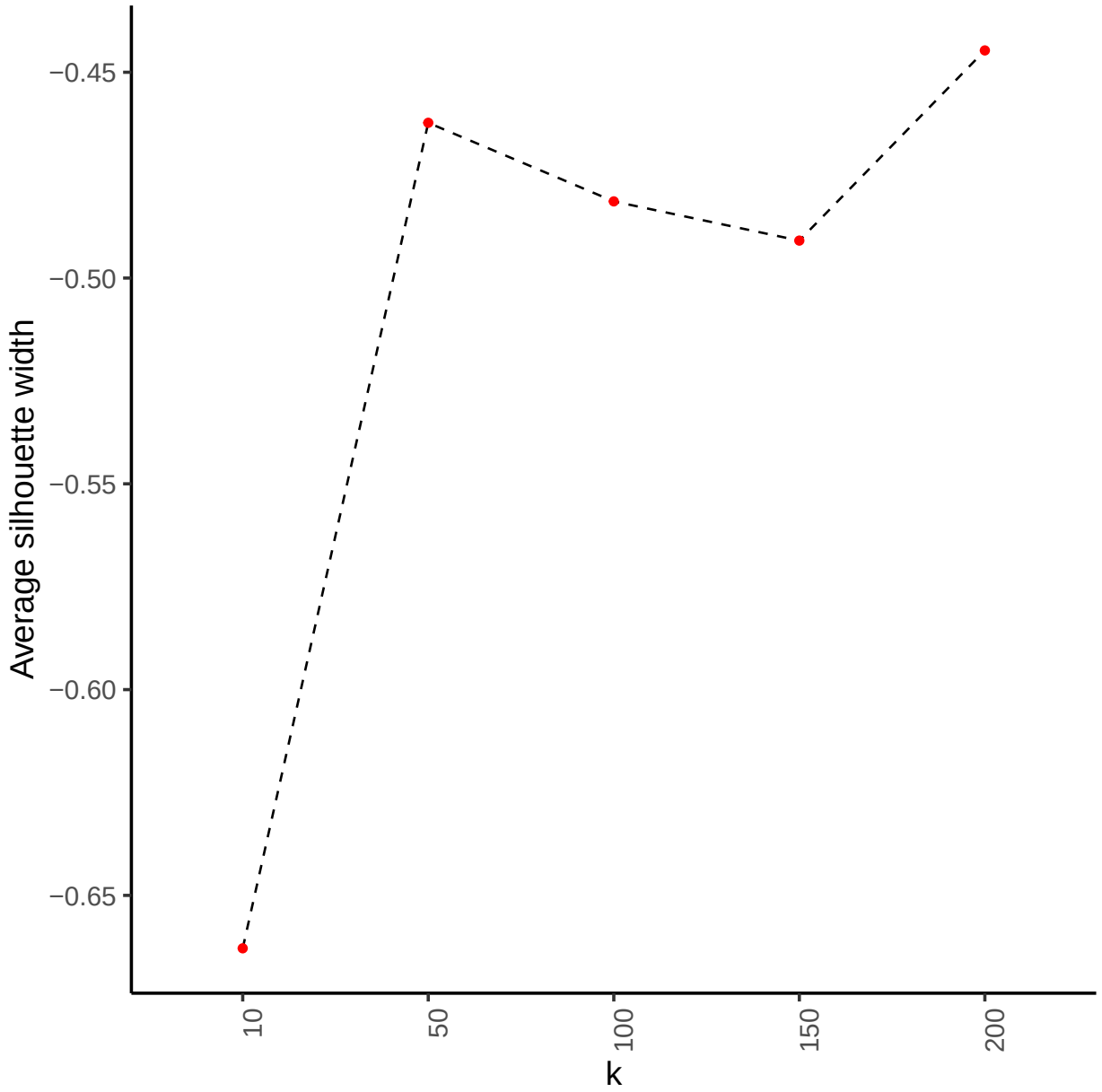
Here, the dataset is put into SummarizedExperiment-class object for subsequent analysis. SummarizedExperiment makes it easier to work with the dataset as a whole and perform specification operations.

```
#SE <- build_SE(input_folder = "~/Data/CytoAnalyze_results/downsample_output" ,
#          experiment_info = "~/Data/CytoAnalyze_results/metadata_tables/experiment_info.csv" ,
#          marker_info = "~/Data/CytoAnalyze_results/metadata_tables/marker_info.csv" ,
#          output_folder = "~/Data/CytoAnalyze_results",
#          export = TRUE)
#SE
```

Clustering parameter fine tuning.

In this step, we aim to determine the optimal k parameter for running clustering algorithm phenograph, using k_sweep(). This function from CytoAnalyze is based on clusterSweep from bluster package. Please check out bluster documentation for more information. The computation time required will depende on the range of k being tested and the number of cells in the dataset.

```
#k = floor(seq(50,300, length.out=10))
#k <- c(90, 130, 200 )
#k_sweep(SE,
#        k,
#        export = TRUE,
#        output_folder = "~/Data/CytoAnalyze_results",
#        assay_name = "exprs"
 #       )
```



The function returns silhouette average plot of the sequence of k values entered. The optimal k should be the value that maximize the silhouette distance between cluster. This, however, tends to favor the global/broad clustering over local data structure, so analyst is encouraged to make a judgment call here. This serves only

to provide a rough value of optimal k.

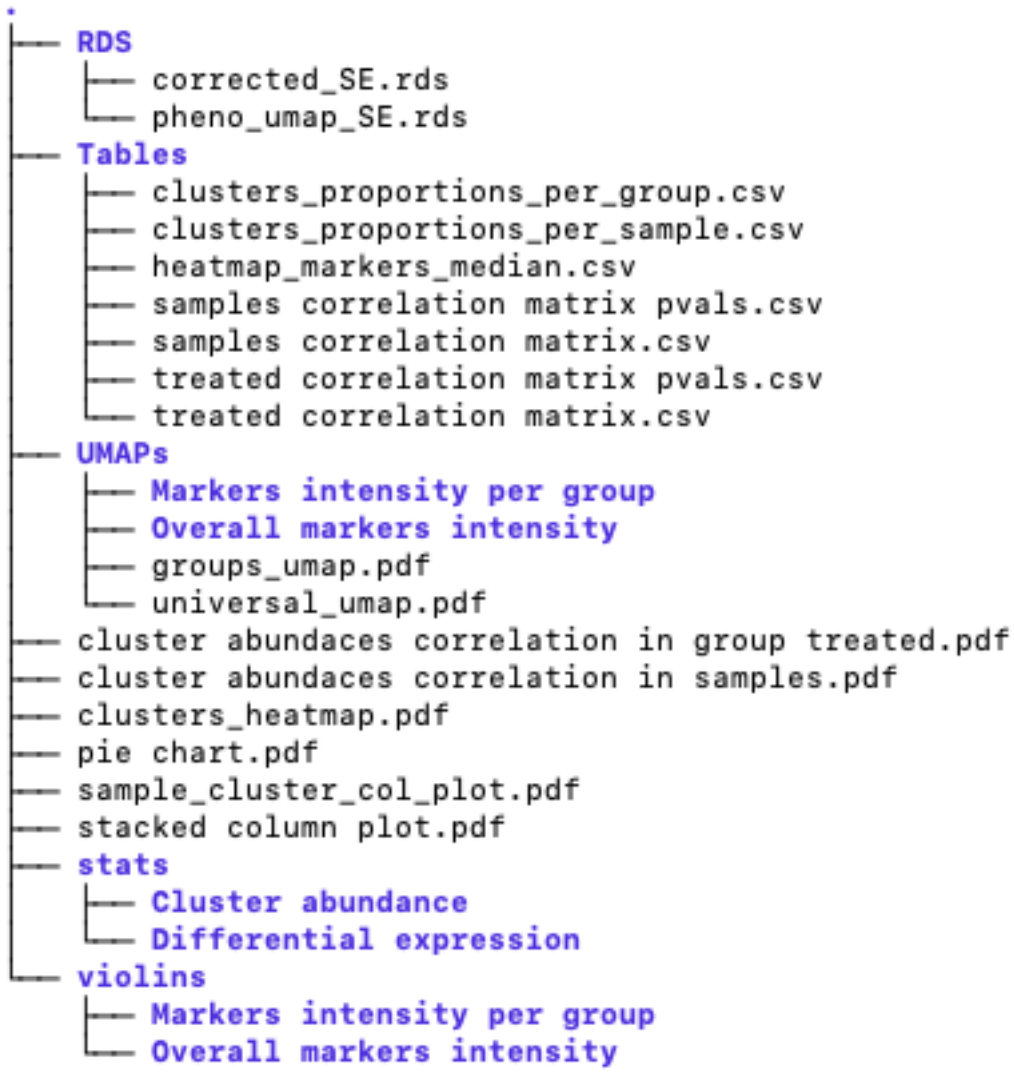
CyTOF workflow/pipeline

This section shows the functionality of cyto_workflow(). Now, after the data has been through transformation, downsampling (if required), and tuning parameter k for optimal clustering as shown above, the data is ready to go through the analysis pipeline. Please note that the above steps are not needed to run the analysis pipeline, it is only recommended, and the analysis pipeline can stand on its own. This structure of the package is due to the fact that the above steps typically need human supervision.

```
cyto_workflow (input_folder = "~/Data/CytoAnalyze_results/downsample_output" ,
               SE = NULL,  # corrected SummarizedExperiment
               output_folder = "~/Data/CytoAnalyze_results/workflow_result",
               marker_info = "~/Data/CytoAnalyze_results/metadata_tables/marker_info.csv",
               experiment_info = "~/Data/CytoAnalyze_results/metadata_tables/experiment_info.csv",
               batch_correct = TRUE,
               markers_correct = c("CD57", "Helios"),
               xdim = 8,         #default value
               ydim = 8,         #default value
               k = 30,           #default value
               n = 5,
               assay_name = "exprs",  #default
               plot_clusters = TRUE,
               groups = "treatment",
               exp_subject = "donor_id",
               plot_samples_prop = TRUE,
               samples = "donor_id",
               plot_clusters_violin = TRUE,
               plot_cluster_corr = TRUE,
               plot_cluster_colsPie = TRUE,
               plot_cluster_hm = TRUE,
               diff_exprs = TRUE,
               count_threshold = 100,
               diff_abund = TRUE,
               diff_limma = TRUE,
               contrast = c("treated", "untreated"),
               paired = FALSE)
```

cyto_workflow() is the wrapper function that streamline CytoAnalyze workflow. The output of workflow pipeline is exported to the specified location. For more details on the function arguments, please check out the package documentations. Here is an example of the output directory structure.

In addition to this wrapper function, CytoAnalyze also contain separate functions that can be used outside of the pipeline.



The function outputs a variety of data visualizations and statistical analysis. Directory RDS will contain all the rds objects exported by the pipeline e.g batch corrected SummarizedExperiment object if performed and SummarizedExperiment object with the added clustering and dimensions reduction data.

Visualization

Directory UMAPS will contain all umap/phenograph figures exported by the pipeline. Based on the input arguments, such as the experimental groups and the markers to analyze, the pipeline will generate a number of umap/phenograph figures. Here are few examples:

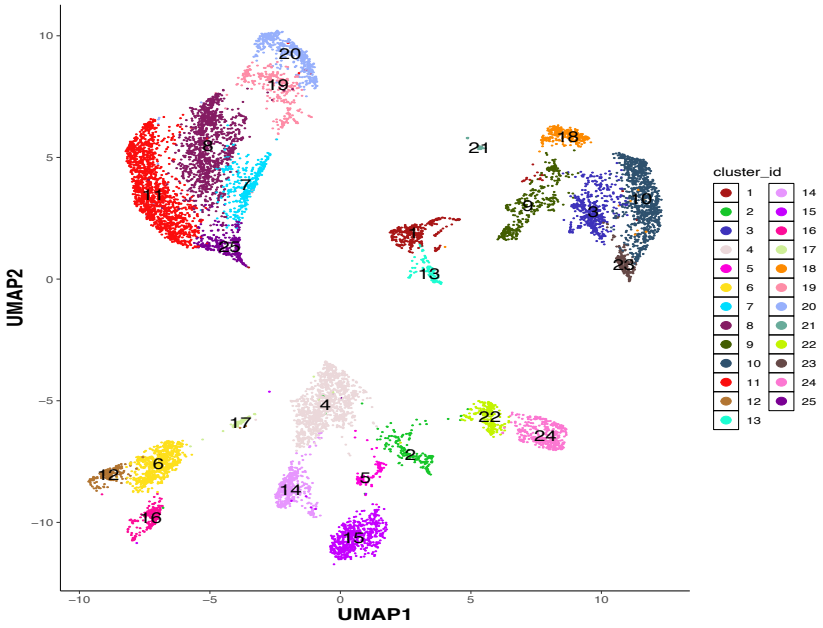


Figure 1: Universal UMAP

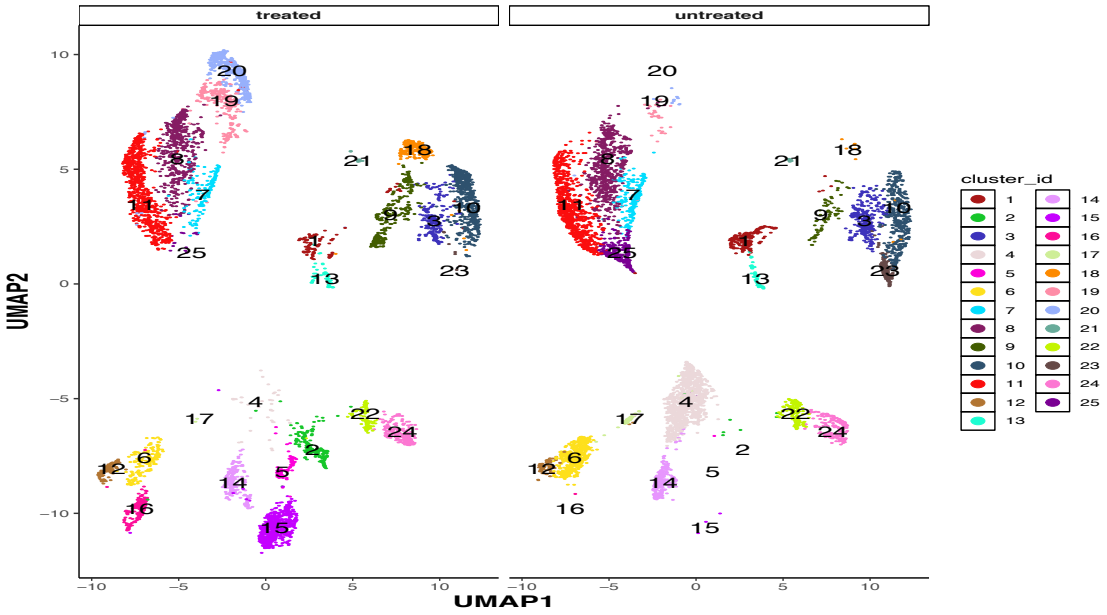


Figure 2: Groups UMAP

Overall marker expression intensity umap/phenograph figures and marker intensity per group figures are also generated for the specified markers. All markers are plots unless otherwise specified.

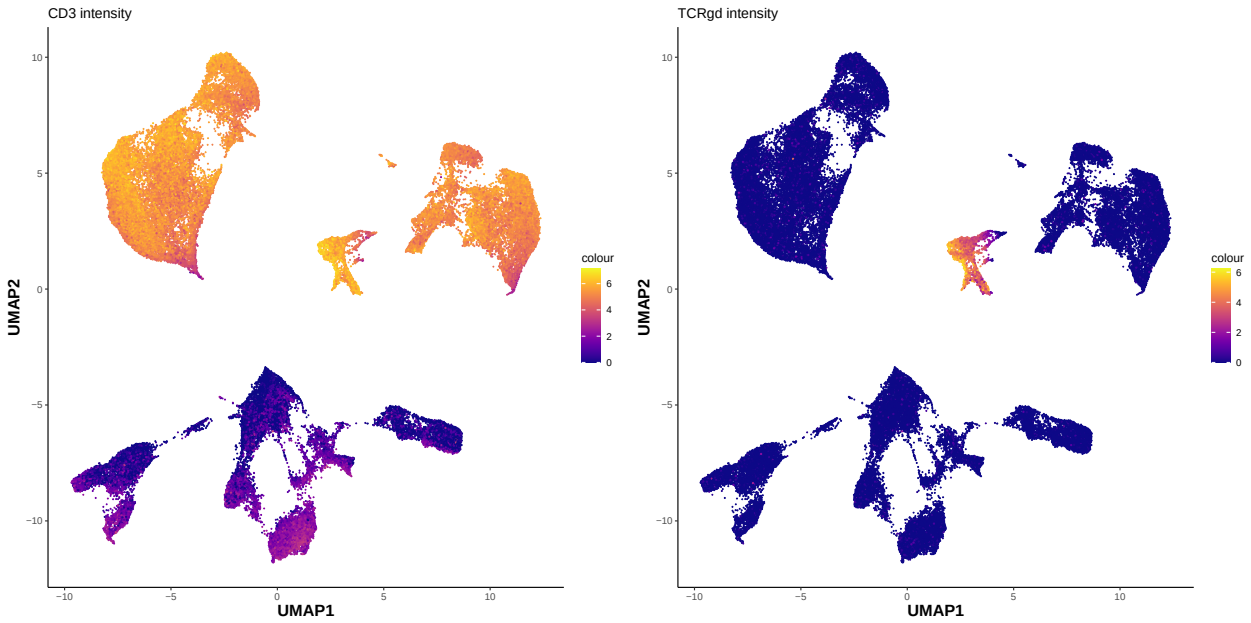


Figure 3: marker intensity UMAP; CD3 and TCRgd

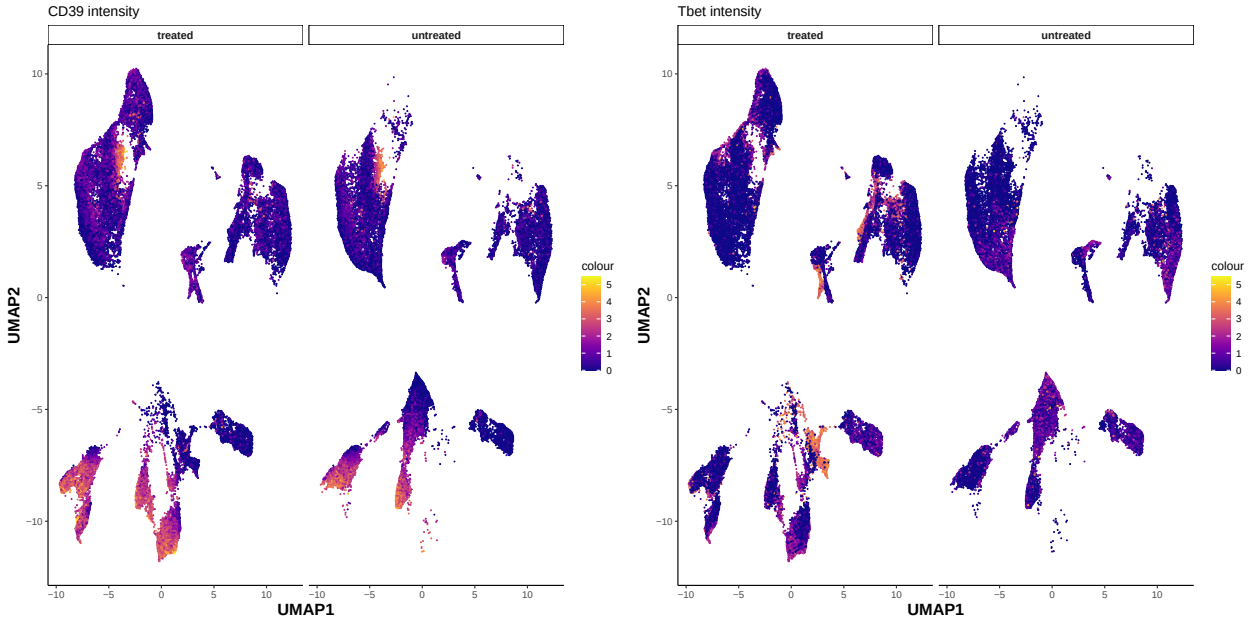


Figure 4: marker intensity UMAP per group; CD39 and Tbet

Another way of showing the overall and per-group marker expression intensity that the pipeline uses is violin plots.

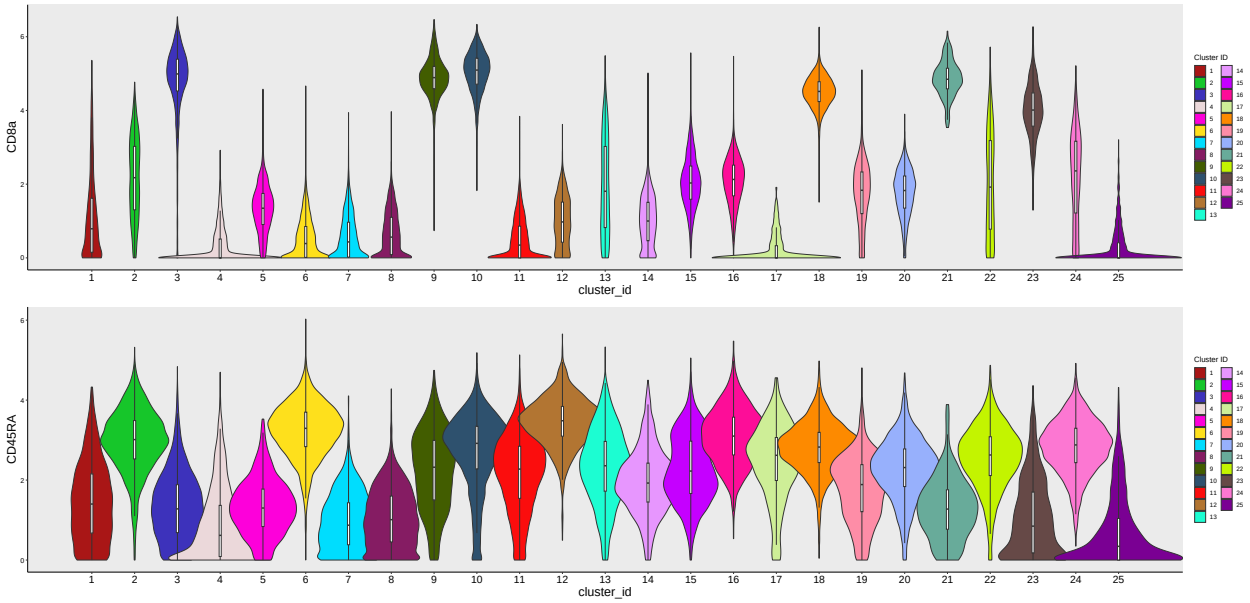


Figure 5: violin plots - overall expression; CD8 and CD45RA

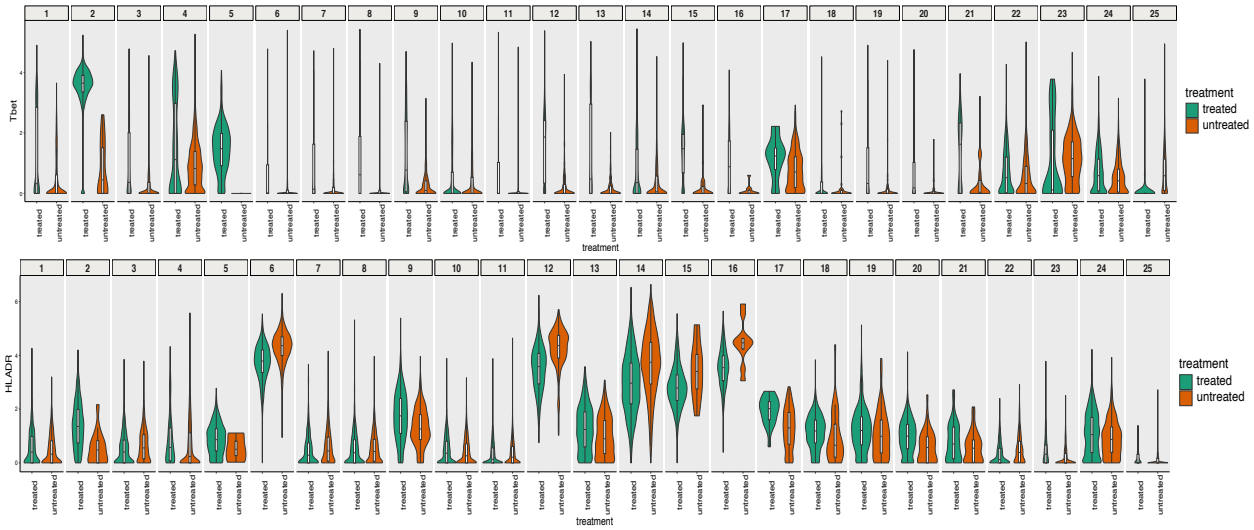


Figure 6: violin plots - per-group expression; Tbet and HLA-DR

The overall markers expression intensity can also be visualized by heatmap

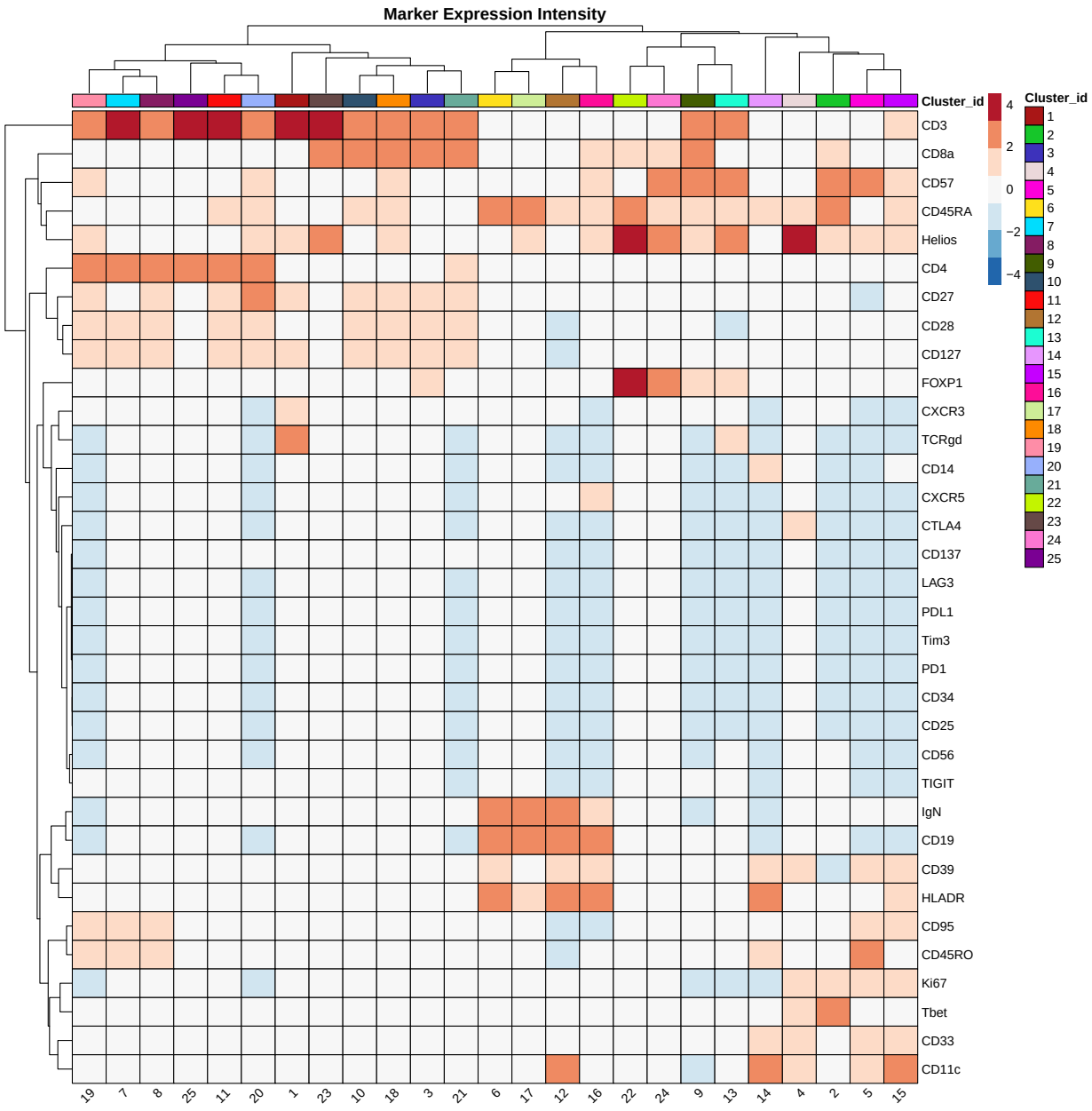


Figure 7: Heatmap

Miscellaneous visualizations

The pipeline generates some visualization for cluster frequencies in the dataset for individual samples and for the specified groups. Also, the tables containing this data is exported in the output location for further analysis.

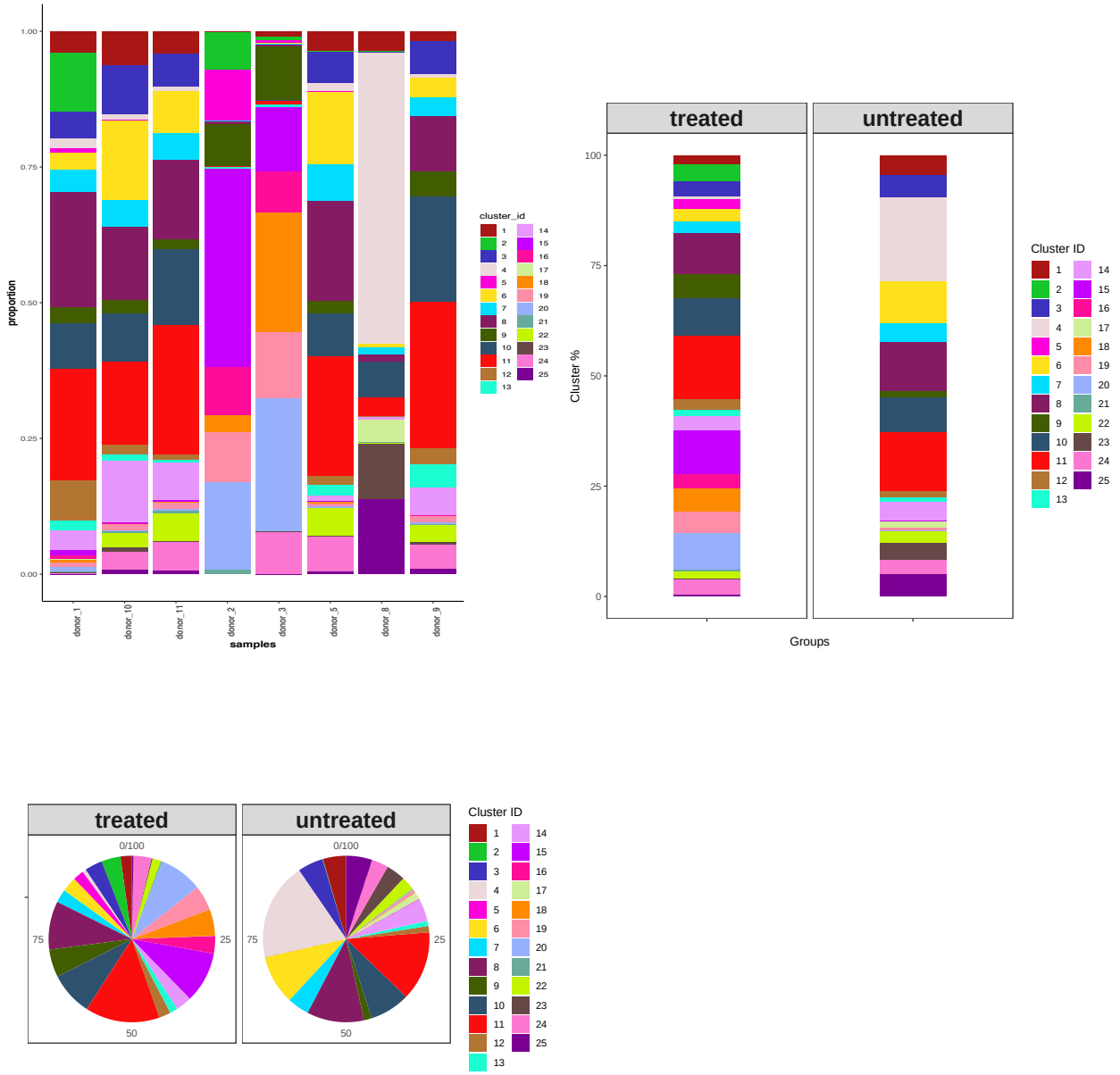


Figure 8: Cluster proportions per sample and per group

Statistical analysis

The workflow includes testing the correlation of cluster abundances in all samples and samples within groups.

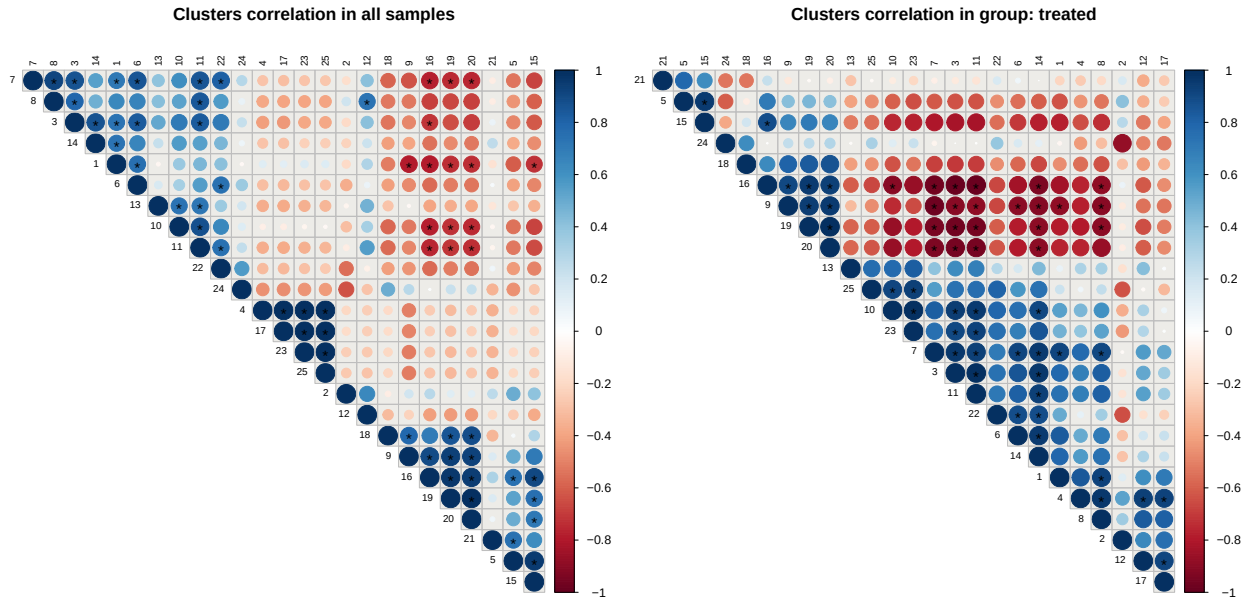


Figure 9: correlation

Clusters that significantly correlate in abundance together in all samples or groups are labelled with asterisk. Note that correlation coefficient of groups that contains 4 or less samples cannot be calculated. The tables for the Pearson's correlation coefficients and p values are exported in folder Tables.

Differential Abundace

The pipeline uses edgeR test from diffcyt to test the differential cluster abundance between the specified groups. Check out diffcyt package for more details. A table with log-fold changes and un-adjusted and adjusted p values (Benjamini-Hochberg method) is exported to the output location in folder stats subfolder Cluster abundance. In addition, column plot is generated and exported for the log-fold change values. If argument paired is set to TRUE, the model formula is adjusted to take in consideration the paired design (with exp_subject argument specifying samples pairing). The comparison contrast is specified in argument contrast. Clusters significant in abundance (below 0.05 adjusted p value) are marked with asterisk. For more information, check out the package documentation.

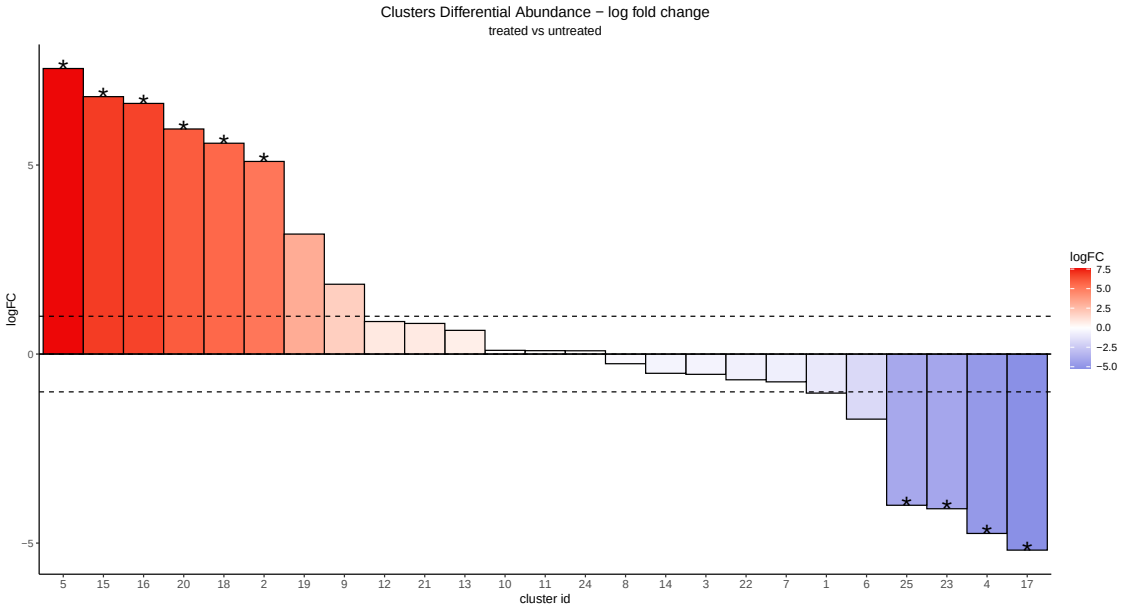


Figure 10: Differential Abundance

Differential expression

The pipeline uses the two multiple regression approaches provided by CytoGLMM package. The bootstrapped generalized linear model for unpaired experimental design and the generalized linear mixed model with random effect assigned to exp_subject to adjust for the autocorrelation of a paired design. The function cyto_diff_exprs() performs either one of these two model -based on the design- by reiterating over all the clusters in the data set. The information from this modelling allows to identify the combination of markers on specific clusters that can predict/classify the response variable e.g treatment, response..etc. By default, the markers included in the modelling are the markers that are labelled as "state" in marker_info table. If a different set of markers is of interest, simply the labels in marker_info or SummarizedExperiment can be changed to include or exclude markers. Forest plots are generated for each cluster in the data set. Also, a table with coefficient median, confidence intervals, p values is exported to the output location folder "Differential expression".

Figure 11: Differential Abundance

Also, for differential expression testing, limma test from diffcyt package can be performed if diff_limma argument is set to TRUE. Experimental desgin can be paired or unpaired. A table with log-fold change and p values for all the marker cluser combination.

For information about the individual functions in CytoAnalyze package and troubleshooting, please check out the package documentations.