Introduction

Overview

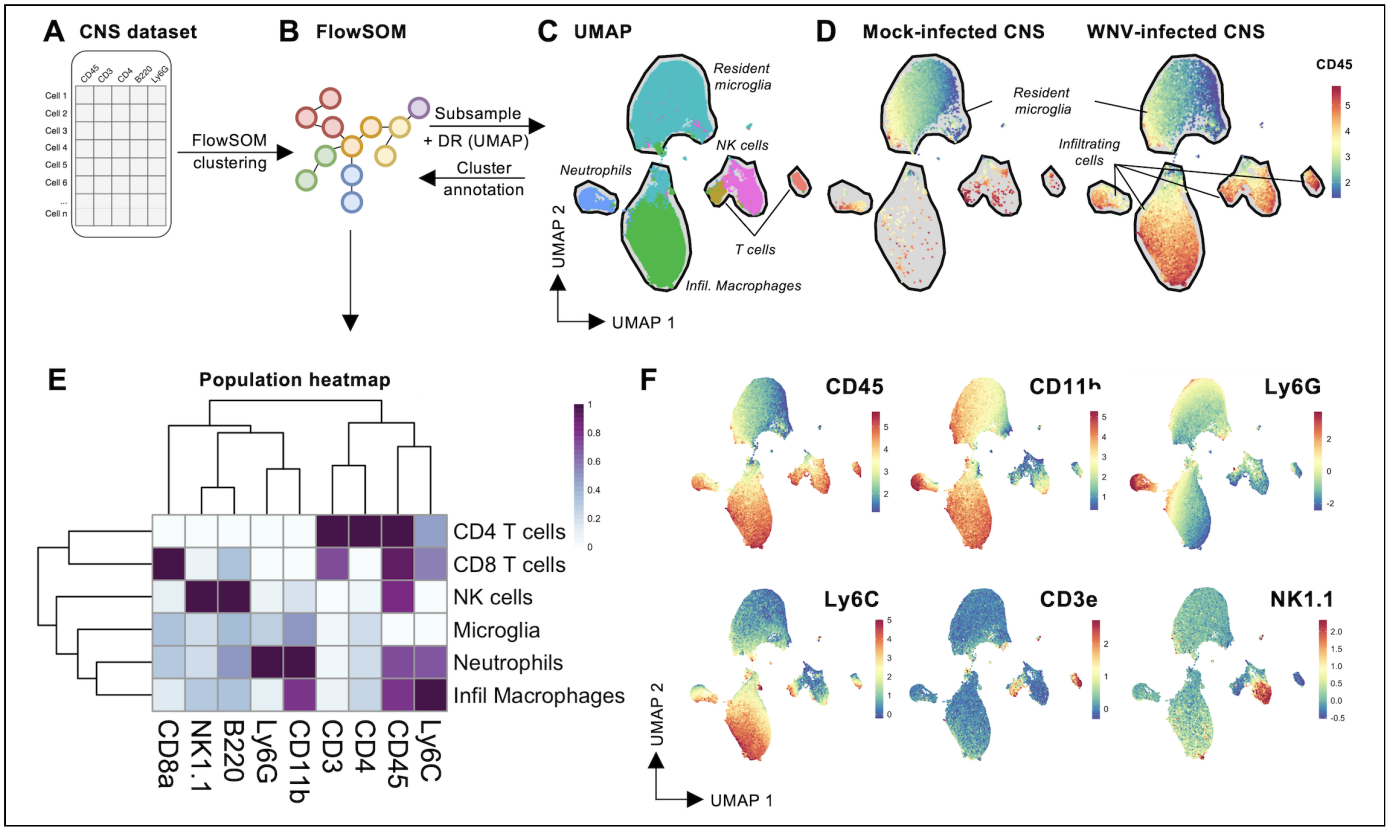
Here we provide a worked example of a 'simple' discovery analysis workflow, where the entire process (data prep, clustering, dimensionality reduction, cluster annotation, plotting, summary data, and statistical analysis) is contained within a single script. The 'simple' workflow is most suitable for fast analysis of small datasets. For larger or more complex datasets, or datasets with multiple batches, we recommend the general discovery workflow, where the data preparation, batch alignment, clustering/dimensionality reduction, and quantitative analysis are separated into separate scripts. The demo dataset used for this worked example are cells extracted from mock- or virally-infected mouse brains, measured by flow cytometry.

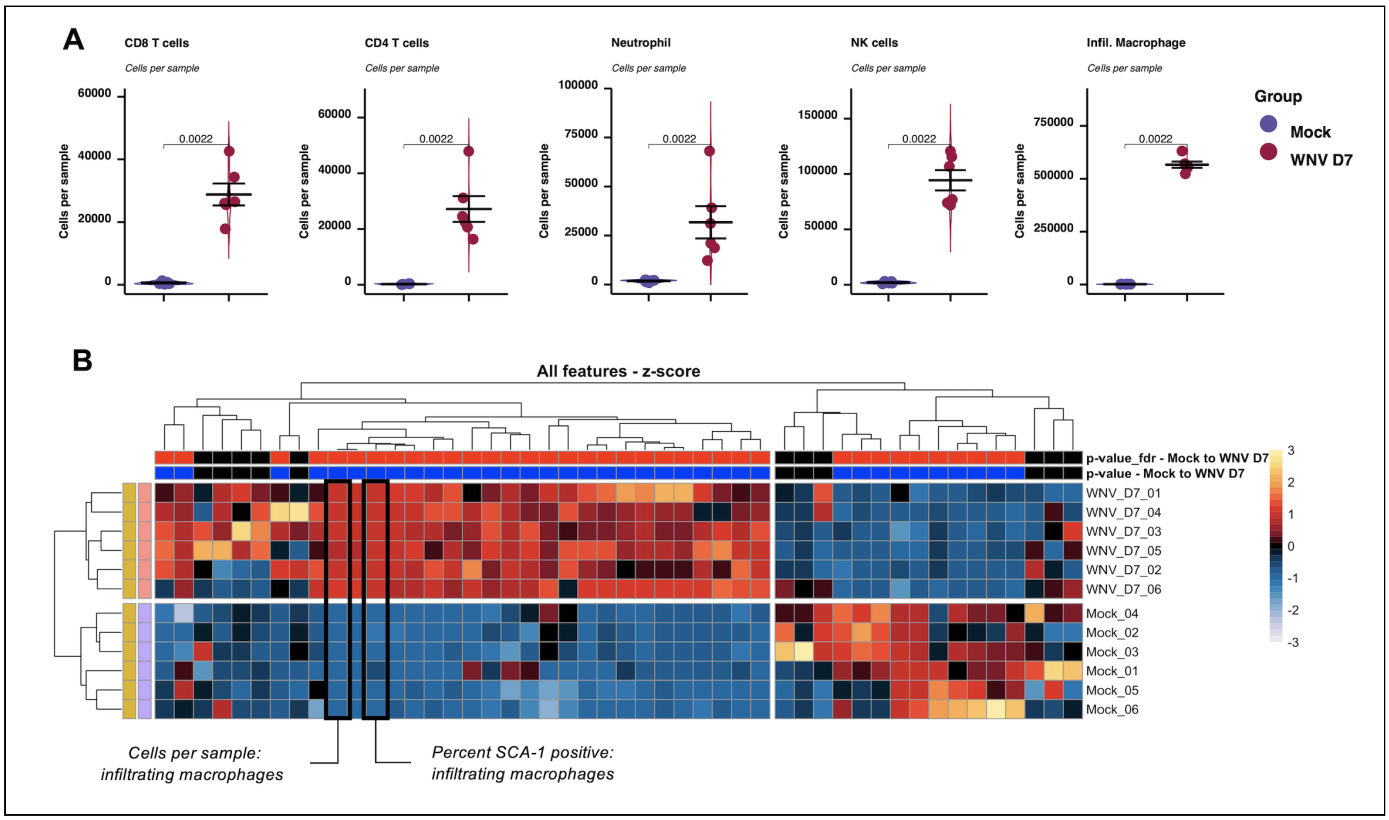
Strategy

The 'simple' and 'general' discovery workflows are designed to facilitate the analysis of large and complex cytometry datasets using the Spectre R package. We've tested up to 30 million cells in a single analysis session so far. The workflow is designed to get around the cell number limitations of tSNE/UMAP. The analysis starts with clustering with FlowSOM – which is fast and scales well to large datasets. The clustered data is then downsampled, and dimensionality reduction is performed with tSNE/UMAP. This allows for visualisation of the data, and the clusters present in the dataset. Once the possible cell types in the datasets have been explored, the clusters can be labelled with the appropriate cellular identities. Finally, we can use the clusters/populations to generate summary statistics (expression levels, frequencies, total counts etc), which allows us to create graphs and heatmaps, facilitating statistical analysis.

Batch alignment

The 'simple' discovery workflow does not include any batch alignment steps. If batch correction needs to be applied, we recommend using the general discovery workflow.


A CNS dataset
B FlowSOM
C UMAP
D Mock-infected CNS
WNV-infected CNS
FlowSOM clustering
Subsample + DR (UMAP)
Cluster annotation
Resident microglia
NK cells
Neutrophils
T cells
Infil. Macrophages
Infiltrating cells
CD45
UMAP 1
UMAP 2
E Population heatmap
CD4 T cells
CD8 T cells
NK cells
Microglia
Neutrophils
Infil Macrophages
CD8a
NK1.1
B220
Ly6G
CD11b
CD3
CD4
CD45
Ly6C
F
CD45
CD11b
Ly6G
Ly6C
CD3e
NK1.1


A
CD8 T cells
CD4 T cells
Neutrophil
NK cells
Infil. Macrophage
Cells per sample
0.0022
Group
Mock
WNV D7
B
All features - z-score
p-value_fdr - Mock to WNV D7
p-value - Mock to WNV D7
WNV_D7_01
WNV_D7_04
WNV_D7_03
WNV_D7_05
WNV_D7_02
WNV_D7_06
Mock_04
Mock_02
Mock_03
Mock_01
Mock_05
Mock_06
Cells per sample: infiltrating macrophages
Percent SCA-1 positive: infiltrating macrophages

Citation

If you use Spectre in your work, please consider citing Ashhurst TM, Marsh-Wakefield F, Putri GH et al. (2020). bioRxiv. 2020.10.22.349563. To continue providing open-source tools such as Spectre, it helps us if we can demonstrate that our efforts are contributing to analysis efforts in the community. Please also consider citing the authors of the individual packages or tools (e.g. CytoNorm, FlowSOM, tSNE, UMAP, etc) that are critical elements of your analysis work. We have provided some generic text that you can use for your methods section with each protocol and on the 'about' page.

Sample methods blurb

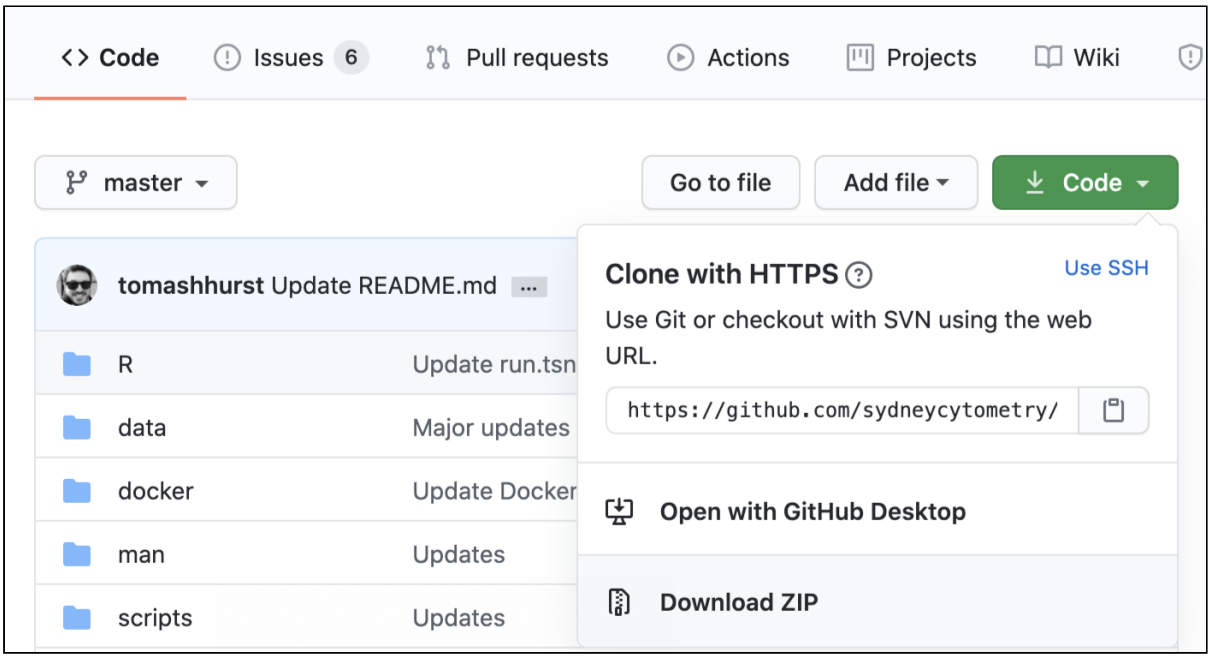
Here is a sample methods blurb for this workflow. You may need to adapt this text to reflect any changes made in your analysis.

Computational analysis of data was performed using the Spectre R package (Ashhurst et al., 2020), with instructions and source code provided at https://github.com/ImmuneDynamics/spectre. Samples were initially prepared in FlowJo, and the population of interest was exported as raw value CSV files. Arcsinh transformation was performed on the data in R using a co-factor of 15 to redistribute the data on a linear scale and compress low end values near zero. The dataset was then merged into a single data.table, with keywords denoting the sample, group, and other factors added to each row (cell). The FlowSOM algorithm (Van Gassen et al., 2015) was then run on the merged dataset to cluster the data, where every cell is assigned to a specific cluster and metacluster. Subsequently, the data was downsampled and analysed by the dimensionality reduction algorithm Uniform Manifold Approximation and Projection (UMAP) (McInnes, Healy, Melville, 2018) for cellular visualisation.

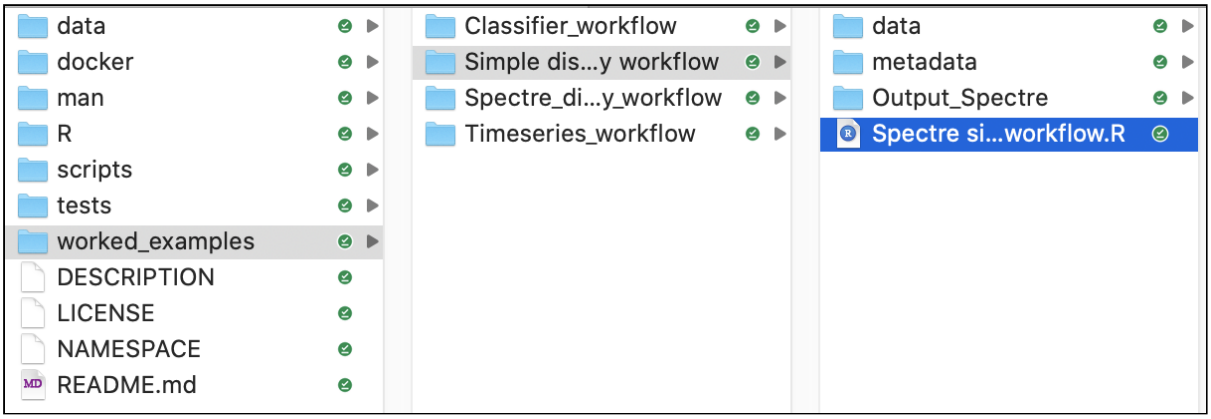
Software and R script preparation

Software: for instructions downloading R, RStudio, and Spectre, please see this section on the home page.

Analysis script: Please visit https://github.com/ImmuneDynamics/Spectre, and download the repository:



You can then find the 'simple discovery workflow' script here:

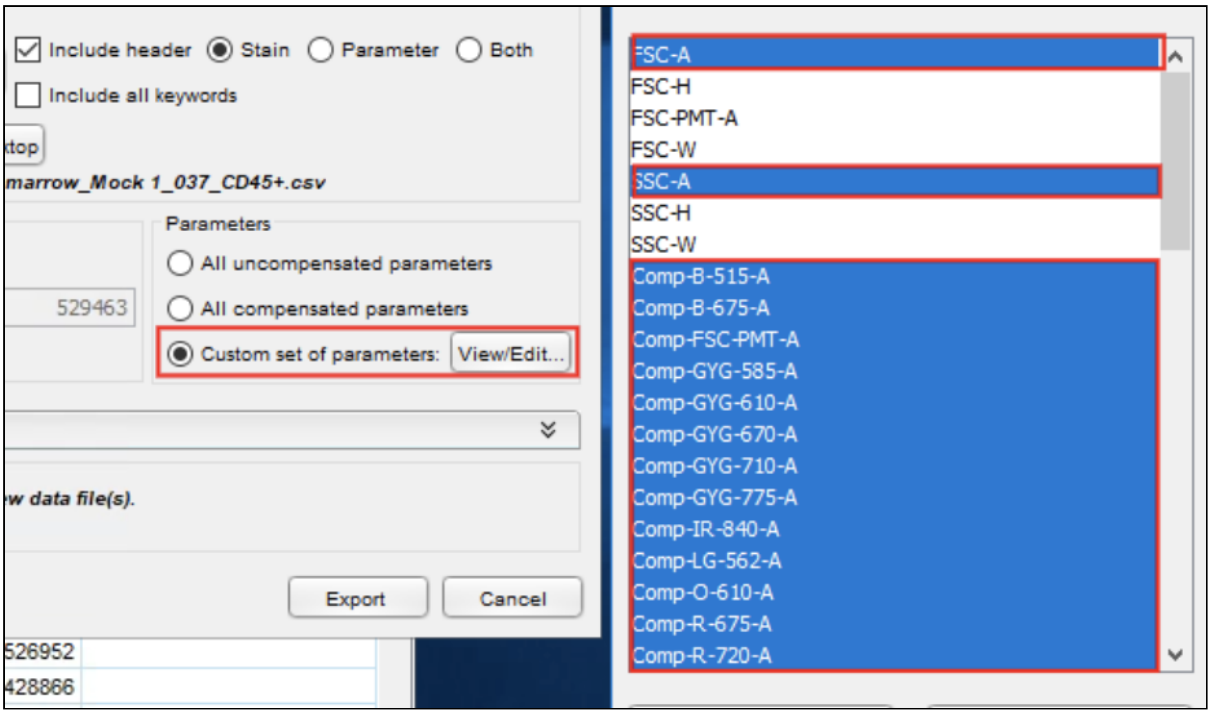


Create a folder for your experiment, and place the script in that folder.

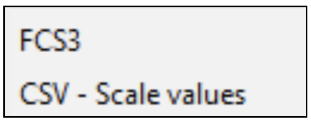
Data preparation and export from FlowJo (or similar)

Export the population of interest (POI) from your files.

Please see this page for more detailed instructions. When using fluorescence data, please ensure you are exporting the data from compensated channels, indicated by 'Comp-Channel Name' (e.g. Comp-B515). Feel free to include any other relevant parameters as well (FSC, SSC, time etc).



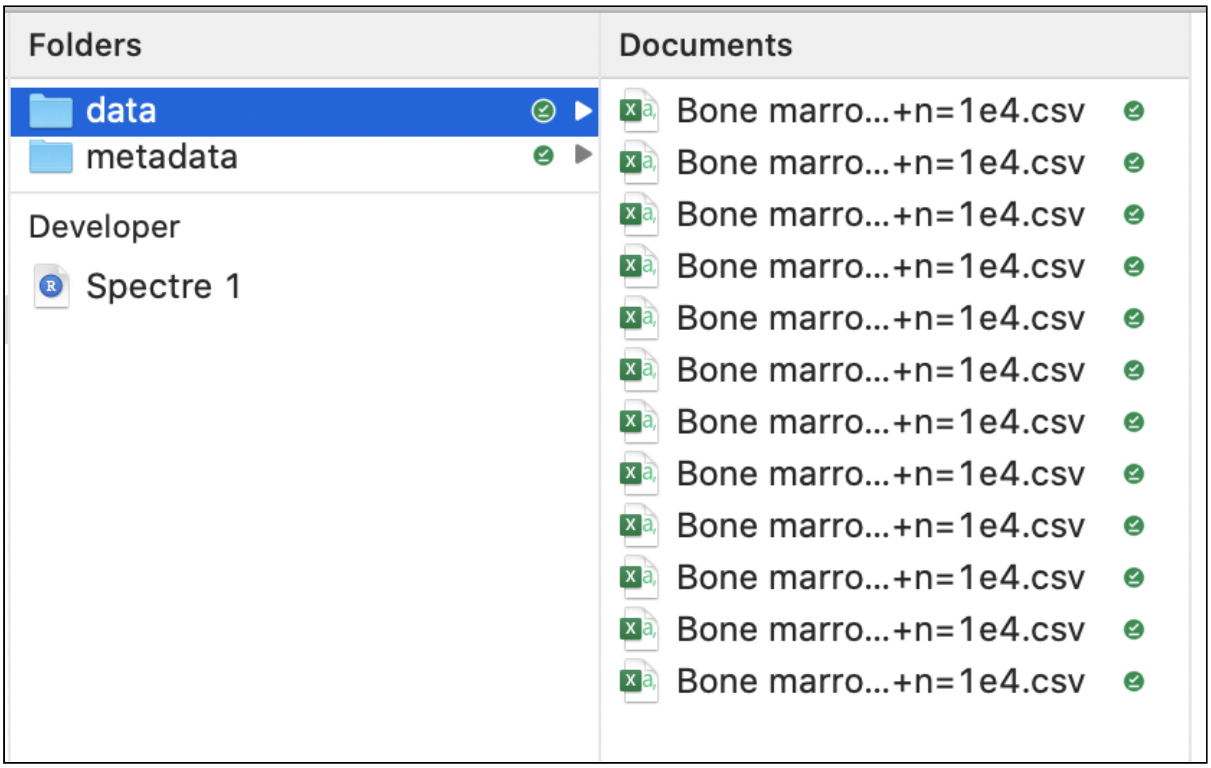
We recommend exporting **CSV 'scale' value** data, or alternatively exporting **FCS files**. These represent the untransformed data values.



You are also welcome to use the CSV 'channel' value data, which uses a form of 'binning' to transform the data onto a linear distribution. Please read this page for a more detailed explanation.

Data folder

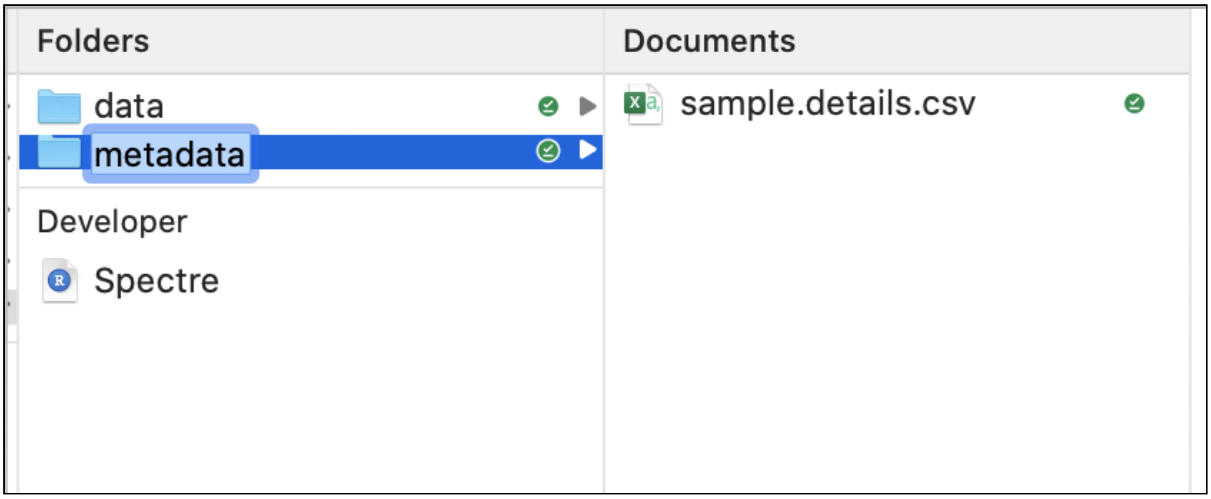
Create a folder within your experiment folder called 'data', and place the exported files there.

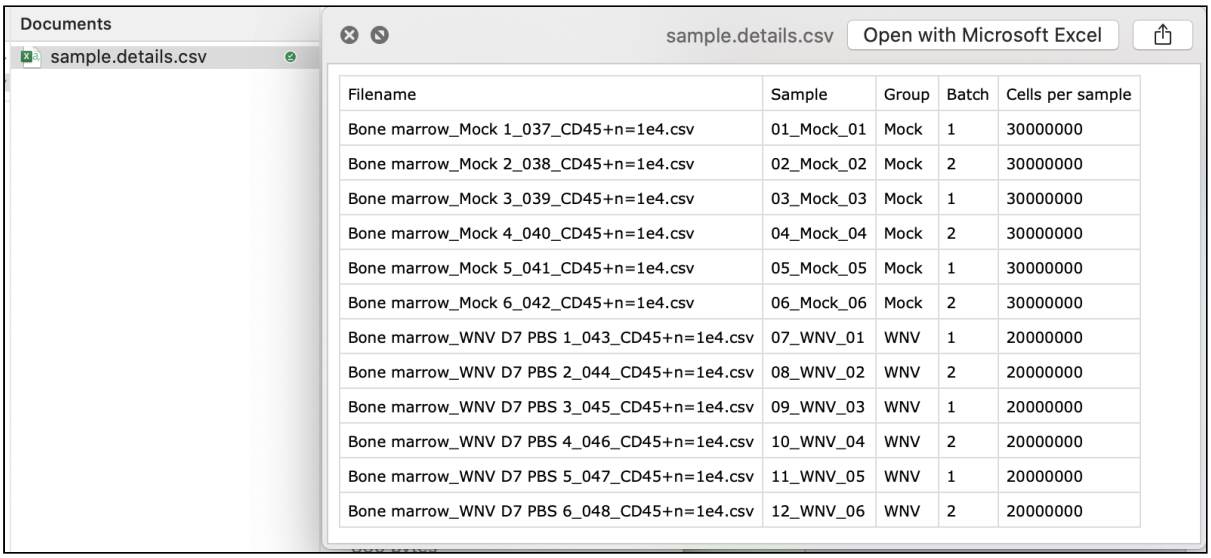


Setup some sample metadata and place it in a folder called 'metadata' or similar

Create a CSV file (using Microsoft Excel or similar) – we have called the file 'sample.details' here. The first column should be called 'Filename' or similar, and should contain the name of one file per row. On a Mac, you can copy the files and 'paste' them into excel – it will copy the name of the file, including extensions (".csv", ".fcs", etc). You can then add as many additional columns as you like, and these can be called whatever you like (e.g. "Sample" could be "SampleName", or "Sample_Name" etc).

- "Sample" is a recommended column, as this can be a more simplified name for each sample
- "Group" is extremely useful for most analyses
- "Batch" is helpful if you have prepared, stained, or run samples in multiple batches. If only a single batch is used, we still recommend entering a 'Batch' column with all rows containing '1'.
- "Cells per sample" is a useful column to add if you intend to generate absolute counts of each population per sample during the generation of summary data, but is not required otherwise.





Filename	Sample	Group	Batch	Cells per sample
Bone marrow_Mock 1_037_CD45+n=1e4.csv	01_Mock_01	Mock	1	30000000
Bone marrow_Mock 2_038_CD45+n=1e4.csv	02_Mock_02	Mock	2	30000000
Bone marrow_Mock 3_039_CD45+n=1e4.csv	03_Mock_03	Mock	1	30000000
Bone marrow_Mock 4_040_CD45+n=1e4.csv	04_Mock_04	Mock	2	30000000
Bone marrow_Mock 5_041_CD45+n=1e4.csv	05_Mock_05	Mock	1	30000000
Bone marrow_Mock 6_042_CD45+n=1e4.csv	06_Mock_06	Mock	2	30000000
Bone marrow_WNV D7 PBS 1_043_CD45+n=1e4.csv	07_WNV_01	WNV	1	20000000
Bone marrow_WNV D7 PBS 2_044_CD45+n=1e4.csv	08_WNV_02	WNV	2	20000000
Bone marrow_WNV D7 PBS 3_045_CD45+n=1e4.csv	09_WNV_03	WNV	1	20000000
Bone marrow_WNV D7 PBS 4_046_CD45+n=1e4.csv	10_WNV_04	WNV	2	20000000
Bone marrow_WNV D7 PBS 5_047_CD45+n=1e4.csv	11_WNV_05	WNV	1	20000000
Bone marrow_WNV D7 PBS 6_048_CD45+n=1e4.csv	12_WNV_06	WNV	2	20000000

1. Load packages and set directories

Open the analysis script in RStudio and open the simple discovery workflow script.

Note

- If you haven't installed Spectre, please visit our Spectre installation page.
- If you aren't familiar with using RStudio or Spectre, please see our RStudio and Spectre basic tutorials.

```
#########################################################################################################
###
#### 1. Load packages, and set working directory
#########################################################################################################
###
```

Load the Spectre and other required libraries

Running library(Spectre) will load the Spectre package. We can then use package.check() to see if the standard dependency packages are installed, and package.load() to load those packages.

```
    ### Load libraries

        library(Spectre)
        Spectre::package.check()    # Check that all required packages are installed
        Spectre::package.load()     # Load required packages
```

Set 'PrimaryDirectory'

Initially, we will set the location of the script as 'PrimaryDirectory'. We'll use this as a sort of 'home page' for where our analysis is going to be performed – including where to find our input data, metadata, and where our output data will go.

```
    ### Set PrimaryDirectory
        dirname(rstudioapi::getActiveDocumentContext()$path)            # Finds the directory where
this script is located
        setwd(dirname(rstudioapi::getActiveDocumentContext()$path))     # Sets the working directory to
where the script is located
        getwd()
        PrimaryDirectory <- getwd()
```

Set 'InputDirectory'

Next we need to set the location of the 'data' folder – where our samples for analysis are stored. In this example they are stored in a sub-folder called 'data'.

> **Setting directories**
>
> If you aren't sure how to navigate directories in R, check out our brief introduction to R tutorial.

```
    ### Set 'input' directory
        setwd(PrimaryDirectory)
        setwd("data/")
        InputDirectory <- getwd()
        setwd(PrimaryDirectory)
```

Set 'MetaDirectory'

We need to set the location of the 'metadata' folder. This is where we can store a CSV file that contains any relevant metadata that we want to embed in our samples. In this example, it is located in a sub-folder called 'metadata'.

```
    ### Set 'metadata' directory
        setwd(PrimaryDirectory)
        setwd("metadata/")
        MetaDirectory <- getwd()
        setwd(PrimaryDirectory)
```

Create 'OutputDirectory'

We need to create a folder where our output data can go once our analysis is finished. In this example we will call this 'Output_Spectre'.

```
### Create output directory
    dir.create("Output_Spectre", showWarnings = FALSE)
    setwd("Output_Spectre")
    OutputDirectory <- getwd()
    setwd(PrimaryDirectory)
```

2. Read and prepare data

```
#########################################################################################################
###
#### 2. Import and prep data
#########################################################################################################
###
```

Read in data

To begin, we will change our working directory to 'InputDirectory' and list all the CSV files in that directory – these should be the sample CSV files.

```
### Import data

    setwd(InputDirectory)
    list.files(InputDirectory, ".csv")
```

We can then read in all of our samples (in this example, one CSV file per sample) into a list called 'data.list'. Spectre uses the data.table framework to store data, which reads, writes, and performs operations on data very quickly.

```
    data.list <- Spectre::read.files(file.loc = InputDirectory,
                                     file.type = ".csv",
                                     do.embed.file.names = TRUE)
```

By default, the read.files() function will generate some other variables, which you can review, by running the do.list.summary() function.

The 'name.table' variable is a table of all the column names for all of your samples (one row per sample, one column per column name). If all of the column names are matching, then this table should be a repeating pattern. If it has been jumbled, then some of your samples have columns that don't appear in other samples. The 'ncol.check' and 'nrow.check' are simple tables indicating the number or columns and rows in each sample.

```
### Check the data

    check <- do.list.summary(data.list)

    check$name.table # Review column names and their subsequent values
    check$ncol.check # Review number of columns (features, markers) in each sample
    check$nrow.check # Review number of rows (cells) in each sample
```

You can review the first 6 rows of the first sample in your data using the following:

```
    data.list[[1]]
```

Merge data.tables

Once the metadata has been added, we can then merge the data into a single data.table using do.merge.files(). By default, columns with matching names will be aligned in the new table, and any columns that are present in some samples, but not others, will be added and filled with 'NA' for any samples that didn't have that column initially.

```
    ### Merge data

        cell.dat <- Spectre::do.merge.files(dat = data.list)
```

Once the data has been merged, we can review the data:

```
cell.dat
```

```
             NK11        CD3      CD45       Ly6G    CD11b       B220       CD8a        Ly6C        CD4
FileName FileNo
     1:   42.3719  40.098700  6885.08  -344.7830 14787.30  -40.2399   83.7175   958.7000  711.0720
CNS_Mock_01      1
     2:   42.9586 119.014000  1780.29  -429.6650  5665.73   86.6673   34.7219   448.2590  307.2720
CNS_Mock_01      1
     3:   59.2366 206.238000 10248.30 -1603.8400 19894.30  427.8310  285.8800  1008.8300  707.0940
CNS_Mock_01      1
     4:  364.9480  -0.233878  3740.04  -815.9800  9509.43  182.4200  333.6050   440.0710  249.7840
CNS_Mock_01      1
     5:  440.2470  40.035200  9191.38    40.5055  5745.82 -211.6940  149.2200    87.4815  867.5700
CNS_Mock_01      1
    ---
169000:  910.8890  72.856100 31466.20  -316.5570 28467.80   -7.7972 -271.8040 12023.7000 1103.0500
 CNS_WNV_D7_06     12
169001:  -10.2642  64.188700 45188.00  -540.5140 22734.00  202.4110 -936.4920  4188.3300  315.9400
 CNS_WNV_D7_06     12
169002: -184.2910  -9.445650 11842.60   -97.9383 17237.00  123.4760 -219.9320  8923.4000 -453.4640
 CNS_WNV_D7_06     12
169003:  248.3860 229.986000 32288.20  -681.1630 19255.80 -656.0540 -201.5880 10365.7000   61.6765
 CNS_WNV_D7_06     12
169004:  738.9810  95.470300 46185.10 -1004.6000 22957.80 -661.6280   72.3356  9704.4700  -31.8532
 CNS_WNV_D7_06     12
```

Read in sample metadata

```
    ### Read in metadata

        setwd(MetaDirectory)

        meta.dat <- fread("sample.details.csv")
        meta.dat
```

3. Arcsinh transformation

Before we perform clustering etc, we need to meaningfully transform the data. For more information on why this is necessary, please see this page.

Data transformations

If you have imported **CSV (channel value)** files exported from FlowJo, then no data transformations are required, and you can skip all of the arcsinh transformation steps and proceed straight to adding the metadata. More information on the FCS, CSV scale, and CSV channel value file types can be found here.

First, check the column names of the dataset.

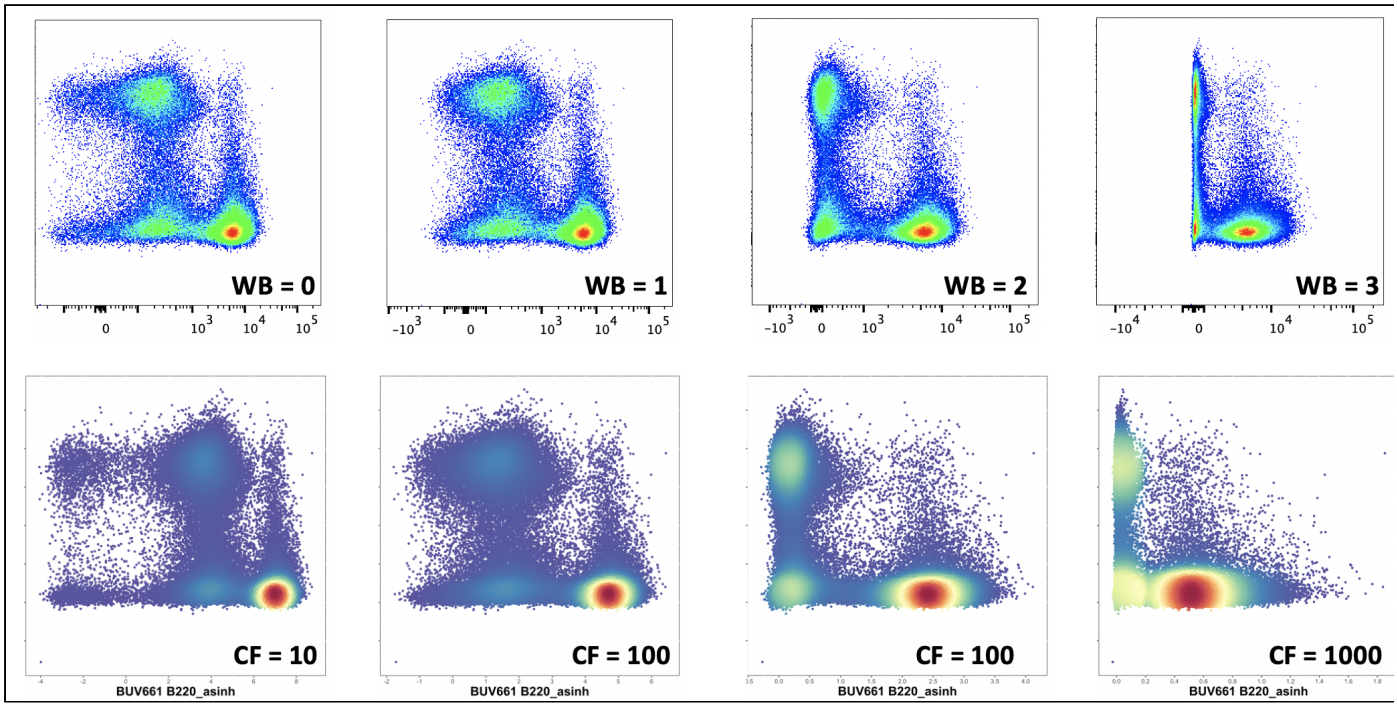
```
#########################################################################################################
###
#### 3. Data transformation
#########################################################################################################
###

    setwd(OutputDirectory)
    dir.create("Output 1 - transformed plots")
    setwd("Output 1 - transformed plots")

    ### Arcsinh transformation

        as.matrix(names(cell.dat))
```

```
      [,1]
 [1,] "NK11"
 [2,] "CD3"
 [3,] "CD45"
 [4,] "Ly6G"
 [5,] "CD11b"
 [6,] "B220"
 [7,] "CD8a"
 [8,] "Ly6C"
 [9,] "CD4"
[10,] "FileName"
[11,] "FileNo"
```

The columns we want to apply arcsinh transformation to are the cellular columns – column 1 to column 9. We can specify those columns using the code below.

```
        to.asinh <- names(cell.dat)[c(1:9)]
        to.asinh
```

Define the cofactor we will use for transformation. As a general recommendation, we suggest using **cofactor = 15 for CyTOF data**, and **cofactor between 100 and 1000 for flow data** (we suggest 500 as a starting point). Here is a quick comparison figure showing how different co-factors compare to bi-exponential transformations performed on an LSR-II. For more detailed information on this choice, and for approaches where different cofactors for different columns might be required, see this page.



In this worked example we will use a cofactor of 500.

```
cofactor <- 500
```

You can also choose a column to use for plotting the transformed result – ideally something that is expressed on a variety of cell types in your dataset.

```
plot.against <- "Ly6C_asinh"
```

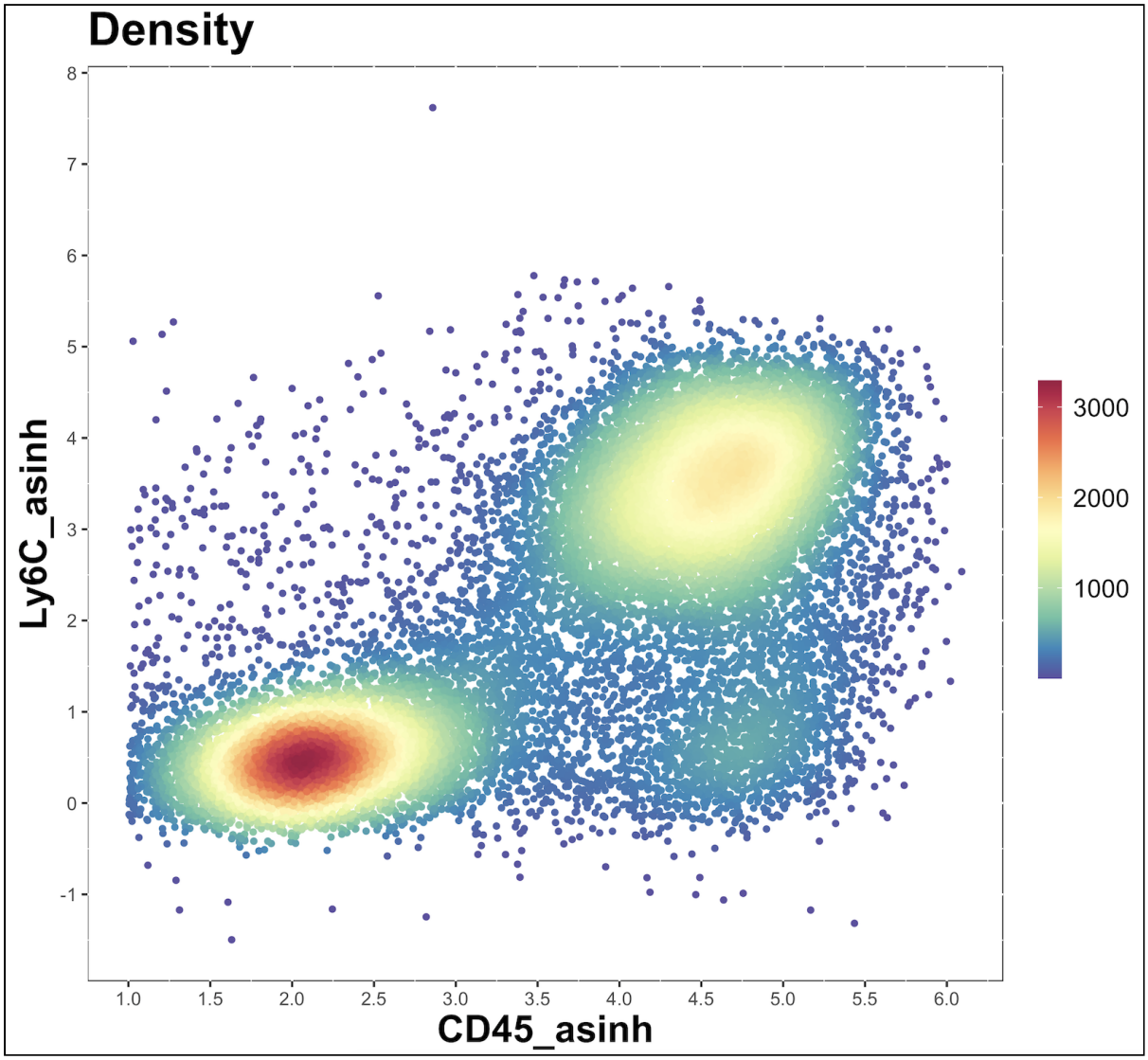
Now we need to apply arcsinh transformation to the data in those columns, using a specific co-factor.

```
cell.dat <- do.asinh(cell.dat, to.asinh, cofactor = cofactor)
transformed.cols <- paste0(to.asinh, "_asinh")
```

We can then make some plots to see if the arcsinh transformation is appropriate

```
for(i in transformed.cols){
  make.colour.plot(do.subsample(cell.dat, 20000), i, plot.against)
}
```

Check the plots and see if you are happy with the transformation. For more detailed guidance, see this page.



If happy, the proceed with analysis. Otherwise, go back to the merging of the data.list (to create cell.dat) and try with another co-factor.

4. Add sample metadata and set preferences

```
################################################################################################################
###
#### 4. Add metadata and set some preferences
################################################################################################################
###
```

We also want to read in and attach some sample metadata, to aid with our analysis. We can set our working directory to MetaDirectory and read in the CSV.

```
    setwd(MetaDirectory)

    ### Metadata

        meta.dat <- fread("sample.details.csv")
        meta.dat
```

Once we have the metadata read into R, we will select only the columns we want to add to our dataset. In this example we only want to include use first four columns (Filename, Sample, Group, and Batch). 'Filename' will be used to for matching between cell.dat and meta.dat, and the other three columns will be the information that gets added to cell.dat

```
        sample.info <- meta.dat[,c(1:4)]
        sample.info

        counts <- meta.dat[,c(2,5)]
        counts
```

Now we can add this information to cell.dat. Essentially, the file names are listed in the metadata table, and we can use that to add any listed metadata in the table to the corresponding files in data.list.

```
        cell.dat <- do.add.cols(cell.dat, "FileName", sample.info, "Filename", rmv.ext = TRUE)
```

We can review the data to ensure the metadata has been correctly embedded.

```
cell.dat
```

Check the column names.

```
    ### Define cellular columns
        as.matrix(names(cell.dat))
```

```
      [,1]
 [1,] "NK11"
 [2,] "CD3"
 [3,] "CD45"
 [4,] "Ly6G"
 [5,] "CD11b"
 [6,] "B220"
 [7,] "CD8a"
 [8,] "Ly6C"
 [9,] "CD4"
[10,] "FileName"
[11,] "FileNo"
[12,] "NK11_asinh"
[13,] "CD3_asinh"
[14,] "CD45_asinh"
[15,] "Ly6G_asinh"
[16,] "CD11b_asinh"
[17,] "B220_asinh"
[18,] "CD8a_asinh"
[19,] "Ly6C_asinh"
[20,] "CD4_asinh"
[21,] "Sample"
[22,] "Group"
[23,] "Batch"
```

Specify columns that represent cellular features (in this case, the arcsinh transformed data, defined by "markername_asinh").

```
        cellular.cols <- names(cell.dat)[c(12:20)]
        as.matrix(cellular.cols)
```

Additionally, specify the columns that will be used to generate cluster and tSNE/UMAP results. Columns that are not specified here will still be analysed, but won't contributed to the generation of clusters. There are a couple of strategies to take here: use all cellular columns for clustering to looks for all possible cell types/states, or use only stably expressed markers to cluster stable phenotypes, which can then be examined for changes in more dynamic markers. For more guidance, see this page.

```
        cluster.cols <- names(cell.dat)[c(12:20)]
        as.matrix(cluster.cols)
```

Specify sample, group, and batch columns

```
        exp.name <- "CNS experiment"
        sample.col <- "Sample"
        group.col <- "Group"
        batch.col <- "Batch"
```

Additionally, we want to specify the downsample targets for dimensionality reduction. This influences how many cells will be *shown* on a tSNE/UMAP plot, and we are specifying the number of cells *per group* to downsample to. Check for the number of cells (rows) in each group:

```
    ### Subsample targets per group

        data.frame(table(cell.dat[[group.col]])) # Check number of cells per sample.
```

```
  Var1   Freq
1 Mock  66992
2  WNV 102012
```

You can then specify the number to downsample to in each group. These must be lower than the total number of cells in each group, and must be provided **in the order the groups appear above**). In this example we want 2000 cells from 'mock' and 20,000 cells from 'WNV', to reflect the number of cells present in each group.

```
        sub.targets <- c(2000, 20000) # target subsample numbers from each group
        sub.targets
```

5. Clustering and dimensionality reduction

```
#############################################################################################################
###
#### 5. Clustering and dimensionality reduction
#############################################################################################################
###
```

We can run clustering using the **run.flowsom** function. In this case we can define the number of desired metaclusters manually, with the **meta.k** argument (in this case we have chosen 8). This can be increased or decreased as required. Typically, overclustering is preferred, as multiple clusters that represent a single cellular population can always be annotated as such. Subsequently, we can write the clustered dataset to disk.

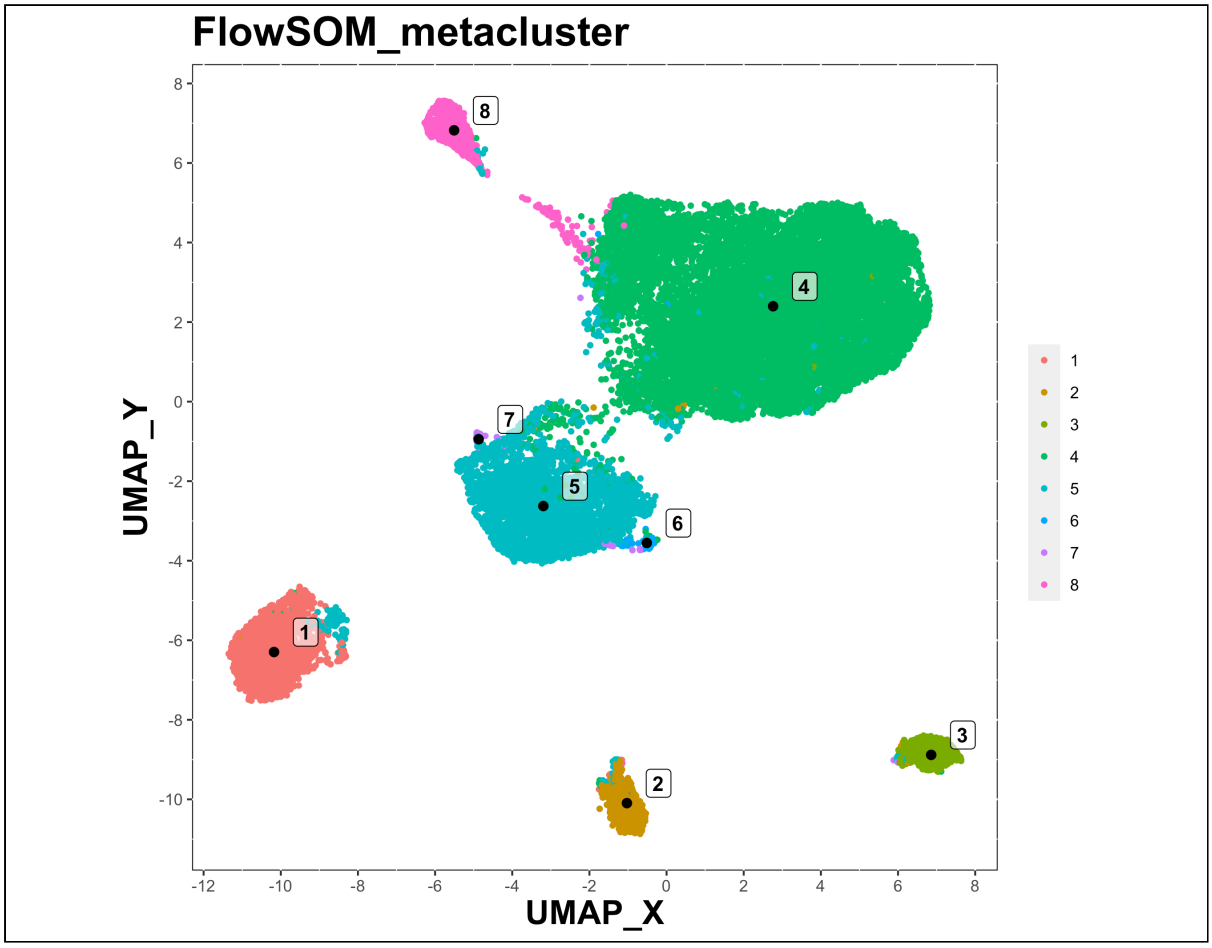
```
    setwd(OutputDirectory)
    dir.create("Output - clustering")
    setwd("Output - clustering")

    ### Clustering

        cell.dat <- run.flowsom(cell.dat, cluster.cols, meta.k = 8)
        fwrite(cell.dat, "clustered.data.csv")
```

We can then run dimensionality reduction on a subset of the data, allow us to visualise the data and resulting clusters. In this case we have used **run.umap**, though other options are available, including **run.fitsne** and **run.tsne**. As before, this subsampled dataset with DR coordinates is saved to disk.

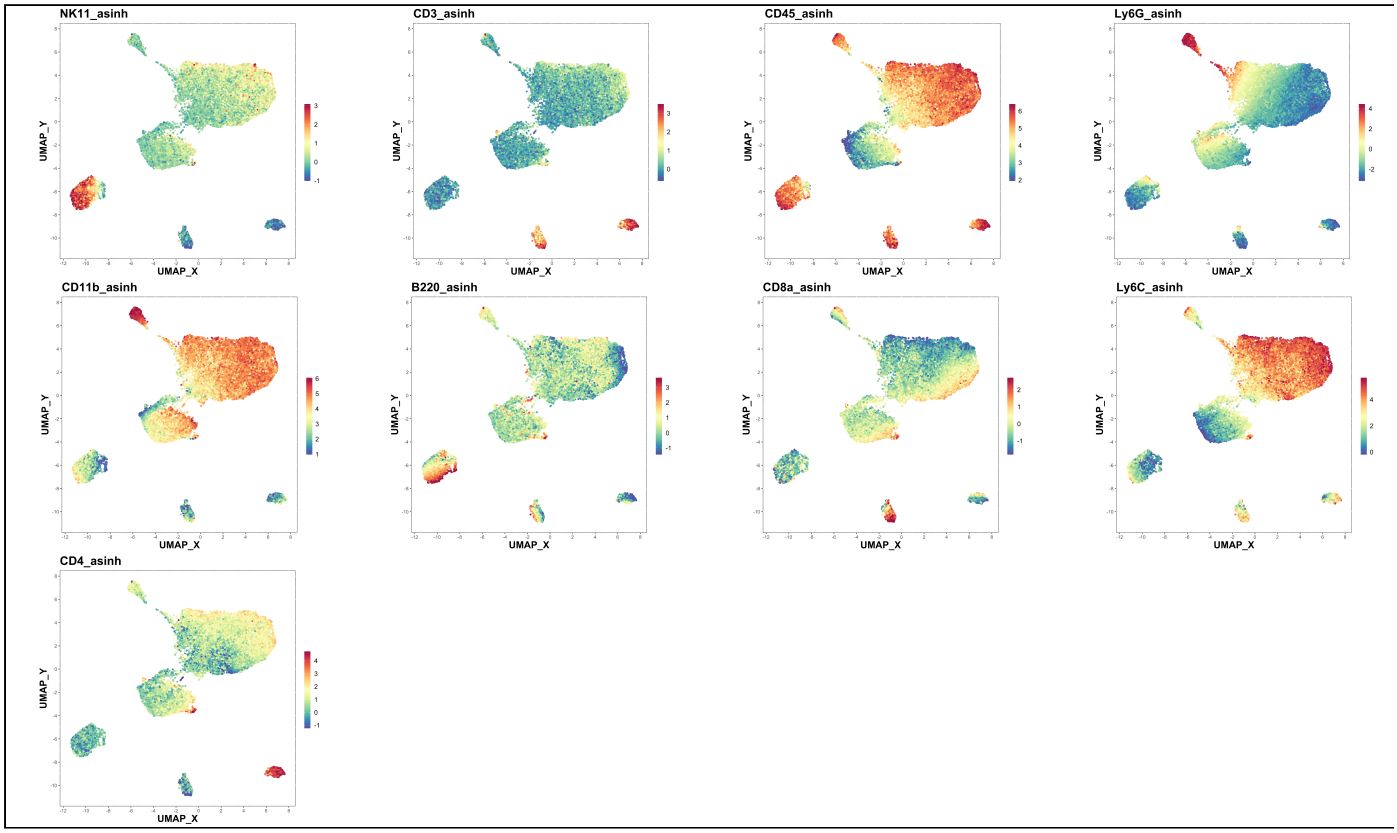
```
    ### Dimensionality reduction

        cell.sub <- do.subsample(cell.dat, sub.targets, group.col)
        cell.sub <- run.umap(cell.sub, cluster.cols)

        fwrite(cell.sub, "clustered.data.DR.csv")
```

We can visualise the DR data to asses which clusters represent cellular populations

```
    ### DR plots

        make.colour.plot(cell.sub, "UMAP_X", "UMAP_Y", "FlowSOM_metacluster", col.type = 'factor',
add.label = TRUE)
```

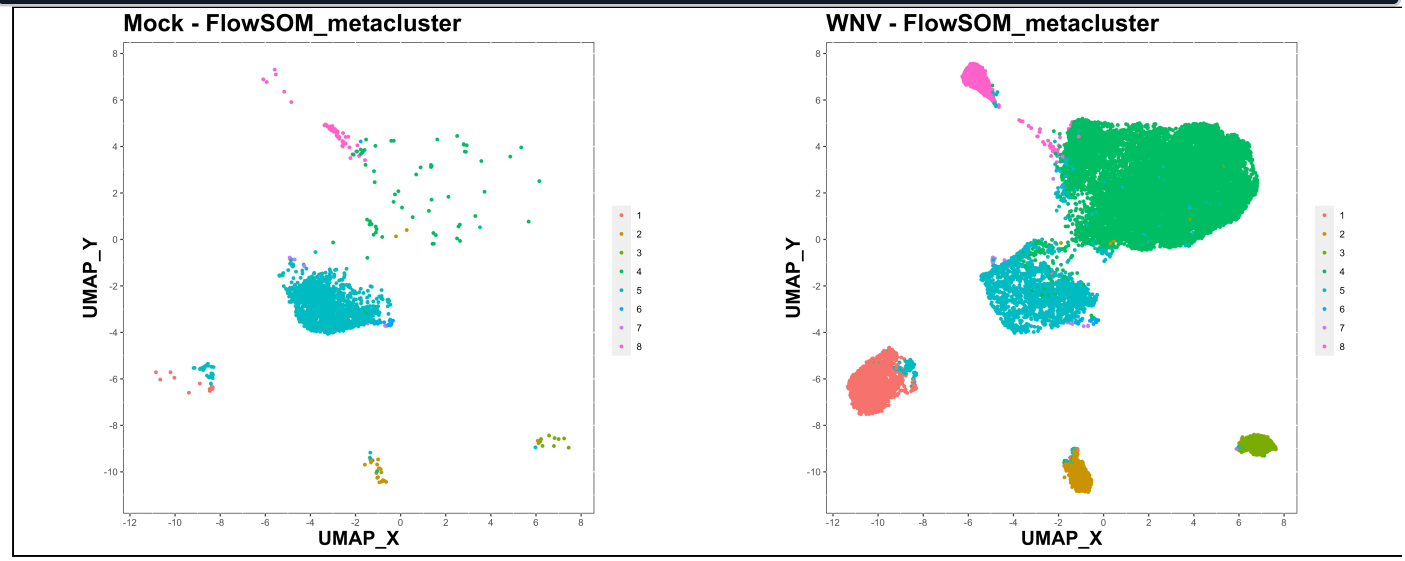



```
make.multi.plot(cell.sub, "UMAP_X", "UMAP_Y", cellular.cols)
```



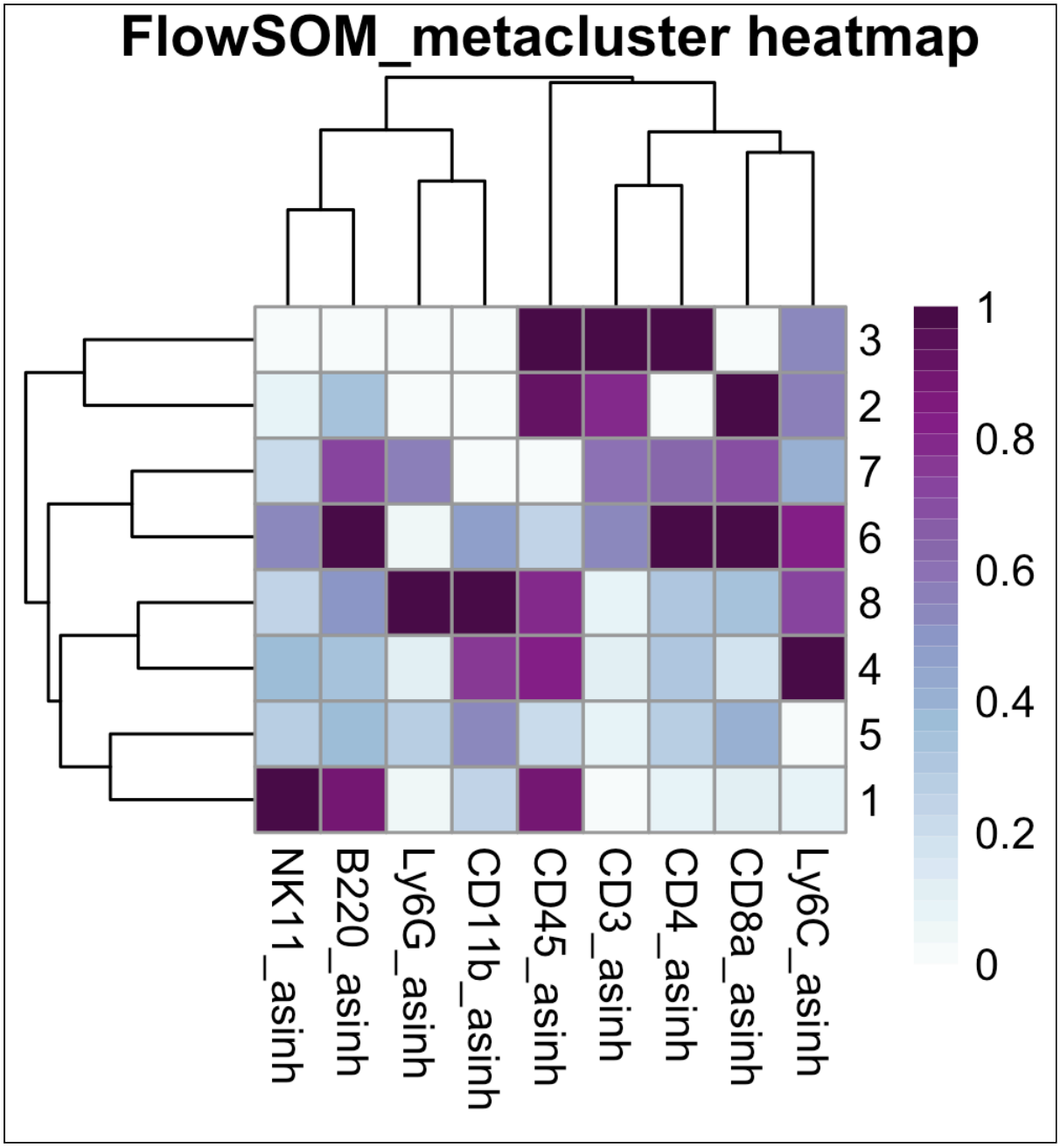
We can also generate some multi plots to compare between experimental groups or batches.

```
make.multi.plot(cell.sub, "UMAP_X", "UMAP_Y", "FlowSOM_metacluster", group.col, col.type = 'factor')
```



We can also produce expression heatmaps to help guide our interpretation of cluster identities.

```
### Expression heatmap

    exp <- do.aggregate(cell.dat, cellular.cols, by = "FlowSOM_metacluster")
    make.pheatmap(exp, "FlowSOM_metacluster", cellular.cols)
```



6. Annotate clusters

```
#########################################################################################################
###
#### 6. Annotate clusters
#########################################################################################################
###
```

Review the cluster labels and marker expression patterns, so you can annotate the clusters. This annotation is optional, as all subsequent steps can be performed on the 'clusters' instead of the 'populations'. Here we can create a list of population names, and then specify which clusters make up that population (e.g. CD4 T cells are contained within cluster '3').

```
    setwd(OutputDirectory)
    dir.create("Output 3 - annotation")
    setwd("Output 3 - annotation")

    ### Annotate

        annots <- list("CD4 T cells" = c(3),
                       "CD8 T cells" = c(2),
                       "NK cells" = c(1),
                       "Neutrophils" = c(8),
                       "Infil Macrophages" = c(4),
                       "Microglia" = c(5,6,7))
```

Once the annotation list is created, we can switch the list into a table format to annotate our data.

```
        annots <- do.list.switch(annots)
        names(annots) <- c("Values", "Population")
        setorderv(annots, 'Values')
        annots
```

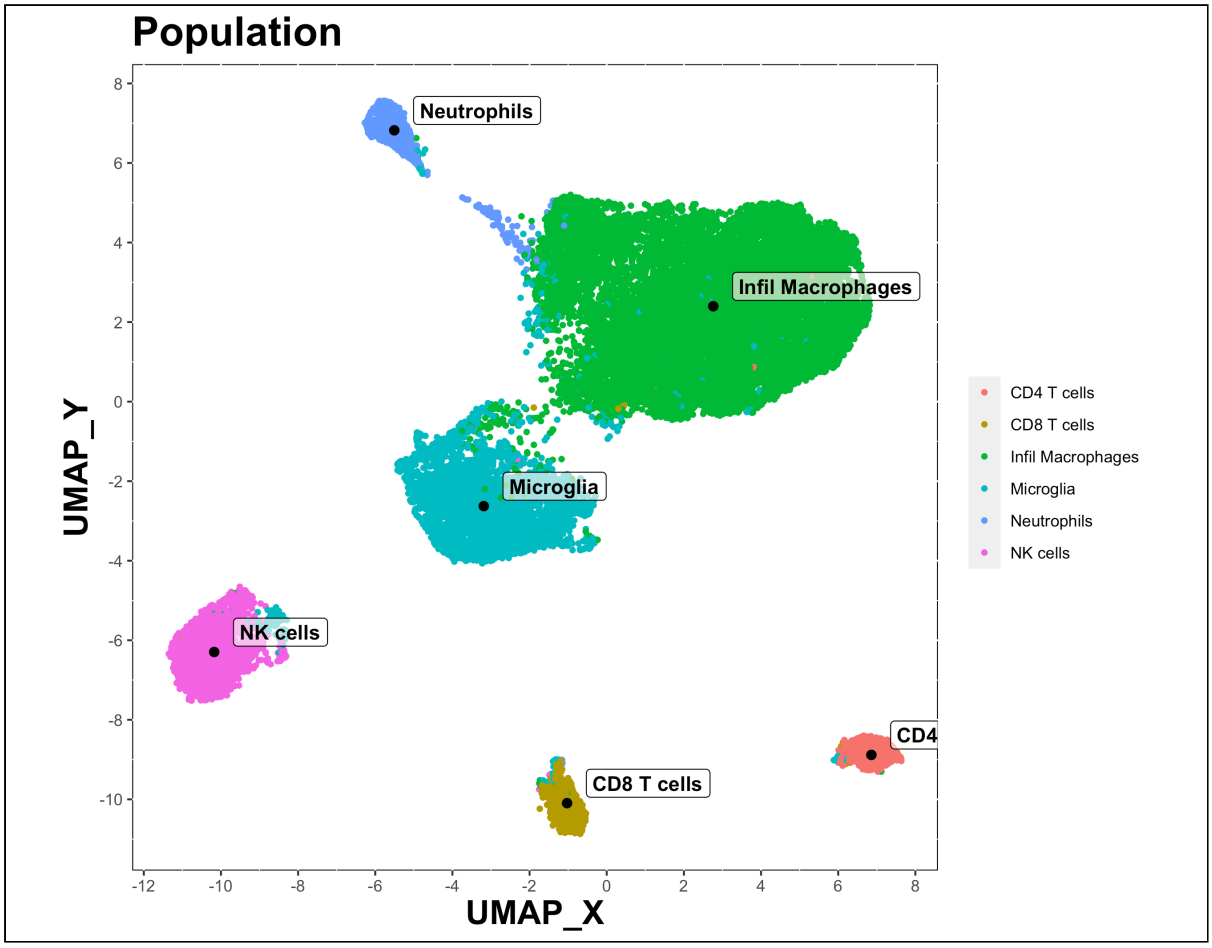
```
   Values        Population
1:      1          NK cells
2:      2       CD8 T cells
3:      3       CD4 T cells
4:      4 Infil Macrophages
5:      5         Microglia
6:      6         Microglia
7:      7         Microglia
8:      8       Neutrophils
```

Using the **do.add.cols** function, we can add the population names to the corresponding clusters.

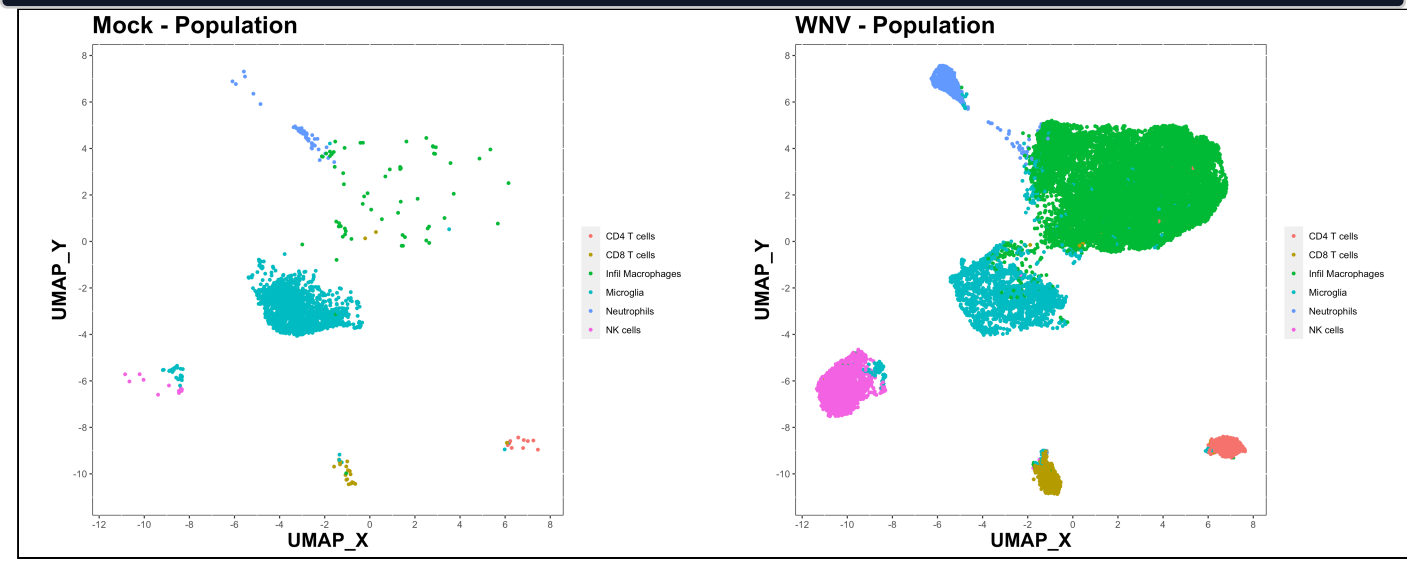
```
    ### Add annotations

        cell.dat <- do.add.cols(cell.dat, "FlowSOM_metacluster", annots, "Values")
        cell.dat

        fwrite(cell.dat, "Annotated.data.csv")

        cell.sub <- do.add.cols(cell.sub, "FlowSOM_metacluster", annots, "Values")
        cell.sub

        fwrite(cell.sub, "Annotated.data.DR.csv")
```

Subsequently, we can visualise the population labels on a UMAP plot.

```
        make.colour.plot(cell.sub, "UMAP_X", "UMAP_Y", "Population", col.type = 'factor', add.label =
TRUE)
```

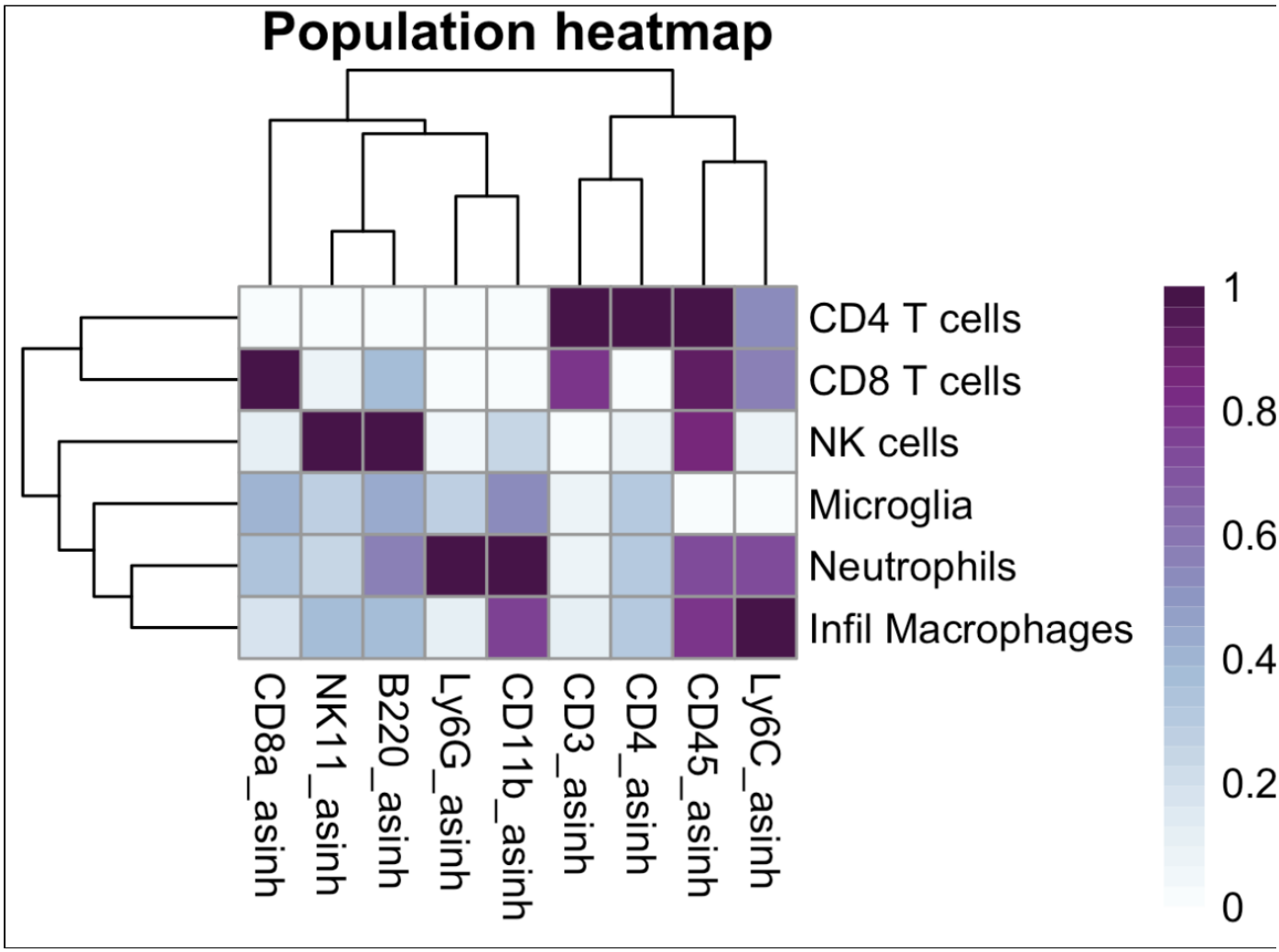



```
make.multi.plot(cell.sub, "UMAP_X", "UMAP_Y", "Population", group.col, col.type = 'factor')
```



We can also generate an expression heatmap to summarise the expression levels of each marker on our populations.

```
    ### Expression heatmap

        rm(exp)
        exp <- do.aggregate(cell.dat, cellular.cols, by = "Population")
        make.pheatmap(exp, "Population", cellular.cols)
```




```
### Write FCS files

        setwd(OutputDirectory)
        setwd("Output 3 - annotation")

        dir.create('FCS files')
        setwd('FCS files')

        write.files(cell.dat,
                    file.prefix = exp.name,
                    divide.by = sample.col,
                    write.csv = FALSE,
                    write.fcs = TRUE)
```

7. Summary data, graphs, statistics

```
#########################################################################################################
###
#### 7. Summary data
#########################################################################################################
###
```

Here we can create 'summary' data for our experiment. This involves calculating the percentage of each population in each sample, along with the corresponding cell counts if the information is available. In addition, we calculate the MFI for selected markers on each population in each sample.

First, set the working directory, and select which columns we will measure the MFI of. In this case, **CD11b_asinh** and **Ly6C_asinh**.

```
    setwd(OutputDirectory)
    dir.create("Output 4 - summary data")
    setwd("Output 4 - summary data")

    ### Setup

        variance.test <- 'kruskal.test'
        pairwise.test <- "wilcox.test"

        comparisons <- list(c("Mock", "WNV"))
        comparisons

        grp.order <- c("Mock", "WNV")
        grp.order
```

We can also specify which columns we wish to measure MFI levels on.

```
    ### Select columns to measure MFI

        as.matrix(cellular.cols)
        dyn.cols <- cellular.cols[c(5,8)]
        dyn.cols
```

Use the new **create.sumtable** function to generate summary data – a data.table of samples (rows) vs measurements (columns).

```
    ### Create summary tables

        sum.dat <- create.sumtable(dat = cell.dat,
                                   sample.col = sample.col,
                                   pop.col = "Population",
                                   use.cols = dyn.cols,
                                   annot.cols = c(group.col, batch.col),
                                   counts = counts)
```

Once the summary data has been generated, we can review it and select which columns to plot. In each case, the column names (i.e. name of each summary measure) are structured as 'MEASURE TYPE -- POPULATION'. This provides a useful structure, as we can use regular expression searches to split the name into just the MEASURE TYPE or POPULATION segment.

```
    ### Review summary data

        sum.dat
        as.matrix(names(sum.dat))
```

```
      [,1]
 [1,] "Sample"
 [2,] "Group"
 [3,] "Batch"
 [4,] "Percent of sample -- CD4 T cells"
 [5,] "Percent of sample -- CD8 T cells"
 [6,] "Percent of sample -- Infil Macrophages"
 [7,] "Percent of sample -- Microglia"
 [8,] "Percent of sample -- Neutrophils"
 [9,] "Percent of sample -- NK cells"
[10,] "Cells per sample -- CD4 T cells"
[11,] "Cells per sample -- CD8 T cells"
[12,] "Cells per sample -- Infil Macrophages"
[13,] "Cells per sample -- Microglia"
[14,] "Cells per sample -- Neutrophils"
[15,] "Cells per sample -- NK cells"
[16,] "MFI of CD11b_asinh -- CD4 T cells"
[17,] "MFI of CD11b_asinh -- CD8 T cells"
[18,] "MFI of CD11b_asinh -- Infil Macrophages"
[19,] "MFI of CD11b_asinh -- Microglia"
[20,] "MFI of CD11b_asinh -- Neutrophils"
[21,] "MFI of CD11b_asinh -- NK cells"
[22,] "MFI of Ly6C_asinh -- CD4 T cells"
[23,] "MFI of Ly6C_asinh -- CD8 T cells"
[24,] "MFI of Ly6C_asinh -- Infil Macrophages"
[25,] "MFI of Ly6C_asinh -- Microglia"
[26,] "MFI of Ly6C_asinh -- Neutrophils"
[27,] "MFI of Ly6C_asinh -- NK cells"
```

Specify which columns we want to plot.

```
plot.cols <- names(sum.dat)[c(4:27)]
plot.cols
```

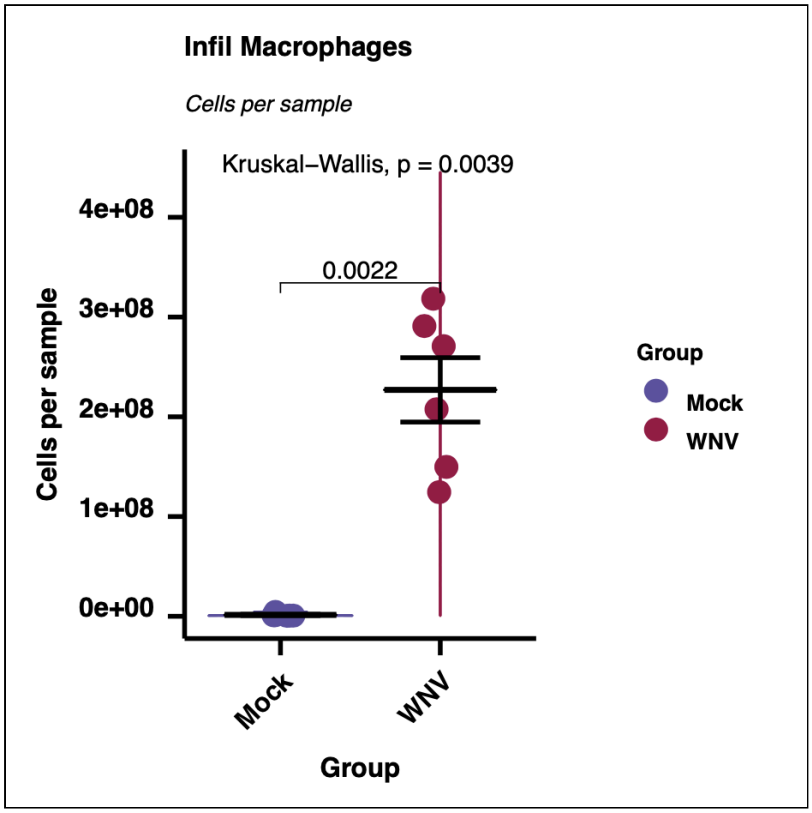
Reorder the data such that sample appear in the specify group order.

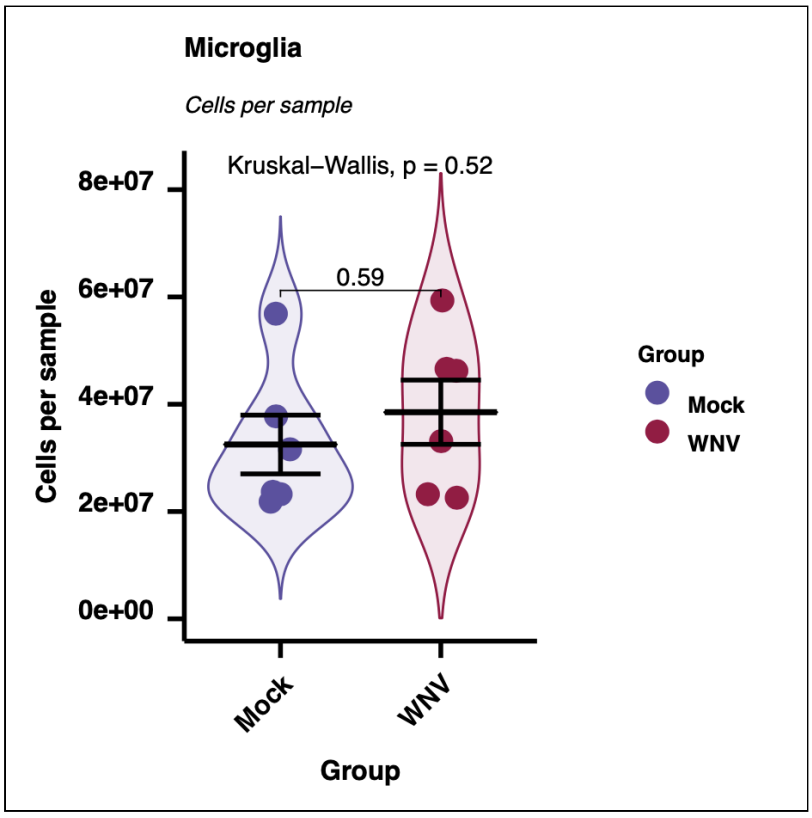
```
### Reorder summary data

sum.dat <- do.reorder(sum.dat, group.col, grp.order)
sum.dat[,c(1:3)]
```

Violin/scatter plots

We can use the **run.autograph** function to create violin/scatter plots with embedded statistic – one per population/ measurement type.

```
    ### Autographs

        for(i in plot.cols){

            measure <- gsub("\\ --.*", "", i)
            measure

            pop <- gsub("^[^--]*.-- ", "", i)
            pop

            make.autograph(sum.dat,
                           x.axis = group.col,
                           y.axis = i,
                           y.axis.label = measure,

                           grp.order = grp.order,
                           my_comparisons = comparisons,

                           Variance_test = variance.test,
                           Pairwise_test = pairwise.test,

                           title = pop,
                           subtitle = measure,
                           filename = paste0(i, '.pdf'))
        }
```

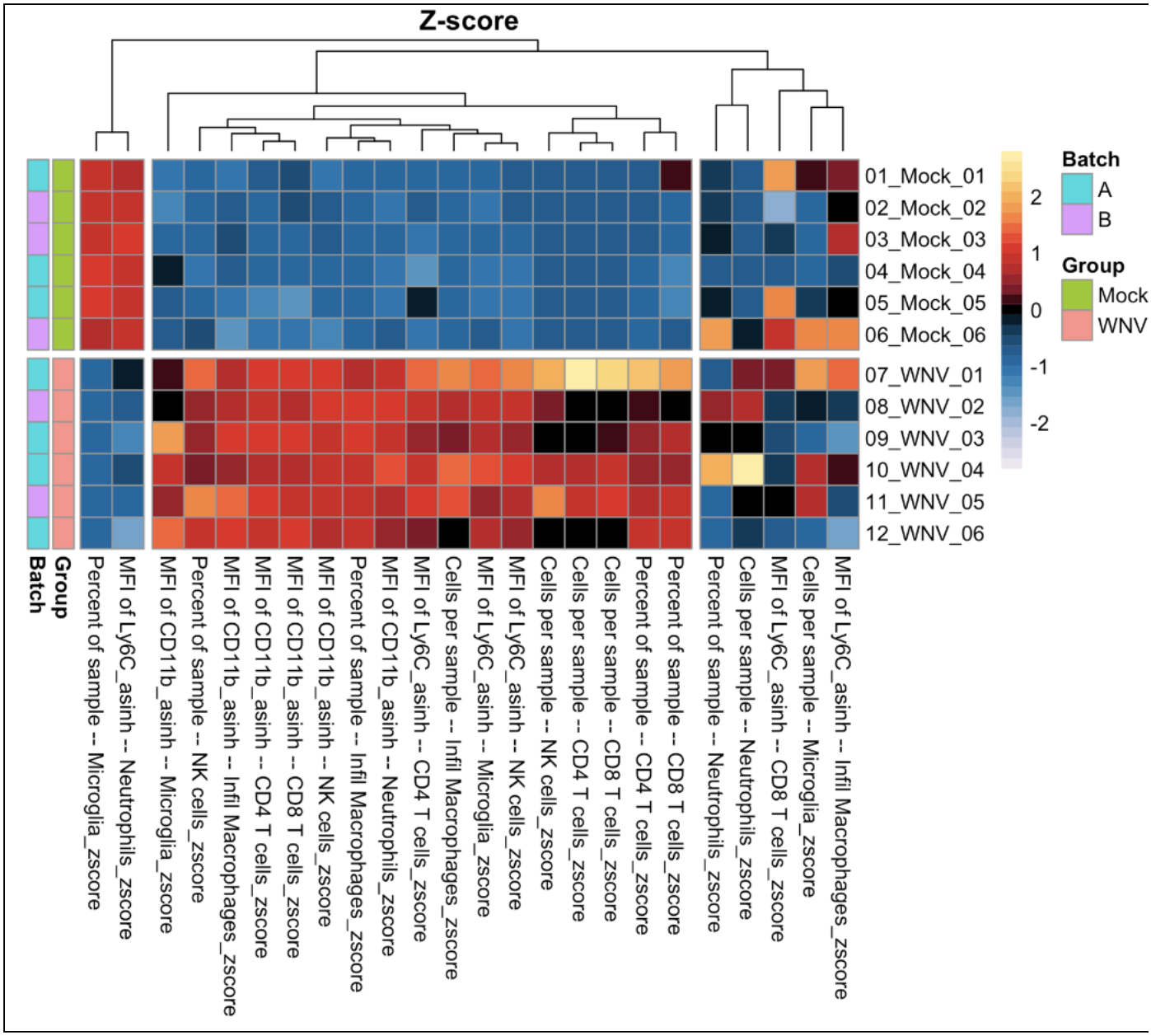




Heatmaps

We can also create a global heatmap show the z-score of each population/measurement type against each sample.

```
### Create a fold change heatmap

        ## Z-score calculation
        sum.dat.z <- do.zscore(sum.dat, plot.cols)

        ## Group
        t.first <- match(grp.order, sum.dat.z[[group.col]])
        t.first <- t.first -1
        t.first

        ## Make heatmap
        make.pheatmap(sum.dat.z,
                      sample.col = sample.col,
                      plot.cols = paste0(plot.cols, '_zscore'),
                      is.fold = TRUE,
                      plot.title = 'Z-score',
                      dendrograms = 'column',
                      row.sep = t.first,
                      cutree_cols = 3)
```



8. Output session info

```
##############################################################################################################
###
#### 8. Output session info and FCS files
##############################################################################################################
###
```

Create "Output-info" directory and save session data

For the final step of our setup, we want to record the session info our R session, and save this in a folder we'll call "Output-info".

```
### Session info and metadata

    setwd(OutputDirectory)
    dir.create("Output - info", showWarnings = FALSE)
    setwd("Output - info")

    sink(file = "session_info.txt", append=TRUE, split=FALSE, type = c("output", "message"))
    session_info()
    sink()

    write(cellular.cols, "cellular.cols.txt")
    write(cluster.cols, "cluster.cols.txt")
```