Introduction

This alignment workflow replaces the 'general discovery' workflow, which is still accessible here.

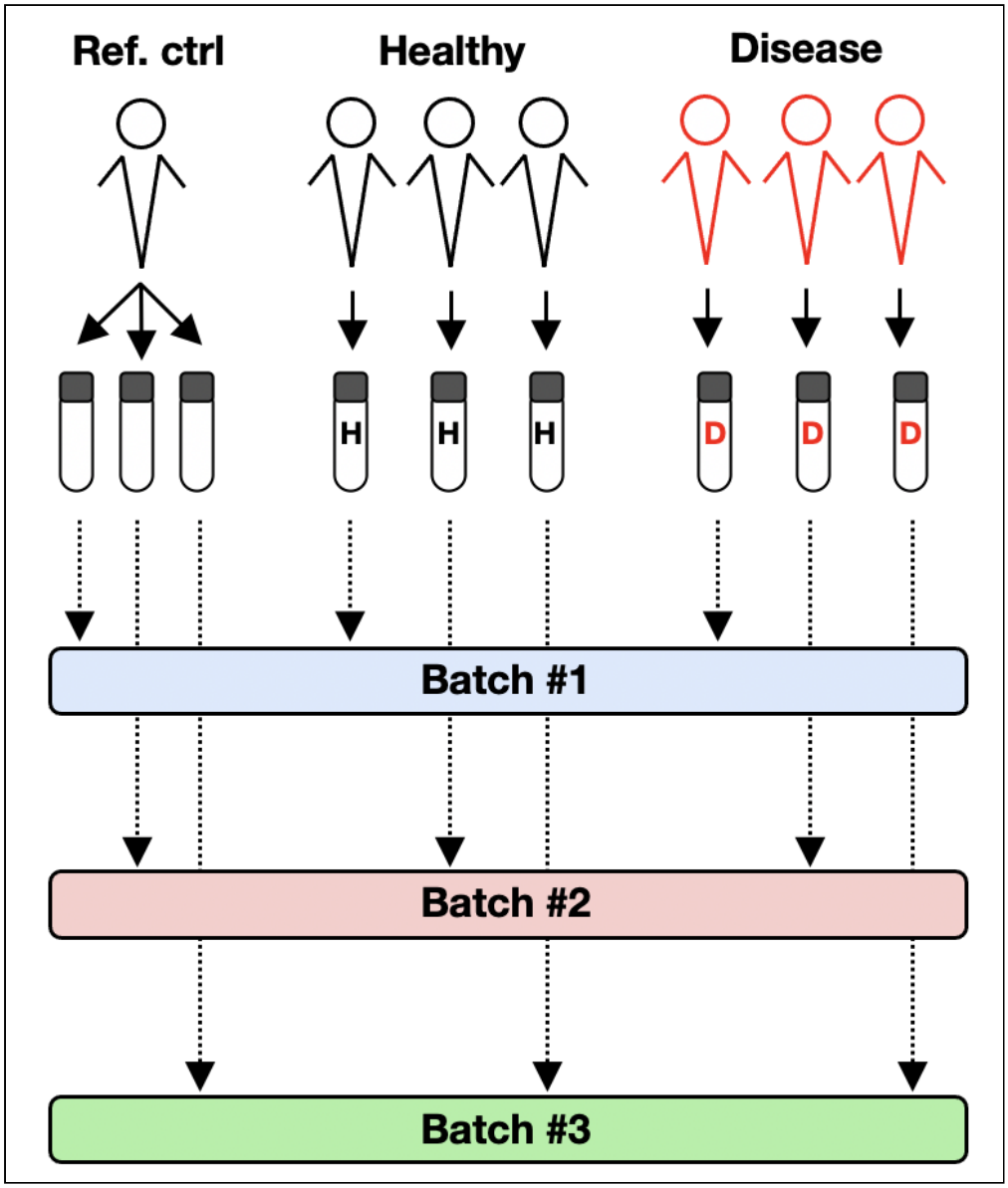
Overview

The batch alignment and analysis workflow builds on the 'simple discovery' workflow by adding a step to facilitate batch alignment. This workflow allows for the correction of technical variation or shifts in signal levels in samples stained and/or acquired across multiple batches. To do this, we have implemented the CytoNorm algorithm (Van Gassen 2020). CytoNorm uses reference control samples that are prepared and recorded along with each batch of samples to identify and correct technical variations between individual batches, while preserving biologically relevant differences. For more information on CytoNorm, see Van Gassen et al 2020, and for more information on our implementation in Spectre, see Ashhurst et al 2021.

The demo dataset used for this worked example are cells extracted from mock- or virally-infected mouse bone marrow, measured by flow cytometry. These samples were stained and acquired in two batches.

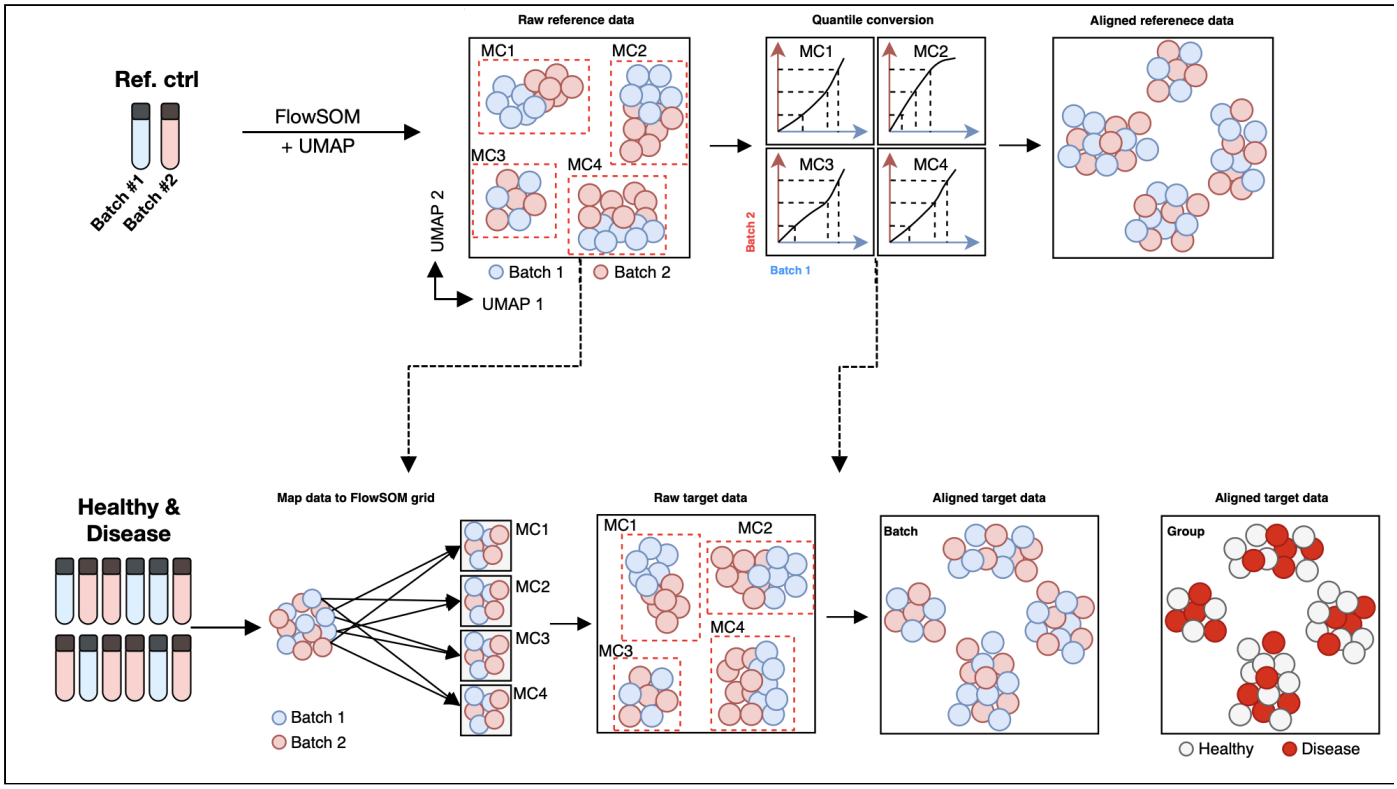
Reference controls for CytoNorm

An example of reference control samples are aliquots of peripheral blood mononuclear cells (PBMCs) that are derived from a single donor at one time point, and cryopreserved (i.e. multiple aliquots of a biologically identical sample). Each time a set of PBMC samples from the study cohort are thawed, stained, and recorded, a reference controls is also thawed, stained, and recorded. Differences in signal level between the reference controls allows CytoNorm to learn the differences in signal levels due to the batches, and correct them, while preserving biological differences between the individual samples. In our demo dataset, we are using bone marrow samples derived from separate mice. Though not derived from the same mouse, these are similar enough that they can be used successfully as reference controls.



Correction of batch effects with CytoNorm

Reference samples are clustered together using FlowSOM, where we attempt to captures cells from matching populations in each batch in a single metacluster (e.g. neutrophils from batch 1 and 2 are captured in metacluster 1, etc). This assumes a reasonably low level of batch effect, consisting of small shifts in the expression levels for one or more markers. Because the proportion of cells in each population of the reference samples are identical, FlowSOM can use quantile distributions for each marker on each metacluser to create a model which will adjust the data values, removing technical variation between batches. This model is then applied to all samples in each batch.



Requirements

The reference controls can only correct batch effects for markers which are actually expressed on the reference controls. For example, some activation markers may be expressed on samples from the 'disease' group, but won't be present on reference controls derived from healthy donors. In this case, we would simply not attempt to perform alignment on that marker. Typically, stable phenotyping markers (e.g. CD4, CD8) would be expressed strongly enough in reference samples, and would be suitable for alignment.

Other approaches

If you have samples derived from multiple batches, but do not have reference controls processed with each batch, then other forms of batch alignment might be possible – see our data integration options for more information.

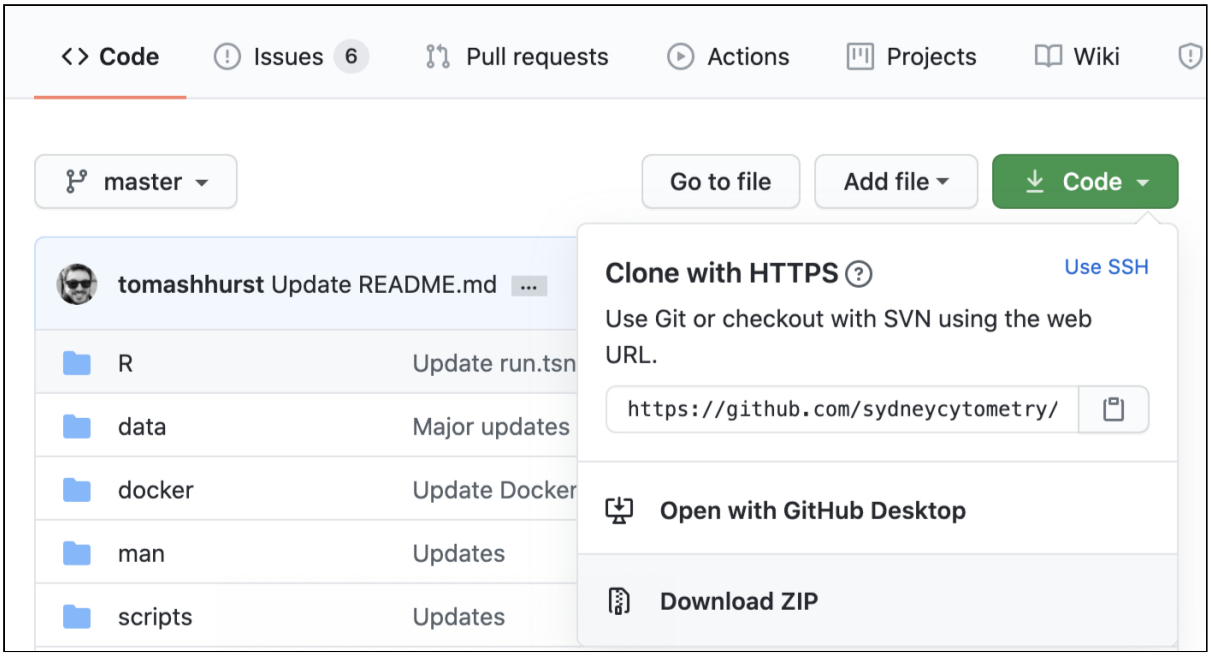
Citation

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.

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:



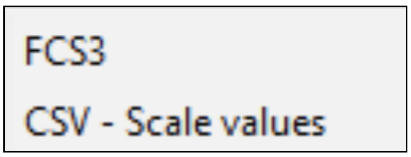
You can then find the 'batch alignment and analysis workflow' script under **'worked examples':**

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

Data preparation and export from FlowJo (or similar)

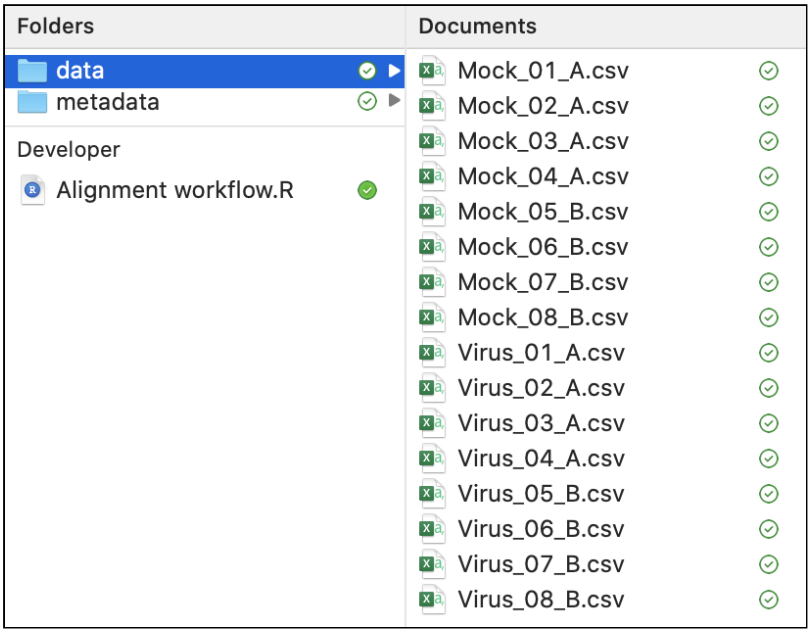
Export the population of interest (POI) from your files.

Please see this page for detailed instructions on exporting data for Spectre. 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 on CSV channel values.*



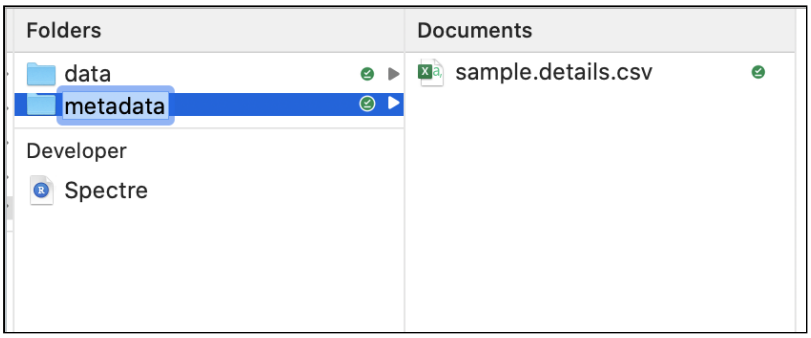
Data folder

Create a folder within your experiment folder called 'data', and place the exported files there (see this page for more info)..



Metadata folder

Create a folder within your experiment folder called 'metadata', and place a 'sample.details.csv' file there (see this page for more info).



FileName	Sample	Group	Batch
Mock_01_A.csv	Mock_01_A	Mock	A
Mock_02_A.csv	Mock_02_A	Mock	A
Mock_03_A.csv	Mock_03_A	Mock	A
Mock_04_A.csv	Mock_04_A	Mock	A
Virus_01_A.csv	WNV_01_A	Virus	A
Virus_02_A.csv	WNV_02_A	Virus	A
Virus_03_A.csv	WNV_03_A	Virus	A
Virus_04_A.csv	WNV_04_A	Virus	A
Mock_05_B.csv	Mock_05_B	Mock	B
Mock_06_B.csv	Mock_06_B	Mock	B
Mock_07_B.csv	Mock_07_B	Mock	B
Mock_08_B.csv	Mock_08_B	Mock	B
Virus_05_B.csv	WNV_05_B	Virus	B
Virus_06_B.csv	WNV_06_B	Virus	B
Virus_07_B.csv	WNV_07_B	Virus	B
Virus_08_B.csv	WNV_08_B	Virus	B

1. Load packages and set directories

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

```
#########################################################################################################
###
#### 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.

```
### Install CytoNorm package

        library(devtools)
        install_github('saeyslab/CytoNorm')

### 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()
        PrimaryDirectory
```

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]]
```

```
       PECy5-5 CD3e PECy7 CD16-32 DL800 Ly6G AF700 CD45 APCCy7 CD48 BUV395 CD11b BUV737 B220 BV605 Ly6C
FileName FileNo
    1:      51.6403      1239.360   -681.257   1752.270    2600.430      627.259  17763.6000   128.9340
 Mock_01_A      1
    2:      58.8824      -140.921   -267.039   1188.640     392.962      304.615  11834.8000   907.3650
 Mock_01_A      1
    3:      60.0077      2191.260   -362.088   1676.960    1441.300     4740.710    185.9470  3915.8200
 Mock_01_A      1
    4:     313.9580      2532.700   -438.388    478.109    1809.220     9440.380   1056.2900  6221.6700
 Mock_01_A      1
    5:     -83.0391      1603.390  -1200.350    564.994    3946.200     1636.110   4579.8700   274.5270
 Mock_01_A      1
   ---
 9996:    -121.9210      4626.660   7377.120   1600.950     197.503    12732.300    -18.9127  3826.8000
 Mock_01_A      1
```

```
 9997:    5635.1700       807.513   -831.751   1750.210    2388.820     1106.890   -394.7910    79.3191
Mock_01_A       1
 9998:     132.8540       185.348    780.195    611.077    1471.310      823.702   6442.3800  -345.6880
Mock_01_A       1
 9999:     -78.8966      2564.800  10150.800   2161.430     264.904    12933.400   -106.9610  8881.1600
Mock_01_A       1
10000:      17.2606       531.683   -801.073    350.229    1963.560      793.921   8161.8700   288.1360
Mock_01_A       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
```

```
        PECy5-5 CD3e PECy7 CD16-32 DL800 Ly6G AF700 CD45 APCCy7 CD48 BUV395 CD11b BUV737 B220 BV605
Ly6C   FileName FileNo
     1:      51.6403     1239.3600   -681.257   1752.270    2600.430      627.259   17763.600
128.934  Mock_01_A      1
     2:      58.8824     -140.9210   -267.039   1188.640     392.962      304.615   11834.800
907.365  Mock_01_A      1
     3:      60.0077     2191.2600   -362.088   1676.960    1441.300     4740.710     185.947
3915.820  Mock_01_A      1
     4:     313.9580     2532.7000   -438.388    478.109    1809.220     9440.380    1056.290
6221.670  Mock_01_A      1
     5:     -83.0391     1603.3900  -1200.350    564.994    3946.200     1636.110    4579.870
274.527  Mock_01_A      1
    ---
159996:     106.3880     1661.1200   -114.892   2138.700    8311.230      127.068     258.936
6161.070 Virus_08_B     16
159997:    1750.2300       65.4275   -843.984  19655.900    7874.900      122.179     550.596
132.445 Virus_08_B     16
159998:    -159.2870     5363.3900  17389.300   2578.650    -362.093    12091.600     121.659
4257.980 Virus_08_B     16
159999:     -62.6836     3132.7100   1941.660   1548.130     577.625    12904.000     338.244
1030.080 Virus_08_B     16
160000:     650.9440     1991.4900    254.429   1677.730    2209.210    22242.300     789.639
12467.000 Virus_08_B     16
```

Read in sample metadata

```
    ### Read in metadata

        setwd(MetaDirectory)

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

```
         FileName    Sample Group Batch
1:  Mock_01_A.csv Mock_01_A  Mock     A
2:  Mock_02_A.csv Mock_02_A  Mock     A
```

```
 3:  Mock_03_A.csv Mock_03_A  Mock     A
 4:  Mock_04_A.csv Mock_04_A  Mock     A
 5: Virus_01_A.csv  WNV_01_A Virus     A
 6: Virus_02_A.csv  WNV_02_A Virus     A
 7: Virus_03_A.csv  WNV_03_A Virus     A
 8: Virus_04_A.csv  WNV_04_A Virus     A
 9:  Mock_05_B.csv Mock_05_B  Mock     B
10:  Mock_06_B.csv Mock_06_B  Mock     B
11:  Mock_07_B.csv Mock_07_B  Mock     B
12:  Mock_08_B.csv Mock_08_B  Mock     B
13: Virus_05_B.csv  WNV_05_B Virus     B
14: Virus_06_B.csv  WNV_06_B Virus     B
15: Virus_07_B.csv  WNV_07_B Virus     B
16: Virus_08_B.csv  WNV_08_B Virus     B
```

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.

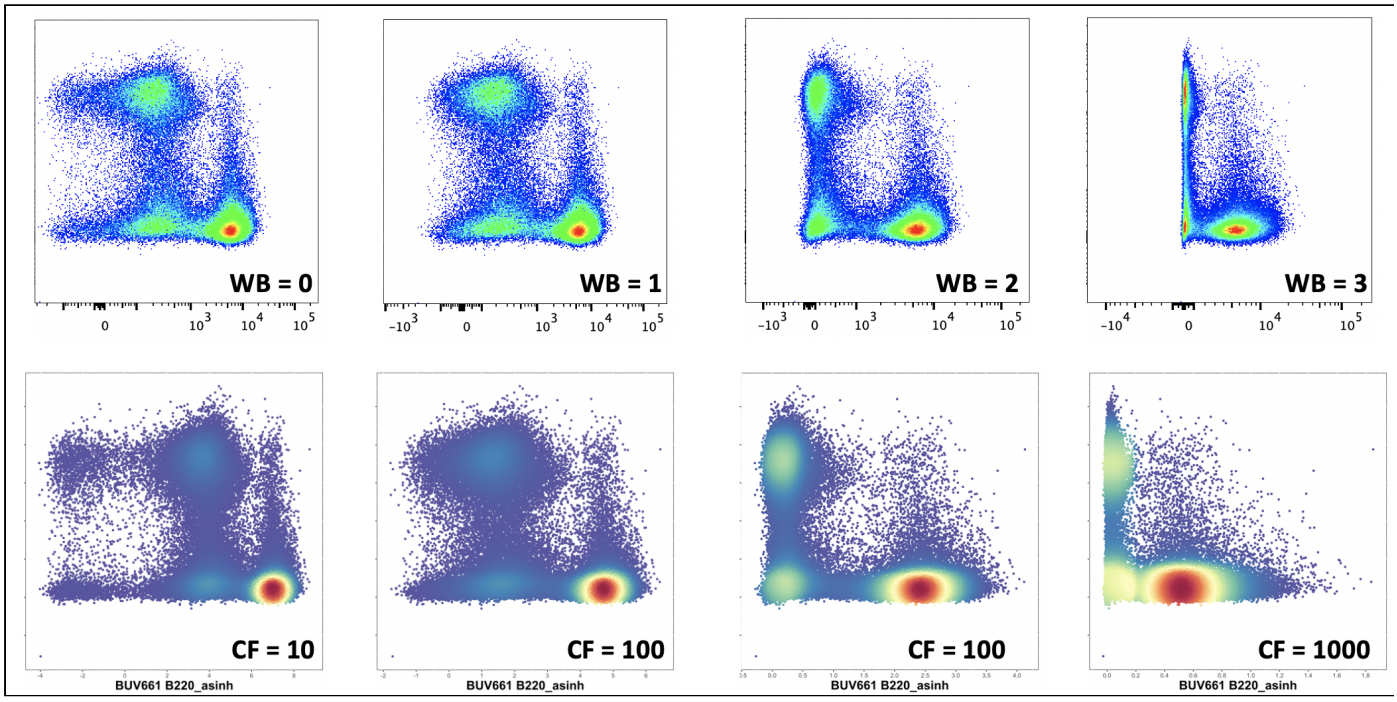
```
#########################################################################################################
###
#### 3. Data transformation
#########################################################################################################
###

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

    ### Arcsinh transformation

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

```
      [,1]
 [1,] "PECy5-5 CD3e"
 [2,] "PECy7 CD16-32"
 [3,] "DL800 Ly6G"
 [4,] "AF700 CD45"
 [5,] "APCCy7 CD48"
 [6,] "BUV395 CD11b"
 [7,] "BUV737 B220"
 [8,] "BV605 Ly6C"
 [9,] "FileName"
[10,] "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(1:8)]
        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 <- "BV605 Ly6C_asinh"
```

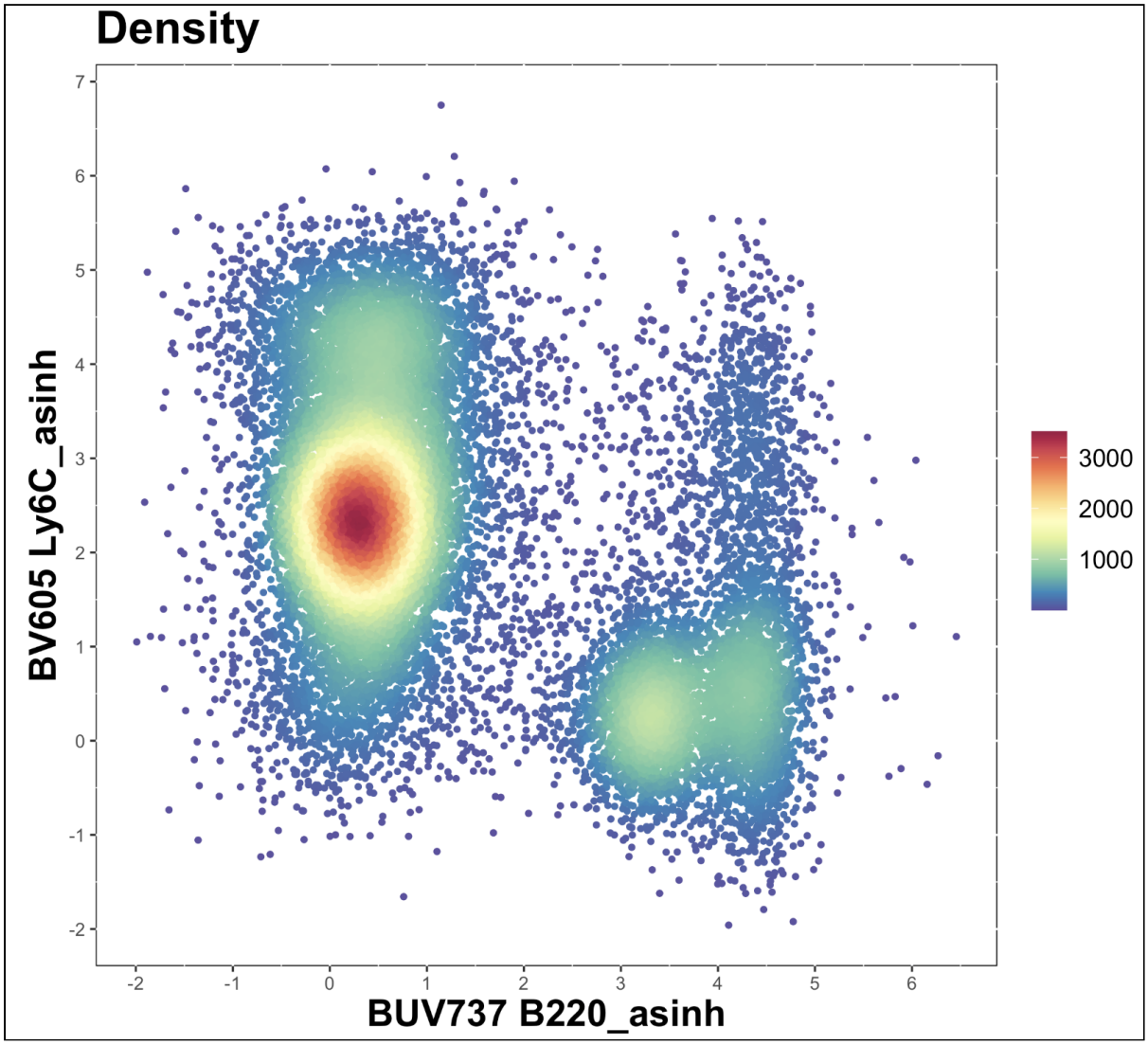
Now we need to apply arcsinh transformation to the data in those columns, using a specific co-factor.

```
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.

```
    ### Add metadata to data.table

        meta.dat
```

```
          FileName    Sample Group Batch
1:  Mock_01_A.csv Mock_01_A  Mock     A
```

```
 2:  Mock_02_A.csv Mock_02_A  Mock     A
 3:  Mock_03_A.csv Mock_03_A  Mock     A
 4:  Mock_04_A.csv Mock_04_A  Mock     A
 5: Virus_01_A.csv  WNV_01_A Virus     A
 6: Virus_02_A.csv  WNV_02_A Virus     A
 7: Virus_03_A.csv  WNV_03_A Virus     A
 8: Virus_04_A.csv  WNV_04_A Virus     A
 9:  Mock_05_B.csv Mock_05_B  Mock     B
10:  Mock_06_B.csv Mock_06_B  Mock     B
11:  Mock_07_B.csv Mock_07_B  Mock     B
12:  Mock_08_B.csv Mock_08_B  Mock     B
13: Virus_05_B.csv  WNV_05_B Virus     B
14: Virus_06_B.csv  WNV_06_B Virus     B
15: Virus_07_B.csv  WNV_07_B Virus     B
16: Virus_08_B.csv  WNV_08_B Virus     B
```

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
```

```
          FileName    Sample Group Batch
 1:  Mock_01_A.csv Mock_01_A  Mock     A
 2:  Mock_02_A.csv Mock_02_A  Mock     A
 3:  Mock_03_A.csv Mock_03_A  Mock     A
 4:  Mock_04_A.csv Mock_04_A  Mock     A
 5: Virus_01_A.csv  WNV_01_A Virus     A
 6: Virus_02_A.csv  WNV_02_A Virus     A
 7: Virus_03_A.csv  WNV_03_A Virus     A
 8: Virus_04_A.csv  WNV_04_A Virus     A
 9:  Mock_05_B.csv Mock_05_B  Mock     B
10:  Mock_06_B.csv Mock_06_B  Mock     B
11:  Mock_07_B.csv Mock_07_B  Mock     B
12:  Mock_08_B.csv Mock_08_B  Mock     B
13: Virus_05_B.csv  WNV_05_B Virus     B
14: Virus_06_B.csv  WNV_06_B Virus     B
15: Virus_07_B.csv  WNV_07_B Virus     B
16: Virus_08_B.csv  WNV_08_B Virus     B
```

In this case we don't have any cell counts in the meta.data table, so we have commented out the creation of 'counts'.

```
# 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
```

```
      ...     BUV737 B220_asinh BV605 Ly6C_asinh    Sample Group Batch
   1: ...             4.2636438        0.2550924 Mock_01_A  Mock     A
   2: ...             3.8577846        1.3575720 Mock_01_A  Mock     A
```

```
     3: ...            0.3638149        2.7553704 Mock_01_A  Mock     A
     4: ...            1.4928782        3.2159434 Mock_01_A  Mock     A
     5: ...            2.9109315        0.5246514 Mock_01_A  Mock     A
    ---
159996: ...            0.4971396        3.2061873  WNV_08_B Virus     B
159997: ...            0.9511485        0.2618862  WNV_08_B Virus     B
159998: ...            0.2409789        2.8385188  WNV_08_B Virus     B
159999: ...            0.6332984        1.4702206  WNV_08_B Virus     B
160000: ...            1.2379493        3.9097814  WNV_08_B Virus     B
```

Check the column names.

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

```
      [,1]
 [1,] "PECy5-5 CD3e"
 [2,] "PECy7 CD16-32"
 [3,] "DL800 Ly6G"
 [4,] "AF700 CD45"
 [5,] "APCCy7 CD48"
 [6,] "BUV395 CD11b"
 [7,] "BUV737 B220"
 [8,] "BV605 Ly6C"
 [9,] "FileName"
[10,] "FileNo"
[11,] "PECy5-5 CD3e_asinh"
[12,] "PECy7 CD16-32_asinh"
[13,] "DL800 Ly6G_asinh"
[14,] "AF700 CD45_asinh"
[15,] "APCCy7 CD48_asinh"
[16,] "BUV395 CD11b_asinh"
[17,] "BUV737 B220_asinh"
[18,] "BV605 Ly6C_asinh"
[19,] "Sample"
[20,] "Group"
[21,] "Batch"
```

Specify columns that represent cellular features (in this case, the arcsinh transformed data, defined by "markername_asinh"). In this case, columns 11 to 18.

```
### Define cellular columns

    as.matrix(names(cell.dat))

    cellular.cols <- names(cell.dat)[c(11:18)]
    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.

```
### Define clustering columns

    as.matrix(names(cell.dat))

    cluster.cols <- names(cell.dat)[c(11:18)]
```

```
        as.matrix(cluster.cols)
```

Specify sample, group, and batch columns

```
    ### Define other key columns

        as.matrix(names(cell.dat))

        exp.name <- "BM 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 that the group names appear in the dataset** (you can determine this by running the **unique** function below.

```
        as.matrix(unique(cell.dat[[group.col]]))
```

```
     [,1]
[1,] "Mock"
[2,] "Virus"
```

In this example we want 10,000 cells from 'mock' and 10,000 cells from 'Virus', to reflect the number of cells present in each group.

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

5. Batch alignment

```
###############################################################################################################
###
#### 5. Batch alignment
###############################################################################################################
###
```

Set an output directory for the alignment results.

```
    setwd(OutputDirectory)
```

```
    dir.create("Output 2 - alignment")
    setwd("Output 2 - alignment")
```

Briefly review the sample.info.

```
    ### Extract reference samples

        sample.info
```

```
          FileName    Sample Group Batch
 1:  Mock_01_A.csv Mock_01_A  Mock     A
 2:  Mock_02_A.csv Mock_02_A  Mock     A
 3:  Mock_03_A.csv Mock_03_A  Mock     A
 4:  Mock_04_A.csv Mock_04_A  Mock     A
 5: Virus_01_A.csv  WNV_01_A Virus     A
 6: Virus_02_A.csv  WNV_02_A Virus     A
 7: Virus_03_A.csv  WNV_03_A Virus     A
 8: Virus_04_A.csv  WNV_04_A Virus     A
 9:  Mock_05_B.csv Mock_05_B  Mock     B
10:  Mock_06_B.csv Mock_06_B  Mock     B
11:  Mock_07_B.csv Mock_07_B  Mock     B
12:  Mock_08_B.csv Mock_08_B  Mock     B
13: Virus_05_B.csv  WNV_05_B Virus     B
14: Virus_06_B.csv  WNV_06_B Virus     B
15: Virus_07_B.csv  WNV_07_B Virus     B
16: Virus_08_B.csv  WNV_08_B Virus     B
```

Briefly review the unique sample names in the dataset.

```
        as.matrix(unique(cell.dat[[sample.col]]))
```

```
      [,1]
 [1,] "Mock_01_A"
 [2,] "Mock_02_A"
 [3,] "Mock_03_A"
 [4,] "Mock_04_A"
 [5,] "Mock_05_B"
 [6,] "Mock_06_B"
 [7,] "Mock_07_B"
 [8,] "Mock_08_B"
 [9,] "WNV_01_A"
[10,] "WNV_02_A"
[11,] "WNV_03_A"
[12,] "WNV_04_A"
[13,] "WNV_05_B"
[14,] "WNV_06_B"
[15,] "WNV_07_B"
[16,] "WNV_08_B"
```

Specify which samples are to be used as the 'reference' samples. In this case we will use "**Mock_01_A**" from batch A, and "**Mock_05_B**" from batch B.

```
        refs <- unique(cell.dat[[sample.col]])[c(1,5)]
        refs
```

```
[1] "Mock_01_A" "Mock_05_B"
```

Now we can create a new dataset (ref.dat) containing only cells from the reference samples (in this example, "**Mock_01_A**" from batch A, and "**Mock_05_B**" from batch B).

```
ref.dat <- do.filter(cell.dat, sample.col, refs)
ref.dat
```

```
       ...  BUV395 CD11b_asinh BUV737 B220_asinh BV605 Ly6C_asinh    Sample Group Batch
    1: ...            1.0504123          4.2636438        0.2550924 Mock_01_A  Mock     A
    2: ...            0.5767234          3.8577846        1.3575720 Mock_01_A  Mock     A
    3: ...            2.9452507          0.3638149        2.7553704 Mock_01_A  Mock     A
    4: ...            3.6319912          1.4928782        3.2159434 Mock_01_A  Mock     A
    5: ...            1.9011864          2.9109315        0.5246514 Mock_01_A  Mock     A
   ---
19996: ...           -0.1535622          3.6552341        0.9755986 Mock_05_B  Mock     B
19997: ...            3.9820237         -0.1828014        2.1916151 Mock_05_B  Mock     B
19998: ...            2.9497215          1.2291302        2.9677227 Mock_05_B  Mock     B
19999: ...            4.1217946          0.8717569        1.9755202 Mock_05_B  Mock     B
20000: ...            3.7923143          0.1008046        1.5941070 Mock_05_B  Mock     B
```

Setup a sub-directory for some pre-alignment plots.

```
### Initial clustering

setwd(OutputDirectory)
setwd("Output 2 - alignment")
dir.create("1 - ref pre-alignment")
setwd("1 - ref pre-alignment")
```

Run **prep.cytonorm** to initialise a cytonorm object, including clustering with FlowSOM. The number of metaclustes will be determined automatically, or you can specify a desired number of metaclusters by using the additional argument meta.k (e.g. meta.k = 20).

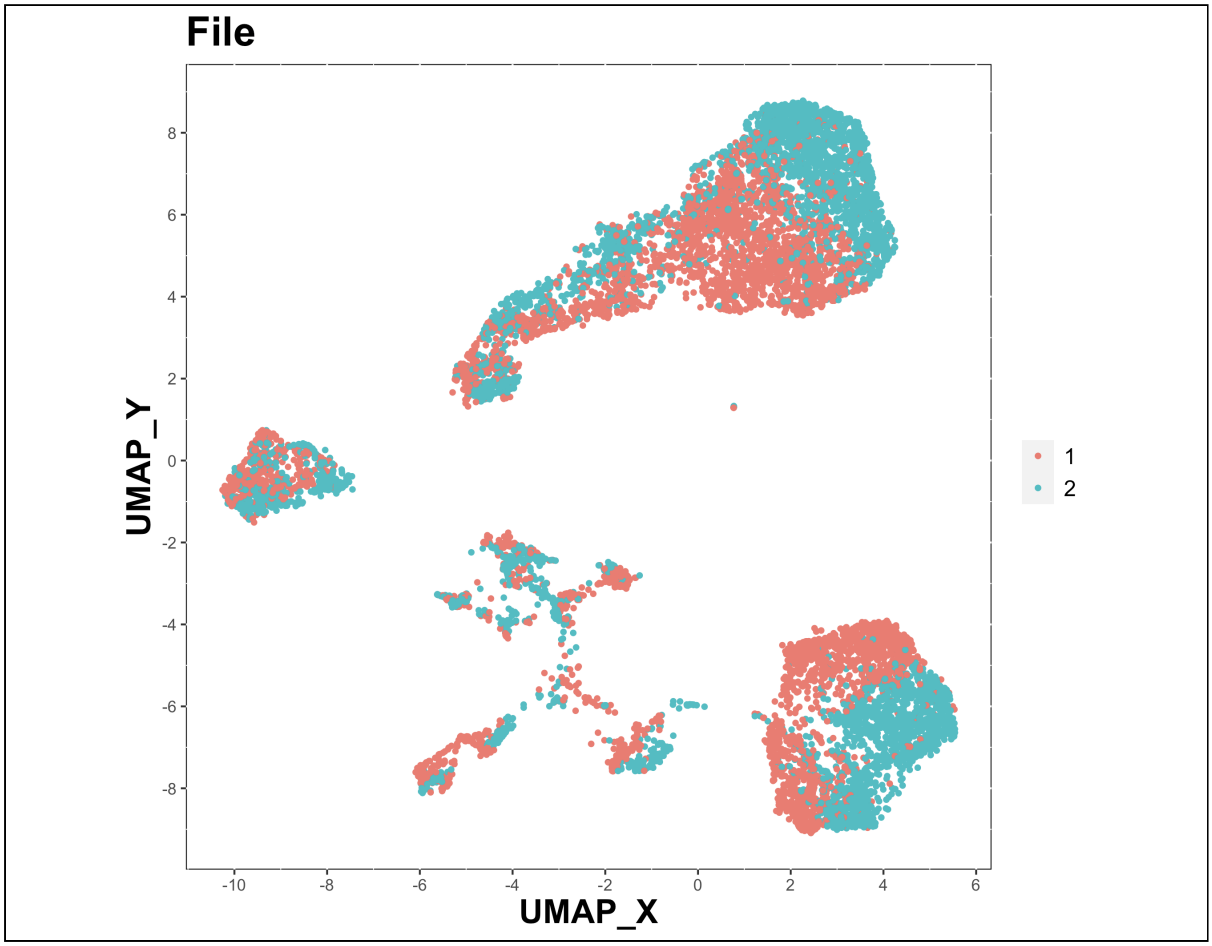
```
cytnrm <- prep.cytonorm(dat = ref.dat,
                        cellular.cols = cellular.cols,
                        cluster.cols = cluster.cols,
                        batch.col = batch.col,
                        sample.col = sample.col)

cytnrm
```

Once this is complete, we can subsample the dataset and plot it using UMAP.

```
cytnrm.sub <- do.subsample(cytnrm$dt, 10000)
cytnrm.sub <- run.umap(cytnrm.sub, use.cols = cluster.cols)
```

First, we will colour the cells by which batch they originated from. We can see batch effects evident by the offset between the red and green cells. In this case they are coloured by 'File'.

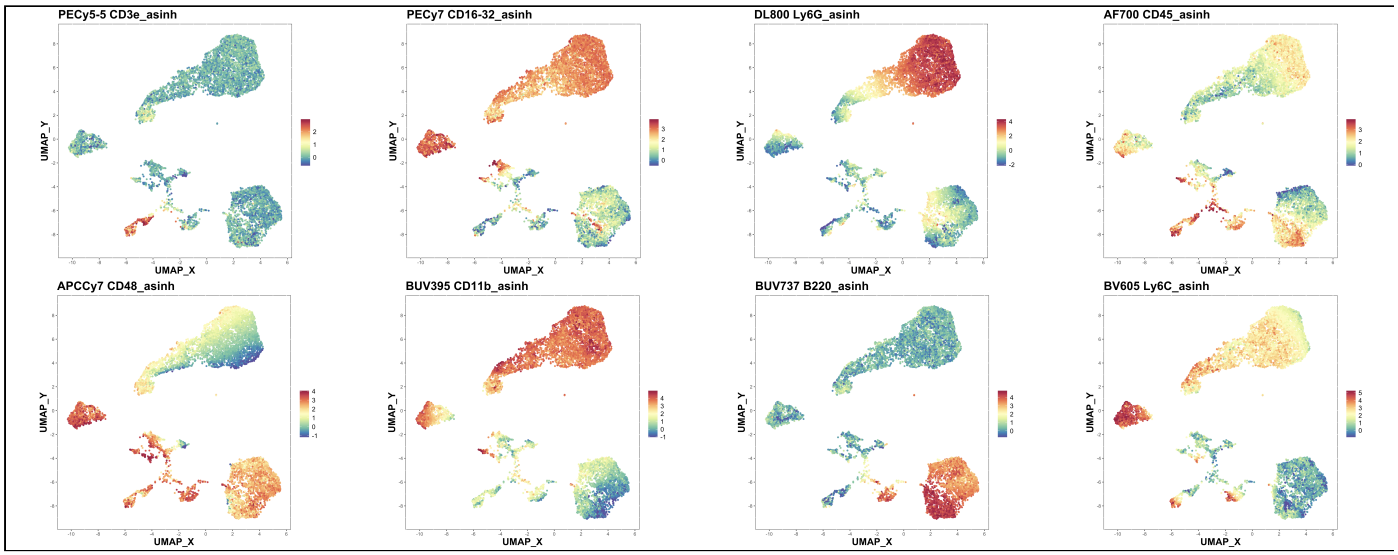
```
make.colour.plot(cytnrm.sub, 'UMAP_X', 'UMAP_Y', 'prep.fsom.metacluster', 'factor', add.label = TRUE)
```



You can see which file number corresponds to each batch by running **cytnrm$files** and **cytnrm$file.nums**.

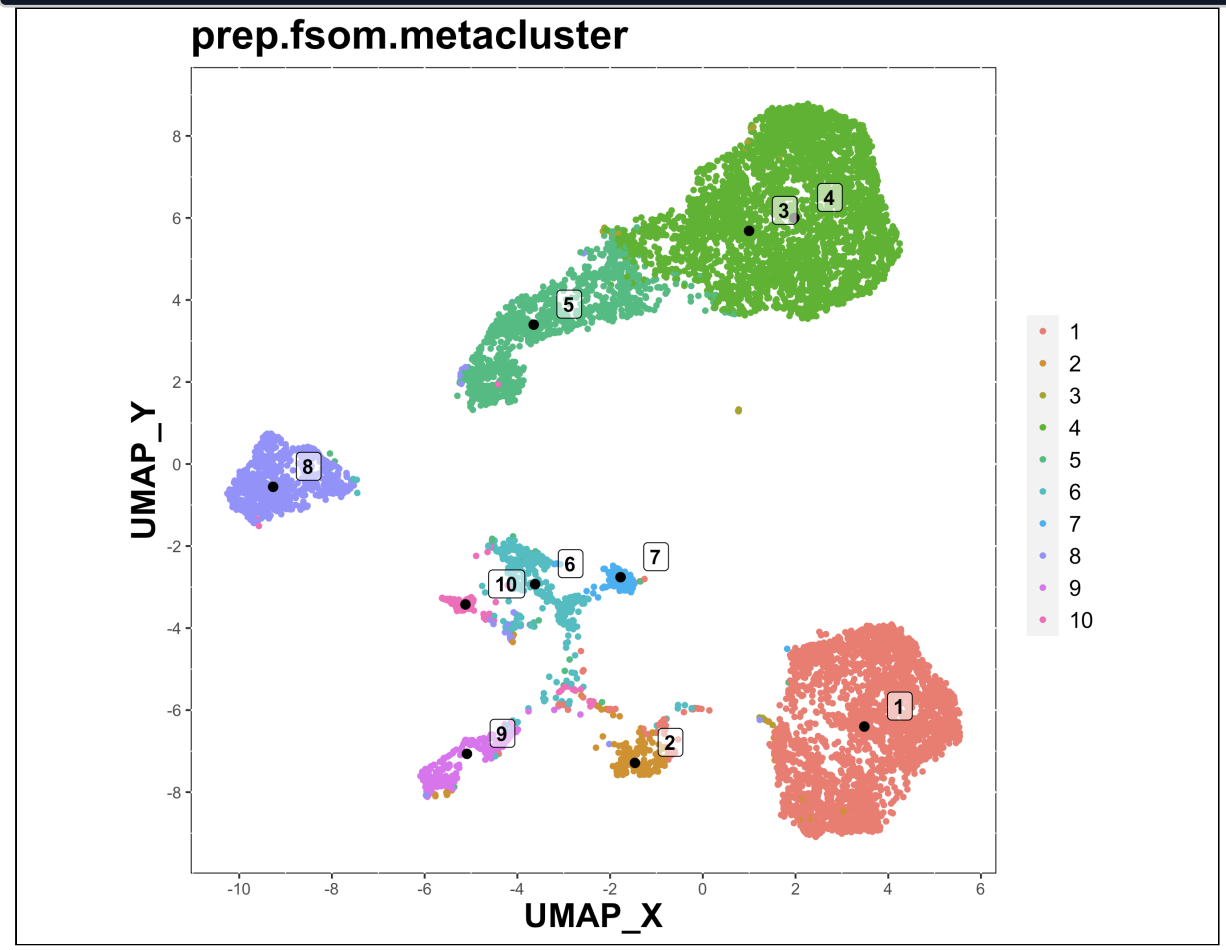
```
>         cytnrm$files
[1] "A" "B"
>         cytnrm$file.nums
[1] 1 2
```

Then, we need to examine the cellular expression across these cells.



Finally, we need to examine the metaclusters that have been generated, and confirm that they capture matching populations across batches (e.g. in this example, metacluster 1 captures B220+ B cells from both batches. If too many or too few metaclusters are generated, you can try running the **prep.cytonorm** function again, and specify a desired number of metaclusters using the **meta.k** argument.

```
make.colour.plot(cytnrm.sub, 'UMAP_X', 'UMAP_Y', 'File', 'factor')
```



If the metaclusters are suitable, we will proceed to actually performing alignment. Because the **train.cytonorm** and **run.cytonorm** functions involve the writing and reading of files, we will set the directory beforehand.

```
### Alignment

setwd(OutputDirectory)
setwd("Output 2 - alignment")
```

Firstly, we need to train the model that will perform quantile alignment for each marker on each metacluster.

```
cytnrm <- train.cytonorm(model = cytnrm, align.cols = cellular.cols)
```

Once complete, this model can be used to align the full dataset.

```
cell.dat <- run.cytonorm(dat = cell.dat, model = cytnrm, batch.col = batch.col)
```

To examine the results, we first need to specify the cellular names containing the aligned data (they will have '_aligned' appended to the end).

```
aligned.cols <- paste0(cellular.cols, '_aligned')
```

Then we can set a new output directory.

```
### Plotting reference data

setwd(OutputDirectory)
setwd("Output 2 - alignment")
dir.create("2 - ref aligned")
setwd("2 - ref aligned")
```

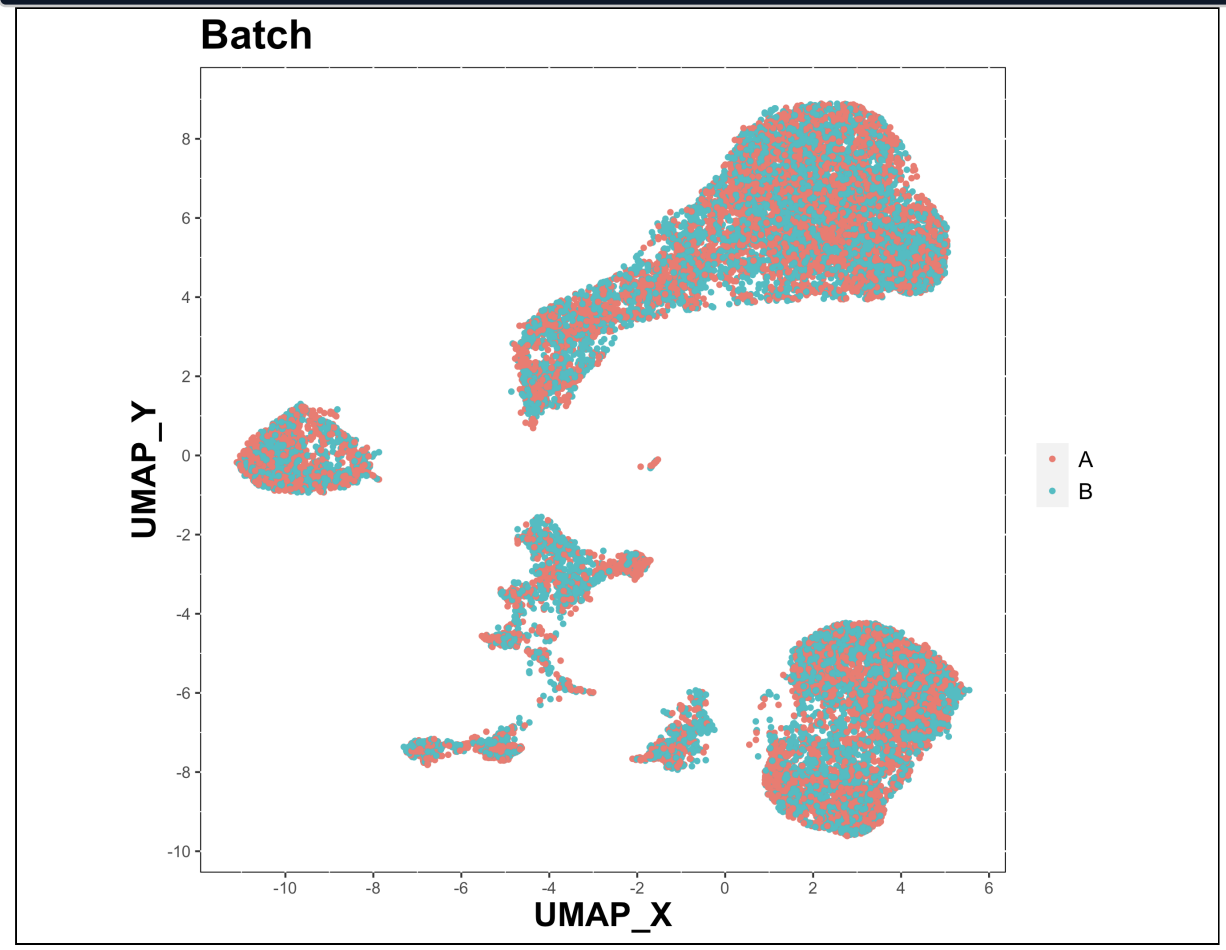
First, we will examine just the reference samples, to see if the alignment looks suitable. We will run UMAP using the new aligned data.

```
ref.sub <- do.filter(cell.dat, sample.col, refs)
ref.sub
```

```
ref.sub <- do.subsample(ref.sub, 20000)
ref.sub <- run.umap(ref.sub, use.cols = aligned.cols)
```

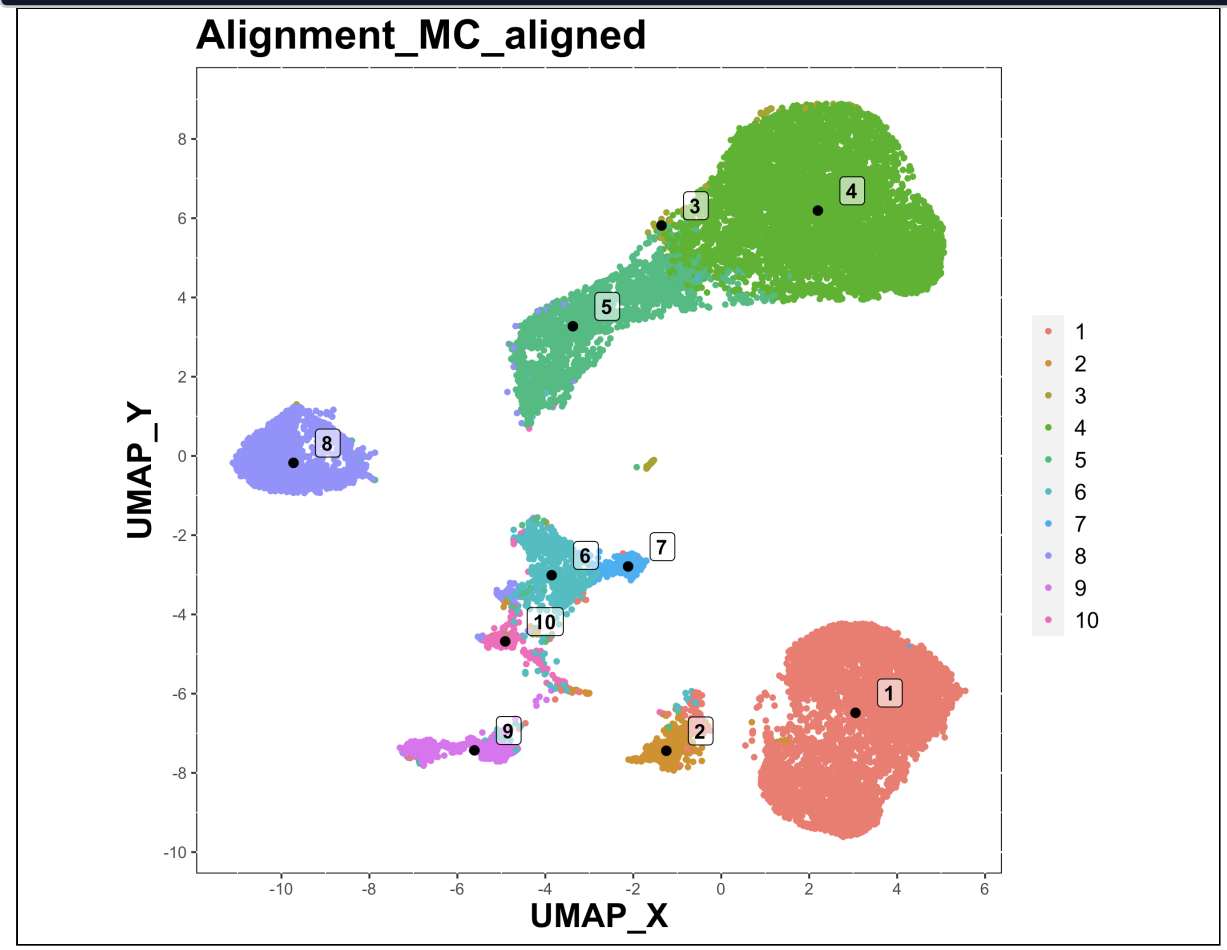
Plotting the data by batch reveals that the two batches are now well integrated with eachother.

```
make.colour.plot(ref.sub, 'UMAP_X', 'UMAP_Y', batch.col, 'factor')
```



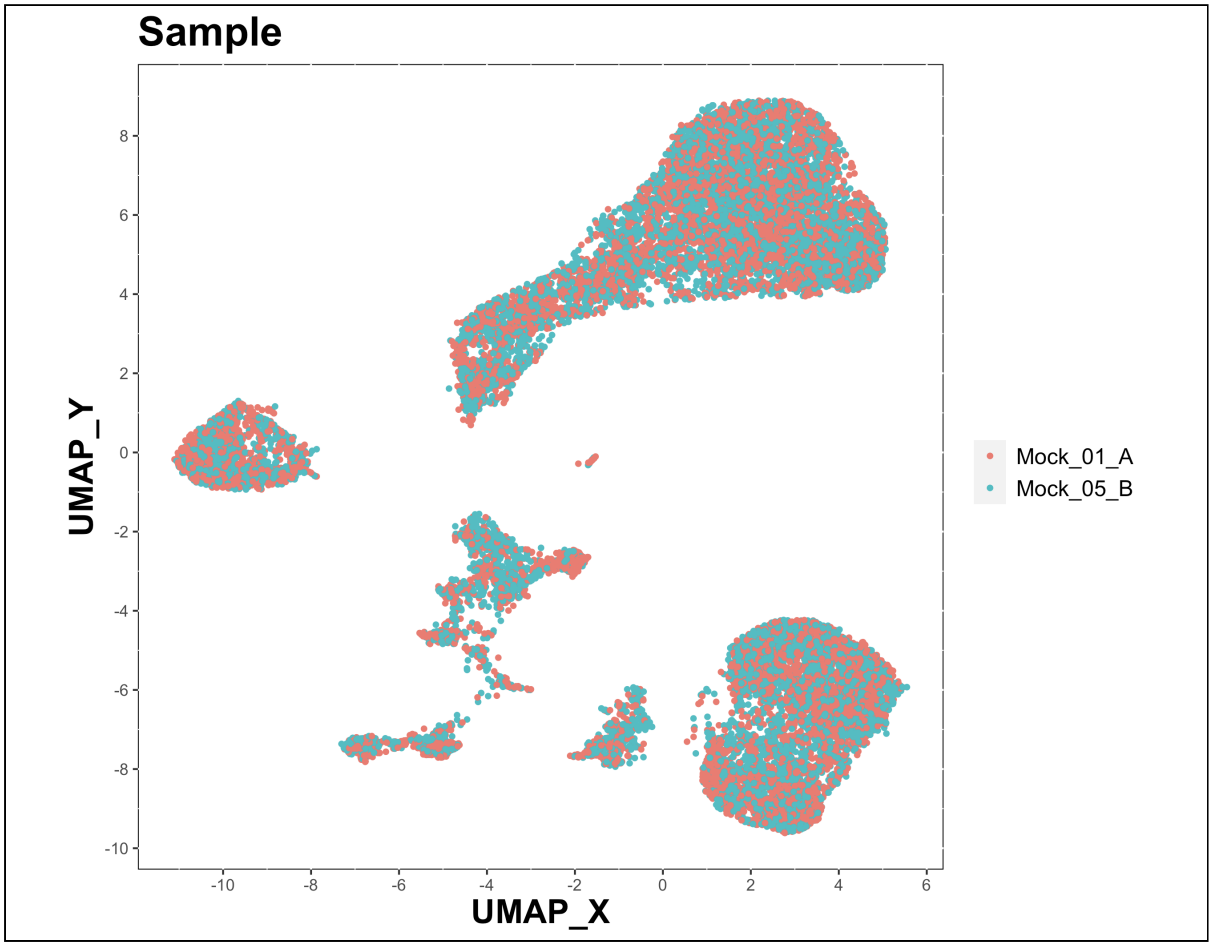
We can also plot the data by metacluster number, to confirm that they look suitable (these are the metaclusters created when we made the alignment model).

```
        make.colour.plot(ref.sub, 'UMAP_X', 'UMAP_Y', 'Alignment_MC_aligned', 'factor', add.label =
TRUE)
```



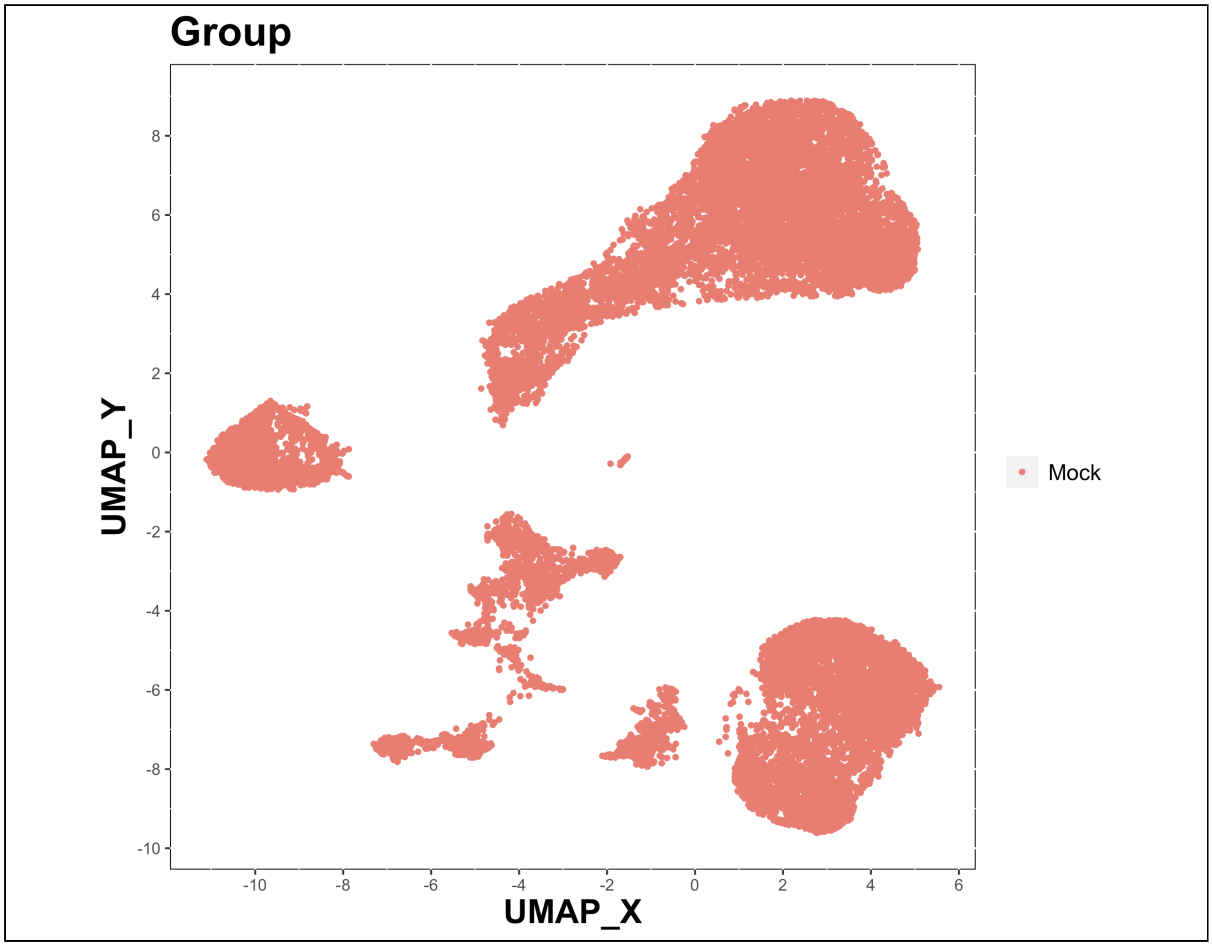
We can also colour the data by the exact samples (should be one per batch).

```
        make.colour.plot(ref.sub, 'UMAP_X', 'UMAP_Y', sample.col, 'factor')
```



As a sanity check, we can also check the experimental groups that these cells come from. The reference samples should ideally all be from a single experimental group, so if there are more than one group in the plot below, there may be a problem with the setup.

```
make.colour.plot(ref.sub, 'UMAP_X', 'UMAP_Y', group.col, 'factor')
```



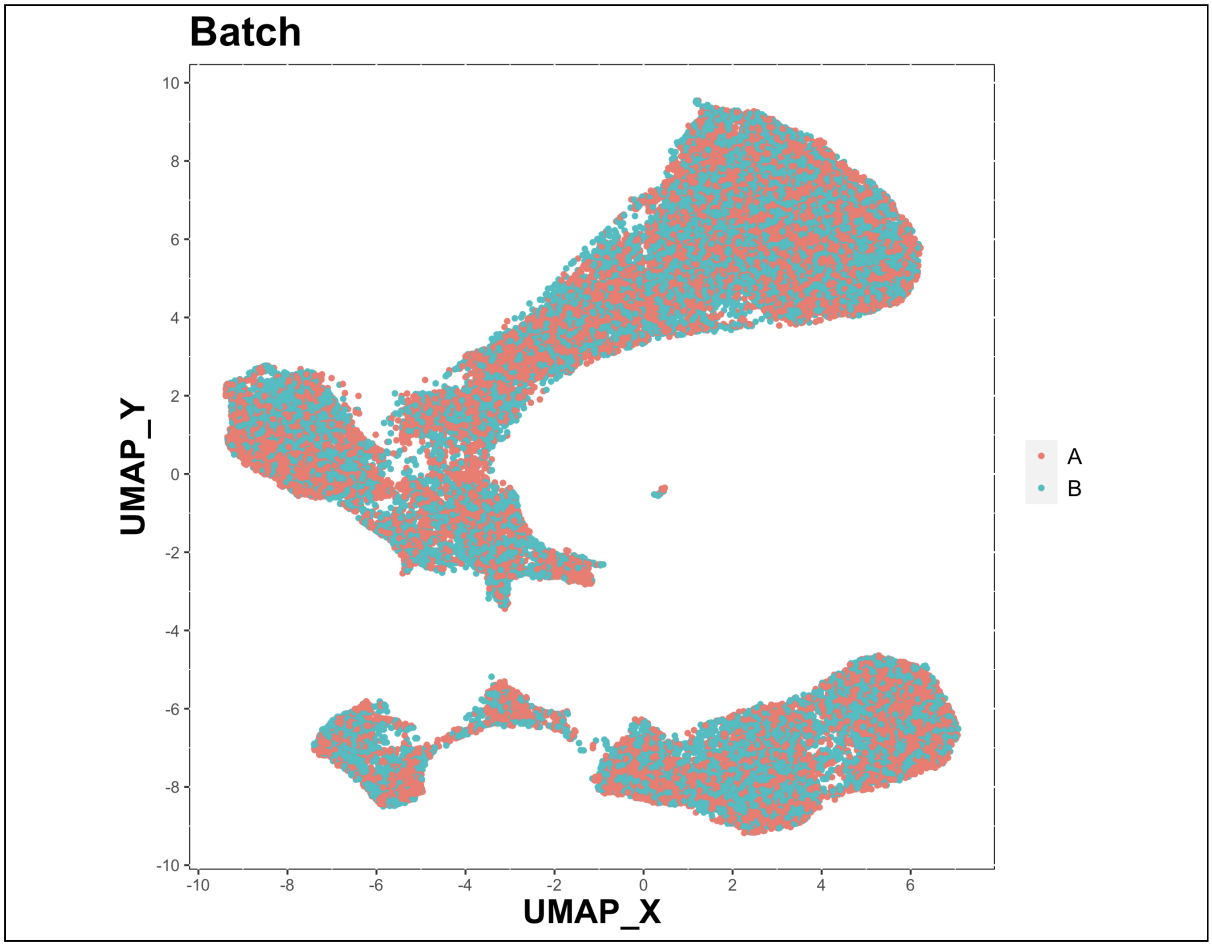
If the alignment looks suitable, we can then run UMAP and make plots using a subset of the whole dataset.

```
### Plotting all data

      setwd(OutputDirectory)
      setwd("Output 2 - alignment")
      dir.create("3 - all aligned")
      setwd("3 - all aligned")
```

```
      aligned.sub <- do.subsample(cell.dat, 50000)
      aligned.sub <- run.umap(aligned.sub, use.cols = aligned.cols)
```

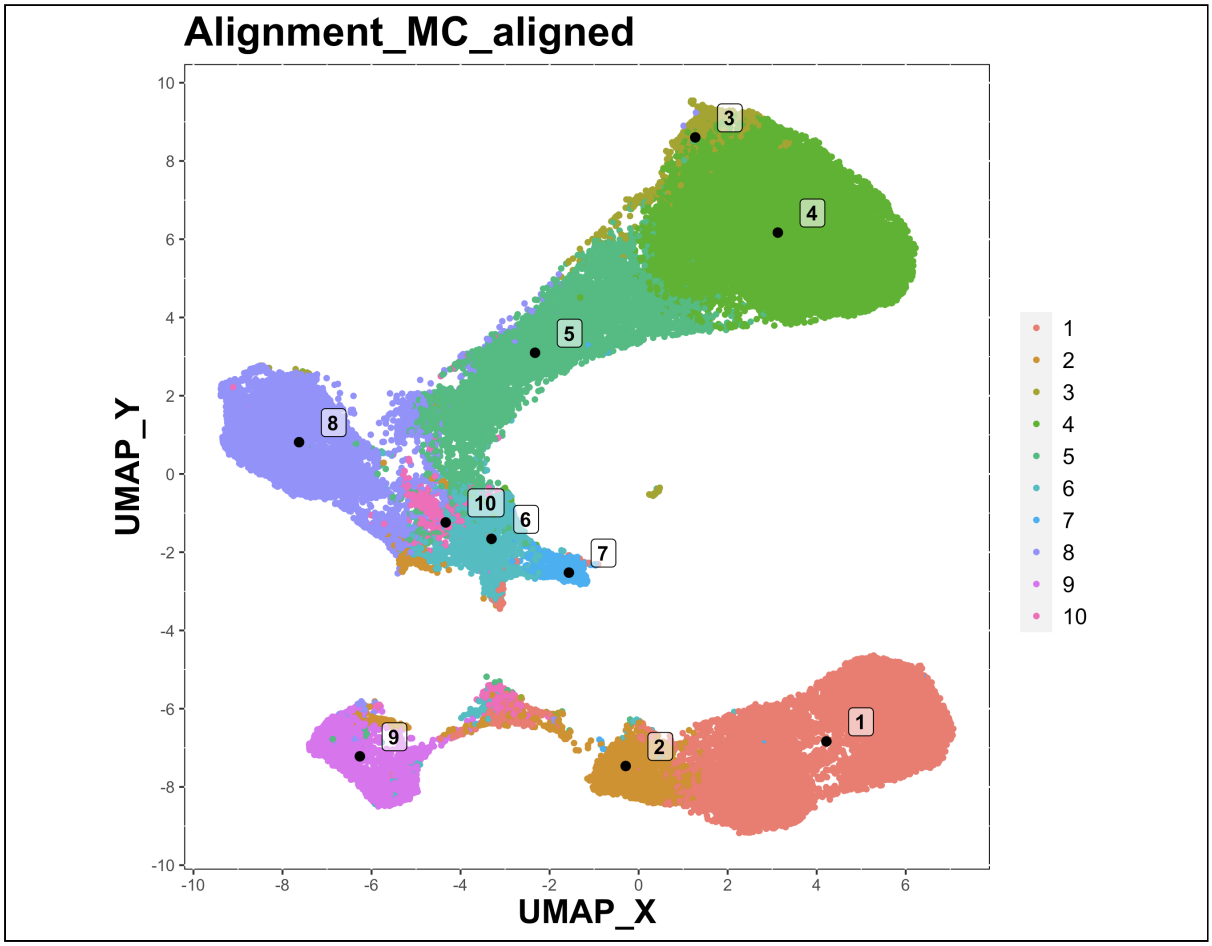
Firstly, we can examine the distribution of cells from each batch.

```
      make.colour.plot(aligned.sub, 'UMAP_X', 'UMAP_Y', batch.col, 'factor')
```



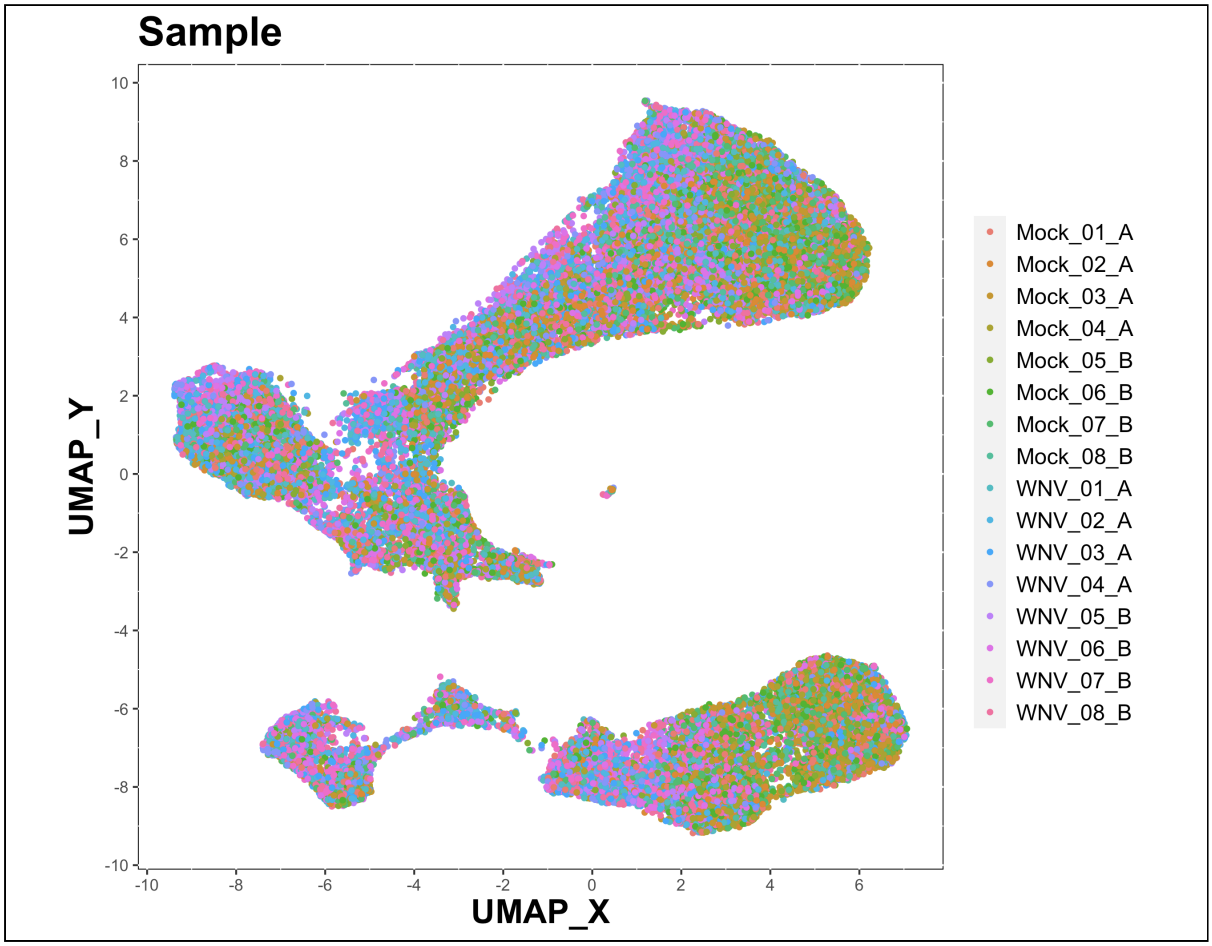
We can also check the metacluster assignments.

```
        make.colour.plot(aligned.sub, 'UMAP_X', 'UMAP_Y', 'Alignment_MC_aligned', 'factor', add.label =
TRUE)
```

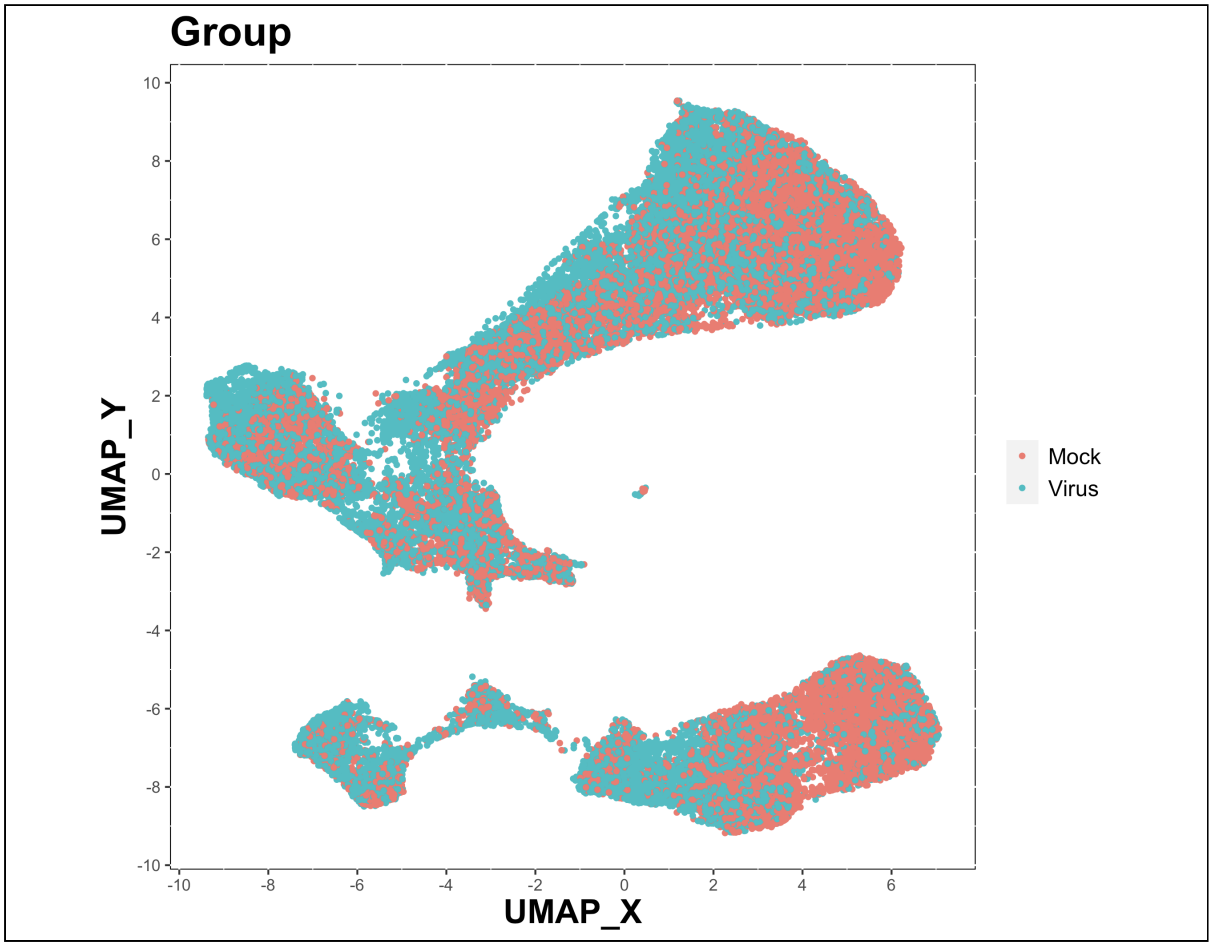


We can then check the distribution of cells from each sample, and each group. In this case the different distribution of cells in each experimental group reflects biologicall relevant changes, and not batch effects.

```
make.colour.plot(aligned.sub, 'UMAP_X', 'UMAP_Y', sample.col, 'factor')
```




```
make.colour.plot(aligned.sub, 'UMAP_X', 'UMAP_Y', group.col, 'factor')
```



For checking later, we can save a copy of the subsetted data to disk as well.

```
        fwrite(aligned.sub, 'aligned.sub.csv')
```

6. Clustering and dimensionality reduction

```
#########################################################################################################
###
#### 6. 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 **30**). 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 3 - clustering")
    setwd("Output 3 - clustering")
```

```
    ### Re-set cellular and clustering cols

        aligned.cellular.cols <- paste0(cellular.cols, '_aligned')
```

```
        aligned.cellular.cols

        aligned.cluster.cols <- paste0(cluster.cols, '_aligned')
        aligned.cluster.cols
```

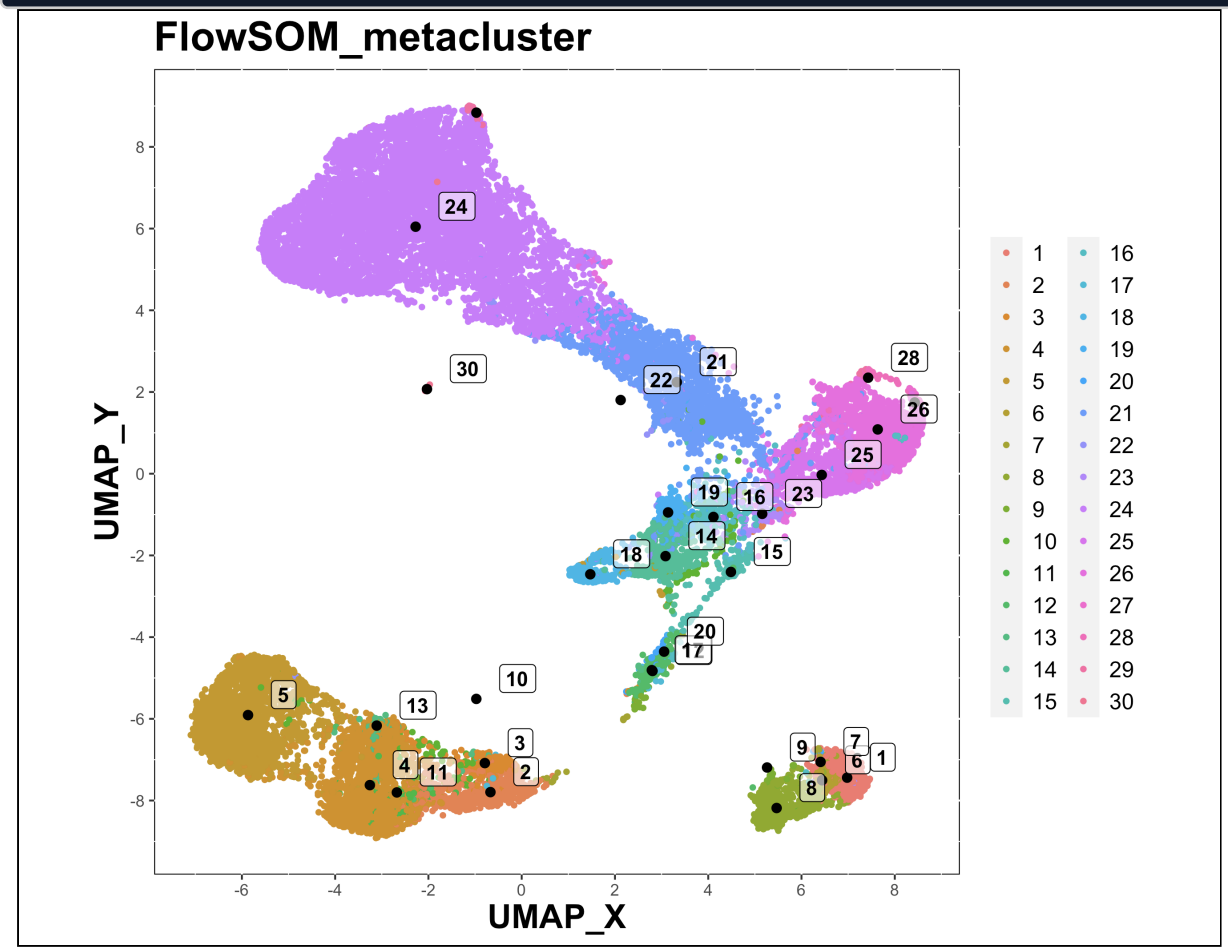
```
    ### Clustering

        cell.dat <- run.flowsom(cell.dat, aligned.cluster.cols, meta.k = 30)
        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.

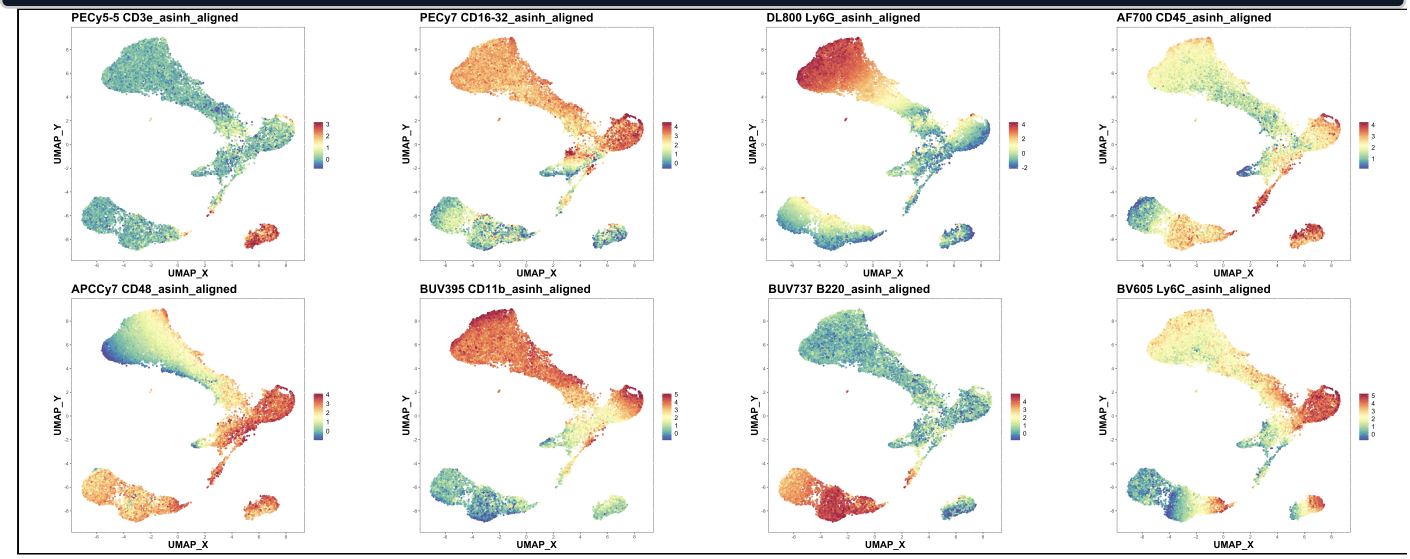
```
    ### Dimensionality reduction

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

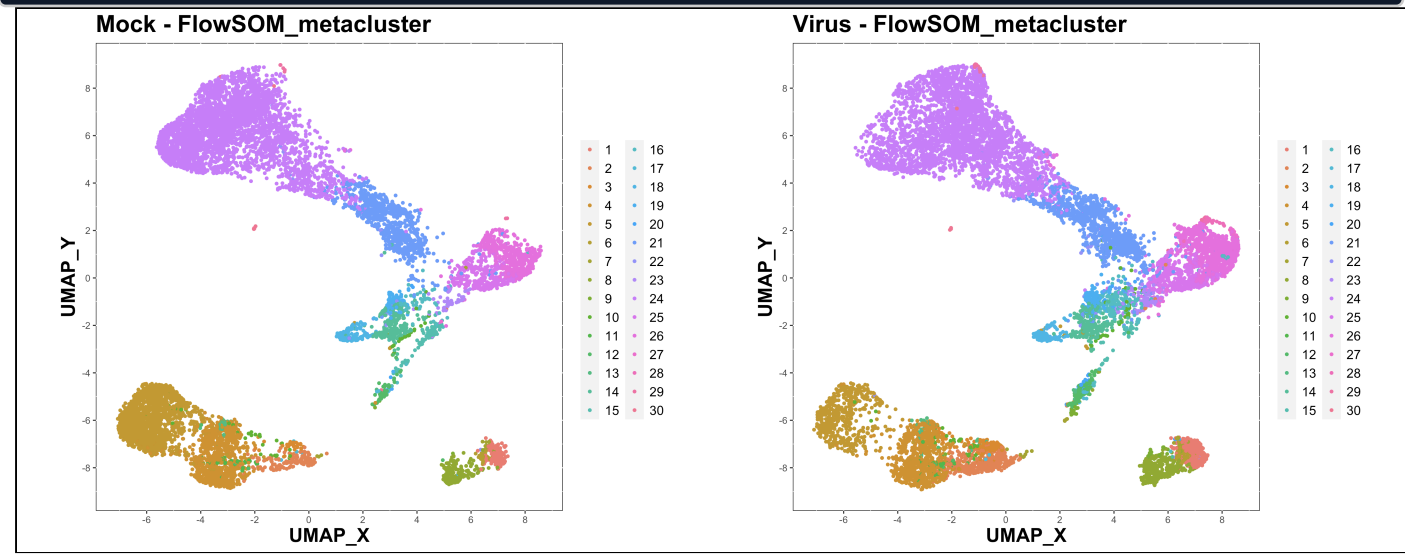


```
make.multi.plot(cell.sub, "UMAP_X", "UMAP_Y", aligned.cellular.cols)
```



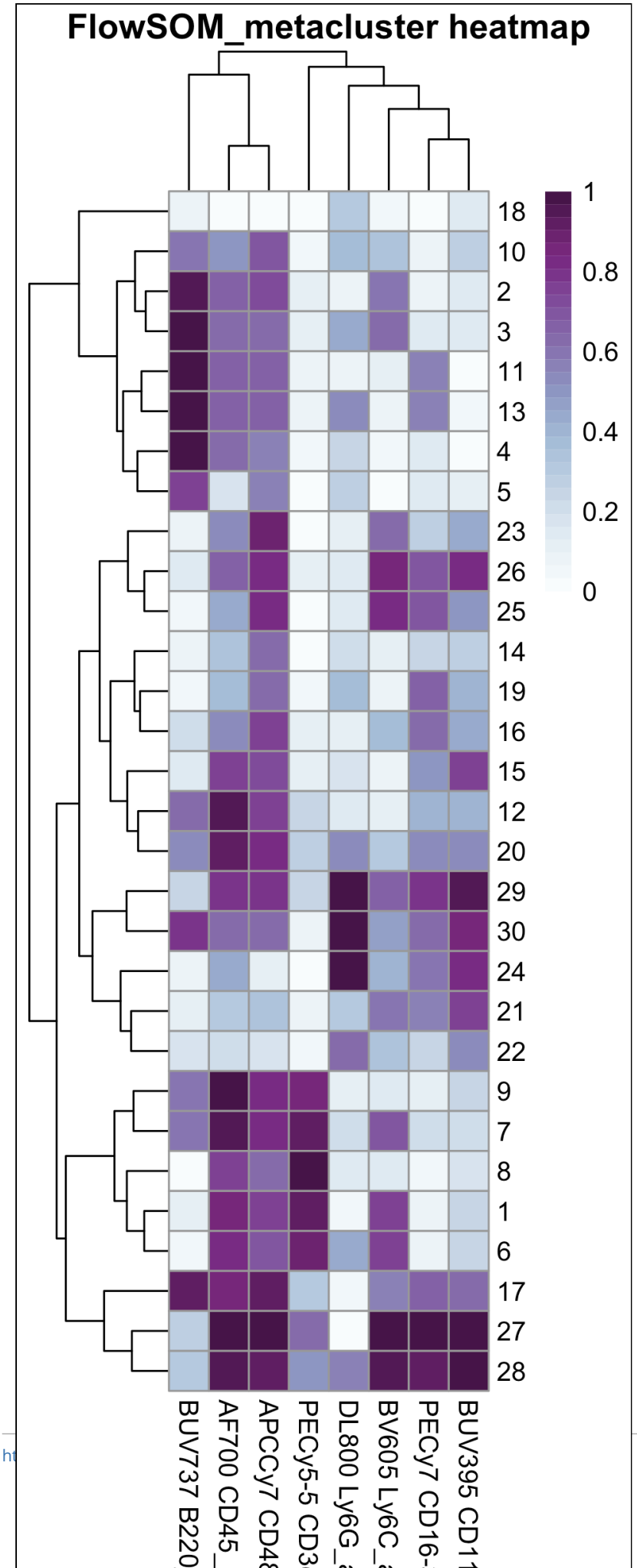
We can also generate some multi plots to compare between experimental groups or batches.

```
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, aligned.cellular.cols, by = "FlowSOM_metacluster")
    make.pheatmap(exp, "FlowSOM_metacluster", aligned.cellular.cols)
```


FlowSOM_metacluster heatmap
18
10
2
3
11
13
4
5
23
26
25
14
19
16
15
12
20
29
30
24
21
22
9
7
8
1
6
17
27
28
1
0.8
0.6
0.4
0.2
0
BUV737 B220
AF700 CD45_
APCCy7 CD4
PECy5-5 CD3
DL800 Ly6G_
BV605 Ly6C_
PECy7 CD16-
BUV395 CD11

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 4 - annotation")
    setwd("Output 4 - annotation")
```

```
    ### Annotate

        annots <- list("Mature neutrophils" = c(24,29),
                       "Immature neutrophils" = c(21,22),
                       "Monocytes" = c(28,26,25),
                       "T cells" = c(9,8,7,6,1),
                       "Mature B cells" = c(3,2,4,11,13),
                       "Immature B cells" = c(5)
        )
```

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              T cells
 2:      2       Mature B cells
 3:      3       Mature B cells
 4:      4       Mature B cells
 5:      5     Immature B cells
 6:      6              T cells
 7:      7              T cells
 8:      8              T cells
 9:      9              T cells
10:     11       Mature B cells
11:     13       Mature B cells
12:     21 Immature neutrophils
13:     22 Immature neutrophils
14:     24   Mature neutrophils
15:     25            Monocytes
16:     26            Monocytes
17:     28            Monocytes
18:     29   Mature neutrophils
```

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

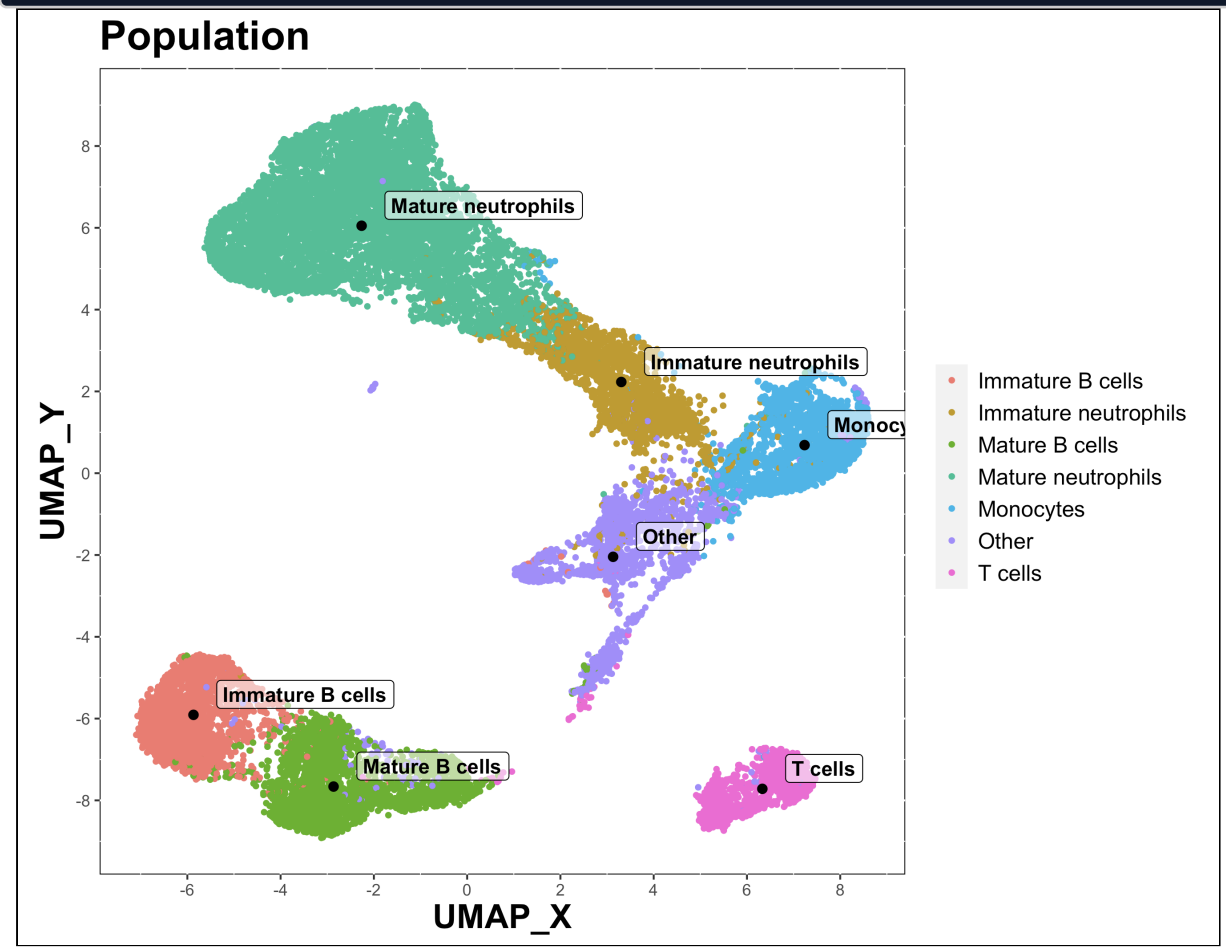
```
    ### Add annotations

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

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

```
    ### Fill in NAs

        cell.dat[['Population']][is.na(cell.dat[, 'Population'])] <- 'Other'
        cell.dat

        cell.sub[['Population']][is.na(cell.sub[, 'Population'])] <- 'Other'
        cell.sub
```

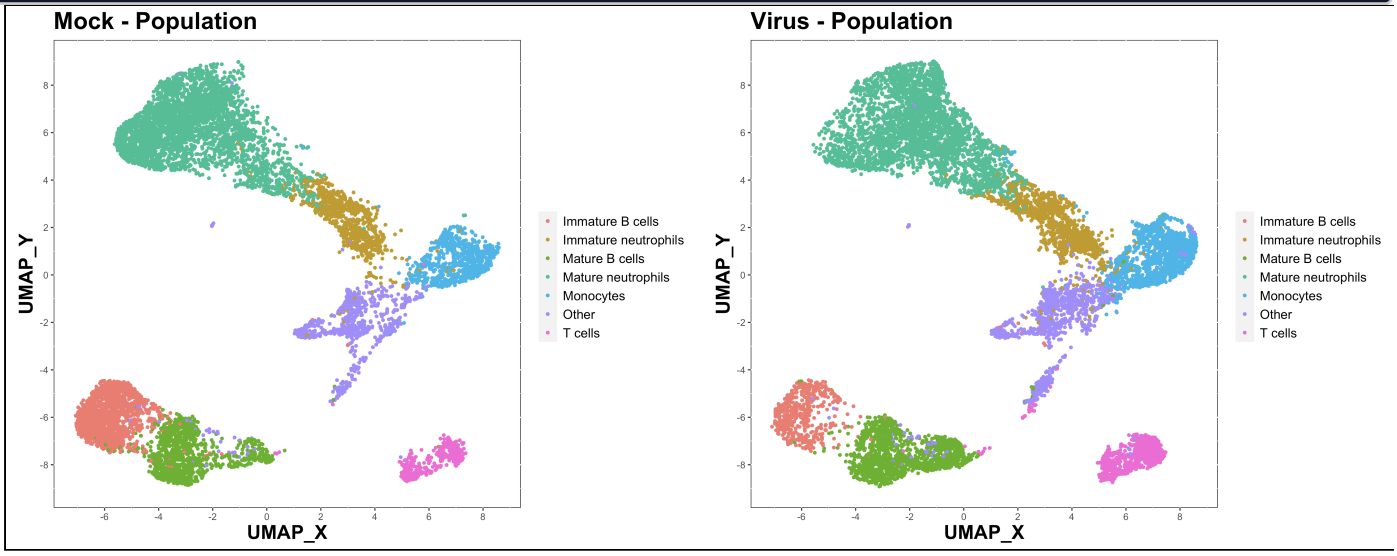
```
    ### Save data and plots

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

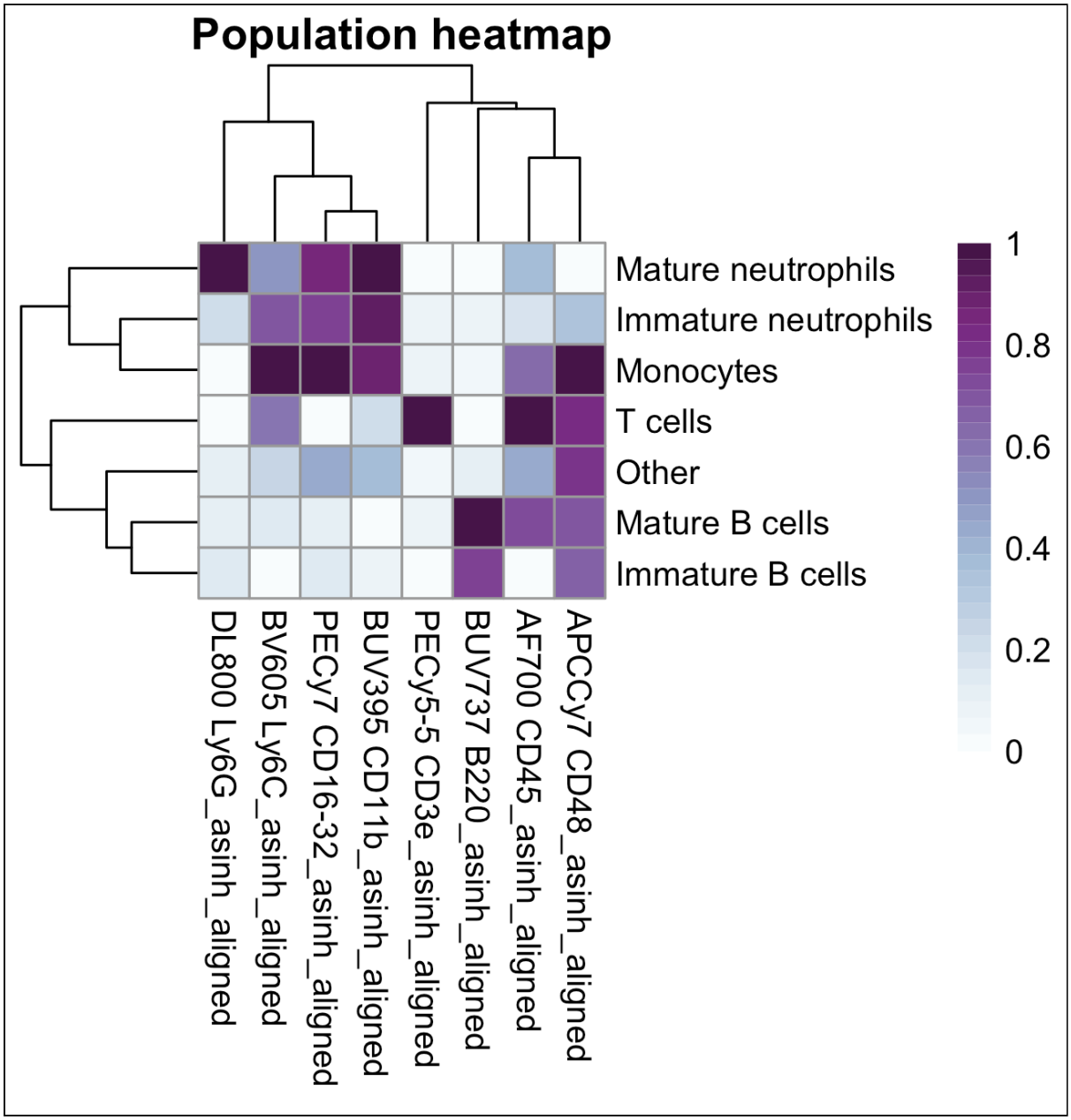


```
make.multi.plot(cell.sub, "UMAP_X", "UMAP_Y", "Population", group.col, col.type = 'factor')
```



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, aligned.cellular.cols, by = "Population")
    make.pheatmap(exp, "Population", aligned.cellular.cols)
```




```
### Write FCS files

    setwd(OutputDirectory)
    setwd("Output 4 - 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)
```

8. Summary data, graphs, statistics

```
#########################################################################################################
###
#### 8. Summary data and statistical analysis
#########################################################################################################
###
```

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 5 - summary data")
    setwd("Output 5 - summary data")
```

```
    ### Setup

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

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

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

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

```
    ### Select columns to measure MFI

        as.matrix(aligned.cellular.cols)
        dyn.cols <- aligned.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).

```
        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.

```
annot.cols <- c(group.col, batch.col)

plot.cols <- names(sum.dat)[c(4:21)]
plot.cols
```

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

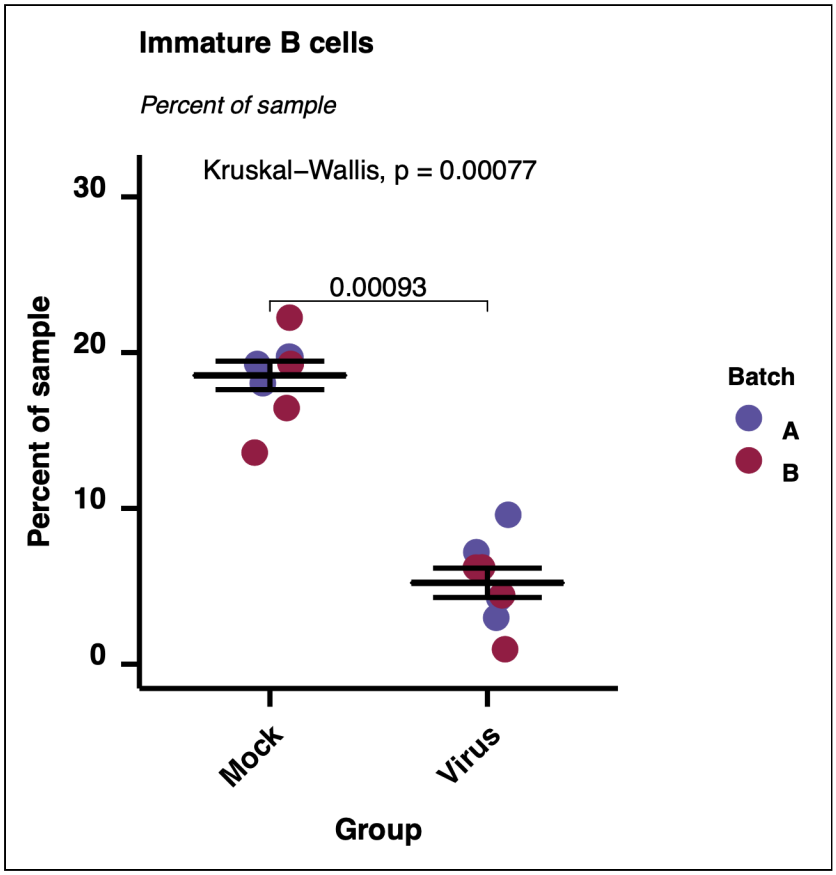
```
### Reorder summary data and SAVE

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

fwrite(sum.dat, 'sum.dat.csv')
```

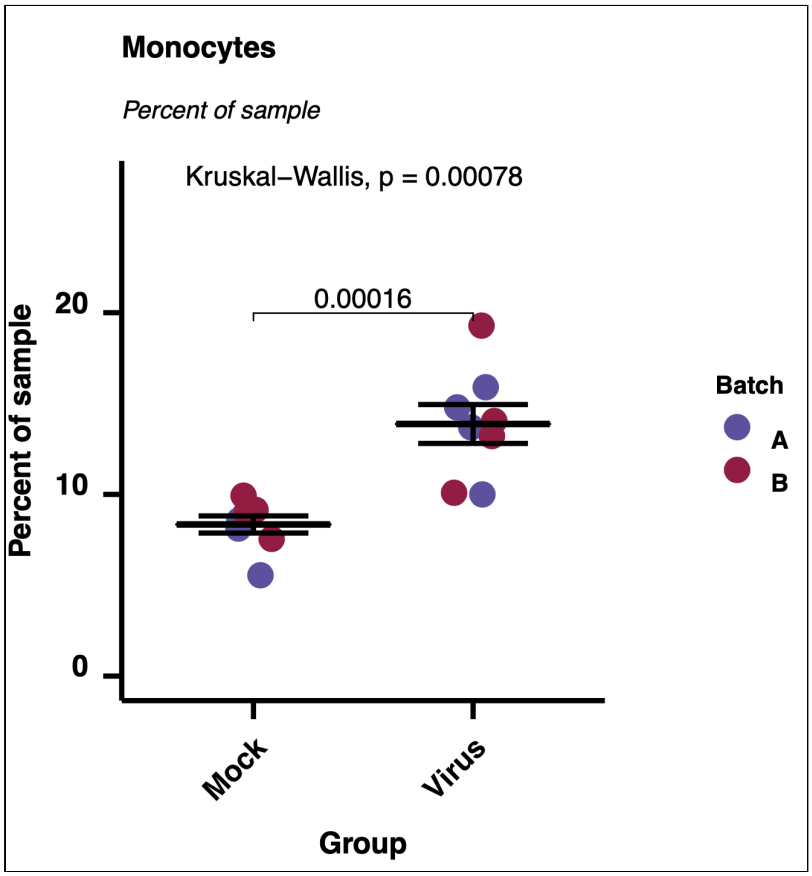
Violin/scatter plots

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,
                       violin = FALSE,
                       colour.by = batch.col,

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

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

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

    }
```

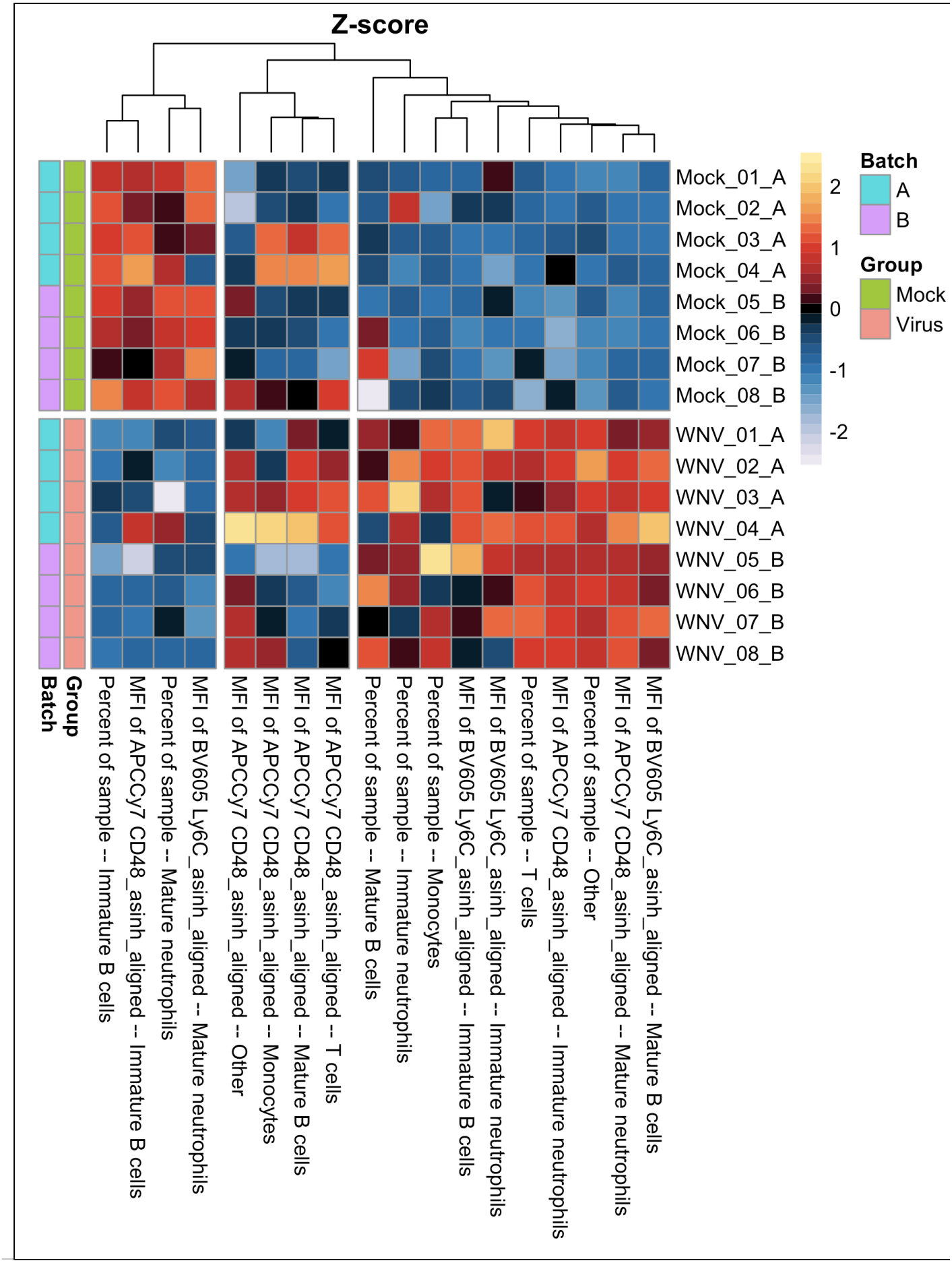

Immature B cells
Percent of sample
Kruskal–Wallis, p = 0.00077
0.00093
Percent of sample
30
20
10
0
Batch
A
B
Mock
Virus
Group


Monocytes
Percent of sample
Kruskal–Wallis, p = 0.00078
0.00016
Percent of sample
20
10
0
Batch
A
B
Mock
Virus
Group

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',
                  annot.cols = annot.cols,
                  dendrograms = 'column',
                  row.sep = t.first,
                  cutree_cols = 3)
```


Z-score
Batch
A
B
Group
Mock
Virus
2
1
0
-1
-2
Mock_01_A
Mock_02_A
Mock_03_A
Mock_04_A
Mock_05_B
Mock_06_B
Mock_07_B
Mock_08_B
WNV_01_A
WNV_02_A
WNV_03_A
WNV_04_A
WNV_05_B
WNV_06_B
WNV_07_B
WNV_08_B
Batch
Group
Percent of sample -- Immature B cells
MFI of APCCy7 CD48_asinh_aligned -- Immature B cells
Percent of sample -- Mature neutrophils
MFI of BV605 Ly6C_asinh_aligned -- Mature neutrophils
MFI of APCCy7 CD48_asinh_aligned -- Other
MFI of APCCy7 CD48_asinh_aligned -- Monocytes
MFI of APCCy7 CD48_asinh_aligned -- Mature B cells
MFI of APCCy7 CD48_asinh_aligned -- T cells
Percent of sample -- Mature B cells
Percent of sample -- Immature neutrophils
Percent of sample -- Monocytes
MFI of BV605 Ly6C_asinh_aligned -- Immature B cells
MFI of BV605 Ly6C_asinh_aligned -- Immature neutrophils
Percent of sample -- T cells
MFI of APCCy7 CD48_asinh_aligned -- Immature neutrophils
Percent of sample -- Other
MFI of APCCy7 CD48_asinh_aligned -- Mature neutrophils
MFI of BV605 Ly6C_asinh_aligned -- Mature B cells

9. Output session info

```
#########################################################################################################
###
#### 9. Output session info
#########################################################################################################
###
```

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