

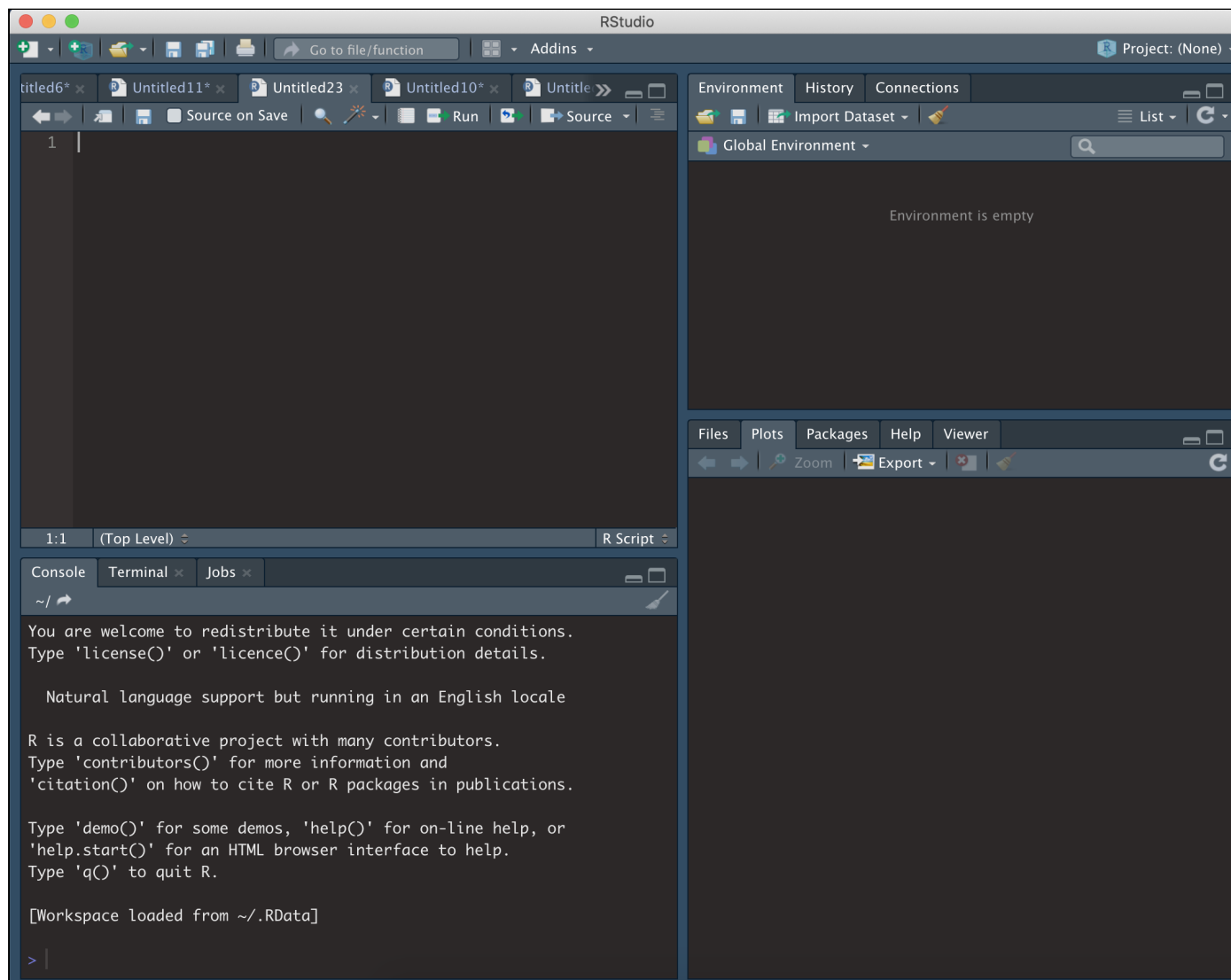
1. Setup tutorial script and dataset

✓ Tips

If you have not installed R or RStudio, or are not familiar with using R, please see the ['getting started'](#) section on the home page.

Open R Studio

To interact with the Spectre package in R, we will use [RStudio](#).



Load Spectre package

To use Spectre, we first need to load the 'Spectre' package, as well as other relevant packages. To do this, follow the instructions below.

- Copy the code on the right into your R script (MyScript.R). Make sure to SAVE.
- Run each line (one by one) by clicking on the line or highlighting the text, and press CMD + return (Mac) or CTRL + Enter (Windows).
- As above, nothing is returned if they are loaded successfully, or an error message is returned if they are not.
- If you have installed Spectre, but the package won't load, then you can visit our installation [troubleshooting page](#).

Input	
<pre>## Load the Spectre packages from library library('Spectre')</pre>	
If successful	If <u>un</u> successful
>	> Error in library("Spectre") : there is no package called 'Spectre'

-

Load other required packages

Rather than having to load each individual package required one-by-one (library('plyr'), library('data.table') etc), we have created two functions to simplify this process:

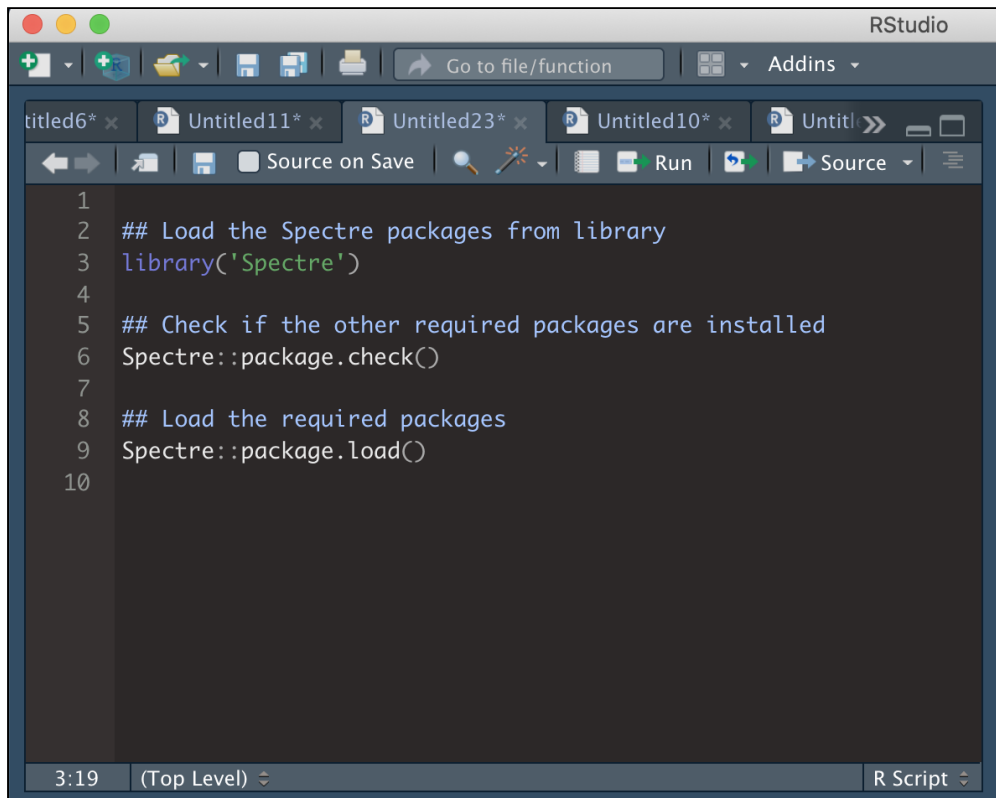
- 1. `packages.check()` will check if all the required packages are installed
- 2. `package.load()` will load all the required packages

Input
<pre>## Check if the other required packages are installed Spectre::package.check() ## Load the required packages Spectre::package.load()</pre>

As each package is loaded, you will see the following:

> Loading required package: PACKAGENAME

So far you should have the following code in your script:



The screenshot shows the RStudio application window. The title bar says 'RStudio'. Below it is a menu bar with 'Go to file/function' and 'Addins'. The next bar shows several open files: 'titled6*', 'Untitled11*', 'Untitled23*', 'Untitled10*', and 'Untitled1'. Below this is a toolbar with icons for navigation and execution, including 'Source on Save', 'Run', and 'Source'. The main editor area contains an R script with the following code:

```
1
2 ## Load the Spectre packages from library
3 library('Spectre')
4
5 ## Check if the other required packages are installed
6 Spectre::package.check()
7
8 ## Load the required packages
9 Spectre::package.load()
10
```

At the bottom of the window, the status bar shows '3:19 (Top Level)' and 'R Script'.

2. Set a working directory and create an output directory

Set a working directory

Normally this would be the location of the files you would like to analyse. For now, you can just this as your desktop or similar. If you aren't sure how to search for directories, please have a look at our [basic R tutorial](#).

Input

```
## Set working directory
setwd("/Users/thomasa/Desktop")

## Check that it has been set correctly
getwd()

## Save the working directory as an object called 'Primary Directory'
PrimaryDirectory <- getwd()
```

Create an output directory

Now we will create a directory where we can save the data and plots we will generate shortly.

Input

```
## Create an output directory
dir.create("Spectre-demo-output")

## Go to that directory and save it as an object called 'Output Directory'
```

```
setwd("Spectre-demo-output")
getwd()
OutputDirectory <- getwd()

## Finally, set the current working directory to 'PrimaryDirectory'
setwd(PrimaryDirectory)
```

3. Load data

Load demo data

Normally, we load some CSV or FCS files from the disk, or a server into R for analysis. In this tutorial, we will skip this step, and use an *included* demo dataset called '**demo.start**'. This is a dataset of 120,000 cells, from 12 bone marrow samples: 6x from mock-infected mice, and 6x West Nile virus (WNV)-infected mice. This amounts to 10,000 cells per sample. The data set is structured as a large data.frame (a table) where each column is a cellular marker (e.g. FITC-CD4, etc), and each row is a cell.

Assign the included **demo.start** dataset to a new object we will call **cell.dat**.

Input

```
cell.dat <- Spectre::demo.asinh
cell.dat <- do.subsample(dat = cell.dat, targets = 10000)
```

You can review the structure of **cell.dat** by using **str()**. You can see the cell.dat is both a 'data.table' and a 'data.frame'.

Input

```
str(cell.dat)
```

Output

```
Classes 'data.table' and 'data.frame': 10000 obs. of 19 variables:
 $ FileName : chr "CNS_Mock_05.csv" "CNS_Mock_04.csv" "CNS_WNV_D7_02.csv" "CNS_WNV_D7_01.csv" ...
 $ NK11 : num 106 133 -178 889 237 ...
 $ CD3 : num 235.1 16.5 1603.4 441.9 143.3 ...
 $ CD45 : num 13575 9534 131586 99332 7806 ...
 $ Ly6G : num -179 -673 -6522 -3229 -605 ...
```

You can review the dimensionality of **cell.dat** by using **dim()**. The first entry returned is the number of rows, and the second is the number of columns.

Input

```
dim(cell.dat)
```

Output

```
[1] 10000 19
```

You can review the first 6 rows (out of the 10,000 rows) of **cell.dat** by using **head()**. Each column is a marker or cellular feature, and each row is a cell.

Input

```
head(cell.dat)
```

```
> head(cell.dat)
  FileName      NK11      CD3      CD45      Ly6G      CD11b
1:  CNS_Mock_05.csv 106.015 235.1390 13574.70 -179.3180 31385.80
2:  CNS_Mock_04.csv 132.794  16.5279  9533.69 -673.4680 22061.20
3: CNS_WNV_D7_02.csv -178.423 1603.4400 131586.00 -6522.1100 1162.98
4: CNS_WNV_D7_01.csv 888.581 441.9300 99331.80 -3228.5300 40167.60
5:  CNS_Mock_04.csv 236.674 143.3300  7805.94 -605.1700 11658.20
6: CNS_WNV_D7_05.csv 735.811 180.2760 97843.00  97.0829 72048.40
      B220      CD8a      Ly6C      CD4 NK11_asinh CD3_asinh
1: -169.421 -394.6730 380.647 509.45900 0.1058174 0.23302438
2:  184.882 282.5120 608.245 1095.47000 0.1324068 0.01652715
3:  893.375 -40.3012 2726.790  -5.03532 -0.1774896 1.25080512
4: -623.009 -764.1700 64542.500 2345.11000 0.8003551 0.42867938
5:  658.958 107.7280  805.445 1937.42000 0.2345184 0.14284373
6: -1014.330 278.5310 52057.300 1490.30000 0.6817573 0.17931353
```

Set some preferences

Input

```
## Look at the names of the columns in the dataset, and take note of the number of each column
as.matrix(names(cell.dat))
```

Output

```
> as.matrix(names(cell.dat))
[,1]
[1,] "FileName"
[2,] "NK11"
[3,] "CD3"
[4,] "CD45"
[5,] "Ly6G"
[6,] "CD11b"
[7,] "B220"
[8,] "CD8a"
[9,] "Ly6C"
[10,] "CD4"
[11,] "NK11_asinh"
[12,] "CD3_asinh"
[13,] "CD45_asinh"
[14,] "Ly6G_asinh"
[15,] "CD11b_asinh"
[16,] "B220_asinh"
[17,] "CD8a_asinh"
[18,] "Ly6C_asinh"
```

```
[19,] "CD4_asinh"
```

Now we can choose the number of each column that we want to use for clustering, rather than having to write out each column name. To do this, we can put the number of the columns in a vector (i.e. `c(5,6,8)` for columns 5, 6, and 8) within the function below. You can replace these with the column numbers you would prefer to use (if you leave it as `c(5,6,8)`, then the columns used for clustering will be CD117, CD16/32, and CD115).

Input

```
## Save the column names that you wish to use for clustering as an object called 'cluster.cols'.
cluster.cols <- names(cell.dat)[c(11:19)]
```

We can check to make sure the names have been saved by running 'cluster.cols'.

Input

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

Output

```
> as.matrix(cluster.cols)
  [,1]
[1,] "NK11_asinh"
[2,] "CD3_asinh"
[3,] "CD45_asinh"
[4,] "Ly6G_asinh"
[5,] "CD11b_asinh"
[6,] "B220_asinh"
[7,] "CD8a_asinh"
[8,] "Ly6C_asinh"
[9,] "CD4_asinh"
```

4. Perform clustering and dimensionality reduction

Now we can perform our clustering and dimensionality reduction. First we are going to cluster the data using FlowSOM.

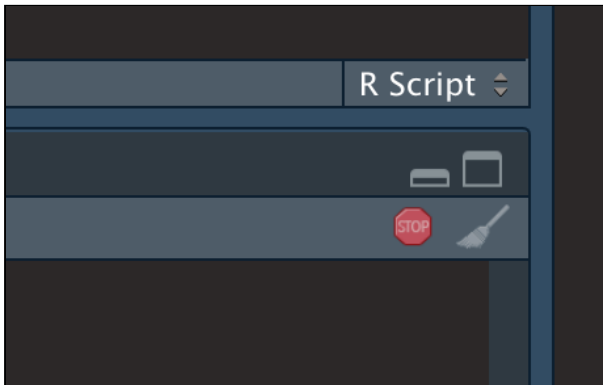
Run clustering (FlowSOM)

We can use the function 'run.flowsom' to run FlowSOM on our 'cell.dat' dataset. For more information on performing clustering in Spectre, see [this page](#). There are two key arguments we need to provide to the function. The first is 'dat', or the dataset to be used. The second is 'clust.cols', which is the columns to be used for clustering. In this case, we want to set **dat** to **cell.dat**, and **clust.cols** to **cluster.cols** (which we just created).

Input

```
## Run FlowSOM
cell.dat <- Spectre::run.flowsom(dat = cell.dat,
                                use.cols = cluster.cols)
```

As the clustering is running, you will see the following red button show up on your RStudio window. That means that RStudio is in the middle of processing something, and it won't respond to other commands while it is working.



While FlowSOM runs, you will progressively see the three following updates:

Creating SOM

Mapping data to SOM

Creating MST

Once FlowSOM has finished (and the red button has gone away) you can check the data to ensure the FlowSOM columns have been added correctly.

Input

```
# Check cell.dat to ensure FlowSOM data correctly attached -- by looking at the last two columns
head(cell.dat)
```

At the end of what's returned, you should see the FlowSOM metaclusters and clusters added to the dataset

Output

	CD4_asinh	FlowSOM_cluster	FlowSOM_metacluster
1:	0.489656167	98	3
2:	0.947296284	120	3
3:	-0.005035299	155	6
4:	1.588118198	1	1
5:	1.415293384	133	3
6:	1.189370553	10	1

Run UMAP

Now we can perform dimensionality reduction on our data for visualisation. For this we are going to use UMAP. For more information on dimensionality reduction and cytometry data, please see [this page](#). There are two key arguments we need to provide to the function. The first is 'dat', or the dataset to be used. The second is 'use.cols', which is the columns to be used for clustering. In this case, we want to set **dat** to **cell.dat**, and **use.cols** to **cluster.cols** (which we just created). UMAP by default doesn't provide progress updates.

Input

```
## Run UMAP
cell.dat <- Spectre::run.umap(dat = cell.dat, use.cols = cluster.cols)
```

It might take 1-2 minutes for UMAP to finish running.

-

Once UMAP has finished (and the red button has gone away) you can check the data to ensure the UMAP columns have been added correctly.

Input

```
## Check cell.dat to ensure the two new UMAP columns have been correctly attached.
head(cell.dat)
```

-

Output

	CD4_asinh	FlowSOM_cluster	FlowSOM_metacluster	UMAP_X	UMAP_Y
1:	0.489656167	98	3	2.663033	4.7166290
2:	0.947296284	120	3	2.255309	3.8056978
3:	-0.005035299	155	6	-4.758126	-0.6016916
4:	1.588118198	1	1	1.474979	-8.3206588
5:	1.415293384	133	3	1.202553	3.8960665
6:	1.189370553	10	1	3.525190	-5.4341879

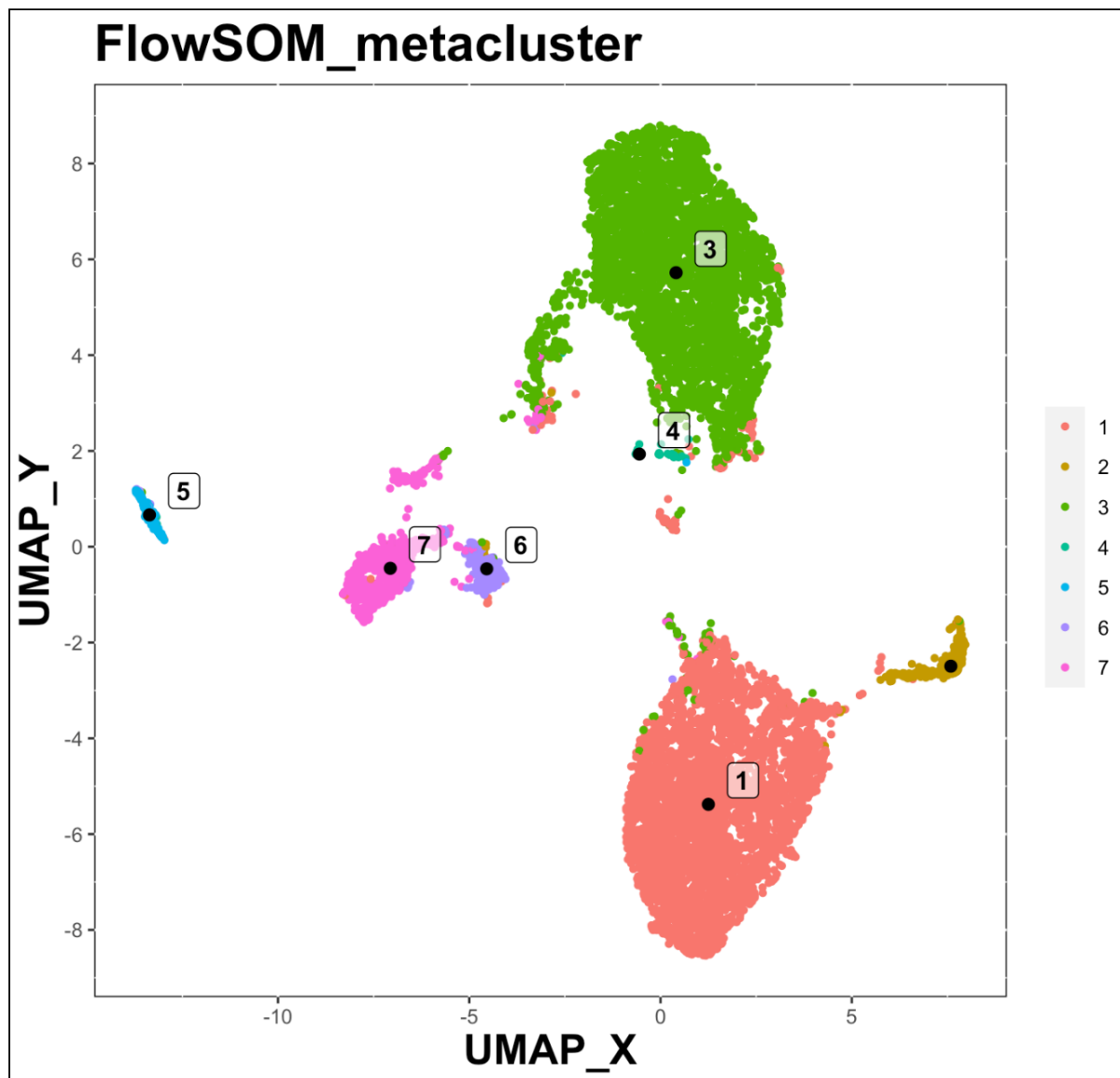
-

Quick visual check

Now that we have run FlowSOM and UMAP, we want to do a quick visual check to make sure everything looks correct. To do this we are going to create a 'factor' plot – a dot plot with our two UMAP columns as the X and Y axis, and the FlowSOM_metacluster as the colour. We are going to add the labels of each cluster to the plot, and we will tell the function *not* to save the image to disk. Running this command should generate a plot in the viewer window in RStudio.

Input

```
## Make a 'factor' plot
Spectre::make.colour.plot(dat = cell.dat,
  x.axis = "UMAP_X",
  y.axis = "UMAP_Y",
  col.axis = "FlowSOM_metacluster",
  col.type = 'factor',
  add.label = TRUE)
```

5. Save data to disk

Now that we have added the cluster and UMAP information to our data, we should save the files and capture our progress. First, let's set our working directory to 'OutputDirectory', so the data goes to the right place.

Input

```
## Set working directory to OutputDirectory
setwd(OutputDirectory)
getwd()
```

Input

```
## Save CSV files
Spectre::write.files(dat = cell.dat,
                     file.prefix = "Sample_CSV_file",
                     write.csv = TRUE,
                     write.fcs = FALSE)
```

-

To further explore this data in FlowJo, let's also save some FCS files.

Input

```
## Save FCS files
Spectre::write.files(dat = cell.dat,
                     file.prefix = "Sample_FCS_file",
                     write.csv = FALSE,
                     write.fcs = TRUE)
```

6. Make some plots

Now we should create some informative plots.

First we will make another factor plot of the FlowSOM metaclusters, but this time we will set 'save.to.disk' to TRUE. Once this has been run, check your working directory for the image.

Input

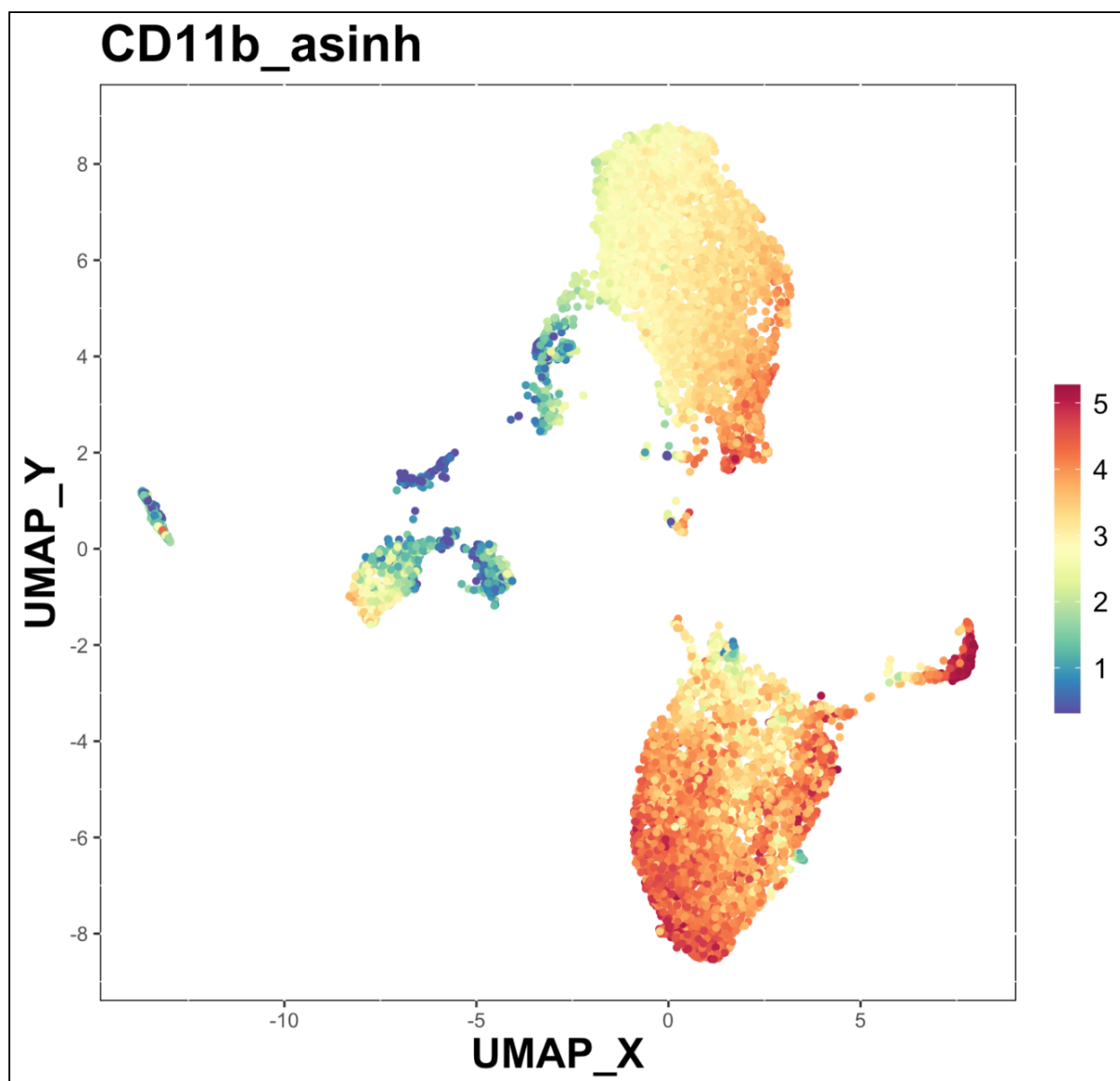
```
## Make a 'factor' plot coloured by cluster
Spectre::make.colour.plot(dat = cell.dat,
                          x.axis = "UMAP_X",
                          y.axis = "UMAP_Y",
                          col.axis = "FlowSOM_metacluster",
                          col.type = 'factor',
                          add.label = TRUE)
```

-

Next we'll make a colour plot showing the expression of a specific marker

Input

```
## Make a colour plot for the expression level of CD11b
Spectre::make.colour.plot(dat = cell.dat,
                          x.axis = "UMAP_X",
                          y.axis = "UMAP_Y",
                          col.axis = "CD11b_asinh",
                          add.label = TRUE)
```



7. Annotate clusters, generate summary data, create heatmaps and graphs (with stats)

From here, we may decide to annotate our clusters (e.g. clusters 1 and 5 become "B cells", etc), create some summary statistics (proportion of each population, total cells of each population per sample, MFI of each marker on each population, etc), and then great some graphs (with statistics) and heatmaps. More information on these processes can be found on the [Spectre Home Page](#).

Where to next?

Now that you've completed the Spectre tutorial, have a look at our workflow types on the [Spectre Home Page](#).