Multiplexed droplet scRNA-seq

Suhas Srinivasan

1. Initialization

This notebook and analysis packages are based on R version 4.3+.
The installation of packages has not been automated due to multiple dependencies from CRAN and Bioconductor.
It is suggested the packages be installed with supervision to fulfil all dependencies.
Additionally, the `celldex` package on first-use will prompt for a cache directory.

NOTE: Running this entire notebook can take 30 to 40 minutes to complete, i.e., Section 5 is most time consuming.

```
#Bioconductor packages
library(SingleCellExperiment)
library(celldex)
library(SingleR)

# CRAN packages
library(stringr)
library(Seurat)
library(SeuratObject)
```

2. Data download

If running on macOS or Windows, the following OS functions are needed: `wget` and `tar`
All the data will be downloaded in the current working directory.

```
system('wget -q https://ftp.ncbi.nlm.nih.gov/geo/series/GSE96nnn/GSE96583/suppl/GSE96583_RAW.tar')
system('wget -q https://ftp.ncbi.nlm.nih.gov/geo/series/GSE96nnn/GSE96583/suppl/GSE96583_batch2.genes.tsv.gz')
system('wget -q https://raw.githubusercontent.com/yelabucsf/demuxlet_paper_code/master/fig3/ye1.ctrl.8.10.sm.best')
system('wget -q https://raw.githubusercontent.com/yelabucsf/demuxlet_paper_code/master/fig3/ye2.stim.8.10.sm.best')
system('tar -xf GSE96583_RAW.tar')
system('rm GSE96583_RAW.tar GSM2560245* GSM2560246* GSM2560247*')
```

3. Data ingestion and filtering

```
# Load data from the extracted sparse matrix files
ctrl_cells = CreateSeuratObject(
              ReadMtx(
                mtx = 'GSM2560248_2.1.mtx.gz', features = 'GSE96583_batch2.genes.tsv.g
z',
                cells = 'GSM2560248_barcodes.tsv.gz', strip.suffix = T
              ),
              project = 'Ctrl'
            )

stim_cells = CreateSeuratObject(
              ReadMtx(
                mtx = 'GSM2560249_2.2.mtx.gz', features = 'GSE96583_batch2.genes.tsv.g
z',
                cells = 'GSM2560249_barcodes.tsv.gz', strip.suffix = T
              ),
              project = 'Stim'
            )

# Load output of demuxlet tool with droplet annotation
ctrl_dmx = read.table('./ye1.ctrl.8.10.sm.best', sep = '\t', header = T)
stim_dmx = read.table('./ye2.stim.8.10.sm.best', sep = '\t', header = T)

# Select only singlets and filter out doublets and ambiguous droplets
ctrl_keep = ctrl_dmx[str_detect(ctrl_dmx$BEST, 'SNG'), ]
stim_keep = stim_dmx[str_detect(stim_dmx$BEST, 'SNG'), ]

# Check filtered ratio
nrow(ctrl_keep)/nrow(ctrl_dmx)
```

```
## [1] 0.8913058
```

```
nrow(stim_keep)/nrow(stim_dmx)
```

```
## [1] 0.8868891
```

The proportion of doublets (~0.11) matches what is noted in the article.

```
ctrl_fltr = ctrl_cells[ , colnames(ctrl_cells) %in% str_remove(ctrl_keep$BARCODE, '-1')]
ctrl_fltr
```

```
## An object of class Seurat
## 35635 features across 13030 samples within 1 assay
## Active assay: RNA (35635 features, 0 variable features)
```

```
stim_fltr = stim_cells[ , colnames(stim_cells) %in% str_remove(stim_keep$BARCODE, '-1')]
stim_fltr
```

```
## An object of class Seurat
## 35635 features across 12812 samples within 1 assay
## Active assay: RNA (35635 features, 0 variable features)
```

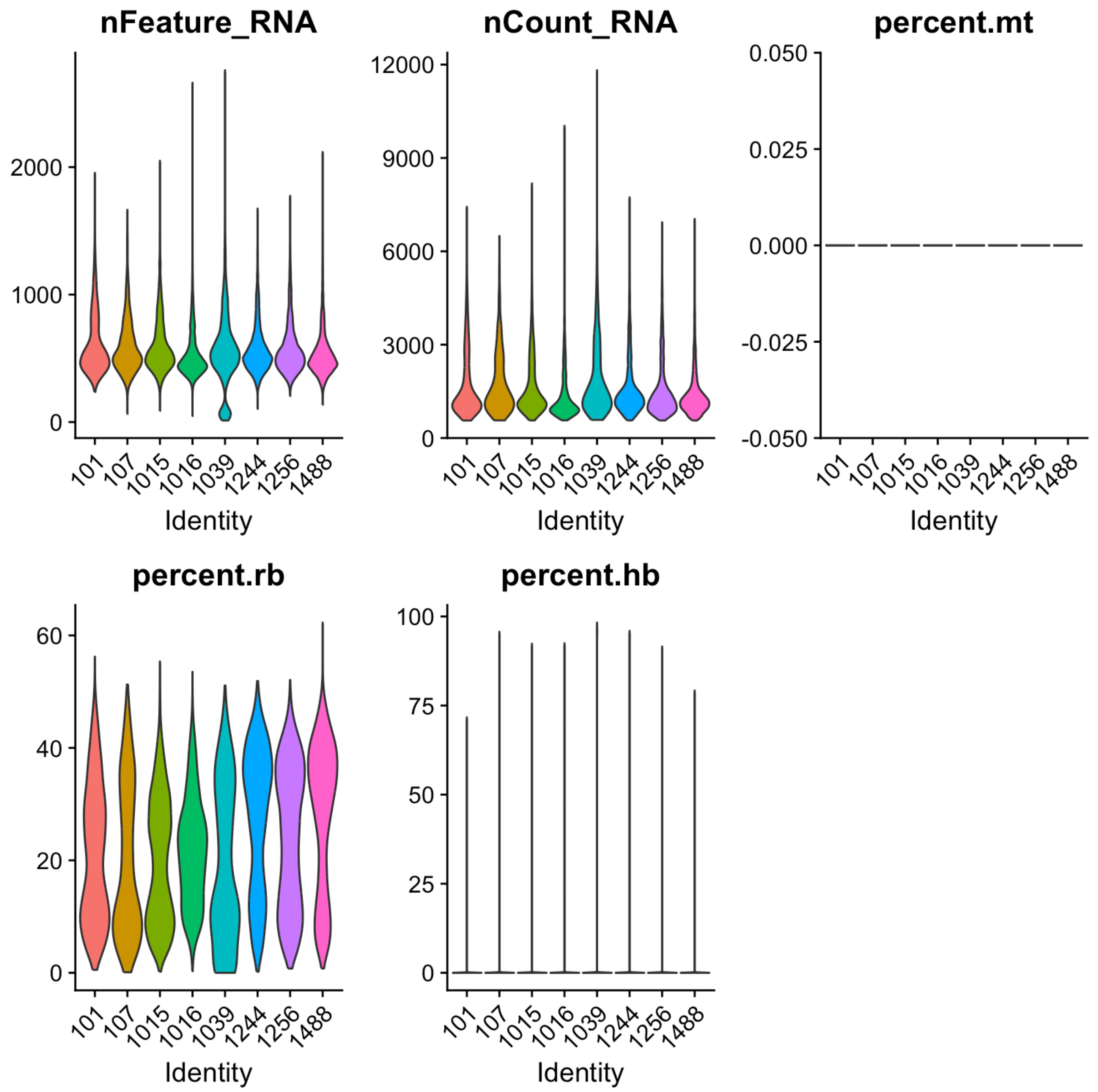
```
# Add Sample ID
ctrl_fltr@meta.data$sample = ctrl_keep$SNG.1ST
stim_fltr@meta.data$sample = stim_keep$SNG.1ST
```

4. Combining cell matrices across conditions

4.1. Quality control

Beginning with a combined matrix for downstream analysis.

```
# Add experiment prefix to Cell IDs while merging
com_cells = merge(x = ctrl_fltr, y = stim_fltr, add.cell.ids = c('Ctrl-', 'Stim-'))

# Perform typical QC for single-cell RNA-seq
# Calculate specific metrics since it is PBMC data
# Mitochondrial, Ribosomal and Haemoglobin genes
com_cells = PercentageFeatureSet(com_cells, '^MT-', col.name = 'percent.mt')
com_cells = PercentageFeatureSet(com_cells, '^RP[SL]', col.name = 'percent.rb')
com_cells = PercentageFeatureSet(com_cells, '^HB[^(P)]', col.name = 'percent.hb')

# Pre-filtering QC metrics
VlnPlot(com_cells, group.by = 'sample', pt.size = 0,
        features = c('nFeature_RNA', 'nCount_RNA', 'percent.mt', 'percent.rb', 'percent.
hb'), ncol = 3)
```



It is observed that the mitochondrial genes have no expression.

```
# Verifying MT genes
mito_genes = sort(rownames(com_cells)[grep('^MT-', rownames(com_cells))])
mito_genes
```

```
##  [1] "MT-ATP6" "MT-ATP8" "MT-CO1"  "MT-CO2"  "MT-CO3"  "MT-CYB"  "MT-ND1"
##  [8] "MT-ND2"  "MT-ND3"  "MT-ND4"  "MT-ND4L" "MT-ND5"  "MT-ND6"
```

The mitochondrial gene names are as expected and all 13 exist, but the counts are indeed 0. Rectifying this would require starting from FASTQ files and quantification to obtain mitochondrial gene expression.

Since this is a peer-reviewed study and authors have accounted for single-cell sequencing challenges continuing with typical QC.

```
dim(com_cells)
```

```
## [1] 35635 25842
```

```
table(com_cells$orig.ident)
```

```
##
##  Ctrl  Stim
## 13030 12812
```

```
table(com_cells$sample)
```

```
##
##  101  107 1015 1016 1039 1244 1256 1488
## 2101 1096 5235 3611 1155 3564 4253 4827
```

```
# Keep cells that have genes quantified between 200 and 2500
# And have ribosomal genes % > 5 and haemoglobin genes % < 2.5
sel_cells = WhichCells(com_cells, expression = nFeature_RNA > 200 & nFeature_RNA < 2500
&
                         percent.rb > 5 & percent.hb < 2.5)

# Keep genes that are expressed in at least 5 cells
sel_genes = rownames(com_cells)[rowSums(com_cells) > 5]
com_cells = subset(com_cells, cells = sel_cells, features = sel_genes)

dim(com_cells)
```

```
## [1] 14460 24362
```

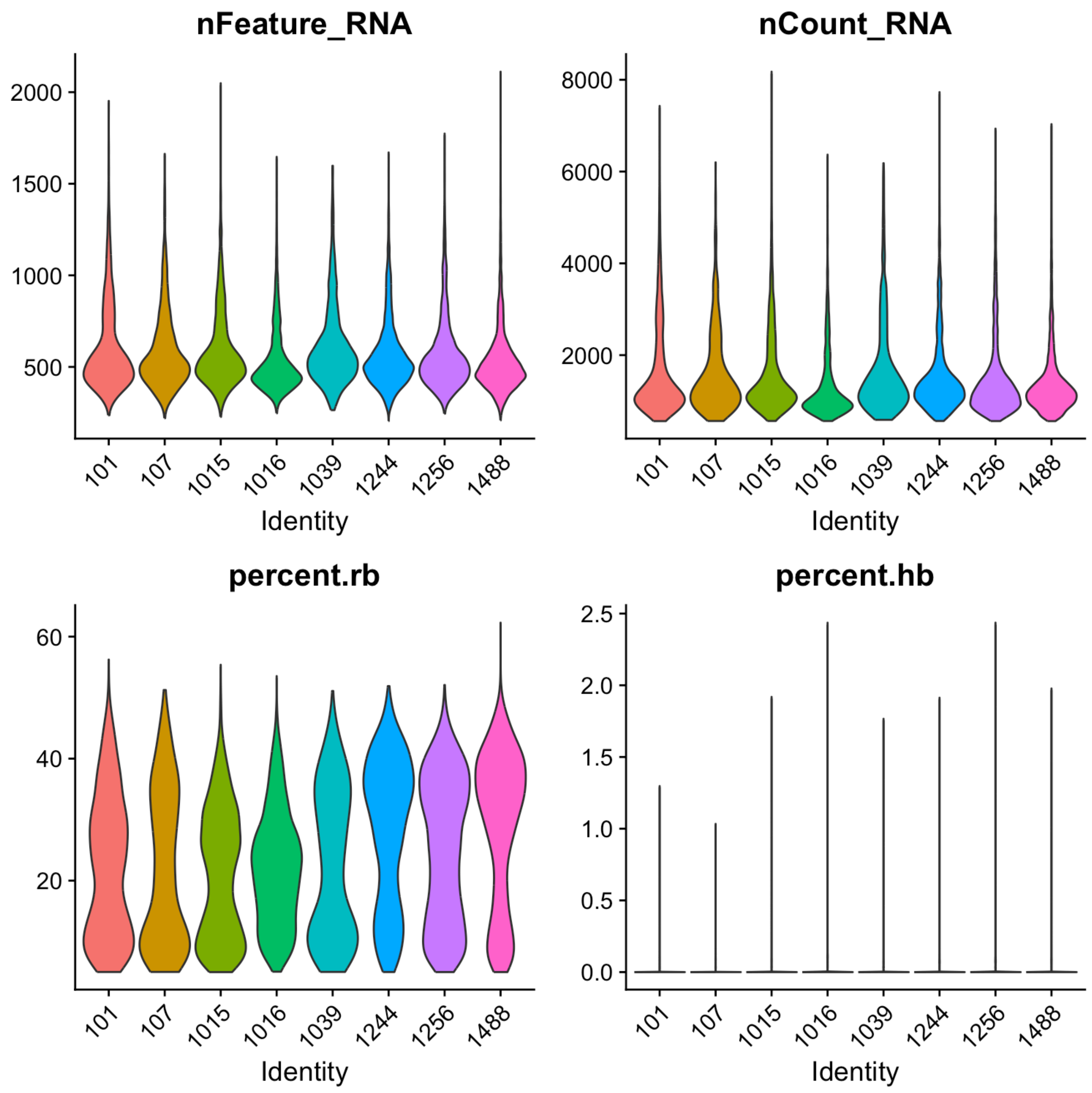
```
table(com_cells$orig.ident)
```

```
##
##  Ctrl  Stim
## 12828 11534
```

```
table(com_cells$sample)
```

```
##
##  101  107 1015 1016 1039 1244 1256 1488
## 1966  969 4878 3558  945 3420 4017 4609
```

```
# Post-filtering QC metrics
VlnPlot(com_cells, group.by = 'sample', pt.size = 0,
        features = c('nFeature_RNA', 'nCount_RNA', 'percent.rb', 'percent.hb'), ncol =
2)
```

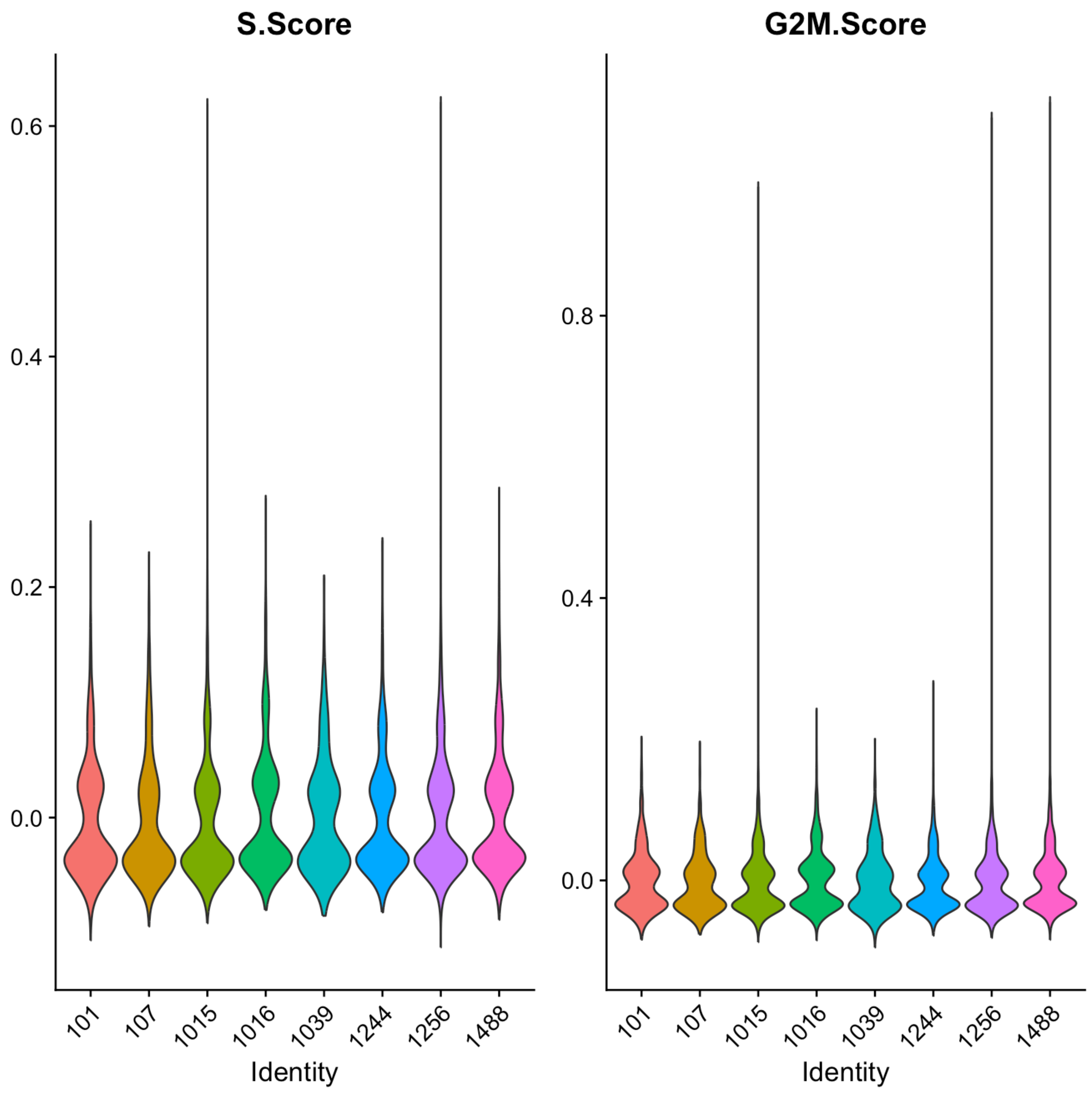


~94% of combined cell dataset still remains.

4.2. Observing cell cycling effects

Noise/heterogeneity introduced due to many cycling cells can cause issues for further analysis.

```
com_cells = NormalizeData(com_cells, verbose = F)

com_cells = CellCycleScoring(com_cells, g2m.features = cc.genes$g2m.genes, s.features =
cc.genes$s.genes)

VlnPlot(com_cells, group.by = 'sample', pt.size = 0, features = c('S.Score', 'G2M.Scor
e'))
```

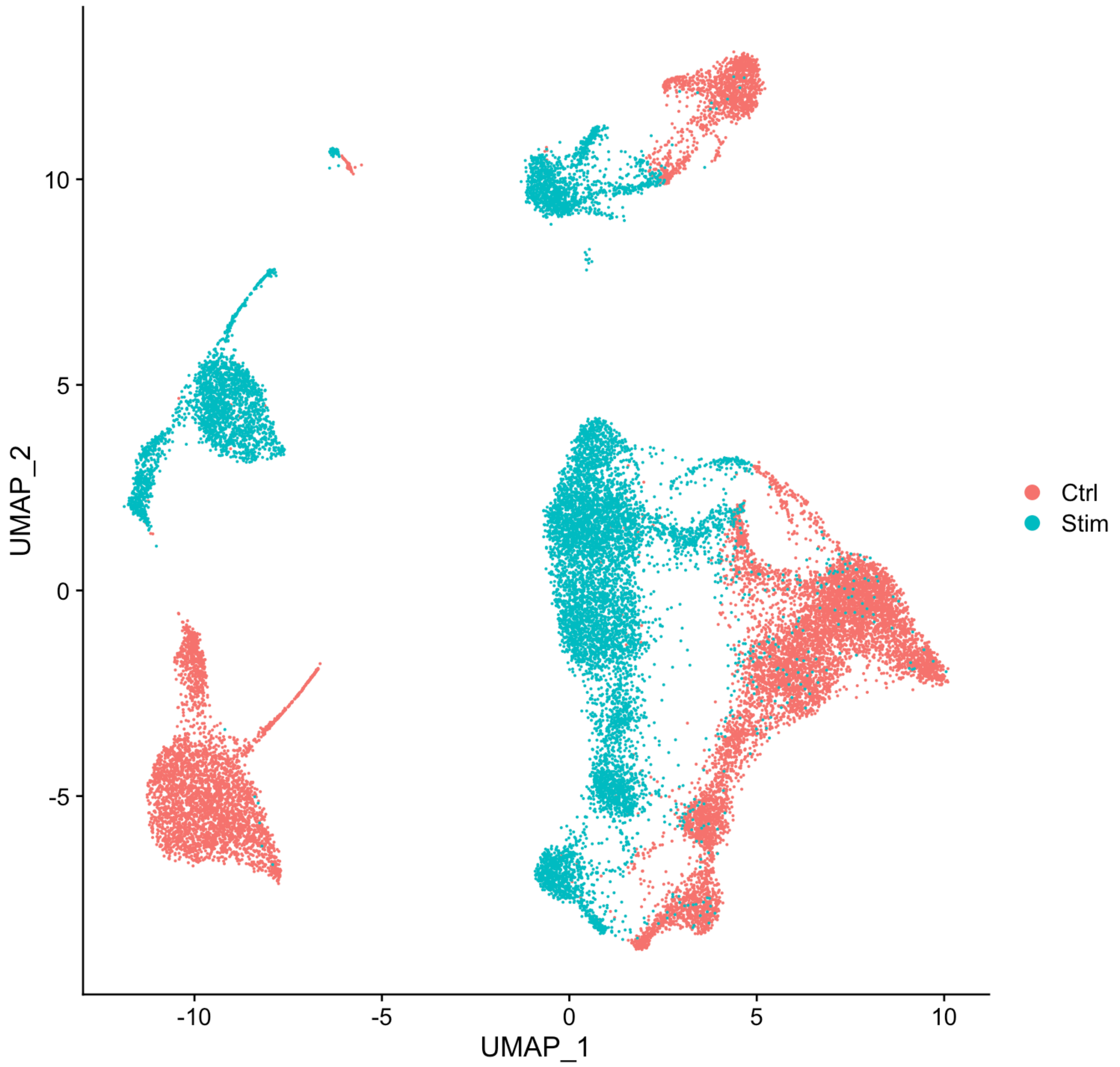


Cycling cells make up a very small population and do not need filtering.

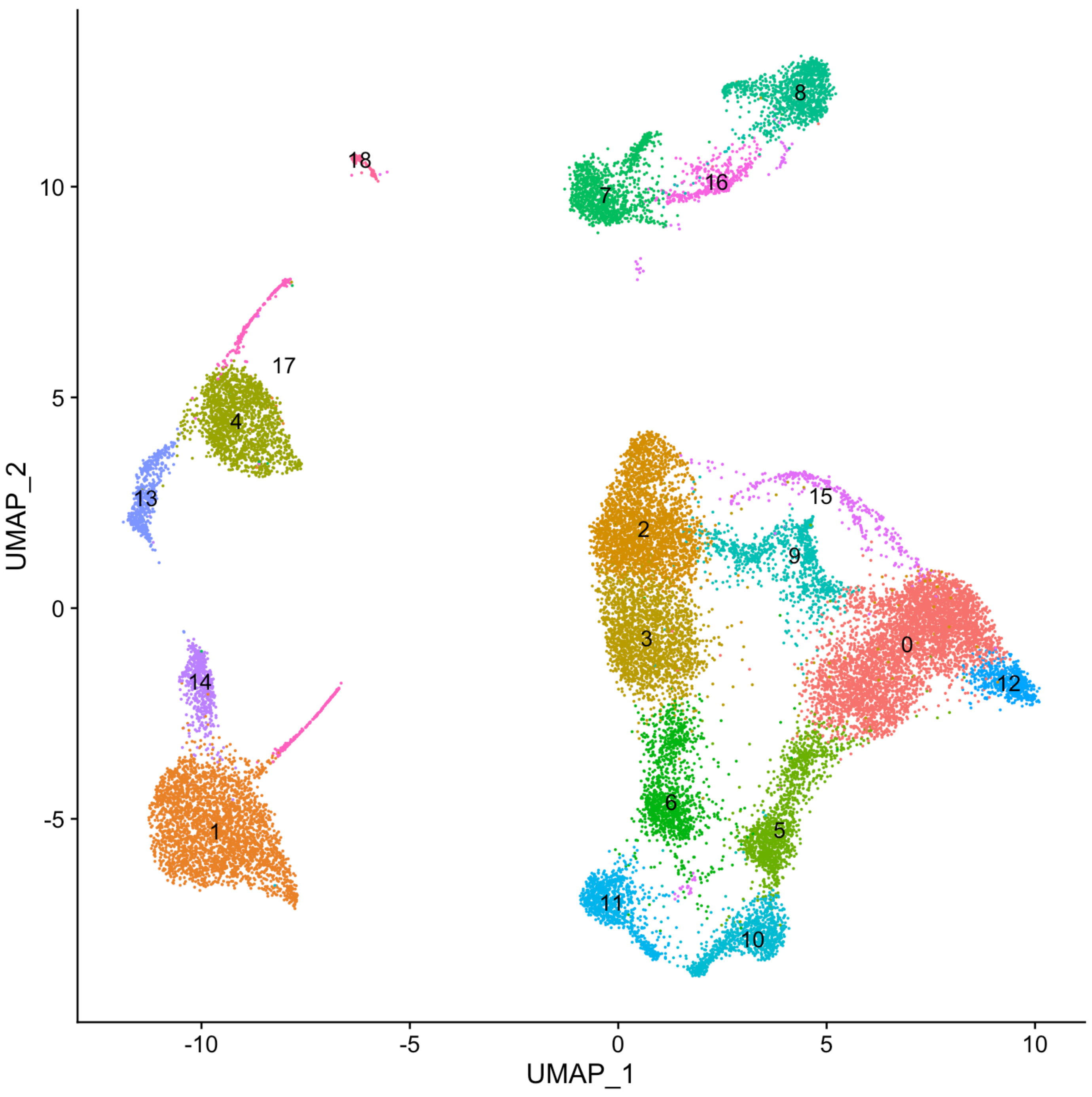
4.3. Dimensionality reduction, embedding and clustering

Perform PCA, embedding and cluster identification with mostly default parameters.
Tuning certain parameters iteratively can provide better embedding and clustering (which is not performed below).

```r
# Find top 2000 variable genes
com_cells = FindVariableFeatures(com_cells, verbose = F)

# Scale and center top features
com_cells = ScaleData(com_cells, verbose = F)

# Perform PCA with 30 components and then UMAP
com_cells = RunPCA(com_cells, npcs = 30, verbose = F)
com_cells = RunUMAP(com_cells, reduction = 'pca', dims = 1:30, verbose = F)

# Construct a nearest neighbors graph for graph clustering
com_cells = FindNeighbors(com_cells, reduction = 'pca', dims = 1:30, verbose = F)
com_cells = FindClusters(com_cells, resolution = 0.5, verbose = F)

# Plot 2D embedding with Ctrl and Stim
UMAPPlot(com_cells, group.by = 'orig.ident') + ggplot2::ggtitle(label = '')
```



```
# Plot 2D embedding with identified clusters
UMAPPlot(com_cells, label = T) + NoLegend()
```



The primary separation of cell groups is by treatment as seen in Fig. 3a of the article.
Additionally, the clustering is occuring within Ctrl and Stim groups. This makes it challenging to identify cell type gene markers.

Thus, it would be better to **harmonize** the Control and Stimulated cells.
This would help to:

1. Obtain low-dimensional representation and clustering without treatment effects
2. Find gene markers across clusters irrespective of IFN-β stimulation
3. Find genes that are up/down regulated by stimulation across clusters

5. Integrating cell matrices across conditions and samples

This relies on Seurat's *anchoring* capabilities.

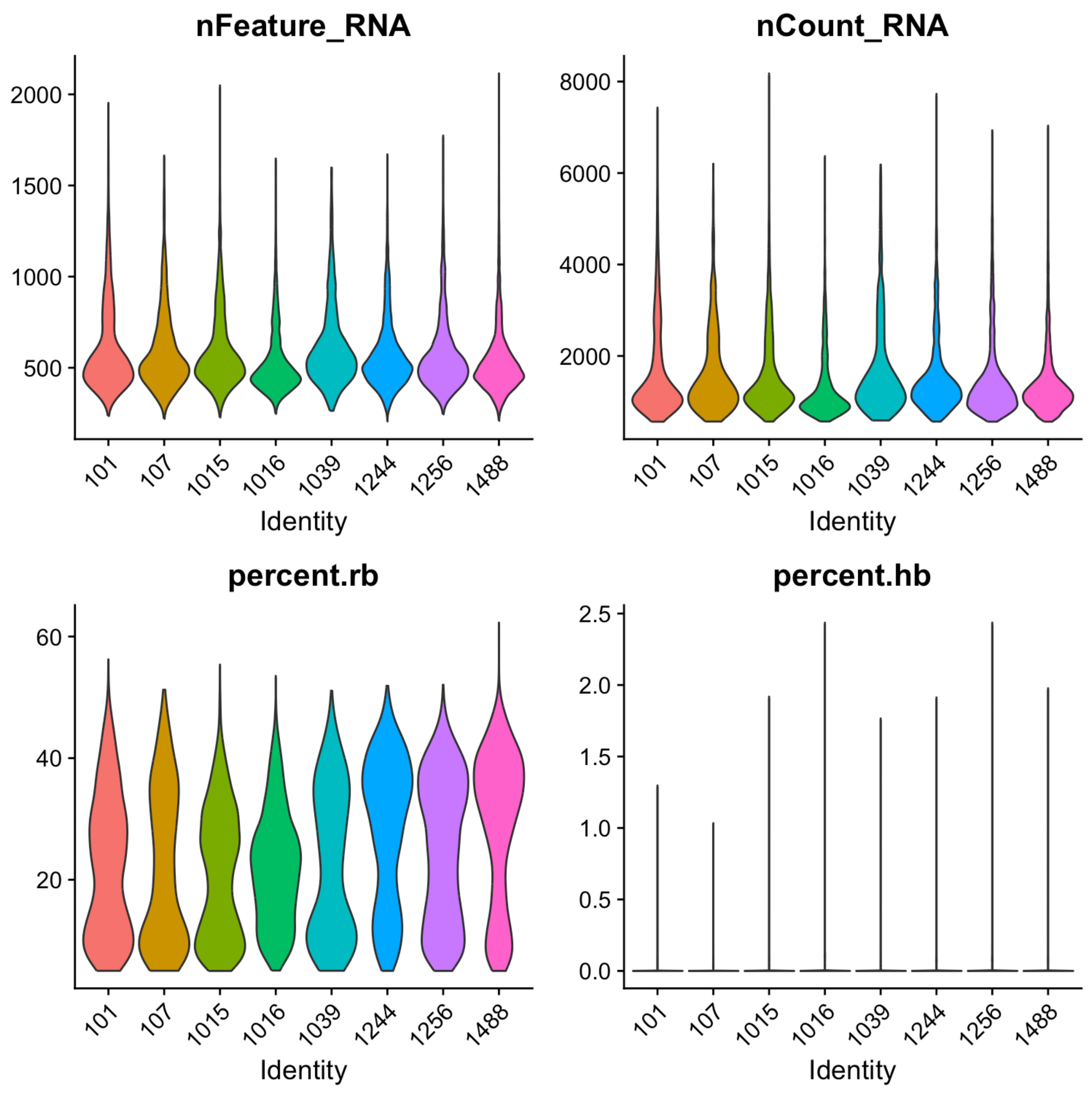
```
# Using the previously filtered cell matrices and split by sample
ctrl_fltr = SplitObject(ctrl_fltr, split.by = 'sample')
stim_fltr = SplitObject(stim_fltr, split.by = 'sample')

# Normalization and top features for each sample
ctrl_fltr = lapply(X = ctrl_fltr, FUN = function(x) {
  x = NormalizeData(x, verbose = F)
  x = FindVariableFeatures(x, verbose = F)
})

stim_fltr = lapply(X = stim_fltr, FUN = function(x) {
  x = NormalizeData(x, verbose = F)
  x = FindVariableFeatures(x, verbose = F)
})
```

WARNING
The below code chunk can take 20 to 30 minutes to complete.

```
anchors = FindIntegrationAnchors(c(ctrl_fltr, stim_fltr), verbose = F)
int_cells = IntegrateData(anchors, verbose = F)
int_cells
```

```
## An object of class Seurat
## 37635 features across 25842 samples within 2 assays
## Active assay: integrated (2000 features, 2000 variable features)
##  1 other assay present: RNA
```

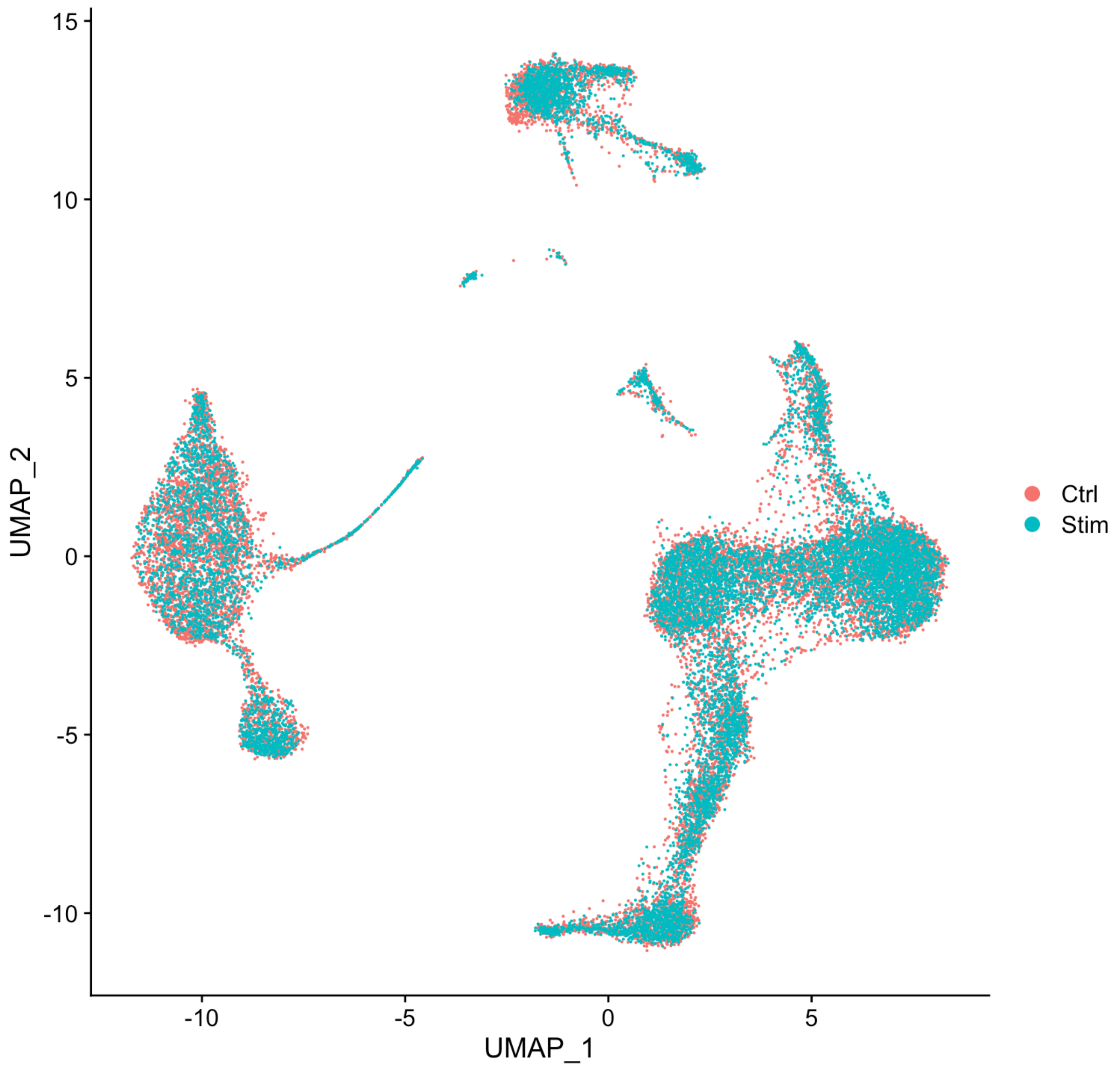
```
# QC for integrated data
DefaultAssay(int_cells) = 'RNA'
int_cells = PercentageFeatureSet(int_cells, '^RP[SL]', col.name = 'percent.rb')
int_cells = PercentageFeatureSet(int_cells, '^HB[^(P)]', col.name = 'percent.hb')
sel_cells = WhichCells(int_cells, expression = nFeature_RNA > 200 & nFeature_RNA < 2500
&
                         percent.rb > 5 & percent.hb < 2.5)
sel_genes = rownames(int_cells)[rowSums(int_cells) > 5]
int_cells = subset(int_cells, cells = sel_cells, features = sel_genes)

VlnPlot(int_cells, group.by = 'sample', pt.size = 0,
        features = c('nFeature_RNA', 'nCount_RNA', 'percent.rb', 'percent.hb'), ncol =
2)
```



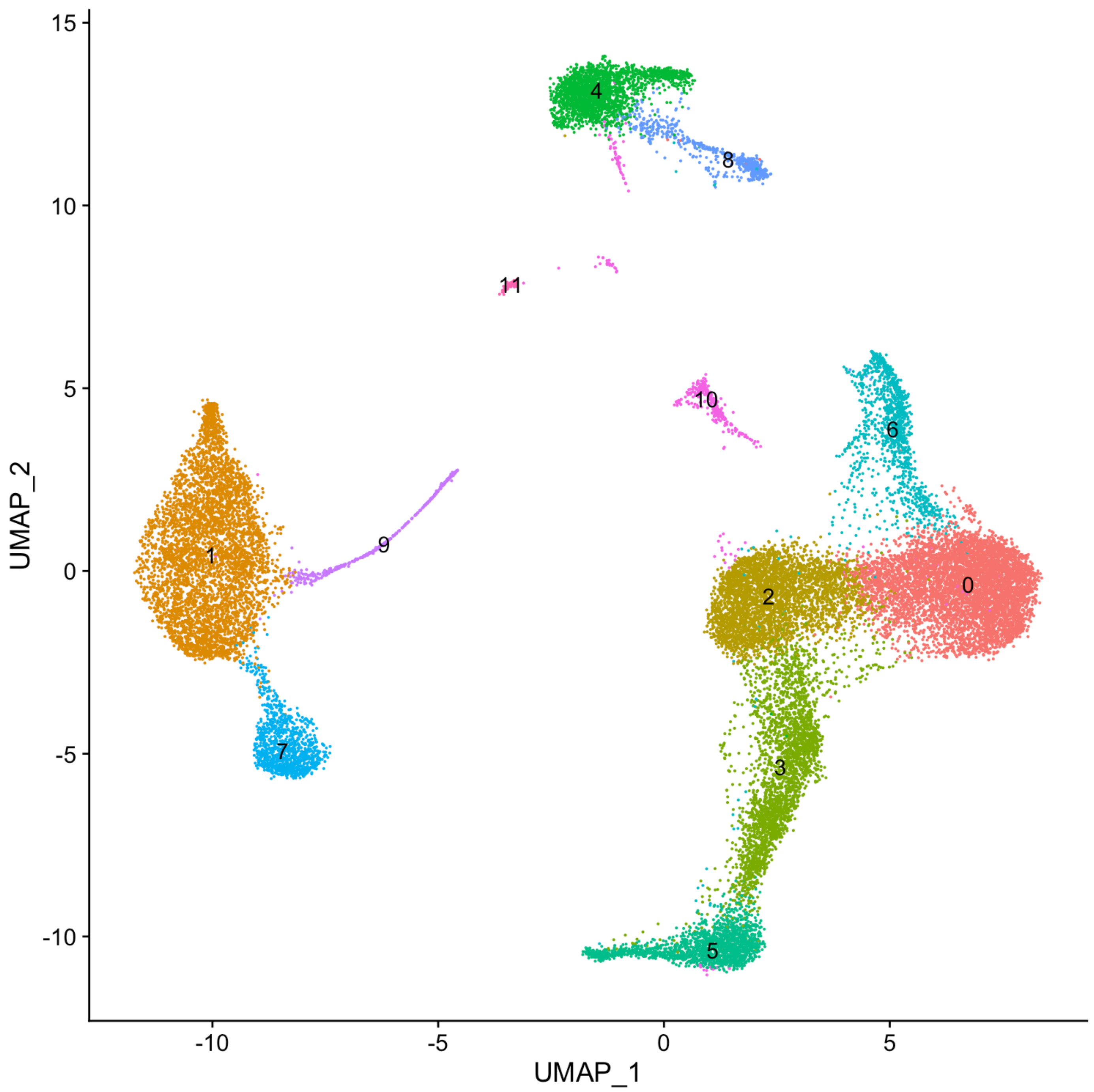
```
int_cells
```

```
## An object of class Seurat
## 17759 features across 24362 samples within 2 assays
## Active assay: RNA (15759 features, 0 variable features)
##  1 other assay present: integrated
```

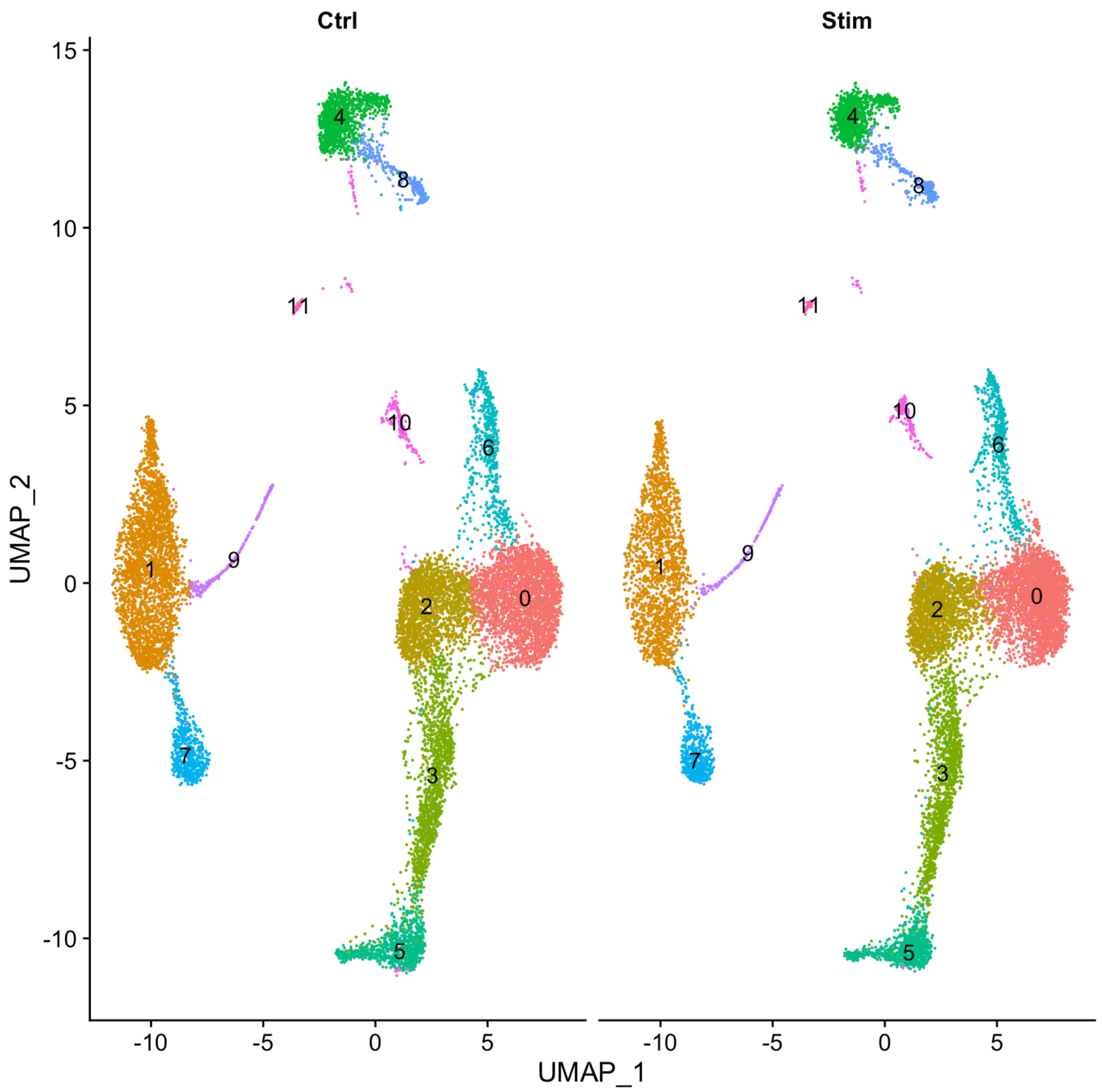
```
DefaultAssay(int_cells) = 'integrated'
int_cells = ScaleData(int_cells, verbose = F)
int_cells = RunPCA(int_cells, npcs = 30, verbose = F)
int_cells = RunUMAP(int_cells, reduction = 'pca', dims = 1:30, verbose = F)
int_cells = FindNeighbors(int_cells, reduction = 'pca', dims = 1:30, verbose = F)
int_cells = FindClusters(int_cells, resolution = 0.5, verbose = F)

UMAPPlot(int_cells, group.by = 'orig.ident') + ggplot2::ggtitle(label = '')
```



```
UMAPPlot(int_cells, label = T) + NoLegend()
```



```
UMAPPlot(int_cells, label = T, split.by = 'orig.ident') + NoLegend()
```



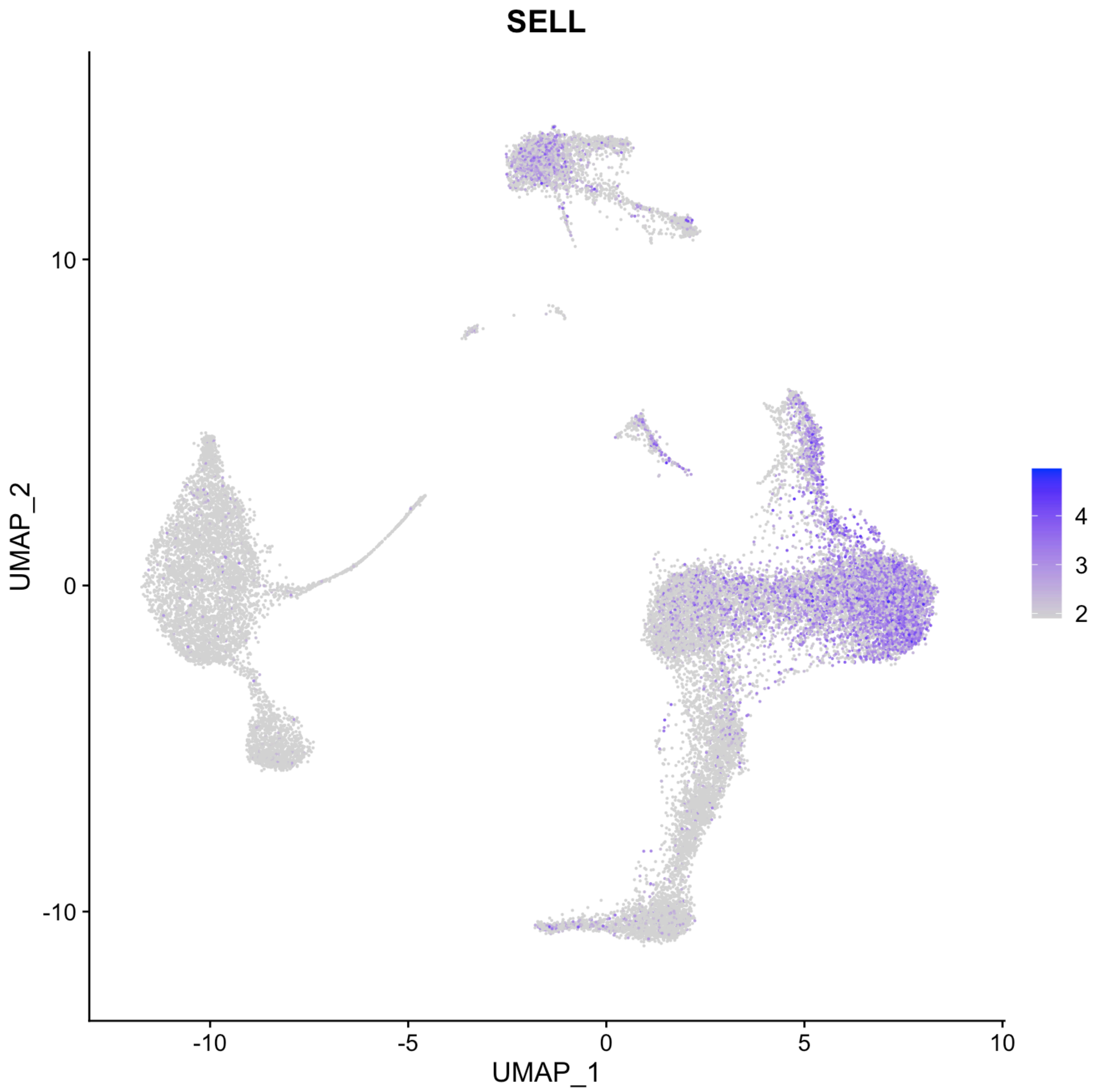
Integration has yielded a harmonized set of cells, in which embedding and clustering is not specific to treatment group.
There are 14 clusters in the whole dataset.

5.1. Gene markers of clusters

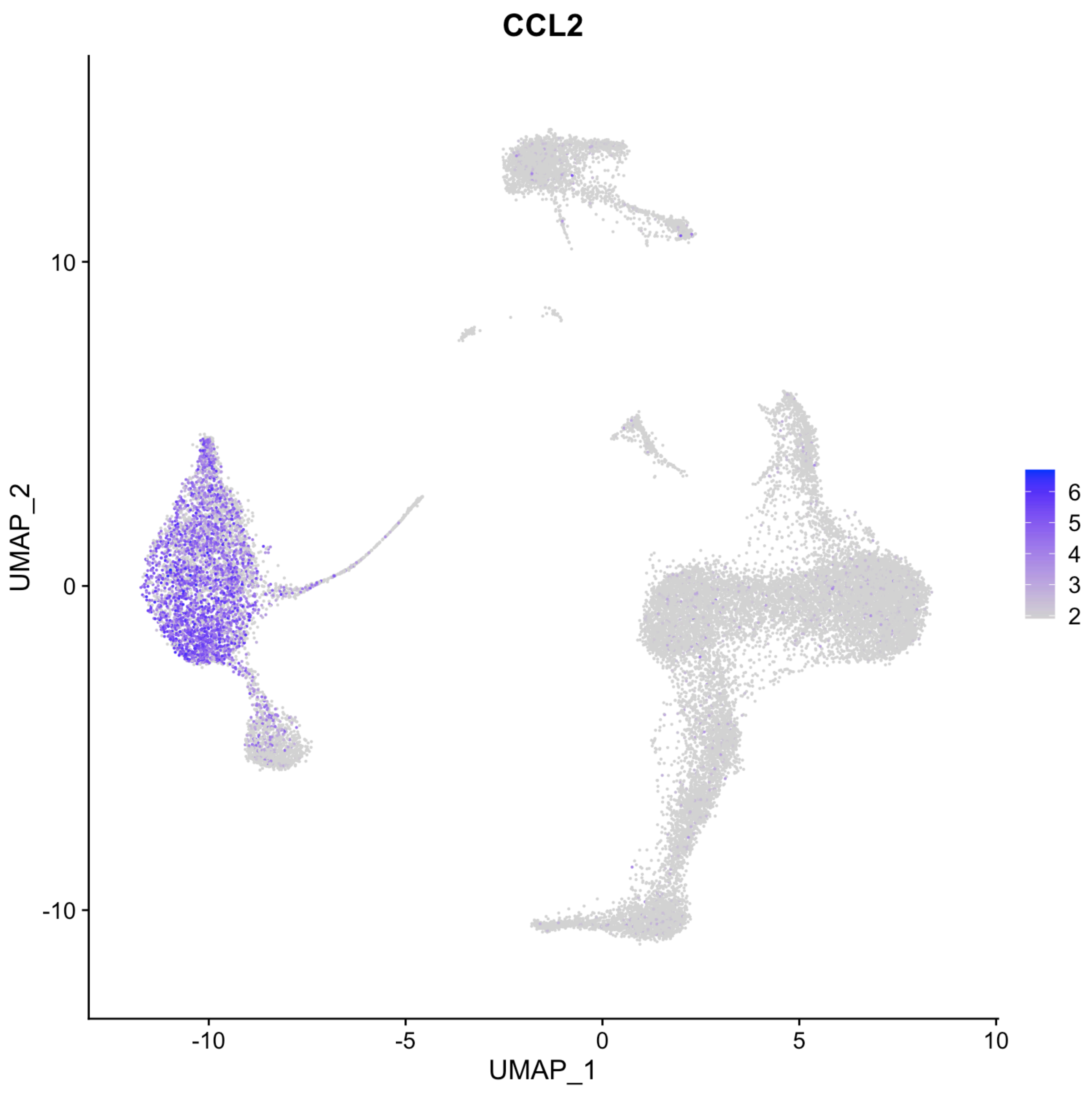
```
DefaultAssay(int_cells) = 'RNA'

# Get top 5 markers for all clusters
get_cons_mrkrs = function(cluster) {
  df = FindConservedMarkers(int_cells, ident.1 = cluster,
                       grouping.var = 'orig.ident',
                       only.pos = T, verbose = F,
                       logfc.threshold = 0.25, min.pct = 0.25, min.diff.pct = 0.25)
                       # Helps to speed up and limit to most significant genes
  df$avg_lfc = (df$Stim_avg_log2FC + df$Stim_avg_log2FC) / 2
  df = df[order(df$minimump_p_val), ]
  df = df[order(df$avg_lfc, decreasing = T), ]
  print(paste0('Cluster-', cluster))
  print(rownames(df)[1:5])
  return(rownames(df)[1:5])
}

clu_mrkrs = c()
max_clu = max(as.numeric(int_cells$seurat_clusters)) - 1
for (i in 0:max_clu) {
  clu_mrkrs = c(clu_mrkrs, get_cons_mrkrs(i))
}
```

```
## [1] "Cluster-0"
## [1] "SELL"   "LTB"    "CCR7"   "GIMAP7" "LDHB"
## [1] "Cluster-1"
## [1] "CCL2"     "C15orf48" "TIMP1"    "SOD2"     "LYZ"
## [1] "Cluster-2"
## [1] "TRAT1"   "ZFP36L2" "CREM"    NA        NA
## [1] "Cluster-3"
## [1] "CCL5" "GZMH" "CD8A" "CD8B" "CST7"
## [1] "Cluster-4"
## [1] "CD79A"   "CD74"    "MS4A1"   "HLA-DRA" "CD83"
## [1] "Cluster-5"
## [1] "GNLY"  "GZMB"  "PRF1"  "NKG7"  "CLIC3"
## [1] "Cluster-6"
## [1] "CACYBP"  "GADD45B" "SRSF2"   "HSPH1"   "CD69"
## [1] "Cluster-7"
## [1] "VMO1"   "FCGR3A" "MS4A7"  "MS4A4A" "CXCL16"
## [1] "Cluster-8"
## [1] "MIR155HG" "HLA-DQA1" "MYC"      "CD83"     "ID3"
## [1] "Cluster-9"
## [1] "TXN"      "HLA-DPB1" "HLA-DRA"  "MARCKSL1" "LYZ"
## [1] "Cluster-10"
## [1] "PPBP"  "PF4"   "GNG11" "SDPR"  "NRGN"
## [1] "Cluster-11"
## [1] "TXN"     "TSPAN13" "ITM2C"   "SEC61B"  "IGJ"
```

```
# Select the top 1 marker for Cluster 0 and 1 respectively for visualization across the
embedding
FeaturePlot(int_cells, features = clu_mrkrs[1], min.cutoff = 'q9')
```



```
FeaturePlot(int_cells, features = clu_mrkrs[6], min.cutoff = 'q9')
```

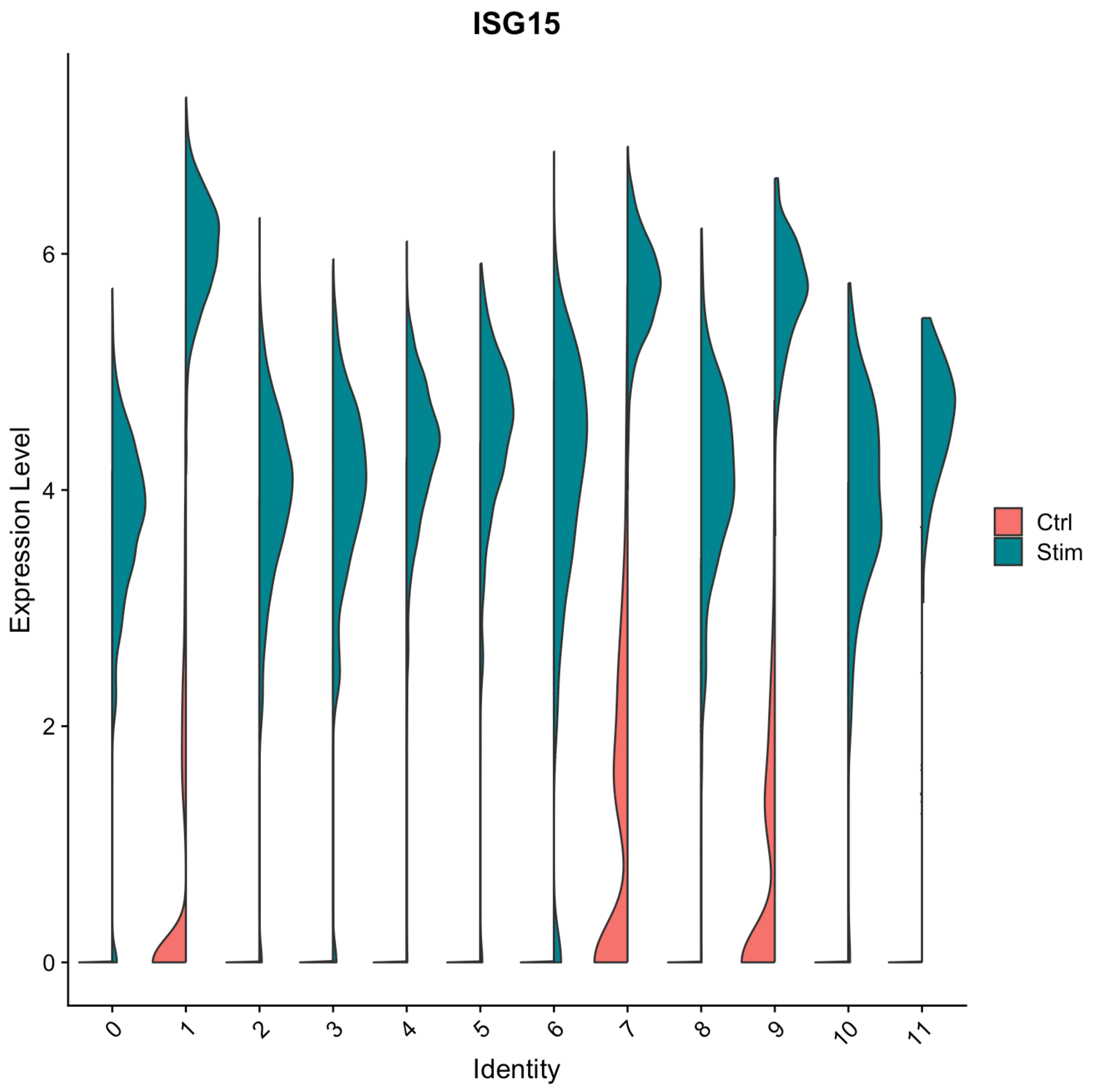


5.2. Genes that are differentially expressed due to interferon stimulation

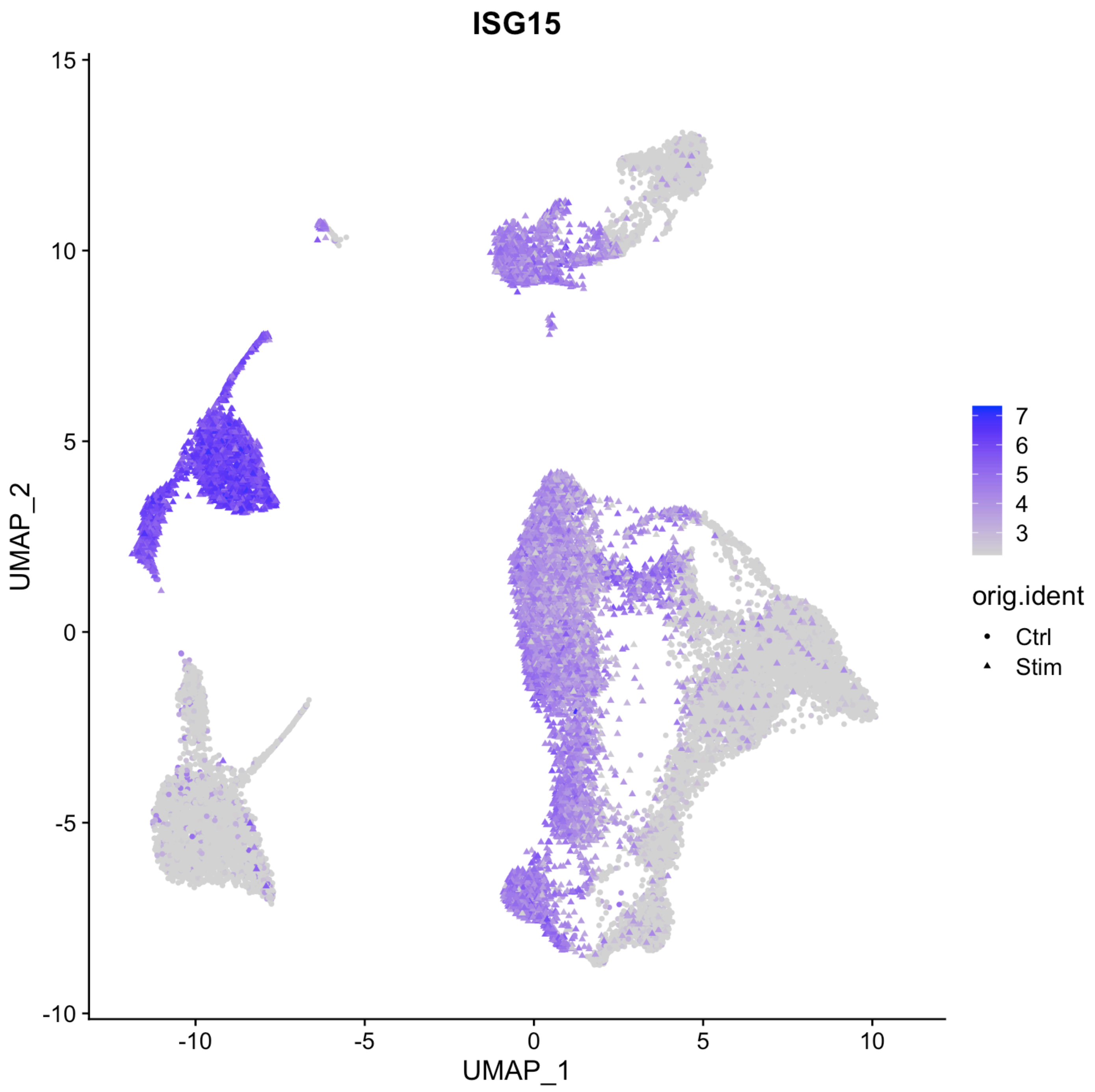
```
int_cells$id_bkp = Idents(int_cells)

Idents(int_cells) = 'orig.ident'

stim_mrkrs = FindMarkers(int_cells, ident.1 = 'Stim', ident.2 = 'Ctrl', only.pos = TRUE,
                        logfc.threshold = 0.5, min.pct = 0.5, min.diff.pct = 0.5, verbos
e = FALSE)
                        # Helps to speed up and limit to most significant genes

head(stim_mrkrs)
```

```
##         p_val avg_log2FC pct.1 pct.2 p_val_adj
## ISG15       0   5.050741 0.984 0.215         0
## IFI6        0   3.816508 0.919 0.120         0
## IFI44L      0   2.589310 0.544 0.034         0
## RSAD2       0   3.425366 0.560 0.019         0
## TNFSF10     0   3.383802 0.623 0.069         0
## LY6E        0   3.175034 0.856 0.143         0
```

```
# Visualize top gene expression profile across conditions
VlnPlot(int_cells, features = rownames(stim_mrkrs)[1], split.by = 'orig.ident', group.by
= 'seurat_clusters',
        split.plot = T, pt.size = 0)
```



```
# Using the previously combined cells with distinct Ctrl and Stim groups to see gene exp
ression
FeaturePlot(com_cells, features = rownames(stim_mrkrs)[1], shape.by = 'orig.ident', pt.s
ize = 1, min.cutoff = 'q9')
```



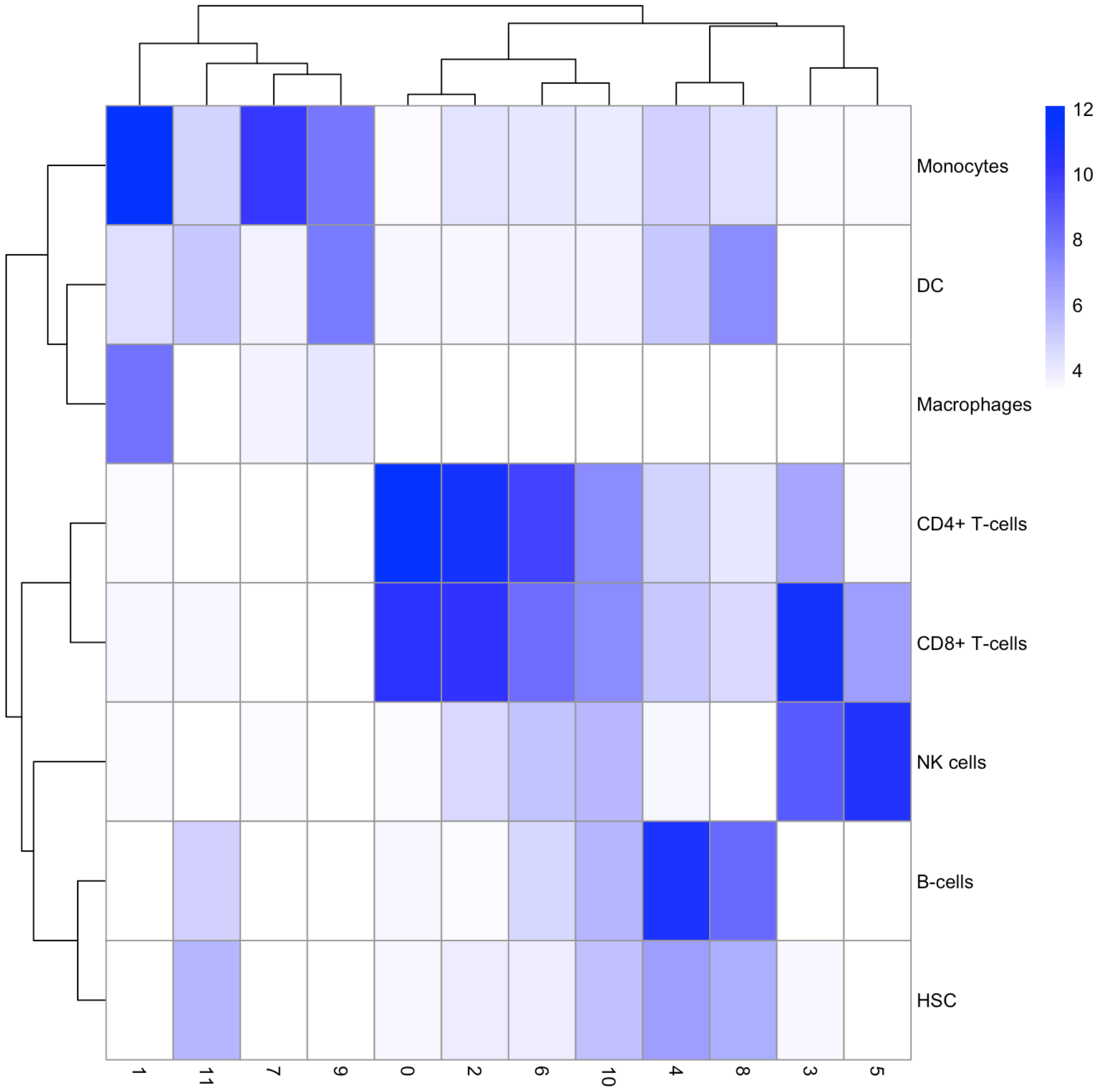
5.3. Cell type identification

Cell type identification can be performed by annotating the clusters with prior knowledge of the above identified gene markers.
This annotation typically requires expert knowledge and/or literature review.
Lately, there are tools available to query a RNA-seq data against a reference RNA-seq dataset to find cell labels.

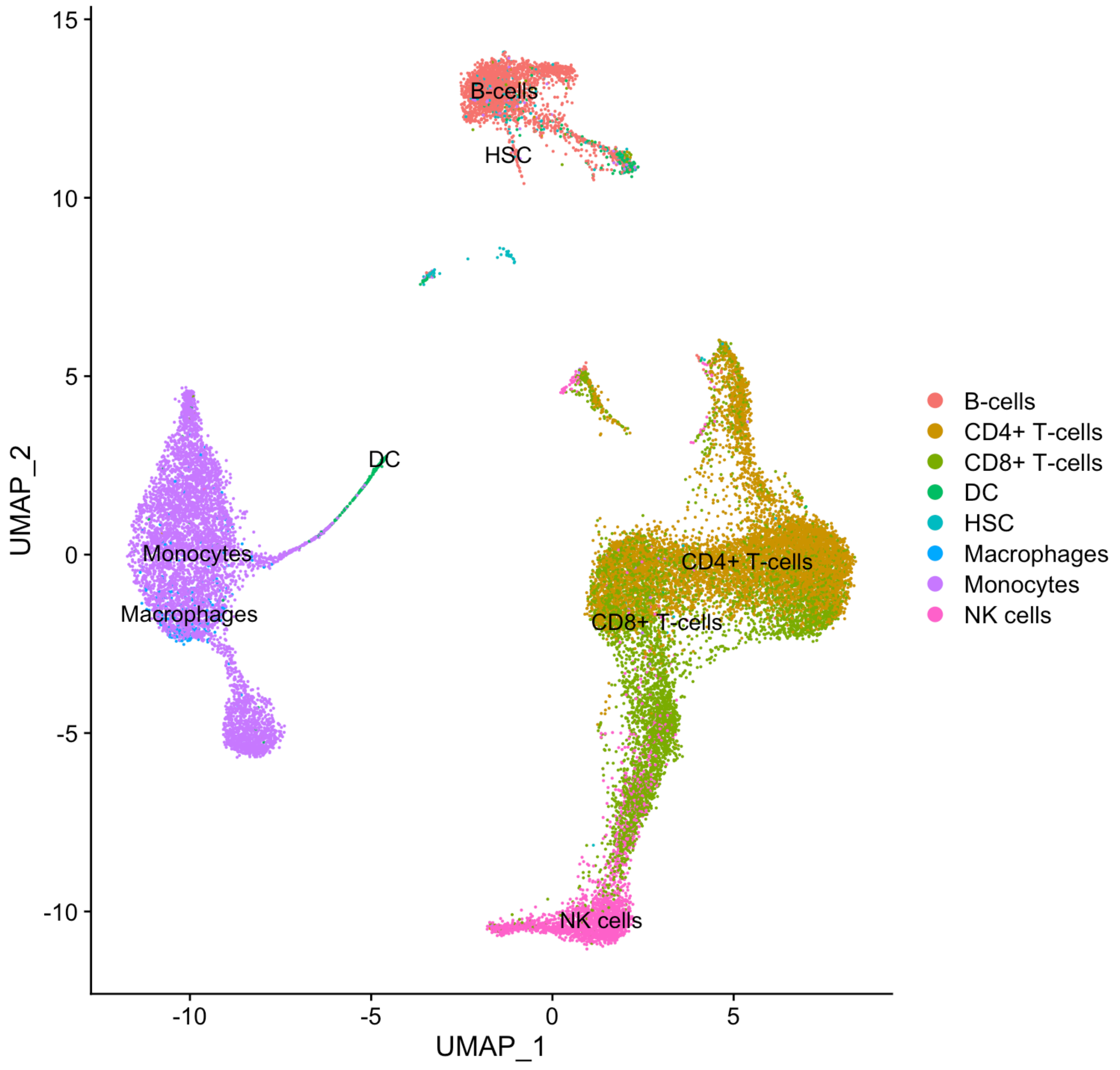
```
# Load cell type data from ENCODE
ref_data = BlueprintEncodeData()

sce = as.SingleCellExperiment(int_cells, assay = 'RNA')

# Use the ENCODE primary cell labels to predict single-cell labels
pred_types = SingleR(test = sce, ref = ref_data, labels = ref_data$label.main)
table(pred_types$labels)
```

```
##
##      B-cells CD4+ T-cells CD8+ T-cells           DC          HSC  Macrophages
##         2342         7573         5523          423          235          259
##    Monocytes     NK cells
##         5736         2271
```

```
# Visualize predicted labels against clusters
tab = table(Assigned = pred_types$labels, Cluster = sce@colData$seurat_clusters)
pheatmap::pheatmap(log2(tab + 10), color = colorRampPalette(c('white', 'blue'))(101))
```



```
int_cells@meta.data$pred_labels = pred_types$labels

UMAPPlot(int_cells, group.by = 'pred_labels', label = T) + ggplot2::ggtitle(label = '')
```



Most of the predicted cell labels match the cell types identified by the authors. Although they do not precisely match all the previously identified clusters directly. This can be improved by iteratively labeling clusters either by expert knowledge or through tools + reference datasets.

---

6. Cleanup

```
system('rm GSM2560248_2.1.mtx.gz GSM2560248_barcodes.tsv.gz GSM2560249_2.2.mtx.gz GSM256
0249_barcodes.tsv.gz')
system('rm GSE96583_batch2.genes.tsv.gz ye1.ctrl.8.10.sm.best ye2.stim.8.10.sm.best')
```