Mafb-restricted local monocyte proliferation precedes lung interstitial macrophage differentiation

12 - cMAF- and Mafb-deficient IM

2022-03-10 20:25:02 +0100

Abstract

Resident tissue macrophages (RTM) are differentiated immune cells populating distinct niches and exhibiting important tissue-supportive functions. RTM maintenance is thought to rely on either monocyte engraftment and differentiation, or RTM self-renewal. Here, we developed an inducible mouse model of lung interstitial macrophage (IM) niche depletion and repopulation to investigate IM development in vivo. Using time-course single-cell RNA-sequencing analyses, bone marrow chimeras and gene targeting, we found that engrafted Ly6C+ classical monocytes could self-renew locally in a CSF1R-dependent manner before their differentiation into RTM. We further showed that the switch from monocyte proliferation towards IM subset specification was controlled by MafB, while c-Maf specifically regulated the identity of the CD206+ IM subset. Our data shed new light on the transcriptional regulation of IM development and provide evidence that, in the mononuclear phagocyte system, self-renewal is not merely restricted to myeloid progenitor cells and mature macrophages, but is also a tightly regulated capability of mature monocytes developing into RTM in vivo.

Contents

1 Description

We would like to know the functions of cMAF and MAFb transcription factors in IM differentiation or IM refilling in empty niche. We analyze the IM with cMAF-KO, MAFb-KO, cMAF/MAF-dKO and control phenotype using scRNAseq (10X Genomics). These mice are Lyz2-Cre induced recombination KO mice, thus gene KO is dependent on the Lyz2 expression.

In this experiment, we had 4 groups of mice:

- Five littermate control mice (Control group);
- Five Lyz2-Cre Maf-flox mice (cMAF-KO group);
- Five Lyz2-Cre Mafb-flox mice (MAFb-KO group);
- Five Lyz2-Cre Maf-flox Mafb-flox mice (dKO group).

Lung monocytes and macrophages were sorted separately and pooled with ratio of 3:7. Poold cells from each sample were stained with unique anti-MHCI-hashtag (Biolegend Hashtag) before pooled sequencing.

The analysis pepline was the same as DT treatment time-course analysis.

2 Demultiplexing by hashtag sequences

```
suppressMessages({
  library(Seurat)
  library(ggpubr)
})
```

2.1 Read data

```
# Load in the UMI matrix (from CR)

umis <- Read10X("../../../counts/scRNAseq/Experiment-7-12-21-ScRNA_NGS21-U976/outs/filtered_feature_bc_matrix/")

# For generating a hashtag count matrix from FASTQ files, please refer to https://github.com/Hoohm/CITE-seq-Count.
# Load in the HTO count matrix
htos <- Read10X("../../../counts/HTO/read_count/", gene.column=1)
```

```
# change cell names to "-1" mode.
colnames(htos) <- paste0(colnames(htos), "-1")

# remove the unmapped as it's not a barcode. It's the last row.
htos <- htos[-nrow(htos), ]

# Select cell barcodes detected by both RNA and HTO
# In the example datasets we have already filtered the cells for you, but perform this step for clarity.
joint.bcs <- intersect(colnames(umis), colnames(htos))

# Subset RNA and HTO counts by joint cell barcodes
umis <- umis[, joint.bcs]
htos <- as.matrix(htos[, joint.bcs])


# Confirm that the HTO have the correct names
```

```
rownames(htos)
```

```
## [1] "HT5_Control-CTTTGTCTTTGTGAG" "HT6_cMAF_KO-TATGCTGCCACGGTA"
## [3] "HT7_MAFb_KO-GAGTCTGCCAGTATC" "HT8_dKO-TATAGAACGCCAGGC"
```

Setup Seurat object and add in the HTO data

```
# Setup Seurat object
hashtag <- CreateSeuratObject(counts = umis)

# Normalize RNA data with log normalization
hashtag <- NormalizeData(hashtag)
# Find and scale variable features
hashtag <- FindVariableFeatures(hashtag, selection.method = 'mean.var.plot
    ')
hashtag <- ScaleData(hashtag, features = VariableFeatures(hashtag))
```

2.2 Adding HTO data as an independent assay

You can read more about working with multi-modal data here

```
# Add HTO data as a new assay independent from RNA
hashtag[['HTO']] <- CreateAssayObject(counts = htos)
# Normalize HTO data, here we use centered log-ratio (CLR) transformation
hashtag <- NormalizeData(hashtag, assay = 'HTO', normalization.method = '
    CLR')
```

2.3 Demultiplex cells based on HTO enrichment

Here we use the Seurat function `HTODemux()` to assign single cells back to their sample origins.

```
# If you have a very large dataset we suggest using k_function = "clara".
    This is a k-medoid clustering function for large applications
# You can also play with additional parameters (see documentation for
    HTODemux()) to adjust the threshold for classification
# Here we are using the default settings
hashtag <- HTODemux(hashtag, assay = "HTO", positive.quantile = 0.94)
```

2.4 Visualize demultiplexing results

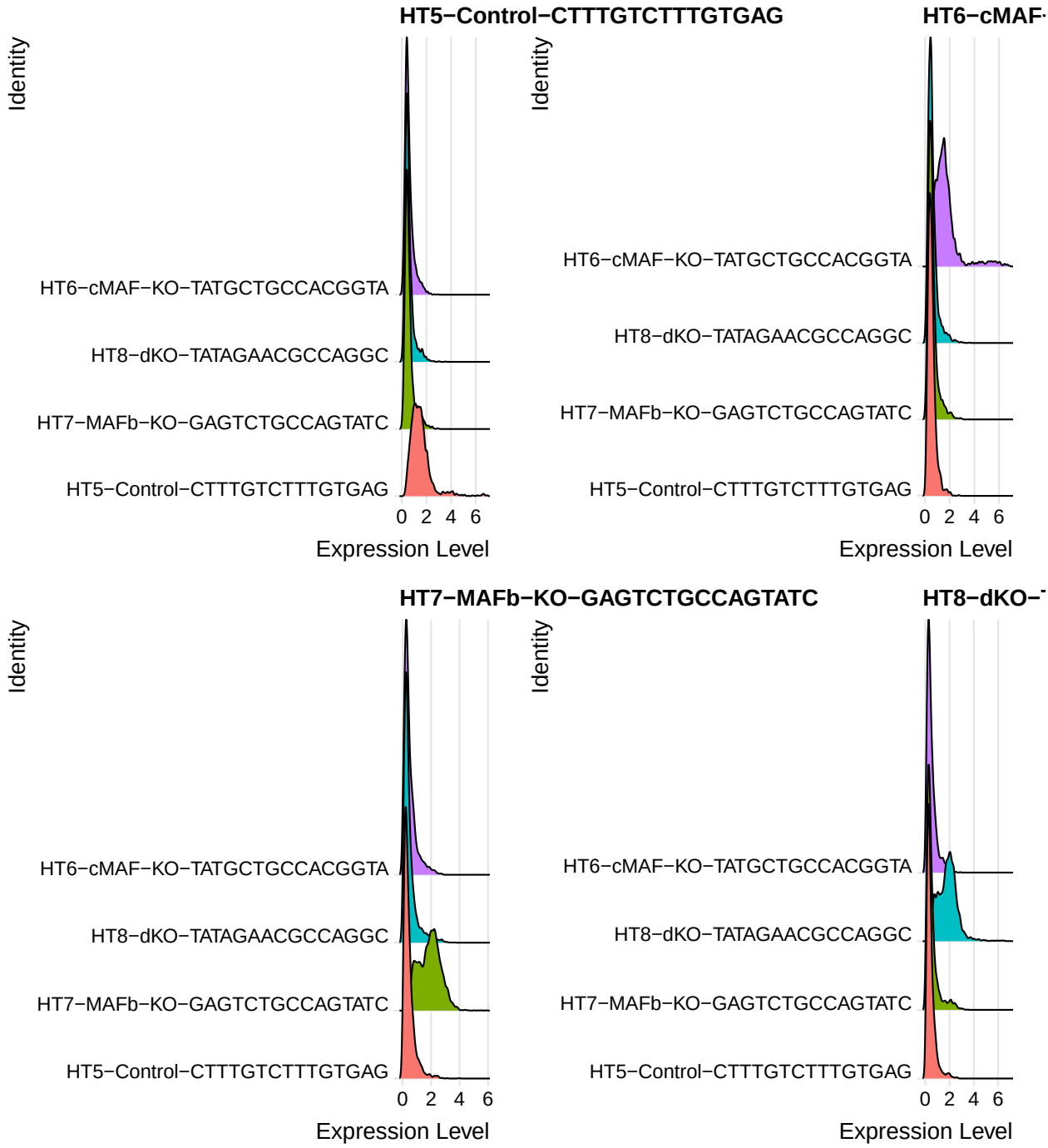
Output from running `HTODemux()` is saved in the object metadata. We can visualize how many cells are classified as singlets, doublets and negative/ambiguous cells.

```
# Global classification results
table(hashtag$HTO_classification.global)
```

```
##
##  Doublet Negative  Singlet
##     2094     2591     6653
```

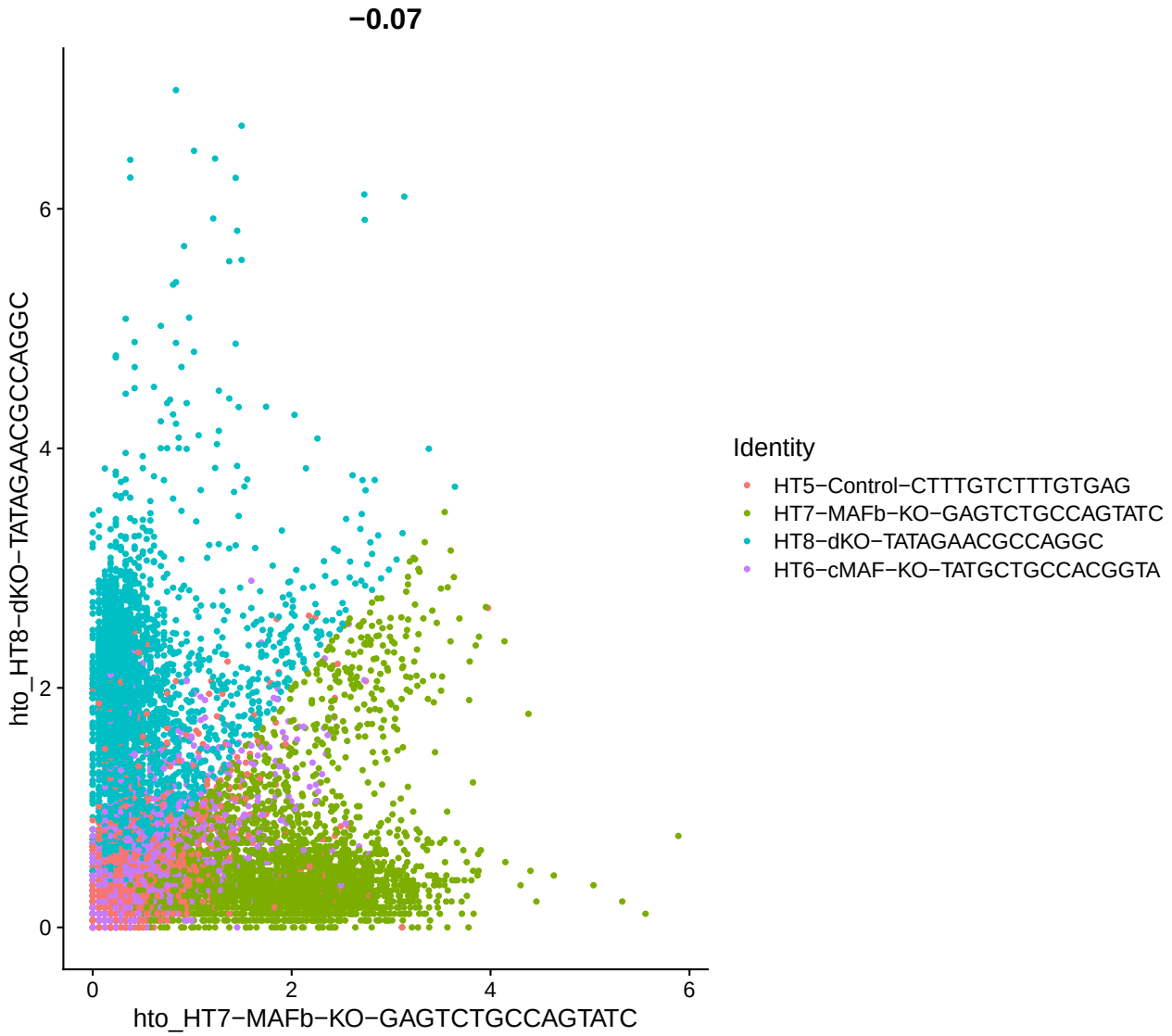
Visualize enrichment for selected HTOs with ridge plots

```
# Group cells based on the max HTO signal
Idents(hashtag) <- 'HTO_maxID'
RidgePlot(hashtag, assay = 'HTO', features = rownames(hashtag[['HTO']]),
    ncol = 2)
```

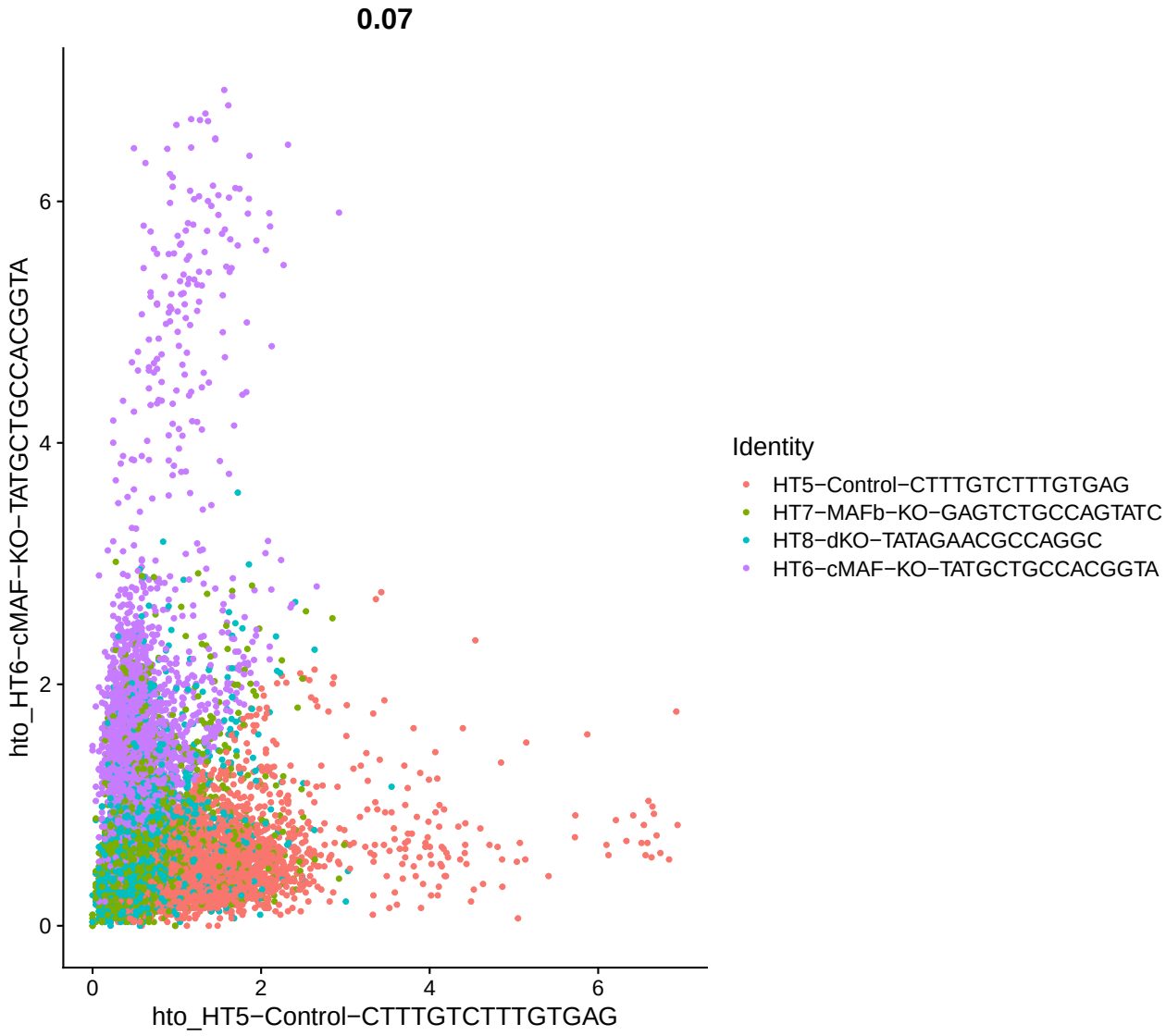


Visualize pairs of HTO signals to confirm mutual exclusivity in singlets

```
FeatureScatter(hashtag, feature1 = 'HT7-MAFb-KO-GAGTCTGCCAGTATC', feature2
    = 'HT8-dKO-TATAGAACGCCAGGC')
```

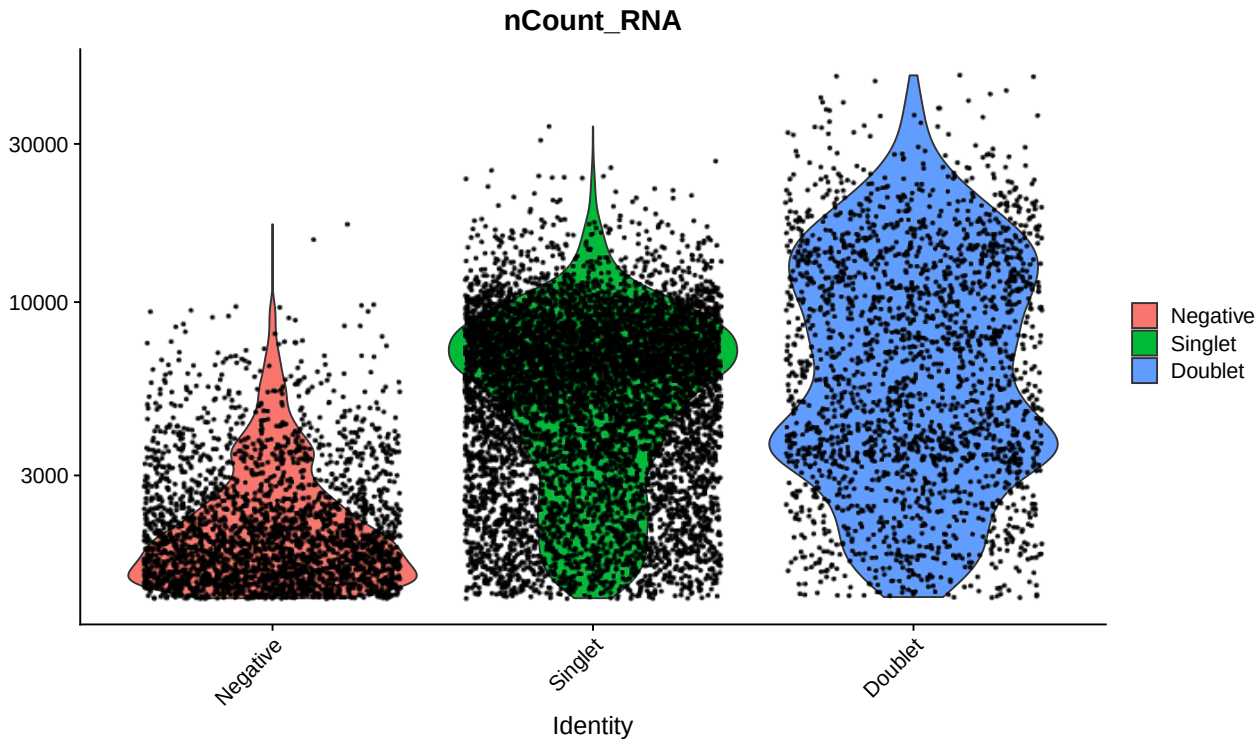



```
FeatureScatter(hashtag, feature1 = 'HT5-Control-CTTTGTCTTTGTGAG', feature2
    = 'HT6-cMAF-KO-TATGCTGCCACGGTA')
```



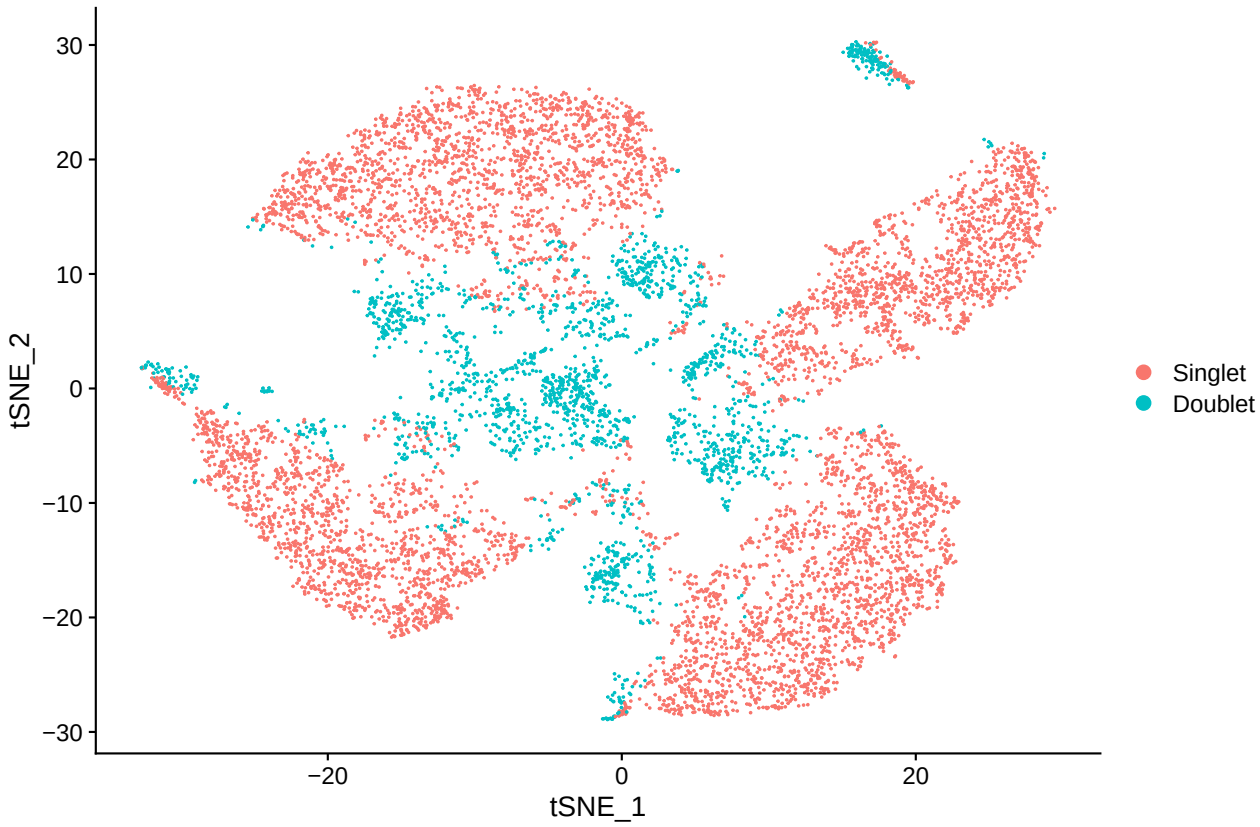
Compare number of UMIs for singlets, doublets and negative cells

```
Idents(hashtag) <- 'HTO_classification.global'
VlnPlot(hashtag, features = 'nCount_RNA', pt.size = 0.1, log = TRUE)
```



Generate a two dimensional tSNE embedding for HTOs.Here we are grouping cells by singlets and doublets for simplicity.

```
#First, we will remove negative cells from the object
hashtag.subset <- subset(hashtag, idents = 'Negative', invert = TRUE)

# Calculate a tSNE embedding of the HTO data
DefaultAssay(hashtag.subset) <- "HTO"
hashtag.subset <- ScaleData(hashtag.subset, features = rownames(hashtag.
    subset), verbose = FALSE)
hashtag.subset <- RunPCA(hashtag.subset, features = rownames(hashtag.
    subset), approx = FALSE)
hashtag.subset <- RunTSNE(hashtag.subset, dims = 1:8, perplexity = 100,
    check_duplicates = FALSE)

DimPlot(hashtag.subset)
```



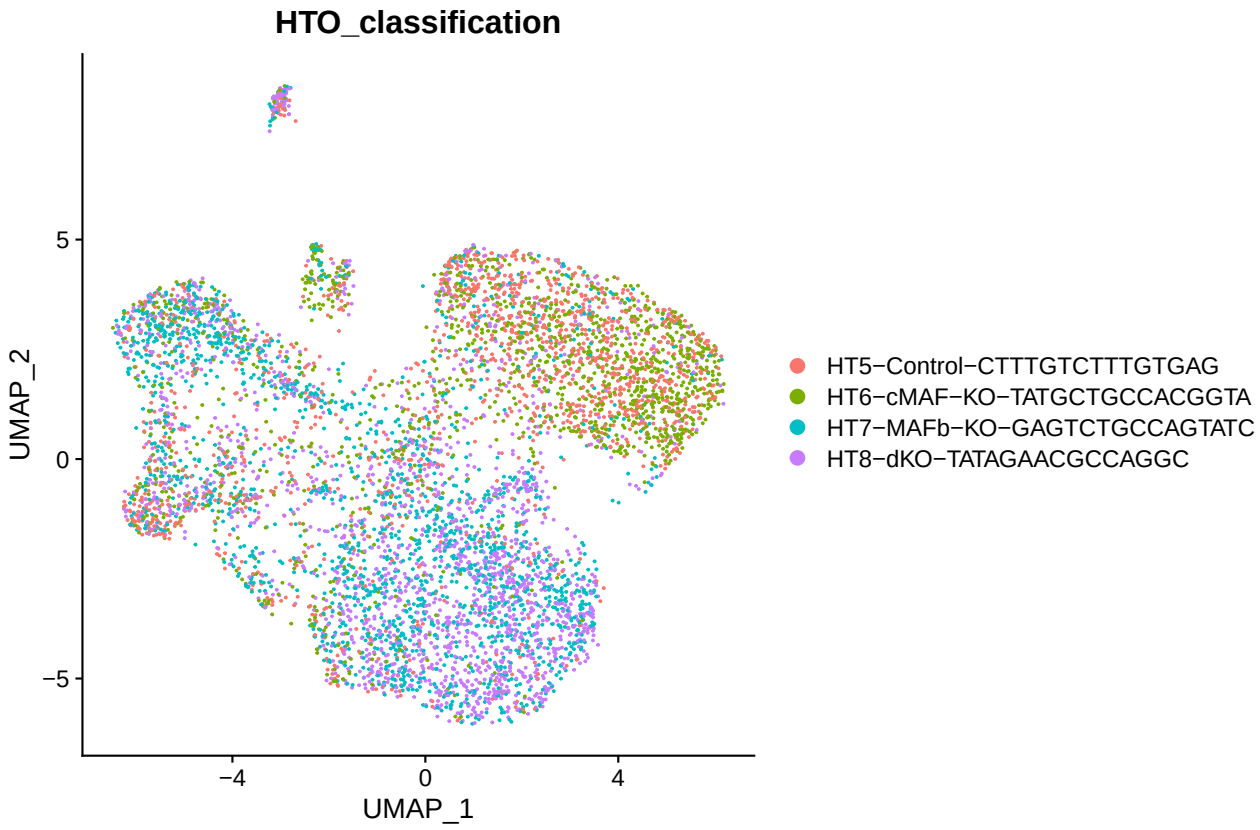
Create an HTO heatmap, based on Figure 1C in the Cell Hashing paper.

```
#To increase the efficiency of plotting, you can subsample cells using the
    num.cells argument
HTOHeatmap(hashtag, assay = 'HTO', ncells = 1000)
```

Cluster and visualize cells using the usual scRNA-seq workflow, and examine for the potential presence of batch effects.

```
# Extract the singlets
singlet <- subset(hashtag, idents = 'Singlet')

# Select the top 1000 most variable features
singlet <- FindVariableFeatures(singlet, selection.method = 'mean.var.plot
    ')

# Scaling RNA data, we only scale the variable features here for
    efficiency
singlet <- ScaleData(singlet, features = VariableFeatures(singlet))

# Run PCA
singlet <- RunPCA(singlet, features = VariableFeatures(singlet))
```

```
# We select the top 10 PCs for clustering and tSNE based on PCElbowPlot
singlet <- FindNeighbors(singlet, reduction = 'pca', dims = 1:10)
singlet <- FindClusters(singlet, resolution = 0.6, verbose = FALSE)
singlet <- RunTSNE(singlet, reduction = 'pca', dims = 1:10)
singlet <- RunUMAP(singlet, reduction = 'pca', dims = 1:10)

# Projecting singlet identities on TSNE visualization
DimPlot(singlet, group.by = "HTO_classification")
```



save to seurat object:

```
saveRDS(singlet, file = "./demultiplexed.seuratObject.rds")
```

3 scRNAseq initiation - QC

Given the low apoptotic rate in the sample, we consider cells with >10% mt genes as apoptotic cells and filter them out from the further analyses.

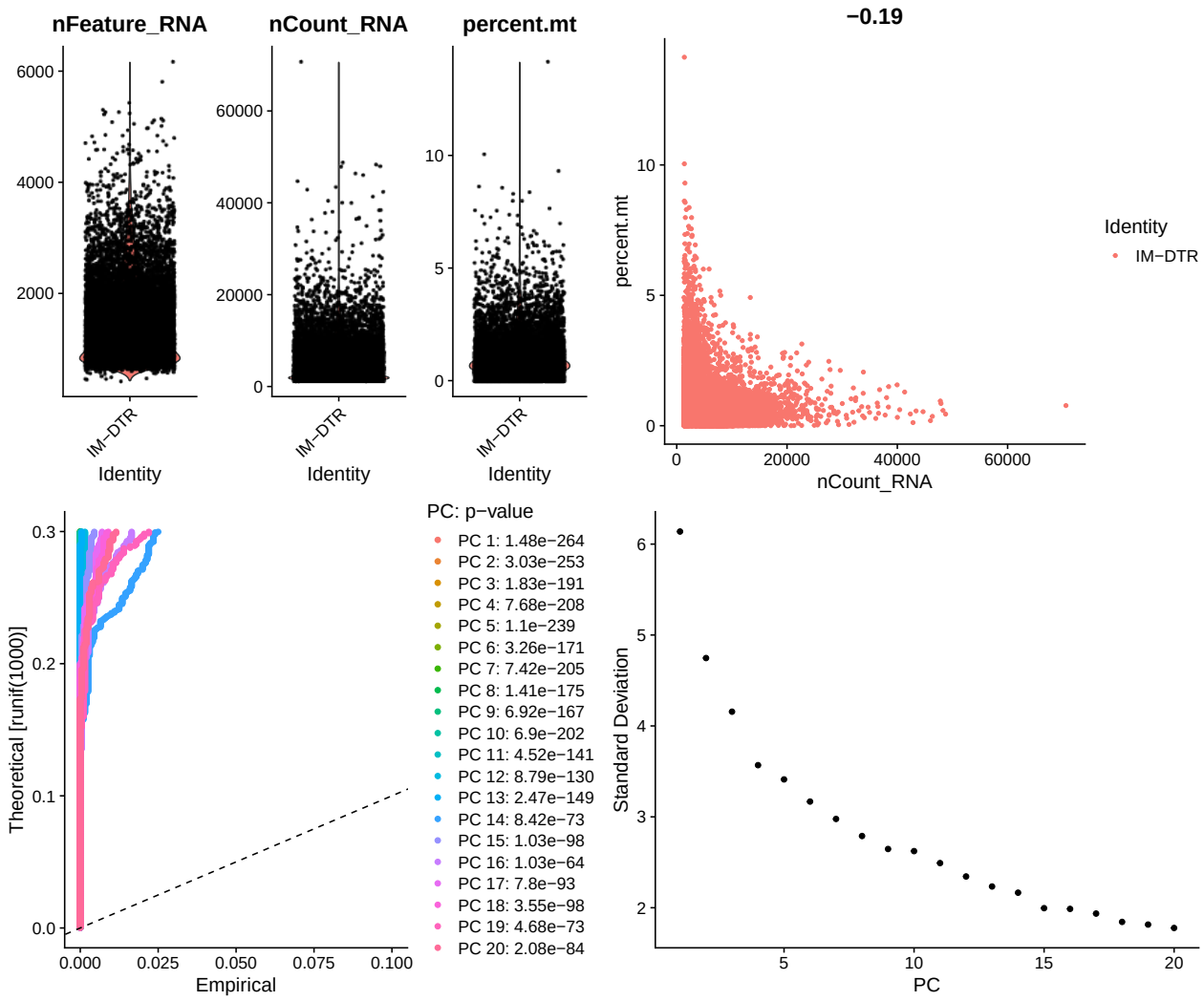
```
# load package and data
library(Seurat)
source("../R/seurat.setup.R")
mt.percentage <- 10
dimensionality <- 1:20
```

3.1 QC

```
IM_Maf <- seurat.setup(path.10x = "/mnt/Data/Single-cell_Analysis/Projects
   /IPL/IM_DTR/IM-DTR_MAF/counts/scRNAseq/Experiment-7-12-21-ScRNA_NGS21-
   U976/outs/filtered_feature_bc_matrix/", project = "IM-DTR",
   dimensionality = dimensionality, mt.percentage = mt.percentage, human =
    FALSE)
```

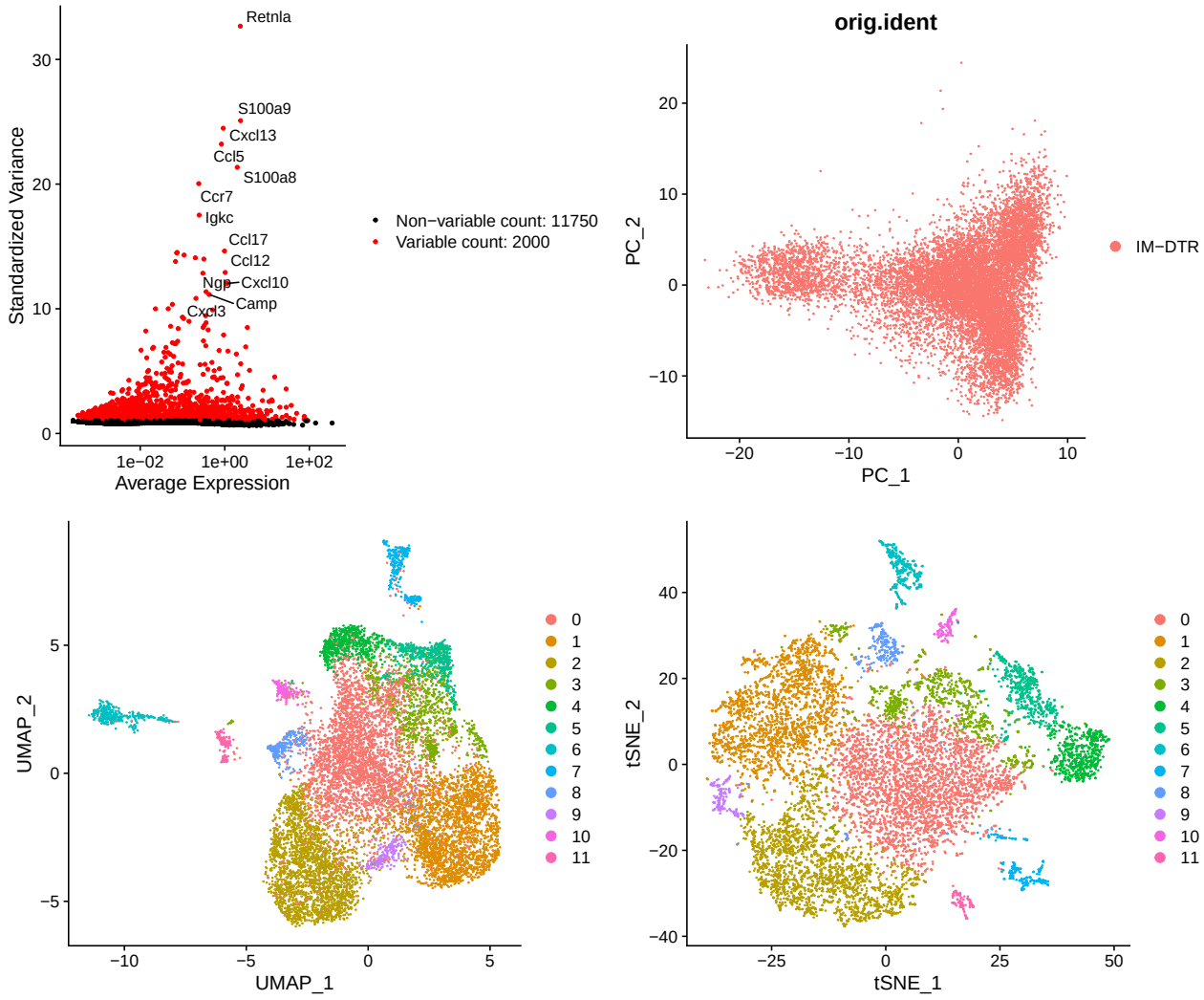
```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 11452
## Number of edges: 367228
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.8751
## Number of communities: 12
## Elapsed time: 1 seconds
```

```
ggarrange(IM_Maf$plots$feature_vln, IM_Maf$plots$RNA_mt.pct_scatter, IM_
   Maf$plots$JackStrawPlot, IM_Maf$plots$ElbowPlot, ncol = 2, nrow = 2)
```




```
ggarrange(IM_Maf$plots$variable_features, IM_Maf$plots$PCA_plot, IM_Maf$
    plots$UMAP_plot, IM_Maf$plots$TSNE_plot, ncol = 2, nrow = 2)
```

1



4 Annotate cells by hashtags

4.1 Load Chromium/HTO data

```
IM_Maf.seuratObject <- IM_Maf$seuratObject
demultiplexed.seuratObject <- singlet
```

Assign HTO annotation to cells

```
common <- intersect(colnames(IM_Maf.seuratObject), colnames(demultiplexed.
    seuratObject))

IM_Maf.seuratObject <- subset(IM_Maf.seuratObject, cells = common)
```

```
# make intersect between hto and rna cells:
hto <- demultiplexed.seuratObject@meta.data$HTO_classification
names(hto) <- colnames(demultiplexed.seuratObject)

# remove the sequence chars
hto <- sub(hto, pattern = "-[C,T,G,A]{0,15}$", replacement = "")
```

```
                                                                    7
# assign to seurat object:                                          8
IM_Maf.seuratObject$group <- hto[colnames(IM_Maf.seuratObject)]     9
```

4.2 Make metadata for samples

```
# cell type                                                         1
IM_Maf.seuratObject$cell.type0 <- "CD45+"                           2
```

Save individual samples for other use

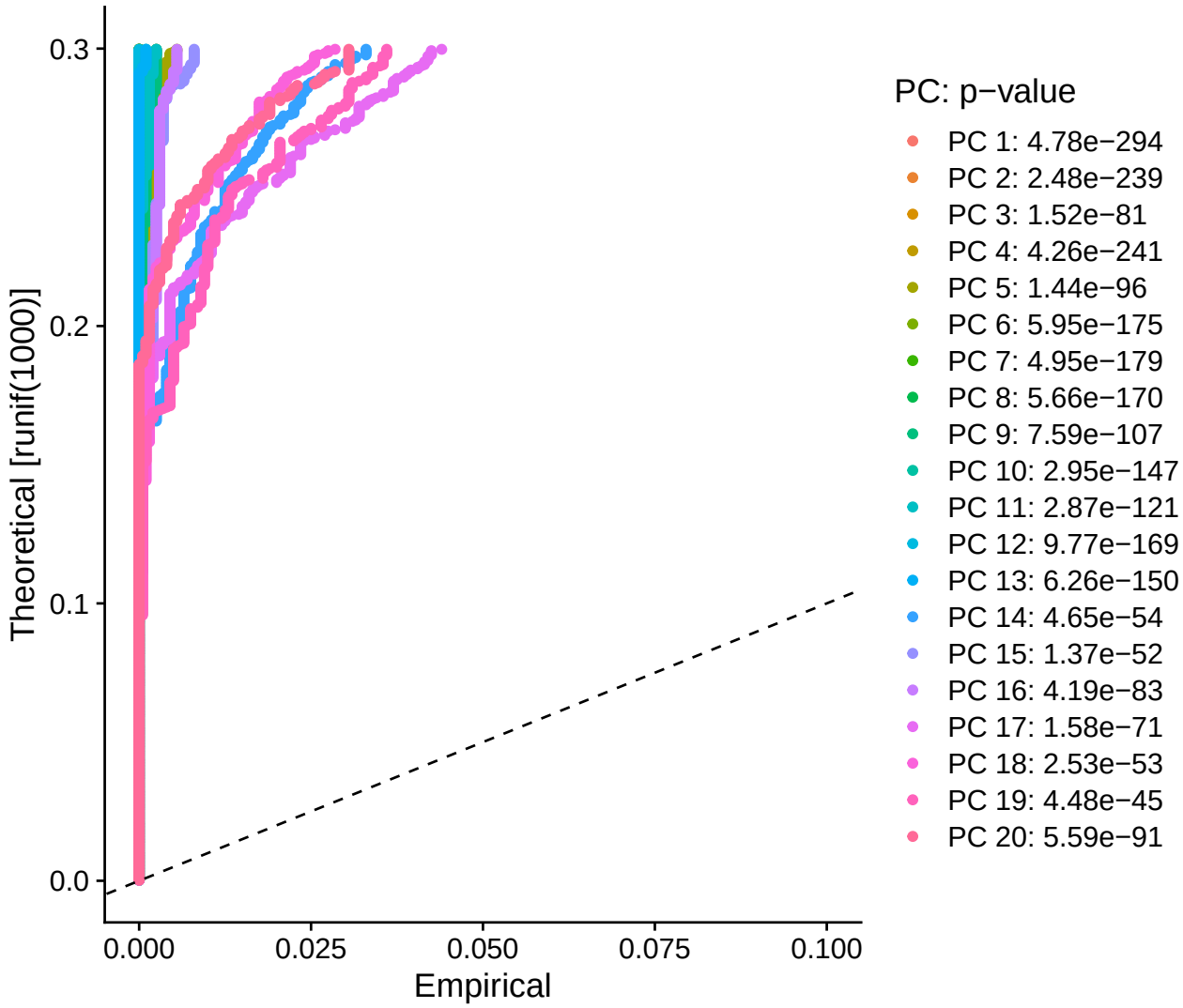
```
obj <- IM_Maf.seuratObject                                                  1
for (i in unique(obj$group)) {                                              2
  obj.sub <- subset(obj, subset = group == i)                               3
  file.name <- paste("IM_mono", i, "seuratObject.rds", sep = ".")           4
  saveRDS(object = obj.sub, file = file.path(".", file.name))               5
}                                                                           6
```

4.3 Data processing and cell clustering

```
IM_Maf.seuratObject <- NormalizeData(IM_Maf.seuratObject)                   1
IM_Maf.seuratObject <- FindVariableFeatures(IM_Maf.seuratObject, selection  2
   .method = "vst", nfeatures = 2000)
IM_Maf.seuratObject <- ScaleData(IM_Maf.seuratObject, features = rownames(  3
   IM_Maf.seuratObject))
```

Linear dimension reduction:

```
IM_Maf.seuratObject <- RunPCA(IM_Maf.seuratObject,                          1
                                        features = VariableFeatures(        2
                                           object = IM_Maf.seuratObject
                                           ))
IM_Maf.seuratObject <- JackStraw(IM_Maf.seuratObject, num.replicate = 100)  3
IM_Maf.seuratObject <- ScoreJackStraw(IM_Maf.seuratObject, dims = 1:20)     4
JackStrawPlot(IM_Maf.seuratObject, dims = 1:20)                             5
```

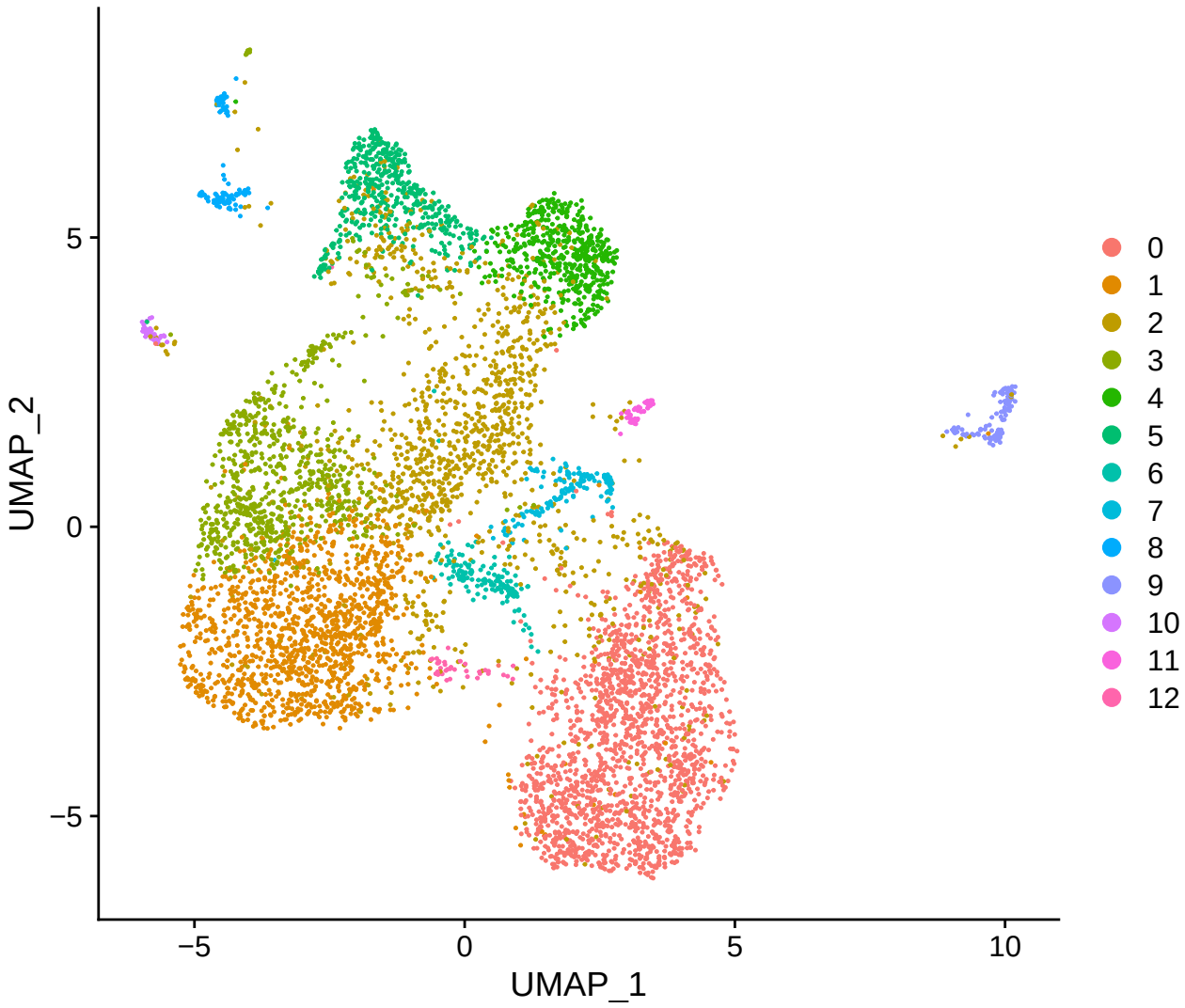



```
IM_Maf.seuratObject <- FindNeighbors(IM_Maf.seuratObject, dims = 1:30)
IM_Maf.seuratObject <- FindClusters(IM_Maf.seuratObject, resolution = 0.5)
```

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 6652
## Number of edges: 247502
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.8791
## Number of communities: 13
## Elapsed time: 0 seconds
```

```
IM_Maf.seuratObject <- RunTSNE(IM_Maf.seuratObject, dims = 1:30)
IM_Maf.seuratObject <- RunUMAP(IM_Maf.seuratObject, dims = 1:30)
```

```
DimPlot(IM_Maf.seuratObject)
```



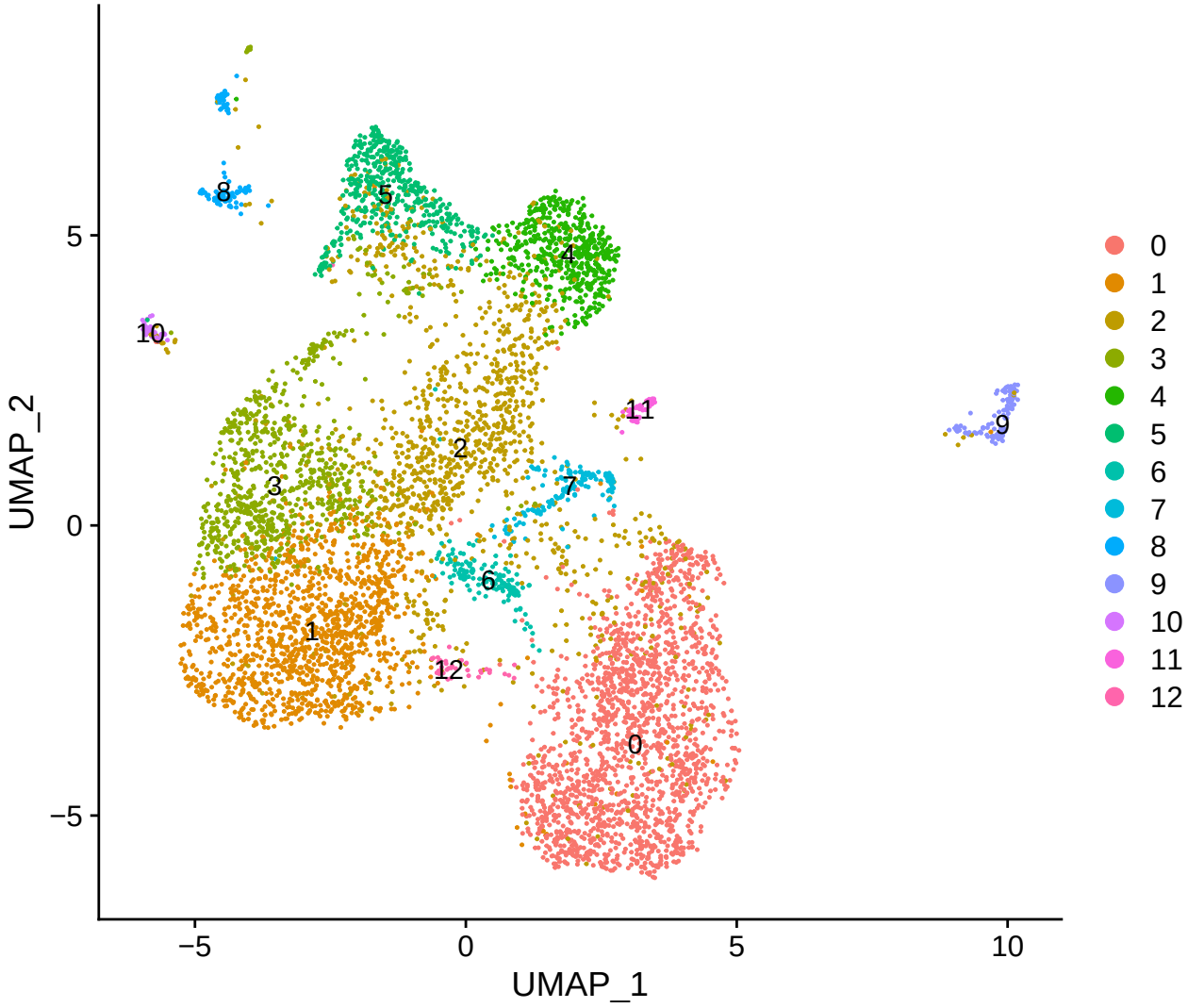
4.4 Celltyping

```
source("../R/seurat2singleR.R")

results.singleR <- seurat2singleR(IM_Maf.seuratObject, ref = "ImmGenData")

saveRDS(results.singleR, file = "./IM_Maf.ImmGenData.singleR.Rds")
saveRDS(IM_Maf.seuratObject, file = "./IM_Maf.seuratObject.rds")
```

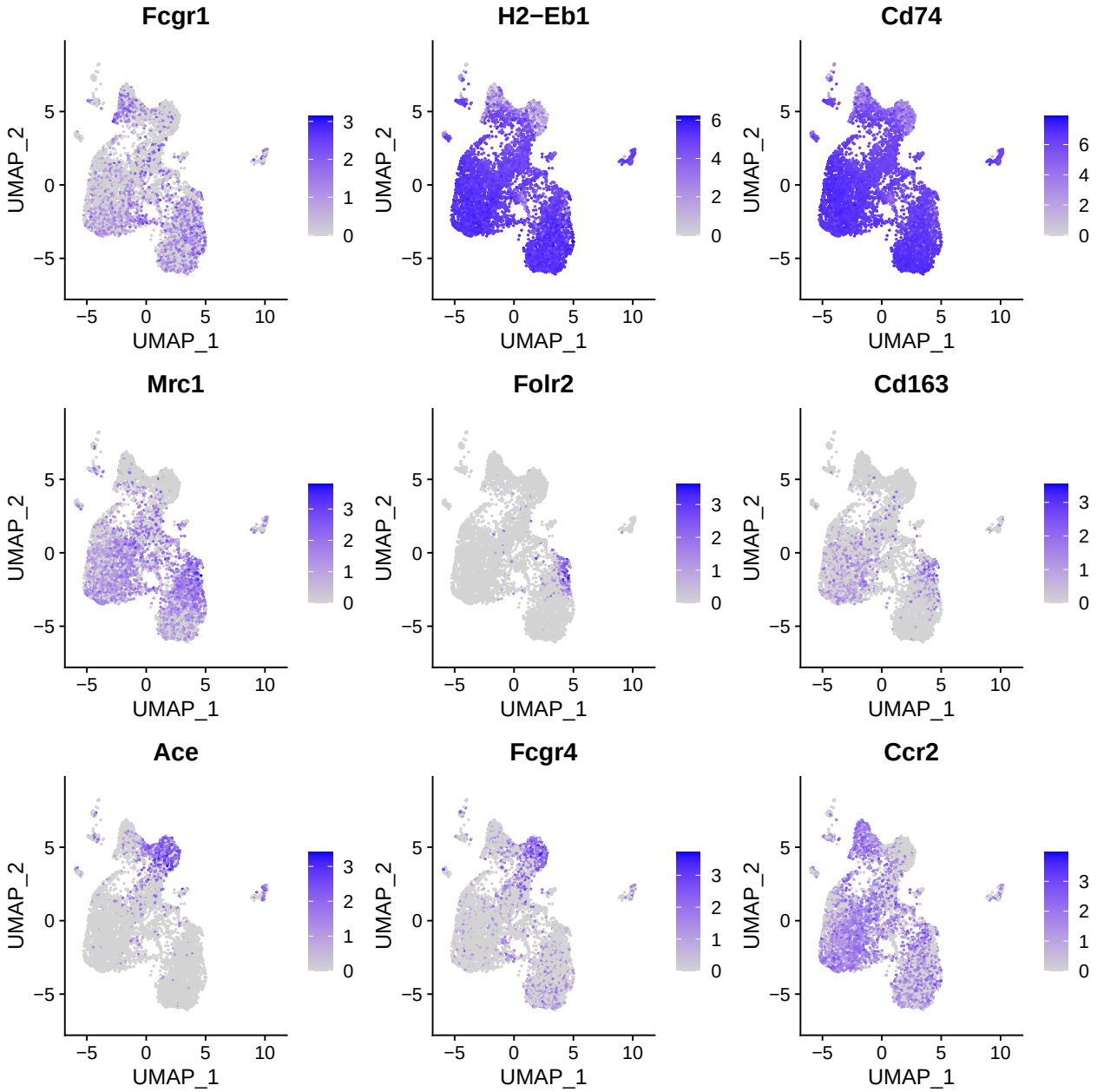
5 Remove contaminated cell types

5.1 Plot with key markers

```
library(ggplot2)
library(dplyr)
results <- IM_Maf.seuratObject
DimPlot(results, reduction = "umap", label = TRUE)
```




```
DefaultAssay(results) <- "RNA"
FeaturePlot(results, features = c("Fcgr1", "H2-Eb1", "Cd74", "Mrc1", "
   Folr2", "Cd163", "Ace", "Fcgr4", "Ccr2"), reduction = "umap")
```



5.2 Use results of SingleR celltyping to annotate cells

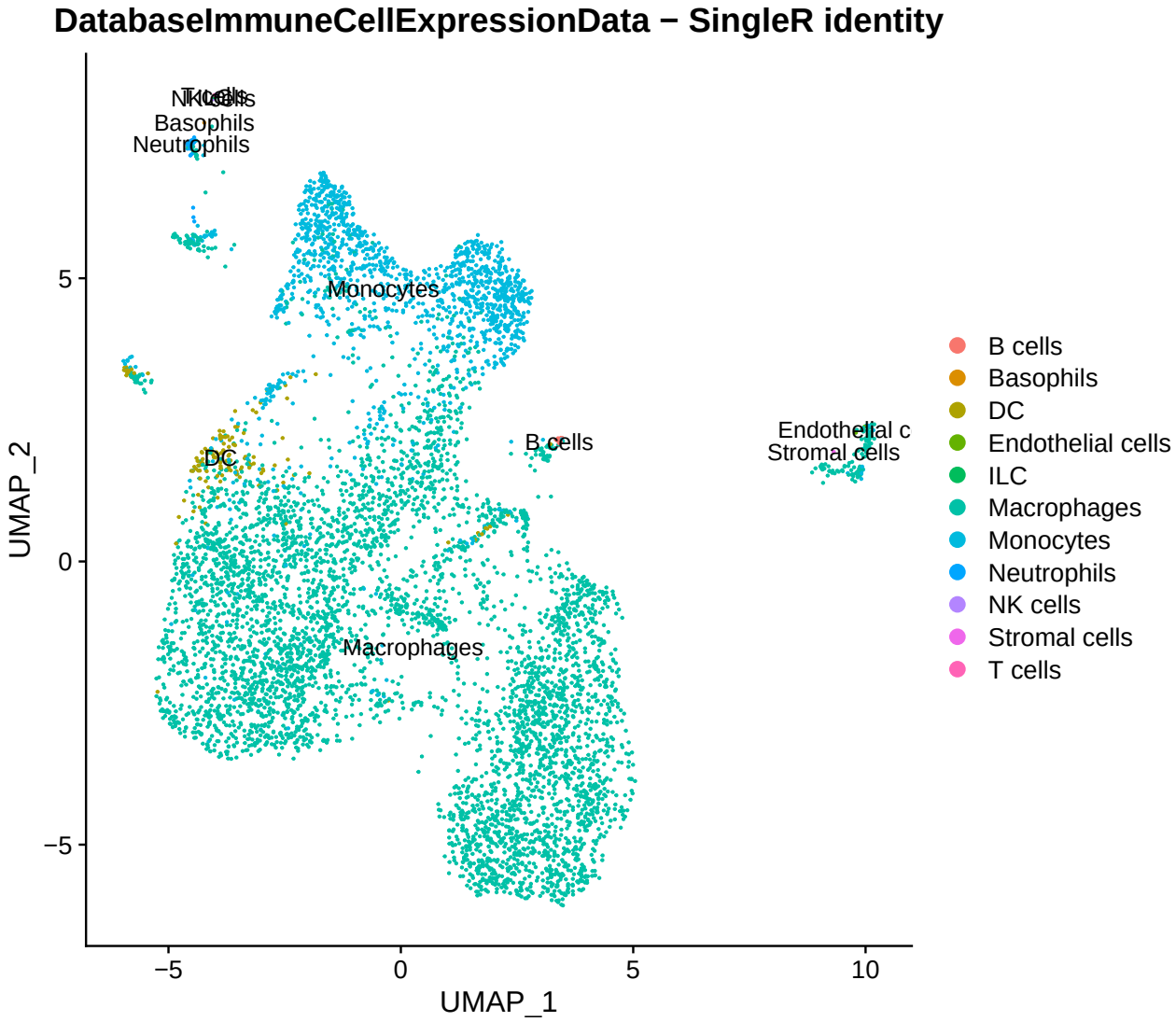
Check if SingleR results contain the same cells:

```
identical(colnames(results), rownames(results.singleR))
```

```
## [1] TRUE
```

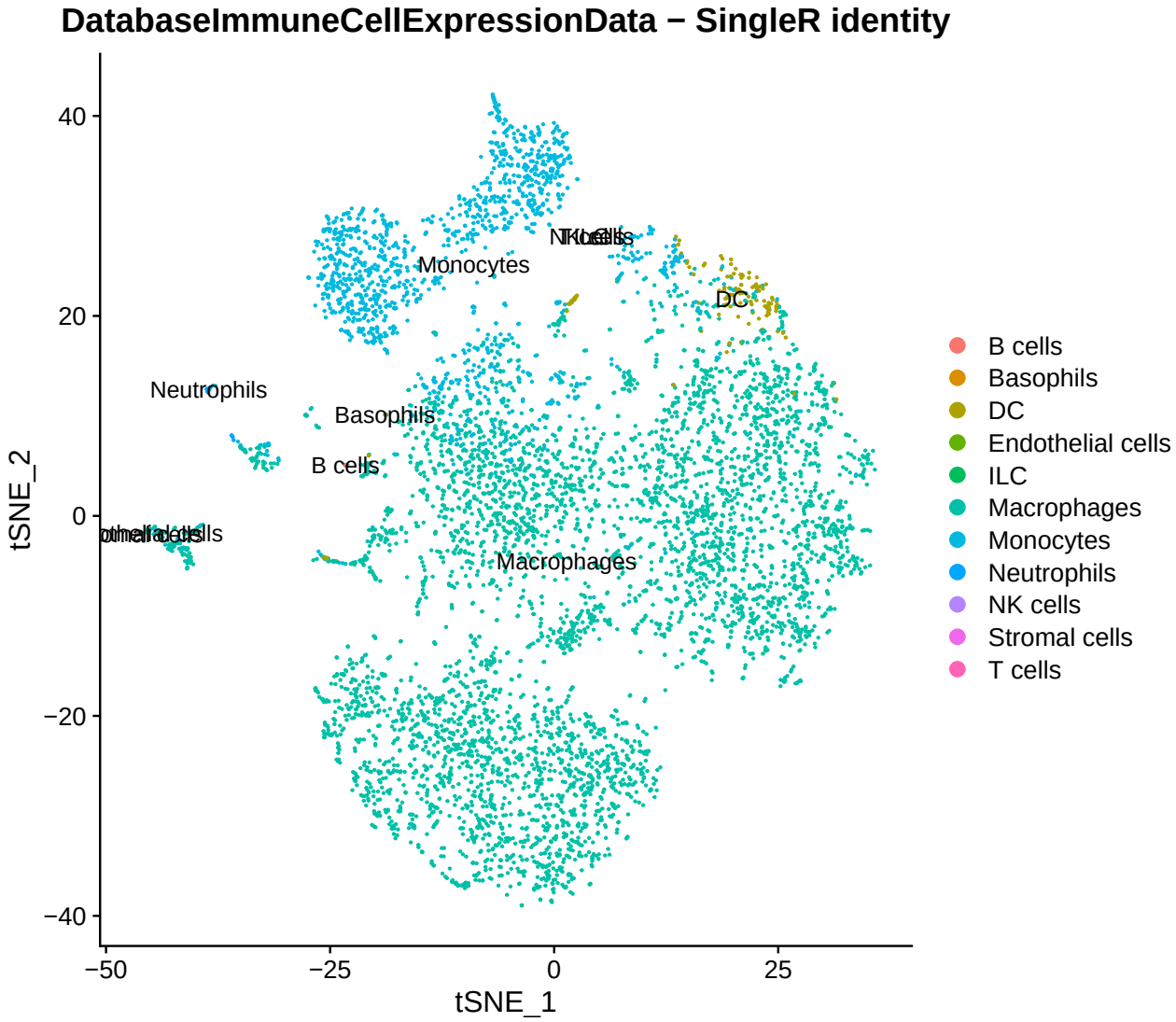
UMAP plot show cell types:

```
results$singleR.celltype <- results.singleR$labels

DimPlot(results, group.by = "singleR.celltype", reduction = "umap", label
    = T) + ggtitle("DatabaseImmuneCellExpressionData - SingleR identity")
```

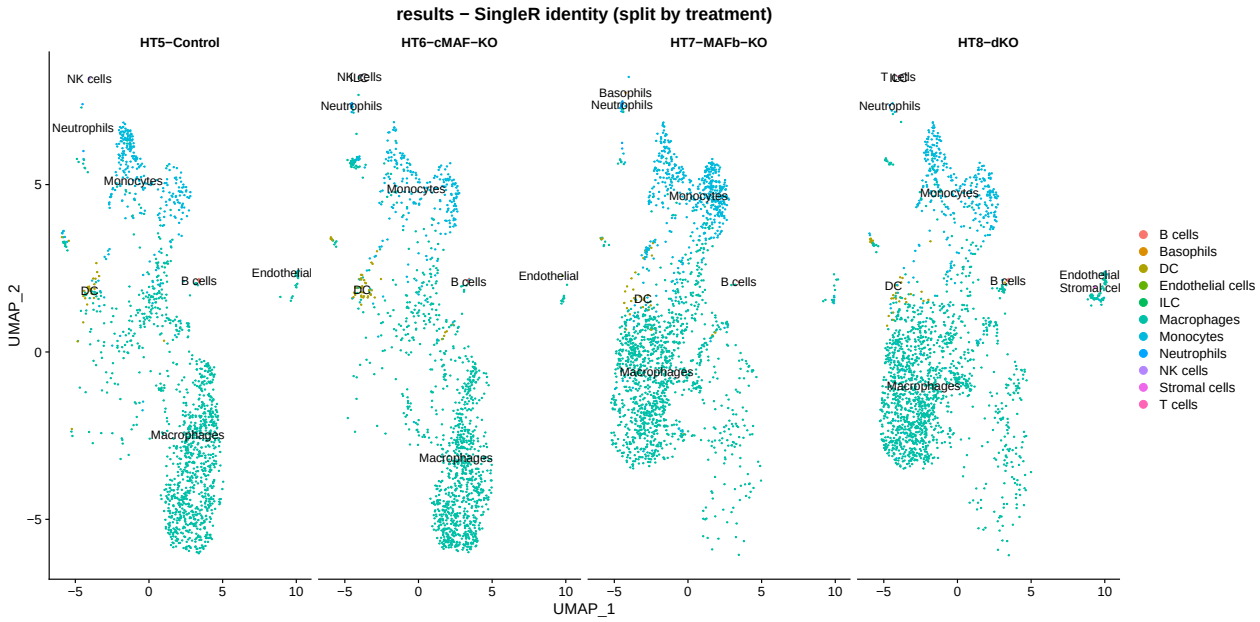


TSNE plot show cell types:

```
DimPlot(results, group.by = "singleR.celltype", reduction = "tsne", label
   = T) + ggtitle("DatabaseImmuneCellExpressionData - SingleR identity")
```




```
DimPlot(results, group.by = "singleR.celltype", reduction = "umap", label
   = T, split.by = "group") + ggtitle("results␣-␣SingleR␣identity␣(split␣
   by␣treatment)")
```



5.3 Remove contamination

Here's all the cell types and number:

```
table(results$singleR.celltype)
```

```
##
##          B cells         Basophils                DC Endothelial cells
##               13                 1               145                   3
##              ILC       Macrophages         Monocytes        Neutrophils
##                2              5049              1403                  30
##          NK cells     Stromal cells           T cells
##                4                 1                 1
```

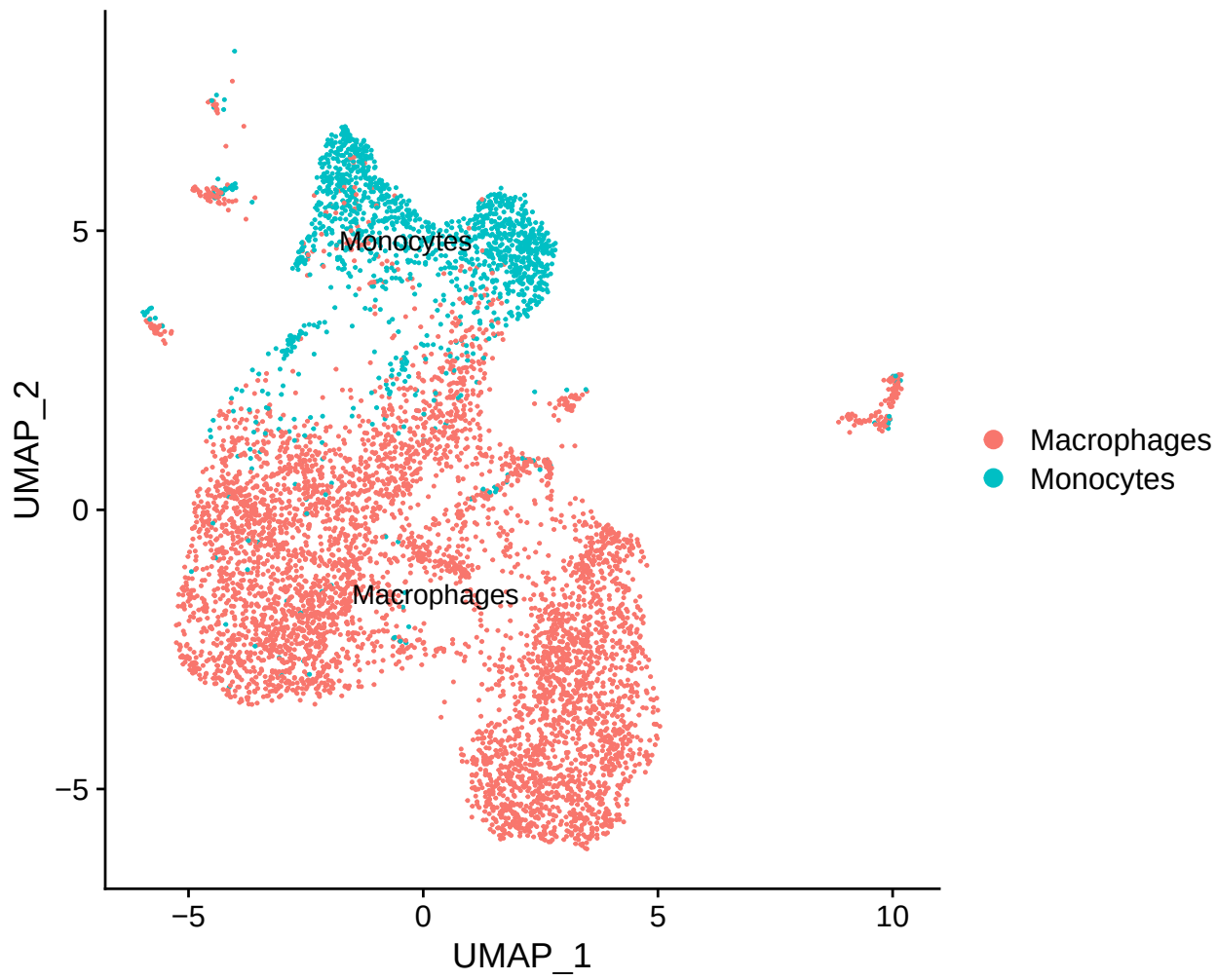
Remove all the other cell types and keep only monocytes and macrophages.

```
results <- subset(results, subset = singleR.celltype %in% c("B cells", "
    Basophils", "Endothelial cells", "Stromal cells", "Neutrophils", "NK
    cells", "DC", "ILC", "T cells"), invert = TRUE) # DC removed!!!
```

```
table(results$group)
```

```
##
## HT5-Control HT6-cMAF-KO HT7-MAFb-KO     HT8-dKO
##        1429        1340        1815        1868
```

```
DimPlot(results, group.by = "singleR.celltype", reduction = "umap", label
    = T) + ggtitle("DatabaseImmuneCellExpressionData - SingleR identity")
```



6 Reanalysis data after contamination removal

Remove old snn:

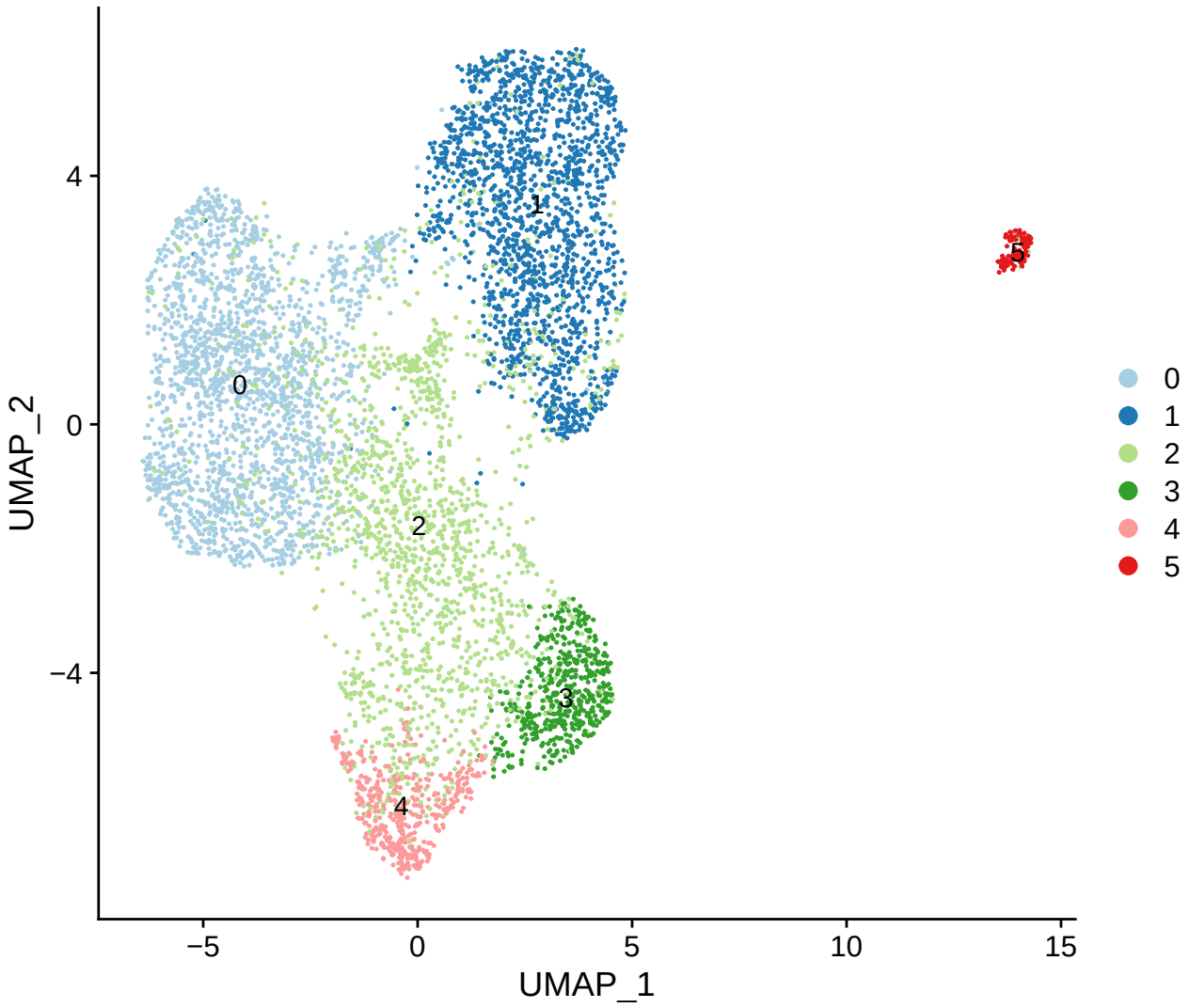
```
meta.names <- names(results@meta.data)
old.snn <- meta.names[startsWith(meta.names, "RNA_snn")]
for (i in old.snn) {
  results[[i]] <- NULL
}
```

```
# DefaultAssay(results) <- "RNA"
results <- NormalizeData(results, verbose=FALSE)
results <- FindVariableFeatures(results, selection.method = "vst",
    nfeatures = 2000, verbose=FALSE)
results <- ScaleData(results, features = rownames(results), verbose=FALSE)
results <- RunPCA(results, features = VariableFeatures(results), verbose=
    FALSE)
results <- RunTSNE(results, dims = 1:10, verbose=FALSE)
results <- RunUMAP(results, dims = 1:10, verbose=FALSE)
```

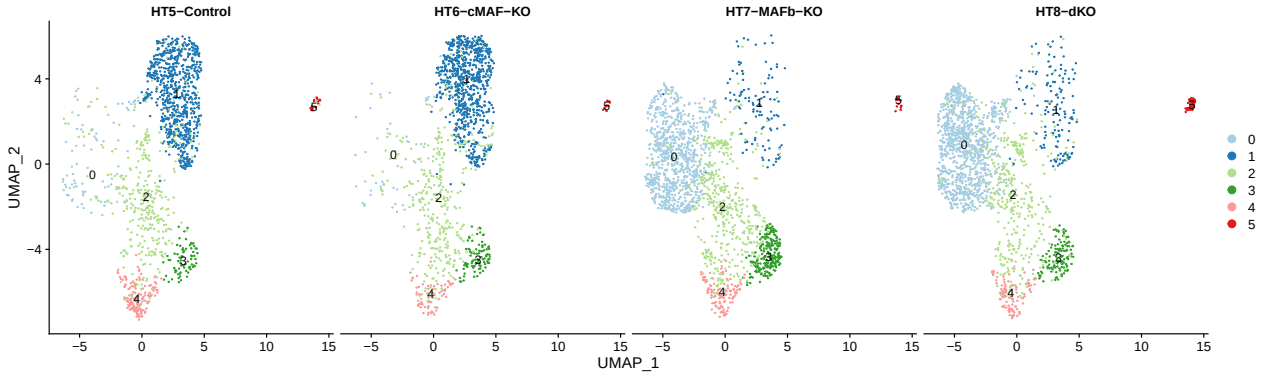
6.1 Cell clustering

```
results <- FindNeighbors(results, dims = 1:10, verbose = FALSE)
results <- FindClusters(results, resolution = 0.12, verbose = FALSE)
```

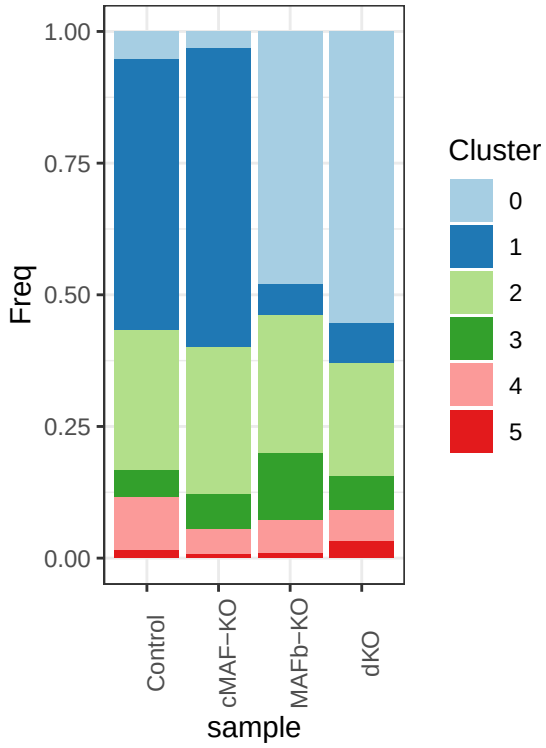
```
DimPlot(results, label = TRUE, cols = "Paired", reduction = "umap")
```




```
DimPlot(results, label = TRUE, split.by = "group", cols = "Paired")
```



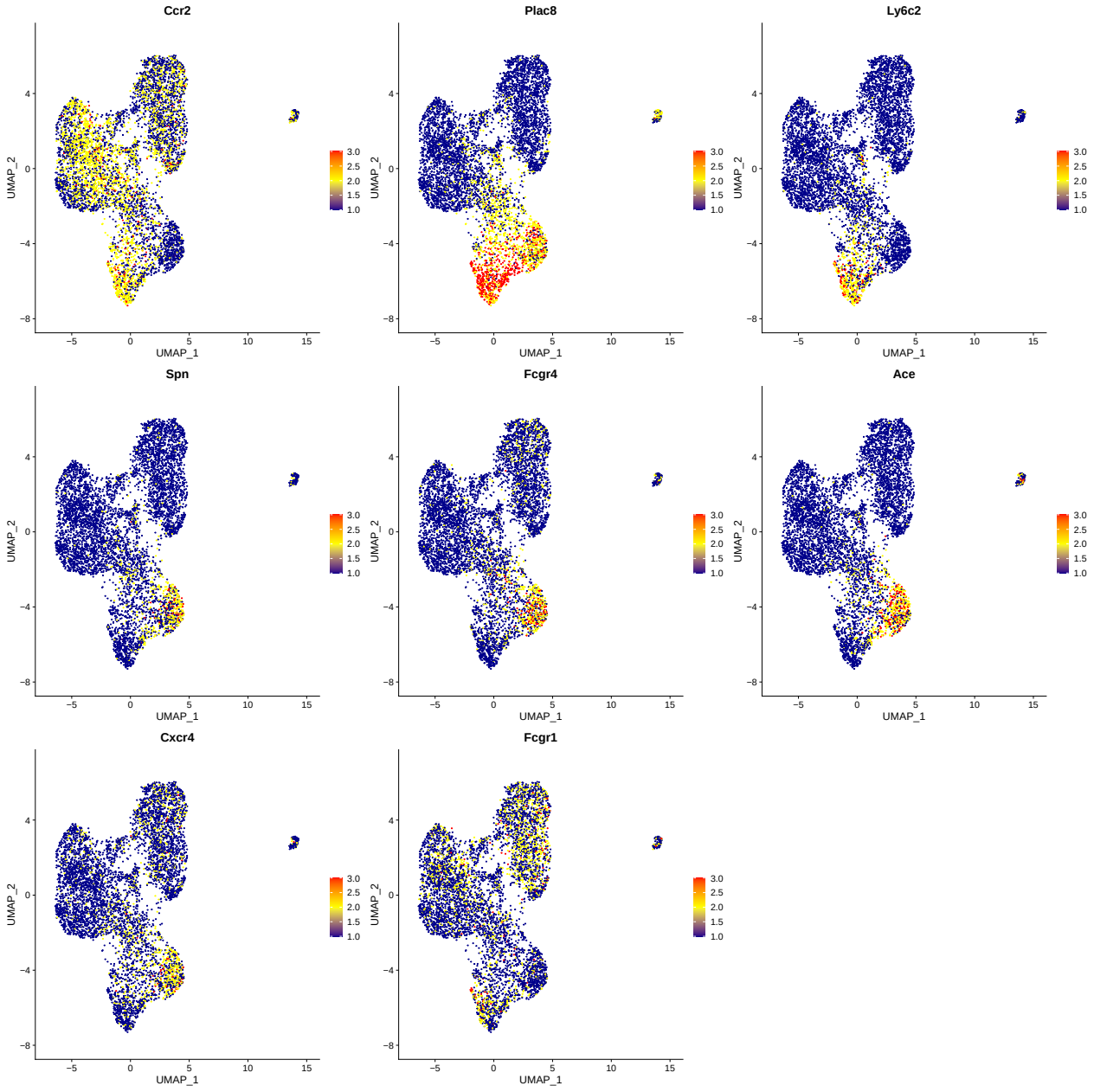

```
source("../R/SeuratFreqTable.R")
freq.celltype.list <- list(
  `Control` = Seurat2CellFreqTable(subset(results, subset = group == "HT5-
     Control"), slotName = "RNA_snn_res.0.12"),
  `cMAF-KO` = Seurat2CellFreqTable(subset(results, subset = group == "HT6-
     cMAF-KO"), slotName = "RNA_snn_res.0.12"),
  `MAFb-KO` = Seurat2CellFreqTable(subset(results, subset = group == "HT7-
     MAFb-KO"), slotName = "RNA_snn_res.0.12"),
  `dKO` = Seurat2CellFreqTable(subset(results, subset = group == "HT8-dKO"
     ), slotName = "RNA_snn_res.0.12")
)

source("../R/barChart.R")
barChart(freq.celltype.list) + labs(fill = "Cluster") + scale_fill_manual(
    values = brewer.pal(6, "Paired")) + theme(axis.text.x = element_text(
    angle = 90))
```



6.2 population characterization

6.2.1 Show expresssion of important monocyte markers

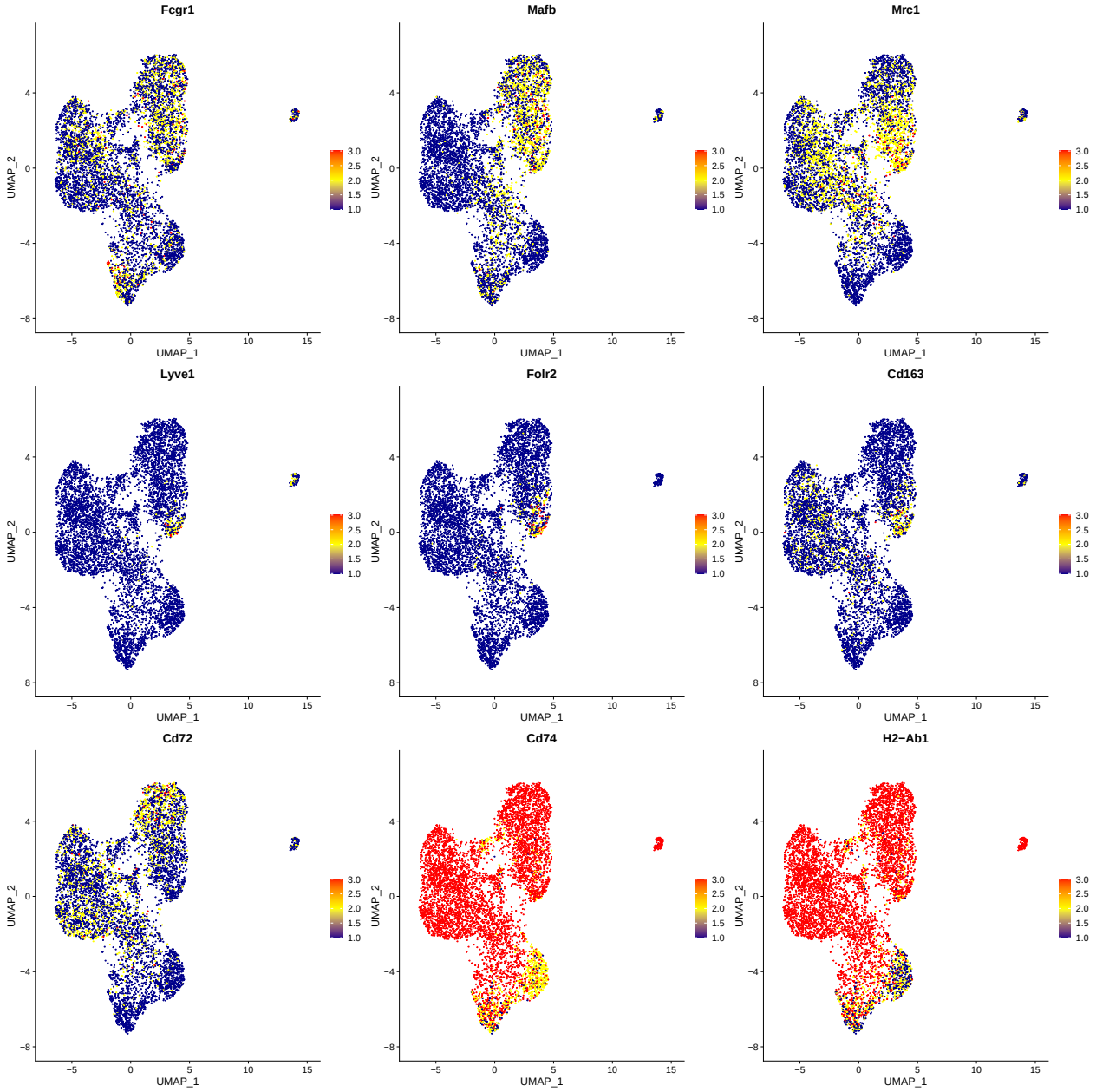
```
FeaturePlot(results, features = c("Ccr2", "Plac8" ,"Ly6c2", "Spn",
                                  "Fcgr4", "Ace", "Cxcr4",
                                  "Fcgr1") ,
          ncol = 3, reduction = "umap", cols = c("dark blue", "yellow", "
             red"))
```



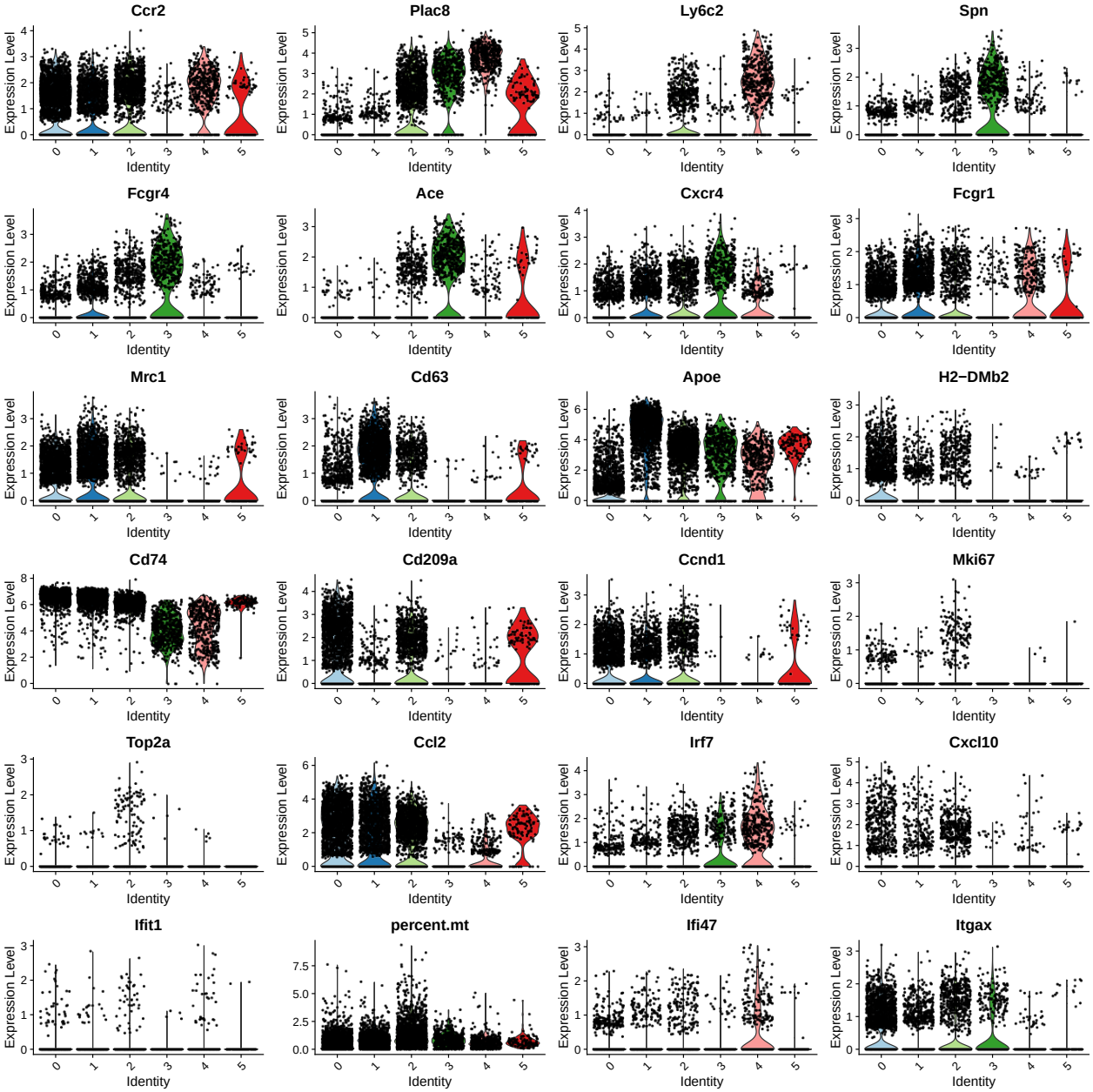
6.2.2 Show expresssion of important IM markers

```
FeaturePlot(results, features = c(
                                  "Fcgr1", "Mafb" ,"Mrc1", "Lyve1", "Folr2",
                                     "Cd163",
```

```
                                "Cd72", "Cd74", "H2-Ab1") ,
          ncol = 3, cols = c("dark blue", "yellow", "red"))
```



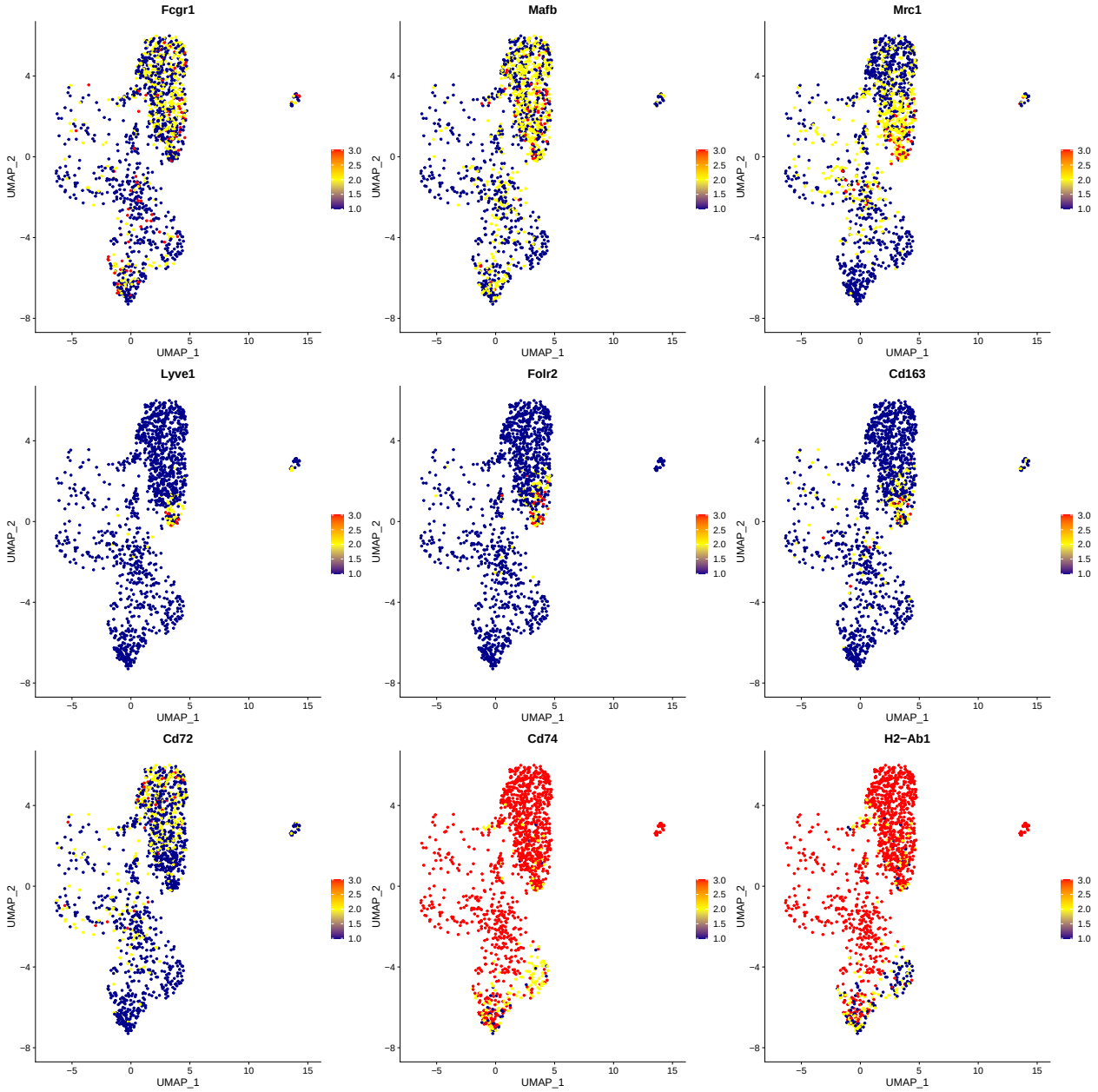

```
VlnPlot(results, features = c("Ccr2", "Plac8" ,"Ly6c2", "Spn",
                              "Fcgr4", "Ace", "Cxcr4",
                              "Fcgr1", "Mrc1", "Cd63", "Apoe",
                              "H2-DMb2", "Cd74", "Cd209a",
                              "Ccnd1","Mki67", "Top2a", "Ccl2",
                              "Irf7", "Cxcl10", "Ifit1", "percent.mt", "
                                 Ifi47", "Itgax") ,
        ncol = 4, cols = brewer.pal(6, "Paired"))
```



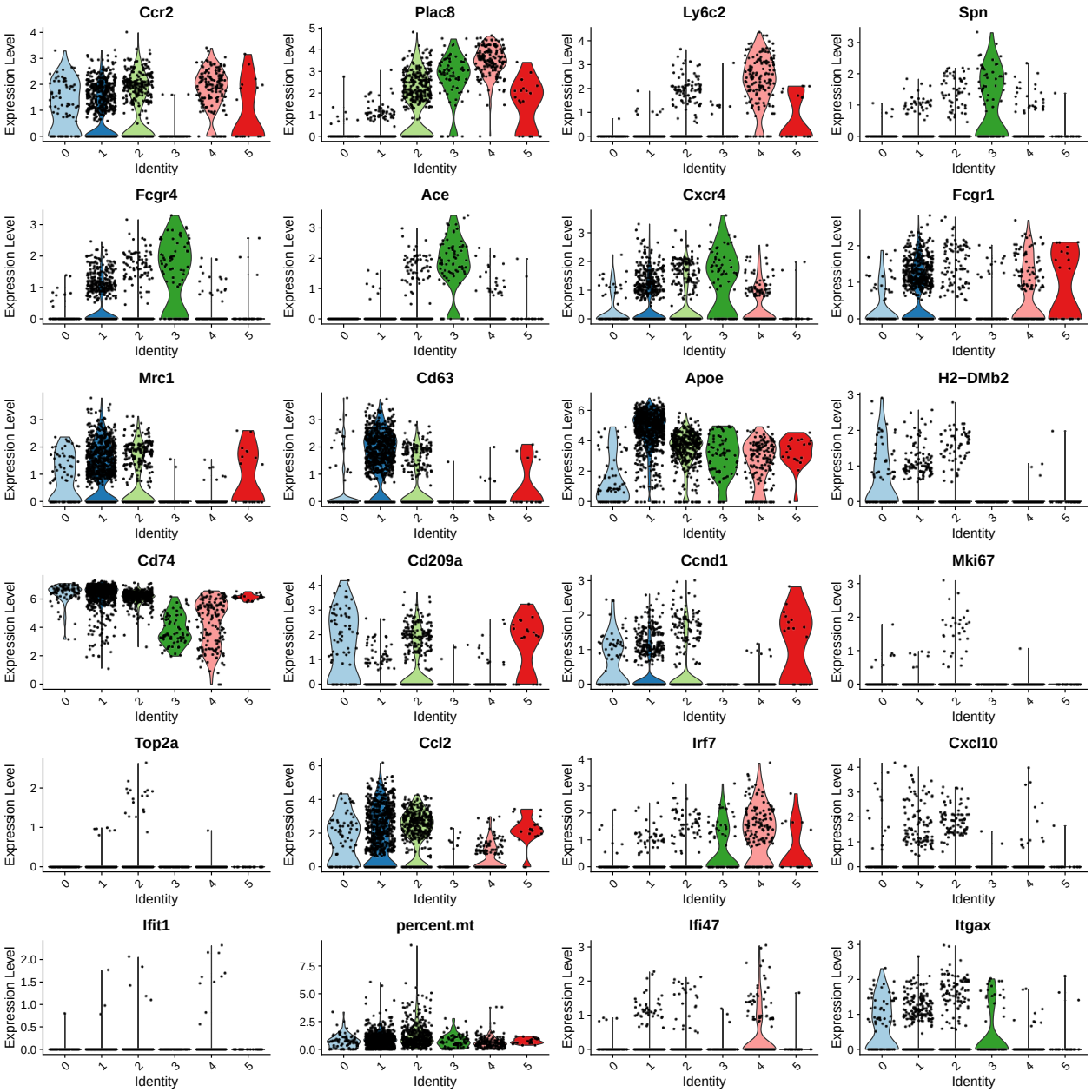
All the IMs are in cluster 1 while cluster 0 only presents Mafb- cells. To confirm we try to identify the populations only in control sample.

6.2.3 Show expresssion of important IM markers in Control sample

```
FeaturePlot(
    subset(results, subset = group == "HT5-Control"),
            features = c("Fcgr1", "Mafb" ,"Mrc1", "Lyve1", "Folr2", "Cd163
               ",
                                "Cd72", "Cd74", "H2-Ab1") ,
        ncol = 3, cols = c("dark blue", "yellow", "red"))
```




```
VlnPlot(
  subset(results, subset = group == "HT5-Control"),
  features = c("Ccr2", "Plac8" ,"Ly6c2", "Spn",
                           "Fcgr4", "Ace", "Cxcr4",
                           "Fcgr1", "Mrc1", "Cd63", "Apoe",
                           "H2-DMb2", "Cd74", "Cd209a",
                           "Ccnd1","Mki67", "Top2a", "Ccl2",
                           "Irf7", "Cxcl10", "Ifit1", "percent.mt", "
                              Ifi47", "Itgax") ,
        ncol = 4, cols = brewer.pal(6, "Paired"))
```



6.2.4 Subset charaterization

cluster 0: Mafb- trapped cluster 1: All IMs (both CD206+ and CD206- IM) cluster 2: Intermediate cluster 3: Patrolling Mono cluster 4: Classical Mono cluster 5: unknown

6.3 Clustering and Annotate CD206+ and CD206- IMs

As Mafb-deficiency introduced a bigger variance which made CD206+/CD206- IMs unable to be clustered separately. We will isolate cluster 1 (contains all IMs) and redo cluster.

```
ims <- subset(results, idents = "1")
```

Normalize and find variable genes

```
ims <- NormalizeData(ims, verbose=FALSE)
```

```
ims <- FindVariableFeatures(ims, selection.method = "vst", nfeatures =
    2000, verbose=FALSE)
ims <- ScaleData(ims, features = rownames(ims), verbose=FALSE)
ims <- RunPCA(ims, features = VariableFeatures(ims), verbose=FALSE)
ims <- RunTSNE(ims, dims = 1:5, verbose=FALSE)
ims <- RunUMAP(ims, dims = 1:5, verbose=FALSE)
```

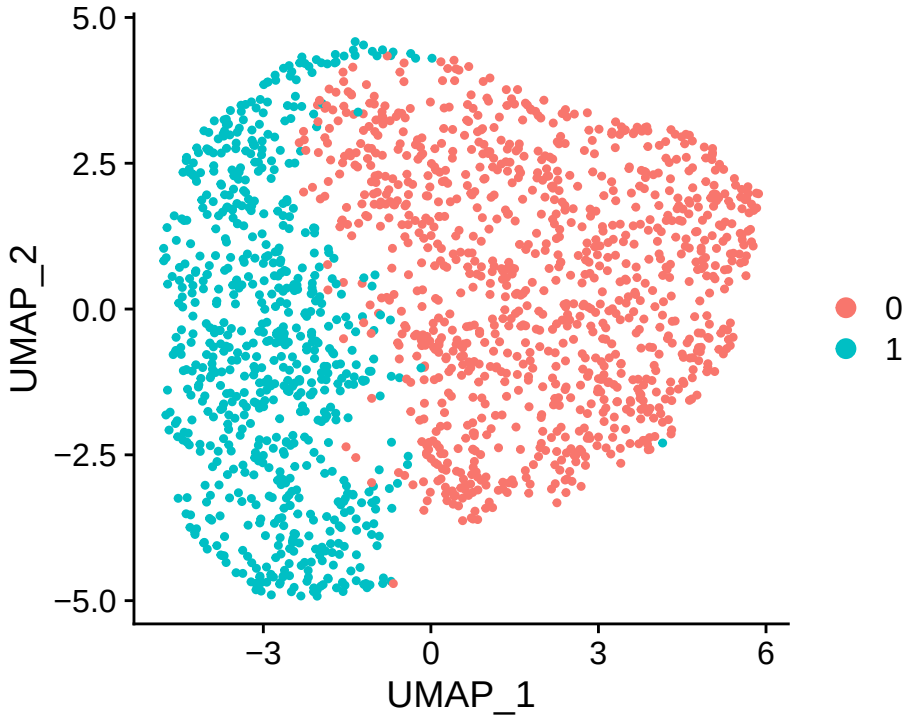
Cell clustering within cluster 1

```
ims <- FindNeighbors(ims, dims = 1:5, verbose = FALSE)
ims <- FindClusters(ims, resolution = 0.2, verbose = FALSE)
```

Show CD206+ and CD206- IMs:

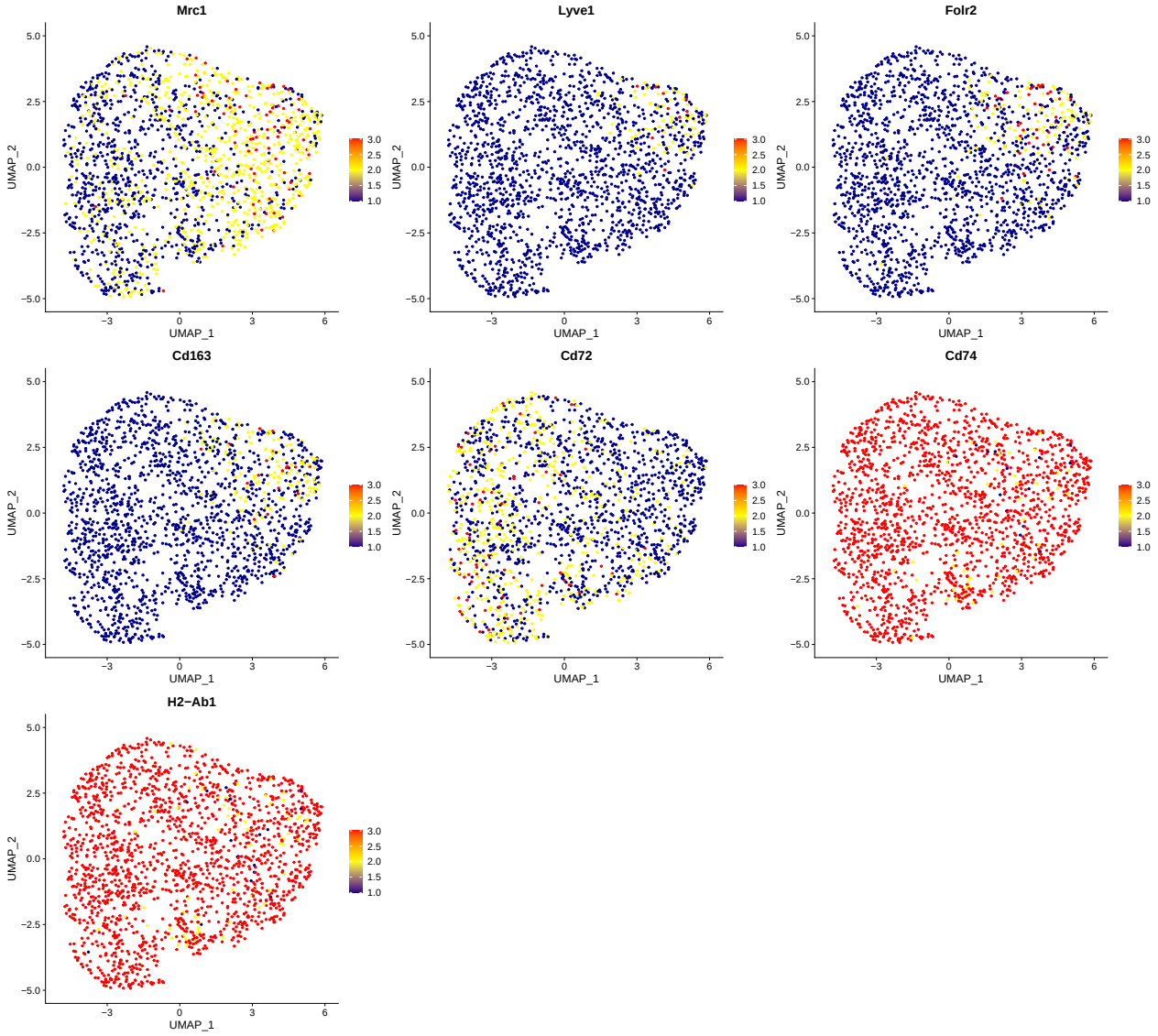
Clusters:

```
DimPlot(ims)
```



Show marker expression:

```
FeaturePlot(ims,
            features = c("Mrc1", "Lyve1", "Folr2", "Cd163",
                         "Cd72", "Cd74", "H2-Ab1") ,
        ncol = 3, cols = c("dark␣blue", "yellow", "red"))
```



Within the cluster1: subcluster 0: CD206+ IMs subcluster 1: CD206- (MHCII hi) IMs

6.4 Annotate with cell types

```
results$cell.type2 <- factor(Idents(results), labels = c("Mafb-deficient",
    "IM", "Intermediate", "Patrolling Mono", "Classical Mono", "Unknown"))

results$cell.type2 <- as.character(results$cell.type2)
```

Overide IM cluster with subcluster annotatinons in new cell.type3 annotation.

```
results$cell.type3 <- results$cell.type2

# overide
results$cell.type3[colnames(ims)] <- as.character(factor(Idents(ims),
   labels = c("CD206+ IMs", "CD206- IMs")))

# Now make annotation into proper factors:
```

```
results$cell.type2 <- as.factor(results$cell.type2)
results$cell.type3 <- factor(results$cell.type3, levels = c("Classical Mono",
                                                            "Patrolling Mono",
                                                            "Intermediate",
                                                            "CD206- IMs",
                                                            "CD206+ IMs",
                                                            "Mafb-deficient",
                                                            "Unknown"))

Idents(results) <- "cell.type3"
levels(results)
```

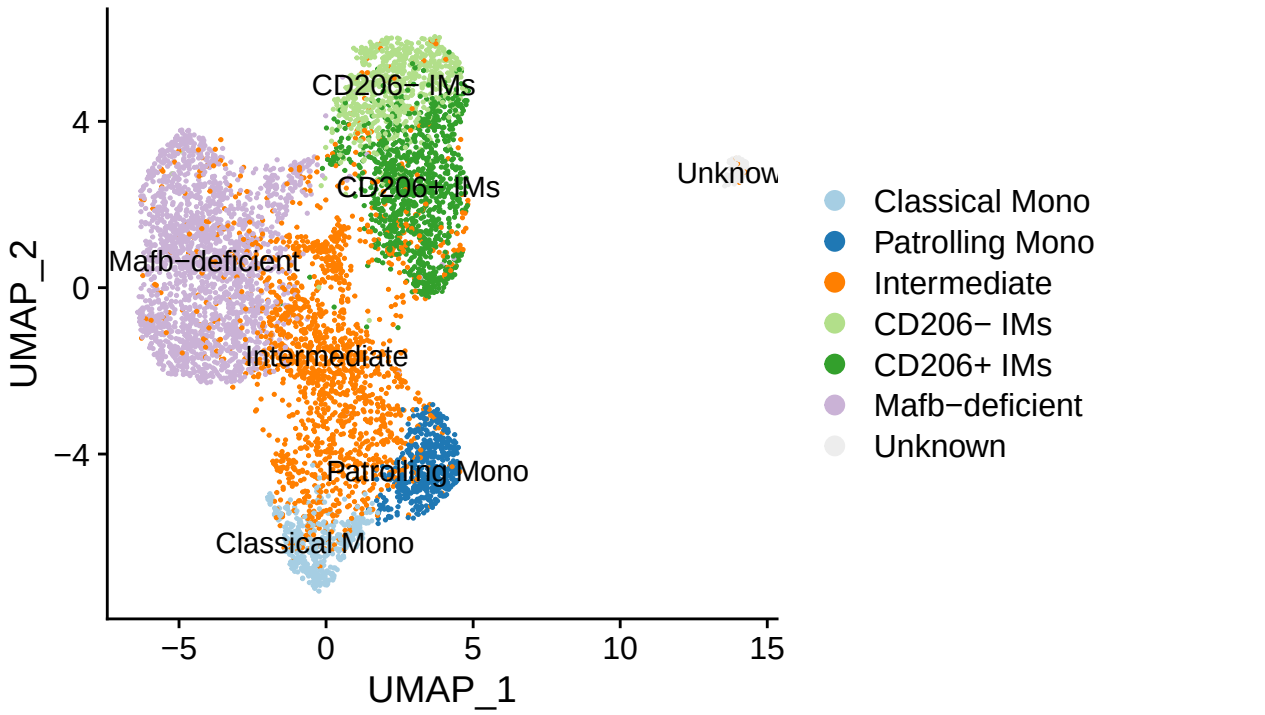
```
## [1] "Classical Mono"  "Patrolling Mono" "Intermediate"    "CD206- IMs"
## [5] "CD206+ IMs"      "Mafb-deficient"  "Unknown"
```

6.5 Make plots with annotated cells

Make color palette for plot (3 new colors were Paired number 8 and 9 from RColorBrewer, and light grey for unknown population):

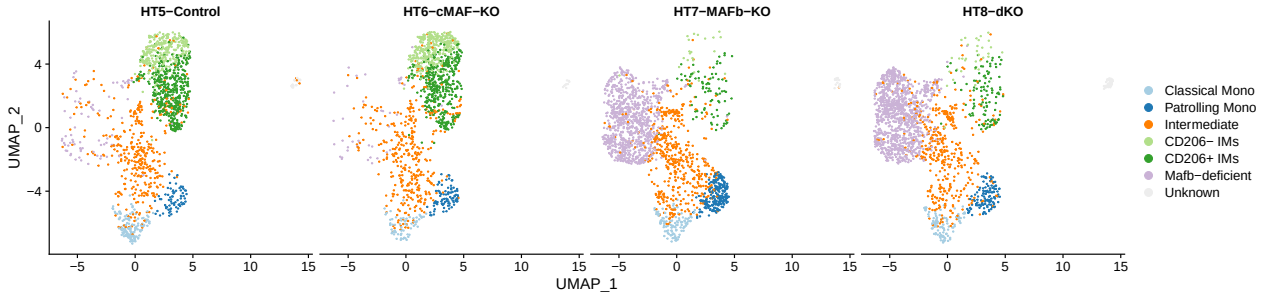
```
pal3 <- c(
  "#A6CEE3", # cMo
  "#1F78B4", # pMo
  "#FF7F00", # Intermediate
  "#B2DF8A", # MHCII IM
  "#33A02C", # CD206 IM
  "#CAB2D6", # Mafb- neo
  "#ededed"# Unknown
)
```

```
DimPlot(results, label = TRUE, cols = pal3)
```




```
ggsave(filename = "../Figures/UMAPplot_All_samplesMaf_label.pdf", width =
    6, height = 4)
```

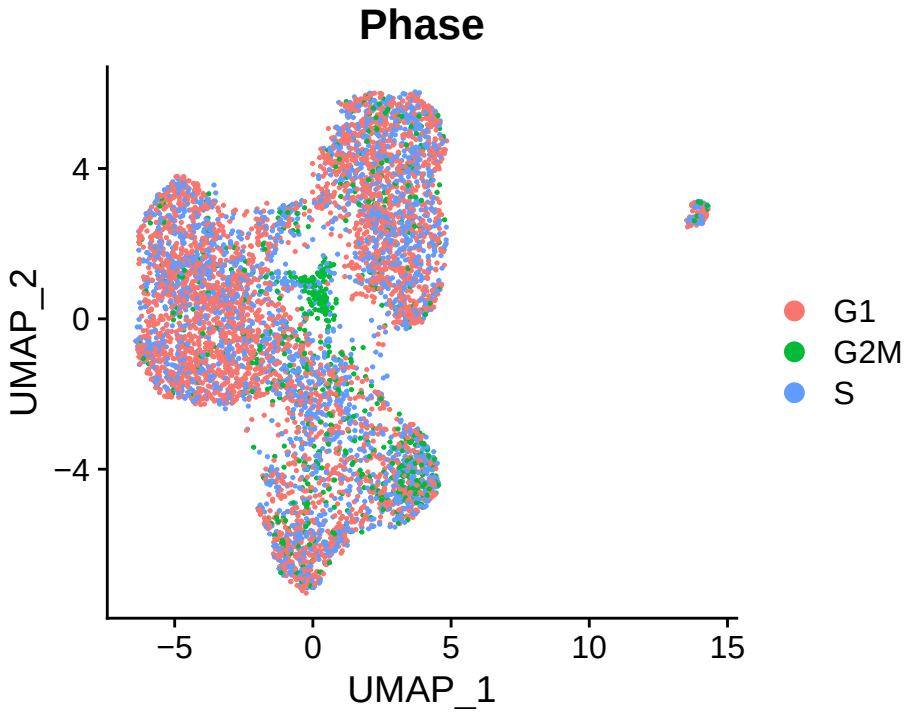
```
DimPlot(results, cols = pal3, split.by = "group")
```




```
ggsave(filename = "../Figures/UMAPplot_all_separate_samplesMaf.pdf", width
    = 16, height = 4)
```

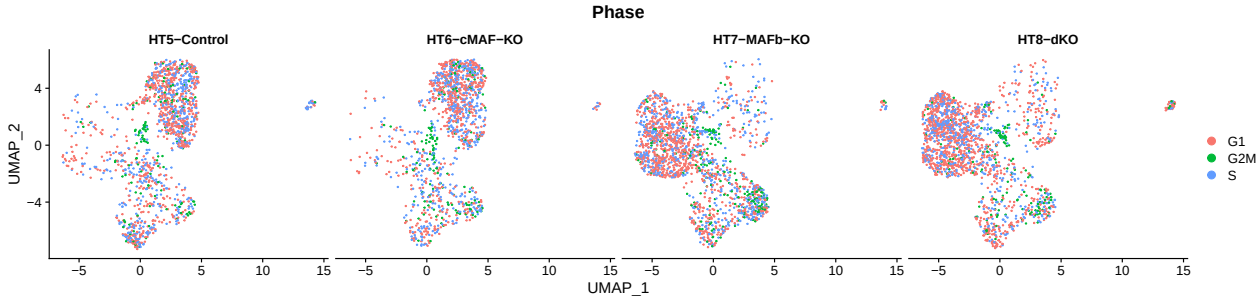
6.6 Cell cycle analysis

```
library(cowplot)
data("geneinfo_human", package = "nichenetr")
s.genes <- nichenetr::convert_human_to_mouse_symbols(cc.genes.updated.2019
    $s.genes)
g2m.genes <- nichenetr::convert_human_to_mouse_symbols(cc.genes.updated
    .2019$g2m.genes)
results <- CellCycleScoring(results, s.features = s.genes, g2m.features =
    g2m.genes, set.ident = FALSE)
DimPlot(results, group.by = "Phase")
```



We see a clear cycling core in Intermediate population.

```
DimPlot(results, group.by = "Phase", reduction = "umap", split.by = "group
    ")
```




```
ggsave(filename = "../Figures/UMAPplot_all_separate_samplesMaf_Phase.pdf",
    width = 16, height = 4)
```

```
saveRDS(results, file = "./All_samples_Maf.seuratObject.Rds")
```

7 Session information

R sesssion:

```
sessionInfo()
```

```
## R version 4.0.3 (2020-10-10)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Ubuntu 20.04.3 LTS
##
```

```
## Matrix products: default
## BLAS:   /usr/lib/x86_64-linux-gnu/openblas-pthread/libblas.so.3
## LAPACK: /usr/lib/x86_64-linux-gnu/openblas-pthread/liblapack.so.3
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_GB.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_GB.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_GB.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_GB.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] parallel  stats4    stats     graphics  grDevices utils
   datasets
## [8] methods   base
##
## other attached packages:
## [1] cowplot_1.1.1       RColorBrewer_1.1-2  S4Vectors_0.28.1
## [4] BiocGenerics_0.36.1 dplyr_1.0.8         ggpubr_0.4.0
## [7] ggplot2_3.3.5       SeuratObject_4.0.4  Seurat_4.1.0
##
## loaded via a namespace (and not attached):
##   [1] utf8_1.2.2            reticulate_1.24       tidyselect_1.1.1
##   [4] htmlwidgets_1.5.4     grid_4.0.3            Rtsne_0.15
##   [7] pROC_1.18.0           munsell_0.5.0         codetools_0.2-18
##  [10] ragg_1.2.2            ica_1.0-2             future_1.24.0
##  [13] miniUI_0.1.1.1        withr_2.4.3           spatstat.random_2.1-0
##  [16] colorspace_2.0-3      highr_0.9             knitr_1.37
##  [19] rstudioapi_0.13       ROCR_1.0-11           ggsignif_0.6.3
##  [22] tensor_1.5            listenv_0.8.0         labeling_0.4.2
##  [25] polyclip_1.10-0       farver_2.1.0          parallelly_1.30.0
##  [28] vctrs_0.3.8           generics_0.1.2        ipred_0.9-12
##  [31] xfun_0.29             randomForest_4.6-14   R6_2.5.1
##  [34] ggbeeswarm_0.6.0      bitops_1.0-7          spatstat.utils_2.3-0
##  [37] assertthat_0.2.1      promises_1.2.0.1      scales_1.1.1
##  [40] nnet_7.3-14           beeswarm_0.4.0        gtable_0.3.0
##  [43] Cairo_1.5-14          globals_0.14.0        goftest_1.2-3
##  [46] timeDate_3043.102     rlang_1.0.1           systemfonts_1.0.4
##  [49] splines_4.0.3         rstatix_0.7.0         lazyeval_0.2.2
##  [52] ModelMetrics_1.2.2.2  checkmate_2.0.0       spatstat.geom_2.3-2
##  [55] broom_0.7.12          yaml_2.3.5            reshape2_1.4.4
##  [58] abind_1.4-5           backports_1.4.1       httpuv_1.6.5
##  [61] Hmisc_4.6-0           caret_6.0-90          DiagrammeR_1.0.8
##  [64] tools_4.0.3           lava_1.6.10           ellipsis_0.3.2
##  [67] spatstat.core_2.4-0   proxy_0.4-26          ggridges_0.5.3
##  [70] Rcpp_1.0.8            plyr_1.8.6            base64enc_0.1-3
##  [73] visNetwork_2.1.0      purrr_0.3.4           rpart_4.1-15
##  [76] deldir_1.0-6          pbapply_1.5-0         zoo_1.8-9
##  [79] nichenetr_1.0.0       ggrepel_0.9.1         cluster_2.1.0
##  [82] magrittr_2.0.2        data.table_1.14.2     RSpectra_0.16-0
##  [85] scattermore_0.8       lmtest_0.9-39         RANN_2.6.1
##  [88] fitdistrplus_1.1-6    matrixStats_0.61.0    hms_1.1.1
##  [91] patchwork_1.1.1       mime_0.12             evaluate_0.15
```

```
##  [94] xtable_1.8-4          jpeg_0.1-9             gridExtra_2.3
##  [97] compiler_4.0.3        tibble_3.1.6           KernSmooth_2.23-20
## [100] crayon_1.5.0          htmltools_0.5.2        mgcv_1.8-33
## [103] later_1.3.0           tzdb_0.2.0             Formula_1.2-4
## [106] tidyr_1.2.0           lubridate_1.8.0        DBI_1.1.2
## [109] MASS_7.3-53           Matrix_1.4-0           car_3.0-12
## [112] readr_2.1.2           cli_3.2.0              gower_1.0.0
## [115] igraph_1.2.11         pkgconfig_2.0.3        foreign_0.8-82
## [118] plotly_4.10.0         spatstat.sparse_2.1-0  recipes_0.2.0
## [121] foreach_1.5.2         vipor_0.4.5            hardhat_0.2.0
## [124] prodlim_2019.11.13    stringr_1.4.0          digest_0.6.29
## [127] sctransform_0.3.3     RcppAnnoy_0.0.19       spatstat.data_2.1-2
## [130] rmarkdown_2.11        leiden_0.3.9           htmlTable_2.4.0
## [133] uwot_0.1.11           shiny_1.7.1            lifecycle_1.0.1
## [136] nlme_3.1-155          jsonlite_1.7.3         carData_3.0-5
## [139] limma_3.46.0          viridisLite_0.4.0      fansi_1.0.2
## [142] pillar_1.7.0          lattice_0.20-41        ggrastr_1.0.1
## [145] fastmap_1.1.0         httr_1.4.2             survival_3.2-7
## [148] glue_1.6.1            fdrtool_1.2.17         png_0.1-7
## [151] iterators_1.0.14      class_7.3-17           stringi_1.7.6
## [154] textshaping_0.3.6     caTools_1.18.2         latticeExtra_0.6-29
## [157] irlba_2.3.5           e1071_1.7-9            future.apply_1.8.1
```