

Seurat

2025-04-04

Contents

Setup the Seurat Object	1
What does data in a count matrix look like?	2
Standard pre-processing workflow	3
QC and selecting cells for further analysis	3
Normalizing the data	6
Apply sctransform normalization	6
Identification of highly variable features (feature selection)	6
Scaling the data	7
Perform linear dimensional reduction	8
Determine the ‘dimensionality’ of the dataset	10
Cluster the cells	11
Run non-linear dimensional reduction (UMAP/tSNE)	12
Finding differentially expressed features (cluster biomarkers)	14
Visualizations of marker feature expression	16
Additions to FeaturePlot	21
Applying themes to plots	26
Assigning cell type identity to clusters	30

Setup the Seurat Object

For this tutorial, we will be analyzing the a dataset of Peripheral Blood Mononuclear Cells (PBMC) freely available from 10X Genomics. There are 2,700 single cells that were sequenced on the Illumina NextSeq 500. The raw data can be found: https://cf.10xgenomics.com/samples/cell/pbmc3k/pbmc3k_filtered_gene_bc_matrices.tar.gz

We start by reading in the data. The `Read10X()` function reads in the output of the cellranger pipeline from 10X, returning a unique molecular identified (UMI) count matrix. The values in this matrix represent the number of molecules for each feature (i.e. gene; row) that are detected in each cell (column). Note that more recent versions of cellranger now also output using the h5 file format, which can be read in using the `Read10X_h5()` function in Seurat.

We next use the count matrix to create a Seurat object. The object serves as a container that contains both data (like the count matrix) and analysis (like PCA, or clustering results) for a single-cell dataset. For example, in Seurat v5, the count matrix is stored in `pbmc[["RNA"]]$counts`.

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
## The following objects are masked from 'package:stats':
##
##   filter, lag
## The following objects are masked from 'package:base':
##
##   intersect, setdiff, setequal, union
```

```
library(Seurat)
```

```
## Loading required package: SeuratObject
## Loading required package: sp
##
## Attaching package: 'SeuratObject'
## The following objects are masked from 'package:base':
##
##   intersect, t
```

```
library(patchwork)
```

```
library(ggplot2)
```

```
library(ggmin)
```

```
# Load the PBMC dataset
```

```
# IMPORTANT: Make sure this Rmd file is saved in the same folder as the "filtered_gene_bc_matrices" dir
```

```
pbmc.data <- Read10X(data.dir = "filtered_gene_bc_matrices/hg19/")
```

```
# Initialize the Seurat object with the raw (non-normalized data).
```

```
pbmc <- CreateSeuratObject(counts = pbmc.data, project = "pbmc3k", min.cells = 3, min.features = 200)
```

```
## Warning: Feature names cannot have underscores ('_'), replacing with dashes
```

```
## ('-')
```

```
pbmc
```

```
## An object of class Seurat
## 13714 features across 2700 samples within 1 assay
## Active assay: RNA (13714 features, 0 variable features)
## 1 layer present: counts
```

What does data in a count matrix look like?

```
# Lets examine a few genes in the first thirty cells
```

```
pbmc.data[c("CD3D", "TCL1A", "MS4A1"), 1:30]
```

```
## 3 x 30 sparse Matrix of class "dgCMatrix"
```

```
## [[ suppressing 30 column names 'AAACATACAACCAC-1', 'AAACATTGAGCTAC-1', 'AAACATTGATCAGC-1' ... ]]
```

```
##
```

```
## CD3D 4 . 10 . . 1 2 3 1 . . 2 7 1 . . 1 3 . 2 3 . . . . 3 4 1 5
```

```
## TCL1A . . . . . . . 1 . . . . . . . . . . 1 . . . . . . .
```

```
## MS4A1 . 6 . . . . . 1 1 1 . . . . . . . . 36 1 2 . . 2 . . . .
```

The . values in the matrix represent 0s (no molecules detected). Since most values in an scRNA-seq matrix are 0, Seurat uses a sparse-matrix representation whenever possible. This results in significant memory and speed savings for Drop-seq/inDrop/10x data.

Standard pre-processing workflow

The steps below encompass the standard pre-processing workflow for scRNA-seq data in Seurat. These represent the selection and filtration of cells based on QC metrics, data normalization and scaling, and the detection of highly variable features.

QC and selecting cells for further analysis

Seurat allows you to easily explore QC metrics and filter cells based on any user-defined criteria. A few QC metrics commonly used by the community include:

- The number of unique genes detected in each cell.
 - Low-quality cells or empty droplets will often have very few genes
 - Cell doublets or multiplets may exhibit an aberrantly high gene count
- Similarly, the total number of molecules detected within a cell (correlates strongly with unique genes)
- The percentage of reads that map to the mitochondrial genome
 - Low-quality / dying cells often exhibit extensive mitochondrial contamination
 - We calculate mitochondrial QC metrics with the `PercentageFeatureSet()` function, which calculates the percentage of counts originating from a set of features
 - We use the set of all genes starting with MT- as a set of mitochondrial genes

```
# The [[ operator can add columns to object metadata. This is a great place to stash QC stats
pbmc[["percent.mt"]] <- PercentageFeatureSet(pbmc, pattern = "^MT-")
```

Where are QC metrics stored in Seurat?

- The number of unique genes and total molecules are automatically calculated during `CreateSeuratObject()`
 - You can find them stored in the object meta data

```
# Show QC metrics for the first 5 cells
head(pbmc@meta.data, 5)
```

```
##          orig.ident nCount_RNA nFeature_RNA percent.mt
## AAACATACAACCAC-1    pbmc3k      2419         779  3.0177759
## AAACATTGAGCTAC-1    pbmc3k      4903        1352  3.7935958
## AAACATTGATCAGC-1    pbmc3k      3147        1129  0.8897363
## AAACCGTGCTTCCG-1    pbmc3k      2639         960  1.7430845
## AAACCGTGTATGCG-1    pbmc3k       980         521  1.2244898
```

- `orig.ident`: The original identity or sample label for the cells.
- `nCount_RNA`: The total number of RNA molecules (UMIs) detected in each cell.
- `nFeature_RNA`: The number of genes detected in that same cell.
- `percent.mt`: The percentage of reads mapping to mitochondrial genes.

In the example below, we visualize QC metrics, and use these to filter cells.

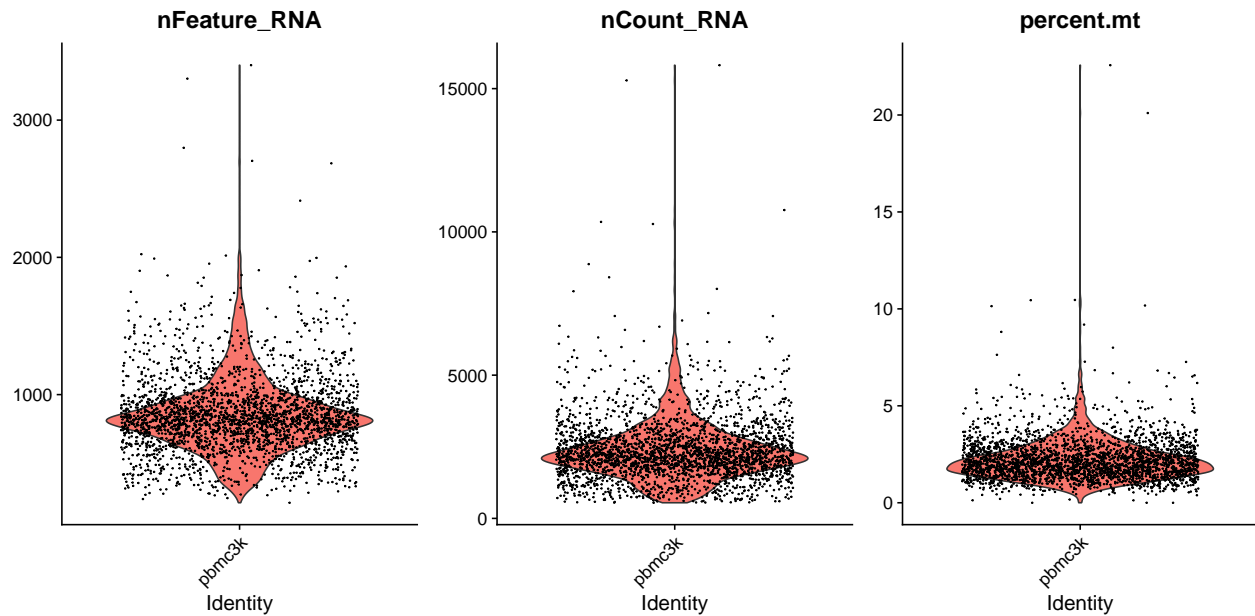
- We filter cells that have unique feature counts over 2,500 or less than 200
- We filter cells that have >5% mitochondrial counts

```
# Visualize QC metrics as a violin plot
```

```
plot_before_QC = VlnPlot(pbmc, features = c("nFeature_RNA", "nCount_RNA", "percent.mt"), ncol = 3)
```

```
## Warning: Default search for "data" layer in "RNA" assay yielded no results;  
## utilizing "counts" layer instead.
```

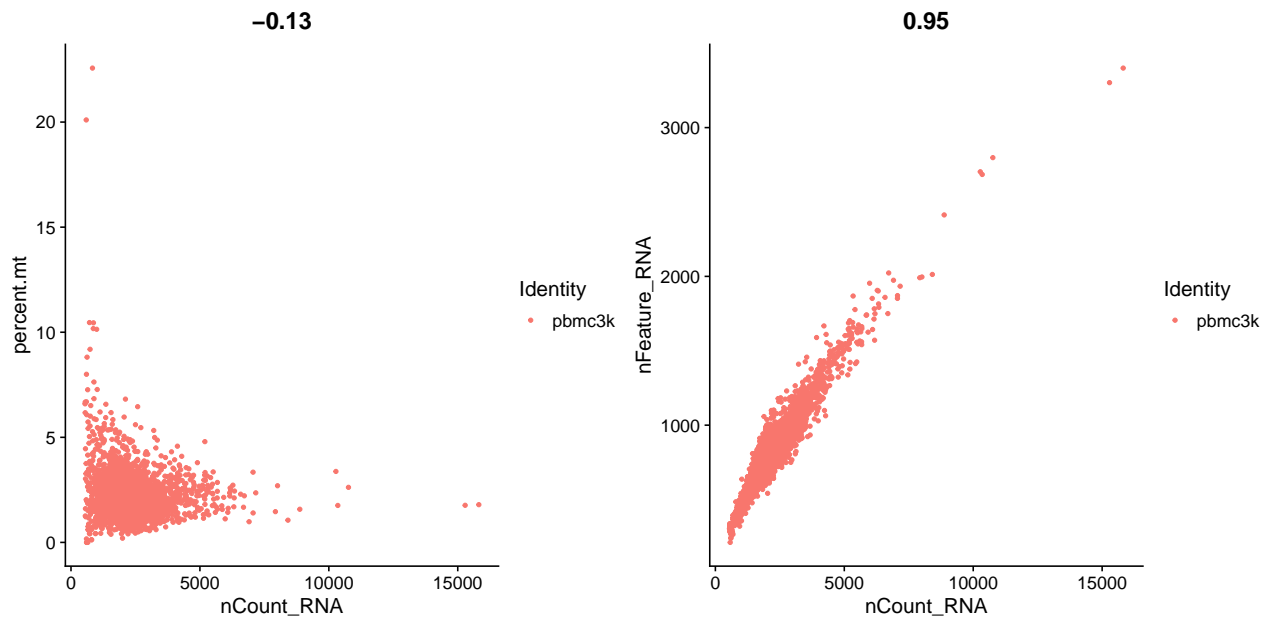
```
plot_before_QC
```



```
# FeatureScatter is typically used to visualize feature-feature relationships, but can be used  
# for anything calculated by the object, i.e. columns in object metadata, PC scores etc.
```

```
# These plots help to identify poor-quality cells or doublets. Cells with high `nCount_RNA` but low `nF
```

```
plot1 <- FeatureScatter(pbmc, feature1 = "nCount_RNA", feature2 = "percent.mt")  
plot2 <- FeatureScatter(pbmc, feature1 = "nCount_RNA", feature2 = "nFeature_RNA")  
plot1 + plot2
```

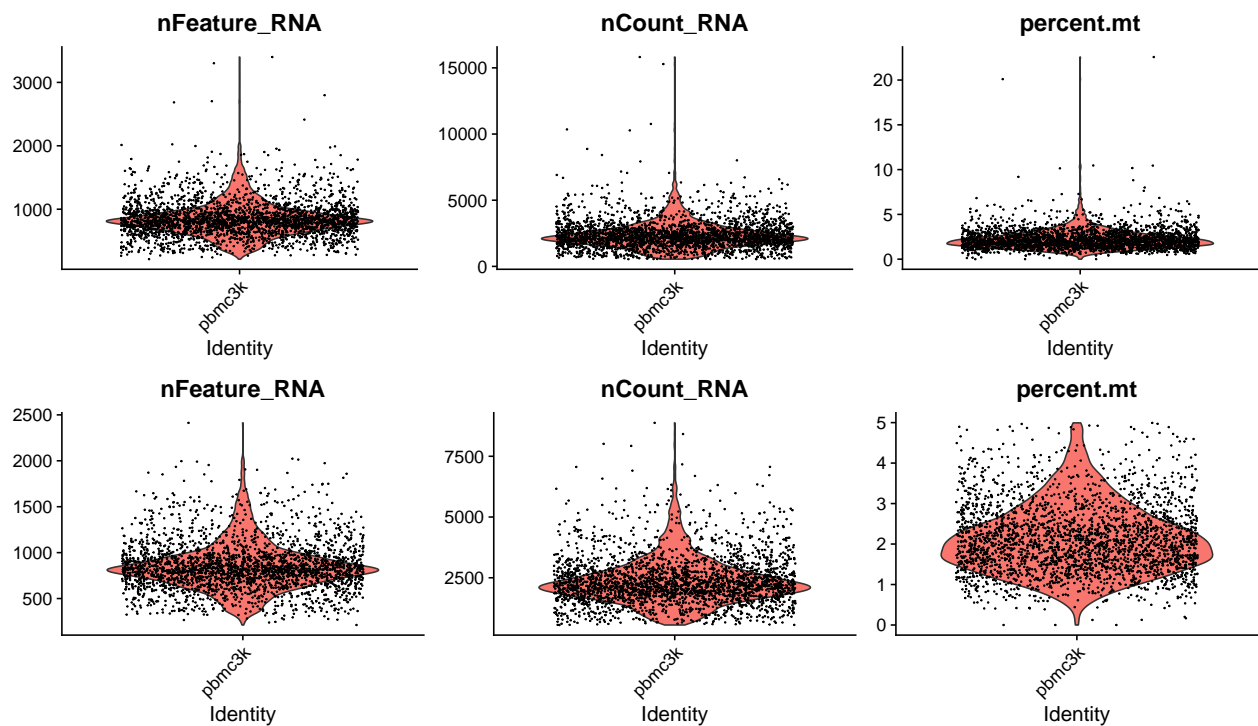
```
pbmc <- subset(pbmc, subset = nFeature_RNA > 200 & nFeature_RNA < 2500 & percent.mt < 5)
```

```
# Visualize QC metrics before and after QC
```

```
plot_after_QC <- VlnPlot(pbmc, features = c("nFeature_RNA", "nCount_RNA", "percent.mt"), ncol = 3)
```

```
## Warning: Default search for "data" layer in "RNA" assay yielded no results;  
## utilizing "counts" layer instead.
```

```
plot_before_QC / plot_after_QC
```



Normalizing the data

After removing unwanted cells from the dataset, the next step is to normalize the data. By default, we employ a global-scaling normalization method “LogNormalize” that normalizes the feature expression measurements for each cell by the total expression, multiplies this by a scale factor (10,000 by default), and log-transforms the result. In Seurat v5, Normalized values are stored in `pbmc[["RNA"]]`\$data.

```
pbmc <- NormalizeData(pbmc, normalization.method = "LogNormalize", scale.factor = 10000)
```

```
## Normalizing layer: counts
```

For clarity, in this previous line of code (and in future commands), we provide the default values for certain parameters in the function call. However, this isn’t required and the same behavior can be achieved the following code.

```
pbmc <- NormalizeData(pbmc)
```

```
## Normalizing layer: counts
```

While this method of normalization is standard and widely used in scRNA-seq analysis, global-scaling relies on an assumption that each cell originally contains the same number of RNA molecules.

- “normalization.method”: Method for normalization.
 - “LogNormalize”: Feature counts for each cell are divided by the total counts for that cell and multiplied by the scale.factor. This is then natural-log transformed using `log1p`
 - “CLR”: Applies a centered log ratio transformation
 - “RC”: Relative counts. Feature counts for each cell are divided by the total counts for that cell and multiplied by the scale.factor. No log-transformation is applied. For counts per million (CPM) set `scale.factor = 1e6`
- While this method of normalization is standard and widely used in scRNA-seq analysis, global-scaling relies on an assumption that each cell originally contains the same number of RNA molecules.
- There is an alternative workflows for the single cell preprocessing that do not make these assumptions: SCTransform() normalization workflow. The use of SCTransform replaces the need to run `NormalizeData`, `FindVariableFeatures`, or `ScaleData` (described below.)

Apply sctransform normalization

- Note that this single command replaces `NormalizeData()`, `ScaleData()`, and `FindVariableFeatures()`.
- Transformed data will be available in the SCT assay, which is set as the default after running `sctransform`
- During normalization, we can also remove confounding sources of variation, for example, mitochondrial mapping percentage

```
# run sctransform
#pbmc <- SCTransform(pbmc, vars.to.regress = "percent.mt", verbose = FALSE)
```

Identification of highly variable features (feature selection)

We next calculate a subset of features that exhibit high cell-to-cell variation in the dataset (i.e, they are highly expressed in some cells, and lowly expressed in others). We found that focusing on these genes in downstream analysis helps to highlight biological signal in single-cell datasets.

By default, we return 2,000 features per dataset. These will be used in downstream analysis, like PCA.

```
pbmc <- FindVariableFeatures(pbmc, selection.method = "vst", nfeatures = 2000)
```

```
## Finding variable features for layer counts
```

```

# Identify the 10 most highly variable genes
top10 <- head(VariableFeatures(pbm), 10)
top10

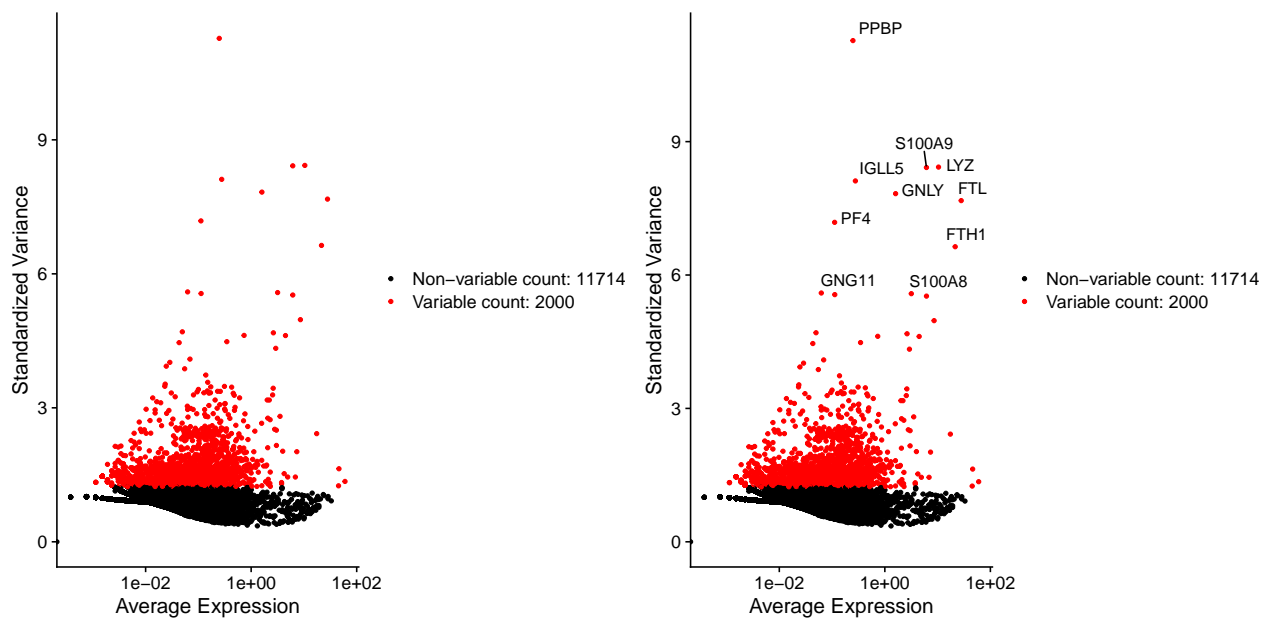
## [1] "PPBP" "LYZ" "S100A9" "IGLL5" "GNLY" "FTL" "PF4" "FTH1"
## [9] "GNG11" "S100A8"

# plot variable features with and without labels
plot1 <- VariableFeaturePlot(pbm)
plot2 <- LabelPoints(plot = plot1, points = top10, repel = TRUE)

## When using repel, set xnuage and ynuage to 0 for optimal results
plot1 + plot2

## Warning in scale_x_log10(): log-10 transformation introduced infinite values.
## log-10 transformation introduced infinite values.

```



Scaling the data

Next, we apply a linear transformation ('scaling') that is a standard pre-processing step prior to dimensional reduction techniques like PCA. The `ScaleData()` function:

- Shifts the expression of each gene, so that the mean expression across cells is 0
- Scales the expression of each gene, so that the variance across cells is 1
 - This step gives equal weight in downstream analyses, so that highly-expressed genes do not dominate
- The results of this are stored in `pbmc[["RNA"]]`\$scale.data
- By default, only variable features are scaled.
- You can specify the features argument to scale additional features

```

all.genes <- rownames(pbm)
pbm <- ScaleData(pbm, features = all.genes)

```

```
## Centering and scaling data matrix
```

Perform linear dimensional reduction

Next we perform PCA on the scaled data. By default, only the previously determined variable features are used as input, but can be defined using features argument if you wish to choose a different subset (if you do want to use a custom subset of features, make sure you pass these to ScaleData first).

For the first principal components, Seurat outputs a list of genes with the most positive and negative loadings, representing modules of genes that exhibit either correlation (or anti-correlation) across single-cells in the dataset.

```
pbmc <- RunPCA(pbmc, features = VariableFeatures(object = pbmc))

## PC_ 1
## Positive: CST3, TYROBP, LST1, AIF1, FTL, FTH1, LYZ, FCN1, S100A9, TYMP
##           FCER1G, CFD, LGALS1, S100A8, CTSS, LGALS2, SERPINA1, IFITM3, SPI1, CFP
##           PSAP, IFI30, SAT1, COTL1, S100A11, NPC2, GRN, LGALS3, GSTP1, PYCARD
## Negative: MALAT1, LTB, IL32, IL7R, CD2, B2M, ACAP1, CD27, STK17A, CTSW
##           CD247, GIMAP5, AQP3, CCL5, SELL, TRAF3IP3, GZMA, MAL, CST7, ITM2A
##           MYC, GIMAP7, HOPX, BEX2, LDLRAP1, GZMK, ETS1, ZAP70, TNFAIP8, RIC3
## PC_ 2
## Positive: CD79A, MS4A1, TCL1A, HLA-DQA1, HLA-DQB1, HLA-DRA, LINC00926, CD79B, HLA-DRB1, CD74
##           HLA-DMA, HLA-DPB1, HLA-DQA2, CD37, HLA-DRB5, HLA-DMB, HLA-DPA1, FCRLA, HVCN1, LTB
##           BLNK, P2RX5, IGLL5, IRF8, SWAP70, ARHGAP24, FCGR2B, SMIM14, PPP1R14A, C16orf74
## Negative: NKG7, PRF1, CST7, GZMB, GZMA, FGFBP2, CTSW, GNLY, B2M, SPON2
##           CCL4, GZMH, FCGR3A, CCL5, CD247, XCL2, CLIC3, AKR1C3, SRGN, HOPX
##           TTC38, APMAP, CTSC, S100A4, IGFBP7, ANXA1, ID2, IL32, XCL1, RHOC
## PC_ 3
## Positive: HLA-DQA1, CD79A, CD79B, HLA-DQB1, HLA-DPB1, HLA-DPA1, CD74, MS4A1, HLA-DRB1, HLA-DRA
##           HLA-DRB5, HLA-DQA2, TCL1A, LINC00926, HLA-DMB, HLA-DMA, CD37, HVCN1, FCRLA, IRF8
##           PLAC8, BLNK, MALAT1, SMIM14, PLD4, LAT2, IGLL5, P2RX5, SWAP70, FCGR2B
## Negative: PPBP, PF4, SDPR, SPARC, GNG11, NRGN, GP9, RGS18, TUBB1, CLU
##           HIST1H2AC, AP001189.4, ITGA2B, CD9, TMEM40, PTCRA, CA2, ACRBP, MMD, TREML1
##           NGFRAP1, F13A1, SEPT5, RUFY1, TSC22D1, MPP1, CMTM5, RP11-367G6.3, MYL9, GP1BA
## PC_ 4
## Positive: HLA-DQA1, CD79B, CD79A, MS4A1, HLA-DQB1, CD74, HLA-DPB1, HIST1H2AC, PF4, TCL1A
##           SDPR, HLA-DPA1, HLA-DRB1, HLA-DQA2, HLA-DRA, PPBP, LINC00926, GNG11, HLA-DRB5, SPARC
##           GP9, AP001189.4, CA2, PTCRA, CD9, NRGN, RGS18, GZMB, CLU, TUBB1
## Negative: VIM, IL7R, S100A6, IL32, S100A8, S100A4, GIMAP7, S100A10, S100A9, MAL
##           AQP3, CD2, CD14, FYB, LGALS2, GIMAP4, ANXA1, CD27, FCN1, RBP7
##           LYZ, S100A11, GIMAP5, MS4A6A, S100A12, FOLR3, TRABD2A, AIF1, IL8, IFI6
## PC_ 5
## Positive: GZMB, NKG7, S100A8, FGFBP2, GNLY, CCL4, CST7, PRF1, GZMA, SPON2
##           GZMH, S100A9, LGALS2, CCL3, CTSW, XCL2, CD14, CLIC3, S100A12, CCL5
##           RBP7, MS4A6A, GSTP1, FOLR3, IGFBP7, TYROBP, TTC38, AKR1C3, XCL1, HOPX
## Negative: LTB, IL7R, CKB, VIM, MS4A7, AQP3, CYTIP, RP11-290F20.3, SIGLEC10, HMOX1
##           PTGES3, LILRB2, MAL, CD27, HN1, CD2, GDI2, ANXA5, CORO1B, TUBA1B
##           FAM110A, ATP1A1, TRADD, PPA1, CCDC109B, ABRACL, CTD-2006K23.1, WARS, VM01, FYB
```

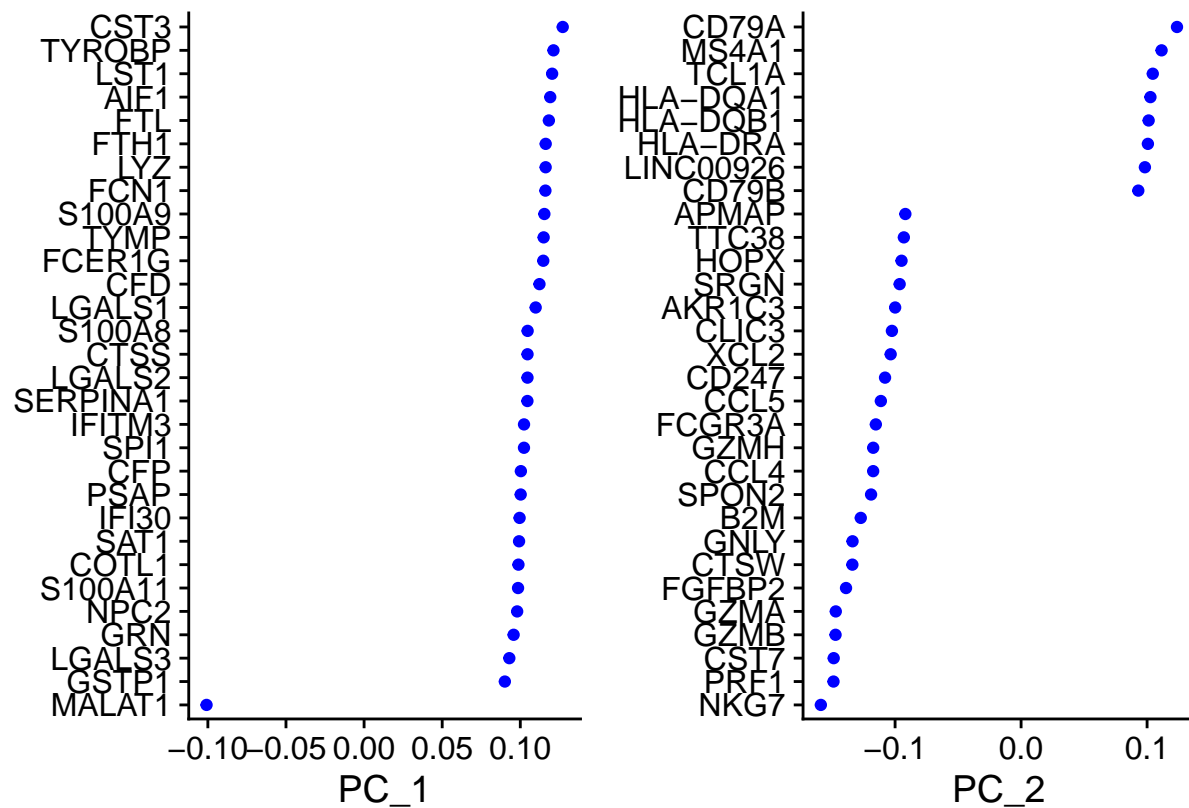
Seurat provides several useful ways of visualizing both cells and features that define the PCA, including VizDimReduction(), DimPlot(), and DimHeatmap()

```
# Examine and visualize PCA results a few different ways
print(pbmc[["pca"]], dims = 1:5, nfeatures = 5)
```

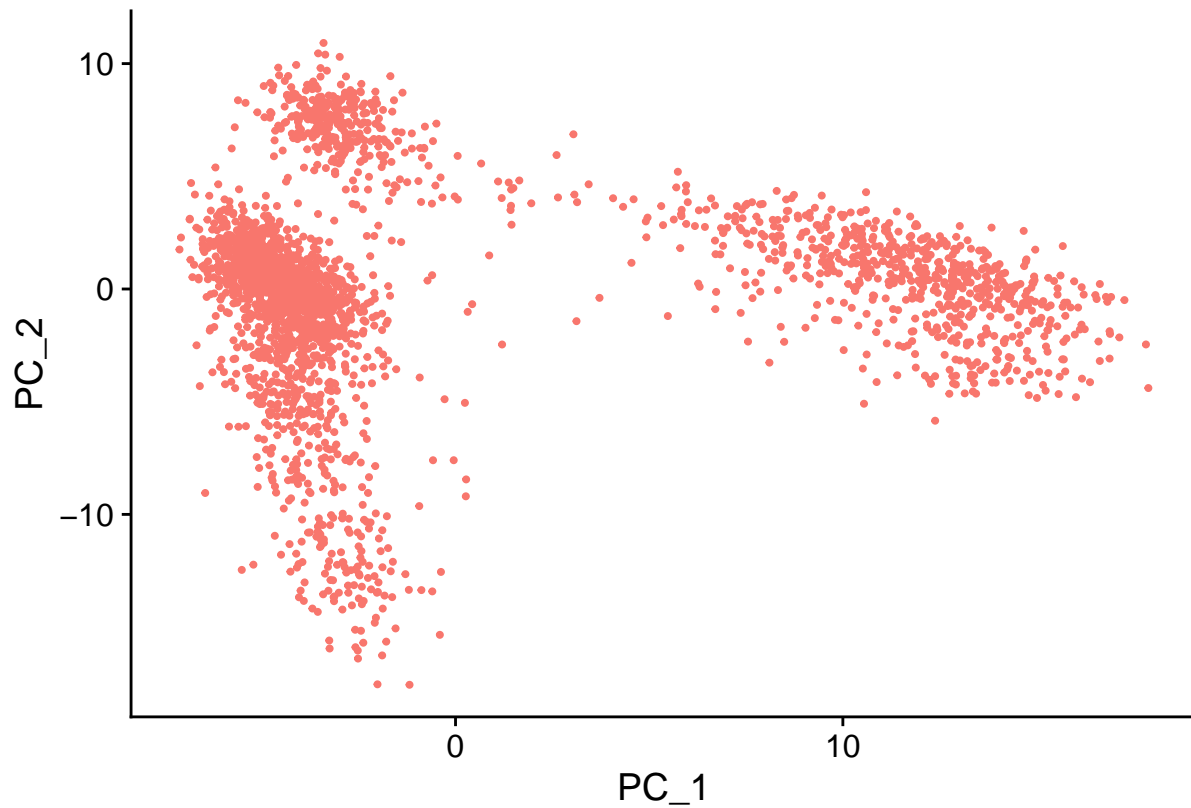
```
## PC_ 1
## Positive: CST3, TYROBP, LST1, AIF1, FTL
## Negative: MALAT1, LTB, IL32, IL7R, CD2
```

```
## PC_ 2
## Positive: CD79A, MS4A1, TCL1A, HLA-DQA1, HLA-DQB1
## Negative: NKG7, PRF1, CST7, GZMB, GZMA
## PC_ 3
## Positive: HLA-DQA1, CD79A, CD79B, HLA-DQB1, HLA-DPB1
## Negative: PPBP, PF4, SDPR, SPARC, GNG11
## PC_ 4
## Positive: HLA-DQA1, CD79B, CD79A, MS4A1, HLA-DQB1
## Negative: VIM, IL7R, S100A6, IL32, S100A8
## PC_ 5
## Positive: GZMB, NKG7, S100A8, FGFBP2, GNLY
## Negative: LTB, IL7R, CKB, VIM, MS4A7
```

```
VizDimLoadings(pbmc, dims = 1:2, reduction = "pca")
```



```
DimPlot(pbmc, reduction = "pca") + NoLegend()
```

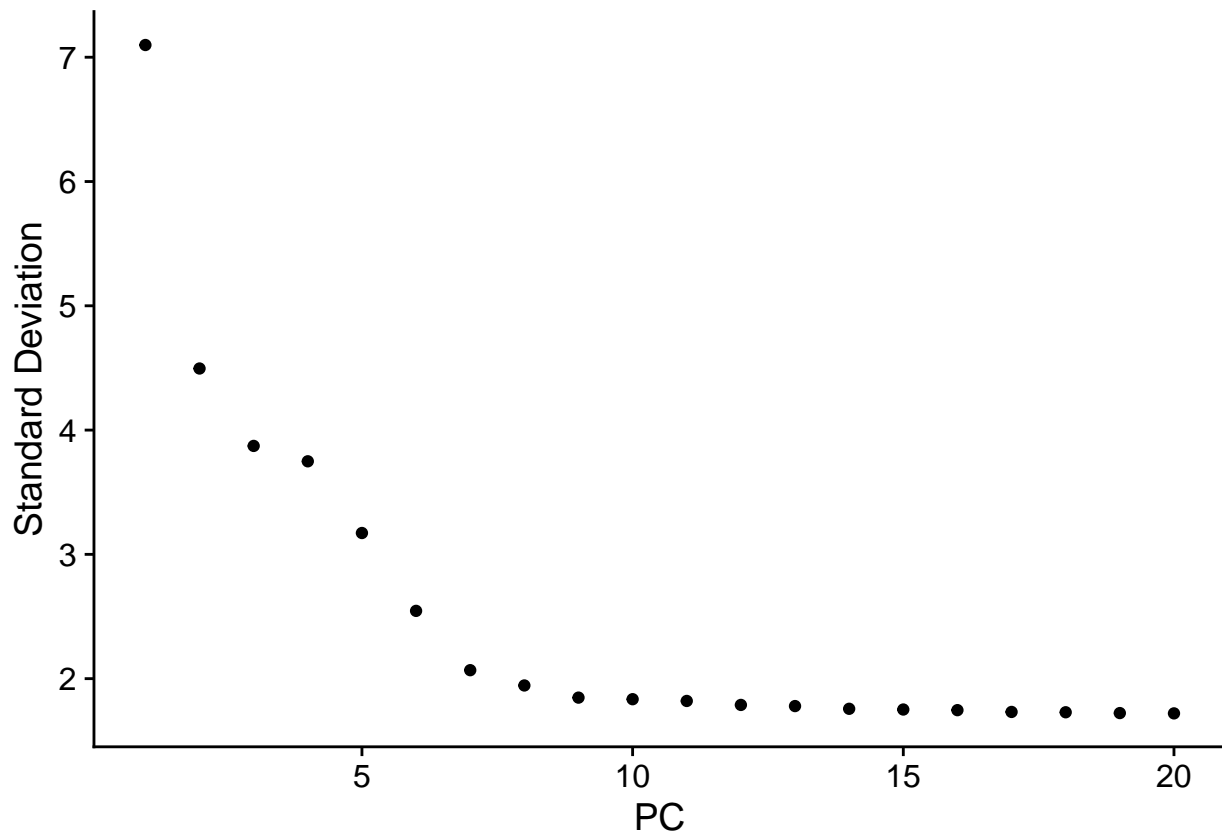


Determine the ‘dimensionality’ of the dataset

To overcome the extensive technical noise in any single feature for scRNA-seq data, Seurat clusters cells based on their PCA scores, with each PC essentially representing a ‘metafeature’ that combines information across a correlated feature set. The top principal components therefore represent a robust compression of the dataset. However, how many components should we choose to include? 10? 20? 100?

We generate an ‘Elbow plot’: a ranking of principal components based on the percentage of variance explained by each one (ElbowPlot() function). In this example, we can observe an ‘elbow’ around PC9-10, suggesting that the majority of true signal is captured in the first 10 PCs.

```
ElbowPlot(pbmc)
```



Identifying the true dimensionality of a dataset – can be challenging/uncertain for the user. We therefore suggest these multiple approaches for users. The first is more supervised, exploring PCs to determine relevant sources of heterogeneity, and could be used in conjunction with GSEA for example. The second (ElbowPlot) The third is a heuristic that is commonly used, and can be calculated instantly. In this example, we might have been justified in choosing anything between PC 7-12 as a cutoff.

We chose 10 here, but encourage users to consider the following:

- Dendritic cell and NK aficionados may recognize that genes strongly associated with PCs 12 and 13 define rare immune subsets (i.e. MZB1 is a marker for plasmacytoid DCs). However, these groups are so rare, they are difficult to distinguish from background noise for a dataset of this size without prior knowledge.
- We encourage users to repeat downstream analyses with a different number of PCs (10, 15, or even 50!). As you will observe, the results often do not differ dramatically.
- We advise users to err on the higher side when choosing this parameter. For example, performing downstream analyses with only 5 PCs does significantly and adversely affect results.

Cluster the cells

Seurat applies a graph-based clustering approach. Importantly, the distance metric which drives the clustering analysis (based on previously identified PCs) remains the same. However, our approach to partitioning the cellular distance matrix into clusters has dramatically improved. Briefly, these methods embed cells in a graph structure - for example a K-nearest neighbor (KNN) graph, with edges drawn between cells with similar feature expression patterns, and then attempt to partition this graph into highly interconnected ‘quasi-cliques’ or ‘communities’.

We first construct a KNN graph based on the euclidean distance in PCA space, and refine the edge weights between any two cells based on the shared overlap in their local neighborhoods (Jaccard similarity). This step

is performed using the `FindNeighbors()` function, and takes as input the previously defined dimensionality of the dataset (first 10 PCs).

To cluster the cells, we next apply modularity optimization techniques such as the Louvain algorithm (default) or SLM, to iteratively group cells together, with the goal of optimizing the standard modularity function. The `FindClusters()` function implements this procedure, and contains a resolution parameter that sets the ‘granularity’ of the downstream clustering, with increased values leading to a greater number of clusters. We find that setting this parameter between 0.4-1.2 typically returns good results for single-cell datasets of around 3K cells. Optimal resolution often increases for larger datasets. The clusters can be found using the `Idents()` function.

```
pbmc <- FindNeighbors(pbmc, dims = 1:10)

## Computing nearest neighbor graph
## Computing SNN
pbmc <- FindClusters(pbmc, resolution = 0.5)

## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 2638
## Number of edges: 95965
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.8723
## Number of communities: 9
## Elapsed time: 0 seconds
# Look at cluster IDs of the first 5 cells
head(Idents(pbmc), 5)

## AAACATACAACCAC-1 AAACATTGAGCTAC-1 AAACATTGATCAGC-1 AAACCGTGCTTCCG-1
##                2                3                2                1
## AAACCGTGTATGCG-1
##                6
## Levels: 0 1 2 3 4 5 6 7 8
```

Run non-linear dimensional reduction (UMAP/tSNE)

Seurat offers several non-linear dimensional reduction techniques, such as tSNE and UMAP, to visualize and explore these datasets. The goal of these algorithms is to learn underlying structure in the dataset, in order to place similar cells together in low-dimensional space. Therefore, cells that are grouped together within graph-based clusters determined above should co-localize on these dimension reduction plots.

While we and others have routinely found 2D visualization techniques like tSNE and UMAP to be valuable tools for exploring datasets, all visualization techniques have limitations, and cannot fully represent the complexity of the underlying data. In particular, these methods aim to preserve local distances in the dataset (i.e. ensuring that cells with very similar gene expression profiles co-localize), but often do not preserve more global relationships. We encourage users to leverage techniques like UMAP for visualization, but to avoid drawing biological conclusions solely on the basis of visualization techniques.

```
pbmc <- RunUMAP(pbmc, dims = 1:10)

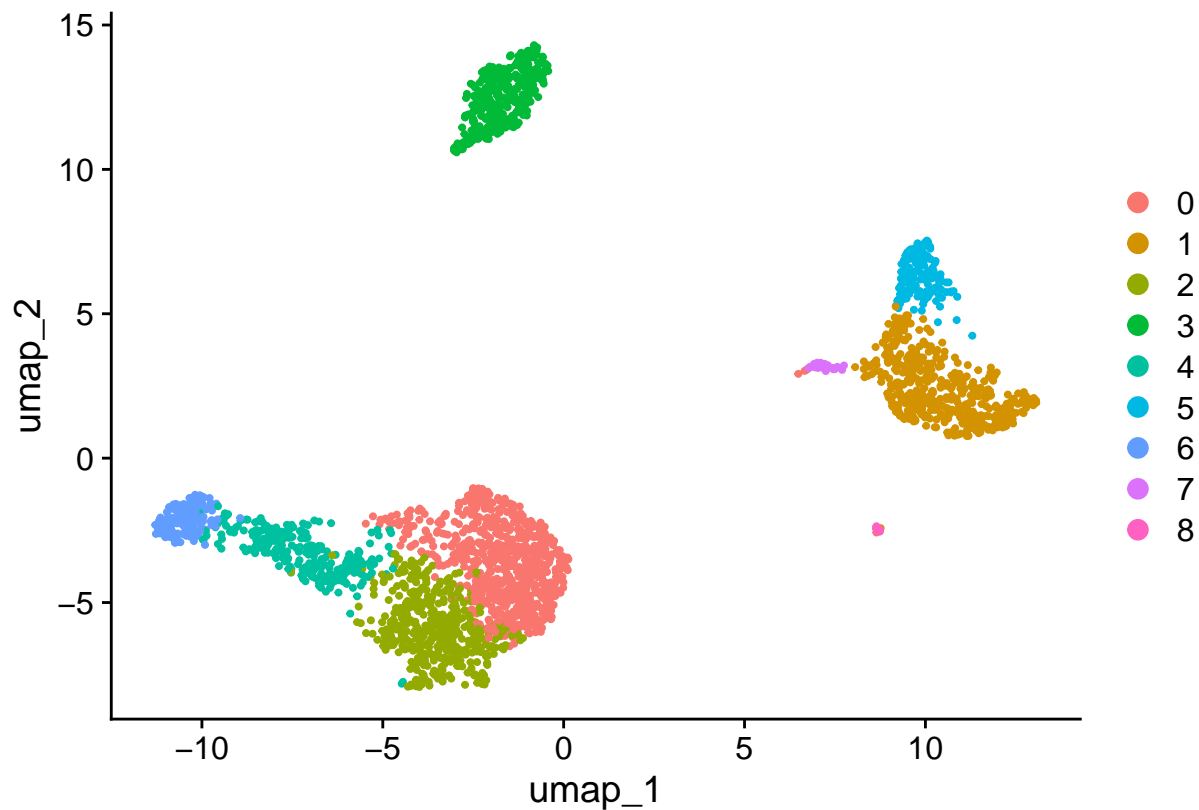
## Warning: The default method for RunUMAP has changed from calling Python UMAP via reticulate to the R
## To use Python UMAP via reticulate, set umap.method to 'umap-learn' and metric to 'correlation'
## This message will be shown once per session
```



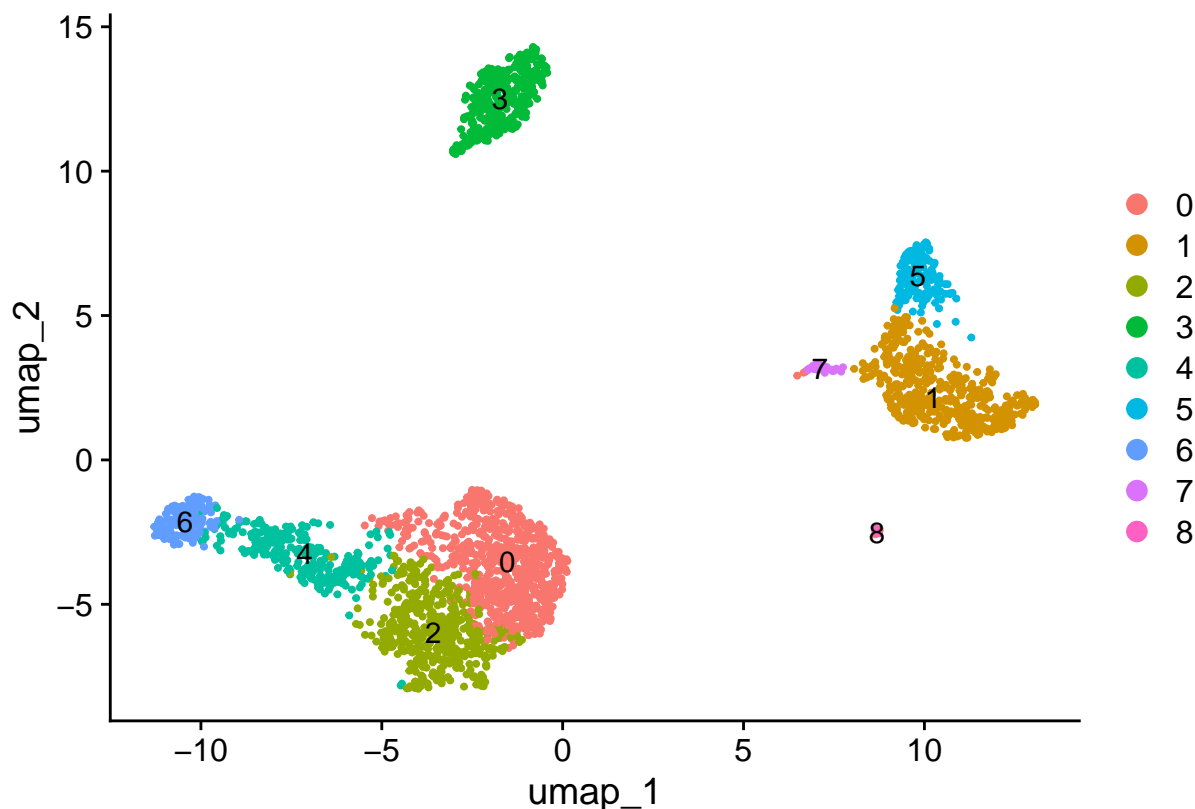
```
## 21:14:40 UMAP embedding parameters a = 0.9922 b = 1.112
## 21:14:40 Read 2638 rows and found 10 numeric columns
## 21:14:40 Using Annoy for neighbor search, n_neighbors = 30
## 21:14:40 Building Annoy index with metric = cosine, n_trees = 50
## 0%   10   20   30   40   50   60   70   80   90  100%
## [----|----|----|----|----|----|----|----|----|----|
## *****|
## 21:14:40 Writing NN index file to temp file /var/folders/wk/p4zj4_993czf2pnjs015dffm0000gp/T//RtmpfV
## 21:14:40 Searching Annoy index using 1 thread, search_k = 3000
## 21:14:41 Annoy recall = 100%
## 21:14:42 Commencing smooth kNN distance calibration using 1 thread with target n_neighbors = 30
## 21:14:43 Initializing from normalized Laplacian + noise (using RSpectra)
## 21:14:43 Commencing optimization for 500 epochs, with 105124 positive edges
## 21:14:49 Optimization finished
```

```
# note that you can set `label = TRUE` or use the LabelClusters function to help label
# individual clusters
```

```
DimPlot(pbmc, reduction = "umap")
```



```
DimPlot(pbmc, reduction = "umap", label = T)
```



Finding differentially expressed features (cluster biomarkers)

Seurat can help you find markers that define clusters via differential expression (DE). By default, Seurat performs differential expression (DE) testing based on the non-parametric Wilcoxon rank sum test. It identifies positive and negative markers of a single cluster (specified in `ident.1`), compared to specific groups of cells (`ident.2`). `FindAllMarkers()` automates this process for all clusters, but you can also test groups of clusters vs. each other, or against all cells.

```
# find all markers of cluster 2 (cluster ID in the above UMAP figure)
cluster2.markers <- FindMarkers(pbmcc, ident.1 = 2)
head(cluster2.markers, n = 5)
```

```
##           p_val avg_log2FC pct.1 pct.2    p_val_adj
## IL32 2.593535e-91  1.3221171 0.949 0.466 3.556774e-87
## LTB  7.994465e-87  1.3450377 0.981 0.644 1.096361e-82
## CD3D 3.922451e-70  1.0562099 0.922 0.433 5.379250e-66
## IL7R 1.130870e-66  1.4256944 0.748 0.327 1.550876e-62
## LDHB 4.082189e-65  0.9765875 0.953 0.614 5.598314e-61
```

The results data frame has the following columns :

- `p_val` : p-value (unadjusted)
- `avg_log2FC` : log fold-change of the average expression between the two groups. Positive values indicate that the feature is more highly expressed in the first group.
- `pct.1` : The percentage of cells where the feature is detected in the first group
- `pct.2` : The percentage of cells where the feature is detected in the second group
- `p_val_adj` : Adjusted p-value, based on Bonferroni correction using all features in the dataset.

If the `ident.2` parameter is omitted or set to `NULL`, `FindMarkers()` will test for differentially expressed features

between the group specified by ident.1 and all other cells. Additionally, the parameter only.pos can be set to TRUE to only search for positive markers, i.e. features that are more highly expressed in the ident.1 group.

```
cluster2.markers <- FindMarkers(pbmcc, ident.1 = 2, ident.2 = NULL, only.pos = TRUE)
head(cluster2.markers, n = 5)
```

```
##           p_val avg_log2FC pct.1 pct.2    p_val_adj
## IL32 2.593535e-91 1.3221171 0.949 0.466 3.556774e-87
## LTB 7.994465e-87 1.3450377 0.981 0.644 1.096361e-82
## CD3D 3.922451e-70 1.0562099 0.922 0.433 5.379250e-66
## IL7R 1.130870e-66 1.4256944 0.748 0.327 1.550876e-62
## LDHB 4.082189e-65 0.9765875 0.953 0.614 5.598314e-61
```

```
# find all markers distinguishing cluster 5 from clusters 0 and 3
cluster5.markers <- FindMarkers(pbmcc, ident.1 = 5, ident.2 = c(0, 3))
head(cluster5.markers, n = 5)
```

```
##           p_val avg_log2FC pct.1 pct.2    p_val_adj
## FCGR3A 2.150929e-209 6.832372 0.975 0.039 2.949784e-205
## IFITM3 6.103366e-199 6.181000 0.975 0.048 8.370156e-195
## CFD 8.891428e-198 6.052575 0.938 0.037 1.219370e-193
## CD68 2.374425e-194 5.493138 0.926 0.035 3.256286e-190
## RP11-290F20.3 9.308287e-191 6.335402 0.840 0.016 1.276538e-186
```

```
# find markers for every cluster compared to all remaining cells, report only the positive ones
pbmc.markers <- FindAllMarkers(pbmcc, only.pos = TRUE)
```

```
## Calculating cluster 0
## Calculating cluster 1
## Calculating cluster 2
## Calculating cluster 3
## Calculating cluster 4
## Calculating cluster 5
## Calculating cluster 6
## Calculating cluster 7
## Calculating cluster 8
```

```
pbmc.markers %>%
  group_by(cluster) %>%
  dplyr::filter(avg_log2FC > 1)
```

```
## # A tibble: 7,046 x 7
## # Groups:   cluster [9]
##           p_val avg_log2FC pct.1 pct.2 p_val_adj cluster gene
##           <dbl>      <dbl> <dbl> <dbl>      <dbl> <fct>   <chr>
## 1 1.74e-109      1.19 0.897 0.593 2.39e-105 0      LDHB
## 2 1.17e- 83      2.37 0.435 0.108 1.60e- 79 0      CCR7
## 3 8.94e- 79      1.09 0.838 0.403 1.23e- 74 0      CD3D
## 4 3.05e- 53      1.02 0.722 0.399 4.19e- 49 0      CD3E
## 5 3.28e- 49      2.10 0.333 0.103 4.50e- 45 0      LEF1
## 6 6.66e- 49      1.25 0.623 0.358 9.13e- 45 0      NOSIP
## 7 9.31e- 44      2.02 0.328 0.11 1.28e- 39 0      PRKCQ-AS1
## 8 4.69e- 43      1.53 0.435 0.184 6.43e- 39 0      PIK3IP1
```

```
## 9 1.47e- 39      2.70 0.195 0.04 2.01e- 35 0      FHIT
## 10 2.44e- 33      1.94 0.262 0.087 3.34e- 29 0      MAL
## # i 7,036 more rows
```

Seurat has several tests for differential expression which can be set with the `test.use` parameter:

- “wilcox” : Wilcoxon rank sum test (default, using ‘presto’ package)
- “wilcox_limma” : Wilcoxon rank sum test (using ‘limma’ package)
- “bimod” : Likelihood-ratio test for single cell feature expression, (McDavid et al., Bioinformatics, 2013)
- “roc” : Standard AUC classifier
- “t” : Student’s t-test
- “poisson” : Likelihood ratio test assuming an underlying negative binomial distribution. Use only for UMI-based datasets
- “negbinom” : Likelihood ratio test assuming an underlying negative binomial distribution. Use only for UMI-based datasets
- “LR” : Uses a logistic regression framework to determine differentially expressed genes. Constructs a logistic regression model predicting group membership based on each- feature individually and compares this to a null model with a likelihood ratio test.
- “MAST” : GLM-framework that treats cellular detection rate as a covariate (Finak et al, Genome Biology, 2015) (Installation instructions)
- “DESeq2” : DE based on a model using the negative binomial distribution (Love et al, Genome Biology, 2014) (Installation instructions) For MAST and DESeq2, please ensure that these packages are installed separately in order to use them as part of Seurat. Once installed, use the `test.use` parameter can be used to specify which DE test to use.

For example, the ROC test returns the ‘classification power’ for any individual marker (ranging from 0 - random, to 1 - perfect).

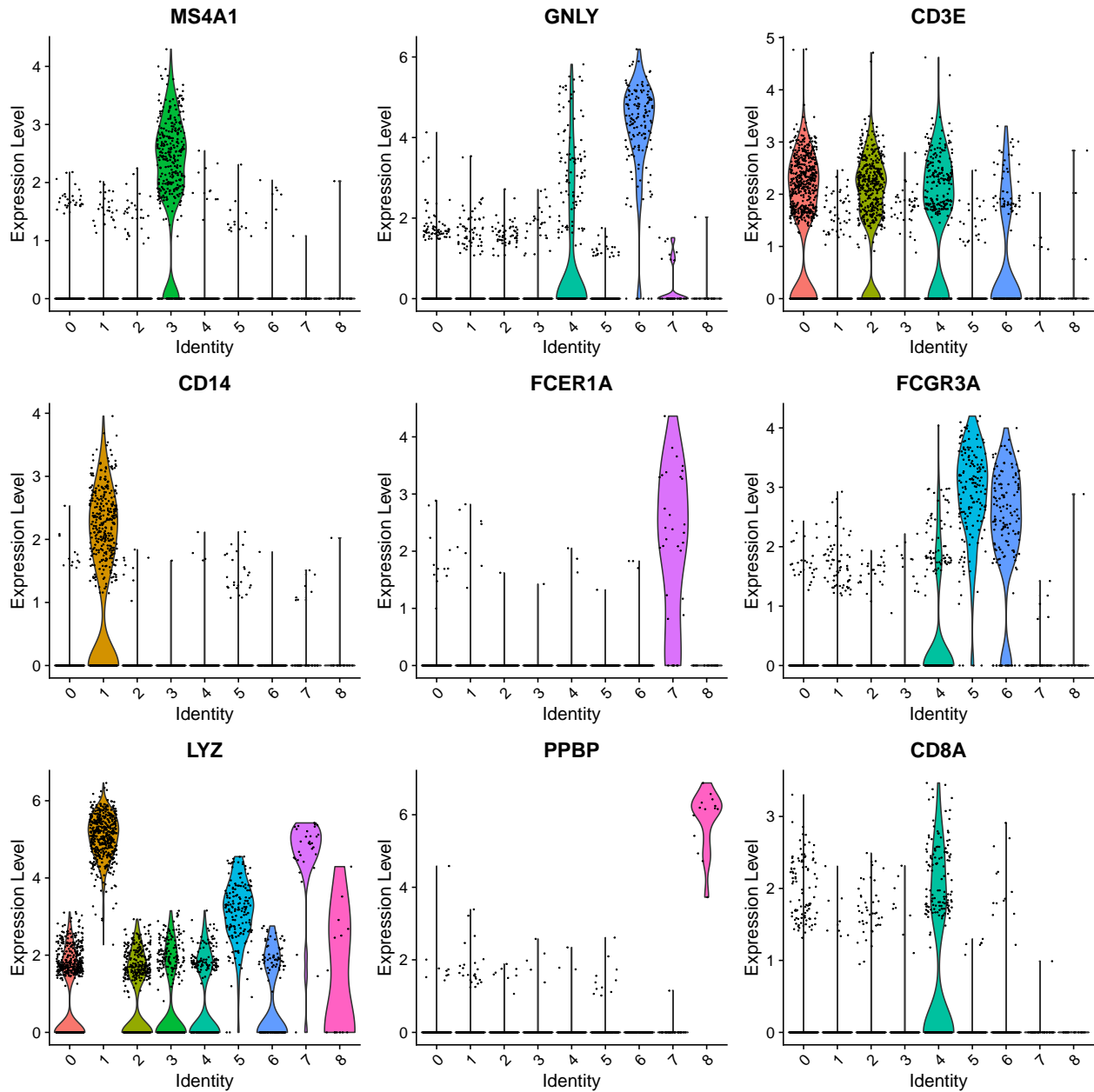
```
cluster0.markers <- FindMarkers(pbmc, ident.1 = 0, logfc.threshold = 0.25, test.use = "roc", only.pos =
```

Visualizations of marker feature expression

We include several tools for visualizing marker expression. `VlnPlot()` (shows expression probability distributions across clusters), and `FeaturePlot()` (visualizes feature expression on a tSNE or PCA plot) are our most commonly used visualizations.

```
feature_genes = c("MS4A1", "GNLY", "CD3E", "CD14", "FCER1A", "FCGR3A", "LYZ", "PPBP", "CD8A")

# Violin plot - Visualize single cell expression distributions in each cluster
VlnPlot(pbmc, features = feature_genes)
```



Ridge Plot - show the distribution of expression values for each gene across all clusters. These are

```
RidgePlot(pbm, features = feature_genes, ncol = 2)
```

```
## Picking joint bandwidth of 0.0236
```

```
## Picking joint bandwidth of 0.0971
```

```
## Picking joint bandwidth of 0.125
```

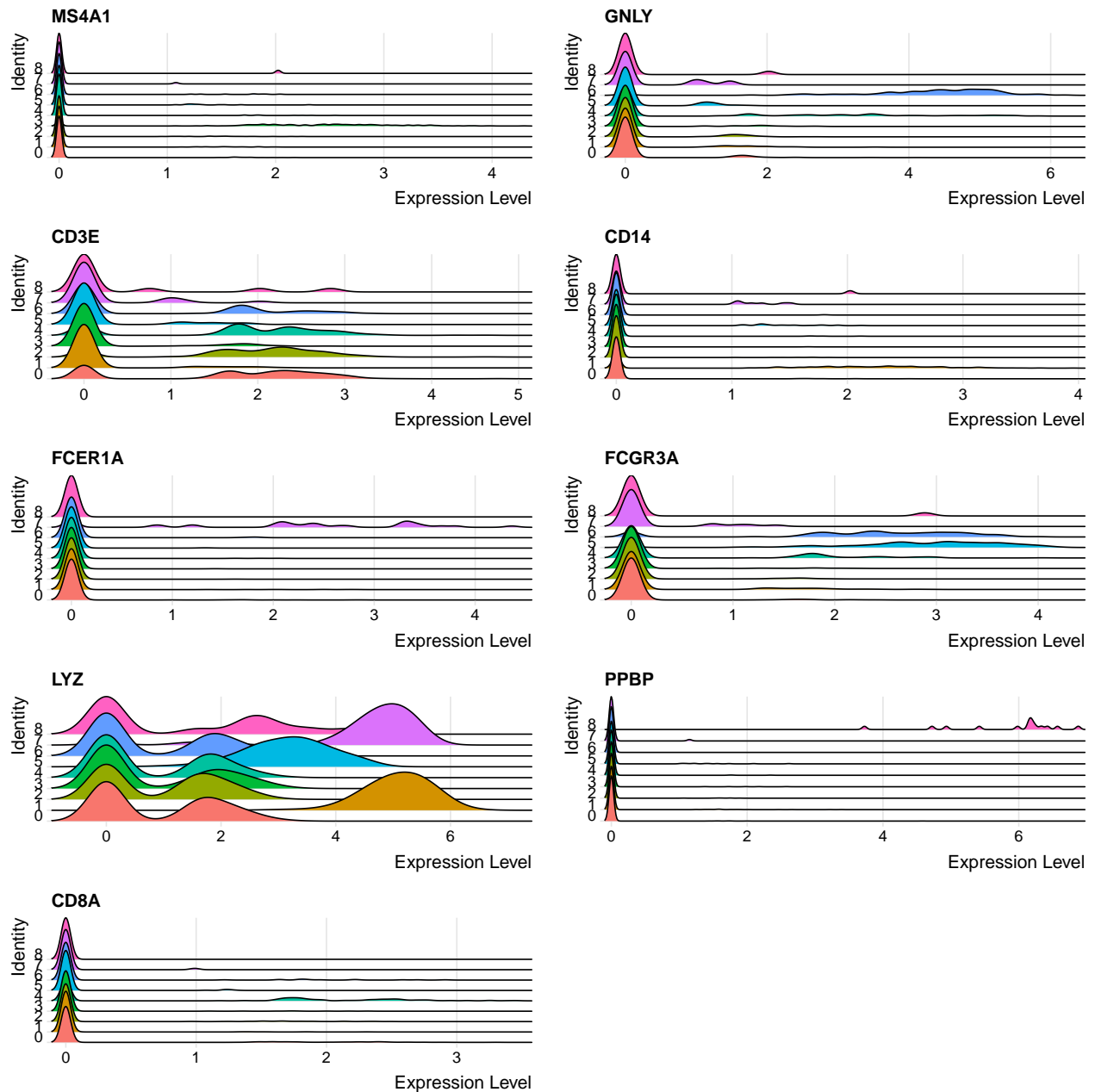
```
## Picking joint bandwidth of 0.0337
```

```
## Picking joint bandwidth of 0.0659
```

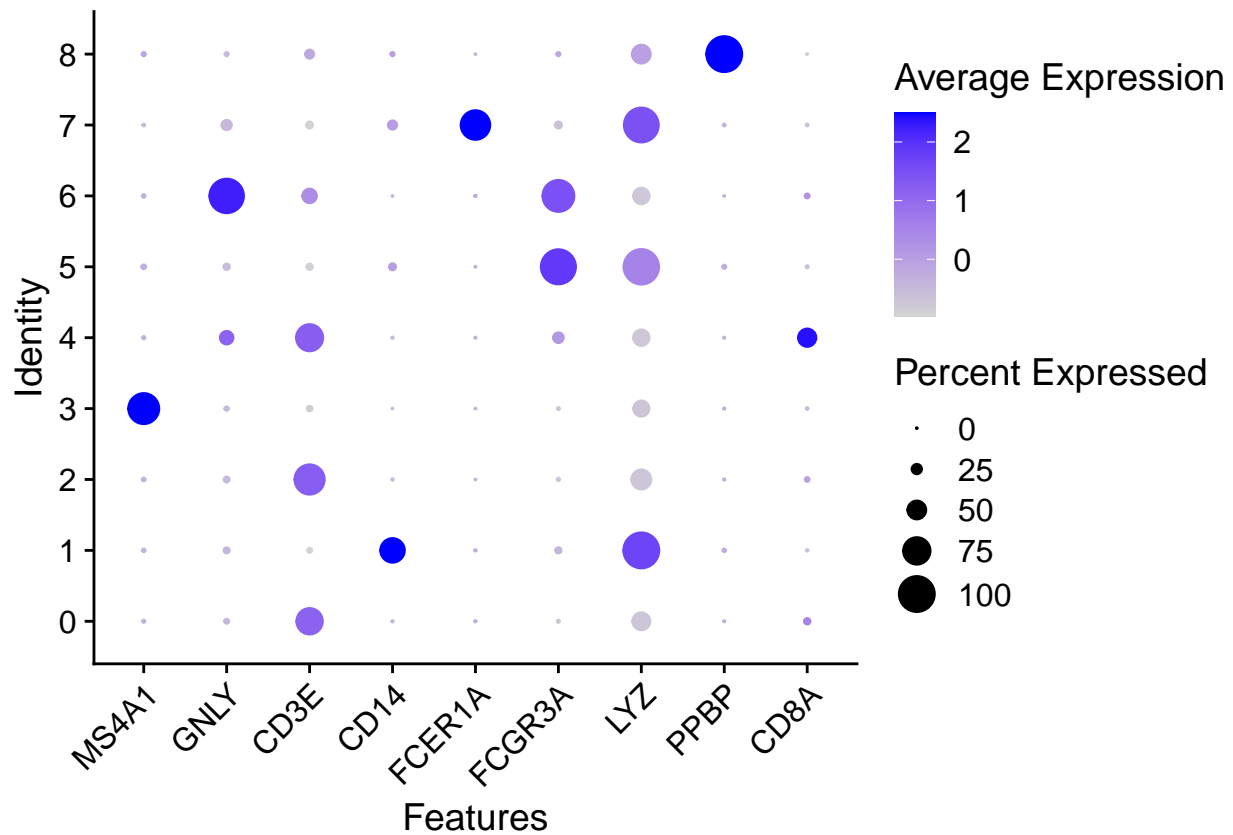
```
## Picking joint bandwidth of 0.0872
```

```
## Picking joint bandwidth of 0.319
```

```
## Picking joint bandwidth of 0.033
## Picking joint bandwidth of 0.0368
```

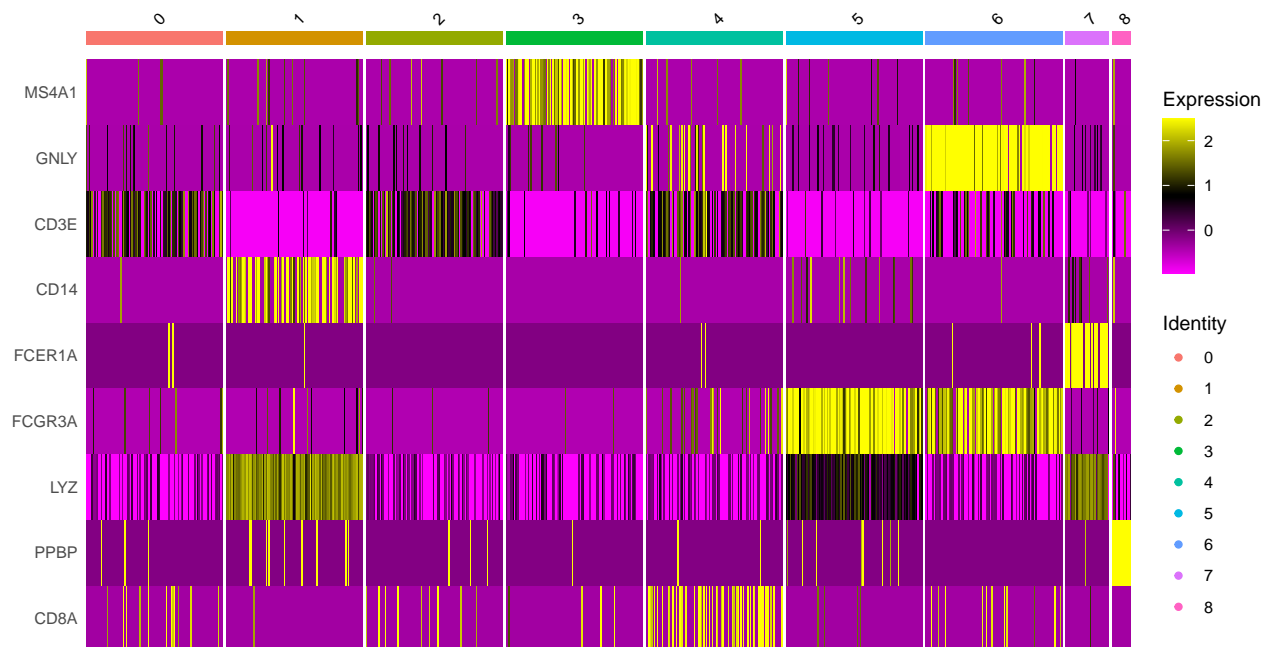


```
# Dot Plot- visualizes both the percentage of expressing cells (dot size) and the average expression level
# feature in each cluster. The color represents the average expression level
DotPlot(pbmc, features = feature_genes) + RotatedAxis()
```



Single cell heatmap of feature expression

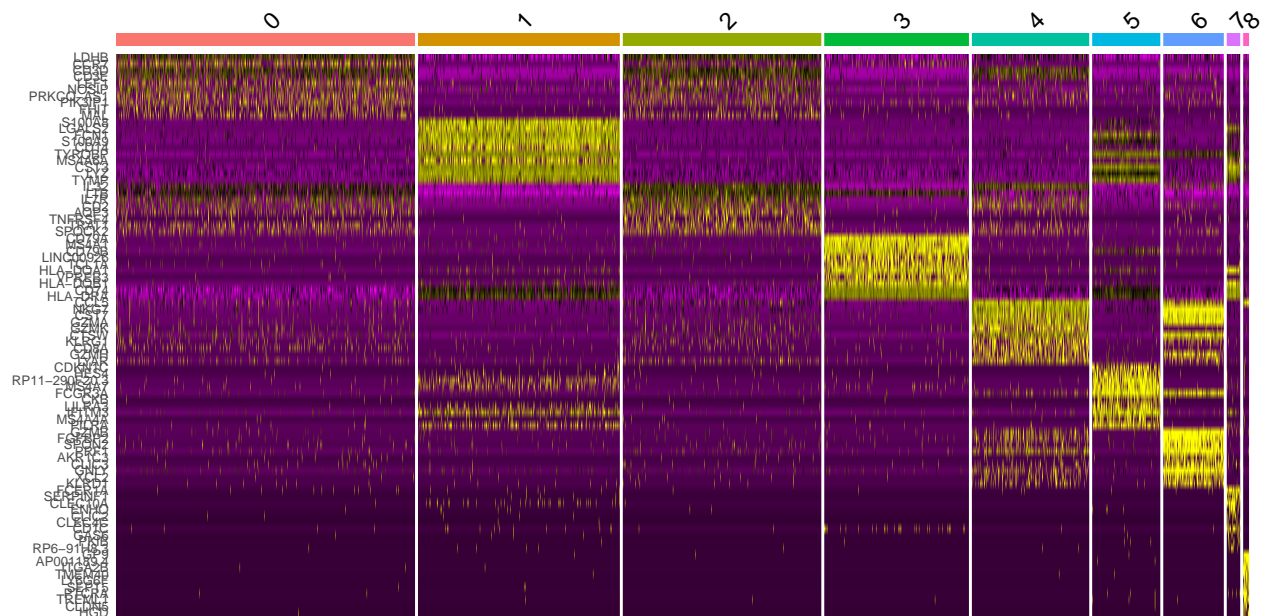
```
DoHeatmap(subset(pbm, downsample = 100), features = feature_genes, size = 3)
```



DoHeatmap() generates an expression heatmap for given cells and features. In this case, we are plotting the top 20 markers (or all markers if less than 20) for each cluster.

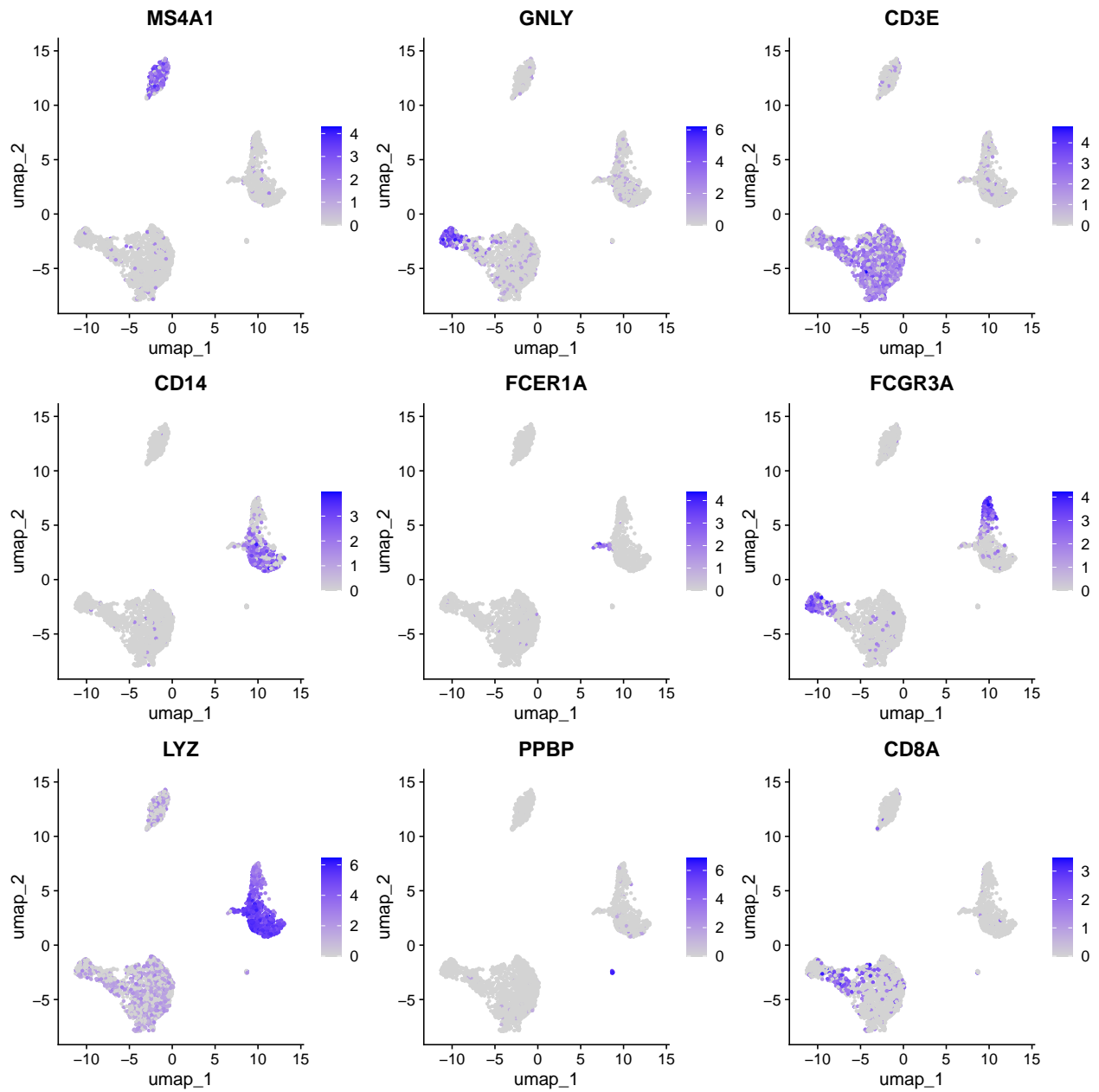
```
# Top 10 markers per cluster heatmap
```

```
pbmc.markers %>%
  group_by(cluster) %>%
  dplyr::filter(avg_log2FC > 1) %>%
  slice_head(n = 10) %>%
  ungroup() -> top10
DoHeatmap(pbmc, features = top10$gene) + NoLegend()
```



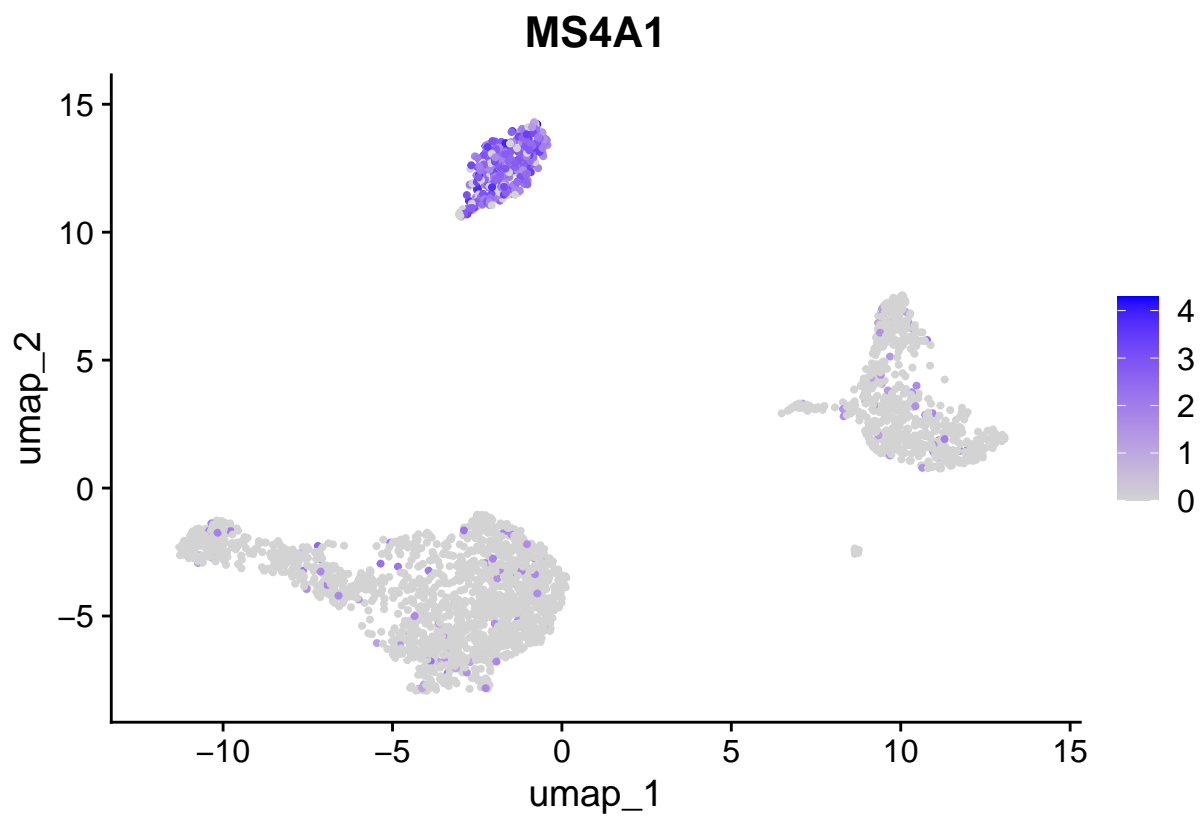
```
# Feature plot - visualize feature expression in low-dimensional space
```

```
FeaturePlot(pbmc, features = feature_genes)
```

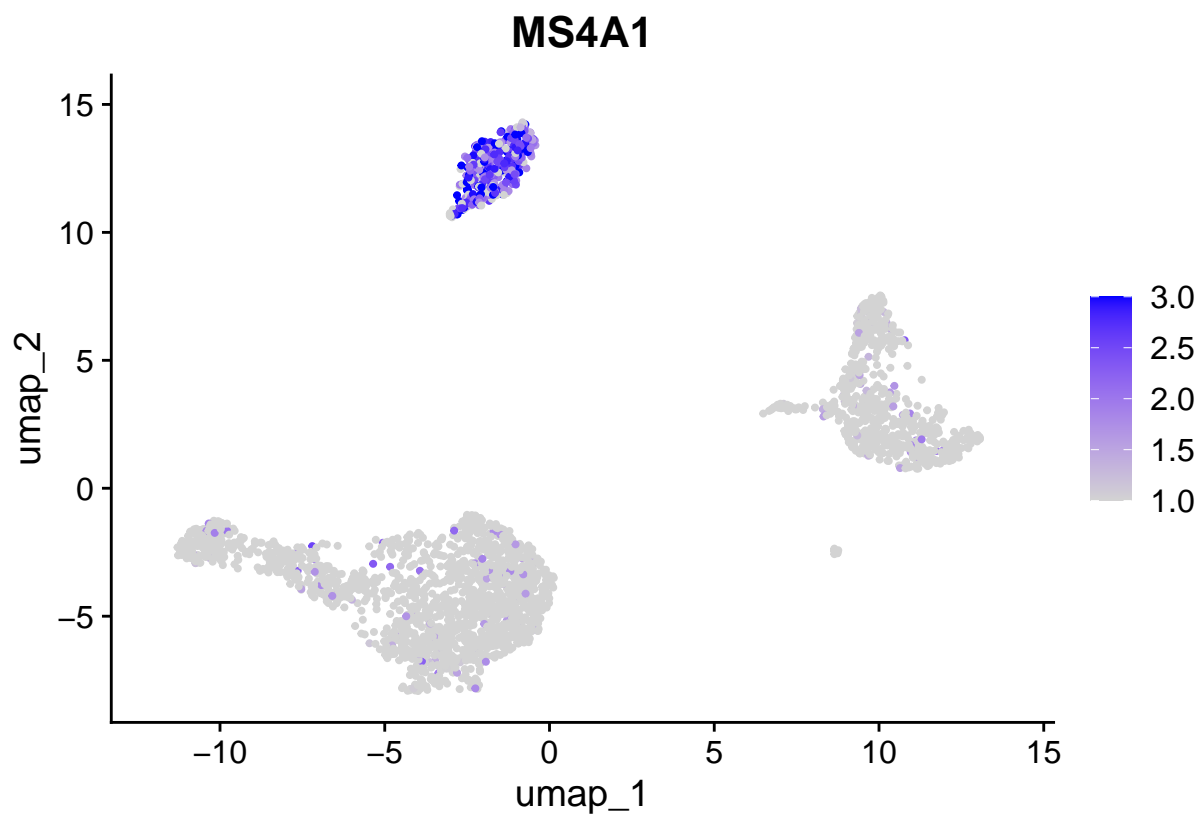



Additions to FeaturePlot

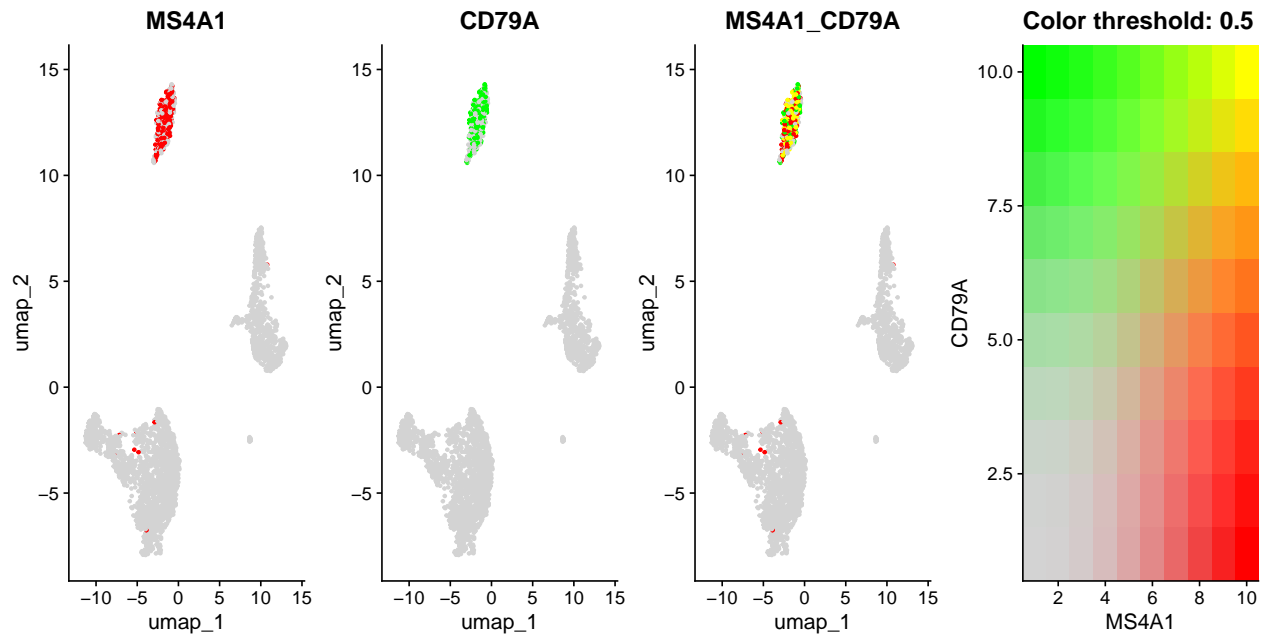
```
# Plot a legend to map colors to expression levels
FeaturePlot(pbm, features = "MS4A1")
```



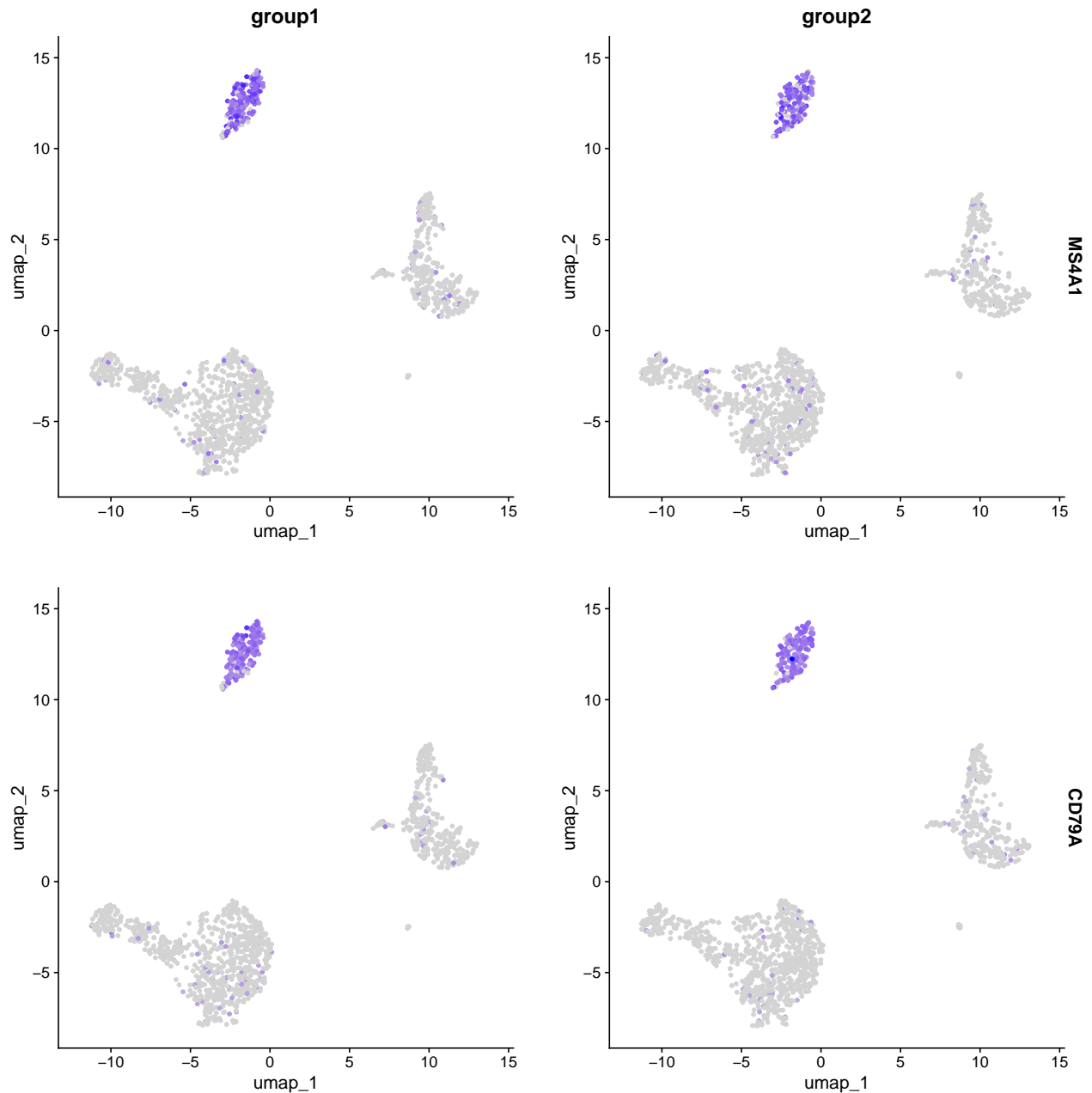
```
# Adjust the contrast in the plot  
FeaturePlot(pbmc, features = "MS4A1", min.cutoff = 1, max.cutoff = 3)
```



```
# Visualize co-expression of two features simultaneously
FeaturePlot(pbm, features = c("MS4A1", "CD79A"), blend = TRUE)
```

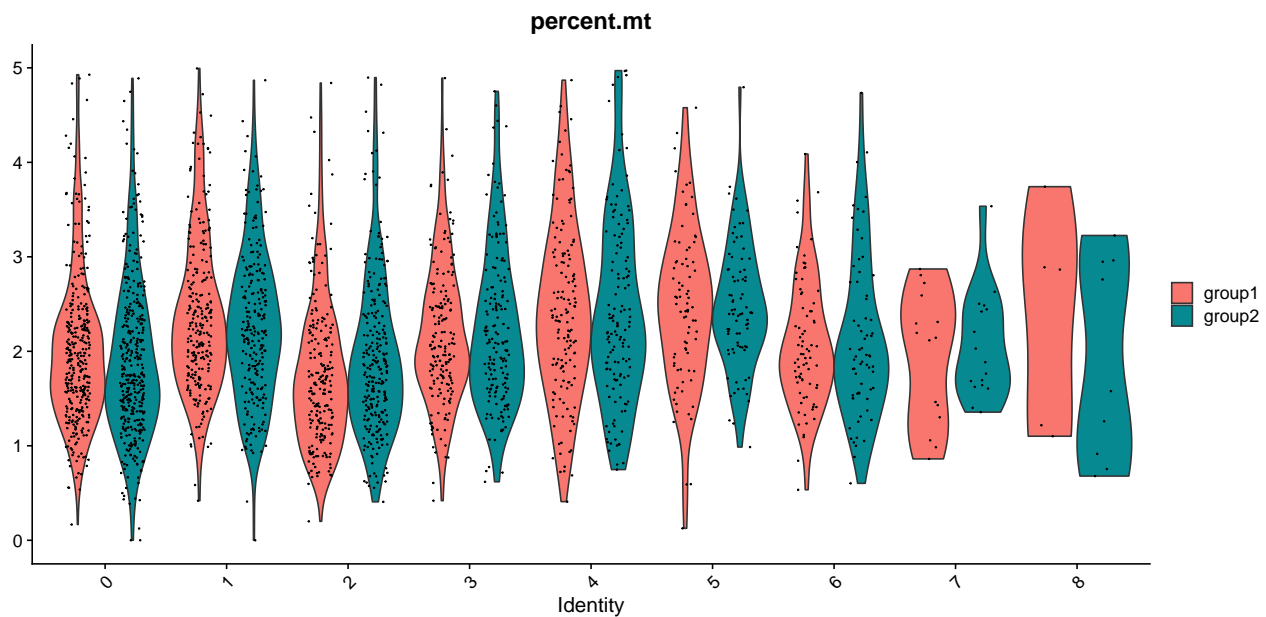


```
# Split visualization to view expression by groups (replaces FeatureHeatmap)
pbmc$groups <- sample(c("group1", "group2"), size = ncol(pbm), replace = TRUE)
FeaturePlot(pbm, features = c("MS4A1", "CD79A"), split.by = "groups")
```

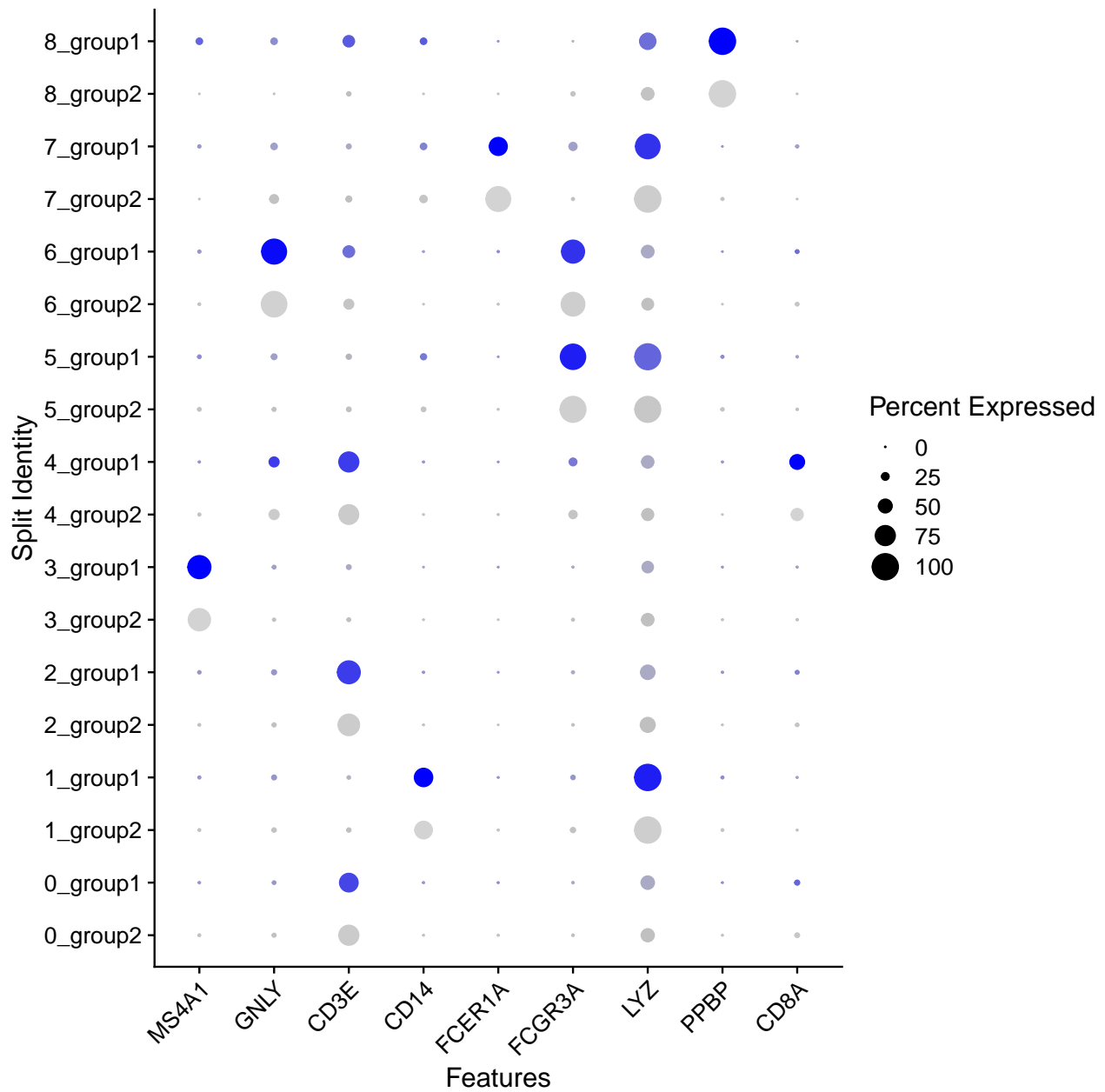


```
# Violin plots can also be split on some variable. Simply add the splitting variable to object
# metadata and pass it to the split.by argument
VlnPlot(pbmc, features = "percent.mt", split.by = "groups")
```

```
## The default behaviour of split.by has changed.
## Separate violin plots are now plotted side-by-side.
## To restore the old behaviour of a single split violin,
## set split.plot = TRUE.
##
## This message will be shown once per session.
```



SplitDotPlotGG has been replaced with the `split.by` parameter for DotPlot
 DotPlot(pbm, features = feature_genes, split.by = "groups") + RotatedAxis()



Applying themes to plots

With Seurat, all plotting functions return ggplot2-based plots by default, allowing one to easily capture and manipulate plots just like any other ggplot2-based plot.

```
library(ggplot2)

baseplot <- DimPlot(pbmcc, reduction = "umap")

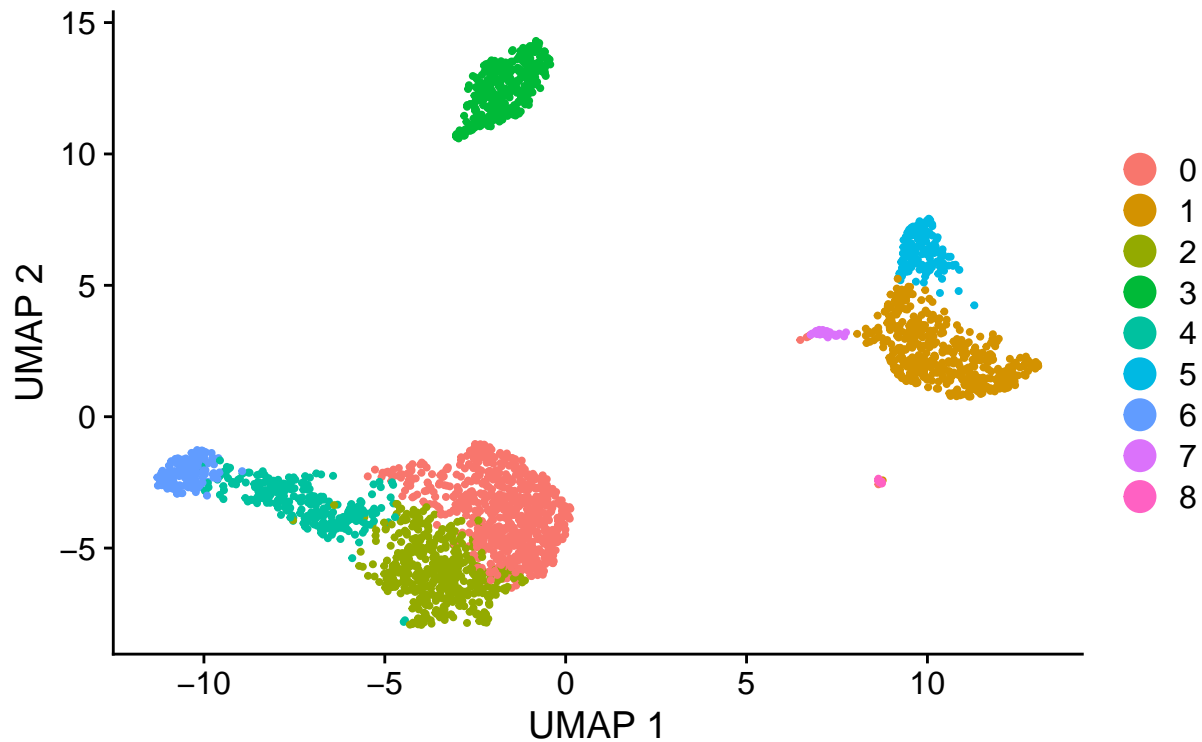
# Customize the plot using ggplot2 layers
baseplot +
  labs(title = "Clustering of 2,700 PBMCs", x = "UMAP 1", y = "UMAP 2") + # Add title and axis labels
  theme(
    plot.title = element_text(size = 18, face = "bold", hjust = 0.5), # Title font size, bold, cen
```

```

axis.title = element_text(size = 14),           # Axis label size
axis.text = element_text(size = 12),           # Tick label size
legend.title = element_text(size = 14),        # Legend title size
legend.text = element_text(size = 12)          # Legend item text size
) +
guides(colour = guide_legend(override.aes = list(size = 5))) # Increase dot size in legend

```

Clustering of 2,700 PBMCs



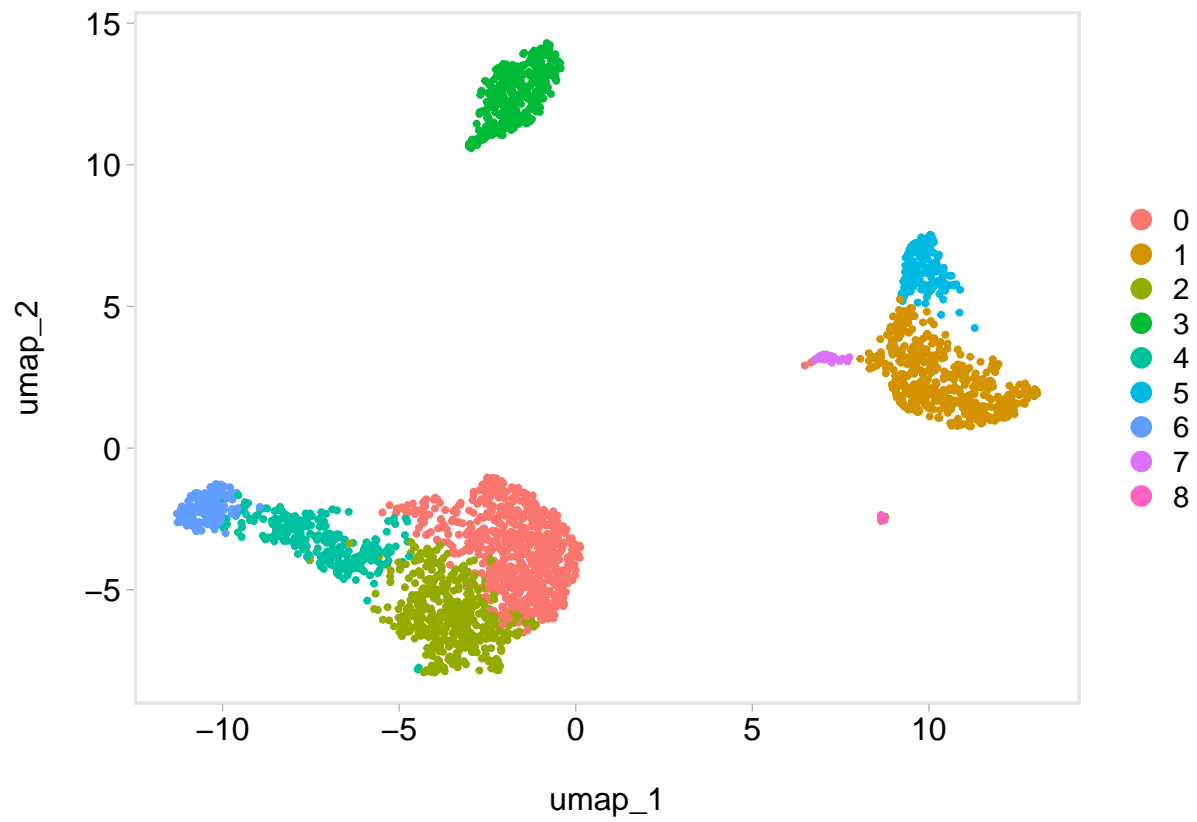
```

# Use community-created themes, overwriting the default Seurat-applied theme Install ggmin
# with remotes::install_github('sjessa/ggmin')

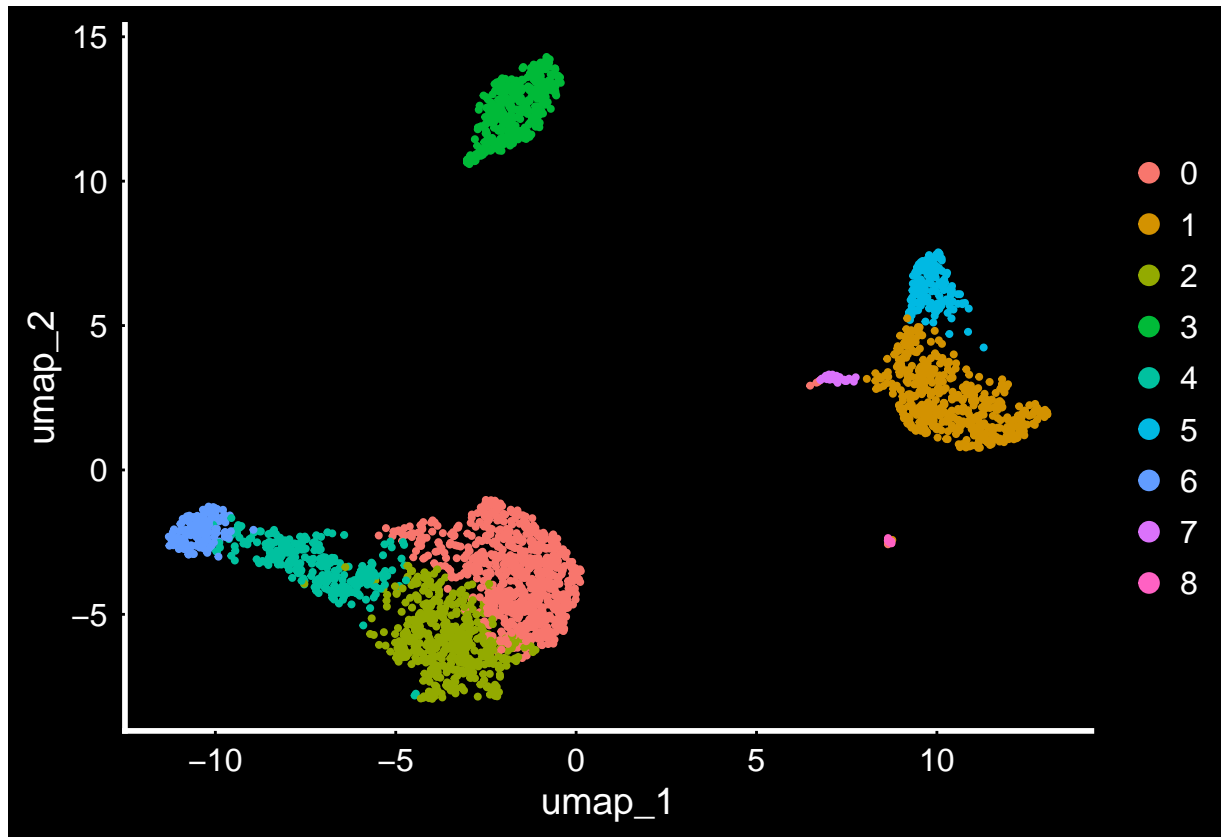
#remotes::install_github('sjessa/ggmin')

baseplot + ggmin::theme_powerpoint()

```



```
# Seurat also provides several built-in themes, such as DarkTheme; for more details see  
# ?SeuratTheme  
baseplot + DarkTheme()
```

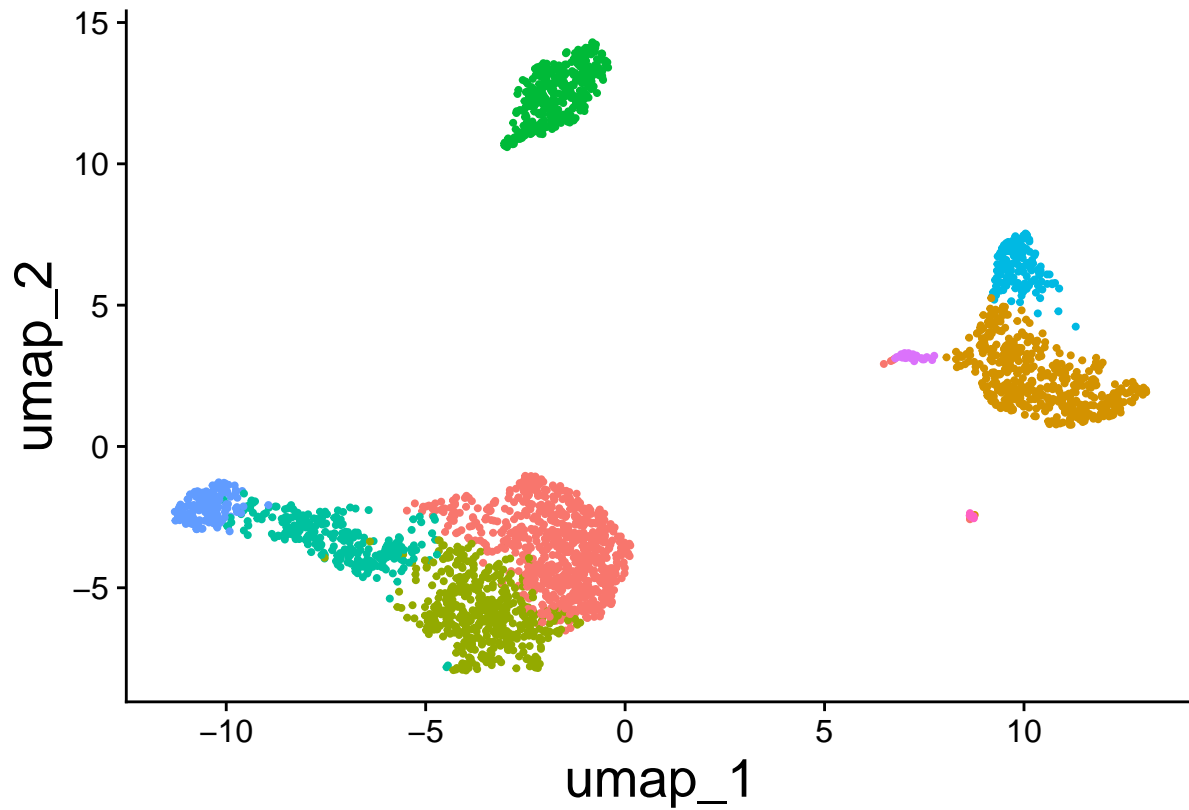



Other theme:

- `theme_minimal()` # Cleaner white background
- `theme_classic()` # No gridlines
- `theme_void()` # No axes at all (for minimalist look)

Chain themes together

```
baseplot + FontSize(x.title = 20, y.title = 20) + NoLegend()
```

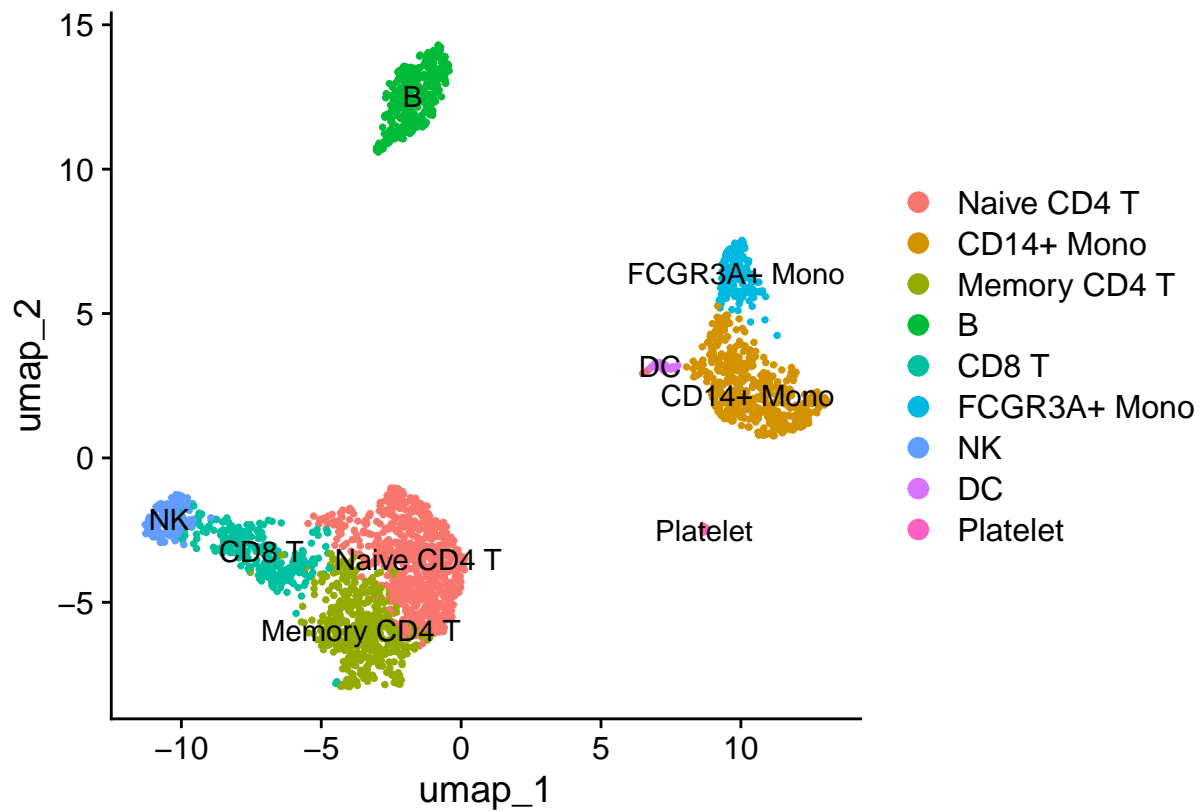


Assigning cell type identity to clusters

Fortunately in the case of this dataset, we can use canonical markers to easily match the unbiased clustering to known cell types:

Cluster ID	Markers	Cell Type
0	IL7R, CCR7	Naive CD4+ T
1	CD14, LYZ	CD14+ Mono
2	IL7R, S100A4	Memory CD4+ T
3	MS4A1	B
4	CD8A	CD8+ T
5	FCGR3A, MS4A7	FCGR3A+ Mono
6	GNLY, NKG7	NK
7	FCER1A, CST3	DC
8	PPBP	Platelet

```
new.cluster.ids <- c("Naive CD4 T", "CD14+ Mono", "Memory CD4 T", "B", "CD8 T", "FCGR3A+ Mono",
  "NK", "DC", "Platelet")
names(new.cluster.ids) <- levels(pbmcc)
pbmcc <- RenameIdents(pbmcc, new.cluster.ids)
DimPlot(pbmcc, reduction = "umap", label = TRUE, pt.size = 0.5) # + NoLegend()
```



```
library(ggplot2)
plot <- DimPlot(pbm, reduction = "umap", label = TRUE, label.size = 4.5) + xlab("UMAP 1") + ylab("UMAP 2")
theme(axis.title = element_text(size = 18), legend.text = element_text(size = 18)) + guides(colour = "celltype")
#ggsave(filename = "./pbmc3k_umap.jpg", height = 7, width = 12, plot = plot, quality = 50)
ggsave(filename = "pbmc3k_umap.jpg", height = 7, width = 12, plot = plot, quality = 50)

saveRDS(pbm, file = "./pbmc3k_final.rds")
```