Preliminary single cell analysis

Jieun Jeong

2025-02-16

This example analyzes data from a single cell experiment in one of the forms accepted by Seurat package, namely a directory with barcode.tsv, the list of cell identifiers, genes.tsv, the list of genes with Ensembl and Symbol rdentifiers, and matrix.mxt, counts (in 3rd column) of gene UMIs or fragments detected for each gene (number in 1st column, row number in genes.tsv) and each cell (number in 2nd column, row number in barcodes.tsv).

We create the initial Seurat object:

```
pa = "/Users/jieun/Work/Git_test/Seurat/data/filtered_gene_bc_matrices/hg19"
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
```

```
## 'SeuratObject' was built under R 4.4.0 but the current version is
## 4.4.2; it is recomended that you reinstall 'SeuratObject' as the ABI
## for R may have changed
```

```
## 'SeuratObject' was built with package 'Matrix' 1.7.0 but the current
## version is 1.7.2; it is recomended that you reinstall 'SeuratObject' as
## the ABI for 'Matrix' may have changed
```

```
##
## Attaching package: 'SeuratObject'
```

```
## The following objects are masked from 'package:base':
##
##     intersect, t
```

```
library(patchwork)
library(ggplot2)

pbmc.data <- Read10X(data.dir = pa)
# 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
```

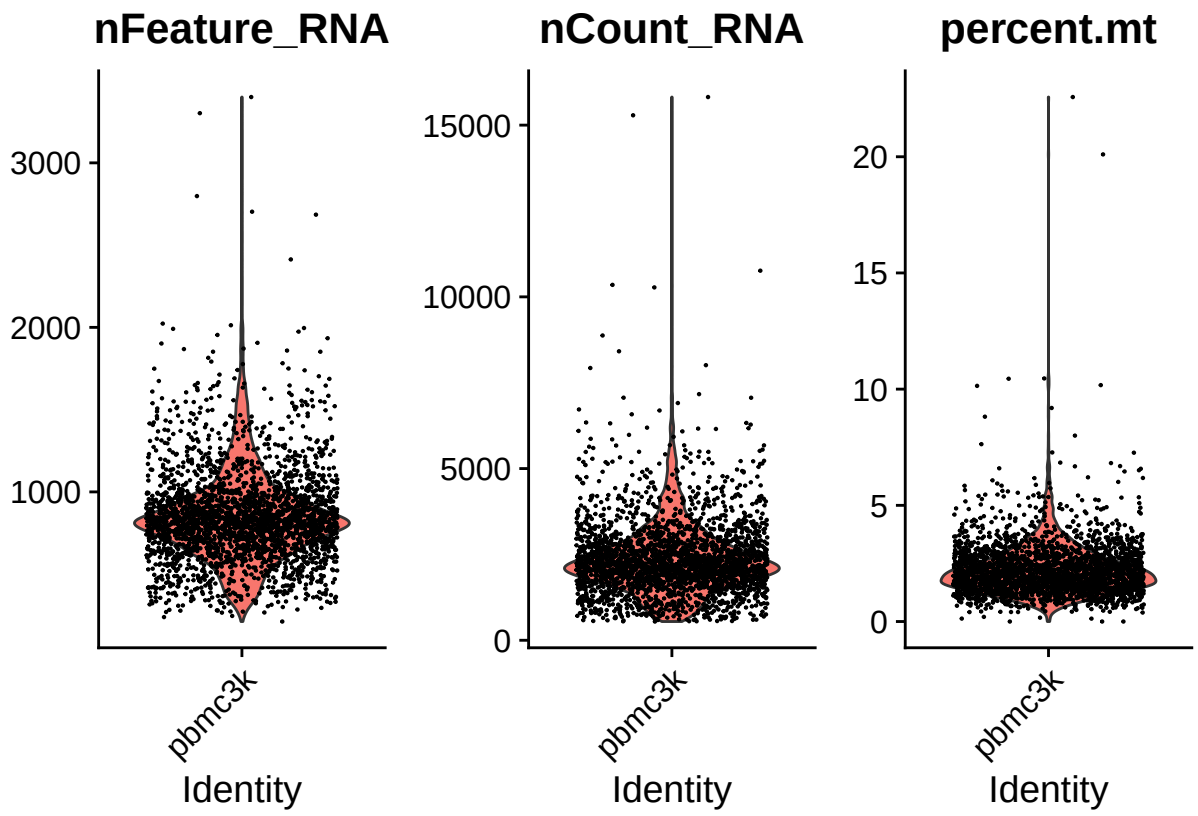
```
## An object of class Seurat
## 13714 features across 2700 samples within 1 assay
## Active assay: RNA (13714 features, 0 variable features)
##  1 layer present: counts
```

Quality contol metrics can be used to exclude some cells and gene from further analysis according to the distributions of the metrics in the data.

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

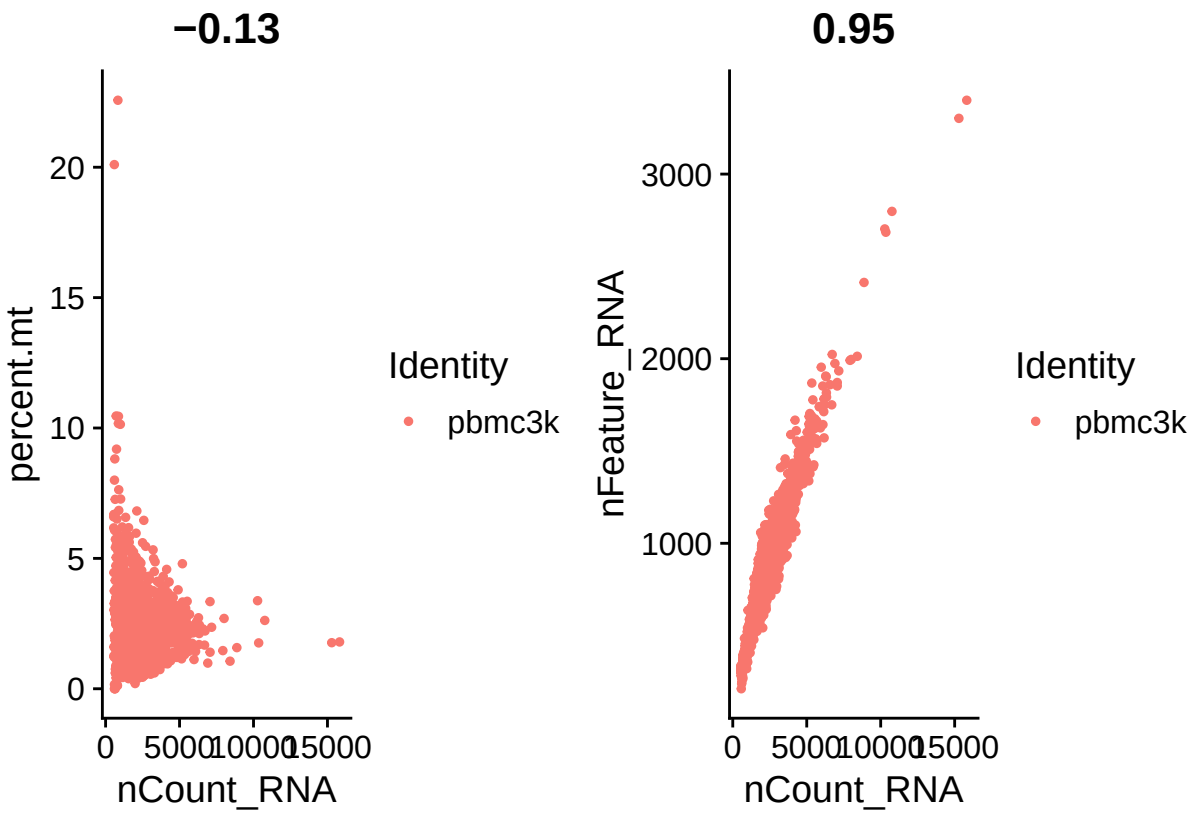
```
pbmc[["percent.mt"]] <- PercentageFeatureSet(pbmc, pattern = "^MT-")

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



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.

```
# Visualize QC metrics as a scatter plot
plot1 <- FeatureScatter(pbmc, feature1 = "nCount_RNA", feature2 = "percent.mt")
plot2 <- FeatureScatter(pbmc, feature1 = "nCount_RNA", feature2 = "nFeature_RNA")
plot1 + plot2
```



One may use median statistics as a guide to filtering parameters.

```
MEDnF <- median(pbmc$nFeature_RNA)
MEDmt <- median(pbmc$percent.mt)
sprintf("MEDnF = %d, MEDmt = %6.4f",MEDnF,MEDmt)
```

```
## [1] "MEDnF = 816, MEDmt = 2.0308"
```

A reasonable choice of limits on `nFeature` is at least $0.25 \times MEDnF$ and at most $3 \times MEDnF$. Similarly, one can require that `percent.mt` is at most $2.5 \times MEDmt$. One can revise those thresholds if the first round of the analysis provides reasons. Improvements in technology are increasing the counts per cell and may require to replace parts of the workflows to fully take advantage of those improvements, but this workflow is "traditional", predating 2022.

```
pbmc <- subset(pbmc, subset = nFeature_RNA > 0.25*MEDnF &
                 nFeature_RNA < 3*MEDnF & percent.mt < 2.5*MEDmt)
pbmc <- NormalizeData(pbmc) # log normalization and scaling
```

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

Improvements in technology are increasing the counts per cell and may require to replace parts of the workflows to fully take advantage of those improvements, but this workflow is "traditional", predating 2022. The next step identifies variable features, i.e. features that help group together similar cells and normalizes the data.

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

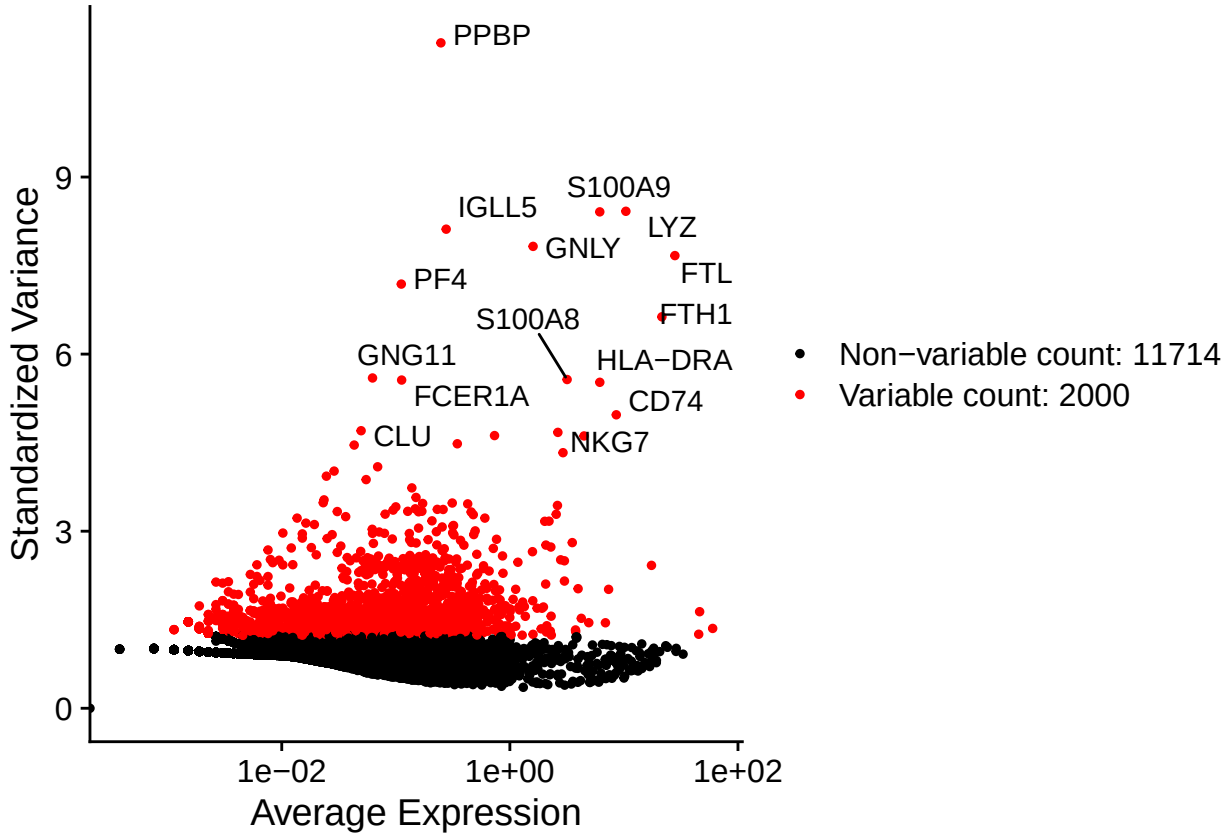
```
## Finding variable features for layer counts
```

```
# Identify most highly variable genes (illustration)
topf <- head(VariableFeatures(pbmc), 15)
# plot variable features with top labels
p <- VariableFeaturePlot(pbmc)
p <- LabelPoints(plot = p, points = topf, repel = TRUE)
```

```
## When using repel, set xnudge and ynudge to 0 for optimal results
```

```
p
```

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




```
# log normalization
allGenes <- rownames(pbmc)
pbmc <- ScaleData(pbmc, features = allGenes)
```

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

Now we run PCA and print a highlight of the result:

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

```
## PC_ 1
## Positive:  MALAT1, LTB, IL32, IL7R, CD2, B2M, ACAP1, STK17A, CTSW, CD247
##     GIMAP5, AQP3, CCL5, SELL, TRAF3IP3, GZMA, MAL, CST7, ITM2A, HOPX
##     MYC, GIMAP7, BEX2, LDLRAP1, GZMK, ETS1, ZAP70, TNFAIP8, RIC3, LYAR
## Negative:  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
## 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, TCL1A, HLA-DPA1
##     HLA-DRB1, PF4, HLA-DQA2, SDPR, HLA-DRA, LINC00926, PPBP, HLA-DRB5, GNG11, SPARC
##     GP9, AP001189.4, CA2, PTCRA, CD9, NRGN, RGS18, GZMB, CLU, TUBB1
## Negative:  VIM, IL7R, S100A6, S100A8, IL32, S100A4, GIMAP7, S100A10, S100A9, MAL
##     AQP3, CD14, CD2, FYB, LGALS2, GIMAP4, ANXA1, RBP7, FCN1, LYZ
##     S100A11, S100A12, GIMAP5, MS4A6A, FOLR3, TRABD2A, AIF1, IL8, TMSB4X, IFI6
## PC_ 5
## Positive:  LTB, IL7R, CKB, VIM, AQP3, MS4A7, CYTIP, RP11-290F20.3, SIGLEC10, HMOX1
##     MAL, PTGES3, LILRB2, HN1, CD2, CORO1B, ANXA5, GDI2, TUBA1B, FAM110A
##     TRADD, PPA1, ATP1A1, ABRACL, CCDC109B, CTD-2006K23.1, FYB, WARS, IL32, TRAF3IP3
## Negative:  GZMB, NKG7, FGFBP2, S100A8, GNLY, CCL4, CST7, PRF1, GZMA, SPON2
##     GZMH, S100A9, LGALS2, CCL3, CTSW, XCL2, CLIC3, CCL5, CD14, S100A12
##     RBP7, MS4A6A, GSTP1, FOLR3, IGFBP7, AKR1C3, TTC38, TYROBP, XCL1, HOPX
```

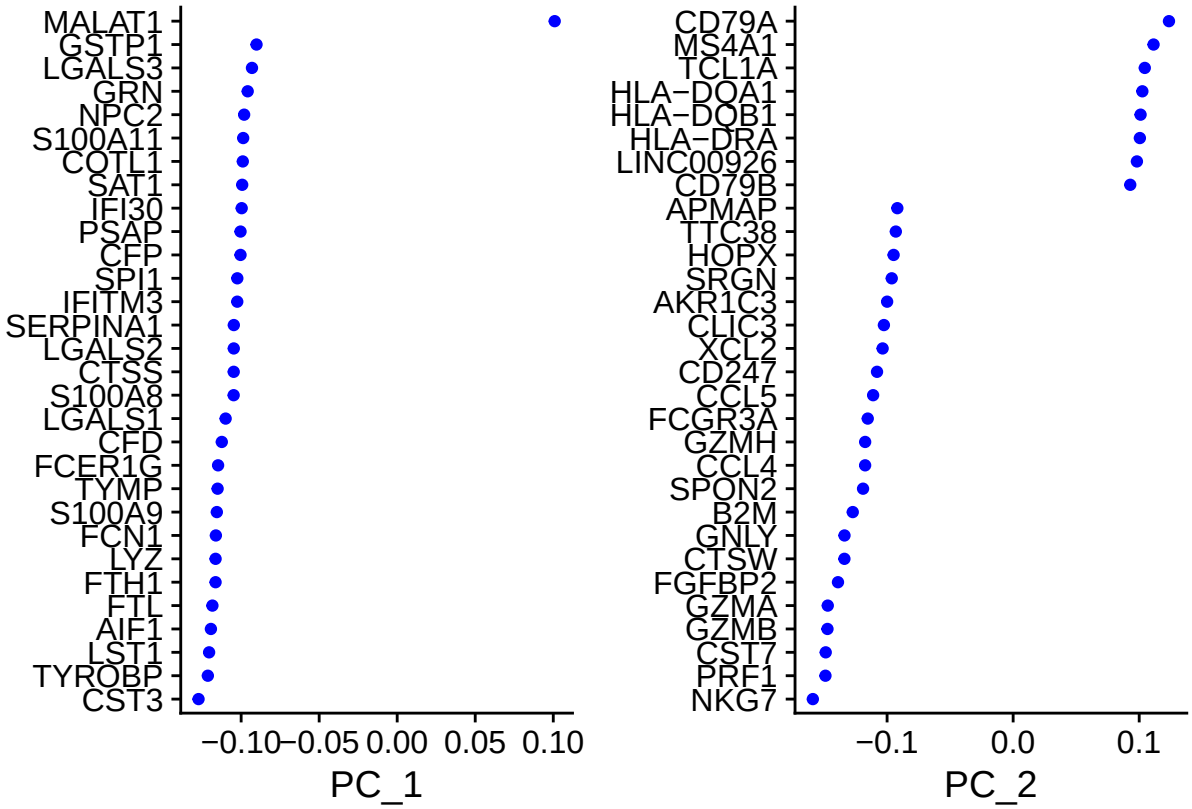
```
# Highlight: top genes in top PC dimensions
print(pbmc[["pca"]], dims = 1:5, nfeatures = 9)
```

```
## PC_ 1
## Positive:  MALAT1, LTB, IL32, IL7R, CD2, B2M, ACAP1, STK17A, CTSW
## Negative:  CST3, TYROBP, LST1, AIF1, FTL, FTH1, LYZ, FCN1, S100A9
## PC_ 2
## Positive:  CD79A, MS4A1, TCL1A, HLA-DQA1, HLA-DQB1, HLA-DRA, LINC00926, CD79B, HLA-DRB1
## Negative:  NKG7, PRF1, CST7, GZMB, GZMA, FGFBP2, CTSW, GNLY, B2M
## PC_ 3
## Positive:  HLA-DQA1, CD79A, CD79B, HLA-DQB1, HLA-DPB1, HLA-DPA1, CD74, MS4A1, HLA-DRB1
## Negative:  PPBP, PF4, SDPR, SPARC, GNG11, NRGN, GP9, RGS18, TUBB1
## PC_ 4
## Positive:  HLA-DQA1, CD79B, CD79A, MS4A1, HLA-DQB1, CD74, HLA-DPB1, HIST1H2AC, TCL1A
```

```
## Negative:  VIM, IL7R, S100A6, S100A8, IL32, S100A4, GIMAP7, S100A10, S100A9
## PC_ 5
## Positive:  LTB, IL7R, CKB, VIM, AQP3, MS4A7, CYTIP, RP11-290F20.3, SIGLEC10
## Negative:  GZMB, NKG7, FGFBP2, S100A8, GNLY, CCL4, CST7, PRF1, GZMA
```

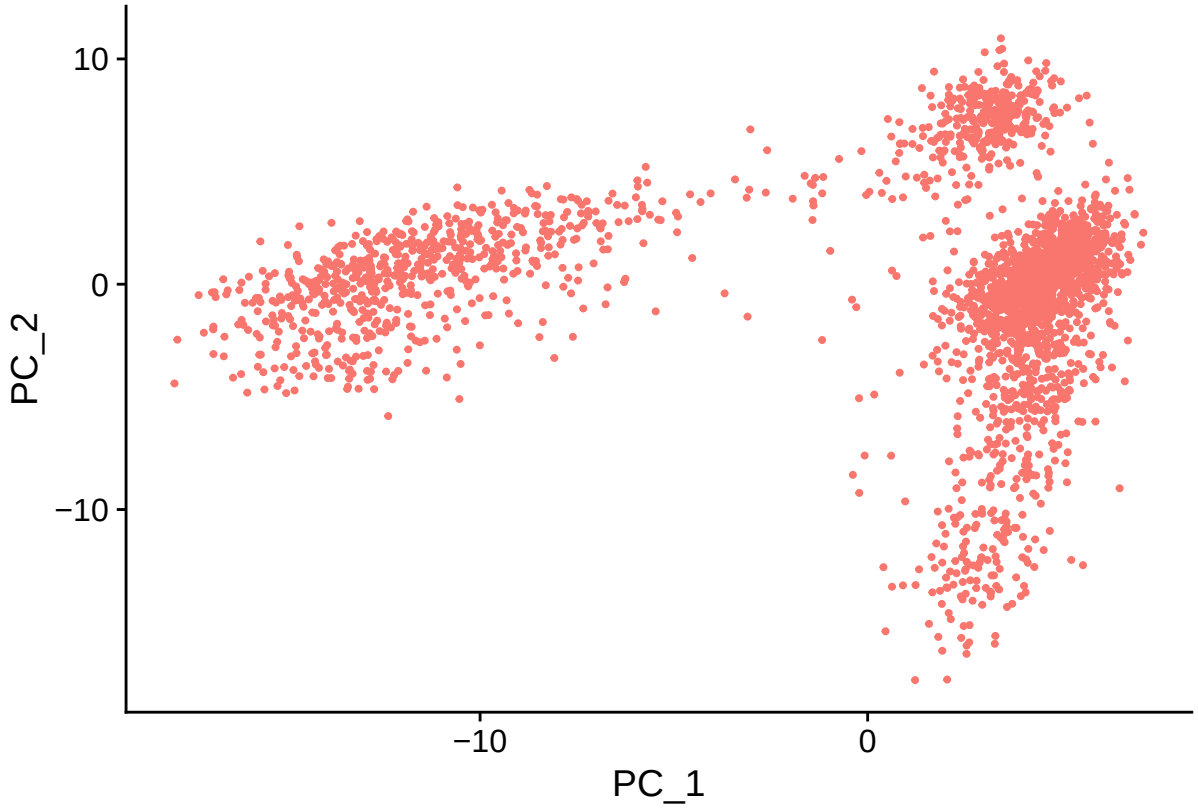
The coefficients of PC's can be visualized too:

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



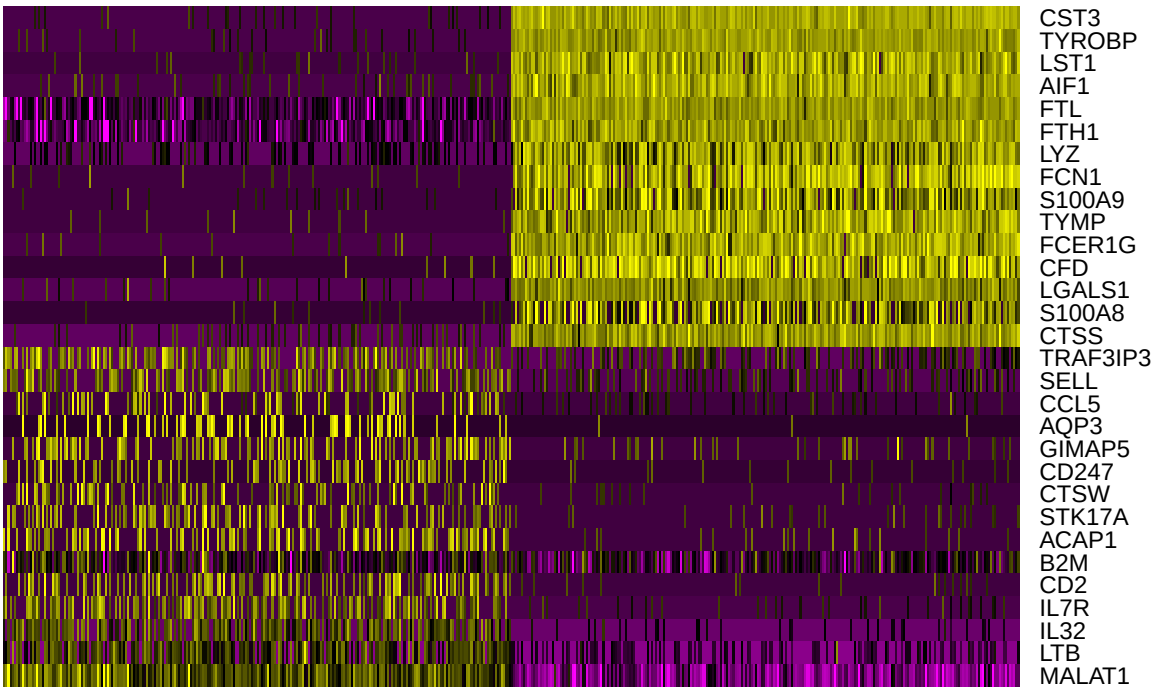
We can also see a standard placement of cells in 2D diagram before clustering

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



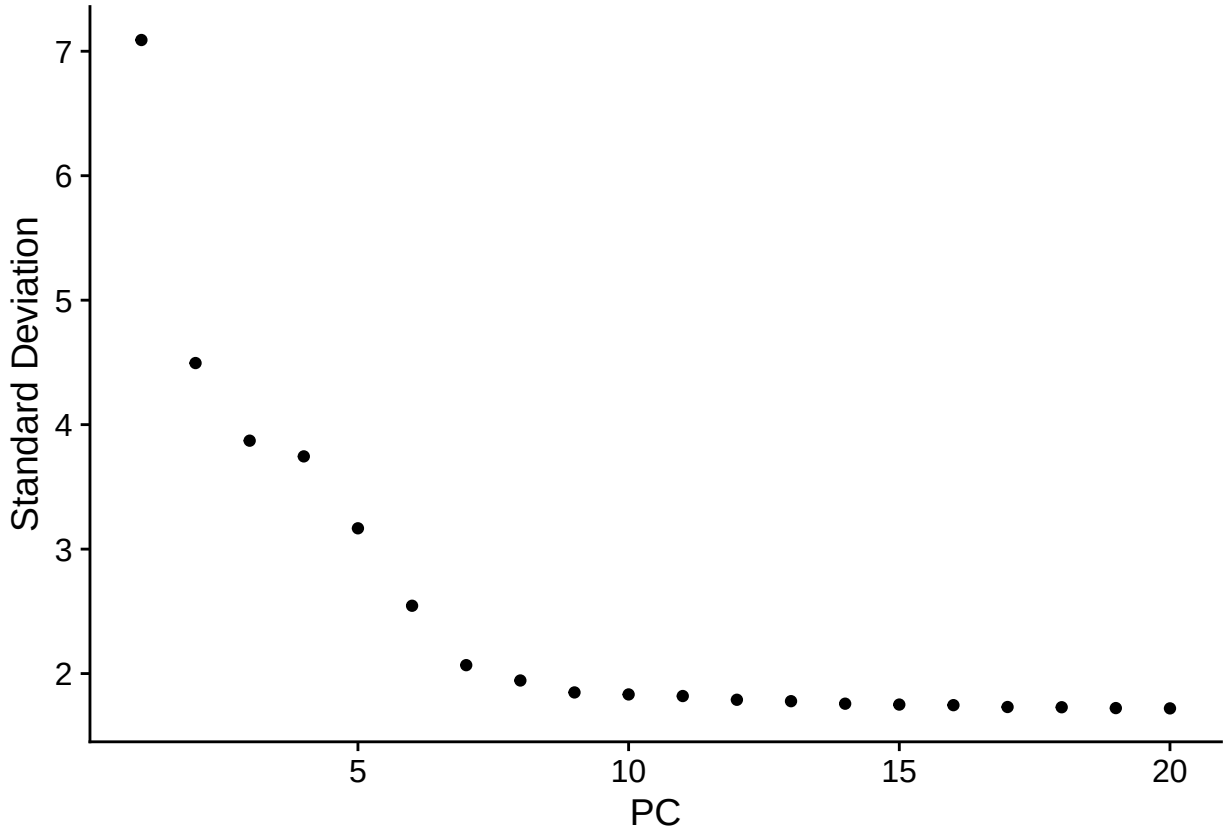
One can also inspect how genes with high coefficient in a PC behave across cells

```
DimHeatmap(pbmc, dims = 1, cells = 500, balanced = TRUE)
```



In this data sets, starting from PC_10 it is hard to see distinct patterns, so perhaps these dimensions are not important. While it is even more clear in Elbow Plot below, 9 or 10 does nor need to be the best number, and Seurat designers encourage to compare results when we use more dimensions.

```
ElbowPlot(pbmc)
```



Now it is the time to prepare the main dish: the clusters!

```
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: 2640
## Number of edges: 95937
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.8727
## Number of communities: 9
## Elapsed time: 0 seconds
```

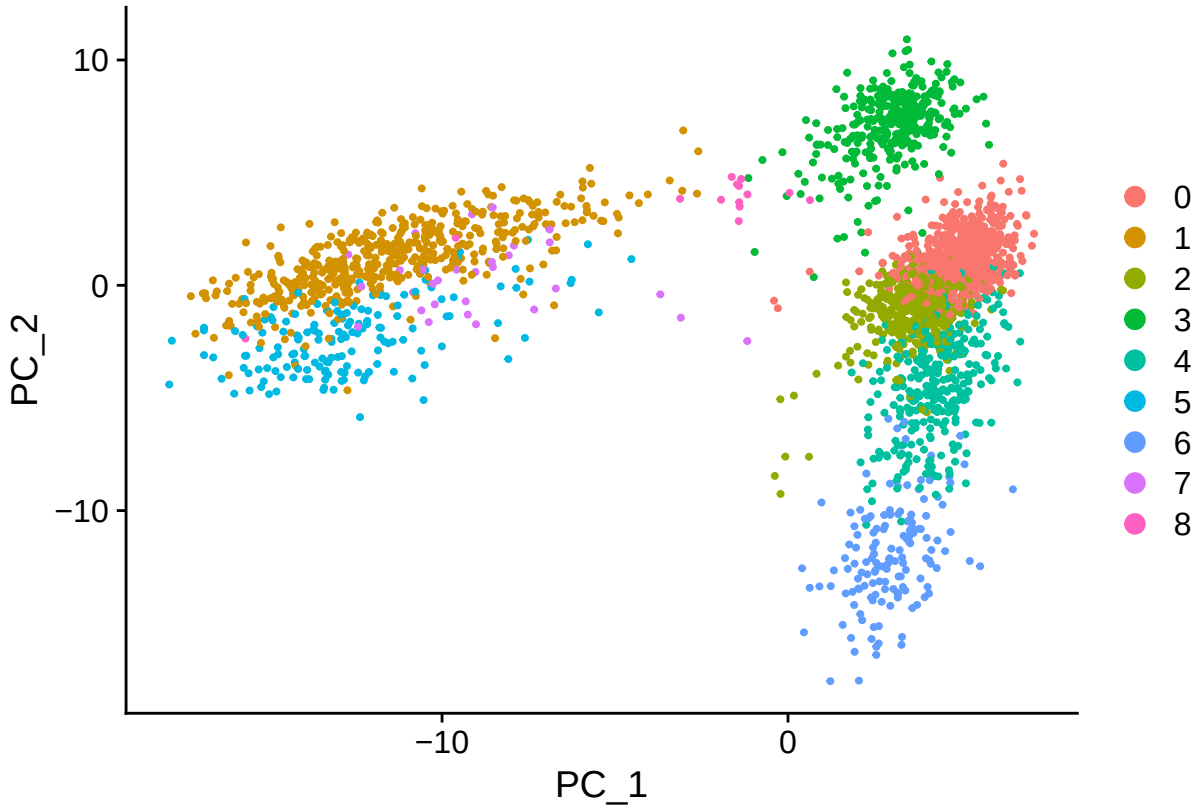
We got clusters numbered from 0, and sizes can be seen simply:

```
table(pbmc$seurat_clusters)
```

```
##
##   0   1   2   3   4   5   6   7   8
## 734 481 420 344 308 162 145  32  14
```

We can also see positions cells according to PC_1 and PC_2

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



but usually a better view is obtained with non-linear coordinates of UMAP. Moreover, after matching clusters with cell types, we can make meanigful legened with the LabelClusters function.

```
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
```

```
## 18:27:01 UMAP embedding parameters a = 0.9922 b = 1.112
```

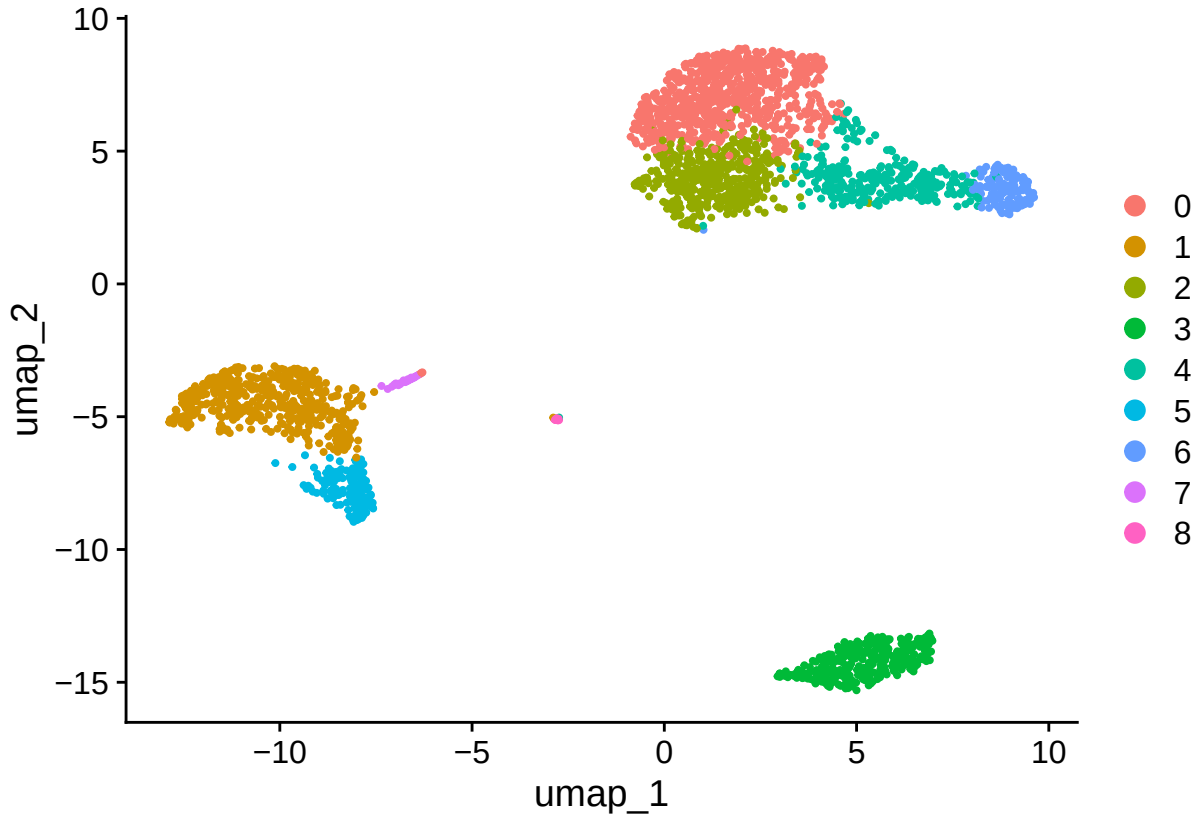
```
## 18:27:01 Read 2640 rows and found 10 numeric columns
```

```
## 18:27:01 Using Annoy for neighbor search, n_neighbors = 30
```

```
## 18:27:01 Building Annoy index with metric = cosine, n_trees = 50
```

```
## 0%   10   20   30   40   50   60   70   80   90   100%

## [----|----|----|----|----|----|----|----|----|----|

## **************************************************|
## 18:27:01 Writing NN index file to temp file /var/folders/72/snzm4vjs6rv_kqq4xv4yrd_w0000gn/T//RtmpcL
## 18:27:01 Searching Annoy index using 1 thread, search_k = 3000
## 18:27:01 Annoy recall = 100%
## 18:27:02 Commencing smooth kNN distance calibration using 1 thread with target n_neighbors = 30
## 18:27:02 Initializing from normalized Laplacian + noise (using RSpectra)
## 18:27:02 Commencing optimization for 500 epochs, with 105138 positive edges
## 18:27:04 Optimization finished
```

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



Markers of cluster allow to identify the respective cell type base on prior knowledge. The function FindwMarkers() allows to find differential genes of every individual cluster against the rest of the cells, but it also allows to find differential genes for any pair of groups of cluster. That may be helpful if we want to find a classifier of all types of cells. Of course, it is not productive to compare every pair of cluster groups, but a figures like two previus one hint what is worth checking.

```
# find all markers of cluster 2
cluster2.markers <- FindMarkers(pbmc, ident.1 = 2)
```

```
## For a (much!) faster implementation of the Wilcoxon Rank Sum Test,
```

```
## (default method for FindMarkers) please install the presto package
## --------------------------------------------
## install.packages('devtools')
## devtools::install_github('immunogenomics/presto')
## --------------------------------------------
## After installation of presto, Seurat will automatically use the more
## efficient implementation (no further action necessary).
## This message will be shown once per session
```

```
head(cluster2.markers, n = 5)
```

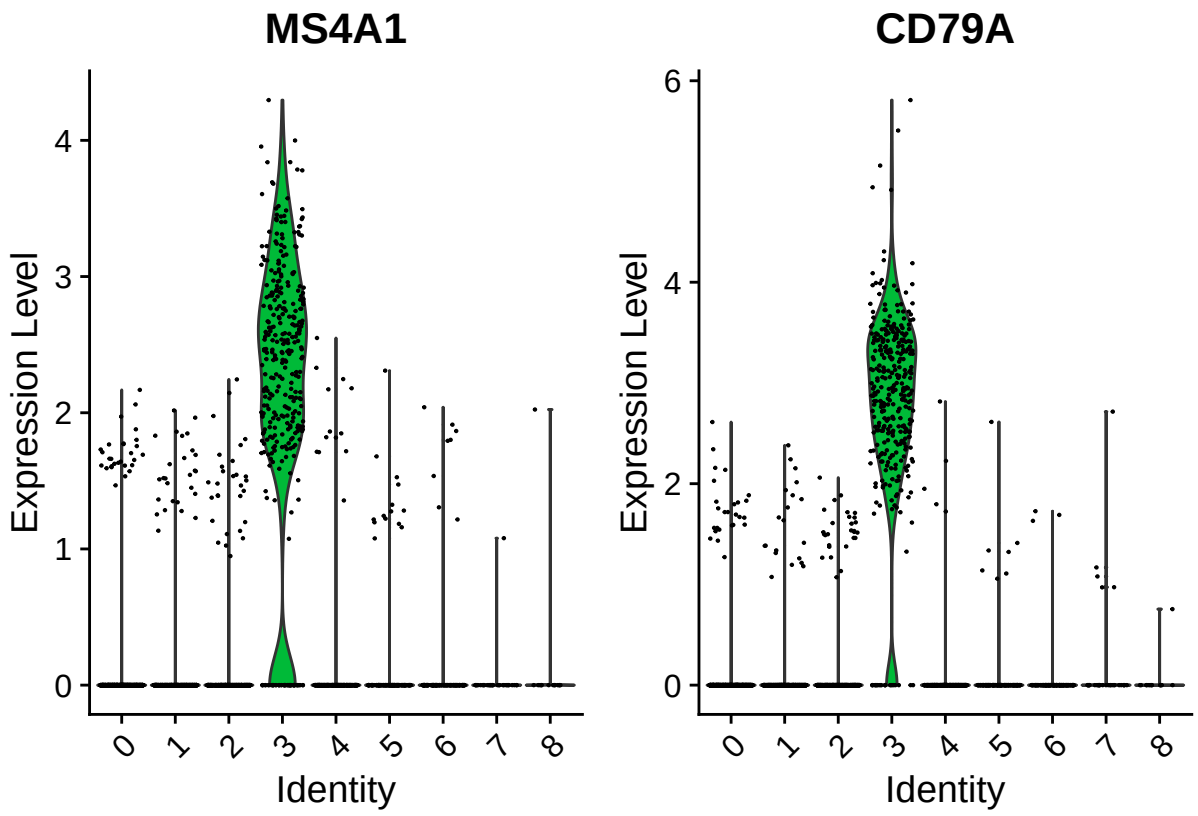
```
##                p_val avg_log2FC pct.1 pct.2    p_val_adj
## IL32    9.265679e-80  1.2375474 0.952 0.477 1.270695e-75
## LTB     7.311890e-79  1.3363006 0.983 0.651 1.002753e-74
## CD3D    3.098892e-63  1.0575939 0.921 0.444 4.249821e-59
## TNFRSF4 1.337503e-60  3.2991377 0.231 0.026 1.834251e-56
## LDHB    4.289876e-59  0.9686025 0.962 0.621 5.883137e-55
```

```
cluster5.markers <- FindMarkers(pbmc,ident.1 = 5, ident.2 = c(1,7))
head(cluster5.markers, n = 5)
```

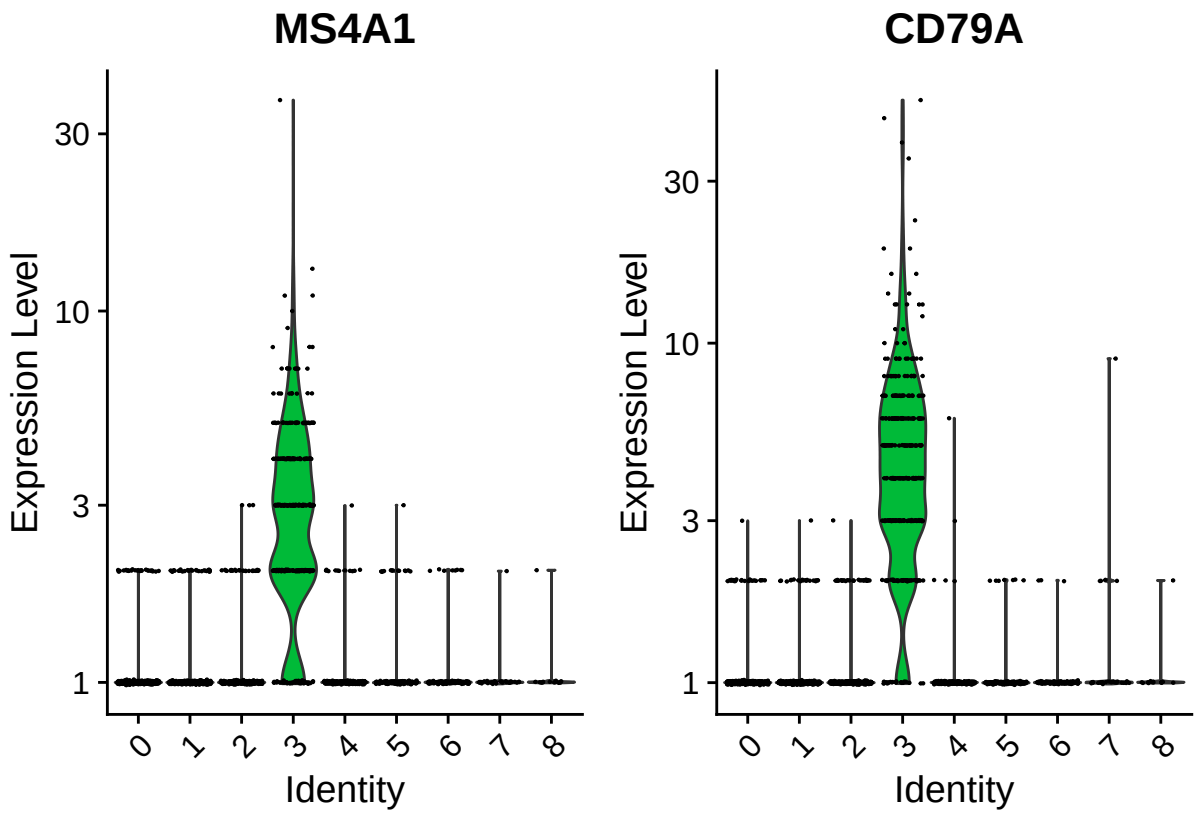
```
##                p_val avg_log2FC pct.1 pct.2     p_val_adj
## FCGR3A 2.568543e-105   5.073180 0.975 0.133 3.522500e-101
## LYZ     1.936836e-74  -2.626094 0.988 0.998  2.656177e-70
## RHOC    9.864456e-69   3.292917 0.864 0.177  1.352812e-64
## IFITM2  4.725826e-65   2.213429 1.000 0.692  6.480998e-61
## RPS19   1.138935e-59   1.088866 1.000 0.990  1.561935e-55
```

Some markers are common to more than one cluster, and some are "negative", so it is interesting how they distribute among cluster. We can find it using three approaches: (a) violin plot of expression levels in every cluster (b) plot of presence in UMAP diagram (c) table of summary statistic of presence in clusters. The last approach is best when we have many clusters, e.g. 25 or more but the first two may allow for quick insights. Method (c) may require a short script, but (a) and (b) have Seurat functions.

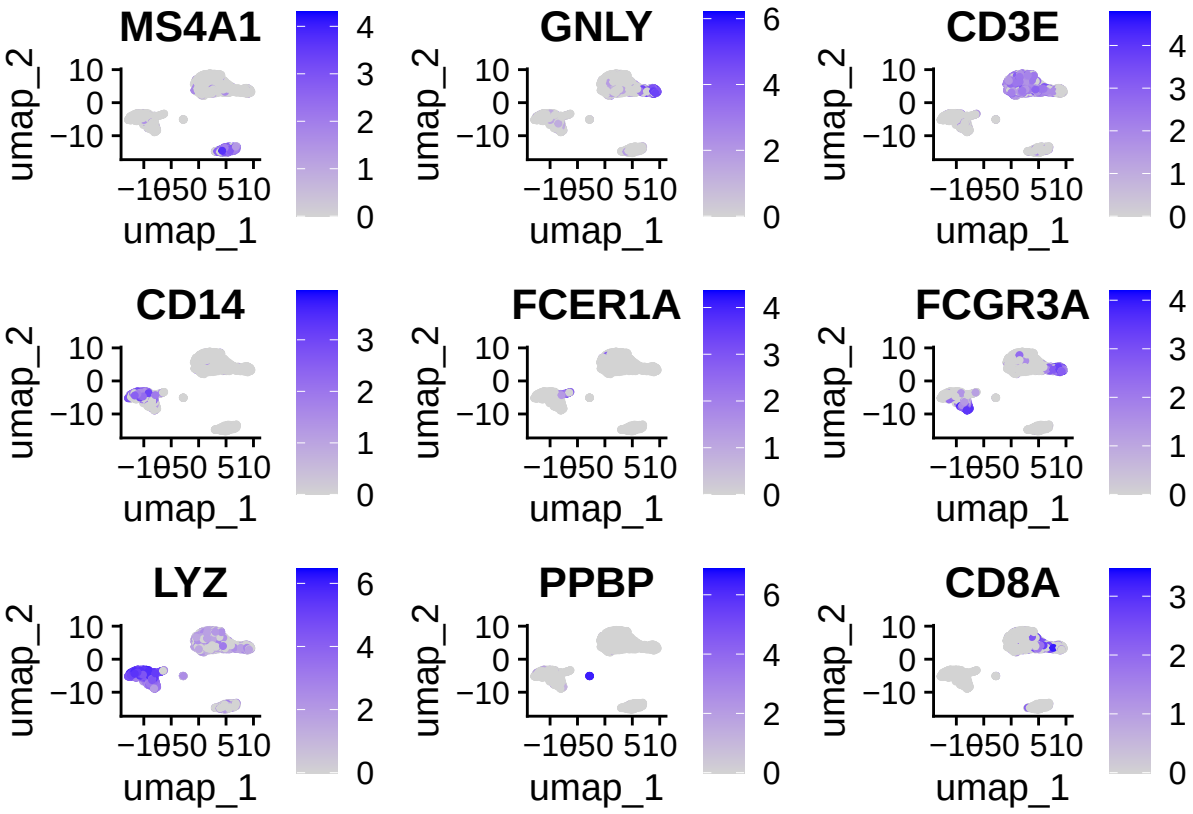
```
VlnPlot(pbmc, features = c("MS4A1", "CD79A"))
```




```
VlnPlot(pbmc, features = c("MS4A1", "CD79A"), layer = "counts", log = TRUE)
```

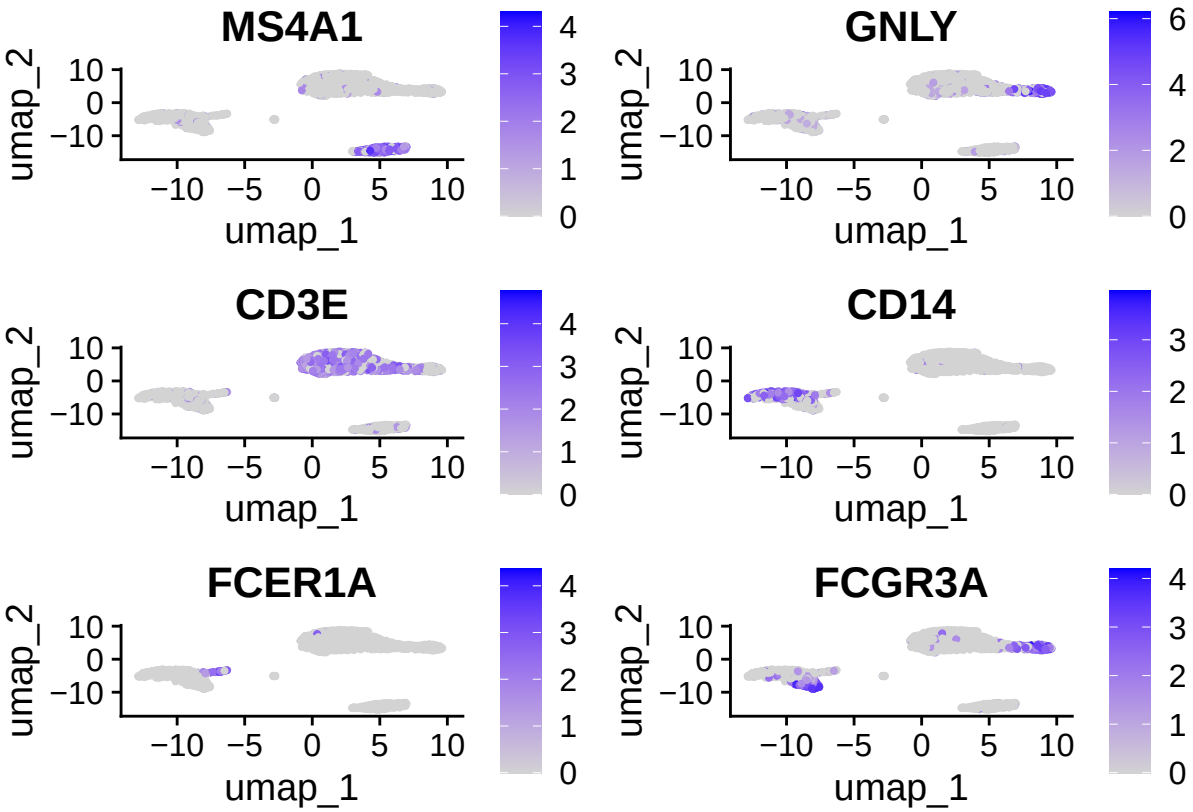



```
FeaturePlot(pbmc, features = c("MS4A1", "GNLY", "CD3E", "CD14", "FCER1A",
                               "FCGR3A", "LYZ", "PPBP","CD8A"))
```



The last plot has too many facets, we may inspect markers in smaller batches:

```
FeaturePlot(pbmc, features = c("MS4A1", "GNLY", "CD3E", "CD14", "FCER1A",
                               "FCGR3A"))
```



Finally, we can have a heatmap of top markers. To read gene names, see it in a separate window and stretch vertically.

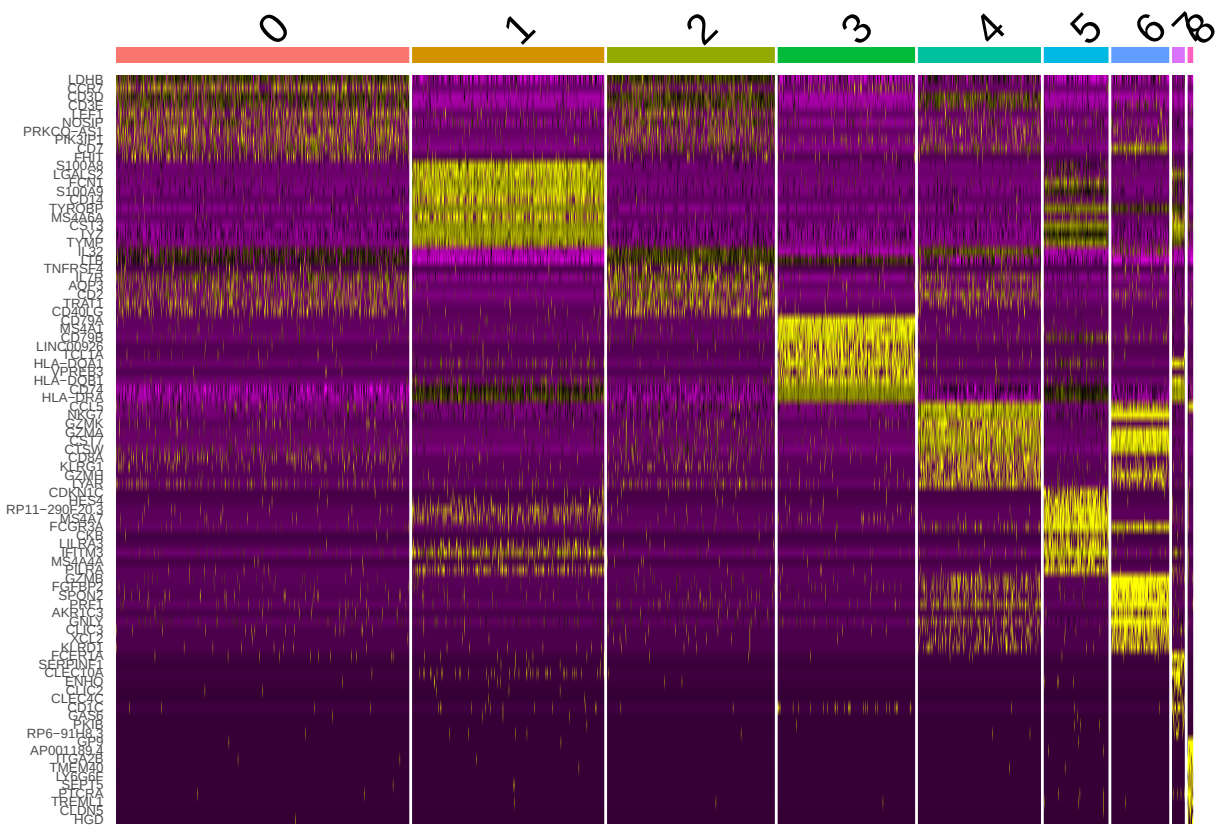
```
# find markers for every cluster compared to all remaining cells,
# report only the positive ones
pbmc.markers <- FindAllMarkers(pbmc, only.pos = TRUE)
```

```
## 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
```

```r
pbmc.markers %>%
    group_by(cluster) %>%
    dplyr::filter(avg_log2FC > 1 & -log10(p_val_adj) > 5)
```

```
## # A tibble: 2,230 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 3.15e-123       1.25 0.91  0.584 4.33e-119 0       LDHB
##  2 1.03e- 87       2.44 0.436 0.104 1.41e- 83 0       CCR7
##  3 2.00e- 82       1.07 0.847 0.393 2.74e- 78 0       CD3D
##  4 1.64e- 61       1.08 0.737 0.389 2.25e- 57 0       CD3E
##  5 4.50e- 52       2.13 0.335 0.1   6.18e- 48 0       LEF1
##  6 1.08e- 51       1.24 0.631 0.351 1.47e- 47 0       NOSIP
##  7 5.03e- 49       2.10 0.335 0.104 6.90e- 45 0       PRKCQ-AS1
##  8 3.20e- 48       1.57 0.444 0.177 4.38e- 44 0       PIK3IP1
##  9 3.14e- 44       1.01 0.582 0.285 4.31e- 40 0       CD7
## 10 8.47e- 44       2.96 0.199 0.037 1.16e- 39 0       FHIT
## # i 2,220 more rows
```

```r
pbmc.markers %>%
    group_by(cluster) %>%
    dplyr::filter(avg_log2FC > 1) %>%
    slice_head(n = 10) %>%
    ungroup() -> top10
DoHeatmap(pbmc, features = top10$gene) + NoLegend() +
  theme(axis.text.y = element_text(size = 5))
```



We see that top 10 markers poorly differenciate between clusters 0, 2 and 4, so we can search for alternatives using powerful subset() function of Seurat.

```
pbmc_024 <- subset(pbmc, idents = c(0, 2, 4))
pbmc_024.markers <- FindAllMarkers(pbmc_024, only.pos = TRUE)
```

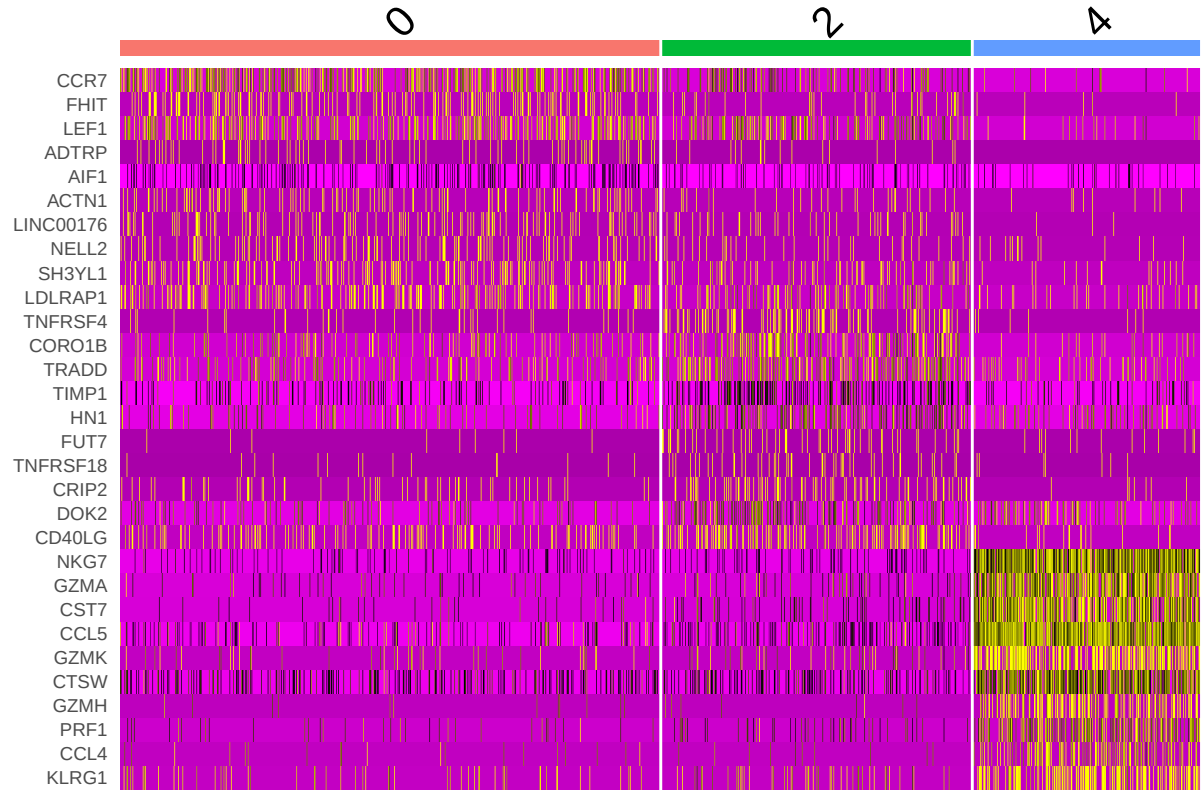
```
## Calculating cluster 0

## Calculating cluster 2

## Calculating cluster 4
```

```
pbmc_024.markers %>%
    group_by(cluster) %>%
    dplyr::filter(avg_log2FC > 1 & -log10(p_val_adj) > 5)
```

```
## # A tibble: 111 x 7
## # Groups:   cluster [3]
##       p_val avg_log2FC pct.1 pct.2 p_val_adj cluster gene
##       <dbl>      <dbl> <dbl> <dbl>     <dbl> <fct>   <chr>
##  1 6.97e-35       1.88 0.436 0.162  9.56e-31 0       CCR7
##  2 1.09e-17       2.44 0.199 0.055  1.49e-13 0       FHIT
##  3 2.77e-11       1.11 0.335 0.198  3.79e- 7 0       LEF1
##  4 3.31e-10       2.53 0.102 0.023  4.54e- 6 0       ADTRP
##  5 3.25e-28       2.57 0.231 0.042  4.46e-24 2       TNFRSF4
```

```
##  6 6.04e-20       1.85 0.324 0.117  8.29e-16 2       CORO1B
##  7 8.67e-19       1.09 0.383 0.155  1.19e-14 2       TRADD
##  8 5.17e-18       1.07 0.386 0.161  7.08e-14 2       TIMP1
##  9 1.87e-16       1.02 0.338 0.134  2.57e-12 2       HN1
## 10 6.89e-16       2.48 0.124 0.021  9.44e-12 2       FUT7
## # i 101 more rows
```

```
pbmc_024.markers %>%
    group_by(cluster) %>%
    dplyr::filter(avg_log2FC > 1) %>%
    slice_head(n = 10) %>%
    ungroup() -> top10
DoHeatmap(pbmc_024, features = top10$gene) + NoLegend() +
  theme(axis.text.y = element_text(size = 7))
```



Now cluster 4 is clearly differenciated, and for 4 vs 6 we can repeat the aproach.