Computational single-cell biology course

Marc Jan Bonder (m.bonder@dkfz.de) and Hakime Öztürk (h.oeztuerk@dkfz-heidelberg.de)

28 April, 2022

Contents

1 Dimensionality reduction and clustering on scRNA-seq data

1.1 Seurat object setup

```
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(ggplot2)
library(Seurat)
library(patchwork)
```

We will use Seurat's example dataset of Peripheral Blood Mononuclear Cells (PBMC) available from 10X Genomics. Please download the raw data from here. Once you download the data, you need to extract it in a folder of your choice.

```
tar -xvf pbmc3k_filtered_gene_bc_matrices.tar
```

Seurat function `Read10X` can be used to read the data from the 10x output.

```
pbmc.data <- Read10X(data.dir = 'data/pbmc3k/filtered_gene_bc_matrices/hg19')
```

We can create a Seurat object by calling the `CreateSeuratObject()` function. We need to provide the arguments such as "counts" matrix, the name of the "project" and state thresholds for filtering the data based on some properties if necessary.

`min.cells`: the minimum number of cells that needs to be present for the feature (i.e. gene) to not to be filtered out from the dataset.

`min.features` : the minimum number of features that need to be detected for a cell to not to be filtered out.

```
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)
```

We can use `rownames` and `colnames` to see the names of genes and cells.

```
# rownames = gene names
head(rownames(pbmc))
## [1] "AL627309.1"    "AP006222.2"    "RP11-206L10.2" "RP11-206L10.9"
```

```
## [5] "LINC00115"     "NOC2L"

# colnames = sample/cell names
head(colnames(pbmc))
## [1] "AAACATACAACCAC-1" "AAACATTGAGCTAC-1" "AAACATTGATCAGC-1" "AAACCGTGCTTCCG-1"
## [5] "AAACCGTGTATGCG-1" "AAACGCACTGGTAC-1"
```

Q1: Let's warm up! Observe the count data of the first 40 cells for genes "FAM132A", "RBP7" and "TP53". (`Hint:` use `GetAssayData` function.)

```
## 3 x 40 sparse Matrix of class "dgCMatrix"
##    [[ suppressing 40 column names 'AAACATACAACCAC-1', 'AAACATTGAGCTAC-1', 'AAACATTGATCAGC-1' ... ]]
##
## FAM132A . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
## RBP7    . . . . . . . . . . . . . . . . . . . . . . 1 5 . . . . . . . . .
## TP53    . . . . . . . . . . . 1 . . . . . . . . . . . . . . . . . . . . .
##
## FAM132A . . . .
## RBP7    . . 4 .
## TP53    . . 1 .
```

The . values in the matrix represent 0s. Since most values in an scRNA-seq matrix are 0, Seurat uses a sparse-matrix representation to speed up calculations and reduce memory usage.

1.2 Pre-processing the data

1.2.1 QC (filtering low quality samples)

One very common practice is to check the percentage of mithocondrial genes per cell. Cells with high numbers of mithocondrial genes are considered to be under stress and hence are low quality.

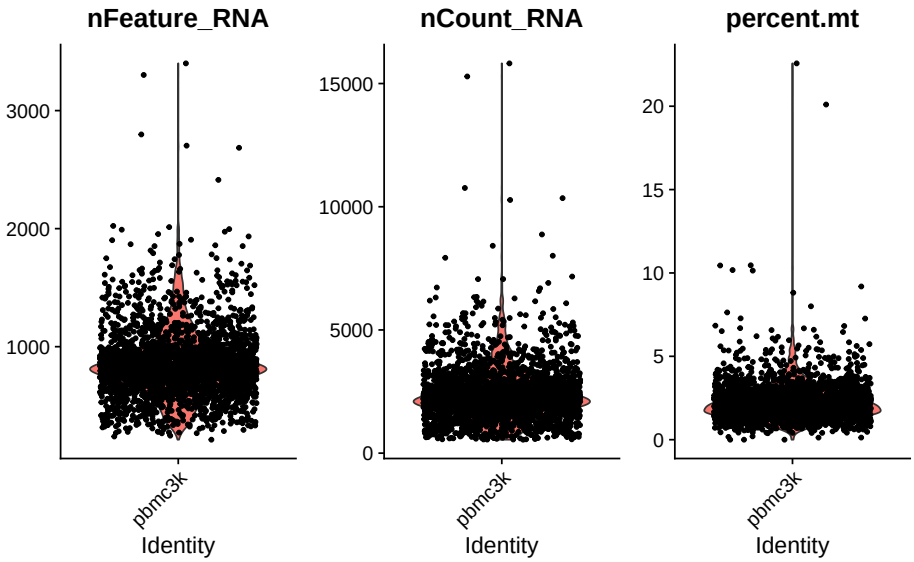
If we are working with gene symbols, mithocondrial gene names start by "MT-".

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

Once we calculated the mithocondrial percentage, it will be stored as a feature in the Seurat object to which we can refer later.

We can check the distribution of different features accross cells visually. * the number of genes expressed in the count matrix * the total counts per cell * the percentage of counts in mitochondrial genes

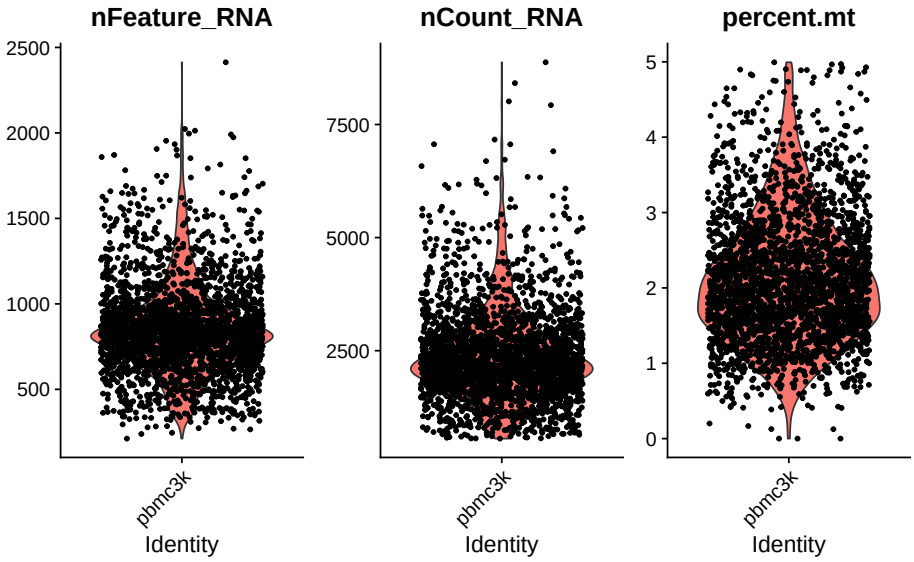
```
VlnPlot(pbmc, features = c("nFeature_RNA", "nCount_RNA", "percent.mt"), ncol = 3)
```



We might want to consider filtering out cells that deviate from the majority of cells (**outlier**) in the distribution of a given feature. Thresholds for these are dataset dependant.

Q2: Let's use `subset()` function to remove outlier cells based on boolean conditions (e.g. &, |) and generate the below plot aftering filtering. Conditions are given below.

```
minFeature_RNA <- 200
maxFeature_RNA <- 2500
maxMTpercent <-5
```



1.2.2 Normalization

Seurat applies 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.

Q3: Do you remember why we use log transformation?

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

`scTransform` is an alternative function to log normalization in recent Seurat versions.

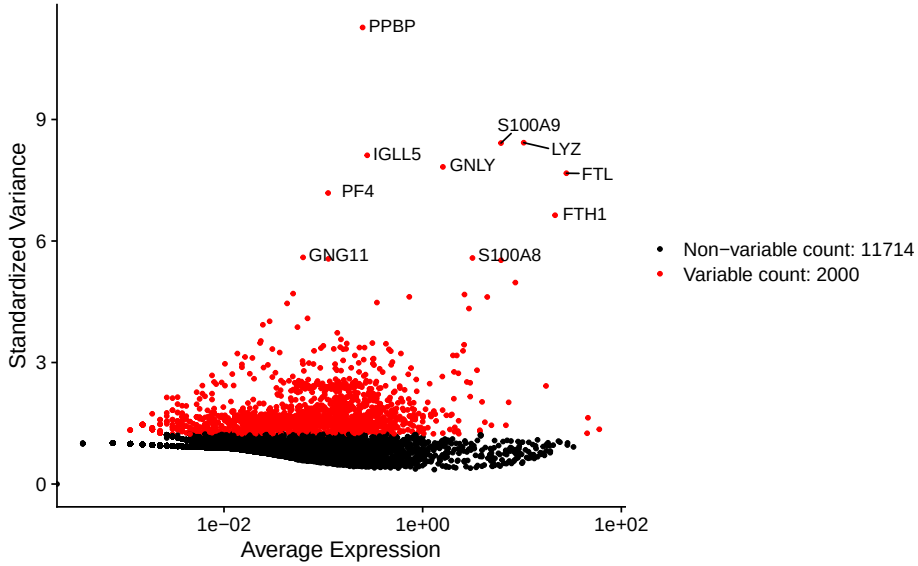
```
#pbmc <- SCTransform(pbmc, vars.to.regress = "percent.mt", verbose = FALSE)
```

1.2.3 Feature selection

Q4: Our feature set (i.e. number of genes) is large. What kind of features do you think should be in the dataset?

Seurat implements `FindVariableFeatures` to model the mean-variance relationship in single-cell data which reduces to dimensions to 2000 by default.

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

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

plot1 <- VariableFeaturePlot(pbmc)
plot2 <- LabelPoints(plot = plot1, points = top10, repel = TRUE)
## When using repel, set xnudge and ynudge to 0 for optimal results
plot2
## Warning: Transformation introduced infinite values in continuous x-axis
```



1.2.4 Scaling

Q5: Seurat implements the `ScaleData` function to scale the data. Do you remember why we need this step?

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

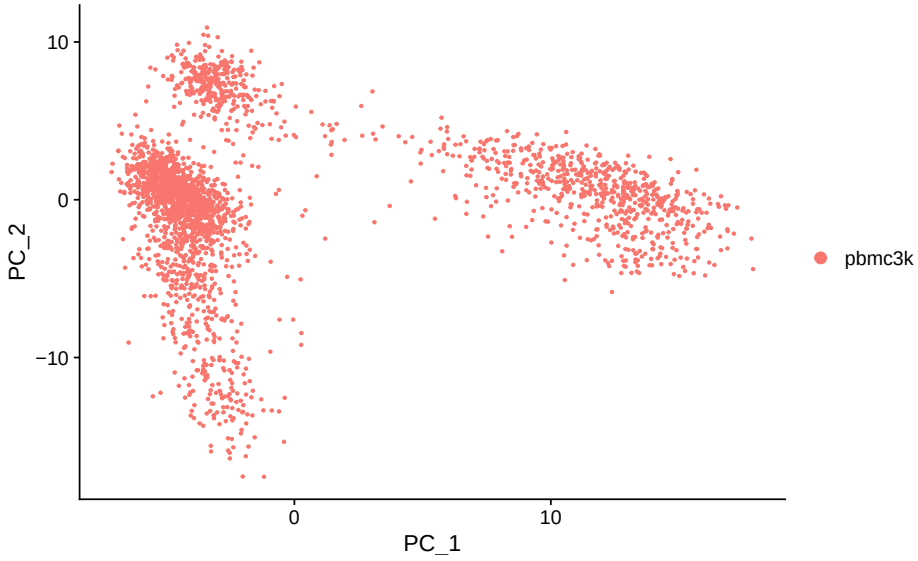
```
## Centering and scaling data matrix

# OR
# pbmc <- ScaleData(pbmc)
```

1.2.5 Dimensionality reduction (feature extraction)

Next step is to compute principal components of our data. We can use all genes or a subset of them to compute them. In this case we use the variable genes that we detected before.

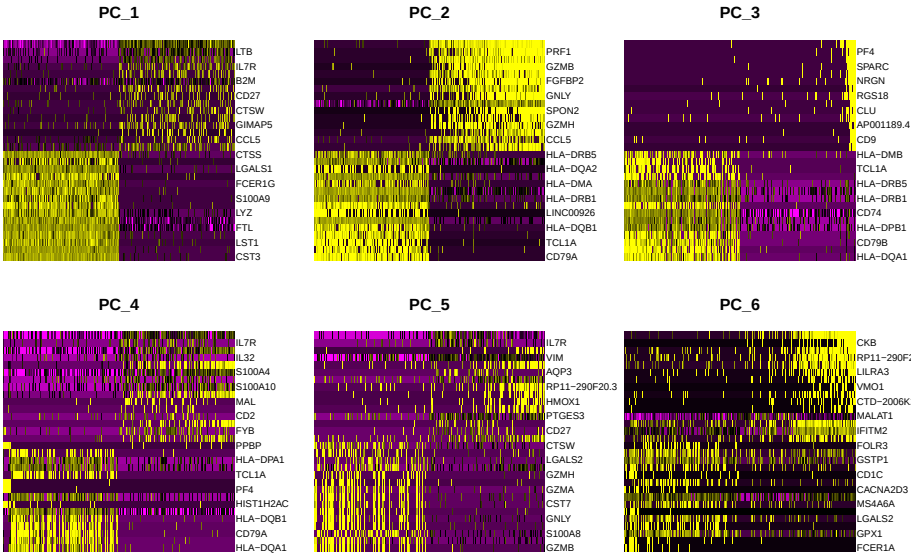
```
pbmc <- RunPCA(pbmc, features = VariableFeatures(object = pbmc))
pbmc[["pca"]]
## A dimensional reduction object with key PC_
##  Number of dimensions: 50
##  Projected dimensional reduction calculated:  FALSE
##  Jackstraw run: FALSE
##  Computed using assay: RNA
```

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

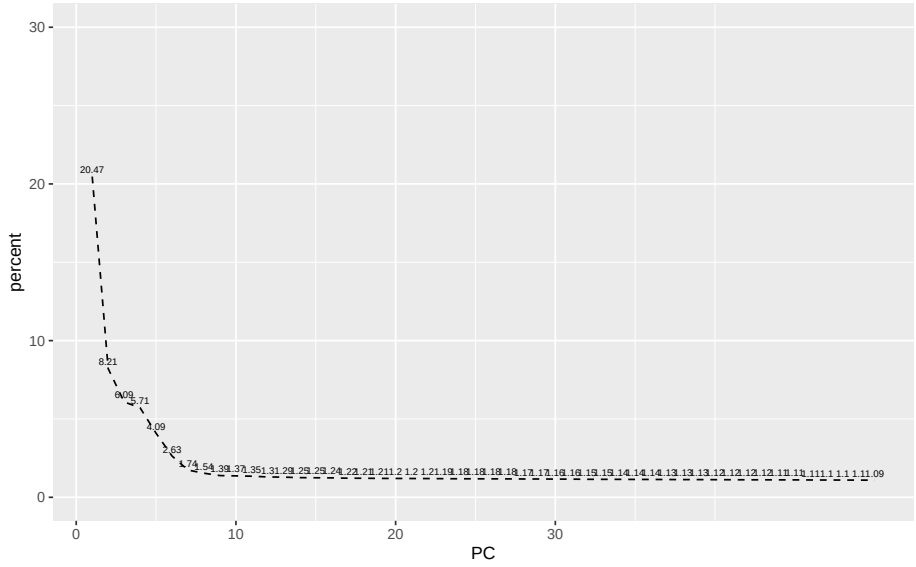


We can visually inspect the genes that correlate the most with the PCs both positively and negatively.

```
DimHeatmap(pbmc, dims = 1:6, cells = 500, balanced = TRUE)
```



Q6: Name the method that we use to determine the dimensionality. Let's try to find out how much of the variance is explained by the PCs. (`Hint:` check `slot Names(pbmc@reductions[["pca"]])` to find the feature might help you with this or name of the method itself can be helpful.)



We can order principal components according to their eigen values or standard deviation. We want to keep those PCs that explain an important proportion of variability in the data.

1.3 Clustering the cells

1.3.1 Graph-based clustering

Seurat adopts a graph-based clustering methodology much similar to `PhenoGraph` approach we discussed earlier.

`FindNeighbors` function performs the following steps:

- (1) build a kNN graph using Euclidean distance in PCA space
- (2) update the edge weights based on the shared neighbors (Jaccard index) to construct a Shared Nearest Neighbor (SNN) graph.

Afterwards, we can use `FindClusters` function to apply a community detection algorithm to identify subgroups. Keep in mind that Seurat uses the Louvain algorithm as a default for this step.

`FindClusters` has a resolution parameter that sets the `granularity` of the downstream clustering, with increased values leading to a greater number of clusters.

The Seurat authors suggest that `granularity values` between 0.4-1.2 usually return good results for sc datasets of around 3K cells. For larger datasets, optimal resolution often increases for larger datasets.

```
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: 95937
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.8723
## Number of communities: 9
## Elapsed time: 0 seconds
```

Based on our PCA analysis we would keep 10 components. The problem with that is that we can't visualize the PCA data from this 10 dimensions in an easy way.

We can now rely of non-linear visualization methods to summarize the PCs in 2/3 dimensions that can be easily visualized.

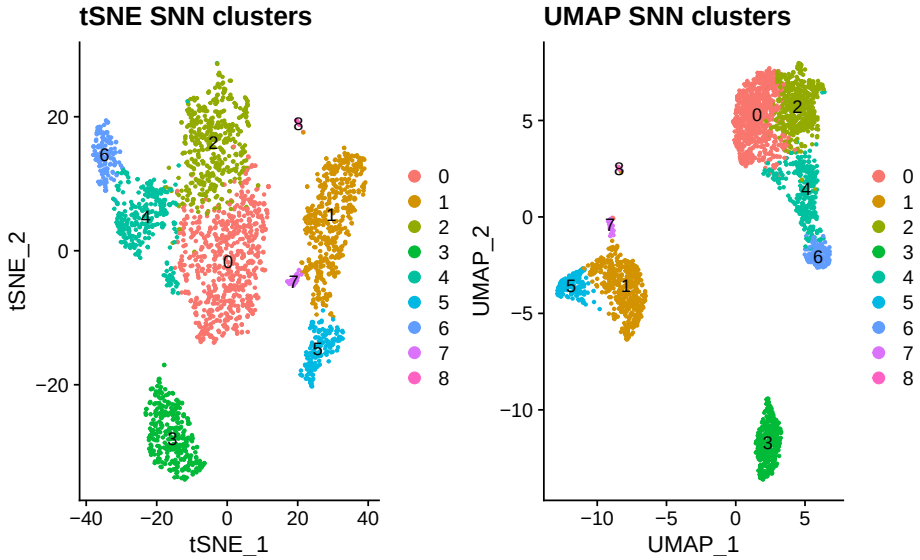
```
pbmc <- RunTSNE(pbmc, dims = 1:10)
```

```
pbmc <- RunUMAP(pbmc, dims = 1:10)
## Warning: The default method for RunUMAP has changed from calling Python UMAP via reticulate to the R-nativ
## To use Python UMAP via reticulate, set umap.method to 'umap-learn' and metric to 'correlation'
## This message will be shown once per session
## 13:40:00 UMAP embedding parameters a = 0.9922 b = 1.112
## 13:40:00 Read 2638 rows and found 10 numeric columns
## 13:40:00 Using Annoy for neighbor search, n_neighbors = 30
## 13:40:00 Building Annoy index with metric = cosine, n_trees = 50
## 0%   10   20   30   40   50   60   70   80   90   100%
## [----|----|----|----|----|----|----|----|----|----|
## **************************************************|
## 13:40:00 Writing NN index file to temp file /var/folders/l5/1nfd7mk14835td6hgk_71p880000gs/T//Rtmpj9FCAq/
## 13:40:00 Searching Annoy index using 1 thread, search_k = 3000
## 13:40:01 Annoy recall = 100%
## 13:40:01 Commencing smooth kNN distance calibration using 1 thread
## 13:40:01 Initializing from normalized Laplacian + noise
```

```
## 13:40:01 Commencing optimization for 500 epochs, with 105142 positive edges
## 13:40:05 Optimization finished
```

We plot the data for the two embeddings and we show the clusters as they were detected on the PCA data.

```
plot1 <- DimPlot(pbmc, reduction = "tsne", label=TRUE ) +
          ggtitle("tSNE SNN clusters")
plot2 <- DimPlot(pbmc, reduction = "umap", label=TRUE ) +
          ggtitle("UMAP SNN clusters")
plot1 + plot2
```




```
saveRDS(pbmc, file = 'data/pbmc3k/pbmc3k_tutorial_out1.rds')
```

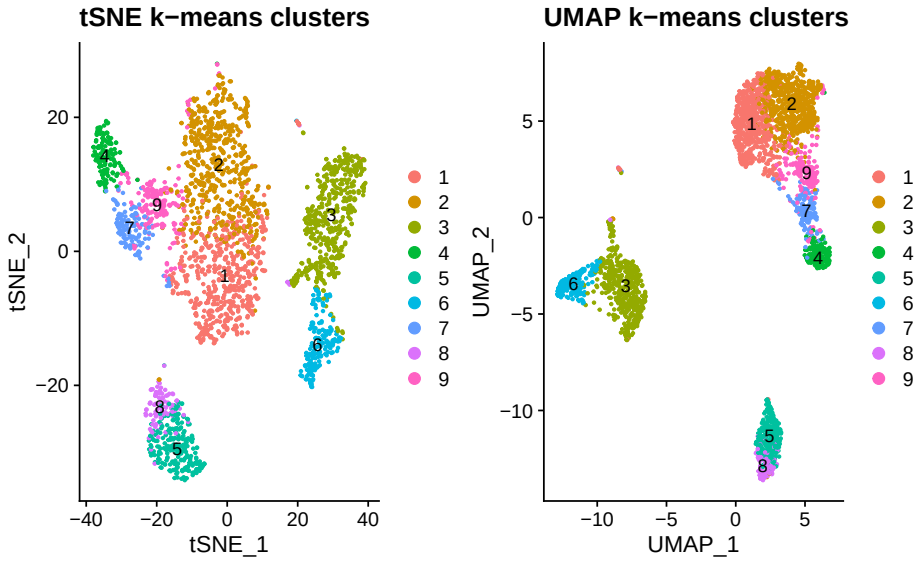
1.3.2 K-means clustering

Seurat does not support K-means? What are we going to do now!? :)

Q7: What do we need to run $K-means$? Let's try to visualize the clusters identified by $K-means$ given $k=9$. (`Hint:` Make use of `Embeddings` function or utilize `reductions` we previously used in the previous examples (e.g. 30 PCs). You can save cluster IDs to `pbmc@meta.data` field.)

```
## [1] 2638   30
##                 orig.ident nCount_RNA nFeature_RNA percent.mt RNA_snn_res.0.5
## AAACATACAACCAC-1     pbmc3k       2419          779  3.0177759               0
## AAACATTGAGCTAC-1     pbmc3k       4903         1352  3.7935958               3
## AAACATTGATCAGC-1     pbmc3k       3147         1129  0.8897363               2
## AAACCGTGCTTCCG-1     pbmc3k       2639          960  1.7430845               1
## AAACCGTGTATGCG-1     pbmc3k        980          521  1.2244898               6
## AAACGCACTGGTAC-1     pbmc3k       2163          781  1.6643551               2
##                 seurat_clusters kmeans.clusters
## AAACATACAACCAC-1               0               9
## AAACATTGAGCTAC-1               3               8
```

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



1.4 Differentially expressed features (cluster bio-markers)

Seurat identifies positive and negative markers of a single cluster (specified in ident.1), compared to all other cells. `FindAllMarkers` automates this process for all clusters, but you can also test groups of clusters vs. each other, or against all cells with `FindMarkers`.

The `min.pct` argument requires a feature to be detected at a minimum percentage in either of the two groups of cells.

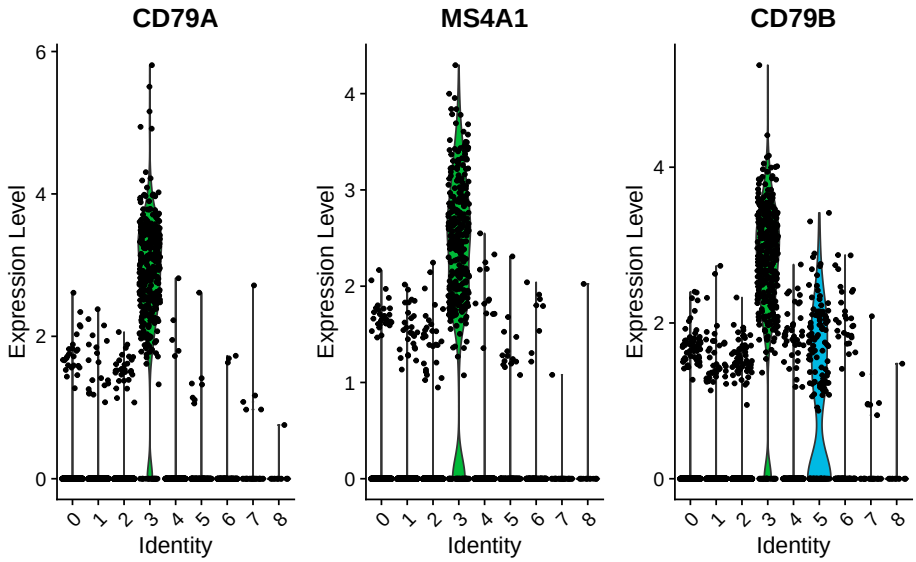
```
# finding all markers of cluster 3
cluster3.markers <- FindMarkers(pbmc, ident.1 = 3, min.pct = 0.25)
## For a more efficient implementation of the Wilcoxon Rank Sum Test,
## (default method for FindMarkers) please install the limma package
## --------------------------------------------
## install.packages('BiocManager')
## BiocManager::install('limma')
## --------------------------------------------
## After installation of limma, Seurat will automatically use the more
## efficient implementation (no further action necessary).
## This message will be shown once per session
head(cluster3.markers, n = 5)
##                  p_val avg_logFC pct.1 pct.2     p_val_adj
## CD79A     0.000000e+00  2.987583 0.936 0.041  0.000000e+00
```

```
## MS4A1      0.000000e+00  2.341555 0.855 0.053  0.000000e+00
## CD79B     2.655974e-274  2.413475 0.916 0.142 3.642403e-270
## LINC00926 2.397625e-272  1.970393 0.564 0.009 3.288103e-268
## TCL1A     9.481783e-271  2.489493 0.622 0.022 1.300332e-266
```

- p_val : p_val (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.

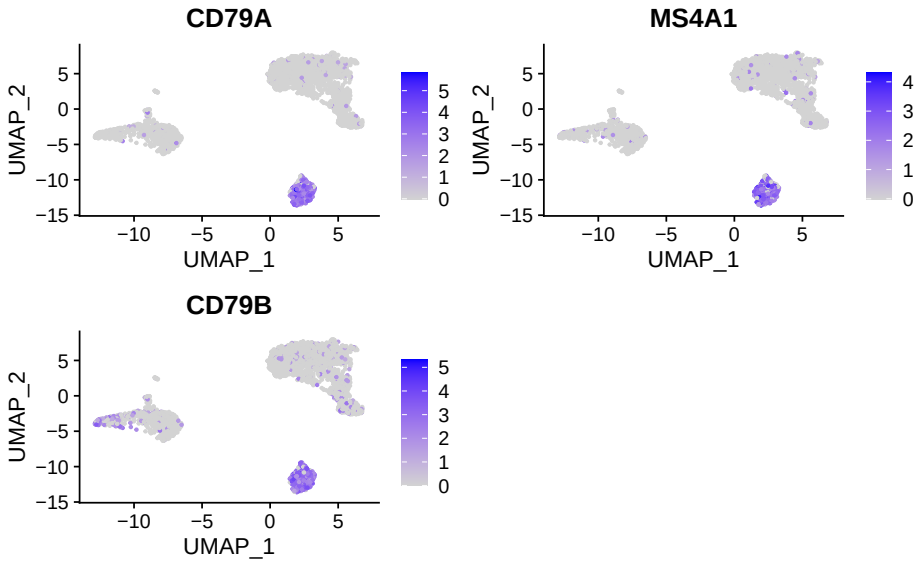
We can plot the genes we identified in the previous step across all clusters in our data.

```
VlnPlot(pbmc, features = rownames(cluster3.markers)[1:3])
```



We can also observe the expression distribution of the genes in the low dimensional embeddings.

```
FeaturePlot(pbmc, features = rownames(cluster3.markers)[1:3])
```



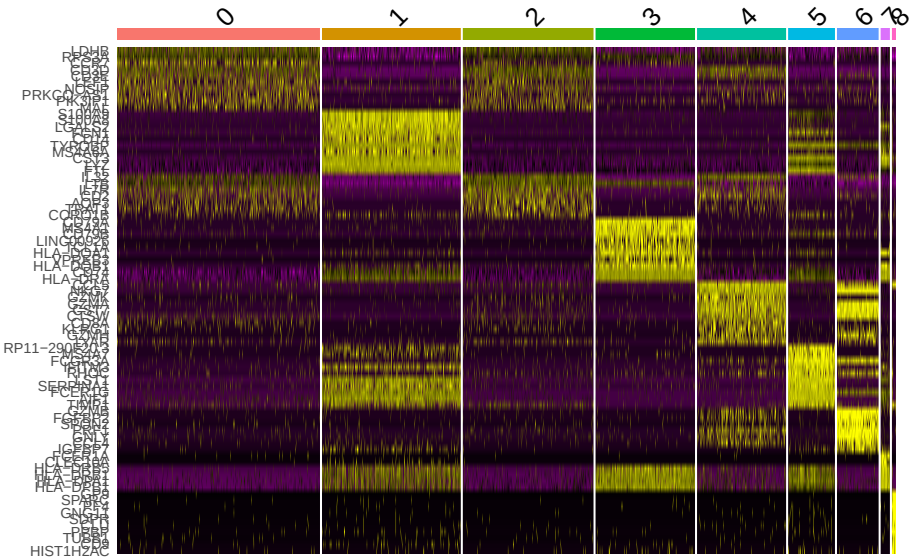
Let's find the gene markers for all clusters.

```
pbmc.markers <- FindAllMarkers(pbmc, only.pos = TRUE, min.pct = 0.25, logfc.threshold = 0.25)
## 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) %>% top_n(n = 2, wt = avg_logFC)
## # A tibble: 18 x 7
## # Groups:   cluster [9]
##        p_val avg_logFC pct.1 pct.2 p_val_adj cluster gene
##        <dbl>     <dbl> <dbl> <dbl>     <dbl> <fct>   <chr>
##  1 7.47e-120     0.770 0.917 0.587 1.02e-115 0       LDHB
##  2 2.81e- 87     0.932 0.443 0.107 3.85e- 83 0       CCR7
##  3 0.            3.86  0.996 0.215 0.        1       S100A9
##  4 0.            3.80  0.975 0.121 0.        1       S100A8
##  5 4.81e- 82     0.870 0.982 0.646 6.60e- 78 2       LTB
##  6 8.95e- 56     0.856 0.423 0.114 1.23e- 51 2       AQP3
##  7 0.            2.99  0.936 0.041 0.        3       CD79A
##  8 9.48e-271     2.49  0.622 0.022 1.30e-266 3       TCL1A
##  9 1.65e-199     2.14  0.958 0.232 2.26e-195 4       CCL5
## 10 4.93e-181     2.15  0.584 0.051 6.76e-177 4       GZMK
## 11 3.51e-184     2.30  0.975 0.134 4.82e-180 5       FCGR3A
## 12 2.03e-125     2.14  1     0.315 2.78e-121 5       LST1
## 13 1.05e-265     3.39  0.986 0.071 1.44e-261 6       GZMB
## 14 6.82e-175     3.41  0.958 0.135 9.36e-171 6       GNLY
## 15 1.48e-220     2.68  0.812 0.011 2.03e-216 7       FCER1A
## 16 1.67e- 21     1.99  1     0.513 2.28e- 17 7       HLA-DPB1
## 17 7.73e-200     5.02  1     0.01  1.06e-195 8       PF4
```

```
## 18 3.68e-110     5.94  1     0.024 5.05e-106 8       PPBP
```

We can plot the top 10 markers per cluster.

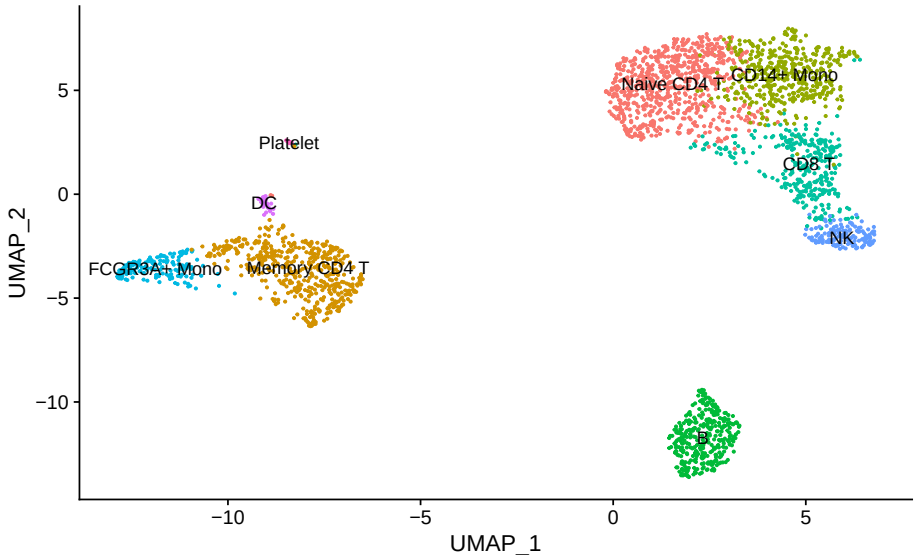
```
top10 <- pbmc.markers %>% group_by(cluster) %>% top_n(n = 10, wt = avg_logFC)
DoHeatmap(pbmc, features = top10$gene) + NoLegend()
```



1.4.1 Cell type identification of clusters

Based on the markers we found, we can rename our clusters to meaningful names. For this dataset, canonical markers match the unbiased clustering to known cell types:

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




```
saveRDS(pbmc, file = 'data/pbmc3k/pbmc3k_tutorial_out2.rds')
```