Experience: single cell RNA-Seq analysis

Load the necessary packages to perform the analysis

```
library(tidyverse)  # collection of packages for tidy data analysis (ggplot, dplyr, ...)
library(Seurat)  # single cell analysis (Check that the version is at least 4.0.0)
library(patchwork)  # combine separate ggplots into the same graphic
```

Analysis of Peripheral Blood Mononuclear Cells

The tutorial is based a dataset of Peripheral Blood Mononuclear Cells (PBMC) freely available from 10X Genomics. The dataset consists of 2,700 single cells that were sequenced on the Illumina NextSeq 500.

Reading in the data.

Load the digital (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).

```
load("./data/pbmc.RData")

pbmc_mat[500:505, 1:30]  # Examine a few genes in the first 30 cells
```

```
## 6 x 30 sparse Matrix of class "dgCMatrix"
##
## FUCA1          . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
## CNR2           . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
## RP11-4M23.3    . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
## PNRC2          . 1 1 . . . . . . . . . . . . . . . . . . . . . . . . . . .
## SRSF10         . . . . . . . . . . . . . 1 . . . . 1 . . . . . . . . . 1 .
## RP11-293P20.2 . . . . . . . . . . . . . . . . . . . . . . . . . . . . . .
```

```
dim(pbmc_mat)  # dimension of the count matrix
```

```
## [1] 32738  2700
```

We are visualizing the first 5 rows of 30 columns. The points represent the zero. The count matrix contains 32738 genes and 2700 cells.

Initialize the Seurat object with the digital count matrix

To proceed with the analysis we need to create a `Seurat Object` that can be handle by `Seurat` package. The necessary input is the count matrix and a name for the project. While creating the object, we are already filtering out any cell that has less than 50 genes (empty cells, all the columns are made of zeros) and the genes that are not present in at least 3 cells. So, we are removing genes that are not expressed because they are always zero and the cells where for some reason didn't work.

```
pbmc <- CreateSeuratObject(counts = pbmc_mat,
                           project = "pbmc", # name of the project
                           min.cells = 3,    # filter for genes (rows)
                           min.features = 50 # filter for cells (columns)
                           )
pbmc
```

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

After the filtering, our data contains 13714 genes and 2700 cells.

Standard workflow

Quality control (QC) and selecting cells for further analysis

A few QC metrics commonly used by the community include:

1. The number of unique genes detected in each cell (low-quality cells or empty droplets will often have very few genes, while cell doublets or multiplets may exhibit an aberrantly high gene count)
2. The total number of molecules detected within a cell (correlates strongly with unique genes)
3. The percentage of reads that map to the mitochondrial genome (low-quality or dying cells often exhibit extensive mitochondrial contamination). If you find a lot of reads in your genome that map to the mitochondrial genome, probably in your cells the mitochondria exploded and the content of the mitochondria was released outside. This usually happens during apoptosis, so the cells is dying or it is already dead.

The `PercentageFeatureSet` function calculates the percentage of counts originating from a set of features (for example, you can use the set of all genes starting with MT- as a set of mitochondrial genes).

The **number of unique genes** (`nFeature_RNA`) and **total molecules** (`nCount_RNA`) are automatically calculated during `CreateSeuratObject`. You can find them stored in the object meta data. Each cell is named according to the unique barcode.

```
# The [[ operator can add columns to the object meta.data,
# ideal to stash QC stats
pbmc[["percent_mt"]] <- PercentageFeatureSet(pbmc, pattern = "^MT-")
# reads that map to the mitochondrial genome

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

```
##                  orig.ident nCount_RNA nFeature_RNA percent_mt
## AAACATACAACCAC       pbmc       2419          779  3.0177759
## AAACATTGAGCTAC       pbmc       4903         1352  3.7935958
## AAACATTGATCAGC       pbmc       3147         1129  0.8897363
## AAACCGTGCTTCCG       pbmc       2639          960  1.7430845
## AAACCGTGTATGCG       pbmc        980          521  1.2244898
```

Visualize QC metrics as a violin plot Here we can see the distribution of the values obtained above (`nCount_RNA`, `nFeature_RNA` and `percent_mt`) for all our cells.

```
p1 <- VlnPlot(pbmc, features = c("nFeature_RNA", "nCount_RNA",
    "percent_mt"), ncol = 3, pt.size = 0.01)

p1
```

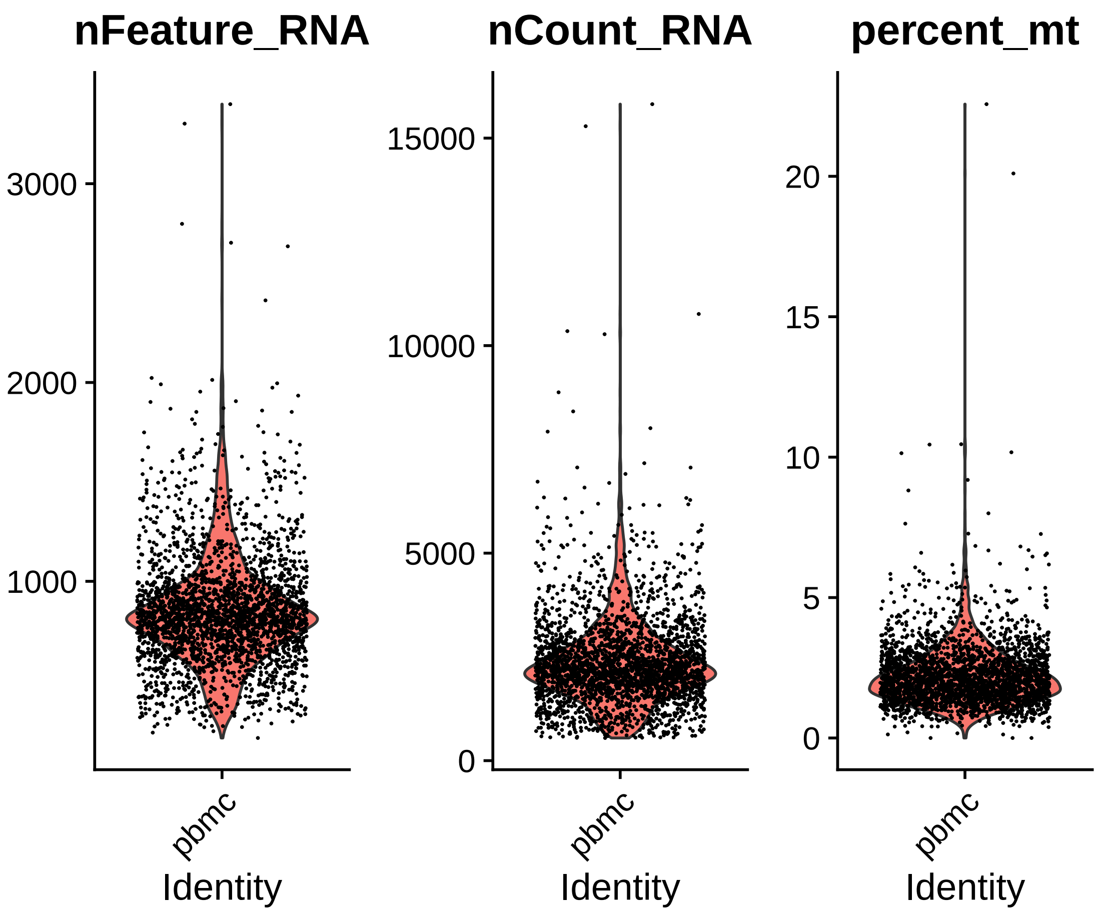


Figure 1: Violin plot representing the distribution of the number of unique genes, total molecules and percentage of reads mapping to mitochondrial genome across all the cells.

The percentage of mitochondrial is acceptable because overall is lower than 5. However, there are some outliers: some cells have a percentage of mitochondrial above 5%, so we will remove them during the quality control step.

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. To see the relationship between those parameters we can also use scatter plots. Usually the number of total genes and unique genes are highly correlated.

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

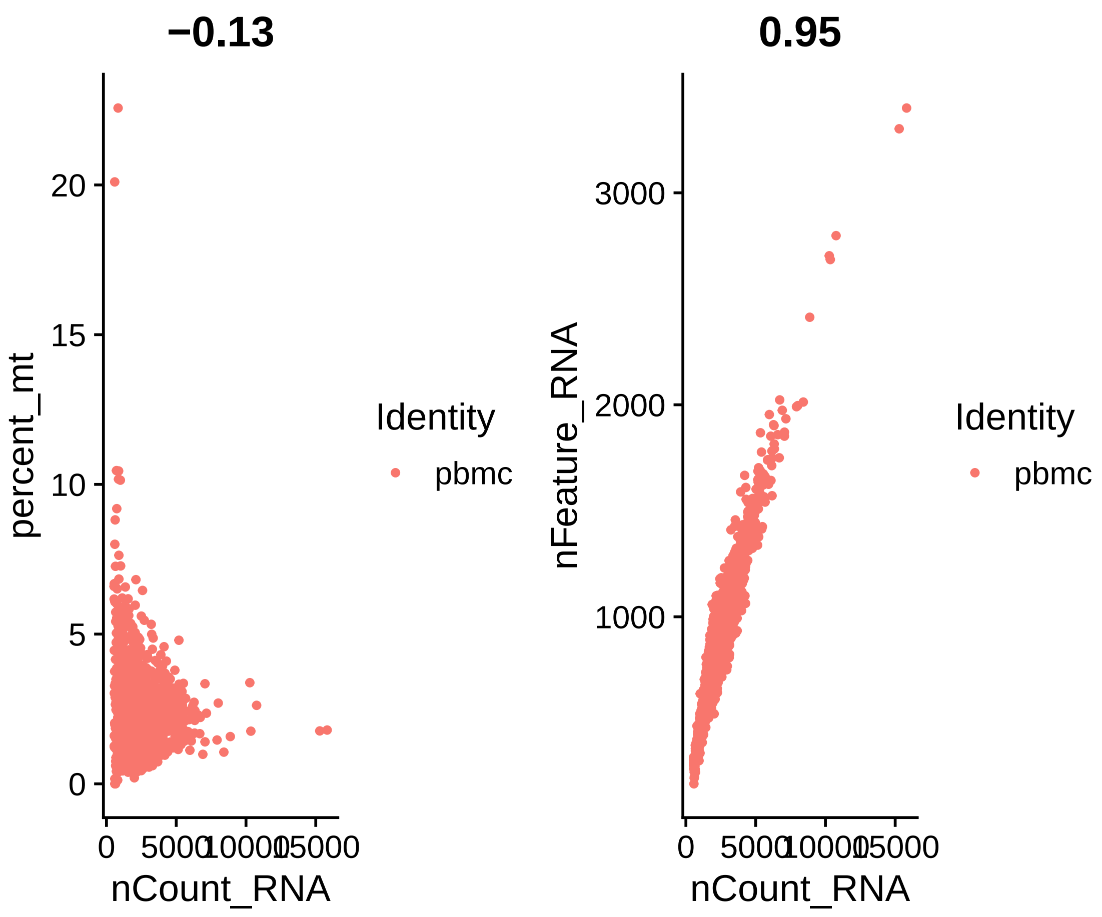


Figure 2: Scatter plot showing feature-feature relationships

Filter cells based on QC values (nFeature and percent_mt) To remove cells with low quality we use the function `subset`. Note that in general, all the functions of Seurat will need in input a Seurat object. This object will always be the same, but as we proceed with our analysis, it will not only contain the raw data, but also the processed data. This allows us to have everything in a single object: it is some kind of list that contains matrices and other objectes that are generated during the analysis.

```
pbmc <- subset(pbmc, subset = nFeature_RNA > 200 &  # lower bound
                  nFeature_RNA < 2500 & # upper bound
                  percent_mt < 5)
pbmc
```

```
## An object of class Seurat
## 13714 features across 2638 samples within 1 assay
## Active assay: RNA (13714 features, 0 variable features)
```

In this case, we are keeping only cells with more than 200 genes (`nFeature_RNA`) but less than 2500 (to remove possible doublets and triplets, that were the outliers in the violin plot), and with a percentage of mitochondrial reads lower than 5% (as mentioned before).

After this filtering step, we have 13714 genes and 2638 cells.

Normalize data

After removing unwanted cells from the dataset, the next step is to normalize the data. Normalization is necessary because, in any kind of experiment, we want **to remove as much as possible technical biases, so differences in signal that are not biological**.

In *bulk RNA-Seq approaches*, the simplest way to normalize a library is to use the total number of reads. So, if you perform 2 experiment and you generated 2 libraries, one with 10 millions reads and one with 20 millions reads, you are assuming that this difference is only technical (you sequenced more the second sample than the first one), and the normalization is applied having as scaling factor the total number of reads. So, you assume that each experiment should generate the same number of reads and any difference observed is technical and doesn't depend on the content of the sample. The same principle is used for *single-cell analysis*, but, in this case, instead of a library, you have a cell: you assume that each cell potentially generate the same number of reads. So, any difference in the total number of reads per cell has to be corrected. This is a heavy hypothesis and not always reasonable for single-cell (because some cells express more than other), but it is the most basic option you have to normalize single-cell data.

By default, Seurat 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 (to reduce skewness). Normalized values are stored in `pbmc[["RNA"]]@data`.

Questionable assumption: each cell should have the same number of reads.

Alternative normalization methods are also available (e.g. `sctransform`). Note that no approaches have been showed to be better than the others.

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

pbmc[["RNA"]]@data[10:15, 1:20]
```

```
## 6 x 20 sparse Matrix of class "dgCMatrix"
##
## HES4         . .        .        .       . .        . . . .        . . .
## RP11-5407.11 . .        .        .       . .        . . . .        . . .
## ISG15        . .        1.429744 3.55831 . 1.726902 . . . 3.339271 . . 1.638876
## AGRN         . .        .        .       . .        . . . .        . . .
## C1orf159     . .        .        .       . .        . . . .        . . .
## TNFRSF18     . 1.625141 .        .       . .        . . . .        . . .
##
## HES4         .        . .        1.157353 . . .
## RP11-5407.11 .        . .        .        . . .
## ISG15        2.861362 . 3.267762 1.157353 . . .
## AGRN         .        . .        .        . . .
## C1orf159     .        . .        .        . . .
## TNFRSF18     .        . .        .        . . .
```

Identification of highly variable features (feature selection)

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

The procedure to select variable features is implemented in the `FindVariableFeatures` function (the procedure models the mean-variance relationship inherent in single-cell data). By default, the function returns the 2,000 most variable features per dataset. These will be used in downstream analysis, like PCA.

We consider that most variable features as the most important and we assume that all the variability is biological and the biological variability is higher than the technical one. This is not always the case, but if we do this step, this is the assumption.

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

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

# plot variable features with labels
plot1 <- VariableFeaturePlot(pbmc)
plot2 <- LabelPoints(plot = plot1, points = top10, repel = TRUE)
plot2
```

Scaling the data

Scaling the data is some kind of normalization on the genes. By scaling, each gene will have the same weight.

By scaling, we apply a linear transformation to the expression levels of each gene, that is a standard preprocessing 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 (z-score transformation)

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`

```r
all_genes <- rownames(pbmc)  # perform scaling on all genes (by default, only the top 2000 are scaled)

pbmc <- ScaleData(pbmc, features = all_genes)

pbmc[["RNA"]]@scale.data[1:5, 1:10]
```

```
##               AAACATACAACCAC AAACATTGAGCTAC AAACATTGATCAGC AAACCGTGCTTCCG
## AL627309.1       -0.05812316    -0.05812316    -0.05812316    -0.05812316
## AP006222.2       -0.03357571    -0.03357571    -0.03357571    -0.03357571
## RP11-206L10.2    -0.04166819    -0.04166819    -0.04166819    -0.04166819
## RP11-206L10.9    -0.03364562    -0.03364562    -0.03364562    -0.03364562
## LINC00115        -0.08223981    -0.08223981    -0.08223981    -0.08223981
##               AAACCGTGTATGCG AAACGCACTGGTAC AAACGCTGACCAGT AAACGCTGGTTCTT
```

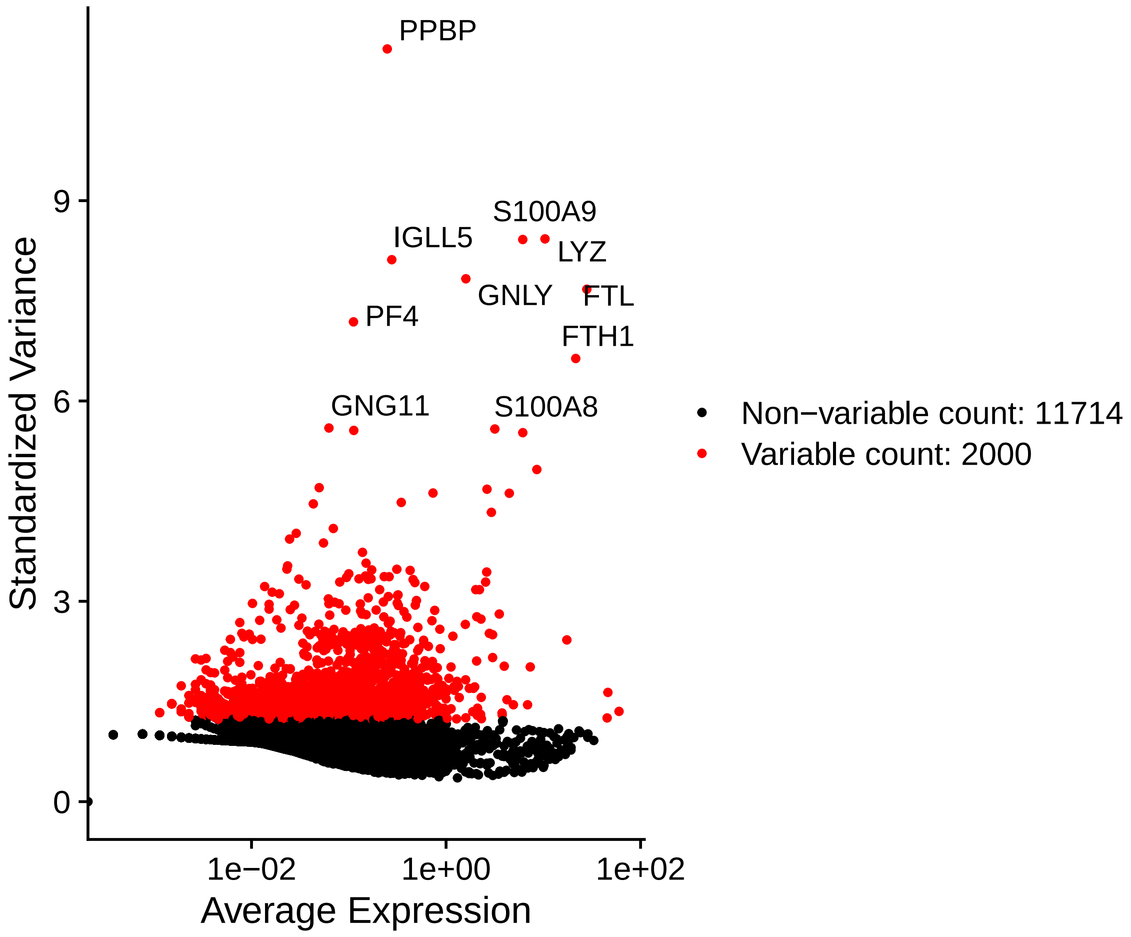


Figure 3: Feature selection

```
## AL627309.1       -0.05812316    -0.05812316    -0.05812316    -0.05812316
## AP006222.2       -0.03357571    -0.03357571    -0.03357571    -0.03357571
## RP11-206L10.2    -0.04166819    -0.04166819    -0.04166819    -0.04166819
## RP11-206L10.9    -0.03364562    -0.03364562    -0.03364562    -0.03364562
## LINC00115        -0.08223981    -0.08223981    -0.08223981    -0.08223981
##               AAACGCTGTAGCCA AAACGCTGTTTCTG
## AL627309.1       -0.05812316    -0.05812316
## AP006222.2       -0.03357571    -0.03357571
## RP11-206L10.2    -0.04166819    -0.04166819
## RP11-206L10.9    -0.03364562    -0.03364562
## LINC00115        -0.08223981    -0.08223981
```

To further reduce, we use dimensionality reduction approaches.

Linear dimensional reduction (PCA)

Dimensionality reduction techniques are employed to reduce data complexity in downstream analyses (e.g. clustering) and for data visualisation.

Principal component analysis (PCA) is the most popular dimension reduction method. For each cell, the original dimensions correspond to the expression values of each gene.

PCA performs an orthogonal transformation of the original dataset to create a set of new, uncorrelated variables or principal components. These principal components are linear combinations of variables in the original dataset. The transformation is defined such that the principal components are ranked with decreasing order of variance. Thus the first principal component amounts to the largest possible variance.

The idea is to discard the principal components with the lowest variance, and effectively reduce the dimensions of the dataset without much loss of information.

By default, in Seurat only the previously determined variable features are used as input for PCA, but this can be defined using features argument if you wish to choose a different subset.

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

PCA is highly interpretable and computationally efficient

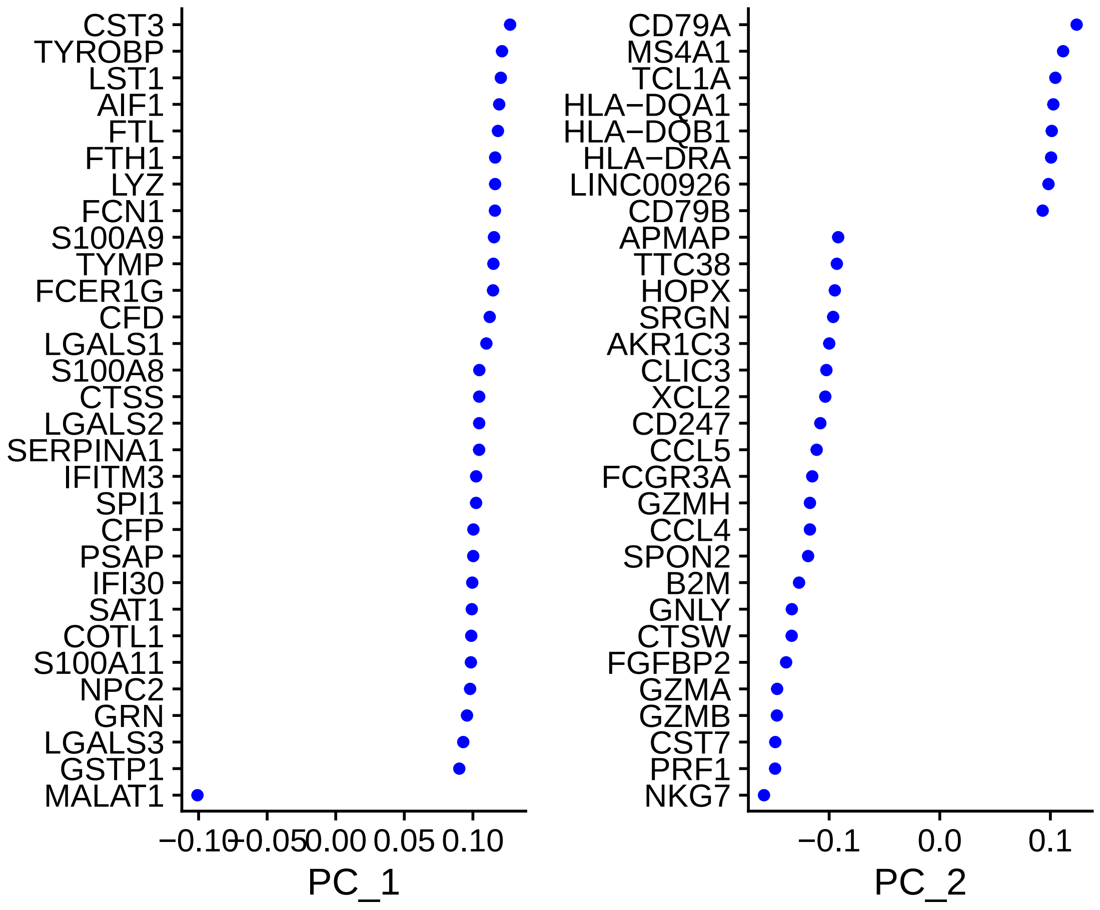
PCA is inappropriate for scRNA-seq data visualization

scRNA-seq data is sparse due to dropout events (weakly expressed genes are missed), meaning there are 60–80% zeroes in the data matrix. It is a highly non-linear structure, while PCA is a linear dimensional reduction technique and therefore deemed very inappropriate for data visualisation. PCA is only used to select ~top 10–50 principal components that can be processed with downstream applications like cluster analysis.

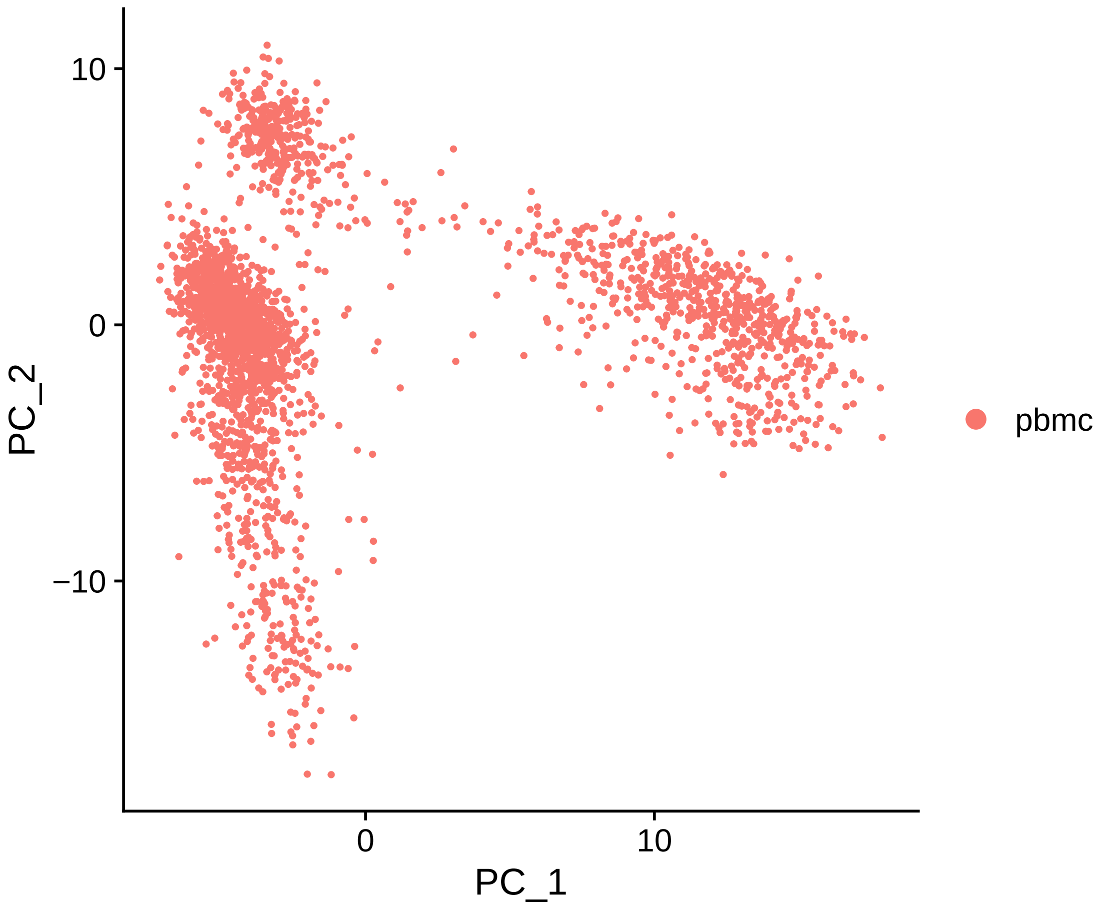
Other dimensionality reduction techniques are used for visualization (t-SNE, UMAP)

Seurat provides several ways of visualizing both cells and features that define the PCA, including `VizDimReduction`, `DimPlot`, and `DimHeatmap`. PCA results are stored in `pbmc[["pca"]]`

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

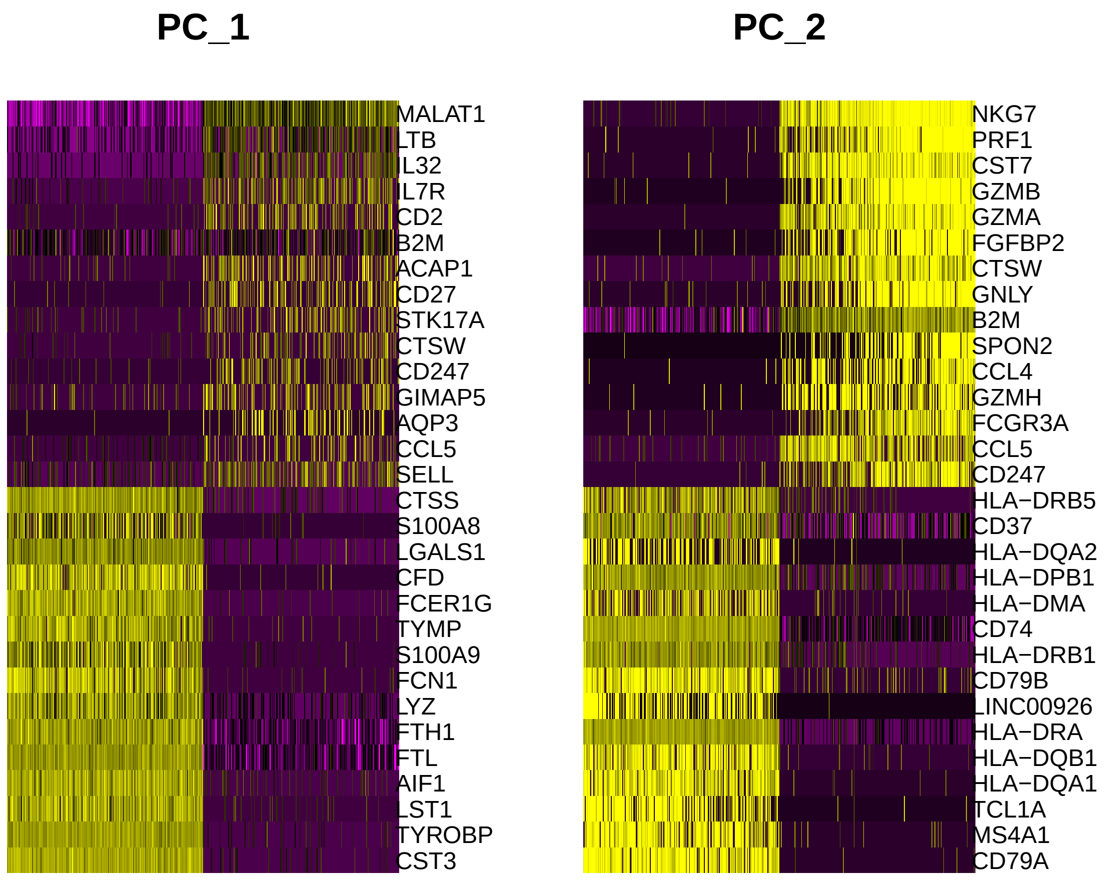


```r
DimPlot(pbmc, reduction = "pca")
```



`DimHeatmap` allows for easy exploration of the primary sources of heterogeneity in a dataset, and can be useful when trying to decide which PCs to include for further downstream analyses. Both cells and features are ordered according to their PCA scores. Setting "cells" to a number plots the 'extreme' cells on both ends of the spectrum, which dramatically speeds plotting for large datasets.

```
DimHeatmap(pbmc, dims = 1:2, cells = 500, balanced = TRUE)
```



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 componenets should we choose to include? 10? 20? 100?*

A heuristic method to decide the number of PC to consider generates an 'Elbow plot': a ranking of principle 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.

```r
ElbowPlot(pbmc)
```

It is called Elbow plot because in an ideal situation, you will see a precise point where, by including more dimensions, you don't increase a lot of the variance.

By looking at the plot obtained in this case, an idea could be to consider the first 8 or 9 dimensions in the analysis. The more dimensions you keep, the more the analysis will be computationally intense.

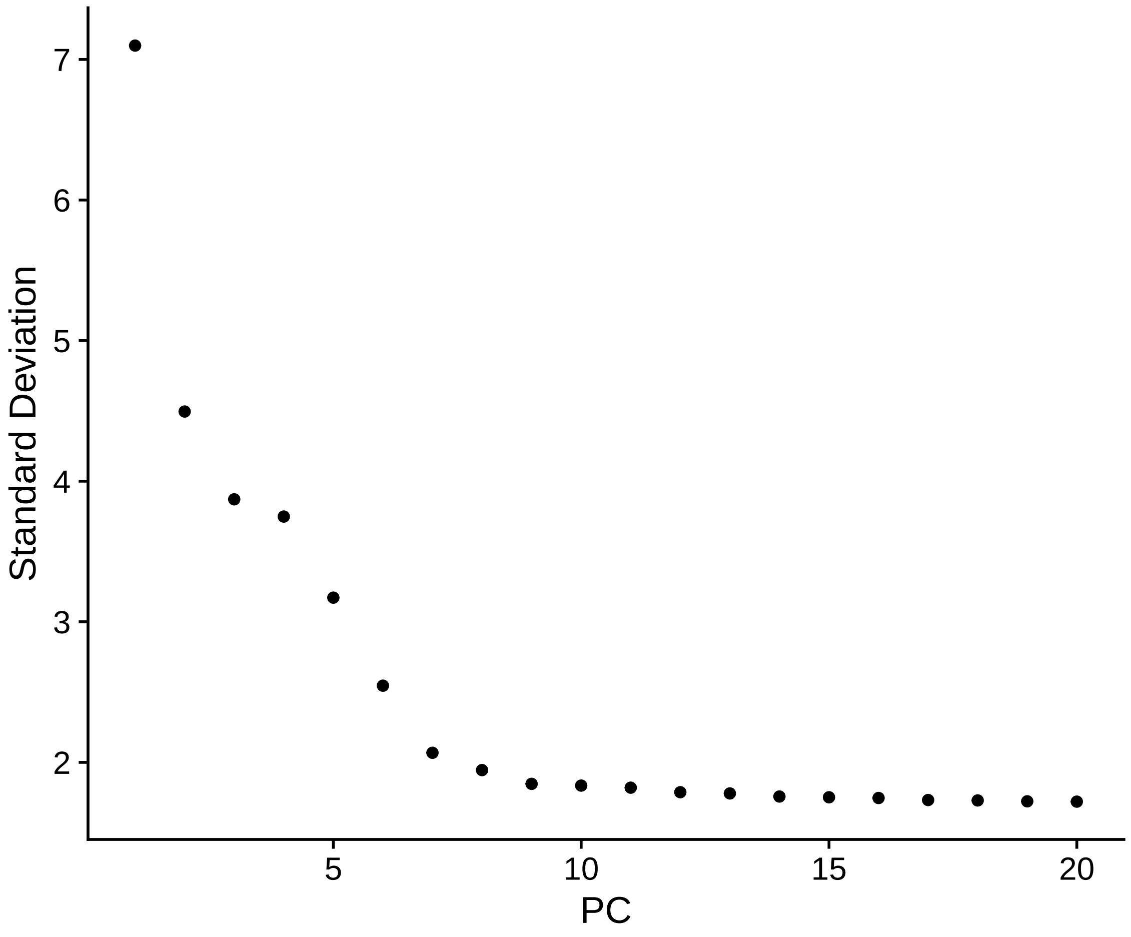


Figure 4: Elbow Plot

Cluster the cells

Seurat v3 applies a graph-based clustering approach. The distance metric which drives the clustering analysis is based on previously identified PCs. The approach to partioning the cellular distance matrix into clusters is the following: we 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) 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)
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: 95927
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.8728
## Number of communities: 9
## Elapsed time: 0 seconds
```

```
head(Idents(pbmc), 5)
```

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

Run non-linear dimensional reduction for visualization (UMAP/tSNE)

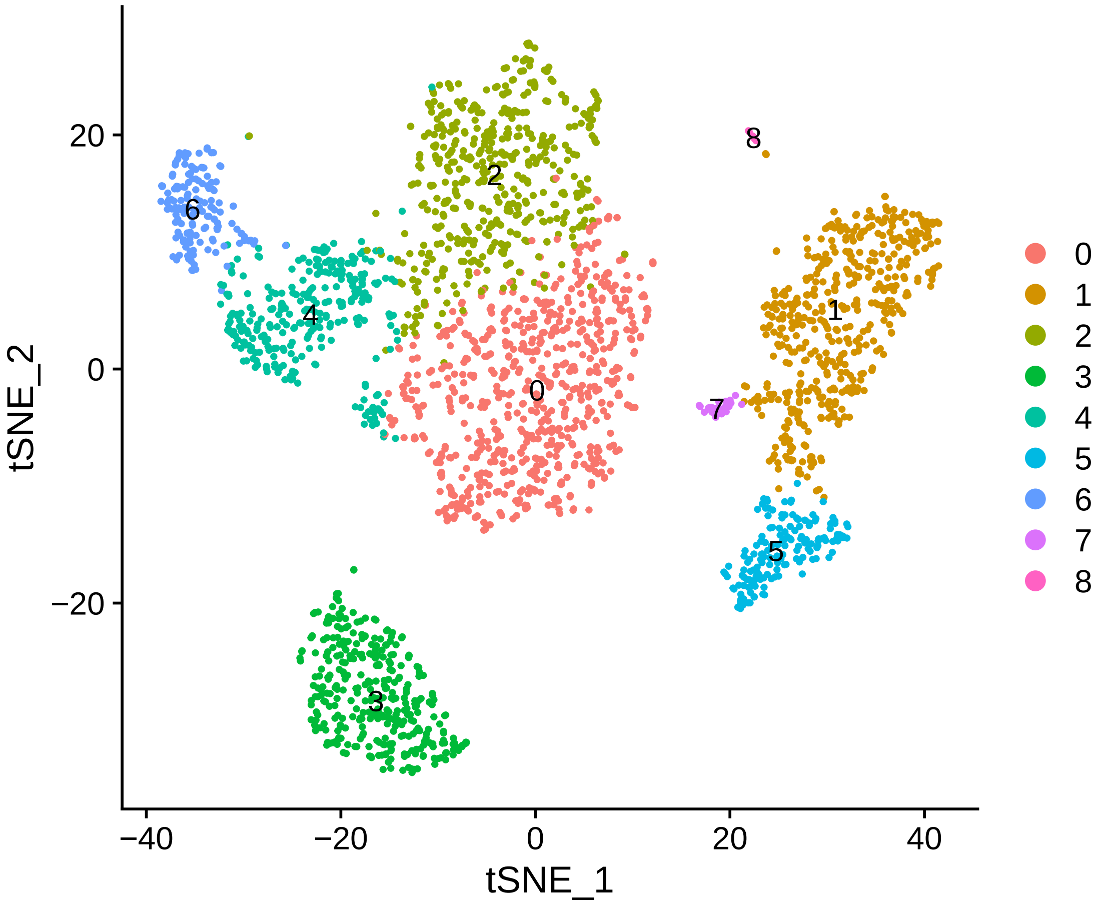
Seurat offers several non-linear dimensional reduction techniques, such as tSNE and UMAP, to visualize and explore these datasets. The goal of these algorithms is to learn the underlying manifold of the data in order to place similar cells together in low-dimensional space. Cells within the graph-based clusters determined above should co-localize on these dimension reduction plots. As input to the UMAP and tSNE, we suggest using the same PCs as input to the clustering analysis.

Visualization with tSNE (t-Stochastic Neighbourhood Embedding) t-SNE is a graph based, non-linear dimensionality reduction technique. It projects high dimensional data onto 2D or 3D components.

Pros - t-SNE powerfully captures the non-linearity in high dimensional datasets and is able to retain the local structures in low dimensions. This is a huge improvement over PCA. t-SNE has been used as a gold standard method for scRNA-seq data visualisation.

Cons - The way t-SNE works, it is impossible for it to preserve the global structure while performing dimension reduction. Only local structures are preserved, while the distances between groups are drastically different depending on the run. - t-SNE embeds data points onto 2 or maximum 3 dimensions only. - For huge datasets, the algorithm takes a long time to run.

```r
pbmc <- RunTSNE(pbmc, dims = 1:10)
DimPlot(pbmc, reduction = "tsne", label = T)
```



Visualization with UMAP (Uniform Manifold Approximation and Projection) UMAP is a dimension reduction technique that can be used for visualisation similarly to t-SNE, but also for general non-linear dimension reduction.

UMAP is a relatively new dimensional reduction technique introduced by McInnes et al in 2018. (McInnes, L, Healy, J, UMAP: Uniform Manifold Approximation and Projection for Dimension Reduction, ArXiv e-prints 1802.03426, 2018)

The algorithm is graph based and principally similar to t-SNE where it constructs a high dimensional graph representation of the data, then optimizes a low-dimensional graph to be as structurally similar as possible.

Pros

- Non linear datasets: UMAP is manifold learning dimension reduction technique and thus captures the non linearity of real world datasets. It is comparable to t-SNE in terms of data visualisation.

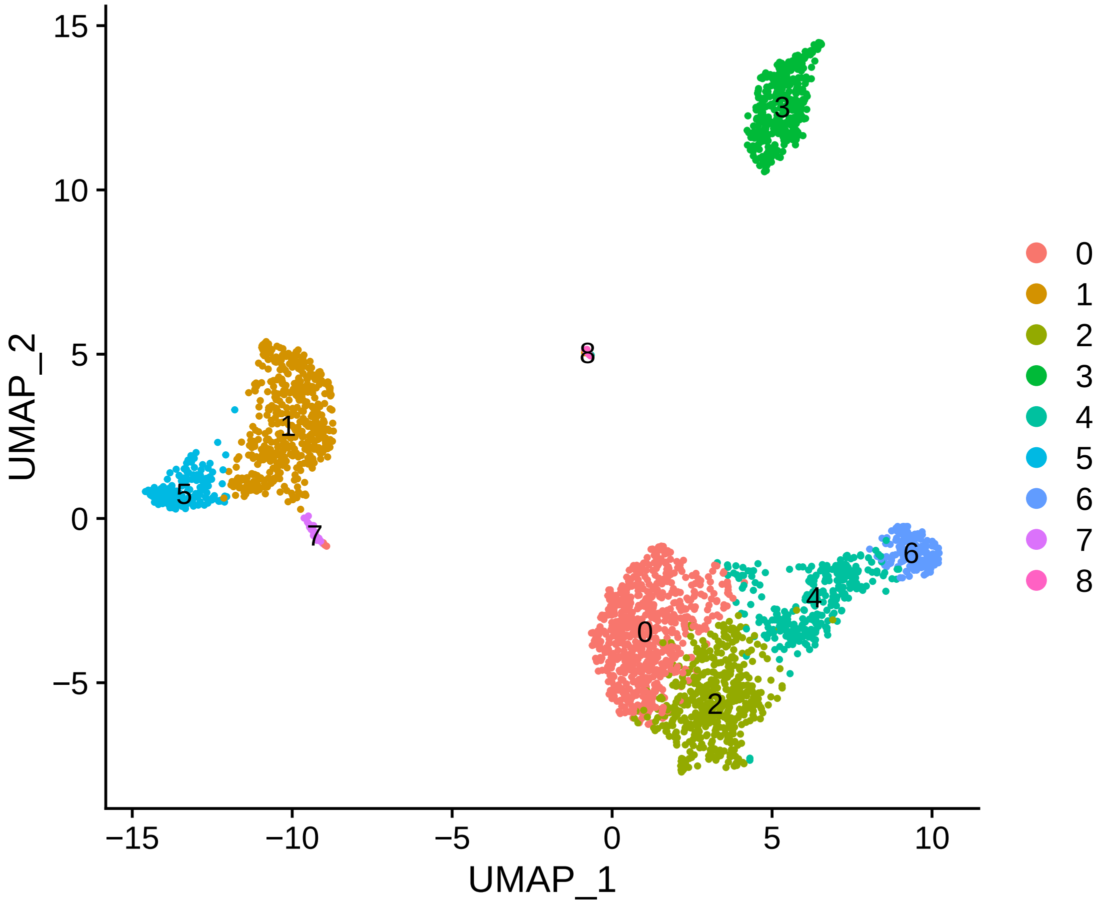
- The mathematical improvements in UMAP allow superior run time performance over t-SNE
- In comparison to t-SNE, UMAP offers better preservation of a data's global structure.
- Unlike t-SNE, UMAP has no computational restrictions on embedding dimensions and can be used as an effective pre-processing step to boost the performance of density based clustering algorithms.

Cons

- Lacks interpretability: Unlike PCA, where the principal components are directions of greatest variance of the source data, the lower dimension embeddings of UMAP lack strong interpretability.
- One of the core assumptions of UMAP is that there exists manifold structure in the data. Because of this, UMAP can tend to find manifold structure within the noise of a dataset.

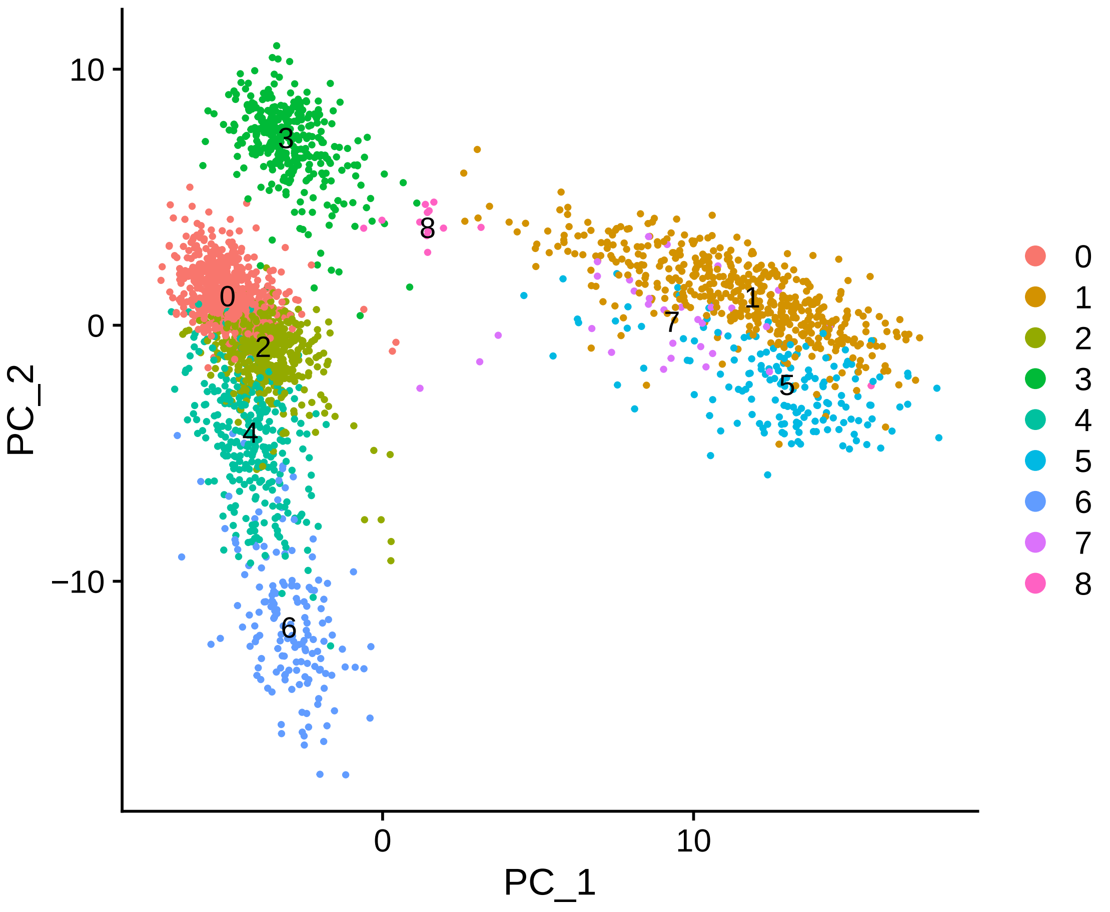
```r
# If you haven't installed UMAP, you can do so via
# reticulate::py_install(packages ='umap-learn')

pbmc <- RunUMAP(pbmc, dims = 1:10)
DimPlot(pbmc, reduction = "umap", label = T)
```



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

Visualization with PCA



Finding differentially expressed features (cluster biomarkers)

Seurat can find markers that define clusters via differential expression. By default, setting only `ident.1`, it identifes 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.

The `min.pct` argument requires a feature to be detected at a minimum percentage in either of the two groups of cells, and the `thresh.test` argument requires a feature to be differentially expressed (on average) by some amount between the two groups.

You can set both of these to 0, but with a dramatic increase in time - since this will test a large number of features that are unlikely to be highly discriminatory.

As another option to speed up these computations, `max.cells.per.ident` can be set. This will downsample each identity class to have no more cells than whatever this is set to. While there is generally going to be a loss in power, the speed increases can be significiant and the most highly differentially expressed features will likely still rise to the top.

The default test used is the Wilcoxon Rank Sum test.

```r
c0_markers <- FindMarkers(pbmc, ident.1 = 0, min.pct = 0.25)
head(c0_markers, n = 5)
```

Find all markers of cluster 0

```
##               p_val avg_log2FC pct.1 pct.2     p_val_adj
## RPS12 1.273332e-143  0.7298951 1.000 0.991 1.746248e-139
## RPS6  6.817653e-143  0.6870694 1.000 0.995 9.349729e-139
## RPS27 4.661810e-141  0.7281575 0.999 0.992 6.393206e-137
## RPL32 8.158412e-138  0.6196246 0.999 0.995 1.118845e-133
## RPS14 5.177478e-130  0.6252832 1.000 0.994 7.100394e-126
```

The results data frame has the following columns :

`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.

The following differential expression tests are currently supported:

wilcox : Wilcoxon rank sum test (default)

bimod : Likelihood-ratio test for single cell feature expression, (McDavid et al., Bioinformatics, 2013)

roc : Standard AUC classifier. For each gene, evaluates (using AUC) a classifier built on that gene alone, to classify between two groups of cells. An AUC value of 1 means that expression values for this gene alone can perfectly classify the two groupings (i.e. Each of the cells in cells.1 exhibit a higher level than each of the cells in cells.2). An AUC value of 0 also means there is perfect classification, but in the other direction. A value of 0.5 implies that the gene has no predictive power to classify the two groups. Returns a 'predictive power' (abs(AUC-0.5) * 2) ranked matrix of putative differentially expressed genes.

t : Student's t-test

poisson : Likelihood ratio test assuming an underlying poisson 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 : Utilizes the MAST package to run the DE testing. GLM-framework that treates cellular detection rate as a covariate (Finak et al, Genome Biology, 2015)

DESeq2 : Identifies differentially expressed genes between two groups of cells based on a model using DESeq2 which uses a negative binomial distribution (Love et al, Genome Biology, 2014)

```r
c3_markers <- FindMarkers(pbmc, ident.1 = 3, ident.2 = c(1, 5),
    min.pct = 0.25)
head(c3_markers, n = 5)
```

Find all markers distinguishing cluster 3 from clusters 2, 5 and 7

```
##                p_val avg_log2FC pct.1 pct.2     p_val_adj
## CD79A  2.453150e-175   4.318558 0.936 0.042 3.364250e-171
## TYROBP 3.150199e-151  -4.997444 0.102 0.995 4.320183e-147
## CST3   2.074856e-149  -4.785573 0.174 0.994 2.845458e-145
## S100A4 2.462265e-149  -4.159048 0.360 1.000 3.376750e-145
## FTL    1.886388e-146  -3.200061 0.994 1.000 2.586992e-142
```

```r
pbmc_markers <- FindAllMarkers(pbmc, only.pos = TRUE, min.pct = 0.25,
    logfc.threshold = 0.25)

pbmc_markers %>%
    group_by(cluster) %>%
    slice_max(n = 2, order_by = avg_log2FC)
```

Find markers for every cluster compared to all remaining cells, report only the positive ones

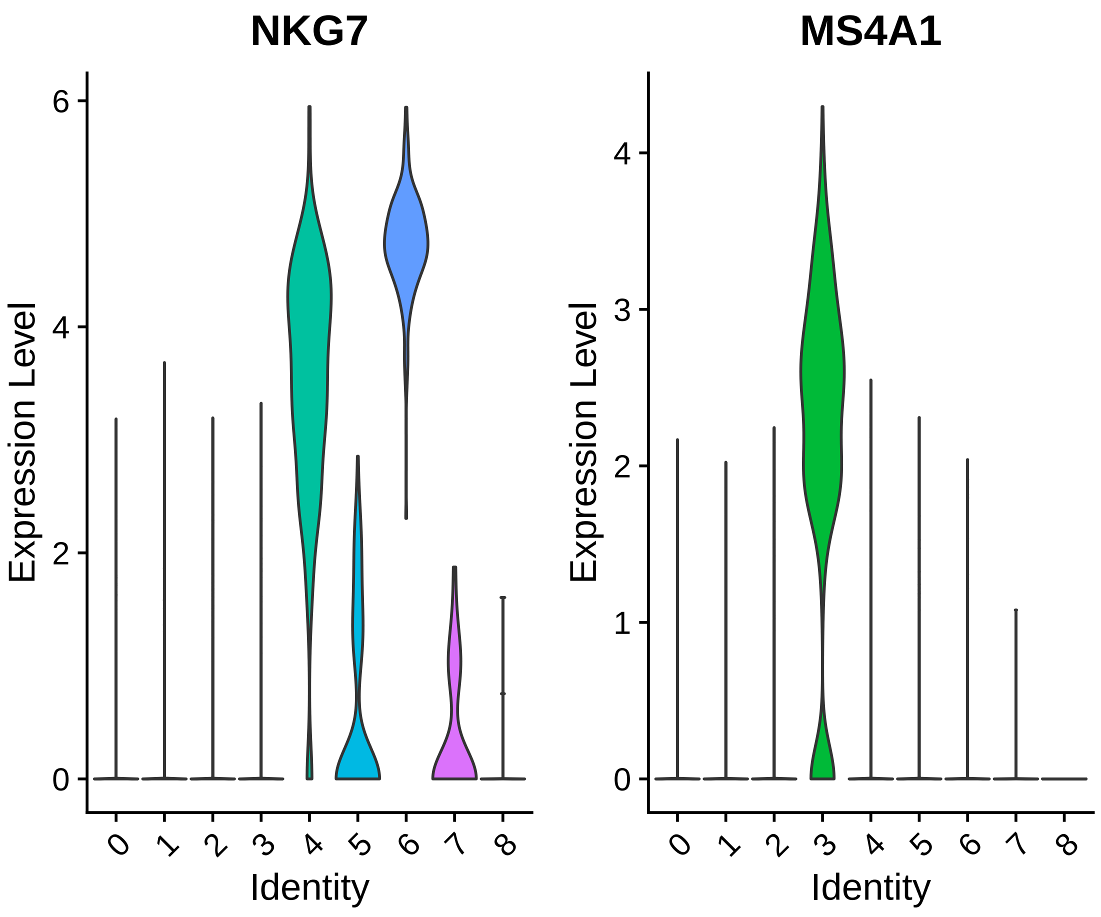
```
## # A tibble: 18 x 7
## # Groups:   cluster [9]
##        p_val avg_log2FC pct.1 pct.2 p_val_adj cluster gene
##        <dbl>      <dbl> <dbl> <dbl>     <dbl> <fct>   <chr>
##  1 9.57e- 88       1.36 0.447 0.108 1.31e- 83 0       CCR7
##  2 3.75e-112       1.09 0.912 0.592 5.14e-108 0       LDHB
##  3 0               5.57 0.996 0.215 0         1       S100A9
##  4 0               5.48 0.975 0.121 0         1       S100A8
##  5 1.06e- 86       1.27 0.981 0.643 1.45e- 82 2       LTB
##  6 2.97e- 58       1.23 0.42  0.111 4.07e- 54 2       AQP3
##  7 0               4.31 0.936 0.041 0         3       CD79A
##  8 9.48e-271       3.59 0.622 0.022 1.30e-266 3       TCL1A
##  9 5.61e-202       3.10 0.983 0.234 7.70e-198 4       CCL5
## 10 7.25e-165       3.00 0.577 0.055 9.95e-161 4       GZMK
## 11 3.51e-184       3.31 0.975 0.134 4.82e-180 5       FCGR3A
## 12 2.03e-125       3.09 1     0.315 2.78e-121 5       LST1
## 13 3.13e-191       5.32 0.961 0.131 4.30e-187 6       GNLY
## 14 7.95e-269       4.83 0.961 0.068 1.09e-264 6       GZMB
## 15 1.48e-220       3.87 0.812 0.011 2.03e-216 7       FCER1A
## 16 1.67e- 21       2.87 1     0.513 2.28e- 17 7       HLA-DPB1
## 17 1.92e-102       8.59 1     0.024 2.63e- 98 8       PPBP
## 18 9.25e-186       7.29 1     0.011 1.27e-181 8       PF4
```

Visualization tools

Seurat offers several tools for visualizing marker expression.

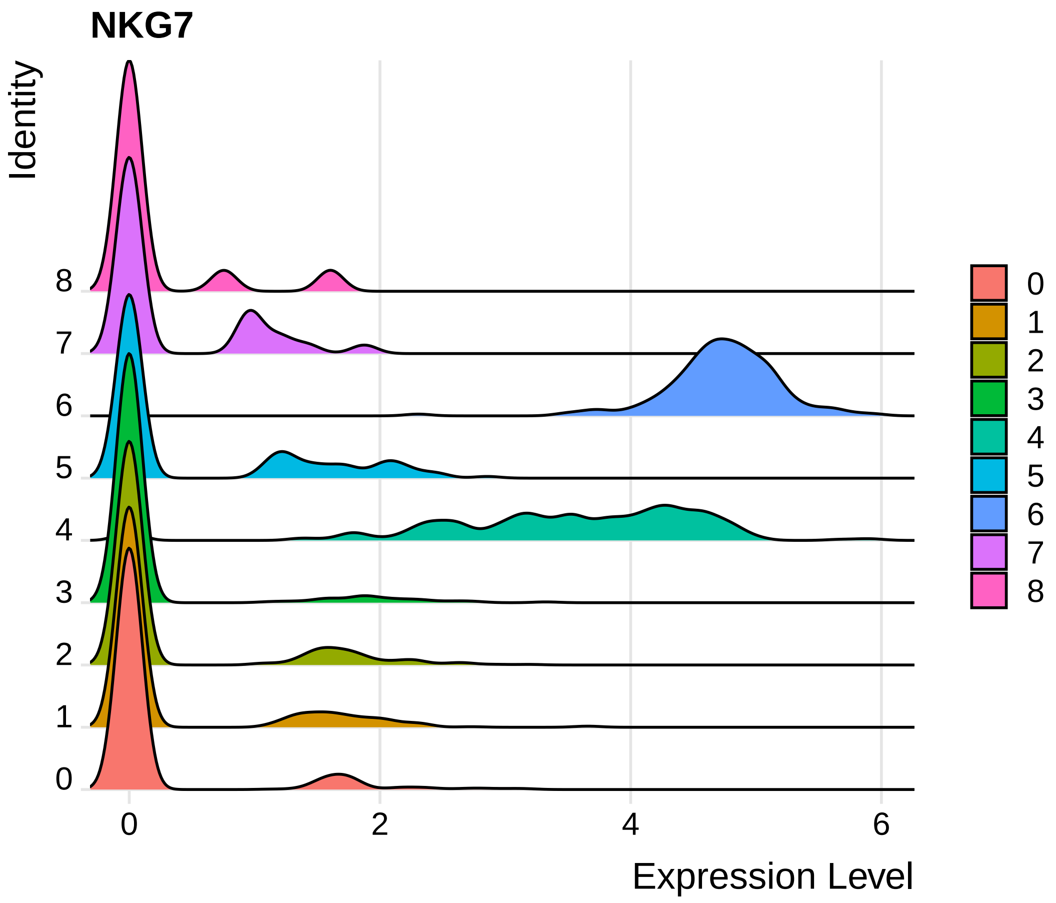
```r
VlnPlot(pbmc, features = c("NKG7", "MS4A1"), pt.size = 0)
```

VlnPlot shows expression probability distributions across clusters



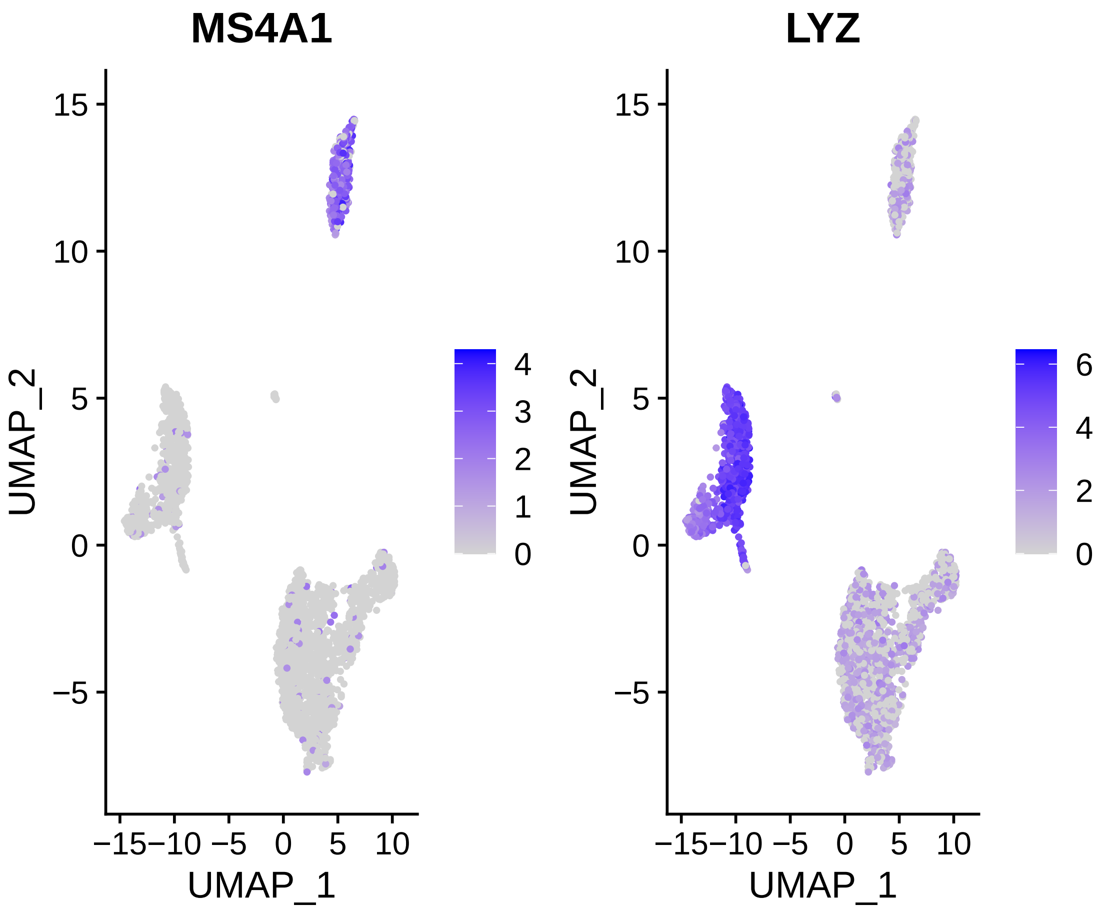
`RidgePlot` also shows expression probability distributions

```r
RidgePlot(pbmc, features = c("NKG7"))
```



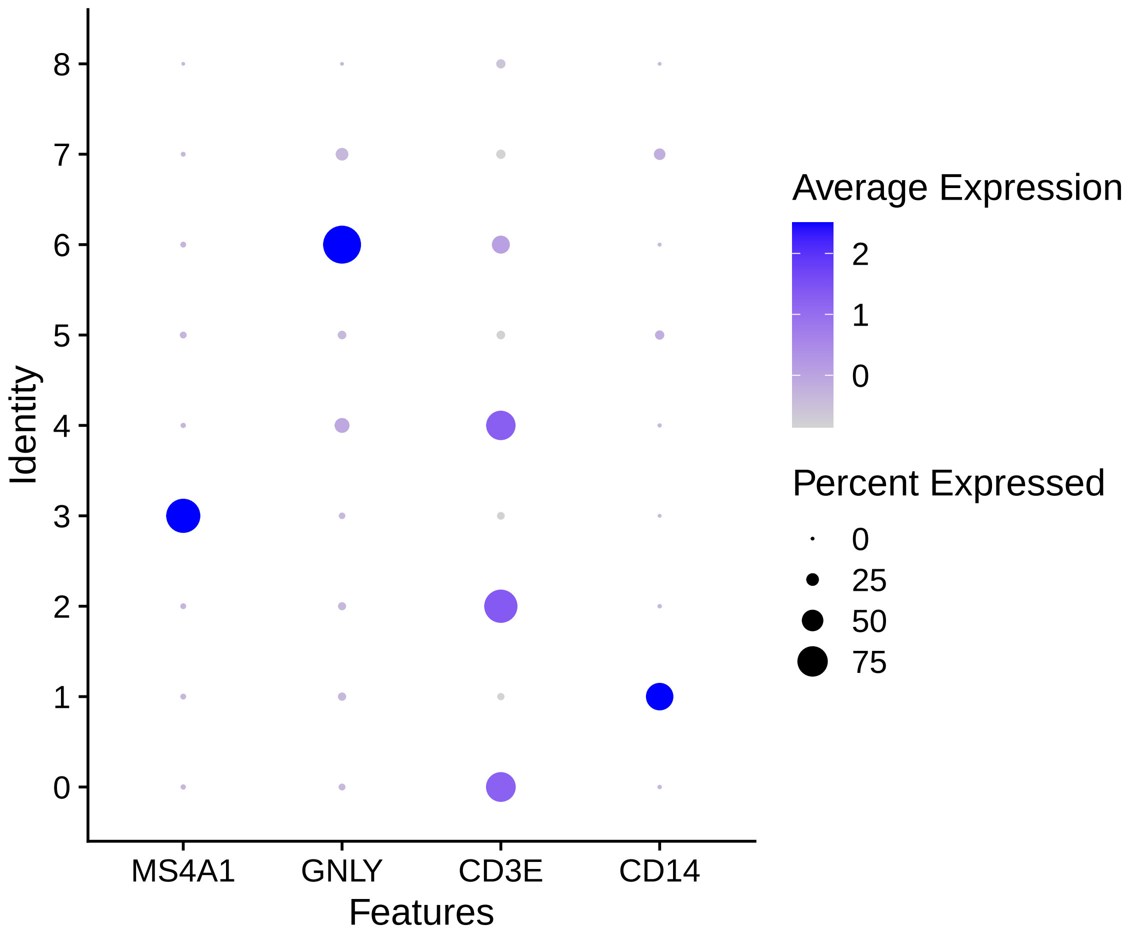
```
FeaturePlot(pbmc, features = c("MS4A1", "LYZ"))
```

FeaturePlot visualizes feature expression on a tSNE, UMAP or PCA plot



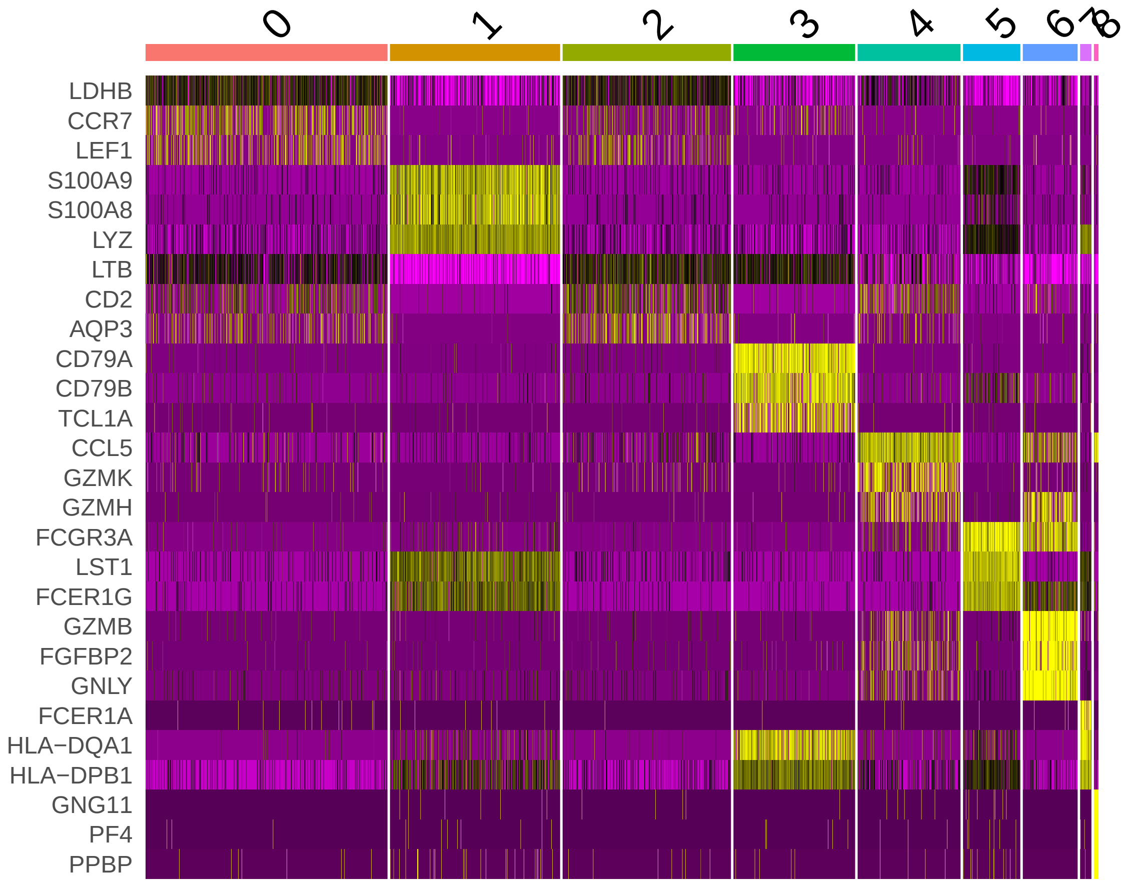
DotPlot Intuitive way of visualizing how feature expression changes across different identity classes (clusters). The size of the dot encodes the percentage of cells within a class, while the color encodes the AverageExpression level of cells within a class (blue is high).

```r
DotPlot(pbmc, features = c("MS4A1", "GNLY", "CD3E", "CD14"))
```



```r
top3 <- pbmc_markers %>%
    group_by(cluster) %>%
    top_n(n = 3, wt = avg_log2FC)
DoHeatmap(pbmc, features = top3$gene) + NoLegend()
```

DoHeatmap generates an expression heatmap for given cells and features. In this case, we are plotting the top 5 markers (or all markers if less than 5) for each cluster.



Assigning cell type identity to clusters

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

Cluster Markers Cell_type

0 IL7R, CCR7 Naive T_cell (CD4+)

1 CD14, LYZ Monocyte (CD14+)

2 IL7R, S100A4 Memory T_cell (CD4+)

3 MS4A1 B_cell

4 CD8A T_cell (CD8+)

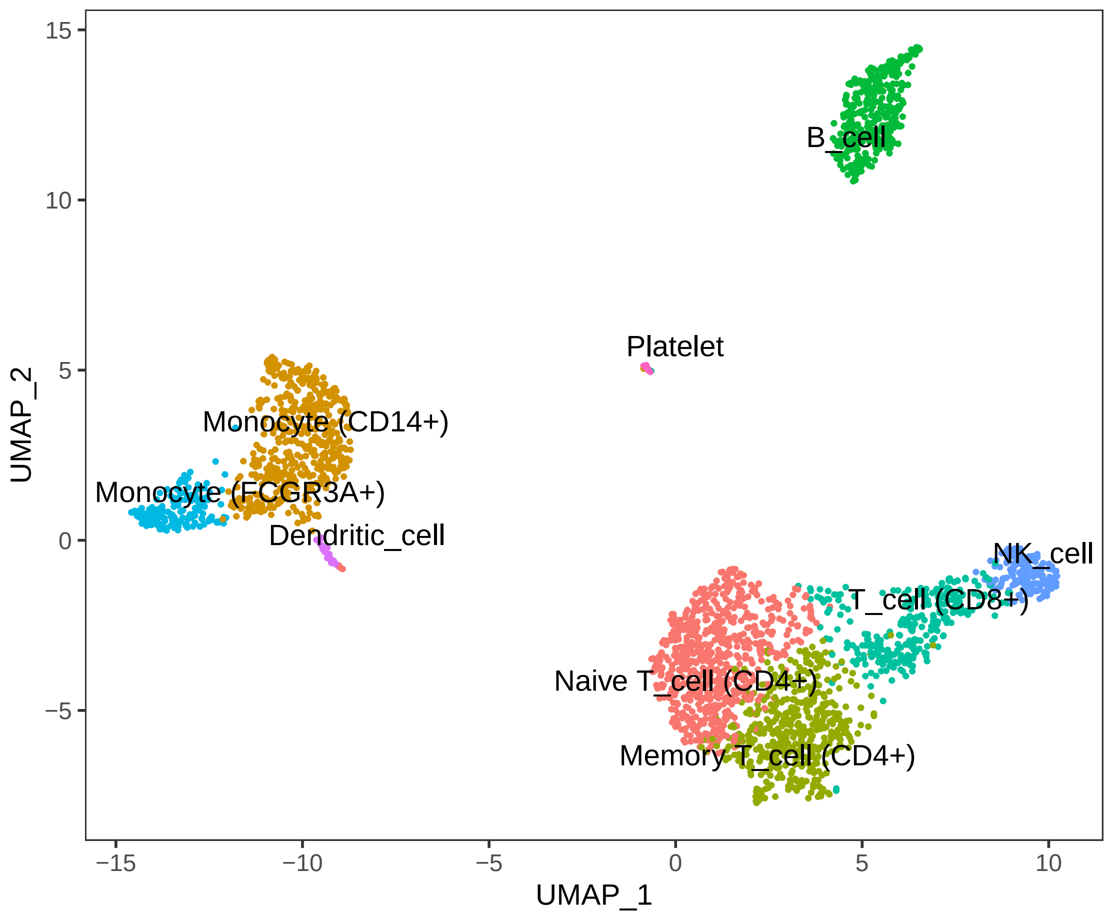
5 FCGR3A, MS4A7 FCGR3A+ Mono

6 GNLY, NKG7 NK_cell

7 FCER1A, CST3 Dendritic_cell

8 PPBP Platelet

```r
new_cluster_ids <- c("Naive T_cell (CD4+)", "Monocyte (CD14+)",
    "Memory T_cell (CD4+)", "B_cell", "T_cell (CD8+)", "Monocyte (FCGR3A+)",
    "NK_cell", "Dendritic_cell", "Platelet")
names(new_cluster_ids) <- levels(pbmc)
pbmc <- RenameIdents(pbmc, new_cluster_ids)
DimPlot(pbmc, reduction = "umap", label = TRUE, pt.size = 0.5,
    repel = T) + theme_bw() + NoLegend() + theme(panel.grid = element_blank())
```


B_cell
Platelet
Monocyte (CD14+)
Monocyte (FCGR3A+)
Dendritic_cell
NK_cell
T_cell (CD8+)
Naive T_cell (CD4+)
Memory T_cell (CD4+)
UMAP_2
UMAP_1