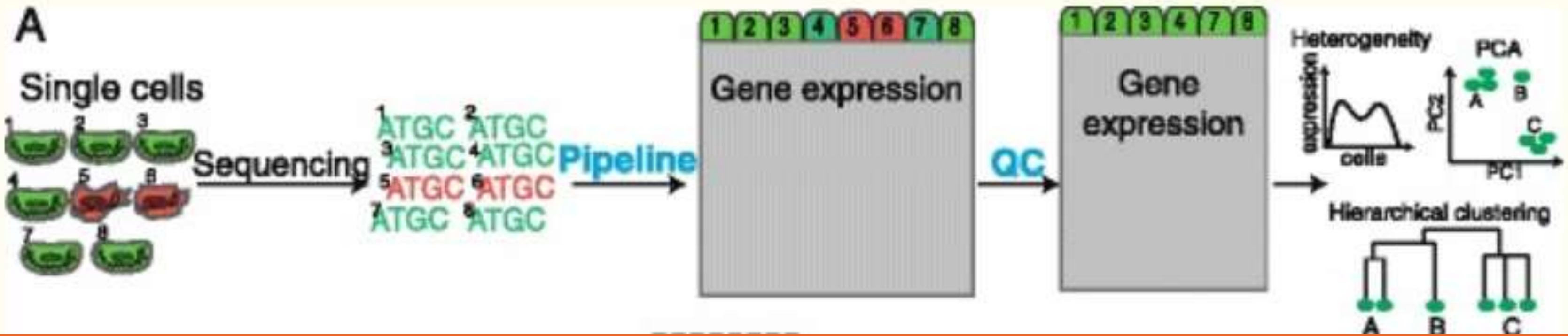
Guided Clustering Tutorial

Mentor:Dr.Zhang

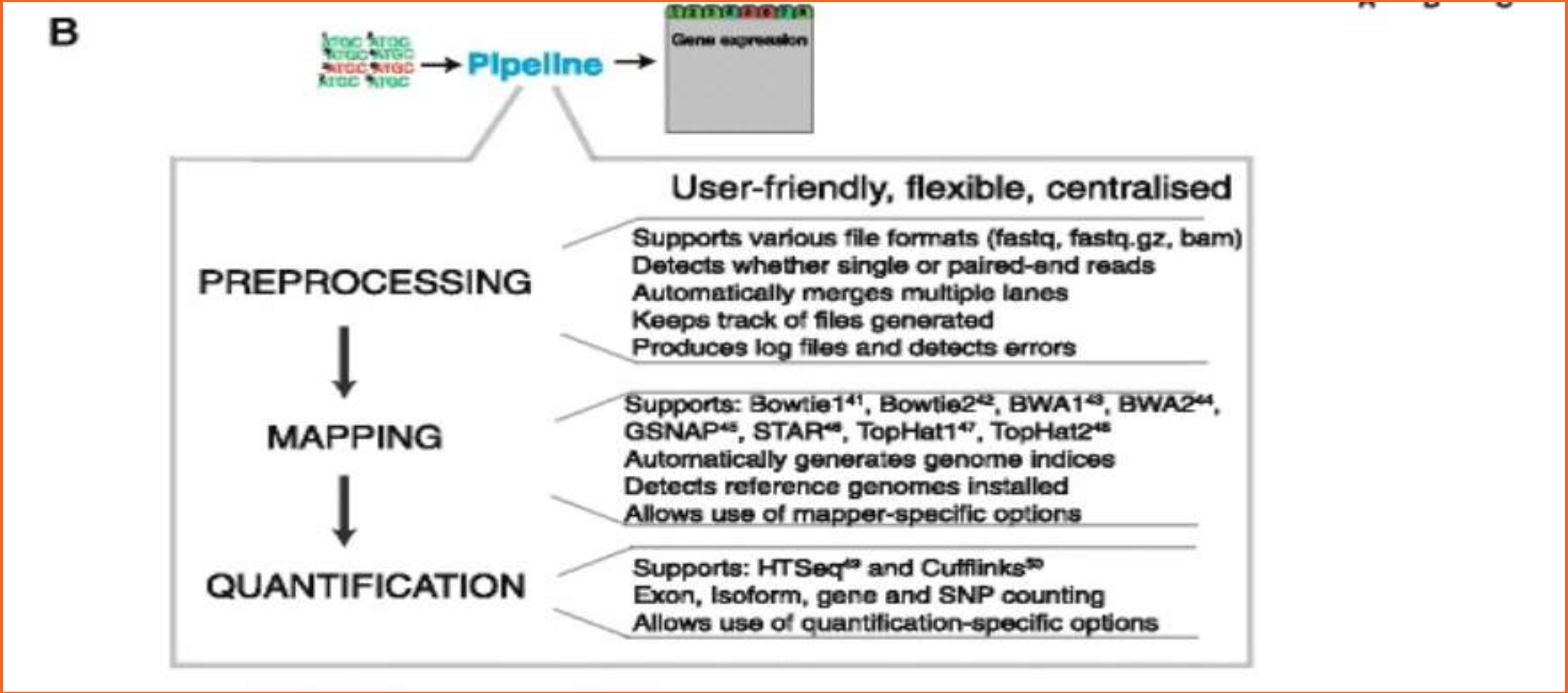
Adithya Ayanam - CTSI STEM Intern

Step A: RNA -Seq Workflow



Green indicates high and red low quality cell

Step B:Computational Pipeline to Read the RNA -Seq data



User-friendly, flexible, centralised

PREPROCESSING

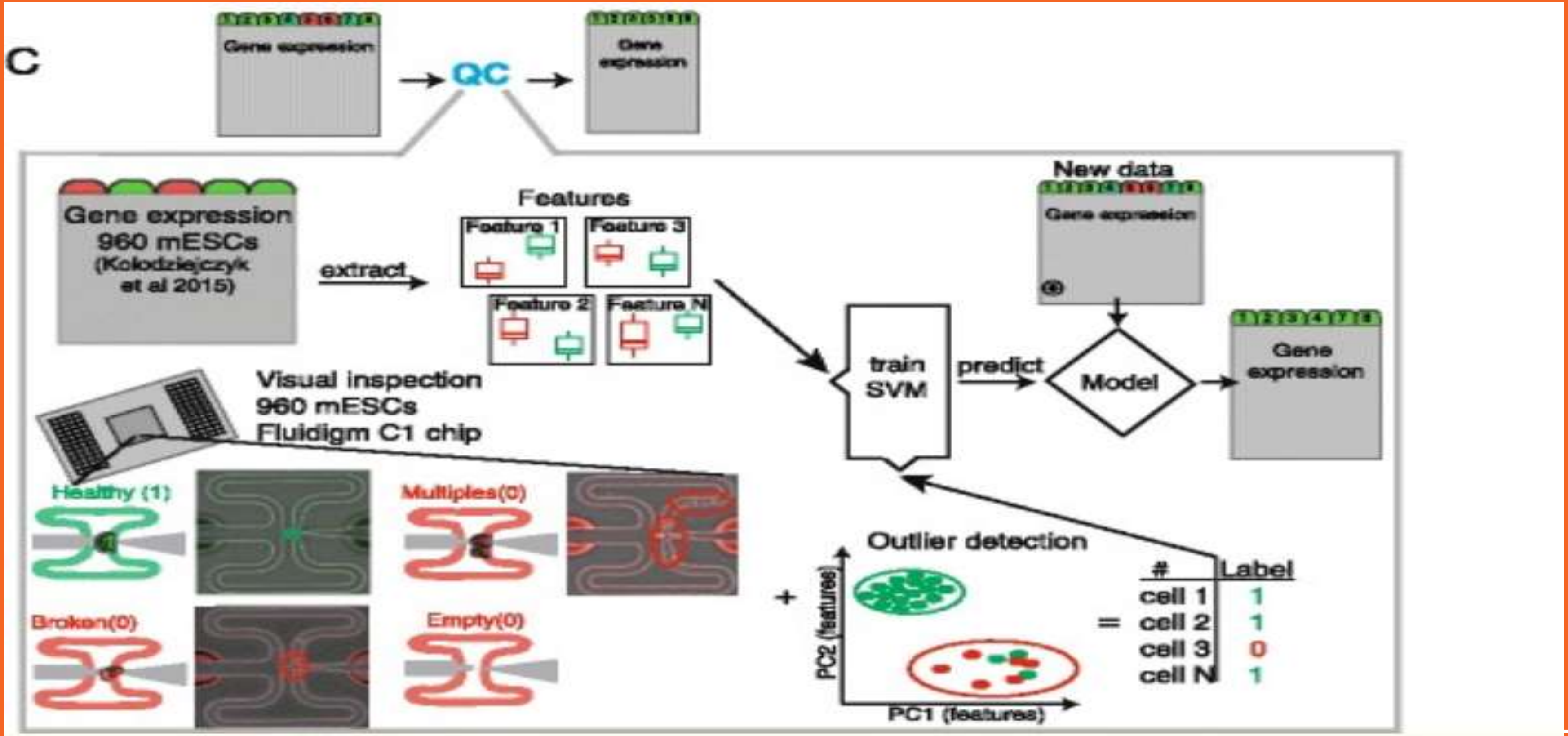
- Supports various file formats (fastq, fastq.gz, bam)
- Detects whether single or paired-end reads
- Automatically merges multiple lanes
- Keeps track of files generated
- Produces log files and detects errors

MAPPING

- Supports: Bowtie1[41], Bowtie2[42], BWA1[43], BWA2[44], GSNAP[45], STAR[46], TopHat1[47], TopHat2[48]
- Automatically generates genome indices
- Detects reference genomes installed
- Allows use of mapper-specific options

QUANTIFICATION

- Supports: HTSeq[49] and Cufflinks[50]
- Exon, Isoform, gene and SNP counting
- Allows use of quantification-specific options


C
Gene expression
QC
Gene expression
Gene expression
960 mESCs
(Kolodziejczyk et al 2015)
extract
Features
Feature 1
Feature 3
Feature 2
Feature N
New data
Gene expression
train SVM
predict
Model
Gene expression
Visual inspection
960 mESCs
Fluidigm C1 chip
Healthy (1)
Multiples(0)
Broken(0)
Empty(0)
Outlier detection
PC2 (features)
PC1 (features)
# Label
cell 1 1
cell 2 1
cell 3 0
cell N 1

Raw Data

Analyze the dataset of Pheripheral Blood Mononuclear Cells(PBMC)

Download the raw data from here.

The Read10X function reads in the output of the cellranger pipeline from 10X

The Read returns the UMI(Unique Moelcular identified) count matrix

The values in the matrix represents number of molecules for each feature(gene) that are detected in each cell.

Set up the Seurat object

Use the count matrix to create the Seurat Object

The Seurat Object act like a container that contains the
    -data (count matrix)
    -Analysis such as PCA or clustering results

Seurat object is a collection of Assay and DimReduc objects

Seurat Represents

- expression data - helds in Assay objects (CITE-seq ADTs, cell hashtags, imputed gene values)
- dimensionality reductions of the expression data. (transformation of the data using dim reduction technique)

Set up the Seurat object

```
library(dplyr)
library(Seurat)
```

Note the file path with backslash ---/data/pbmc3k/filtered_gene_bc_matrices/hg19/

```
# Load the PBMC dataset
pbmc <- Read10X(data.dir = "your file location ")

# Initialize the Seurat object with the raw (non-normalized data).
pbmc <- CreateSeuratObject(counts = pbmc, project = "pbmc3k", min.cells = 3,
min.features = 200)
pbmc
```

Set up the Seurat object

```
pbmc
```

Output

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

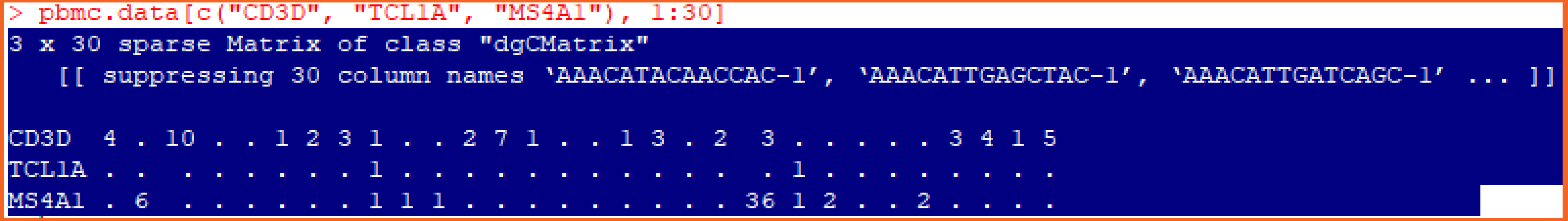
To Examine few genes in the first thirty cells

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

Set up the Seurat object

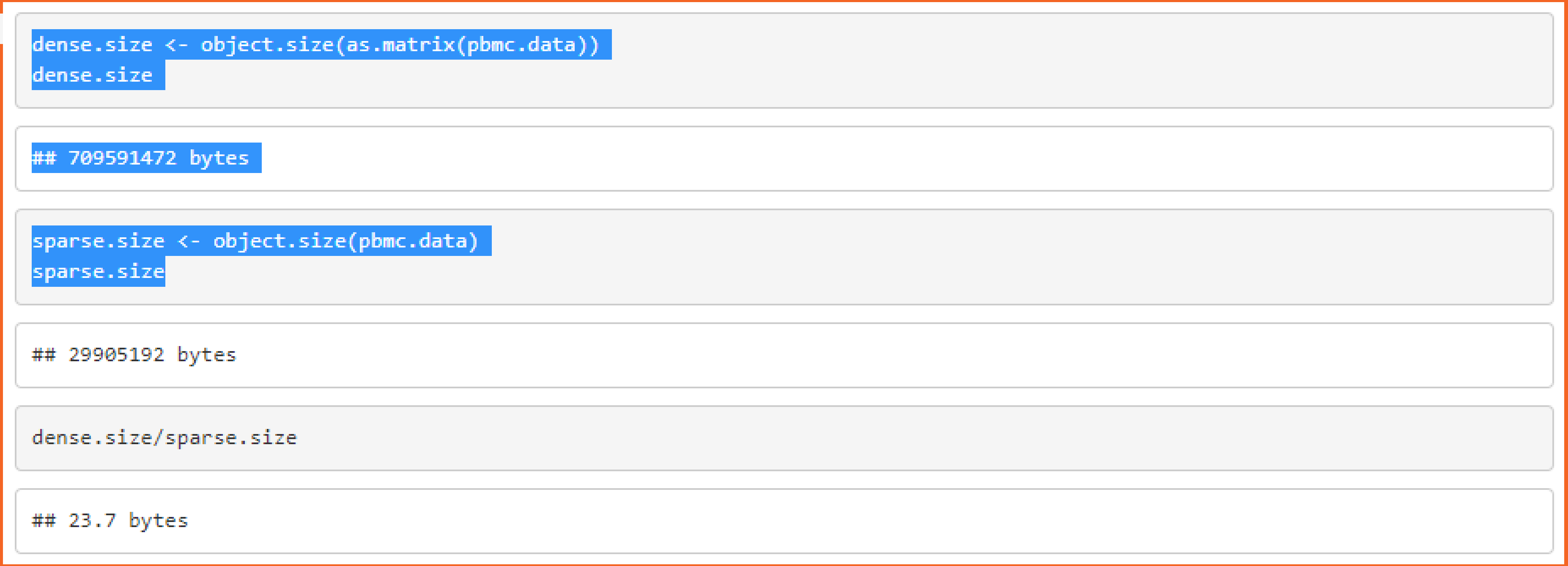
Interpretation of the output

- The “.” values in the matrix represent 0s(no molecules detected).
- Most values in an scRNA-seq matrix are 0, Seurat uses a sparse-matrix representation whenever possible.
- Significant memory and speed is saved for Drop-seq/inDrop/10x data.



```
> 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 . . . .
```

Set up the Seurat object



```r
dense.size <- object.size(as.matrix(pbmc.data))
dense.size
```

```
## 709591472 bytes
```

```r
sparse.size <- object.size(pbmc.data)
sparse.size
```

```
## 29905192 bytes
```

```r
dense.size/sparse.size
```

```
## 23.7 bytes
```

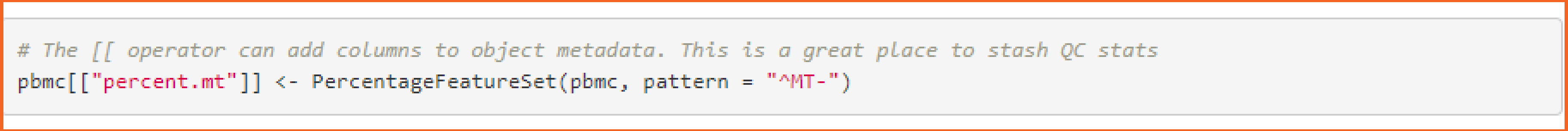
QC and selecting cells for further analysis

- 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

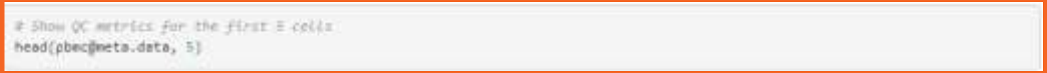
QC and selecting cells for further analysis



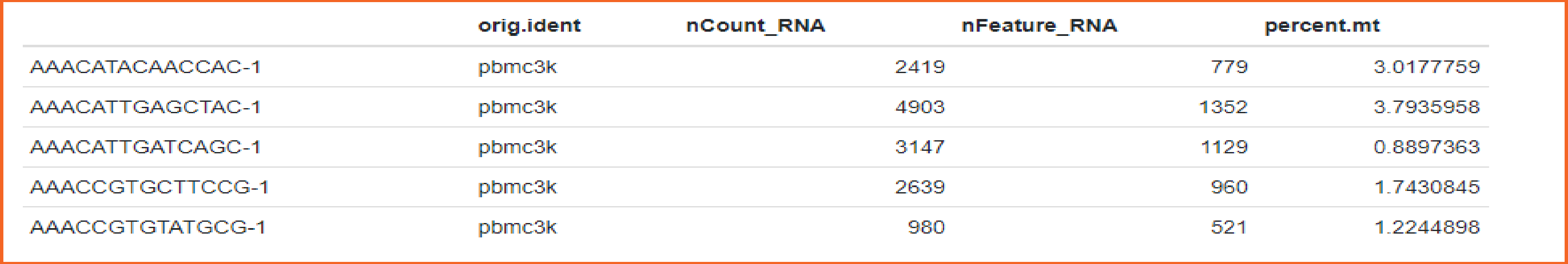
```
# The [[ operator can add columns to object metadata. This is a great place to stash QC stats
pbmc[["percent.mt"]] <- PercentageFeatureSet(pbmc, pattern = "^MT-")
```

The number of unique genes and total molecules are automatically calculated during CreateSeuratObject

- You can find them stored in the object meta data



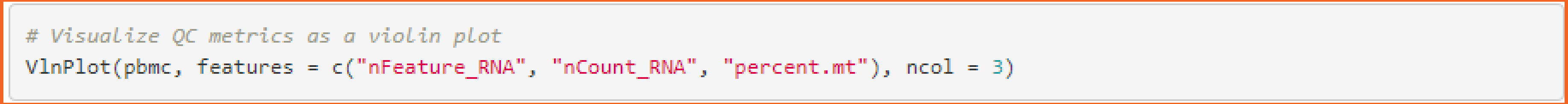
```
# Show QC metrics for the first 5 cells
head(pbmc@meta.data, 5)
```



	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

Visualize QC metrics

#A Violin Plot is used to visualise the distribution of the data and its probability density.This is done to view the QC metrics

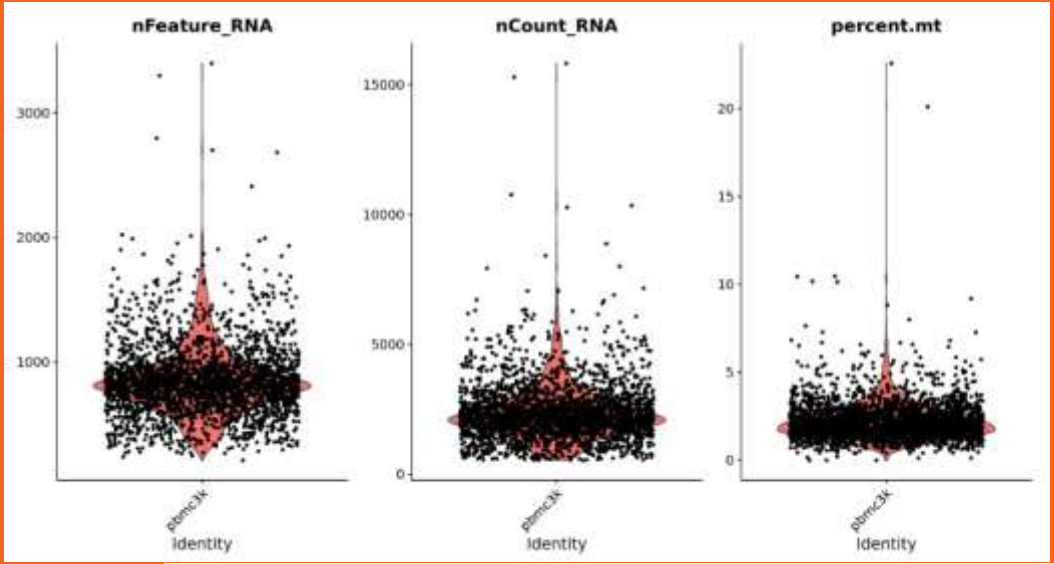


```r
# Visualize QC metrics as a violin plot
VlnPlot(pbmc, features = c("nFeature_RNA", "nCount_RNA", "percent.mt"), ncol = 3)
```

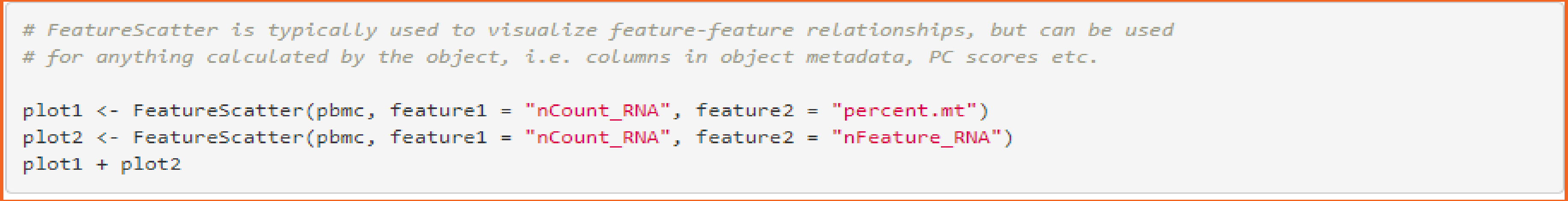
- We filter cells that have unique feature counts over 2,500 or less than 200
- We filter cells that have >5% mitochondrial counts

Visualize QC metrics

```
# Visualize QC metrics as a violin plot
VlnPlot(pbmc, features = c("nFeature_RNA", "nCount_RNA", "percent.mt"), ncol = 3)
```

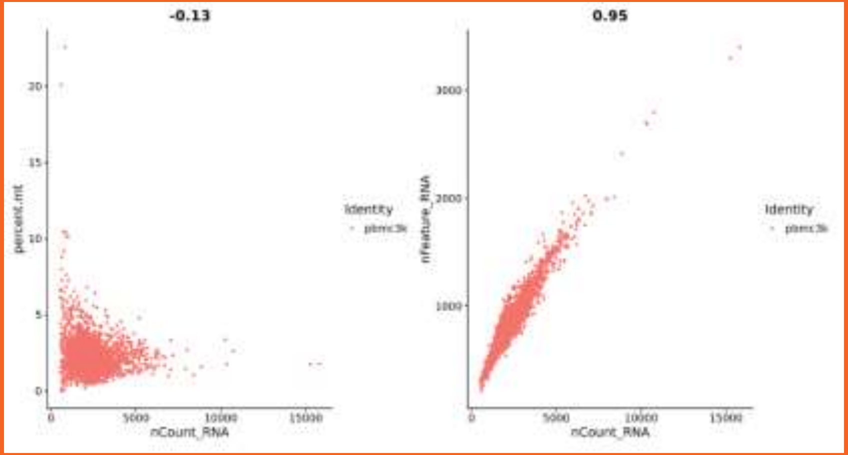


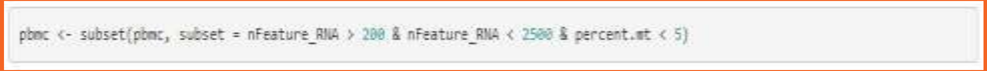
Visualize QC metrics



```
# 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.

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

- Subset is a function to filter the unwanted cells in the dataset.
- This function calculates the features that exhibit high cell-to cell variation in the dataset.
- some cells have highly expressed genes and some has lowly expressed gene





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

Normalize the data

- Normalize the data .
- Usually LogNormalize is used.
- This normalizes the feature expression measurements for each cell by the total expression,
- Multiplies this by a scale factor (10,000 by default),
- Log-transforms the result (satijalab).

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

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

Identify Variable Feature Selection

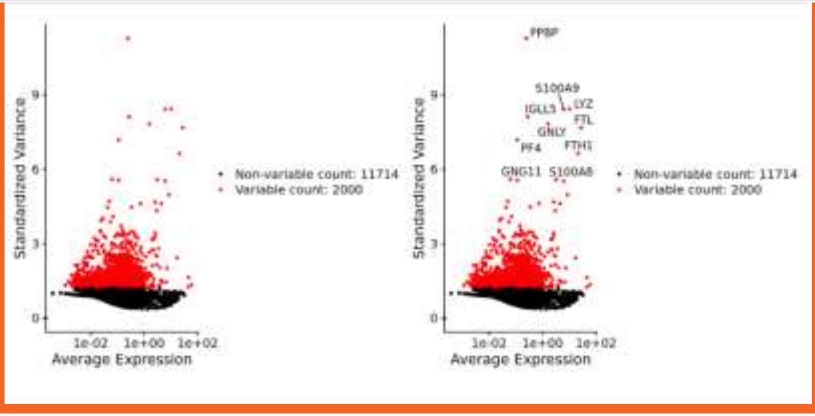
##

- Identify features that are outliers on a 'mean variability plot'.
- We use the vst selection method .
- vst: First, fits a line to the relationship of log(variance) and log(mean)
- Using local polynomial regression (loess). Then standardizes the feature values
- Using the observed mean and expected variance (given by the fitted line).
- Feature variance is then calculated on the standardized values after
- Clipping to a maximum (see clip.max parameter - default value is auto).
- Other methods are mean.var.plot(mvp)-identofy variable features while controlling
- The strong relationship between variability and average expression,
- dispersion(disp)- select genes with highest dispersion values.

##

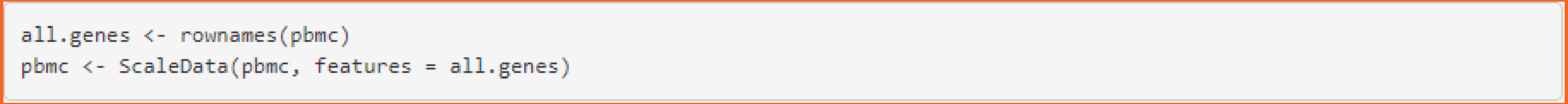
Identify Variable Feature Selection

```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 and without labels
plot1 <- VariableFeaturePlot(pbmc)
plot2 <- LabelPoints(plot = plot1, points = top10, repel = TRUE)
plot1 + plot2
```



Scaling the data

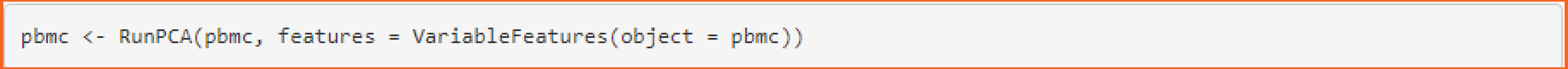
- Next, we apply a linear transformation (scaling) that is a standard pre-processing step prior to dimensional reduction techniques like PCA.
- The ScaleData function:
    - Shifts the expression of each gene, so that the mean expression across cells is 0
    - Scales the expression of each gene, so 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



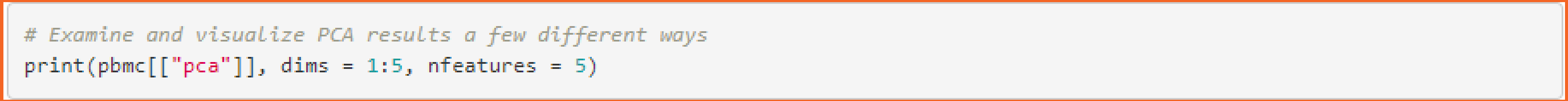
```
all.genes <- rownames(pbmc)
pbmc <- ScaleData(pbmc, features = all.genes)
```

Perform Linear Dimensional Reduction

- Through this line of code the PCA and clustering (analysis methods) results will be unaffected.
- The heatmaps (DoHeatmap function) will require genes for it to be scaled.
- After that we will perform PCA on the scaled data.
- The previously determined variable features are usually used as an input
- Through the use of the features argument you can choose another subset.

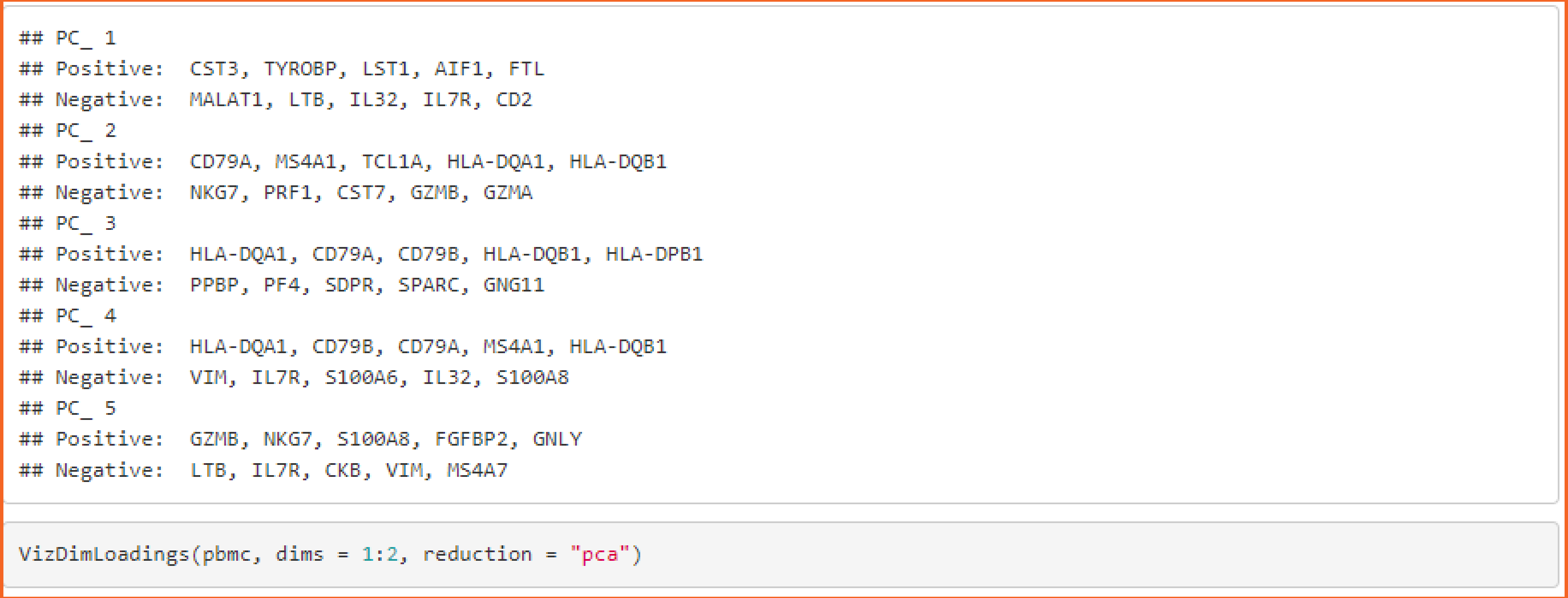


```
pbmc <- RunPCA(pbmc, features = VariableFeatures(object = pbmc))
```



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

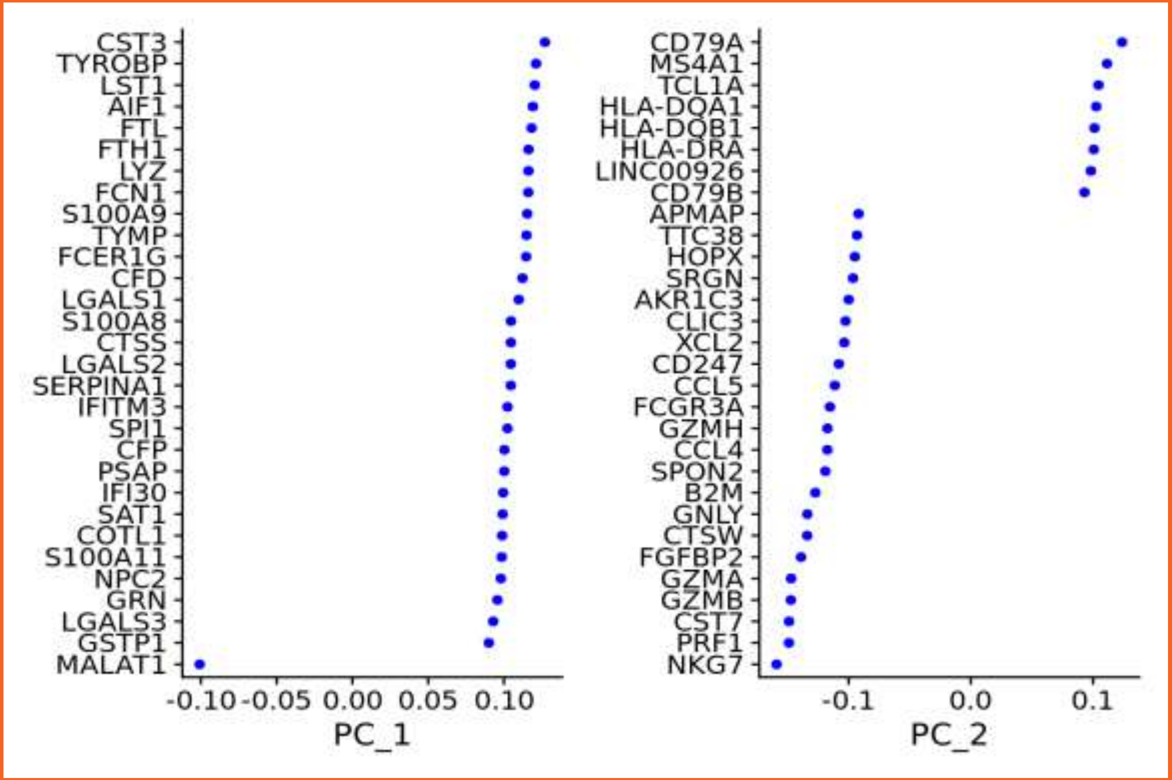
Perform Linear Dimensional Reduction



```
## 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
```

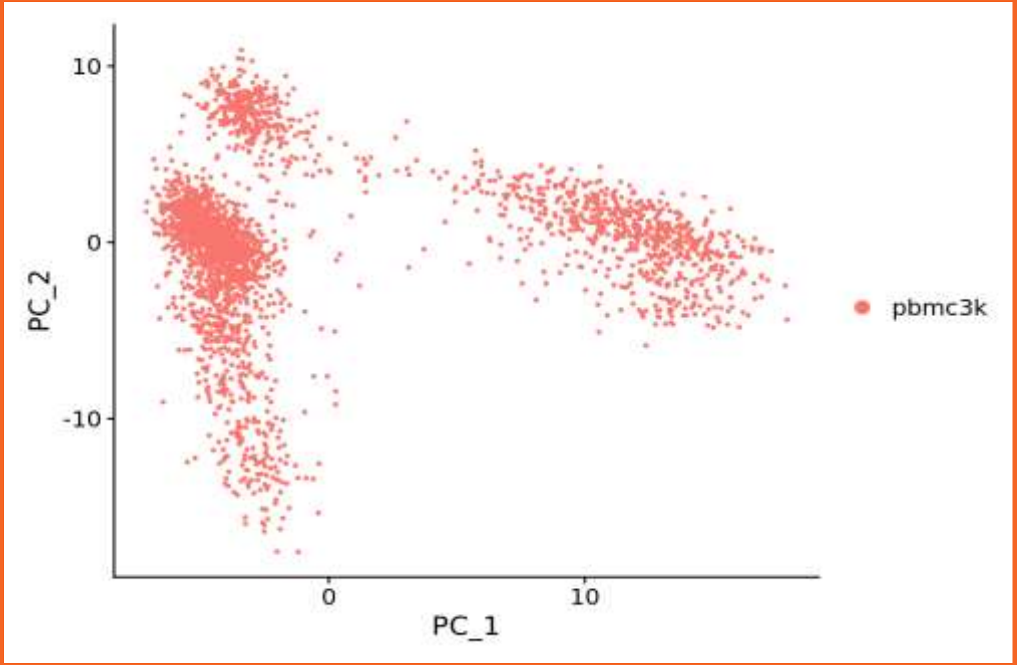
```
VizDimLoadings(pbmc, dims = 1:2, reduction = "pca")
```

Perform Linear Dimensional Reduction



Perform Linear Dimensional Reduction

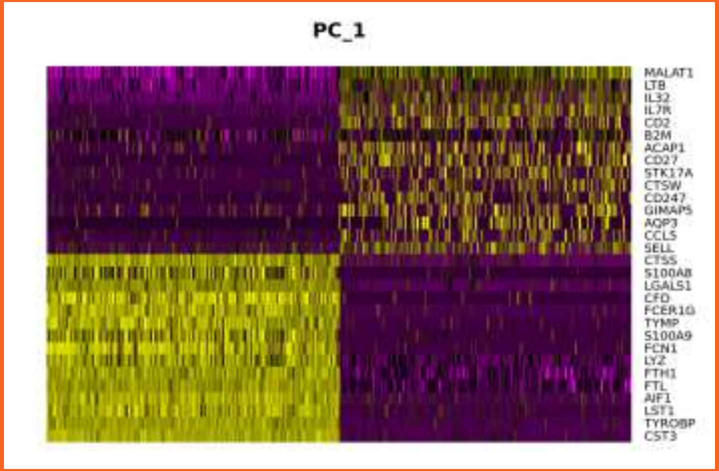
```
DimPlot(pbmc, reduction = "pca")
```



Perform Linear Dimensional Reduction

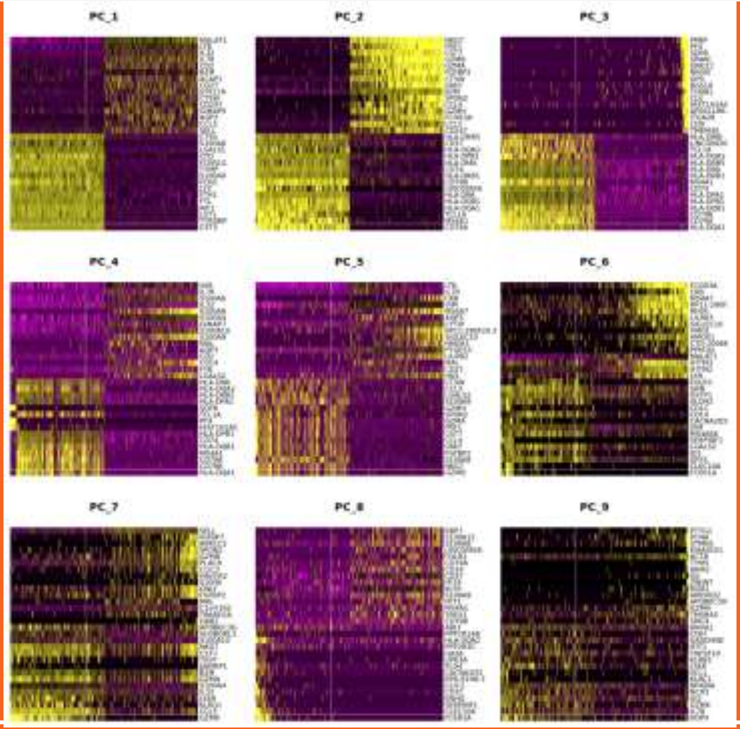
- Draws a heatmap focusing on a principal component.
- Both cells and genes are sorted by their principal component scores.
- Allows for nice visualization of sources of heterogeneity in the dataset.

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

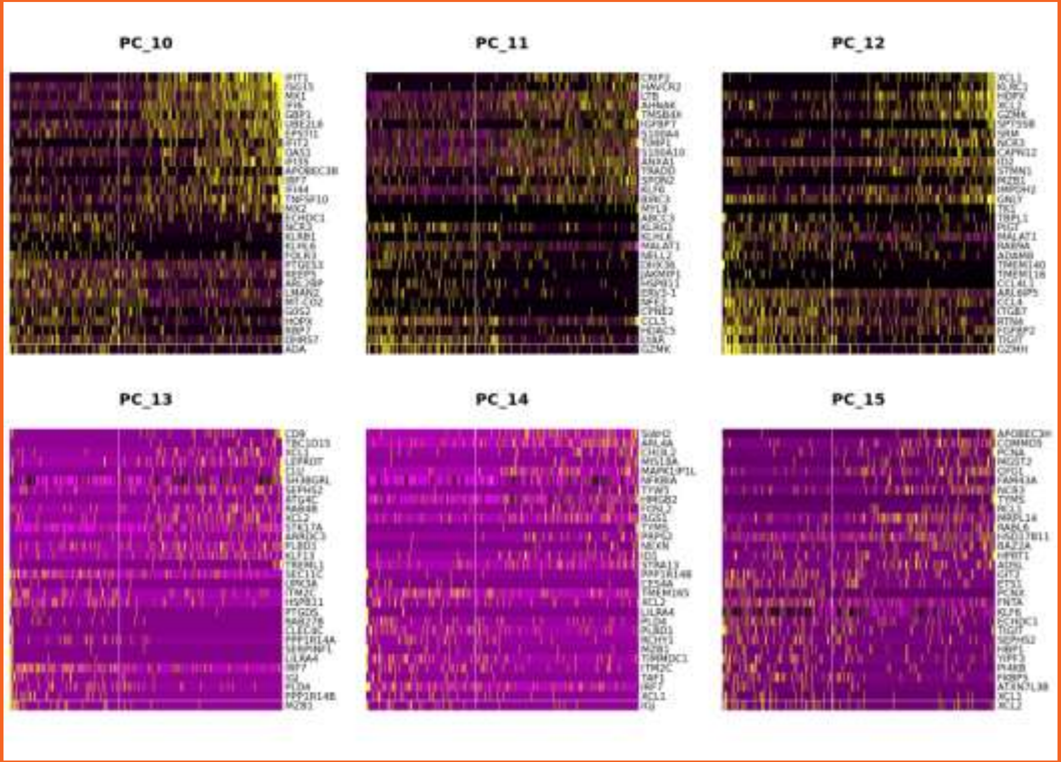


Perform Linear Dimensional Reduction

```
DimHeatmap(pbmc, dims = 1:15, cells = 500, balanced = TRUE)
```



Perform Linear Dimensional Reduction



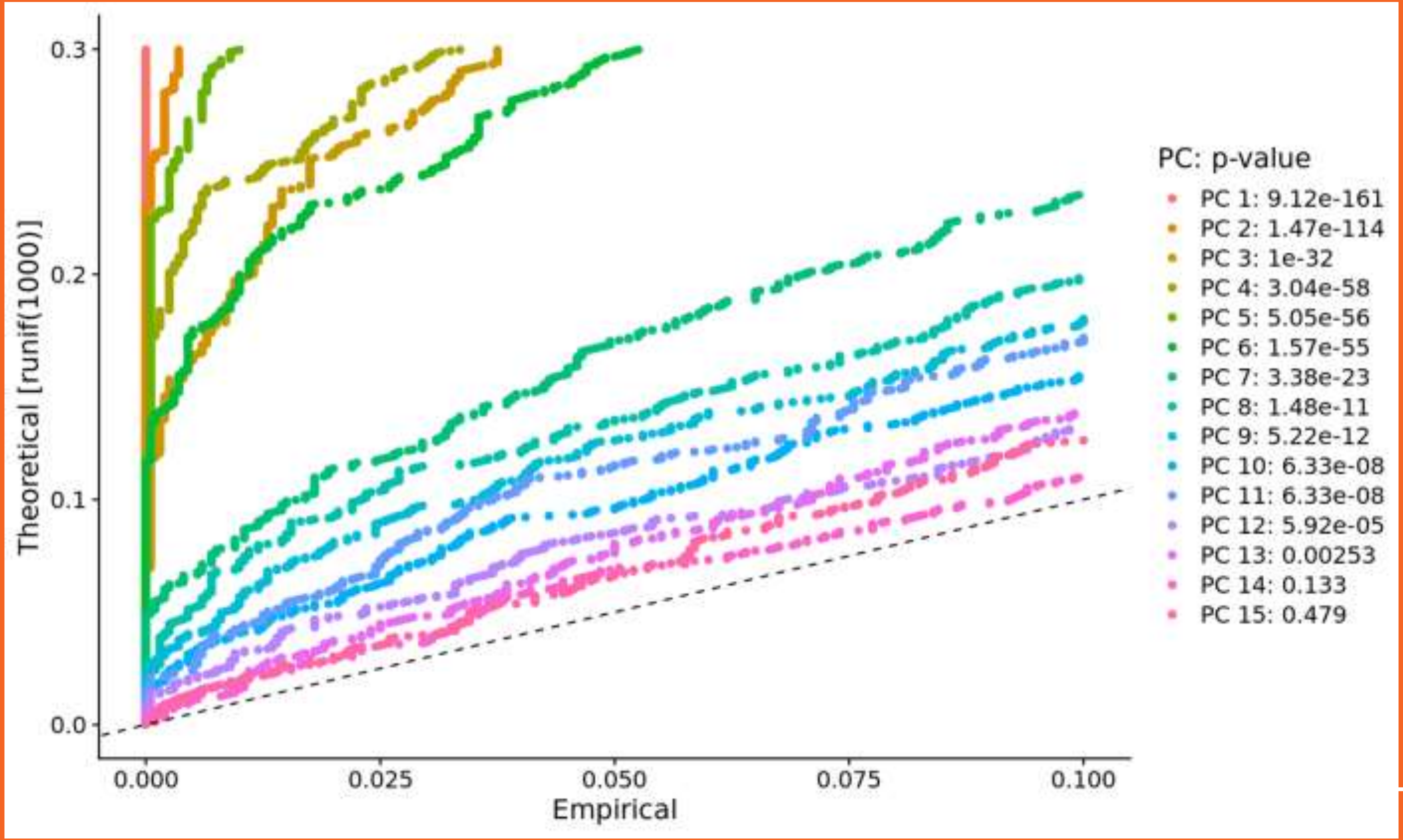
Determine the Dimensionality of the Dataset

- Determine Statistical Significance of PCA Scores.
- Randomly permutes a subset of data, and calculates projected
- PCA scores for these 'random' genes.
- Then compares the PCA scores for the 'random' genes with the observed PCA scores to determine statistical signifance.
- End result is a p-value for each gene's association with each principal component.

```
# NOTE: This process can take a long time for big datasets, comment out for expediency. More
# approximate techniques such as those implemented in ElbowPlot() can be used to reduce
# computation time
pbmc <- JackStraw(pbmc, num.replicate = 100)
pbmc <- ScoreJackStraw(pbmc, dims = 1:20)
```

```
JackStrawPlot(pbmc, dims = 1:15)
```

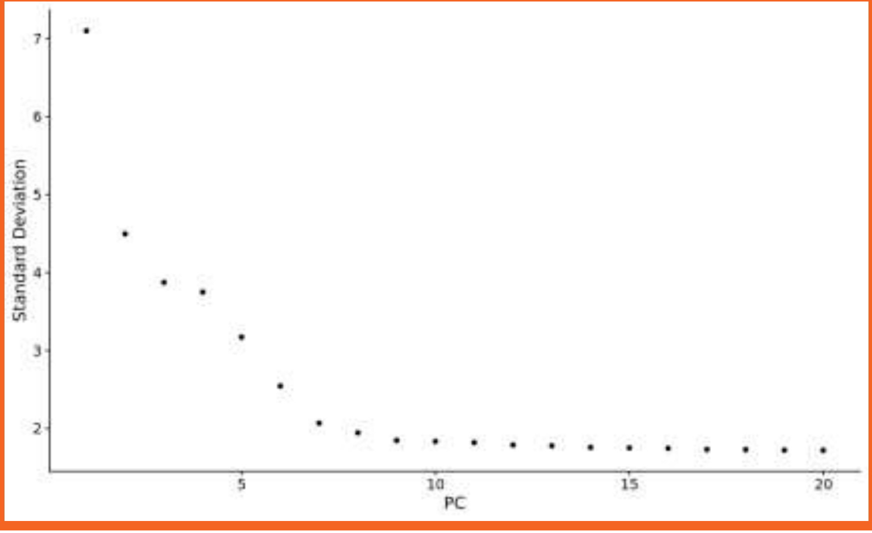
Determine the Dimensionality of the Dataset



Determine the Dimensionality of the Dataset

- Elbow Method for finding the optimal number of clusters
- This method is based on the percentage of variance.

```
ElbowPlot(pbmc)
```



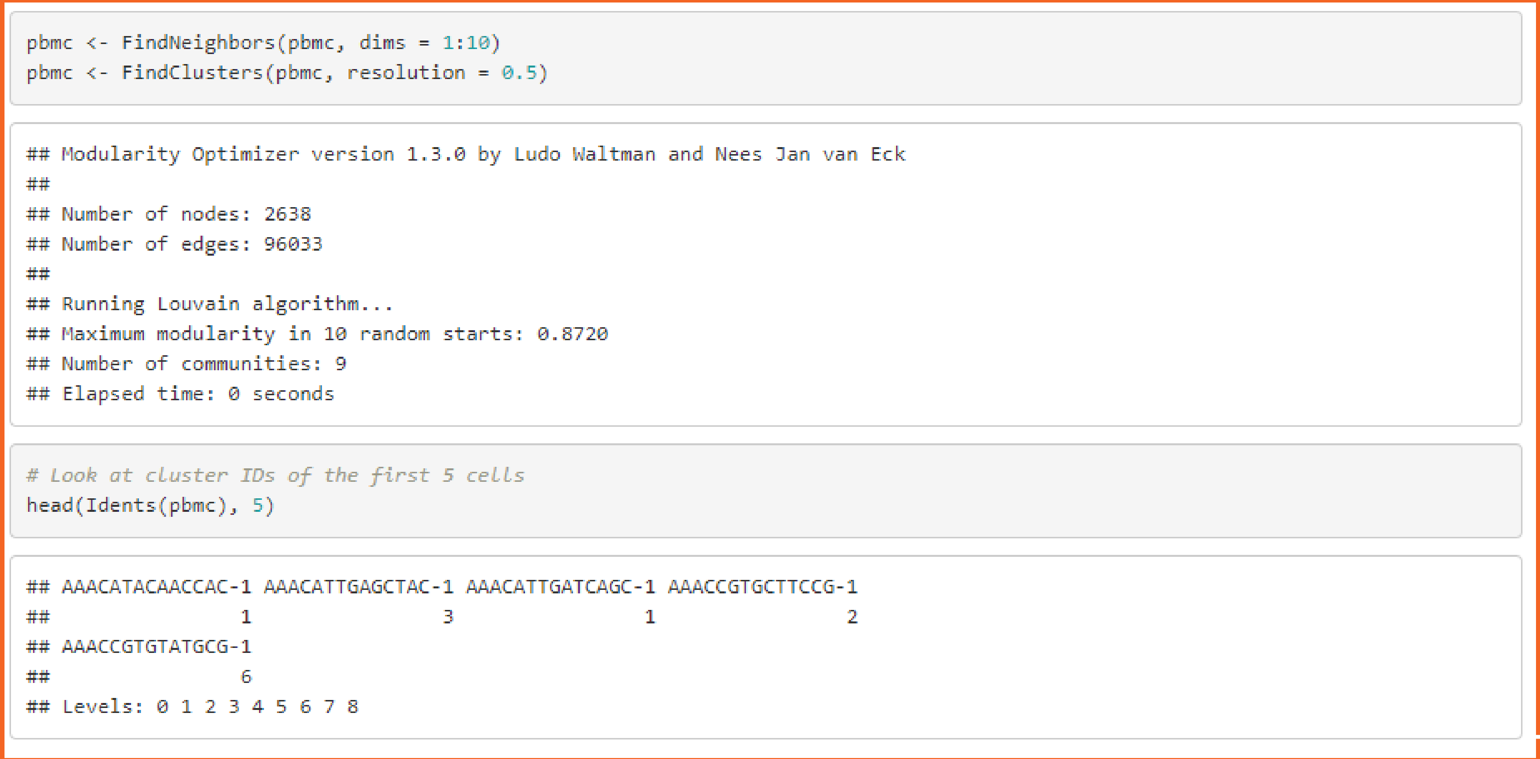
Cluster the cells

- Constructs a Shared Nearest Neighbor (SNN) Graph for a given dataset.

- We first determine the k-nearest neighbors of each cell.

- We use this knn graph to construct the SNN graph by calculating
    - the neighborhood overlap (Jaccard index) between every cell
    - and its k.param nearest neighbors.

Cluster the cells



```r
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: 96033
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.8720
## Number of communities: 9
## Elapsed time: 0 seconds
```

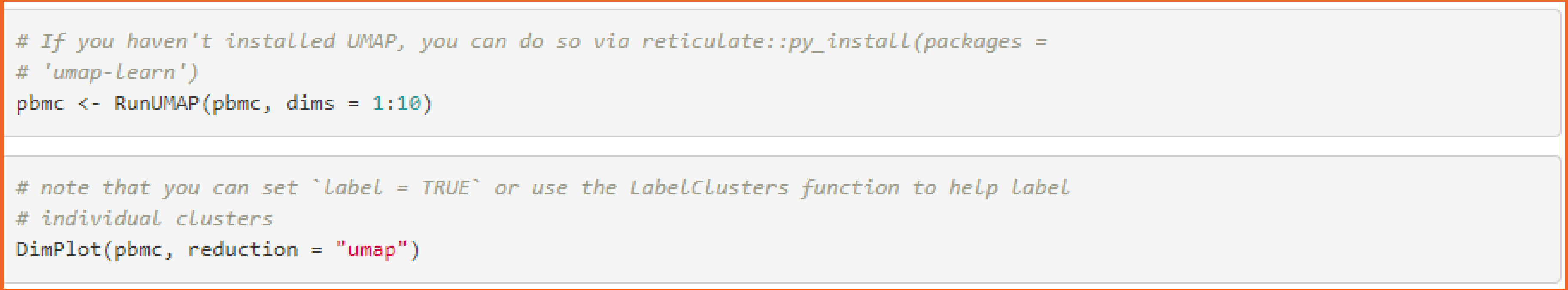
```r
# Look at cluster IDs of the first 5 cells
head(Idents(pbmc), 5)
```

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

Run non-linear Dimensional Reduction

UMAP - A non Linear Technique to explore dataset

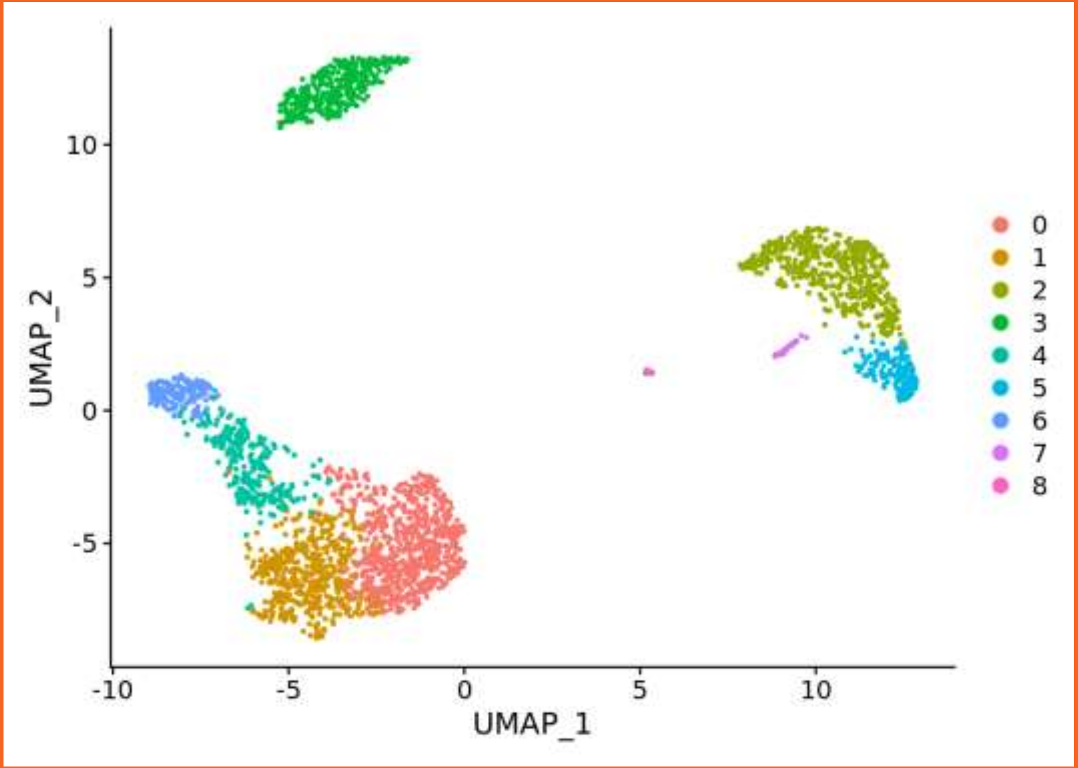
- Runs the Uniform Manifold Approximation and Projection (UMAP) dimensional reduction technique.
- To run, you must first install the umap-learn python package (e.g. via pip install umap-learn).
- Details on this package can be found here: https://github.com/lmcinnes/umap.



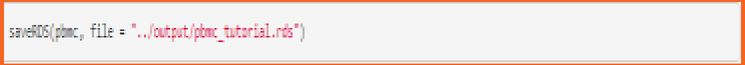
```r
# If you haven't installed UMAP, you can do so via reticulate::py_install(packages =
# 'umap-learn')
pbmc <- RunUMAP(pbmc, dims = 1:10)
```

```r
# note that you can set `label = TRUE` or use the LabelClusters function to help label
# individual clusters
DimPlot(pbmc, reduction = "umap")
```

Run non-linear Dimensional Reduction

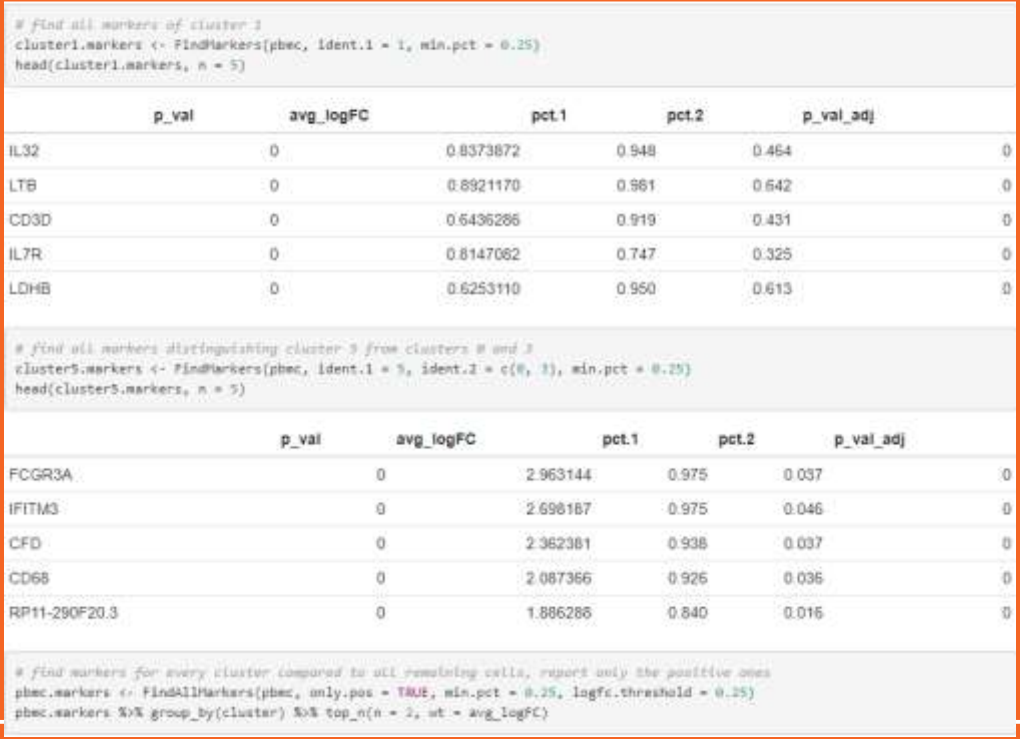


Save the object to make it easy to load it back



```
saveRDS(pbmc, file = "../output/pbmc_tutorial.rds")
```

Find Differentially expressed features- (Cluster biomarkers)



```
# find all markers of cluster 1
cluster1.markers <- FindMarkers(pbmc, ident.1 = 1, min.pct = 0.25)
head(cluster1.markers, n = 5)
```

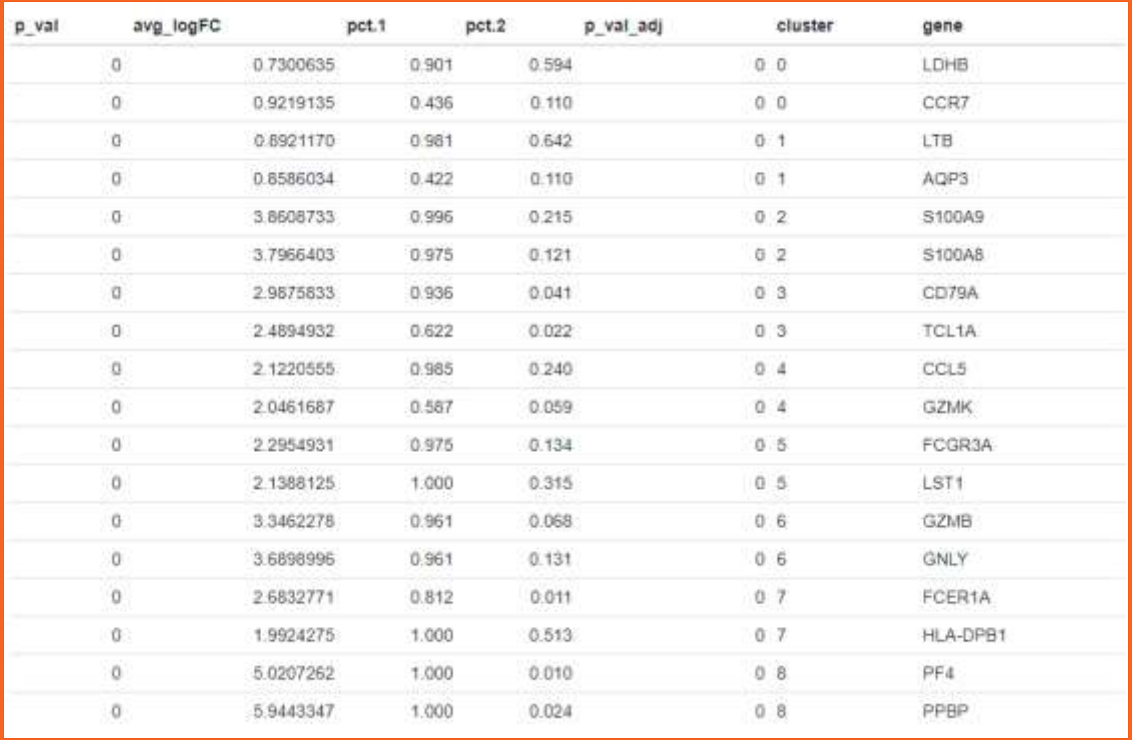
	p_val	avg_logFC	pct.1	pct.2	p_val_adj
IL32	0	0.8373872	0.948	0.464	0
LTB	0	0.8921170	0.981	0.642	0
CD3D	0	0.6436286	0.919	0.431	0
IL7R	0	0.8147082	0.747	0.325	0
LDHB	0	0.6253110	0.950	0.613	0

```
# find all markers distinguishing cluster 5 from clusters 0 and 3
cluster5.markers <- FindMarkers(pbmc, ident.1 = 5, ident.2 = c(0, 3), min.pct = 0.25)
head(cluster5.markers, n = 5)
```

	p_val	avg_logFC	pct.1	pct.2	p_val_adj
FCGR3A	0	2.963144	0.975	0.037	0
IFITM3	0	2.698187	0.975	0.046	0
CFD	0	2.362381	0.938	0.037	0
CD68	0	2.087366	0.926	0.036	0
RP11-290F20.3	0	1.886288	0.840	0.016	0

```
# find markers for every cluster compared to all remaining cells, report only the positive ones
pbmc.markers <- FindAllMarkers(pbmc, only.pos = TRUE, min.pct = 0.25, logfc.threshold = 0.25)
pbmc.markers %>% group_by(cluster) %>% top_n(n = 2, wt = avg_logFC)
```

Find Differentially expressed features- (Cluster biomarkers)



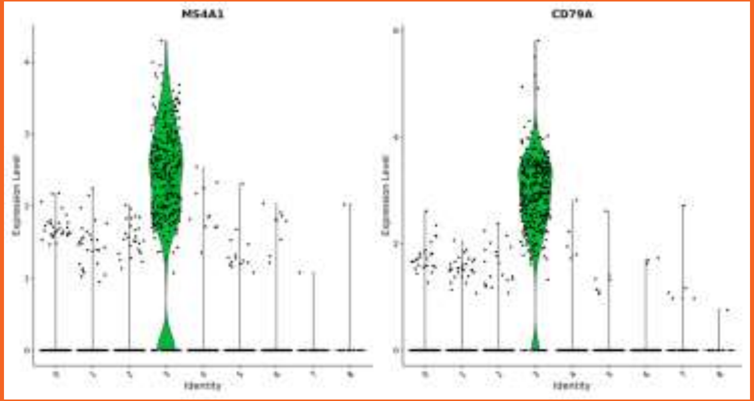
p_val	avg_logFC	pct.1	pct.2	p_val_adj	cluster	gene
0	0.7300635	0.901	0.594	0	0	LDHB
0	0.9219135	0.436	0.110	0	0	CCR7
0	0.8921170	0.981	0.642	0	1	LTB
0	0.8586034	0.422	0.110	0	1	AQP3
0	3.8608733	0.996	0.215	0	2	S100A9
0	3.7966403	0.975	0.121	0	2	S100A8
0	2.9875833	0.936	0.041	0	3	CD79A
0	2.4894932	0.622	0.022	0	3	TCL1A
0	2.1220555	0.985	0.240	0	4	CCL5
0	2.0461687	0.587	0.059	0	4	GZMK
0	2.2954931	0.975	0.134	0	5	FCGR3A
0	2.1388125	1.000	0.315	0	5	LST1
0	3.3462278	0.961	0.068	0	6	GZMB
0	3.6898996	0.961	0.131	0	6	GNLY
0	2.6832771	0.812	0.011	0	7	FCER1A
0	1.9924275	1.000	0.513	0	7	HLA-DPB1
0	5.0207262	1.000	0.010	0	8	PF4
0	5.9443347	1.000	0.024	0	8	PPBP

Find Differentially expressed features- (Cluster biomarkers)

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

```
cluster1.markers <- FindMarkers(pbmc, ident.1 = 0, logfc.threshold = 0.25, test.use = "roc", only.pos = TRUE)
```

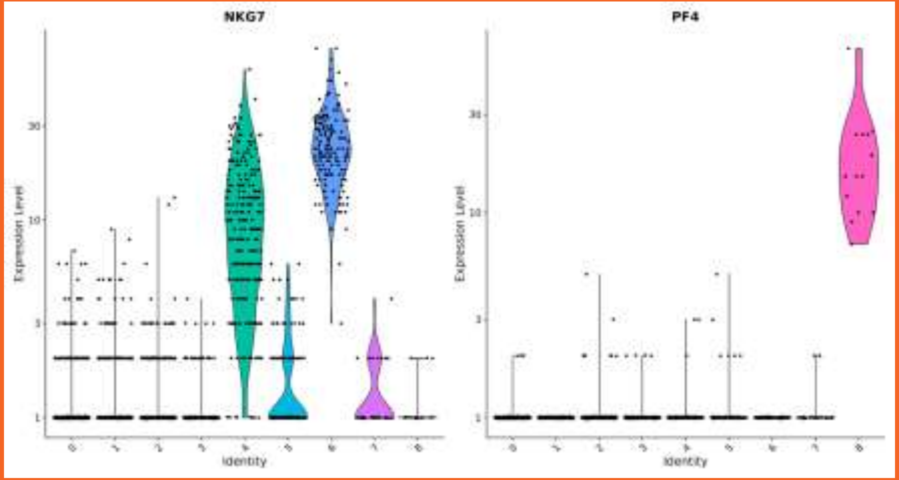
```
VlnPlot(pbmc, features = c("MS4A1", "CD79A"))
```



Find Differentially expressed features- (Cluster biomarkers)

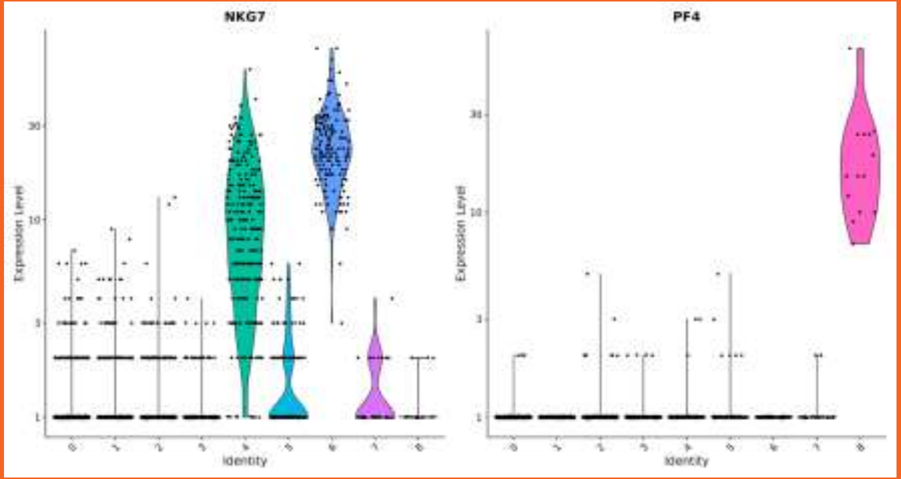
- Finds markers for every cluster compared to all remaining cells, report only the positive ones

```
# you can plot raw counts as well
VlnPlot(pbmc, features = c("NKG7", "PF4"), slot = "counts", log = TRUE)
```



Find Differentially expressed features- (Cluster biomarkers)

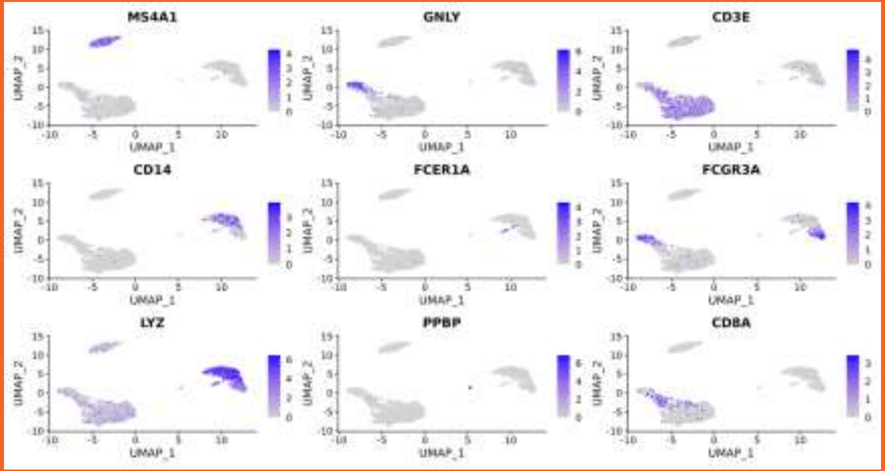
- The FeaturePlot function visualizes feature expression on a tSNE or PCA plot.



Find Differentially expressed features- (Cluster biomarkers)

- The FeaturePlot function visualizes feature expression on a tSNE or PCA plot.

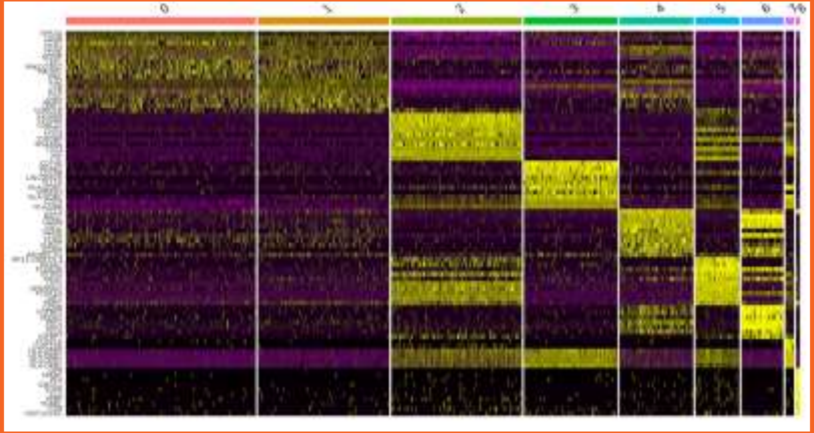
```
FeaturePlot(pbmc, features = c("MS4A1", "GNLY", "CD3E", "CD14", "FCER1A", "FCGR3A", "LYZ", "PPBP",
    "CD8A"))
```



Find Differentially expressed features-(Cluster biomarkers)

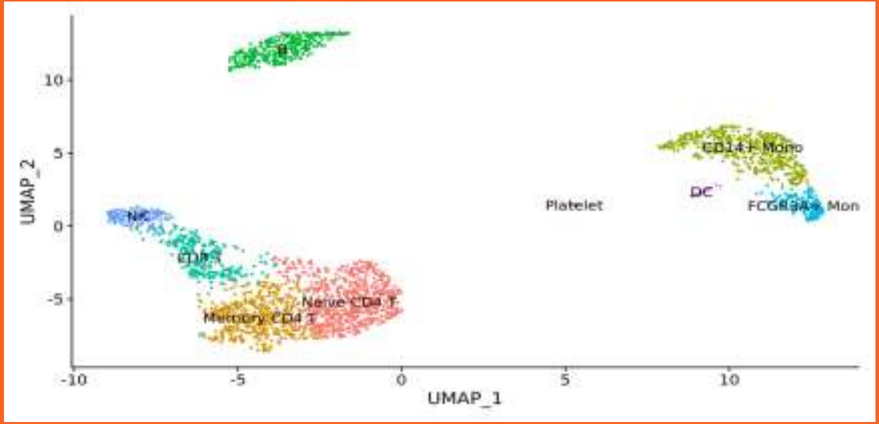
- The DoHeatMap creates a heatmap for the given cells and features.

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



Assign cell type identity to clusters.

```
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 = "../output/pbmc3k_final.rds")
```