Seurat Tutorials

Basic: https://satijalab.org/seurat/articles/pbmc3k_tutorial.html How to asign HTO info to seurat object: https://satijalab.org/seurat/articles/multimodal_vignette.html How to demultiplex data based on HTO: https://satijalab.org/seurat/articles/hashing_vignette.html To combine two 10x runs: https://satijalab.org/seurat/articles/merge_vignette.html To use SingleR: https://www.singlecellcourse.org/single-cell-rna-seq-analysis-using-seurat.html#cell-type-annotation-using-singler Pseudotime with monocle3: https://rpubs.com/mahima_bose/Seurat_and_Monocle3_p

Loading libraries

install.packages("Seurat") if (!require("BiocManager", quietly = TRUE)) install.packages("BiocManager") BiocManager::install("SingleR") BiocManager::install("celldex") BiocManager::install("SingleCellExperiment") BiocManager::install("DropletUtils") BiocManager::install("dittoSeq")

```
library(ggrepel)
```

```
## Loading required package: ggplot2
```

```
library(Seurat)
```

```
## Loading required package: SeuratObject
```

```
## Loading required package: sp
```

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

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

```
library(ggplot2)
library(SingleR)
```

```
## Loading required package: SummarizedExperiment
```

```
## Loading required package: MatrixGenerics
```

```
## Loading required package: matrixStats
```

```
##
## Attaching package: 'MatrixGenerics'
```

```
## The following objects are masked from 'package:matrixStats':
##
##     colAlls, colAnyNAs, colAnys, colAvgsPerRowSet, colCollapse,
##     colCounts, colCummaxs, colCummins, colCumprods, colCumsums,
##     colDiffs, colIQRDiffs, colIQRs, colLogSumExps, colMadDiffs,
##     colMads, colMaxs, colMeans2, colMedians, colMins, colOrderStats,
##     colProds, colQuantiles, colRanges, colRanks, colSdDiffs, colSds,
##     colSums2, colTabulates, colVarDiffs, colVars, colWeightedMads,
##     colWeightedMeans, colWeightedMedians, colWeightedSds,
##     colWeightedVars, rowAlls, rowAnyNAs, rowAnys, rowAvgsPerColSet,
##     rowCollapse, rowCounts, rowCummaxs, rowCummins, rowCumprods,
##     rowCumsums, rowDiffs, rowIQRDiffs, rowIQRs, rowLogSumExps,
##     rowMadDiffs, rowMads, rowMaxs, rowMeans2, rowMedians, rowMins,
##     rowOrderStats, rowProds, rowQuantiles, rowRanges, rowRanks,
##     rowSdDiffs, rowSds, rowSums2, rowTabulates, rowVarDiffs, rowVars,
##     rowWeightedMads, rowWeightedMeans, rowWeightedMedians,
##     rowWeightedSds, rowWeightedVars
```

```
## Loading required package: GenomicRanges
```

```
## Loading required package: stats4
```

```
## Loading required package: BiocGenerics
```

```
##
## Attaching package: 'BiocGenerics'
```

```
## The following object is masked from 'package:SeuratObject':
##
##     intersect
```

```
## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs
```

```
## The following objects are masked from 'package:base':
##
##     anyDuplicated, aperm, append, as.data.frame, basename, cbind,
##     colnames, dirname, do.call, duplicated, eval, evalq, Filter, Find,
##     get, grep, grepl, intersect, is.unsorted, lapply, Map, mapply,
##     match, mget, order, paste, pmax, pmax.int, pmin, pmin.int,
##     Position, rank, rbind, Reduce, rownames, sapply, setdiff, sort,
##     table, tapply, union, unique, unsplit, which.max, which.min
```

```
## Loading required package: S4Vectors
```

```
##
## Attaching package: 'S4Vectors'
```

```
## The following object is masked from 'package:utils':
##
##     findMatches
```

```
## The following objects are masked from 'package:base':
##
##     expand.grid, I, unname
```

```
## Loading required package: IRanges
```

```
##
## Attaching package: 'IRanges'
```

```
## The following object is masked from 'package:sp':
##
##     %over%
```

```
## Loading required package: GenomeInfoDb
```

```
## Loading required package: Biobase
```

```
## Welcome to Bioconductor
##
##     Vignettes contain introductory material; view with
##     'browseVignettes()'. To cite Bioconductor, see
##     'citation("Biobase")', and for packages 'citation("pkgname")'.
```

```
##
## Attaching package: 'Biobase'
```

```
## The following object is masked from 'package:MatrixGenerics':
##
##     rowMedians
```

```
## The following objects are masked from 'package:matrixStats':
##
##     anyMissing, rowMedians
```

```
##
## Attaching package: 'SummarizedExperiment'
```

```
## The following object is masked from 'package:Seurat':
```

```
##
##     Assays
```

```
## The following object is masked from 'package:SeuratObject':
##
##     Assays
```

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
```

```
## The following object is masked from 'package:Biobase':
##
##     combine
```

```
## The following objects are masked from 'package:GenomicRanges':
##
##     intersect, setdiff, union
```

```
## The following object is masked from 'package:GenomeInfoDb':
##
##     intersect
```

```
## The following objects are masked from 'package:IRanges':
##
##     collapse, desc, intersect, setdiff, slice, union
```

```
## The following objects are masked from 'package:S4Vectors':
##
##     first, intersect, rename, setdiff, setequal, union
```

```
## The following objects are masked from 'package:BiocGenerics':
##
##     combine, intersect, setdiff, union
```

```
## The following object is masked from 'package:matrixStats':
##
##     count
```

```
## The following objects are masked from 'package:stats':
##
##     filter, lag
```

```
## The following objects are masked from 'package:base':
##
```

```
##     intersect, setdiff, setequal, union
```

```
library(celldex)
```

```
##
## Attaching package: 'celldex'
```

```
## The following objects are masked from 'package:SingleR':
##
##     BlueprintEncodeData, DatabaseImmuneCellExpressionData,
##     HumanPrimaryCellAtlasData, ImmGenData, MonacoImmuneData,
##     MouseRNAseqData, NovershternHematopoieticData
```

```
library(RColorBrewer)
library(SingleCellExperiment)
library(DropletUtils)
library(Matrix)
```

```
##
## Attaching package: 'Matrix'
```

```
## The following object is masked from 'package:S4Vectors':
##
##     expand
```

```
library(knitr)
library(reshape2)
library(dittoSeq)
library(viridis)
```

```
## Loading required package: viridisLite
```

```
library(RColorBrewer)
library(wesanderson)
library(DoubletFinder)
library(SeuratDisk)
```

```
## Registered S3 method overwritten by 'SeuratDisk':
##   method            from
##   as.sparse.H5Group Seurat
```

```
library(httr)
```

```
##
## Attaching package: 'httr'
```

```
## The following object is masked from 'package:Biobase':
##
##     content
```

```
library(scDblFinder)
library(tibble)
library(tidyr)
```

```
##
## Attaching package: 'tidyr'
```

```
## The following object is masked from 'package:reshape2':
##
##     smiths
```

```
## The following objects are masked from 'package:Matrix':
##
##     expand, pack, unpack
```

```
## The following object is masked from 'package:S4Vectors':
##
##     expand
```

```
library(MAST)
library(patchwork)
library(ggsignif)
library(ggtext)
```

Importing 10X data

The function Read10X() imports sparse matrix generated by Cellranger into the variable rawdata. The variable rawdata is a unique molecular identified (UMI) count matrix. The values in this matrix represent the number of molecules for each feature (i.e. gene; row) that are detected in each cell (column).

In our case, this object contain two list: - Gene Expression (GEX) - Antibody capture (HTO)

```
data_BM <- Read10X(data.dir = "../Data/sample_filtered_feature_bc_matrix_BM/")
```

```
## 10X data contains more than one type and is being returned as a list containing matrices of
 each type.
```

```
data_SP <- Read10X(data.dir = "../Data/sample_filtered_feature_bc_matrix_SP/")
```

```
## 10X data contains more than one type and is being returned as a list containing matrices of
 each type.
```

Create Seurat object

In the following code genes detected in less than 3 cells are filtered out.

```
Seurat_Object_BM <- CreateSeuratObject(counts = data_BM$`Gene Expression`, project = "10x_A12_
BM", min.cells = 3)
Seurat_Object_BM
```

```
## An object of class Seurat
## 19884 features across 19486 samples within 1 assay
## Active assay: RNA (19884 features, 0 variable features)
##  1 layer present: counts
```

```
Seurat_Object_SP <- CreateSeuratObject(counts = data_SP$`Gene Expression`, project = "10x_A12_
SP", min.cells = 3)
Seurat_Object_SP
```

```
## An object of class Seurat
## 17662 features across 7698 samples within 1 assay
## Active assay: RNA (17662 features, 0 variable features)
##  1 layer present: counts
```

Add HTO information to seurat object

The seurat object contains an assay storing RNA measurement, but we add another assay with HTO info.

```
# create a new assay to store HTO information
HTO_assay <- CreateAssayObject(counts = data_BM$`Antibody Capture`)
```

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

```
# add this assay to the previously created Seurat object
Seurat_Object_BM$HTO <- HTO_assay

# Validate that the object now contains an assay with HTO info
rownames(Seurat_Object_BM$HTO)
```

```
## [1] "BM-WT-1"  "BM-A12-1" "BM-WT-2"  "BM-WT-3"  "BM-A12-2" "BM-A12-3" "huK"
```

```
# Note that all operations below are performed on the RNA assay Set and verify that the defaul
t assay is RNA.
DefaultAssay(Seurat_Object_BM)
```

```
## [1] "RNA"
```

```
# create a new assay to store HTO information
HTO_assay <- CreateAssayObject(counts = data_SP$`Antibody Capture`)
```

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

```
# add this assay to the previously created Seurat object
Seurat_Object_SP$HTO <- HTO_assay

# Validate that the object now contains an assay with HTO info
rownames(Seurat_Object_SP$HTO)
```

```
## [1] "SP-WT-1"  "SP-A12-1" "SP-WT-2"  "SP-WT-3"  "SP-A12-2" "SP-A12-3" "huK"
```

```
# Note that all operations below are performed on the RNA assay Set and verify that the defaul
t assay is RNA.
DefaultAssay(Seurat_Object_SP)
```

```
## [1] "RNA"
```

QUALITY CONTROL

First, we want to study the percentage of reads that map to mitochondrial genome, since Low-quality / dying cells often exhibit extensive mitochondrial contamination

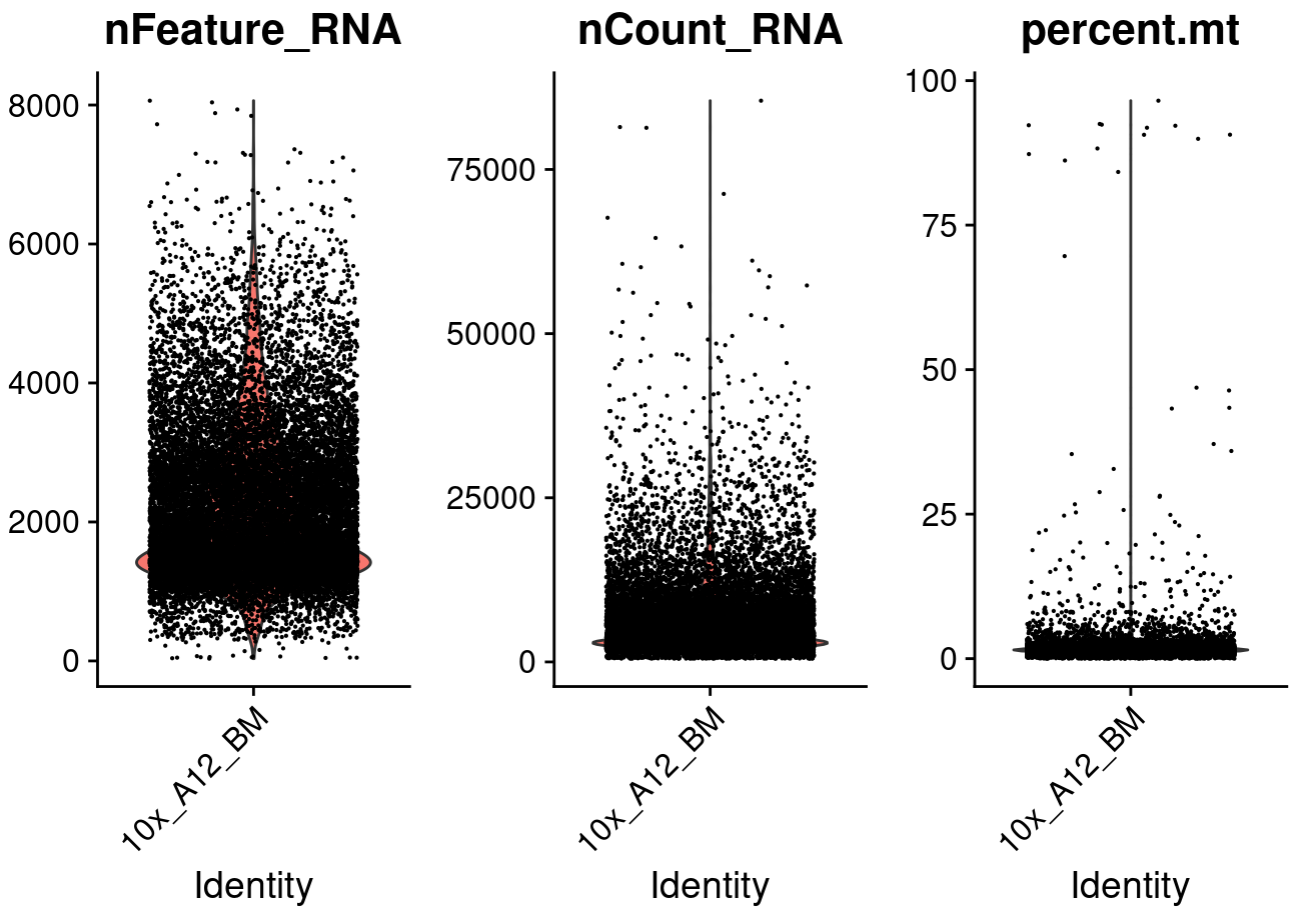
```
# The [[ operator can add columns to object metadata. This is a great place to stash QC stats
Seurat_Object_BM[["percent.mt"]] <- PercentageFeatureSet(Seurat_Object_BM, pattern = "^mt-")
```

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

Visualize QC metrics:

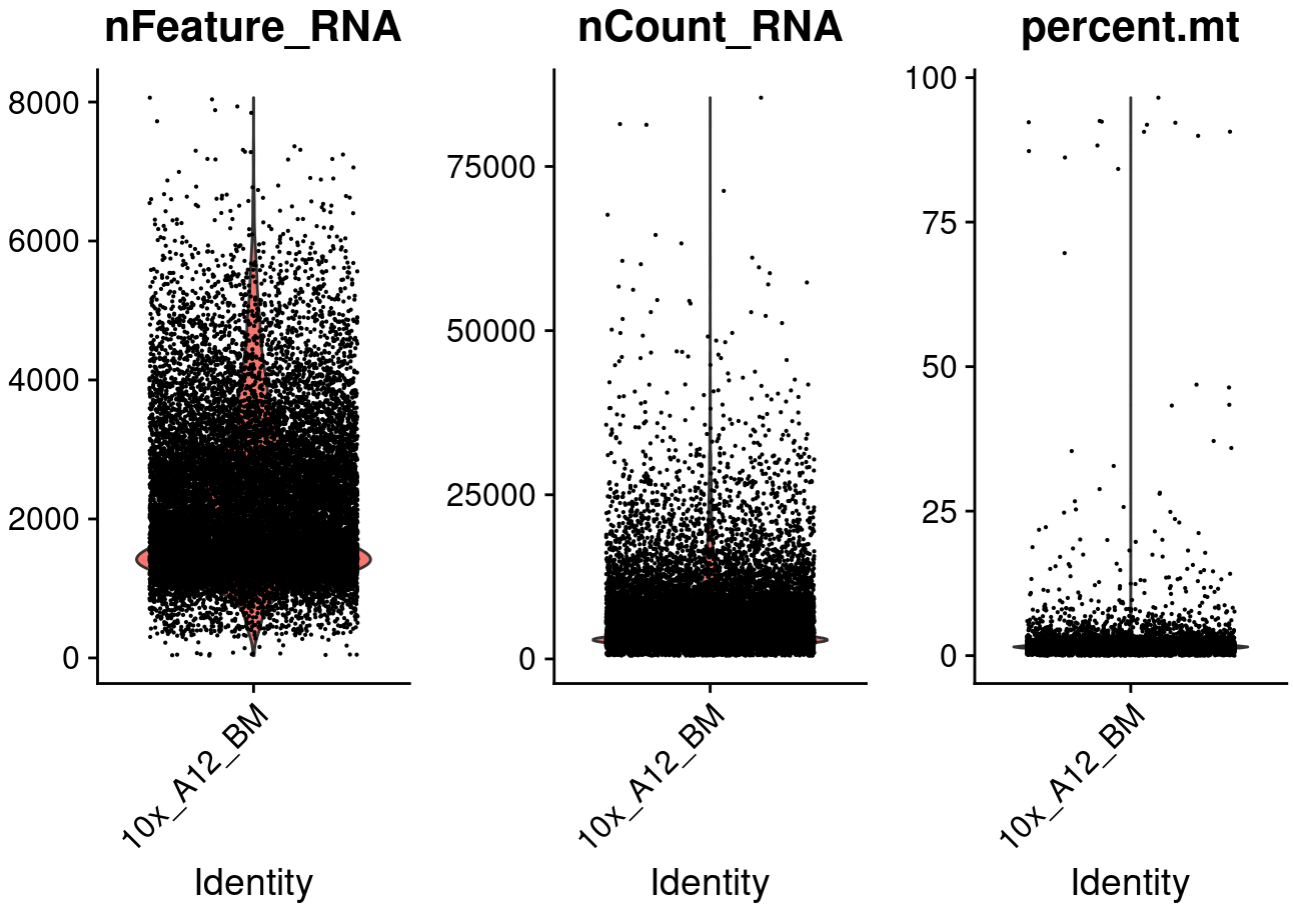
```
# violin plot
Idents(Seurat_Object_BM) <- Seurat_Object_BM$orig.ident
VlnPlot(Seurat_Object_BM, features = c("nFeature_RNA", "nCount_RNA", "percent.mt"), ncol = 3)
```

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




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

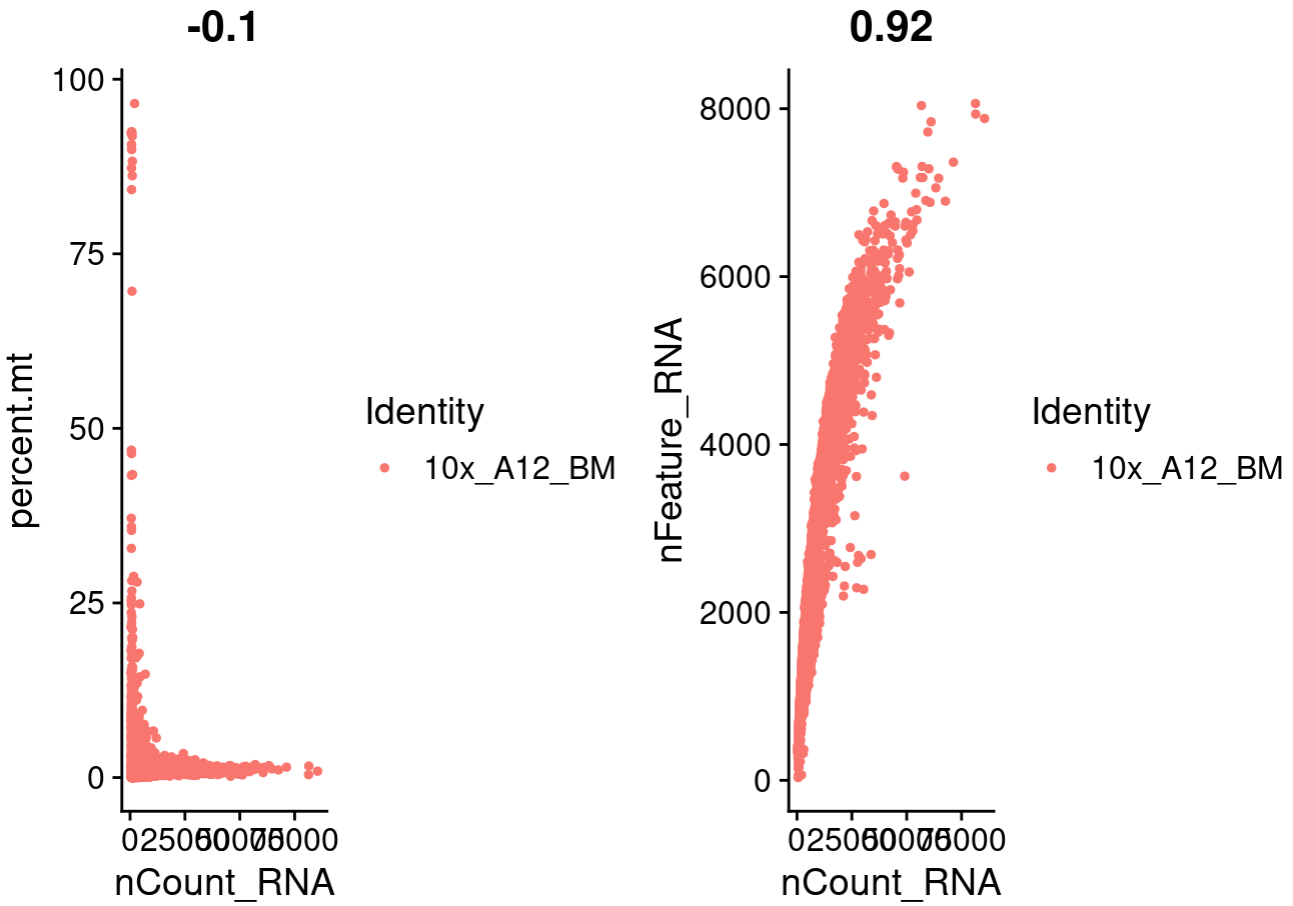
```
## Warning: Default search for "data" layer in "RNA" assay yielded no results;
## utilizing "counts" layer instead.
```




```
mean(Seurat_Object_BM$nFeature_RNA)
```

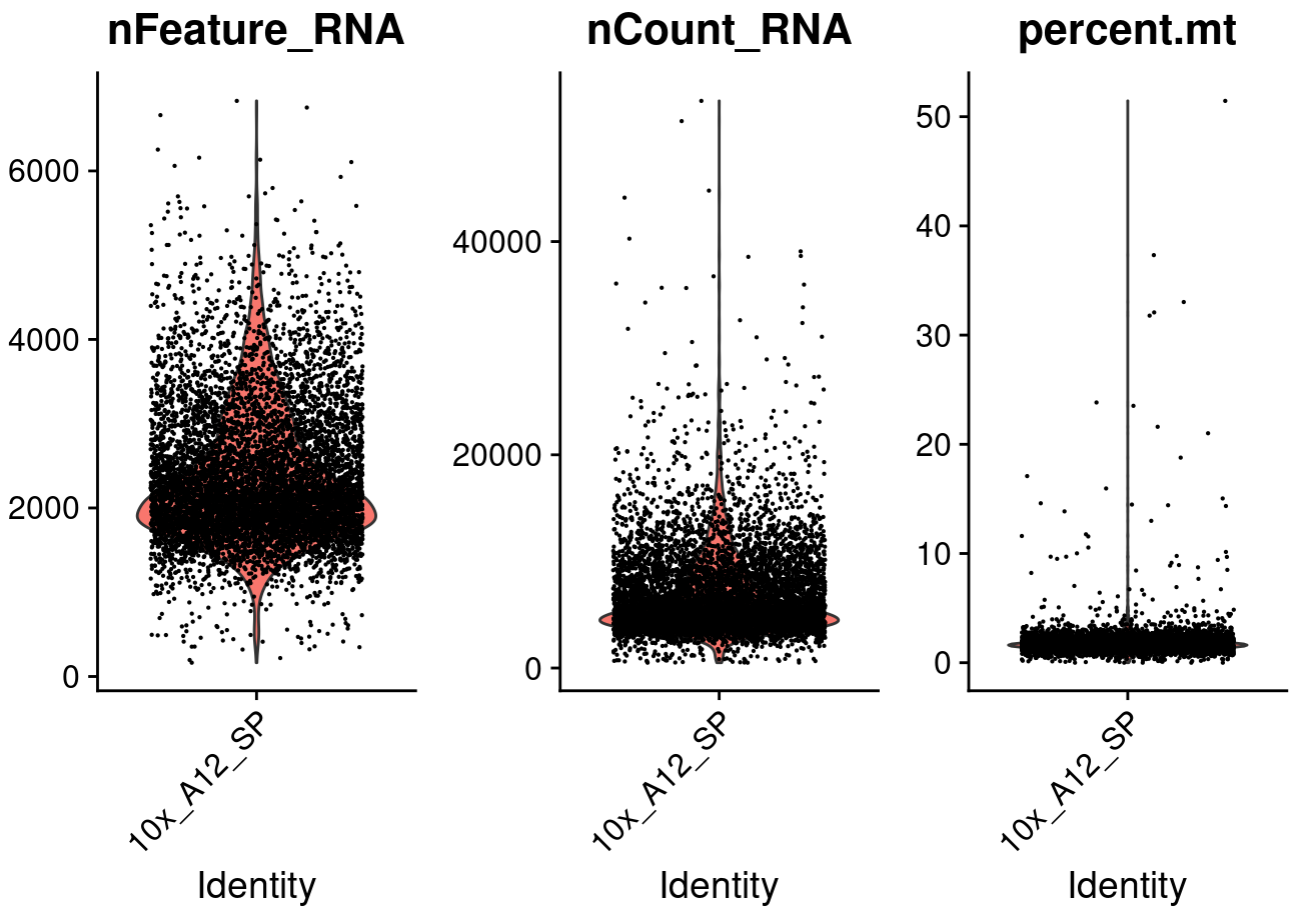
```
## [1] 2157.838
```

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




```
# violin plot
Idents(Seurat_Object_SP) <- Seurat_Object_SP$orig.ident
VlnPlot(Seurat_Object_SP, features = c("nFeature_RNA", "nCount_RNA", "percent.mt"), ncol = 3)
```

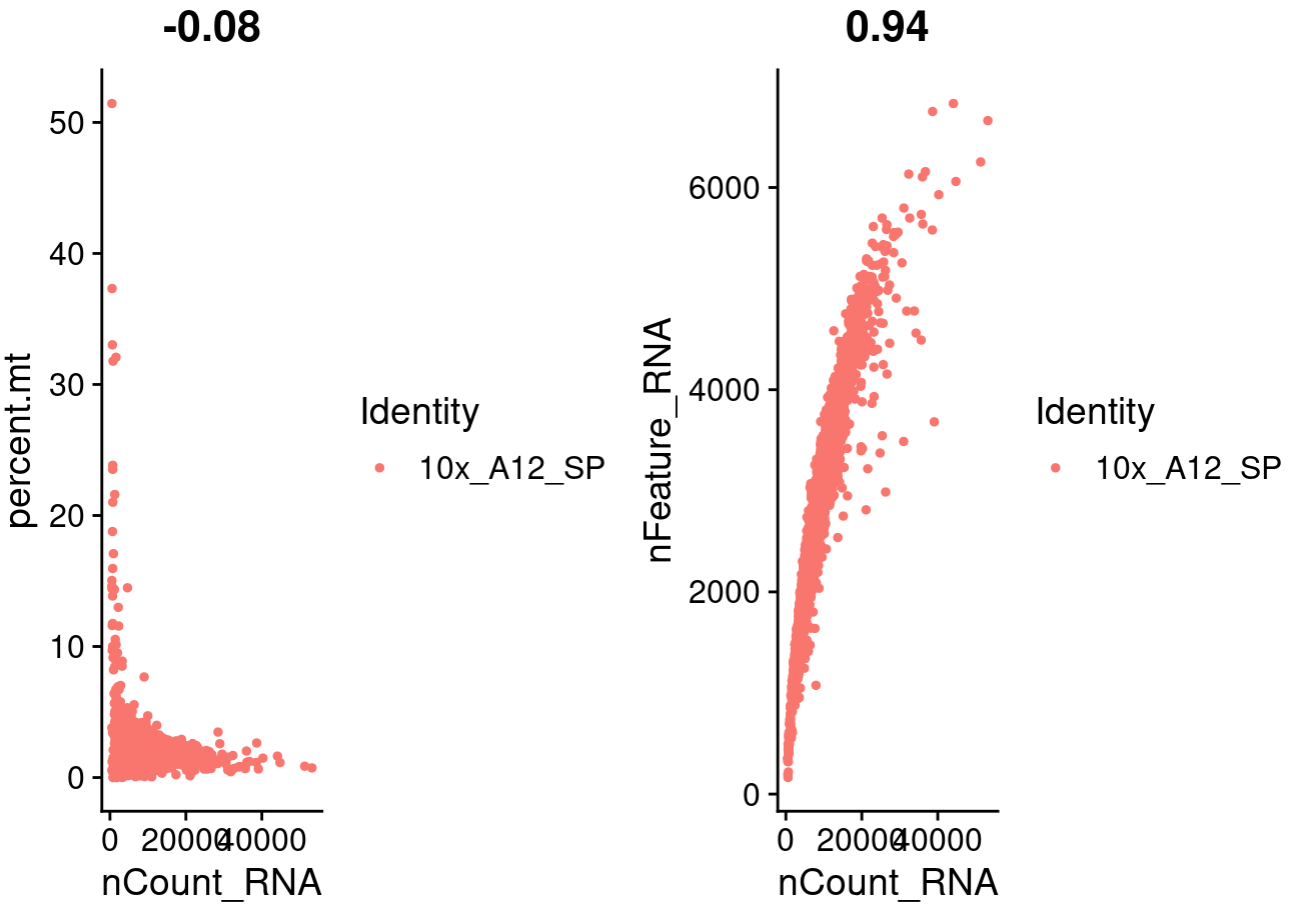
```
## Warning: Default search for "data" layer in "RNA" assay yielded no results;
## utilizing "counts" layer instead.
```




```
mean(Seurat_Object_SP$nFeature_RNA)
```

```
## [1] 2355.735
```

```
# 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.

plot3 <- FeatureScatter(Seurat_Object_SP, feature1 = "nCount_RNA", feature2 = "percent.mt")
plot4 <- FeatureScatter(Seurat_Object_SP, feature1 = "nCount_RNA", feature2 = "nFeature_RNA")
plot3 + plot4
```

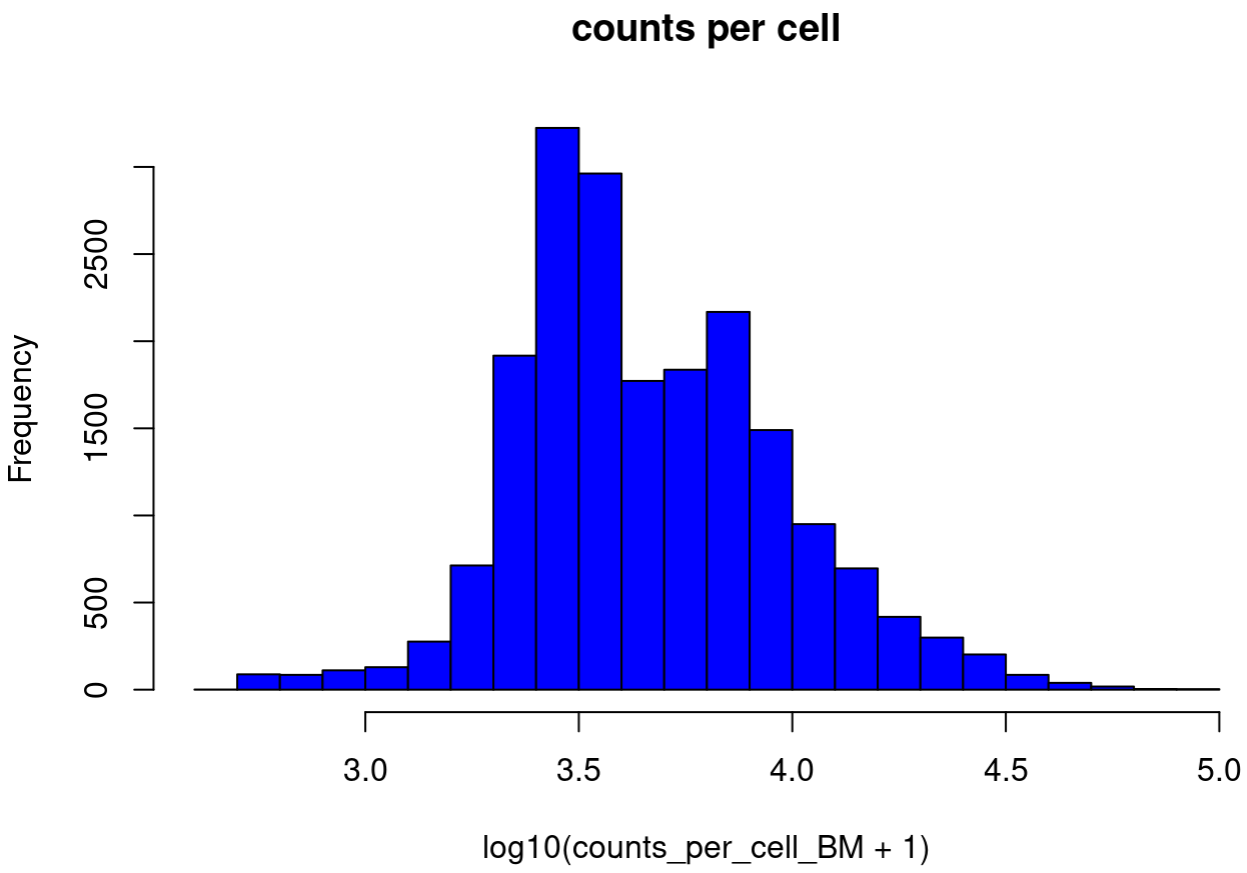


Other representations useful for quality control:

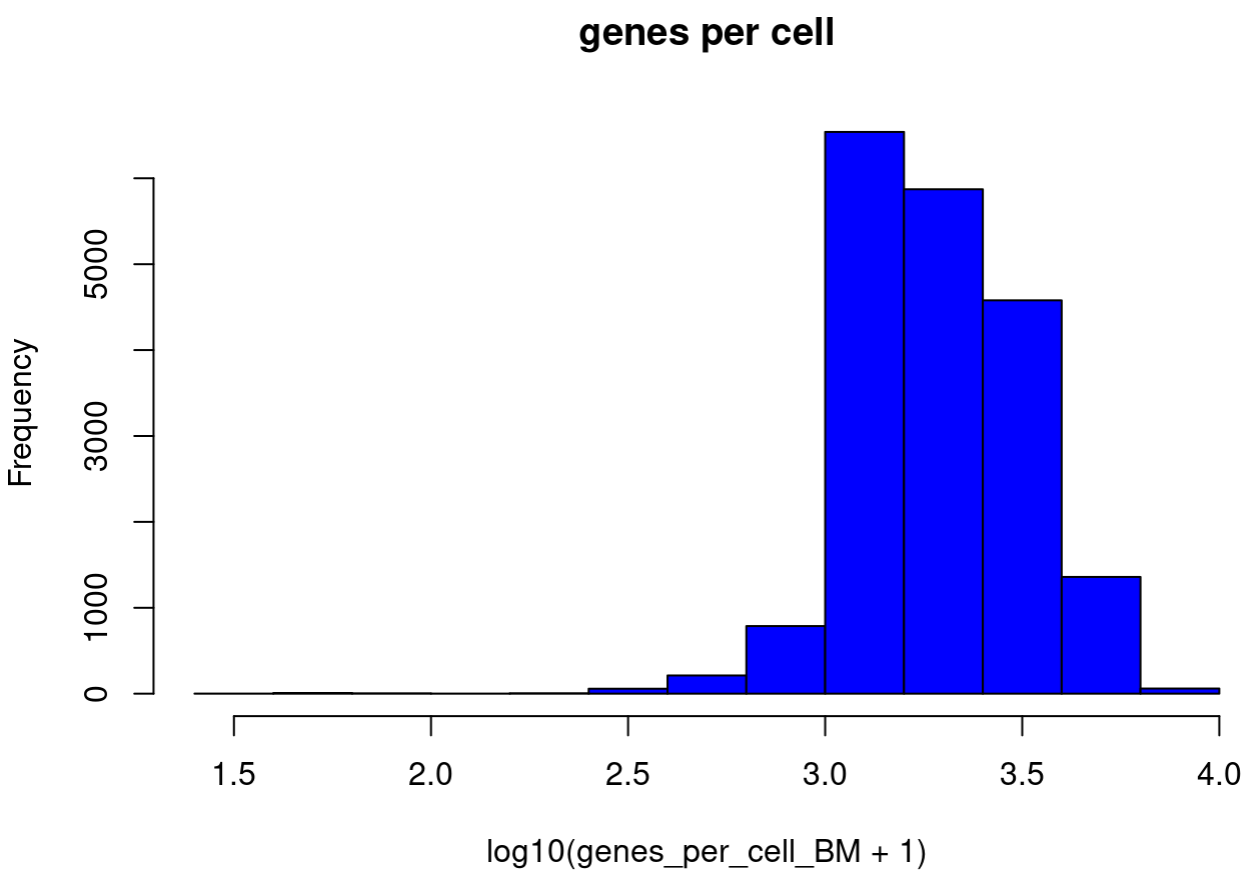
```
counts_BM <- data_BM$`Gene Expression`
dim(counts)
```

```
## NULL
```

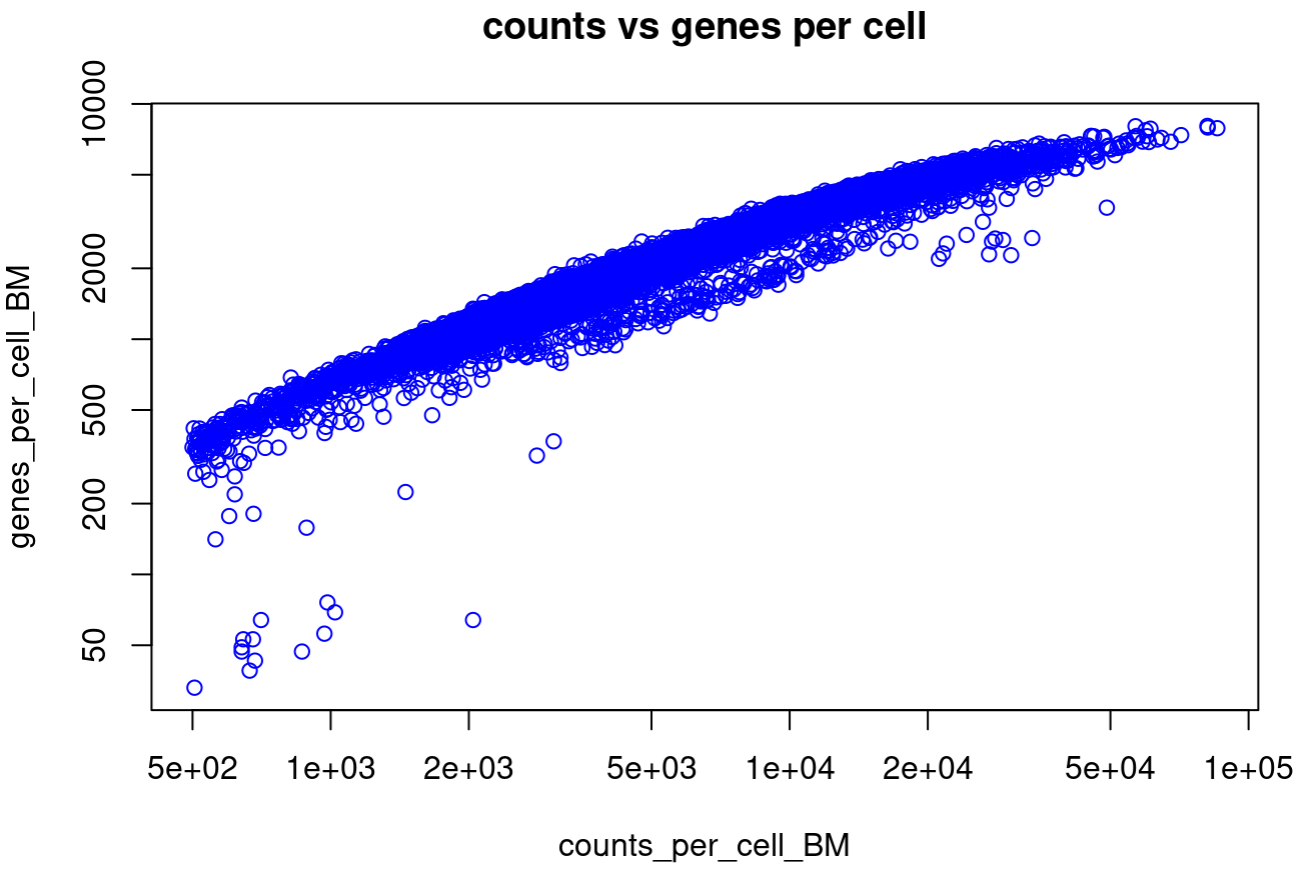
```
counts_per_cell_BM <- Matrix::colSums(counts_BM)
counts_per_gene_BM <- Matrix::rowSums(counts_BM)
genes_per_cell_BM <- Matrix::colSums(counts_BM > 0) # count a gene only if it has non-zero rea
ds mapped.
cells_per_gene_BM <- Matrix::rowSums(counts_BM > 0) # only count cells where the gene is expre
ssed

hist(log10(counts_per_cell_BM + 1),main='counts per cell',col='blue')
```



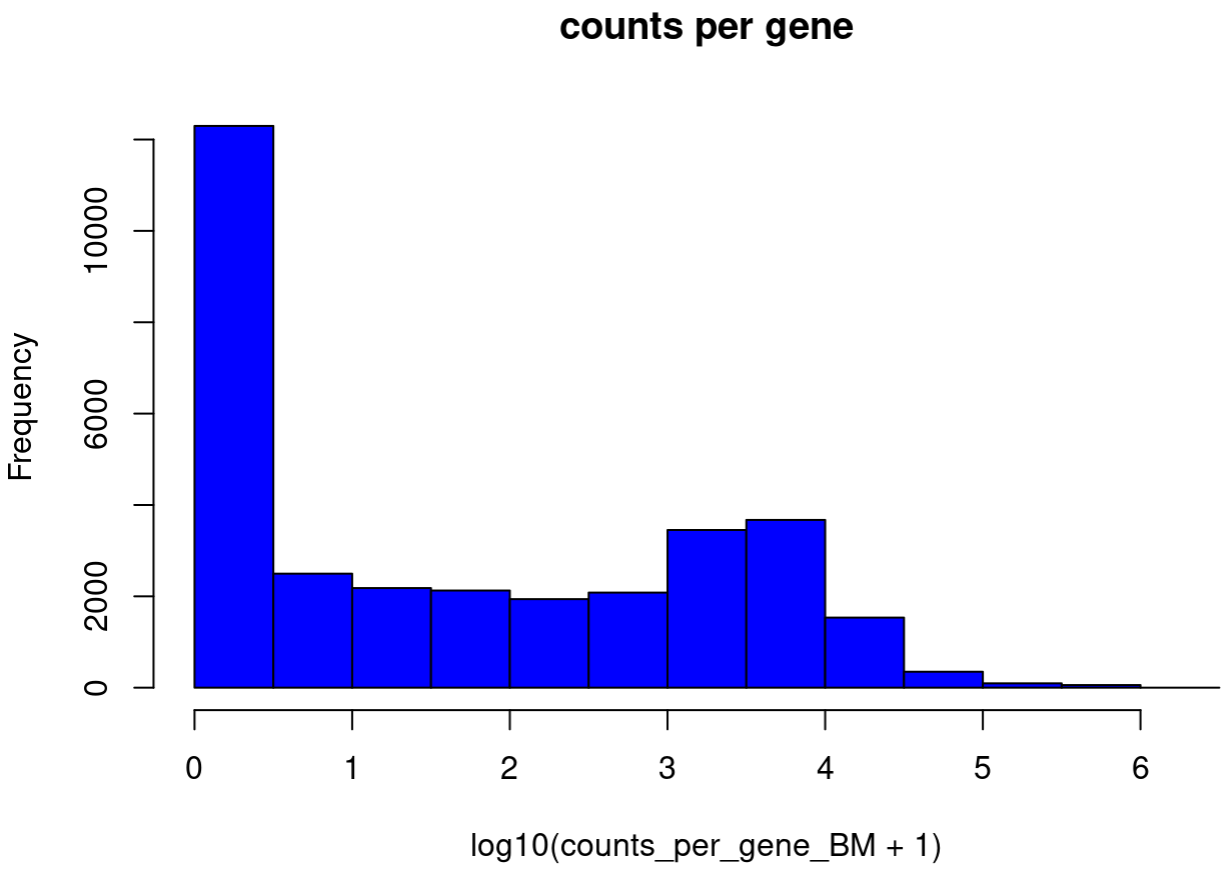

```
hist(log10(genes_per_cell_BM + 1), main='genes per cell', col='blue')
```




```
plot(counts_per_cell_BM, genes_per_cell_BM, log='xy', col='blue')
title('counts vs genes per cell')
```



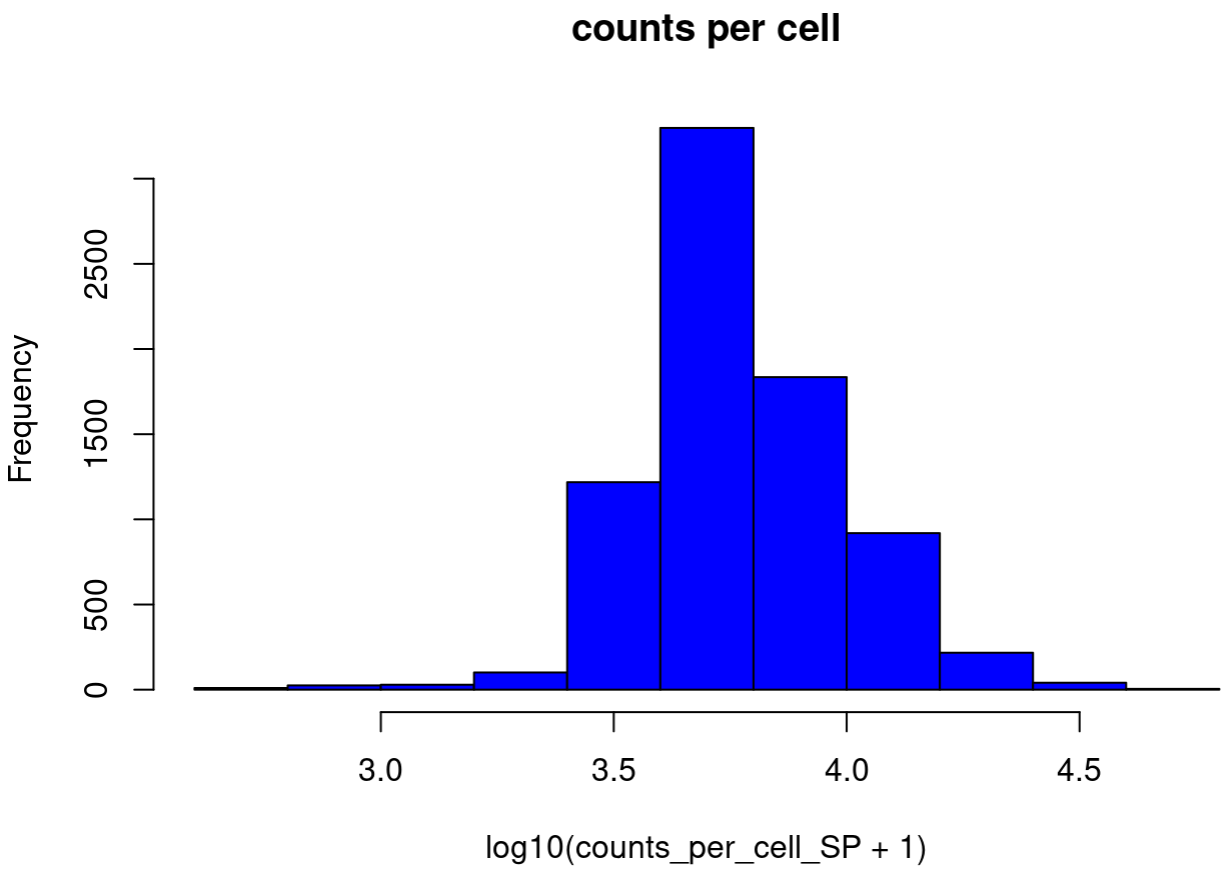

```
hist(log10(counts_per_gene_BM + 1),main='counts per gene',col='blue')
```



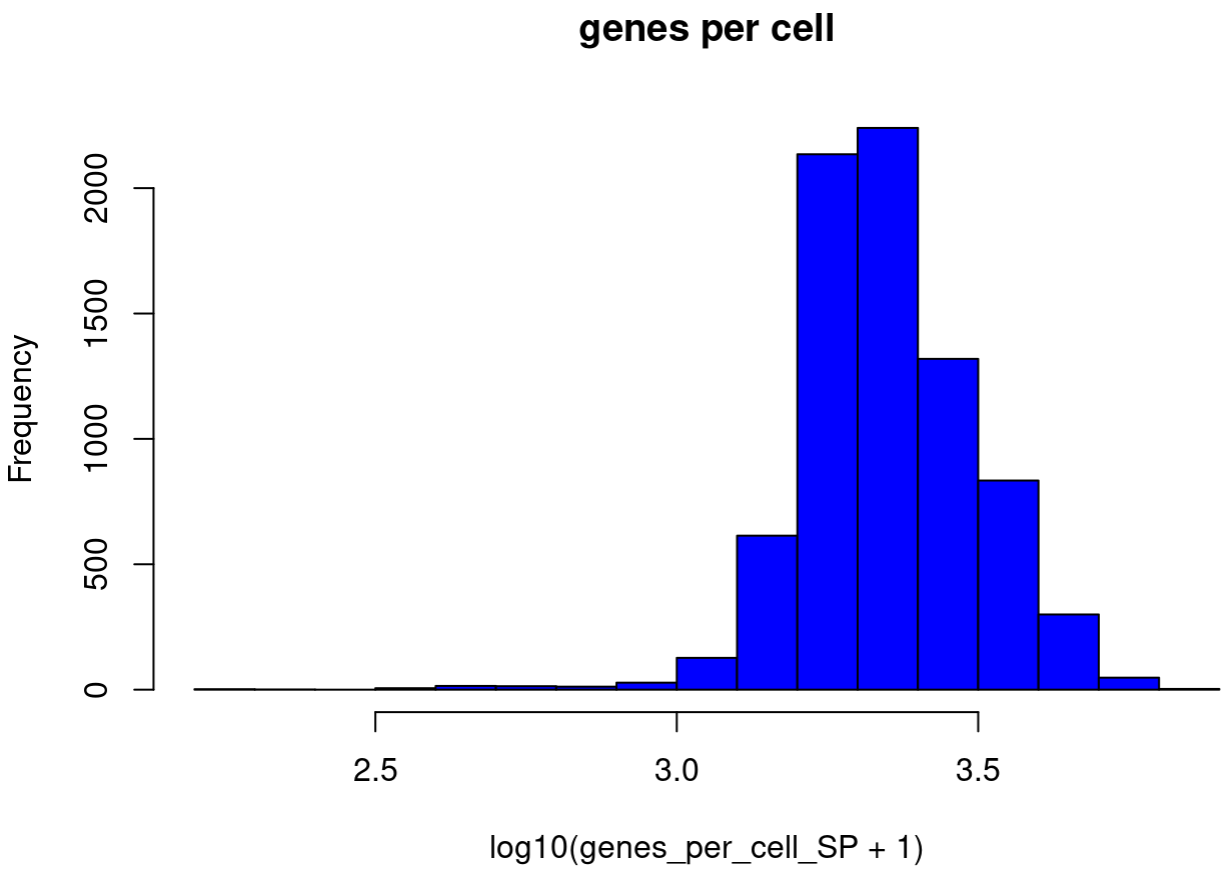

```
counts_SP<- data_SP$`Gene Expression`
dim(counts_SP)
```

```
## [1] 32286  7698
```

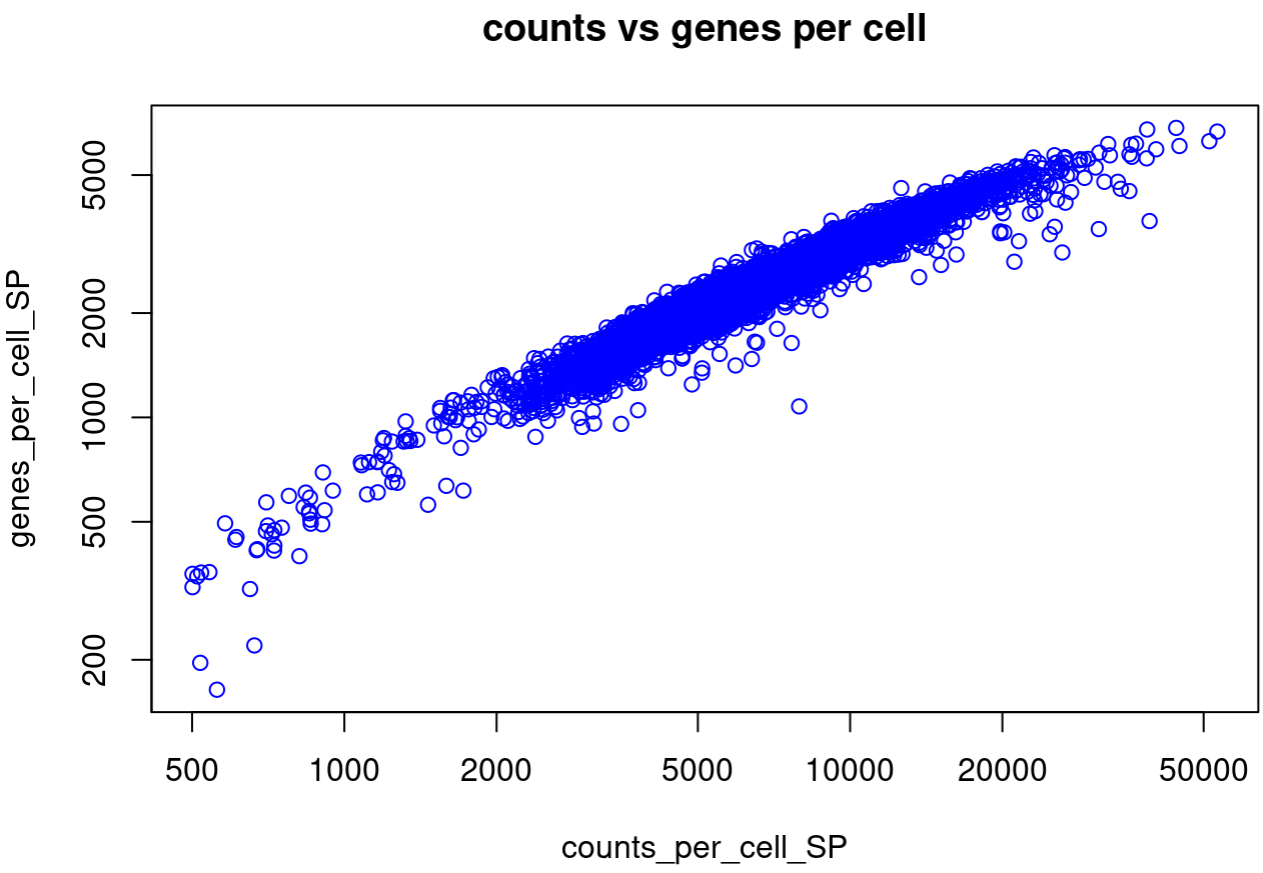
```
counts_per_cell_SP <- Matrix::colSums(counts_SP)
counts_per_gene_SP <- Matrix::rowSums(counts_SP)
genes_per_cell_SP <- Matrix::colSums(counts_SP>0) # count a gene only if it has non-zero reads
 mapped.
cells_per_gene_SP <- Matrix::rowSums(counts_SP>0) # only count cells where the gene is express
ed

hist(log10(counts_per_cell_SP+1),main='counts per cell',col='blue')
```



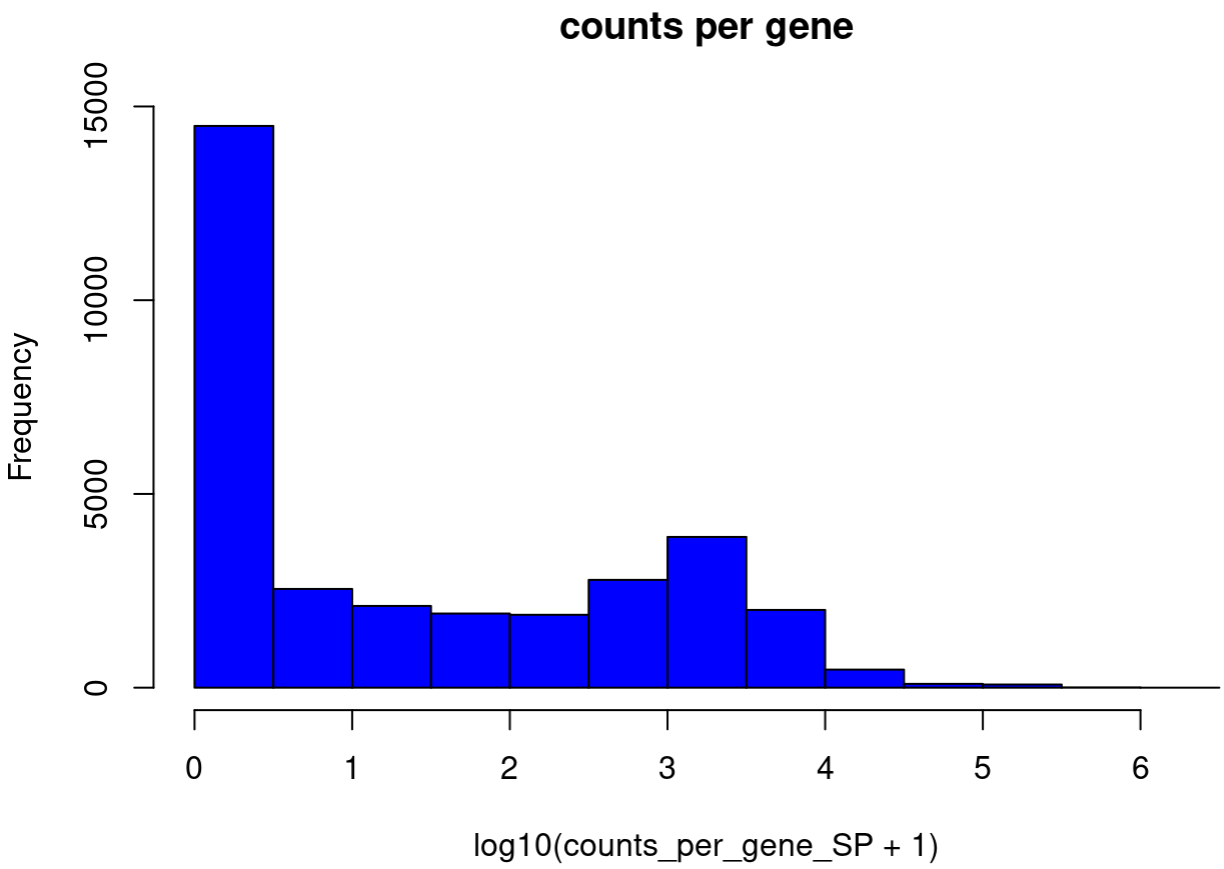

```
hist(log10(genes_per_cell_SP+1), main='genes per cell', col='blue')
```




```
plot(counts_per_cell_SP, genes_per_cell_SP, log='xy', col='blue')
title('counts vs genes per cell')
```

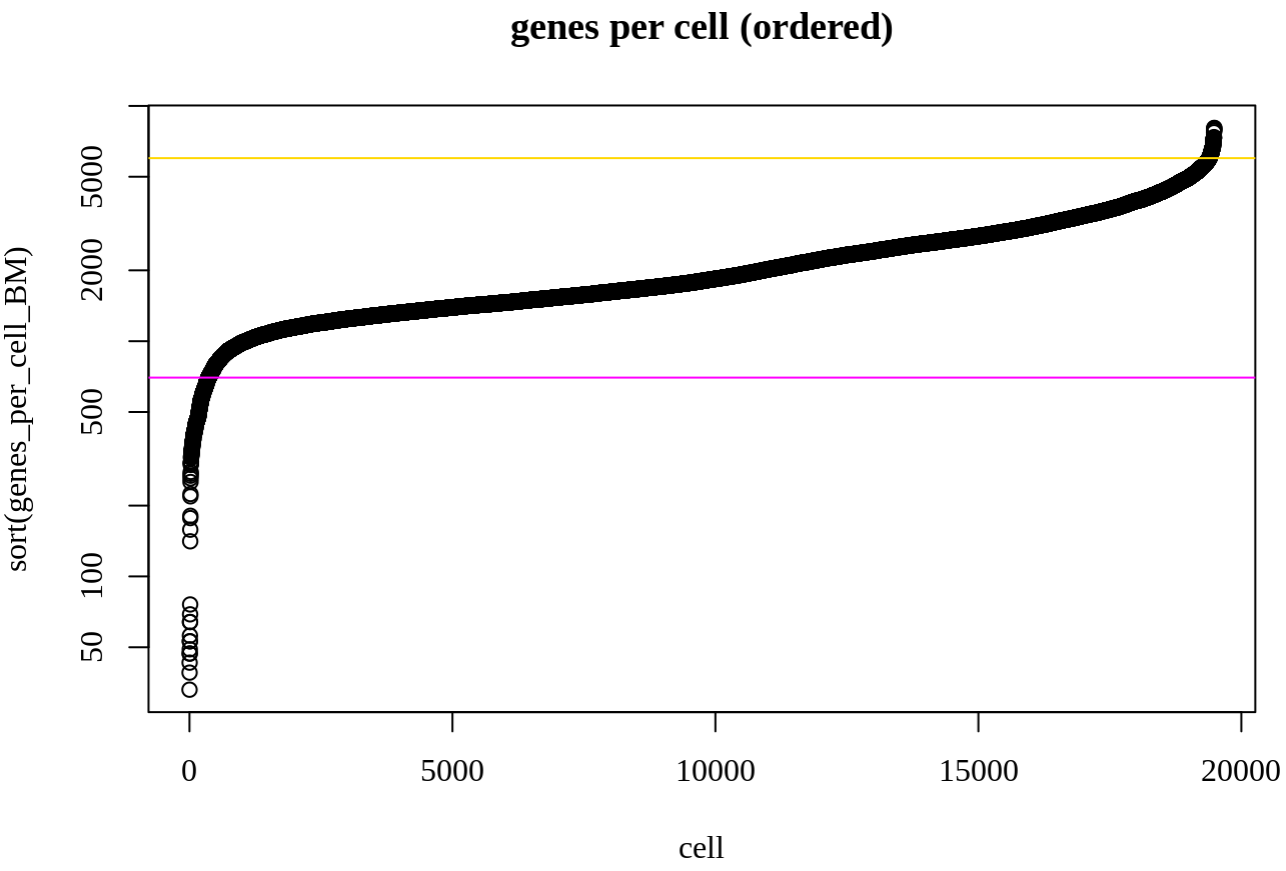



```
hist(log10(counts_per_gene_SP+1),main='counts per gene',col='blue')
```

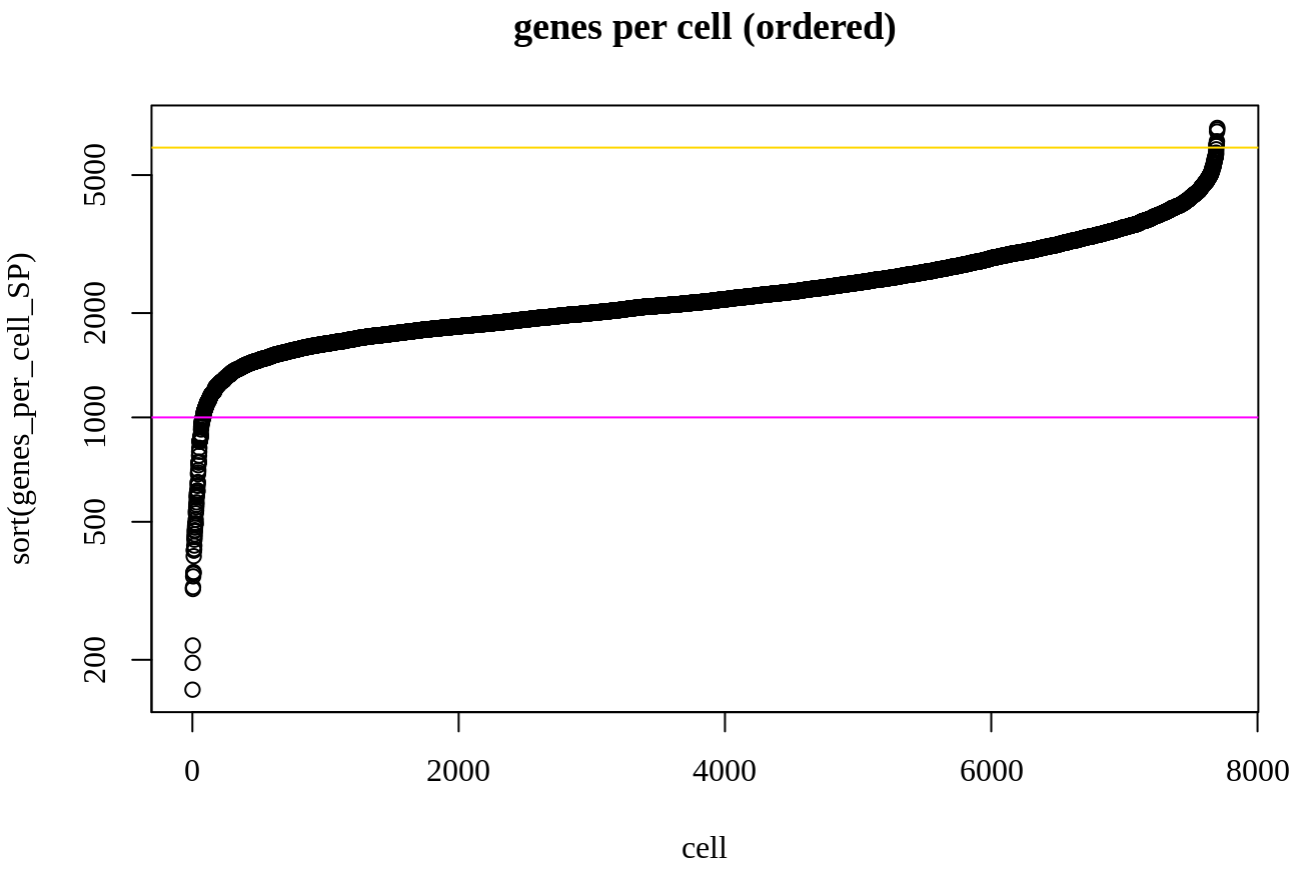


To establish the min and max genes per cells to avoid outliers: In our case we do not want to be very restrictive with max gene number since there are plasma cells with a lot of immunoglobuline transcripts.

```
# Set upper and lower thresholds for genes per cell:
MIN_GENES_PER_CELL_BM <- 700  ## user-defined setting
MAX_GENES_PER_CELL_BM <- 6000  ## user-defined setting

# now replot with the thresholds being shown:
plot(sort(genes_per_cell_BM), xlab='cell', log='y', main='genes per cell (ordered)',
  family = "Times New Roman")
abline(h=MIN_GENES_PER_CELL_BM, col='magenta')  # lower threshold
abline(h=MAX_GENES_PER_CELL_BM, col='gold') # upper threshold
```




```
# Parameters
MIN_GENES_PER_CELL_SP <- 1000
MAX_GENES_PER_CELL_SP <- 6000

plot(
  sort(genes_per_cell_SP),
  xlab = 'cell',
  log = 'y',
  main = 'genes per cell (ordered)',
  family = "Times New Roman"
)
abline(h = MIN_GENES_PER_CELL_SP, col = 'magenta')   # lower threshold
abline(h = MAX_GENES_PER_CELL_SP, col = 'gold')      # upper threshold
```

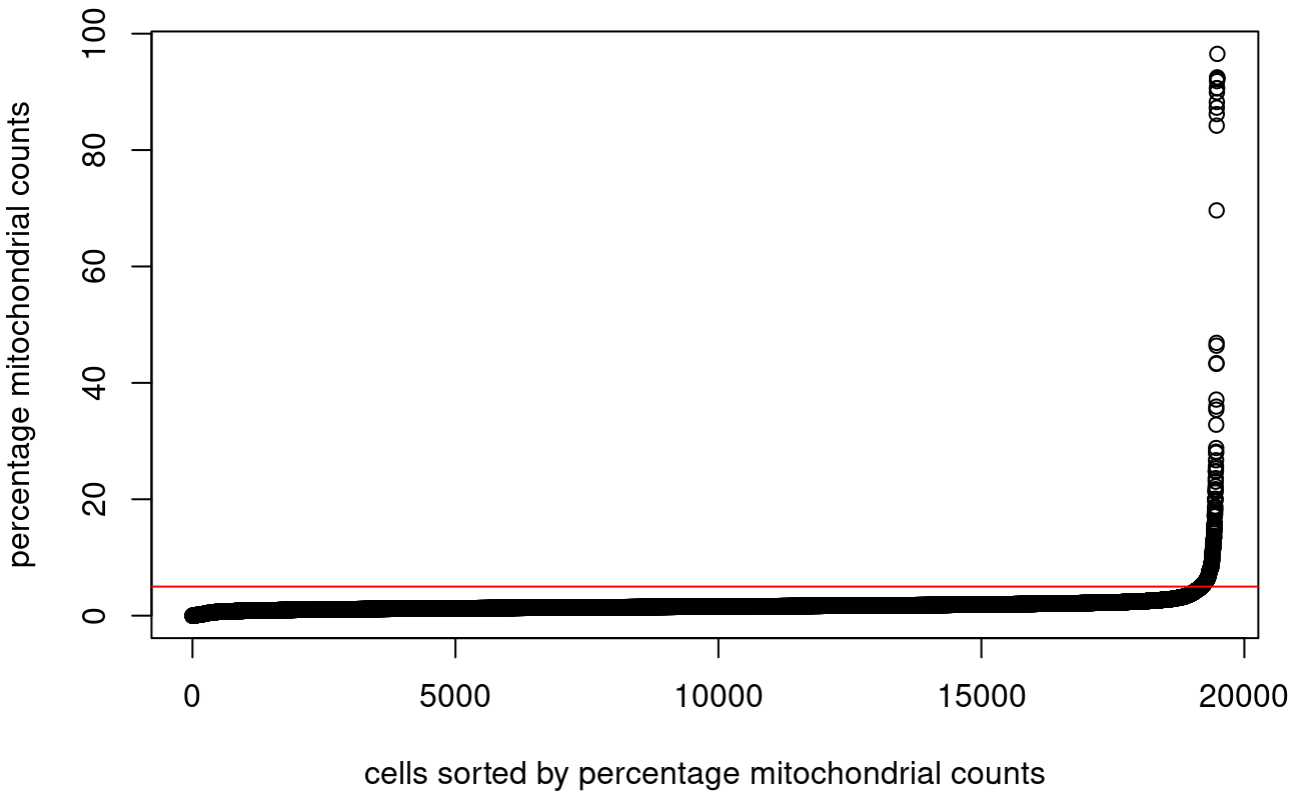


To establish the threshold for mithocondrial genes:

```
# define the mitochondrial genes
mito_genes_BM <- grep("^mt-", rownames(counts_BM) , ignore.case=T, value=T)
print(mito_genes_BM)
```

```
##  [1] "mt-Nd1"  "mt-Nd2"  "mt-Co1"  "mt-Co2"  "mt-Atp8" "mt-Atp6" "mt-Co3"
##  [8] "mt-Nd3"  "mt-Nd4l" "mt-Nd4"  "mt-Nd5"  "mt-Nd6"  "mt-Cytb"
```

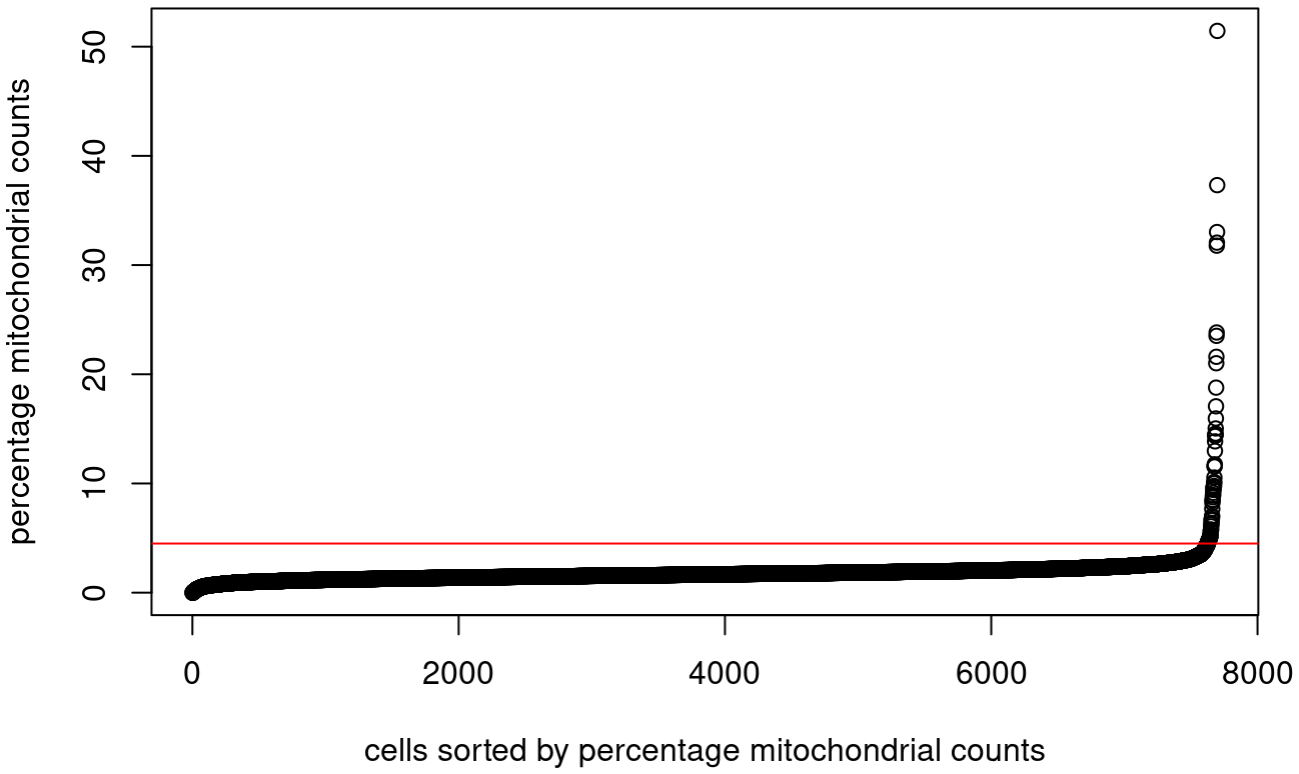
```
# compute pct mito
mito_gene_read_counts_BM = Matrix::colSums(counts_BM[mito_genes_BM,])
pct_mito_BM = mito_gene_read_counts_BM / counts_per_cell_BM * 100

MAX_PCT_MITO_BM <- 5   ## user-defined setting

plot(sort(pct_mito_BM), xlab = "cells sorted by percentage mitochondrial counts", ylab = "perc
entage mitochondrial counts")
abline(h=MAX_PCT_MITO_BM, col='red')
```




```
# define the mitochondrial genes
mito_genes_SP <- grep("^mt-", rownames(counts_SP) , ignore.case=T, value=T)
print(mito_genes_SP)
```

```
##  [1] "mt-Nd1"  "mt-Nd2"  "mt-Co1"  "mt-Co2"  "mt-Atp8" "mt-Atp6" "mt-Co3"
##  [8] "mt-Nd3"  "mt-Nd4l" "mt-Nd4"  "mt-Nd5"  "mt-Nd6"  "mt-Cytb"
```

```
# compute pct mito
mito_gene_read_counts_SP = Matrix::colSums(counts_SP[mito_genes_SP,])
pct_mito_SP = mito_gene_read_counts_SP / counts_per_cell_SP * 100

MAX_PCT_MITO_SP <- 4.5   ## user-defined setting

plot(sort(pct_mito_SP), xlab = "cells sorted by percentage mitochondrial counts", ylab = "perc
entage mitochondrial counts")
abline(h=MAX_PCT_MITO_SP, col='red')
```



Filtering based on QC metrics

Filtering is a subjective step which totally depends on the type of data one has. Based on the previous plots we have established the following filters: We filter cells that have unique feature counts over 6000 or less than 500 We filter cells that have >5% mitochondrial counts

```
Seurat_Object_BM <- subset(Seurat_Object_BM, subset = nFeature_RNA > 700 & nFeature_RNA < 6000
 & percent.mt < 5)
```

```
Seurat_Object_SP <- subset(Seurat_Object_SP, subset = nFeature_RNA > 1000 & nFeature_RNA < 600
0 & percent.mt < 4.5)
```

Normalizing the data

By default, we employ a global-scaling normalization method “LogNormalize” that normalizes the feature expression measurements for each cell by the total expression, multiplies this by a scale factor (10,000 by default), and log-transforms the result. Normalized values are stored in pbmc[[“RNA”]]@data.

LOGNORMALIZE Method Steps:

- Library Size Factor Calculation: For each cell, the total expression (sum of counts) is calculated. This total is used as a library size factor.
- Scaling: Each gene count in a cell is divided by the total counts (library size) of that cell. This step converts the

counts into relative proportions (or fractions) of the total counts for each cell.

- Scaling to a Common Value: These proportions are then multiplied by a scale factor (often set to 10,000) to bring them to a common scale. This helps in making the data more interpretable and comparable across cells.
- Log Transformation: Finally, a logarithmic transformation is applied, typically log1p, which is log(1 + x). This transformation helps in compressing the range of the data and stabilizing the variance.

On the other hand, HTO or ADT data is normalized with the “CLR” method:

- Very useful for Compositional Data Constant Sum: The sum of all parts (e.g., all gene counts in a cell) is constant. This sum can vary between samples, but within a sample, the sum of all parts is fixed. Relative Proportions: The important aspect is not the absolute value of each part, but its relative proportion compared to the total. Interdependence of Parts: Since the sum of all parts is constant, an increase in one part must be compensated by a decrease in another part(s).

We normalize both GEX and HTO data

GEX is normalized by “LogNormalize” (default)

```
Seurat_Object_BM <- NormalizeData(Seurat_Object_BM)
```

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

```
Seurat_Object_BM <- NormalizeData(Seurat_Object_BM, assay = "HTO", normalization.method = "CLR
")
```

```
## Normalizing across features
```

```
Seurat_Object_SP <- NormalizeData(Seurat_Object_SP)
```

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

```
Seurat_Object_SP <- NormalizeData(Seurat_Object_SP, assay = "HTO", normalization.method = "CLR
")
```

```
## Normalizing across features
```

Demultiplex data based on HTO enrichment

Demultiplex cells and identify cross-sample doublets. Observe the number of singlets, doublets and negative cells https://satijalab.org/seurat/articles/multimodal_vignette

The number written in HTODemux (0.99) has to be carefully selected. If this number is very high, only cells with very high reads of HTO will be considered as positive. This increases doublets (as it is difficult to be considered a positive cell, cells with WT-3 high and WT-4 very-high for example, will only be considered as WT4). Still, we need to find an equilibrium being restrictive but not in excess.

I have chosen to set this threshold to 0.9999 in spleen as the distribution of HTO counts is more clear and we need to be

restrictive in taking cells that are real hulgk+. In bone marrow, we have been less restrictive due to the fact that hto counts for some mice were low in some cases.

```
Seurat_Object_BM <- HTODemux(Seurat_Object_BM, assay = "HTO", positive.quantile = 0.99)
```

```
## As of Seurat v5, we recommend using AggregateExpression to perform pseudo-bulk analysis.
## First group.by variable `ident` starts with a number, appending `g` to ensure valid variabl
e names
## Cutoff for BM-WT-1 : 29 reads
##
## Cutoff for BM-A12-1 : 43 reads
##
## Cutoff for BM-WT-2 : 25 reads
##
## Cutoff for BM-WT-3 : 161 reads
##
## Cutoff for BM-A12-2 : 195 reads
##
## Cutoff for BM-A12-3 : 29 reads
##
## Cutoff for huK : 23 reads
##
## This message is displayed once per session.
```

```
Seurat_Object_SP <- HTODemux(Seurat_Object_SP, assay = "HTO", positive.quantile = 0.99999)
```

```
## Cutoff for SP-WT-1 : 61 reads
```

```
## Cutoff for SP-A12-1 : 135 reads
```

```
## Cutoff for SP-WT-2 : 452 reads
```

```
## Cutoff for SP-WT-3 : 891 reads
```

```
## Cutoff for SP-A12-2 : 129 reads
```

```
## Cutoff for SP-A12-3 : 122 reads
```

```
## Cutoff for huK : 36 reads
```

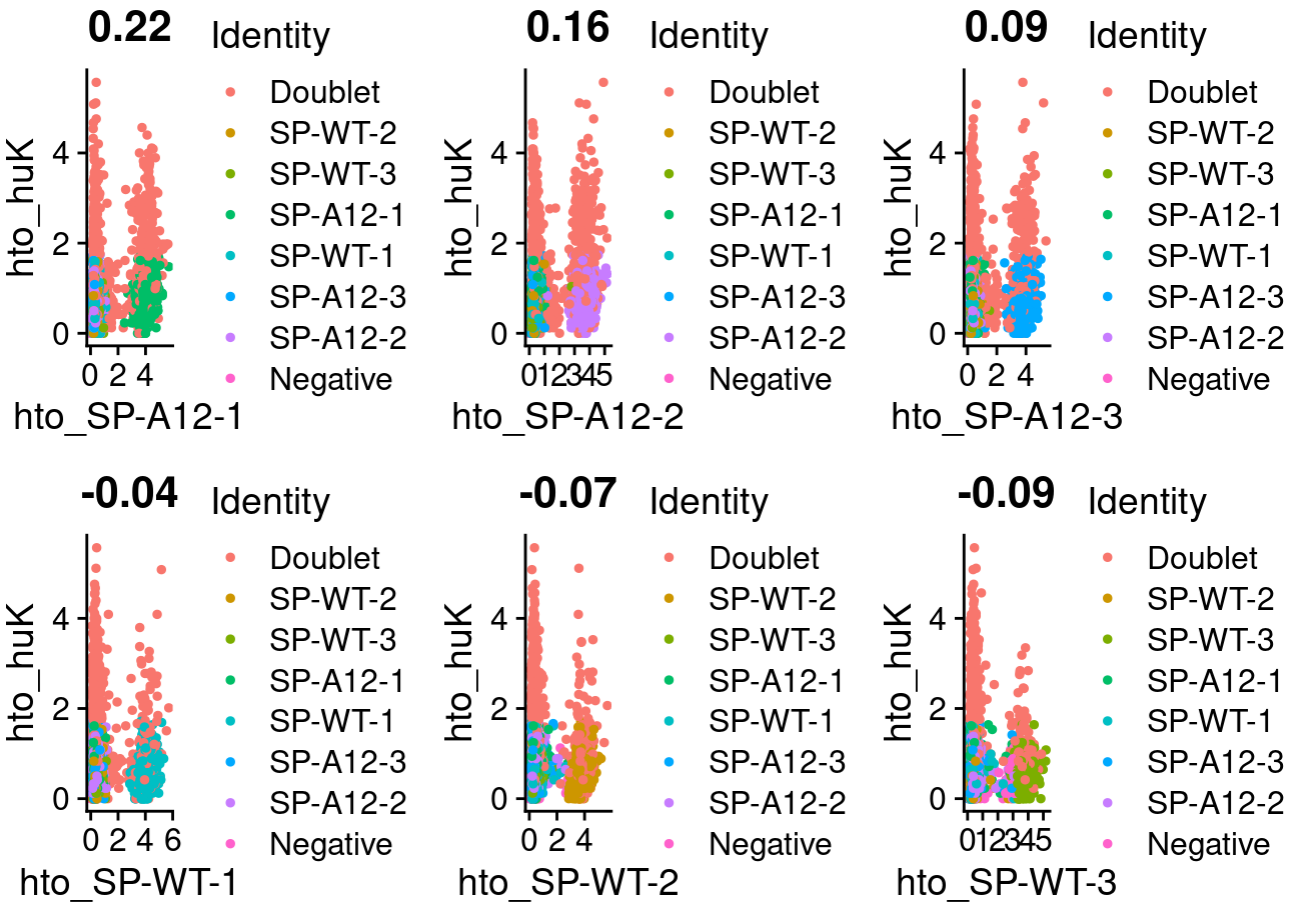
```
# Global classification results
table_result <- table(Seurat_Object_SP$HTO_classification)
table_result_sorted <- sort(table_result, decreasing = FALSE)
table_result_sorted
```

```
##
```

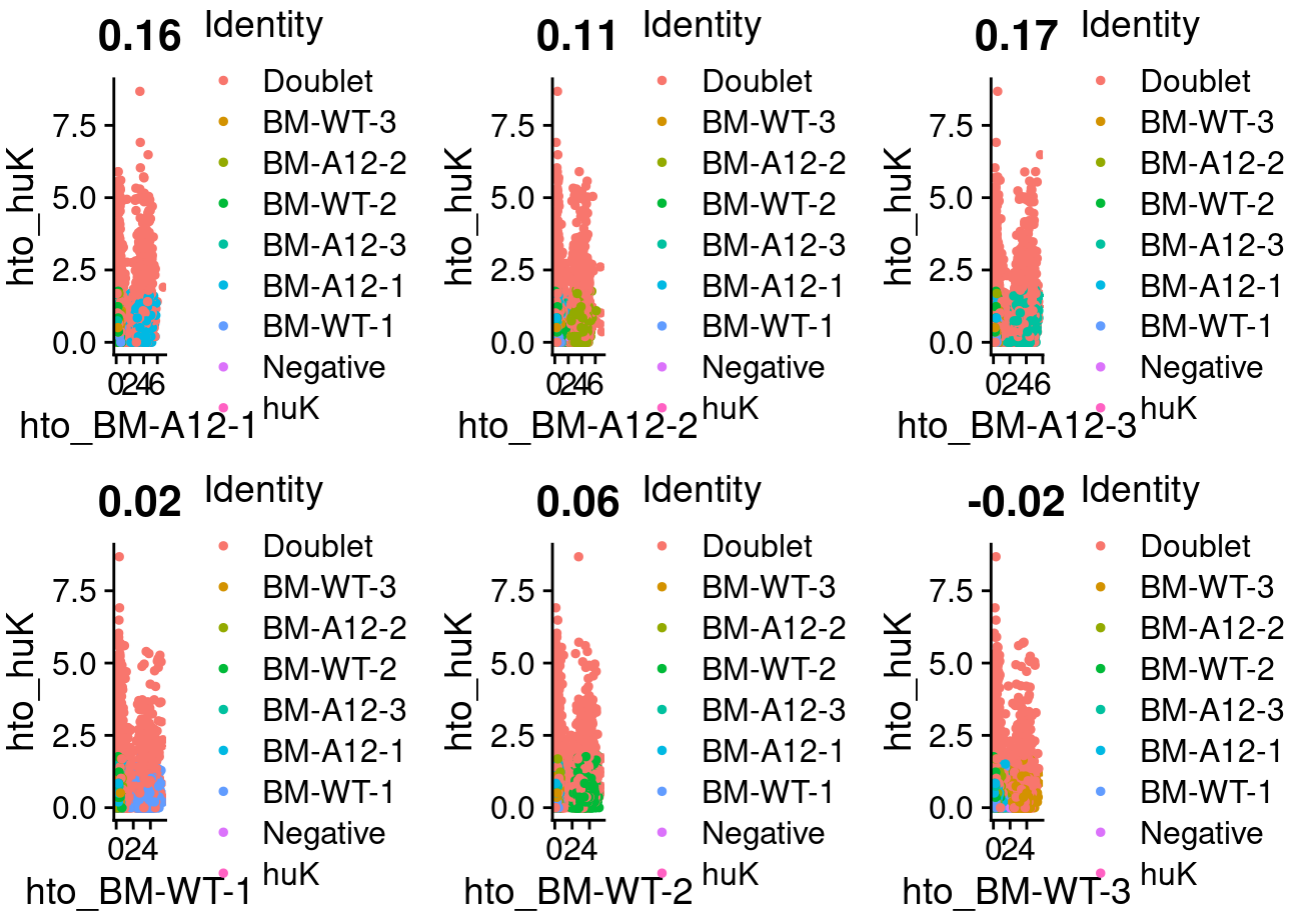
```
##       huK_SP-WT-3       huK_SP-WT-2       huK_SP-WT-1   SP-WT-2_SP-WT-3
##                 2                 6                19                47
##      huK_SP-A12-3  SP-A12-1_SP-WT-3  SP-A12-3_SP-WT-3  SP-A12-3_SP-WT-2
##                53                63                66                67
##  SP-A12-2_SP-WT-3  SP-A12-2_SP-WT-2  SP-A12-1_SP-WT-2  SP-A12-2_SP-WT-1
##                69                71                75                78
##   SP-WT-1_SP-WT-2   SP-WT-1_SP-WT-3      huK_SP-A12-2 SP-A12-1_SP-A12-3
##                81                82                85                89
##  SP-A12-1_SP-WT-1 SP-A12-2_SP-A12-3 SP-A12-1_SP-A12-2      huK_SP-A12-1
##                89                96                99               101
##  SP-A12-3_SP-WT-1          Negative           SP-WT-3           SP-WT-1
##               105               127               938               973
##          SP-A12-1           SP-WT-2          SP-A12-3          SP-A12-2
##               975              1025              1028              1069
```

Verify that the positive quantile is selected properly

```
plot1 <- FeatureScatter(Seurat_Object_SP, feature1 = "hto_SP-A12-1", feature2 = "hto_huK", pt.
size = 1)
plot2 <- FeatureScatter(Seurat_Object_SP, feature1 = "hto_SP-A12-2", feature2 = "hto_huK", pt.
size = 1)
plot3 <- FeatureScatter(Seurat_Object_SP, feature1 = "hto_SP-A12-3", feature2 = "hto_huK", pt.
size = 1)
plot4 <- FeatureScatter(Seurat_Object_SP, feature1 = "hto_SP-WT-1", feature2 = "hto_huK", pt.s
ize = 1)
plot5 <- FeatureScatter(Seurat_Object_SP, feature1 = "hto_SP-WT-2", feature2 = "hto_huK", pt.s
ize = 1)
plot6 <- FeatureScatter(Seurat_Object_SP, feature1 = "hto_SP-WT-3", feature2 = "hto_huK", pt.s
ize = 1)
(plot1 + plot2 + plot3 + plot4 + plot5 + plot6)
```




```
plot1 <- FeatureScatter(Seurat_Object_BM, feature1 = "hto_BM-A12-1", feature2 = "hto_huK", pt.
size = 1)
plot2 <- FeatureScatter(Seurat_Object_BM, feature1 = "hto_BM-A12-2", feature2 = "hto_huK", pt.
size = 1)
plot3 <- FeatureScatter(Seurat_Object_BM, feature1 = "hto_BM-A12-3", feature2 = "hto_huK", pt.
size = 1)
plot4 <- FeatureScatter(Seurat_Object_BM, feature1 = "hto_BM-WT-1", feature2 = "hto_huK", pt.s
ize = 1)
plot5 <- FeatureScatter(Seurat_Object_BM, feature1 = "hto_BM-WT-2", feature2 = "hto_huK", pt.s
ize = 1)
plot6 <- FeatureScatter(Seurat_Object_BM, feature1 = "hto_BM-WT-3", feature2 = "hto_huK", pt.s
ize = 1)
(plot1 + plot2 + plot3 + plot4 + plot5 + plot6)
```



Let's convert our Seurat object to single cell experiment (SCE) for convenience.

```
sce_SP <- as.SingleCellExperiment(DietSeurat(Seurat_Object_SP))
```

```
## Warning: Layer 'scale.data' is empty
```

```
sce_SP
```

```
## class: SingleCellExperiment
## dim: 17662 7578
## metadata(0):
## assays(2): counts logcounts
## rownames(17662): Xkr4 Mrpl15 ... AC149090.1 A12
## rowData names(0):
## colnames(7578): AAACCTGAGAAGGGTA-1 AAACCTGAGACGACGT-1 ...
##   TTTGTCATCTGCTTGC-1 TTTGTCATCTTGCCGT-1
## colData names(13): orig.ident nCount_RNA ... hash.ID ident
## reducedDimNames(0):
## mainExpName: RNA
## altExpNames(1): HTO
```

```
sce_BM <- as.SingleCellExperiment(DietSeurat(Seurat_Object_BM))
```

```
## Warning: Layer 'scale.data' is empty
```

```
sce_BM
```

```
## class: SingleCellExperiment
## dim: 19884 18874
## metadata(0):
## assays(2): counts logcounts
## rownames(19884): Xkr4 Rp1 ... AC149090.1 A12
## rowData names(0):
## colnames(18874): AAACCTGAGCATCATC-1 AAACCTGAGCCCAATT-1 ...
##   TTTGTCATCGCAGGCT-1 TTTGTCATCGCCAAAT-1
## colData names(13): orig.ident nCount_RNA ... hash.ID ident
## reducedDimNames(0):
## mainExpName: RNA
## altExpNames(1): HTO
```

scDblFinder

Doublet depletion scDblFinder has two main modes for generating artificial doublets: a random one (clusters=FALSE, now default) and a cluster-based one (clusters=TRUE or providing your own clusters - the approach from previous versions). In practice, we observed that both approaches perform well (and better than alternatives). We suggest using the cluster-based approach when the datasets are segregated into clear clusters, and the random one for the rest (e.g. developmental trajectories).

```
sce_BM <- scDblFinder(sce_BM)
```

```
## Creating ~15100 artificial doublets...
```

```
## Dimensional reduction
```

```
## Evaluating kNN...
```

```
## Training model...
```

```
## iter=0, 4178 cells excluded from training.
```

```
## iter=1, 4234 cells excluded from training.
```

```
## iter=2, 4209 cells excluded from training.
```

```
## Threshold found:0.487
```

```
## 3663 (19.4%) doublets called
```

```
table(sce_BM$scDblFinder.class)
```

```
##
## singlet doublet
##   15211    3663
```

```
doublets_scDblFinder_BM <- colnames(sce_BM[, sce_BM$scDblFinder.class == "doublet"])

sce_SP <- scDblFinder(sce_SP)
```

```
## Creating ~6063 artificial doublets...
```

```
## Dimensional reduction
```

```
## Evaluating kNN...
```

```
## Training model...
```

```
## iter=0, 974 cells excluded from training.
```

```
## iter=1, 958 cells excluded from training.
```

```
## iter=2, 949 cells excluded from training.
```

```
## Threshold found:0.496
```

```
## 922 (12.2%) doublets called
```

```
table(sce_SP$scDblFinder.class)
```

```
##
## singlet doublet
##    6656     922
```

```
doublets_scDblFinder_SP <- colnames(sce_SP[, sce_SP$scDblFinder.class == "doublet"])
```

Eliminating doublets from HTO analysis with scDblFinder

```
Seurat_Object_BM_singlets <- subset(Seurat_Object_BM, cells = setdiff(Cells(Seurat_Object_BM),
 doublets_scDblFinder_BM))
```

```
Seurat_Object_SP_singlets <- subset(Seurat_Object_SP, cells = setdiff(Cells(Seurat_Object_SP),
 doublets_scDblFinder_SP))
```

HTO signal

Group cells based on the max HTO signal

```
Idents(Seurat_Object_BM) <- "HTO_maxID"
RidgePlot(Seurat_Object_BM, assay = "HTO", features = rownames(Seurat_Object_BM[["HTO"]])[1:7]
, ncol = 2)
```

```
## Picking joint bandwidth of 0.0516
```

```
## Picking joint bandwidth of 0.127
```

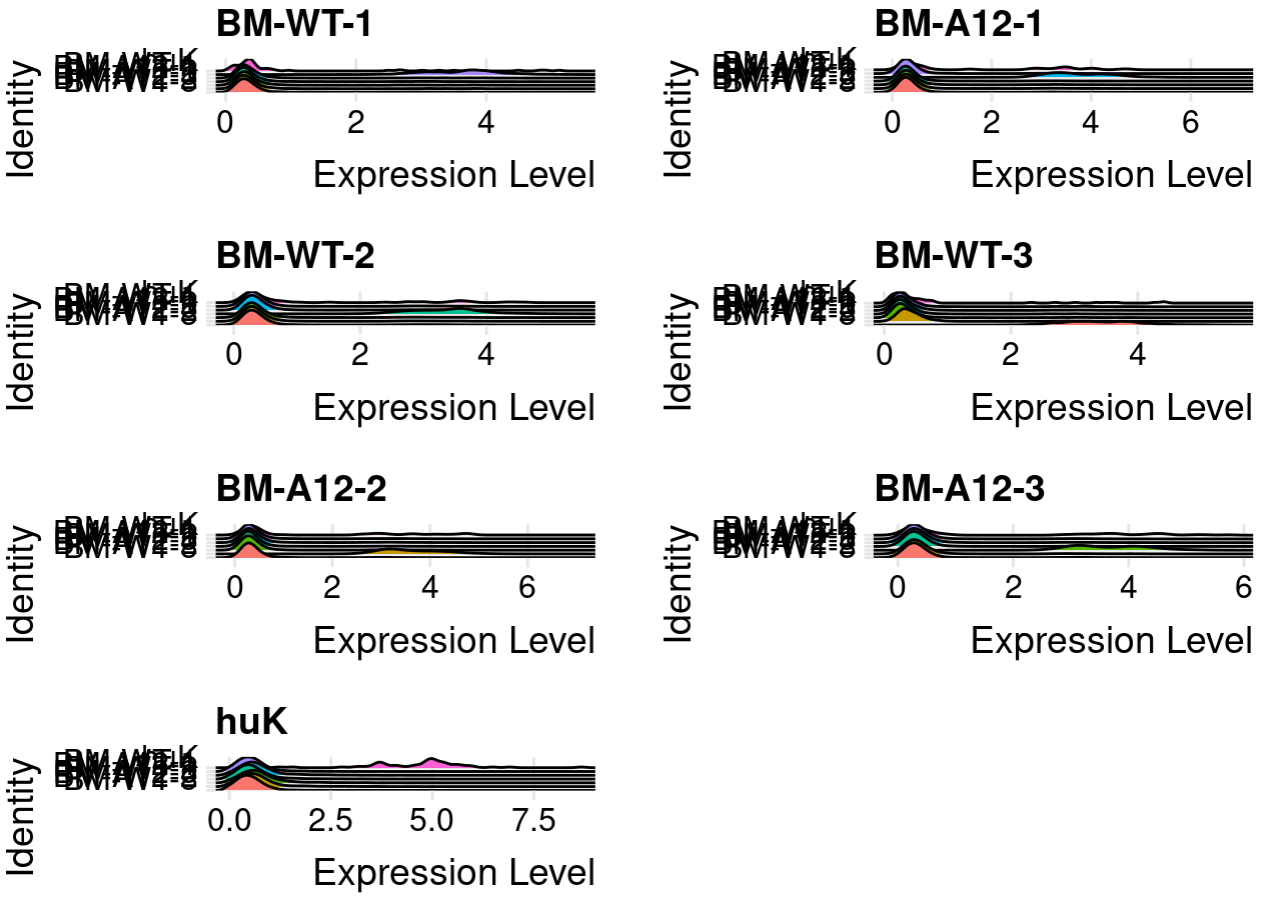
```
## Picking joint bandwidth of 0.0959
```

```
## Picking joint bandwidth of 0.0567
```

```
## Picking joint bandwidth of 0.131
```

```
## Picking joint bandwidth of 0.142
```

```
## Picking joint bandwidth of 0.113
```




```
Idents(Seurat_Object_SP) <- "HTO_maxID"
RidgePlot(Seurat_Object_SP, assay = "HTO", features = rownames(Seurat_Object_SP[["HTO"]])[1:7]
, ncol = 2)
```

```
## Picking joint bandwidth of 0.0334
```

```
## Picking joint bandwidth of 0.0503
```

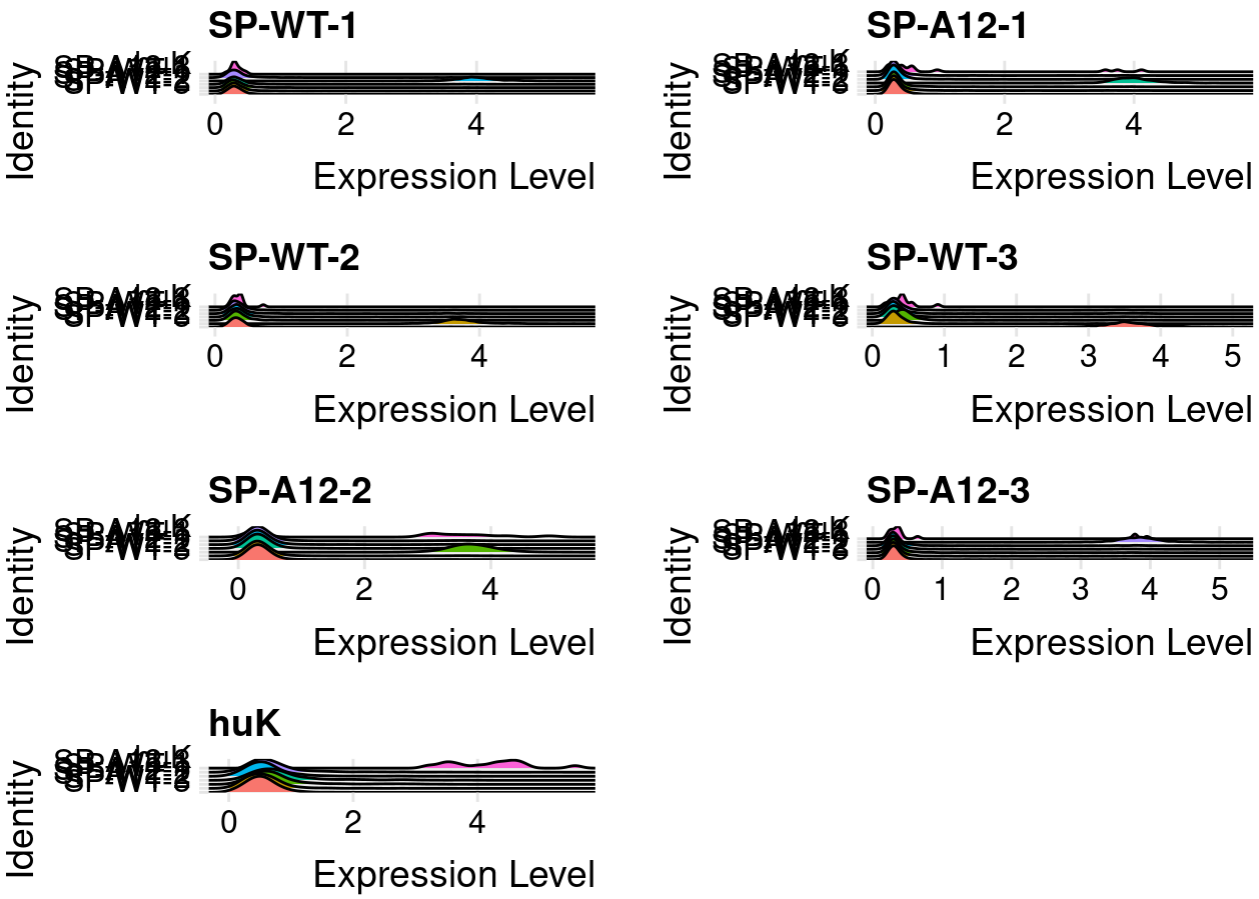
```
## Picking joint bandwidth of 0.0326
```

```
## Picking joint bandwidth of 0.0357
```

```
## Picking joint bandwidth of 0.157
```

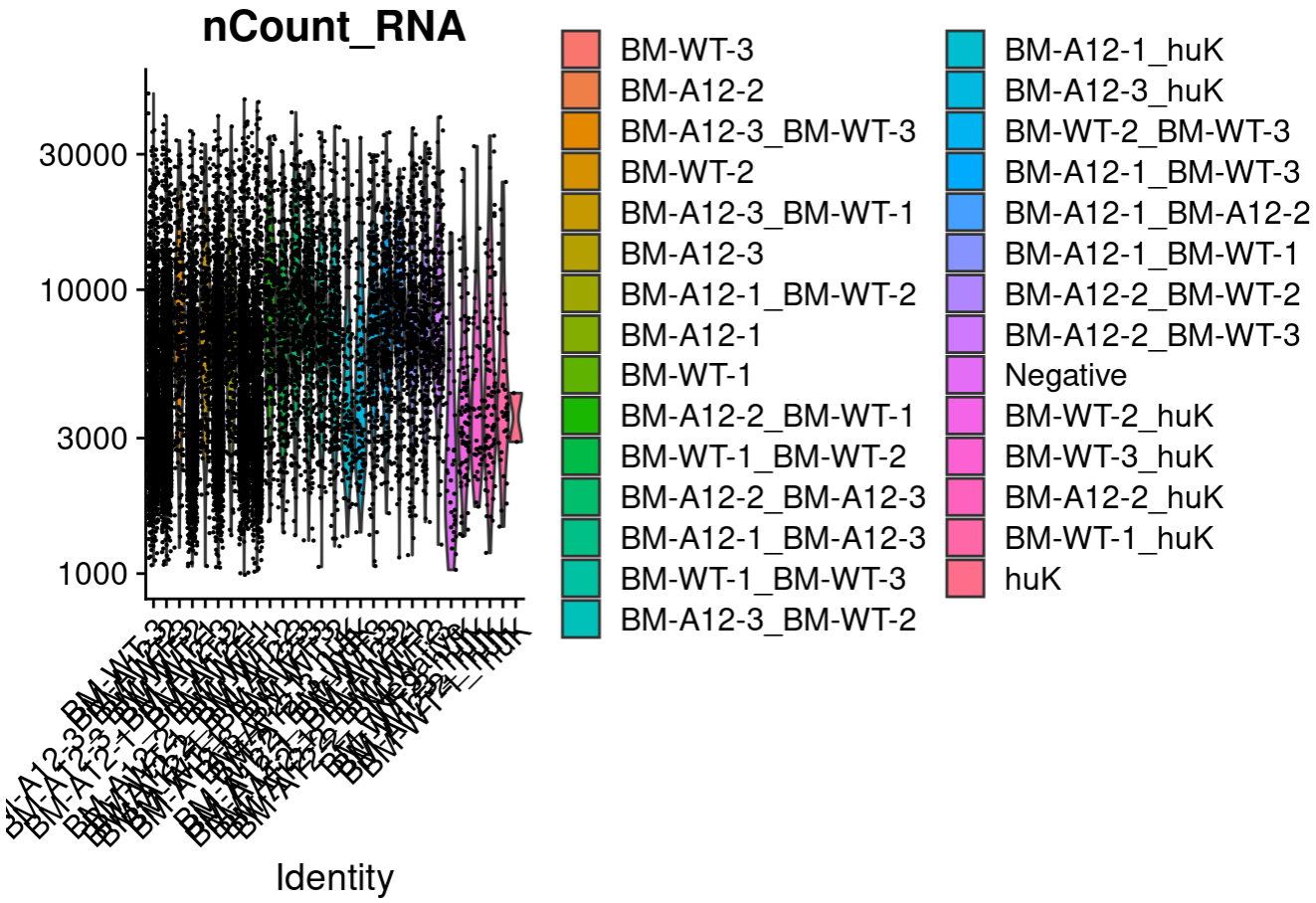
```
## Picking joint bandwidth of 0.0368
```

```
## Picking joint bandwidth of 0.108
```

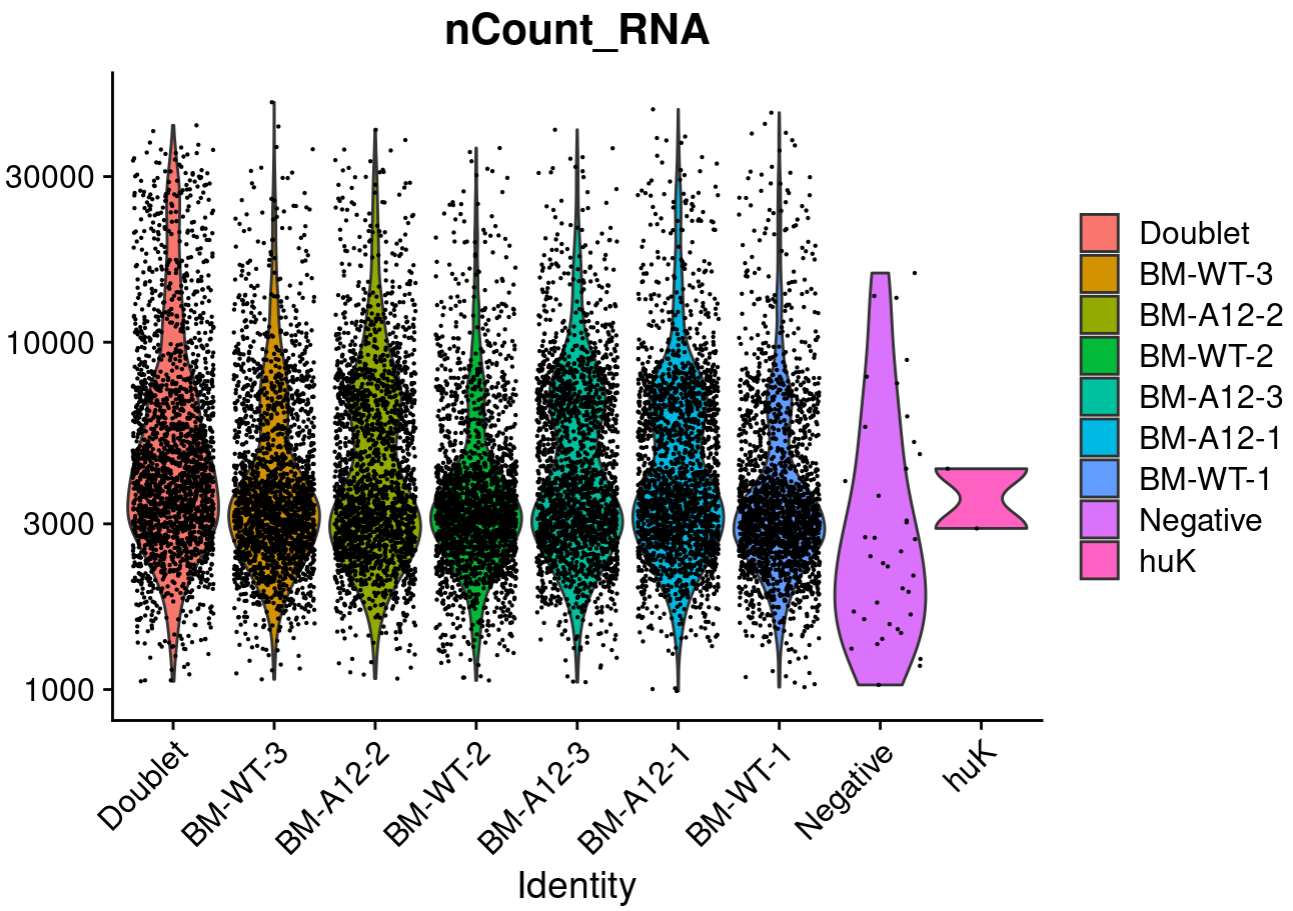


Compare number of UMIs for singlets, doublets and negative cells

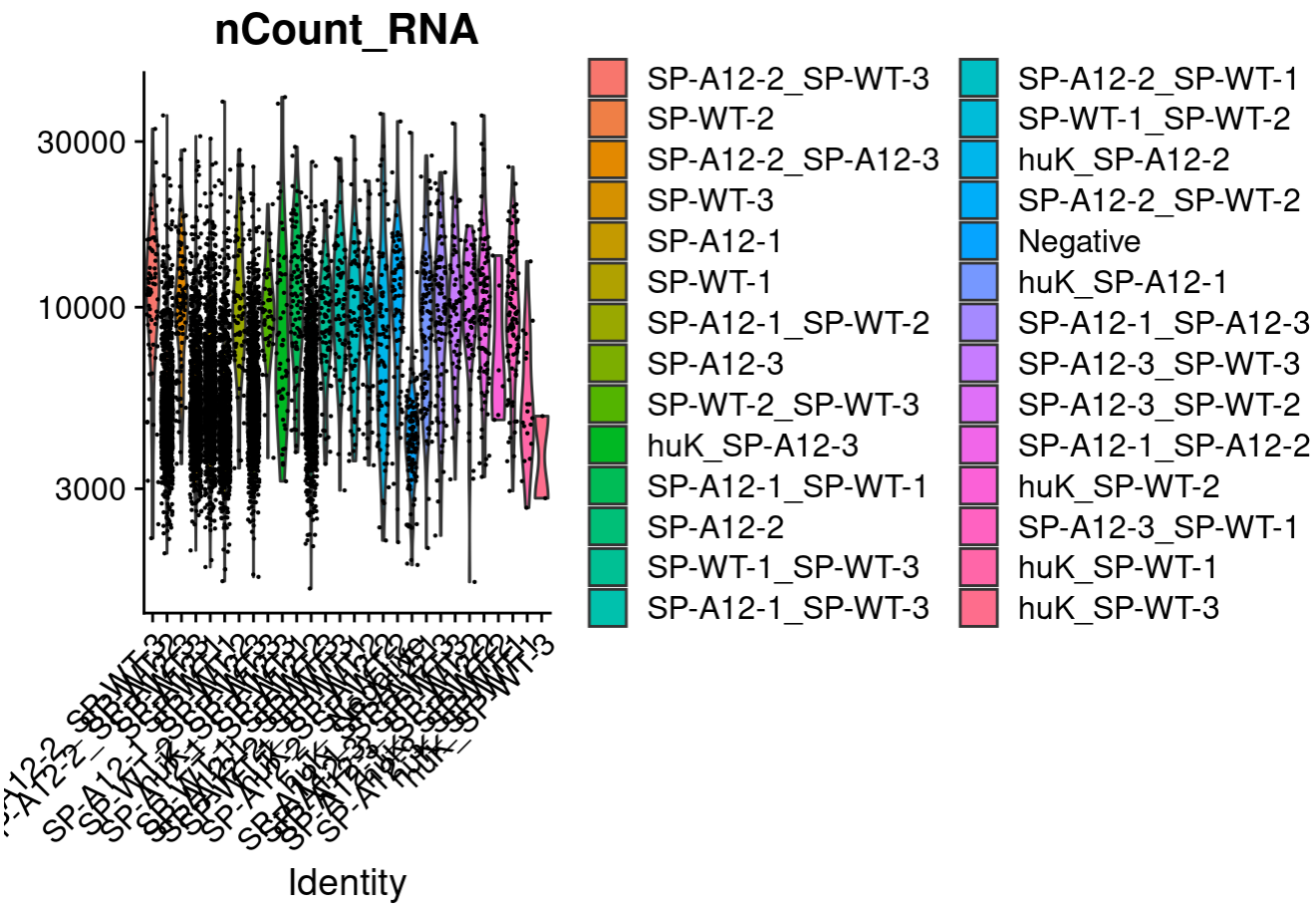
```
Idents(Seurat_Object_BM) <- "HTO_classification"
VlnPlot(Seurat_Object_BM, features = "nCount_RNA", pt.size = 0.1, log = TRUE)
```



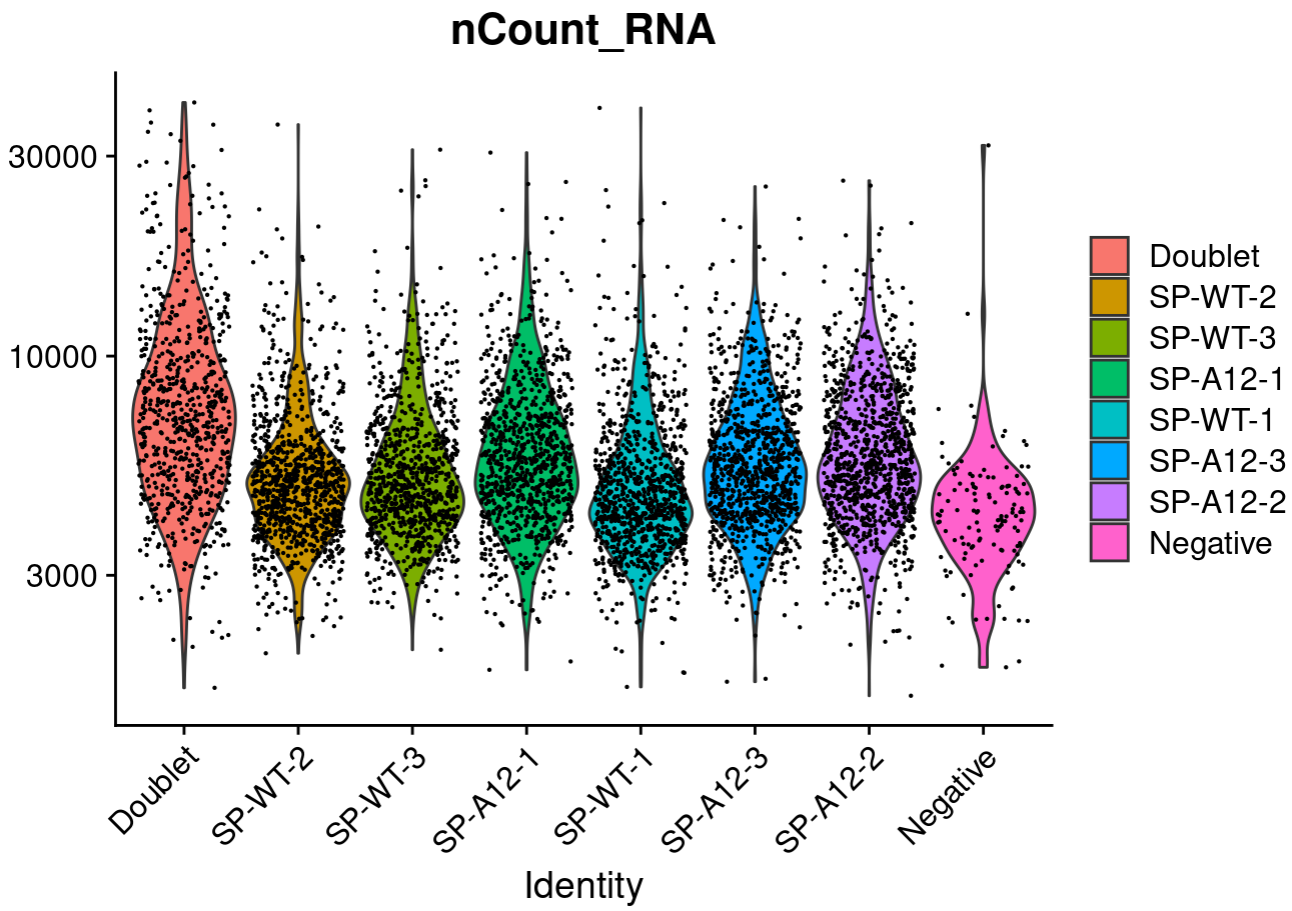

```
VlnPlot(Seurat_Object_BM_singlets, features = "nCount_RNA", pt.size = 0.1, log = TRUE)
```




```
Idents(Seurat_Object_SP) <- "HTO_classification"
VlnPlot(Seurat_Object_SP, features = "nCount_RNA", pt.size = 0.1, log = TRUE)
```

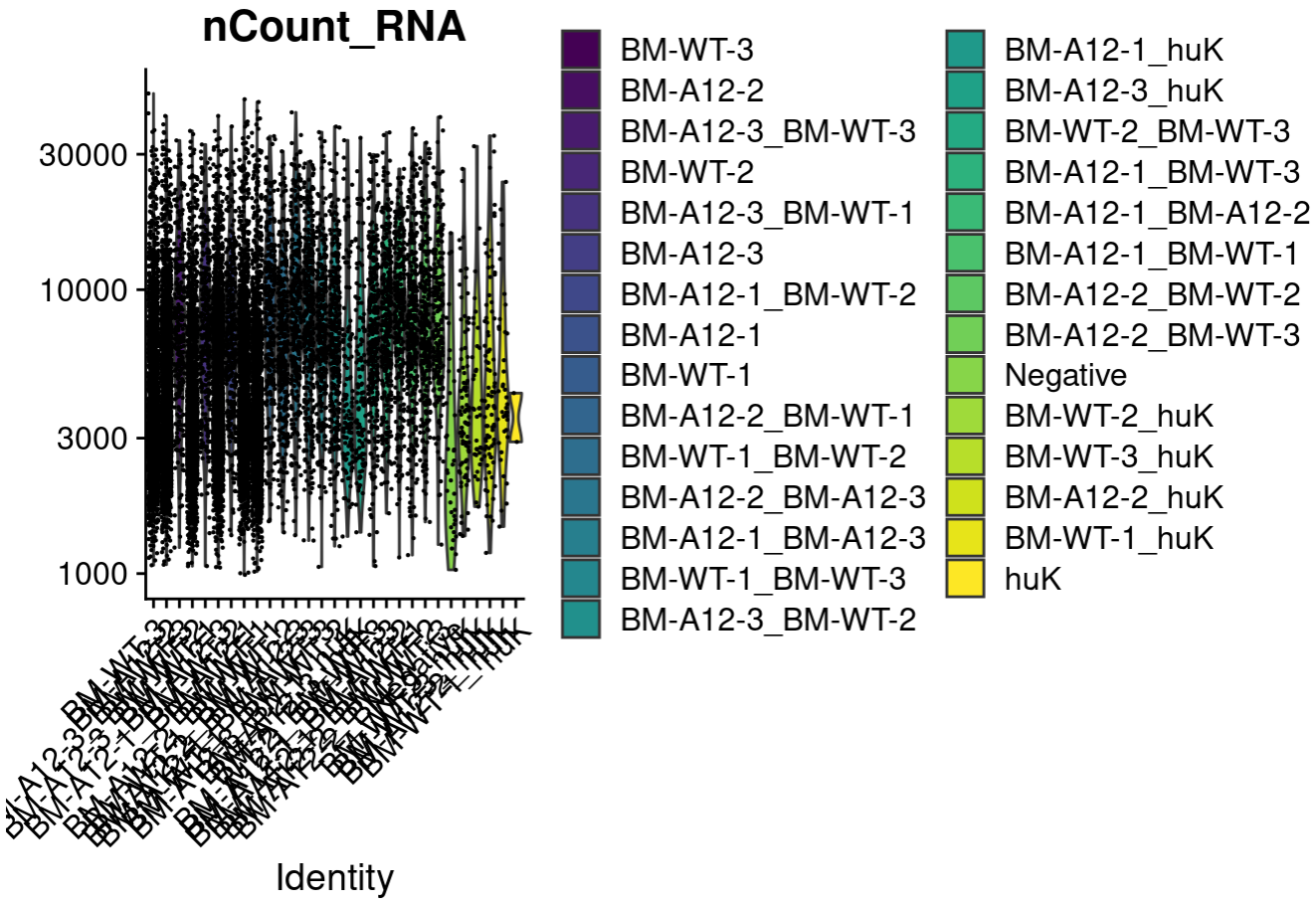



```
VlnPlot(Seurat_Object_SP_singlets, features = "nCount_RNA", pt.size = 0.1, log = TRUE)
```

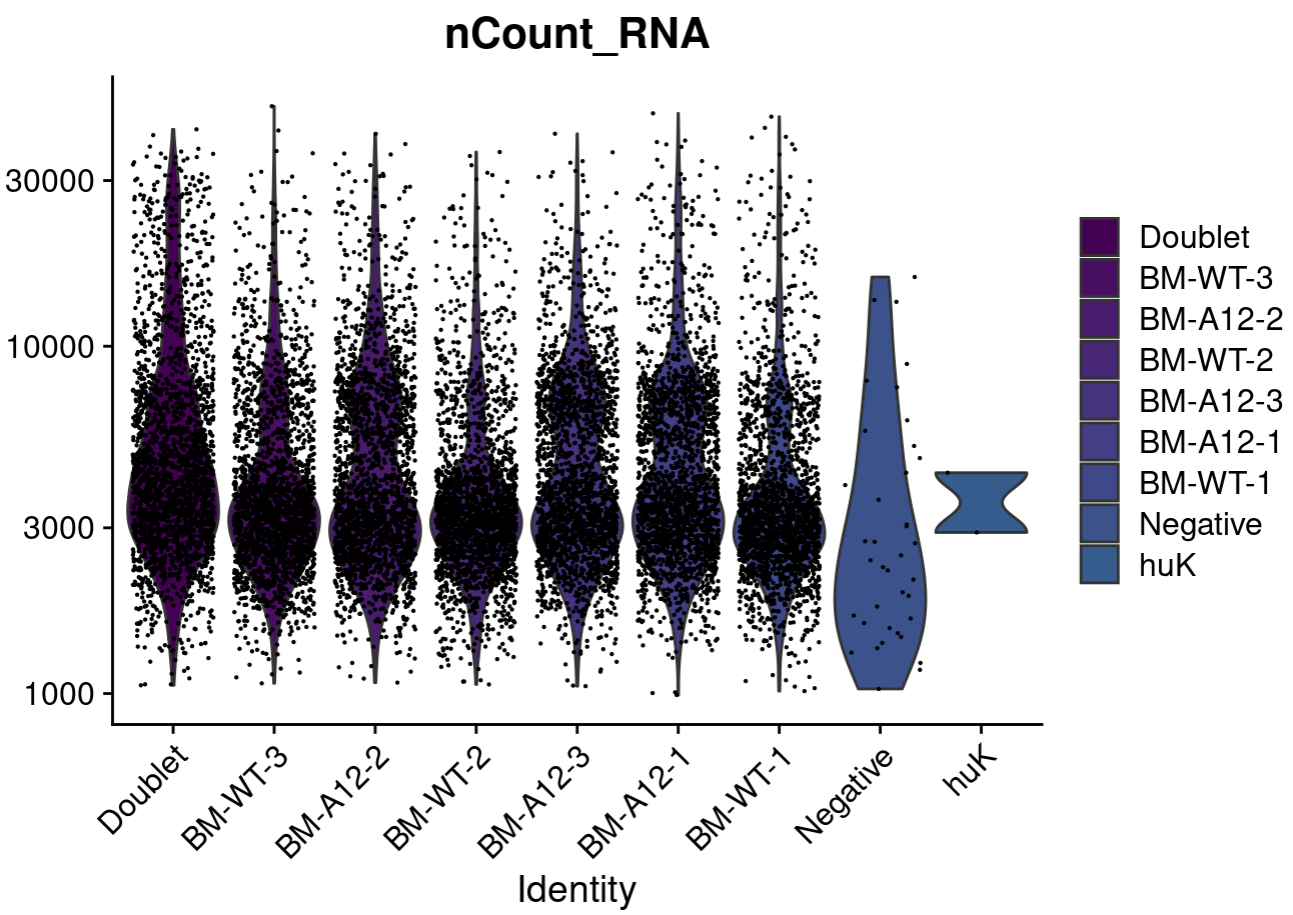


Compare number of UMIs for combinations of different antibodies

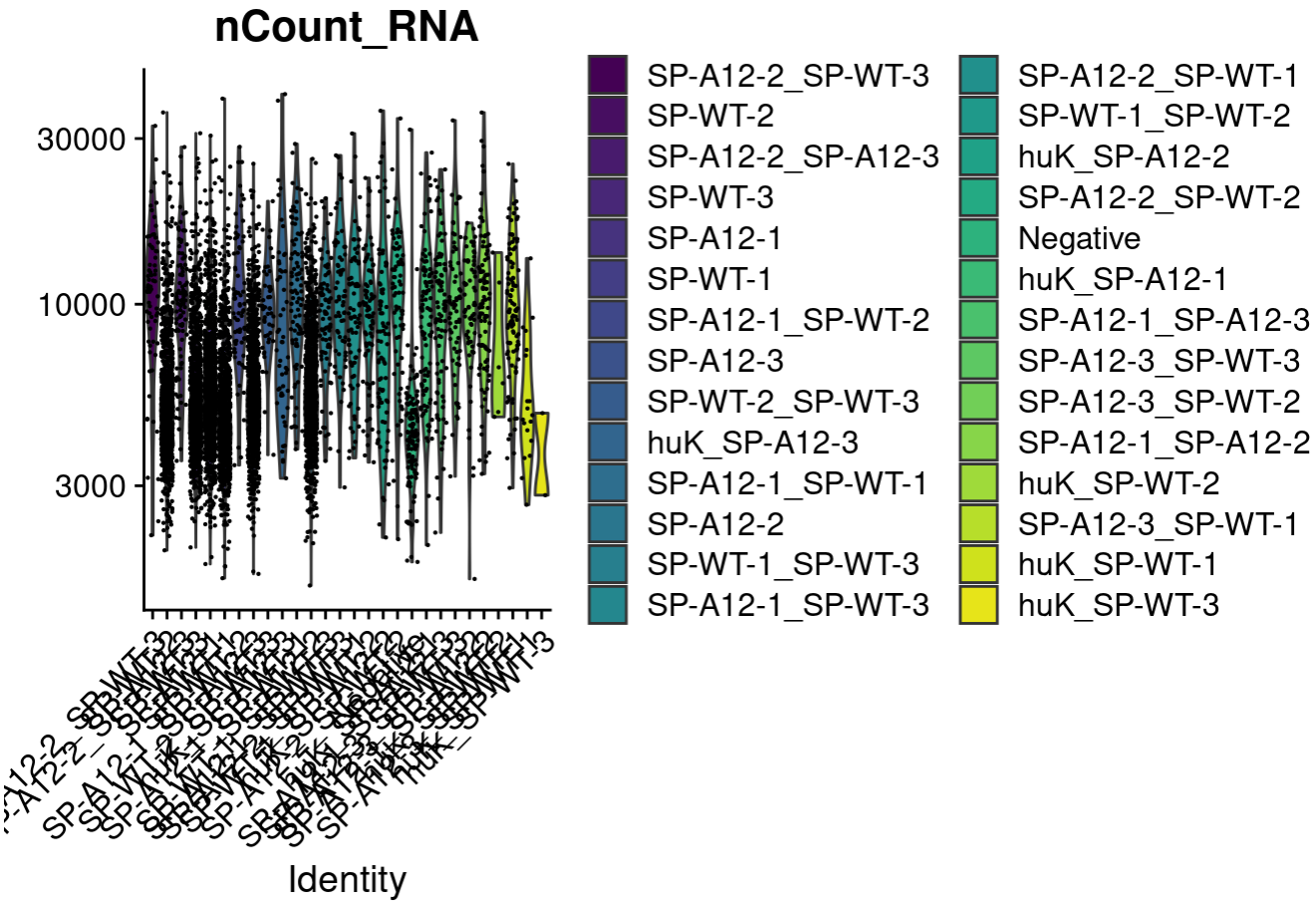
```
my_palette <- viridis(29)
VlnPlot(Seurat_Object_BM, features = "nCount_RNA", pt.size = 0.1, log = TRUE) +
  scale_fill_manual(values = my_palette)
```



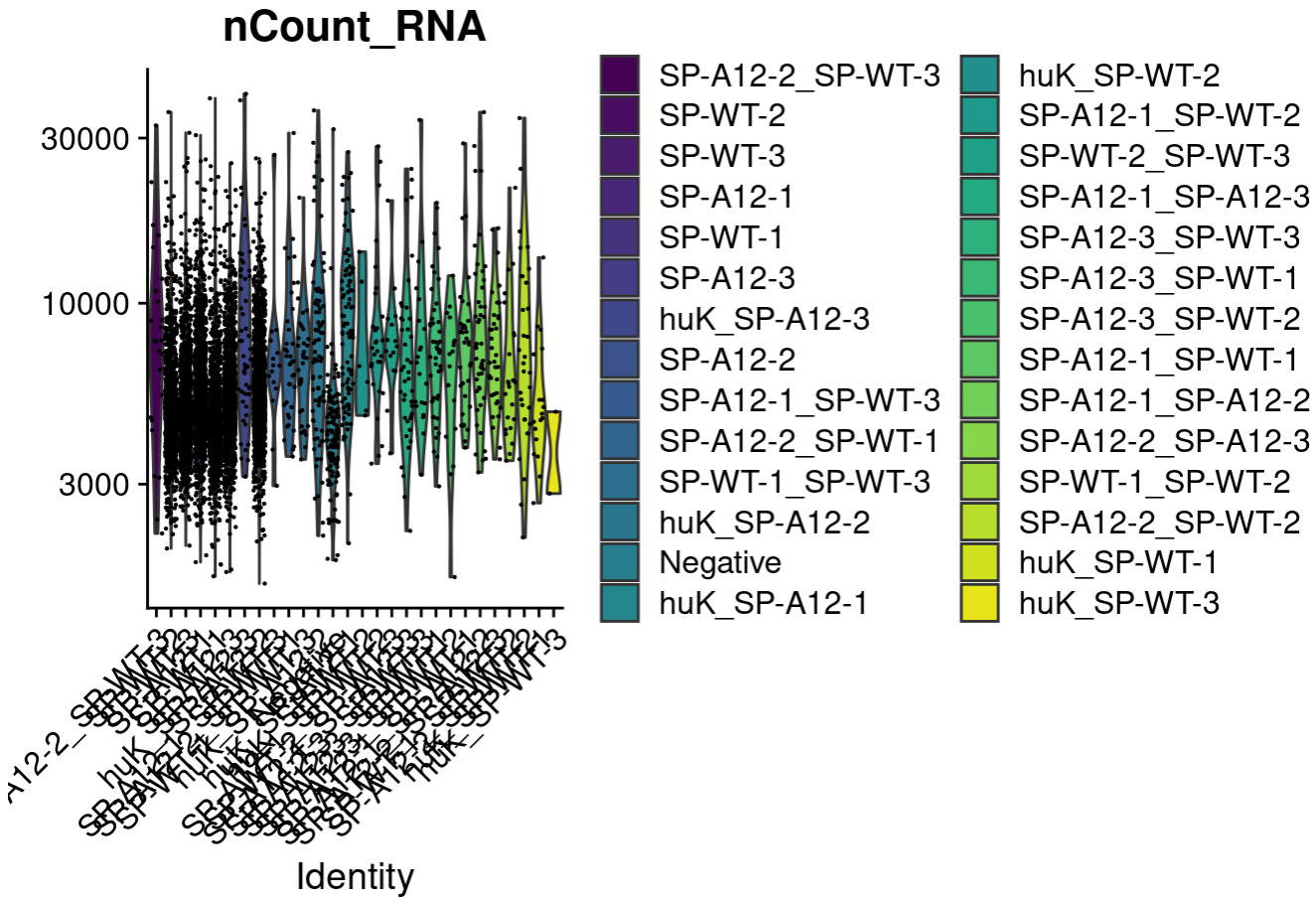

```
VlnPlot(Seurat_Object_BM_singlets, features = "nCount_RNA", pt.size = 0.1, log = TRUE) +
  scale_fill_manual(values = my_palette)
```




```
Idents(Seurat_Object_SP) <- "HTO_classification"
Idents(Seurat_Object_SP_singlets) <- "HTO_classification"
VlnPlot(Seurat_Object_SP, features = "nCount_RNA", pt.size = 0.1, log = TRUE)+
  scale_fill_manual(values = my_palette)
```




```
VlnPlot(Seurat_Object_SP_singlets, features = "nCount_RNA", pt.size = 0.1, log = TRUE)+
  scale_fill_manual(values = my_palette)
```



Keep HTO combinations that reflect singlets

```
Seurat_Object_BM_selected <- subset(Seurat_Object_BM_singlets, HTO_classification %in% c("BM-WT-2", "BM-WT-3", "BM-A12-3", "BM-A12-1", "BM-A12-2", "BM-WT-1", "BM-A12-1_huK", "BM-A12-3_huK", "BM-A12-1_huK", "BM-WT-2_huK", "BM-A12-2_huK", "BM-WT-1_huK", "BM-WT-3_huK"))
Seurat_Object_BM_selected
```

```
## An object of class Seurat
## 19891 features across 13409 samples within 2 assays
## Active assay: RNA (19884 features, 0 variable features)
##  2 layers present: counts, data
##  1 other assay present: HTO
```

```
Idents(Seurat_Object_BM_selected) <- "HTO_classification"
RidgePlot(Seurat_Object_BM_selected, assay = "HTO", features = rownames(Seurat_Object_BM_selected[["HTO"]])[1:6], ncol = 2)
```

```
## Picking joint bandwidth of 0.0707
```

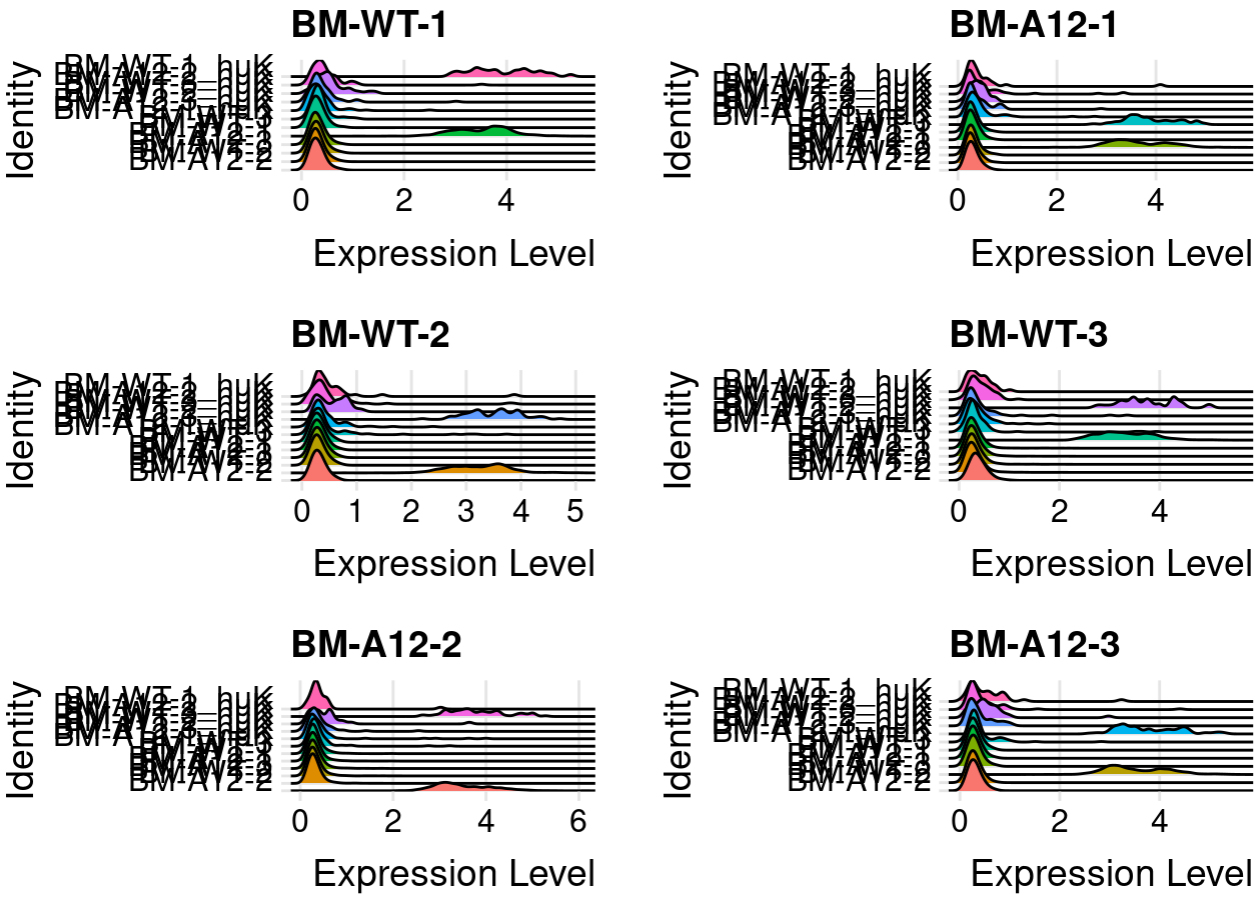
```
## Picking joint bandwidth of 0.0637
```

```
## Picking joint bandwidth of 0.0686
```

```
## Picking joint bandwidth of 0.0705
```

```
## Picking joint bandwidth of 0.0689
```

```
## Picking joint bandwidth of 0.0761
```




```
Seurat_Object_SP_selected <- subset(Seurat_Object_SP_singlets, HTO_classification %in% c("SP-WT-3", "SP-WT-2", "SP-A12-3", "SP-A12-2", "SP-WT-1", "SP-A12-1", "huK_SP-A12-1", "huK_SP-A12-2", "huK_SP-A12-3", "huK_SP-WT-3", "huK_SP-WT-2", "huK_SP-WT-1"))
Seurat_Object_SP_selected
```

```
## An object of class Seurat
## 17669 features across 6076 samples within 2 assays
## Active assay: RNA (17662 features, 0 variable features)
##  2 layers present: counts, data
##  1 other assay present: HTO
```

```
Idents(Seurat_Object_SP_selected) <- "HTO_classification"
RidgePlot(Seurat_Object_SP_selected, assay = "HTO", features = rownames(Seurat_Object_SP_selected[["HTO"]])[1:6], ncol = 2)
```

```
## Picking joint bandwidth of 0.0476
```

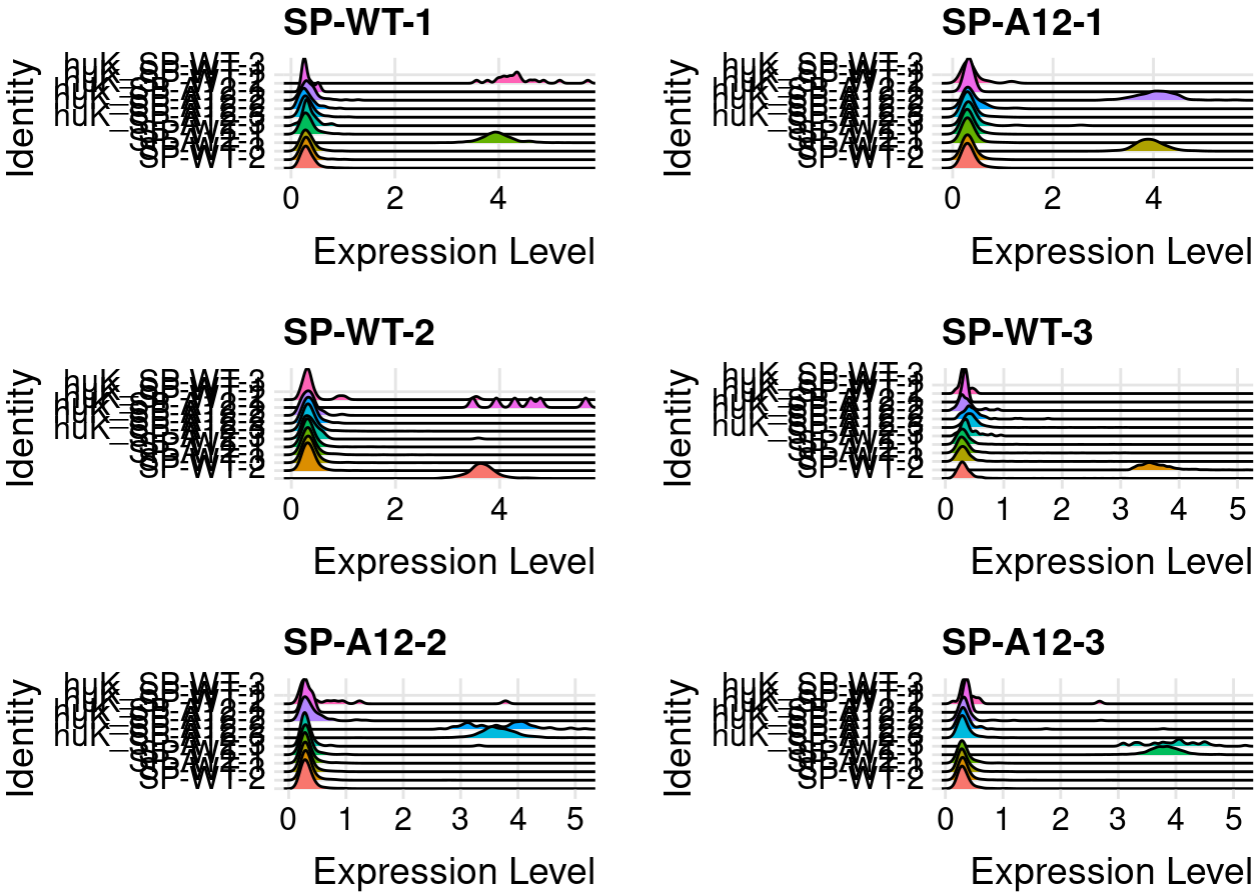
```
## Picking joint bandwidth of 0.0992
```

```
## Picking joint bandwidth of 0.0613
```

```
## Picking joint bandwidth of 0.0305
```

```
## Picking joint bandwidth of 0.0556
```

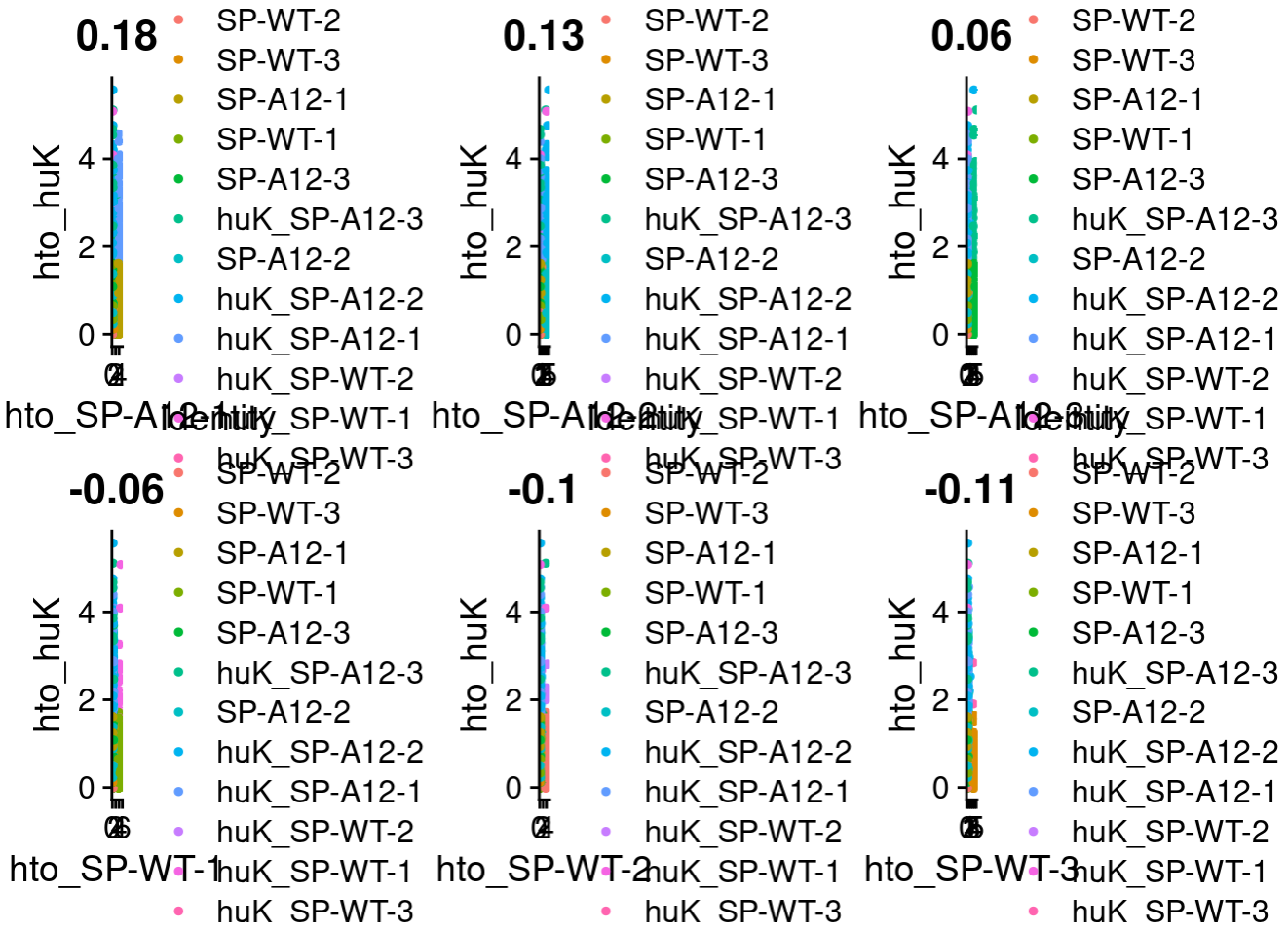
```
## Picking joint bandwidth of 0.0458
```



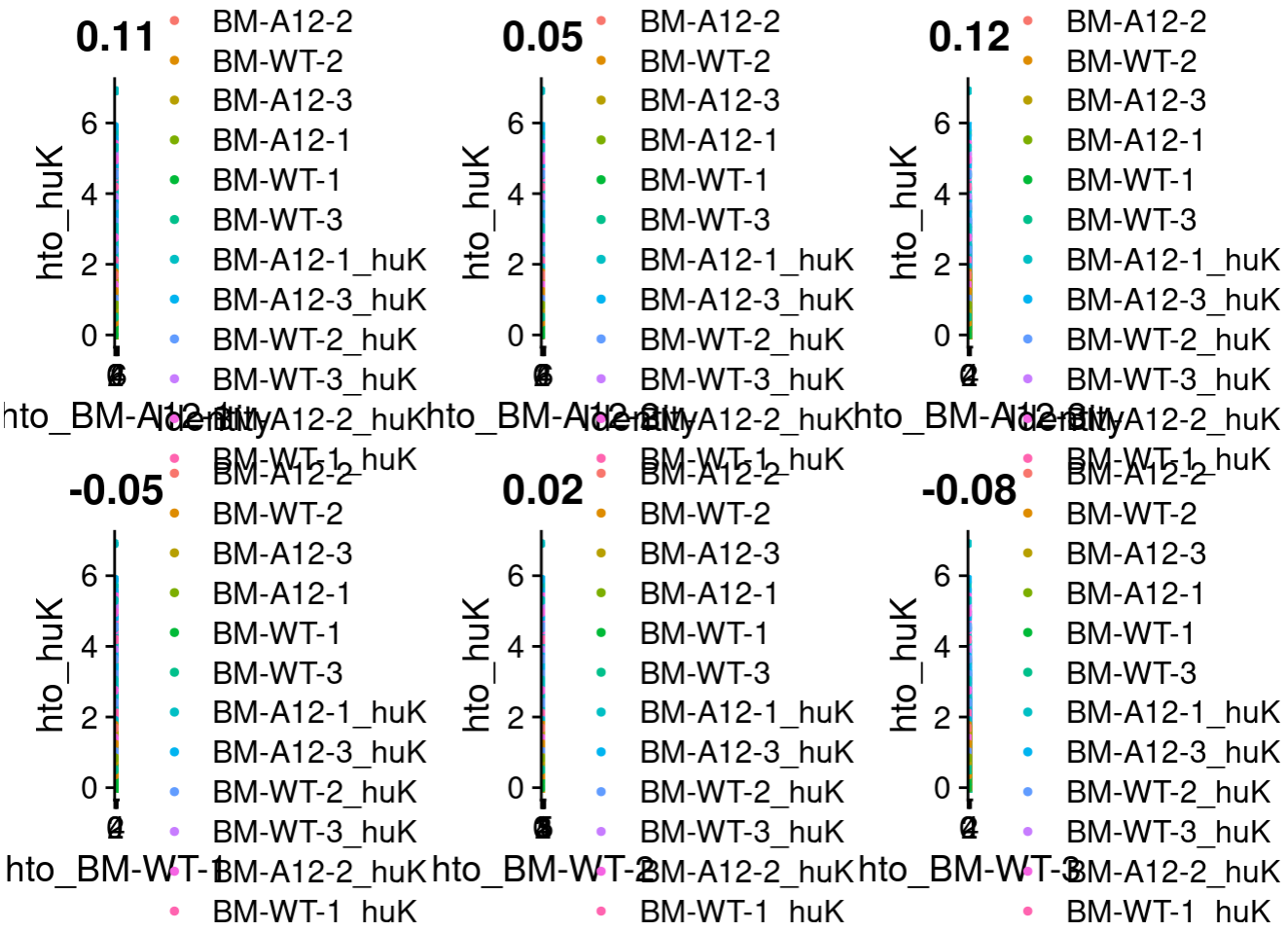
We are going to verify if doublets make sense (combination of huIgk with A12 HTOs)

```
plot1 <- FeatureScatter(Seurat_Object_SP_selected, feature1 = "hto_SP-A12-1", feature2 = "hto_
huK", pt.size = 1)
plot2 <- FeatureScatter(Seurat_Object_SP_selected, feature1 = "hto_SP-A12-2", feature2 = "hto_
huK", pt.size = 1)
plot3 <- FeatureScatter(Seurat_Object_SP_selected, feature1 = "hto_SP-A12-3", feature2 = "hto_
huK", pt.size = 1)
plot4 <- FeatureScatter(Seurat_Object_SP_selected, feature1 = "hto_SP-WT-1", feature2 = "hto_h
uK", pt.size = 1)
```

```
plot5 <- FeatureScatter(Seurat_Object_SP_selected, feature1 = "hto_SP-WT-2", feature2 = "hto_huK", pt.size = 1)
plot6 <- FeatureScatter(Seurat_Object_SP_selected, feature1 = "hto_SP-WT-3", feature2 = "hto_huK", pt.size = 1)
(plot1 + plot2 + plot3 + plot4 + plot5 + plot6)
```




```
plot1 <- FeatureScatter(Seurat_Object_BM_selected, feature1 = "hto_BM-A12-1", feature2 = "hto_huK", pt.size = 1)
plot2 <- FeatureScatter(Seurat_Object_BM_selected, feature1 = "hto_BM-A12-2", feature2 = "hto_huK", pt.size = 1)
plot3 <- FeatureScatter(Seurat_Object_BM_selected, feature1 = "hto_BM-A12-3", feature2 = "hto_huK", pt.size = 1)
plot4 <- FeatureScatter(Seurat_Object_BM_selected, feature1 = "hto_BM-WT-1", feature2 = "hto_huK", pt.size = 1)
plot5 <- FeatureScatter(Seurat_Object_BM_selected, feature1 = "hto_BM-WT-2", feature2 = "hto_huK", pt.size = 1)
plot6 <- FeatureScatter(Seurat_Object_BM_selected, feature1 = "hto_BM-WT-3", feature2 = "hto_huK", pt.size = 1)
(plot1 + plot2 + plot3 + plot4 + plot5 + plot6)
```



Identification of highly variable features (feature selection)

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

By default FindVariableFeatures() function return 2,000 features per dataset. These will be used in downstream analysis, like PCA.

```
DefaultAssay(Seurat_Object_SP_selected) <- "RNA"
Seurat_Object_SP_selected  <- FindVariableFeatures(Seurat_Object_SP_selected , selection.metho
d = "vst", nfeatures = 3000)
```

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

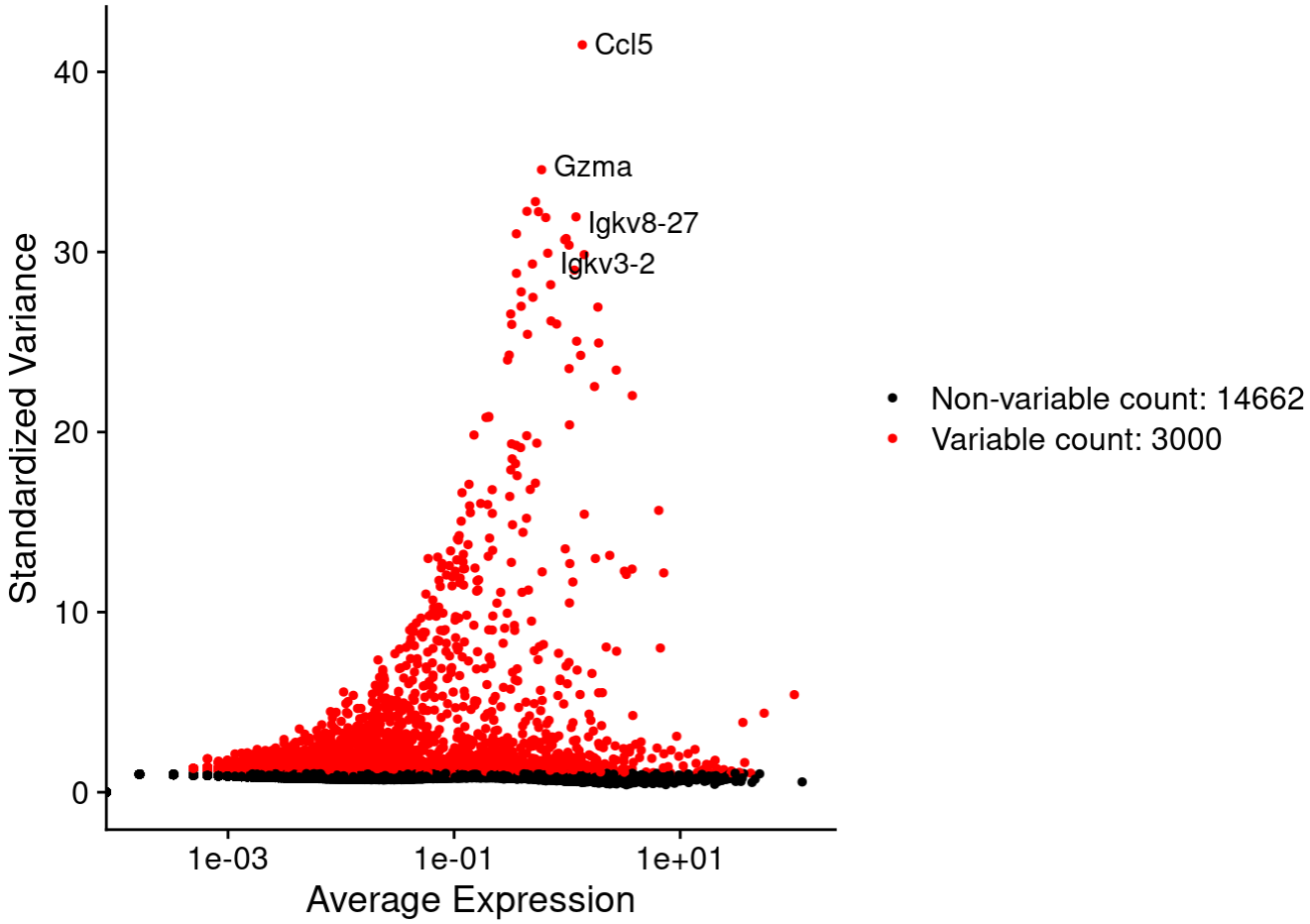
```
# Identify the 10 most highly variable genes
top10 <- head(VariableFeatures(Seurat_Object_SP_selected), 10)

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

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

```
plot2
```

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

```
## Warning: ggrepel: 6 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```




```
DefaultAssay(Seurat_Object_BM_selected) <- "RNA"
Seurat_Object_BM_selected <- FindVariableFeatures(Seurat_Object_BM_selected, selection.method
= "vst", nfeatures = 3000)
```

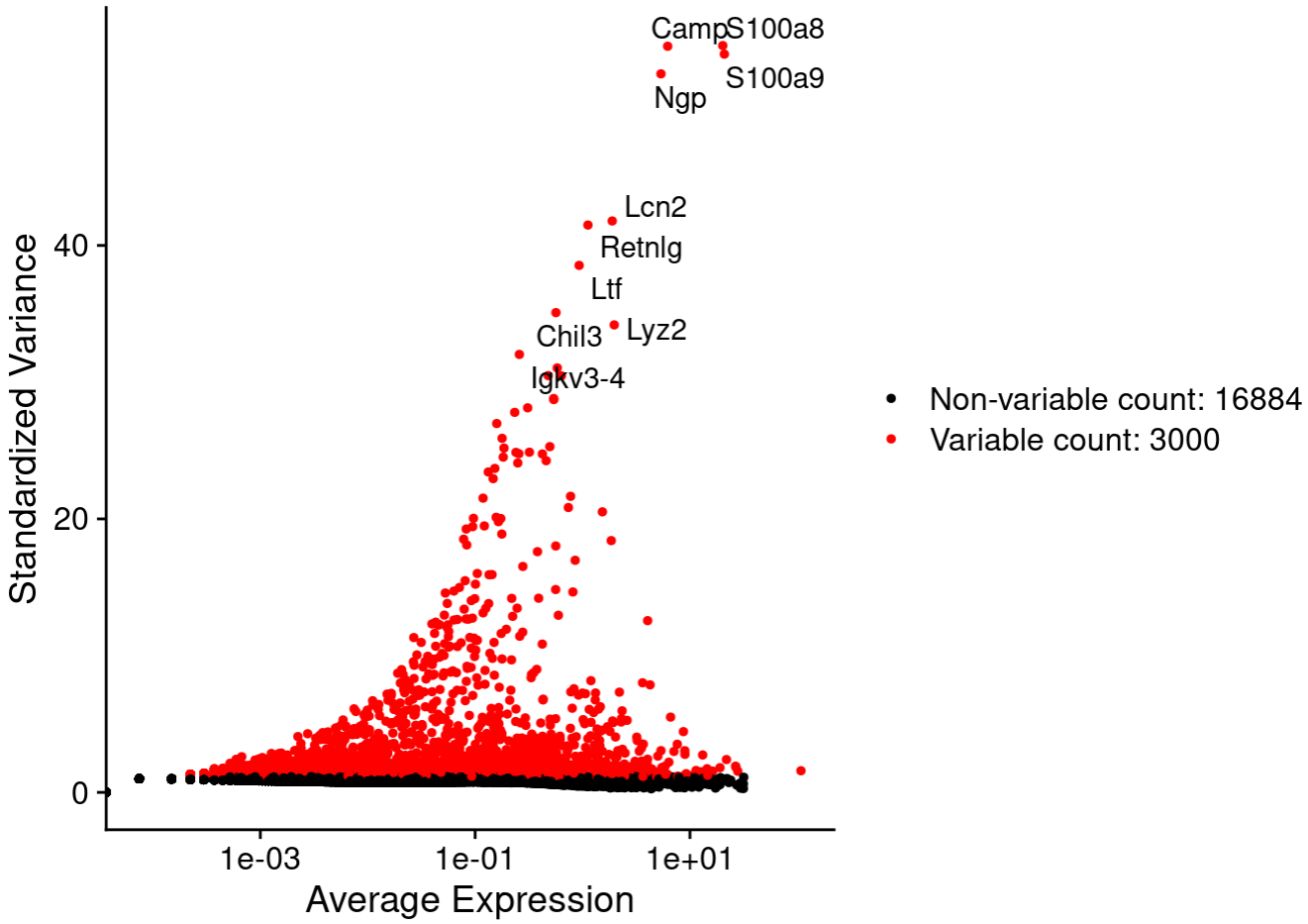
```
## Finding variable features for layer counts
```

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

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

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

```
plot2
```

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



Scaling the data

Next, we apply a linear transformation (‘scaling’) that is a standard pre-processing step prior to dimensional reduction techniques like PCA. The ScaleData() function: - Shifts the expression of each gene, so that the mean expression across cells is 0 - Scales the expression of each gene, so that the variance across cells is 1 (This step gives equal weight in downstream analyses, so that highly-expressed genes do not dominate) - The results of this are stored in pbmc[[“RNA”]]@scale.data

Although for PCA and clustering we are going to use only the Variable features, for other analysis such as Seurat Heatmaps it is important to have all the genes in the heatmeap to be scaled so the highly-expressed genes don’t dominate the heatmap. That’s hay we normalize all the genes.

NormalizeData

The NormalizeData function is used to normalize gene expression data within each cell. Normalizing the data is crucial in scRNA-seq to ensure that differences in sequencing depth (number of reads per cell) and cell size do not improperly affect the analysis results. The main approaches to normalization include:

Total Normalization: Divides the gene expression of each cell by the total gene expression for that cell, thereby normalizing for sequencing depth.

Reference Cell Normalization: Divides the gene expression of each cell by that of a reference cell (e.g., a cell with average gene expression).

ScaleData

The ScaleData function (also known as standardization) is applied after normalization to transform and standardize gene expression data to a common scale. This ensures that all features (genes) have comparable variance before performing statistical analyses like PCA (Principal Component Analysis). Standardization typically involves:

Z-Score Standardization: Subtracts the mean and divides by the standard deviation for each gene across all cells, so that each gene has a mean of zero and a standard deviation of one.

```
Seurat_Object_SP_selected <- ScaleData(Seurat_Object_SP_selected, features = VariableFeatures(
Seurat_Object_BM_selected))
```

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

```
Seurat_Object_BM_selected <- ScaleData(Seurat_Object_BM_selected, features = VariableFeatures(
Seurat_Object_BM_selected))
```

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

PCA

Next we perform PCA on the scaled data. By default, only the previously determined variable features are used as input, but can be defined using features argument if you wish to choose a different subset. We deplete the A12 gene from the clustering analysis in order to avoid bias.

```
variable_genes <- setdiff(VariableFeatures(Seurat_Object_SP_selected), c("A12"))
Seurat_Object_SP_selected <- RunPCA(Seurat_Object_SP_selected, features = variable_genes)
```

```
## Warning: The following 13 features requested have zero variance; running
## reduction without them: Gzma, Fkbp11, Slpi, Spon1, Cr2, Ppa1, Lyz2, Gm33489,
## Vcp, Pex11a, Hesx1, Pip4k2a, Acat3
```

```
## Warning in PrepDR5(object = object, features = features, layer = layer, : The
## following features were not available: Mybl1, Aicda, Slit3, Trp53inp1, H13,
## Gcsam, Sec11c, Atp1b1, Txndc11, Il7r, Mef2b, Dnajc3, Ddost, Sdc1, Hspa5,
## Trav12-3, Hmces, Ankrd55, Spcs1, 1810046K07Rik, Lpp, Dock4, Neil1, Dnajb11,
## Rftn1, Fam214a, Rgs1, Ddc, Irf4, Cpeb3, Gm31718, Ighv1-71, Gm20400, Apoc1,
## Ppib, Plxdc1, Snx24, Ackr3, D5Ertd615e, Sec61b, Smco4, Krtcap2, Tmed10, Pim1,
## AY036118, Glis3, Dlgap2, Lmo4, Ube2e2, Trav9-2, Frmd4b, Galnt3, Gadd45b,
## Slc1a3, Rhobtb1, Adgrv1, Sash1, Itga9, Sdc3, Ankub1, Asb2, S1pr2, Cpne4,
## Ranbp17, Cobl, Pafah1b3, Hexb, Ftl1, E330020D12Rik, Ahr, Igf1, Ssr3, Prss30,
## Selenop, Dgki, Ostc, Mreg, Nrn1, Creb3l2, Pdia3, Trav13n-1, Samd4, Ighv1-4,
## Adamdec1, A530064D06Rik, Dtx1, Marcks, Spcs2, Tex14, Ighv1-13.1, Hist1h3b,
```

```
## Wdfy3, Rab13, Gm43305, Epb41l3, Rassf6, Fcgr1, Nsg2, Myof, Myadm, Hfe, Gm12649,
## Cd1d1, Ccdc141, Gphn, S100a10, Nbea, Ksr2, Adam11, Xlr5a, Extl3, Hist1h2bp,
## Pycard, Gm40841, Gm5820, Cst6, Tbxas1, Spic, Sub1, Gas7, Fam135a, S1pr3, Cnpy2,
## Rrbp1, Maf, Gm43698, Tbc1d4, Mfhas1, Dpp10, Thsd7a, Cd5, Gm45552, Igkv3-11,
## Bhlhe40, Copz2, Clptm1l, C6, Ighv5-1, Igkv12-42, Sema6d, Sec61a1, Atg4a,
## Slc35b1, Abcc3, Nid1, Npc2, Cdh13, Dph5, Adgrl2, 2900052N01Rik, Syndig1l,
## Adgre1, Cttnbp2nl, Ppp1r14a, Cd274, Hyou1, Ankrd35, Rhpn1, Cd14, Adrb1, Saraf,
## Dusp9, Cacna1e, Trav6n-6, Adgrl3, Camkmt, Fgfr1, Pkd2l2, Pipox, Gstt1, Gng4,
## Zfp521, Neo1, Ift20, Epha7, Rin2, Banf2os, Cpd, Zeb2, Cacna1d, Nos1, Slco2b1,
## Wnt10b, Ccdc28b, Ccr4, Gm32200, Gm31508, Ptpn3, Mybpc1, Top1, Sox4, Krt80,
## Trav6-5, Tmem273, Acoxl, Wdr95, Gm20528, Prkcsh, Zc3h12b, Mpzl1, Crb1, Akr1b7,
## Cdhr5, Rfx4, Cep85, Rflna, Fam129c, Arhgap24, 4933427D06Rik, Emp2, Gm1673,
## Sel1l, Adssl1, Fhit, Gm11630, Polh, Nrg2, Ugt1a7c, Mydgf, Cntn5, Gm12678,
## A430010J10Rik, Hspa13, Gm32184, Prlr, Nr3c2, Trav16n, Bhlhe41, Gm39091, Dnajb9,
## Jun, E230029C05Rik, Thnsl2, Gpr35, Gusb, Coro2b, Gm33843, Parp8, Myo9a,
## Fam184a, Dnajc7, Tenm4, Lhfpl4, Selenok, 1700034P13Rik, Thbd, A330084C13Rik,
## Apol7c, Bcl2a1d, Fscn1, Dusp14, Tspan15, Etl4, Vpreb3, Nt5e, Emp3, Fndc3a,
## A830005F24Rik, Itgae, Fzd6, Arid3a, Dusp16, Gpc1, Akap12, Pls1, Litaf, Klrb1,
## Ube2j1, Kcnj10, Dpysl5, Slc11a1, Gm20756, Cd163, Dst, Trabd2b, Hfm1, Kifc3,
## AC133103.1, Trat1, Gnaz, Pilrb1, Car5a, Gjb2, Gm17057, Sgsm3, Rsu1, Trps1,
## Lman2, Col14a1, 1700062C10Rik, Fam83b, Hes5, Ifi47, Surf4, Arhgap8, Nfatc1,
## 1700029I15Rik, Dao, Gm47754, Zbtb20, 4933406I18Rik, Mycl, Il12rb2, Gstt3,
## Gpm6a, Bcl2l11, Trav19, Sema7a, Otos, Slc45a3, Dysf, Mgl2, Cd300e, Plcb4, Pls3,
## Pdlim4, Gpr162, Sgip1, Hs3st2, Hbb-bh1, 5430425K12Rik, Fmn1, Actg1, Immp2l,
## Tesc, Trbj1-1, Grip1, Herpud1, Sec61g, Gm14341, Gm43303, Slc22a13, Gm5427,
## Rab34, Samd12, Hs6st3, Pdlim1, Trbj2-1, C1qtnf6, Sned1, Zdhhc14, Prkca, Fam20c,
## Sntg2, Mxd4, Gm38037, She, Gpr4, Nlrp1a, Sp140, Aldh3b1, Cldn7, Ptpn14, Cmpk1,
## Gm32312, Kctd14, Selenos, Tmed2, Kdm2b, Ssr1, Gpr155, Rgl1, Tmcc3, Ptpn13,
## Gxylt2, Kcnq1ot1, Ntrk3, Rcbtb2, Timd2, Rgs7bp, Klhl4, Tubb2b, Plxnb2, Ccr7,
## Cd6, Creg2, Col4a4, Atp6v1g3, C1ql3, Fibcd1, Dll4, Slc12a5, Frem2, S100a2,
## Fhl5, Gm11209, Stpg1, Mmel1, Gm43566, Gm40304, Cxcl9, Oas1g, Ppp1r9a, Prokr1,
## Kcnj8, Atp4a, Hmx2, BC048644, Cep126, Gpr83, Rgl3, Chst2, Gm48486, Lrrc75b,
## Madcam1, 1700030O20Rik, Gm36862, Cdhr3, Lrrc72, Kcnh5, Gm36264, Nhlrc1,
## Rnf144b, Gm41031, Gm47849, Gm15287, Vstm4, Traj58, Gm17035, Pdzrn4,
## 5730414N17Rik, Pkp2, B3galt5, Togaram2, Ltbp1, Gm50146, Kctd12b, Cap2, Ddit4,
## Gh, Klk8, Lztfl1, Lrrc59, St8sia1, Hmgn3, Atp8a2, Cntnap5c, Gm41071, Gm13387,
## Muc1, Gm11817, Gm12861, Gm47328, Ccdc192, Pik3r4, Tshz3, Ccdc151, Gm2447,
## Sorbs2, Tgfbr3, Rab11fip4, Stat1, Slc16a5, Tagap, Entpd1, Klra10, Pdgfc,
## Gm26704, Cd86, Hrh1, Gm43914, Hormad2, Nova1, Ighv6-4, Trav1, Mrap, Tnfsf12,
## Cd44, Bid, 4933424M12Rik, Cacna1h, Gm43400, Galnt17, Cadps2, Xcr1, Olfr889,
## Leprotl1, Cpq, Art2b, Iglc3, Trav14-2, Prdm16, Gm5535, Mnd1, Alpl, Spint2,
## Clec1a, Cep112, Gm39383, Lcn4, Bet1, 2310008N11Rik, Hcrtr2, Hpgds, Rad54b,
## Phf11b, Adam22, Tmem248, Gm11827, Rcn3, Trim30c, Gm4610, Mras, Arhgap29,
## Ldlrad3, Syn3, Ccr6, Tnf, Matk, Insig1, Marveld1, Trpm8, Gm33994, Trim55,
## S100a3, 4930555A03Rik, Gm12972, Reep1, Gm44970, Cyp2s1, Myh14, Anpep, Dkk3,
## 3930402G23Rik, B930018H19Rik, 9530059O14Rik, Rab36, Gal3st1, Myl7, Med9os,
## Vsnl1, AI463170, Gpr141b, Dsp, Gm20767, Fam83h, A4galt, Tnp2, Epha3, Trim40,
## Catsperd, 1700065O20Rik, Gm19500, Prelid3a, 3010001F23Rik, Tmem214, Maml2,
## Tbc1d9, Nlrc5, Lrrc49, Sh3bp4, B3gnt9, Pak1, Klf2, Map2, Tmem51, Trerf1,
## Dennd5b, Gm46224, 4930471E19Rik, Cysltr1, Kcp, Gm45866, Gm30382, Ifi213,
## A930005H10Rik, AI847159, Acvr1, Rgs6, Gm49086, Gm43434, Ramp3, Rab30,
## A430093F15Rik, Lamc2, 4930534H03Rik, Irf1, Srprb, Arl4a, Abr, Epha2, Ephx1,
## Tmem256, A3galt2, Ms4a4a, Gm20743, Trpm5, Abcg3, Gm13822, Fndc3b, Uap1l1,
## Spry2, Gm44110, Tns3, Mir155hg, Tnfsf14, 2410018L13Rik, AI662270, Adgrg5,
```

```
## Rasgef1b, Trav14d-3-dv8, Zbp1, Npas2, Gm973, Gm28942, A230020J21Rik, Upp2,
## Gm13657, Gm14019, C030034L19Rik, Iqgap3, A930002I21Rik, Mybphl, 4933405D12Rik,
## Cth, Gm12862, Padi3, Tmem82, Zfp988, Crmp1, 1700027F09Rik, Fzd9, Gm26597,
## Trbj1-6, Trbj2-2, Lmcd1, Nrip3, 4931431B13Rik, Proz, K230015D01Rik, Kank2,
## Gm48293, Prtg, Stac, 4930520O04Rik, Pomgnt2, 4930579P08Rik, Arhgef25, Apon,
## Rasgef1c, Gm43951, Serpinb1c, 1700037F03Rik, Gm30108, 2610528A11Rik, Gm49249,
## Gm41290, Rapgef3os2, Cldn14, Rnf39, Stap2, 1700122C19Rik, Pygm, Irf2bp2, Rgs2,
## Bach2, Ppan, Sec24a, Klf12, Mlec, Erlec1, Dock7, A630014C17Rik, Ahcyl2,
## Slc25a19, C130026I21Rik, Ppm1e, Cbfa2t3, Gpr176, Sytl2, Txlnb, Dach2, Rpl32,
## Tmtc1, Gm47015, Mtf2, Cyp51, Erp44, Chka, Ube2a, Dnajc15, Fap, Pdgfa, Cav2,
## Itga5, Ctbp2, Gm41804, Gm829, Gm32479, Sostdc1, Il2ra, Ptprj, Sil1, Pdgfb,
## Pbx3, Enpp1, A330040F15Rik, Gm13710, Trim16, Stt3a, Gne, Vipr1, Scfd2, Gm34215,
## Sspn, Mcam, Irf6, Rtn4rl2, Spsb1, Podxl2, 4930550C14Rik, Tmem266, Acvr2b,
## F630206G17Rik, Peli3, AC149090.1, Utrn, 2210408F21Rik, Pakap.1, Bhlha15, Erp29,
## Cyp39a1, Cecr2, Sbf2, Nfasc, Gm14124, Vax2os, 4930452B06Rik, Trav10n, Nfkbiz,
## St13, Ubash3b, Hspa2, Slamf1, Bcl6, Mtdh, Trav13n-4, Arl11, Clec2i, Gm50386,
## Soga1, Dnah12, Tmem131, Nans, Id3, Ddt, Ddx43, Snrnp25, AW011738, Hdac8, Mia2,
## Oasl1, Irgm1, H2-Q7, Mapk6, 2510009E07Rik, Ifi208, Nacc2, Xaf1, Lynx1, Sh2b2,
## Tgfbr1, Hist1h2ag, Gm48857, 5730507C01Rik, Gns, Plec, Pkp4, Slc15a2, Farp2,
## Os9, BC035044, Dll1, Apobec2, Gm16124, Amigo2, Otulinl, Fads3, Fas, Slc7a5,
## Arl5c, Cdh17, Cd96, Ppfibp2, Lta, Mbd4, Nrp2, Hepacam2, Smad3, Gm10552, Hivep1,
## Slc44a1, Plxnc1, Lima1, Gm42031, Il5ra, Platr27, B3galnt1, Pinlyp, Rnf180,
## Lrp5, Nbas, Marco, Gm37126, Kcns1, Bmp7, A330015K06Rik, Chil6, Gm10961, Stbd1,
## Trbj1-4, Wdr54, Gm44646, Me3, Khdc3, Gm48653, Pnma1, Serpinb1b, C1qtnf9,
## Pou4f1, Gm48972, Csmd3, 1700085D07Rik, 8430426J06Rik, Gm41609, Gm50287, Nmnat2,
## Sestd1, Fer1l4, Kcna2, Gm11802, Heyl, Trbj2-5, Lama4, Timm8a2, Gm49959,
## Slc9a3r2, Scd4, Snhg18, Rarg, Fancd2, Srp9, Ighj3, Tgtp2, A530040E14Rik, Ampd1,
## Prx, Gm28905, AI463229, Zfp9, Aldh1a3, Tram2, Gpr55, Alpk1, Tnfaip3, Cxcr4,
## Gm16083, Echdc3, Junb, Smim24, Phtf2, Nfkbie, Kdm6b, Tent5a, 5033421B08Rik,
## Stard9, Magt1, Clec2g, Adap2, Nsl1, Eif2ak2, Mirt1, Evc, Gm47918, Dnah11, Gcat,
## 4930417O13Rik, Gm11998, Robo1, Tnfrsf25, Rapgef5, Siah2, Mtfr2, Lap3, Inpp4b,
## Ndufa1, A530032D15Rik, Smagp, Fdft1, Baiap3, Pik3r5, Cuedc1, Fhad1, Tbkbp1,
## Klhl30, Hic1, Adamtsl4, Hip1, Gm4258, Tpst2, 6330415G19Rik, Lrrc25, Serpinf1,
## Slc22a15, Klrc3, Lyst, Ifi206, Mfsd10, Gtf2i, Sema5a, Ercc6l2, Mbd2, Mrpl57,
## Rilpl1, Isyna1, Sh3rf1, Renbp, Srp19, Rps26, Psme2, Fam89a, Tmem258, Dnase1l3,
## Atat1, B3gnt7, Uba5, Tsc22d3, Rassf8, Bc1, Zc3h7a, Tbc1d2b, Adamts6, Mcoln2,
## Lgmn, Ppp4r2, Heg1, 2010300C02Rik, Cblc, Sept11, Mical3, Mob3b, 2810408I11Rik,
## Upk1a, Gm36278, Rps6kl1, Cmbl, Gm45029, Col17a1, Il6ra, Sik1, Spry1, Hist1h2bk,
## Gm1604a, Mtln, Fbxw7, Grwd1, Tpk1, Apol7e, Grk3, Xrcc1, Dusp2, Srpr, Fer, Grk5,
## Rpia, Ccpg1, Thrb, Mboat1, Tgif1, Wdr49, Rgs9, Arl15, Edaradd, Il6st, Pno1,
## Gm28376, Cdyl2, Abcb1b, Kdelr1, Atp6v0a1, Vegfb, Igtp, Tspan13, Nop2, Timm23,
## Ndufa12, Pkd1l3, Slc7a7, Clic4, Rasgef1a, Ska3, Abtb2, Plcd3, Fbxw13, Derl1,
## Utp20, Gsto2, Bmt2, Mrpl17, Prss12, As3mt, Atrnl1, Gmppa, Ypel2, Pard3bos3,
## Rgs7, Gm14051, Srsf12, Rims3, Kp
```

```
## PC_ 1
## Positive:  Nkg7, Ctsw, Ccl5, Klrd1, Il2rb, Klrk1, Gm2682, Klre1, Ms4a4b, Klrb1c
##     Ncr1, Cd7, Fcer1g, Samd3, Fyb, Il18rap, Klrc2, Eomes, Txk, Prkcq
##     Cst7, Ccr2, Ccr5, Anxa2, Osbpl3, Sh2d1a, 1700025G04Rik, Klra8, Lcp2, Prkch
## Negative:  Iglc2, Fcmr, Igkc, Ighm, Ifi30, Gm30211, Mzb1, Cd24a, Ms4a4c, Cd83
##     Bcar3, Ly6a, 2010309G21Rik, Blvrb, Il4i1, Iglc1, Rpsa, Ifi27l2a, Hck, Cybb
##     Plaur, Rps2, Bfsp2, Iglv3, Plac8, Il9r, Hes1, Kmo, Rps12, Cd200
## PC_ 2
```

```
## Positive:  Mki67, Pclaf, Top2a, Birc5, Ccna2, Neil3, Kif11, Rrm2, Kif15, Spc24
##     Uhrf1, Cdca3, Cks1b, Clspn, Ncapg2, Stmn1, Tyms, Cdk1, E2f8, Tpx2
##     Cenpm, Kntc1, Tcf19, Ncapg, Mpeg1, Grn, Nusap1, Stil, Basp1, Dtl
## Negative:  Ccl5, Nkg7, Klre1, Ncr1, Klrb1c, Klrk1, Il2rb, Samd3, Ctsw, Ms4a4b
##     Gm2682, Klrc2, Il18rap, Klra8, Stat4, Txk, Eomes, Ifitm10, Aoah, Klra7
##     Dok2, Klri2, Cst7, Klrb1f, Sytl3, Prf1, Prkcq, Itga2, Sh2d1a, Klra4
## PC_ 3
## Positive:  Pclaf, Mki67, Top2a, Stmn1, Ccna2, Birc5, Rrm2, Neil3, Kif11, Spc24
##     Cdk1, Cdca3, Uhrf1, Clspn, E2f8, Kif15, Tpx2, Tcf19, Kntc1, Stil
##     Esco2, Cenpm, Ckap2l, Ncapg, Tyms, Cks1b, Cdca8, Pbk, Ncaph, Ska1
## Negative:  Siglech, Ctsl, Cd300c, Grm8, Alox5ap, Mpeg1, Ccnd1, Csf2rb2, Klra17, Runx2
##     Gm21762, Ppfia4, Cox6a2, Ccr9, Lair1, Smim5, Tbc1d8, Fmnl2, Lrp8, Lag3
##     Ddr1, Upb1, P2ry14, Gm34680, Ramp1, Klk1, Pdzd4, Eldr, Clec9a, Dntt
## PC_ 4
## Positive:  Vim, Mki67, Birc5, Kif11, Pclaf, Ccna2, Scimp, Neil3, Tpx2, Cst3
##     Knl1, Cdk1, Rrm2, Cdca3, Top2a, Cdc25b, Itgb7, Hmgb2, Kif15, Txndc5
##     Stmn1, Hist1h2ae, E2f8, Ska1, Hist1h1b, H2afv, Csrp2, Clspn, Plk1, Cenpe
## Negative:  Rps2, Hsp90ab1, Mif, Srm, Npm1, Fcrl5, Nme2, Nme1, C1qbp, Mettl1
##     Ppp1r14b, Rplp1, Rplp0, Hspe1, Rps18, Ncl, Fbl, Eif4a1, Pa2g4, Atxn1
##     Rps12, Nhp2, Gnl3, Hspd1, Myc, Eif5a, Apex1, Bzw2, Pdia4, Rpl41
## PC_ 5
## Positive:  Cd3e, Cd3d, Cd3g, Themis, Trbc2, Bcl11b, Lat, Thy1, Trac, Lef1
##     Tcf7, Camk4, Itk, Sntb1, Cd247, Cd8b1, Cd27, Prkcq, Igfbp4, Gstp3
##     Slfn1, Dapl1, Cd28, Cd8a, Als2cl, Rgcc, Fyb, Ms4a6b, Cd4, Skap1
## Negative:  Ncr1, Klrb1c, Klrk1, Klre1, Tyrobp, Fcrl5, Klra8, Fcer1g, Klrc2, Klra4
##     Ccl5, Klrb1f, Klri2, Prf1, Osbpl3, Klra7, Pdia4, Tm6sf1, Itgax, Aoah
##     Il18rap, Adamts14, Itga2, Plac8, Mzb1, Sytl3, I830077J02Rik, Ccr2, S1pr5, Cd9
```

```
variable_genes <- setdiff(VariableFeatures(Seurat_Object_BM_selected), c("A12"))
Seurat_Object_BM_selected <- RunPCA(Seurat_Object_BM_selected, features = variable_genes)
```

```
## PC_ 1
## Positive:  Iglc2, H2-Eb1, H2-Aa, Ms4a1, Rnase6, Ctsh, Fcmr, Fth1, Ly6a, Iglc1
##     Cybb, March1, Pld4, Ifi30, Fcer2a, Kmo, Ncf1, Kynu, Scimp, Gm15987
##     Pltp, Bmyc, Gpr183, Plekhm3, Tyrobp, Gm42418, Ms4a4c, Cyth4, Plaur, Agbl1
## Negative:  Top2a, Pclaf, Mki67, Birc5, Rrm2, Stmn1, Ccna2, Spc24, Cdk1, Pbk
##     Cdca3, Kif11, Hist1h2ap, Nusap1, Hist1h2ae, Neil3, Cdca8, Asf1b, Hist1h1b, Cks1b
##     Esco2, Tuba1b, Lockd, Fbxo5, Cenpm, Tpx2, H2afx, Clspn, Kif15, Hist1h1a
## PC_ 2
## Positive:  Igkc, Gm30211, Cd24a, Ighm, Ms4a1, Iglc2, Fcmr, Kcnq5, Iglc1, Lrmp
##     Cplx2, Rag1, Tmem108, 2010309G21Rik, Gfra1, Fcer2a, Myl4, E2f2, Pde2a, Uchl1
##     Gm37065, Gm32569, Csrp2, Slamf7, Mgst1, Klhl14, Agbl1, Mzb1, Iglv1, Serpinb1a
## Negative:  Siglech, Mpeg1, Runx2, Ctsl, Fyb, Lair1, Bst2, Cd7, Tyrobp, Cd300c
##     Alox5ap, Fcer1g, Ppfia4, Ccnd1, Cox6a2, Smim5, Cd68, Upb1, Tbc1d8, Ptms
##     Ccr9, Ly6c2, Tex2, Pltp, Ramp1, Pacsin1, Grm8, Ctsb, Lgals1, Clec12a
## PC_ 3
## Positive:  Ighm, Rpl41, Rpl28, Ptma, Rpsa, Rplp1, Rpl15, Rps18, Rplp0, Ppia
##     Rps2, Rps12, Rpl36a, Rplp2, Irf8, Rpl14, Hsp90ab1, Fth1, Rnase6, Tcf4
##     Rps6, Mzb1, Npm1, Siglech, Pld4, Bst2, Hsp90b1, Cox6a2, H2-Aa, Rpl4
## Negative:  S100a8, Lcn2, Hp, S100a9, Ifitm6, Lyz2, Ngp, Mcemp1, Pygl, Anxa1
##     Cd177, Camp, Trem3, Mmp9, Gda, Ltf, Ly6g, Chil1, Clec4a2, Wfdc21
##     Chil3, Lrg1, Hdc, Igsf6, Gsr, Ltb4r1, Retnlg, Cd63, C3, Anxa3
## PC  4
```

```
## Positive:  Ighm, S100a9, Lcn2, S100a8, Cd24a, Ifitm6, Ngp, Cd177, Hp, Wfdc21
##     Camp, Ltf, Mcemp1, Alox5ap, Trem3, Ly6g, Lyz2, Mmp9, Gpx1, Chil1
##     Gda, Cst3, Pygl, Ifi27l2a, Cebpd, Chil3, Slpi, Mgst1, Lrg1, Mzb1
## Negative:  Nkg7, Ccl5, Il2rb, Ms4a4b, Gm2682, Klrb1c, Klre1, Klrk1, Ctsw, Skap1
##     Ncr1, Samd3, Ctla2a, Txk, Sh2d1a, Xcl1, Klra7, Eomes, Id2, Tcf7
##     Tox, Lck, Gzma, Dok2, Cd247, AW112010, Cst7, Ctla2b, Klrc2, Tbx21
## PC_ 5
## Positive:  Cenpf, Cep55, Pimreg, Aspm, Ube2c, Cdc20, Cdkn3, Sapcd2, Kif23, Pif1
##     Mis18bp1, Cenpe, Depdc1b, Plk1, Racgap1, Tpx2, Kif22, Nek2, Sgo2a, Parpbp
##     Kif18b, Nusap1, Ect2, Hmmr, Ckap2l, Kif11, Prc1, Gas2l3, Prr11, Kif2c
## Negative:  Klf1, Ermap, Car1, Hebp1, Gata1, Rhag, Gstm5, Rhd, Slc25a21, Aqp1
##     Ces2g, Ank1, Tspan33, Tal1, Smim1, Cldn13, Gypa, Hemgn, Sptb, Slc38a5
##     Icam4, Tspo2, Hba-a1, Hbb-bs, Add2, Hbb-bt, Spta1, Lmna, Cdc6, Trim10
```

Cluster the cells

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 18 PCs).

To cluster the cells, we next apply modularity optimization techniques 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. 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.

Cell Type annotation using SingleR

We used the ImmGen database in order to automatically annotate immune cells in our data.

```
# Download Immgen data
immgen.se <-ImmGenData()
```

```
## see ?celldex and browseVignettes('celldex') for documentation
```

```
## loading from cache
```

```
## see ?celldex and browseVignettes('celldex') for documentation
```

```
## loading from cache
```

```
sce_SP <- as.SingleCellExperiment(DietSeurat(Seurat_Object_SP_selected))
sce_SP
```

```
## class: SingleCellExperiment
## dim: 17662 6076
## metadata(0):
```

```
## assays(2): counts logcounts
## rownames(17662): Xkr4 Mrpl15 ... AC149090.1 A12
## rowData names(0):
## colnames(6076): AAACCTGAGACGACGT-1 AAACCTGAGCTAGTGG-1 ...
##   TTTGTCATCGTTTAGG-1 TTTGTCATCTTGCCGT-1
## colData names(13): orig.ident nCount_RNA ... hash.ID ident
## reducedDimNames(0):
## mainExpName: RNA
## altExpNames(1): HTO
```

```
sce_BM <- as.SingleCellExperiment(DietSeurat(Seurat_Object_BM_selected))
sce_BM
```

```
## class: SingleCellExperiment
## dim: 19884 13409
## metadata(0):
## assays(2): counts logcounts
## rownames(19884): Xkr4 Rp1 ... AC149090.1 A12
## rowData names(0):
## colnames(13409): AAACCTGAGCCCAATT-1 AAACCTGAGGAATTAC-1 ...
##   TTTGTCATCGCAGGCT-1 TTTGTCATCGCCAAAT-1
## colData names(13): orig.ident nCount_RNA ... hash.ID ident
## reducedDimNames(0):
## mainExpName: RNA
## altExpNames(1): HTO
```

Label main

To study the main immune cell types

SingleR es una herramienta bioinformática utilizada para realizar análisis de transferencia de conocimiento en datos de expresión génica, especialmente en el contexto de datos de secuenciación de ARN (RNA-seq) de célula única. Su objetivo principal es comparar perfiles de expresión génica de un conjunto de datos de prueba con perfiles de expresión de referencia de conjuntos de datos públicos o previamente caracterizados.

```
immgen_SP.main <- SingleR(test = sce_SP,assay.type.test = 1,ref = immgen.se,labels = immgen.se
$label.main)
table(immgen_SP.main$pruned.labels)
```

```
##
##      B cells B cells, pro           DC          ILC  Macrophages    Monocytes
##         5806            4           29            7            2            1
##     NK cells          NKT      T cells          Tgd
##           71            2           73            4
```

```
immgen_BM.main <- SingleR(test = sce_BM,assay.type.test = 1,ref = immgen.se,labels = immgen.se
$label.main)
table(immgen_BM.main$pruned.labels)
```

```
##
```

```
##      B cells B cells, pro    Basophils           DC  Fibroblasts          ILC
##        10388          928            6         1289            3           39
##  Macrophages    Monocytes  Neutrophils     NK cells          NKT   Stem cells
##            5           38          240          297           24           84
##      T cells          Tgd
##           27           17
```

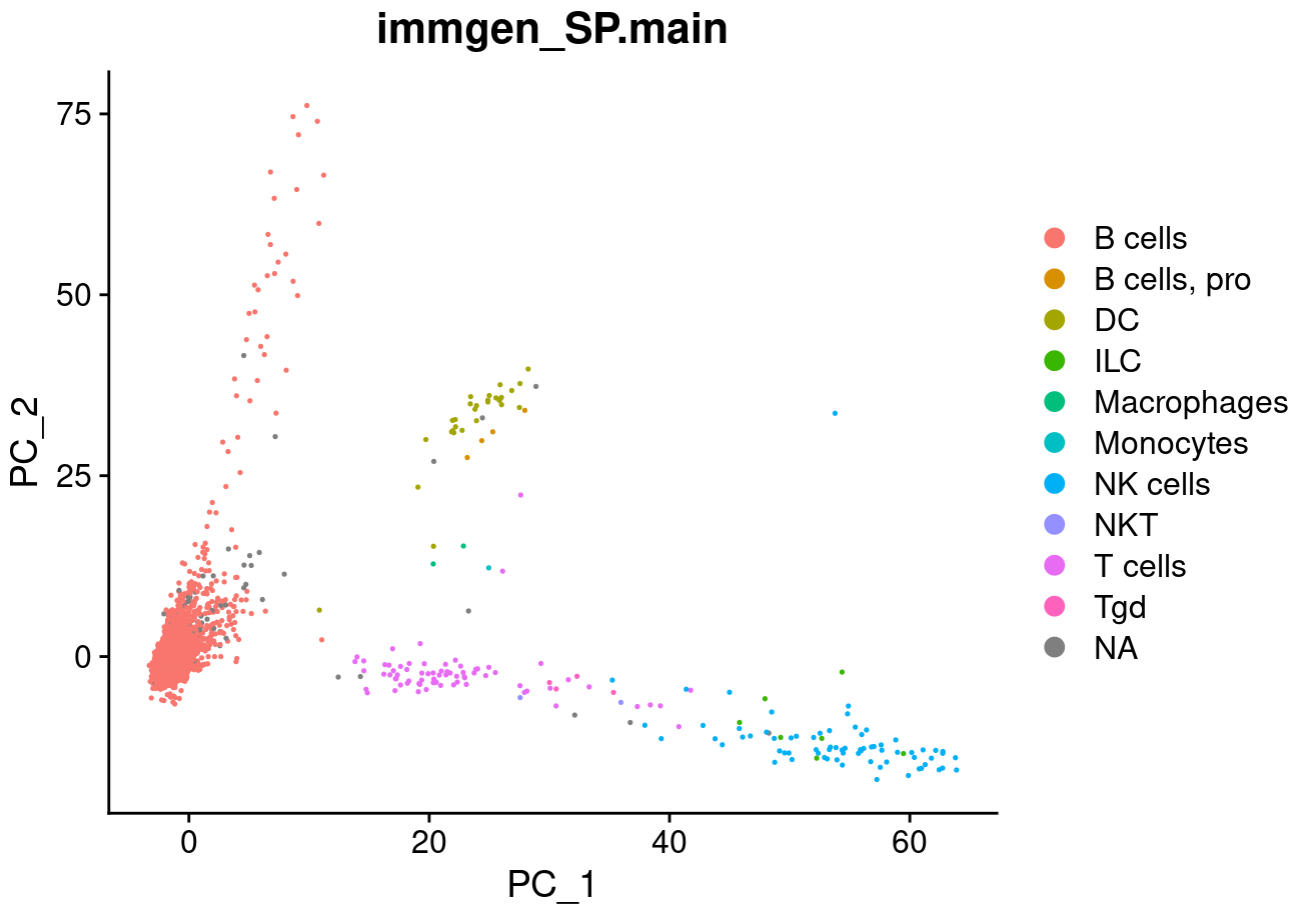
Add annotations to seurat object

```
Seurat_Object_SP_selected@meta.data$immgen_SP.main <- immgen_SP.main$pruned.labels
```

```
Seurat_Object_BM_selected@meta.data$immgen_BM.main <- immgen_BM.main$pruned.labels
```

See where are the non-B cells in the umap plot:

```
DimPlot(Seurat_Object_SP_selected, reduction = "pca", group.by = "immgen_SP.main")
```

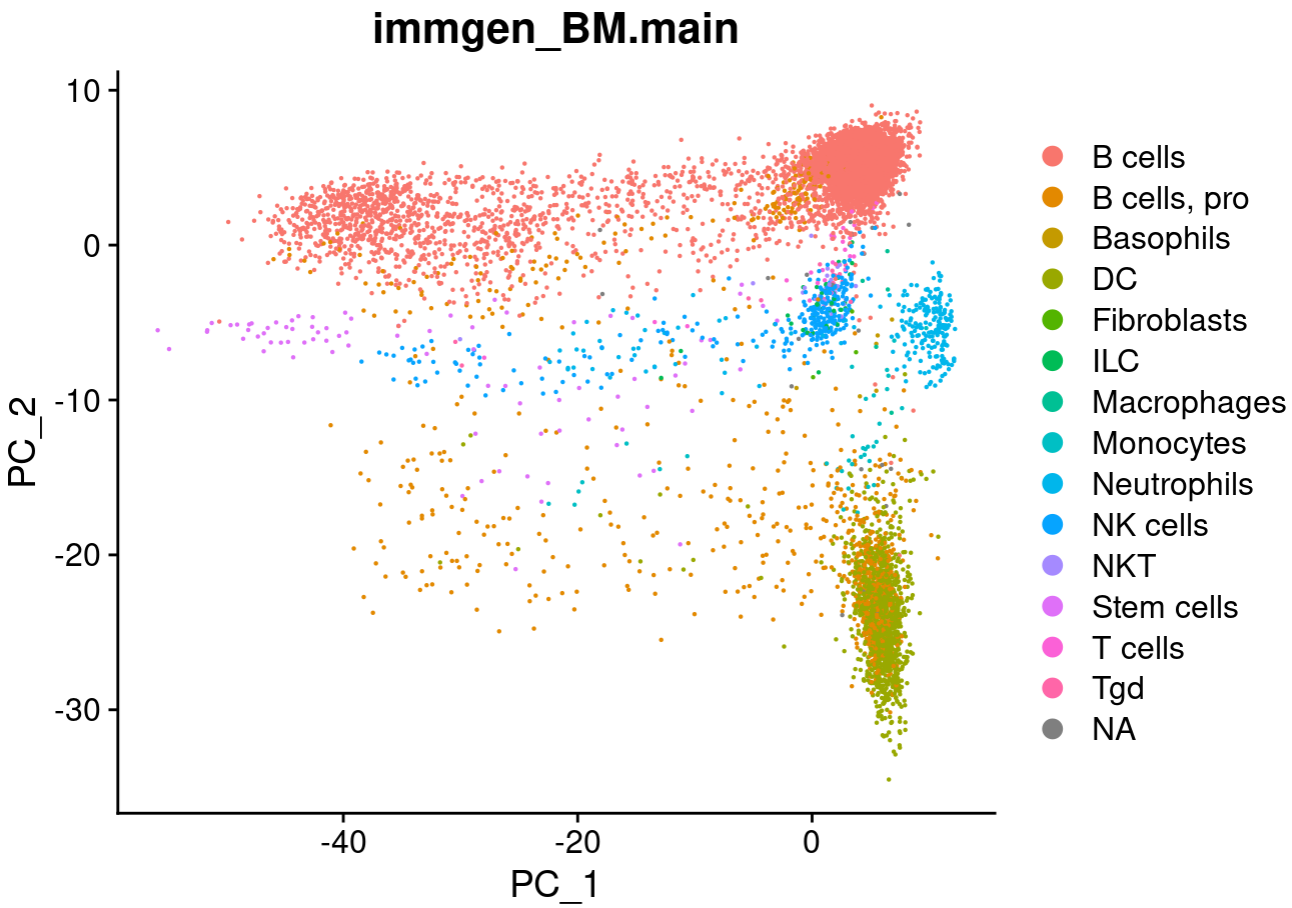



```
table(Seurat_Object_SP_selected$immgen_SP.main)
```

```
##
##      B cells B cells, pro           DC          ILC  Macrophages    Monocytes
##         5806            4           29            7            2            1
##     NK cells          NKT      T cells          Tgd
```

```
##           71            2           73            4
```

```
DimPlot(Seurat_Object_BM_selected, reduction = "pca", group.by = "immgen_BM.main")
```



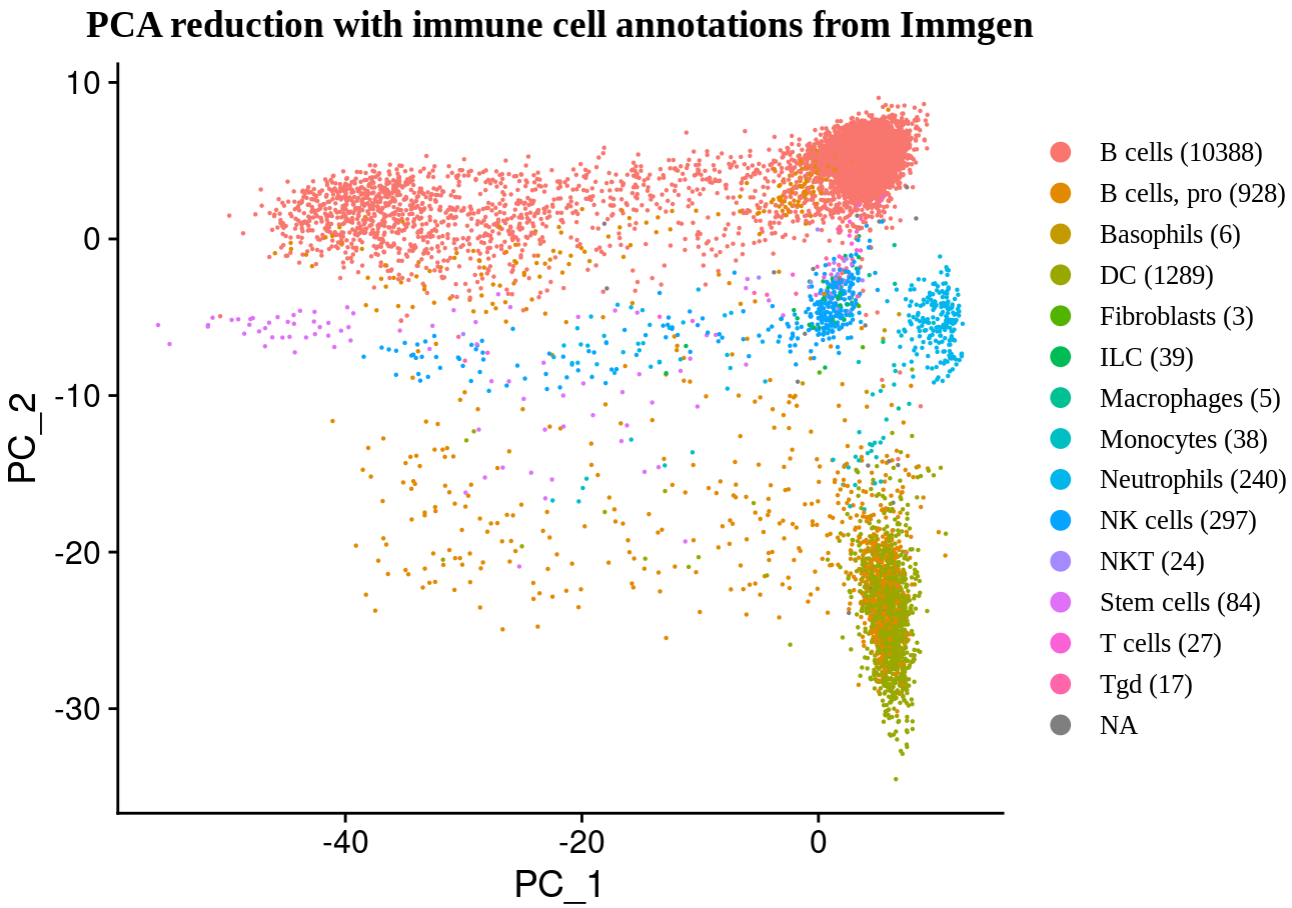

```
table(Seurat_Object_BM_selected$immgen_BM.main)
```

```
##
##      B cells B cells, pro    Basophils           DC  Fibroblasts          ILC
##        10388          928            6         1289            3           39
##  Macrophages    Monocytes  Neutrophils     NK cells          NKT   Stem cells
##            5           38          240          297           24           84
##      T cells          Tgd
##           27           17
```

Calculate the number of cells per cell type

```
# Calculate the number of cells per group
cell_counts <- table(Seurat_Object_BM_selected$immgen_BM.main)

# Create labels by combining the group name and the number of cells
new_labels <- paste0(names(cell_counts), " (", cell_counts, ")")
names(new_labels) <- names(cell_counts)  # Assign the original names as names for the new labe
ls
```

```
# Create the DimPlot with automatic colors, title, and customized legend
DimPlot(
    Seurat_Object_BM_selected,
    reduction = "pca",
    group.by = "immgen_BM.main"
) +
    scale_color_discrete(labels = new_labels) +  # Labels with cell counts and automatic color
s
    ggtitle("PCA reduction with immune cell annotations from Immgen") +  # Title
    theme(
        legend.text = element_text(size = 10, family = "Times New Roman"),  # Legend font
        legend.title = element_text(size = 12, family = "Times New Roman"), # Legend title fon
t
        plot.title = element_text(size = 14, family = "Times New Roman")    # Title customizat
ion
    )
```



Asign cell type to cell identity

```
Idents(Seurat_Object_SP_selected) <- Seurat_Object_SP_selected$immgen_SP.main
```

```
Idents(Seurat_Object_BM_selected) <- Seurat_Object_BM_selected$immgen_BM.main
```

Select only B cells

```
Seurat_Object_SP_selected_Bcells<- subset(Seurat_Object_SP_selected, idents = c("B cells"))
Seurat_Object_SP_selected_Bcells
```

```
## An object of class Seurat
## 17669 features across 5806 samples within 2 assays
## Active assay: RNA (17662 features, 3000 variable features)
##  3 layers present: counts, data, scale.data
##  1 other assay present: HTO
##  1 dimensional reduction calculated: pca
```

```
Seurat_Object_BM_selected_Bcells<- subset(Seurat_Object_BM_selected, idents = c("B cells", "B
cells, pro"))
Seurat_Object_BM_selected_Bcells
```

```
## An object of class Seurat
## 19891 features across 11316 samples within 2 assays
## Active assay: RNA (19884 features, 3000 variable features)
##  3 layers present: counts, data, scale.data
##  1 other assay present: HTO
##  1 dimensional reduction calculated: pca
```

Redo Scaling, PCA and clustering for B cells

Identifucation of highly variable features of B cells

```
Seurat_Object_SP_selected_Bcells <- FindVariableFeatures(Seurat_Object_SP_selected_Bcells, sel
ection.method = "vst", nfeatures = 3000)
```

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

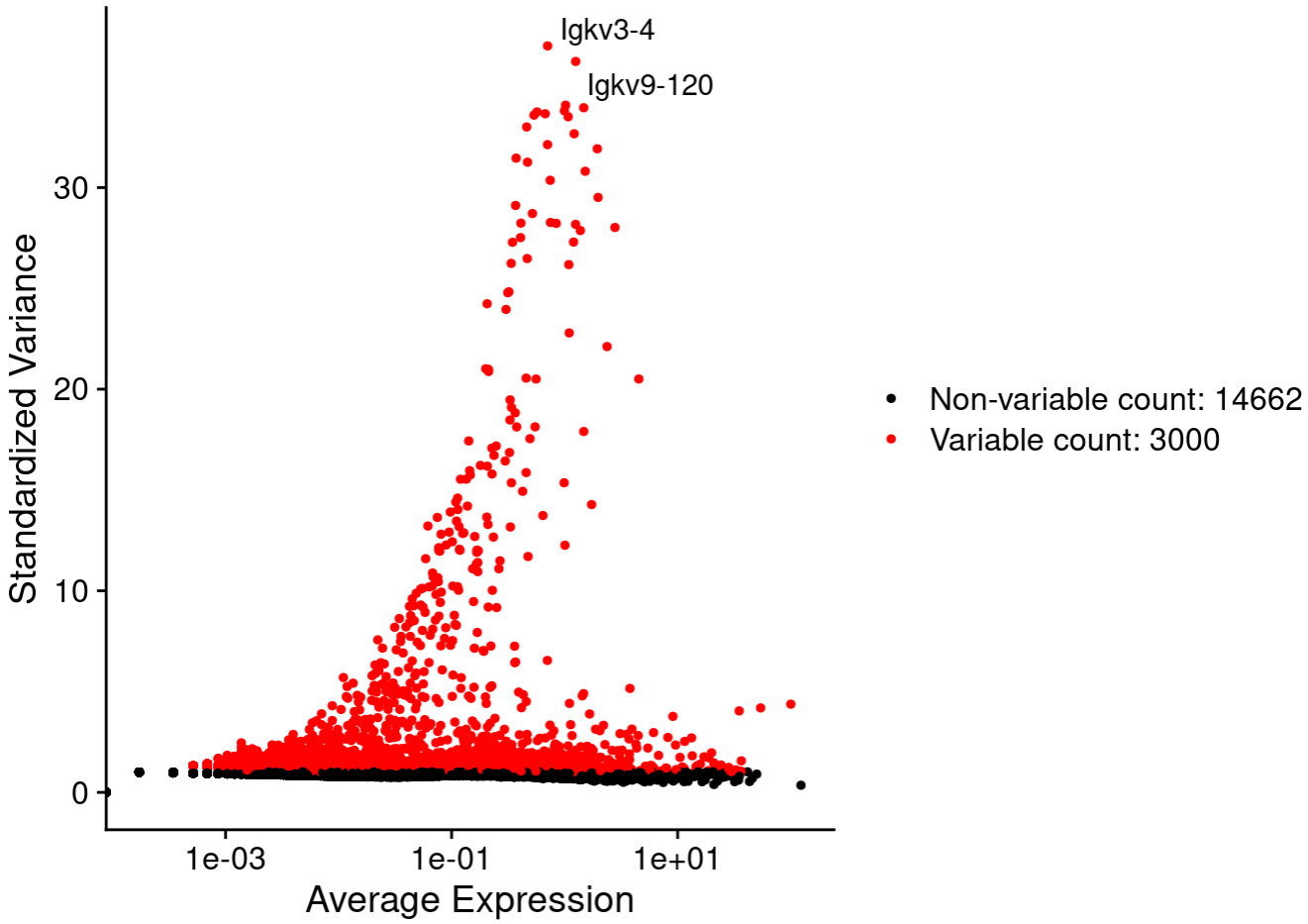
```
# Identify the 10 most highly variable genes
top10 <- head(VariableFeatures(Seurat_Object_SP_selected_Bcells), 10)

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

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

```
plot2
```

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

```
## Warning: ggrepel: 8 unlabeled data points (too many overlaps). Consider
## increasing max.overlaps
```




```
Seurat_Object_BM_selected_Bcells <- FindVariableFeatures(Seurat_Object_BM_selected_Bcells, sel
ection.method = "vst", nfeatures = 3000)
```

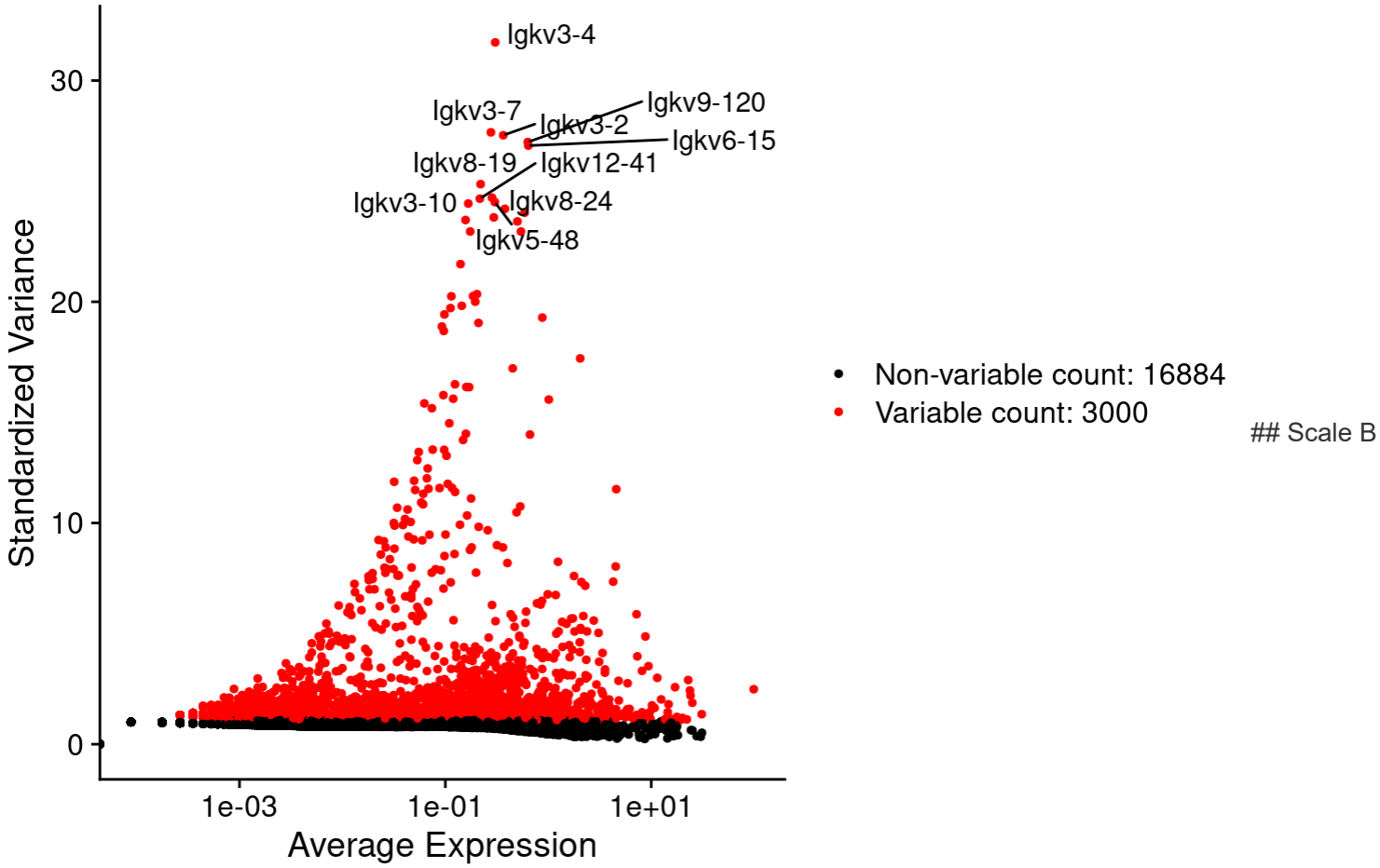
```
## Finding variable features for layer counts
```

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

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

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

```
plot2
```

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



Scale B

cells

```
variable_genes_SP <- setdiff(VariableFeatures(Seurat_Object_SP_selected_Bcells), c("A12"))
Seurat_Object_SP_selected_Bcells <- ScaleData(Seurat_Object_SP_selected_Bcells, features = row
.names(Seurat_Object_SP_selected_Bcells))
```

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

```
## Warning: Different features in new layer data than already exists for
## scale.data
```

```
s.genes <- cc.genes$s.genes
g2m.genes <- cc.genes$g2m.genes

Seurat_Object_BM_selected_Bcells <- CellCycleScoring(Seurat_Object_BM_selected_Bcells, s.featu
res = s.genes, g2m.features = g2m.genes, set.ident = TRUE)
```

```
## Warning: The following features are not present in the object: MCM5, PCNA,
## TYMS, FEN1, MCM2, MCM4, RRM1, UNG, GINS2, MCM6, CDCA7, DTL, PRIM1, UHRF1,
## MLF1IP, HELLS, RFC2, RPA2, NASP, RAD51AP1, GMNN, WDR76, SLBP, CCNE2, UBR7,
## POLD3, MSH2, ATAD2, RAD51, RRM2, CDC45, CDC6, EXO1, TIPIN, DSCC1, BLM,
## CASP8AP2, USP1, CLSPN, POLA1, CHAF1B, BRIP1, E2F8, not searching for symbol
## synonyms
```

```
## Warning: The following features are not present in the object: HMGB2, CDK1,
## NUSAP1, UBE2C, BIRC5, TPX2, TOP2A, NDC80, CKS2, NUF2, CKS1B, MKI67, TMPO,
## CENPF, TACC3, FAM64A, SMC4, CCNB2, CKAP2L, CKAP2, AURKB, BUB1, KIF11, ANP32E,
## TUBB4B, GTSE1, KIF20B, HJURP, CDCA3, HN1, CDC20, TTK, CDC25C, KIF2C, RANGAP1,
## NCAPD2, DLGAP5, CDCA2, CDCA8, ECT2, KIF23, HMMR, AURKA, PSRC1, ANLN, LBR,
## CKAP5, CENPE, CTCF, NEK2, G2E3, GAS2L3, CBX5, CENPA, not searching for symbol
## synonyms
```

```
## Warning in AddModuleScore(object = object, features = features, name = name, :
## Could not find enough features in the object from the following feature lists:
## S.Score Attempting to match case...Could not find enough features in the object
## from the following feature lists: G2M.Score Attempting to match case...
```

```
Seurat_Object_BM_selected_Bcells$CC.Difference <- Seurat_Object_BM_selected_Bcells$S.Score - S
eurat_Object_BM_selected_Bcells$G2M.Score


variable_genes_BM <- setdiff(VariableFeatures(Seurat_Object_BM_selected_Bcells), c("A12"))
Seurat_Object_BM_selected_Bcells <- ScaleData(Seurat_Object_BM_selected_Bcells, vars.to.regres
s = c("CC.Difference"), features = row.names(Seurat_Object_BM_selected_Bcells))
```

```
## Regressing out CC.Difference
```

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

```
## Warning: Different features in new layer data than already exists for
## scale.data
```

PCA of B cells

```
Seurat_Object_SP_selected_Bcells <- RunPCA(Seurat_Object_SP_selected_Bcells, features = variab
le_genes_SP) #
```

```
## PC_ 1
## Positive:  Fcrl5, Atxn1, Dtx1, Marcks, Myof, I830077J02Rik, Rgs10, Pdia4, Cd9, Dph5
##     Nebl, Plac8, Zc3h12c, S1pr3, Asb2, Cr2, Sema7a, Ackr3, Nedd4, Cd1d1
##     Arhgap24, Pik3r4, Rsu1, Grn, Rplp1, Tbc1d9, Rplp0, Mfhas1, Camkmt, Plxnc1
## Negative:  Klf2, Fcer2a, Cd55, Bach2, Jund, Tsc22d3, Vim, S100a10, Vpreb3, Emp3
##     Serpinb1a, Cmah, Ptma, Ms4a4c, Fcmr, Rasgrp2, Ccr7, Itgb7, Fry, Rasa3
##     St6gal1, Ier2, Pgap1, Rflnb, Id3, Cpm, Txndc5, Cd200, Klf3, Tuba1a
## PC_ 2
## Positive:  Cr2, I830077J02Rik, Myof, Asb2, Tm6sf1, Dtx1, Cd1d1, St6galnac3, S1pr3, Maml2
##     Il9r, Fcrl5, Immp2l, Rpl39, Camkmt, Rsu1, Rpl13, Atxn1, Pik3r4, Pde4d
##     Ptpn14, Rpl27a, Tbc1d9, Ackr3, Rplp1, Nebl, Pdia4, A930005H10Rik, Nedd9, Trps1
## Negative:  Mki67, Pclaf, Top2a, Ptma, Stmn1, Neil3, Kif11, Birc5, Ccna2, Rrm2
##     Kif15, Uhrf1, Cdca3, Spc24, E2f8, Cks1b, Cdk1, Clspn, Tpx2, Jchain
##     Ncapg, Basp1, Cenpm, Stil, Ckap2l, Aicda, Kntc1, Ncaph, Ppia, Pbk
## PC_ 3
```

```
## Positive:  Ppia, Hsp90ab1, Srm, Mif, Rps2, Nme1, Mettl1, Nme2, C1qbp, Fbl
##     Eif5a, Ppp1r14b, Ncl, Ptma, Apex1, Eif4a1, Nhp2, Hspe1, Ranbp1, Psme2
##     Pa2g4, Timm8a1, Npm1, Gnl3, Ppan, Nolc1, Hspd1, Atp5g1, Ppa1, Gar1
## Negative:  Mki67, Pclaf, Top2a, Kif11, Birc5, Ccna2, Neil3, Jchain, Tpx2, Cdca3
##     Nusap1, Aicda, Mybl1, Zbtb20, Kif15, Cdk1, Rrm2, Ckap2l, Knl1, E2f8
##     S1pr2, Clspn, Basp1, Ncapg, Nuggc, Cenpe, Kif23, Anxa2, Pbk, Spc24
## PC_ 4
## Positive:  Actn1, Gm31718, Pstpip2, Dnm3, Apoe, Fcer1g, Bhlhe41, Cpd, Rftn1, Zbtb32
##     Ccdc28b, Lgals1, Iglc1, A630001O12Rik, Ahnak, Sox5, Cblc, Zeb2, S100a10, A430093F15Rik
##     Ahr, Cd300lf, Rassf4, Aldh3b1, Fcmr, Apba1, Lysmd2, Plac8, Ighg2b, Nfatc1
## Negative:  Cr2, Pxdc1, Rpl13, Rpsa, Pclaf, Tmem108, Ccr6, Tespa1, Fcer2a, Neil3
##     Pakap.1, Asb2, Dtx1, Ell3, Tyms, Ffar2, Hes1, Rrm2, Ccna2, E2f8
##     Spc24, I830077J02Rik, Uhrf1, Nedd9, Clspn, Cdk1, Rps2, Pkib, Birc5, Il9r
## PC_ 5
## Positive:  Rpsa, Rpl13, Rpl41, Rpl37, Rpl39, Rpl35, Rpl32, Rps26, Eef1a1, Rpl27a
##     Rplp0, Rps27a, Rpl28, Rps18, Rps11, Rpl10a, Rpl29, Rps12, Rplp1, Rpl36a
##     Rps2, Rpl3, Rpl15, Rpl7a, Rpl14, Rack1, Eef1b2, Rps6, Rpl4, Apoe
## Negative:  Nedd9, Ptprj, Tmem131l, Agbl1, Ifi30, Hck, Nedd4, Wfdc21, Lyst, Fgd2
##     Malt1, Dennd4a, Mctp2, Ppp1r14b, Vpreb3, Hnrnpa2b1, Lynx1, Hivep3, Hspa5, Wdr43
##     Mybbp1a, Dusp2, Zdhhc14, Cd180, Srm, Klhl14, Hnrnpu, Cplx2, Larp1, Nolc1
```

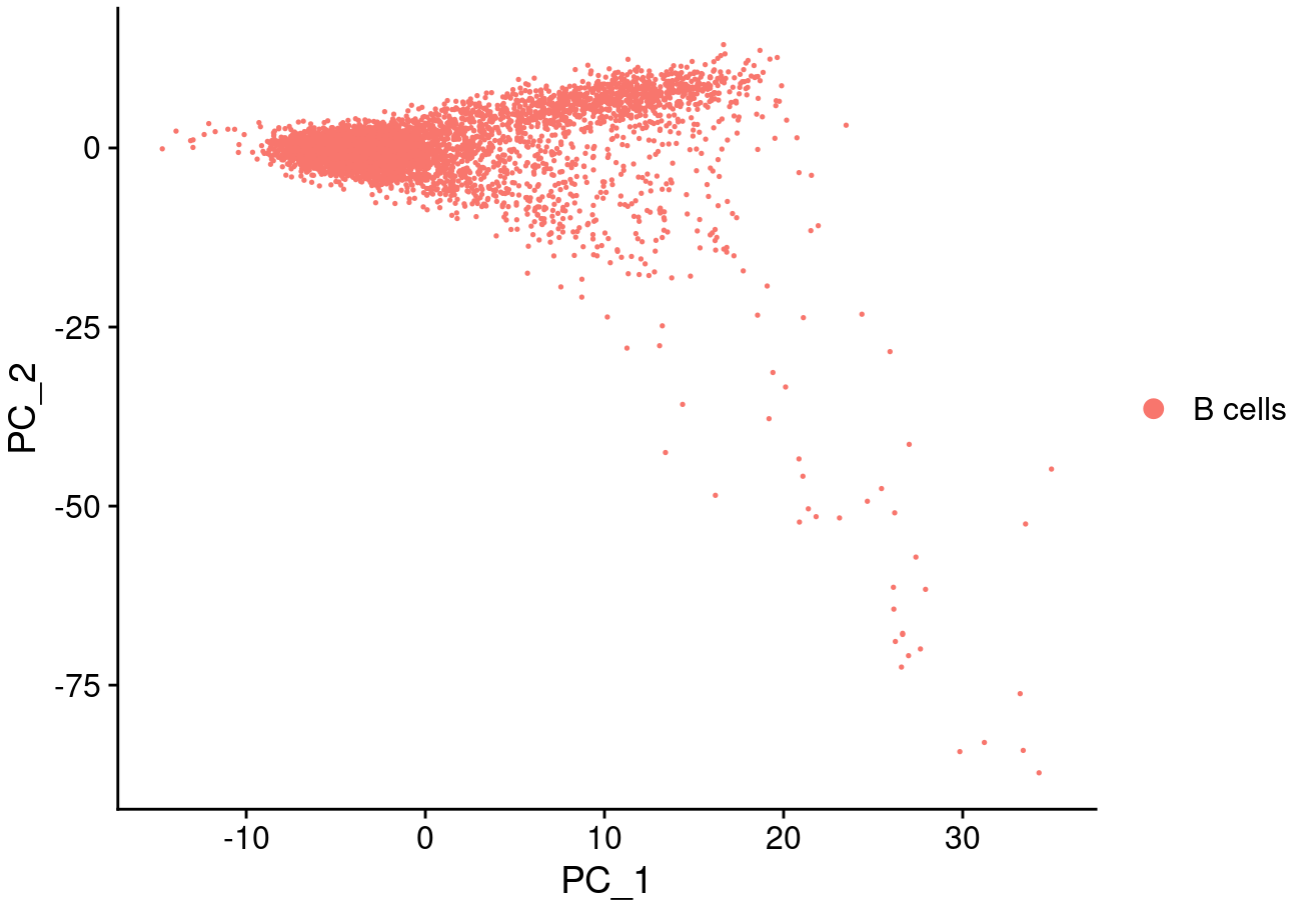
```
Seurat_Object_BM_selected_Bcells <- RunPCA(Seurat_Object_BM_selected_Bcells, features = variab
le_genes_BM)
```

```
## PC_ 1
## Positive:  Ly6d, Cd74, Iglc2, Cd52, Ms4a1, H2-Eb1, H2-Aa, Iglc1, Ighm, Fcer2a
##     March1, Gm15987, Agbl1, Gimap3, Ifi30, 2010309G21Rik, Gpr183, Ms4a4c, Ly6a, Inpp4b
##     Gimap4, Gimap7, Gm43388, Pglyrp1, Iglv1, Apoe, Hck, Gm42418, Iglv3, Rnase6
## Negative:  Pclaf, Top2a, Mki67, Rrm2, Birc5, Stmn1, Ccna2, Spc24, Clspn, Cdca3
##     Dut, Cdk1, Kif11, Pbk, Asf1b, Cdca8, Nusap1, Tuba1b, Lig1, Uhrf1
##     Neil3, Hist1h2ap, Hist1h2ae, Tyms, Ube2c, Esco2, Fbxo5, Hist1h1b, Tpx2, Lockd
## PC_ 2
## Positive:  Siglech, Mpeg1, Cd7, Runx2, Fyb, Fcer1g, Ctsl, Lair1, Alox5ap, Ppfia4
##     Smim5, Cd300c, Ccnd1, Cox6a2, Tbc1d8, Bst2, Tyrobp, Tex2, Ccr9, Upb1
##     Gm34680, Csf2rb2, Cd68, Klrd1, Ly6c2, Paqr5, Ptms, Pacsin1, Fmnl2, Ppm1h
## Negative:  Cd24a, Gm30211, Kcnq5, Lrmp, Cplx2, Csrp2, Ighm, Ptma, E2f2, Tmem108
##     Arntl, Pde2a, Slamf7, Rrm2, Elof1, Gfra1, Hist1h1a, Ezh2, Hist1h1e, Hist1h1b
##     Zfpm1, Lgals9, Dck, Hist1h2ae, Ccna2, Rsph1, Esco2, Uhrf1, Hist1h2ap, Pclaf
## PC_ 3
## Positive:  Rag1, Myb, Cplx2, Il7r, Gfra1, Gm32569, Gm37065, 2610307P16Rik, Erg, Slc12a3
##     Lrmp, Cd93, Prep, Vpreb1, Pde2a, Lgals9, Uchl1, Bst1, Igll1, Cd24a
##     Lef1, Rag2, Slamf7, Il2ra, Arntl, Ptma, Plxdc1, Capsl, Gm4258, Elof1
## Negative:  H2-Eb1, H2-Aa, Cd74, Cd52, Fcer2a, Iglc2, Rpsa, Rpl13, Ly6a, Ms4a1
##     Rplp1, March1, Gimap3, Rps2, Ctsh, Rps20, Rps8, Ms4a4c, Gimap4, Gimap7
##     Gpr183, Rpl39, Rplp2, Pglyrp1, Gm15987, Apoe, Rpl36a, Rps18, Rpl14, Cr2
## PC_ 4
## Positive:  Hist1h2ab, Hist1h2ap, Hist1h2ak, Ly6d, Neil3, Esco2, Hist1h1b, Hist1h1e, Hist1h1
a, Hist1h2ae
##     Hist1h1d, E2f8, Mxd3, Gm20628, Nusap1, Hist2h2ac, Pbk, Ighm, Kif11, E2f7
##     Agbl1, Hist1h3c, Hist1h2bb, Ccnf, Hist1h2bj, Mctp2, Spc25, Hist1h2af, Hist1h1c, Fbxo5
## Negative:  Vpreb2, Ifitm2, Vpreb1, Igll1, Myl10, Dntt, Arpp21, Lef1, Impdh1, Rps2
##     Gm30948, Rps27a, Rpl13, Rpl41, Rpl32, Eef1a1, Rps18, Rps26, Anxa1, Grb7
##     Ttll11, Rpl39, Rps8, Rpl10a, Egfl7, Drc7, Rplp2, Rpl15, Rplp0, Rpsa
```

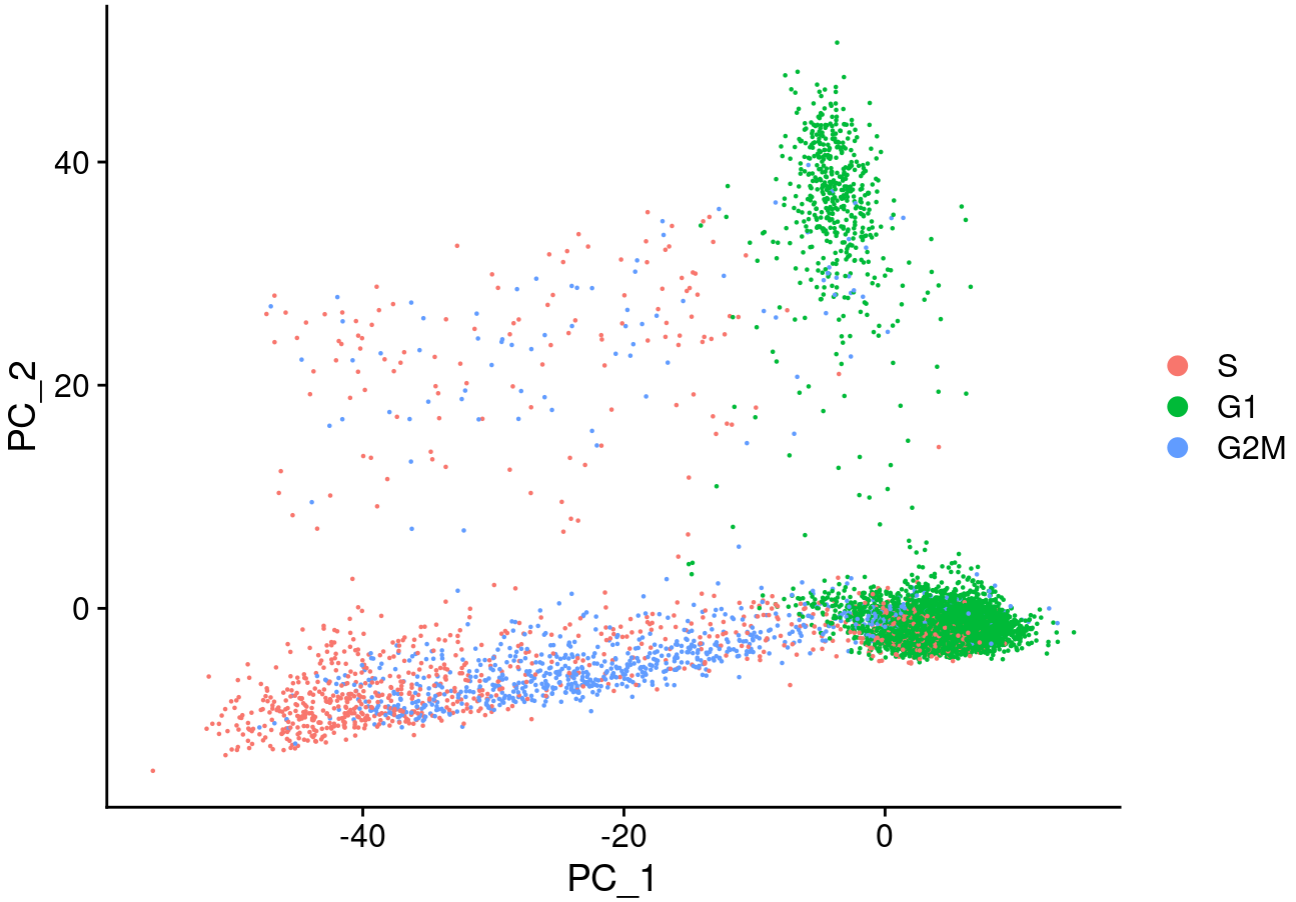
```
## PC_ 5
## Positive:  Cd24a, Ly6d, Hck, Ccnd2, Hsp90b1, Eif5a, Ifi30, Ighm, Wnt10a, Ptma
##     Mif, Nme1, Hsp90ab1, Klhl14, Nhp2, Rbm3, Ppia, Atp5g1, Srm, Agbl1
##     Ppp1r14b, Slc25a5, Them6, Atp5g3, C1qbp, Hspe1, Grn, Srsf3, Iglc1, Hnrnpa2b1
## Negative:  Arpp21, Dntt, Gm30948, Vpreb2, Vpreb1, Myl10, Igll1, Grb7, Selenom, Erg
##     Impdh1, Sell, Ifitm2, Lef1, Utrn, H2-Eb1, Frmd6, Gimap4, A630023P12Rik, Eng
##     Lax1, Ttll11, Csgalnact1, Auts2, Mctp1, Heyl, Drc7, March1, Tmem91, H2-Aa
```

Representation:

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




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



Determine the 'dimensionality' of the dataset and obtain evidence to choose number of principal components

```
#Seurat_Object_SP_selected_Bcells <- JackStraw(Seurat_Object_SP_selected_Bcells, num.replicate
 = 100, dims = 40)
```

```
#Seurat_Object_BM_selected_Bcells <- JackStraw(Seurat_Object_BM_selected_Bcells, num.replicate
 = 100, dims = 40)
```

```
#Seurat_Object_SP_selected_Bcells <- ScoreJackStraw(Seurat_Object_SP_selected_Bcells, dims = 1
:40)
```

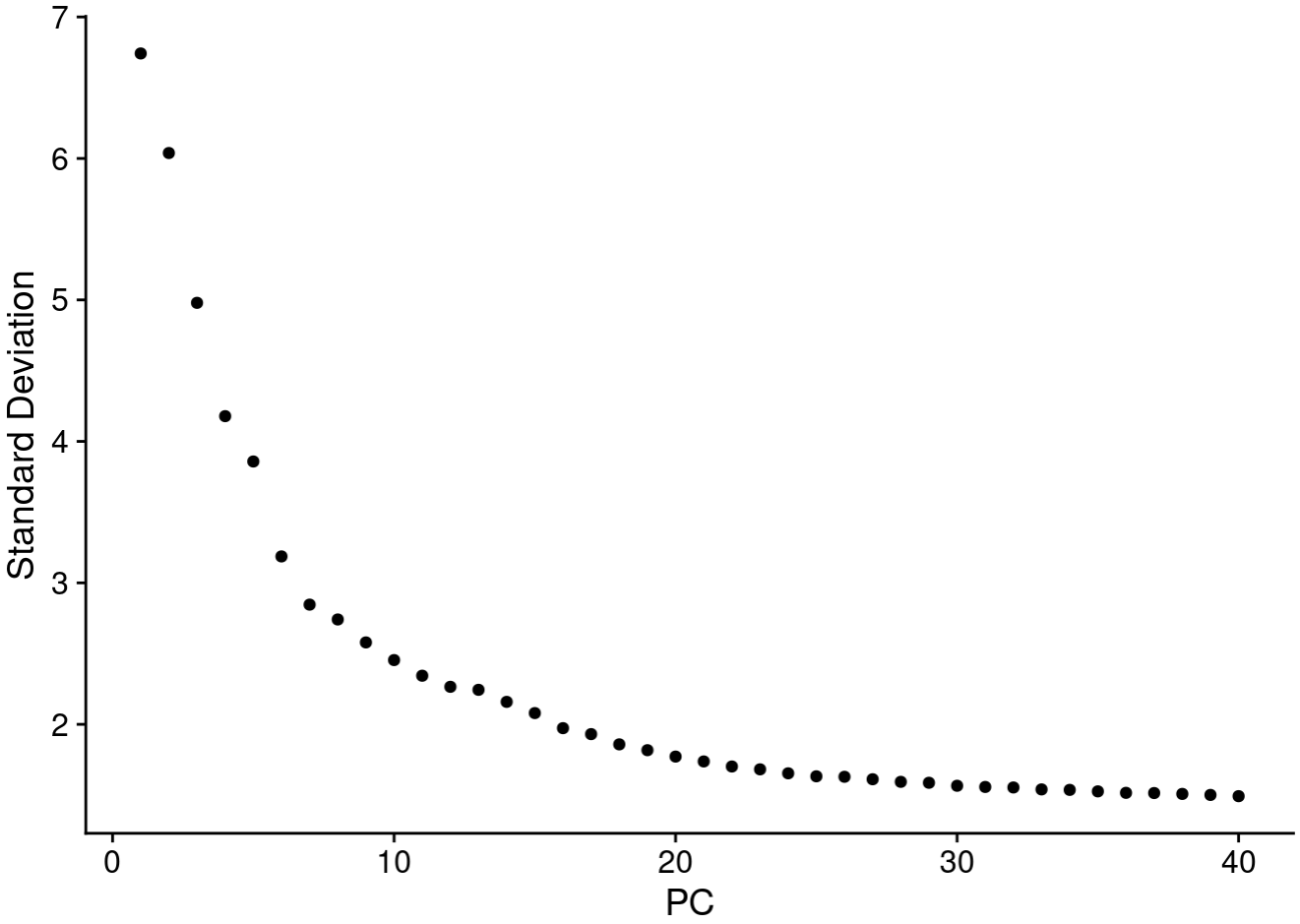
```
#Seurat_Object_BM_selected_Bcells <- ScoreJackStraw(Seurat_Object_BM_selected_Bcells, dims = 1
:40)
```

```
#JackStrawPlot(Seurat_Object_SP_selected_Bcells, dims = 1:40)
```

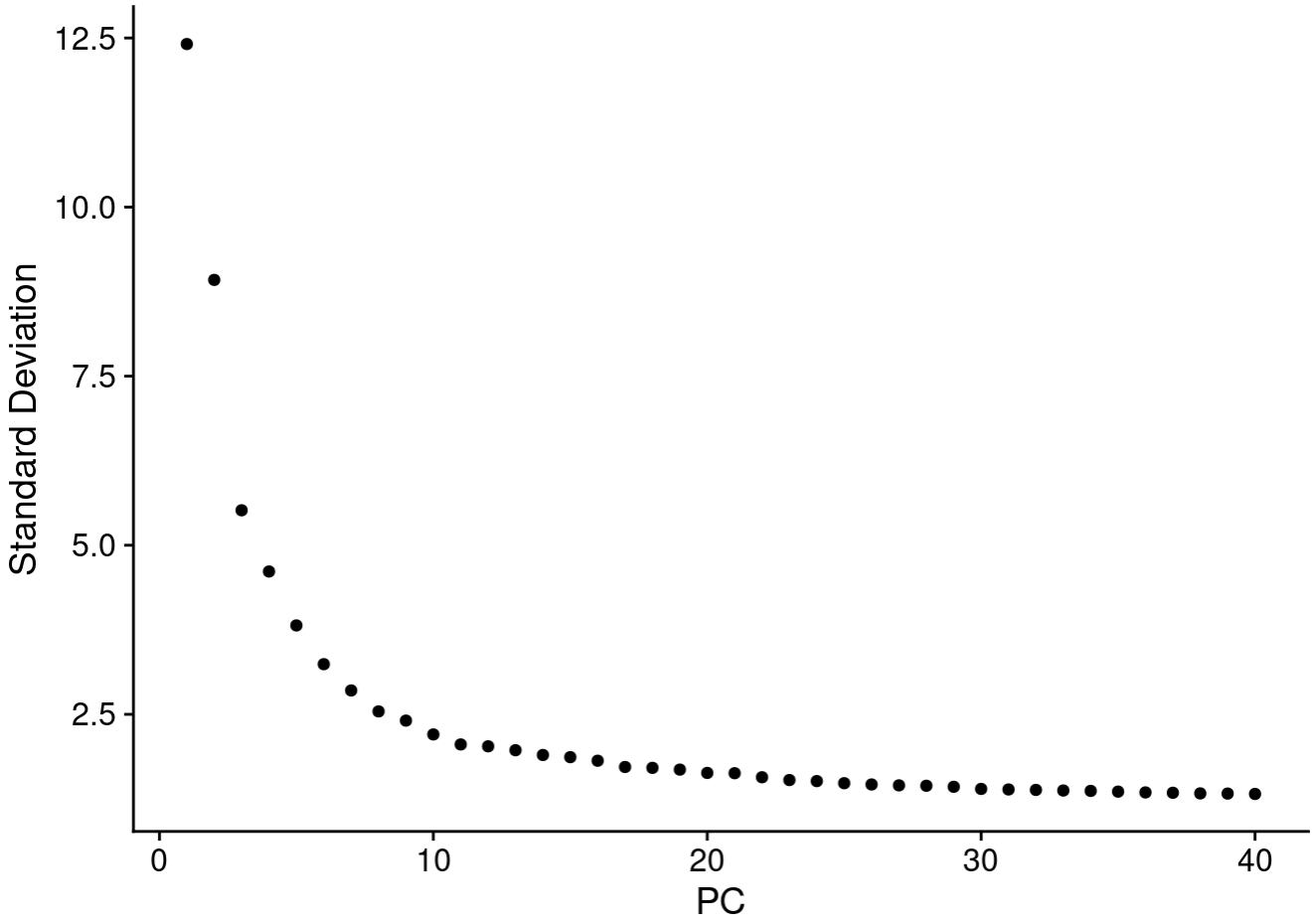
```
#JackStrawPlot(Seurat_Object_BM_selected_Bcells, dims = 1:40) # No has corrido esto
```

An heuristic method generates an 'Elbow plot': a ranking of principle components based on the percentage of variance explained by each one (ElbowPlot() function).

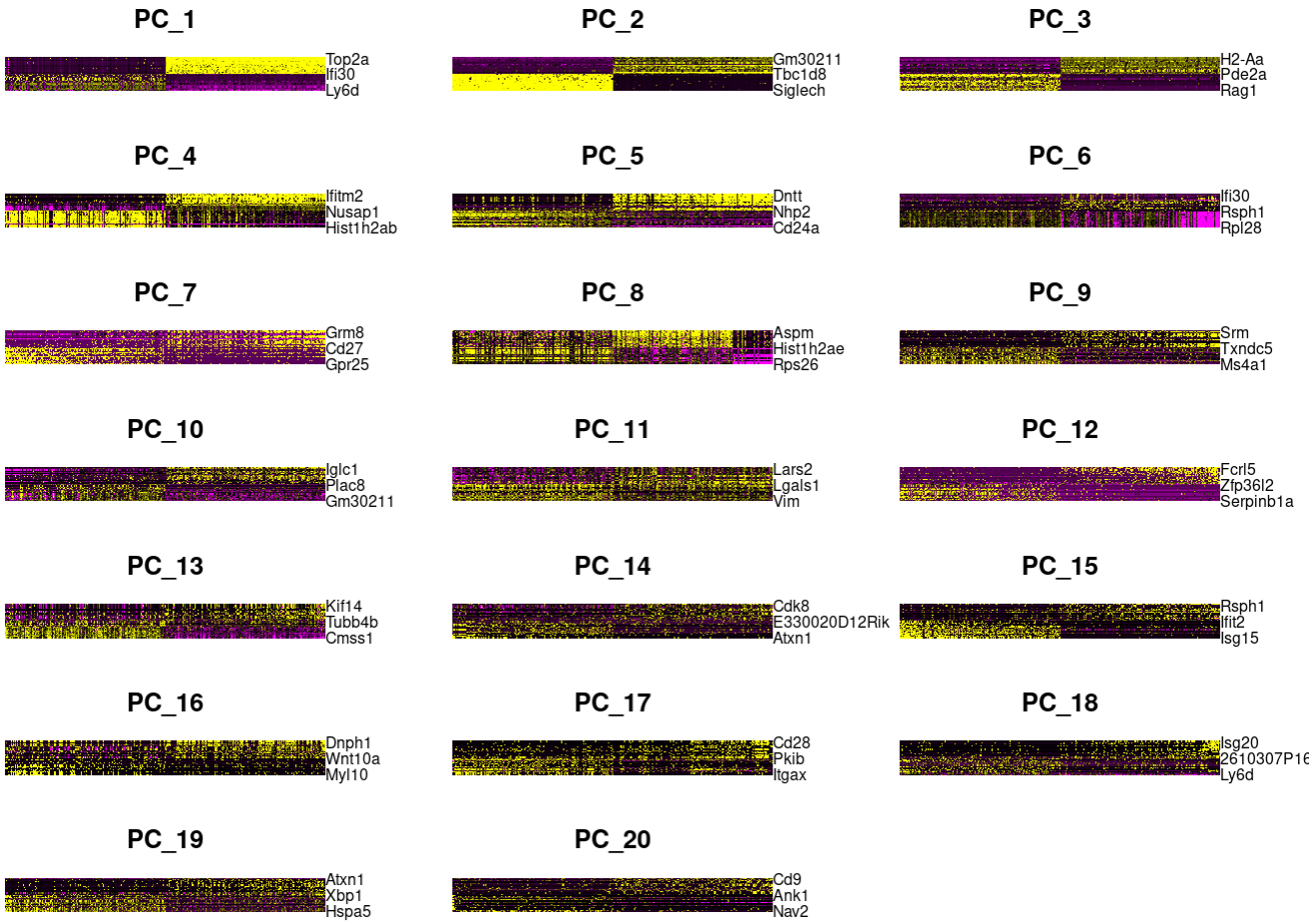
```
ElbowPlot(Seurat_Object_SP_selected_Bcells, ndims=40)
```



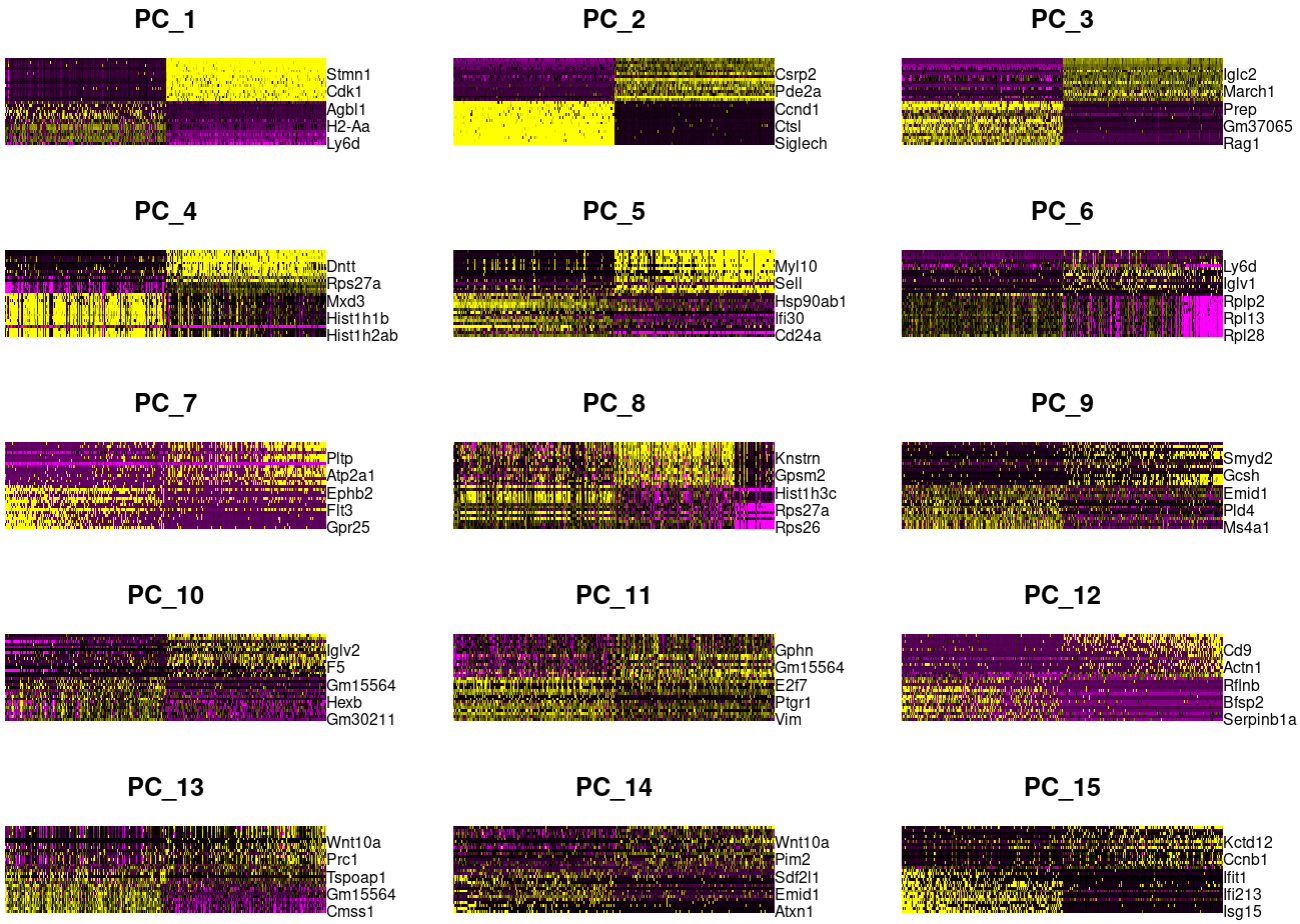

```
ElbowPlot(Seurat_Object_BM_selected_Bcells, ndims=40)
```




```
DimHeatmap(Seurat_Object_BM_selected_Bcells, dims = 1:20, cells = 500, balanced = TRUE)
```




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



Cluster the cells

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.

To cluster the cells, we next apply modularity optimization techniques 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.

B cells clusters

```
Seurat_Object_SP_selected_Bcells <- FindNeighbors(Seurat_Object_SP_selected_Bcells, dims = 1:2
0)
```

```
## Computing nearest neighbor graph
```

```
## Computing SNN
```

```
Seurat_Object_SP_selected_Bcells <- FindClusters(Seurat_Object_SP_selected_Bcells, resolution
= 0.6)
```

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 5806
## Number of edges: 193034
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.7929
## Number of communities: 11
## Elapsed time: 0 seconds
```

B cells clusters

```
Seurat_Object_BM_selected_Bcells <- FindNeighbors(Seurat_Object_BM_selected_Bcells, dims = 1:15)
```

```
## Computing nearest neighbor graph
```

```
## Computing SNN
```

```
Seurat_Object_BM_selected_Bcells <- FindClusters(Seurat_Object_BM_selected_Bcells, resolution = 0.9)
```

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 11316
## Number of edges: 369215
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.8359
## Number of communities: 18
## Elapsed time: 1 seconds
```

Run non-linear dimensional reduction (UMAP/tSNE)

```
Seurat_Object_SP_selected_Bcells <- RunUMAP(Seurat_Object_SP_selected_Bcells, dims = 1:20)
```

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

```
## 13:30:26 UMAP embedding parameters a = 0.9922 b = 1.112
```

```
## 13:30:26 Read 5806 rows and found 20 numeric columns
```

```
## 13:30:26 Using Annoy for neighbor search, n_neighbors = 30
```

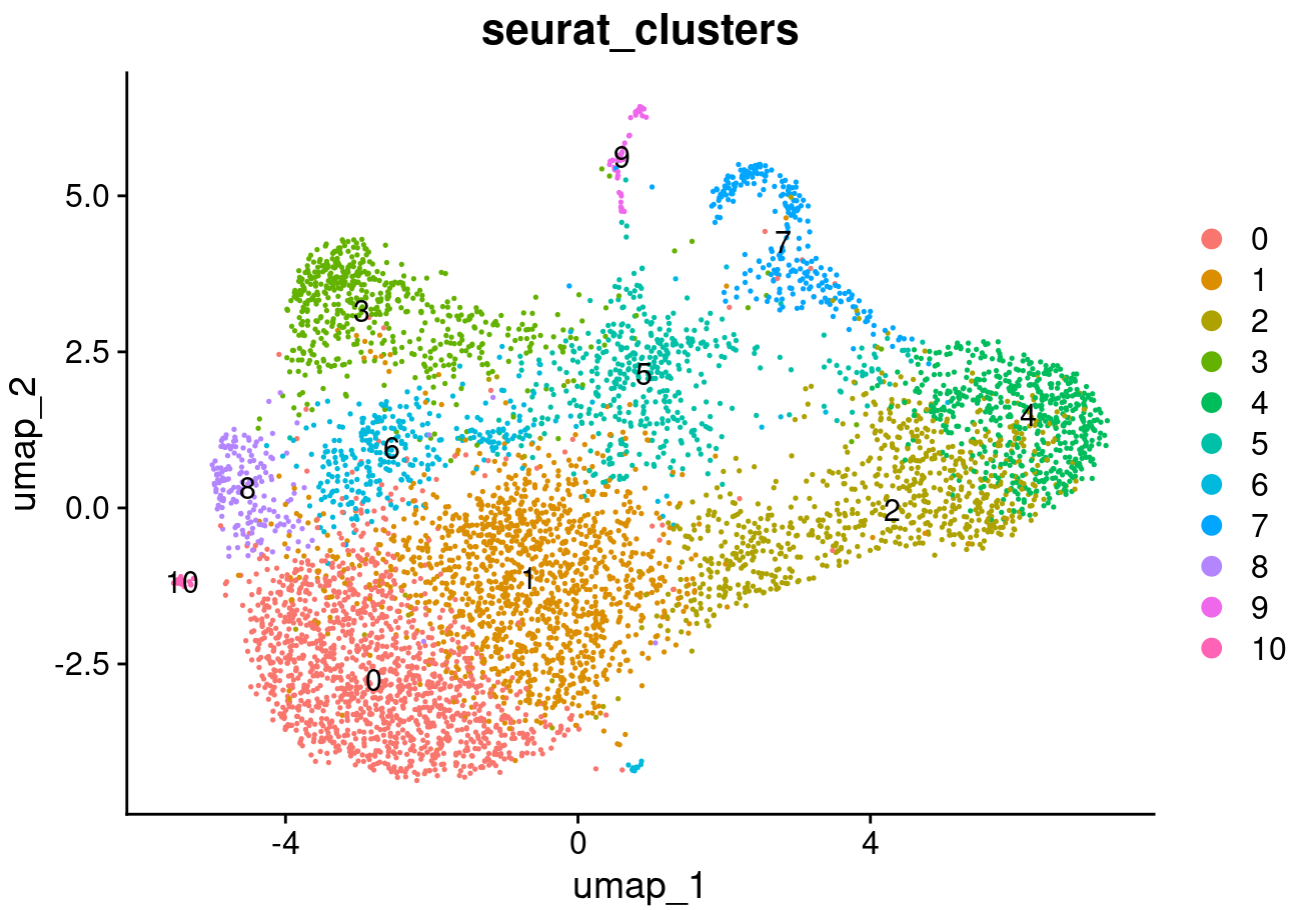
```
## 13:30:26 Building Annoy index with metric = cosine, n_trees = 50
```

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

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

```
## **************************************************|
## 13:30:27 Writing NN index file to temp file /tmp/RtmpZuvVCA/file12ff491de2c17a
## 13:30:27 Searching Annoy index using 1 thread, search_k = 3000
## 13:30:29 Annoy recall = 100%
## 13:30:30 Commencing smooth kNN distance calibration using 1 thread with target n_neighbors
= 30
## 13:30:32 Initializing from normalized Laplacian + noise (using RSpectra)
## 13:30:32 Commencing optimization for 500 epochs, with 242234 positive edges
## 13:30:39 Optimization finished
```

```
DimPlot(Seurat_Object_SP_selected_Bcells, reduction = "umap", label=T, group.by = "seurat_clus
ters")
```




```
Seurat_Object_BM_selected_Bcells <- RunUMAP(Seurat_Object_BM_selected_Bcells, dims = 1:15)
```

```
## 13:30:40 UMAP embedding parameters a = 0.9922 b = 1.112
```

```
## 13:30:40 Read 11316 rows and found 15 numeric columns
```

```
## 13:30:40 Using Annoy for neighbor search, n_neighbors = 30
```

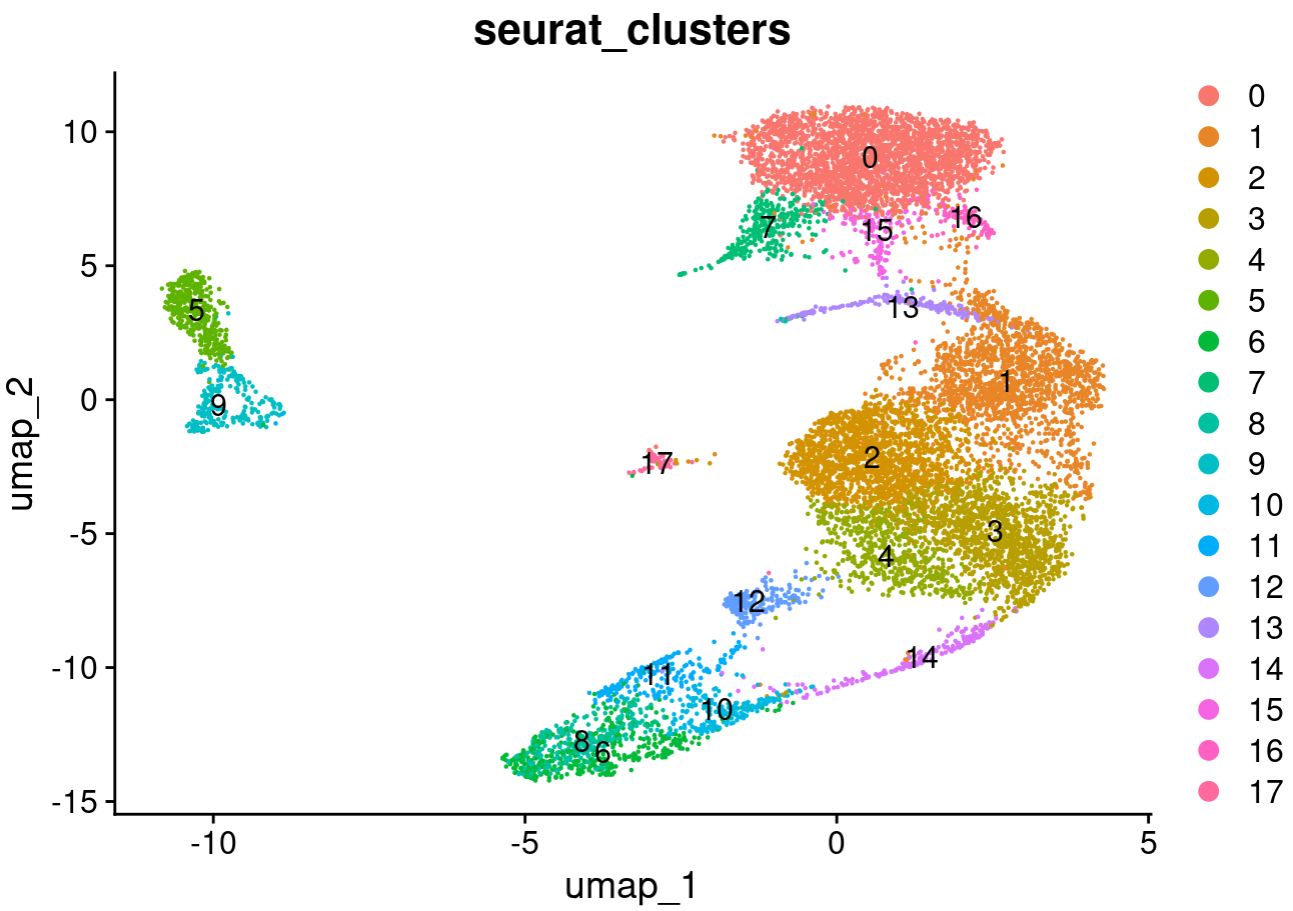
```
## 13:30:40 Building Annoy index with metric = cosine, n_trees = 50
```

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

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

```
## **************************************************|
## 13:30:41 Writing NN index file to temp file /tmp/RtmpZuvVCA/file12ff49c722308
## 13:30:41 Searching Annoy index using 1 thread, search_k = 3000
## 13:30:44 Annoy recall = 100%
## 13:30:45 Commencing smooth kNN distance calibration using 1 thread with target n_neighbors
= 30
## 13:30:47 Initializing from normalized Laplacian + noise (using RSpectra)
## 13:30:48 Commencing optimization for 200 epochs, with 474586 positive edges
## 13:30:53 Optimization finished
```

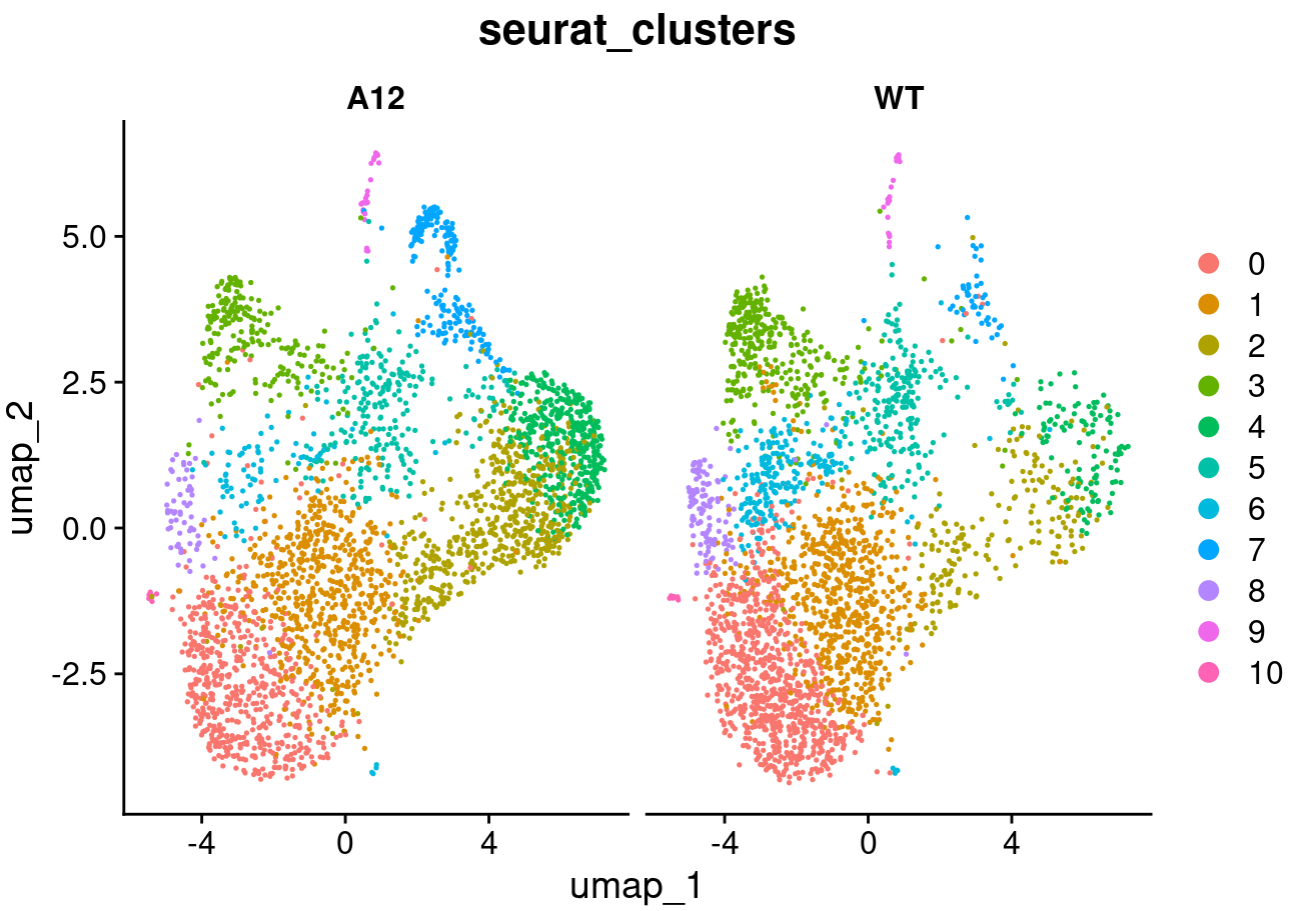
```
DimPlot(Seurat_Object_BM_selected_Bcells, reduction = "umap", label=T, group.by = "seurat_clus
ters")
```



Assign the corresponding genotype to each cell

```
Seurat_Object_SP_selected_Bcells$genotype <- "NA"

A12_cells <- Cells(Seurat_Object_SP_selected_Bcells)[which(Seurat_Object_SP_selected_Bcells$HTO_classification %in% c("SP-A12-1","SP-A12-2","SP-A12-3","huK_SP-A12-2","huK_SP-A12-1","huK_SP-A12-3"))]
Seurat_Object_SP_selected_Bcells@meta.data[A12_cells, "genotype"] = "A12"

WT_cells <- Cells(Seurat_Object_SP_selected_Bcells)[which(Seurat_Object_SP_selected_Bcells$HTO_classification %in% c("SP-WT-1","SP-WT-2","SP-WT-3","huK_SP-WT-2","huK_SP-WT-1","huK_SP-WT-3"))]
Seurat_Object_SP_selected_Bcells@meta.data[WT_cells, "genotype"] = "WT"
```

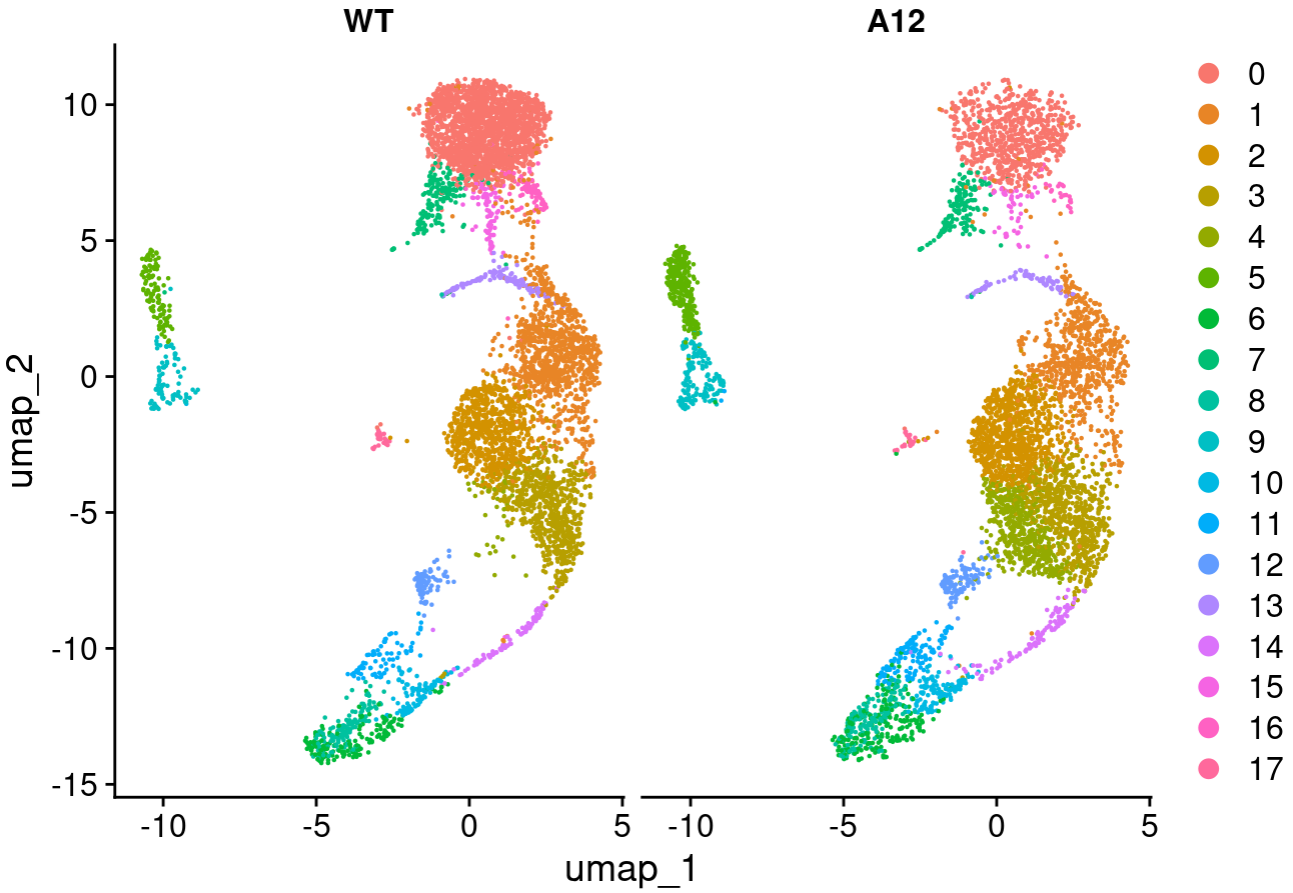
```
DimPlot(Seurat_Object_SP_selected_Bcells, reduction = "umap", split.by = "genotype", group.by = "seurat_clusters")
```




```
Seurat_Object_BM_selected_Bcells$genotype <- "NA"

A12_cells <- Cells(Seurat_Object_BM_selected_Bcells)[which(Seurat_Object_BM_selected_Bcells$HT
O_classification %in% c("BM-A12-1","BM-A12-2","BM-A12-3","BM-A12-2_huK","BM-A12-1_huK","BM-A12
-3_huK"))]
Seurat_Object_BM_selected_Bcells@meta.data[A12_cells, "genotype"] = "A12"

WT_cells <- Cells(Seurat_Object_BM_selected_Bcells)[which(Seurat_Object_BM_selected_Bcells$HTO
_classification %in% c("BM-WT-1","BM-WT-2","BM-WT-3","BM-WT-1_huK","BM-WT-2_huK","BM-WT-3_huK"
))]
Seurat_Object_BM_selected_Bcells@meta.data[WT_cells, "genotype"] = "WT"
```

```
Seurat_Object_BM_selected_Bcells$genotype <- factor(Seurat_Object_BM_selected_Bcells$genotype,
 levels = c("WT", "A12"))
Seurat_Object_SP_selected_Bcells$genotype <- factor(Seurat_Object_SP_selected_Bcells$genotype,
 levels = c("WT", "A12"))
```

```
Idents(Seurat_Object_BM_selected_Bcells) <- Seurat_Object_BM_selected_Bcells$seurat_clusters
DimPlot(Seurat_Object_BM_selected_Bcells, reduction = "umap", split.by = "genotype")
```



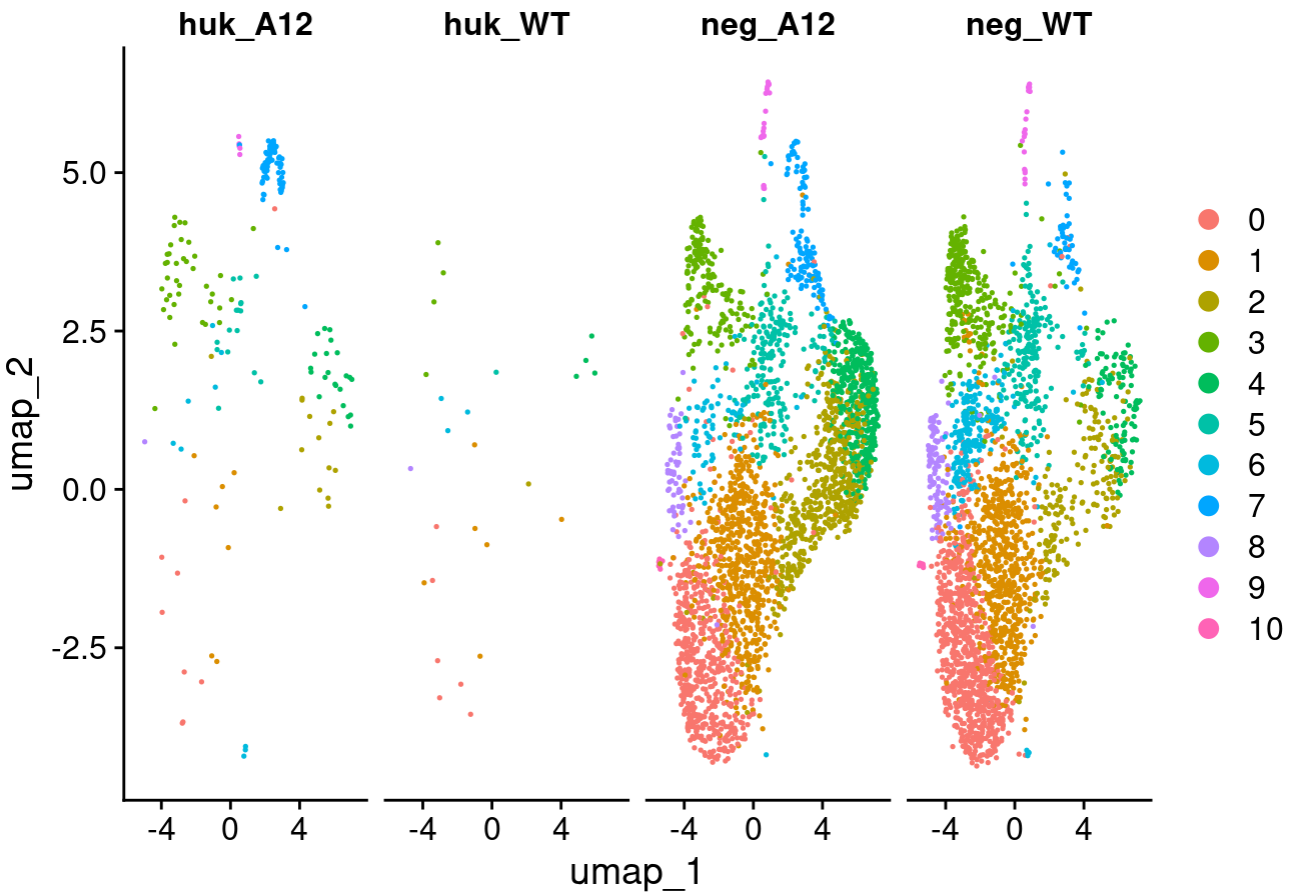
Assign hulgk CITE-seq information to each cell

```
Seurat_Object_SP_selected_Bcells$huIgk <- "NA"

huk_A12_cells <- Cells(Seurat_Object_SP_selected_Bcells)[which(Seurat_Object_SP_selected_Bcell
s$HTO_classification %in% c("huK_SP-A12-2","huK_SP-A12-1","huK_SP-A12-3"))]
Seurat_Object_SP_selected_Bcells@meta.data[huk_A12_cells, "huIgk"] = "huk_A12"

huk_WT_cells <- Cells(Seurat_Object_SP_selected_Bcells)[which(Seurat_Object_SP_selected_Bcells
$HTO_classification %in% c("huK_SP-WT-2","huK_SP-WT-1","huK_SP-WT-3"))]
Seurat_Object_SP_selected_Bcells@meta.data[huk_WT_cells, "huIgk"] = "huk_WT"

neg_A12_cells <- Cells(Seurat_Object_SP_selected_Bcells)[which(Seurat_Object_SP_selected_Bcell
s$HTO_classification %in% c("SP-A12-1","SP-A12-2","SP-A12-3"))]
Seurat_Object_SP_selected_Bcells@meta.data[neg_A12_cells, "huIgk"] = "neg_A12"

neg_WT_cells <- Cells(Seurat_Object_SP_selected_Bcells)[which(Seurat_Object_SP_selected_Bcells
$HTO_classification %in% c("SP-WT-1","SP-WT-2","SP-WT-3"))]
Seurat_Object_SP_selected_Bcells@meta.data[neg_WT_cells, "huIgk"] = "neg_WT"

table(Seurat_Object_SP_selected_Bcells$huIgk)
```

```
##
## huk_A12  huk_WT neg_A12  neg_WT
```

```
##     199      26    2830    2751
```

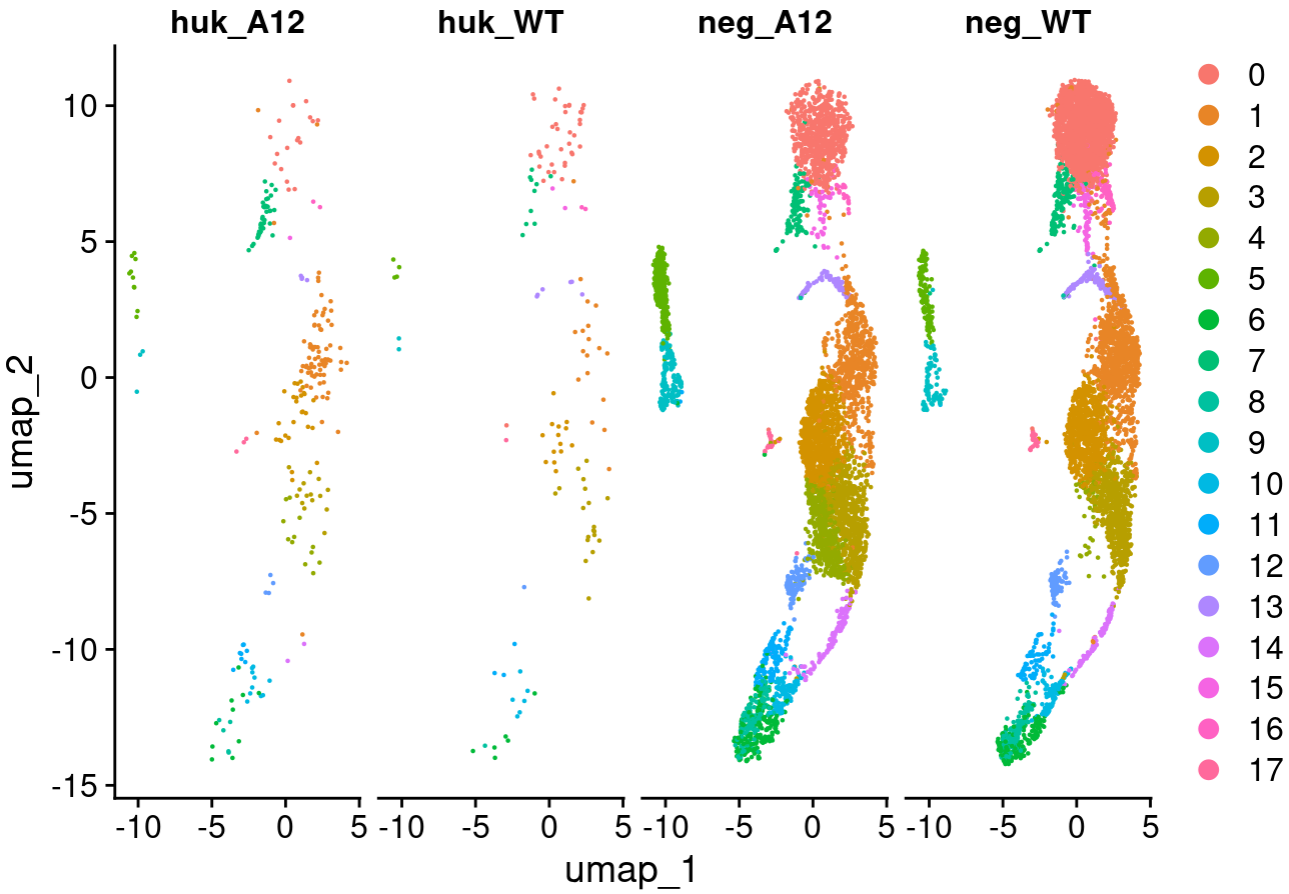
```
DimPlot(Seurat_Object_SP_selected_Bcells, reduction = "umap", split.by = "huIgk")
```




```
Seurat_Object_BM_selected_Bcells$huIgk <- "NA"

huk_A12_cells <- Cells(Seurat_Object_BM_selected_Bcells)[which(Seurat_Object_BM_selected_Bcell
s$HTO_classification %in% c("BM-A12-2_huK","BM-A12-1_huK","BM-A12-3_huK"))]
Seurat_Object_BM_selected_Bcells@meta.data[huk_A12_cells, "huIgk"] = "huk_A12"

huk_WT_cells <- Cells(Seurat_Object_BM_selected_Bcells)[which(Seurat_Object_BM_selected_Bcells
$HTO_classification %in% c("BM-WT-1_huK","BM-WT-2_huK","BM-WT-3_huK"))]
Seurat_Object_BM_selected_Bcells@meta.data[huk_WT_cells, "huIgk"] = "huk_WT"

neg_A12_cells <- Cells(Seurat_Object_BM_selected_Bcells)[which(Seurat_Object_BM_selected_Bcell
s$HTO_classification %in% c("BM-A12-1","BM-A12-2","BM-A12-3"))]
Seurat_Object_BM_selected_Bcells@meta.data[neg_A12_cells, "huIgk"] = "neg_A12"

neg_WT_cells <- Cells(Seurat_Object_BM_selected_Bcells)[which(Seurat_Object_BM_selected_Bcells
$HTO_classification %in% c("BM-WT-1","BM-WT-2","BM-WT-3"))]
Seurat_Object_BM_selected_Bcells@meta.data[neg_WT_cells, "huIgk"] = "neg_WT"

table(Seurat_Object_BM_selected_Bcells$huIgk)
```

```
##
## huk_A12  huk_WT neg_A12  neg_WT
##     255     133    5040    5888
```

```
DimPlot(Seurat_Object_BM_selected_Bcells, reduction = "umap", split.by = "huIgk")
```




```
Seurat_Object_BM_selected_Bcells$huIgk <- factor(Seurat_Object_BM_selected_Bcells$huIgk, level
s = c("neg_WT", "huk_WT", "neg_A12", "huk_A12"))
Seurat_Object_SP_selected_Bcells$huIgk <- factor(Seurat_Object_SP_selected_Bcells$huIgk, level
s = c("neg_WT", "huk_WT", "neg_A12", "huk_A12"))
```

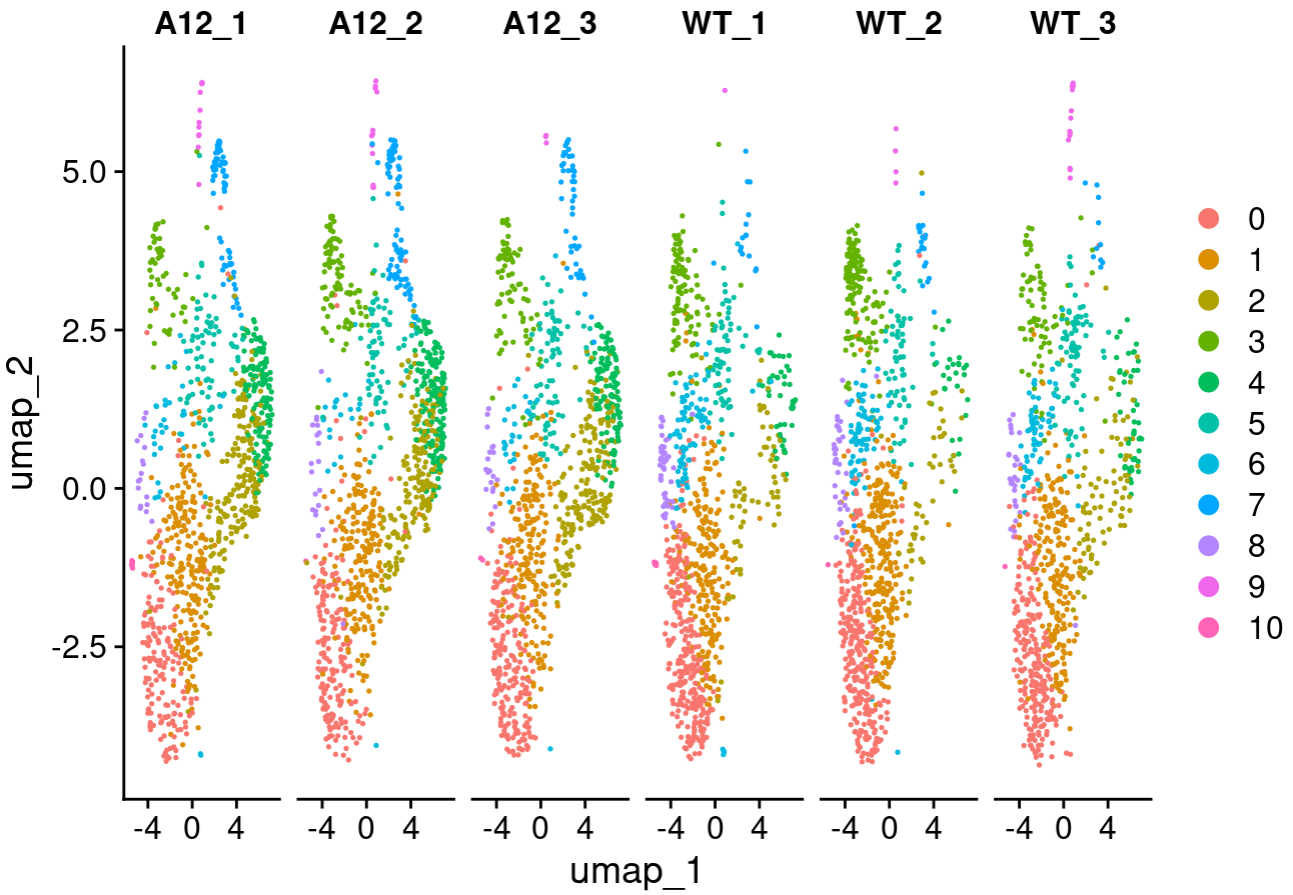
Assign the corresponding mice of origin to each cell

```
Seurat_Object_SP_selected_Bcells$mice <- "NA"

WT_1_cells <- Cells(Seurat_Object_SP_selected_Bcells)[which(Seurat_Object_SP_selected_Bcells$H
TO_classification %in% c("huK_SP-WT-1","SP-WT-1"))]
Seurat_Object_SP_selected_Bcells@meta.data[WT_1_cells, "mice"] = "WT_1"

WT_2_cells <- Cells(Seurat_Object_SP_selected_Bcells)[which(Seurat_Object_SP_selected_Bcells$H
TO_classification %in% c("huK_SP-WT-2","SP-WT-2"))]
```

```
Seurat_Object_SP_selected_Bcells@meta.data[WT_2_cells, "mice"] = "WT_2"

WT_3_cells <- Cells(Seurat_Object_SP_selected_Bcells)[which(Seurat_Object_SP_selected_Bcells$H
TO_classification %in% c("huK_SP-WT-3","SP-WT-3"))]
Seurat_Object_SP_selected_Bcells@meta.data[WT_3_cells, "mice"] = "WT_3"

A12_1_cells <- Cells(Seurat_Object_SP_selected_Bcells)[which(Seurat_Object_SP_selected_Bcells$
HTO_classification %in% c("huK_SP-A12-1","SP-A12-1"))]
Seurat_Object_SP_selected_Bcells@meta.data[A12_1_cells, "mice"] = "A12_1"

A12_2_cells <- Cells(Seurat_Object_SP_selected_Bcells)[which(Seurat_Object_SP_selected_Bcells$
HTO_classification %in% c("huK_SP-A12-2","SP-A12-2"))]
Seurat_Object_SP_selected_Bcells@meta.data[A12_2_cells, "mice"] = "A12_2"

A12_3_cells <- Cells(Seurat_Object_SP_selected_Bcells)[which(Seurat_Object_SP_selected_Bcells$
HTO_classification %in% c("huK_SP-A12-3","SP-A12-3"))]
Seurat_Object_SP_selected_Bcells@meta.data[A12_3_cells, "mice"] = "A12_3"

table(Seurat_Object_SP_selected_Bcells$mice)
```

```
##
## A12_1 A12_2 A12_3  WT_1  WT_2  WT_3
##   982  1056   991   945   966   866
```

```
DimPlot(Seurat_Object_SP_selected_Bcells, reduction = "umap", split.by = "mice")
```

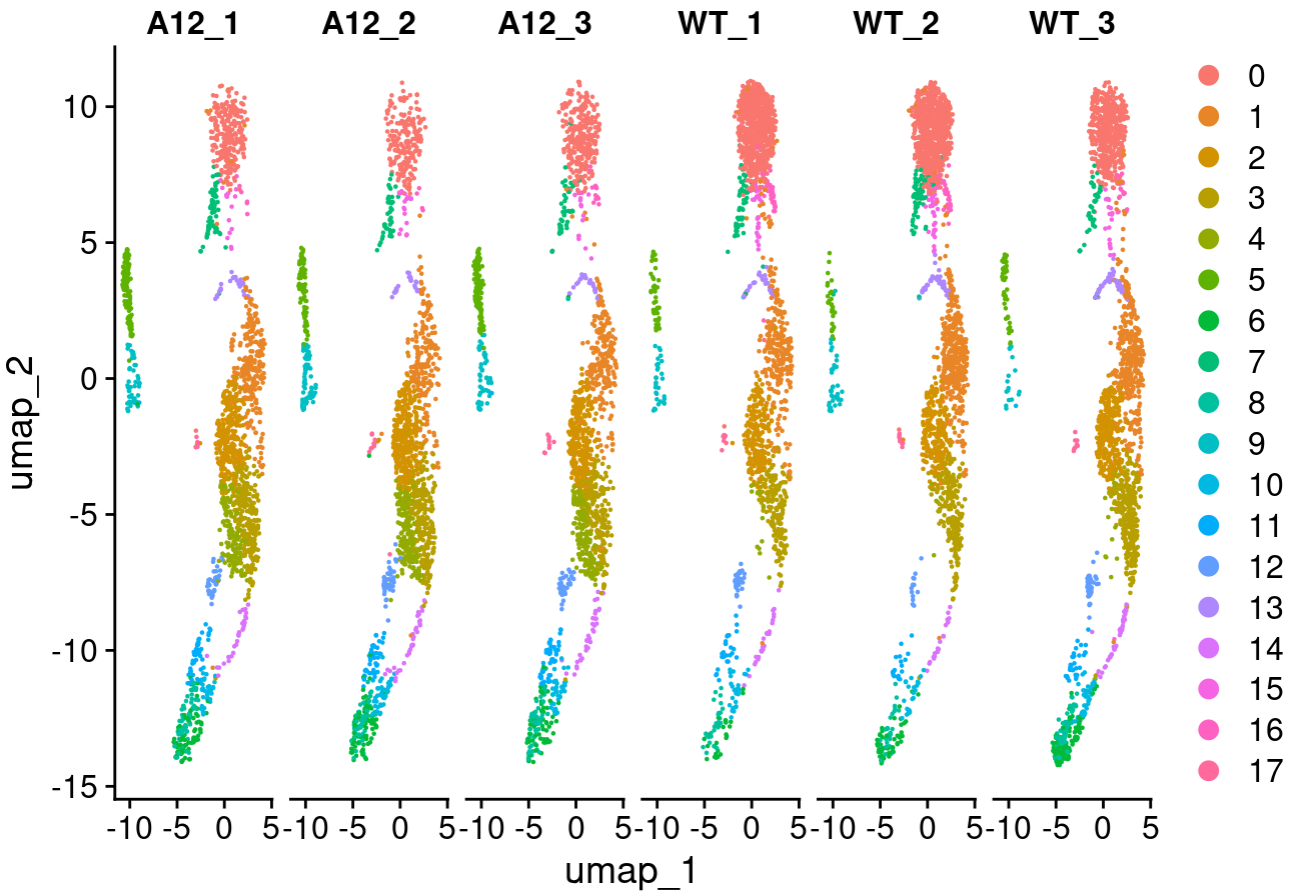



```
Seurat_Object_BM_selected_Bcells$mice <- "NA"

WT_1_cells <- Cells(Seurat_Object_BM_selected_Bcells)[which(Seurat_Object_BM_selected_Bcells$H
TO_classification %in% c("BM-WT-1_huK","BM-WT-1"))]
Seurat_Object_BM_selected_Bcells@meta.data[WT_1_cells, "mice"] = "WT_1"

WT_2_cells <- Cells(Seurat_Object_BM_selected_Bcells)[which(Seurat_Object_BM_selected_Bcells$H
TO_classification %in% c("BM-WT-2_huK","BM-WT-2"))]
Seurat_Object_BM_selected_Bcells@meta.data[WT_2_cells, "mice"] = "WT_2"

WT_3_cells <- Cells(Seurat_Object_BM_selected_Bcells)[which(Seurat_Object_BM_selected_Bcells$H
TO_classification %in% c("BM-WT-3_huK","BM-WT-3"))]
Seurat_Object_BM_selected_Bcells@meta.data[WT_3_cells, "mice"] = "WT_3"

A12_1_cells <- Cells(Seurat_Object_BM_selected_Bcells)[which(Seurat_Object_BM_selected_Bcells$
HTO_classification %in% c("BM-A12-1_huK","BM-A12-1"))]
Seurat_Object_BM_selected_Bcells@meta.data[A12_1_cells, "mice"] = "A12_1"

A12_2_cells <- Cells(Seurat_Object_BM_selected_Bcells)[which(Seurat_Object_BM_selected_Bcells$
HTO_classification %in% c("BM-A12-2_huK","BM-A12-2"))]
Seurat_Object_BM_selected_Bcells@meta.data[A12_2_cells, "mice"] = "A12_2"

A12_3_cells <- Cells(Seurat_Object_BM_selected_Bcells)[which(Seurat_Object_BM_selected_Bcells$
HTO_classification %in% c("BM-A12-3_huK","BM-A12-3"))]
Seurat_Object_BM_selected_Bcells@meta.data[A12_3_cells, "mice"] = "A12_3"
```

```
table(Seurat_Object_BM_selected_Bcells$mice)
```

```
##
## A12_1 A12_2 A12_3  WT_1  WT_2  WT_3
##  1767  1768  1760  1867  2108  2046
```

```
DimPlot(Seurat_Object_BM_selected_Bcells, reduction = "umap", split.by = "mice")
```




```
Seurat_Object_BM_selected_Bcells$mice <- factor(Seurat_Object_BM_selected_Bcells$mice, levels
= c("WT_1", "WT_2", "WT_3", "A12_1","A12_2","A12_3"))
Seurat_Object_SP_selected_Bcells$mice <- factor(Seurat_Object_SP_selected_Bcells$mice, levels
= c("WT_1", "WT_2", "WT_3", "A12_1","A12_2","A12_3"))
```

Immgen fine A12

Focus on specific B cell subsets present in each organ (spleen or bone marrow)

Let's convert our Seurat object to single cell experiment (SCE) for convenience.

```
sce_SP_Bcells <- as.SingleCellExperiment(DietSeurat(Seurat_Object_SP_selected_Bcells))
sce_SP_Bcells
```

```
## class: SingleCellExperiment
```

```
## dim: 17662 5806
## metadata(0):
## assays(3): counts logcounts scaledata
## rownames(17662): Xkr4 Mrpl15 ... AC149090.1 A12
## rowData names(0):
## colnames(5806): AAACCTGAGACGACGT-1 AAACCTGAGCTAGTGG-1 ...
##   TTTGTCATCCGCATCT-1 TTTGTCATCTTGCCGT-1
## colData names(19): orig.ident nCount_RNA ... mice ident
## reducedDimNames(0):
## mainExpName: RNA
## altExpNames(1): HTO
```

```
sce_BM_Bcells <- as.SingleCellExperiment(DietSeurat(Seurat_Object_BM_selected_Bcells))
sce_BM_Bcells
```

```
## class: SingleCellExperiment
## dim: 19884 11316
## metadata(0):
## assays(3): counts logcounts scaledata
## rownames(19884): Xkr4 Rp1 ... AC149090.1 A12
## rowData names(0):
## colnames(11316): AAACCTGAGCCCAATT-1 AAACCTGAGGAATTAC-1 ...
##   TTTGTCATCGCAGGCT-1 TTTGTCATCGCCAAAT-1
## colData names(24): orig.ident nCount_RNA ... mice ident
## reducedDimNames(0):
## mainExpName: RNA
## altExpNames(1): HTO
```

```
bcell_filter <- immgen.se$label.main %in% c("B cells", "B cells, pro")
immgen_Bcells.se <- immgen.se[, bcell_filter]

# Only bone marrow B cell populations
bone_marrow_filter <- immgen_Bcells.se$label.fine %in% c("B cells (B.FrE)", "B cells (preB.FrD
)", "B cells (B.FrF)", "B cells (preB.FrC)", "B cells (proB.FrBC)", "B cells, pro (proB.FrA)",
 "B cells (proB.CLP)")
immgen_Bcells_BoneMarrow.se <- immgen_Bcells.se[, bone_marrow_filter]

# Only splenic B cell populations
SP_filter <- immgen_Bcells.se$label.fine %in% c("B cells (B.Fo)", "B cells (B.GC)","B cells (B
.MZ)", "B cells (B.MEM)", "B cells (B.T1)", "B cells (B.T2)", "B cells (B.T3)", "B cells (B1b)
", "B cells (B1a)")
immgen_Bcells_SP.se <- immgen_Bcells.se[, SP_filter]
```

```
immgen_SP.fine <- SingleR(test = sce_SP_Bcells, assay.type.test = 1,ref = immgen_Bcells_SP.se
,labels = immgen_Bcells_SP.se$label.fine)
table(immgen_SP.fine$pruned.labels)
```

```
##
## B cells (B.Fo) B cells (B.GC) B cells (B.MZ) B cells (B.T1) B cells (B.T2)
##           2955             41           1164             89            497
```

```
## B cells (B.T3)  B cells (B1a)  B cells (B1b)
##            794            244              9
```

```
immgen_BM.fine <- SingleR(test = sce_BM_Bcells, assay.type.test = 1,ref = immgen_Bcells_BoneMa
rrow.se,labels = immgen_Bcells_BoneMarrow.se$label.fine)
table(immgen_BM.fine$pruned.labels)
```

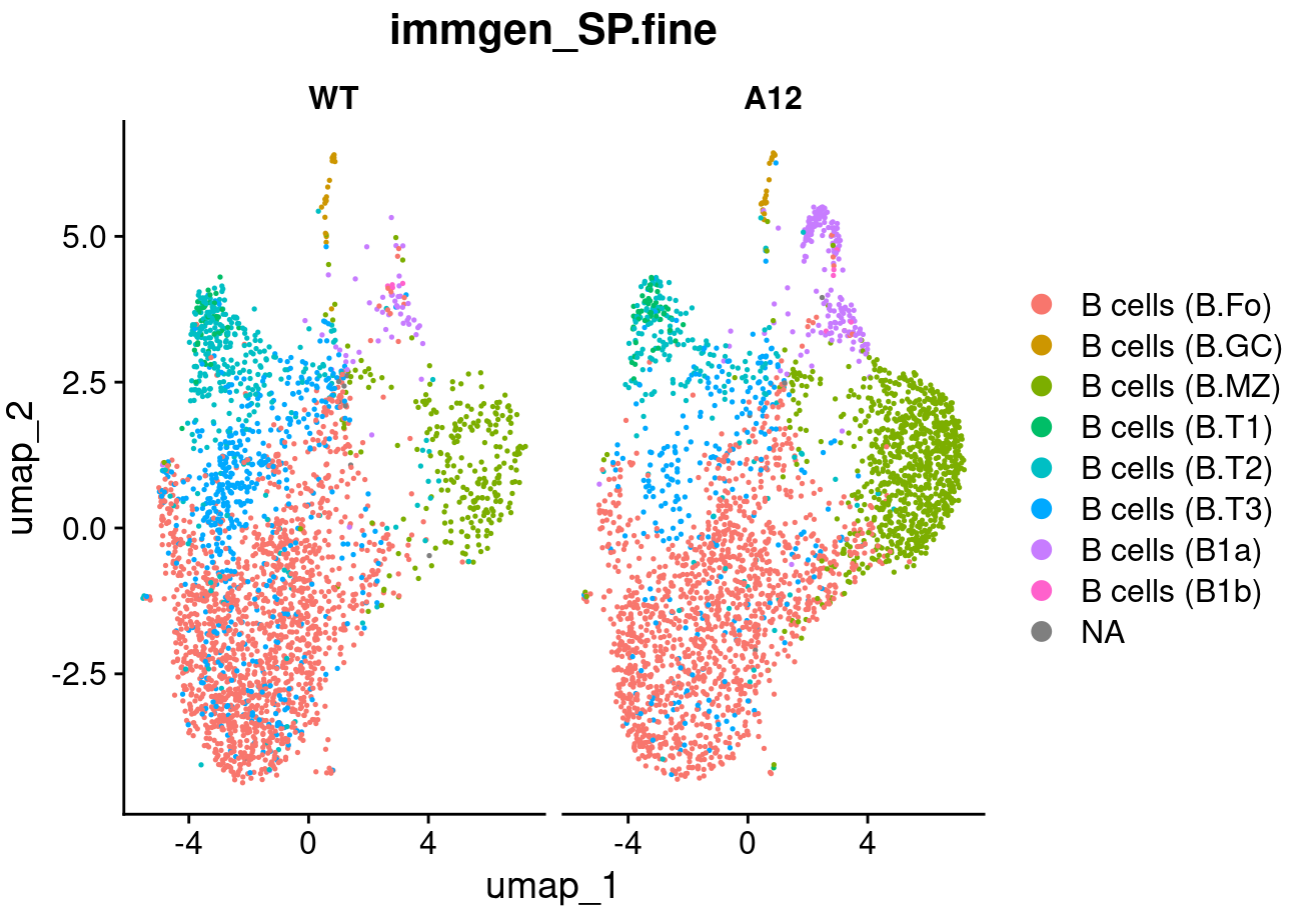
```
##
##         B cells (B.FrE)         B cells (B.FrF)      B cells (preB.FrC)
##                    1959                    3484                    1049
##      B cells (preB.FrD)      B cells (proB.CLP)     B cells (proB.FrBC)
##                    3645                     360                     415
## B cells, pro (proB.FrA)
##                     341
```

Add annotations to seurat object

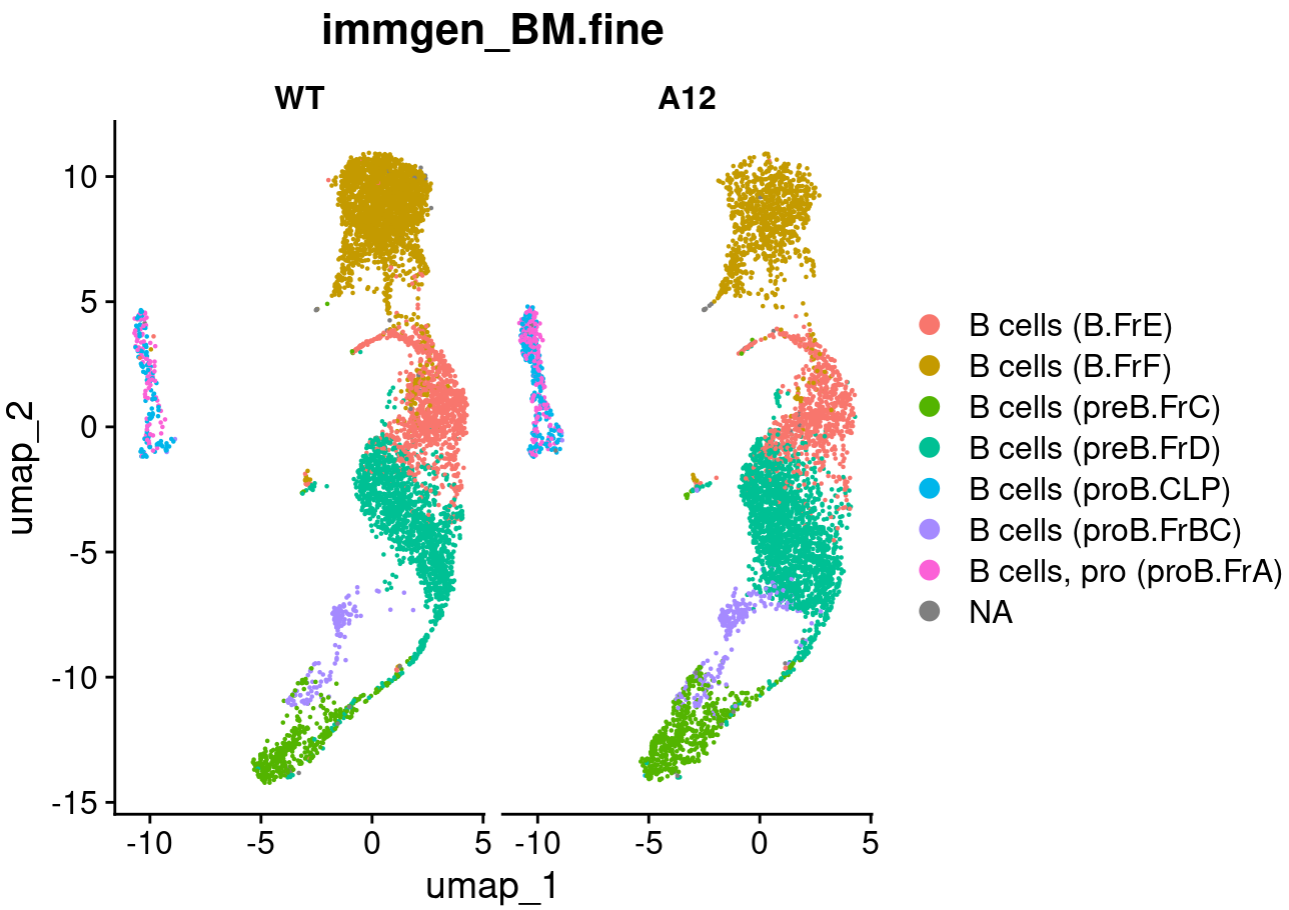
```
Seurat_Object_SP_selected_Bcells@meta.data$immgen_SP.fine <- immgen_SP.fine$pruned.labels
Seurat_Object_SP_selected_Bcells$immgen_SP.fine <- na.omit(Seurat_Object_SP_selected_Bcells$im
mgen_SP.fine)
```

```
Seurat_Object_BM_selected_Bcells@meta.data$immgen_BM.fine <- immgen_BM.fine$pruned.labels
Seurat_Object_BM_selected_Bcells$immgen_BM.fine <- na.omit(Seurat_Object_BM_selected_Bcells$im
mgen_BM.fine)
```

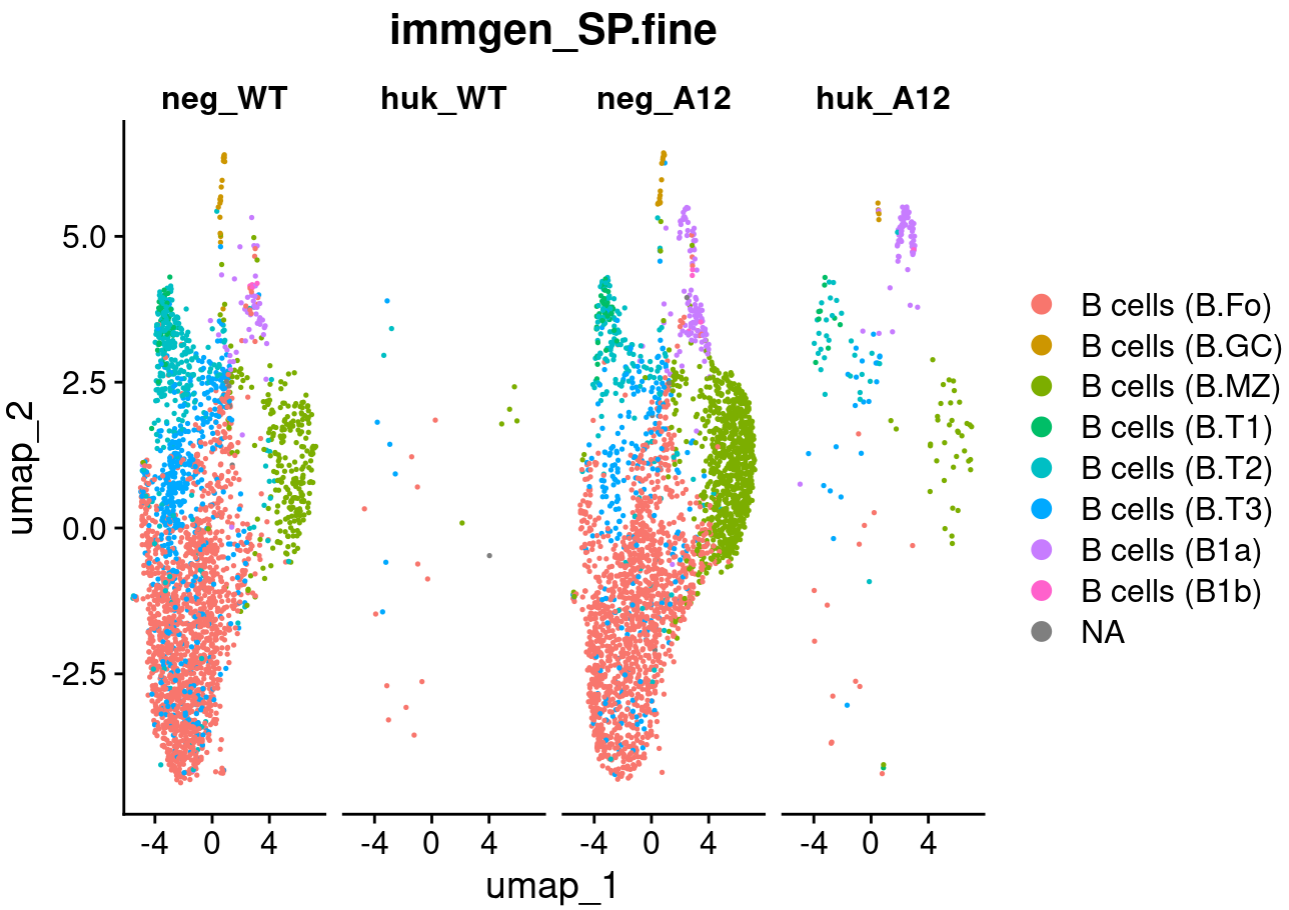
```
DimPlot(Seurat_Object_SP_selected_Bcells, reduction = "umap", group.by = "immgen_SP.fine", spl
it.by = "genotype")
```



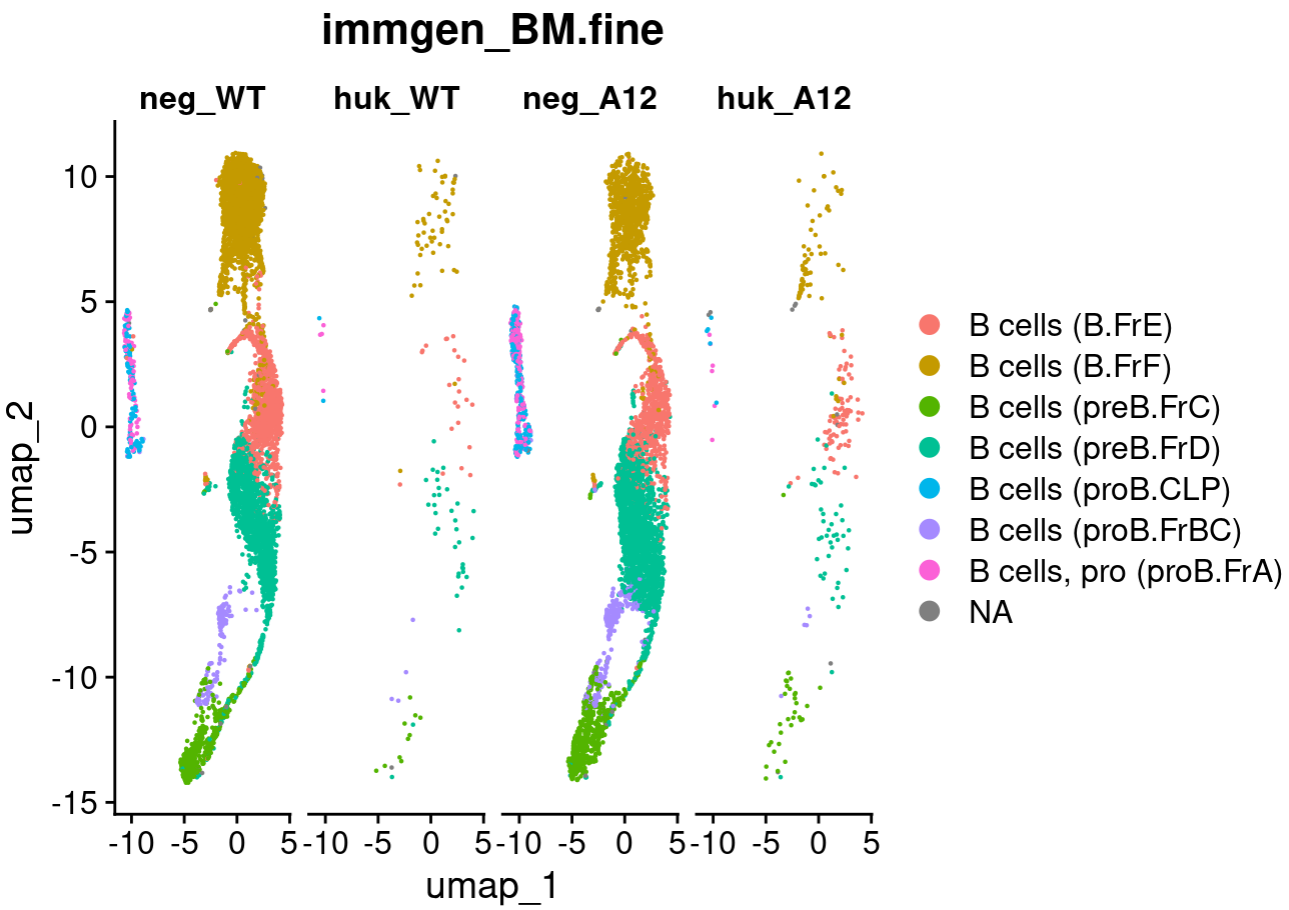

```
DimPlot(Seurat_Object_BM_selected_Bcells, reduction = "umap", group.by = "immgen_BM.fine", spl
it.by = "genotype")
```



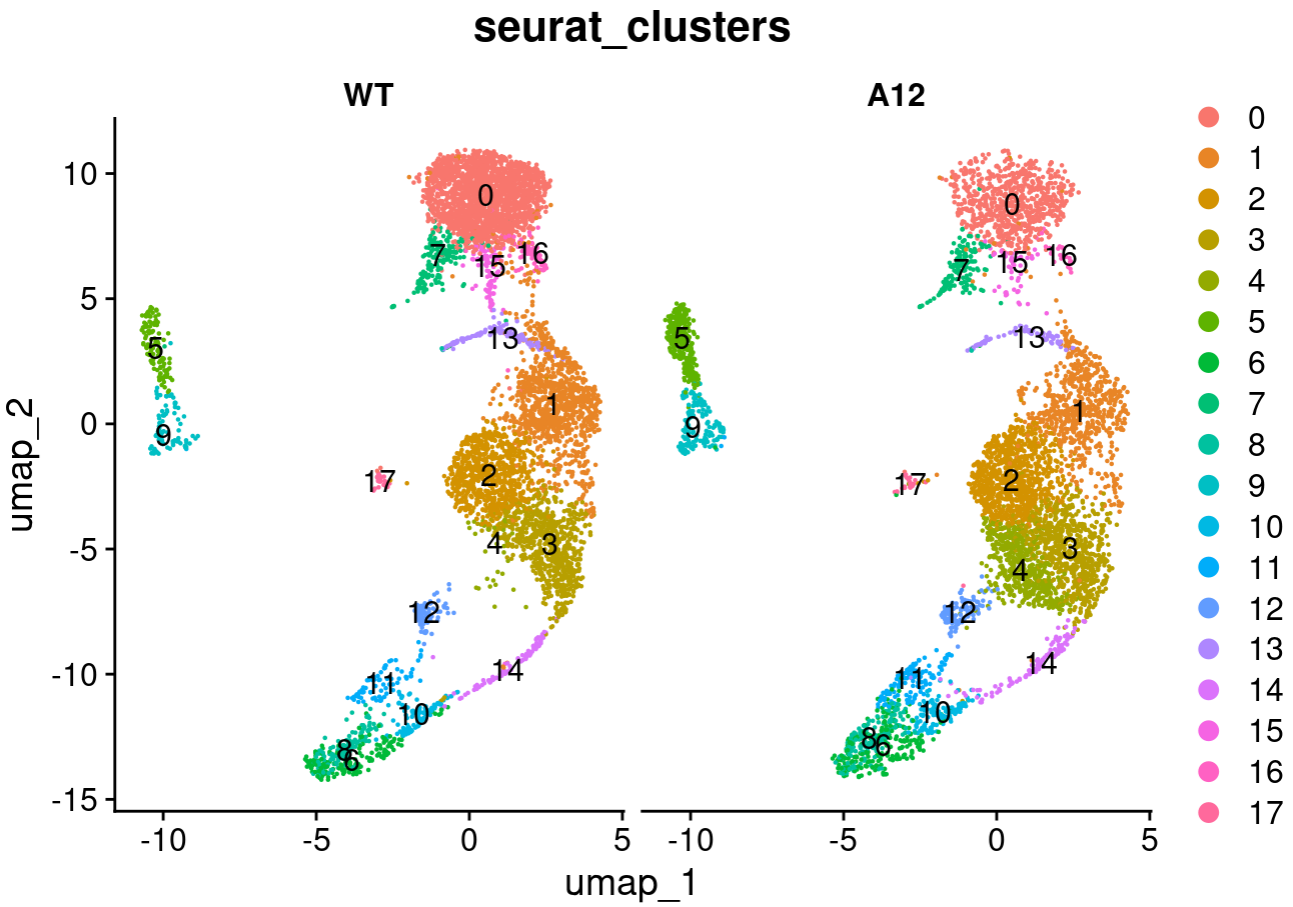

```
DimPlot(Seurat_Object_SP_selected_Bcells, reduction = "umap", group.by = "immgen_SP.fine", spl
it.by = "huIgk")
```



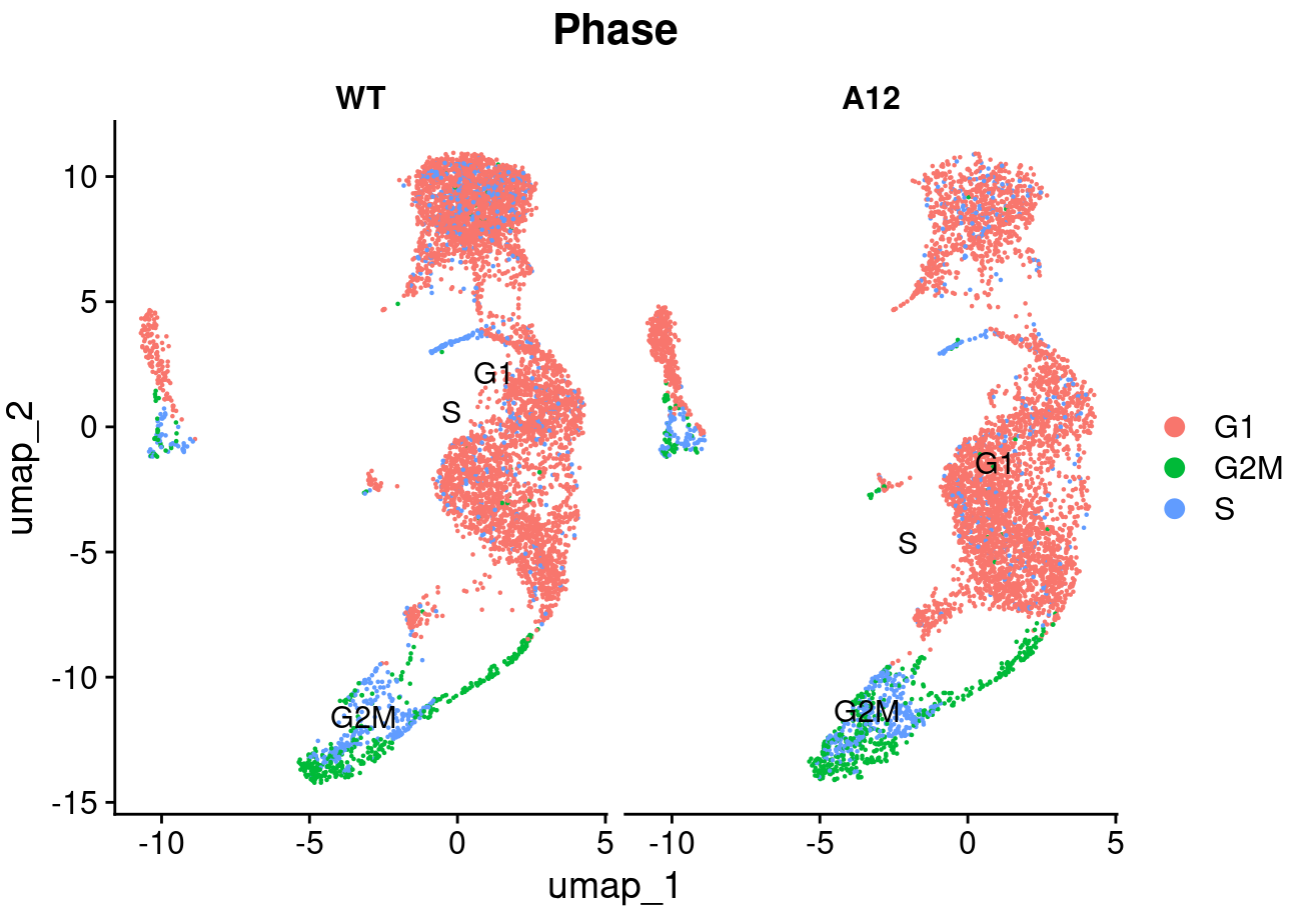

```
DimPlot(Seurat_Object_BM_selected_Bcells, reduction = "umap", group.by = "immgen_BM.fine", spl
it.by = "huIgk")
```



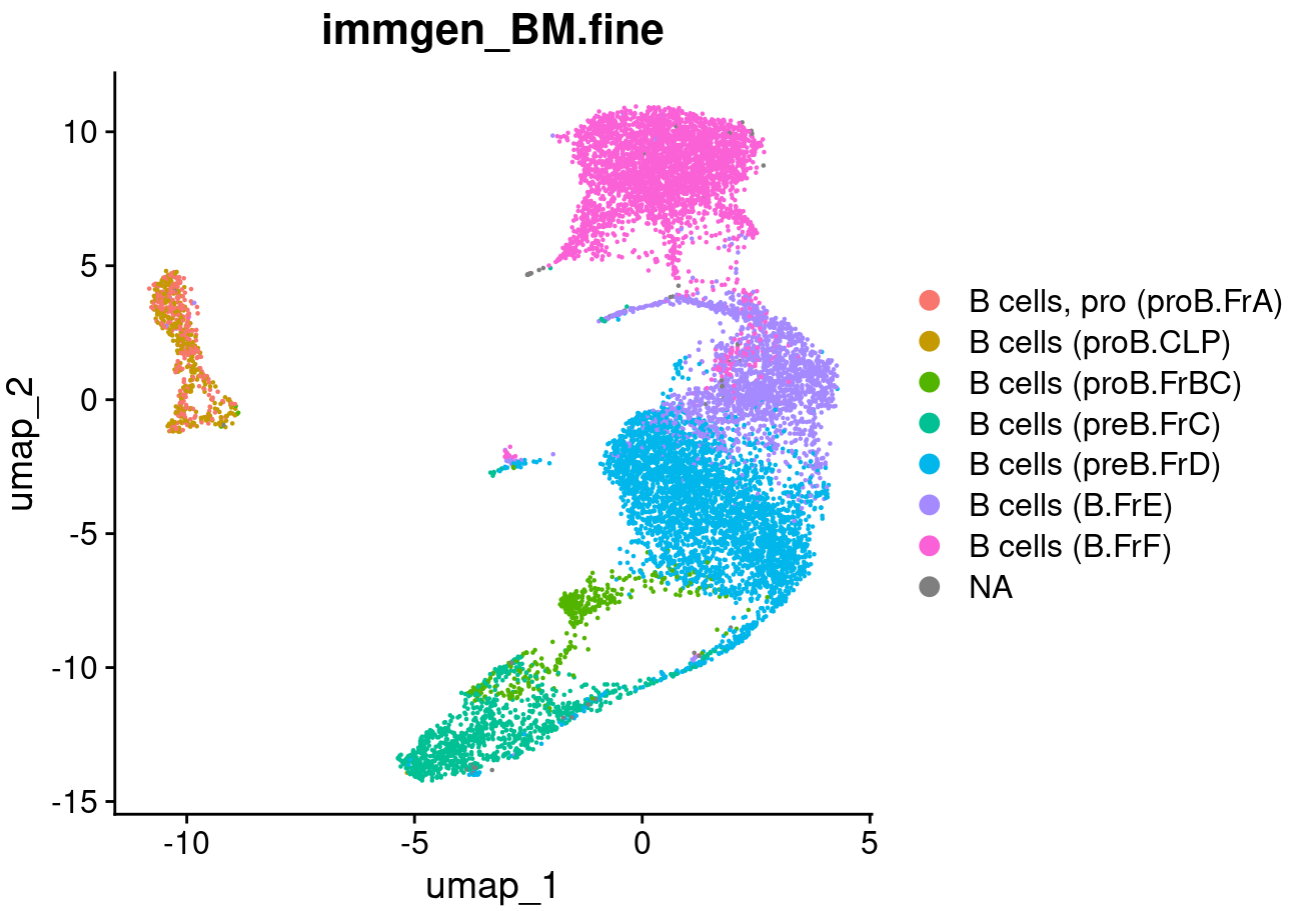

```
DimPlot(Seurat_Object_BM_selected_Bcells, reduction = "umap", group.by = "seurat_clusters", sp
lit.by = "genotype", label = TRUE)
```




```
DimPlot(Seurat_Object_BM_selected_Bcells, reduction = "umap", group.by = "Phase", split.by = "
genotype", label = TRUE)
```



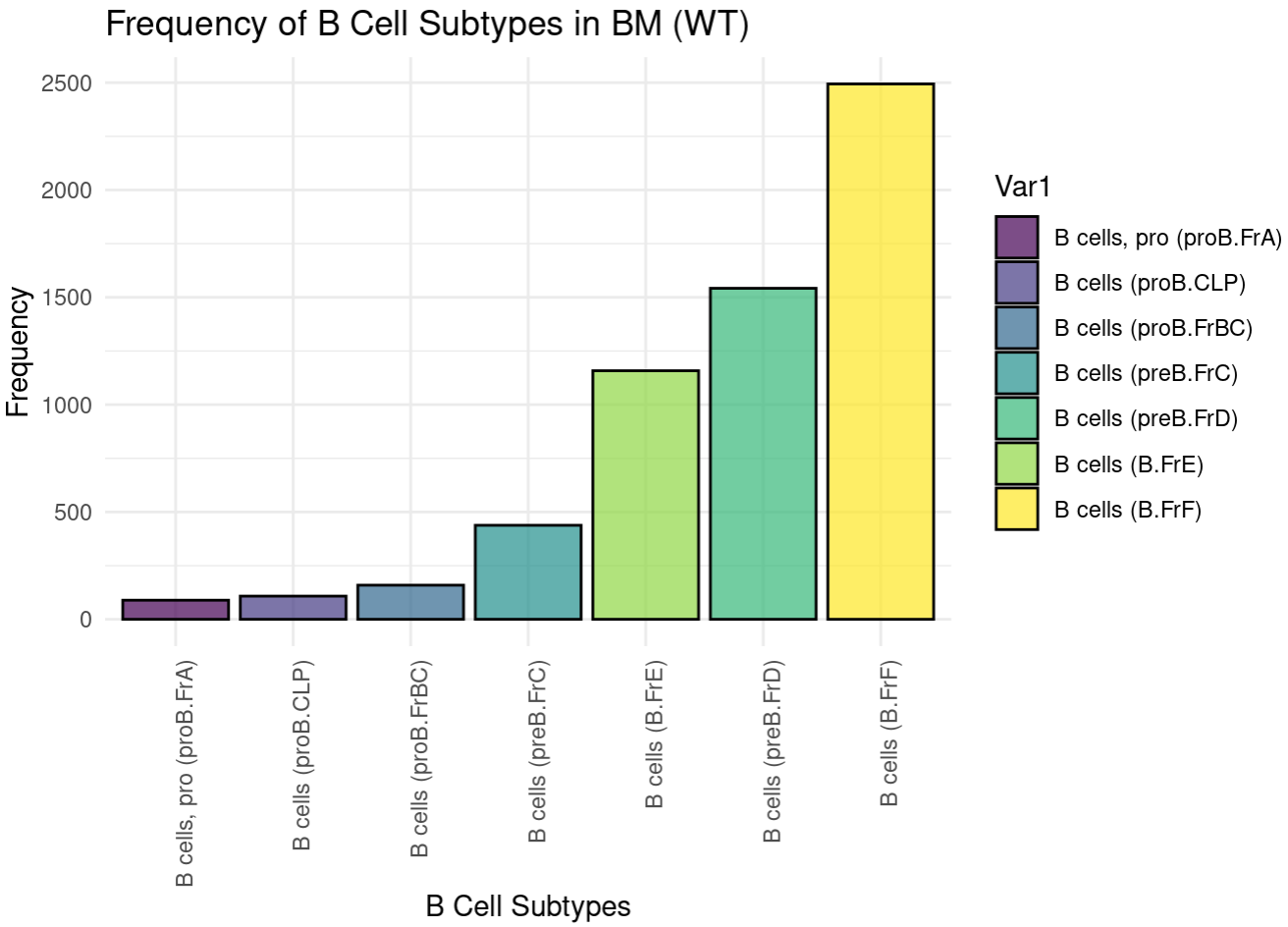

```
# Reorder levels manualy in the Seurat object
Seurat_Object_BM_selected_Bcells$immgen_BM.fine <- factor(Seurat_Object_BM_selected_Bcells$imm
gen_BM.fine,
                                levels = c("B cells, pro (proB.FrA)",
                                           "B cells (proB.CLP)",
                                           "B cells (proB.FrBC)",
                                           "B cells (preB.FrC)",
                                           "B cells (preB.FrD)",
                                           "B cells (B.FrE)",
                                           "B cells (B.FrF)"))
DimPlot(Seurat_Object_BM_selected_Bcells, group.by = "immgen_BM.fine")
```



Histogram of B cell subtypes

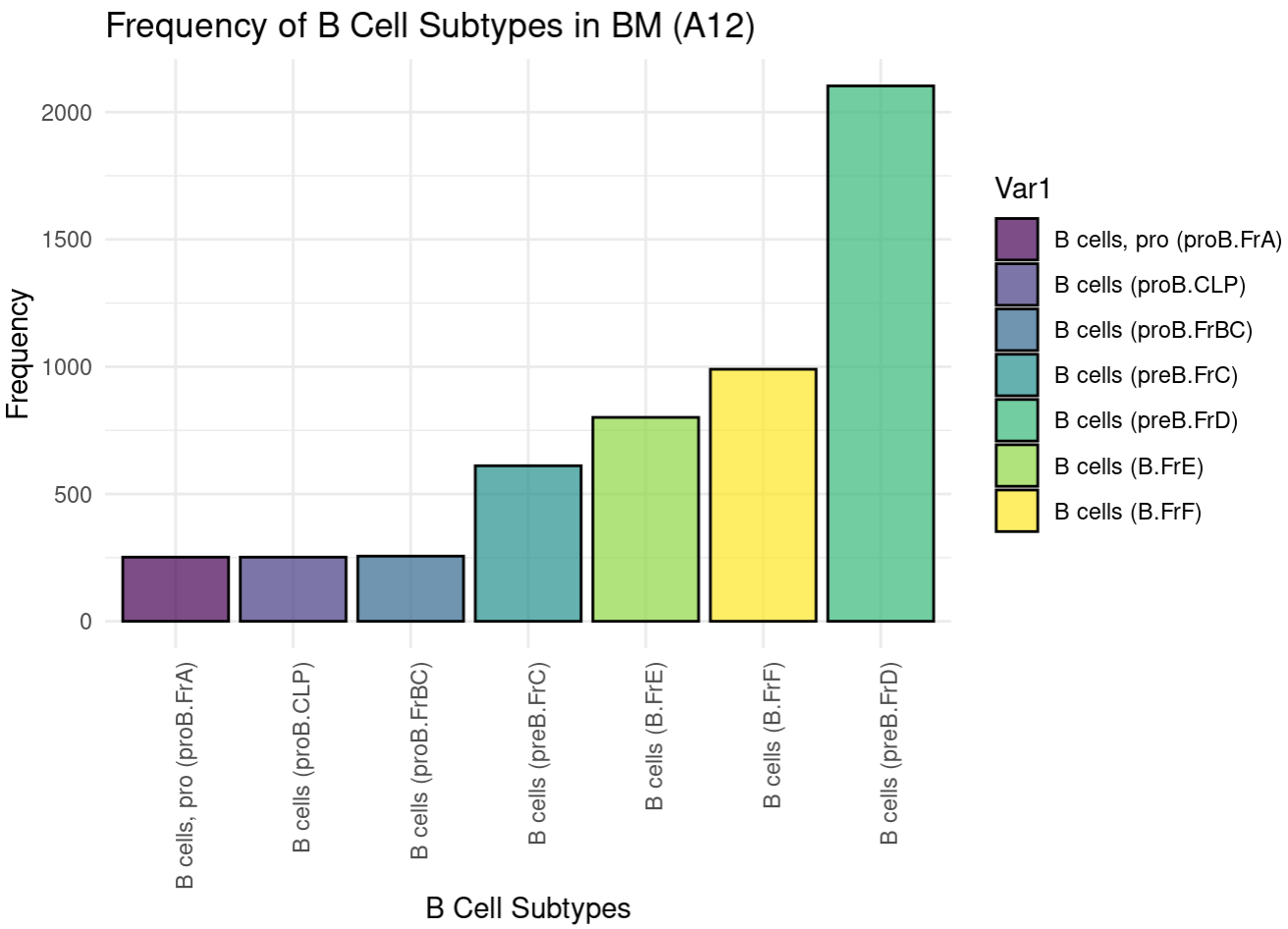
```
Seurat_Object_BM_A12 <- subset(Seurat_Object_BM_selected_Bcells, genotype == "A12")
Seurat_Object_BM_WT <- subset(Seurat_Object_BM_selected_Bcells, genotype == "WT")

### FOR WT ###

# Get all unique categories from Seurat_Object_BM_WT$immgen_BM.fine
cell_subtypes <- unique(Seurat_Object_BM_WT$immgen_BM.fine)

# Calculate the frequency of each category in Seurat_Object_BM_WT$immgen_BM.fine
freq_table_WT <- table(Seurat_Object_BM_WT$immgen_BM.fine)

# Convert the frequency table into a data frame and sort it from highest to lowest
freq_df_WT <- as.data.frame(freq_table_WT)
freq_df_WT <- freq_df_WT[order(-freq_df_WT$Freq), ]

# Create a bar plot with ggplot2 using colors based on the categories
bar_plot <- ggplot(freq_df_WT, aes(x = reorder(Var1, Freq), y = Freq, fill = Var1)) +
  geom_bar(stat = "identity", color = "black", alpha = 0.7) +  # Bar plot
  scale_fill_viridis_d() +  # Continuous color scale (viridis)
  labs(title = "Frequency of B Cell Subtypes in BM (WT)", x = "B Cell Subtypes", y = "Frequenc
y") +  # Title and axis labels
  theme_minimal() +  # Theme style
  theme(axis.text.x = element_text(angle = 90, hjust = 1))  # Rotate x-axis labels
```

```
# Display the bar plot
print(bar_plot)
```



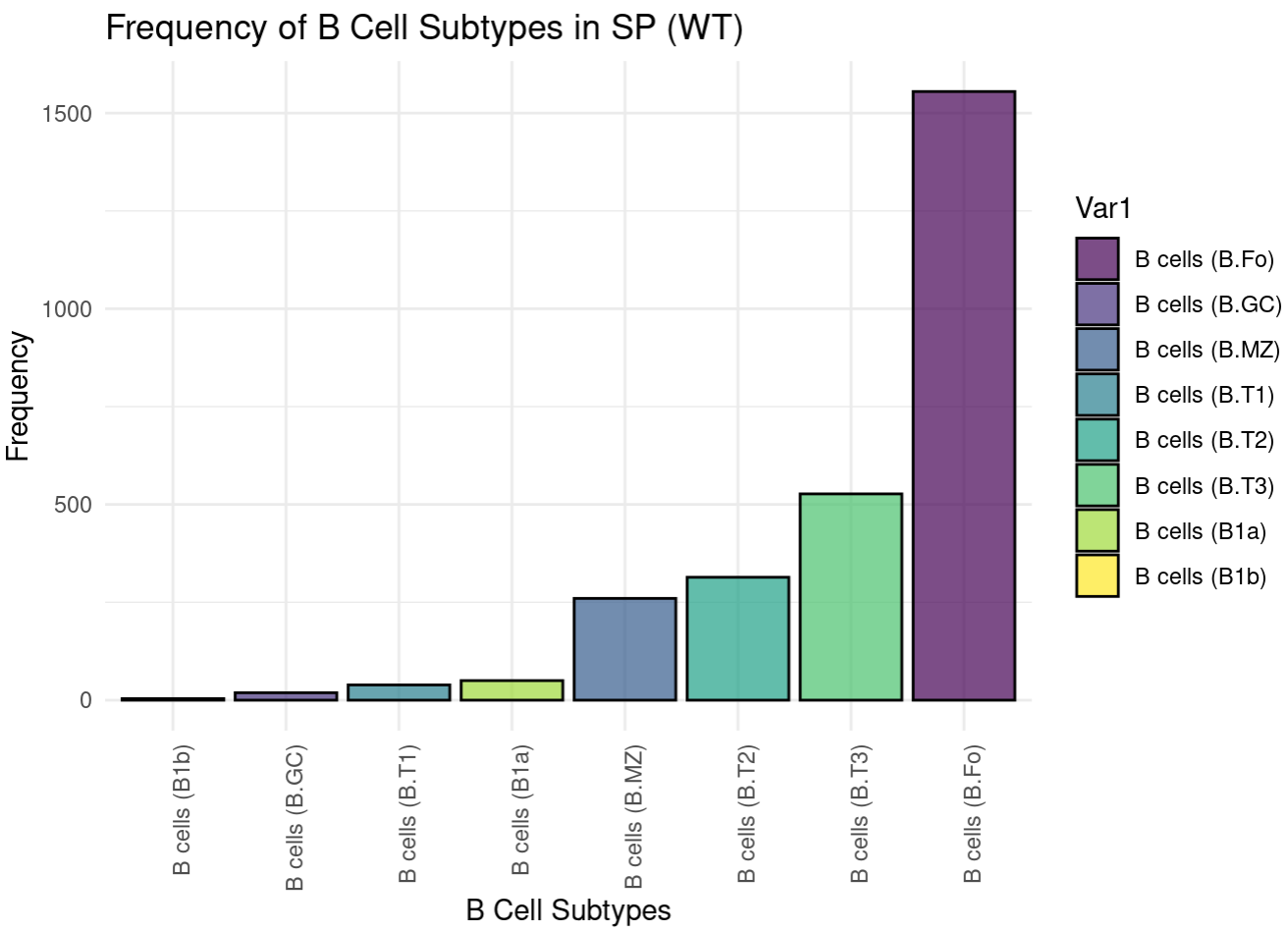

```
### FOR A12 ###

# Get all unique categories from Seurat_Object_BM_A12$immgen_BM.fine
cell_subtypes <- unique(Seurat_Object_BM_A12$immgen_BM.fine)

# Calculate the frequency of each category in Seurat_Object_BM_A12$immgen_BM.fine
freq_table_A12 <- table(Seurat_Object_BM_A12$immgen_BM.fine)

# Convert the frequency table into a data frame and sort it from highest to lowest
freq_df_A12 <- as.data.frame(freq_table_A12)
freq_df_A12 <- freq_df_A12[order(-freq_df_A12$Freq), ]

# Create a bar plot with ggplot2 using colors based on the categories
bar_plot <- ggplot(freq_df_A12, aes(x = reorder(Var1, Freq), y = Freq, fill = Var1)) +
  geom_bar(stat = "identity", color = "black", alpha = 0.7) +  # Bar plot
  scale_fill_viridis_d() +  # Continuous color scale (viridis)
  labs(title = "Frequency of B Cell Subtypes in BM (A12)", x = "B Cell Subtypes", y = "Frequen
cy") +  # Title and axis labels
  theme_minimal() +  # Theme style
  theme(axis.text.x = element_text(angle = 90, hjust = 1))  # Rotate x-axis labels

# Display the bar plot
```

```
print(bar_plot)
```

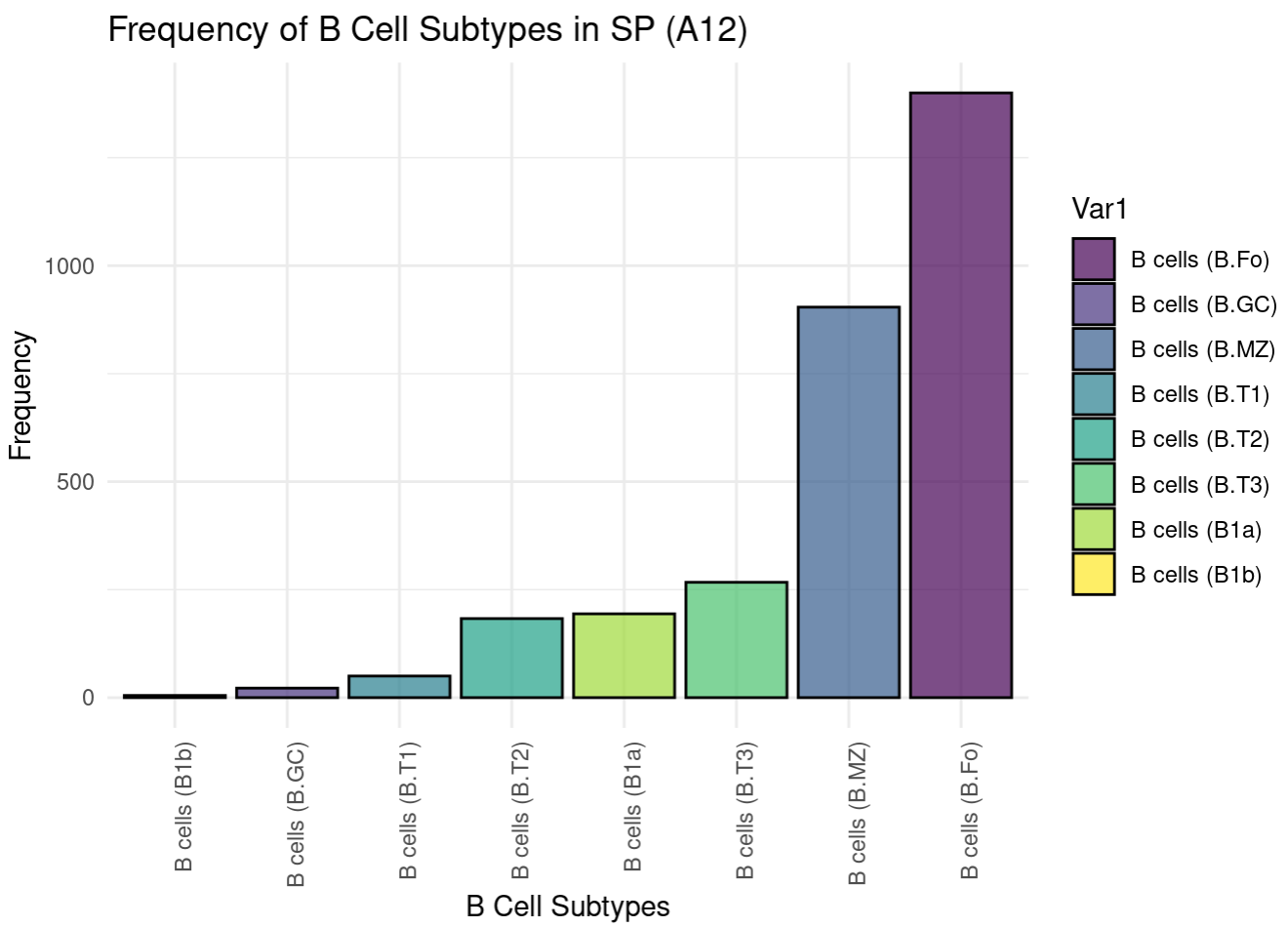



```
Seurat_Object_SP_A12 <- subset(Seurat_Object_SP_selected_Bcells, genotype == "A12")
Seurat_Object_SP_WT <- subset(Seurat_Object_SP_selected_Bcells, genotype == "WT")

### FOR WT ###

# Get all unique categories from Seurat_Object_SP_WT$immgen_SP.fine
cell_subtypes <- unique(Seurat_Object_SP_WT$immgen_SP.fine)

# Calculate the frequency of each category in Seurat_Object_SP_WT$immgen_SP.fine
freq_table_WT <- table(Seurat_Object_SP_WT$immgen_SP.fine)

# Convert the frequency table into a data frame and sort it from highest to lowest
freq_df_WT <- as.data.frame(freq_table_WT)
freq_df_WT <- freq_df_WT[order(-freq_df_WT$Freq), ]

# Create a bar plot with ggplot2 using colors based on the categories
bar_plot <- ggplot(freq_df_WT, aes(x = reorder(Var1, Freq), y = Freq, fill = Var1)) +
  geom_bar(stat = "identity", color = "black", alpha = 0.7) +  # Bar plot
  scale_fill_viridis_d() +  # Continuous color scale (viridis)
  labs(title = "Frequency of B Cell Subtypes in SP (WT)", x = "B Cell Subtypes", y = "Frequenc
y") +  # Title and axis labels
  theme_minimal() +  # Theme style
  theme(axis.text.x = element_text(angle = 90, hjust = 1))  # Rotate x-axis labels
```

```
# Display the bar plot
print(bar_plot)
```




```
### FOR A12 ###

# Get all unique categories from Seurat_Object_SP_A12$immgen_SP.fine
cell_subtypes <- unique(Seurat_Object_SP_A12$immgen_SP.fine)

# Calculate the frequency of each category in Seurat_Object_SP_A12$immgen_SP.fine
freq_table_A12 <- table(Seurat_Object_SP_A12$immgen_SP.fine)

# Convert the frequency table into a data frame and sort it from highest to lowest
freq_df_A12 <- as.data.frame(freq_table_A12)
freq_df_A12 <- freq_df_A12[order(-freq_df_A12$Freq), ]

# Create a bar plot with ggplot2 using colors based on the categories
bar_plot <- ggplot(freq_df_A12, aes(x = reorder(Var1, Freq), y = Freq, fill = Var1)) +
  geom_bar(stat = "identity", color = "black", alpha = 0.7) +  # Bar plot
  scale_fill_viridis_d() +  # Continuous color scale (viridis)
  labs(title = "Frequency of B Cell Subtypes in SP (A12)", x = "B Cell Subtypes", y = "Frequen
cy") +  # Title and axis labels
  theme_minimal() +  # Theme style
  theme(axis.text.x = element_text(angle = 90, hjust = 1))  # Rotate x-axis labels

# Display the bar plot
print(bar_plot)
```



Cellular cycle

```
CC_BM <- CellCycleScoring(Seurat_Object_BM_selected_Bcells, s.features = cc.genes$s.genes, g2m
.features = cc.genes$g2m.genes)
```

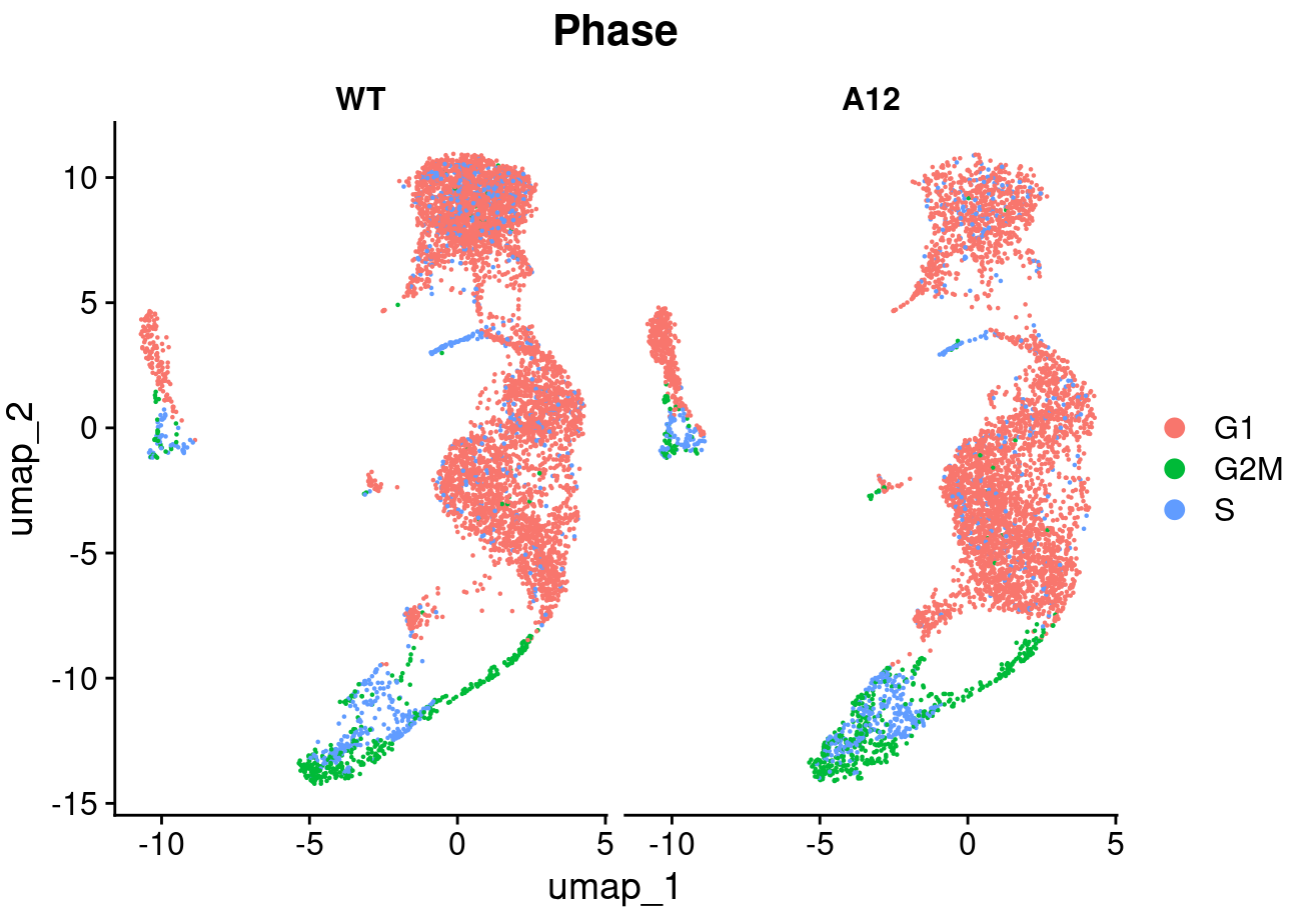
```
## Warning: The following features are not present in the object: MCM5, PCNA,
## TYMS, FEN1, MCM2, MCM4, RRM1, UNG, GINS2, MCM6, CDCA7, DTL, PRIM1, UHRF1,
## MLF1IP, HELLS, RFC2, RPA2, NASP, RAD51AP1, GMNN, WDR76, SLBP, CCNE2, UBR7,
## POLD3, MSH2, ATAD2, RAD51, RRM2, CDC45, CDC6, EXO1, TIPIN, DSCC1, BLM,
## CASP8AP2, USP1, CLSPN, POLA1, CHAF1B, BRIP1, E2F8, not searching for symbol
## synonyms
```

```
## Warning: The following features are not present in the object: HMGB2, CDK1,
## NUSAP1, UBE2C, BIRC5, TPX2, TOP2A, NDC80, CKS2, NUF2, CKS1B, MKI67, TMPO,
## CENPF, TACC3, FAM64A, SMC4, CCNB2, CKAP2L, CKAP2, AURKB, BUB1, KIF11, ANP32E,
## TUBB4B, GTSE1, KIF20B, HJURP, CDCA3, HN1, CDC20, TTK, CDC25C, KIF2C, RANGAP1,
## NCAPD2, DLGAP5, CDCA2, CDCA8, ECT2, KIF23, HMMR, AURKA, PSRC1, ANLN, LBR,
## CKAP5, CENPE, CTCF, NEK2, G2E3, GAS2L3, CBX5, CENPA, not searching for symbol
## synonyms
```

```
## Warning in AddModuleScore(object = object, features = features, name = name, :
## Could not find enough features in the object from the following feature lists:
```

```
## S.Score Attempting to match case...Could not find enough features in the object
## from the following feature lists: G2M.Score Attempting to match case...
```

```
DimPlot(CC_BM, group.by = "Phase", split.by = "genotype")
```



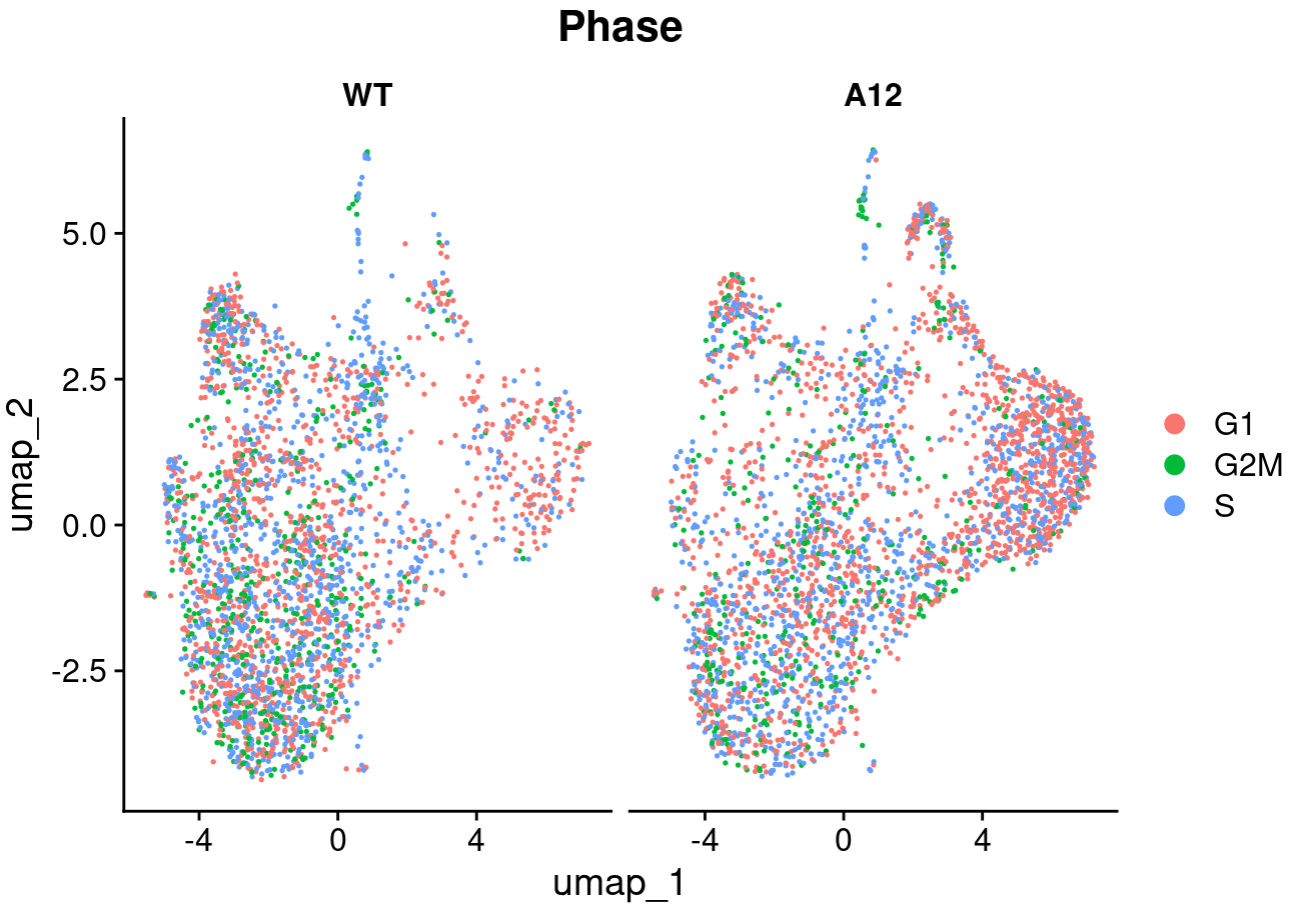

```
CC_SP <- CellCycleScoring(Seurat_Object_SP_selected_Bcells, s.features = cc.genes$s.genes, g2m
.features = cc.genes$g2m.genes)
```

```
## Warning: The following features are not present in the object: MCM5, PCNA,
## TYMS, FEN1, MCM2, MCM4, RRM1, UNG, GINS2, MCM6, CDCA7, DTL, PRIM1, UHRF1,
## MLF1IP, HELLS, RFC2, RPA2, NASP, RAD51AP1, GMNN, WDR76, SLBP, CCNE2, UBR7,
## POLD3, MSH2, ATAD2, RAD51, RRM2, CDC45, CDC6, EXO1, TIPIN, DSCC1, BLM,
## CASP8AP2, USP1, CLSPN, POLA1, CHAF1B, BRIP1, E2F8, not searching for symbol
## synonyms
```

```
## Warning: The following features are not present in the object: HMGB2, CDK1,
## NUSAP1, UBE2C, BIRC5, TPX2, TOP2A, NDC80, CKS2, NUF2, CKS1B, MKI67, TMPO,
## CENPF, TACC3, FAM64A, SMC4, CCNB2, CKAP2L, CKAP2, AURKB, BUB1, KIF11, ANP32E,
## TUBB4B, GTSE1, KIF20B, HJURP, CDCA3, HN1, CDC20, TTK, CDC25C, KIF2C, RANGAP1,
## NCAPD2, DLGAP5, CDCA2, CDCA8, ECT2, KIF23, HMMR, AURKA, PSRC1, ANLN, LBR,
## CKAP5, CENPE, CTCF, NEK2, G2E3, GAS2L3, CBX5, CENPA, not searching for symbol
## synonyms
```

```
## Warning in AddModuleScore(object = object, features = features, name = name, :
## Could not find enough features in the object from the following feature lists:
## S.Score Attempting to match case...Could not find enough features in the object
## from the following feature lists: G2M.Score Attempting to match case...
```

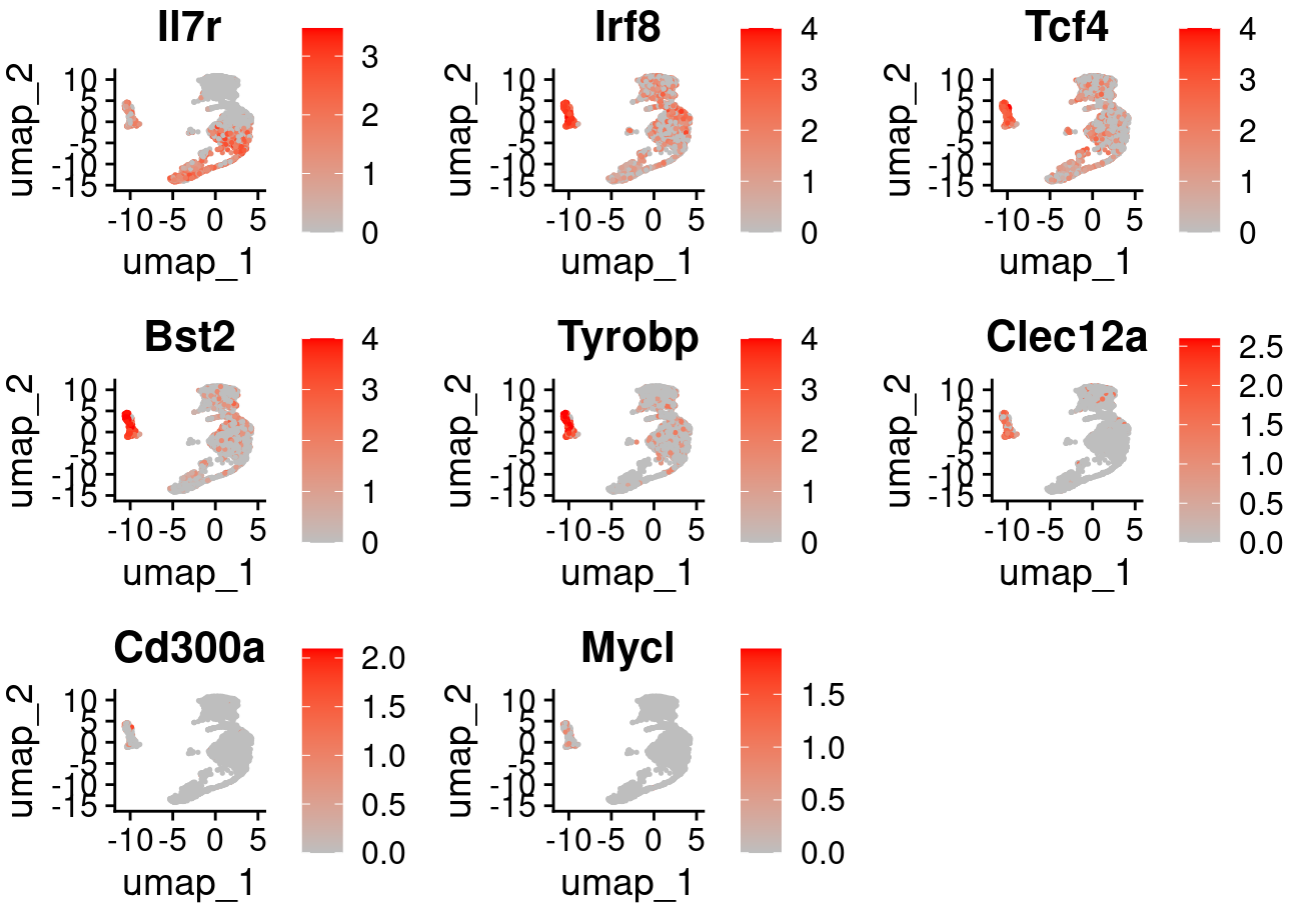
```
DimPlot(CC_SP, group.by = "Phase", split.by = "genotype")
```



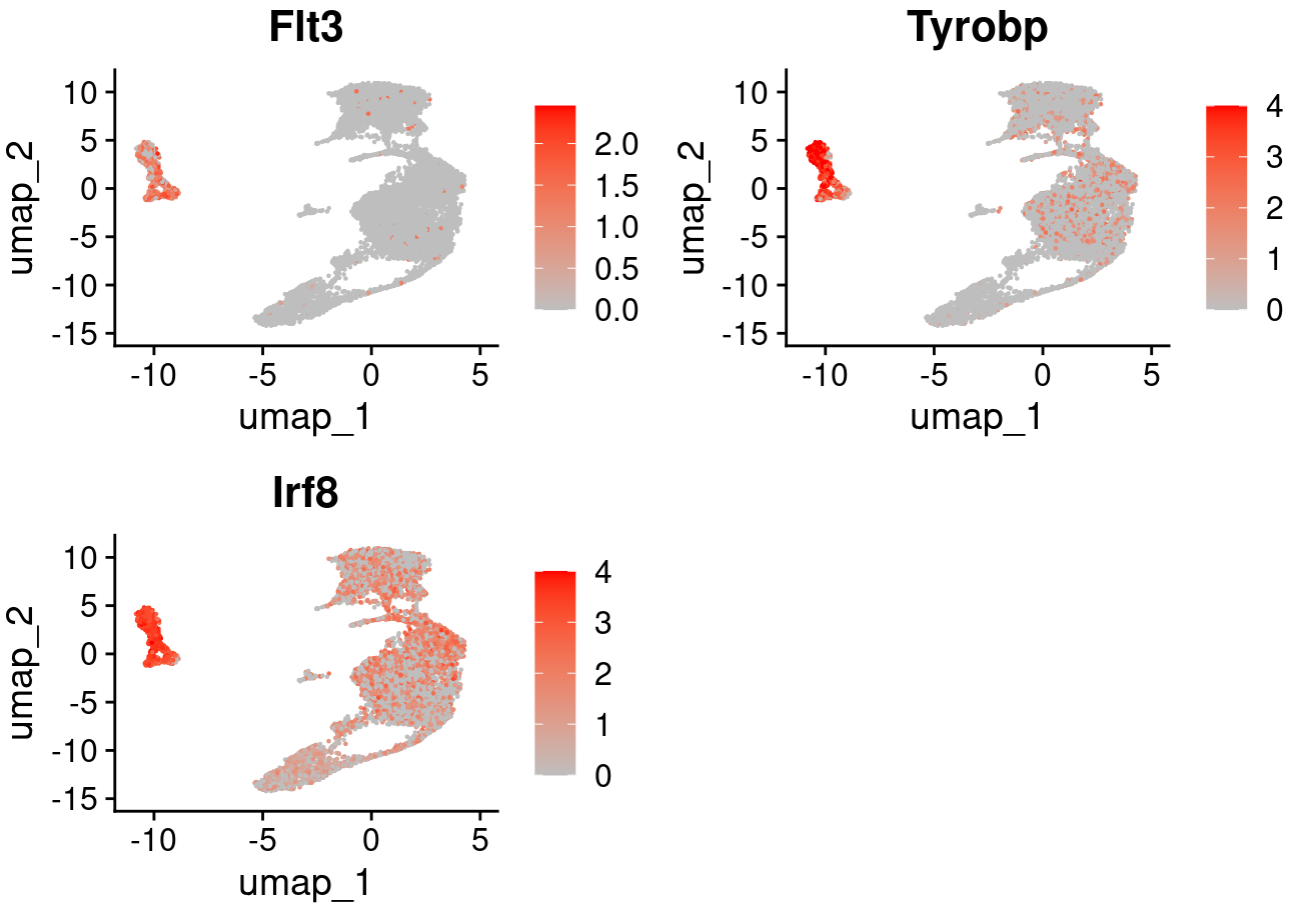
Assign celltype based on reference

Pre-ProB cell identification

```
FeaturePlot(Seurat_Object_BM_WT, features = c("Il7r","Irf8","Tcf4","Bst2","Tyrobp","Clec12a","
Cd300a","Mycl"), max.cutoff = 4,
    cols = c("grey", "red"))
```

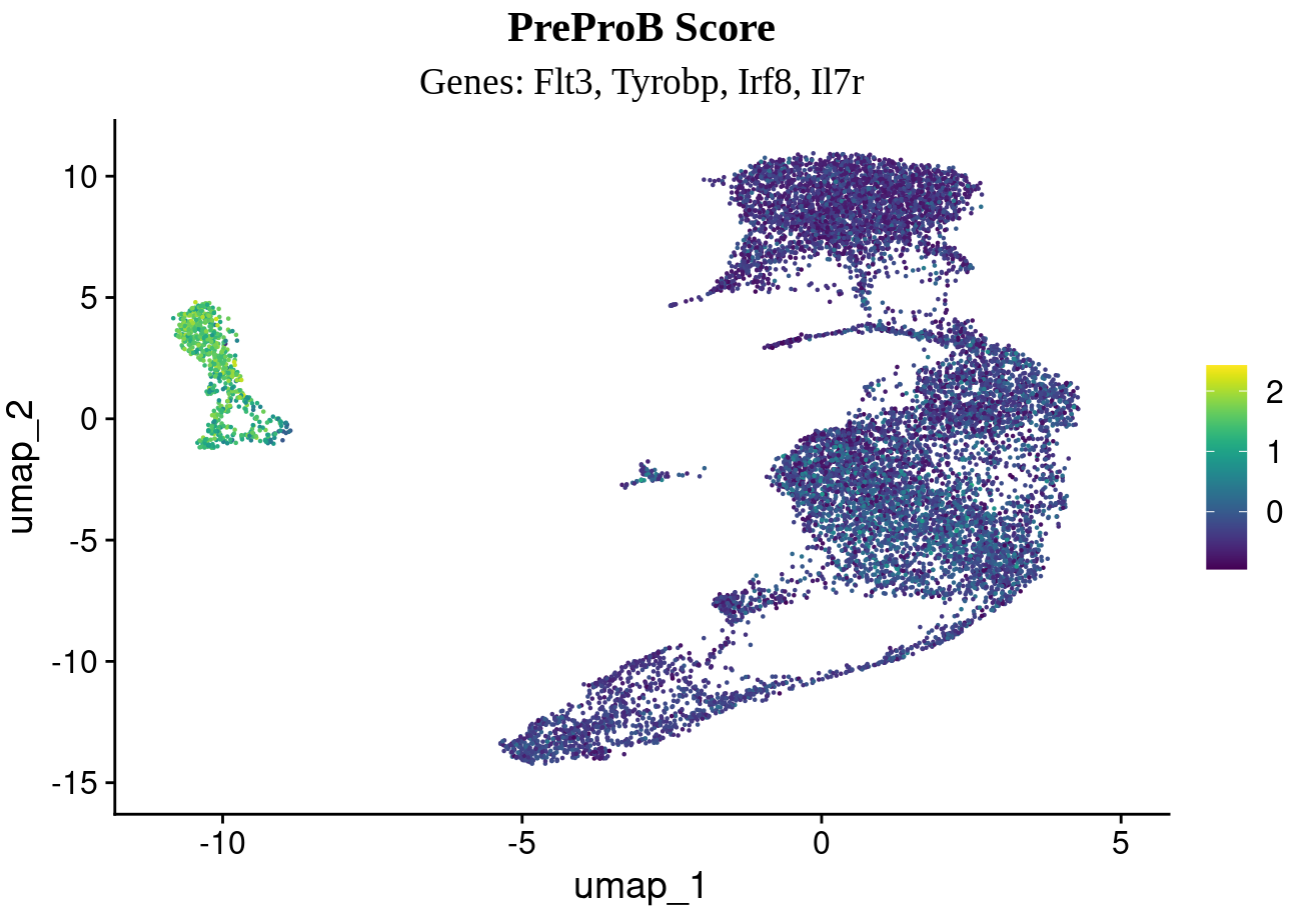



```
FeaturePlot(Seurat_Object_BM_selected_Bcells, features = c("Flt3", "Tyrobp","Irf8"), max.cutof
f = 4,
    cols = c("grey", "red"))
```



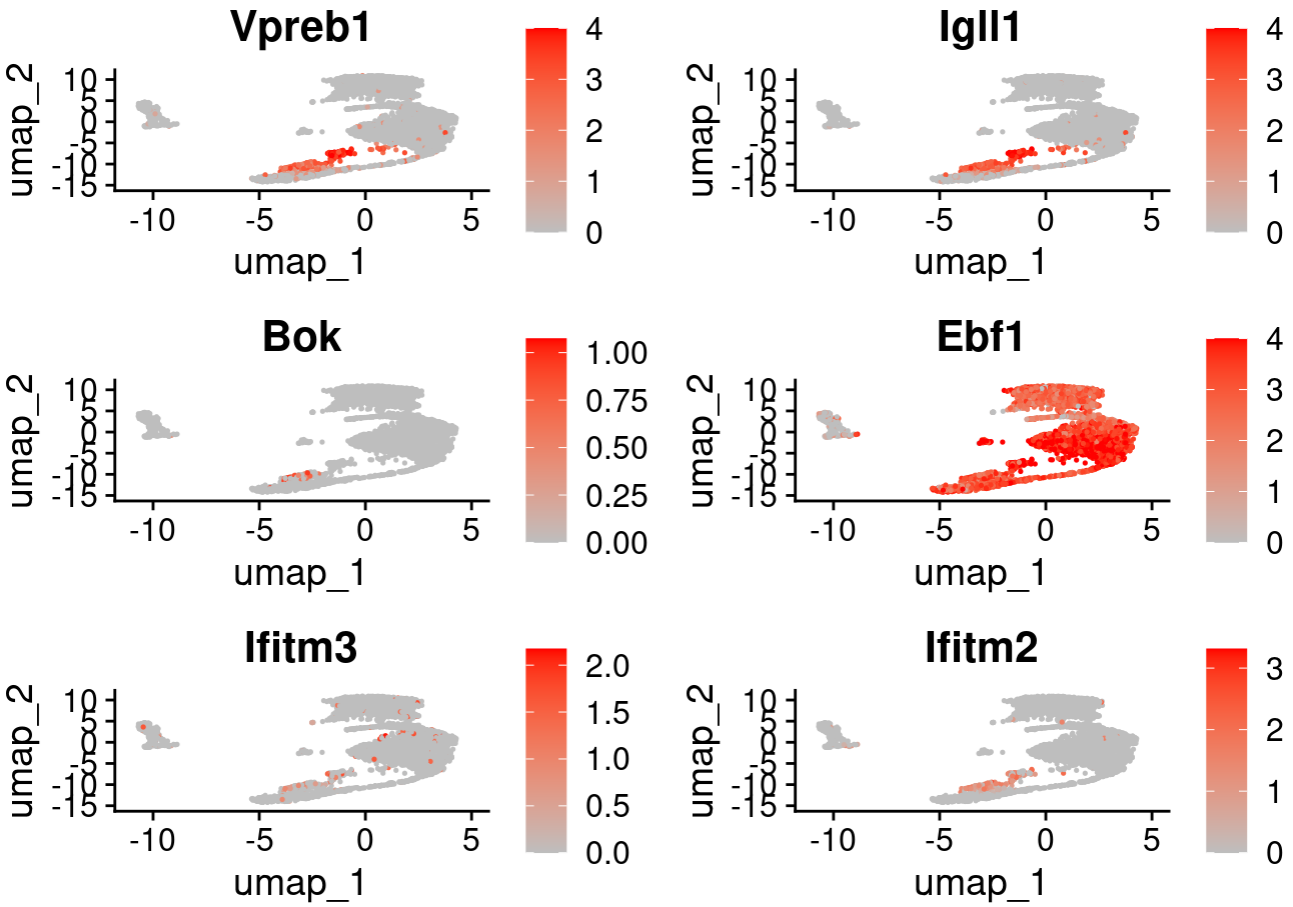

```
# List of genes for Pre-ProB signature
PrePro_marker_gene_list <- list(c("Flt3", "Tyrobp", "Irf8", "Il7r"))

# Calculate module for specified genes
Seurat_Object_BM_selected_Bcells <- AddModuleScore(
  object = Seurat_Object_BM_selected_Bcells,
  features = PrePro_marker_gene_list,
  name = "PrePro_score"
)

# Create feature Plot
FeaturePlot(Seurat_Object_BM_selected_Bcells, features = "PrePro_score1") +
  scale_color_viridis(option = "D") +
  ggtitle("PreProB Score", subtitle = "Genes: Flt3, Tyrobp, Irf8, Il7r") +
  theme(
    plot.title = element_text(family = "Times New Roman", size = 16),
    plot.subtitle = element_text(family = "Times New Roman", size = 14, hjust = 0.5)
  )
```

```
## Scale for colour is already present.
## Adding another scale for colour, which will replace the existing scale.
```

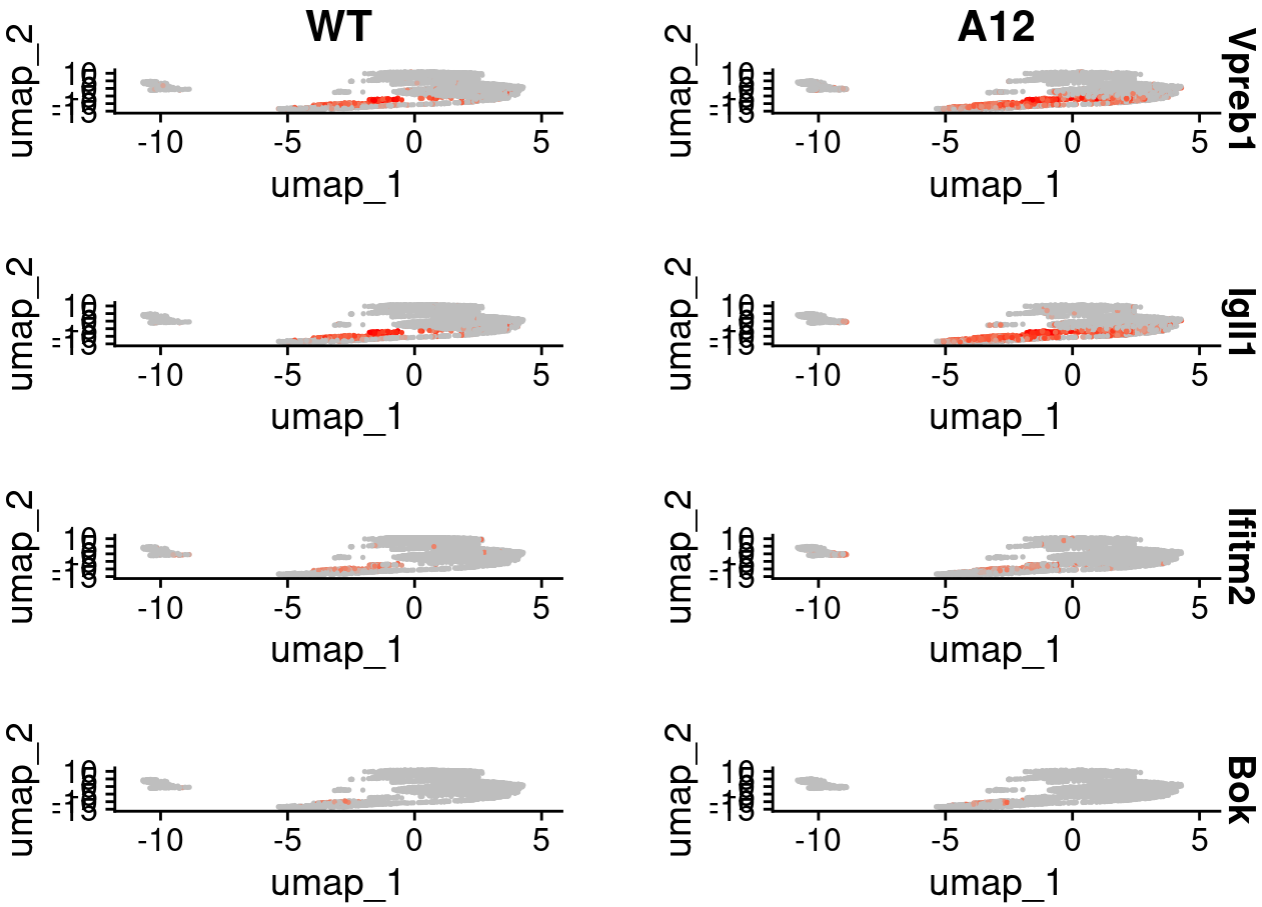


ProB

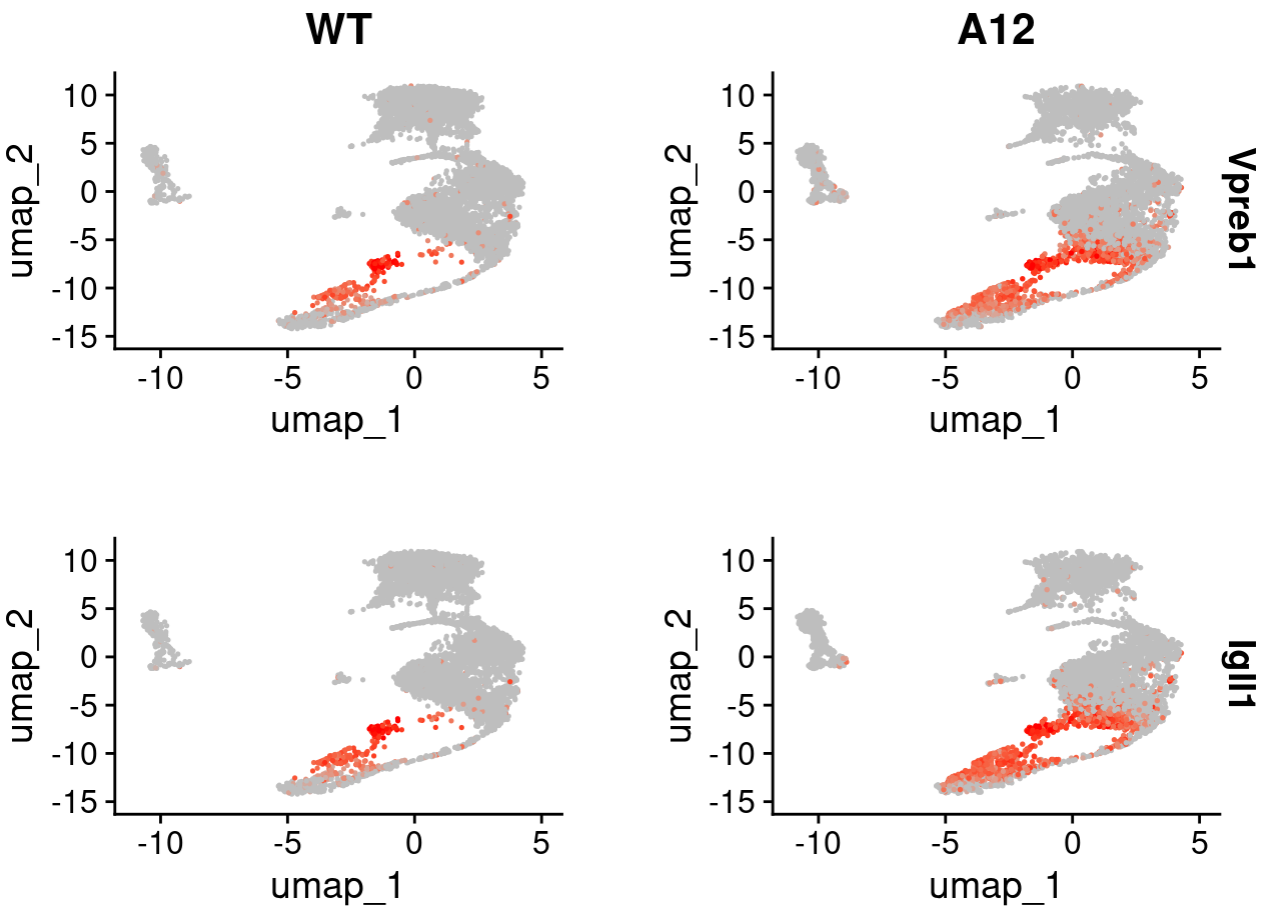
```
FeaturePlot(Seurat_Object_BM_WT, features = c("Vpreb1", "Igll1","Bok","Ebf1","Ifitm3","Ifitm2"
), max.cutoff = 4,
    cols = c("grey", "red"))
```




```
FeaturePlot(Seurat_Object_BM_selected_Bcells, features = c("Vpreb1","Igll1","Ifitm2", "Bok"),
split.by = "genotype", max.cutoff = 4,
    cols = c("grey", "red"))
```

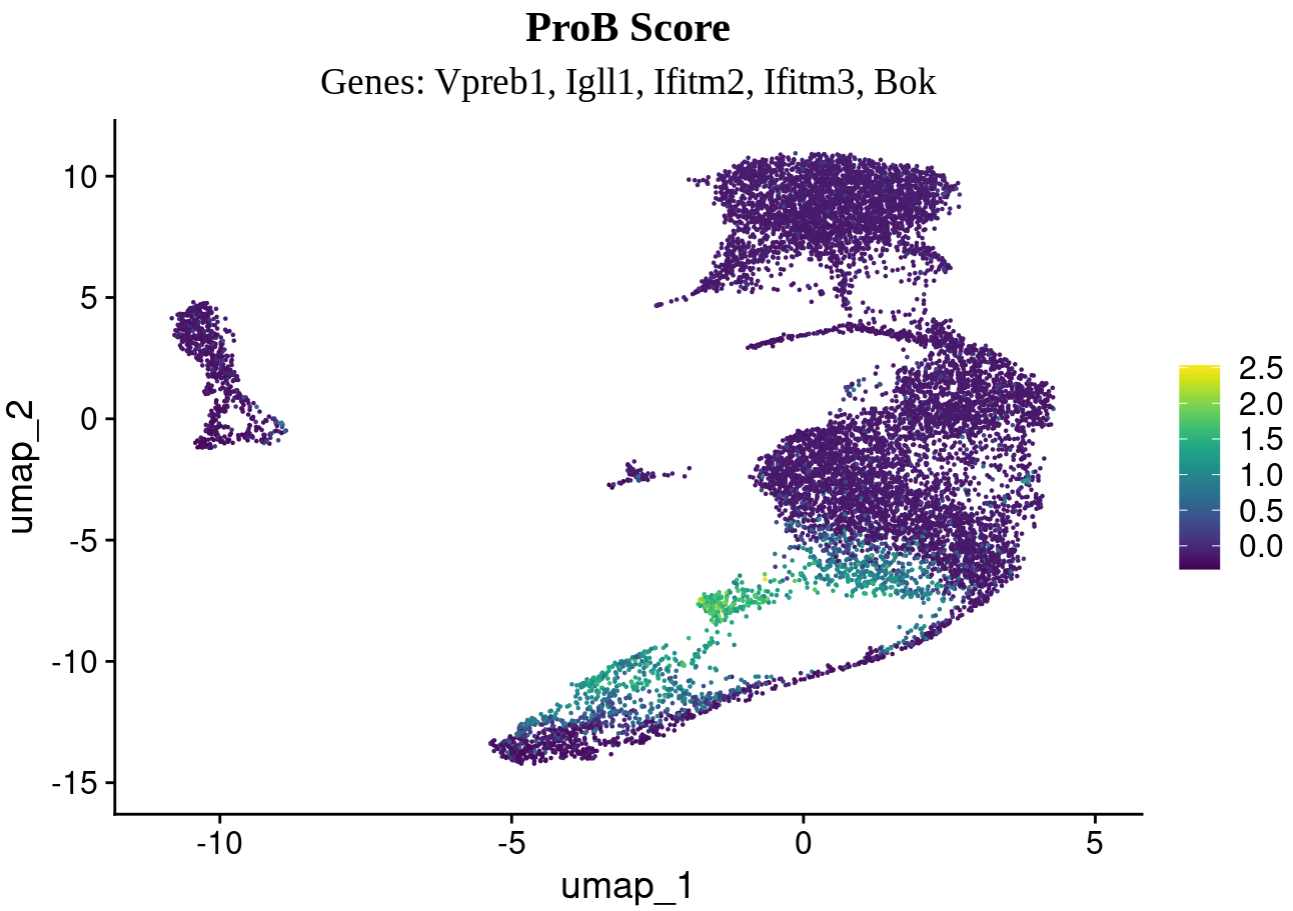



```
FeaturePlot(Seurat_Object_BM_selected_Bcells, features = c("Vpreb1","Igll1"), split.by = "geno
type", max.cutoff = 4,
    cols = c("grey", "red"))
```



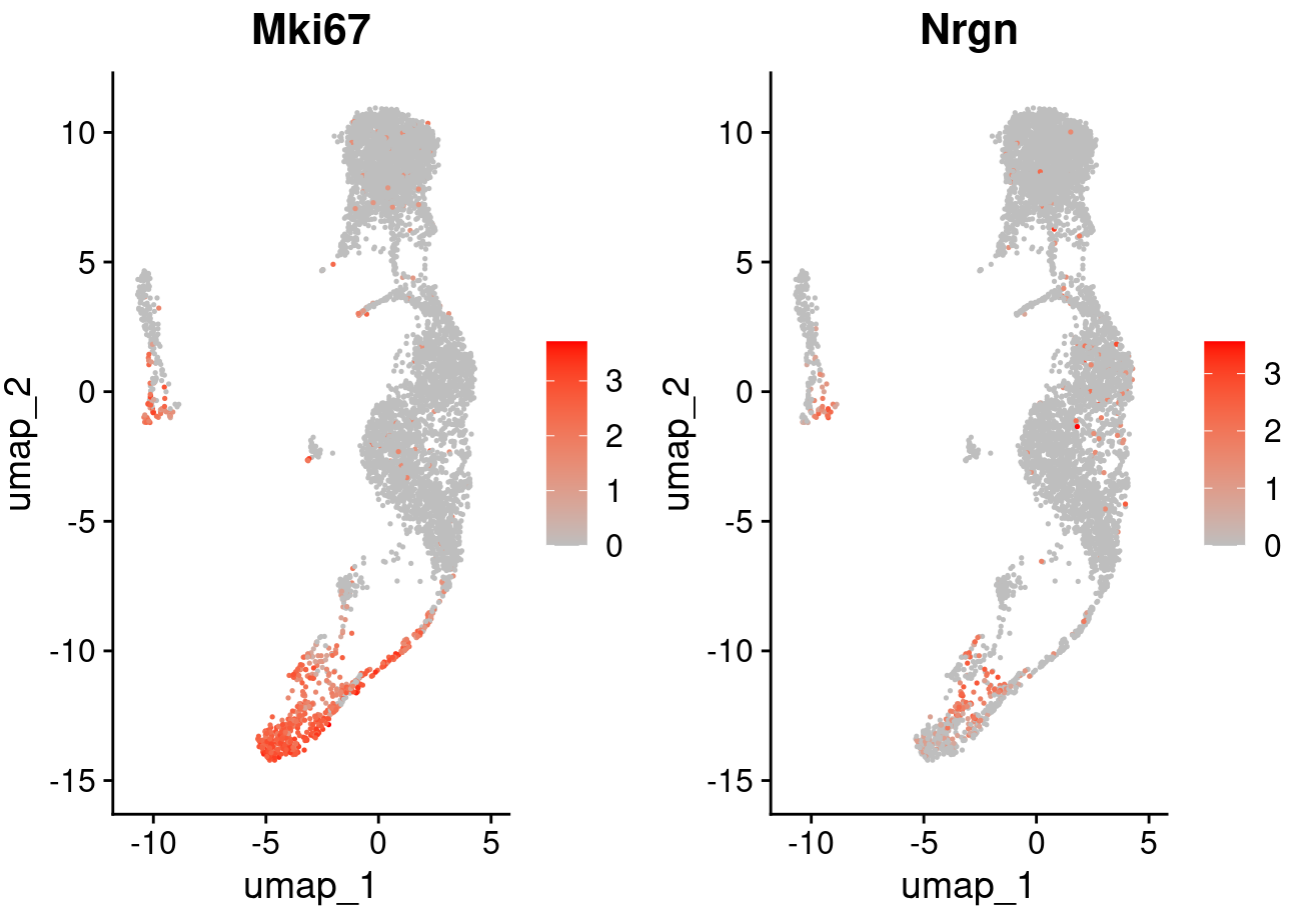

```
# List of genes for ProB subset
Pro_marker_gene_list <- list(c("Vpreb1","Igll1","Ifitm2", "Bok", "Ifitm3"))

# Calculate module for specified genes
Seurat_Object_BM_selected_Bcells <- AddModuleScore(
  object = Seurat_Object_BM_selected_Bcells,
  features = Pro_marker_gene_list,
  name = "Pro_score"
)

# Create feature Plot
FeaturePlot(Seurat_Object_BM_selected_Bcells, features = "Pro_score1") +
  scale_color_viridis(option = "D") +
  ggtitle("ProB Score", subtitle = "Genes: Vpreb1, Igll1, Ifitm2, Ifitm3, Bok") +
  theme(
    plot.title = element_text(family = "Times New Roman", size = 16),
    plot.subtitle = element_text(family = "Times New Roman", size = 14, hjust = 0.5)
  )
```

```
## Scale for colour is already present.
## Adding another scale for colour, which will replace the existing scale.
```

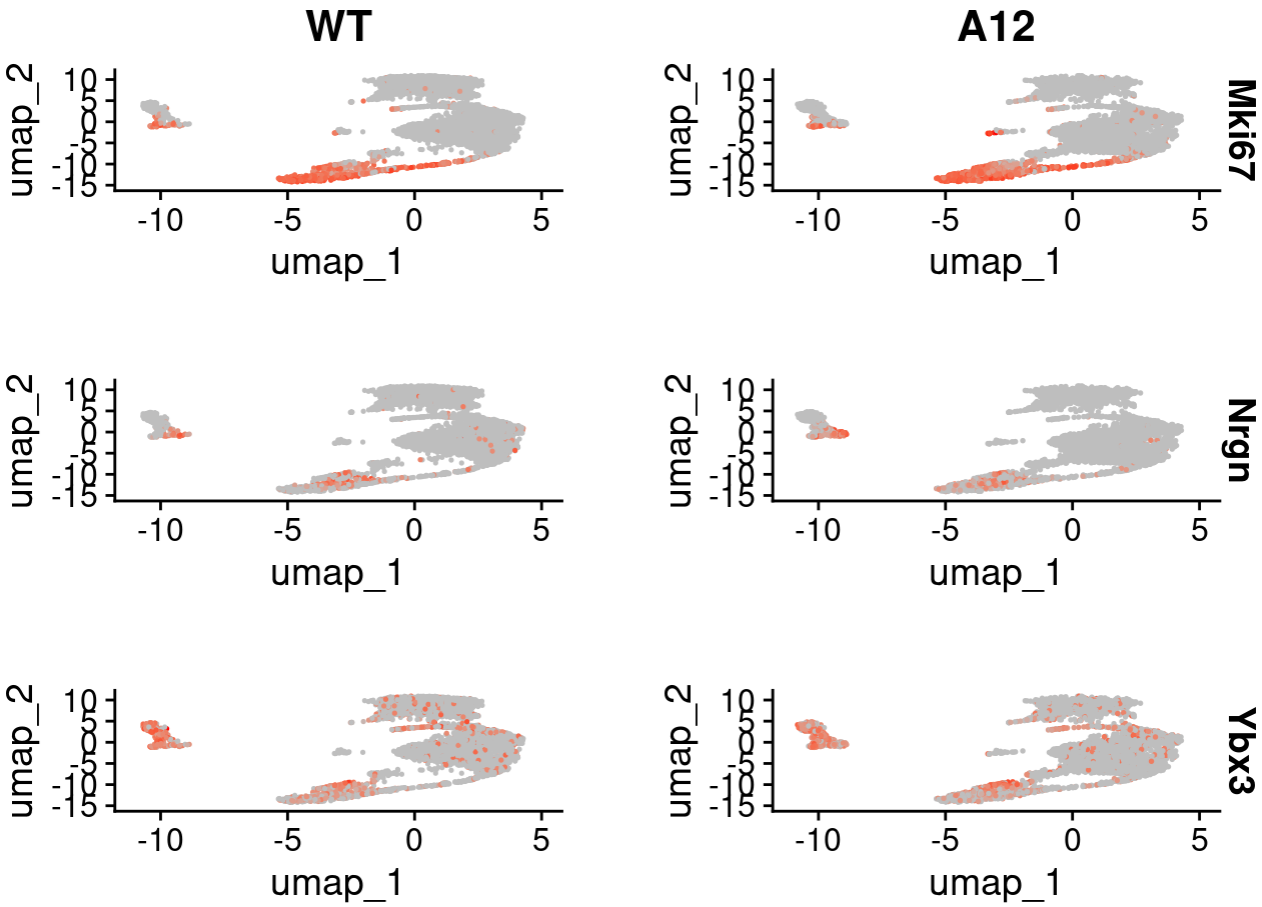


Large PreB identification

```
FeaturePlot(Seurat_Object_BM_WT, features = c("Mki67", "Nrgn"), max.cutoff = 4,
    cols = c("grey", "red"))
```

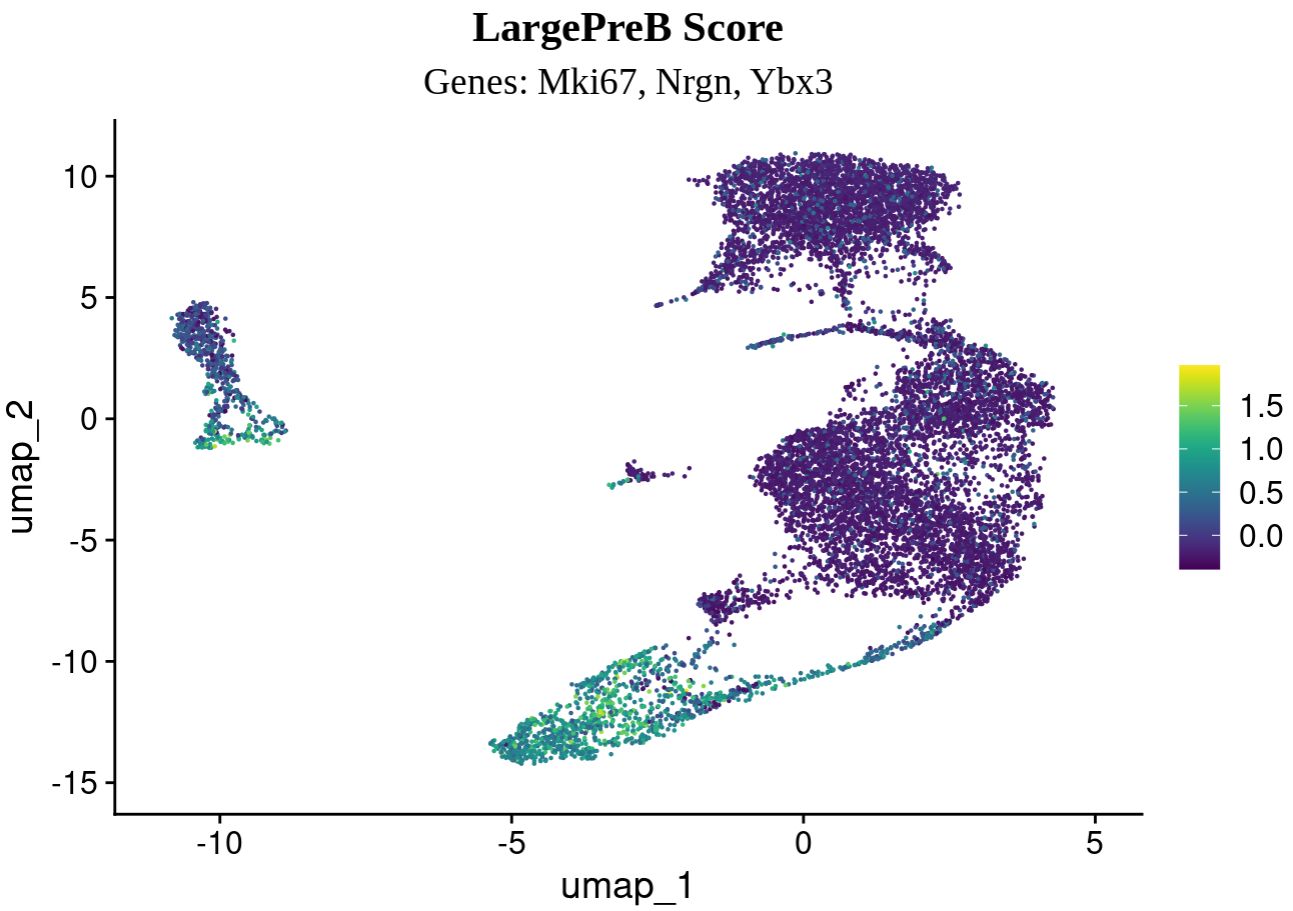



```
FeaturePlot(Seurat_Object_BM_selected_Bcells, features = c("Mki67", "Nrgn", "Ybx3"), split.by
= "genotype", max.cutoff = 4,
    cols = c("grey", "red"))
```



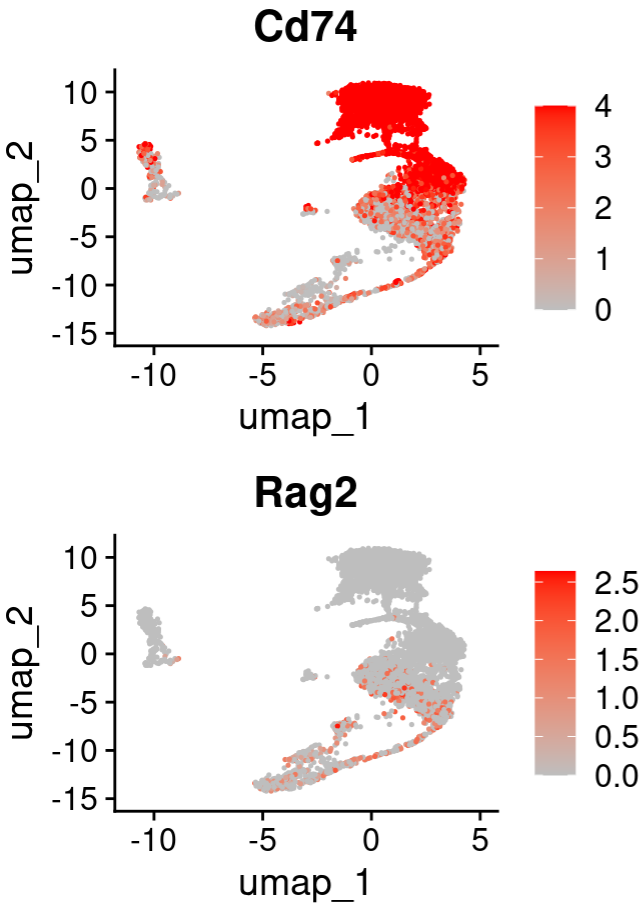

```
# List of genes for Large PreB signature
LargePreB_marker_gene_list <- list(c("Mki67", "Nrgn", "Ybx3"))

# Calculate module for specified genes
Seurat_Object_BM_selected_Bcells <- AddModuleScore(
  object = Seurat_Object_BM_selected_Bcells,
  features = LargePreB_marker_gene_list,
  name = "LargePreB_score"
)

# Create feature Plot
FeaturePlot(Seurat_Object_BM_selected_Bcells, features = "LargePreB_score1") +
  scale_color_viridis(option = "D") +
  ggtitle("LargePreB Score", subtitle = "Genes: Mki67, Nrgn, Ybx3") +
  theme(
    plot.title = element_text(family = "Times New Roman", size = 16),
    plot.subtitle = element_text(family = "Times New Roman", size = 14, hjust = 0.5)
  )
```

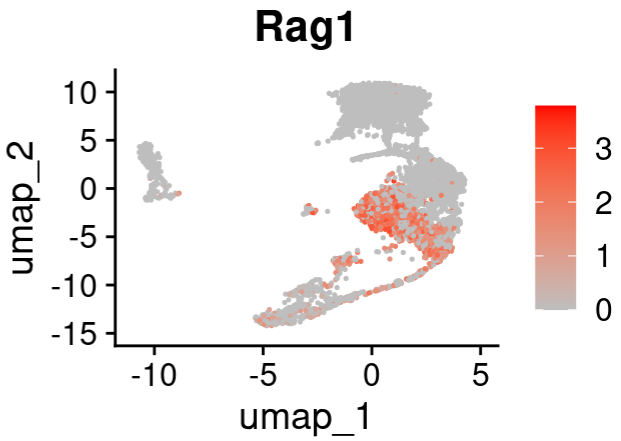
```
## Scale for colour is already present.
## Adding another scale for colour, which will replace the existing scale.
```



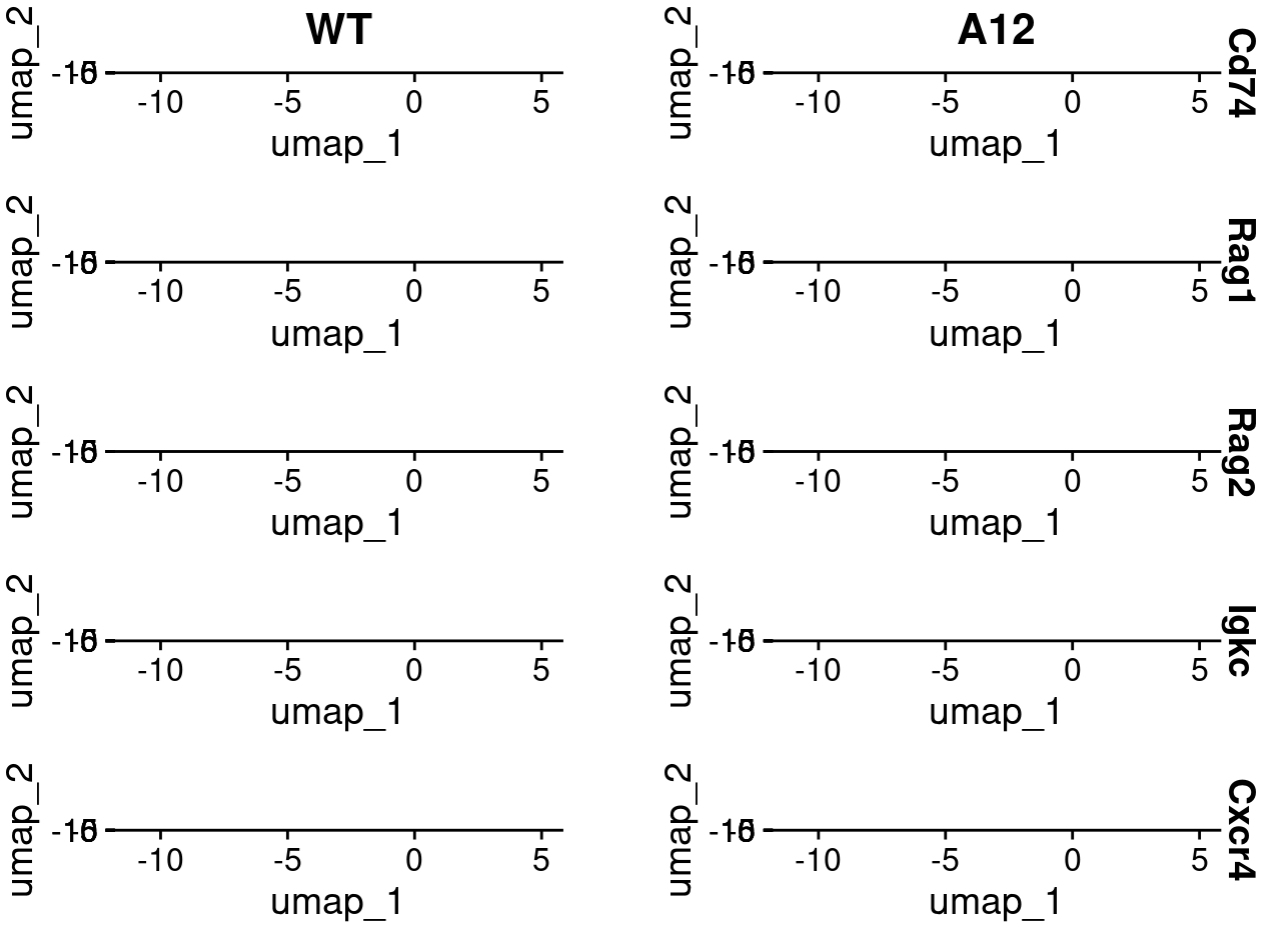
Small preB gene signature

```
FeaturePlot(Seurat_Object_BM_WT, features = c("Cd74", "Rag1", "Rag2"), max.cutoff = 4,
    cols = c("grey", "red"))
```



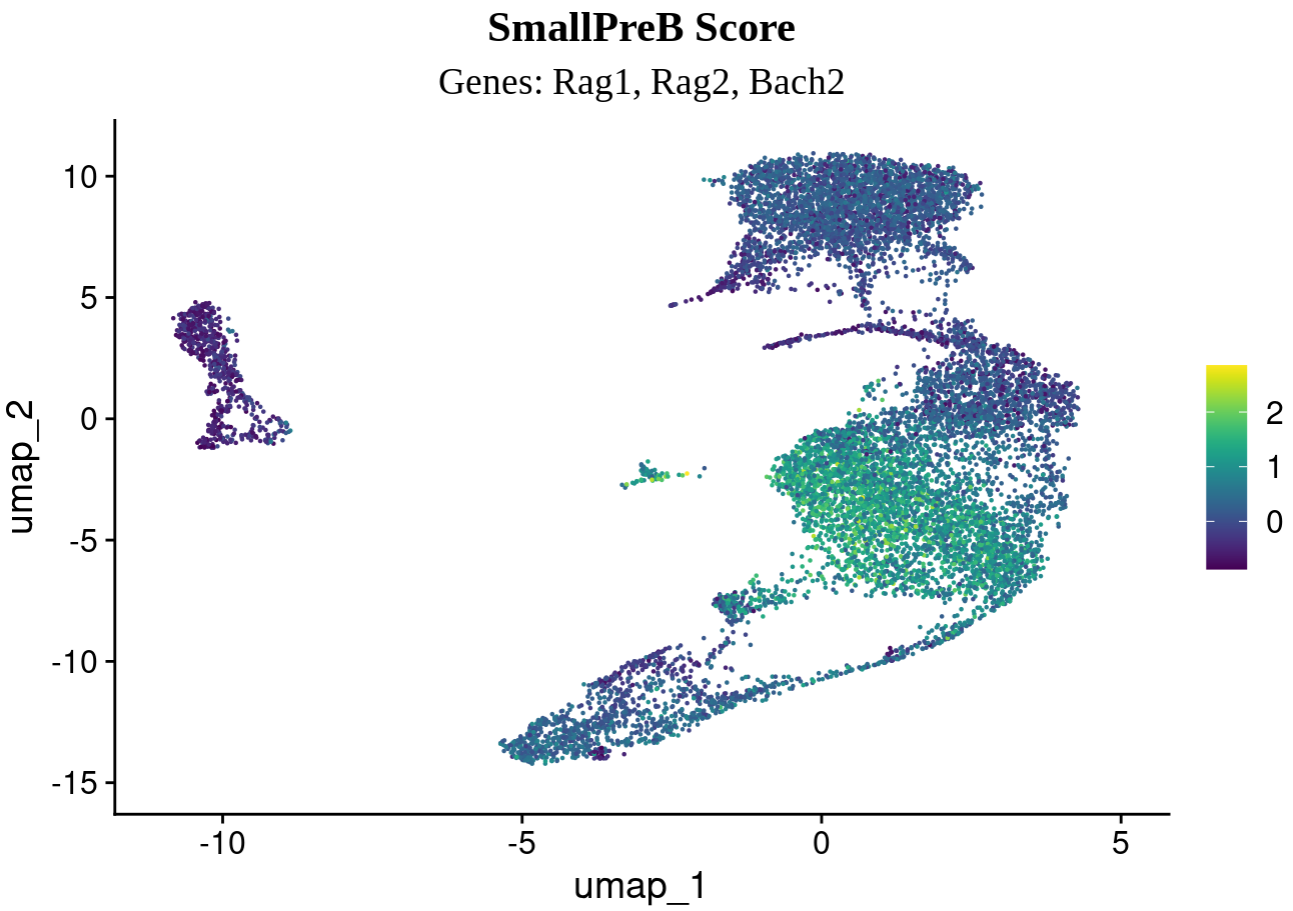



```
FeaturePlot(Seurat_Object_BM_selected_Bcells, features = c("Cd74", "Rag1", "Rag2", "Igkc", "Cxcr4"), split.by = "genotype", max.cutoff = 4,
    cols = c("grey", "red"))
```



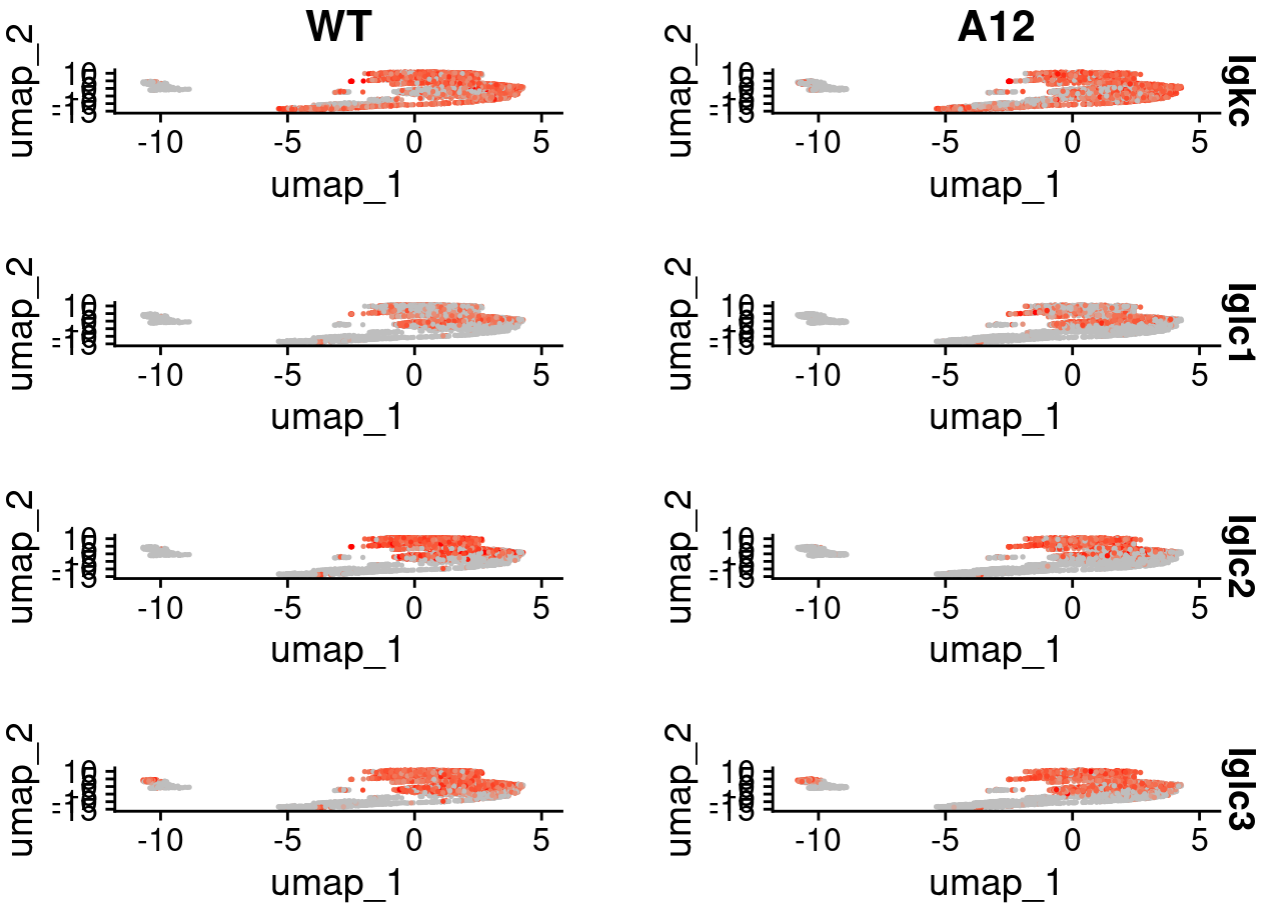

```
# List of genes for small PreB
SmallPreB_marker_gene_list <- list(c("Rag1", "Rag2", "Bach2"))

# Calculate module for specified genes
Seurat_Object_BM_selected_Bcells <- AddModuleScore(
  object = Seurat_Object_BM_selected_Bcells,
  features = SmallPreB_marker_gene_list,
  name = "SmallPreB_score"
)

# Create feature Plot
FeaturePlot(Seurat_Object_BM_selected_Bcells, features = "SmallPreB_score1") +
  scale_color_viridis(option = "D") +
  ggtitle("SmallPreB Score", subtitle = "Genes: Rag1, Rag2, Bach2") +
  theme(
    plot.title = element_text(family = "Times New Roman", size = 16),
    plot.subtitle = element_text(family = "Times New Roman", size = 14, hjust = 0.5)
  )
```

```
## Scale for colour is already present.
## Adding another scale for colour, which will replace the existing scale.
```

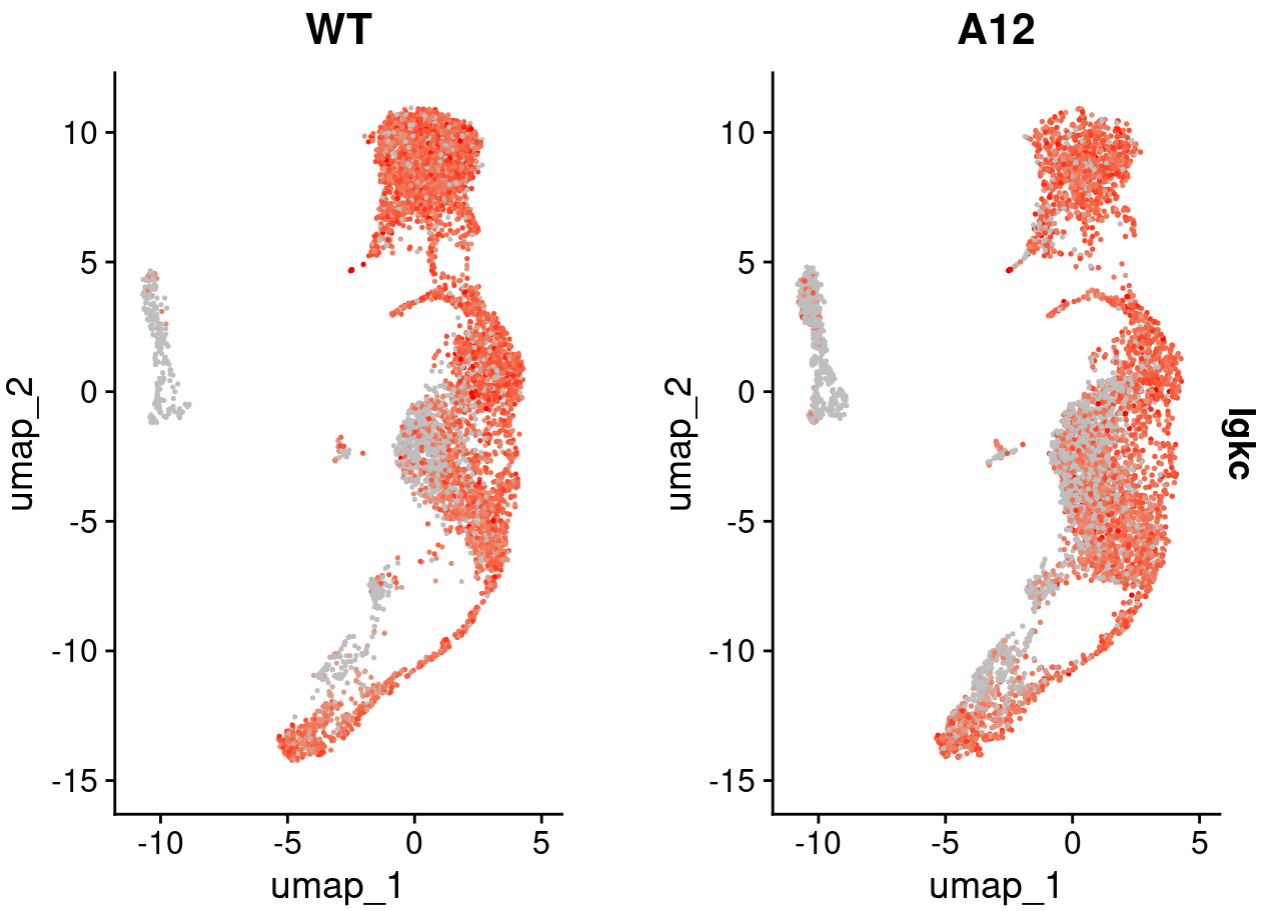


Identificación Immature

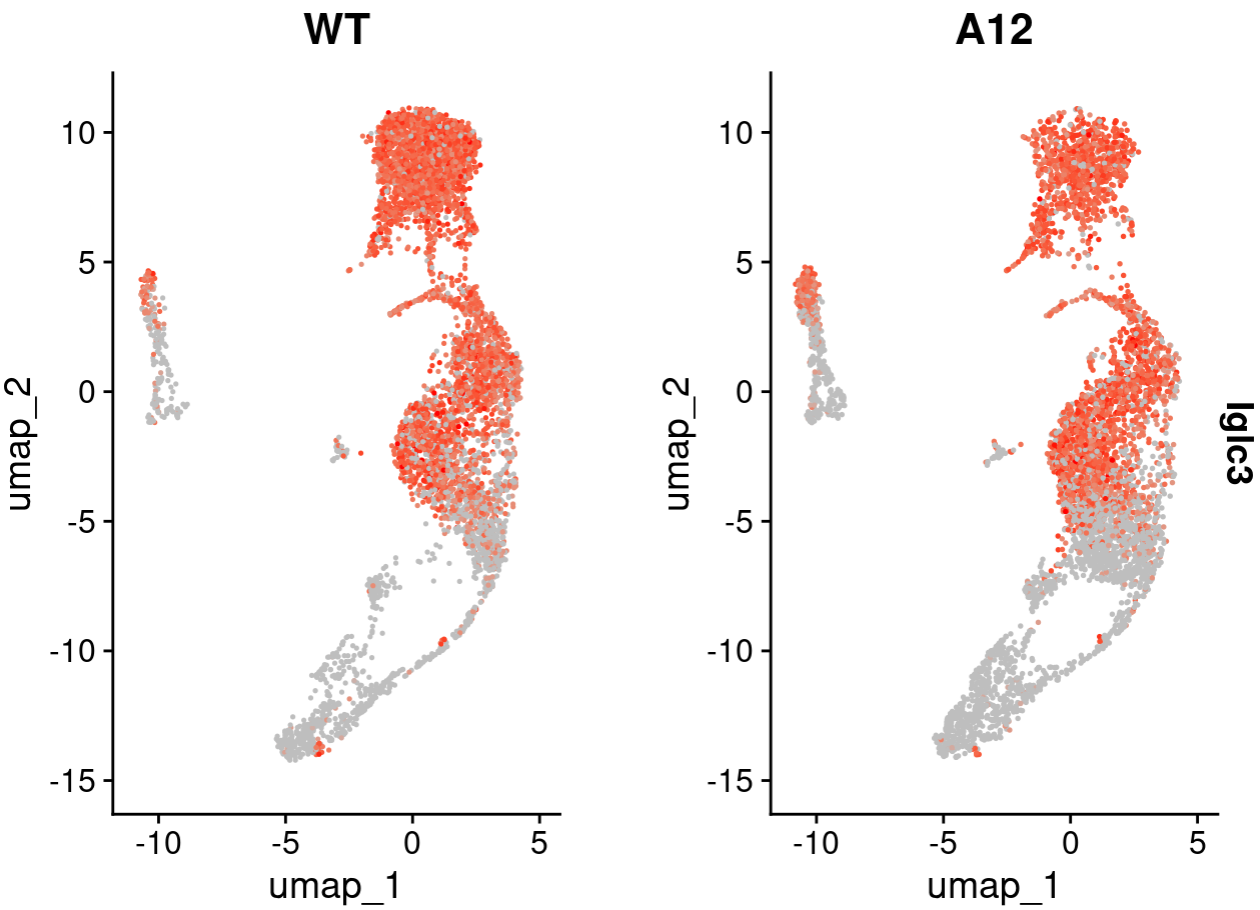
```
FeaturePlot(Seurat_Object_BM_selected_Bcells, features = c("Igkc", "Iglc1", "Iglc2", "Iglc3"),
 split.by = "genotype", max.cutoff = 4,
    cols = c("grey", "red"))
```



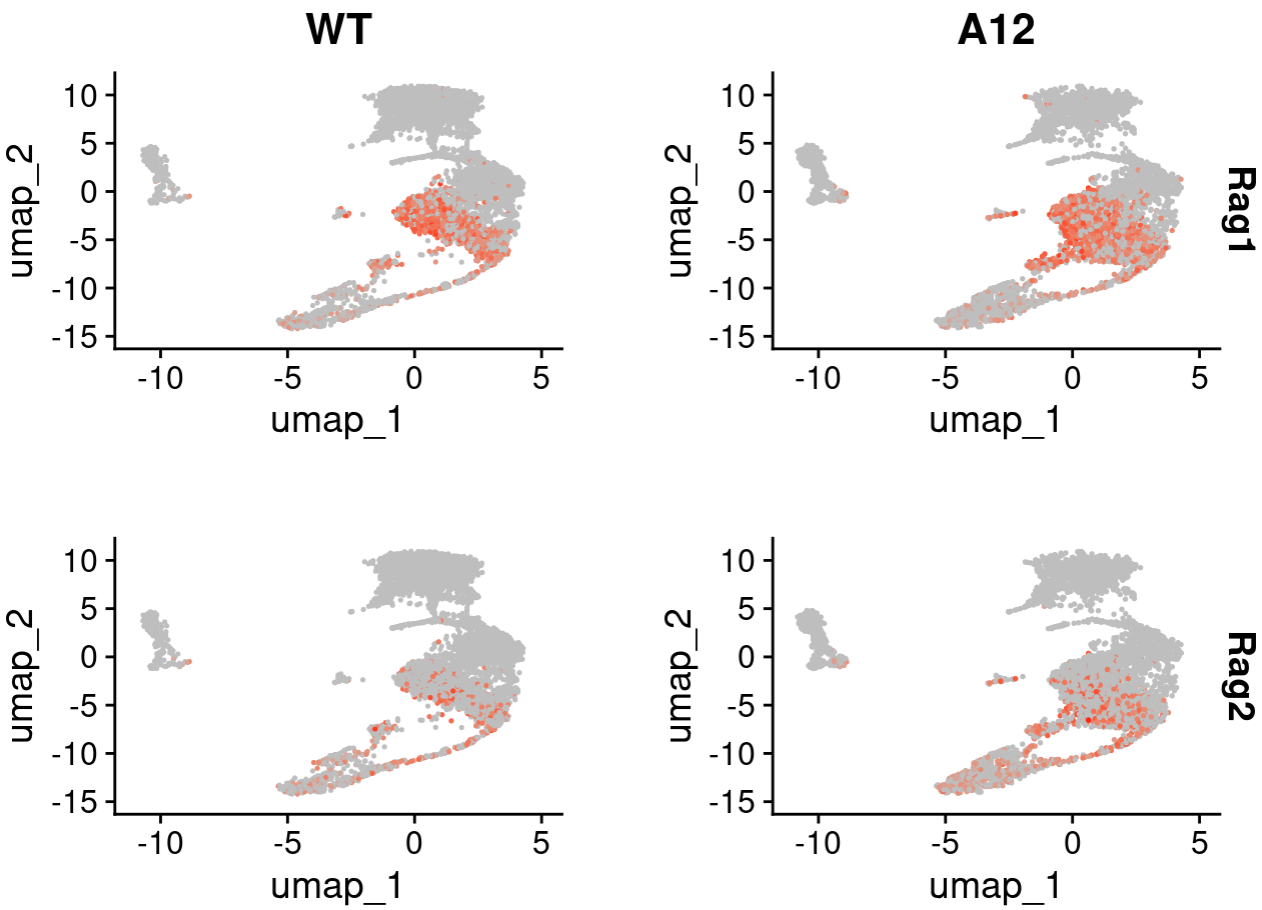

```
FeaturePlot(Seurat_Object_BM_selected_Bcells, features = c("Igkc"), split.by = "genotype", max
.cutoff = 4,
    cols = c("grey", "red"))
```



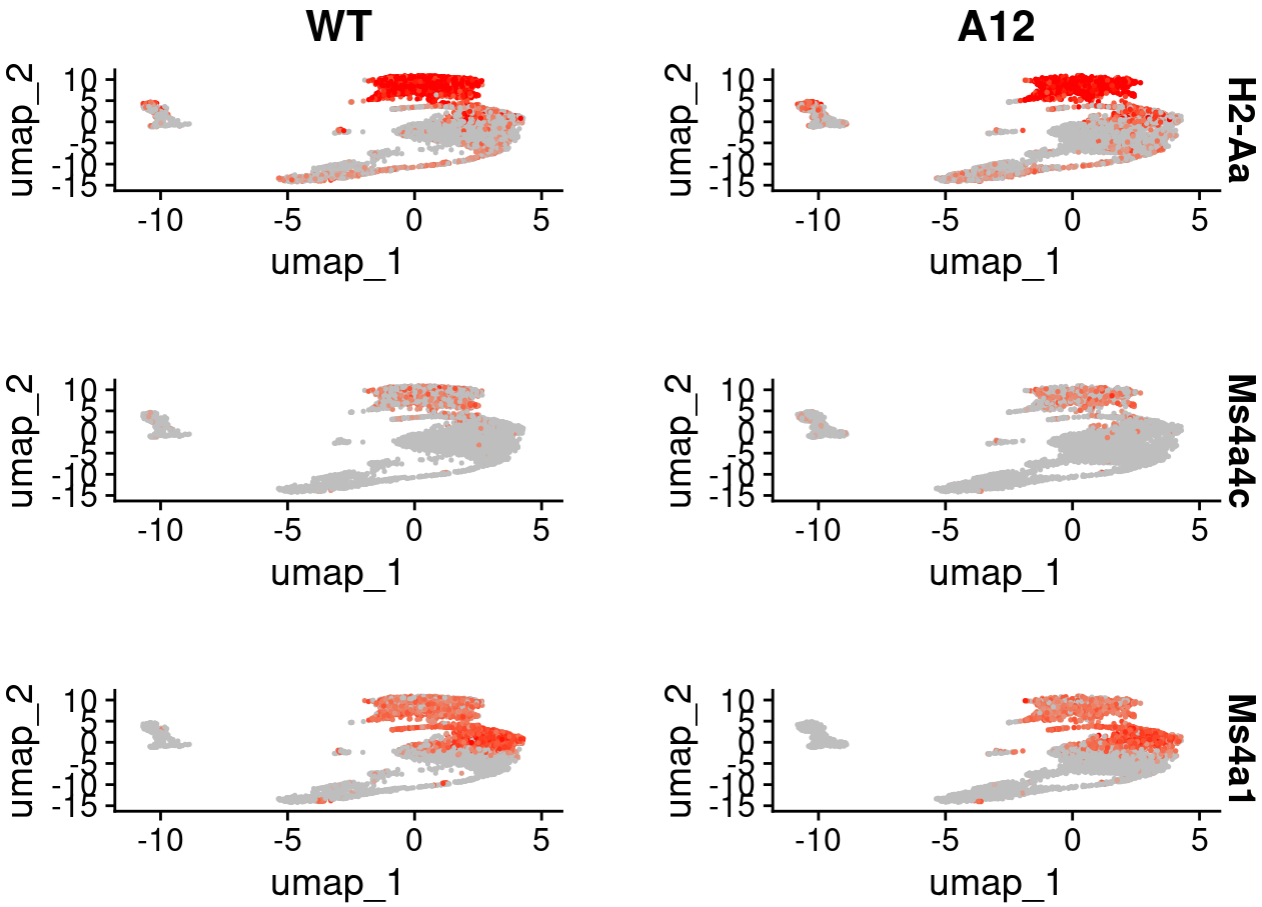

```
FeaturePlot(Seurat_Object_BM_selected_Bcells, features = c("Iglc3"), split.by = "genotype", ma
x.cutoff = 4,
    cols = c("grey", "red"))
```



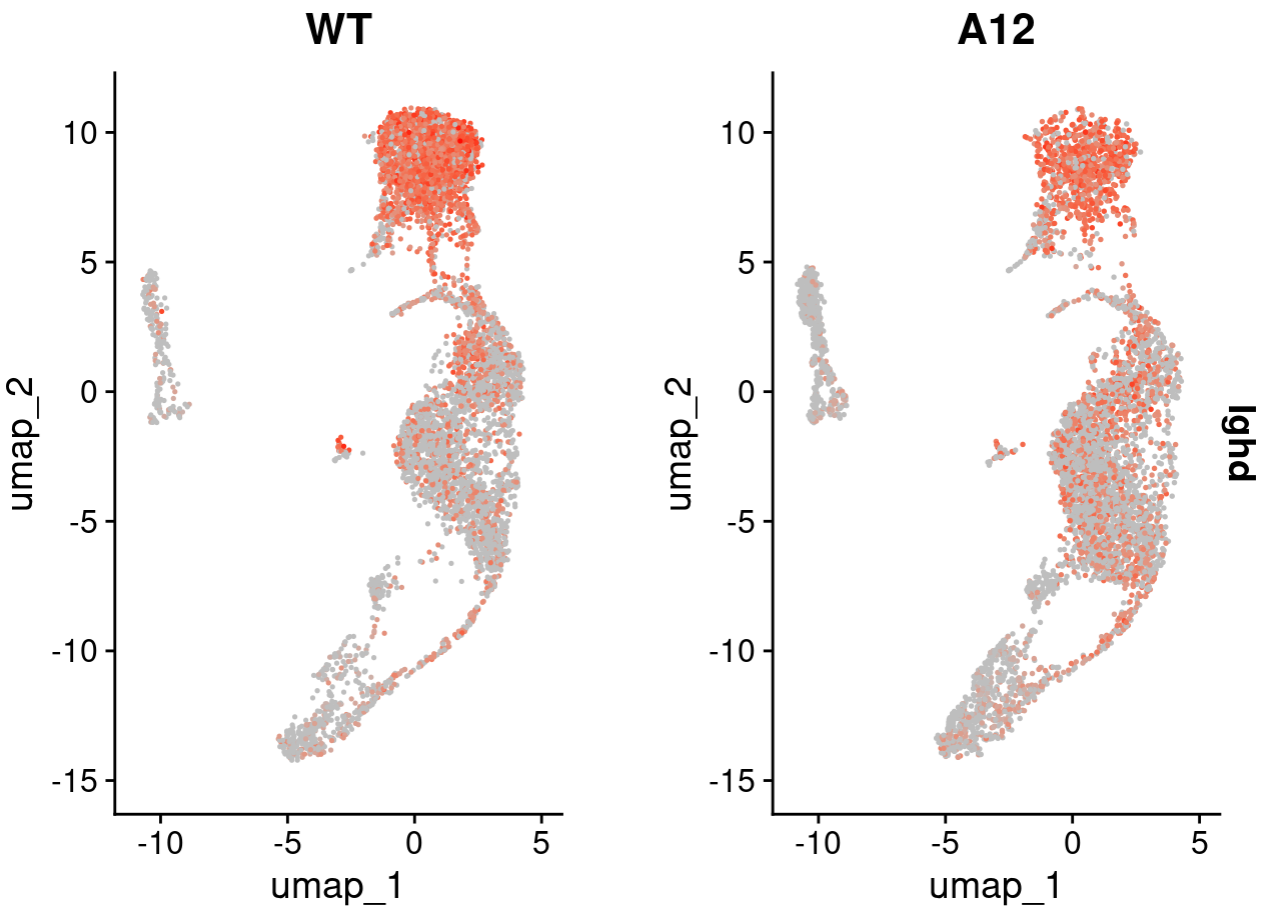

```
FeaturePlot(Seurat_Object_BM_selected_Bcells, features = c("Rag1", "Rag2"), split.by = "genoty
pe", max.cutoff = 4,
    cols = c("grey", "red"))
```




```
FeaturePlot(Seurat_Object_BM_selected_Bcells, features = c("H2-Aa", "Ms4a4c", "Ms4a1"), split.
by = "genotype", max.cutoff = 4,
    cols = c("grey", "red"))
```




```
FeaturePlot(Seurat_Object_BM_selected_Bcells, features = c("Ighd"), split.by = "genotype", max
.cutoff = 4,
    cols = c("grey", "red"))
```

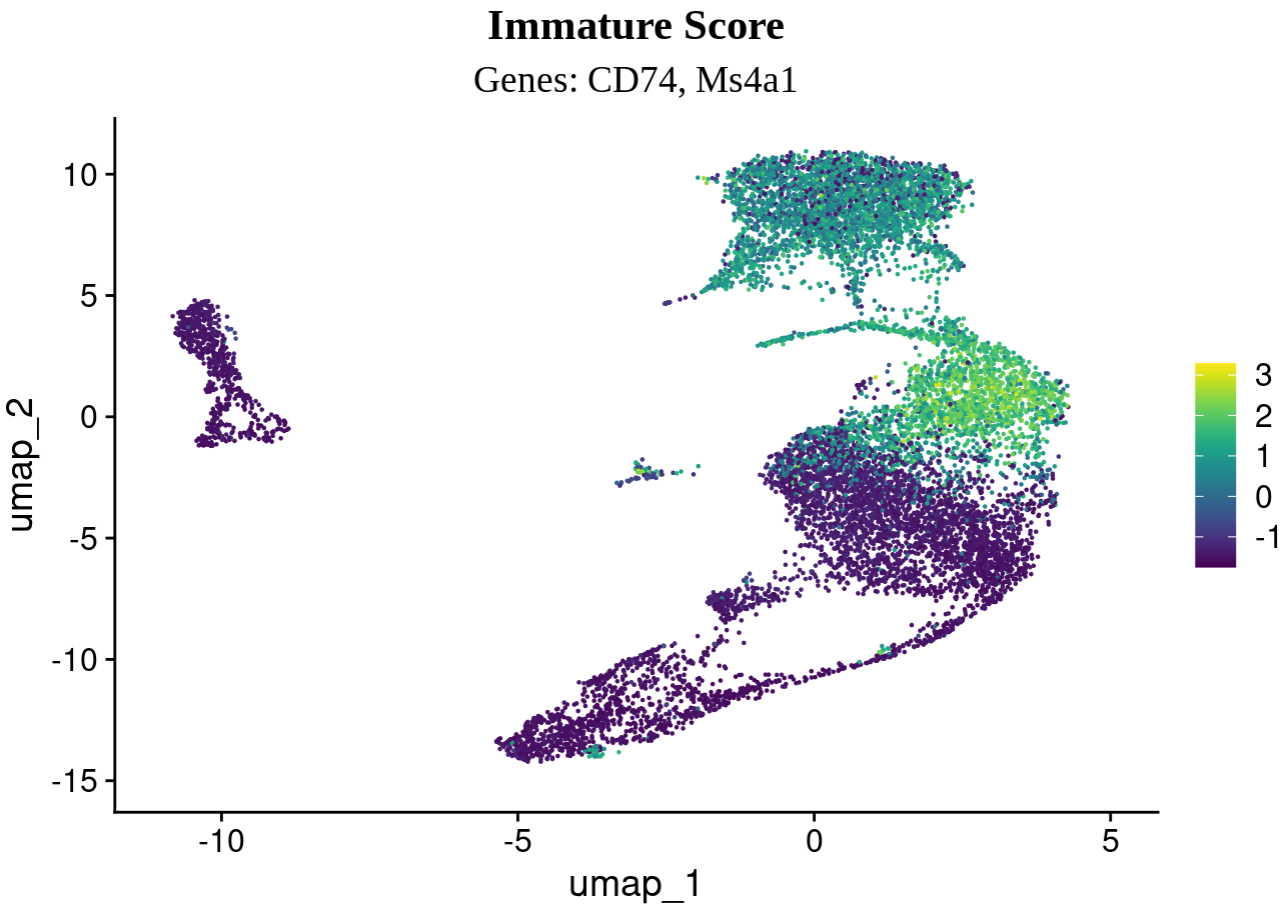



```
# List of genes for Immature
Immature_marker_gene_list <- list(c("CD74", "Ms4a1"))

# Calculate module for specified genes
Seurat_Object_BM_selected_Bcells <- AddModuleScore(
  object = Seurat_Object_BM_selected_Bcells,
  features = Immature_marker_gene_list,
  name = "Immature_score"
)
```

```
## Warning: The following features are not present in the object: CD74, not
## searching for symbol synonyms
```

```
# Create feature plot
FeaturePlot(Seurat_Object_BM_selected_Bcells, features = "Immature_score1") +
  scale_color_viridis(option = "D") +
  ggtitle("Immature Score", subtitle = "Genes: CD74, Ms4a1") +
  theme(
    plot.title = element_text(family = "Times New Roman", size = 16),
    plot.subtitle = element_text(family = "Times New Roman", size = 14, hjust = 0.5)
  )
```

```
## Scale for colour is already present.
## Adding another scale for colour, which will replace the existing scale.
```



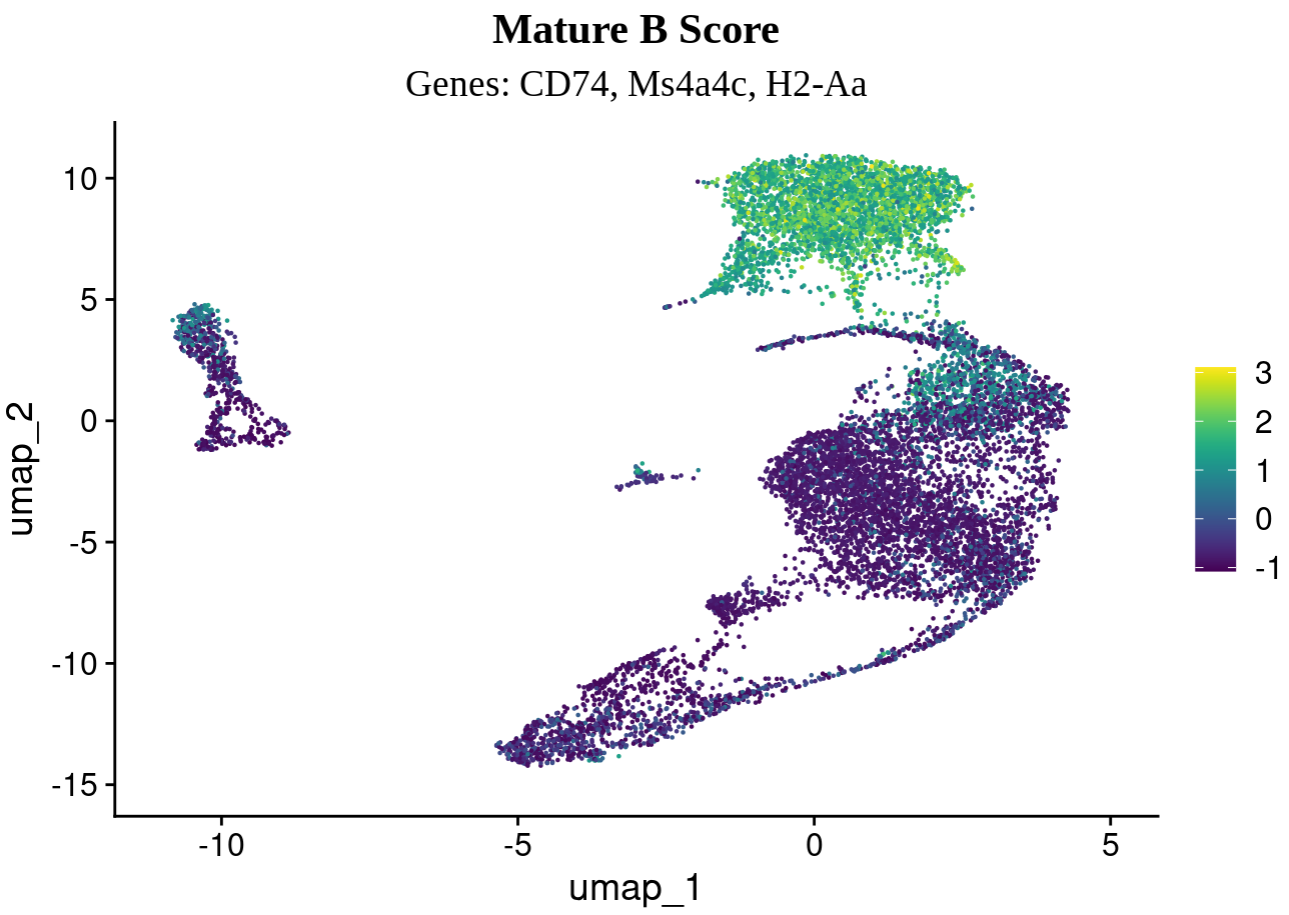

```
# List of genes for Mature B
Mature_marker_gene_list <- list(c("CD74", "H2-Aa", "Ms4a4c"))

#Calculate module for specified genes
Seurat_Object_BM_selected_Bcells <- AddModuleScore(
  object = Seurat_Object_BM_selected_Bcells,
  features = Mature_marker_gene_list,
  name = "Mature_score"
)
```

```
## Warning: The following features are not present in the object: CD74, not
## searching for symbol synonyms
```

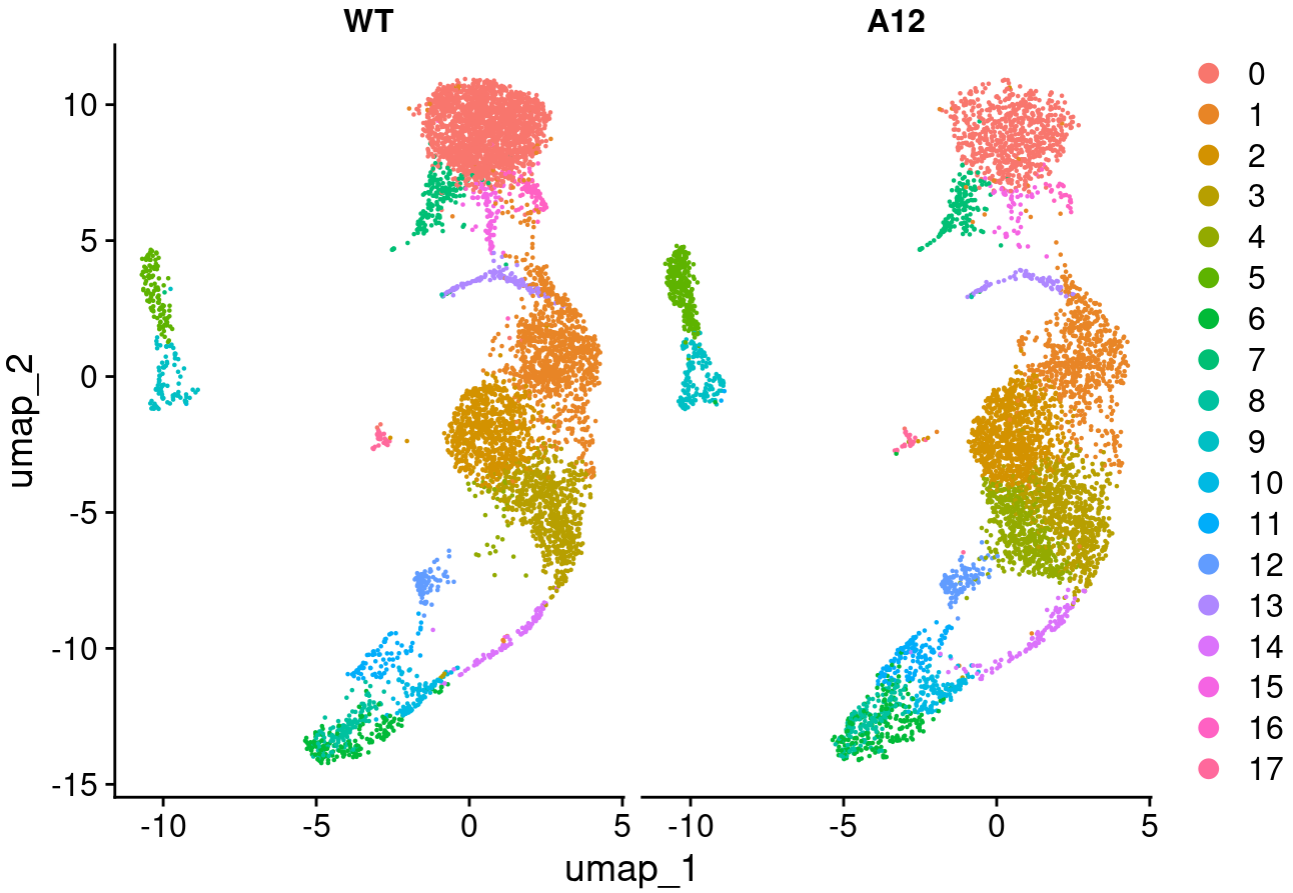
```
# Create Feature Plot
FeaturePlot(Seurat_Object_BM_selected_Bcells, features = "Mature_score1") +
  scale_color_viridis(option = "D") +
  ggtitle("Mature B Score", subtitle = "Genes: CD74, Ms4a4c, H2-Aa") +
  theme(
    plot.title = element_text(family = "Times New Roman", size = 16),
    plot.subtitle = element_text(family = "Times New Roman", size = 14, hjust = 0.5)
  )
```

```
## Scale for colour is already present.
```

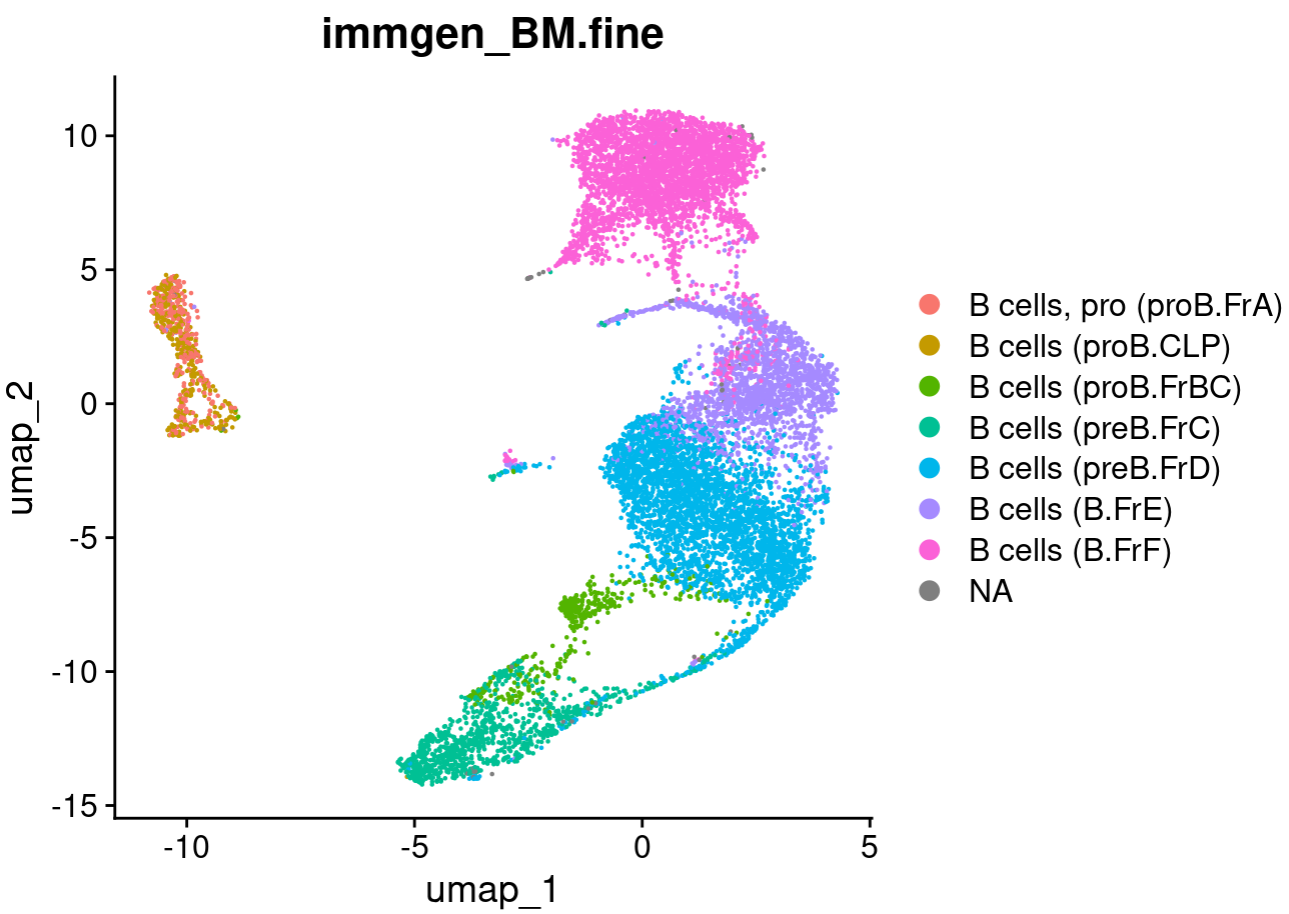
```
## Adding another scale for colour, which will replace the existing scale.
```



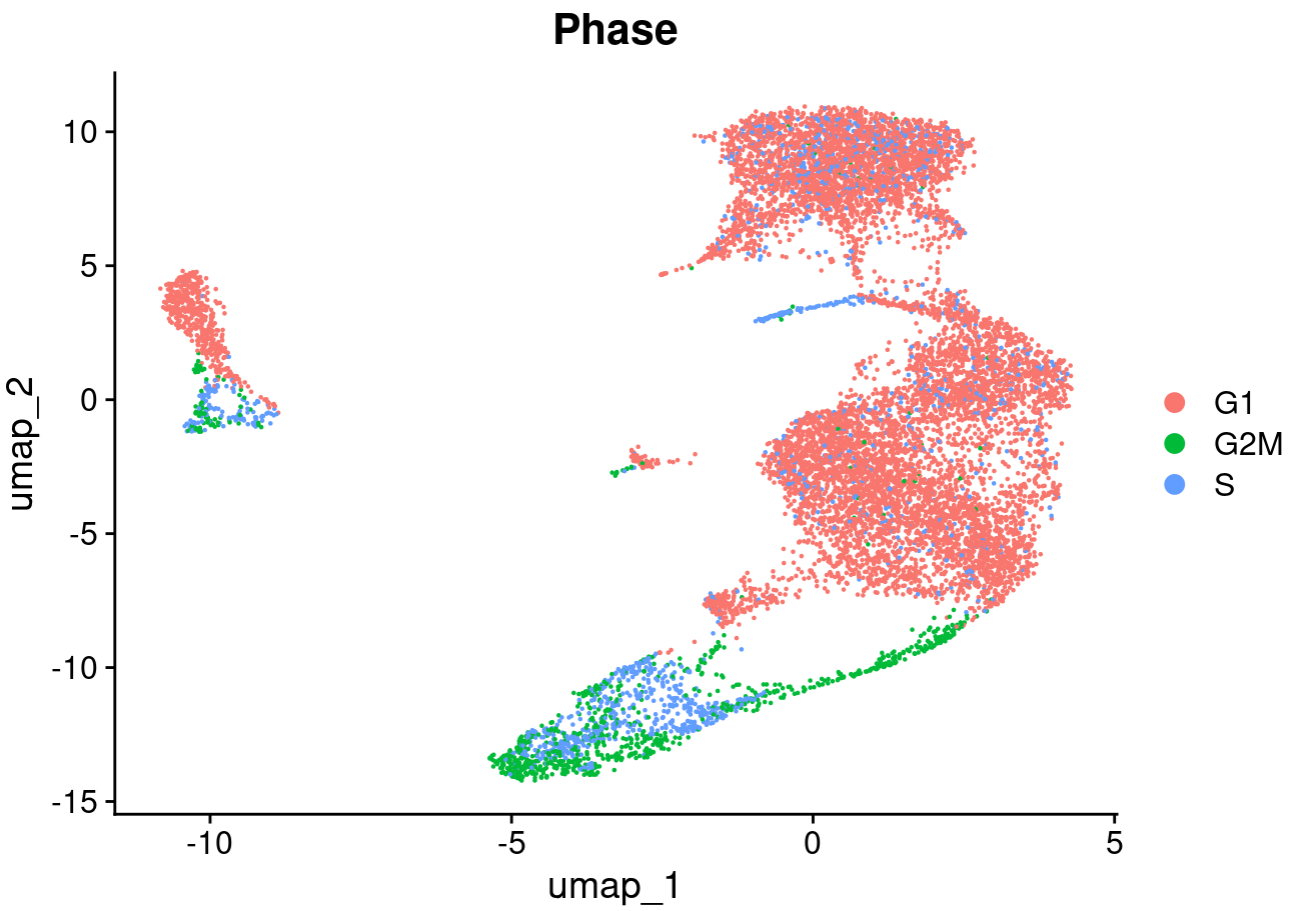

```
DimPlot(Seurat_Object_BM_selected_Bcells, split.by = "genotype")
```




```
DimPlot(Seurat_Object_BM_selected_Bcells, group.by = "immgen_BM.fine")
```

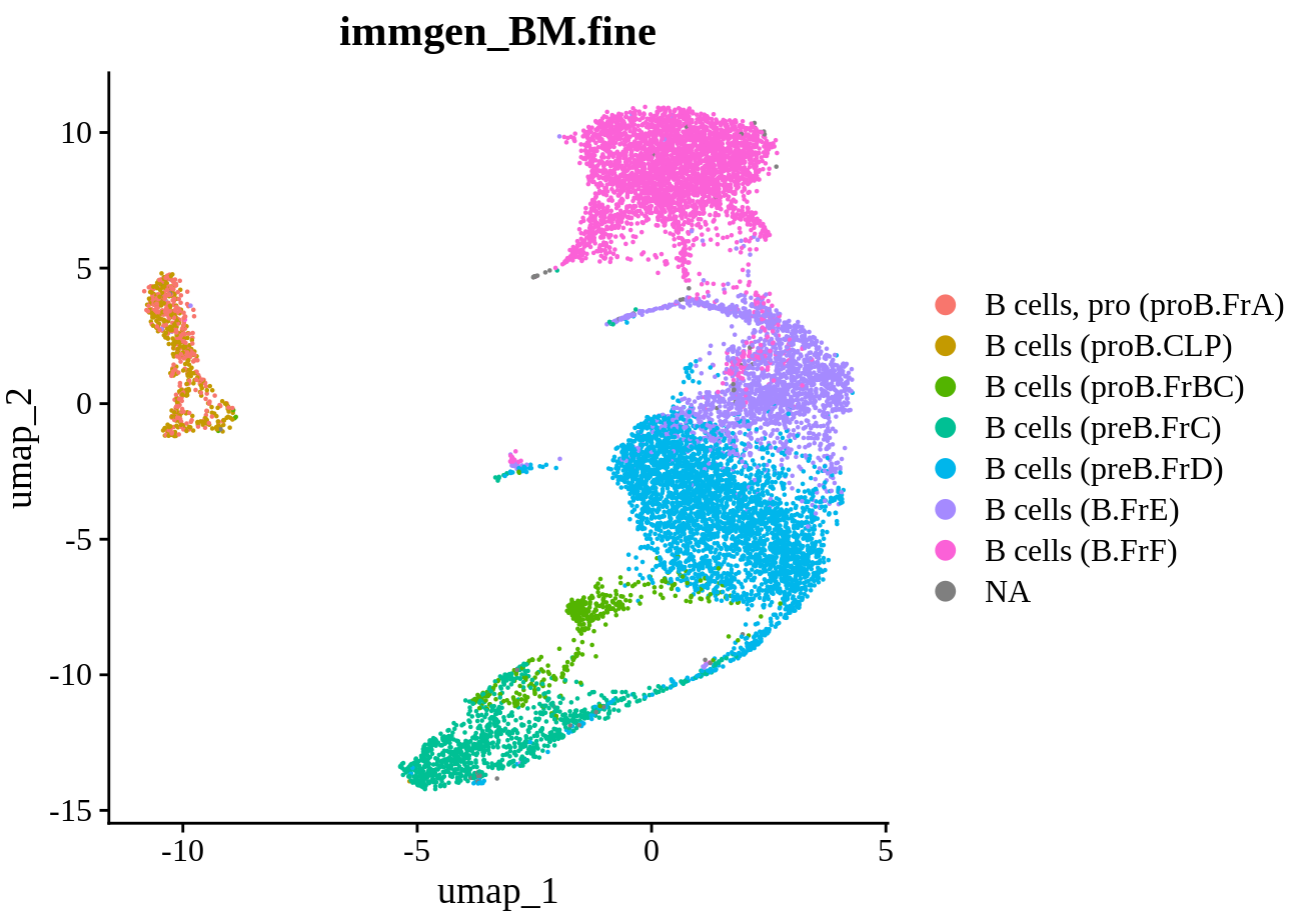



```
DimPlot(Seurat_Object_BM_selected_Bcells, group.by = "Phase")
```

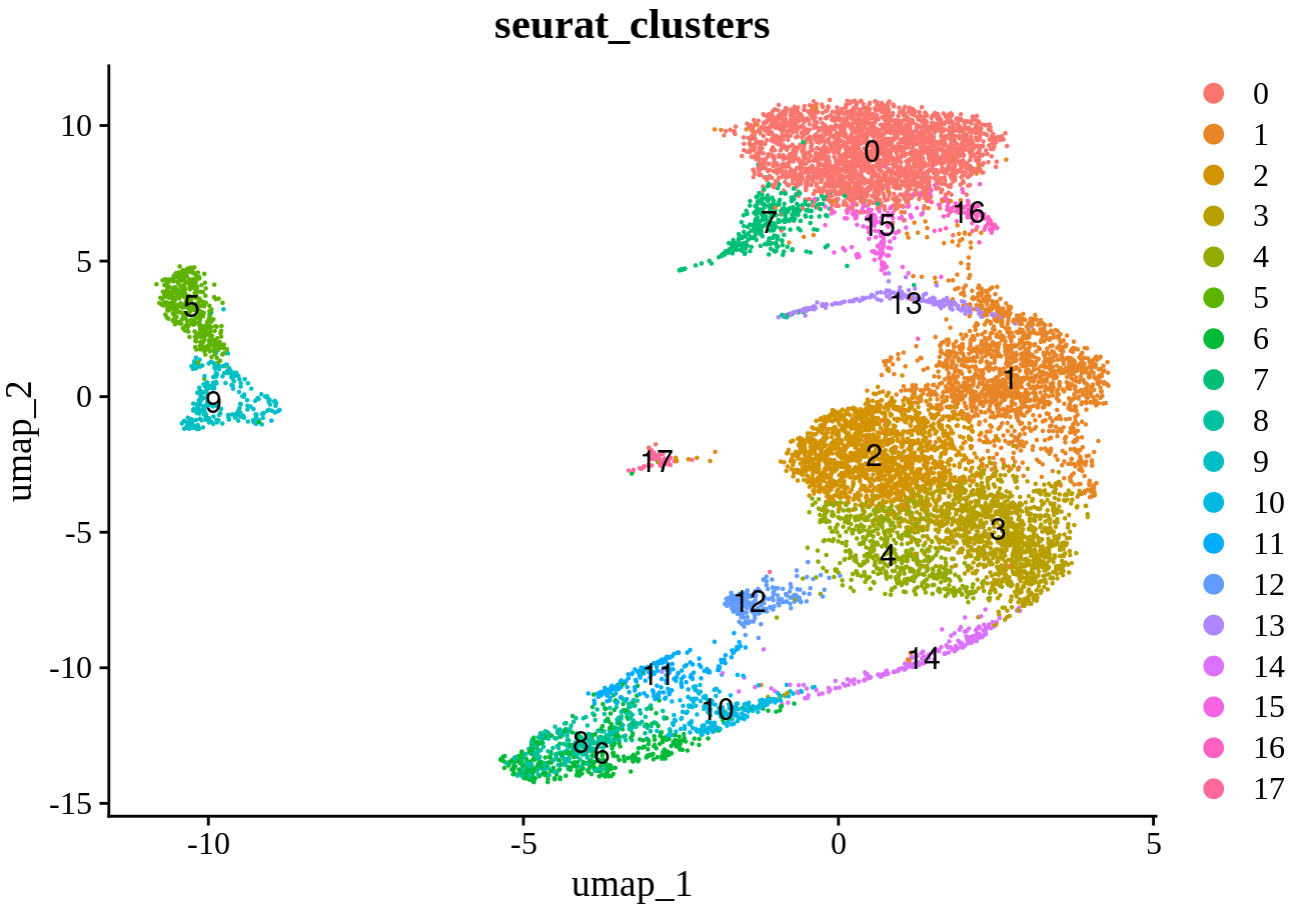


Rename clusters and analysis for BM

```
DimPlot(Seurat_Object_BM_selected_Bcells, reduction = "umap", group.by = "immgen_BM.fine") +
    theme(
        text = element_text(family = "Times New Roman")
    )
```




```
DimPlot(Seurat_Object_BM_selected_Bcells, reduction = "umap", group.by = "seurat_clusters", la
bel = TRUE) +
    theme(
        text = element_text(family = "Times New Roman")
    )
```

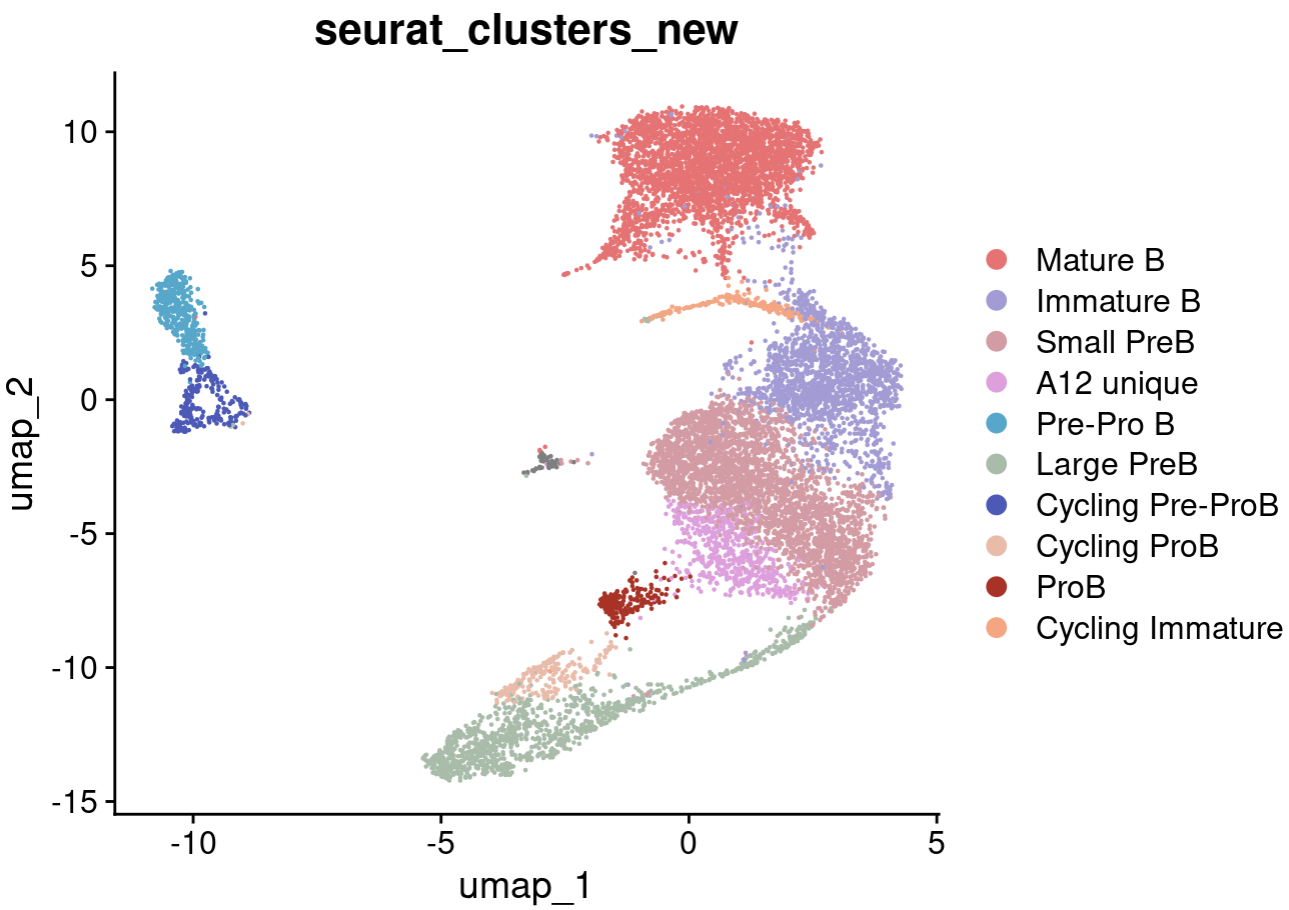



```
Seurat_Object_BM_selected_Bcells_idents <- RenameIdents(Seurat_Object_BM_selected_Bcells, "0"="Mature B", "1"="Immature B", "2"="Small PreB", "3"="Small PreB", "4"= "A12 unique", "5"="Pre-Pro B", "6"="Large PreB", "7"="Mature B", "8"="Large PreB", "9"="Cycling Pre-ProB", "10"="Large PreB", "11"= "Cycling ProB", "12" = "ProB", "13"="Cycling Immature", "14" = "Large PreB", "15" = "Mature B", "16" = "Mature B", "17" = "Highly Mitochondrial")
```

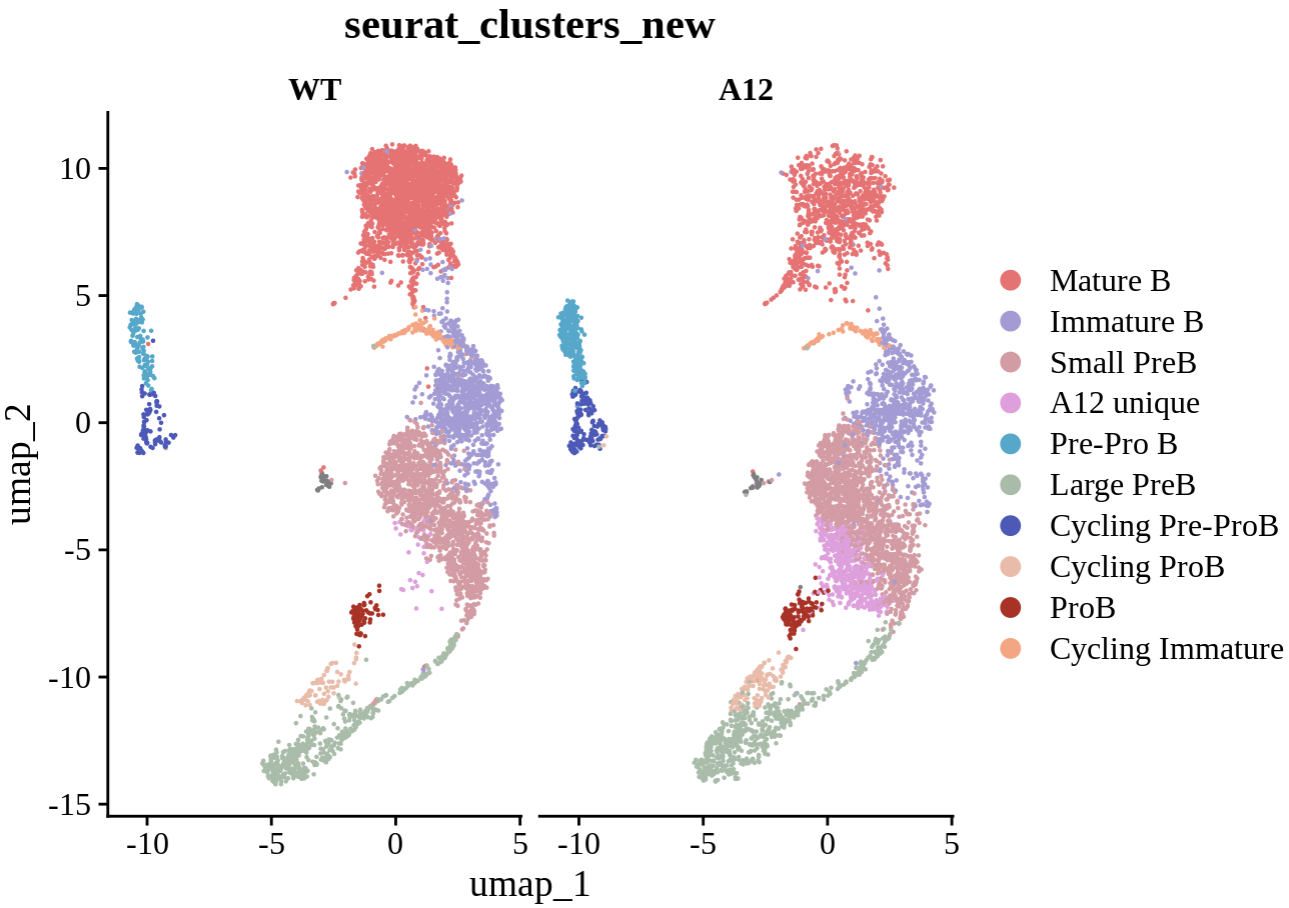
```
Seurat_Object_BM_selected_Bcells_idents$seurat_clusters_new <- Idents(Seurat_Object_BM_selected_Bcells_idents)
```

```
# Now create the DimPlot with specified colors
DimPlot(
    Seurat_Object_BM_selected_Bcells_idents,
    reduction = "umap",
    group.by = "seurat_clusters_new",
    cols=  c(
    "Mature B" = "#E57373",
    "Cycling Immature" = "#F4A582",
    "A12 unique" = "#DDA0DD",
    "Immature B" = "#A39BD3",
    "Small PreB" = "#D39BA3",
    "Large PreB" = "#A9BBA9",
    "ProB" = "#A93226",
    "Cycling ProB" = "#E9BBA9",
    "Pre-Pro B" = "#56A7C9",
    "Cycling Pre-ProB" = "#4C59B6"
```

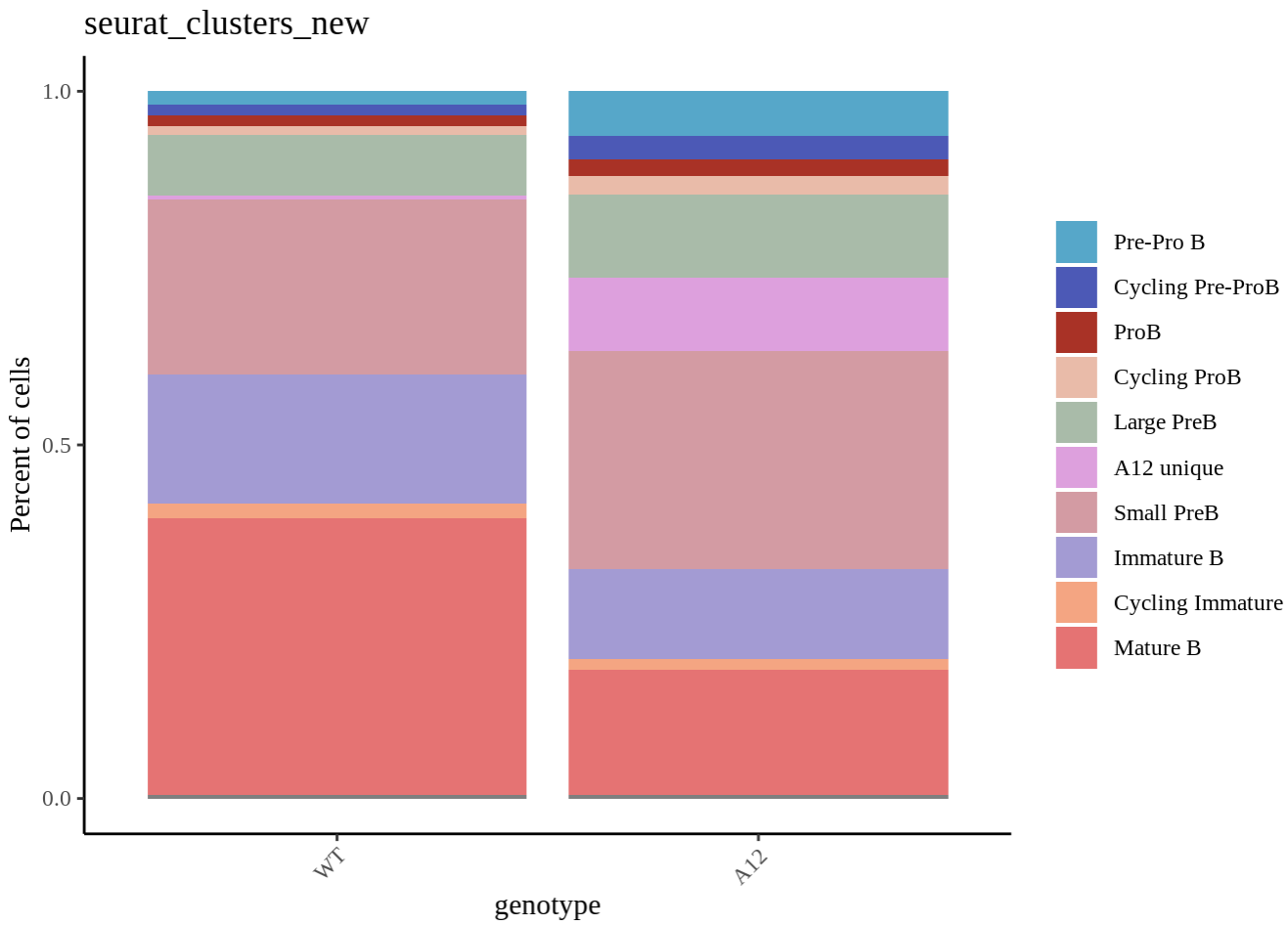
```
)
)
```




```
DimPlot(
    Seurat_Object_BM_selected_Bcells_idents,
    reduction = "umap",
    group.by = "seurat_clusters_new",
    cols=  c(
    "Mature B" = "#E57373",
    "Cycling Immature" = "#F4A582",
    "A12 unique" = "#DDA0DD",
    "Immature B" = "#A39BD3",
    "Small PreB" = "#D39BA3",
    "Large PreB" = "#A9BBA9",
    "ProB" = "#A93226",
    "Cycling ProB" = "#E9BBA9",
    "Pre-Pro B" = "#56A7C9",
    "Cycling Pre-ProB" = "#4C59B6"
), split.by = "genotype"
) +
    theme(
        text = element_text(family = "Times New Roman")
    )
```




```
dittoBarPlot(
    object = Seurat_Object_BM_selected_Bcells_idents,
    scale = "percent",
    var = "seurat_clusters_new",
    color.panel =  c(
    "Mature B" = "#E57373",
    "Cycling Immature" = "#F4A582",
    "A12 unique" = "#DDA0DD",
    "Immature B" = "#A39BD3",
    "Small PreB" = "#D39BA3",
    "Large PreB" = "#A9BBA9",
    "ProB" = "#A93226",
    "Cycling ProB" = "#E9BBA9",
    "Pre-Pro B" = "#56A7C9",
    "Cycling Pre-ProB" = "#4C59B6"
), group.by = "genotype", x.reorder = c(2,1), var.labels.reorder = c(9,3,10,4,7,1,11,6,2,8,5))
+
    theme(
        text = element_text(family = "Times New Roman")
    )
```




```
# Extract data from Seurat object
data <- Seurat_Object_BM_selected_Bcells_idents@meta.data
# Calculate percentages for each combination of genotype and seurat_clusters
data_summary <- data %>%
  group_by(genotype, seurat_clusters) %>%
  summarize(count = n()) %>%
  mutate(percentage = count / sum(count) * 100)
```

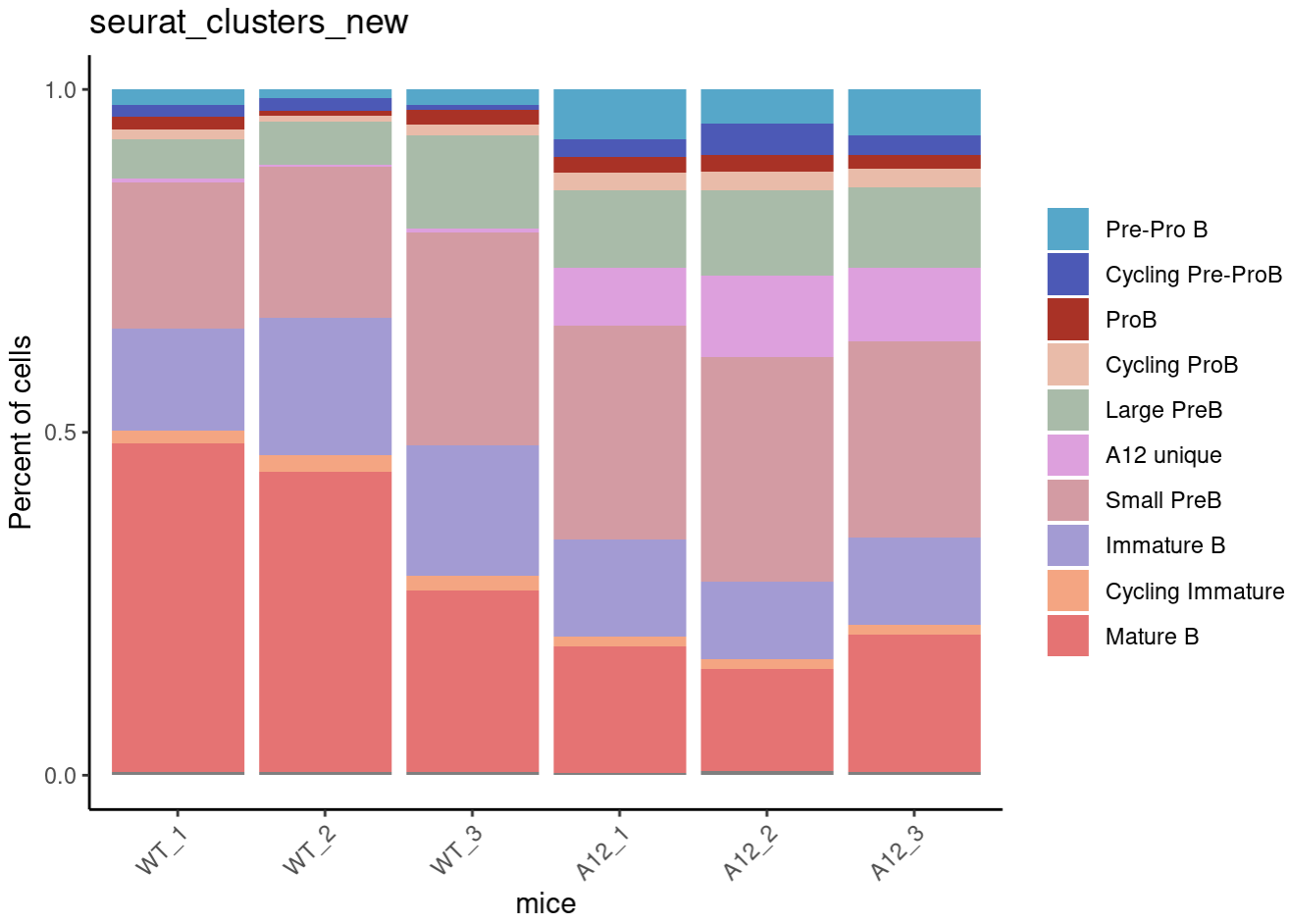
```
## `summarise()` has grouped output by 'genotype'. You can override using the
## `.groups` argument.
```

```
data_summary
```

```
## # A tibble: 36 × 4
## # Groups:   genotype [2]
##   genotype seurat_clusters count percentage
##   <fct>    <fct>           <int>      <dbl>
## 1 WT       0                2003     33.3
## 2 WT       1                1093     18.2
## 3 WT       2                 764     12.7
## 4 WT       3                 730     12.1
## 5 WT       4                  32      0.531
## 6 WT       5                 115      1.91
## 7 WT       6                 178      2.96
```

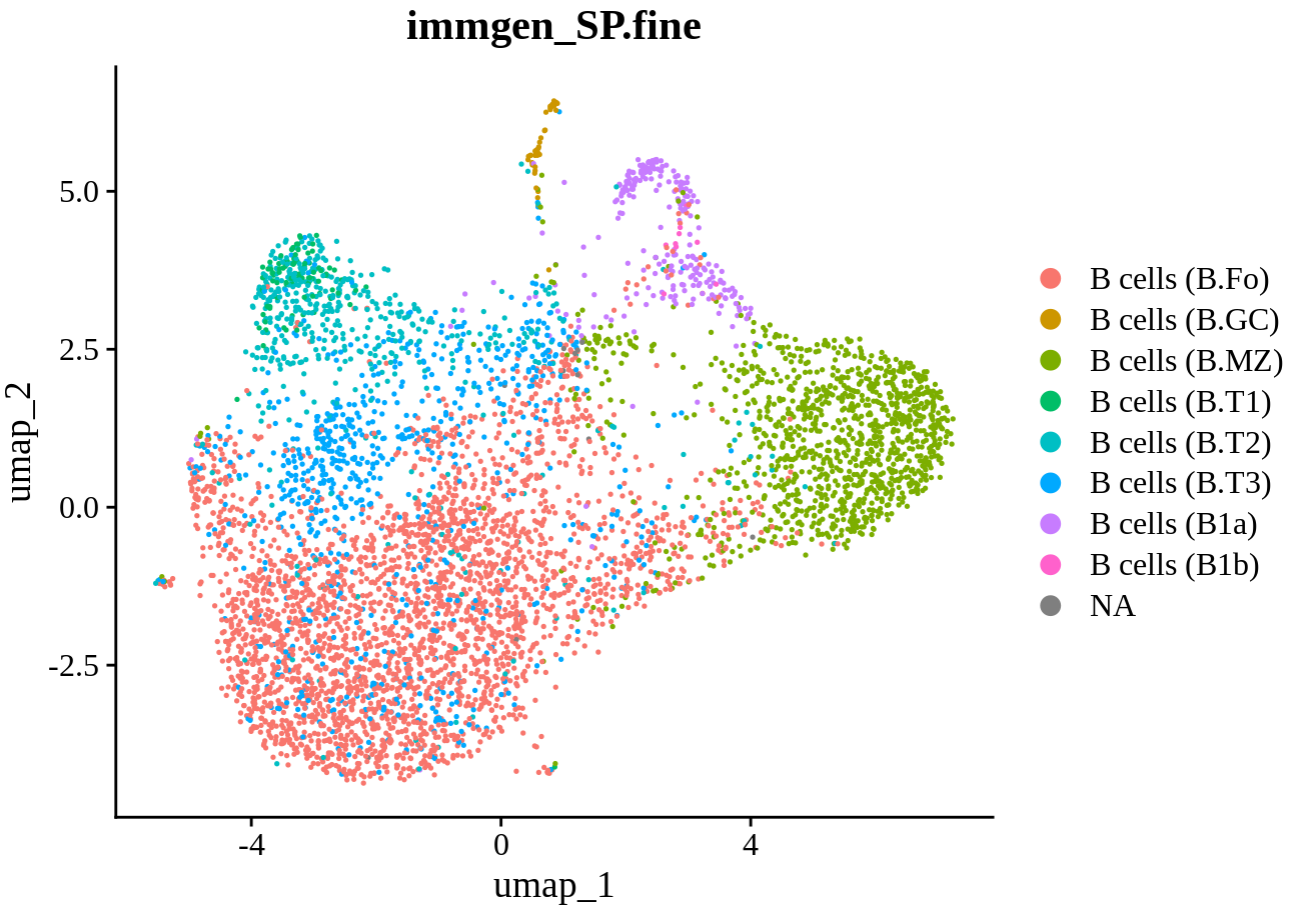
```
## 8 WT      7           179     2.97
## 9 WT      8           153     2.54
## 10 WT     9            87     1.44
## # ℹ 26 more rows
```

```
# Just a quick example with the built in Seurat data:
dittoBarPlot(
    object = Seurat_Object_BM_selected_Bcells_idents,
    scale = "percent",
    var = "seurat_clusters_new",
    color.panel = c(
    "Mature B" = "#E57373",
    "Cycling Immature" = "#F4A582",
    "A12 unique" = "#DDA0DD",
    "Immature B" = "#A39BD3",
    "Small PreB" = "#D39BA3",
    "Large PreB" = "#A9BBA9",
    "ProB" = "#A93226",
    "Cycling ProB" = "#E9BBA9",
    "Pre-Pro B" = "#56A7C9",
    "Cycling Pre-ProB" = "#4C59B6"
),
    group.by = "mice", x.reorder = c(4,5,6,1,2,3), var.labels.reorder = c(9,3,10,4,7,1,11,6,2,
8,5))
```

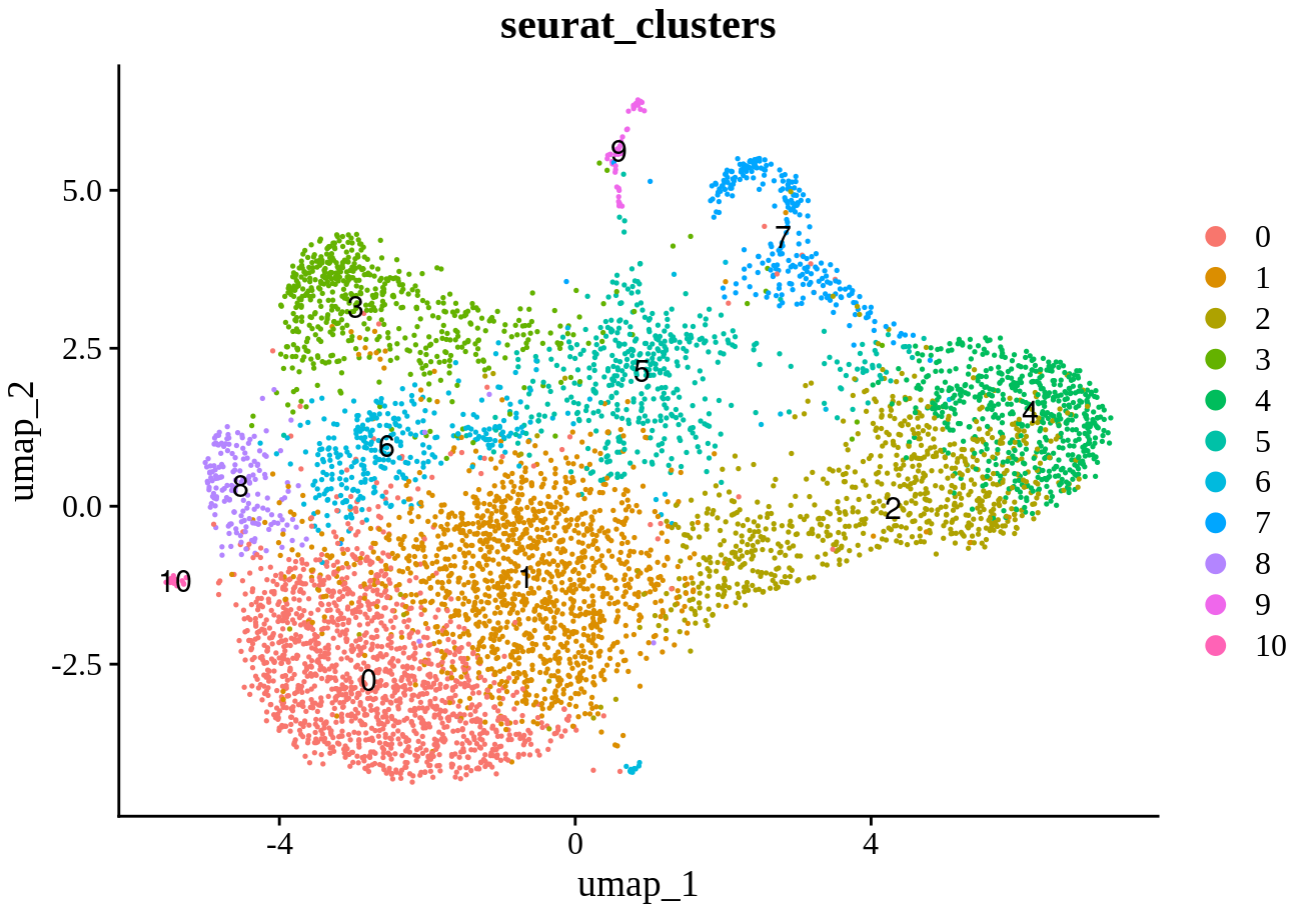


Rename clusters and analysis for SP

```
DimPlot(Seurat_Object_SP_selected_Bcells, reduction = "umap", group.by = "immgen_SP.fine") +
    theme(
        text = element_text(family = "Times New Roman")
    )
```



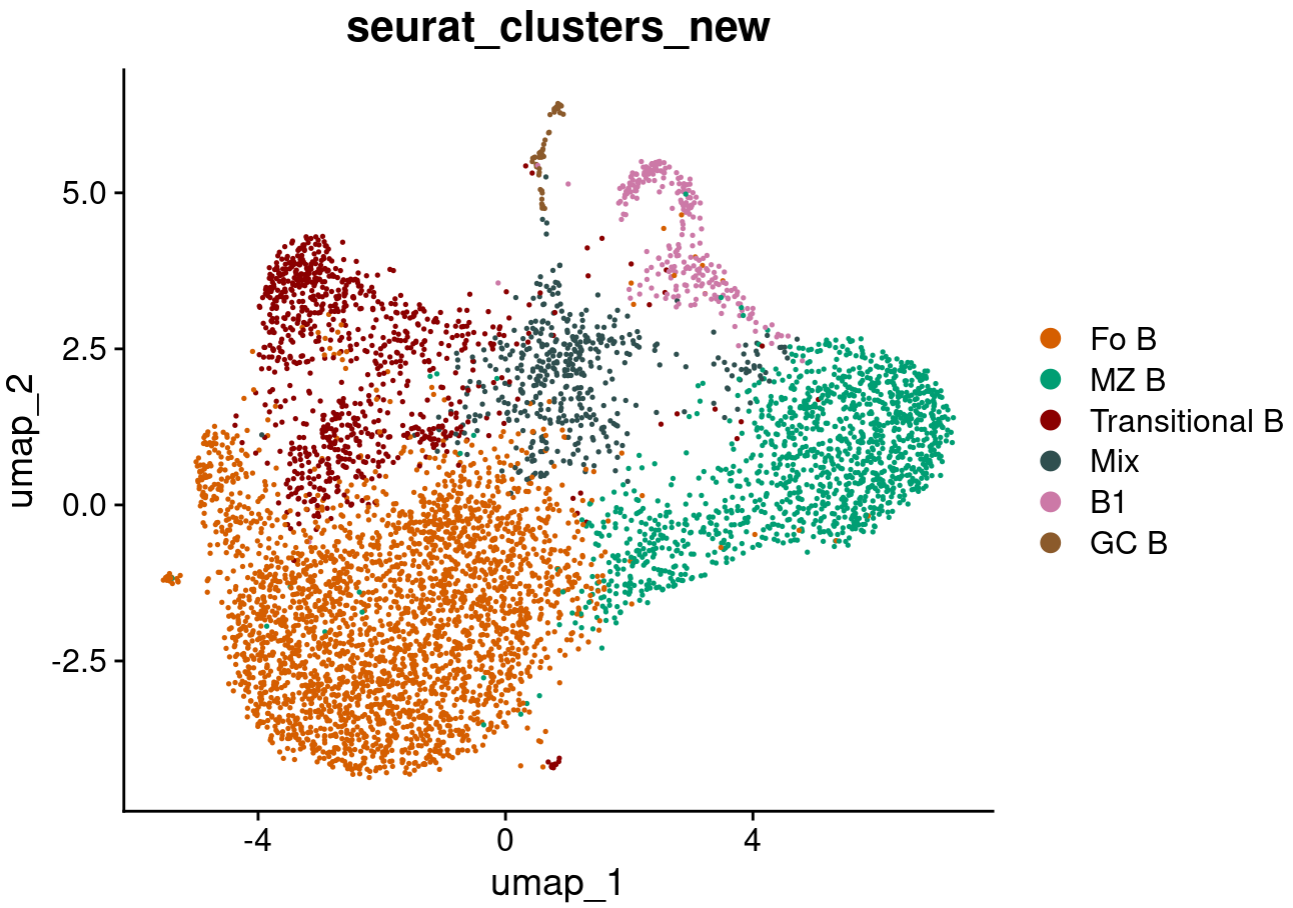

```
DimPlot(Seurat_Object_SP_selected_Bcells, reduction = "umap", group.by = "seurat_clusters", la
bel = TRUE) +
    theme(
        text = element_text(family = "Times New Roman")
    )
```



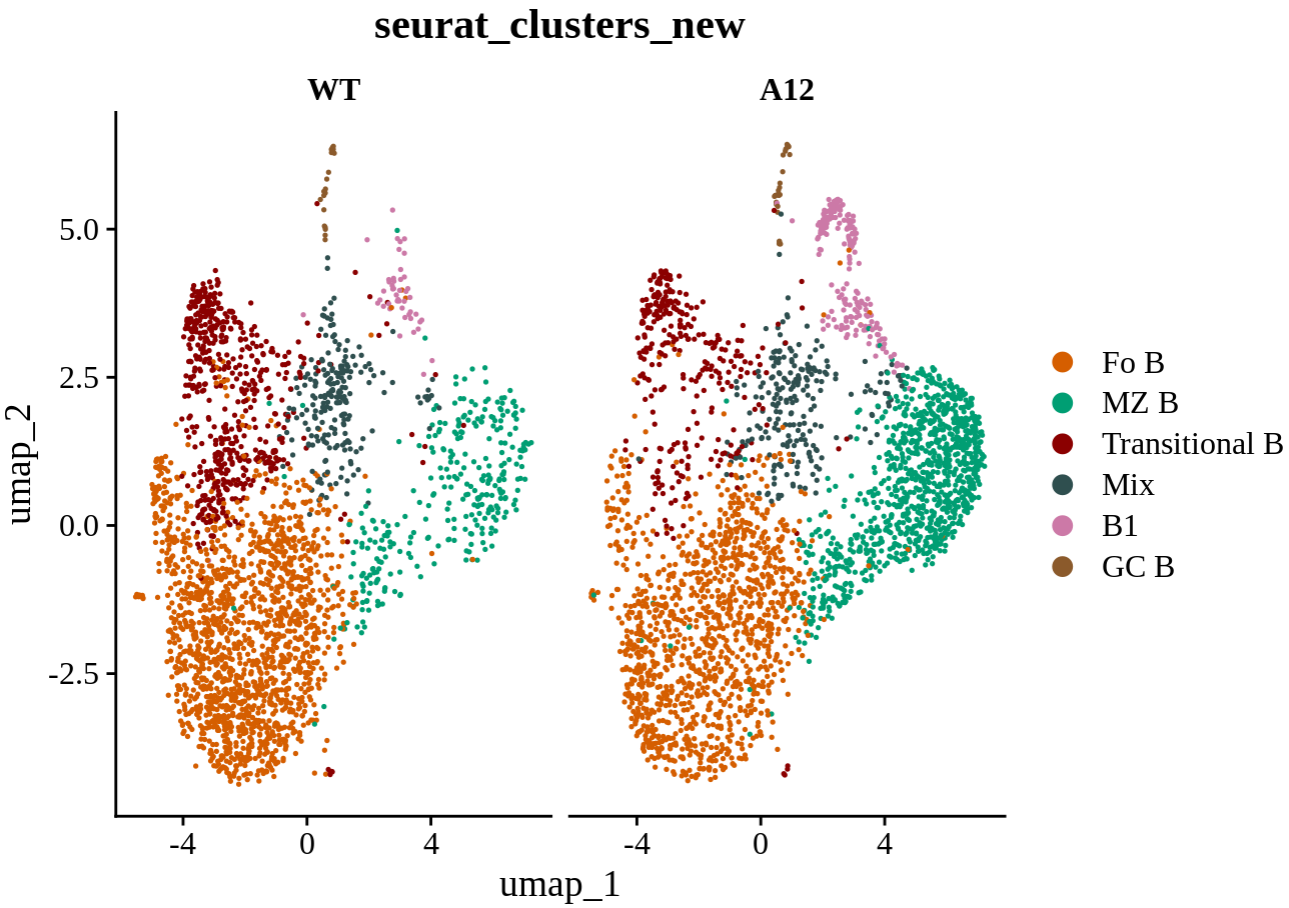

```
Seurat_Object_SP_selected_Bcells_idents <- RenameIdents(Seurat_Object_SP_selected_Bcells, "0"="Fo B", "1"="Fo B", "2" = "MZ B", "3"="Transitional B", "4"= "MZ B", "5"="Mix", "6"="Transitional B", "7" = "B1", "8" = "Fo B", "9" = "GC B", "10" = "Fo B")
```

```
Seurat_Object_SP_selected_Bcells_idents$seurat_clusters_new <- Idents(Seurat_Object_SP_selected_Bcells_idents)
```

```
#plot the cells from Blimp using the transferred clustering
DimPlot(Seurat_Object_SP_selected_Bcells_idents, reduction = "umap", cols= c("Fo B" = "#D55E00", "Transitional B" = "darkred", "MZ B" = "#009E73", "B1" = "#CC79A7", "Mix" = "darkslategray", "GC B" = "tan4"), group.by = "seurat_clusters_new")
```




```
DimPlot(Seurat_Object_SP_selected_Bcells_idents, reduction = "umap", cols= c("Fo B" = "#D55E00
", "Transitional B" = "darkred", "MZ B" = "#009E73", "B1" = "#CC79A7", "Mix" = "darkslategray"
, "GC B" = "tan4"), group.by = "seurat_clusters_new", split.by = "genotype") +
    theme(
        text = element_text(family = "Times New Roman")  # Cambia la fuente a Times New Roman
    )
```




```
data_summary <- Seurat_Object_SP_selected_Bcells_idents@meta.data %>%
  group_by(genotype, seurat_clusters_new) %>%
  summarize(count = n()) %>%
  mutate(percentage = count / sum(count) * 100)
```

```
## `summarise()` has grouped output by 'genotype'. You can override using the
## `.groups` argument.
```

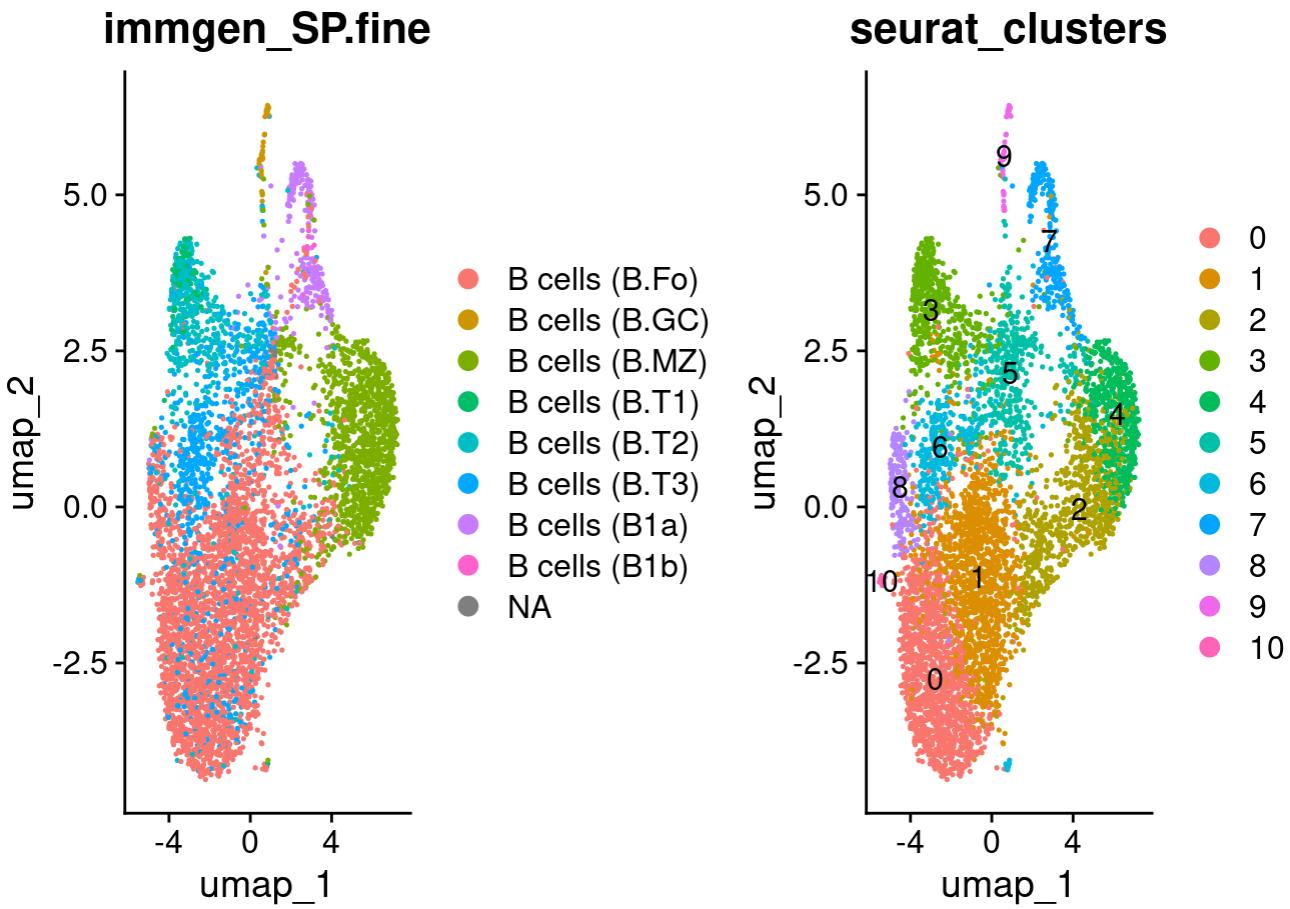
```
data_summary[data_summary$genotype == "A12",]
```

```
## # A tibble: 6 × 4
## # Groups:   genotype [1]
##   genotype seurat_clusters_new count percentage
##   <fct>    <fct>                <int>      <dbl>
## 1 A12      Fo B                  1264     41.7
## 2 A12      MZ B                  1005     33.2
## 3 A12      Transitional B         305     10.1
## 4 A12      Mix                    228      7.53
## 5 A12      B1                     201      6.64
## 6 A12      GC B                    26      0.858
```

```
data_summary[data_summary$genotype == "WT",]
```

```
## # A tibble: 6 × 4
## # Groups:   genotype [1]
##   genotype seurat_clusters_new count percentage
##   <fct>    <fct>                <int>      <dbl>
## 1 WT       Fo B                  1627     58.6
## 2 WT       MZ B                   312     11.2
## 3 WT       Transitional B         555     20.0
## 4 WT       Mix                    216      7.78
## 5 WT       B1                      47      1.69
## 6 WT       GC B                    20      0.720
```

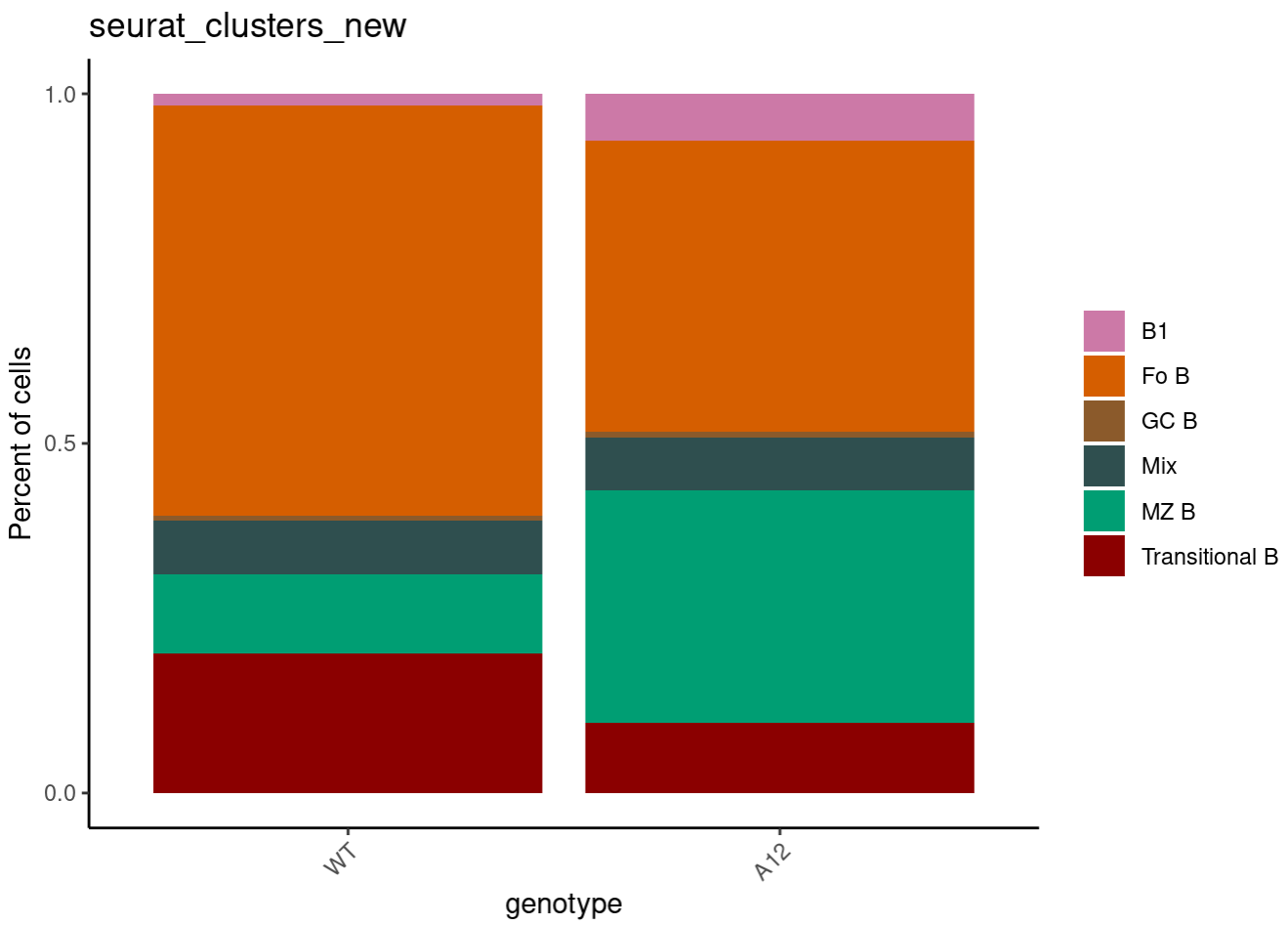
```
DimPlot(Seurat_Object_SP_selected_Bcells_idents, reduction = "umap", group.by = "immgen_SP.fine") + DimPlot(Seurat_Object_SP_selected_Bcells_idents, reduction = "umap", group.by = "seurat_clusters", label = T)
```



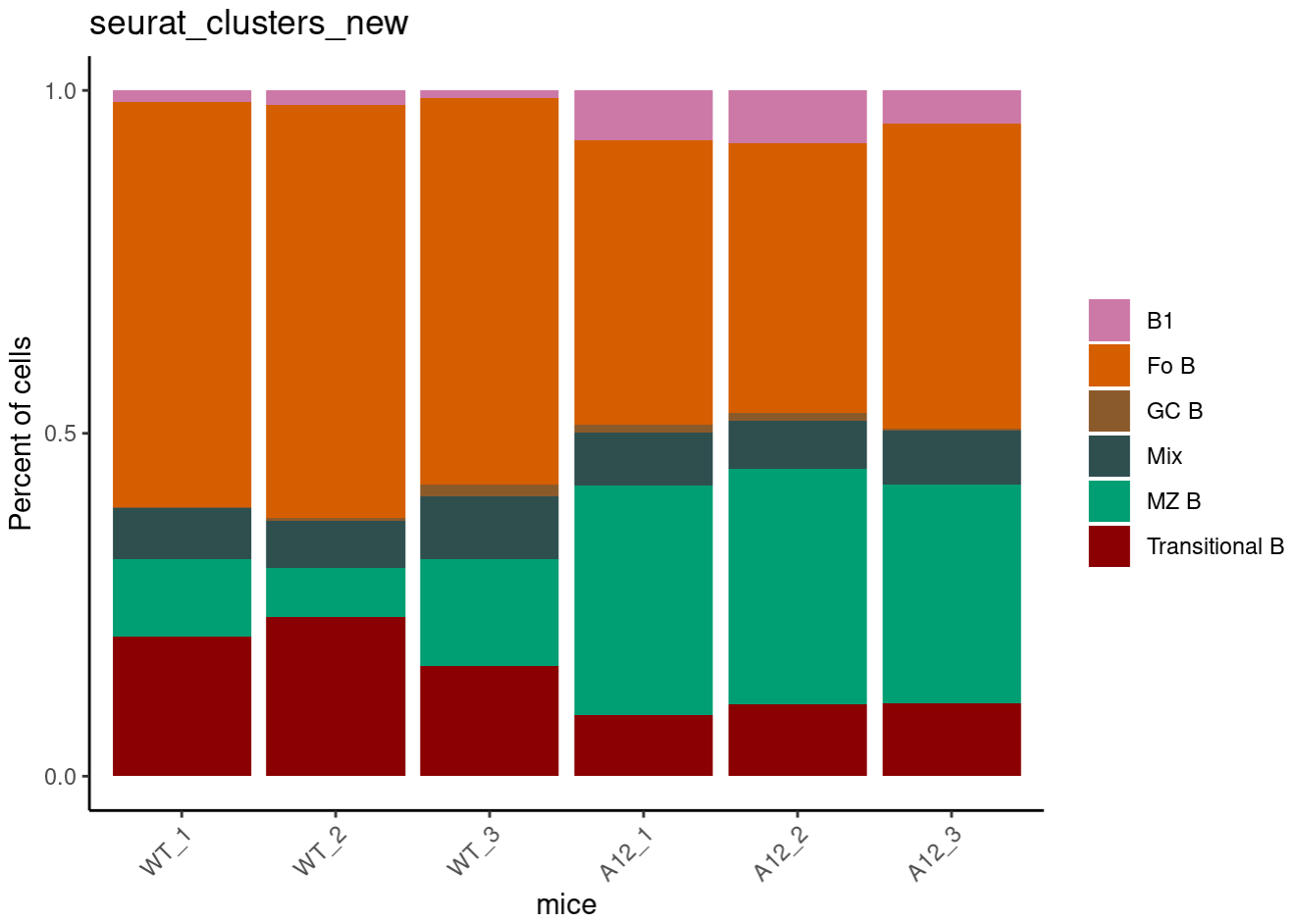

```
Seurat_Object_SP_selected_Bcells_idents$seurat_clusters_new<- Idents(Seurat_Object_SP_selected_Bcells_idents)
```

```
# Just a quick example with the built in Seurat data:
dittoBarPlot(
    object = Seurat_Object_SP_selected_Bcells_idents,
    scale = "percent",
    var = "seurat_clusters_new",
    color.panel =  c("Fo B" = "#D55E00", "Transitional B" = "darkred", "MZ B" = "#009E73", "B1
```

```
" = "#CC79A7", "Mix" = "darkslategray", "GC B" = "tan4"),
    group.by = "genotype", x.reorder = c(2,1))
```



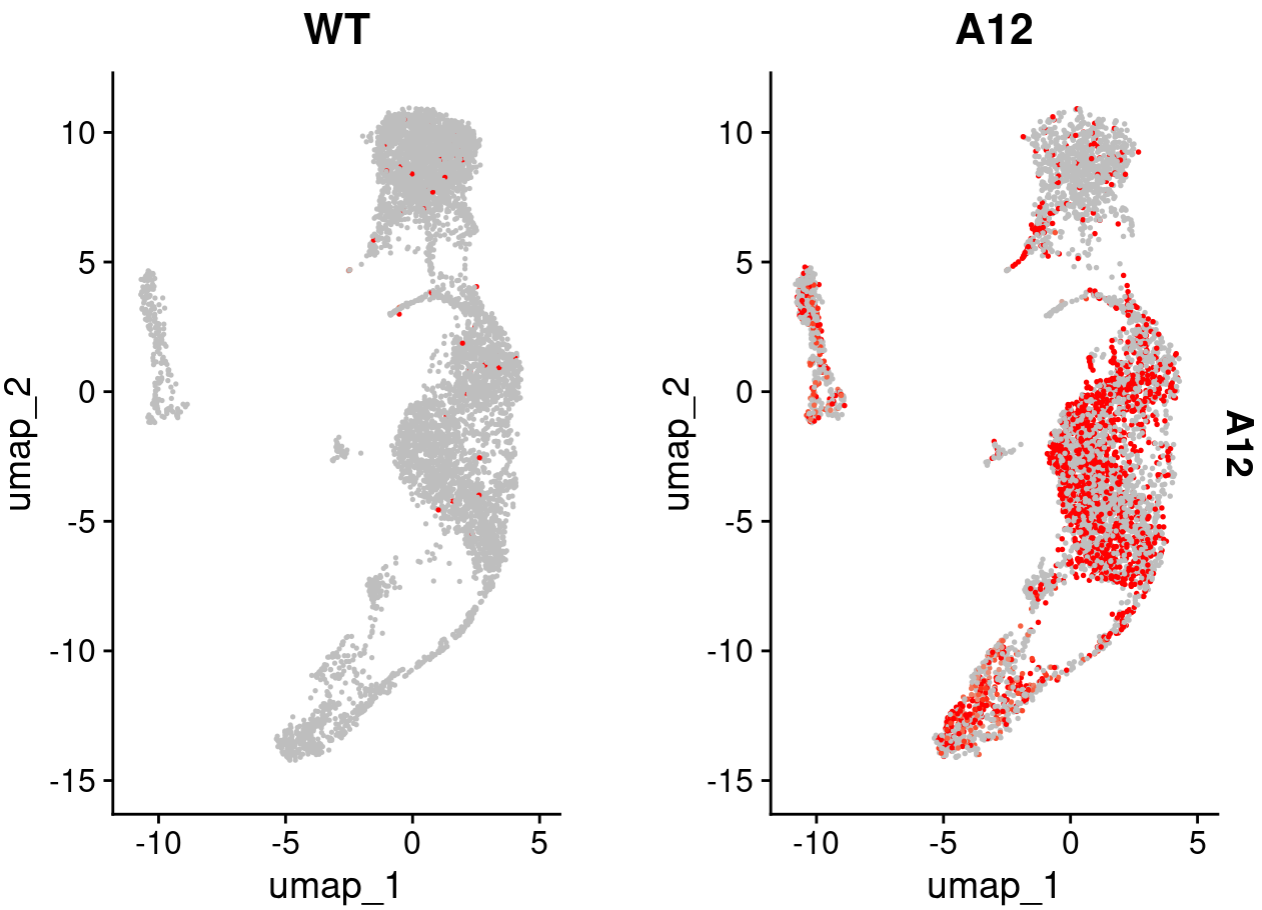

```
# Just a quick example with the built in Seurat data:
dittoBarPlot(
    object = Seurat_Object_SP_selected_Bcells_idents,
    scale = "percent",
    var = "seurat_clusters_new",
    color.panel = c("Fo B" = "#D55E00", "Transitional B" = "darkred", "MZ B" = "#009E73", "B1"
 = "#CC79A7", "Mix" = "darkslategray", "GC B" = "tan4"),
    group.by = "mice", x.reorder = c(4,5,6,1,2,3))
```



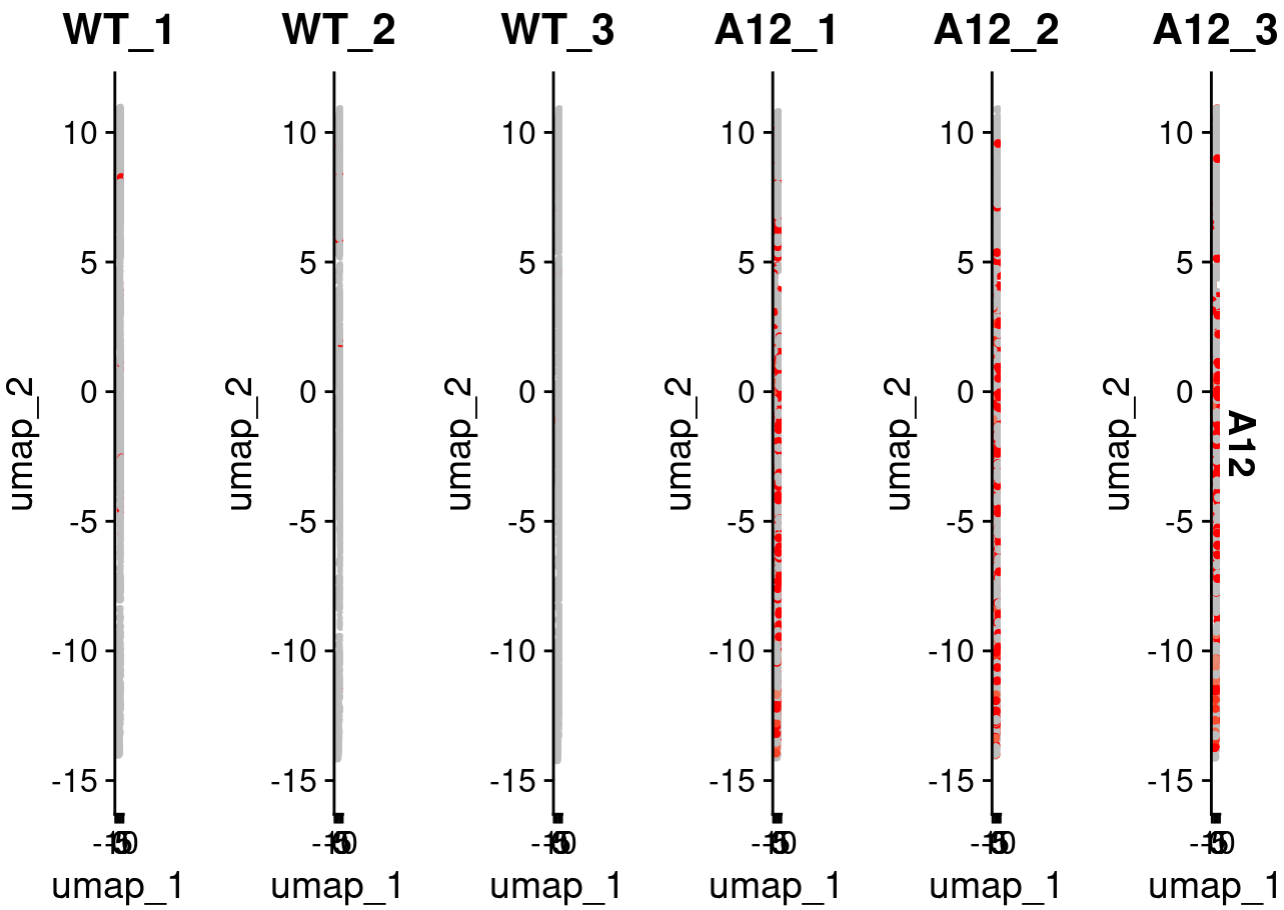
Gene Expression

A12 expression

```
FeaturePlot(Seurat_Object_BM_selected_Bcells_idents, features = "A12", max.cutoff = 1, split.by = "genotype",
    cols = c("grey", "red"))
```

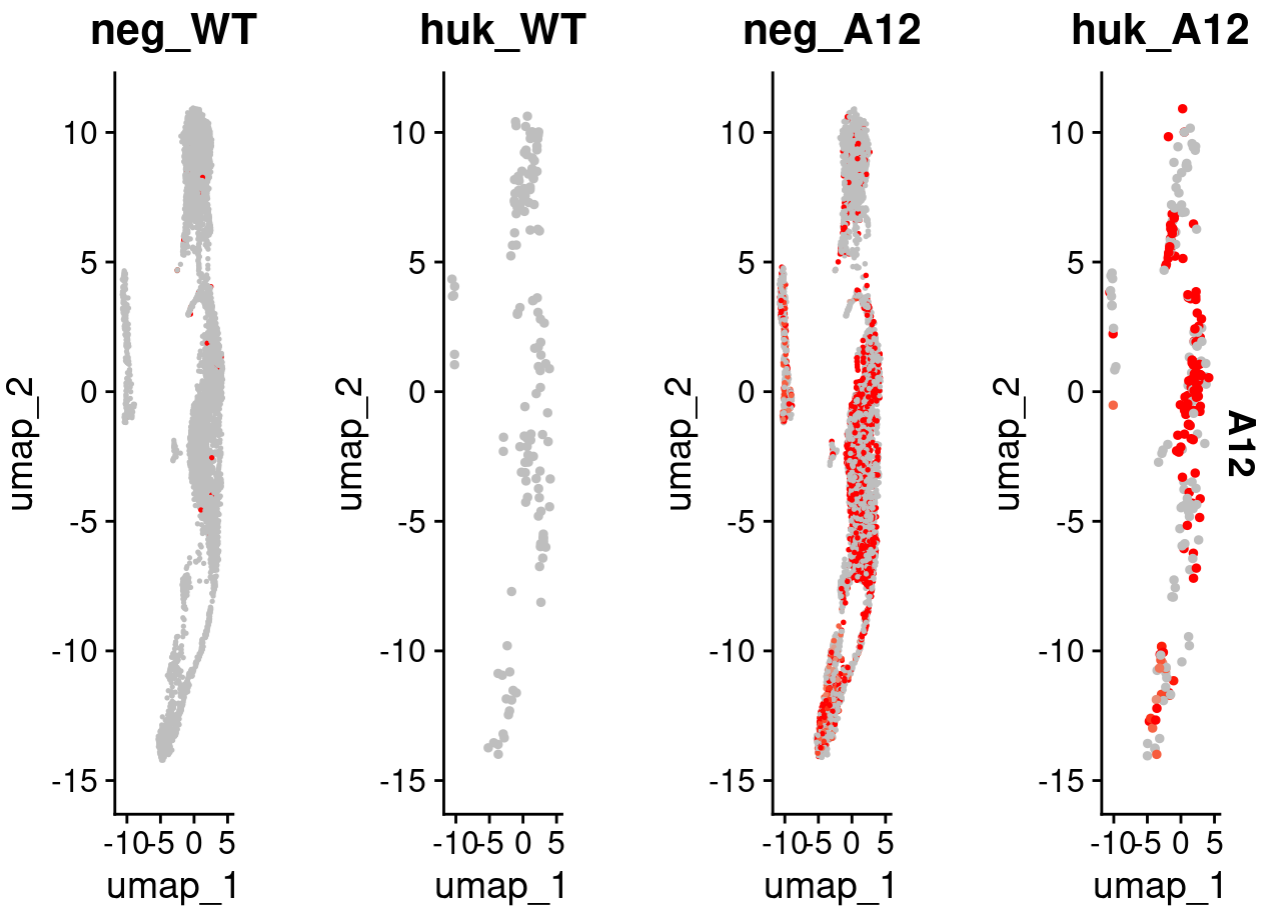



```
FeaturePlot(Seurat_Object_BM_selected_Bcells_idents, features = "A12", max.cutoff = 1, split.by = "mice",
    cols = c("grey", "red"))
```

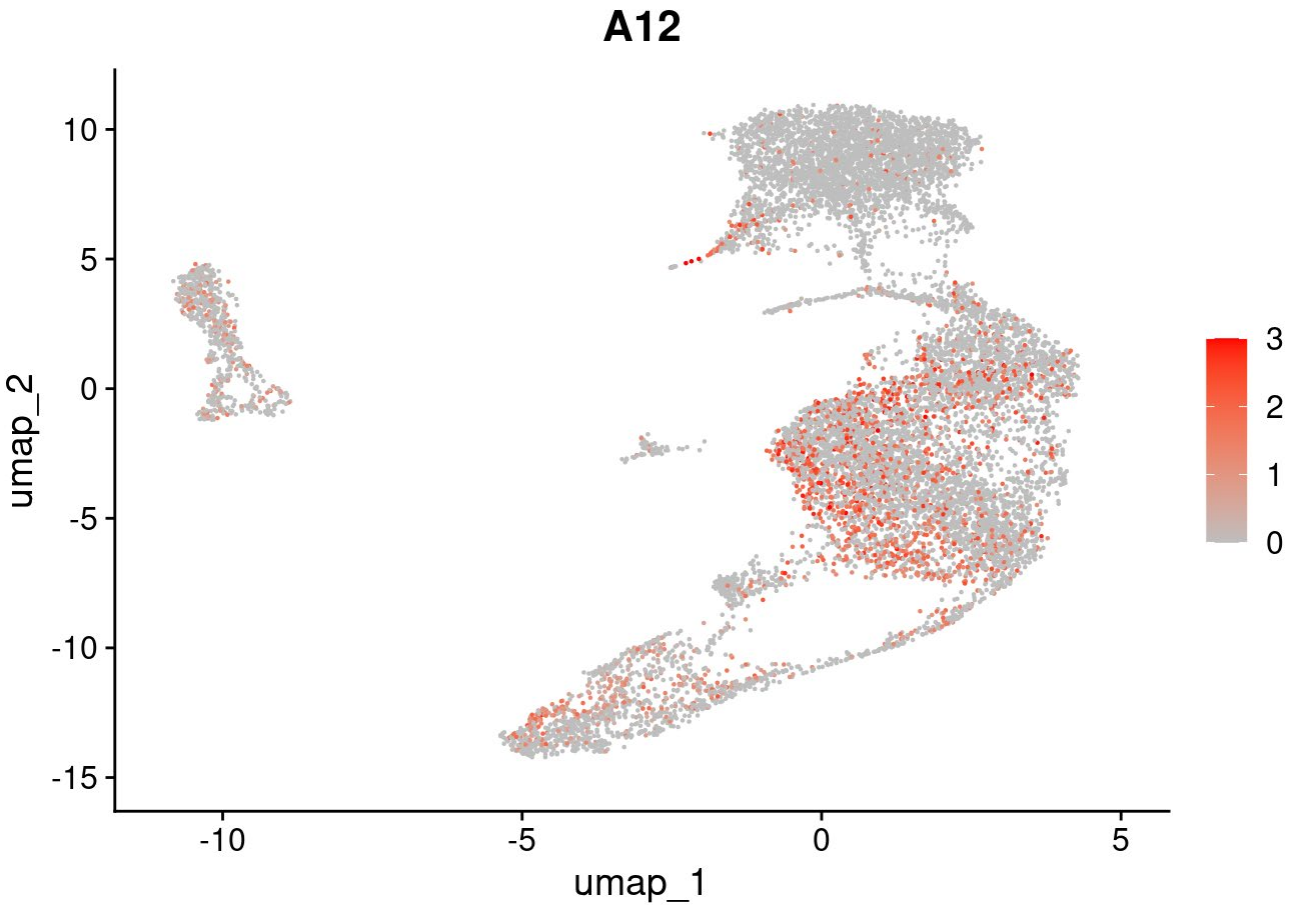



```
FeaturePlot(Seurat_Object_BM_selected_Bcells_idents, features = "A12", max.cutoff = 1, split.by = "huIgk",
    cols = c("grey", "red"))
```

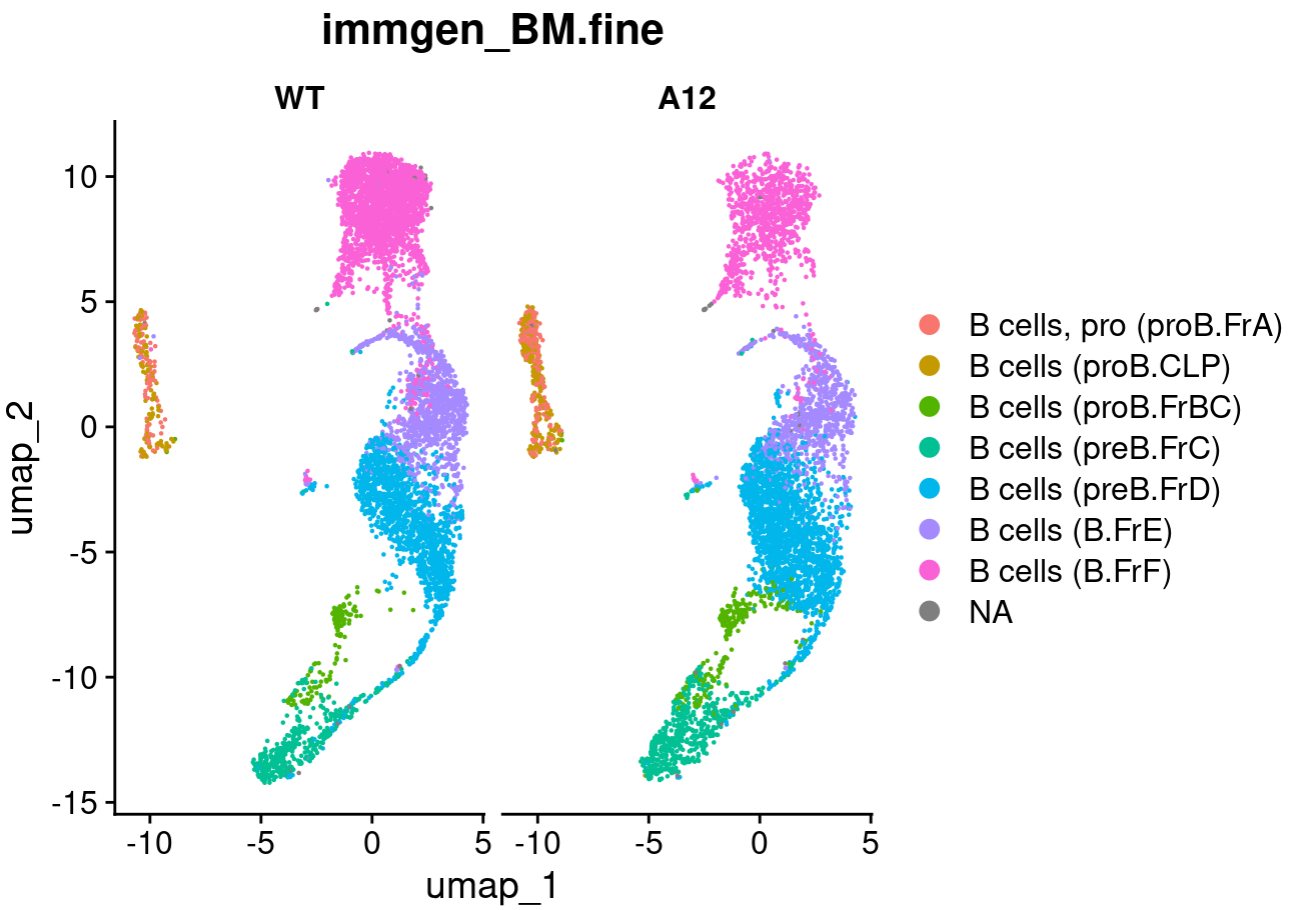
```
## Warning: All cells have the same value (0) of "A12"
```



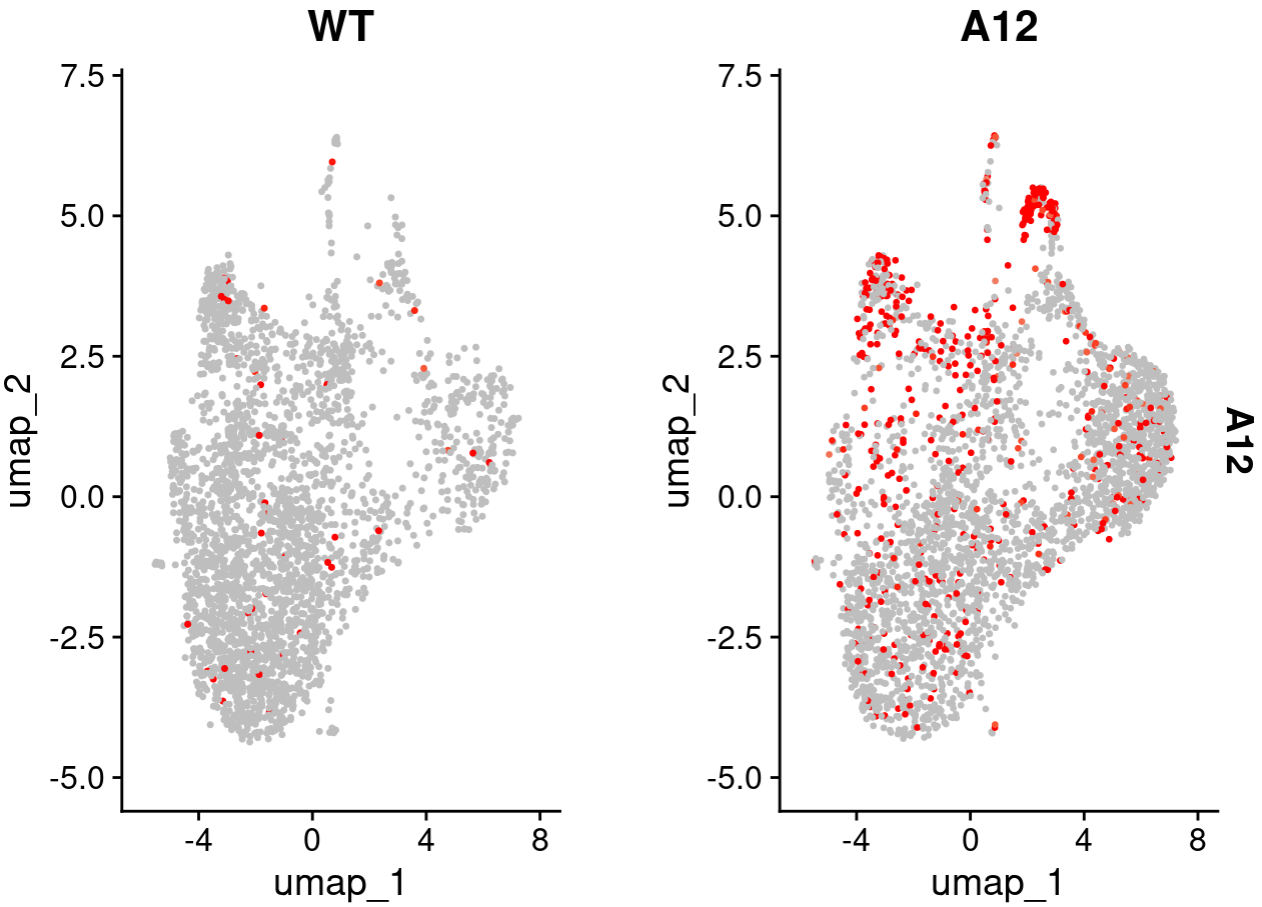

```
FeaturePlot(Seurat_Object_BM_selected_Bcells_idents, features = "A12", max.cutoff = 3,
    cols = c("grey", "red"))
```



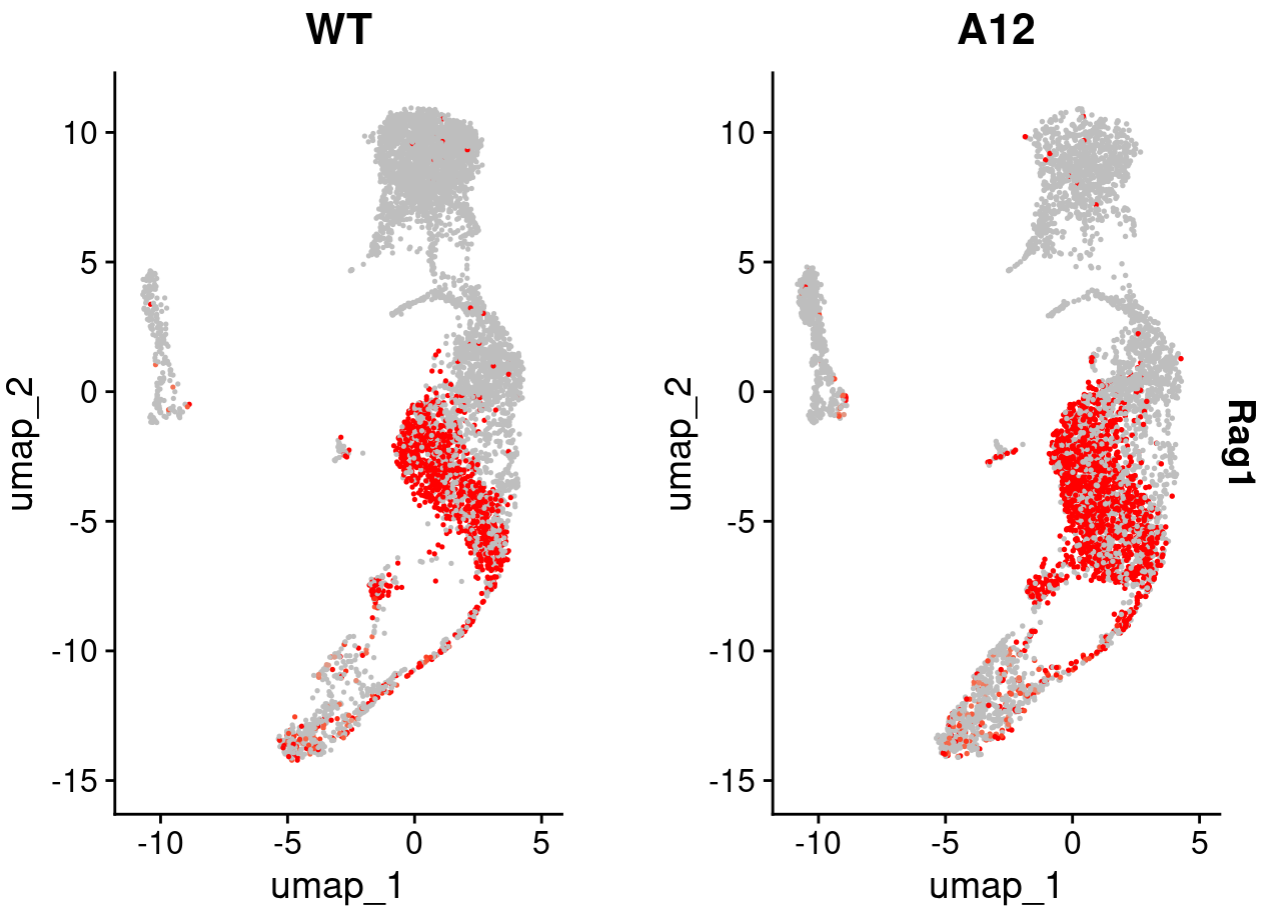

```
DimPlot(Seurat_Object_BM_selected_Bcells_idents, group.by = "immgen_BM.fine", split.by = "geno
type")
```



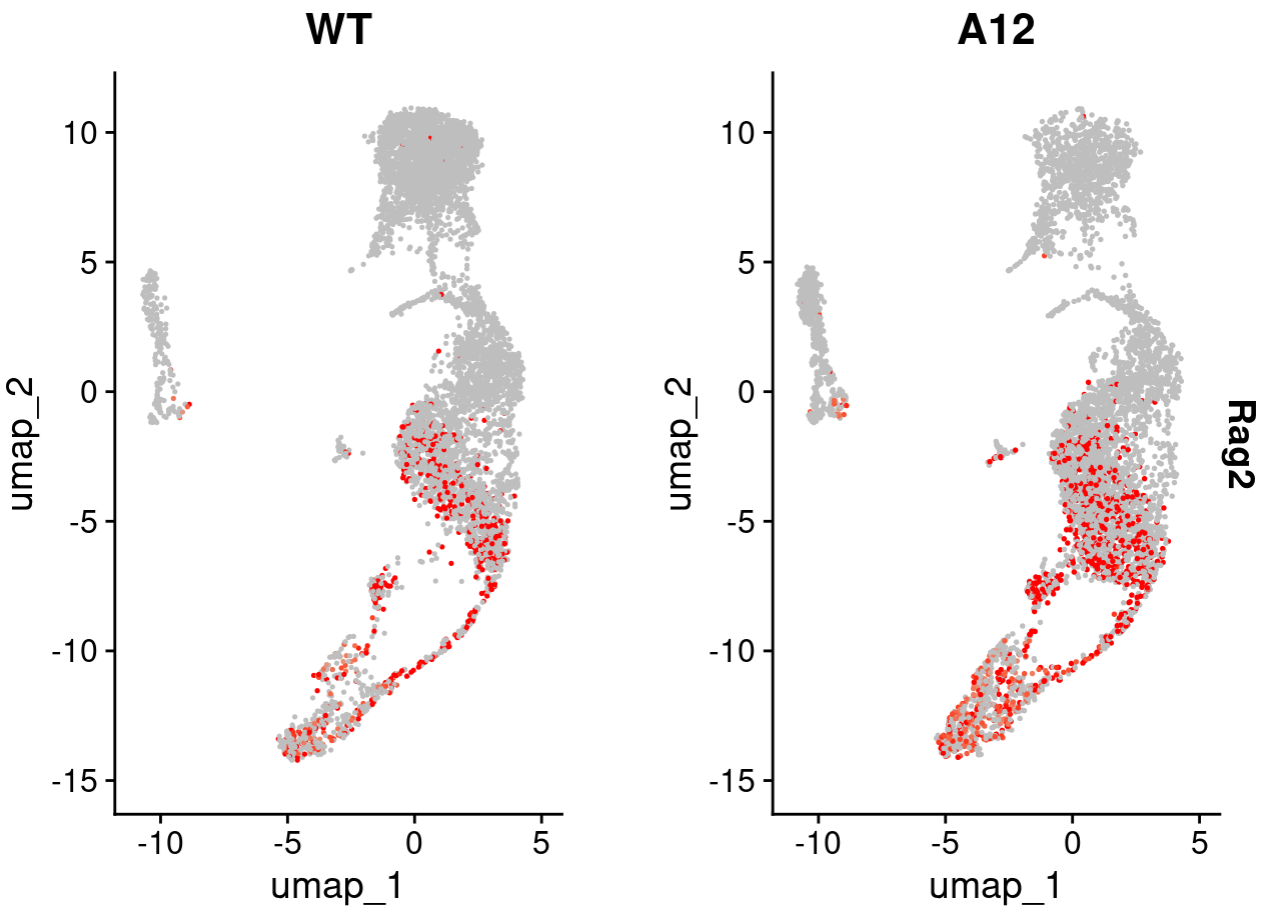

```
FeaturePlot(Seurat_Object_SP_selected_Bcells_idents, features = "A12", max.cutoff = 1, split.by = "genotype",
    cols = c("grey", "red"))
```



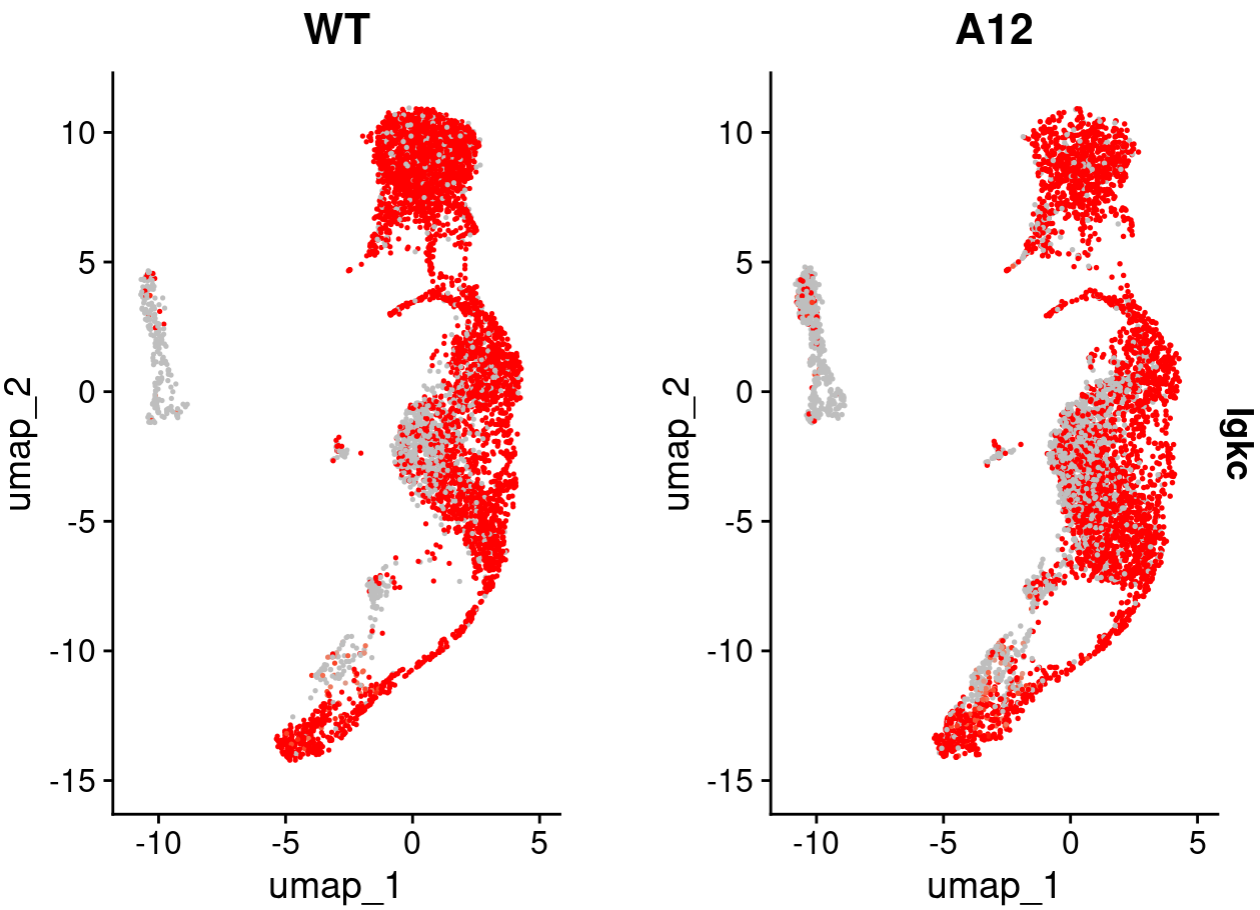

```
FeaturePlot(Seurat_Object_BM_selected_Bcells_idents, features = "Rag1", max.cutoff = 1, split.
by = "genotype",
    cols = c("grey", "red"))
```




```
FeaturePlot(Seurat_Object_BM_selected_Bcells_idents, features = "Rag2", max.cutoff = 1, split.
by = "genotype",
    cols = c("grey", "red"))
```

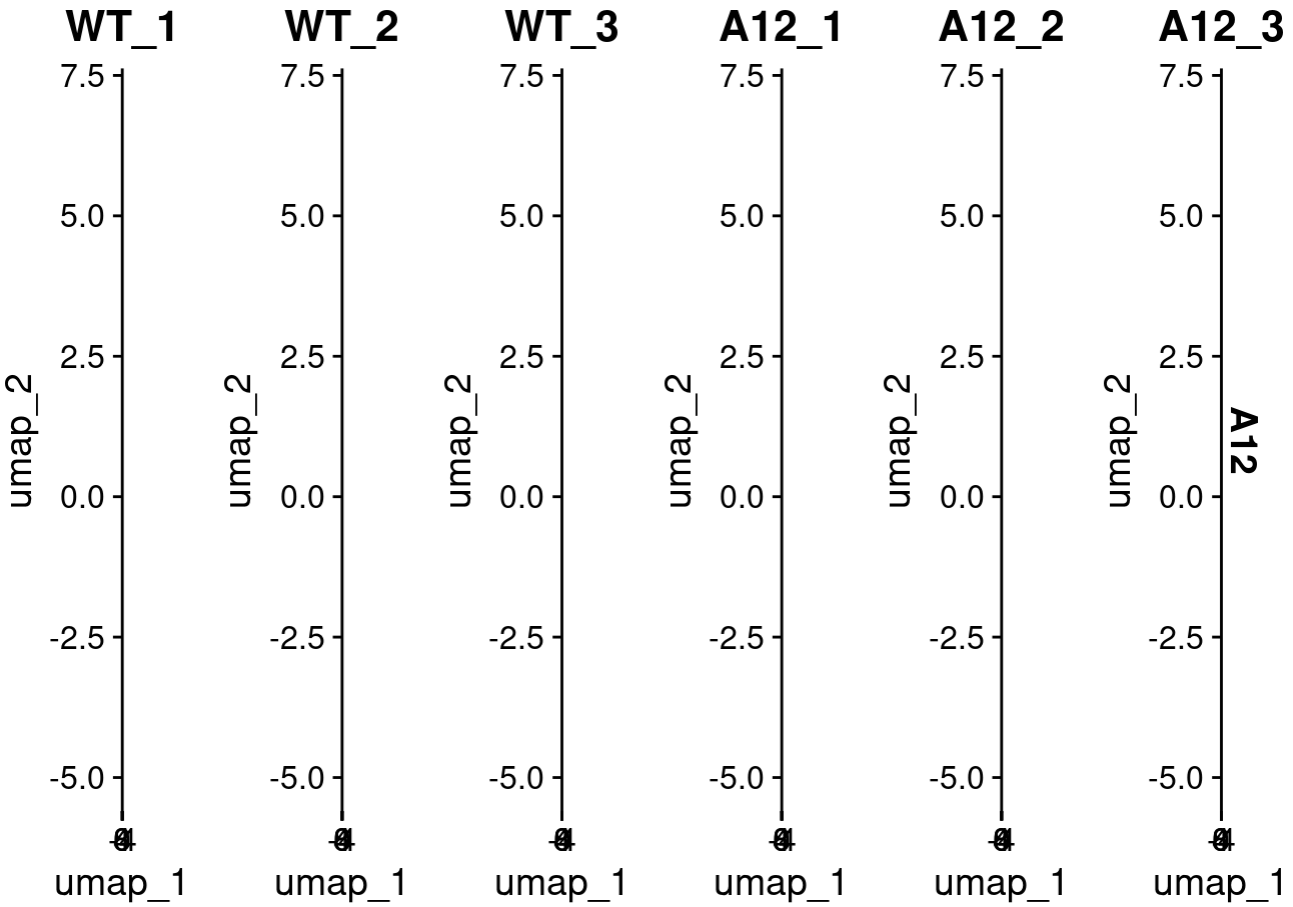



```
FeaturePlot(Seurat_Object_BM_selected_Bcells_idents, features = "Igkc", max.cutoff = 1, split.
by = "genotype",
    cols = c("grey", "red"))
```



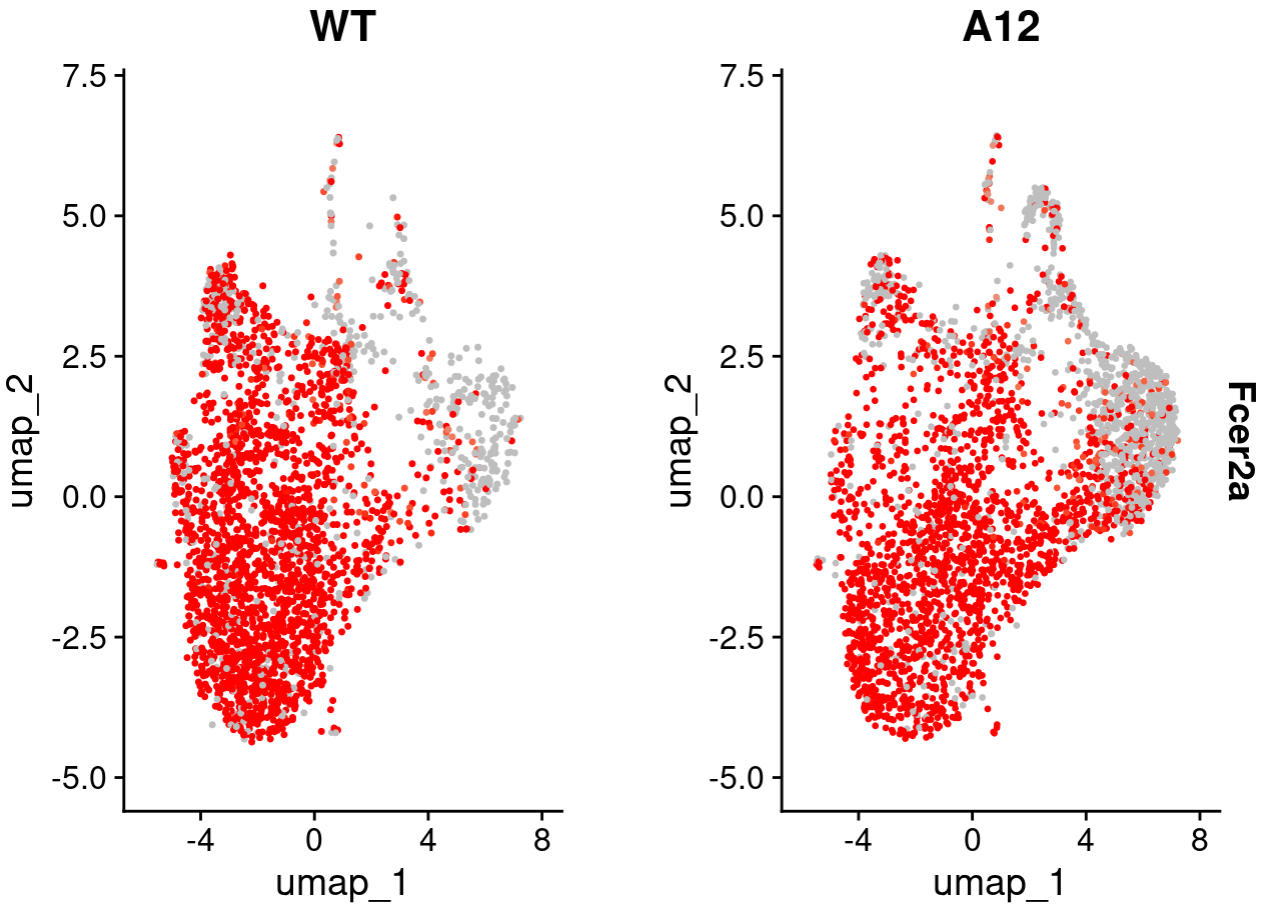

```
FeaturePlot(Seurat_Object_SP_selected_Bcells_idents, features = "A12", max.cutoff = 1, split.by = "mice",
    cols = c("grey", "red"))
```

Expresión

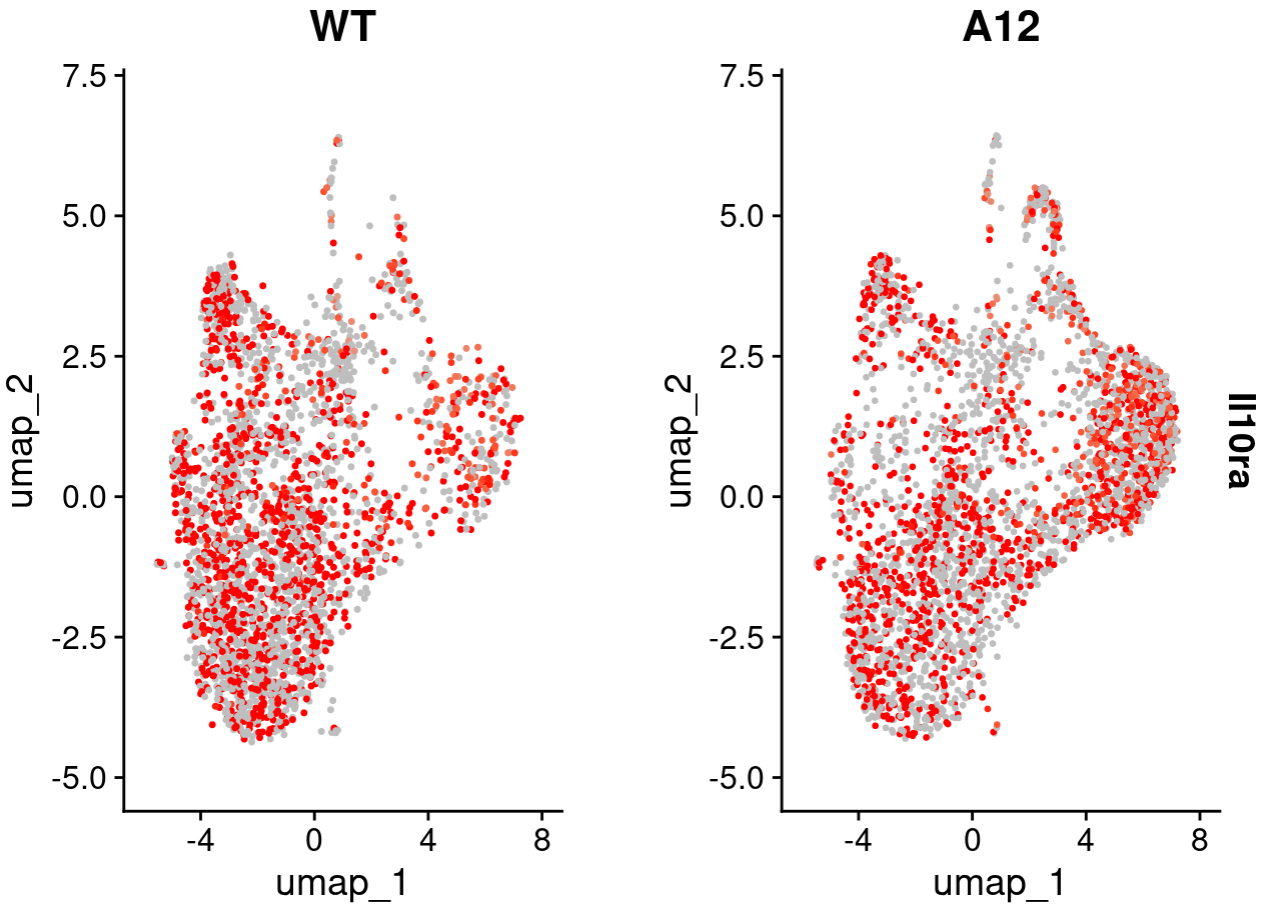


de CD21 y CD23

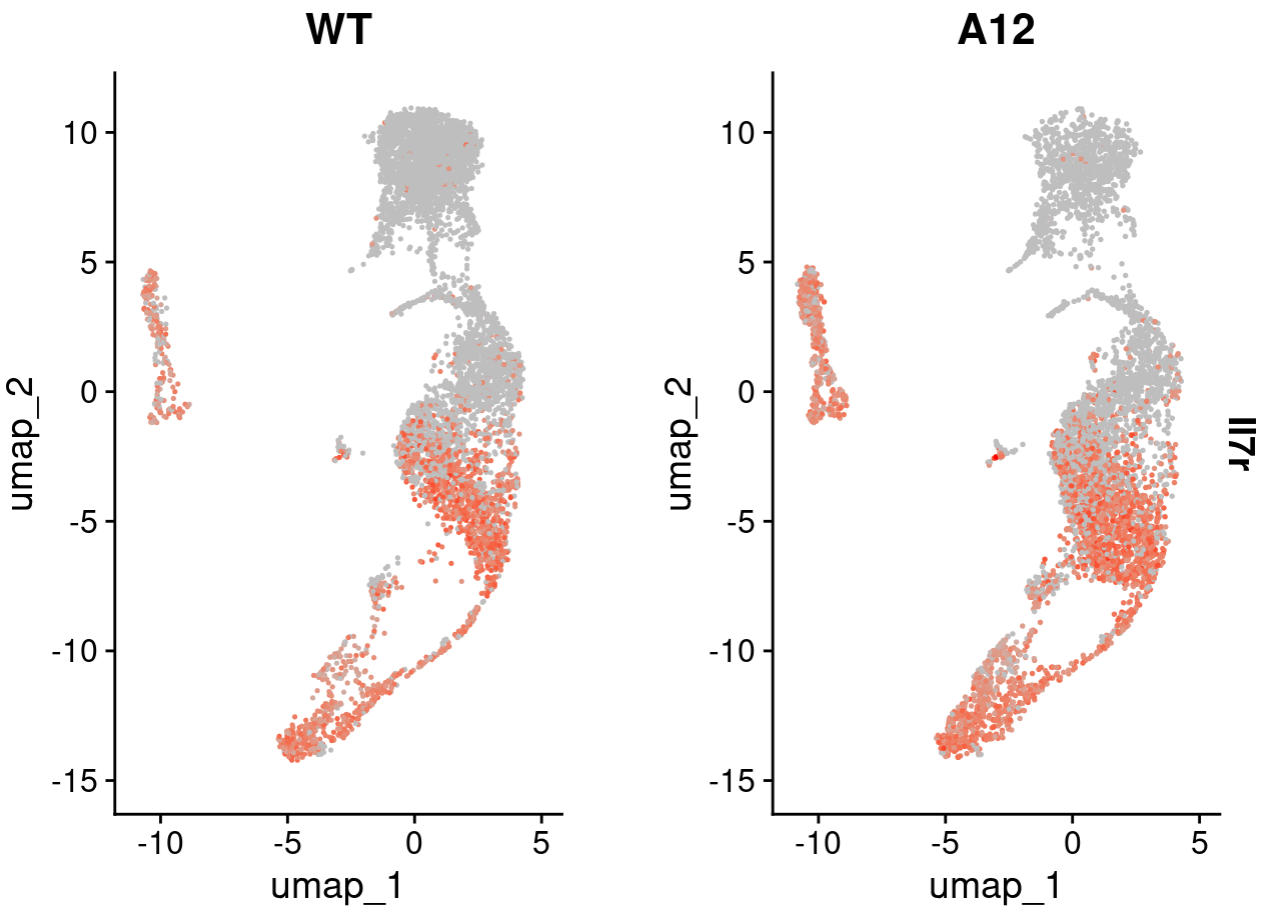
```
FeaturePlot(Seurat_Object_SP_selected_Bcells_idents, features = "Fcer2a", max.cutoff = 1, spli
t.by = "genotype",
    cols = c("grey", "red"))
```



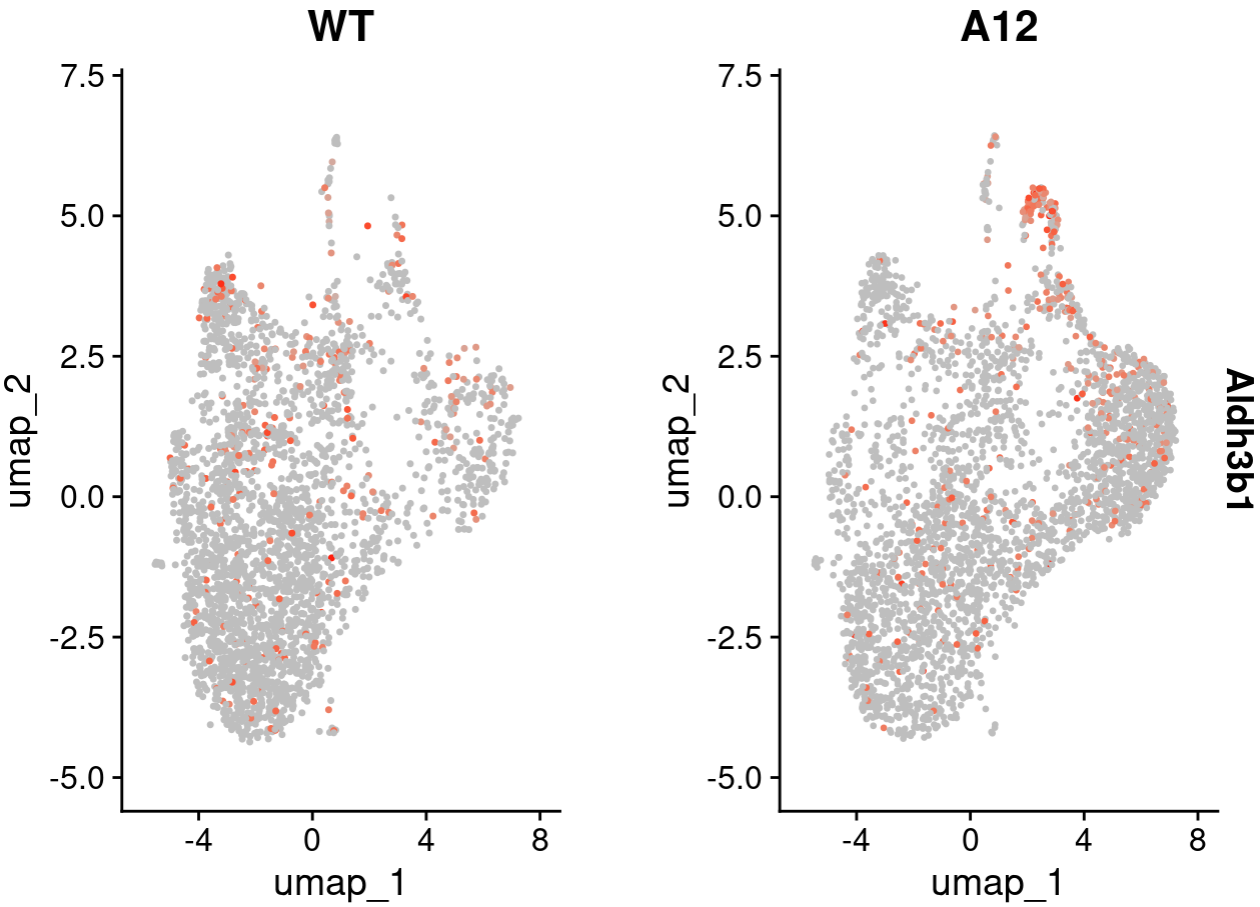

```
FeaturePlot(Seurat_Object_SP_selected_Bcells_idents, features = "Il10ra", max.cutoff = 1, spli
t.by = "genotype",
    cols = c("grey", "red"))
```




```
FeaturePlot(Seurat_Object_BM_selected_Bcells_idents, features = "Il7r", max.cutoff = 5, split.
by = "genotype",
    cols = c("grey", "red"))
```




```
FeaturePlot(Seurat_Object_SP_selected_Bcells_idents, features = "Aldh3b1", max.cutoff = 2, spl
it.by = "genotype",
    cols = c("grey", "red"))
```



Most Relevant Genes per Cluster

Generate subsets for A12 and WT

```
Seurat_Object_BM_A12 <- subset(Seurat_Object_BM_selected_Bcells_idents, genotype == "A12")
Seurat_Object_BM_WT <- subset(Seurat_Object_BM_selected_Bcells_idents, genotype == "WT")
Seurat_Object_SP_A12 <- subset(Seurat_Object_SP_selected_Bcells_idents, genotype == "A12")
Seurat_Object_SP_WT <- subset(Seurat_Object_SP_selected_Bcells_idents, genotype == "WT")
```

The min.pct argument requires a feature to be detected at a minimum percentage in either of the two groups of cells. We report only the positive ones.

Wilcox spleen WT analysis

```
all_markers_SP <- FindAllMarkers(object = Seurat_Object_SP_WT, only.pos = T, min.pct = 0.25, m
in.diff.pct = 0.1, thresh.use = 0.25)
```

```
## Calculating cluster Fo B
```

```
## For a (much!) faster implementation of the Wilcoxon Rank Sum Test,
## (default method for FindMarkers) please install the presto package
```