CRISPRClean Data Analysis Using R

2022-11-15

R Markdown

This is an R Markdown document. Markdown is a simple formatting syntax for authoring HTML, PDF, and MS Word documents. For more details on using R Markdown see http://rmarkdown.rstudio.com.

Import both control and depleted samples

We are going to be using the Seurat toolkit for this downstream analysis. First we will load Seurat and import our samples in .h5 format

```
library(Seurat)
```

```
## Attaching SeuratObject

## Attaching sp
```

```
pbmc_control.mtx <- Read10X_h5("~/R/control_filtered_no_mask.h5")

pbmc_depleted.mtx <- Read10X_h5("~/R/depleted_filtered_no_mask_rep3.h5")
```

Create a Seurat Object

Keep cells expressing a minimum of 200 features. Keep genes expressed in a minimum of 3 cells.

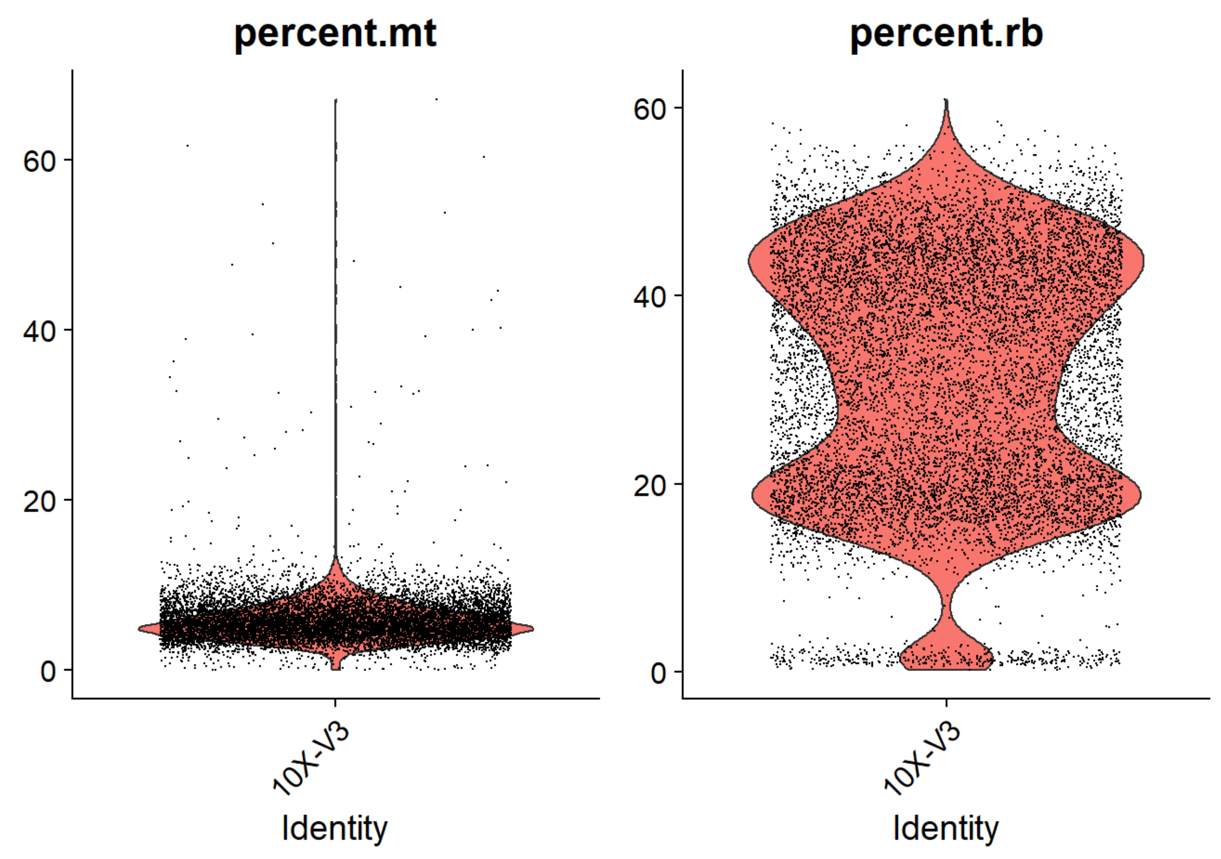
```
pbmc_control.so <- CreateSeuratObject(pbmc_control.mtx, min.cells = 3, min.fe
atures = 200, project = "10X-V3")

pbmc_depleted.so <- CreateSeuratObject(pbmc_depleted.mtx, min.cells = 3, min.
features = 200, project = "CRISPRClean")
```

Calculate percentage of UMI’s contributed by mitochondrial genes

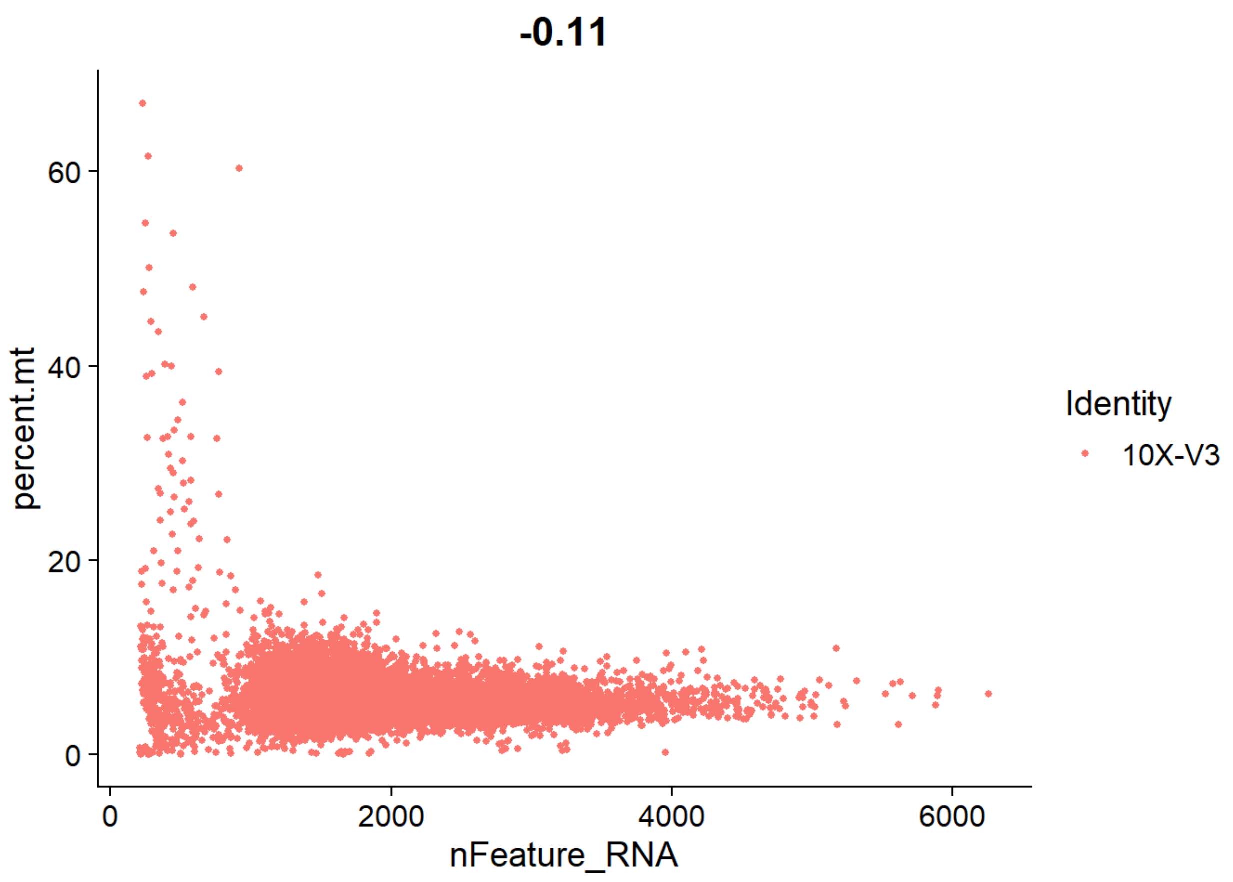
We will look at violin plots of UMIs/cell, Genes/cell, and Mito reads/cell We will also look at a feature scatter plot showing how percentage of mito reads relates to # of genes/cell

```
#control sample

#percent mito
```

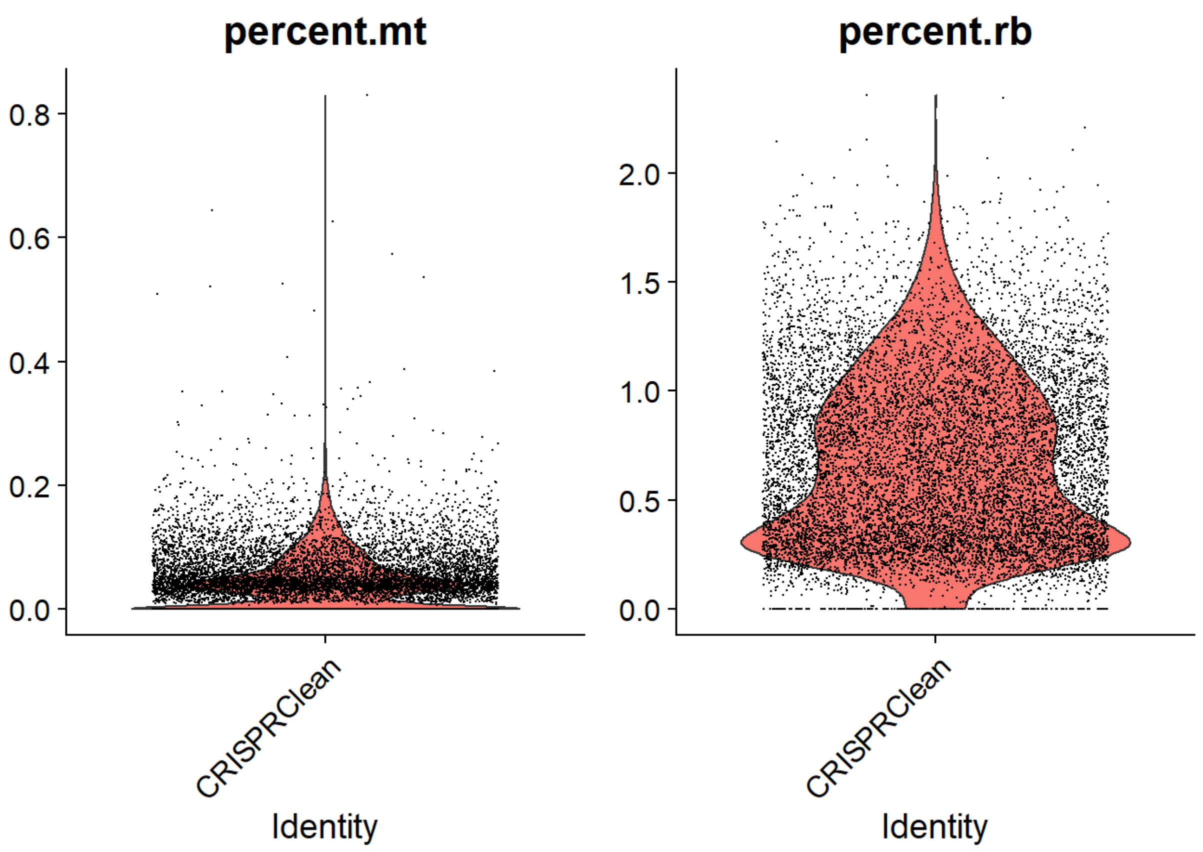
```
pbmc_control.so <- PercentageFeatureSet(pbmc_control.so, pattern = "^MT-", co
l.name = 'percent.mt')

#percent ribo

pbmc_control.so <- PercentageFeatureSet(pbmc_control.so, pattern = "^RP[SL][[
:digit:]]|^RPLP[[:digit:]]|^RPSA", col.name = 'percent.rb')


#fraction of mito and ribo reads per cell

VlnPlot(pbmc_control.so, features = c('percent.mt','percent.rb'), group.by =
'orig.ident')
```



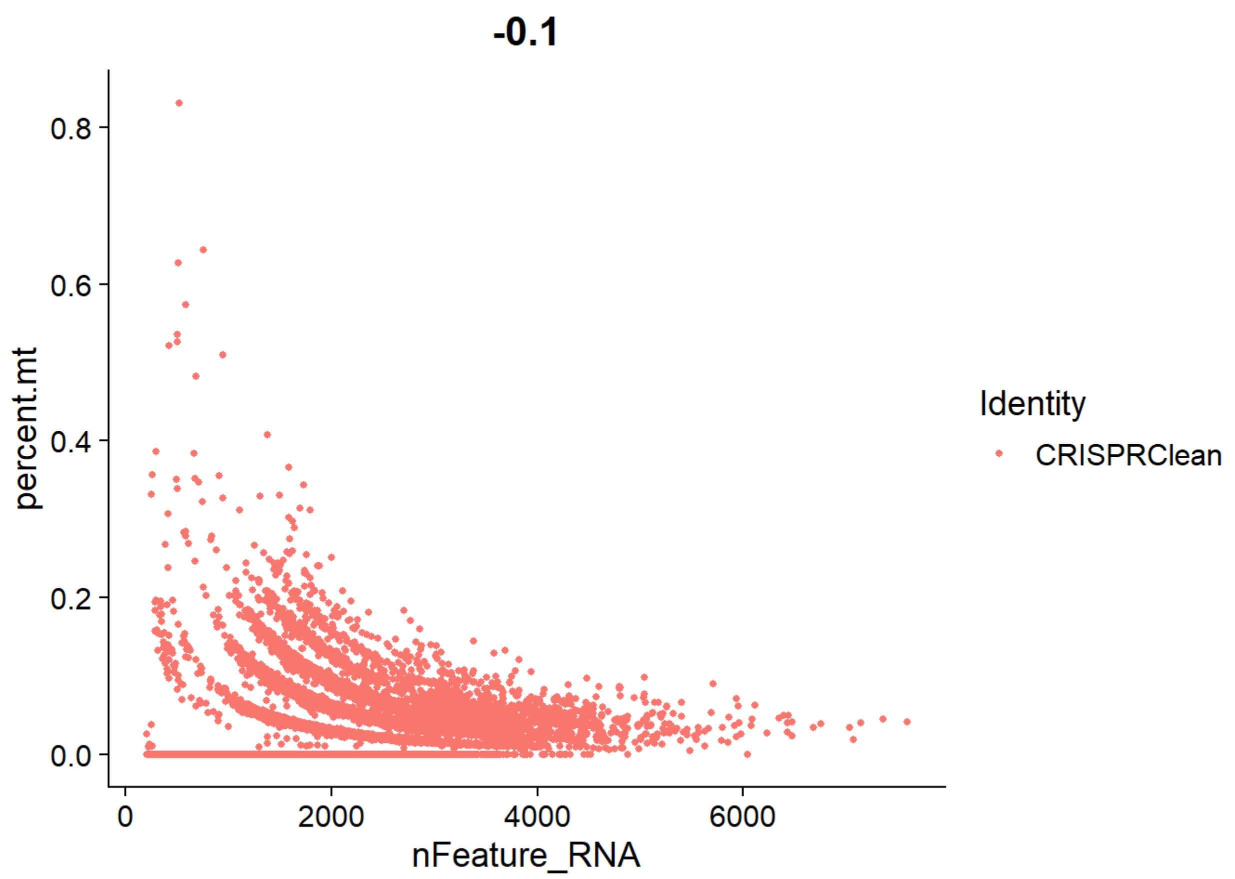
```
FeatureScatter(pbmc_control.so, feature1 = 'nFeature_RNA', feature2 = 'percen
t.mt', group.by = 'orig.ident')
```



```
#depleted sample

#percent mito

pbmc_depleted.so <- PercentageFeatureSet(pbmc_depleted.so, pattern = "^MT-",
col.name = 'percent.mt')

#percent ribo

pbmc_depleted.so <- PercentageFeatureSet(pbmc_depleted.so, pattern = "^RP[SL]
[[:digit:]]|^RPLP[[:digit:]]|^RPSA", col.name = 'percent.rb')


VlnPlot(pbmc_depleted.so, features = c('percent.mt','percent.rb'), group.by =
'orig.ident')
```



```
FeatureScatter(pbmc_depleted.so, feature1 = 'nFeature_RNA', feature2 = 'perce
nt.mt', group.by = 'orig.ident')
```



we can see that we removed a large fraction of mito and ribo reads with depletion

Using miQC for identifying dead cells based on mitochondiral content

We will attempt to build a flexmix model using MiQC (Hippen et. al., 2021) to filter low quality/dead cells. If this flexmix model fails to build (not a high proportion of dead cells) we will then keep cells in the 95th percentile of % mitochondrial reads

```
library(SeuratWrappers)

#control sample

pbmc_control.so <- RunMiQC(pbmc_control.so, percent.mt = 'percent.mt', backup
.option = 'percentile', backup.percentile = 0.95)
```

```
## Warning in RunMiQC(pbmc_control.so, percent.mt = "percent.mt", backup.opti
on =

## "percentile", : flexmix returned only 1 cluster

## defaulting to backup.percentile for filtering

## Warning: Adding a command log without an assay associated with it
```

```
table(pbmc_control.so$miQC.keep)
```

```
##
## discard    keep
##     575   10909
```

We were unable to build flexmix model for this sample, but that’s okay. We filter cells based on being within the 95th percentile of fraction of mitochondrial reads

We want to keep the filtering consistent, so we will filter the 95th percentile in the depleted sample as well

```
library(dplyr)
```

```
##
## Attaching package: 'dplyr'
## The following objects are masked from 'package:stats':
##
##     filter, lag
## The following objects are masked from 'package:base':
##
##     intersect, setdiff, setequal, union
```

```
pbmc_depleted.so@meta.data <- pbmc_depleted.so@meta.data %>% mutate(miQC.keep
= ifelse(test = pbmc_depleted.so@meta.data$percent.mt <= quantile(pbmc_deplet
ed.so@meta.data$percent.mt, probs = 0.95), yes = 'keep', no = 'discard'))

table(pbmc_depleted.so$miQC.keep)
```

```
##
## discard    keep
##     577   10961
```

We can see that we are removing approximately the same number of dead cells in both the control and depleted condition

Filter dead cells

```
#control

ncol(pbmc_control.so)
```

```
## [1] 11484
```

```
pbmc_control.so <- pbmc_control.so[, pbmc_control.so@meta.data[, "miQC.keep"]
== 'keep']

ncol(pbmc_control.so)
```

```
## [1] 10909
#depleted

ncol(pbmc_depleted.so)
## [1] 11538
pbmc_depleted.so <- pbmc_depleted.so[, pbmc_depleted.so@meta.data[, "miQC.kee
p"] == 'keep']

ncol(pbmc_depleted.so)
## [1] 10961
```

Doublet Removal

We are going to use the doublet removal toolkit scDblFinder (Germain et. al., 2022). We first need to go through an initial round of clustering to simulate artificial doublets and subsequent removal

For clustering, we are going to use the SCTransform workflow We want to perform clustering using residual default cutoff of 1.3 rather than selecting a fixed number of highly variable genes. We will also be regressing out the percentage of mito reads so as to not affect clustering

We will also be scoring cell cycle genes to eventually regress out as well

initial control clustering

```
#SCTransform

pbmc_control.so <- SCTransform(pbmc_control.so, variable.features.n = NULL, v
ariable.features.rv.th = 1.3, vars.to.regress = 'percent.mt')
## Calculating cell attributes from input UMI matrix: log_umi
## Variance stabilizing transformation of count matrix of size 19878 by 10909
## Model formula is y ~ log_umi
## Get Negative Binomial regression parameters per gene
## Using 2000 genes, 5000 cells
##

  |

  |                                                                      |
0%

  |

  |==================                                                    |  2
5%

  |

  |===================================                                   |  5
0%

  |
  |=====================================================                  |  7
5%
  |
  |======================================================================| 10
0%
## Found 82 outliers - those will be ignored in fitting/regularization step
## Second step: Get residuals using fitted parameters for 19878 genes
##
  |
  |                                                                      |
0%
  |
  |==                                                                    |
2%
  |
  |====                                                                  |
5%
  |
  |=====                                                                 |
8%
  |
  |=======                                                               |  1
0%
  |
  |=========                                                             |  1
2%
  |
  |==========                                                            |  1
5%
  |
  |============                                                          |  1
8%
  |
  |==============                                                        |  2
0%
  |
  |================                                                      |  2
2%
  |
  |==================                                                    |  2
5%

  |
  |===================                                                   |  2
8%

  |
  |=====================                                                 |  3
0%

  |
  |=======================                                               |  3
2%

  |
  |========================                                              |  3
5%

  |
  |==========================                                            |  3
8%

  |
  |============================                                          |  4
0%

  |
  |==============================                                        |  4
2%

  |
  |================================                                      |  4
5%

  |
  |=================================                                     |  4
8%

  |
  |===================================                                   |  5
0%

  |
  |=====================================                                 |  5
2%

  |
  |======================================                                |  5
5%

  |
  |========================================                                |  5
8%
  |
  |==========================================                              |  6
0%
  |
  |============================================                            |  6
2%
  |
  |==============================================                          |  6
5%
  |
  |===============================================                         |  6
8%
  |
  |=================================================                       |  7
0%
  |
  |===================================================                     |  7
2%
  |
  |====================================================                    |  7
5%
  |
  |======================================================                  |  7
8%
  |
  |========================================================                |  8
0%
  |
  |==========================================================              |  8
2%
  |
  |============================================================            |  8
5%
  |
  |=============================================================           |  8
8%
  |
  |===============================================================       |  9
0%

  |

  |=================================================================     |  9
2%

  |

  |==================================================================    |  9
5%

  |

  |====================================================================  |  9
8%

  |

  |======================================================================| 10
0%

## Computing corrected count matrix for 19878 genes

##

  |

  |                                                                      |
0%

  |

  |==                                                                    |
2%

  |

  |====                                                                  |
5%

  |

  |=====                                                                 |
8%

  |

  |=======                                                               |  1
0%

  |

  |=========                                                             |  1
2%

  |

  |==========                                                            |  1
5%

  |
  |============                                                              |  1
8%

  |
  |==============                                                            |  2
0%

  |
  |================                                                          |  2
2%

  |
  |==================                                                        |  2
5%

  |
  |===================                                                       |  2
8%

  |
  |=====================                                                     |  3
0%

  |
  |=======================                                                   |  3
2%

  |
  |========================                                                  |  3
5%

  |
  |==========================                                                |  3
8%

  |
  |============================                                              |  4
0%

  |
  |==============================                                            |  4
2%

  |
  |================================                                          |  4
5%

  |
  |=================================                                         |  4
8%

  |
  |===================================                                   |  5
0%

  |
  |=====================================                                 |  5
2%

  |
  |======================================                                |  5
5%

  |
  |========================================                              |  5
8%

  |
  |==========================================                            |  6
0%

  |
  |============================================                          |  6
2%

  |
  |==============================================                        |  6
5%

  |
  |===============================================                       |  6
8%

  |
  |=================================================                     |  7
0%

  |
  |===================================================                   |  7
2%

  |
  |====================================================                  |  7
5%

  |
  |======================================================                |  7
8%

  |
  |========================================================              |  8
0%

  |
  |==========================================================            |  8
2%
  |
  |============================================================          |  8
5%
  |
  |=============================================================         |  8
8%
  |
  |===============================================================       |  9
0%
  |
  |=================================================================     |  9
2%
  |
  |==================================================================    |  9
5%
  |
  |====================================================================  |  9
8%
  |
  |======================================================================| 10
0%
## Calculating gene attributes
## Wall clock passed: Time difference of 4.147565 mins
## Determine variable features
## Place corrected count matrix in counts slot
## Regressing out percent.mt
## Centering data matrix
## Set default assay to SCT
```

```
#cell cyclce scoring
pbmc_control.so <- CellCycleScoring(pbmc_control.so, s.features = cc.genes.up
dated.2019$s.genes, g2m.features = cc.genes.updated.2019$g2m.genes)
#PCA
pbmc_control.so <- RunPCA(pbmc_control.so)
```

```
## PC_ 1
## Positive:  NKG7, GNLY, CCL5, GZMA, IL7R, LTB, IL32, GZMB, RPS27, KLRB1
##     KLRD1, CST7, FGFBP2, RPS12, PRF1, RPL13, HOPX, RPS18, TRBC2, TRBC1
##     GZMK, CD3D, RPL3, RPL10, RPS27A, ARL4C, TRAC, RPS29, PCED1B-AS1, RPS3
## Negative:  LYZ, S100A9, S100A8, CTSS, VCAN, FCN1, CST3, NEAT1, MNDA, FTL
##     S100A12, AIF1, IFI30, FOS, TYROBP, HLA-DRA, PSAP, LST1, CD14, SAT1
##     S100A6, FTH1, CYBB, FGL2, COTL1, S100A11, S100A4, CD74, MS4A6A, CEBPD
## PC_ 2
## Positive:  PPBP, PF4, NRGN, GP1BB, GNG11, CAVIN2, TUBB1, HIST1H2AC, GP9, R
GS18
##     PRKAR2B, ACRBP, PTCRA, TSC22D1, CMTM5, TMEM40, CCL5, MAP3K7CL, MMD, OD
C1
##     C2orf88, F13A1, CLU, CLDN5, RUFY1, LGALSL, LIMS1, AC127502.2, TREML1,
SPARC
## Negative:  RPL13, LYZ, RPS12, EEF1A1, RPS27, S100A9, RPL10, TPT1, RPS18, S
100A8
##     RPL30, RPL32, RPLP1, RPS14, RPL41, LTB, RPL34, RPL11, HLA-DRA, RPS3A
##     RPS6, RPS8, RPS27A, RPS4X, RPS23, RPS29, IL7R, RPS15A, RPS3, CTSS
## PC_ 3
## Positive:  HLA-DRA, CD74, MS4A1, IGHM, IGKC, CD79A, HLA-DPB1, HLA-DPA1, IG
HD, TCL1A
##     BANK1, HLA-DQA1, HLA-DQB1, LTB, CD79B, IGLC2, RALGPS2, HLA-DRB1, TNFRS
F13C, LINC00926
##     LINC02397, FCRL1, FCER2, VPREB3, IGLC3, RPL13, CD22, RPS27, RPS12, RPS
18
## Negative:  GNLY, NKG7, GZMB, GZMA, CCL5, FGFBP2, KLRD1, PRF1, CST7, HOPX
##     CLIC3, KLRF1, SPON2, KLRB1, CCL4, TRDC, CTSW, GZMH, TYROBP, IL2RB
##     CD160, CMC1, KLRK1, FCER1G, FCGR3A, GZMM, XCL2, CD7, AKR1C3, MATK
## PC_ 4
## Positive:  IL7R, S100A8, VCAN, S100A9, CD3D, S100A12, TRAC, CD3E, IL32, BC
L11B
##     CD3G, TPT1, RPS12, PRKCQ-AS1, LDHB, GZMK, TCF7, TRBC2, TRBC1, GIMAP7
##     RPL13, FOS, MAL, FYB1, LTB, NOSIP, LEF1, RPS14, RCAN3, RPL30
## Negative:  HLA-DRA, CD74, IGKC, IGHM, MS4A1, HLA-DPA1, HLA-DPB1, CD79A, GN
LY, NKG7
##     GZMB, HLA-DQA1, HLA-DQB1, HLA-DRB1, TCL1A, IGHD, BANK1, CD79B, CLIC3,
RALGPS2
##     FGFBP2, IGLC2, TCF4, KLRD1, JCHAIN, GZMA, HLA-DRB5, BCL11A, PRF1, IRF8
## PC_ 5
## Positive:  S100A8, VCAN, S100A9, S100A12, FOS, IGHM, LYZ, IGKC, MS4A1, CD7
9A
##     MNDA, TCL1A, IGHD, CD14, CYP1B1, BANK1, THBS1, RALGPS2, NCF1, IGLC2
##     CSF3R, CD36, PLBD1, GNLY, MEGF9, LINC02397, SLC2A3, LINC00926, FCRL1,
TNFRSF13C

## Negative:  FCGR3A, CDKN1C, LST1, AIF1, CST3, MS4A7, IFITM3, HLA-DPA1, SAT1
, IFI30

##     FCER1G, FTL, COTL1, C1QA, TCF7L2, SMIM25, HLA-DPB1, FTH1, LYPD2, CSF1R

##     CALHM6, RHOC, HLA-DRB1, TMSB4X, CEBPB, HES4, SERPINA1, HMOX1, TUBA1B,
MTSS1
```

```
#Find k-nearest neighbors using the first 30 dimensions

pbmc_control.so <- FindNeighbors(pbmc_control.so, dims = 1:30)
```

```
## Computing nearest neighbor graph

## Computing SNN
```

```
#generate UMAP coordinates

pbmc_control.so <- RunUMAP(pbmc_control.so, dims = 1:30)
```

```
## Warning: The default method for RunUMAP has changed from calling Python UM
AP via reticulate to the R-native UWOT using the cosine metric

## To use Python UMAP via reticulate, set umap.method to 'umap-learn' and met
ric to 'correlation'

## This message will be shown once per session

## 15:57:06 UMAP embedding parameters a = 0.9922 b = 1.112

## 15:57:06 Read 10909 rows and found 30 numeric columns

## 15:57:06 Using Annoy for neighbor search, n_neighbors = 30

## 15:57:06 Building Annoy index with metric = cosine, n_trees = 50

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

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

## **************************************************|

## 15:57:09 Writing NN index file to temp file C:\Users\dante\AppData\Local\T
emp\Rtmp2Xrtmf\file37e0406a671a

## 15:57:09 Searching Annoy index using 1 thread, search_k = 3000

## 15:57:16 Annoy recall = 100%

## 15:57:17 Commencing smooth kNN distance calibration using 1 thread

## 15:57:18 Initializing from normalized Laplacian + noise

## 15:57:20 Commencing optimization for 200 epochs, with 452860 positive edge
s

## 15:57:44 Optimization finished
```

```
#Find clusters using the louvain algorithm with multilevel refinement. It is
recommended to overcluster the data first when using scDblFinder

pbmc_control.so <- FindClusters(pbmc_control.so, resolution = 1.2, algorithm
= 2)
```

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10909
## Number of edges: 381970
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.8536
## Number of communities: 28
## Elapsed time: 3 seconds
```

initial depleted clustering

```
#SCTransform
pbmc_depleted.so <- SCTransform(pbmc_depleted.so, variable.features.n = NULL,
variable.features.rv.th = 1.3, vars.to.regress = 'percent.mt')
```

```
## Calculating cell attributes from input UMI matrix: log_umi
## Variance stabilizing transformation of count matrix of size 19898 by 10961
## Model formula is y ~ log_umi
## Get Negative Binomial regression parameters per gene
## Using 2000 genes, 5000 cells
##
  |
  |                                                                      |
0%
  |
  |==================                                                    |  2
5%
  |
  |===================================                                   |  5
0%
  |
  |====================================================                  |  7
5%
  |
  |======================================================================| 10
0%
## Found 74 outliers - those will be ignored in fitting/regularization step
## Second step: Get residuals using fitted parameters for 19898 genes
```

```
##
   |
   |                                                                          |
0%
   |
   |==                                                                        |
2%
   |
   |====                                                                      |
5%
   |
   |=====                                                                     |
8%
   |
   |=======                                                                   |   1
0%
   |
   |=========                                                                 |   1
2%
   |
   |==========                                                                |   1
5%
   |
   |============                                                              |   1
8%
   |
   |==============                                                            |   2
0%
   |
   |================                                                          |   2
2%
   |
   |==================                                                        |   2
5%
   |
   |===================                                                       |   2
8%
   |
  |=====================                                                 |  3
0%

  |
  |=======================                                               |  3
2%

  |
  |========================                                              |  3
5%

  |
  |==========================                                            |  3
8%

  |
  |============================                                          |  4
0%

  |
  |==============================                                        |  4
2%

  |
  |================================                                      |  4
5%

  |
  |=================================                                     |  4
8%

  |
  |===================================                                   |  5
0%

  |
  |=====================================                                 |  5
2%

  |
  |======================================                                |  5
5%

  |
  |========================================                              |  5
8%

  |
  |==========================================                            |  6
0%

  |
  |============================================                           |  6
2%

  |
  |==============================================                         |  6
5%

  |
  |===============================================                        |  6
8%

  |
  |=================================================                      |  7
0%

  |
  |===================================================                    |  7
2%

  |
  |====================================================                   |  7
5%

  |
  |======================================================                 |  7
8%

  |
  |========================================================               |  8
0%

  |
  |==========================================================             |  8
2%

  |
  |============================================================           |  8
5%

  |
  |=============================================================          |  8
8%

  |
  |===============================================================        |  9
0%

  |
  |=================================================================      |  9
2%

  |
  |==================================================================    |  9
5%

  |

  |====================================================================  |  9
8%

  |

  |======================================================================| 10
0%

## Computing corrected count matrix for 19898 genes

##

  |

  |                                                                      |
0%

  |

  |==                                                                    |
2%

  |

  |====                                                                  |
5%

  |

  |=====                                                                 |
8%

  |

  |=======                                                               |  1
0%

  |

  |=========                                                             |  1
2%

  |

  |==========                                                            |  1
5%

  |

  |============                                                          |  1
8%

  |

  |==============                                                        |  2
0%

  |
  |================                                                      |  2
2%
  |
  |==================                                                    |  2
5%
  |
  |===================                                                   |  2
8%
  |
  |=====================                                                 |  3
0%
  |
  |=======================                                               |  3
2%
  |
  |========================                                              |  3
5%
  |
  |==========================                                            |  3
8%
  |
  |============================                                          |  4
0%
  |
  |==============================                                        |  4
2%
  |
  |================================                                      |  4
5%
  |
  |=================================                                     |  4
8%
  |
  |===================================                                   |  5
0%
  |
  |=====================================                                 |  5
2%
  |
  |======================================                                |  5
5%

  |

  |========================================                              |  5
8%

  |

  |==========================================                            |  6
0%

  |

  |============================================                          |  6
2%

  |

  |==============================================                        |  6
5%

  |

  |===============================================                       |  6
8%

  |

  |=================================================                     |  7
0%

  |

  |===================================================                   |  7
2%

  |

  |====================================================                  |  7
5%

  |

  |======================================================                |  7
8%

  |

  |========================================================              |  8
0%

  |

  |==========================================================            |  8
2%

  |

  |============================================================          |  8
5%

  |
  |=============================================================          |  8
8%
  |
  |===============================================================        |  9
0%
  |
  |=================================================================      |  9
2%
  |
  |==================================================================     |  9
5%
  |
  |====================================================================   |  9
8%
  |
  |======================================================================| 10
0%
## Calculating gene attributes
## Wall clock passed: Time difference of 2.525177 mins
## Determine variable features
## Place corrected count matrix in counts slot
## Regressing out percent.mt
## Centering data matrix
## Set default assay to SCT
```

```
#cell cyclce scoring
pbmc_depleted.so <- CellCycleScoring(pbmc_depleted.so, s.features = cc.genes.
updated.2019$s.genes, g2m.features = cc.genes.updated.2019$g2m.genes)
#PCA
pbmc_depleted.so <- RunPCA(pbmc_depleted.so)
```

```
## PC_ 1
## Positive:  NKG7, GNLY, IL7R, GZMA, LTB, CCL5, GZMB, KLRB1, CST7, FGFBP2
##     PRF1, MALAT1, TRBC1, HOPX, CD3D, IL32, ARL4C, CD3E, TRAC, CD7
##     GZMK, ETS1, KLRD1, CD2, CTSW, BCL11B, GZMM, CLIC3, PCED1B-AS1, KLRF1
## Negative:  LYZ, S100A9, VCAN, HLA-DRA, MNDA, AIF1, S100A12, IFI30, CD74, C
D14
##     CTSS, CYBB, FTH1, S100A8, FCN1, COTL1, FGL2, HLA-DRB1, HLA-DPA1, MS4A6
A
##     GRN, TKT, FOS, CEBPD, FTL, MPEG1, TYMP, NCF2, CST3, S100A6
## PC_ 2
## Positive:  GNLY, NKG7, GZMA, GZMB, IL7R, FGFBP2, LTB, KLRB1, MALAT1, CST7
##     PRF1, TMSB10, HOPX, CD52, KLRD1, CLIC3, ARL4C, KLRF1, CTSW, IFITM2
##     CYBA, ZFP36L2, CD7, HCST, SPON2, VIM, GIMAP7, IL32, TRBC1, TRDC
## Negative:  NRGN, PF4, GP1BB, GNG11, CAVIN2, TUBB1, HIST1H2AC, GP9, RGS18,
ACRBP
##     CMTM5, PTCRA, TSC22D1, PRKAR2B, PPBP, TMEM40, CLU, C2orf88, MAP3K7CL,
MMD
##     CLDN5, F13A1, ODC1, LGALSL, SPARC, TREML1, MTURN, RUFY1, AC147651.1, C
TTN
## PC_ 3
## Positive:  LTB, HLA-DRA, CD79A, MS4A1, CD74, IGHM, IGKC, IL7R, CD52, HLA-D
PB1
##     IGHD, TCL1A, BANK1, HLA-DQB1, HLA-DQA1, HLA-DPA1, CD79B, FCRL1, LINC02
397, RALGPS2
##     IGLC2, TNFRSF13C, CD37, FCER2, LINC00926, VPREB3, TRAC, CD3E, HLA-DRB1
, AFF3
## Negative:  GNLY, NKG7, GZMB, GZMA, FGFBP2, CCL5, PRF1, CST7, KLRD1, HOPX
##     CLIC3, KLRF1, SPON2, KLRB1, S100A9, LYZ, CTSW, TRDC, CCL4, FCGR3A
##     GZMH, CMC1, KLRK1, VCAN, CD160, IL2RB, MATK, AKR1C3, GZMM, EFHD2
## PC_ 4
## Positive:  HLA-DRA, CD74, MS4A1, CD79A, IGHM, IGKC, HLA-DPB1, GNLY, NKG7,
HLA-DPA1
##     HLA-DQA1, HLA-DQB1, TCL1A, IGHD, GZMB, BANK1, HLA-DRB1, CD79B, FCRL1,
LINC02397
##     RALGPS2, IGLC2, GZMA, TNFRSF13C, FCER2, LINC00926, FGFBP2, VPREB3, BCL
11A, CLIC3
## Negative:  IL7R, S100A9, VCAN, CD3E, TRAC, LYZ, CD3D, BCL11B, S100A12, VIM
##     CD3G, TRBC1, PRKCQ-AS1, LTB, FYB1, GIMAP7, MAL, MNDA, S100A8, NOSIP
##     LEF1, SARAF, RCAN3, TRAT1, TCF7, CCR7, INPP4B, GIMAP4, LIME1, LINC0086
1
## PC_ 5
## Positive:  PLD4, SERPINF1, LILRA4, IL3RA, ITM2C, CCDC50, UGCG, JCHAIN, GZM
B, PPP1R14B
##     TPM2, IRF8, IRF7, LRRC26, CLEC4C, DNASE1L3, SCT, TCF4, MZB1, DERL3
##     SMPD3, LINC00996, FCGR3A, SCN9A, MAP1A, EPHB1, HLA-DPA1, SCAMP5, CDKN1
C, NPC2
## Negative:  MS4A1, CD79A, VCAN, IGHM, S100A9, IGHD, S100A12, FCRL1, BANK1,
CD79B
```

```
##     S100A8, TCL1A, FCER2, IGLC2, LINC02397, RALGPS2, LINC00926, VPREB3, TN
FRSF13C, LYZ
##     MNDA, GNLY, CD22, NKG7, IGLC3, PAX5, CD14, CD37, FOS, THBS1
```

```
#Find k-nearest neighbors using the first 30 dimensions
pbmc_depleted.so <- FindNeighbors(pbmc_depleted.so, dims = 1:30)
```

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

```
#generate UMAP coordinates
pbmc_depleted.so <- RunUMAP(pbmc_depleted.so, dims = 1:30)
```

```
## 16:01:08 UMAP embedding parameters a = 0.9922 b = 1.112
## 16:01:08 Read 10961 rows and found 30 numeric columns
## 16:01:08 Using Annoy for neighbor search, n_neighbors = 30
## 16:01:08 Building Annoy index with metric = cosine, n_trees = 50
## 0%   10   20   30   40   50   60   70   80   90   100%
## [----|----|----|----|----|----|----|----|----|----|
## **************************************************|
## 16:01:10 Writing NN index file to temp file C:\Users\dante\AppData\Local\T
emp\Rtmp2Xrtmf\file37e07dd436fe
## 16:01:10 Searching Annoy index using 1 thread, search_k = 3000
## 16:01:13 Annoy recall = 100%
## 16:01:15 Commencing smooth kNN distance calibration using 1 thread
## 16:01:16 Initializing from normalized Laplacian + noise
## 16:01:17 Commencing optimization for 200 epochs, with 454818 positive edge
s
## 16:01:31 Optimization finished
```

```
#Find clusters using the louvain algorithm with multilevel refinement. It is
recommended to overcluster the data first when using scDblFinder
pbmc_depleted.so <- FindClusters(pbmc_depleted.so, resolution = 1.2, algorith
m = 2)
```

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10961
## Number of edges: 382358
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.8656
## Number of communities: 28
## Elapsed time: 2 seconds
```

scDblFinder

we use the natural log normalized features to simulate artificial doublets. The number of top features corresponds to the number of highly variable genes. We are also using the same number of dimensions (30) used in the first clustering iterations. The expected doublet rate is assumed to be 1% per thousand cells captured which is appropriate for 10x datasets.

```
library(scDblFinder)
library(SingleCellExperiment)
```

```
## Loading required package: SummarizedExperiment
## Loading required package: MatrixGenerics
## Loading required package: matrixStats
##
## Attaching package: 'matrixStats'
## The following object is masked from 'package:dplyr':
##
##     count
##
## 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 objects are masked from 'package:dplyr':
##
##     combine, intersect, setdiff, union
## The following objects are masked from 'package:stats':
##
##     IQR, mad, sd, var, xtabs
## The following objects are masked from 'package:base':
##
##     anyDuplicated, 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 objects are masked from 'package:dplyr':
##
##     first, rename
## The following objects are masked from 'package:base':
##
##     expand.grid, I, unname
## Loading required package: IRanges
##
## Attaching package: 'IRanges'
## The following objects are masked from 'package:dplyr':
```

```
##
##     collapse, desc, slice
## The following object is masked from 'package:sp':
##
##     %over%
## The following object is masked from 'package:grDevices':
##
##     windows
## 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:SeuratObject':
##
##     Assays
## The following object is masked from 'package:Seurat':
##
##     Assays
```

```
#natural log normalize the raw counts data. SCTransform counts data uses pear
son residuals which can only be used for clustering/visualization
pbmc_control.so <- NormalizeData(pbmc_control.so, assay = 'RNA')
#run scdblfinder
```

```
scdblfinder.control <- scDblFinder(as.SingleCellExperiment(pbmc_control.so, a
ssay = 'RNA'), clusters = 'seurat_clusters',
                                    dbr = NULL, nfeatures = length(pbmc_control.s
o@assays$SCT@var.features),
                                    dims = 30, includePCs = 30, processing = "nor
mFeatures")
```

```
## 28 clusters
## Creating ~8728 artificial doublets...
## Dimensional reduction
## Evaluating kNN...
## Training model...
## iter=0, 885 cells excluded from training.
## iter=1, 664 cells excluded from training.
## iter=2, 604 cells excluded from training.
## Threshold found:0.471
## 604 (5.5%) doublets called
```

```
#natural log normalize the raw counts data. SCTransform counts data uses pear
son residuals which can only be used for clustering/visualization
pbmc_depleted.so <- NormalizeData(pbmc_depleted.so, assay = 'RNA')
#run scdblfinder
scdblfinder.depleted <- scDblFinder(as.SingleCellExperiment(pbmc_depleted.so,
assay = 'RNA'), clusters = 'seurat_clusters',
                                    dbr = NULL, nfeatures = length(pbmc_depleted.
so@assays$SCT@var.features),
                                    dims = 30, includePCs = 30, processing = "nor
mFeatures")
```

```
## 28 clusters
## Creating ~8769 artificial doublets...
## Dimensional reduction
## Evaluating kNN...
## Training model...
## iter=0, 1009 cells excluded from training.
## iter=1, 619 cells excluded from training.
## iter=2, 505 cells excluded from training.
## Threshold found:0.538
## 496 (4.5%) doublets called
```

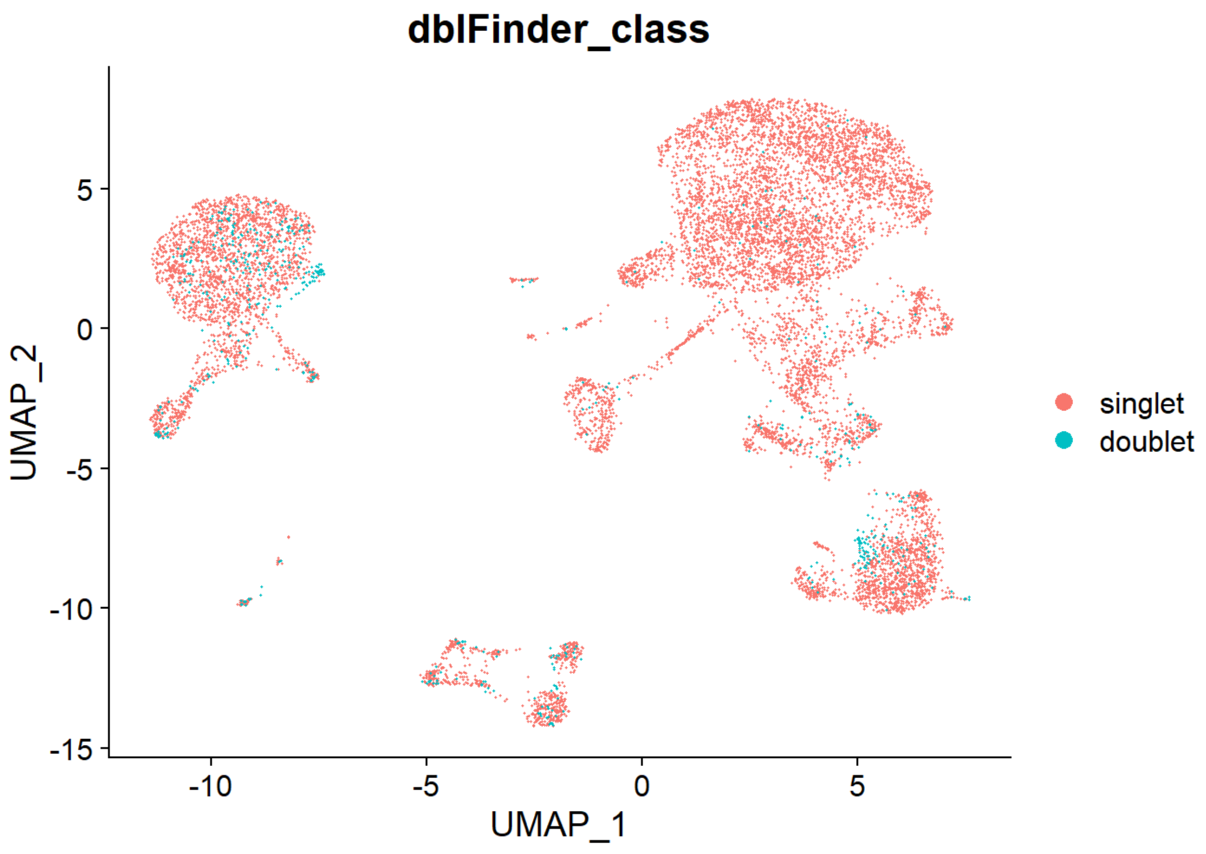
scDblFinder creates a SingleCellExperiment object with a metadata column ‘dblFinder_class’ containing the ‘singlet’ or ‘doublet’ call information per cell. We will extract this column from the SingleCellExperiment object and add it to our seurat object.

We will also look at both DimPlots and violin plots as a QC metric to ensure that the called doublets meet 2 requirements: On average doublets should have approximately twice the UMI/cell and genes/cell

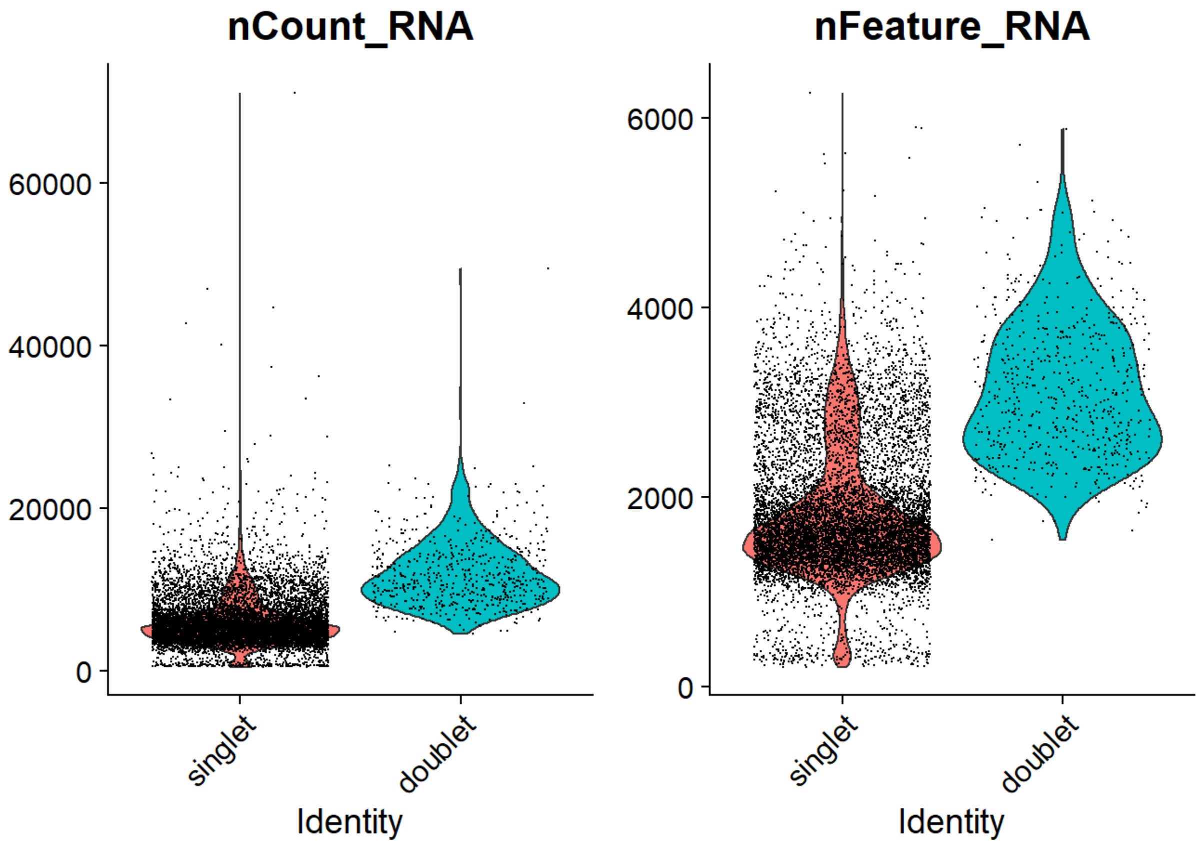
```
#control

pbmc_control.so@meta.data$dblFinder_class <- scdblfinder.control$scDblFinder.
class

table(pbmc_control.so@meta.data$dblFinder_class)
```

```
##
## singlet doublet
##   10305     604
```

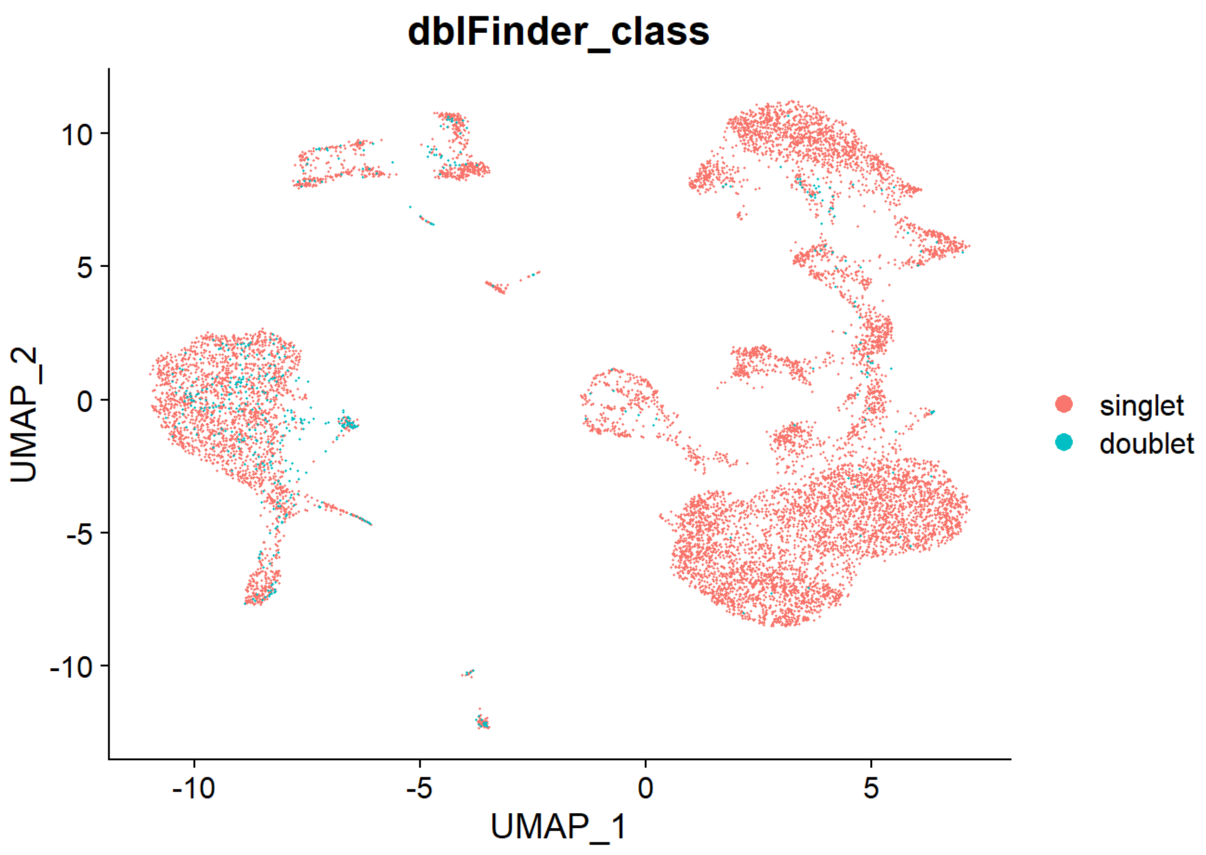
```
DimPlot(pbmc_control.so, group.by = "dblFinder_class", order = T)
```



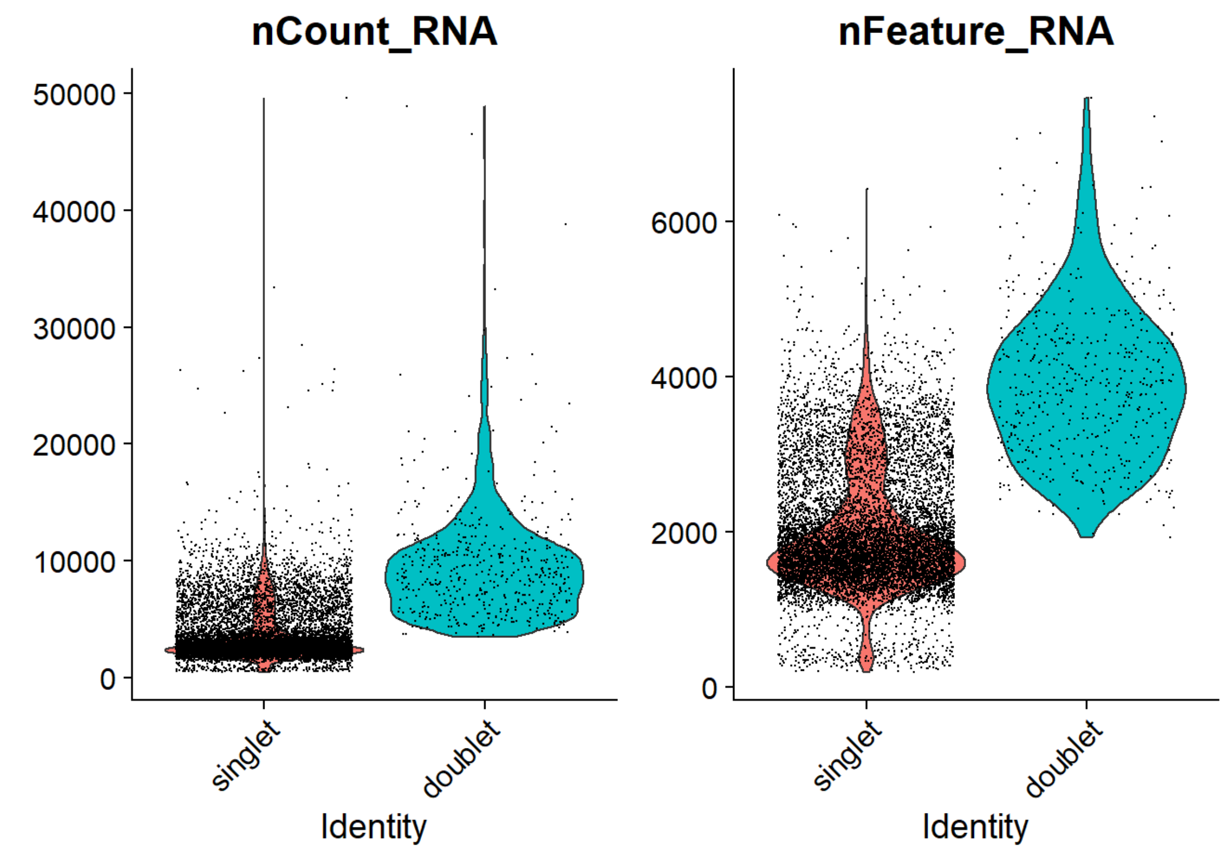
```
VlnPlot(pbmc_control.so, features = c("nCount_RNA", "nFeature_RNA"), group.by
= "dblFinder_class")
```



```
#depleted
pbmc_depleted.so@meta.data$dblFinder_class <- scdblfinder.depleted$scDblFinde
r.class
table(pbmc_depleted.so@meta.data$dblFinder_class)
##
## singlet doublet
##   10465     496
DimPlot(pbmc_depleted.so, group.by = "dblFinder_class", order = T)
```



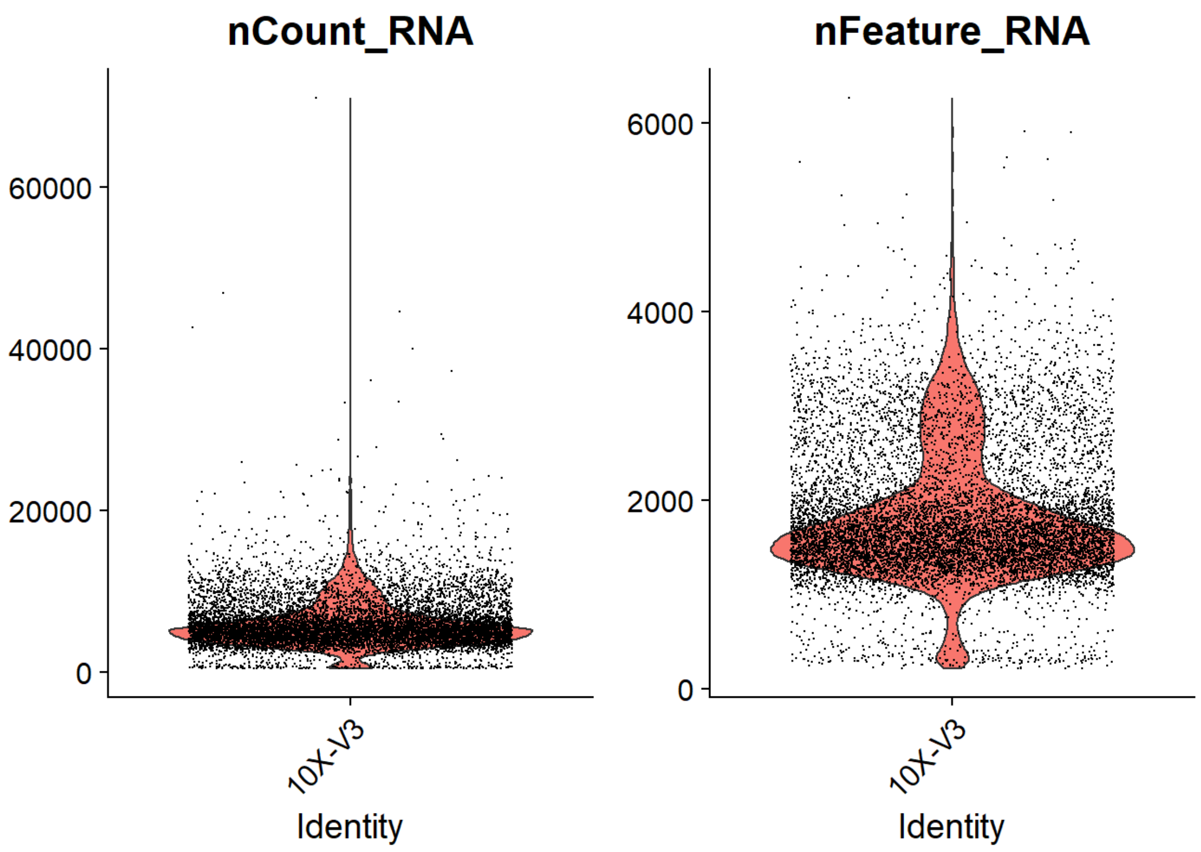
```
VlnPlot(pbmc_depleted.so, features = c("nCount_RNA", "nFeature_RNA"), group.b
y = "dblFinder_class")
```



We can see that the called doublets represent distinct populations within some clusters. Additionally, the number of UMIs/cell and genes/cell is roughly twice as much which gives us confidence in the accuracy of called doublets

We will now remove these cells called as doublets from the Seurat object We are also going to look at what the distribution of UMIs/cell and genes/cell looks like after doublet removal

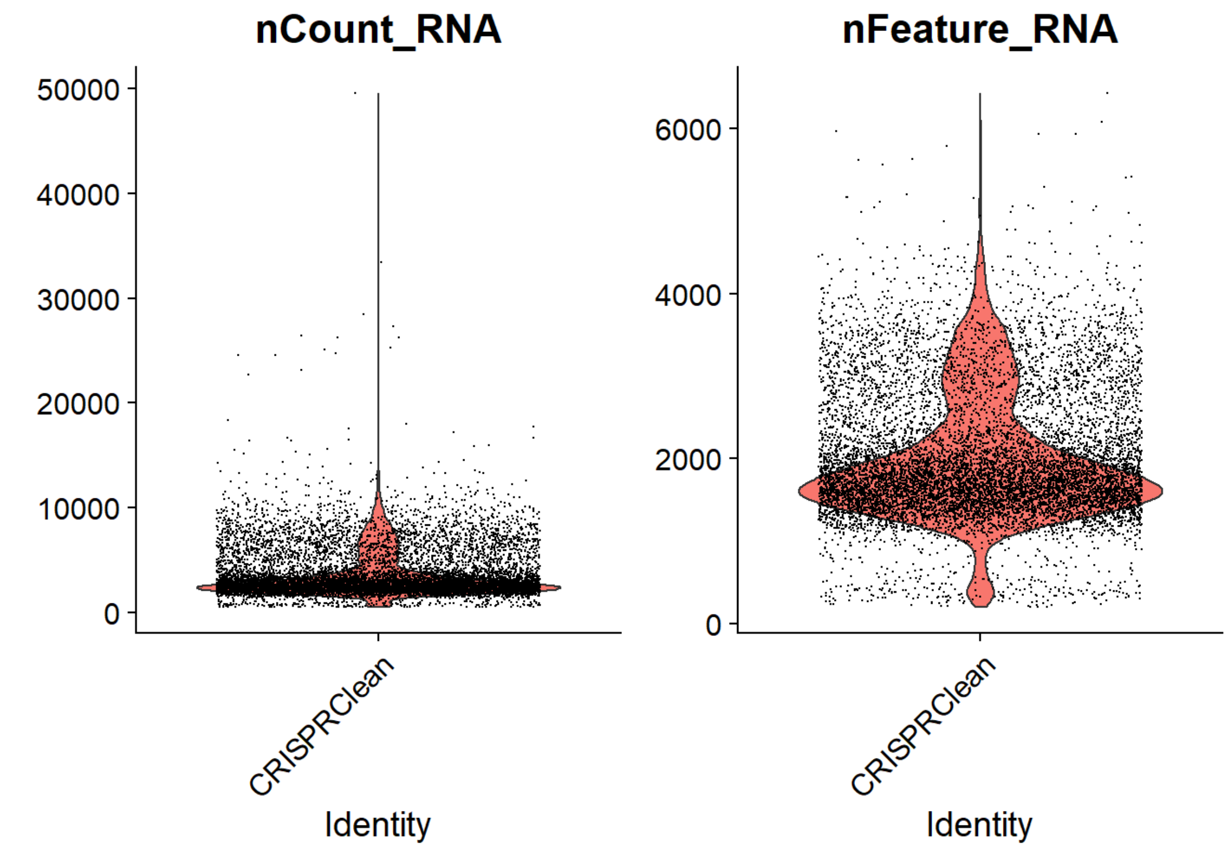
```
#control

pbmc_control.so <- pbmc_control.so[, pbmc_control.so@meta.data[, "dblFinder_c
lass"] == "singlet"]

VlnPlot(pbmc_control.so, features = c('nCount_RNA','nFeature_RNA'), group.by
= 'orig.ident')
```



```
ncol(pbmc_control.so)
```

```
## [1] 10305
```

```
#depleted

pbmc_depleted.so <- pbmc_depleted.so[, pbmc_depleted.so@meta.data[, "dblFinde
r_class"] == "singlet"]

VlnPlot(pbmc_depleted.so, features = c('nCount_RNA','nFeature_RNA'), group.by
= 'orig.ident')
```



```
ncol(pbmc_depleted.so)
## [1] 10465
```

After doublet removal, we can see we still have some cells with outlier UMI counts. These could be homotypic doublets or possibly introduce some unwanted variation for cell clustering. Given that there are very few, we will be conservative and filter cells in the top 1% of UMI counts

As a result, we will create a new metadata dataframe with some key metrics

```
pbmc_control.meta <- pbmc_control.so@meta.data

#we are going to add two columns: One column for cells with high sequencing d
epth and another column quantifying the complexity of each cell


pbmc_control.meta <- pbmc_control.meta %>% mutate(highdepth = ifelse(test = p
bmc_control.meta$nCount_RNA <= quantile(pbmc_control.meta$nCount_RNA, probs =
0.99), yes = 'keep', no = 'discard'), complexity = log10(pbmc_control.meta$nF
eature_RNA)/log10(pbmc_control.meta$nCount_RNA))


#add new metadata to Seurat object

pbmc_control.so@meta.data <- pbmc_control.meta
```

```
#for now let's remove the cells with high sequencing depth
table(pbmc_control.so@meta.data$highdepth)
```

```
##
## discard    keep
##     104   10201
```

```
pbmc_control.so <- pbmc_control.so[, pbmc_control.so@meta.data[, "highdepth"]
== "keep"]
ncol(pbmc_control.so)
```

```
## [1] 10201
```

```
pbmc_depleted.meta <- pbmc_depleted.so@meta.data
#we are going to add two columns: One column for cells with high sequencing d
epth and another column quantifying the complexity of each cell


pbmc_depleted.meta <- pbmc_depleted.meta %>% mutate(highdepth = ifelse(test =
pbmc_depleted.meta$nCount_RNA <= quantile(pbmc_depleted.meta$nCount_RNA, prob
s = 0.99), yes = 'keep', no = 'discard'), complexity = log10(pbmc_depleted.me
ta$nFeature_RNA)/log10(pbmc_depleted.meta$nCount_RNA))


#add new metadata to Seurat object
pbmc_depleted.so@meta.data <- pbmc_depleted.meta


#for now let's remove the cells with high sequencing depth
table(pbmc_depleted.so@meta.data$highdepth)
```

```
##
## discard    keep
##     105   10360
```

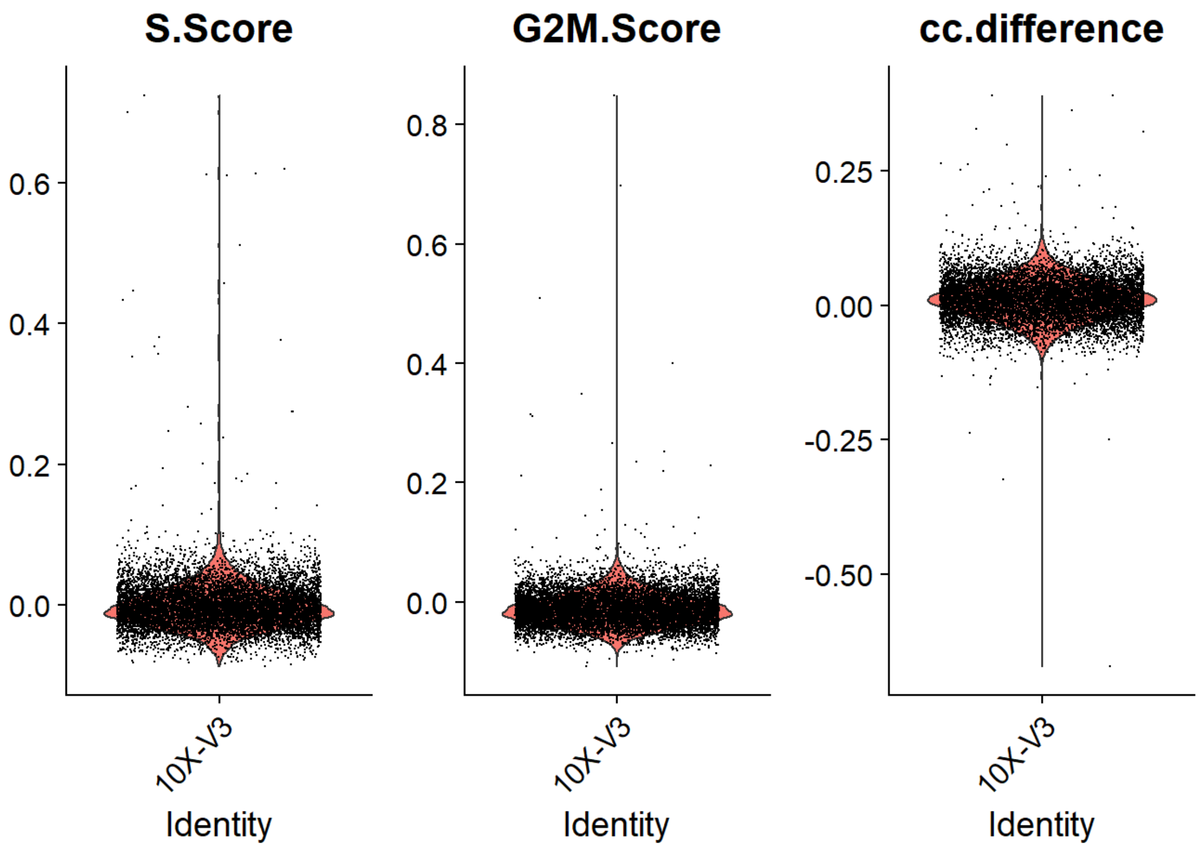
```
pbmc_depleted.so <- pbmc_depleted.so[, pbmc_depleted.so@meta.data[, "highdept
h"] == "keep"]
ncol(pbmc_depleted.so)
```

```
## [1] 10360
```

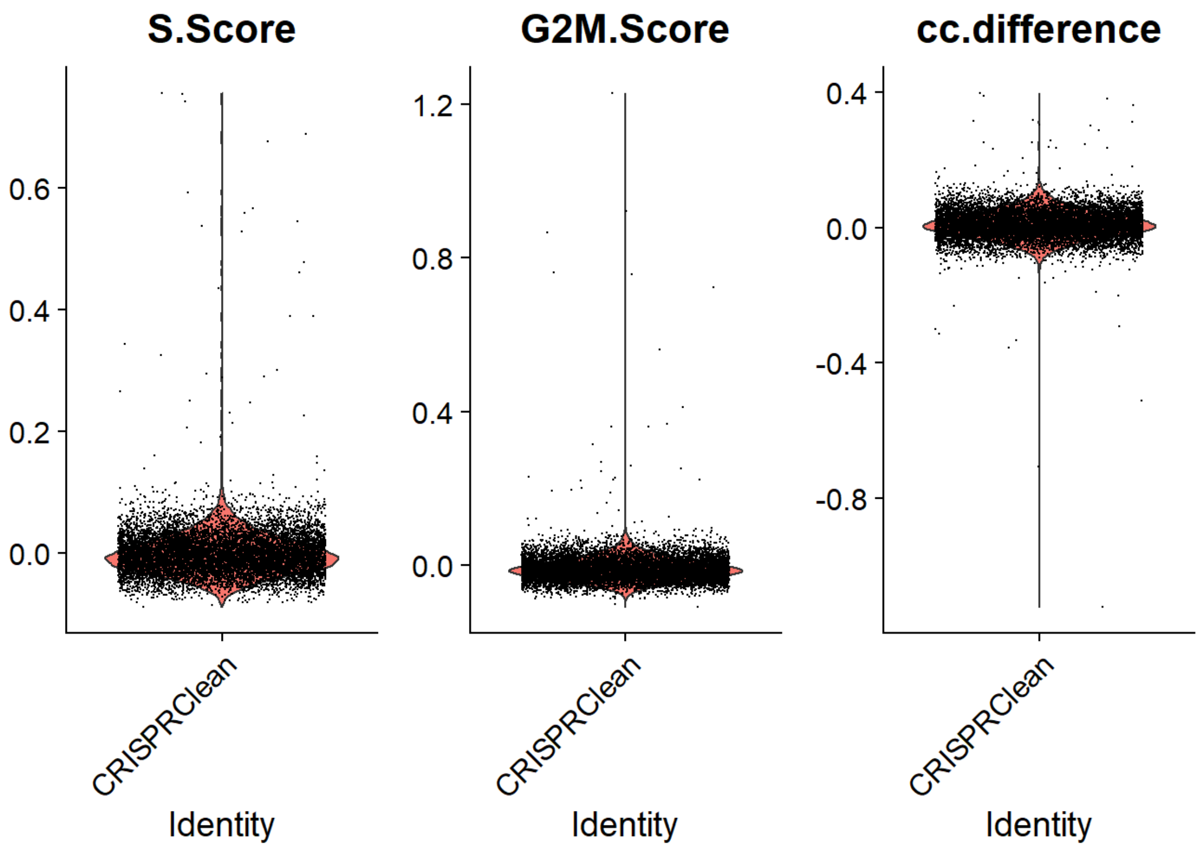
Checking cell cycle scoring of cells

```
#change the default assay to RNA
DefaultAssay(pbmc_control.so) = 'RNA'
DefaultAssay(pbmc_depleted.so) = 'RNA'
```

```
#we quantified cells is S and G2 phase earlier, so we will look to see the pr
oportion of cycling cells

pbmc_control.so$cc.difference <- pbmc_control.so$S.Score - pbmc_control.so$G2
M.Score

VlnPlot(pbmc_control.so, features = c('S.Score','G2M.Score','cc.difference'),
group.by = 'orig.ident')
```



```
pbmc_depleted.so$cc.difference <- pbmc_depleted.so$S.Score - pbmc_depleted.so
$G2M.Score

VlnPlot(pbmc_depleted.so, features = c('S.Score','G2M.Score','cc.difference')
, group.by = 'orig.ident')
```



We can see that there are very few cells in the cycling phase. As a result, we will can choose to regress out cell cycle related influence.

2nd round of clustering

we will proceed with clustering as we did before, with a few slight modifications In the vars.to.regress, we are going to regress out percent mito as well as percent ribosomal. We are also regressing out cell cycle influence which is dependent upon the experiment at hand. If we wanted to regress out G1 phase cells from cycling cells, we would use 'cc.difference' to regress out. However, in this example, we are regressing out all cell cycle influence.

```
library(ggplot2)

#SCTransform and regress out percent mito and cell cycle score

pbmc_control.so <- SCTransform(pbmc_control.so, variable.features.n = NULL, v
ariable.features.rv.th = 1.3, vars.to.regress = c('percent.mt','S.Score','G2M
.Score','percent.rb'))
```

```
## Calculating cell attributes from input UMI matrix: log_umi

## Variance stabilizing transformation of count matrix of size 19388 by 10201

## Model formula is y ~ log_umi

## Get Negative Binomial regression parameters per gene
## Using 2000 genes, 5000 cells
##
  |
  |                                                                      |
0%
  |
  |==================                                                    |  2
5%
  |
  |===================================                                   |  5
0%
  |
  |====================================================                  |  7
5%
  |
  |======================================================================| 10
0%
## There are 1 estimated thetas smaller than 1e-07 - will be set to 1e-07
## Found 91 outliers - those will be ignored in fitting/regularization step
## Second step: Get residuals using fitted parameters for 19388 genes
##
  |
  |                                                                      |
0%
  |
  |==                                                                    |
3%
  |
  |====                                                                  |
5%
  |
  |=====                                                                 |
8%
  |
  |=======                                                               |  1
0%
  |
  |=========                                                             |  1
3%
```

```
  |
  |===========                                                            |  1
5%
  |
  |=============                                                          |  1
8%
  |
  |==============                                                         |  2
1%
  |
  |================                                                       |  2
3%
  |
  |==================                                                     |  2
6%
  |
  |====================                                                   |  2
8%
  |
  |======================                                                 |  3
1%
  |
  |=======================                                                |  3
3%
  |
  |=========================                                              |  3
6%
  |
  |===========================                                            |  3
8%
  |
  |=============================                                          |  4
1%
  |
  |===============================                                        |  4
4%
  |
  |================================                                       |  4
6%
```

```
  |
  |==================================                                    |  4
9%
  |
  |====================================                                  |  5
1%
  |
  |======================================                                |  5
4%
  |
  |=======================================                               |  5
6%
  |
  |=========================================                             |  5
9%
  |
  |===========================================                           |  6
2%
  |
  |=============================================                         |  6
4%
  |
  |===============================================                       |  6
7%
  |
  |================================================                      |  6
9%
  |
  |==================================================                    |  7
2%
  |
  |====================================================                  |  7
4%
  |
  |======================================================                |  7
7%
  |
  |========================================================              |  7
9%
```

```
  |
  |=========================================================             |  8
2%
  |
  |===========================================================           |  8
5%
  |
  |=============================================================         |  8
7%
  |
  |===============================================================       |  9
0%
  |
  |=================================================================     |  9
2%
  |
  |==================================================================    |  9
5%
  |
  |====================================================================  |  9
7%
  |
  |======================================================================| 10
0%
## Computing corrected count matrix for 19388 genes
##
  |
  |                                                                      |
0%
  |
  |==                                                                    |
3%
  |
  |====                                                                  |
5%
  |
  |=====                                                                 |
8%
  |
  |=======                                                               |  1
0%

  |
  |=========                                                             |  1
3%

  |
  |===========                                                           |  1
5%

  |
  |=============                                                         |  1
8%

  |
  |==============                                                        |  2
1%

  |
  |================                                                      |  2
3%

  |
  |==================                                                    |  2
6%

  |
  |====================                                                  |  2
8%

  |
  |======================                                                |  3
1%

  |
  |=======================                                               |  3
3%

  |
  |=========================                                             |  3
6%

  |
  |===========================                                           |  3
8%

  |
  |=============================                                         |  4
1%

  |
  |===============================                                       |  4
4%

  |

  |================================                                      |  4
6%

  |

  |==================================                                    |  4
9%

  |

  |====================================                                  |  5
1%

  |

  |======================================                                |  5
4%

  |

  |=======================================                               |  5
6%

  |

  |=========================================                             |  5
9%

  |

  |===========================================                           |  6
2%

  |

  |=============================================                         |  6
4%

  |

  |===============================================                       |  6
7%

  |

  |================================================                      |  6
9%

  |

  |==================================================                    |  7
2%

  |

  |====================================================                  |  7
4%

  |
  |======================================================                |  7
7%
  |
  |========================================================              |  7
9%
  |
  |=========================================================             |  8
2%
  |
  |===========================================================           |  8
5%
  |
  |=============================================================         |  8
7%
  |
  |===============================================================       |  9
0%
  |
  |=================================================================     |  9
2%
  |
  |==================================================================    |  9
5%
  |
  |====================================================================  |  9
7%
  |
  |======================================================================| 10
0%
## Calculating gene attributes
## Wall clock passed: Time difference of 3.783353 mins
## Determine variable features
## Place corrected count matrix in counts slot
## Regressing out percent.mt, S.Score, G2M.Score, percent.rb
## Centering data matrix
## Set default assay to SCT
```

```
#PCA
pbmc_control.so <- RunPCA(pbmc_control.so)
```

```
## PC_ 1
## Positive:  LYZ, S100A9, S100A8, CTSS, VCAN, FCN1, CST3, NEAT1, MNDA, HLA-D
RA
##     AIF1, IFI30, S100A12, FOS, FTL, TYROBP, PSAP, LST1, S100A6, CD74
##     CD14, CYBB, FGL2, S100A4, S100A11, COTL1, MS4A6A, TKT, LGALS1, HLA-DPA
1
## Negative:  CCL5, NKG7, GNLY, PPBP, PF4, GNG11, GP1BB, CAVIN2, TUBB1, HIST1
H2AC
##     NRGN, GP9, TSC22D1, ACRBP, PRKAR2B, PTCRA, GZMA, CMTM5, GZMB, ODC1
##     TMEM40, MAP3K7CL, MMD, C2orf88, RGS18, CLU, TUBA4A, KLRD1, CLDN5, FGFB
P2
## PC_ 2
## Positive:  NKG7, GNLY, GZMB, GZMA, FGFBP2, KLRD1, PRF1, CST7, HOPX, CLIC3
##     KLRB1, KLRF1, SPON2, CCL5, CCL4, CTSW, TRDC, GZMH, IL2RB, KLRK1
##     CMC1, CD160, GZMM, CD7, FCGR3A, XCL2, MATK, CD247, AKR1C3, IFITM2
## Negative:  NRGN, PPBP, PF4, GP1BB, TUBB1, GNG11, CAVIN2, HLA-DRA, RGS18, H
IST1H2AC
##     GP9, PRKAR2B, ACRBP, PTCRA, CMTM5, TSC22D1, F13A1, ODC1, TMEM40, CD74
##     MAP3K7CL, OAZ1, C2orf88, FTH1, MMD, CLU, RUFY1, LYZ, LIMS1, TAGLN2
## PC_ 3
## Positive:  HLA-DRA, CD74, IGKC, MS4A1, IGHM, CD79A, HLA-DPB1, HLA-DPA1, HL
A-DQA1, HLA-DQB1
##     TCL1A, IGHD, BANK1, CD79B, HLA-DRB1, RALGPS2, IGLC2, LINC02397, TNFRSF
13C, TCF4
##     BCL11A, FCRL1, LINC00926, SPIB, FCER2, VPREB3, JCHAIN, CD22, HLA-DRB5,
IGLC3
## Negative:  S100A8, S100A9, VCAN, LYZ, S100A12, IL7R, FCN1, MNDA, FOS, CTSS
##     NEAT1, S100A4, S100A6, CD14, NRGN, CCL5, PF4, PPBP, GNG11, IL32
##     GP1BB, THBS1, CAVIN2, TUBB1, RGS18, ANXA1, CYP1B1, CD3D, S100A11, RGS1
0
## PC_ 4
## Positive:  GNLY, NKG7, GZMB, LYZ, S100A9, FGFBP2, S100A8, KLRD1, CCL5, PRF
1
##     CLIC3, HOPX, GZMA, VCAN, TYROBP, KLRF1, SPON2, CST7, HLA-DRA, TRDC
##     FCN1, CTSW, CCL4, FCER1G, S100A12, CTSS, CD74, KLRB1, CD160, MNDA
## Negative:  IL32, IL7R, GZMK, LTB, TRAC, CD3D, TRBC2, CD3G, BCL11B, CD52
##     CD3E, ITGB1, TRBC1, RTKN2, CD2, LIME1, HBB, SYNE2, MAF, HBA1
##     CD27, GPR183, CDKN1C, HBA2, AQP3, INPP4B, CD84, BCL2, TIGIT, CDC14A
## PC_ 5
## Positive:  FCGR3A, LST1, AIF1, CDKN1C, CST3, HLA-DPA1, MS4A7, SAT1, IFITM3
, FCER1G
##     IFI30, FTL, COTL1, TCF7L2, SMIM25, C1QA, HLA-DPB1, FTH1, HLA-DRA, CSF1
R
##     RHOC, LYPD2, CALHM6, HLA-DRB1, IFITM2, PSAP, CEBPB, TYROBP, SERPINA1,
HES4
## Negative:  S100A8, VCAN, S100A12, S100A9, FOS, IGHM, IGKC, MS4A1, CD79A, L
YZ
##     MNDA, IGHD, TCL1A, CYP1B1, CD14, THBS1, BANK1, NCF1, RALGPS2, CSF3R
##     PLBD1, MEGF9, CD36, SLC2A3, IGLC2, LINC02397, FCRL1, RGS2, LINC00926,
TNFRSF13C
```

```
#we can quantitately measure the optimal number of PCAs to use for downstream
analysis
pct <- pbmc_control.so[['pca']]@stdev / sum(pbmc_control.so[['pca']]@stdev) *
100
cumu <- cumsum(pct)
sort(which((pct[1:length(pct) - 1] - pct[2:length(pct)]) > 0.1), decreasing =
T)[1] + 1
```

```
## [1] 15
```

```
#Find k-nearest neighbors using the first 30 dimensions. For now, we will use
30 dimensions which is the default setting and generally provides good cluste
ring results. However, we can always reanalyze clustering results with the ca
lculated number
pbmc_control.so <- FindNeighbors(pbmc_control.so, dims = 1:30)
```

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

```
#generate UMAP coordinates
pbmc_control.so <- RunUMAP(pbmc_control.so, dims = 1:30)
```

```
## 16:20:08 UMAP embedding parameters a = 0.9922 b = 1.112
## 16:20:08 Read 10201 rows and found 30 numeric columns
## 16:20:08 Using Annoy for neighbor search, n_neighbors = 30
## 16:20:08 Building Annoy index with metric = cosine, n_trees = 50
## 0%   10   20   30   40   50   60   70   80   90   100%
## [----|----|----|----|----|----|----|----|----|----|
## **************************************************|
## 16:20:10 Writing NN index file to temp file C:\Users\dante\AppData\Local\T
emp\Rtmp2Xrtmf\file37e061ac4943
## 16:20:10 Searching Annoy index using 1 thread, search_k = 3000
## 16:20:14 Annoy recall = 100%

## 16:20:21 Commencing smooth kNN distance calibration using 1 thread

## 16:20:23 Initializing from normalized Laplacian + noise

## 16:20:24 Commencing optimization for 200 epochs, with 425180 positive edge
s

## 16:20:37 Optimization finished
```

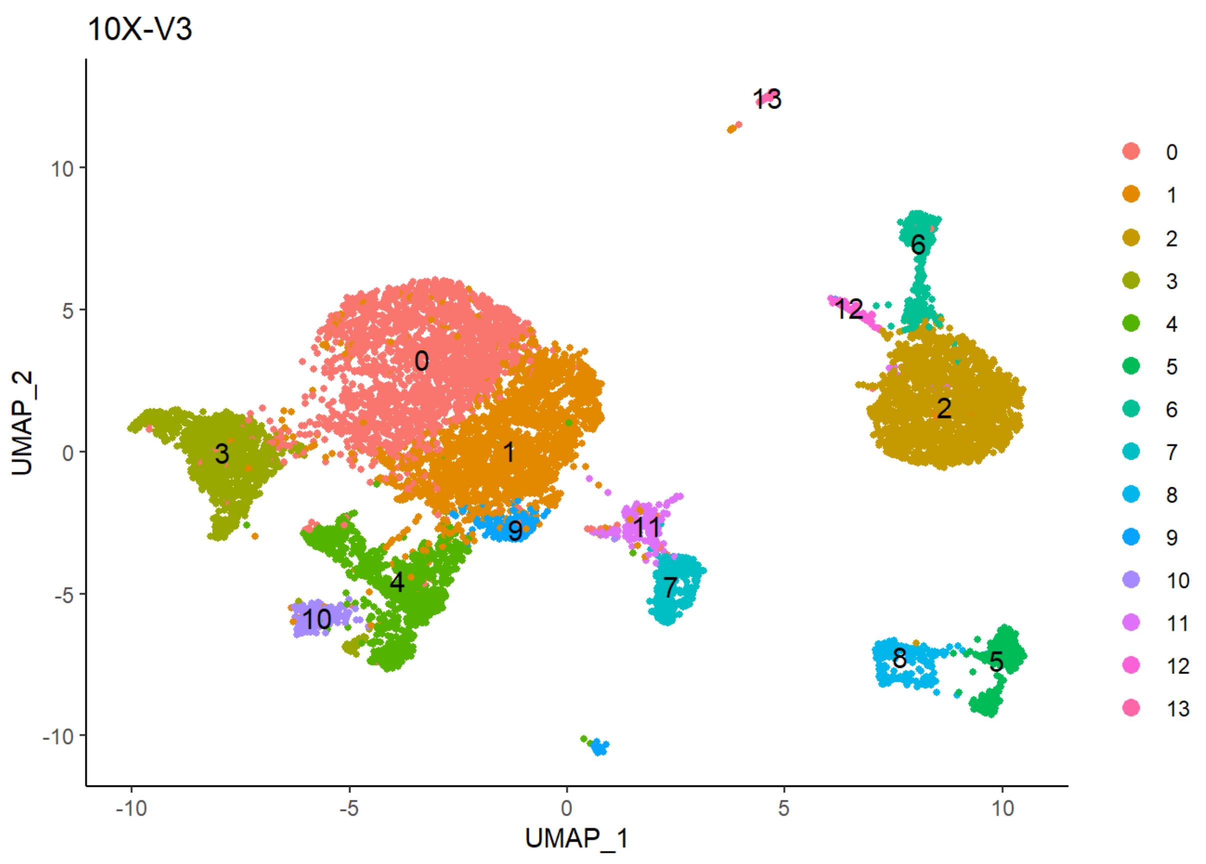
```
#iteratively find clusters at a range of resolutions to be stored in metadata

pbmc_control.so <- FindClusters(pbmc_control.so, resolution = seq(0.05,1,0.05
), algorithm = 2)
```

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck

##

## Number of nodes: 10201

## Number of edges: 353697

##

## Running Louvain algorithm with multilevel refinement...

## Maximum modularity in 10 random starts: 0.9803

## Number of communities: 6

## Elapsed time: 2 seconds

## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck

##

## Number of nodes: 10201

## Number of edges: 353697

##

## Running Louvain algorithm with multilevel refinement...

## Maximum modularity in 10 random starts: 0.9668

## Number of communities: 8

## Elapsed time: 2 seconds

## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck

##

## Number of nodes: 10201

## Number of edges: 353697

##

## Running Louvain algorithm with multilevel refinement...

## Maximum modularity in 10 random starts: 0.9548

## Number of communities: 10
## Elapsed time: 2 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10201
## Number of edges: 353697
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.9438
## Number of communities: 11
## Elapsed time: 2 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10201
## Number of edges: 353697
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.9328
## Number of communities: 13
## Elapsed time: 2 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10201
## Number of edges: 353697
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.9260
## Number of communities: 14
## Elapsed time: 2 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10201
## Number of edges: 353697
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.9192
## Number of communities: 14
## Elapsed time: 2 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10201
## Number of edges: 353697
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.9123
## Number of communities: 14
## Elapsed time: 2 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10201
## Number of edges: 353697
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.9057
## Number of communities: 15
## Elapsed time: 1 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10201
## Number of edges: 353697
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.8993
## Number of communities: 16
## Elapsed time: 2 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10201
## Number of edges: 353697
```

```
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.8942
## Number of communities: 16
## Elapsed time: 2 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10201
## Number of edges: 353697
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.8883
## Number of communities: 16
## Elapsed time: 2 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10201
## Number of edges: 353697
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.8832
## Number of communities: 17
## Elapsed time: 2 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10201
## Number of edges: 353697
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.8781
## Number of communities: 17
## Elapsed time: 2 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
```

```
## Number of nodes: 10201
## Number of edges: 353697
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.8735
## Number of communities: 19
## Elapsed time: 2 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10201
## Number of edges: 353697
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.8691
## Number of communities: 21
## Elapsed time: 2 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10201
## Number of edges: 353697
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.8651
## Number of communities: 22
## Elapsed time: 2 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10201
## Number of edges: 353697
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.8612
## Number of communities: 22
## Elapsed time: 2 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10201
## Number of edges: 353697
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.8582
## Number of communities: 23
## Elapsed time: 2 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10201
## Number of edges: 353697
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.8541
## Number of communities: 22
## Elapsed time: 2 seconds
```

```
#normalize raw counts data
pbmc_control.so <- NormalizeData(pbmc_control.so, assay = 'RNA')
#visualize UMAP and pick a resolution to use from column of dataframe
DimPlot(pbmc_control.so, label = T, pt.size = 1, group.by = 'SCT_snn_res.0.4'
) + theme_classic() + ggtitle('10X-V3')
```



```
#total number of clusters
length(table(pbmc_control.so@meta.data$SCT_snn_res.0.4))
```

```
## [1] 14
```

```
#SCTransform and regress out percent mito and cell cycle score
pbmc_depleted.so <- SCTransform(pbmc_depleted.so, variable.features.n = NULL,
variable.features.rv.th = 1.3, vars.to.regress = c('percent.mt','S.Score','G2
M.Score','percent.rb'))
```

```
## Calculating cell attributes from input UMI matrix: log_umi

## Variance stabilizing transformation of count matrix of size 19494 by 10360

## Model formula is y ~ log_umi

## Get Negative Binomial regression parameters per gene

## Using 2000 genes, 5000 cells

##

  |

  |                                                                      |
0%

  |
  |==================                                                        |  2
5%
  |
  |===================================                                       |  5
0%
  |
  |=====================================================                     |  7
5%
  |
  |==========================================================================| 10
0%
## Found 86 outliers - those will be ignored in fitting/regularization step
## Second step: Get residuals using fitted parameters for 19494 genes
##
  |
  |                                                                          |
0%
  |
  |==                                                                        |
3%
  |
  |====                                                                      |
5%
  |
  |=====                                                                     |
8%
  |
  |=======                                                                   |  1
0%
  |
  |=========                                                                 |  1
3%
  |
  |===========                                                               |  1
5%
  |
  |=============                                                             |  1
8%
  |
  |==============                                                        |  2
1%

  |
  |================                                                      |  2
3%

  |
  |==================                                                    |  2
6%

  |
  |====================                                                  |  2
8%

  |
  |======================                                                |  3
1%

  |
  |=======================                                               |  3
3%

  |
  |=========================                                             |  3
6%

  |
  |===========================                                           |  3
8%

  |
  |=============================                                         |  4
1%

  |
  |===============================                                       |  4
4%

  |
  |================================                                      |  4
6%

  |
  |==================================                                    |  4
9%

  |
  |====================================                                  |  5
1%

  |
  |======================================                                    |  5
4%
  |
  |=======================================                                   |  5
6%
  |
  |=========================================                                 |  5
9%
  |
  |===========================================                               |  6
2%
  |
  |=============================================                             |  6
4%
  |
  |===============================================                           |  6
7%
  |
  |================================================                          |  6
9%
  |
  |==================================================                        |  7
2%
  |
  |====================================================                      |  7
4%
  |
  |======================================================                    |  7
7%
  |
  |========================================================                  |  7
9%
  |
  |=========================================================                 |  8
2%
  |
  |===========================================================               |  8
5%
  |
  |=============================================================         |  8
7%

  |

  |===============================================================       |  9
0%

  |

  |=================================================================     |  9
2%

  |

  |==================================================================    |  9
5%

  |

  |====================================================================  |  9
7%

  |

  |======================================================================| 10
0%

## Computing corrected count matrix for 19494 genes

##

  |

  |                                                                      |
0%

  |

  |==                                                                    |
3%

  |

  |====                                                                  |
5%

  |

  |=====                                                                 |
8%

  |

  |=======                                                               |  1
0%

  |

  |=========                                                             |  1
3%

  |
  |===========                                                           |  1
5%

  |
  |=============                                                         |  1
8%

  |
  |==============                                                        |  2
1%

  |
  |================                                                      |  2
3%

  |
  |==================                                                    |  2
6%

  |
  |====================                                                  |  2
8%

  |
  |======================                                                |  3
1%

  |
  |=======================                                               |  3
3%

  |
  |=========================                                             |  3
6%

  |
  |===========================                                           |  3
8%

  |
  |=============================                                         |  4
1%

  |
  |===============================                                       |  4
4%

  |
  |================================                                      |  4
6%

  |
  |==================================                                        |  4
9%
  |
  |====================================                                      |  5
1%
  |
  |======================================                                    |  5
4%
  |
  |=======================================                                   |  5
6%
  |
  |=========================================                                 |  5
9%
  |
  |===========================================                               |  6
2%
  |
  |=============================================                             |  6
4%
  |
  |===============================================                           |  6
7%
  |
  |================================================                          |  6
9%
  |
  |==================================================                        |  7
2%
  |
  |====================================================                      |  7
4%
  |
  |======================================================                    |  7
7%
  |
  |========================================================                  |  7
9%
  |
  |==========================================================            |  8
2%
  |
  |============================================================          |  8
5%
  |
  |==============================================================        |  8
7%
  |
  |================================================================      |  9
0%
  |
  |==================================================================    |  9
2%
  |
  |===================================================================   |  9
5%
  |
  |=====================================================================  |  9
7%
  |
  |=======================================================================| 10
0%
## Calculating gene attributes
## Wall clock passed: Time difference of 3.101769 mins
## Determine variable features
## Place corrected count matrix in counts slot
## Regressing out percent.mt, S.Score, G2M.Score, percent.rb
## Centering data matrix
## Set default assay to SCT
```

```
#PCA
pbmc_depleted.so <- RunPCA(pbmc_depleted.so)
```

```
## PC_ 1
## Positive:  NKG7, GNLY, GZMA, CCL5, GZMB, FGFBP2, CST7, PRF1, KLRB1, HOPX
##     KLRD1, CLIC3, KLRF1, CTSW, SPON2, PF4, TRDC, CCL4, CAVIN2, GNG11
##     GP1BB, KLRK1, GZMM, TUBB1, CMC1, GZMH, CD7, GZMK, HIST1H2AC, ARL4C
## Negative:  LYZ, S100A9, VCAN, HLA-DRA, MNDA, CD74, AIF1, IFI30, S100A12, C
D14
```

```
##     CTSS, CYBB, HLA-DPA1, HLA-DRB1, FCN1, S100A8, COTL1, FGL2, HLA-DPB1, M
S4A6A
##     TKT, GRN, MPEG1, FOS, FTL, TYMP, NCF2, VIM, CST3, CEBPD
## PC_ 2
## Positive:  NRGN, GP1BB, PF4, GNG11, TUBB1, CAVIN2, HIST1H2AC, GP9, RGS18,
ACRBP
##     CMTM5, PTCRA, TSC22D1, PRKAR2B, PPBP, TMEM40, CLU, C2orf88, MAP3K7CL,
MMD
##     ODC1, F13A1, CLDN5, SPARC, LGALSL, MTURN, TREML1, RUFY1, AC147651.1, L
IMS1
## Negative:  GNLY, NKG7, GZMB, GZMA, FGFBP2, PRF1, CST7, HOPX, KLRD1, KLRB1
##     CLIC3, KLRF1, SPON2, CTSW, TRDC, CCL4, GZMH, CMC1, KLRK1, FCGR3A
##     GZMM, CD160, IL2RB, CD7, MATK, IFITM2, AKR1C3, HCST, ID2, XCL2
## PC_ 3
## Positive:  S100A9, LYZ, VCAN, S100A12, MNDA, IL7R, S100A8, CD14, AIF1, VIM
##     FOS, S100A6, FCN1, CTSS, CEBPD, TKT, FYB1, FTH1, RGS2, IFI30
##     ANXA1, S100A11, FGL2, THBS1, MS4A6A, PLBD1, CYP1B1, GIMAP7, CD3E, NCF2
## Negative:  HLA-DRA, CD74, MS4A1, CD79A, IGHM, IGKC, HLA-DPB1, HLA-DPA1, HL
A-DQA1, HLA-DQB1
##     IGHD, BANK1, TCL1A, CD79B, FCRL1, LINC02397, RALGPS2, HLA-DRB1, TNFRSF
13C, IGLC2
##     FCER2, LINC00926, VPREB3, BCL11A, AFF3, CD22, IGLC3, CD37, TCF4, PAX5
## PC_ 4
## Positive:  IL7R, LTB, TRAC, CD52, CD3D, BCL11B, CD3E, CD3G, IL32, TRBC1
##     MALAT1, GZMK, ITGB1, INPP4B, RCAN3, BCL2, MAL, CD27, LIME1, CRIP1
##     TRAT1, CD2, GPR183, AQP3, RTKN2, ETS1, LEF1, BIRC3, PBXIP1, FOXP1
## Negative:  GNLY, NKG7, GZMB, LYZ, FGFBP2, S100A9, GZMA, PRF1, CCL5, CLIC3
##     HOPX, KLRD1, CST7, SPON2, VCAN, KLRF1, HLA-DRA, FCGR3A, CTSW, TRDC
##     IFI30, MNDA, CCL4, AIF1, CD74, NRGN, EFHD2, CD160, S100A12, GZMH
## PC_ 5
## Positive:  VCAN, S100A12, S100A9, MS4A1, CD79A, S100A8, IGHM, IGHD, LYZ, F
CRL1
##     MNDA, BANK1, TCL1A, FOS, CD14, FCER2, LINC02397, THBS1, RALGPS2, IGLC2
##     CYP1B1, CD79B, LINC00926, VPREB3, TNFRSF13C, CD22, PLBD1, IGLC3, RGS2,
NCF1
## Negative:  FCGR3A, CDKN1C, HLA-DPA1, AIF1, PLD4, MS4A7, IL3RA, IFI30, SERP
INF1, GZMB
##     LILRA4, PPP1R14B, UGCG, NPC2, COTL1, CCDC50, ITM2C, IRF7, TPM2, LRRC26
##      HLA-DRA, SCT, CLEC4C, CST3, DNASE1L3, HLA-DPB1, TCF7L2, IFITM3, SMIM25
, IRF8
```

```
#we can quantitately measure the optimal number of PCAs to use for downstream
analysis

pct <- pbmc_depleted.so[['pca']]@stdev / sum(pbmc_depleted.so[['pca']]@stdev)
* 100

cumu <- cumsum(pct)

sort(which((pct[1:length(pct) - 1] - pct[2:length(pct)]) > 0.1), decreasing =
T)[1] + 1
```

```
## [1] 14
```

```
#Find k-nearest neighbors using the first 30 dimensions. For now, we will use
30 dimensions which is the default setting and generally provides good cluste
ring results. However, we can always reanalyze clustering results with the ca
lculated number

pbmc_depleted.so <- FindNeighbors(pbmc_depleted.so, dims = 1:30)
```

```
## Computing nearest neighbor graph

## Computing SNN
```

```
#generate UMAP coordinates

pbmc_depleted.so <- RunUMAP(pbmc_depleted.so, dims = 1:30)
```

```
## 16:26:26 UMAP embedding parameters a = 0.9922 b = 1.112

## 16:26:26 Read 10360 rows and found 30 numeric columns

## 16:26:26 Using Annoy for neighbor search, n_neighbors = 30

## 16:26:26 Building Annoy index with metric = cosine, n_trees = 50

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

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

## **************************************************|

## 16:26:28 Writing NN index file to temp file C:\Users\dante\AppData\Local\T
emp\Rtmp2Xrtmf\file37e07bd42c4f

## 16:26:28 Searching Annoy index using 1 thread, search_k = 3000

## 16:26:33 Annoy recall = 100%

## 16:26:39 Commencing smooth kNN distance calibration using 1 thread

## 16:26:41 Initializing from normalized Laplacian + noise

## 16:26:42 Commencing optimization for 200 epochs, with 434748 positive edge
s

## 16:26:55 Optimization finished
```

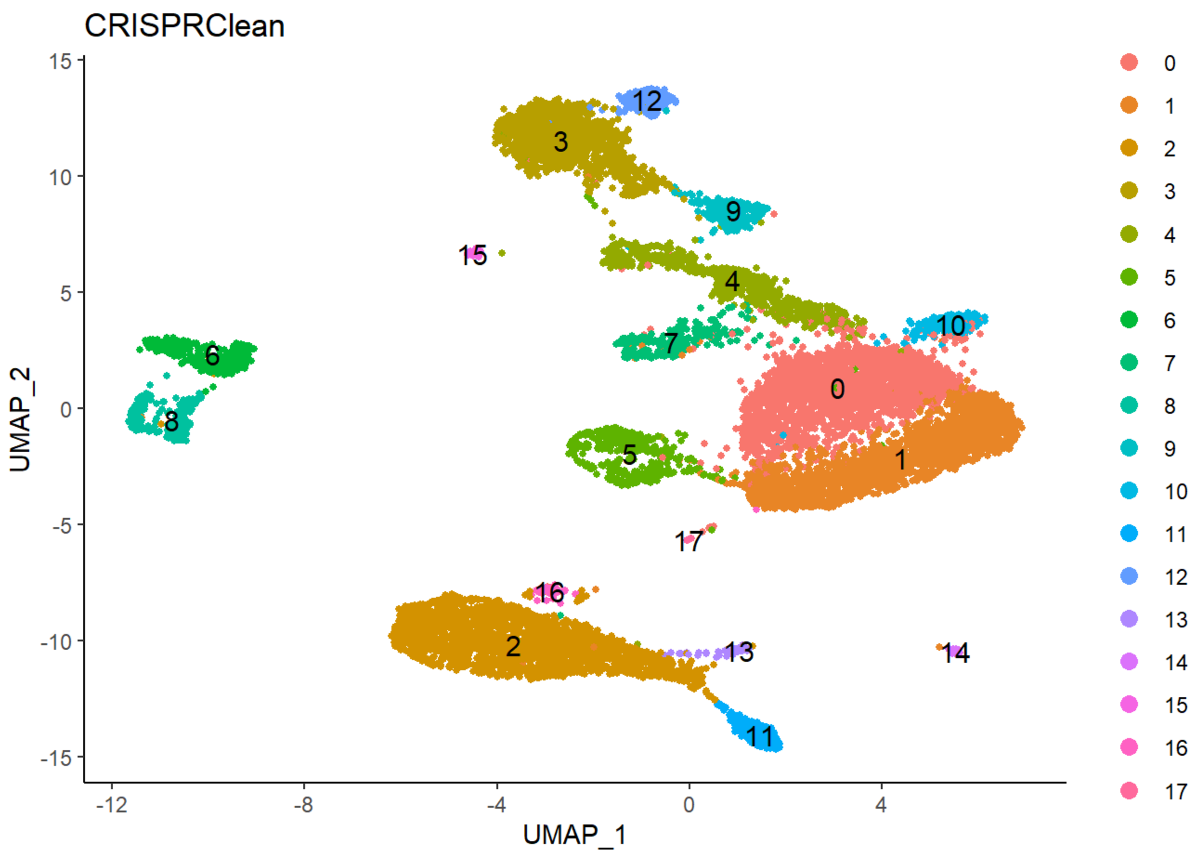
```
#iteratively find clusters at a range of resolutions to be stored in metadata

pbmc_depleted.so <- FindClusters(pbmc_depleted.so, resolution = seq(0.05,1,0.
05), algorithm = 2)
```

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10360
## Number of edges: 363195
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.9810
## Number of communities: 8
## Elapsed time: 2 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10360
## Number of edges: 363195
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.9695
## Number of communities: 10
## Elapsed time: 1 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10360
## Number of edges: 363195
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.9592
## Number of communities: 12
## Elapsed time: 2 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10360
## Number of edges: 363195
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.9492
## Number of communities: 15
## Elapsed time: 2 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10360
## Number of edges: 363195
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.9419
## Number of communities: 17
## Elapsed time: 2 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10360
## Number of edges: 363195
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.9358
## Number of communities: 18
## Elapsed time: 2 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10360
## Number of edges: 363195
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.9297
## Number of communities: 18
## Elapsed time: 2 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10360
## Number of edges: 363195
##
```

```
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.9235
## Number of communities: 18
## Elapsed time: 2 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10360
## Number of edges: 363195
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.9177
## Number of communities: 19
## Elapsed time: 2 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10360
## Number of edges: 363195
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.9127
## Number of communities: 19
## Elapsed time: 2 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10360
## Number of edges: 363195
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.9079
## Number of communities: 19
## Elapsed time: 2 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10360
## Number of edges: 363195
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.9030
## Number of communities: 20
## Elapsed time: 2 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10360
## Number of edges: 363195
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.8984
## Number of communities: 20
## Elapsed time: 2 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10360
## Number of edges: 363195
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.8936
## Number of communities: 21
## Elapsed time: 2 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10360
## Number of edges: 363195
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.8889
## Number of communities: 21
## Elapsed time: 2 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
```

```
##
## Number of nodes: 10360
## Number of edges: 363195
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.8843
## Number of communities: 21
## Elapsed time: 2 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10360
## Number of edges: 363195
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.8797
## Number of communities: 21
## Elapsed time: 2 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10360
## Number of edges: 363195
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.8754
## Number of communities: 22
## Elapsed time: 2 seconds
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10360
## Number of edges: 363195
##
## Running Louvain algorithm with multilevel refinement...
## Maximum modularity in 10 random starts: 0.8716
## Number of communities: 22
## Elapsed time: 2 seconds

## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck

##

## Number of nodes: 10360

## Number of edges: 363195

##

## Running Louvain algorithm with multilevel refinement...

## Maximum modularity in 10 random starts: 0.8677

## Number of communities: 22

## Elapsed time: 2 seconds
```

```
#normalize raw counts data

pbmc_depleted.so <- NormalizeData(pbmc_depleted.so, assay = 'RNA')

#visualize UMAP and pick a resolution to use from column of dataframe

DimPlot(pbmc_depleted.so, label = T, pt.size = 1, group.by = 'SCT_snn_res.0.4
') + theme_classic() + ggtitle('CRISPRClean')
```



```
#total number of clusters
```

```
length(table(pbmc_depleted.so@meta.data$SCT_snn_res.0.4))
```

```
## [1] 18
```

we can see that we obtain 4 more additional clusters after depletion!