scRNAseq - using seurat

Adapted from this dataset: https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE197177 (https://www.ncbi.nlm.nih.gov/geo/query/acc.cgi?acc=GSE197177)

2025-04-18

Input:

```
# Create Seurat Objects for the input samples:
# C1, 2, 3 refers to case 1, 2 and 3 from the reference paper.

C1_YF <- Read10X(data.dir = 'C1_YF')
C1_YF <- CreateSeuratObject(counts =C1_YF, project = "C1_YF", min.cells = 3, min.features = 200)

C1_ZY <- Read10X(data.dir = 'C1_ZY')
C1_ZY <- CreateSeuratObject(counts = C1_ZY, project = "C1_ZY", min.cells = 3, min.features = 20
0)

C2_YF <- Read10X(data.dir = 'C2_YF')
C2_YF <- CreateSeuratObject(counts =C2_YF, project = "C2_YF", min.cells = 3, min.features = 200)

C2_ZC <- Read10X(data.dir = 'C2_ZC')
C2_ZC <- CreateSeuratObject(counts = C2_ZC, project = "C2_ZC", min.cells = 3, min.features = 20
0)

C2_ZY <- Read10X(data.dir = 'C2_ZY')
C2_ZY <- CreateSeuratObject(counts =C2_ZY, project = "C2_ZY", min.cells = 3, min.features = 200)

C3_YF <- Read10X(data.dir = 'C3_YF')
C3_YF <- CreateSeuratObject(counts = C3_YF, project = "C3_YF", min.cells = 3, min.features = 20
0)

C3_ZY <- Read10X(data.dir = 'C3_ZY')
C3_ZY <- CreateSeuratObject(counts =C3_ZY, project = "C3_ZY", min.cells = 3, min.features = 200)

C4_ZY <- Read10X(data.dir = 'C4_ZY')
C4_ZY <- CreateSeuratObject(counts = C4_ZY, project = "C4_ZY", min.cells = 3, min.features = 20
0)
```

QC:

```
# Compute percent.mt for each sample:

C1_YF[["percent.mt"]] <- PercentageFeatureSet(C1_YF, pattern = "^MT-")
C1_ZY[["percent.mt"]] <- PercentageFeatureSet(C1_ZY, pattern = "^MT-")
C2_YF[["percent.mt"]] <- PercentageFeatureSet(C2_YF, pattern = "^MT-")
C2_ZC[["percent.mt"]] <- PercentageFeatureSet(C2_ZC, pattern = "^MT-")
C2_ZY[["percent.mt"]] <- PercentageFeatureSet(C2_ZY, pattern = "^MT-")
C3_YF[["percent.mt"]] <- PercentageFeatureSet(C3_YF, pattern = "^MT-")
C3_ZY[["percent.mt"]] <- PercentageFeatureSet(C3_ZY, pattern = "^MT-")
C4_ZY[["percent.mt"]] <- PercentageFeatureSet(C4_ZY, pattern = "^MT-")
```

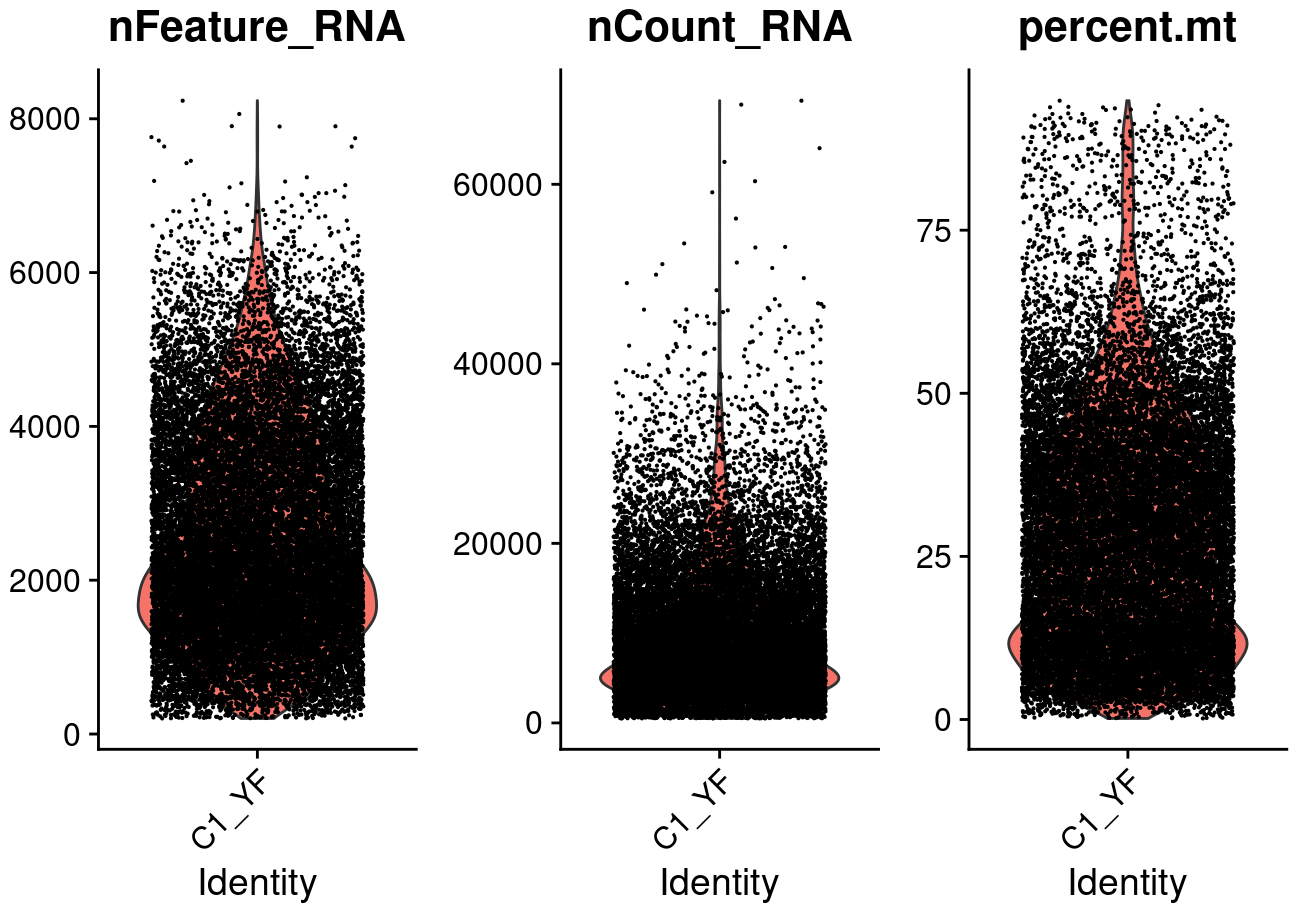
Visualize QC metrics as violin plots for each sample:

```
VlnPlot(C1_YF, 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.
```

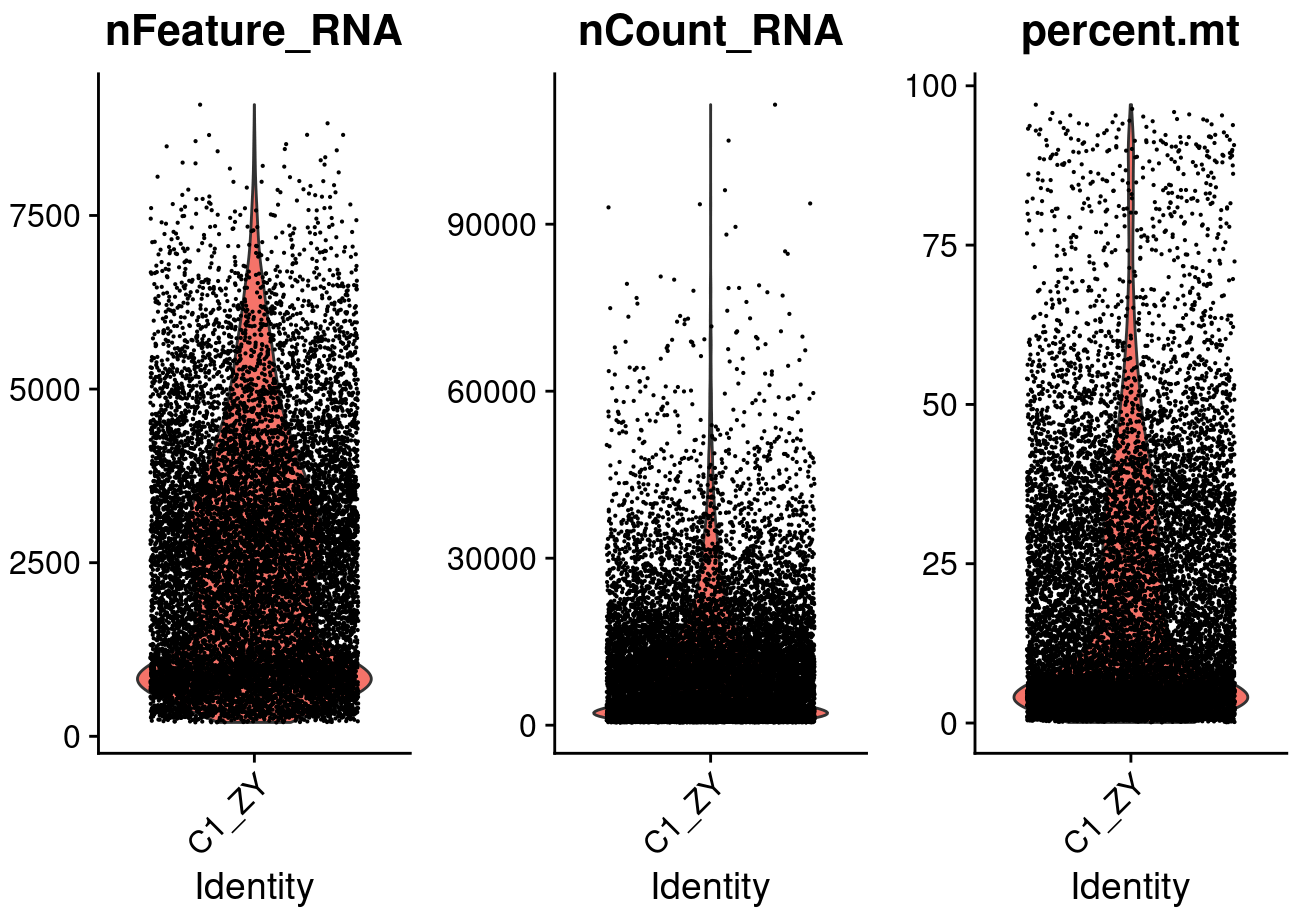
```
## Warning: The `slot` argument of `FetchData()` is deprecated as of SeuratObject 5.0.0.
## ℹ Please use the `layer` argument instead.
## ℹ The deprecated feature was likely used in the Seurat package.
##   Please report the issue at <https://github.com/satijalab/seurat/issues>.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
## Warning: `PackageCheck()` was deprecated in SeuratObject 5.0.0.
## ℹ Please use `rlang::check_installed()` instead.
## ℹ The deprecated feature was likely used in the Seurat package.
##   Please report the issue at <https://github.com/satijalab/seurat/issues>.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```



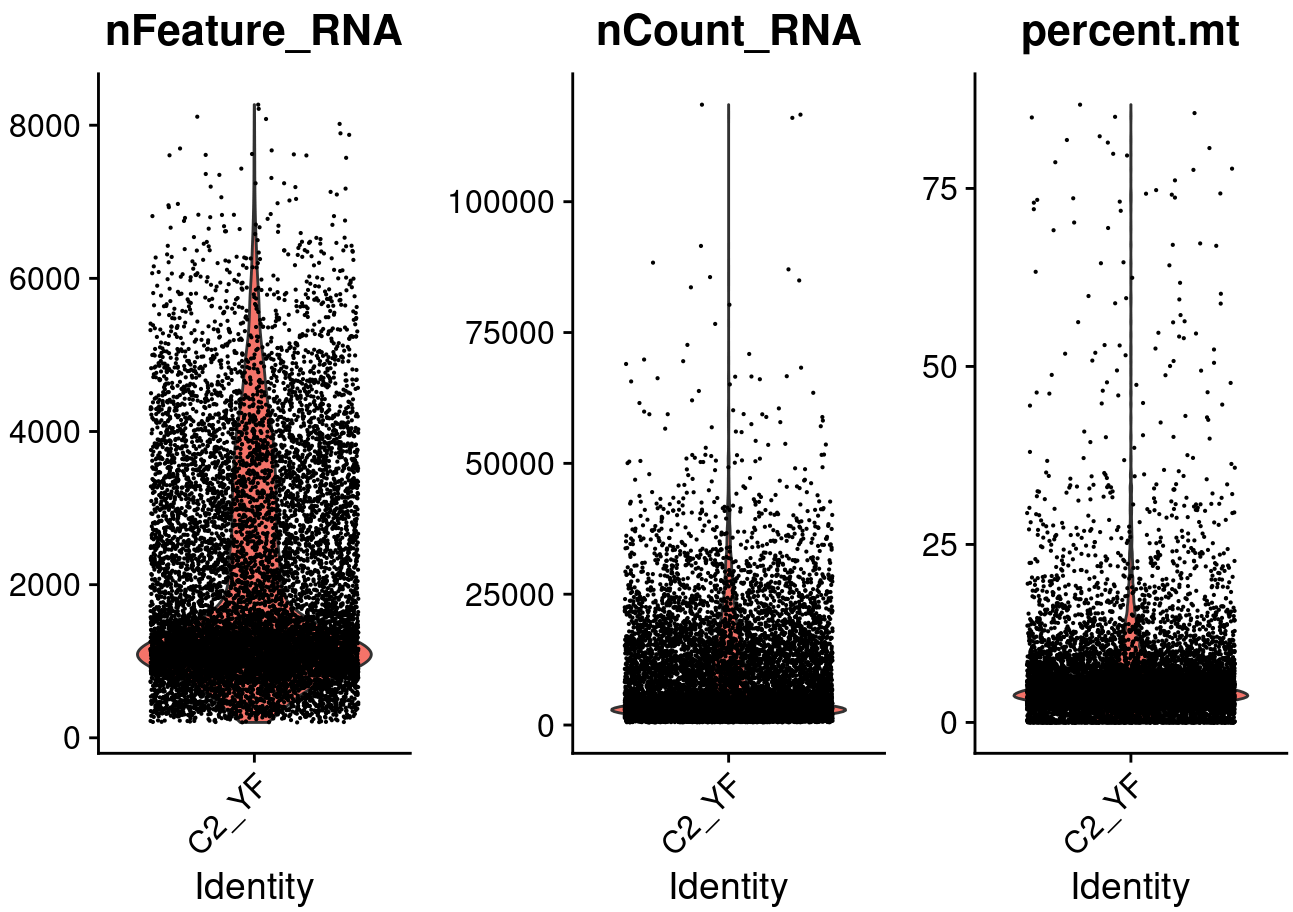

```
VlnPlot(C1_ZY, features = c("nFeature_RNA", "nCount_RNA", "percent.mt"), ncol = 3)
```

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



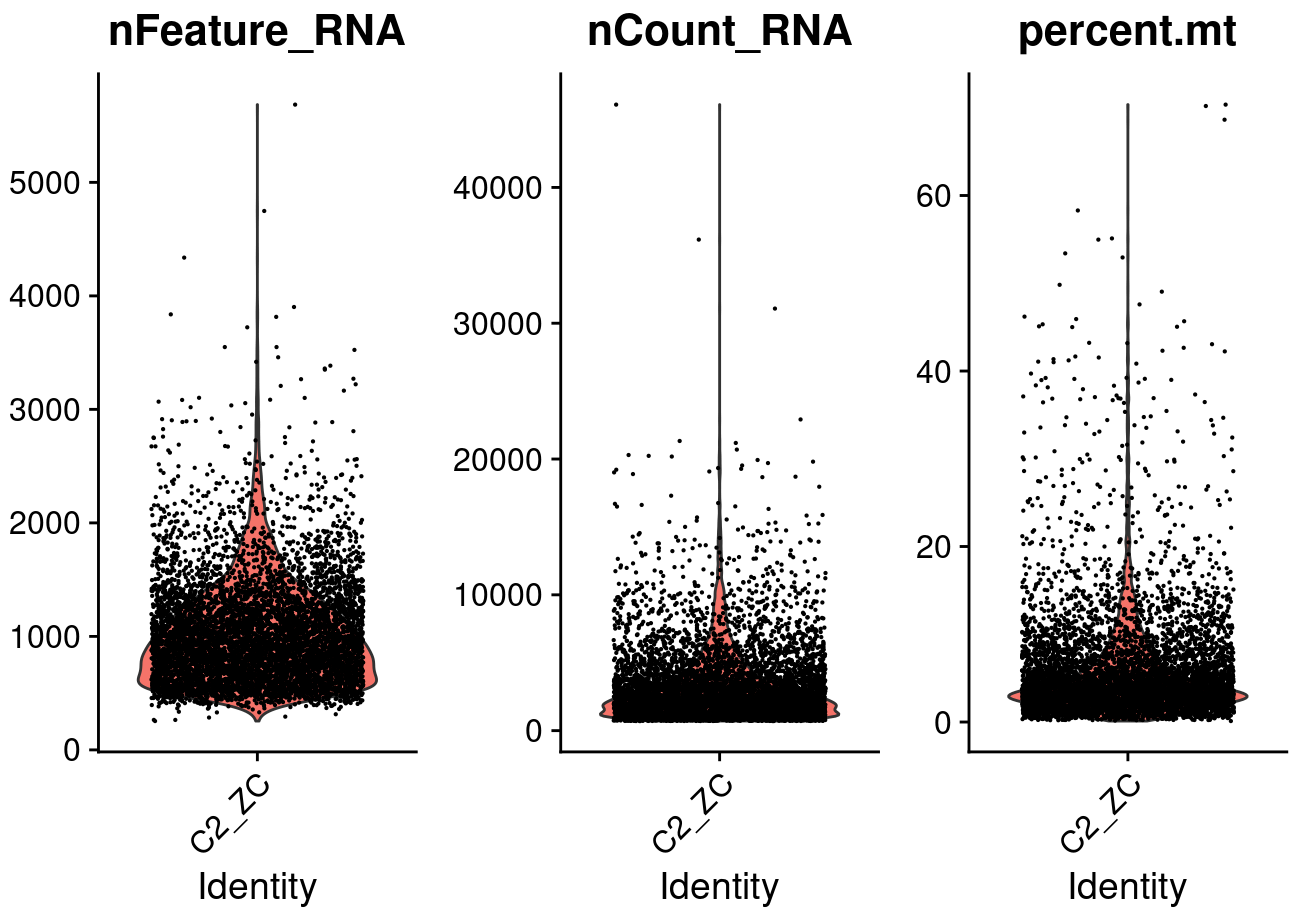

```
VlnPlot(C2_YF, features = c("nFeature_RNA", "nCount_RNA", "percent.mt"), ncol = 3)
```

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



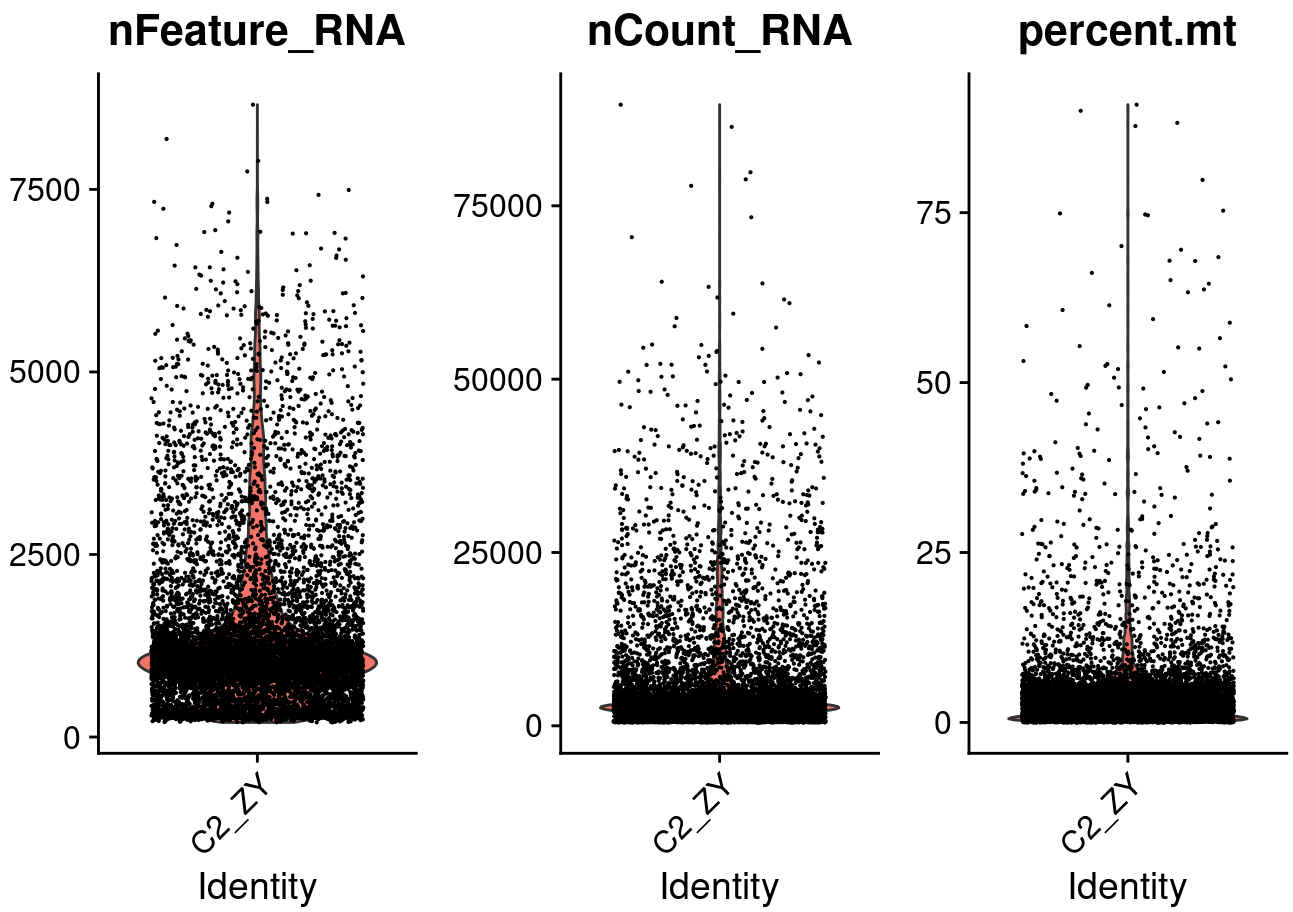

```
VlnPlot(C2_ZC, features = c("nFeature_RNA", "nCount_RNA", "percent.mt"), ncol = 3)
```

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



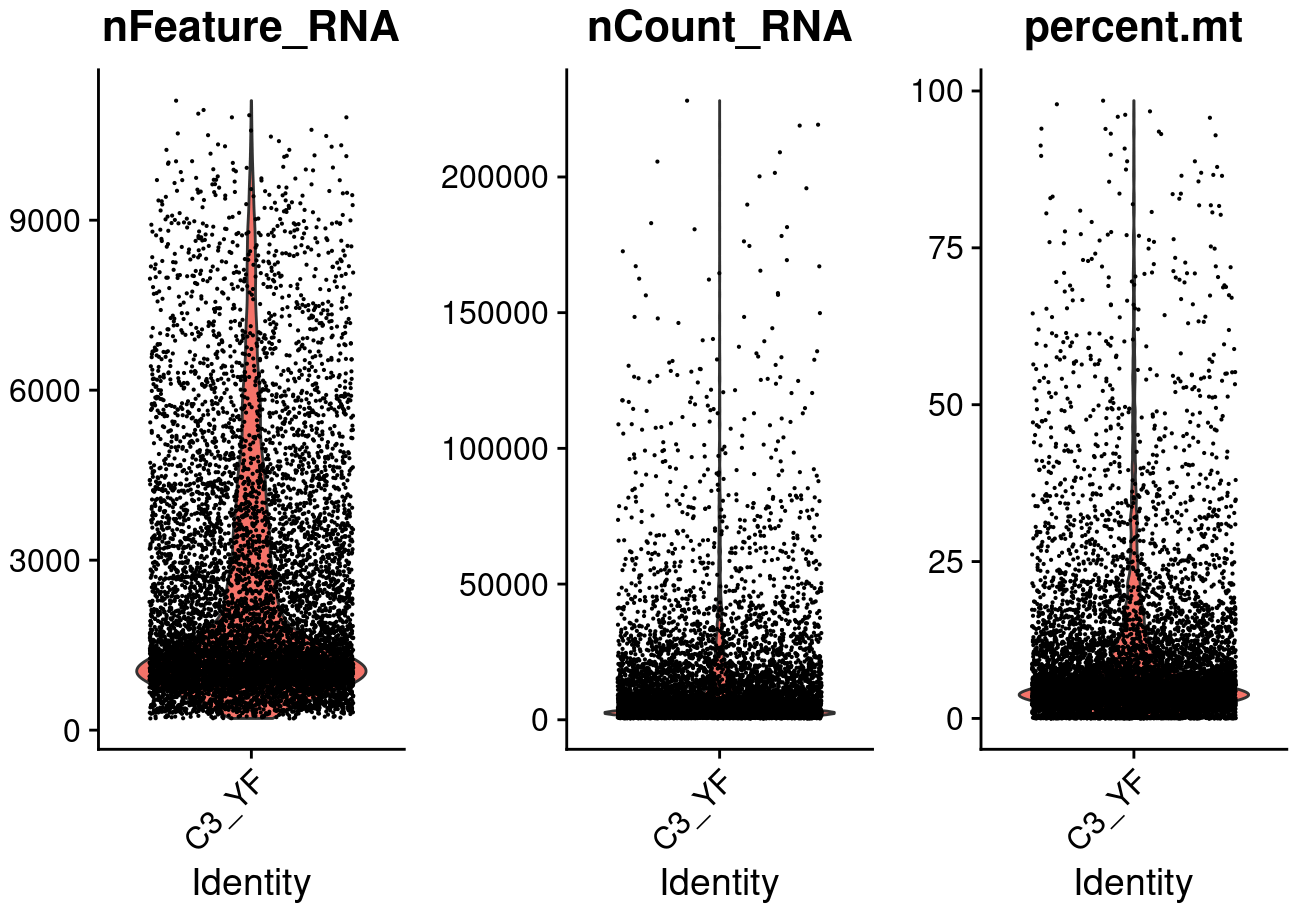

```
VlnPlot(C2_ZY, features = c("nFeature_RNA", "nCount_RNA", "percent.mt"), ncol = 3)
```

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



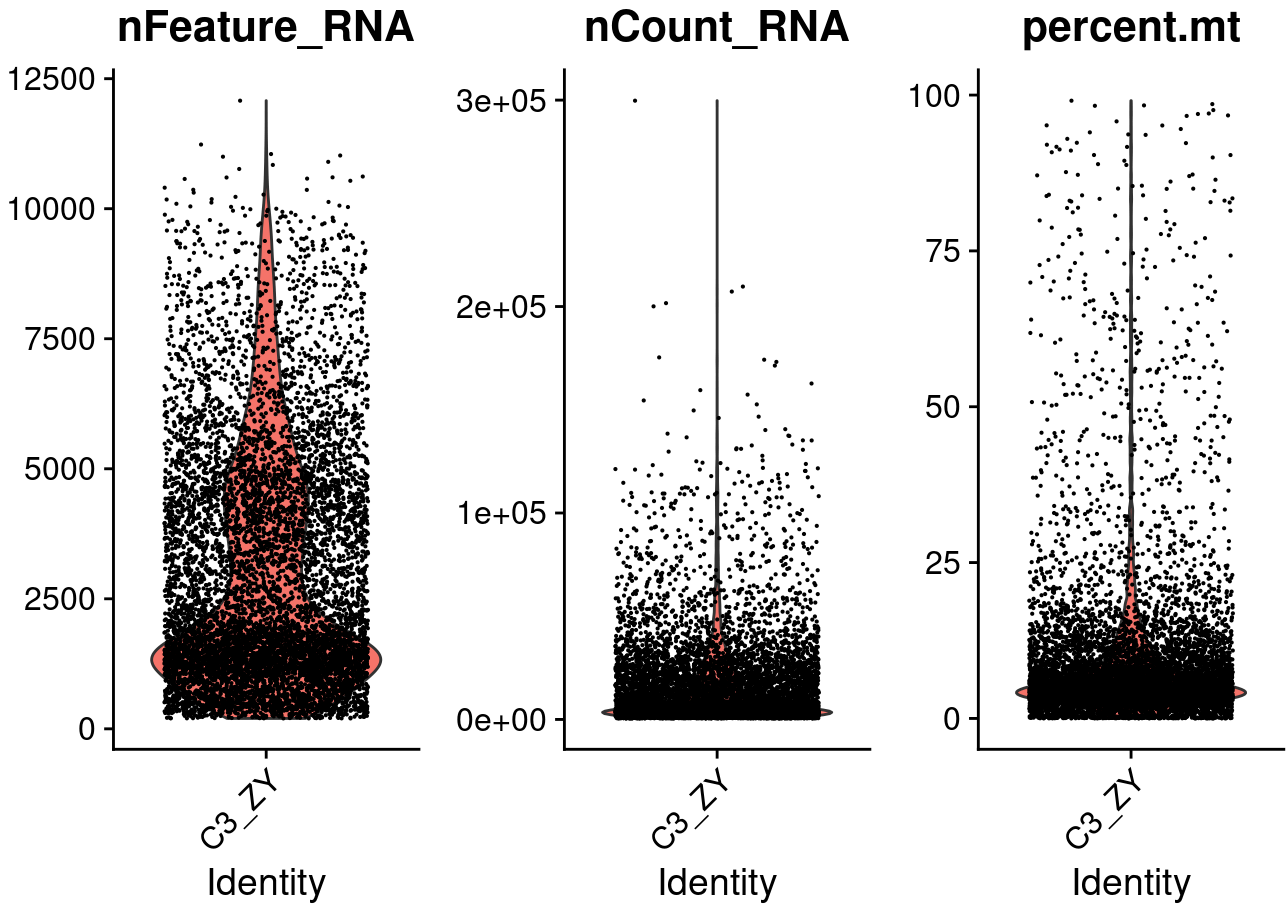

```
VlnPlot(C3_YF, features = c("nFeature_RNA", "nCount_RNA", "percent.mt"), ncol = 3)
```

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



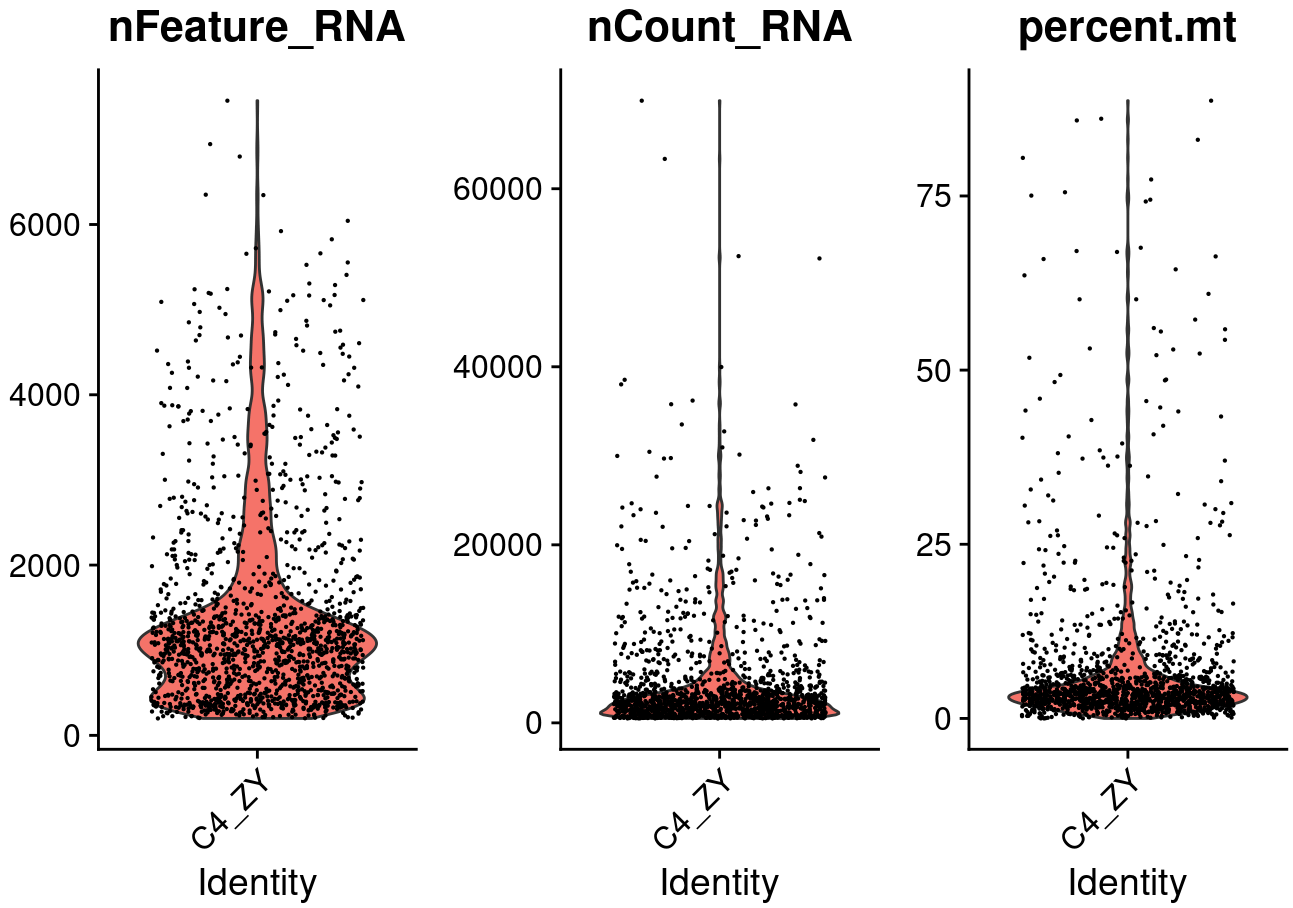

```
VlnPlot(C3_ZY, features = c("nFeature_RNA", "nCount_RNA", "percent.mt"), ncol = 3)
```

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




```
VlnPlot(C4_ZY, features = c("nFeature_RNA", "nCount_RNA", "percent.mt"), ncol = 3)
```

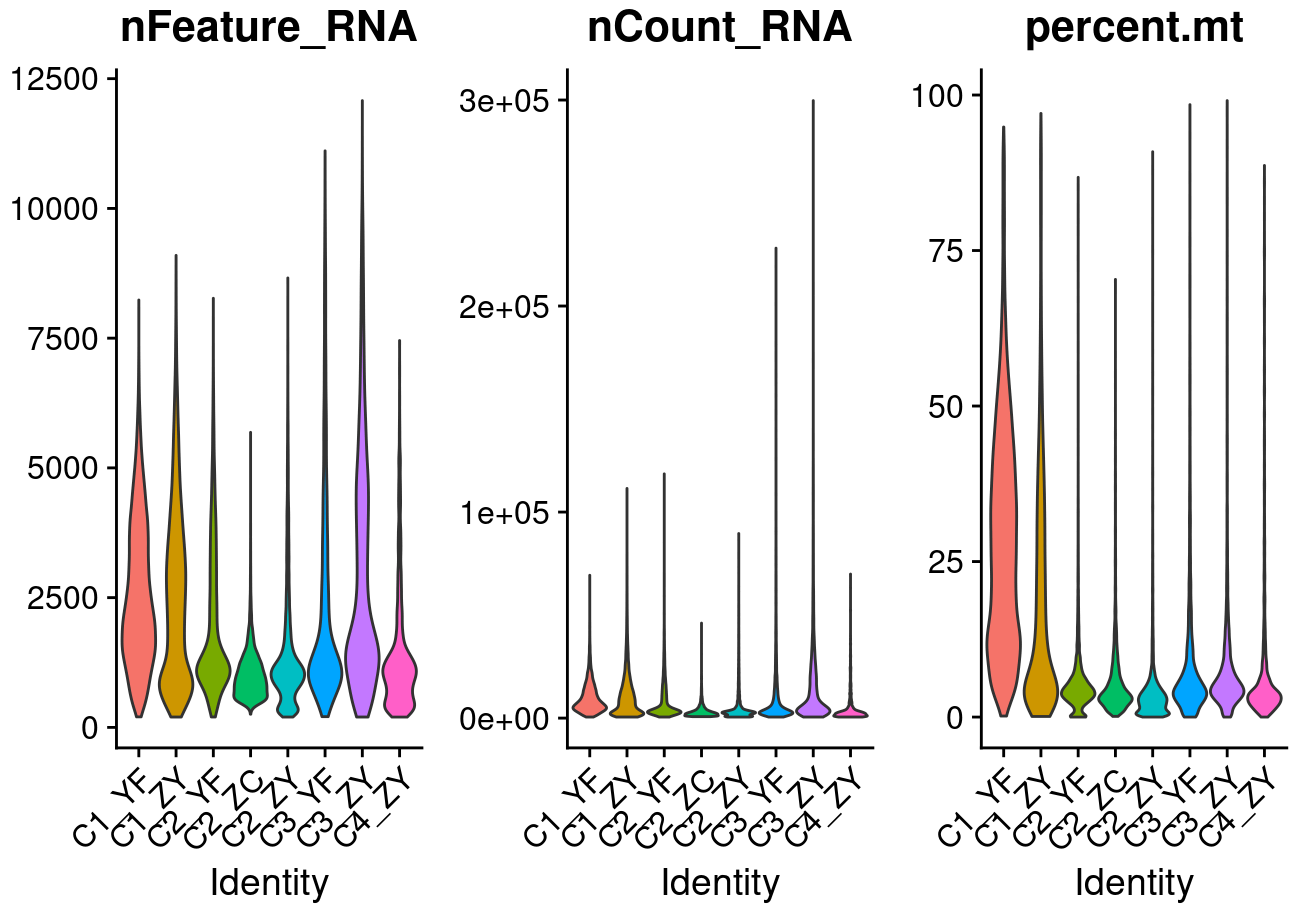
```
## Warning: Default search for "data" layer in "RNA" assay yielded no results;
## utilizing "counts" layer instead.
```



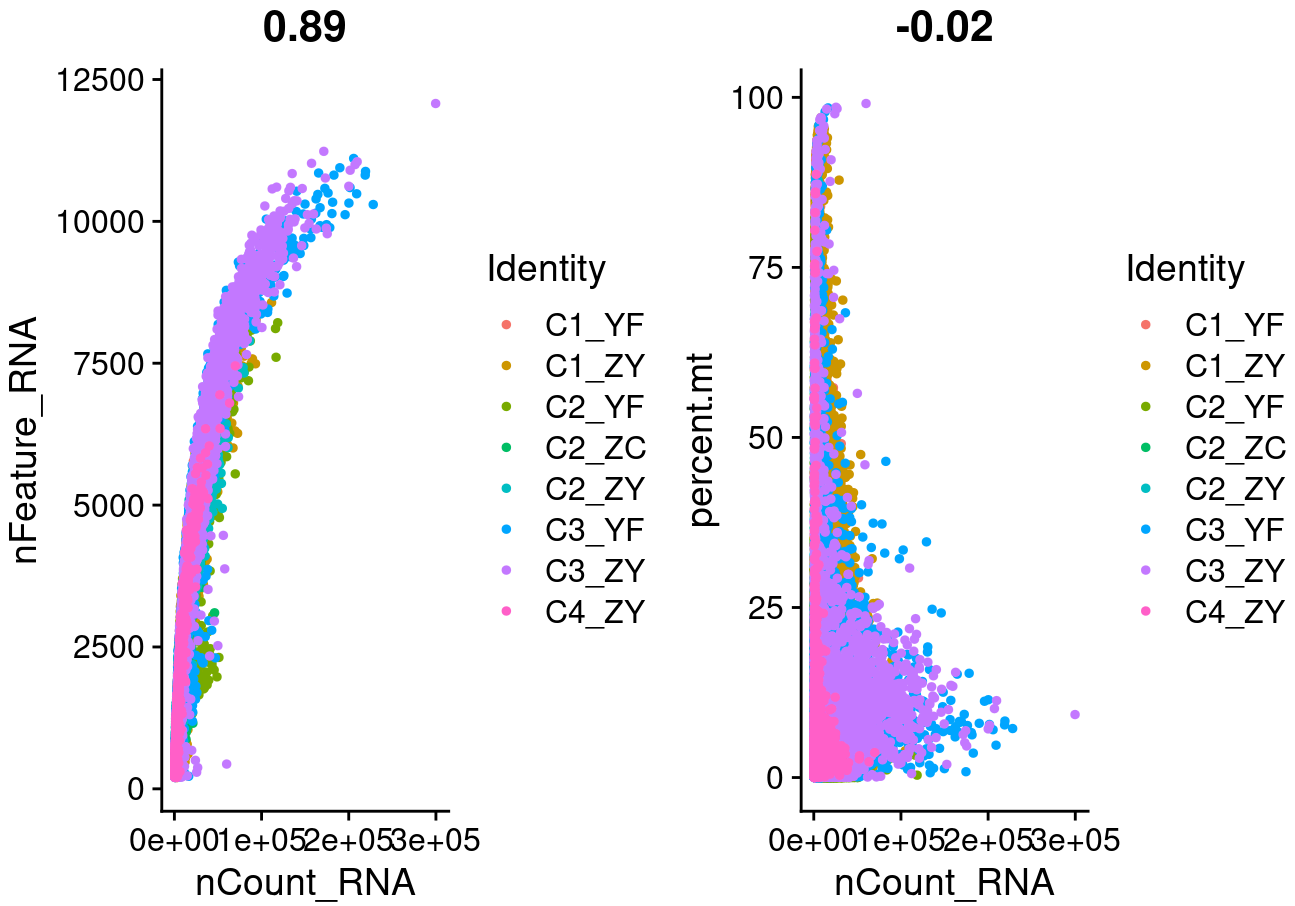
An integrated violin plot:

```
merged <- merge(
  C1_YF, #Anchor obj
  y = list(C1_ZY, C2_YF, C2_ZC, C2_ZY, C3_YF, C3_ZY, C4_ZY),
  add.cell.ids = c("C1_YF","C1_ZY","C2_YF","C2_ZC","C2_ZY","C3_YF","C3_ZY","C4_ZY")
)
VlnPlot(merged,
        features = c("nFeature_RNA","nCount_RNA","percent.mt"),
        group.by = "orig.ident",
        pt.size = 0,
        ncol = 3)
```

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




```
# Scatter plots to inspect feature-feature relationships:
plot1 <- FeatureScatter(merged, "nCount_RNA", "nFeature_RNA")

plot2 <- FeatureScatter(merged, "nCount_RNA", "percent.mt")
plot1 + plot2
```



Summary table of samples before filtering:

```
samples <- list(
  C1_YF = C1_YF, C1_ZY = C1_ZY, C2_YF = C2_YF, C2_ZC = C2_ZC,
  C2_ZY = C2_ZY, C3_YF = C3_YF, C3_ZY = C3_ZY, C4_ZY = C4_ZY
)
summary_before <- data.frame(
  sample = names(samples),
  cells  = sapply(samples, ncol),
  genes  = sapply(samples, nrow)
)
print(summary_before)
```

```
##       sample cells genes
## C1_YF  C1_YF 21481 21640
## C1_ZY  C1_ZY 13567 21572
## C2_YF  C2_YF 11717 21715
## C2_ZC  C2_ZC  6605 18492
## C2_ZY  C2_ZY  9352 22523
## C3_YF  C3_YF  9169 22668
## C3_ZY  C3_ZY  8532 22970
## C4_ZY  C4_ZY  1516 16470
```

Filter cells by QC thresholds:

```
# keep cells with >200  and percent.mt  <20
filtered_samples <- lapply(
  samples,
  function(obj) subset(obj, subset = nFeature_RNA > 200 & percent.mt < 20)
)

## Summary table after filtering:
summary_after  <- data.frame(
  sample = names(filtered_samples),
  cells  = sapply(filtered_samples, ncol),
  genes_detected = sapply(filtered_samples, function(obj){
    counts <- GetAssayData(obj, assay="RNA", slot="counts")
    sum(rowSums(counts > 0) > 0)
  }
  ))
```

```
## Warning: The `slot` argument of `GetAssayData()` is deprecated as of SeuratObject 5.0.0.
## ℹ Please use the `layer` argument instead.
## This warning is displayed once every 8 hours.
## Call `lifecycle::last_lifecycle_warnings()` to see where this warning was
## generated.
```

```
print(summary_after)
```

```
##       sample cells genes_detected
## C1_YF  C1_YF  8160          21482
## C1_ZY  C1_ZY  8497          21511
## C2_YF  C2_YF 11348          21715
## C2_ZC  C2_ZC  6398          18492
## C2_ZY  C2_ZY  9100          22523
## C3_YF  C3_YF  8176          22666
## C3_ZY  C3_ZY  7808          22969
## C4_ZY  C4_ZY  1406          16469
```

Log-normalize & find HVGs

```
normalized_samples <- lapply(filtered_samples, function(obj) {
  NormalizeData(obj, normalization.method="LogNormalize", scale.factor=1e4) %>%
    FindVariableFeatures(selection.method="vst", nfeatures=2000)
})
```

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

Example on first sample

```
obj1   <- normalized_samples[[1]]
top10  <- head(VariableFeatures(obj1), 10)
bot10  <- tail(VariableFeatures(obj1), 10)
print(top10, caption="Top 10 highly variable genes")
```

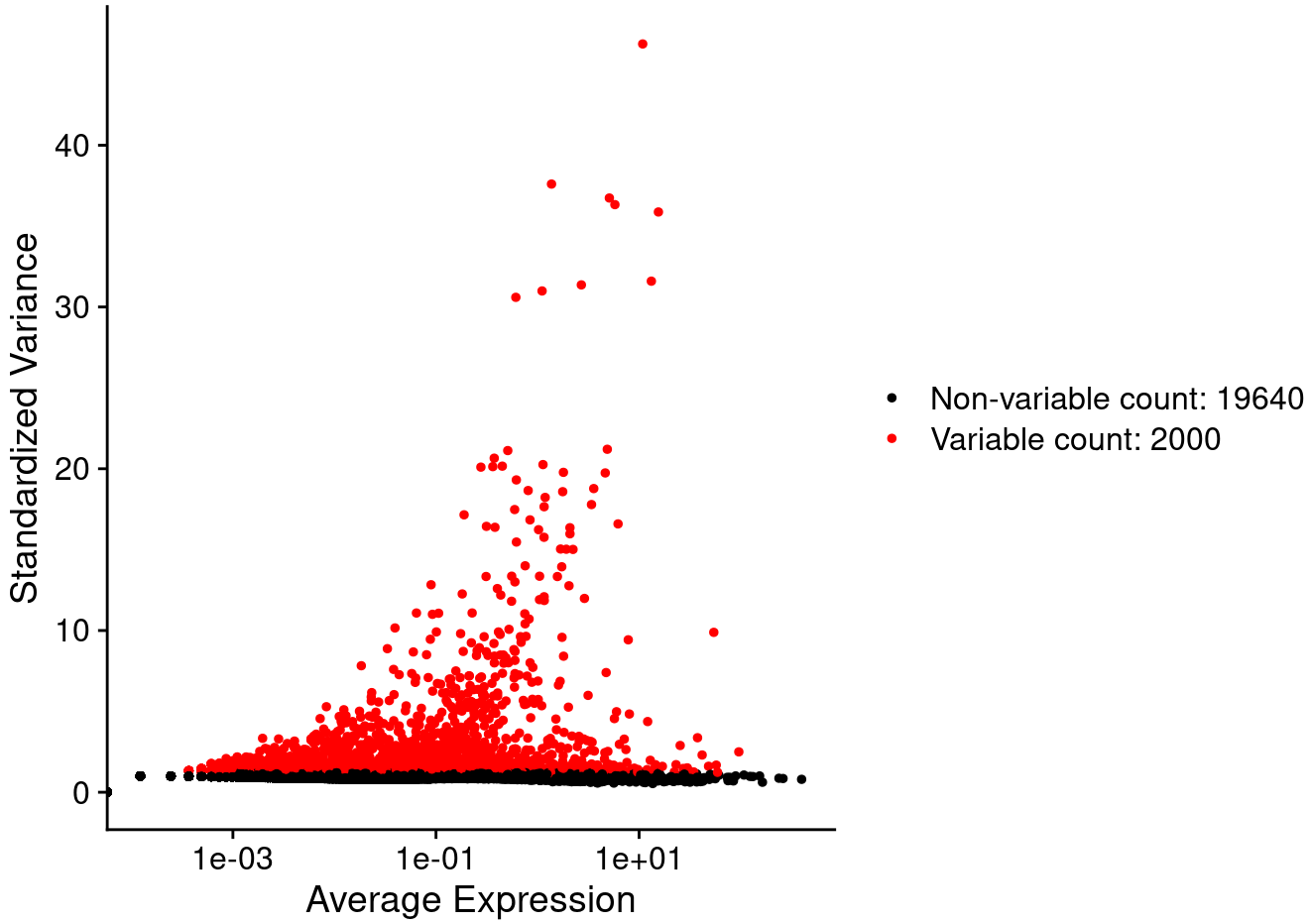
```
##  [1] "SPP1"   "JCHAIN" "IGHA1"  "APOE"   "IGKC"   "IGLC2"  "APOC1"  "CXCL14"
##  [9] "MMP12"  "COL1A1"
```

```
print(bot10, caption="Bottom 10 least variable genes")
```

```
##  [1] "ZNF521"     "FA2H"       "OIP5"       "HIST1H4H"   "TSPAN1"
##  [6] "ASNS"       "ETS2"       "LRRK2"      "CD300E"     "AP001160.2"
```

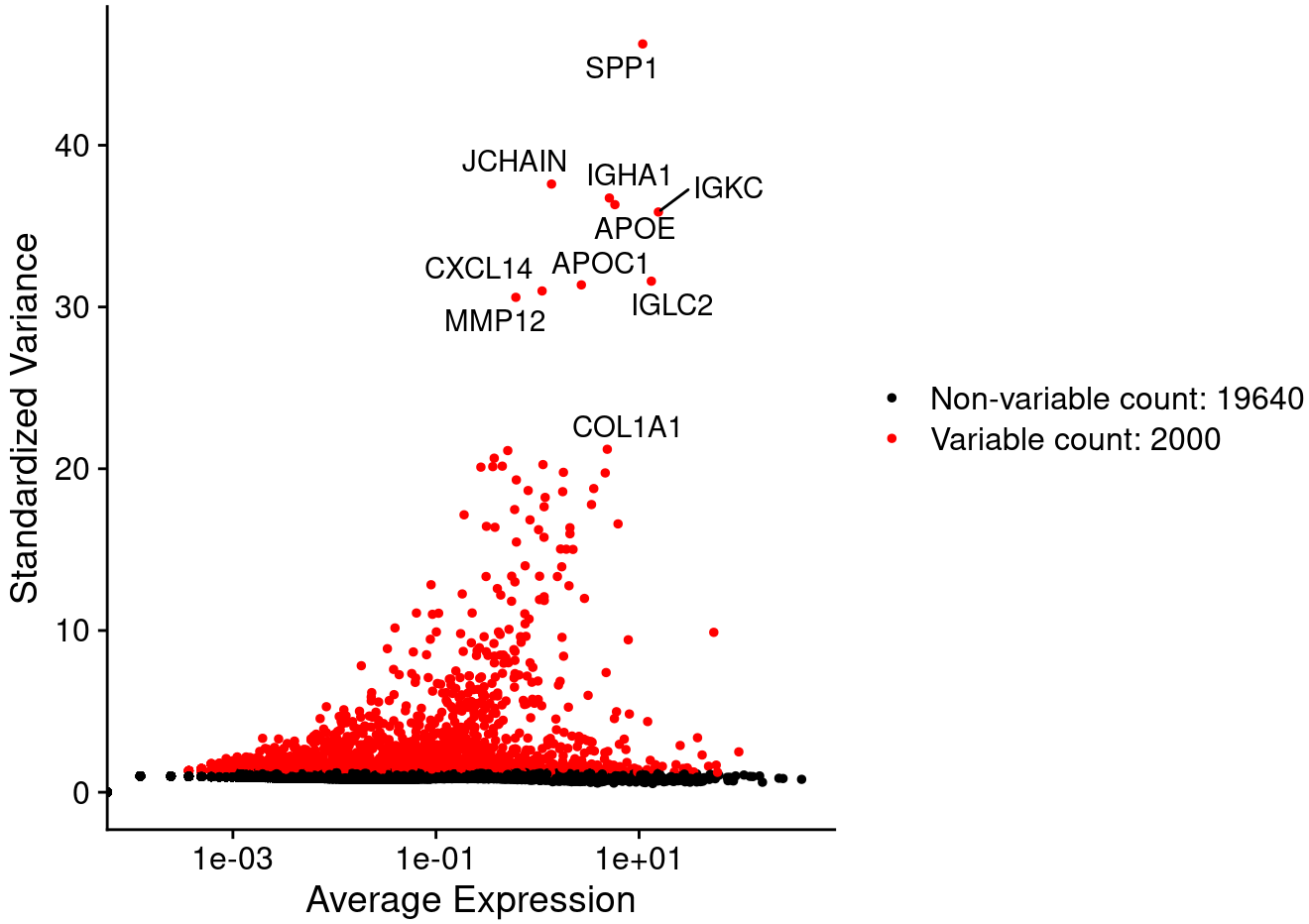
```
p1     <- VariableFeaturePlot(obj1)
p2     <- LabelPoints(p1, points=top10, repel=TRUE, xnudge=0, ynudge=0)
p3     <- LabelPoints(p1, points=bot10, repel=TRUE, xnudge=0, ynudge=0)
print(p1)
```

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



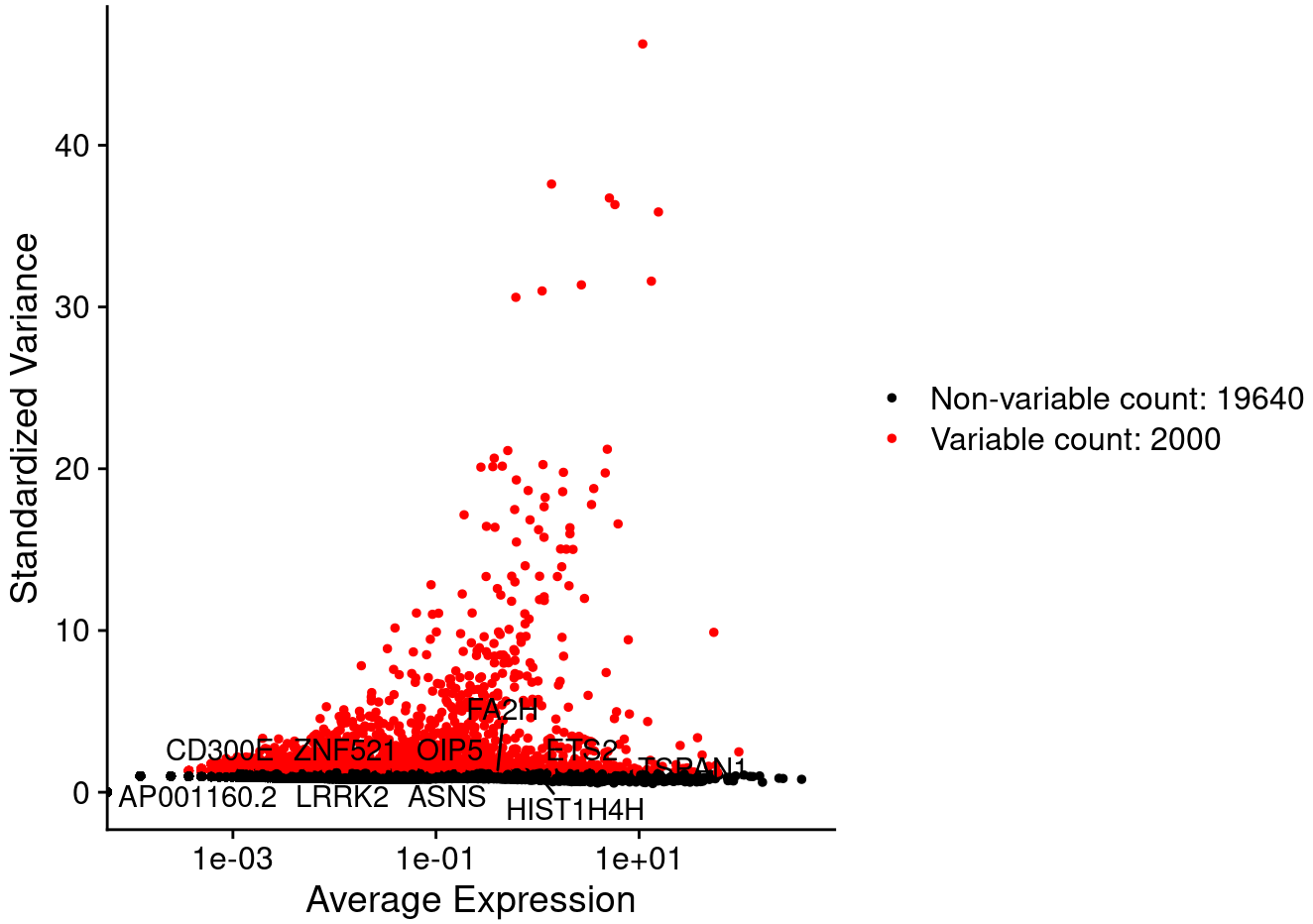

```
print(p2, caption="Top10-HVG")
```

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




```
print(p3, caption="Least variable genes")
```

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



Scale data and PCA:

```
pca_samples <- lapply(normalized_samples, function(obj) {
  obj %>%
    ScaleData(vars.to.regress=c("nCount_RNA","percent.mt")) %>% ##Regressed out count and percen
t data as described in the publication.

    RunPCA(features=VariableFeatures(object=obj), verbose=FALSE)
})
```

```
## Regressing out nCount_RNA, percent.mt
```

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

```
## Regressing out nCount_RNA, percent.mt
```

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

```
## Regressing out nCount_RNA, percent.mt
```

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

```
## Regressing out nCount_RNA, percent.mt
```

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

```
## Regressing out nCount_RNA, percent.mt
```

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

```
## Regressing out nCount_RNA, percent.mt
```

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

```
## Regressing out nCount_RNA, percent.mt
```

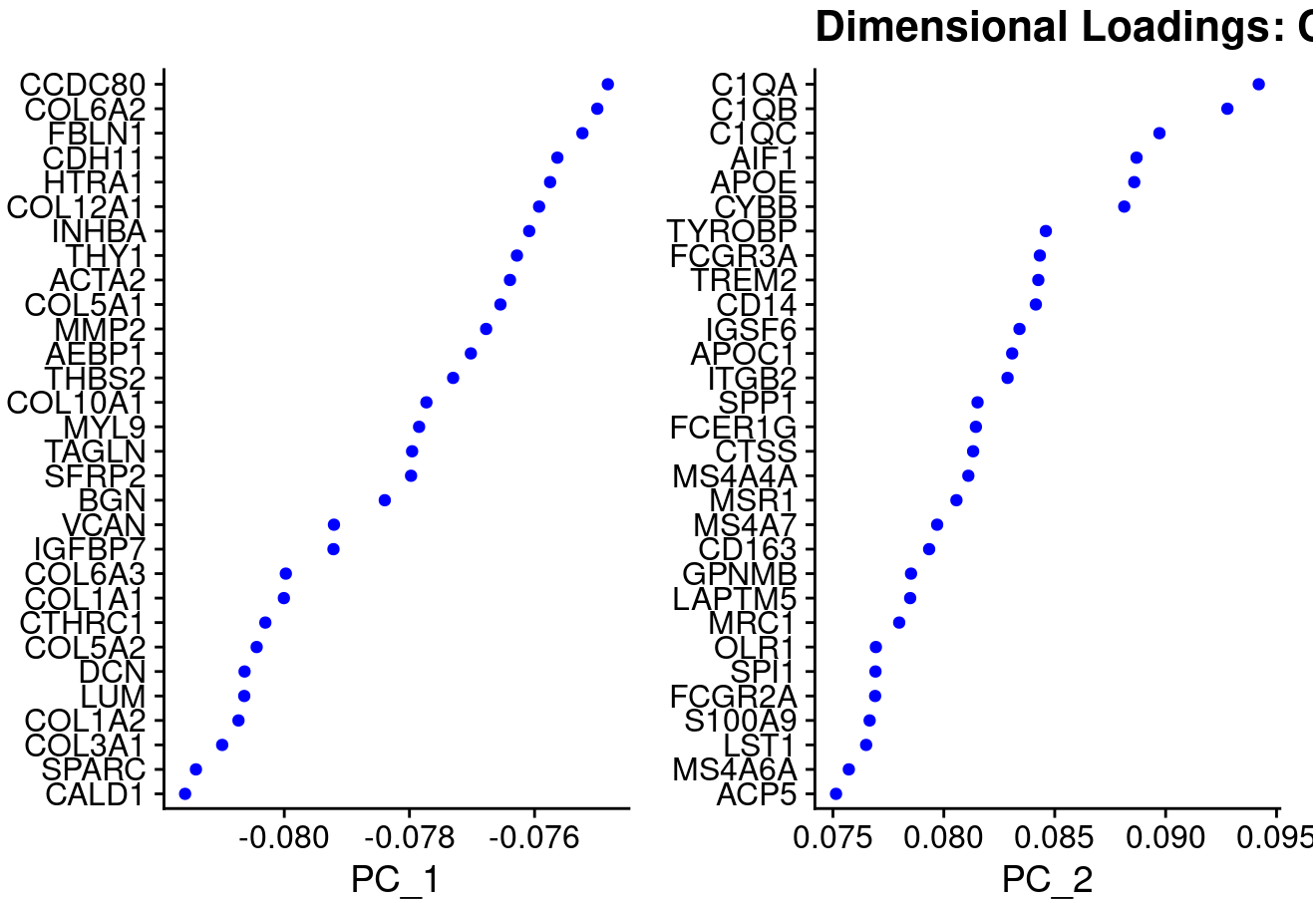
```
## Centering and scaling data matrix
```

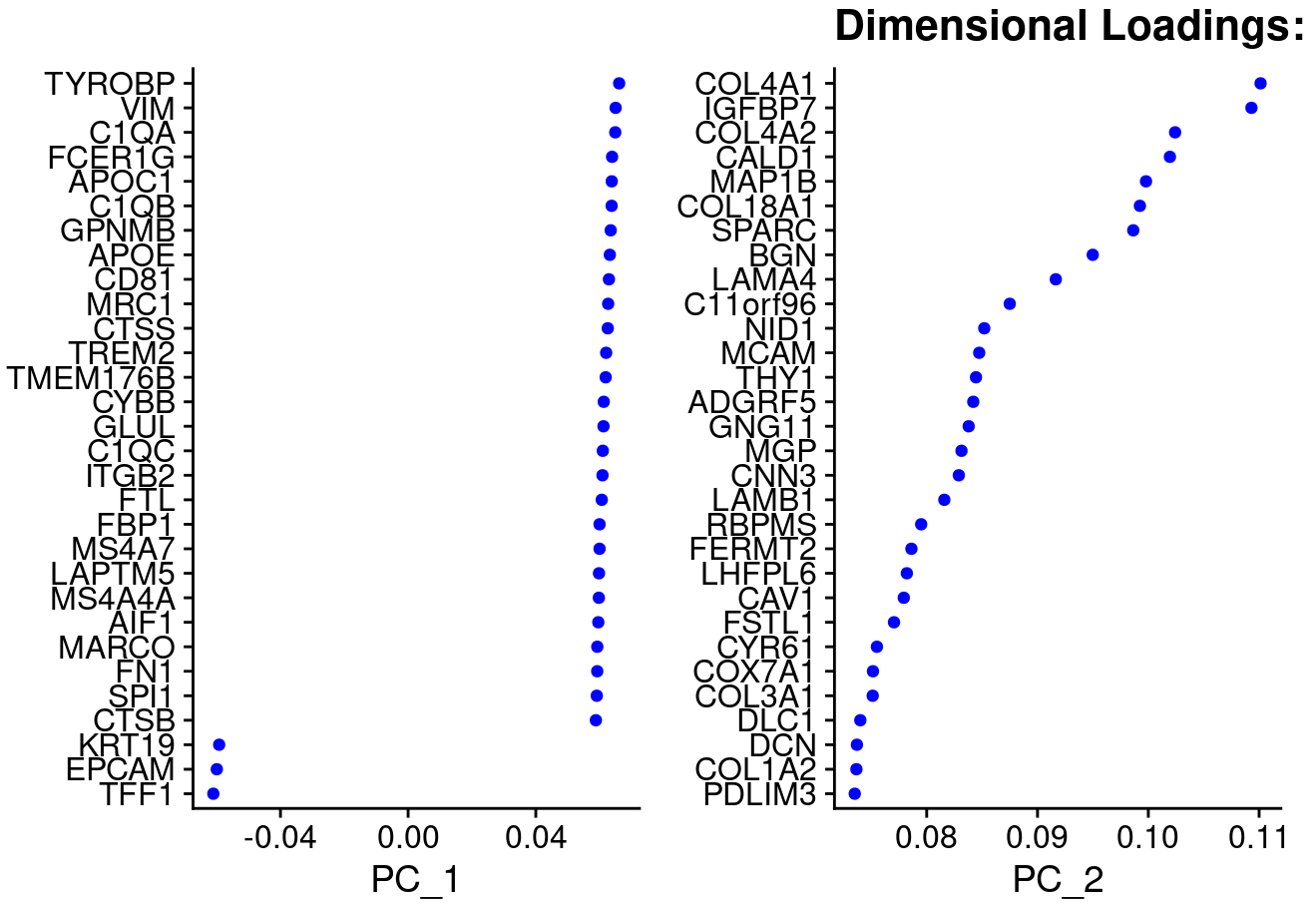
```
## Regressing out nCount_RNA, percent.mt
```

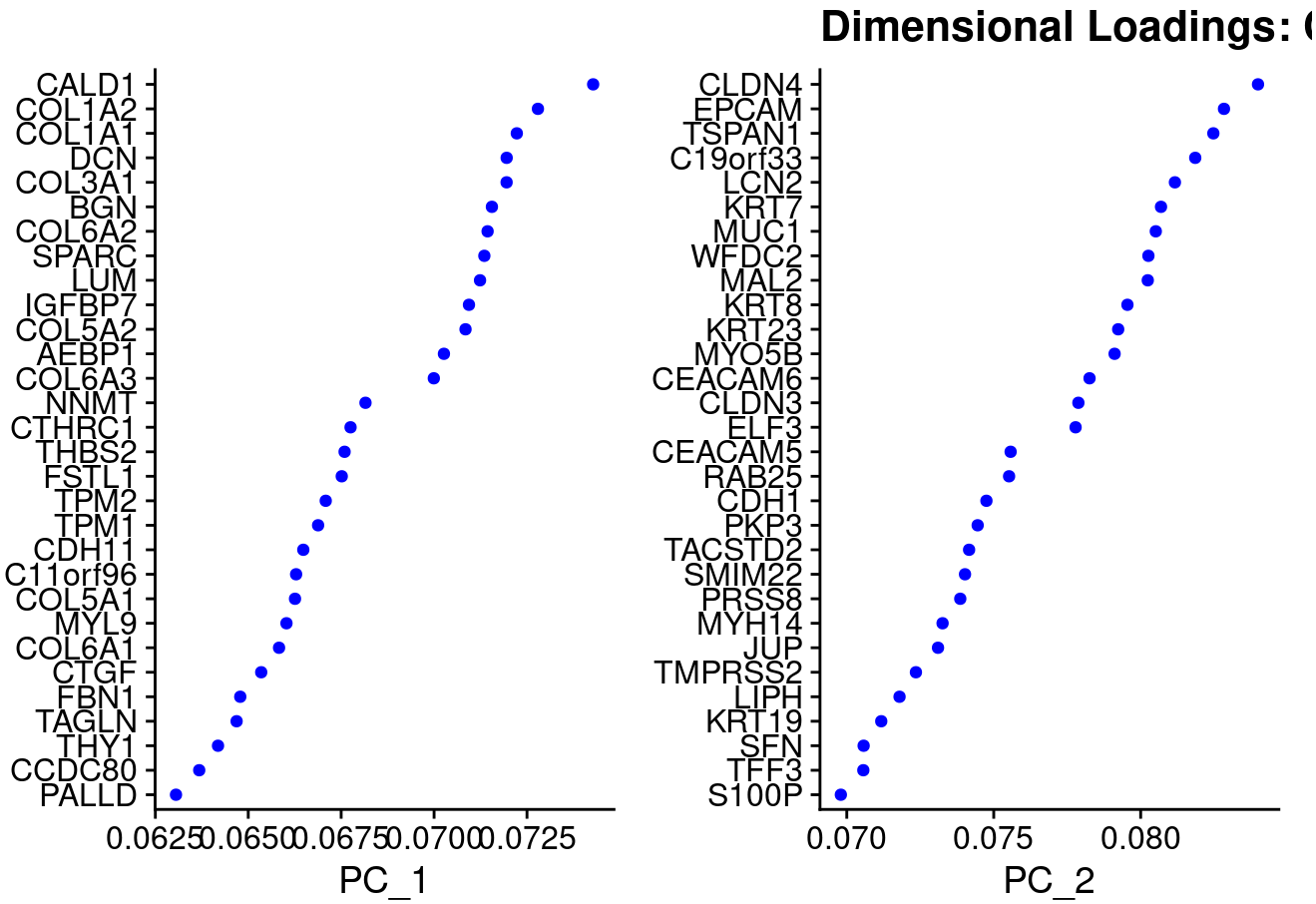
```
## Centering and scaling data matrix
```

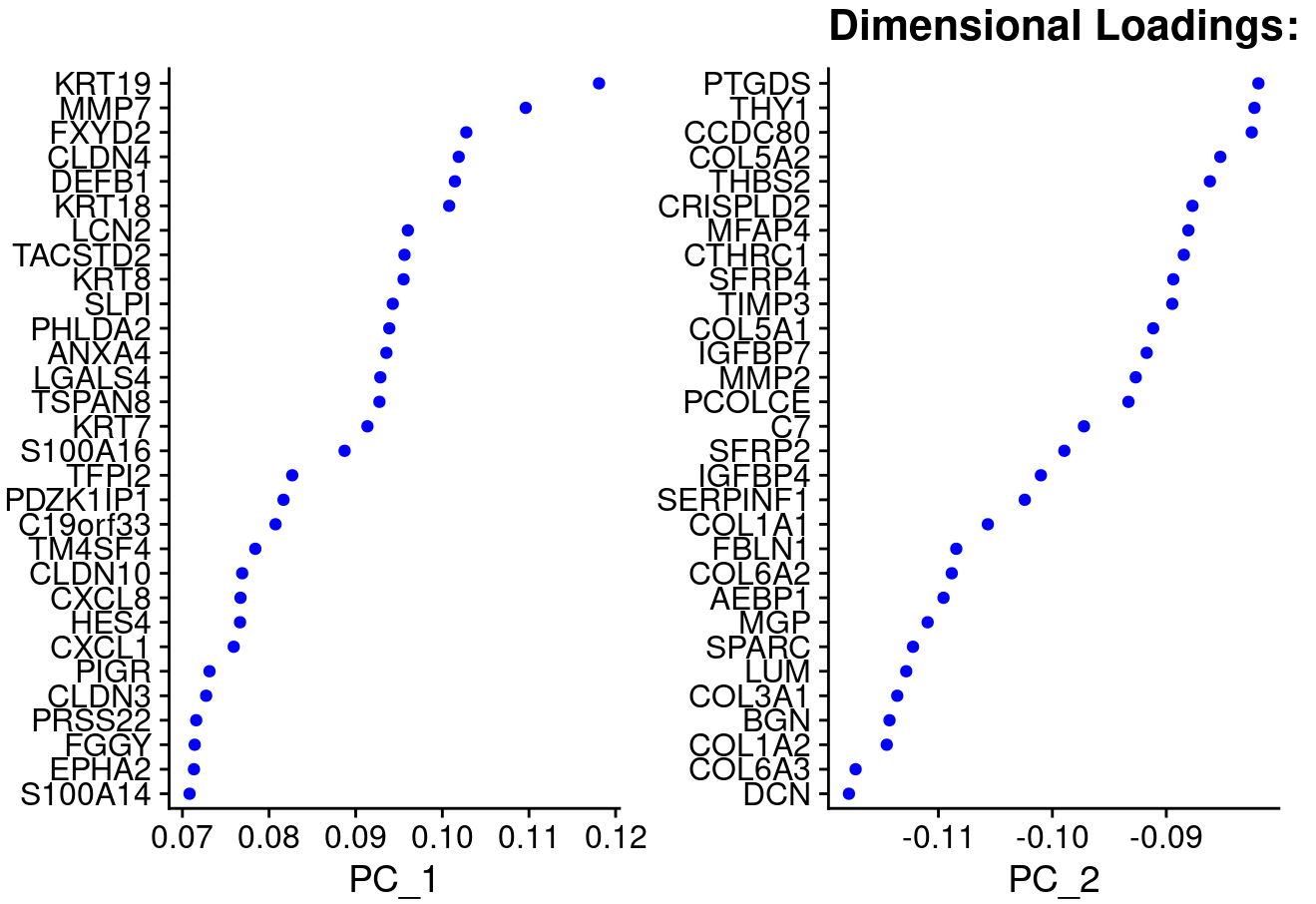
Visualize dimension loadings:

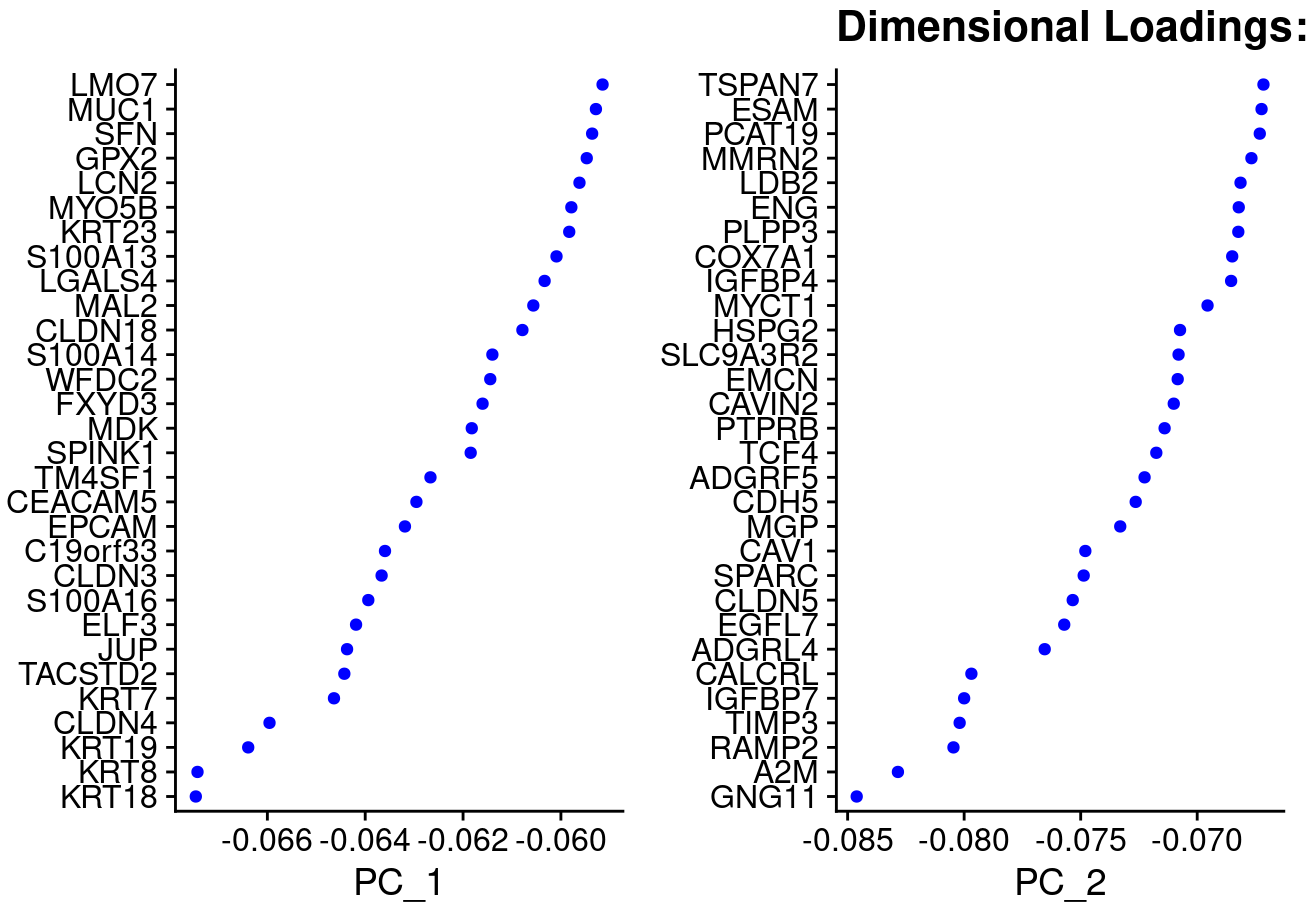
```
# One plot per sample
lapply(names(pca_samples), function(nm) {
  obj <- pca_samples[[nm]]
  print(
    VizDimLoadings(obj, dims = 1:2, reduction = "pca") +
      ggtitle(paste0("Dimensional Loadings: ", nm))
  )
})
```

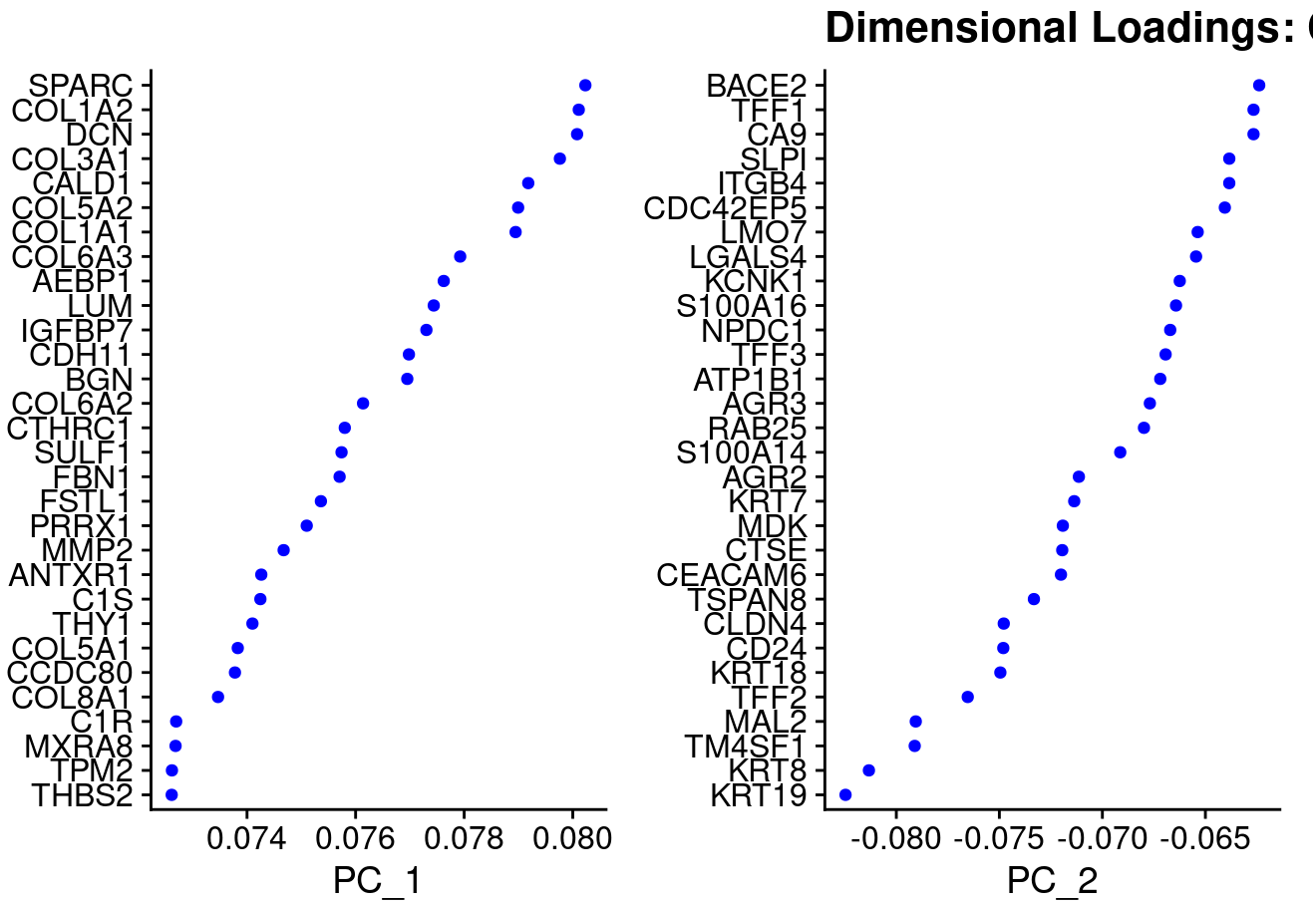

Dimensional Loadings: C
CCDC80
COL6A2
FBLN1
CDH11
HTRA1
COL12A1
INHBA
THY1
ACTA2
COL5A1
MMP2
AEBP1
THBS2
COL10A1
MYL9
TAGLN
SFRP2
BGN
VCAN
IGFBP7
COL6A3
COL1A1
CTHRC1
COL5A2
DCN
LUM
COL1A2
COL3A1
SPARC
CALD1
-0.080
-0.078
-0.076
PC_1
C1QA
C1QB
C1QC
AIF1
APOE
CYBB
TYROBP
FCGR3A
TREM2
CD14
IGSF6
APOC1
ITGB2
SPP1
FCER1G
CTSS
MS4A4A
MSR1
MS4A7
CD163
GPNMB
LAPTM5
MRC1
OLR1
SPI1
FCGR2A
S100A9
LST1
MS4A6A
ACP5
0.075
0.080
0.085
0.090
0.095
PC_2


Dimensional Loadings:
TYROBP
VIM
C1QA
FCER1G
APOC1
C1QB
GPNMB
APOE
CD81
MRC1
CTSS
TREM2
TMEM176B
CYBB
GLUL
C1QC
ITGB2
FTL
FBP1
MS4A7
LAPTM5
MS4A4A
AIF1
MARCO
FN1
SPI1
CTSB
KRT19
EPCAM
TFF1
-0.04
0.00
0.04
PC_1
COL4A1
IGFBP7
COL4A2
CALD1
MAP1B
COL18A1
SPARC
BGN
LAMA4
C11orf96
NID1
MCAM
THY1
ADGRF5
GNG11
MGP
CNN3
LAMB1
RBPMS
FERMT2
LHFPL6
CAV1
FSTL1
CYR61
COX7A1
COL3A1
DLC1
DCN
COL1A2
PDLIM3
0.08
0.09
0.10
0.11
PC_2

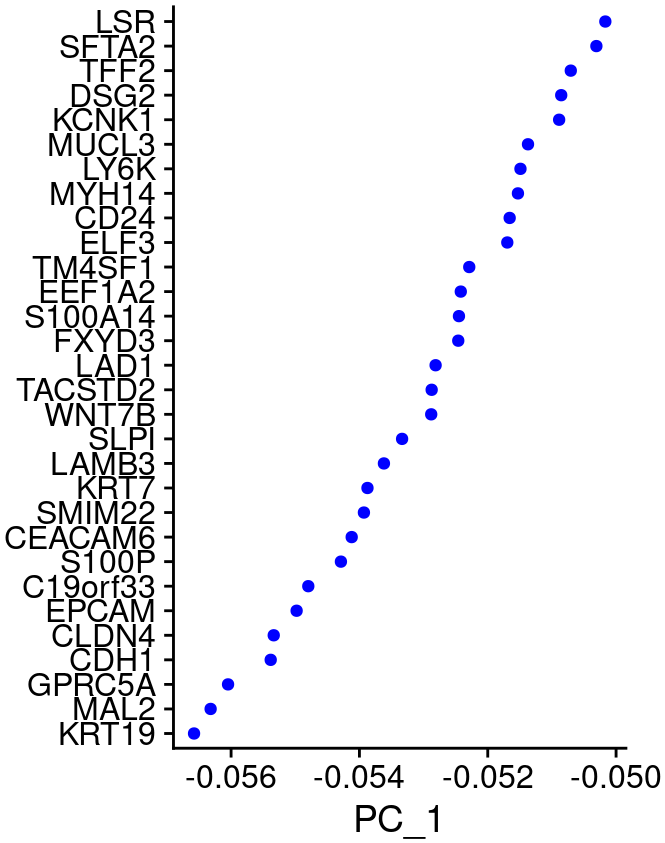

Dimensional Loadings: (
CALD1
COL1A2
COL1A1
DCN
COL3A1
BGN
COL6A2
SPARC
LUM
IGFBP7
COL5A2
AEBP1
COL6A3
NNMT
CTHRC1
THBS2
FSTL1
TPM2
TPM1
CDH11
C11orf96
COL5A1
MYL9
COL6A1
CTGF
FBN1
TAGLN
THY1
CCDC80
PALLD
0.06250.06500.06750.07000.0725
PC_1
CLDN4
EPCAM
TSPAN1
C19orf33
LCN2
KRT7
MUC1
WFDC2
MAL2
KRT8
KRT23
MYO5B
CEACAM6
CLDN3
ELF3
CEACAM5
RAB25
CDH1
PKP3
TACSTD2
SMIM22
PRSS8
MYH14
JUP
TMPRSS2
LIPH
KRT19
SFN
TFF3
S100P
0.070 0.075 0.080
PC_2

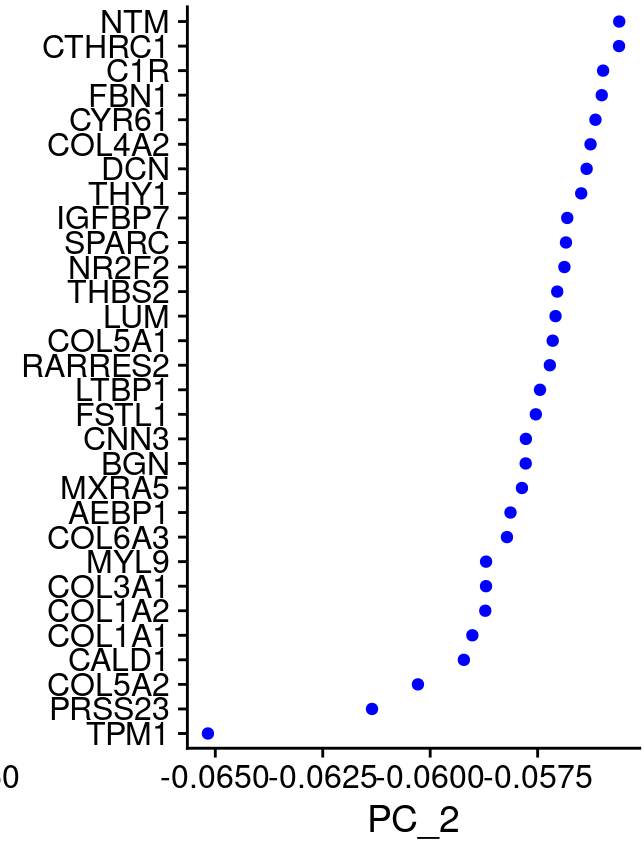

Dimensional Loadings:
KRT19
MMP7
FXYD2
CLDN4
DEFB1
KRT18
LCN2
TACSTD2
KRT8
SLPI
PHLDA2
ANXA4
LGALS4
TSPAN8
KRT7
S100A16
TFPI2
PDZK1IP1
C19orf33
TM4SF4
CLDN10
CXCL8
HES4
CXCL1
PIGR
CLDN3
PRSS22
FGGY
EPHA2
S100A14
0.07 0.08 0.09 0.10 0.11 0.12
PC_1
PTGDS
THY1
CCDC80
COL5A2
THBS2
CRISPLD2
MFAP4
CTHRC1
SFRP4
TIMP3
COL5A1
IGFBP7
MMP2
PCOLCE
C7
SFRP2
IGFBP4
SERPINF1
COL1A1
FBLN1
COL6A2
AEBP1
MGP
SPARC
LUM
COL3A1
BGN
COL1A2
COL6A3
DCN
-0.11 -0.10 -0.09
PC_2


Dimensional Loadings:
LMO7
MUC1
SFN
GPX2
LCN2
MYO5B
KRT23
S100A13
LGALS4
MAL2
CLDN18
S100A14
WFDC2
FXYD3
MDK
SPINK1
TM4SF1
CEACAM5
EPCAM
C19orf33
CLDN3
S100A16
ELF3
JUP
TACSTD2
KRT7
CLDN4
KRT19
KRT8
KRT18
-0.066 -0.064 -0.062 -0.060
PC_1
TSPAN7
ESAM
PCAT19
MMRN2
LDB2
ENG
PLPP3
COX7A1
IGFBP4
MYCT1
HSPG2
SLC9A3R2
EMCN
CAVIN2
PTPRB
TCF4
ADGRF5
CDH5
MGP
CAV1
SPARC
CLDN5
EGFL7
ADGRL4
CALCRL
IGFBP7
TIMP3
RAMP2
A2M
GNG11
-0.085 -0.080 -0.075 -0.070
PC_2

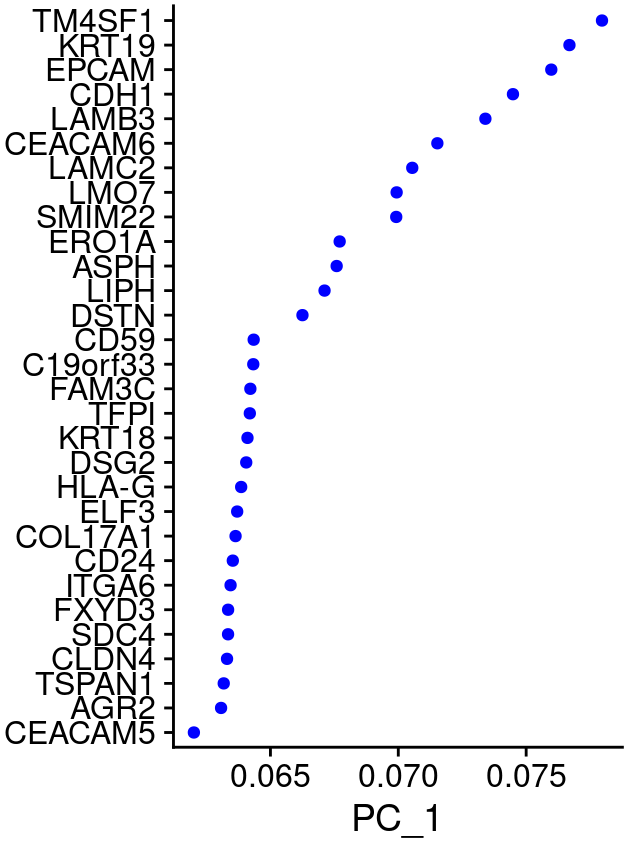

Dimensional Loadings:
SPARC
COL1A2
DCN
COL3A1
CALD1
COL5A2
COL1A1
COL6A3
AEBP1
LUM
IGFBP7
CDH11
BGN
COL6A2
CTHRC1
SULF1
FBN1
FSTL1
PRRX1
MMP2
ANTXR1
C1S
THY1
COL5A1
CCDC80
COL8A1
C1R
MXRA8
TPM2
THBS2
0.074 0.076 0.078 0.080
PC_1
BACE2
TFF1
CA9
SLPI
ITGB4
CDC42EP5
LMO7
LGALS4
KCNK1
S100A16
NPDC1
TFF3
ATP1B1
AGR3
RAB25
S100A14
AGR2
KRT7
MDK
CTSE
CEACAM6
TSPAN8
CLDN4
CD24
KRT18
TFF2
MAL2
TM4SF1
KRT8
KRT19
-0.080 -0.075 -0.070 -0.065
PC_2

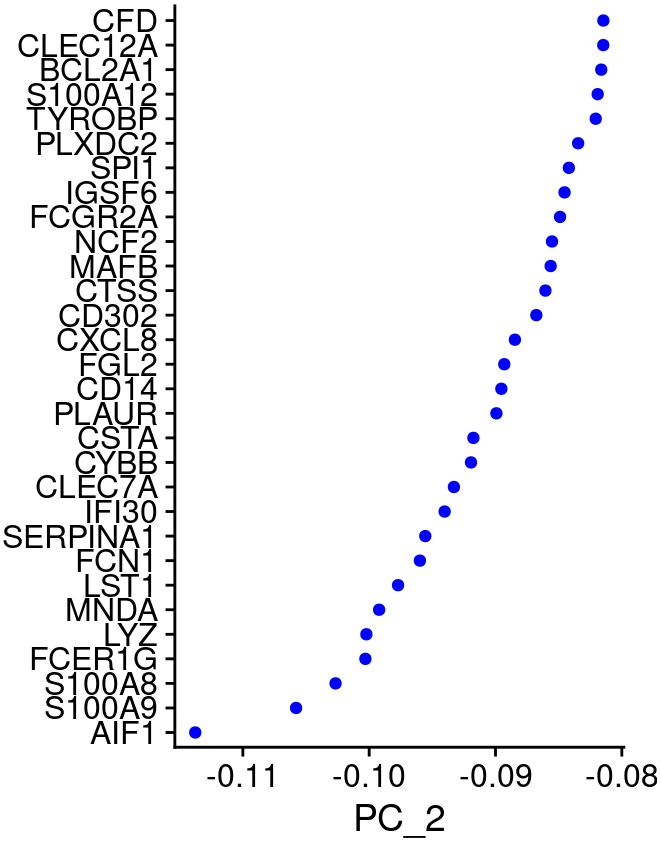
Dimensional Loadings:



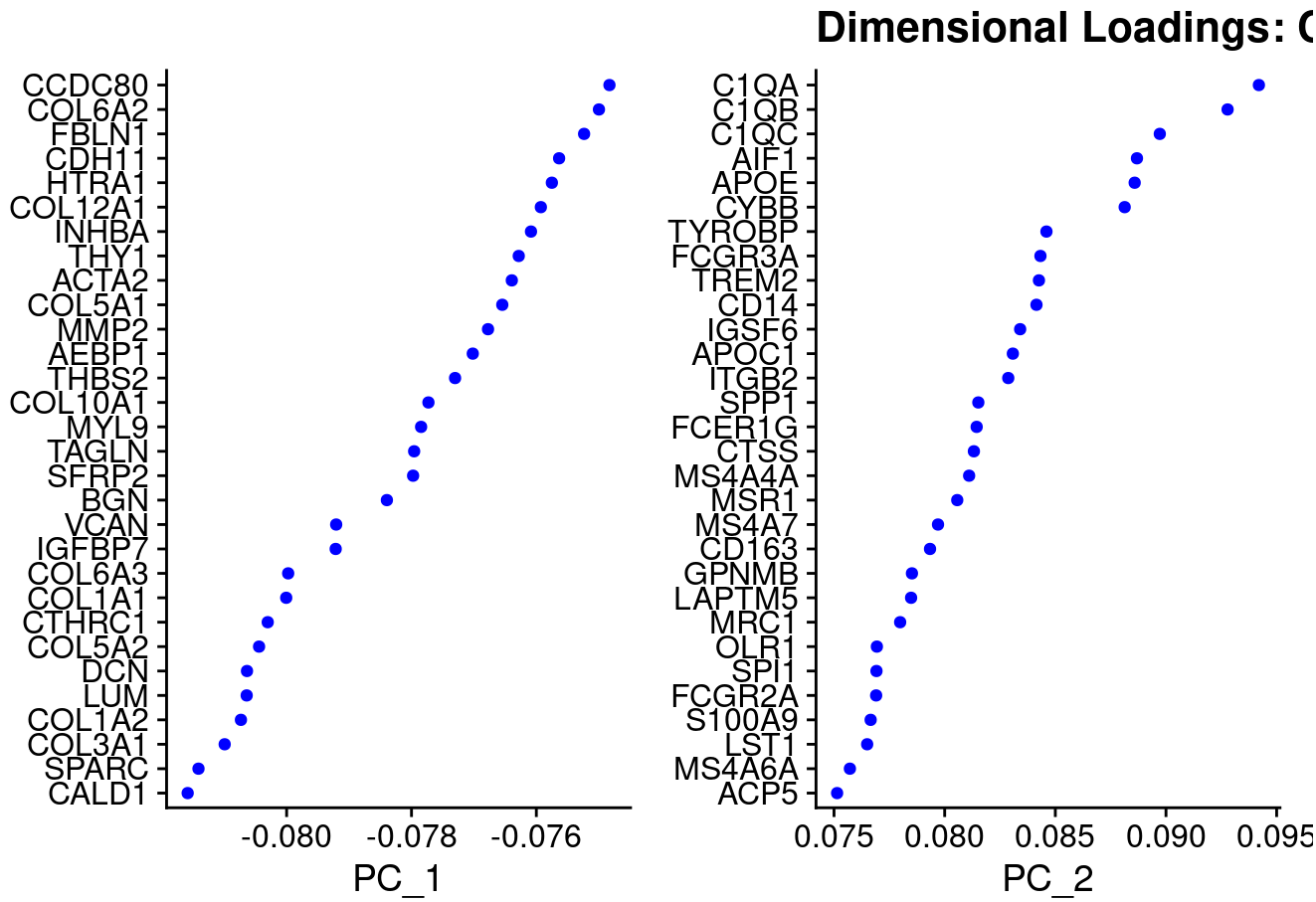


Dimensional Loadings:

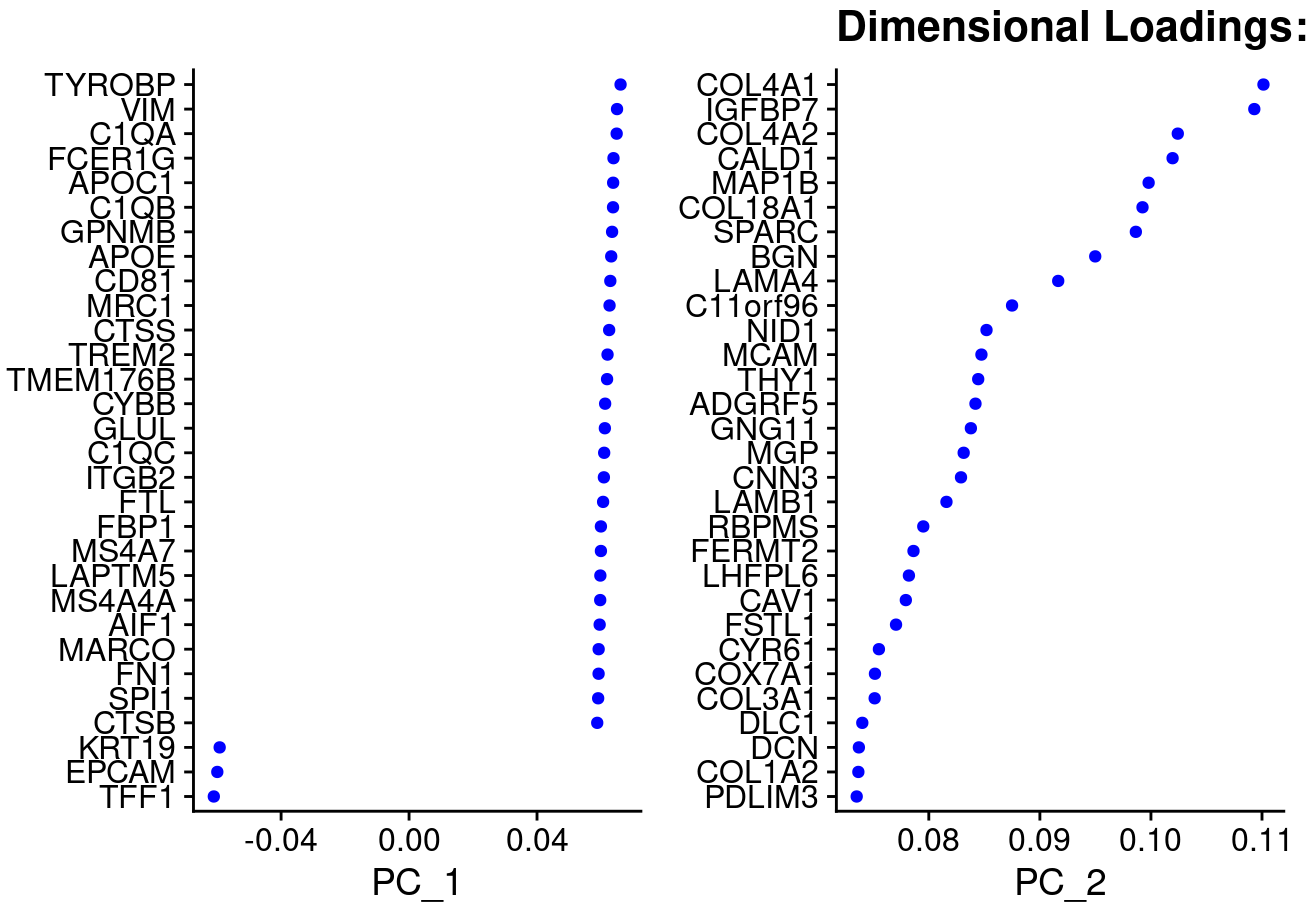




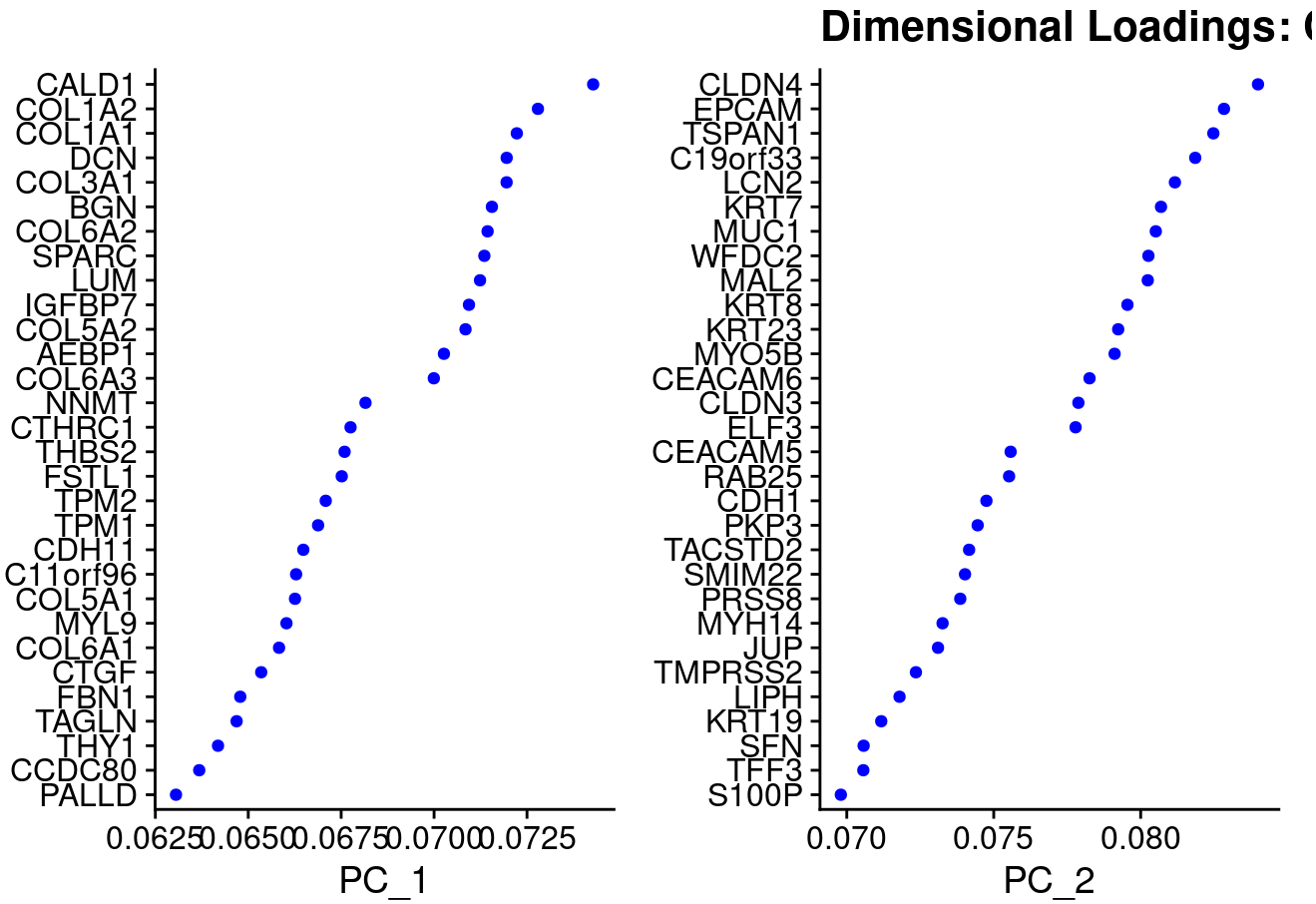
```
## [[1]]
```


Dimensional Loadings: C
CCDC80
COL6A2
FBLN1
CDH11
HTRA1
COL12A1
INHBA
THY1
ACTA2
COL5A1
MMP2
AEBP1
THBS2
COL10A1
MYL9
TAGLN
SFRP2
BGN
VCAN
IGFBP7
COL6A3
COL1A1
CTHRC1
COL5A2
DCN
LUM
COL1A2
COL3A1
SPARC
CALD1
-0.080
-0.078
-0.076
PC_1
C1QA
C1QB
C1QC
AIF1
APOE
CYBB
TYROBP
FCGR3A
TREM2
CD14
IGSF6
APOC1
ITGB2
SPP1
FCER1G
CTSS
MS4A4A
MSR1
MS4A7
CD163
GPNMB
LAPTM5
MRC1
OLR1
SPI1
FCGR2A
S100A9
LST1
MS4A6A
ACP5
0.075
0.080
0.085
0.090
0.095
PC_2

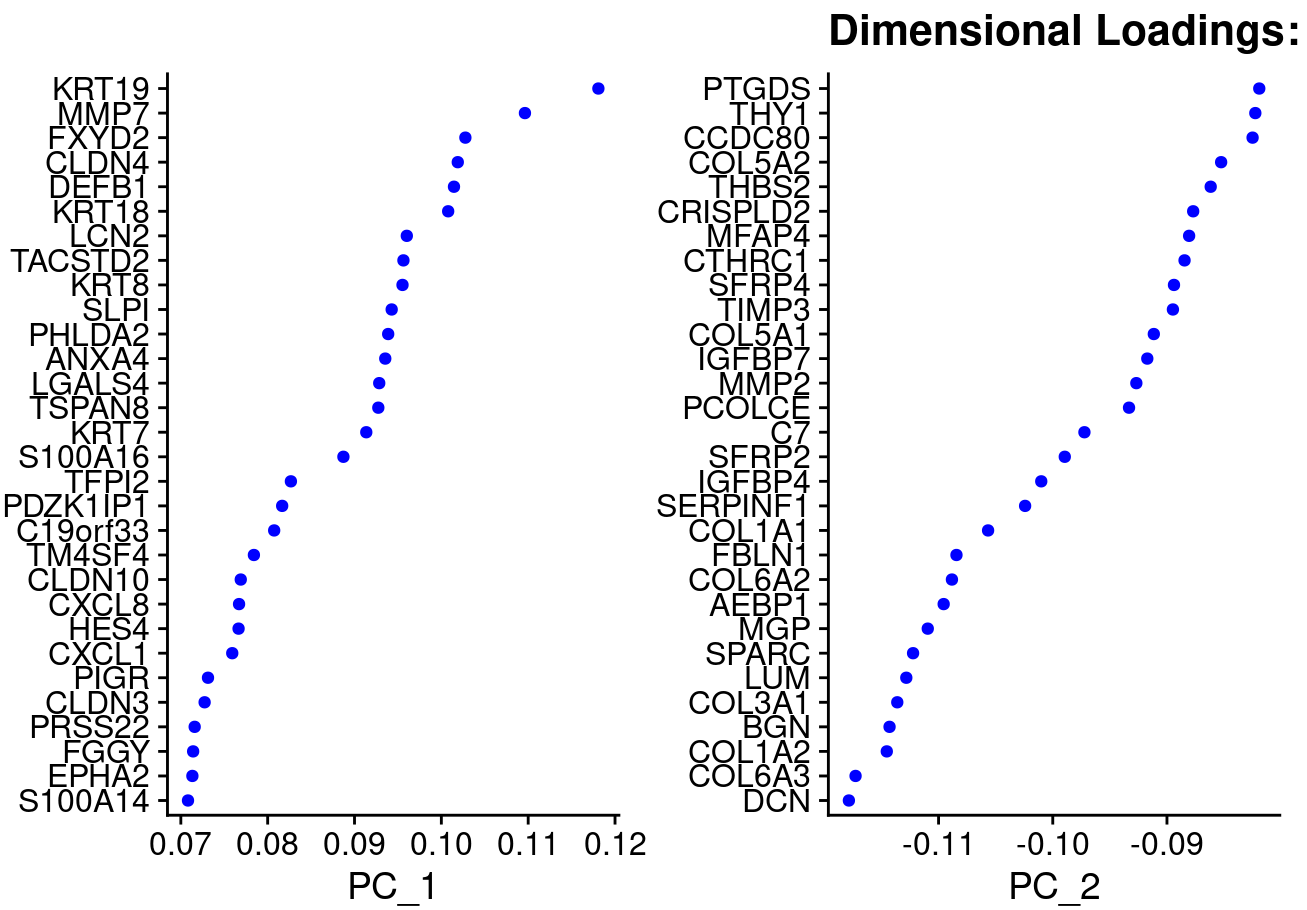
```
##
## [[2]]
```


Dimensional Loadings:
TYROBP
VIM
C1QA
FCER1G
APOC1
C1QB
GPNMB
APOE
CD81
MRC1
CTSS
TREM2
TMEM176B
CYBB
GLUL
C1QC
ITGB2
FTL
FBP1
MS4A7
LAPTM5
MS4A4A
AIF1
MARCO
FN1
SPI1
CTSB
KRT19
EPCAM
TFF1
-0.04
0.00
0.04
PC_1
COL4A1
IGFBP7
COL4A2
CALD1
MAP1B
COL18A1
SPARC
BGN
LAMA4
C11orf96
NID1
MCAM
THY1
ADGRF5
GNG11
MGP
CNN3
LAMB1
RBPMS
FERMT2
LHFPL6
CAV1
FSTL1
CYR61
COX7A1
COL3A1
DLC1
DCN
COL1A2
PDLIM3
0.08
0.09
0.10
0.11
PC_2

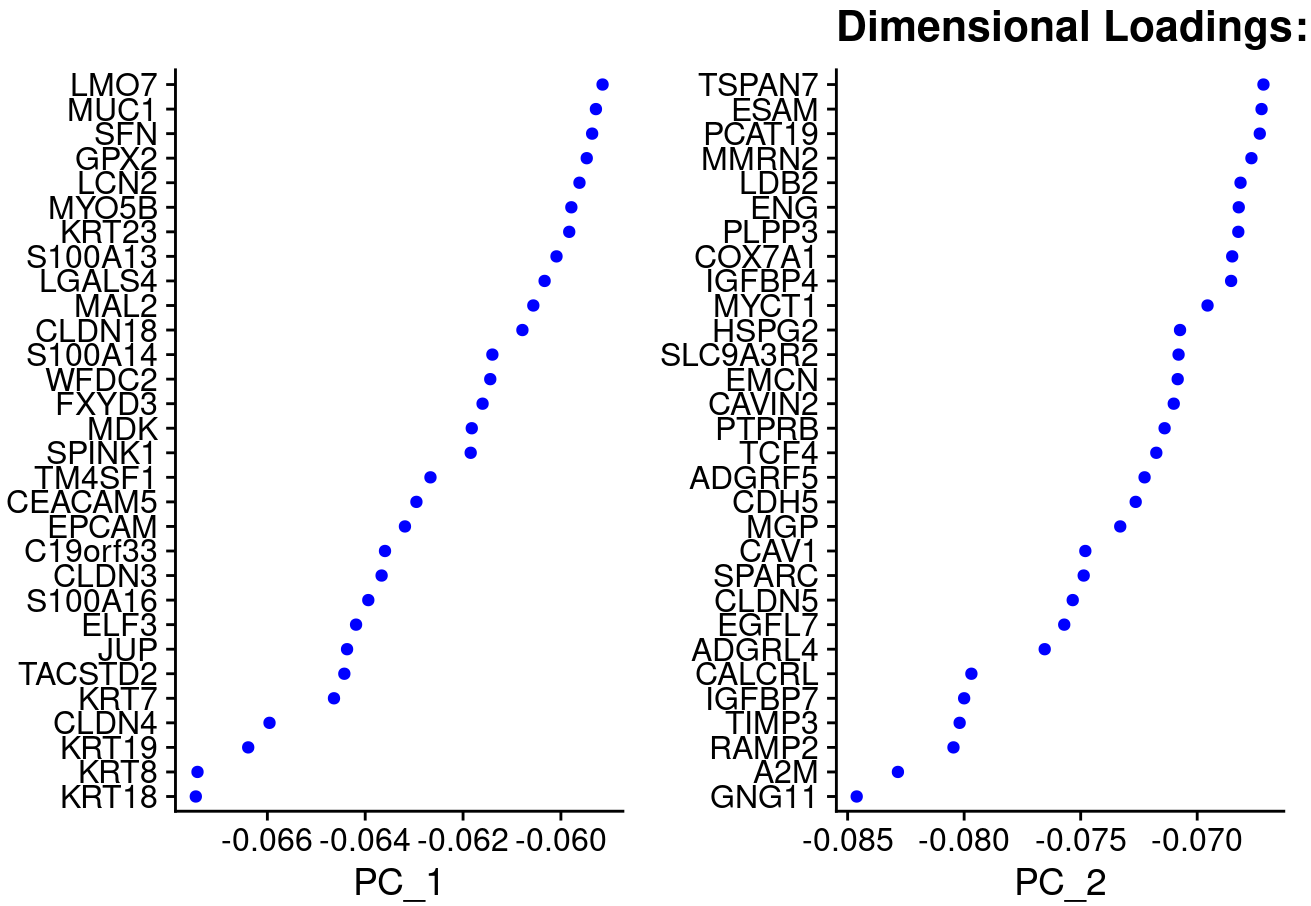
```
##
## [[3]]
```


Dimensional Loadings: (
CALD1
COL1A2
COL1A1
DCN
COL3A1
BGN
COL6A2
SPARC
LUM
IGFBP7
COL5A2
AEBP1
COL6A3
NNMT
CTHRC1
THBS2
FSTL1
TPM2
TPM1
CDH11
C11orf96
COL5A1
MYL9
COL6A1
CTGF
FBN1
TAGLN
THY1
CCDC80
PALLD
0.06250.06500.06750.07000.0725
PC_1
CLDN4
EPCAM
TSPAN1
C19orf33
LCN2
KRT7
MUC1
WFDC2
MAL2
KRT8
KRT23
MYO5B
CEACAM6
CLDN3
ELF3
CEACAM5
RAB25
CDH1
PKP3
TACSTD2
SMIM22
PRSS8
MYH14
JUP
TMPRSS2
LIPH
KRT19
SFN
TFF3
S100P
0.070
0.075
0.080
PC_2

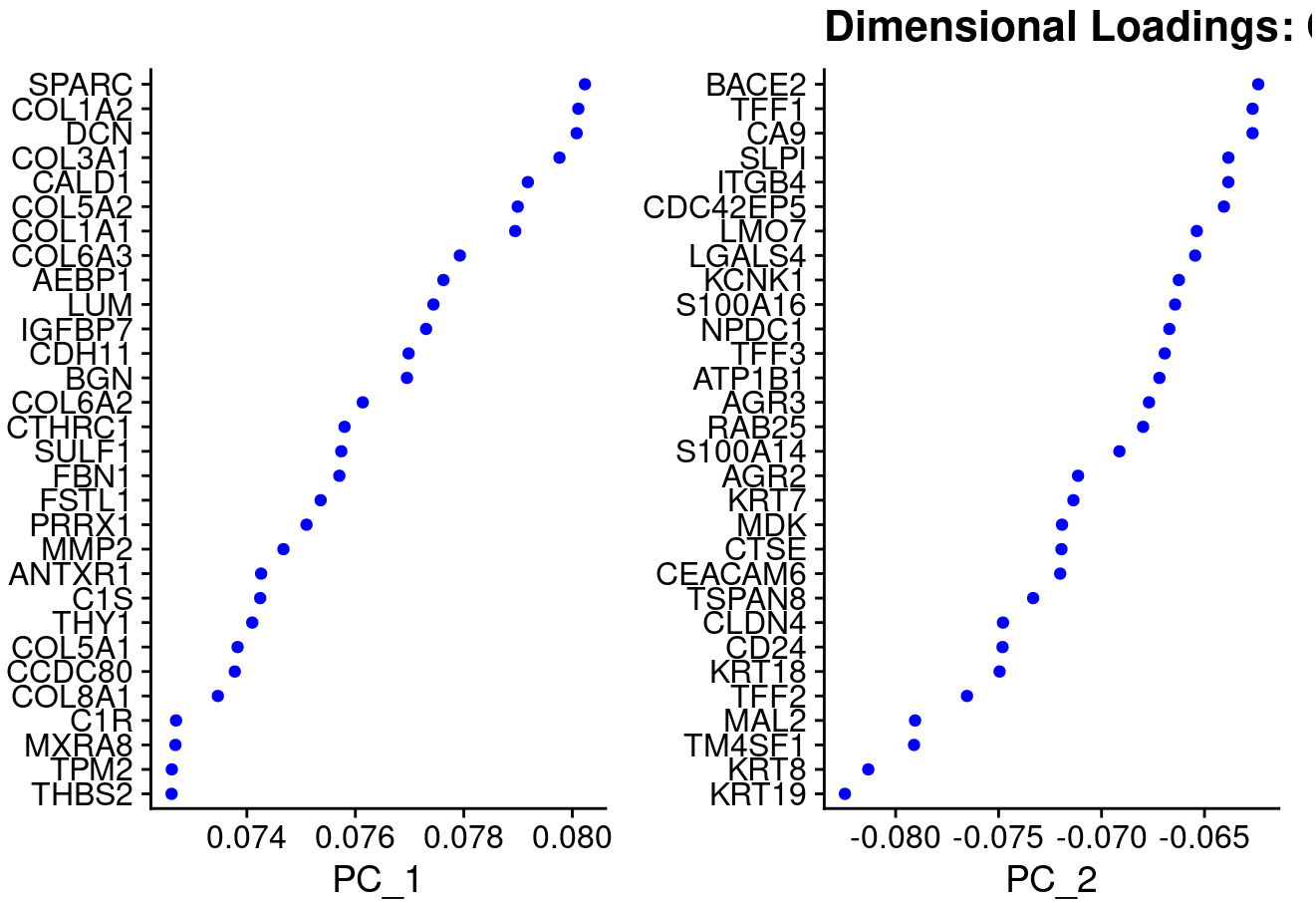
```
##
## [[4]]
```


Dimensional Loadings:
KRT19
MMP7
FXYD2
CLDN4
DEFB1
KRT18
LCN2
TACSTD2
KRT8
SLPI
PHLDA2
ANXA4
LGALS4
TSPAN8
KRT7
S100A16
TFPI2
PDZK1IP1
C19orf33
TM4SF4
CLDN10
CXCL8
HES4
CXCL1
PIGR
CLDN3
PRSS22
FGGY
EPHA2
S100A14
0.07
0.08
0.09
0.10
0.11
0.12
PC_1
PTGDS
THY1
CCDC80
COL5A2
THBS2
CRISPLD2
MFAP4
CTHRC1
SFRP4
TIMP3
COL5A1
IGFBP7
MMP2
PCOLCE
C7
SFRP2
IGFBP4
SERPINF1
COL1A1
FBLN1
COL6A2
AEBP1
MGP
SPARC
LUM
COL3A1
BGN
COL1A2
COL6A3
DCN
-0.11
-0.10
-0.09
PC_2

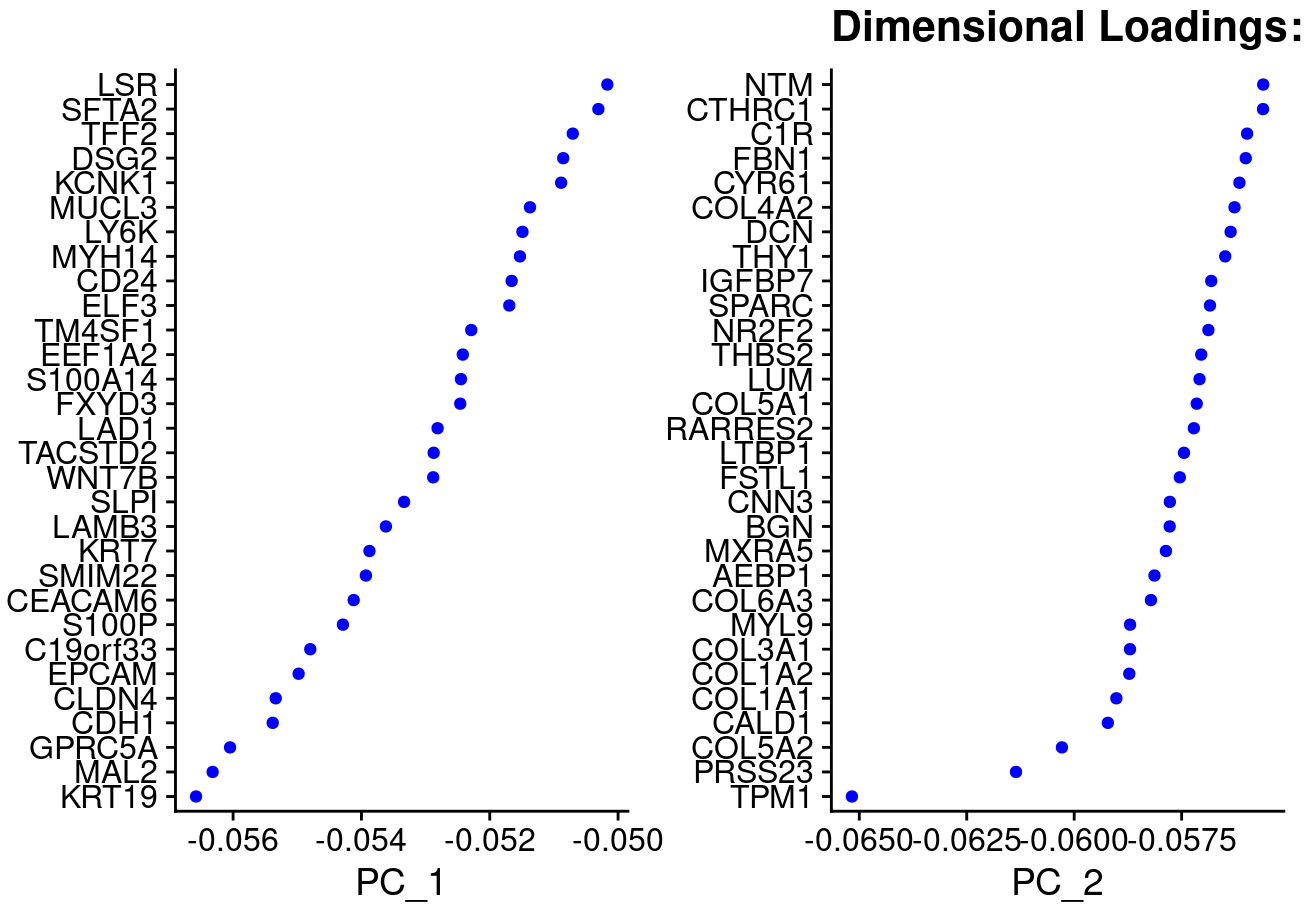
```
##
## [[5]]
```


Dimensional Loadings:
LMO7
MUC1
SFN
GPX2
LCN2
MYO5B
KRT23
S100A13
LGALS4
MAL2
CLDN18
S100A14
WFDC2
FXYD3
MDK
SPINK1
TM4SF1
CEACAM5
EPCAM
C19orf33
CLDN3
S100A16
ELF3
JUP
TACSTD2
KRT7
CLDN4
KRT19
KRT8
KRT18
-0.066 -0.064 -0.062 -0.060
PC_1
TSPAN7
ESAM
PCAT19
MMRN2
LDB2
ENG
PLPP3
COX7A1
IGFBP4
MYCT1
HSPG2
SLC9A3R2
EMCN
CAVIN2
PTPRB
TCF4
ADGRF5
CDH5
MGP
CAV1
SPARC
CLDN5
EGFL7
ADGRL4
CALCRL
IGFBP7
TIMP3
RAMP2
A2M
GNG11
-0.085 -0.080 -0.075 -0.070
PC_2

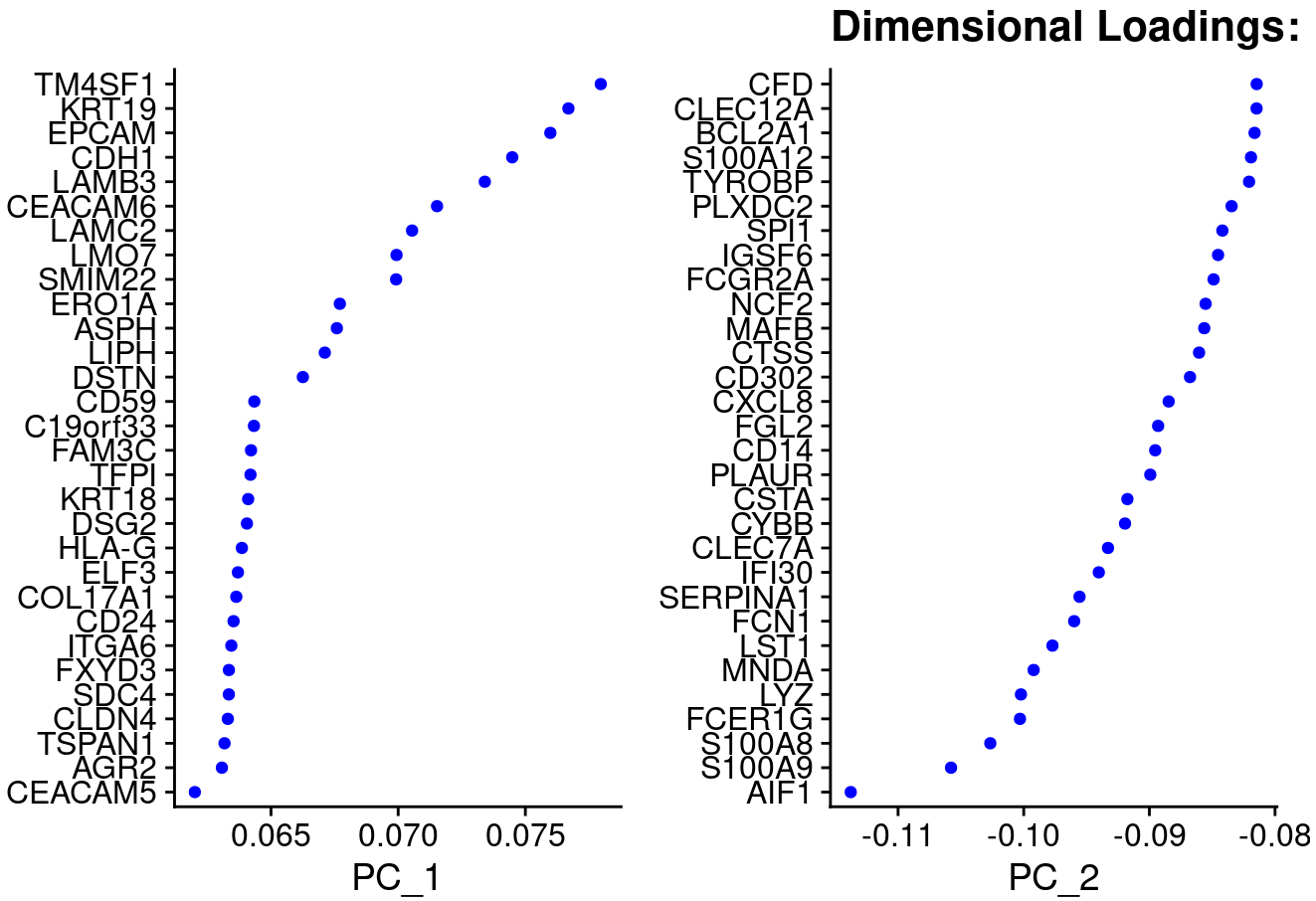
```
##
## [[6]]
```


Dimensional Loadings:
SPARC
COL1A2
DCN
COL3A1
CALD1
COL5A2
COL1A1
COL6A3
AEBP1
LUM
IGFBP7
CDH11
BGN
COL6A2
CTHRC1
SULF1
FBN1
FSTL1
PRRX1
MMP2
ANTXR1
C1S
THY1
COL5A1
CCDC80
COL8A1
C1R
MXRA8
TPM2
THBS2
0.074
0.076
0.078
0.080
PC_1
BACE2
TFF1
CA9
SLPI
ITGB4
CDC42EP5
LMO7
LGALS4
KCNK1
S100A16
NPDC1
TFF3
ATP1B1
AGR3
RAB25
S100A14
AGR2
KRT7
MDK
CTSE
CEACAM6
TSPAN8
CLDN4
CD24
KRT18
TFF2
MAL2
TM4SF1
KRT8
KRT19
-0.080
-0.075
-0.070
-0.065
PC_2

```
##
## [[7]]
```


Dimensional Loadings:
LSR
SFTA2
TFF2
DSG2
KCNK1
MUCL3
LY6K
MYH14
CD24
ELF3
TM4SF1
EEF1A2
S100A14
FXYD3
LAD1
TACSTD2
WNT7B
SLPI
LAMB3
KRT7
SMIM22
CEACAM6
S100P
C19orf33
EPCAM
CLDN4
CDH1
GPRC5A
MAL2
KRT19
-0.056
-0.054
-0.052
-0.050
PC_1
NTM
CTHRC1
C1R
FBN1
CYR61
COL4A2
DCN
THY1
IGFBP7
SPARC
NR2F2
THBS2
LUM
COL5A1
RARRES2
LTBP1
FSTL1
CNN3
BGN
MXRA5
AEBP1
COL6A3
MYL9
COL3A1
COL1A2
COL1A1
CALD1
COL5A2
PRSS23
TPM1
-0.0650
-0.0625
-0.0600
-0.0575
PC_2

```
##
## [[8]]
```



Determine optimal PCs and an optimal pK:

```
pc_nums <- lapply(pca_samples, function(obj) {
  sdev      <- obj[["pca"]]@stdev
  pct       <- sdev / sum(sdev) * 100
  cum_pct   <- cumsum(pct)
  c1        <- which(cum_pct > 90 & pct < 5)[1]
  c2        <- sort(which(diff(pct) > 0.1), decreasing=TRUE)[1] + 1
  min(c(c1,c2), length(sdev))
})

pk_values <- 0.09
```

Doublet Finder:

```
## Tried to use DoubletFinder, but didn't work out [package bugs :(] . So using scDblFinder:
library(scDblFinder)
library(SingleCellExperiment)

sce_list <- lapply(pca_samples, function(seu){
  # Convert THIS Seurat object to an SCE
  sce <- as.SingleCellExperiment(seu)
  # Run scDblFinder
  sce <- scDblFinder(sce)
  # Return the annotated SCE
  sce
})
```

```
## Creating ~6528 artificial doublets...
```

```
## Dimensional reduction
```

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

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

```
## iter=0, 1083 cells excluded from training.
```

```
## iter=1, 1240 cells excluded from training.
```

```
## iter=2, 1267 cells excluded from training.
```

```
## Threshold found:0.473
```

```
## 909 (11.1%) doublets called
```

```
## Creating ~6798 artificial doublets...
```

```
## Dimensional reduction
```

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

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

```
## iter=0, 1252 cells excluded from training.
```

```
## iter=1, 1455 cells excluded from training.
```

```
## iter=2, 1506 cells excluded from training.
```

```
## Threshold found:0.527
```

```
## 1094 (12.9%) doublets called
```

```
## Creating ~9079 artificial doublets...
```

```
## Dimensional reduction
```

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

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

```
## iter=0, 1748 cells excluded from training.
```

```
## iter=1, 1839 cells excluded from training.
```

```
## iter=2, 1846 cells excluded from training.
```

```
## Threshold found:0.414
```

```
## 1367 (12%) doublets called
```

```
## Creating ~5119 artificial doublets...
```

```
## Dimensional reduction
```

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

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

```
## iter=0, 589 cells excluded from training.
```

```
## iter=1, 520 cells excluded from training.
```

```
## iter=2, 437 cells excluded from training.
```

```
## Threshold found:0.39
```

```
## 350 (5.5%) doublets called
```

```
## Creating ~7280 artificial doublets...
```

```
## Dimensional reduction
```

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

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

```
## iter=0, 1371 cells excluded from training.
```

```
## iter=1, 1461 cells excluded from training.
```

```
## iter=2, 1511 cells excluded from training.
```

```
## Threshold found:0.419
```

```
## 1158 (12.7%) doublets called
```

```
## Creating ~6541 artificial doublets...
```

```
## Dimensional reduction
```

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

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

```
## iter=0, 1330 cells excluded from training.
```

```
## iter=1, 1487 cells excluded from training.
```

```
## iter=2, 1451 cells excluded from training.
```

```
## Threshold found:0.409
```

```
## 962 (11.8%) doublets called
```

```
## Creating ~6247 artificial doublets...
```

```
## Dimensional reduction
```

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

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

```
## iter=0, 1433 cells excluded from training.
```

```
## iter=1, 1511 cells excluded from training.
```

```
## iter=2, 1512 cells excluded from training.
```

```
## Threshold found:0.437
```

```
## 900 (11.5%) doublets called
```

```
## Creating ~1500 artificial doublets...
```

```
## Dimensional reduction
```

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

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

```
## iter=0, 150 cells excluded from training.
```

```
## iter=1, 155 cells excluded from training.
```

```
## iter=2, 157 cells excluded from training.
```

```
## Threshold found:0.671
```

```
## 68 (4.8%) doublets called
```

```
# Now each element of sce_list has scDblFinder.score & .class
# To inspect sample 1:
table(sce_list[[1]]$scDblFinder.class)
```

```
##
## singlet doublet
##    7251     909
```

```
# Tabulate:
doublet_summary <- lapply(names(sce_list), function(samp) {
  sce <- sce_list[[samp]]
  tbl <- table(sce$scDblFinder.class)
  data.frame(
    sample   = samp,
    class    = names(tbl),
    count    = as.integer(tbl),
    fraction = as.numeric(prop.table(tbl)),
    row.names = NULL
  )
}) %>% bind_rows()

print(doublet_summary)
```

```
##    sample   class count   fraction
## 1   C1_YF singlet  7251 0.88860294
## 2   C1_YF doublet   909 0.11139706
## 3   C1_ZY singlet  7403 0.87124868
## 4   C1_ZY doublet  1094 0.12875132
## 5   C2_YF singlet  9981 0.87953824
## 6   C2_YF doublet  1367 0.12046176
## 7   C2_ZC singlet  6048 0.94529540
## 8   C2_ZC doublet   350 0.05470460
## 9   C2_ZY singlet  7942 0.87274725
## 10  C2_ZY doublet  1158 0.12725275
## 11  C3_YF singlet  7214 0.88233855
## 12  C3_YF doublet   962 0.11766145
## 13  C3_ZY singlet  6908 0.88473361
## 14  C3_ZY doublet   900 0.11526639
## 15  C4_ZY singlet  1338 0.95163585
## 16  C4_ZY doublet    68 0.04836415
```

Cluster and visualize:

```
obj_with_clusters <- lapply(pca_samples, function(obj) {
  obj <- FindNeighbors(obj, dims = 1:20)
  obj <- FindClusters(obj, resolution = 0.5)
  obj <- RunUMAP(obj, dims = 1:20)
  return(obj)
})
```

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

```
## Computing SNN
```

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 8160
## Number of edges: 258430
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.8670
## Number of communities: 14
## Elapsed time: 0 seconds
```

```
## Warning: The default method for RunUMAP has changed from calling Python UMAP via reticulate t
o the R-native UWOT using the cosine metric
## To use Python UMAP via reticulate, set umap.method to 'umap-learn' and metric to 'correlatio
n'
## This message will be shown once per session
```

```
## 22:03:26 UMAP embedding parameters a = 0.9922 b = 1.112
```

```
## 22:03:26 Read 8160 rows and found 20 numeric columns
```

```
## 22:03:26 Using Annoy for neighbor search, n_neighbors = 30
```

```
## 22:03:26 Building Annoy index with metric = cosine, n_trees = 50
```

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

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

```
## **************************************************|
## 22:03:27 Writing NN index file to temp file /scratch/5516304.1.academic/RtmpTRHoBg/file2da0b1
6988e1de
## 22:03:27 Searching Annoy index using 1 thread, search_k = 3000
## 22:03:29 Annoy recall = 100%
## 22:03:30 Commencing smooth kNN distance calibration using 1 thread with target n_neighbors =
30
## 22:03:31 Initializing from normalized Laplacian + noise (using RSpectra)
## 22:03:32 Commencing optimization for 500 epochs, with 336312 positive edges
## 22:03:40 Optimization finished
## Computing nearest neighbor graph
## Computing SNN
```

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 8497
## Number of edges: 297356
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.8987
## Number of communities: 16
## Elapsed time: 0 seconds
```

```
## 22:03:43 UMAP embedding parameters a = 0.9922 b = 1.112
## 22:03:43 Read 8497 rows and found 20 numeric columns
## 22:03:43 Using Annoy for neighbor search, n_neighbors = 30
## 22:03:43 Building Annoy index with metric = cosine, n_trees = 50
## 0%   10   20   30   40   50   60   70   80   90   100%
## [----|----|----|----|----|----|----|----|----|----|
## **************************************************|
## 22:03:44 Writing NN index file to temp file /scratch/5516304.1.academic/RtmpTRHoBg/file2da0b1
5e7fca72
## 22:03:44 Searching Annoy index using 1 thread, search_k = 3000
## 22:03:46 Annoy recall = 100%
## 22:03:47 Commencing smooth kNN distance calibration using 1 thread with target n_neighbors =
30
## 22:03:49 Initializing from normalized Laplacian + noise (using RSpectra)
## 22:03:49 Commencing optimization for 500 epochs, with 358890 positive edges
## 22:03:57 Optimization finished
## Computing nearest neighbor graph
## Computing SNN
```

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 11348
## Number of edges: 407997
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.9005
## Number of communities: 16
## Elapsed time: 1 seconds
```

```
## 22:04:02 UMAP embedding parameters a = 0.9922 b = 1.112
## 22:04:02 Read 11348 rows and found 20 numeric columns
## 22:04:02 Using Annoy for neighbor search, n_neighbors = 30
## 22:04:02 Building Annoy index with metric = cosine, n_trees = 50
## 0%   10   20   30   40   50   60   70   80   90   100%
## [----|----|----|----|----|----|----|----|----|----|
## **************************************************|
## 22:04:03 Writing NN index file to temp file /scratch/5516304.1.academic/RtmpTRHoBg/file2da0b1
224dbc4a
## 22:04:03 Searching Annoy index using 1 thread, search_k = 3000
## 22:04:06 Annoy recall = 100%
## 22:04:07 Commencing smooth kNN distance calibration using 1 thread with target n_neighbors =
30
## 22:04:09 Initializing from normalized Laplacian + noise (using RSpectra)
## 22:04:09 Commencing optimization for 200 epochs, with 500540 positive edges
## 22:04:14 Optimization finished
## Computing nearest neighbor graph
## Computing SNN
```

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 6398
## Number of edges: 246149
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.9200
## Number of communities: 15
## Elapsed time: 0 seconds
```

```
## 22:04:16 UMAP embedding parameters a = 0.9922 b = 1.112
## 22:04:16 Read 6398 rows and found 20 numeric columns
## 22:04:16 Using Annoy for neighbor search, n_neighbors = 30
## 22:04:16 Building Annoy index with metric = cosine, n_trees = 50
## 0%   10   20   30   40   50   60   70   80   90   100%
## [----|----|----|----|----|----|----|----|----|----|
## **************************************************|
## 22:04:17 Writing NN index file to temp file /scratch/5516304.1.academic/RtmpTRHoBg/file2da0b1
1c8f2c37
## 22:04:17 Searching Annoy index using 1 thread, search_k = 3000
## 22:04:19 Annoy recall = 100%
## 22:04:20 Commencing smooth kNN distance calibration using 1 thread with target n_neighbors =
30
## 22:04:22 Initializing from normalized Laplacian + noise (using RSpectra)
## 22:04:22 Commencing optimization for 500 epochs, with 287608 positive edges
## 22:04:28 Optimization finished
## Computing nearest neighbor graph
## Computing SNN
```

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 9100
## Number of edges: 328099
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.8920
## Number of communities: 15
## Elapsed time: 1 seconds
```

```
## 22:04:32 UMAP embedding parameters a = 0.9922 b = 1.112
## 22:04:32 Read 9100 rows and found 20 numeric columns
## 22:04:32 Using Annoy for neighbor search, n_neighbors = 30
## 22:04:32 Building Annoy index with metric = cosine, n_trees = 50
## 0%   10   20   30   40   50   60   70   80   90   100%
## [----|----|----|----|----|----|----|----|----|----|
## **************************************************|
## 22:04:33 Writing NN index file to temp file /scratch/5516304.1.academic/RtmpTRHoBg/file2da0b1
2d079897
## 22:04:33 Searching Annoy index using 1 thread, search_k = 3000
## 22:04:35 Annoy recall = 100%
## 22:04:36 Commencing smooth kNN distance calibration using 1 thread with target n_neighbors =
30
## 22:04:38 Initializing from normalized Laplacian + noise (using RSpectra)
## 22:04:38 Commencing optimization for 500 epochs, with 397244 positive edges
## 22:04:47 Optimization finished
## Computing nearest neighbor graph
## Computing SNN
```

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 8176
## Number of edges: 287924
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.8976
## Number of communities: 13
## Elapsed time: 0 seconds
```

```
## 22:04:50 UMAP embedding parameters a = 0.9922 b = 1.112
## 22:04:50 Read 8176 rows and found 20 numeric columns
## 22:04:50 Using Annoy for neighbor search, n_neighbors = 30
## 22:04:50 Building Annoy index with metric = cosine, n_trees = 50
## 0%   10   20   30   40   50   60   70   80   90   100%
## [----|----|----|----|----|----|----|----|----|----|
## **************************************************|
## 22:04:51 Writing NN index file to temp file /scratch/5516304.1.academic/RtmpTRHoBg/file2da0b1
13ae7624
## 22:04:51 Searching Annoy index using 1 thread, search_k = 3000
## 22:04:53 Annoy recall = 100%
## 22:04:54 Commencing smooth kNN distance calibration using 1 thread with target n_neighbors =
30
## 22:04:56 Initializing from normalized Laplacian + noise (using RSpectra)
## 22:04:56 Commencing optimization for 500 epochs, with 355170 positive edges
## 22:05:05 Optimization finished
## Computing nearest neighbor graph
## Computing SNN
```

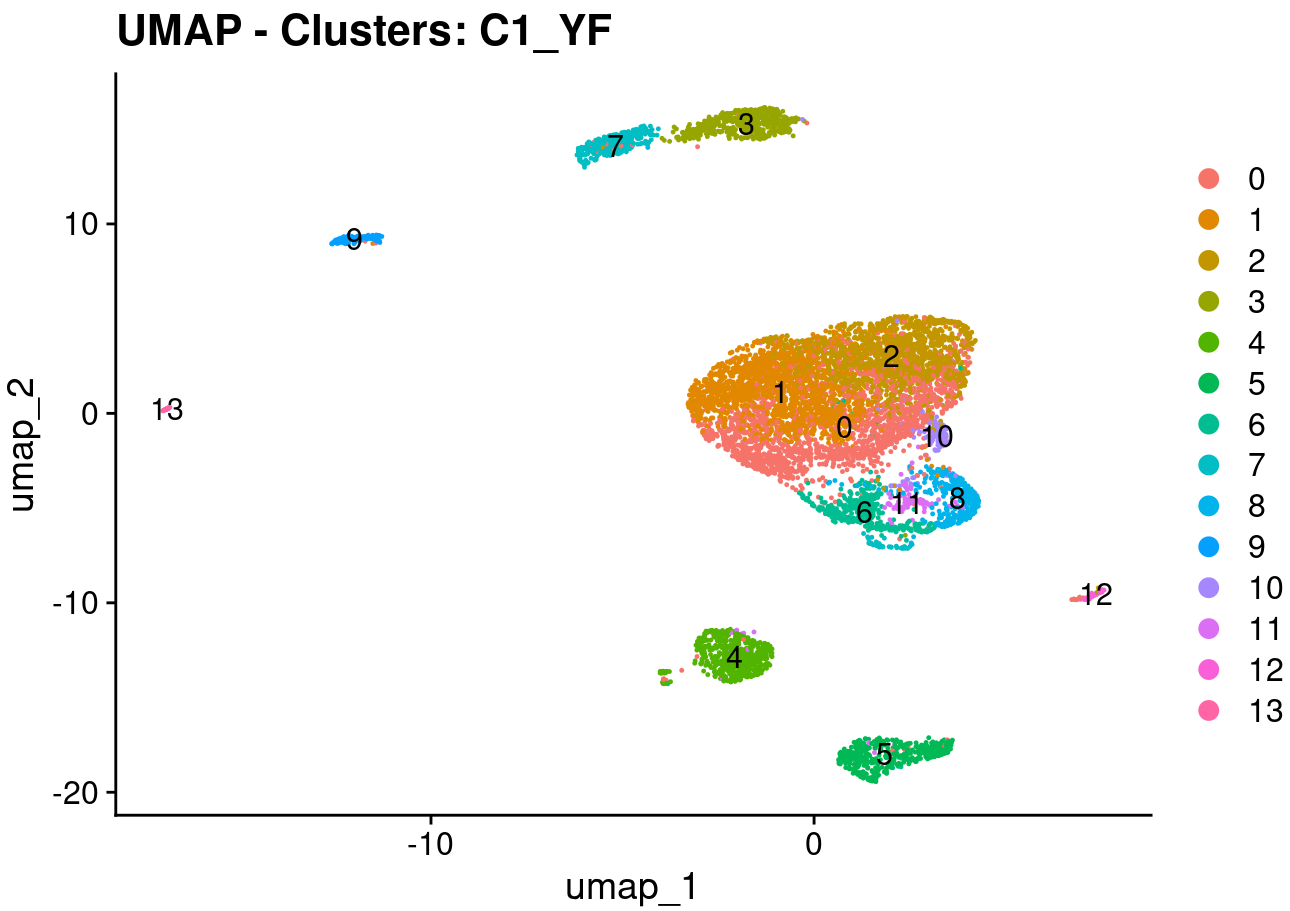
```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 7808
## Number of edges: 278673
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.8986
## Number of communities: 11
## Elapsed time: 0 seconds
```

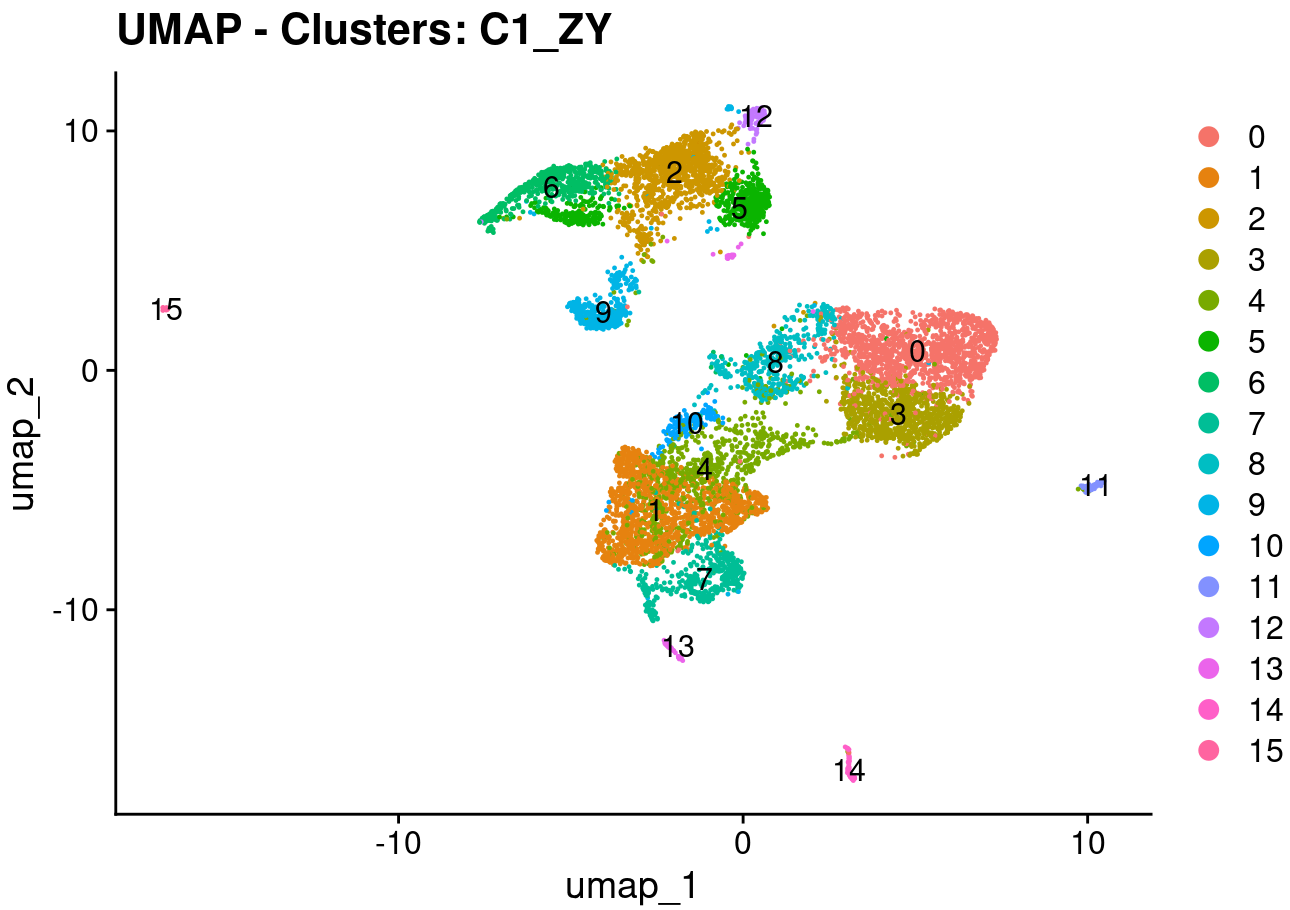
```
## 22:05:07 UMAP embedding parameters a = 0.9922 b = 1.112
## 22:05:07 Read 7808 rows and found 20 numeric columns
## 22:05:07 Using Annoy for neighbor search, n_neighbors = 30
## 22:05:07 Building Annoy index with metric = cosine, n_trees = 50
## 0%   10   20   30   40   50   60   70   80   90   100%
## [----|----|----|----|----|----|----|----|----|----|
## **************************************************|
## 22:05:08 Writing NN index file to temp file /scratch/5516304.1.academic/RtmpTRHoBg/file2da0b1
5e736194
## 22:05:08 Searching Annoy index using 1 thread, search_k = 3000
## 22:05:10 Annoy recall = 100%
## 22:05:11 Commencing smooth kNN distance calibration using 1 thread with target n_neighbors =
30
## 22:05:13 Initializing from normalized Laplacian + noise (using RSpectra)
## 22:05:13 Commencing optimization for 500 epochs, with 330388 positive edges
## 22:05:21 Optimization finished
## Computing nearest neighbor graph
## Computing SNN
```

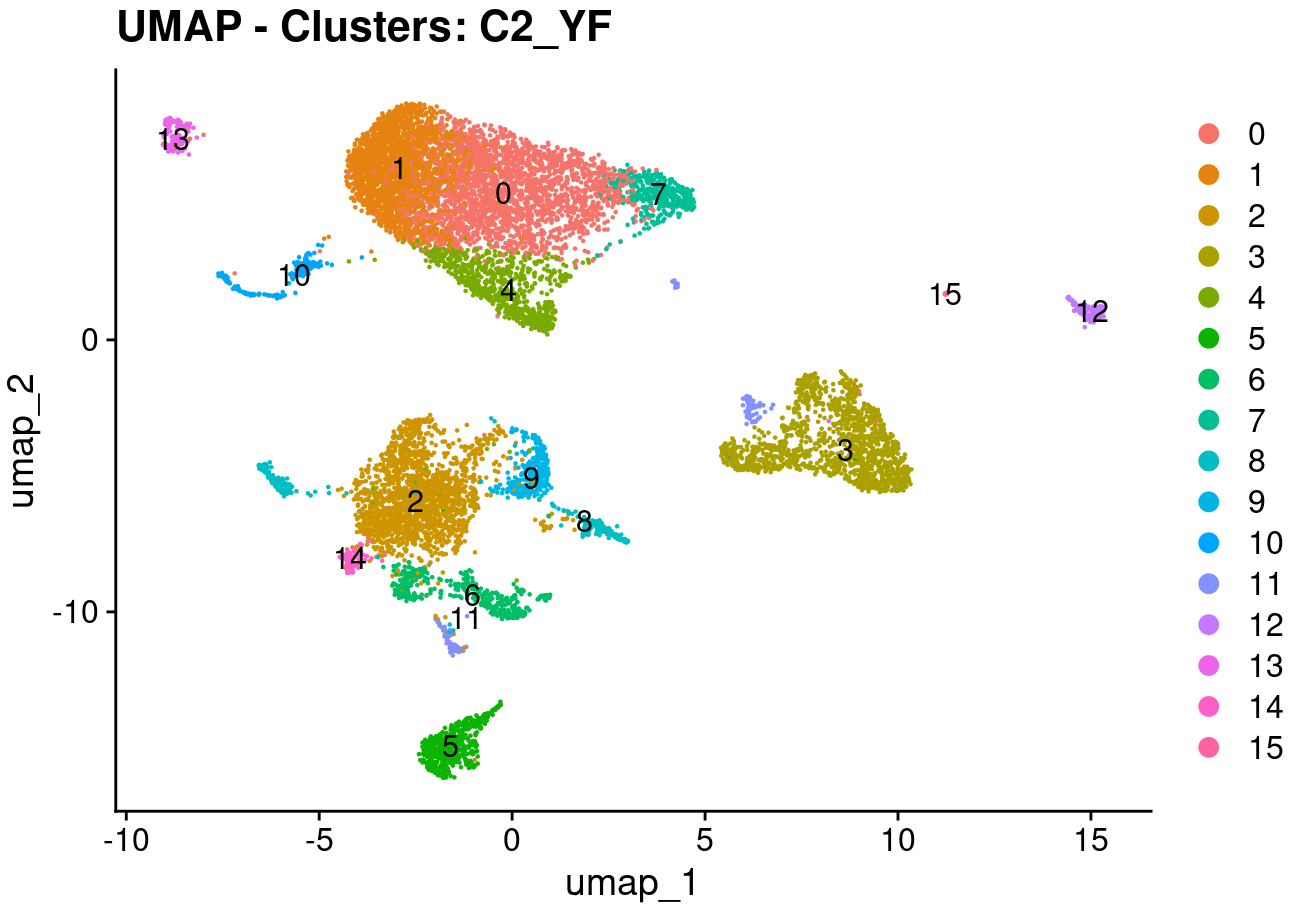
```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 1406
## Number of edges: 46431
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.8657
## Number of communities: 10
## Elapsed time: 0 seconds
```

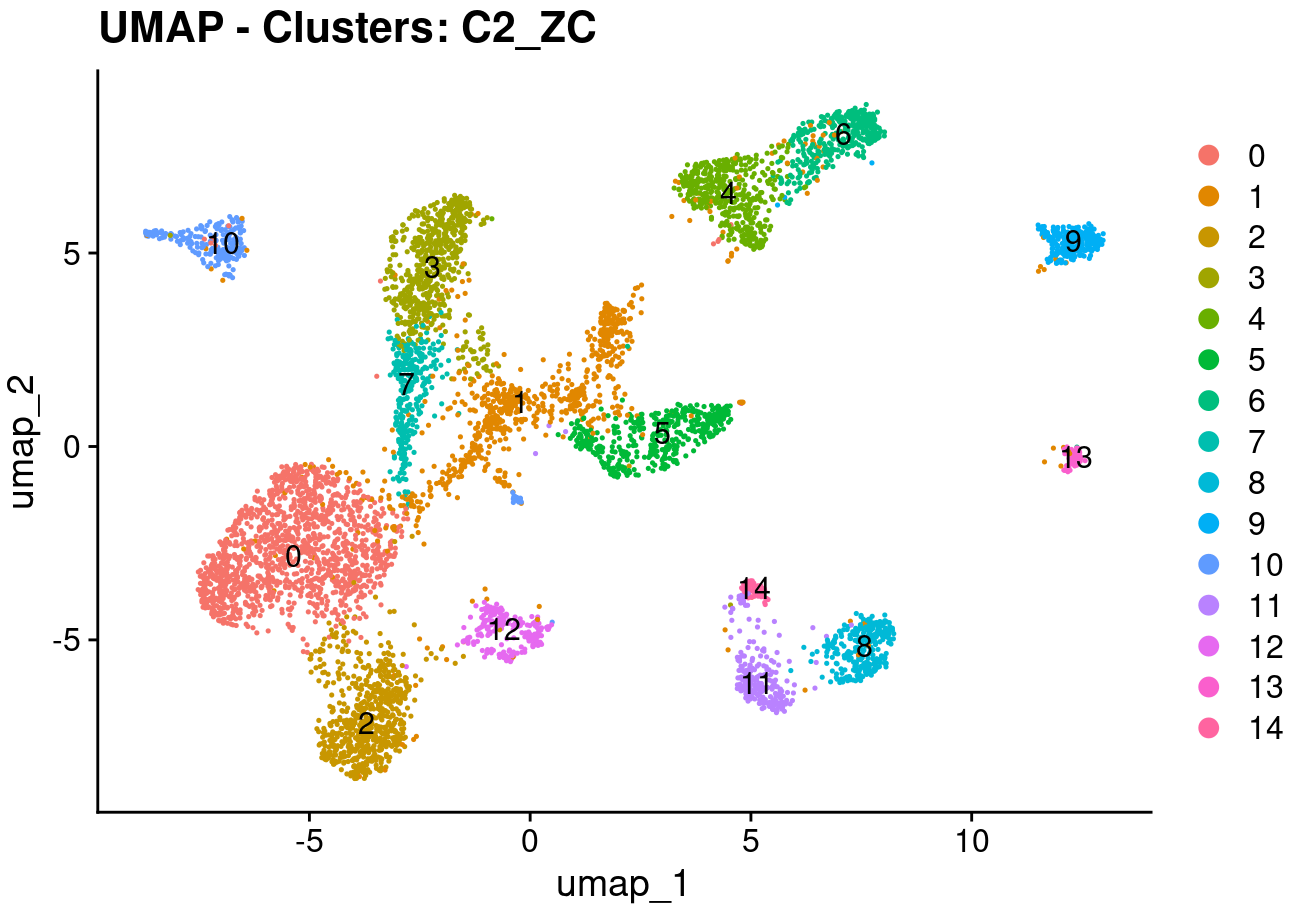
```
## 22:05:21 UMAP embedding parameters a = 0.9922 b = 1.112
## 22:05:21 Read 1406 rows and found 20 numeric columns
## 22:05:21 Using Annoy for neighbor search, n_neighbors = 30
## 22:05:21 Building Annoy index with metric = cosine, n_trees = 50
## 0%   10   20   30   40   50   60   70   80   90   100%
## [----|----|----|----|----|----|----|----|----|----|
## **************************************************|
## 22:05:22 Writing NN index file to temp file /scratch/5516304.1.academic/RtmpTRHoBg/file2da0b1
72f23141
## 22:05:22 Searching Annoy index using 1 thread, search_k = 3000
## 22:05:22 Annoy recall = 100%
## 22:05:23 Commencing smooth kNN distance calibration using 1 thread with target n_neighbors =
30
## 22:05:24 Initializing from normalized Laplacian + noise (using RSpectra)
## 22:05:24 Commencing optimization for 500 epochs, with 54588 positive edges
## 22:05:26 Optimization finished
```

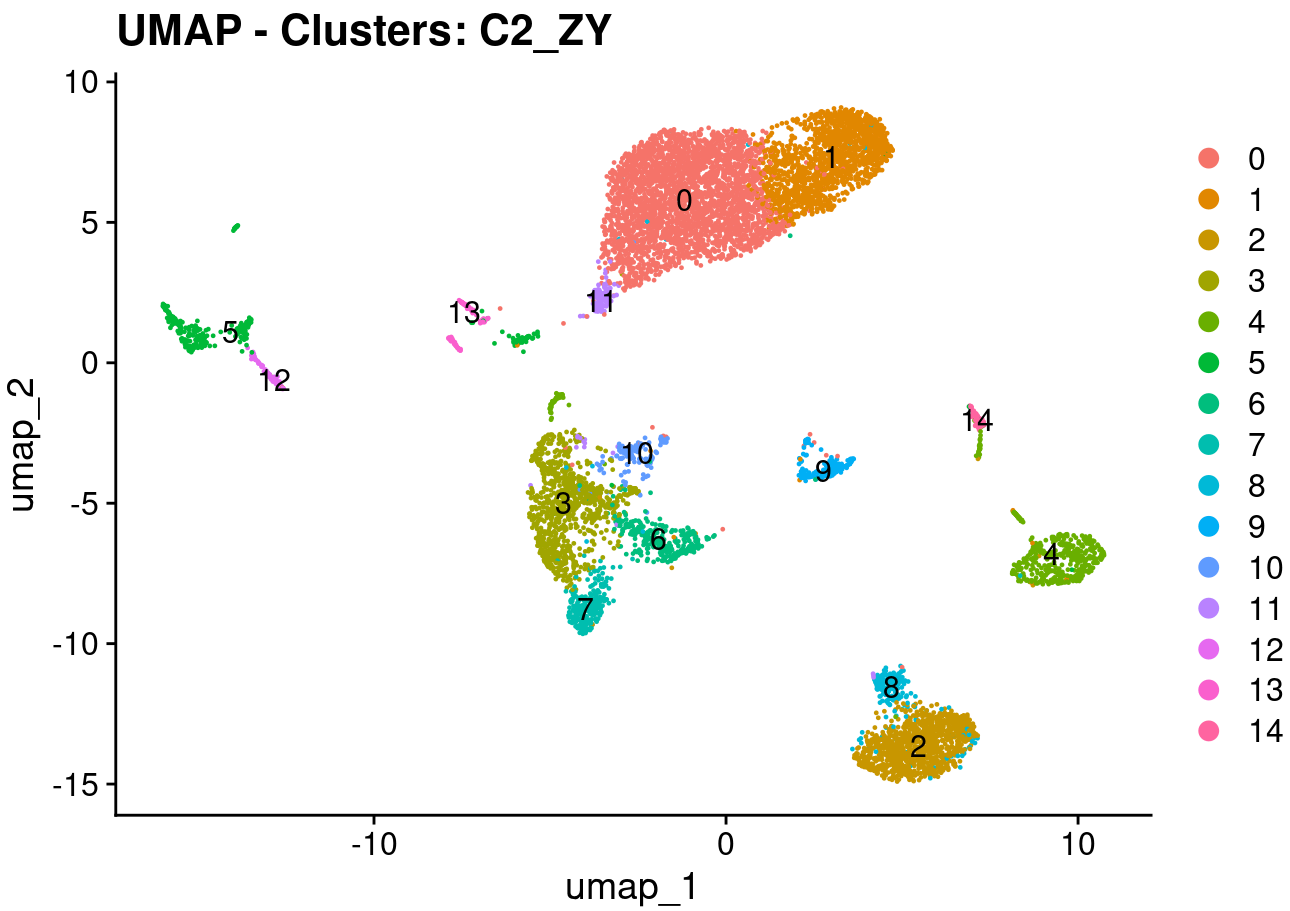
```
# Visualize UMAP and clusters per sample
lapply(names(obj_with_clusters), function(nm) {
  obj <- obj_with_clusters[[nm]]
  print(DimPlot(obj, reduction = "umap", label = TRUE) +
          ggtitle(paste0("UMAP - Clusters: ", nm)))
})
```

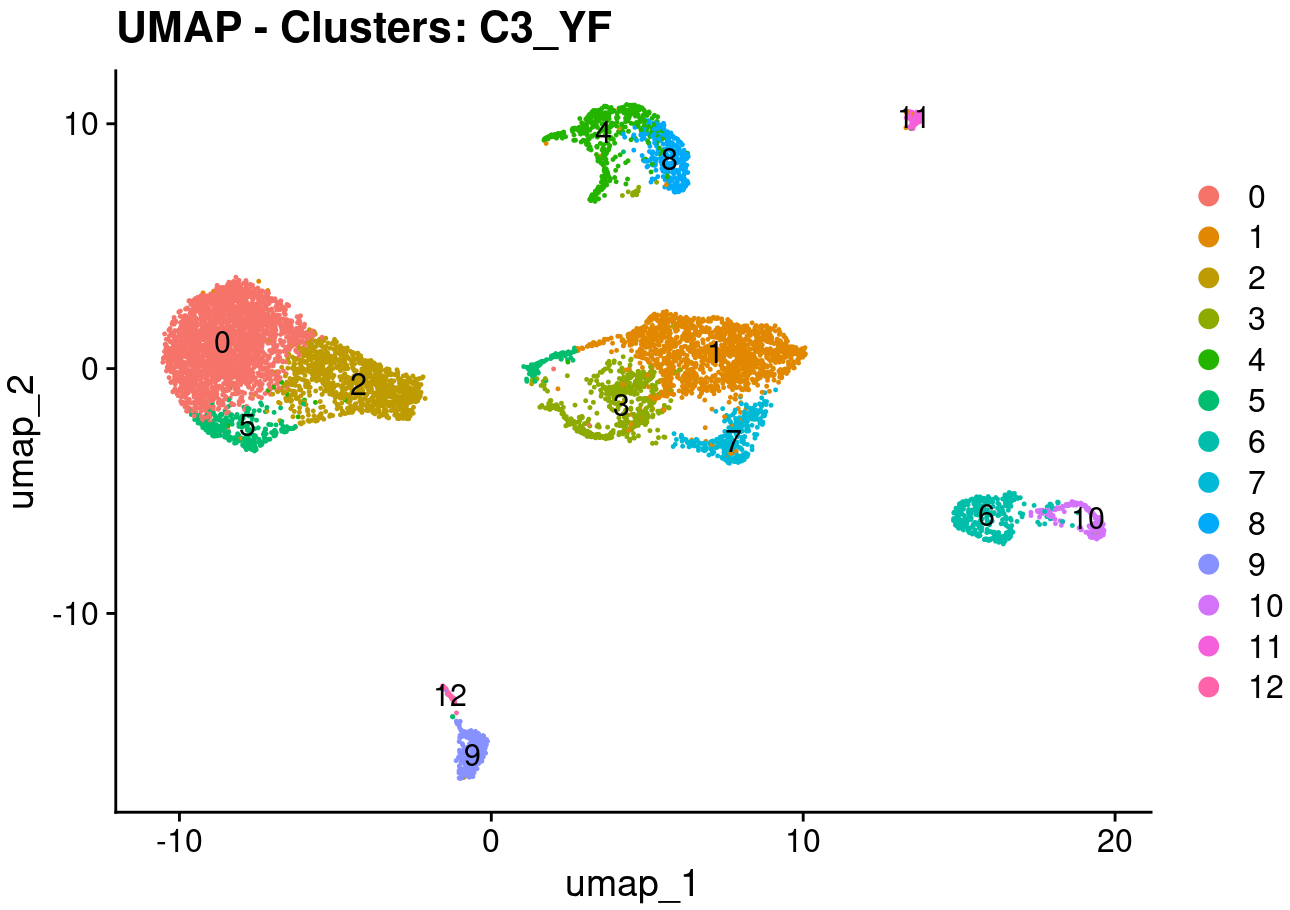

UMAP - Clusters: C1_YF
umap_2
umap_1
10
0
-10
-20
-10
0
3
7
9
2
1
0
10
13
6
11
8
12
4
5
0
1
2
3
4
5
6
7
8
9
10
11
12
13

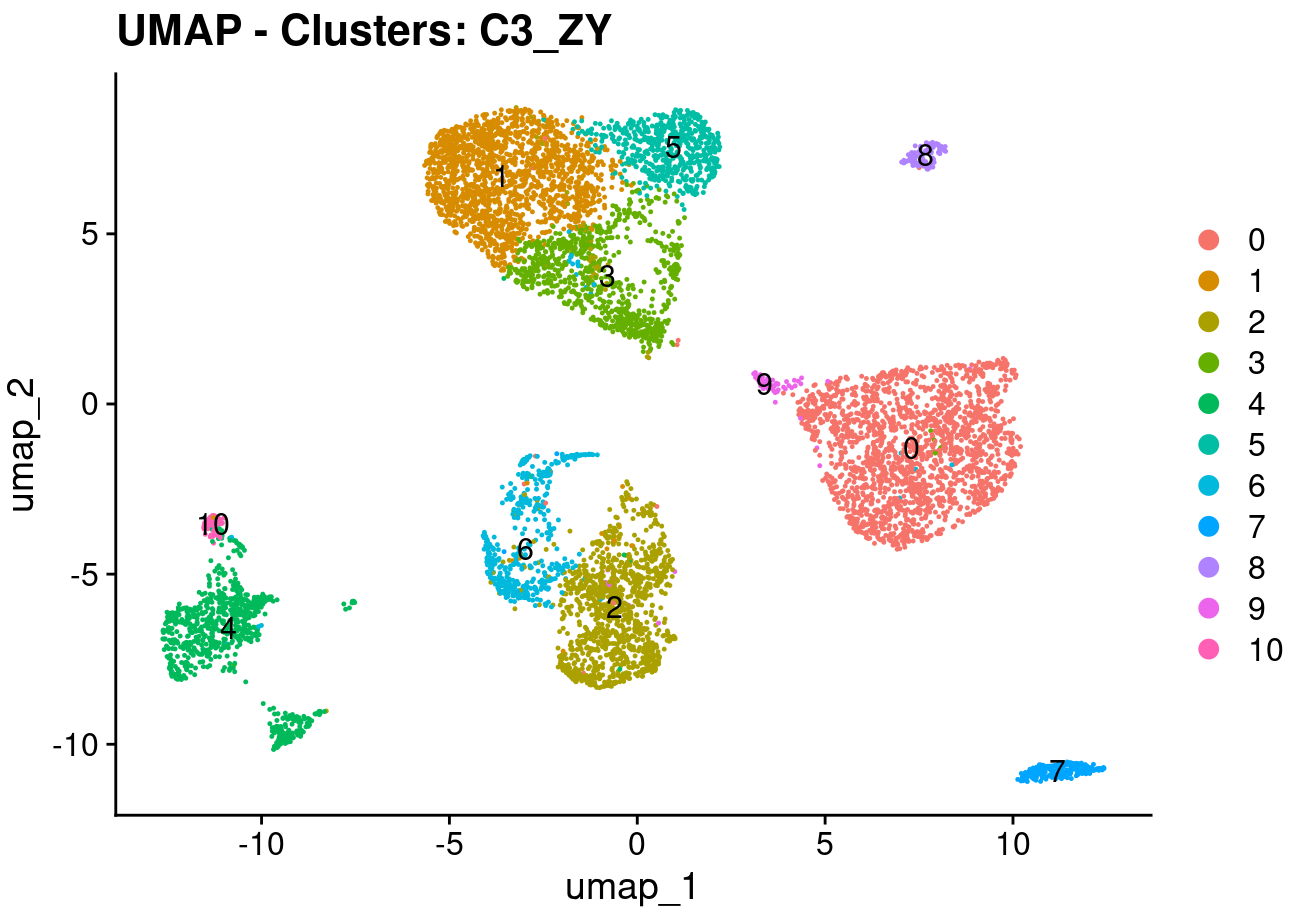

UMAP - Clusters: C1_ZY
umap_2
umap_1
10
0
-10
-10
0
10
12
2
6
5
15
9
0
8
3
10
4
11
1
7
13
14
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

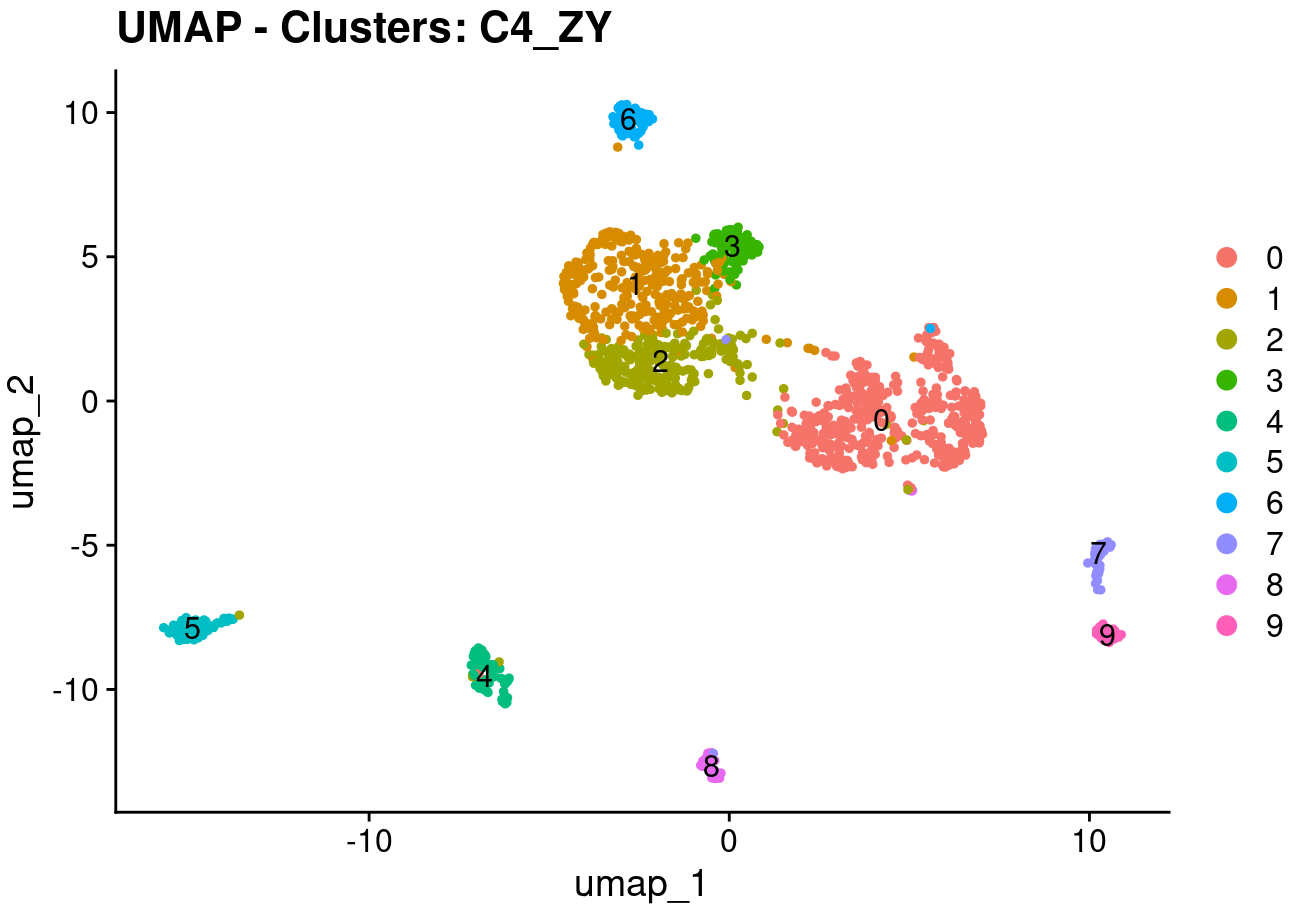

UMAP - Clusters: C2_YF
umap_2
umap_1
-10
-5
0
5
10
15
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15


UMAP - Clusters: C2_ZC
umap_2
umap_1
-5
0
5
10
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14

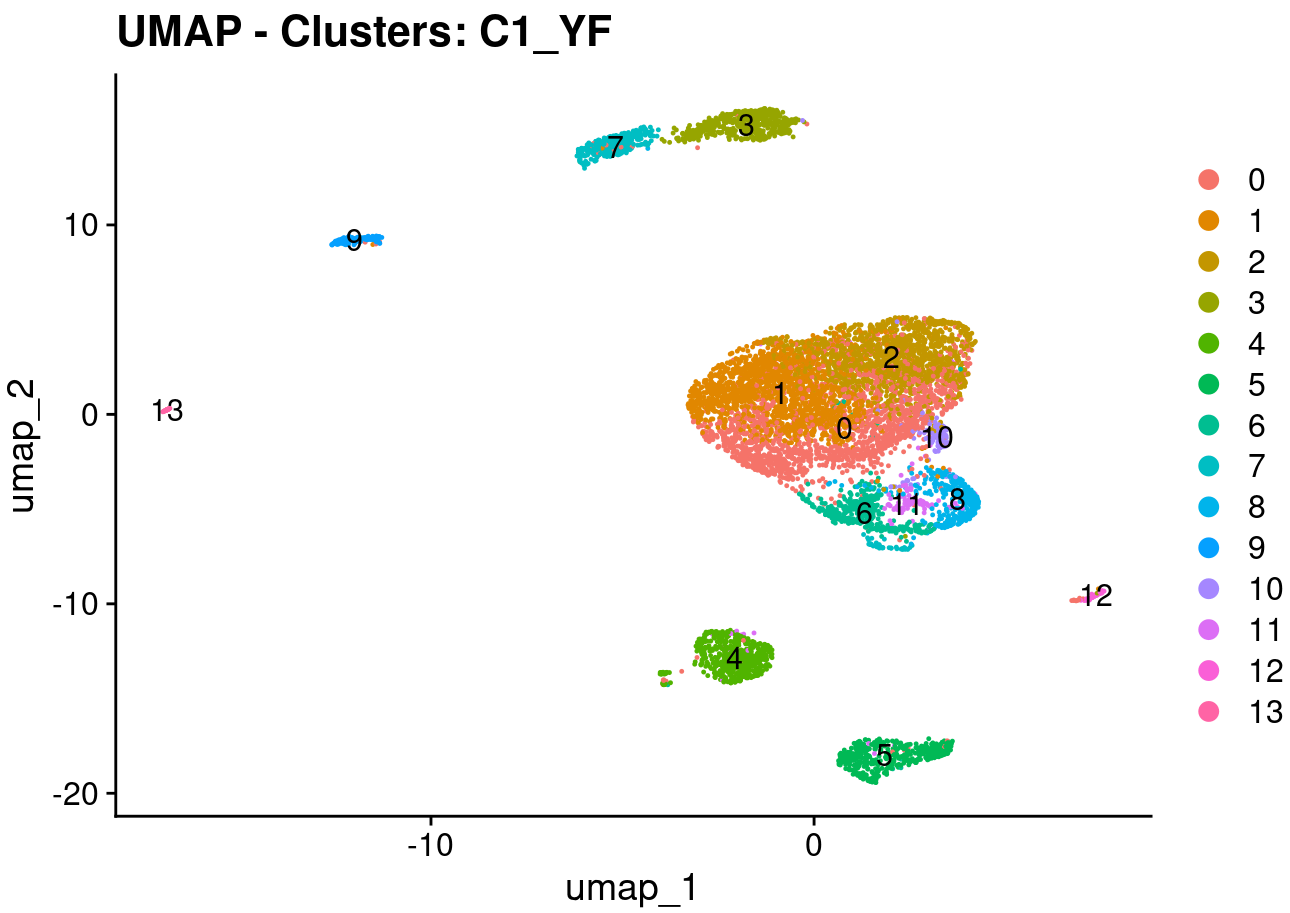

UMAP - Clusters: C2_ZY
umap_2
umap_1
10
5
0
-5
-10
-15
-10
0
10
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14


UMAP - Clusters: C3_YF
umap_2
umap_1
10
0
-10
-10
0
10
20
0
1
2
3
4
5
6
7
8
9
10
11
12

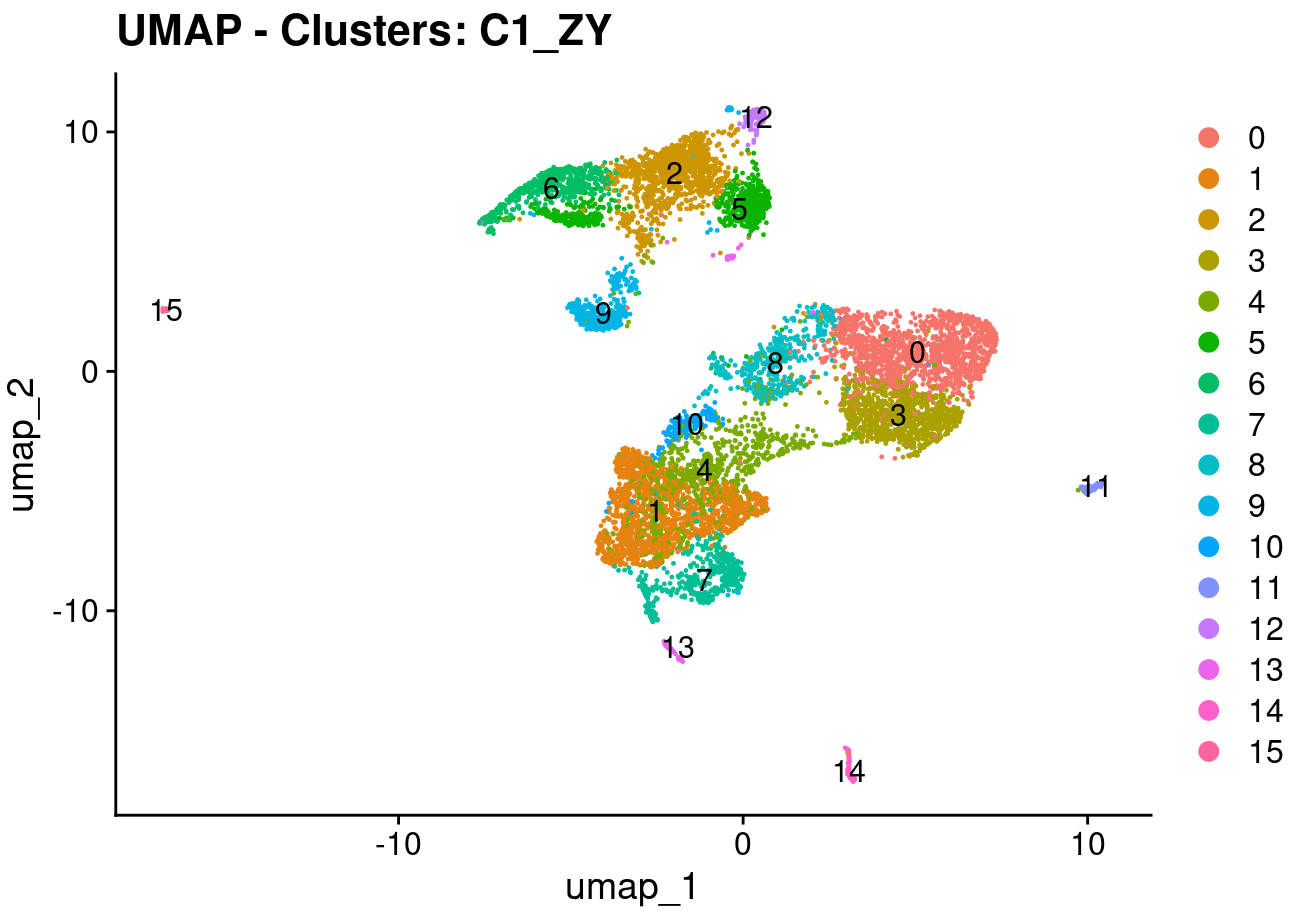

UMAP - Clusters: C3_ZY
umap_2
umap_1
5
0
-5
-10
-10
-5
0
5
10
1
5
8
3
9
0
10
6
2
4
7
0
1
2
3
4
5
6
7
8
9
10


UMAP - Clusters: C4_ZY
umap_2
umap_1
10
5
0
-5
-10
-10
0
10
6
3
1
2
0
7
5
9
4
8
0
1
2
3
4
5
6
7
8
9

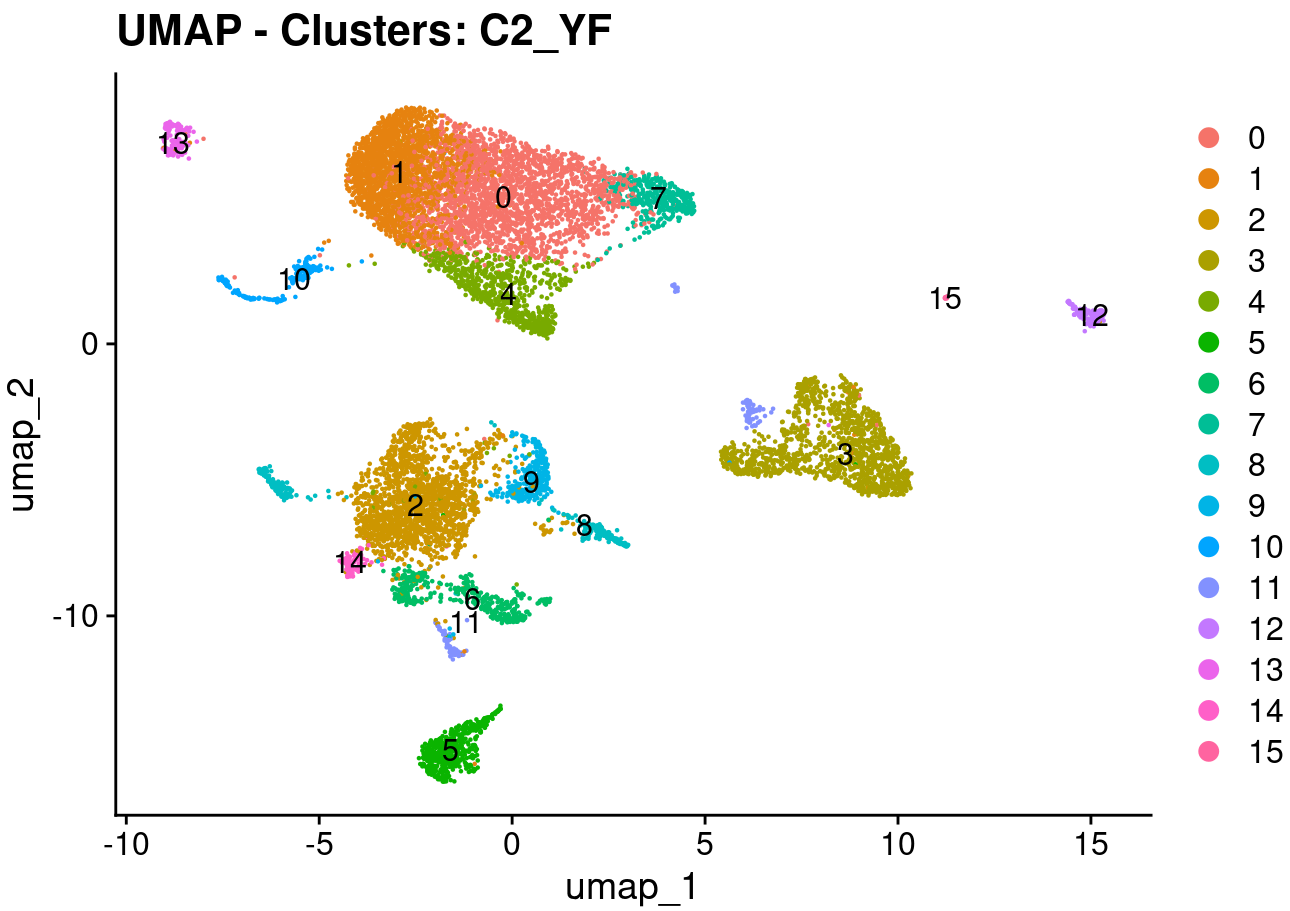
```
## [[1]]
```


UMAP - Clusters: C1_YF
umap_2
umap_1
10
0
-10
-20
-10
0
3
7
9
2
1
0
10
6
11
8
13
12
4
5
0
1
2
3
4
5
6
7
8
9
10
11
12
13

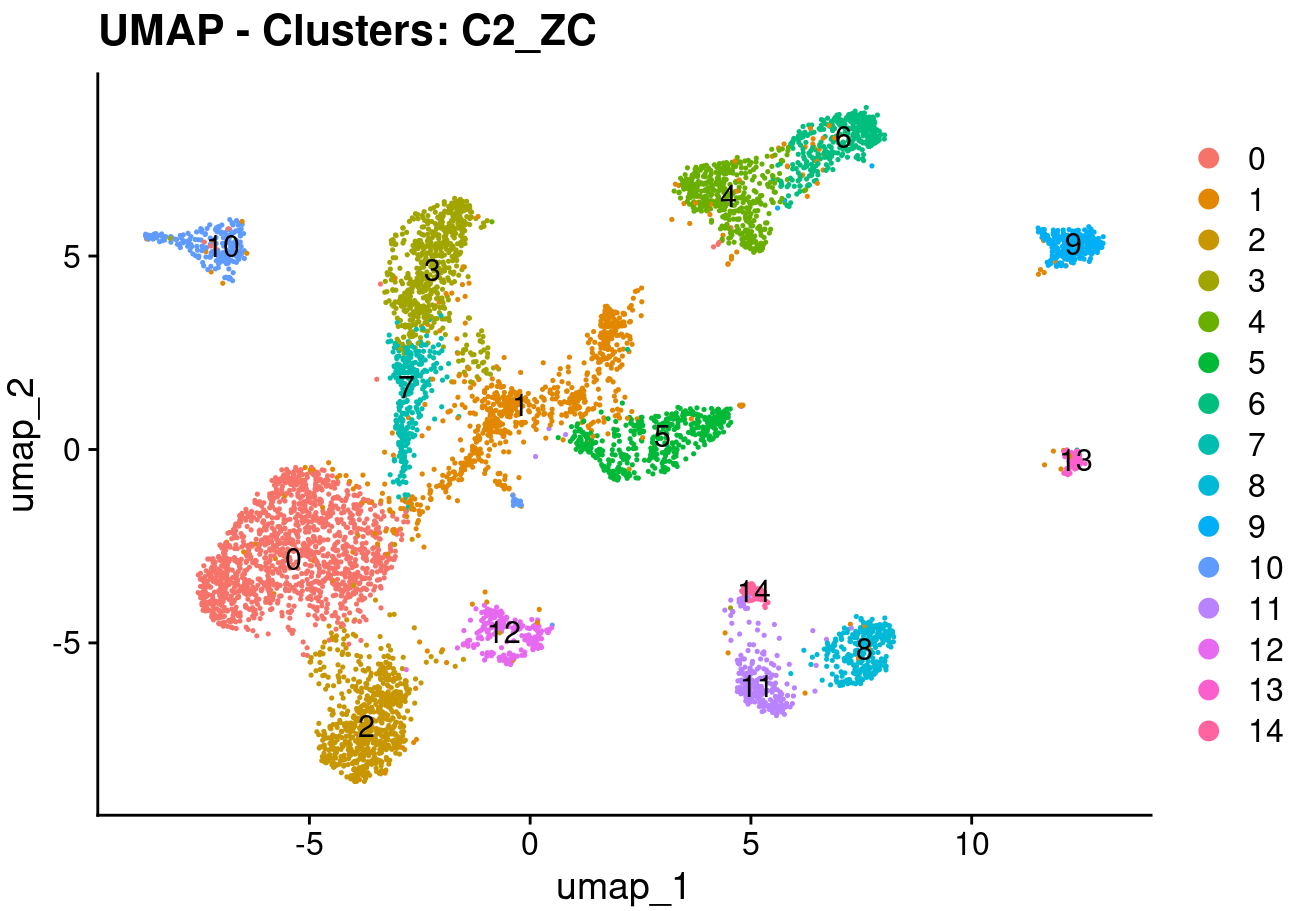
```
##
## [[2]]
```


UMAP - Clusters: C1_ZY
umap_2
umap_1
10
0
-10
-10
0
10
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

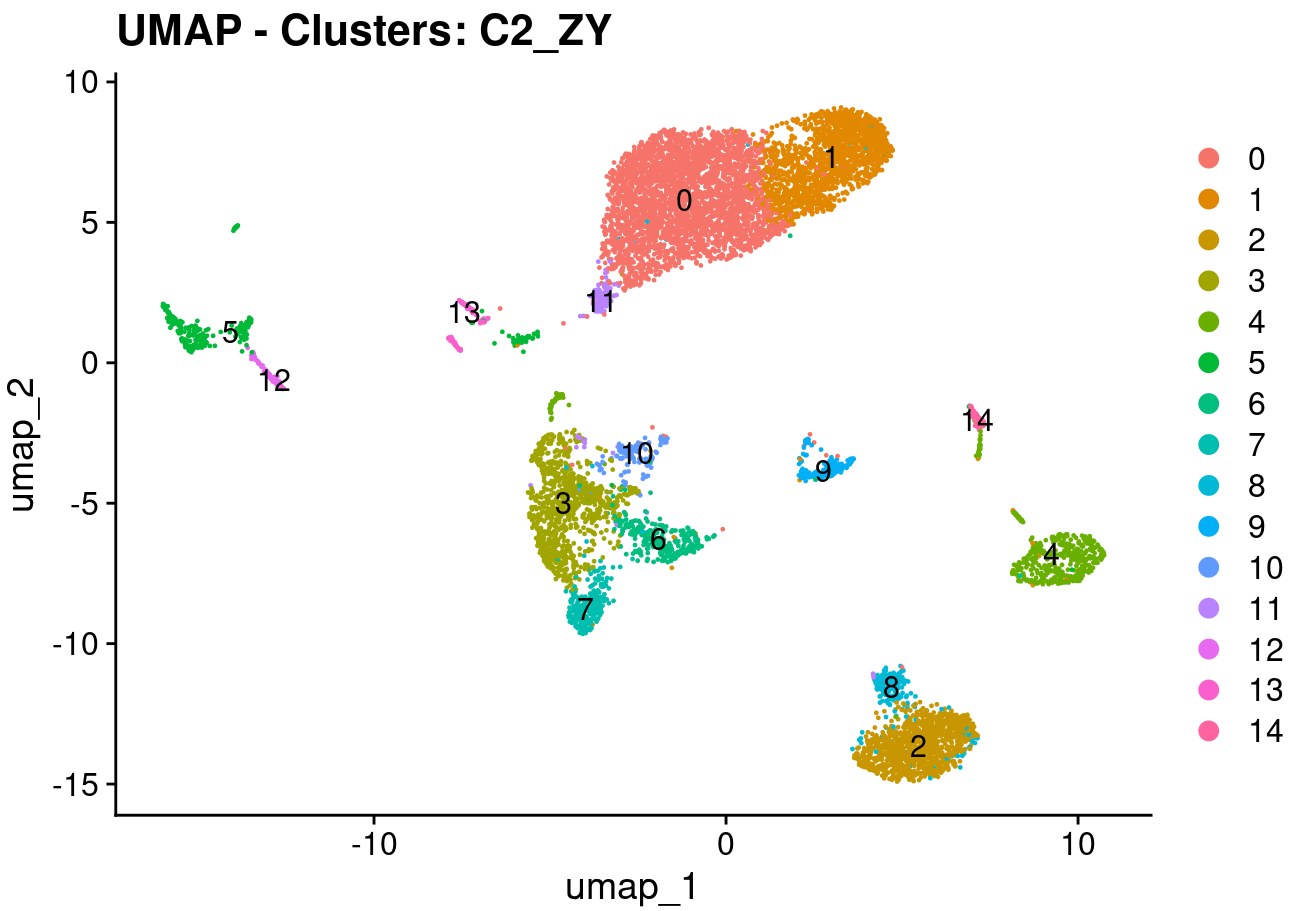
```
##
## [[3]]
```


UMAP - Clusters: C2_YF
umap_2
umap_1
0
-10
-10
-5
0
5
10
15
13
1
0
7
10
4
15
12
3
2
9
8
14
6
11
5
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14
15

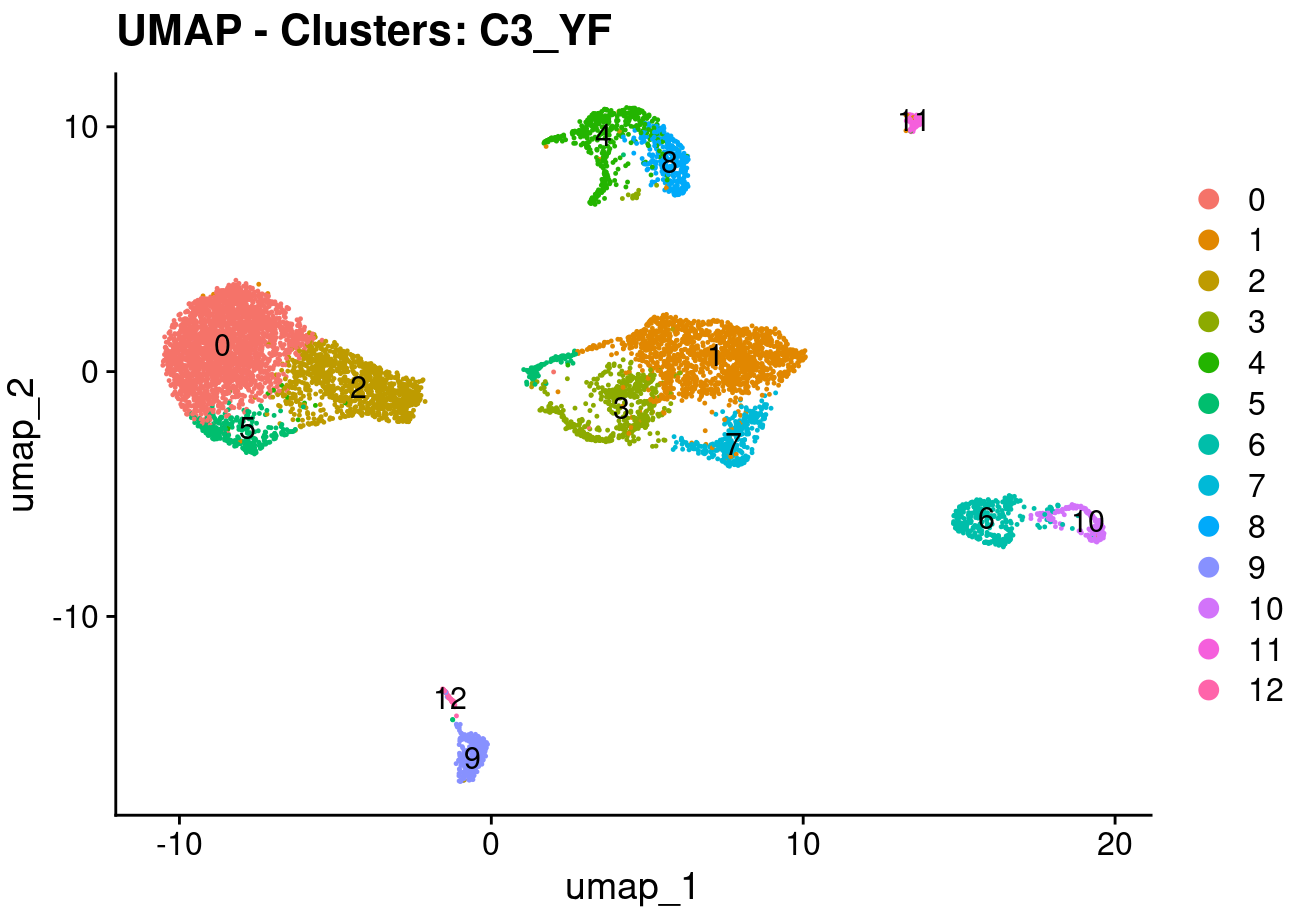
```
##
## [[4]]
```


UMAP - Clusters: C2_ZC
umap_2
umap_1
5
0
-5
-5
0
5
10
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14

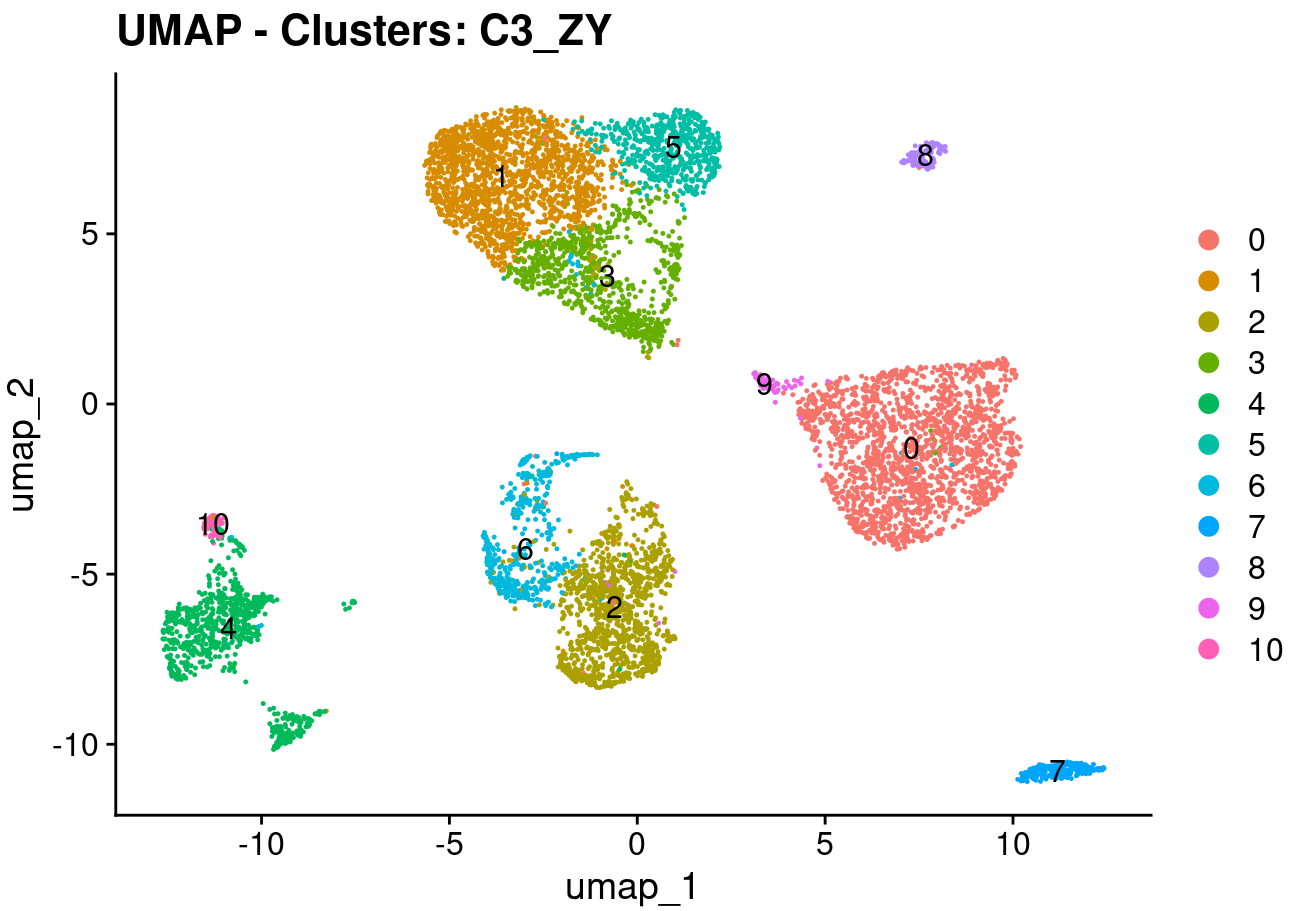
```
##
## [[5]]
```


UMAP - Clusters: C2_ZY
umap_2
umap_1
10
5
0
-5
-10
-15
-10
0
10
0
1
2
3
4
5
6
7
8
9
10
11
12
13
14

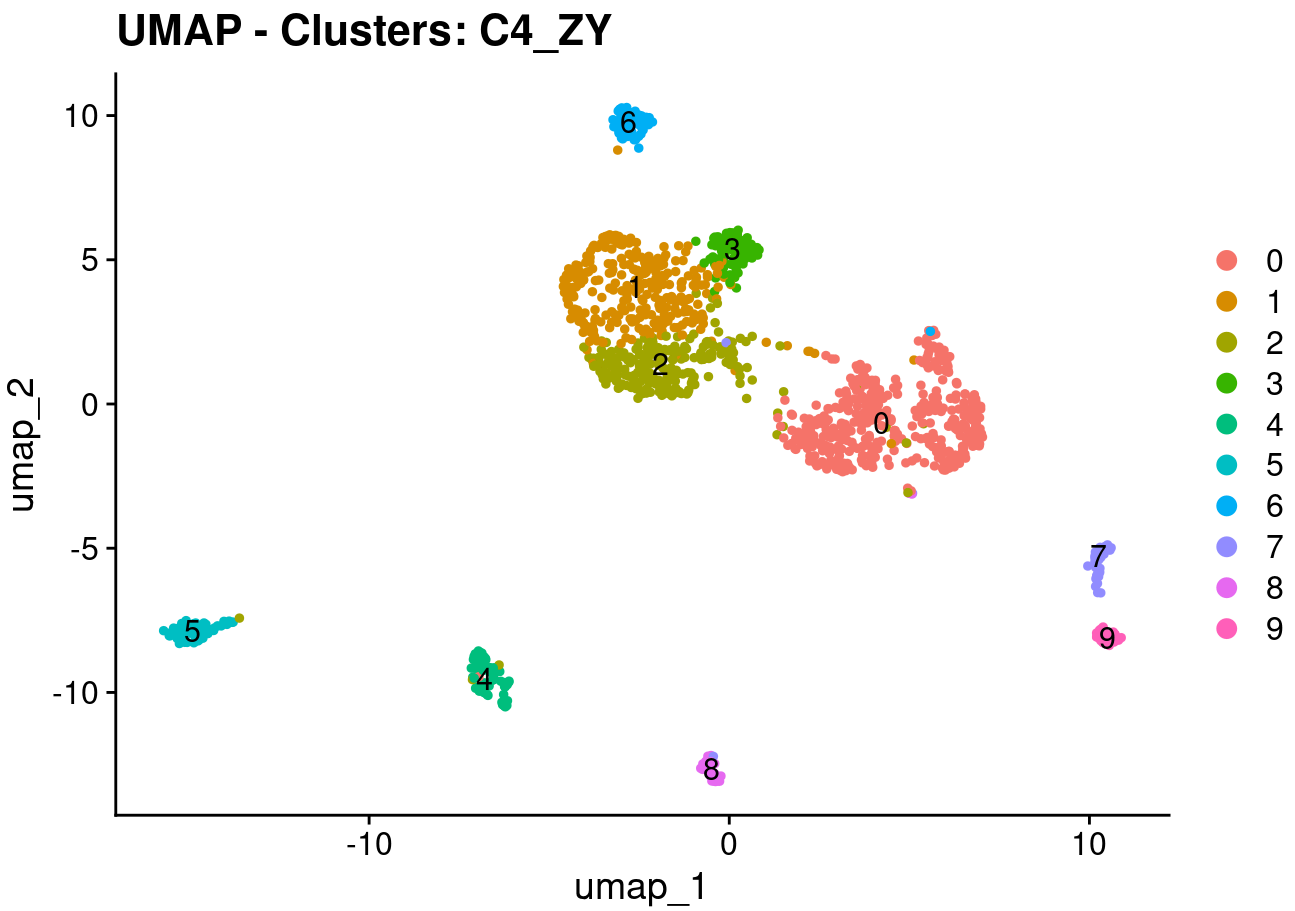
```
##
## [[6]]
```


UMAP - Clusters: C3_YF
umap_2
umap_1
10
0
-10
-10
0
10
20
0
1
2
3
4
5
6
7
8
9
10
11
12

```
##
## [[7]]
```


UMAP - Clusters: C3_ZY
umap_2
umap_1
5
0
-5
-10
-10
-5
0
5
10
1
5
8
3
9
0
10
6
2
4
7
0
1
2
3
4
5
6
7
8
9
10

```
##
## [[8]]
```




```
# Count cells per cluster and per sample
cluster_summary <- lapply(names(obj_with_clusters), function(nm) {
  obj <- obj_with_clusters[[nm]]
  tab <- table(Idents(obj))
  data.frame(sample = nm, cluster = names(tab), count = as.integer(tab))
}) %>% bind_rows()

print(cluster_summary)
```

```
##     sample cluster count
## 1    C1_YF       0  1813
## 2    C1_YF       1  1729
## 3    C1_YF       2  1532
## 4    C1_YF       3   549
## 5    C1_YF       4   520
## 6    C1_YF       5   442
## 7    C1_YF       6   388
## 8    C1_YF       7   361
## 9    C1_YF       8   333
## 10   C1_YF       9   152
## 11   C1_YF      10   126
## 12   C1_YF      11   110
## 13   C1_YF      12    71
## 14   C1_YF      13    34
## 15   C1_ZY       0  1422
## 16   C1_ZY       1  1411
## 17   C1_ZY       2  1024
## 18   C1_ZY       3   881
## 19   C1_ZY       4   855
## 20   C1_ZY       5   603
## 21   C1_ZY       6   518
## 22   C1_ZY       7   444
## 23   C1_ZY       8   433
## 24   C1_ZY       9   382
## 25   C1_ZY      10   153
## 26   C1_ZY      11    95
## 27   C1_ZY      12    93
## 28   C1_ZY      13    84
## 29   C1_ZY      14    69
## 30   C1_ZY      15    30
## 31   C2_YF       0  2416
## 32   C2_YF       1  2243
## 33   C2_YF       2  1792
## 34   C2_YF       3  1360
## 35   C2_YF       4   819
## 36   C2_YF       5   501
## 37   C2_YF       6   447
## 38   C2_YF       7   396
## 39   C2_YF       8   312
## 40   C2_YF       9   304
## 41   C2_YF      10   207
## 42   C2_YF      11   155
## 43   C2_YF      12   149
## 44   C2_YF      13   145
## 45   C2_YF      14    87
## 46   C2_YF      15    15
## 47   C2_ZC       0  1387
## 48   C2_ZC       1   831
## 49   C2_ZC       2   671
## 50   C2_ZC       3   609
## 51   C2_ZC       4   442
```

```
## 52   C2_ZC       5   422
## 53   C2_ZC       6   350
## 54   C2_ZC       7   304
## 55   C2_ZC       8   279
## 56   C2_ZC       9   263
## 57   C2_ZC      10   244
## 58   C2_ZC      11   239
## 59   C2_ZC      12   209
## 60   C2_ZC      13    82
## 61   C2_ZC      14    66
## 62   C2_ZY       0  3158
## 63   C2_ZY       1  1584
## 64   C2_ZY       2  1085
## 65   C2_ZY       3   912
## 66   C2_ZY       4   500
## 67   C2_ZY       5   334
## 68   C2_ZY       6   307
## 69   C2_ZY       7   272
## 70   C2_ZY       8   207
## 71   C2_ZY       9   175
## 72   C2_ZY      10   135
## 73   C2_ZY      11   133
## 74   C2_ZY      12   112
## 75   C2_ZY      13    97
## 76   C2_ZY      14    89
## 77   C3_YF       0  2593
## 78   C3_YF       1  1524
## 79   C3_YF       2  1075
## 80   C3_YF       3   504
## 81   C3_YF       4   488
## 82   C3_YF       5   385
## 83   C3_YF       6   374
## 84   C3_YF       7   328
## 85   C3_YF       8   296
## 86   C3_YF       9   290
## 87   C3_YF      10   184
## 88   C3_YF      11    86
## 89   C3_YF      12    49
## 90   C3_ZY       0  1861
## 91   C3_ZY       1  1663
## 92   C3_ZY       2  1143
## 93   C3_ZY       3   821
## 94   C3_ZY       4   780
## 95   C3_ZY       5   577
## 96   C3_ZY       6   467
## 97   C3_ZY       7   233
## 98   C3_ZY       8   104
## 99   C3_ZY       9    87
## 100  C3_ZY      10    72
## 101  C4_ZY       0   443
## 102  C4_ZY       1   305
## 103  C4_ZY       2   238
```

```
## 104  C4_ZY       3    91
## 105  C4_ZY       4    72
## 106  C4_ZY       5    70
## 107  C4_ZY       6    61
## 108  C4_ZY       7    48
## 109  C4_ZY       8    40
## 110  C4_ZY       9    38
```

Identify marker genes per cluster (example on first object):

```
obj1 <- obj_with_clusters[[1]]
markers_obj1 <- FindAllMarkers(obj1, only.pos = TRUE, min.pct = 0.25, logfc.threshold = 0.25)
```

```
## Calculating cluster 0
```

```
## For a (much!) faster implementation of the Wilcoxon Rank Sum Test,
## (default method for FindMarkers) please install the presto package
## --------------------------------------------
## install.packages('devtools')
## devtools::install_github('immunogenomics/presto')
## --------------------------------------------
## After installation of presto, Seurat will automatically use the more
## efficient implementation (no further action necessary).
## This message will be shown once per session
```

```
## Calculating cluster 1
```

```
## Calculating cluster 2
```

```
## Calculating cluster 3
```

```
## Calculating cluster 4
```

```
## Calculating cluster 5
```

```
## Calculating cluster 6
```

```
## Calculating cluster 7
```

```
## Calculating cluster 8
```

```
## Calculating cluster 9
```

```
## Calculating cluster 10
```

```
## Calculating cluster 11
```

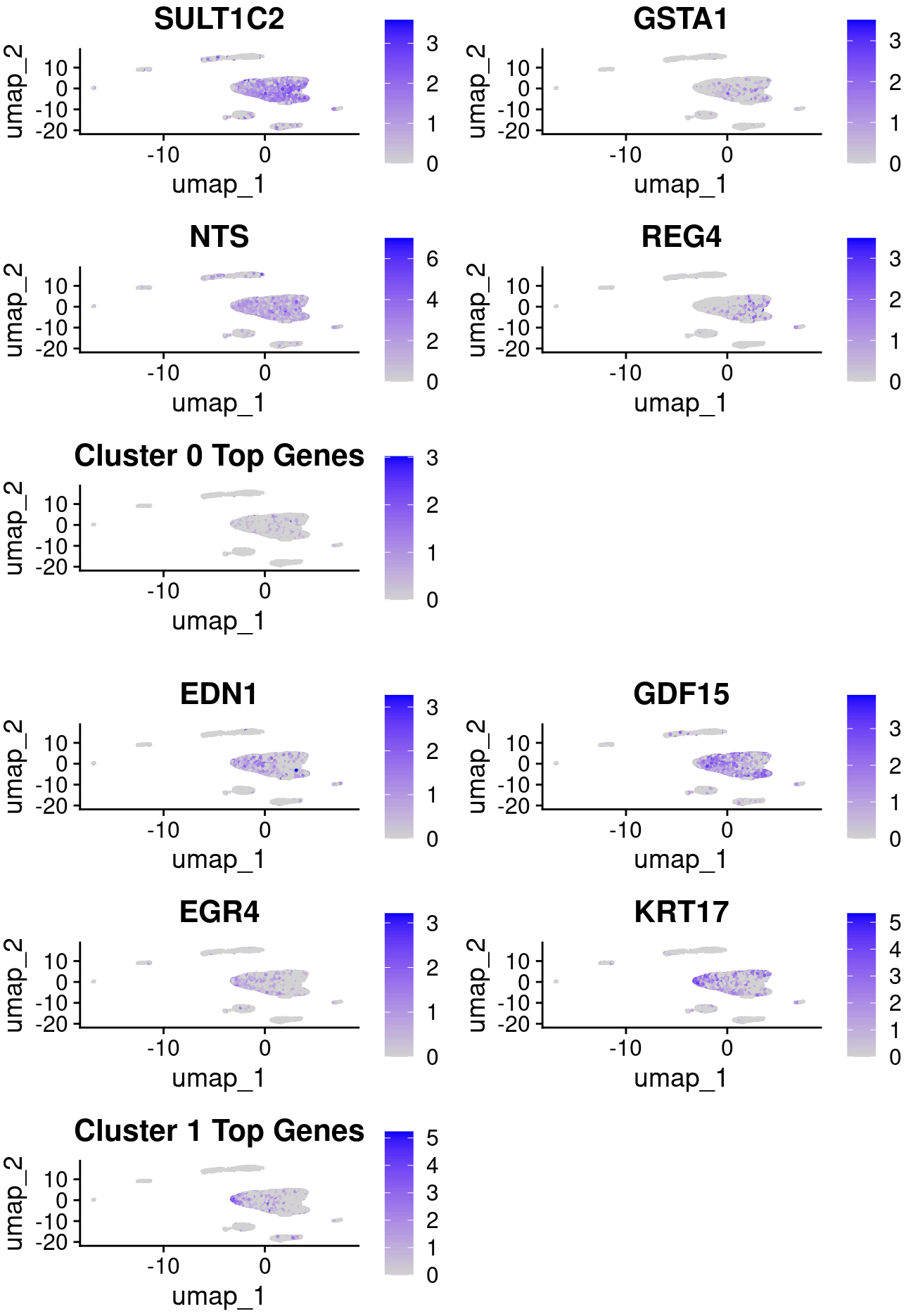
```
## Calculating cluster 12
```

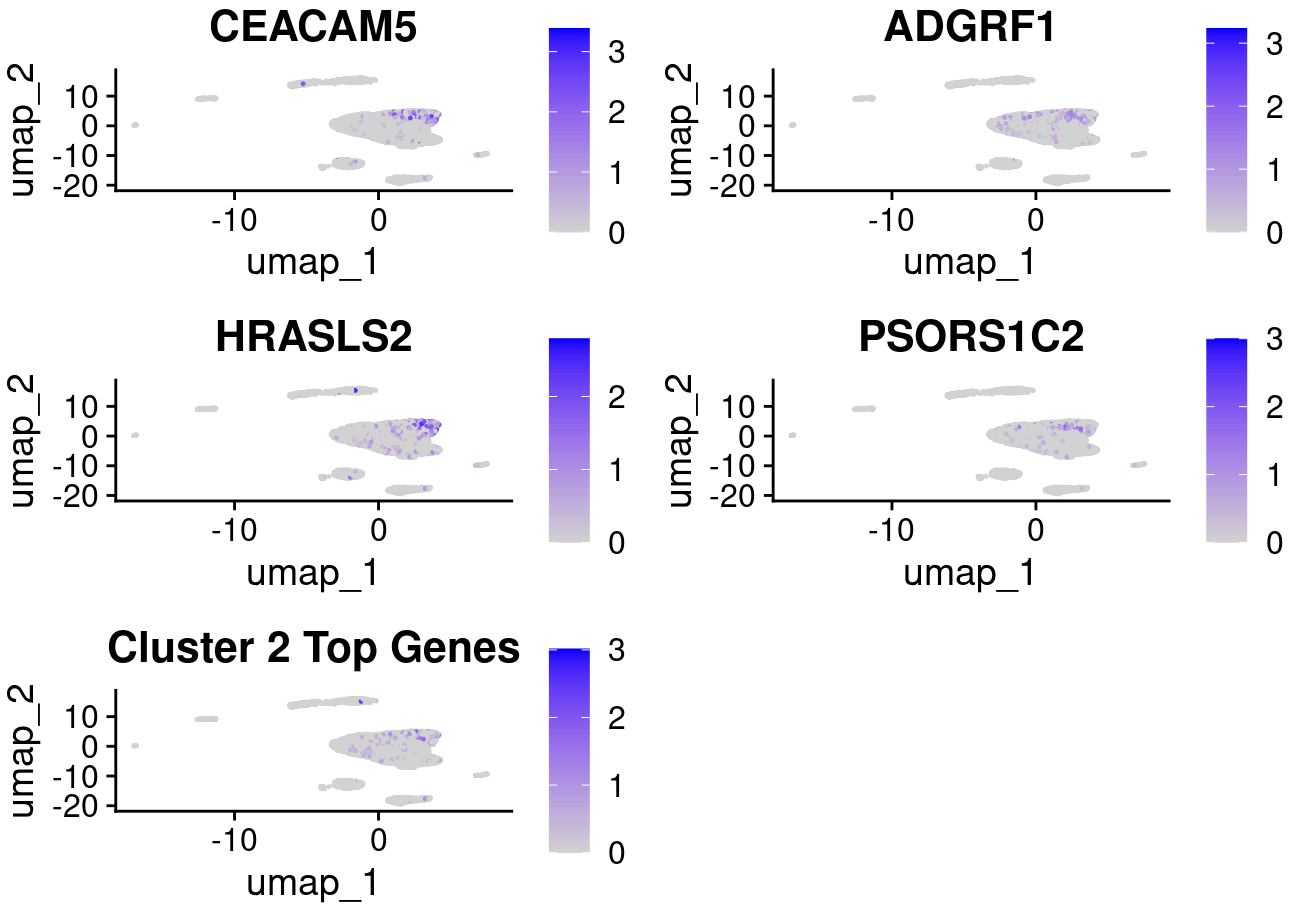
```
## Calculating cluster 13
```

```
top5_markers <- markers_obj1 %>% group_by(cluster) %>% top_n(n = 5, wt = avg_log2FC)

print(top5_markers)
```

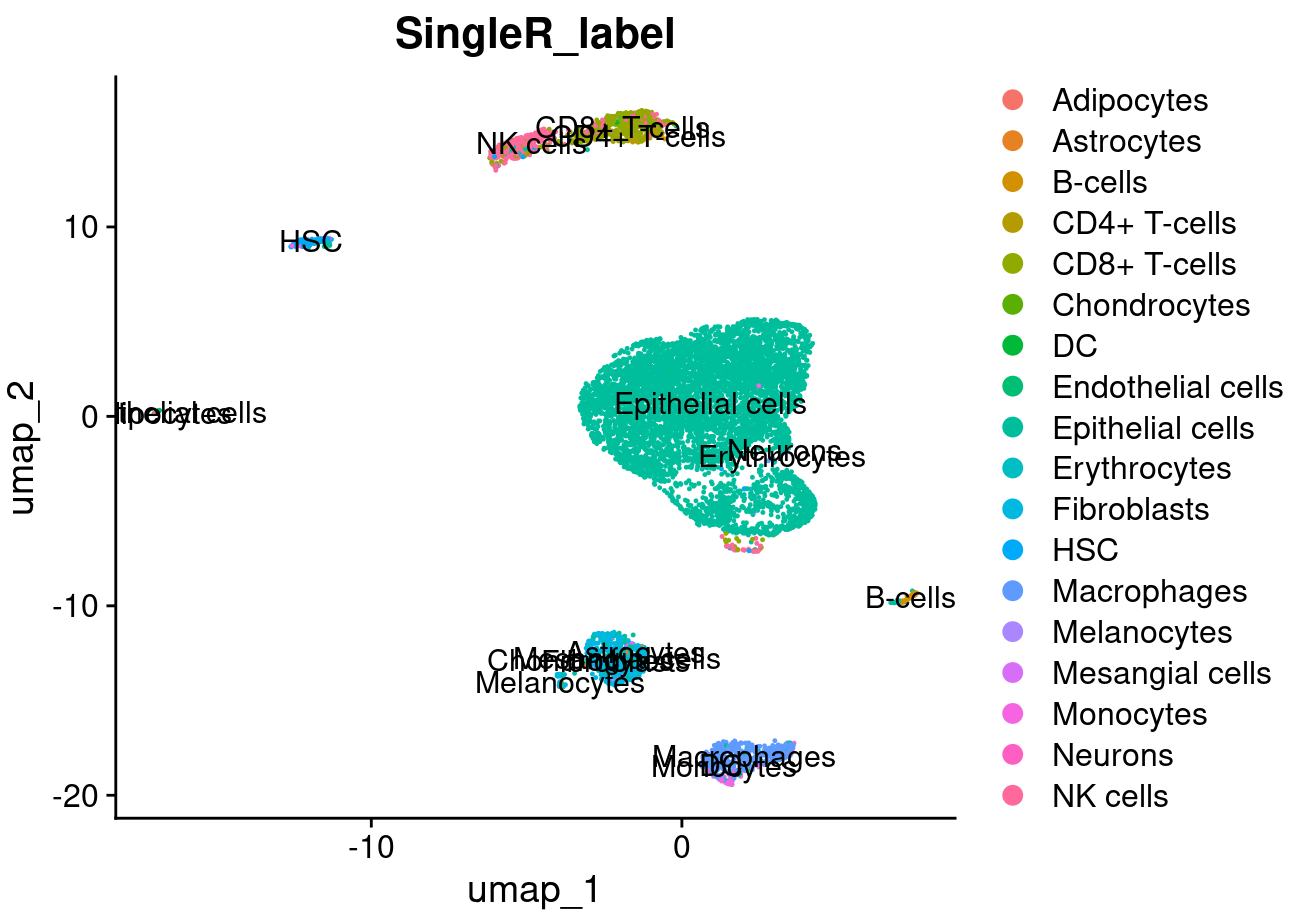
```
## # A tibble: 70 × 7
## # Groups:   cluster [14]
##        p_val avg_log2FC pct.1 pct.2 p_val_adj cluster gene
##        <dbl>      <dbl> <dbl> <dbl>     <dbl> <fct>   <chr>
##  1 5.63e-202      0.980 0.934 0.595 1.22e-197 0       SULT1C2
##  2 1.09e- 97      1.18  0.437 0.189 2.36e- 93 0       GSTA1
##  3 1.22e- 70      0.996 0.912 0.648 2.64e- 66 0       NTS
##  4 4.94e- 69      1.00  0.366 0.171 1.07e- 64 0       REG4
##  5 2.25e- 61      1.10  0.272 0.111 4.87e- 57 0       PNPLA3
##  6 1.51e-201      1.63  0.667 0.308 3.26e-197 1       EDN1
##  7 4.19e-197      1.39  0.828 0.502 9.06e-193 1       GDF15
##  8 2.94e-153      1.75  0.521 0.222 6.36e-149 1       EGR4
##  9 3.99e-127      1.56  0.629 0.357 8.63e-123 1       KRT17
## 10 4.38e-126      1.69  0.486 0.216 9.47e-122 1       MMP7
## # ℹ 60 more rows
```

```
# 5. Visualize marker expression for key clusters (example with top 3 clusters by size)
top_clusters <- cluster_summary %>%
  filter(sample == names(obj_with_clusters)[1]) %>%
  top_n(3, count) %>%
  pull(cluster) %>%
  unique()

for (cl in top_clusters) {
  top_genes <- markers_obj1 %>% filter(cluster == cl) %>% top_n(5, avg_log2FC) %>% pull(gene)
  print(FeaturePlot(obj1, features = top_genes, reduction = "umap") +
          ggtitle(paste("Cluster", cl, "Top Genes")))
}
```


SULT1C2
GSTA1
NTS
REG4
Cluster 0 Top Genes
EDN1
GDF15
EGR4
KRT17
Cluster 1 Top Genes
umap_1
umap_2




```
# 6. Automated annotation (example with SingleR, using Blueprint+ENCODE as ref)

ref <- BlueprintEncodeData()
sce <- as.SingleCellExperiment(obj1)
preds <- SingleR(test = sce, ref = ref, labels = ref$label.main)

# Add labels to Seurat object
obj1$SingleR_label <- preds$labels

# Plot UMAP with labels
DimPlot(obj1, group.by = "SingleR_label", label = TRUE, reduction = "umap")
```


SingleR_label
umap_2
umap_1
10
0
-10
-20
-10
0
HSC
Epithelial cells
B-cells
Adipocytes
Astrocytes
B-cells
CD4+ T-cells
CD8+ T-cells
Chondrocytes
DC
Endothelial cells
Epithelial cells
Erythrocytes
Fibroblasts
HSC
Macrophages
Melanocytes
Mesangial cells
Monocytes
Neurons
NK cells

```
#manual_labels <- c(
 # "Adipocytes" ,             # 0: SULT1C2, GSTA1, NTS, REG4, PNPLA3
  #"Inflammatory Epithelium",      # 1: EDN1, GDF15, EGR4, KRT17, MMP7
  #"CEA+ Tumor Epithelium",        # 2: CEACAM5, ADGRF1, HRASLS2, PSORS1C2, FGFBP1
#  "CD8+ Cytotoxic T Cells",       # 3: IFNG, CD3G, GZMH, CD8B, CD8A
 # "Myofibroblasts / CAF",         # 4: ASPN, HTRA3, OMD, VGLL3, LRRC15
#  "Myeloid / Monocyte-like",      # 5: FPR3, FCGR1A, CLEC5A, CSF2RA, LINC02345
 # "Proliferating Epithelium (S)", # 6: DTL, MCM10, MND1, CDC45, EXO1
#  "NK / γδ T Cells",              # 7: TNFRSF18, GNLY, TMIGD2, TRDC, KRT81
 # "Cycling Cells (G2/M)",         # 8: CCNB1, CDC20, CCNB2, CENPA, NEK2
#  "Mast Cells",                   # 9: CPA3, TPSAB1, TPSB2, MS4A2, HDC
 # "Unannotated",                  # 10
 # "Unannotated",                  # 11
#  "Unannotated",                  # 12
 # "Unannotated"                   # 13
#)

# Name the labels with character cluster identities
#names(manual_labels) <- as.character(0:13)

# Assign to metadata
#obj1$manual_label <- manual_labels[as.character(Idents(obj1))]

# Plot with UMAP
#DimPlot(obj1, group.by = "manual_label", label = TRUE, reduction = "umap") +
 # ggtitle("Manual Annotation of Cell Types")
```

```
# Merge Normalized Samples:
merged_all <- merge(
  x = normalized_samples[[1]],
  y = normalized_samples[-1],
  add.cell.ids = names(normalized_samples),
  project = "MergedProject"
)

# Scale and PCA:
merged_all <- ScaleData(merged_all, vars.to.regress = c("nCount_RNA", "percent.mt"))
```

```
## Regressing out nCount_RNA, percent.mt
```

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

```
merged_all <- RunPCA(merged_all, features = VariableFeatures(merged_all))
```

```
## PC_ 1
## Positive:  EPCAM, MUC1, AGR2, S100A14, SFTA2, LGALS4, DSG2, TSPAN1, TSPAN8, KRT8
##     FXYD3, SLPI, ELF3, KRT19, TFF2, CTSE, GPRC5A, CLDN18, SLC44A4, MAL2
##     CLDN4, SMIM22, AGR3, ITGB4, NQO1, TFF1, C19orf33, TFF3, CEACAM6, GPX2
## Negative:  TNFAIP3, TYROBP, RGS1, PLIN2, ALOX5AP, C1QC, MS4A7, C1QA, C1QB, AIF1
##     FCGR3A, MRC1, GPNMB, FCER1G, MSR1, RGCC, GPR183, FCGR2A, CD163, CTSS
##     MS4A4A, GLUL, CYBB, HLA-DQA1, APOE, OLR1, CD14, LAIR1, PLXDC2, VSIG4
## PC_ 2
## Positive:  GZMA, IL7R, ISG20, GZMK, FKBP11, IL2RB, RARRES3, TIGIT, GZMB, CCND2
##     PMEPA1, IFNG, FBXO32, ICOS, GNLY, BATF, CD27, FKBP10, TNFRSF18, TNFAIP3
##     XCL2, BGN, CAV1, COL5A1, CTLA4, RGS16, AEBP1, DCBLD2, ENAH, KLK7
## Negative:  CD68, C1QA, MS4A7, CTSB, MRC1, C1QC, C1QB, AIF1, MSR1, FTL
##     CTSD, FCER1G, CYBB, HLA-DRA, TYROBP, MS4A4A, CD14, APOE, HLA-DRB1, APOC1
##     CAPG, LYZ, GPNMB, FCGR3A, CD163, FCGR2A, VSIG4, TREM2, NPC2, FBP1
## PC_ 3
## Positive:  ISG20, IL7R, S100A4, TNFAIP3, GZMA, TXNIP, TFF1, LYZ, ALOX5AP, RGS1
##     CCL4, TFF2, TFF3, AGR2, GZMK, GCHFR, PHGR1, PLAC8, CTSE, GPX2
##     ZG16B, TMEM238, AREG, BCL2A1, RAB11FIP1, CLDN18, HLA-DRB1, AKR7A3, HPGD, C15orf48
## Negative:  SPARC, IGFBP7, BGN, AEBP1, COL6A2, COL5A1, FSTL1, MYL9, TAGLN, NNMT
##     TIMP3, TPM2, CNN3, C11orf96, COL4A2, COL6A1, COL4A1, RARRES2, CTGF, CTSK
##     GPX8, CYR61, FN1, MGP, VCAN, TCF4, INHBA, LAMA4, IGFBP4, COX7A1
## PC_ 4
## Positive:  MKI67, BIRC5, TOP2A, UBE2C, TK1, TPX2, CDKN3, PCLAF, GTSE1, ASPM
##     DLGAP5, CCNB2, TYMS, RRM2, NUSAP1, CDK1, ZWINT, CENPF, HMMR, NUF2
##     CMBL, CEP55, CCNB1, UBE2T, DTYMK, NTS, CDC20, DIAPH3, KIF4A, MAD2L1
## Negative:  MMP7, KLK7, KLK11, LY6K, PRSS2, EEF1A2, SERPINA1, LCN2, PRSS22, TM4SF1
##     SPINK1, DEFB1, PRSS1, KRT7, COL17A1, VSTM2L, SPRR3, IGKC, IGHG4, LIF
##     IGHA1, IGLC2, CD24, LAMC2, SPRR1B, FAM129B, WNT7B, LAMB3, MT1G, TRIM29
## PC_ 5
## Positive:  MKI67, TOP2A, UBE2C, BIRC5, ASPM, TPX2, PCLAF, HMMR, DLGAP5, GTSE1
##     ANLN, CENPF, CDK1, RRM2, CCNB2, MMP7, TYMS, NUSAP1, NUF2, CEP55
##     CCNA2, SPC25, MAD2L1, STMN1, KIF23, KIF4A, NCAPG, ZWINT, CDKN3, HJURP
## Negative:  MMP23B, TM4SF5, JSRP1, MUC3A, AC005550.2, CMBL, FBXO2, PHGR1, HEPH, VSIG2
##     CLDN18, NTS, TFF1, ADIRF, LINC01133, AKR7A3, UCA1, ZG16B, TFF3, TXNIP
##     MUCL3, PTGDS, TFF2, SDR16C5, HSPA1A, TSPAN1, STARD10, CAPS, BCAS1, S100A4
```

```
#  Add Sample Metadata (for Harmony):
merged_all$sample <- merged_all$orig.ident

# Run Harmony Integration
merged_all <- RunHarmony(merged_all, group.by.vars = "sample")
```

```
## Transposing data matrix
```

```
## Initializing state using k-means centroids initialization
```

```
## Harmony 1/10
```

```
## Harmony 2/10
```

```
## Harmony 3/10
```

```
## Harmony converged after 3 iterations
```

```
merged_all <- FindNeighbors(merged_all, reduction = "harmony", dims = 1:25)
```

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

```
## Computing SNN
```

```
merged_all <- FindClusters(merged_all, resolution = 0.5)
```

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 60893
## Number of edges: 2039231
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.9146
## Number of communities: 19
## Elapsed time: 18 seconds
```

```
merged_all <- RunUMAP(merged_all, reduction = "harmony", dims = 1:25)
```

```
## 22:23:40 UMAP embedding parameters a = 0.9922 b = 1.112
```

```
## 22:23:40 Read 60893 rows and found 25 numeric columns
```

```
## 22:23:40 Using Annoy for neighbor search, n_neighbors = 30
```

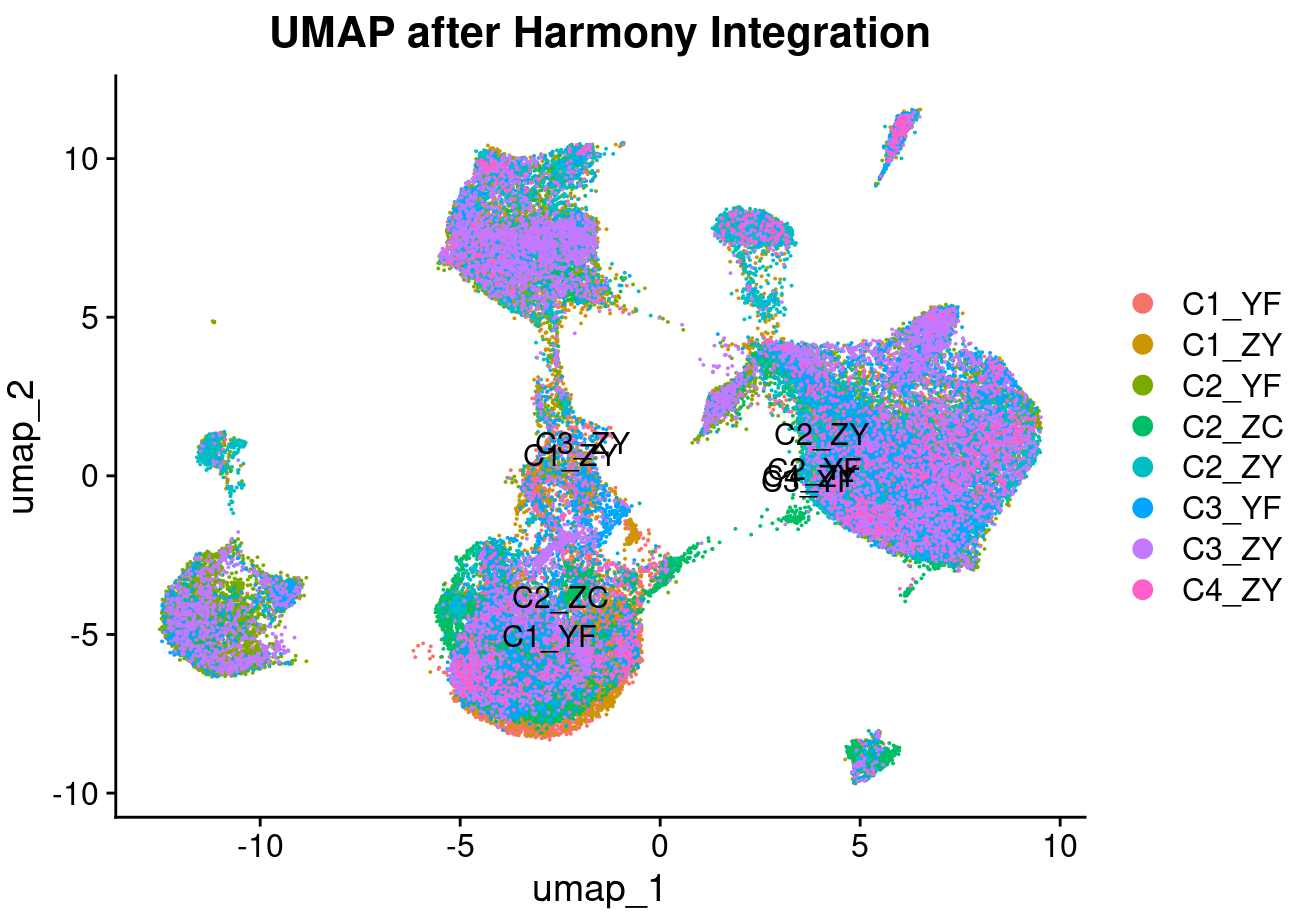
```
## 22:23:40 Building Annoy index with metric = cosine, n_trees = 50
```

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

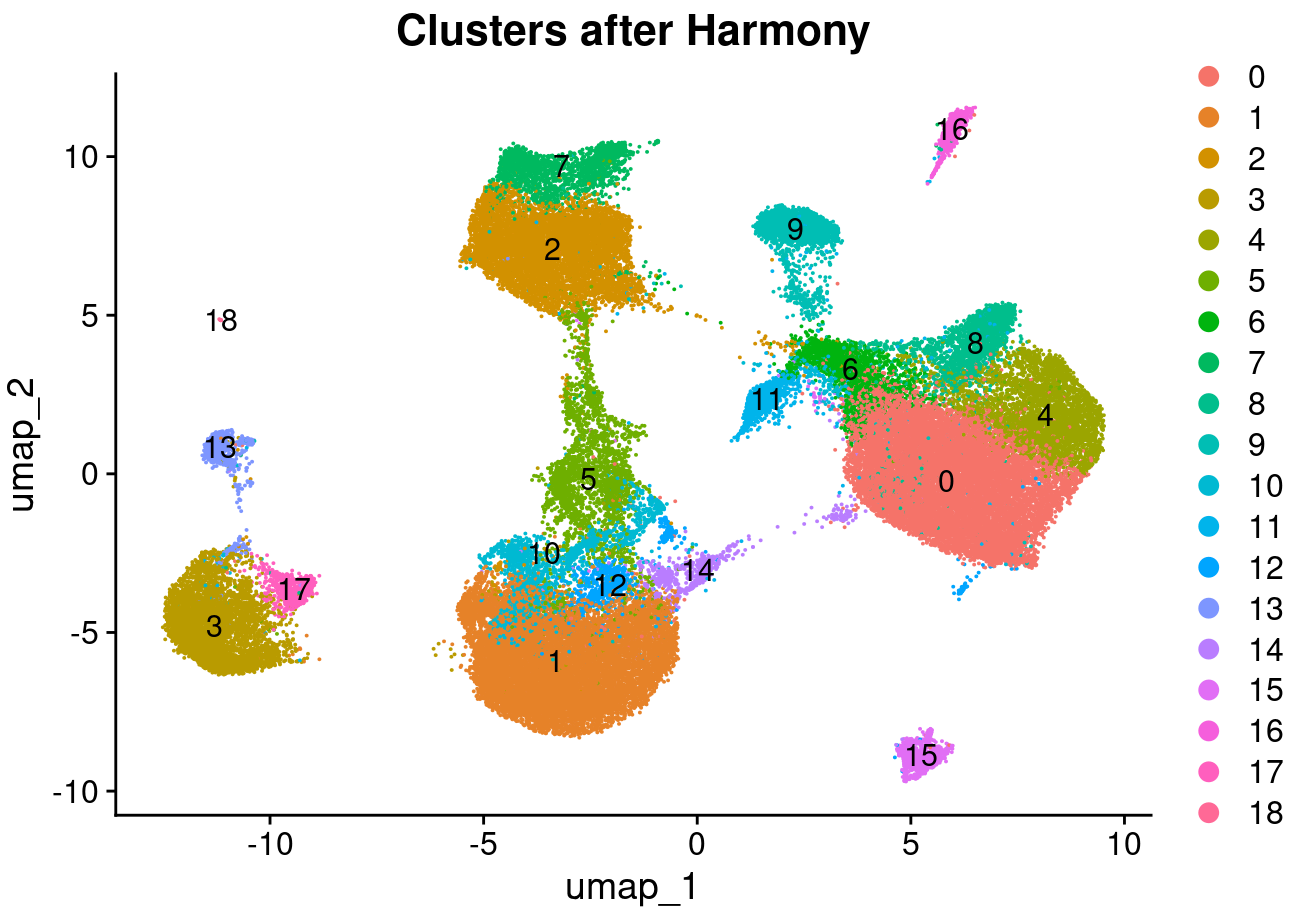
```
## [----|----|----|----|----|----|----|----|----|----|
```

```
## ************************************************|
## 22:23:47 Writing NN index file to temp file /scratch/5516304.1.academic/RtmpTRHoBg/file2da0b1
6c6e4a7c
## 22:23:47 Searching Annoy index using 1 thread, search_k = 3000
## 22:24:09 Annoy recall = 100%
## 22:24:10 Commencing smooth kNN distance calibration using 1 thread with target n_neighbors =
30
## 22:24:14 Initializing from normalized Laplacian + noise (using RSpectra)
## 22:24:17 Commencing optimization for 200 epochs, with 2780682 positive edges
## 22:24:41 Optimization finished
```

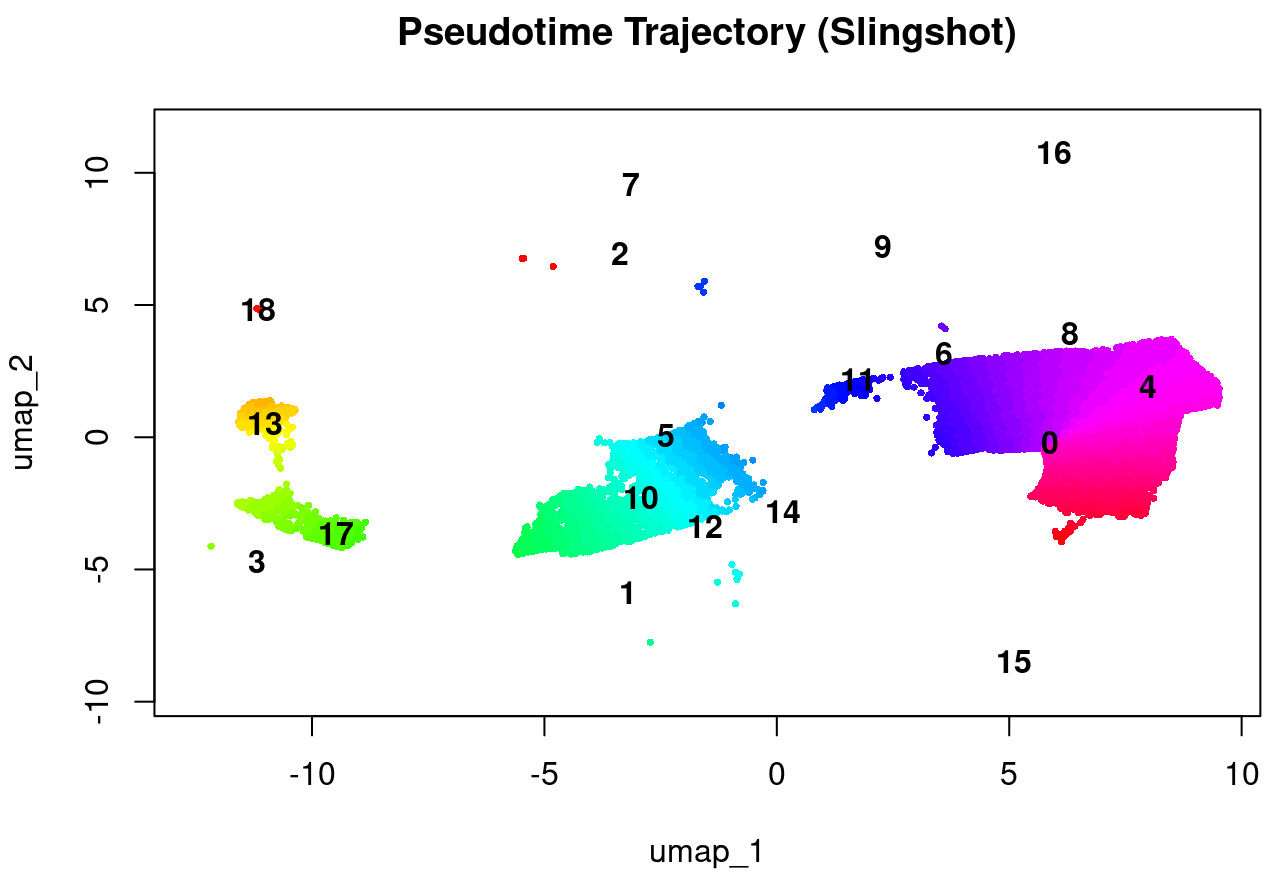
```
# Visualize Integrated UMAP
DimPlot(merged_all, group.by = "sample", reduction = "umap", label = TRUE) +
  ggtitle("UMAP after Harmony Integration")
```



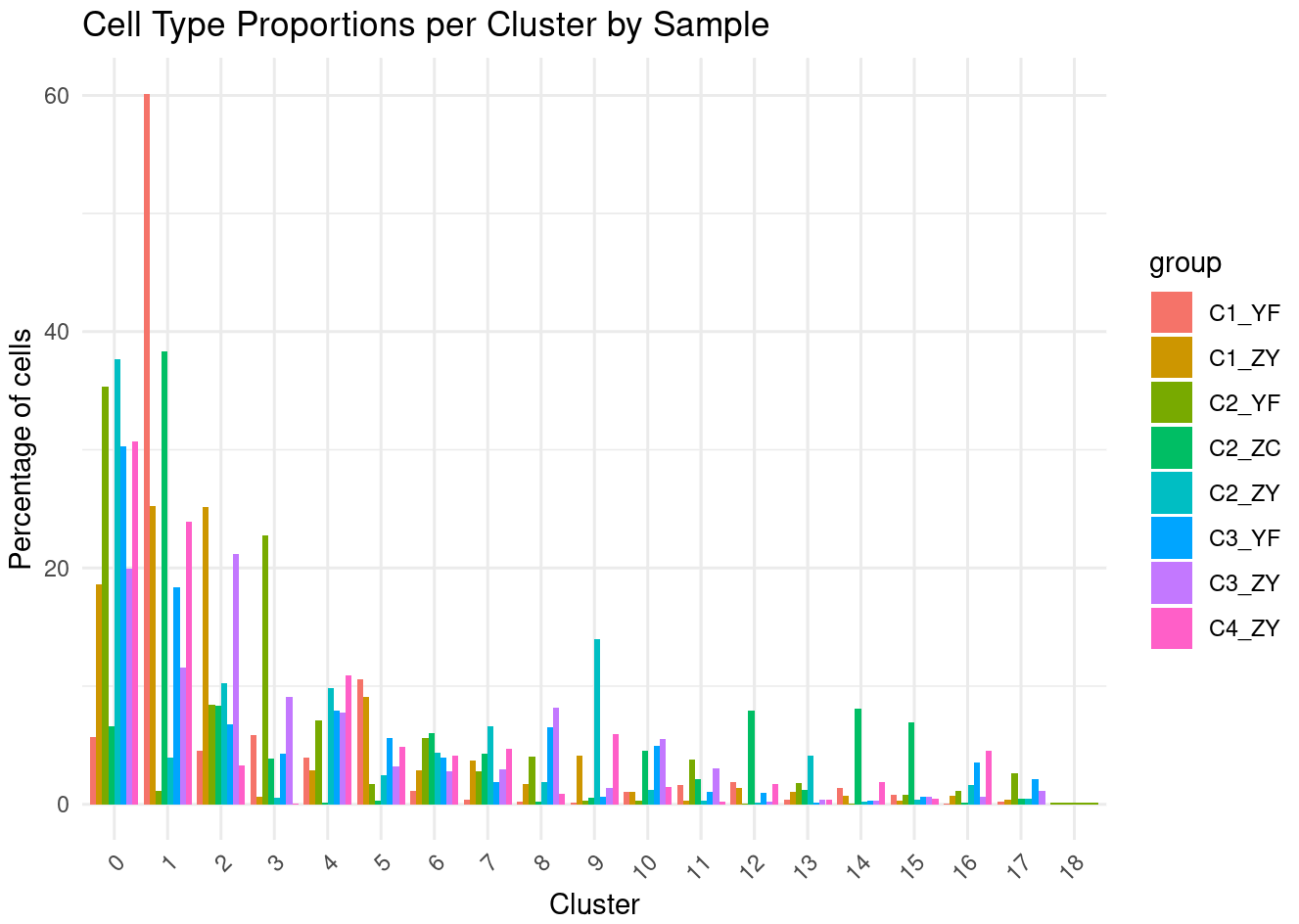

```
DimPlot(merged_all, group.by = "seurat_clusters", reduction = "umap", label = TRUE) +
  ggtitle("Clusters after Harmony")
```




```
# Fix for Seurat v5 multi-layer assay
merged_all <- JoinLayers(merged_all)

# Pseudotime Analysis with Slingshot:
sce_harmony <- as.SingleCellExperiment(merged_all)
sce_harmony <- slingshot(sce_harmony, clusterLabels = 'seurat_clusters', reducedDim = 'UMAP')

# Plot Pseudotime Trajectory:
plot(
  reducedDims(sce_harmony)$UMAP,
  col = rainbow(100)[cut(sce_harmony$slingPseudotime_1, 100)],
  pch = 16, cex = 0.5,
  main = "Pseudotime Trajectory (Slingshot)"
)

centers <- aggregate(
  reducedDims(sce_harmony)$UMAP,
  by = list(cluster = sce_harmony$seurat_clusters),
  FUN = mean
)
text(centers[,2], centers[,3], labels = centers$cluster, font = 2)
```




```
# Cell Type Proportion Analysis
# Extract metadata as a clean dataframe
meta_df <- merged_all@meta.data
meta_df$seurat_clusters <- Idents(merged_all)
meta_df$group <- merged_all$sample

# Plot cell type proportions per sample:
meta_df %>%
  dplyr::count(group, seurat_clusters) %>%
  group_by(group) %>%
  mutate(percentage = n / sum(n) * 100) %>%
  ggplot(aes(x = as.factor(seurat_clusters), y = percentage, fill = group)) +
  geom_bar(stat = "identity", position = "dodge") +
  theme_minimal() +
  ylab("Percentage of cells") +
  xlab("Cluster") +
  ggtitle("Cell Type Proportions per Cluster by Sample") +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```



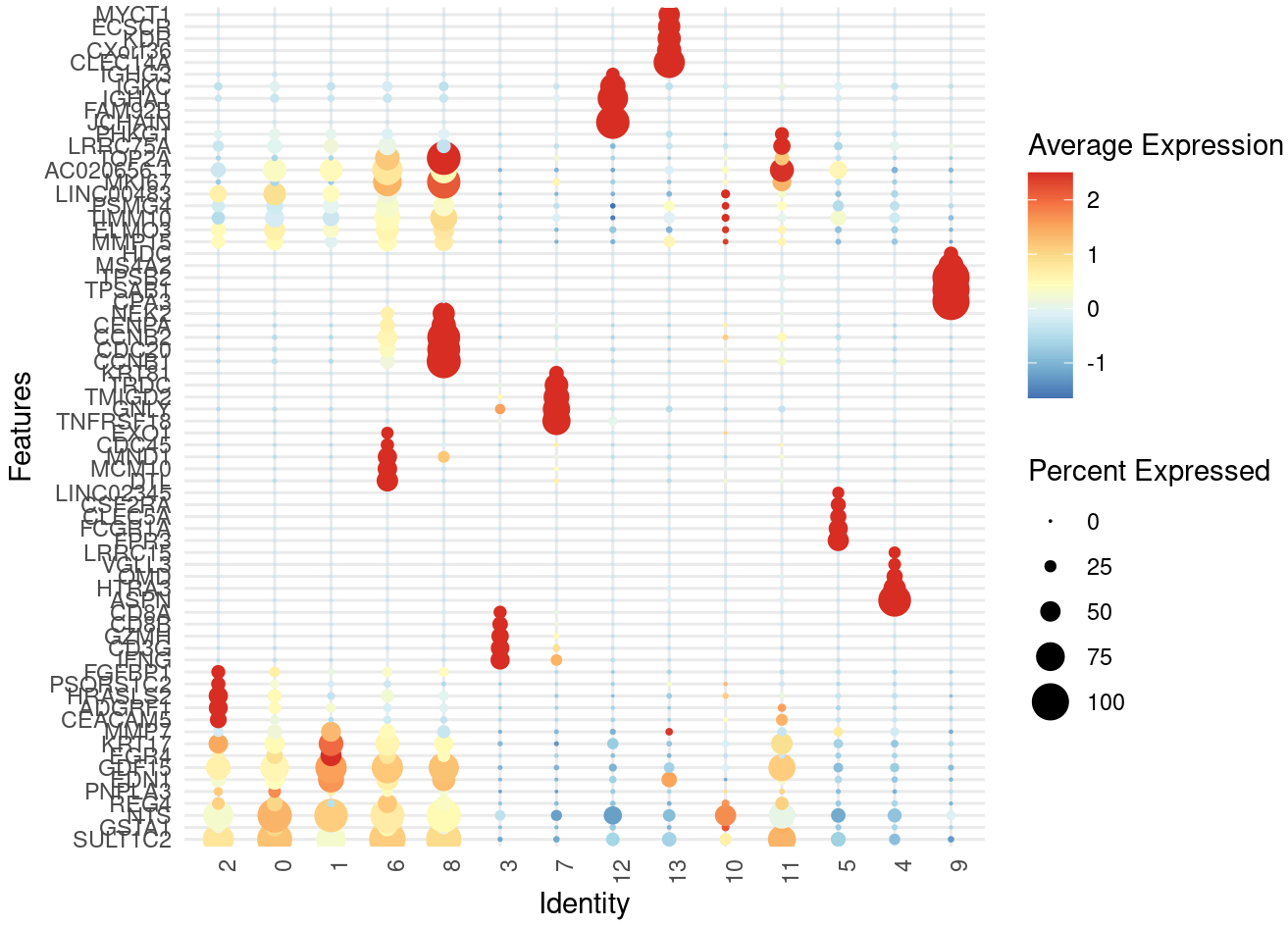

```
# --- Replicating Vignette Figures: Fig1d and Fig2a ---

# ----------------------------
# Fig1d: DotPlot of marker genes across clusters
# ----------------------------

# Ensure identity class is correctly set
Idents(obj1) <- factor(obj1$seurat_clusters)

# Use only marker genes present in dataset
top5_genes <- intersect(unique(top5_markers$gene), rownames(obj1))
if (length(top5_genes) == 0) stop("No valid marker genes found in obj1")

# Compute average expression across clusters
avg <- AggregateExpression(obj1, features = top5_genes, assays = "RNA")$RNA
```

```
## First group.by variable `ident` starts with a number, appending `g` to ensure valid variable
names
## This message is displayed once every 8 hours.
```

```
# Z-score transformation for heatmap clustering
sdat <- t(scale(t(avg[top5_genes, ])))
phtm <- pheatmap::pheatmap(sdat, cluster_rows = FALSE, silent = TRUE)

# Derive cluster order from heatmap
cluster_order <- gsub("^g", "", phtm$tree_col$labels[phtm$tree_col$order])
Idents(obj1) <- factor(obj1$seurat_clusters, levels = cluster_order)

# Reorder genes by peak cluster expression
mxid <- factor(colnames(sdat)[apply(sdat, 1, which.max)], levels = cluster_order)
gord <- rownames(sdat)[order(mxid)]

# Generate DotPlot
fig1d <- DotPlot(obj1, features = gord) + coord_flip() +
  scale_color_gradientn(colours = rev(brewer.pal(9, "RdYlBu")), guide = "colourbar") +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 90, hjust = 1))
```

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

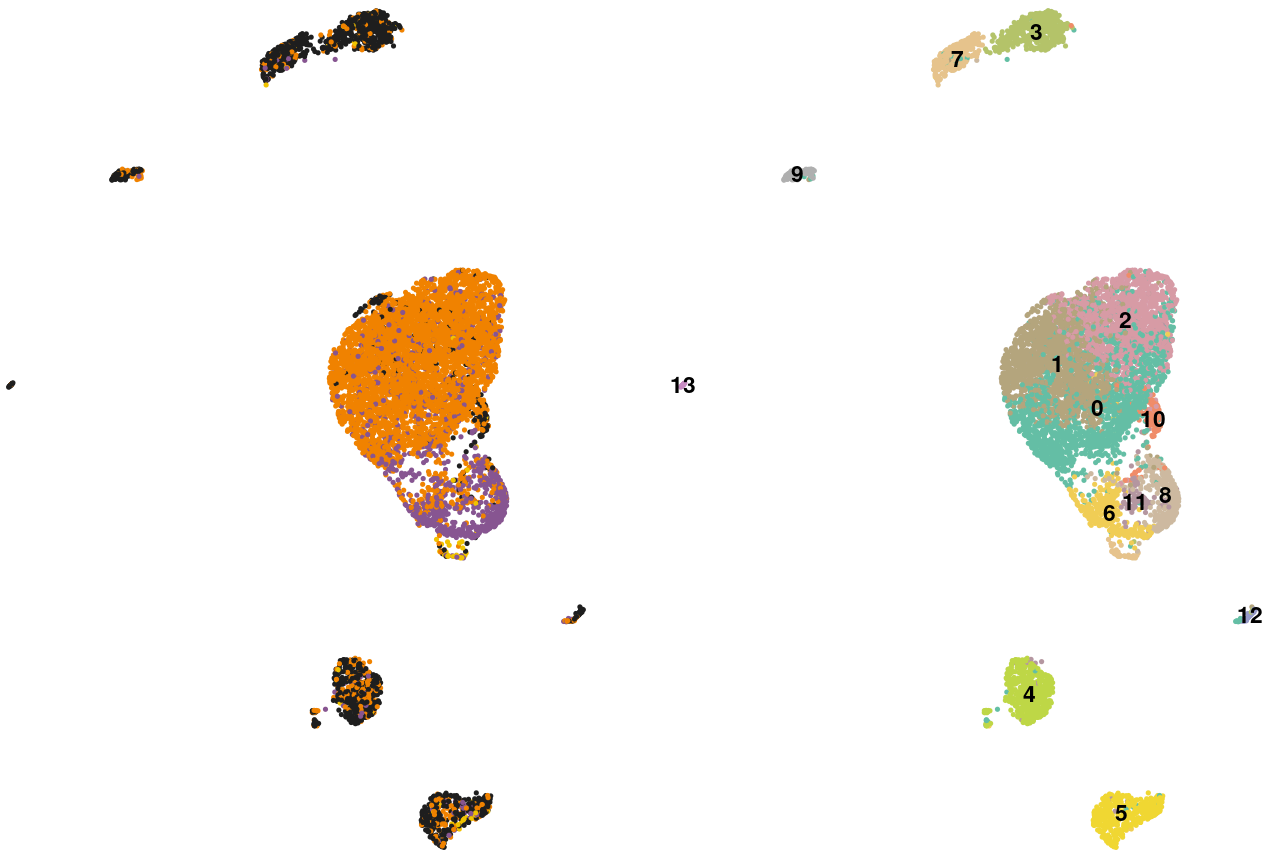
```
print(fig1d)
```



```
# ----------------------------
# Fig2a: UMAPs colored by marker expression and cluster identity
# ----------------------------

# Create marker flag based on EPCAM and MKI67 expression
epcam_expr <- GetAssayData(obj1, assay = "RNA", slot = "data")["EPCAM", ]
mki67_expr <- GetAssayData(obj1, assay = "RNA", slot = "data")["MKI67", ]

obj1$flag <- ifelse(epcam_expr > 0 & mki67_expr > 0, "EPCAM+MKI67+",
              ifelse(epcam_expr > 0, "EPCAM+",
              ifelse(mki67_expr > 0, "MKI67+", "Other")))
obj1$flag <- factor(obj1$flag, levels = c("Other", "MKI67+", "EPCAM+MKI67+", "EPCAM+"))

# Extract UMAP coordinates with metadata
umap_data <- data.frame(
  Embeddings(obj1, "umap"),
  cluster = as.character(Idents(obj1)),
  marker = obj1$flag,
  group = obj1$orig.ident
)
colnames(umap_data)[1:2] <- c("x", "y")

# Define color schemes
mkcols <- c("#222222", "#F3C300", "#875692", "#F38400")
ecols <- colorRampPalette(brewer.pal(8, "Set2"))(length(unique(umap_data$cluster)))
groupCols <- brewer.pal(length(unique(umap_data$group)), "Dark2")
```

```
## Warning in brewer.pal(length(unique(umap_data$group)), "Dark2"): minimal value for n is 3, re
turning requested palette with 3 different levels
```

```
# Fig2a (left): UMAP colored by marker flag
pp1 <- ggplot(umap_data, aes(x, y)) +
  geom_point(aes(color = marker), size = 0.25) +
  scale_color_manual(values = mkcols) +
  theme_pubr() + NoAxes() + NoLegend()

# Compute cluster centroids for labeling
centroids <- umap_data %>%
  group_by(cluster) %>%
  summarise(x = median(x), y = median(y))

# Fig2a (right): UMAP colored by cluster, with cluster labels
pp2 <- ggplot(umap_data, aes(x, y)) +
  geom_point(aes(color = cluster), size = 0.25) +
  scale_color_manual(values = ecols) +
  geom_text(data = centroids, aes(label = cluster), color = "black", size = 3, fontface = "bol
d") +
  theme_pubr() + NoAxes() + NoLegend()

# Display both UMAPs side-by-side
print(ggarrange(pp1, pp2, ncol = 2))
```



```
sink("sessionInfo.txt")
sessionInfo()
sink()
```