hw10

Adriana Bukała

5/18/2021

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

Data preprocessing

Load **raw** gene/cell matrix data.

```
pbmc.data <- Read10X(data.dir = "data/matrices_mex/hg19/")
```

Initialize the Seurat object with the raw (non-normalized data).

```
pbmc <- CreateSeuratObject(counts = pbmc.data, project = "pbmc3k", min.cells = 3, min.features = 200)
```

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

```
pbmc
```

```
## An object of class Seurat
## 17822 features across 69981 samples within 1 assay
## Active assay: RNA (17822 features, 0 variable features)
```

Let's examine a few genes in the first thirty cells

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

```
## 3 x 30 sparse Matrix of class "dgCMatrix"
```

```
##    [[ suppressing 30 column names 'AAACATACAAAACG-1', 'AAACATACAAAAGC-1', 'AAACATACAAACAG-1' ... ]]
```

```
##
## CD3D  . . . . . .
## TCL1A . . . . . .
## MS4A1 . . . . . .
```

```
max(pbmc.data)
```

```
## [1] 260
```

```
dim(pbmc.data)
```

```
## [1]   32738 5898240
```

The [[operator can add columns to object metadata. This is a great place to stash QC stats

```
pbmc[["percent.mt"]] <- PercentageFeatureSet(pbmc, pattern = "^MT-")
```

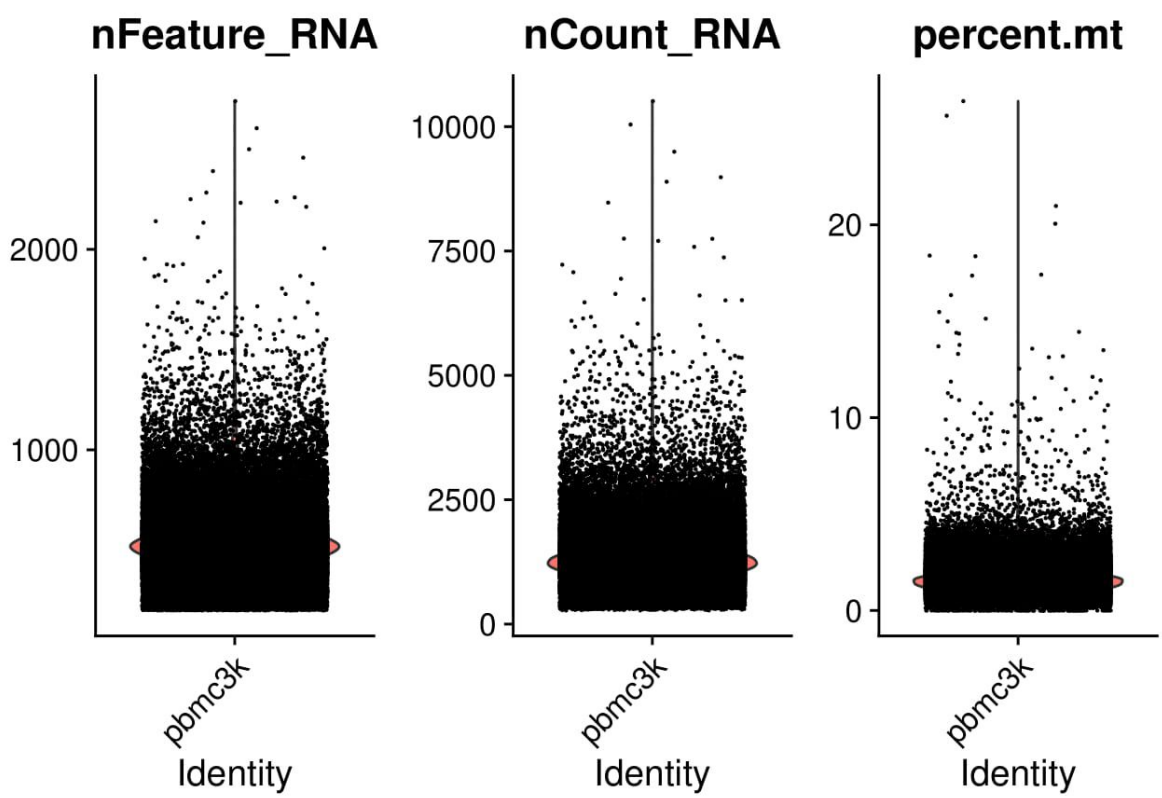
Show QC metrics for the first 5 cells

```
head(pbmc@meta.data, 5)
```

```
##                  orig.ident nCount_RNA nFeature_RNA percent.mt
## AAACATACACCCAA-1     pbmc3k       1216          498   2.220395
## AAACATACCCCTCA-1     pbmc3k       1265          472   2.450593
## AAACATACCGGAGA-1     pbmc3k       1322          542   1.512859
## AAACATACTAACCG-1     pbmc3k        854          349   1.053864
## AAACATACTCTTCA-1     pbmc3k       1252          446   1.757188
```

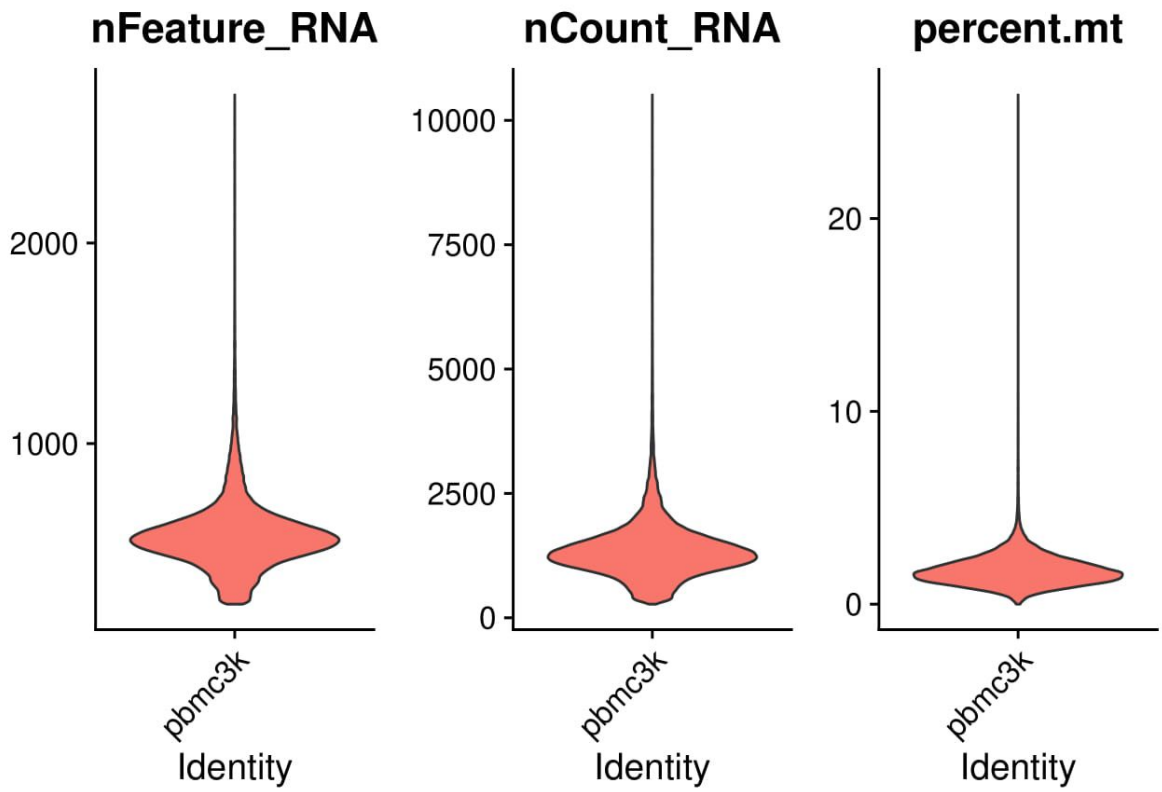
Visualize QC metrics as a violin plot.

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



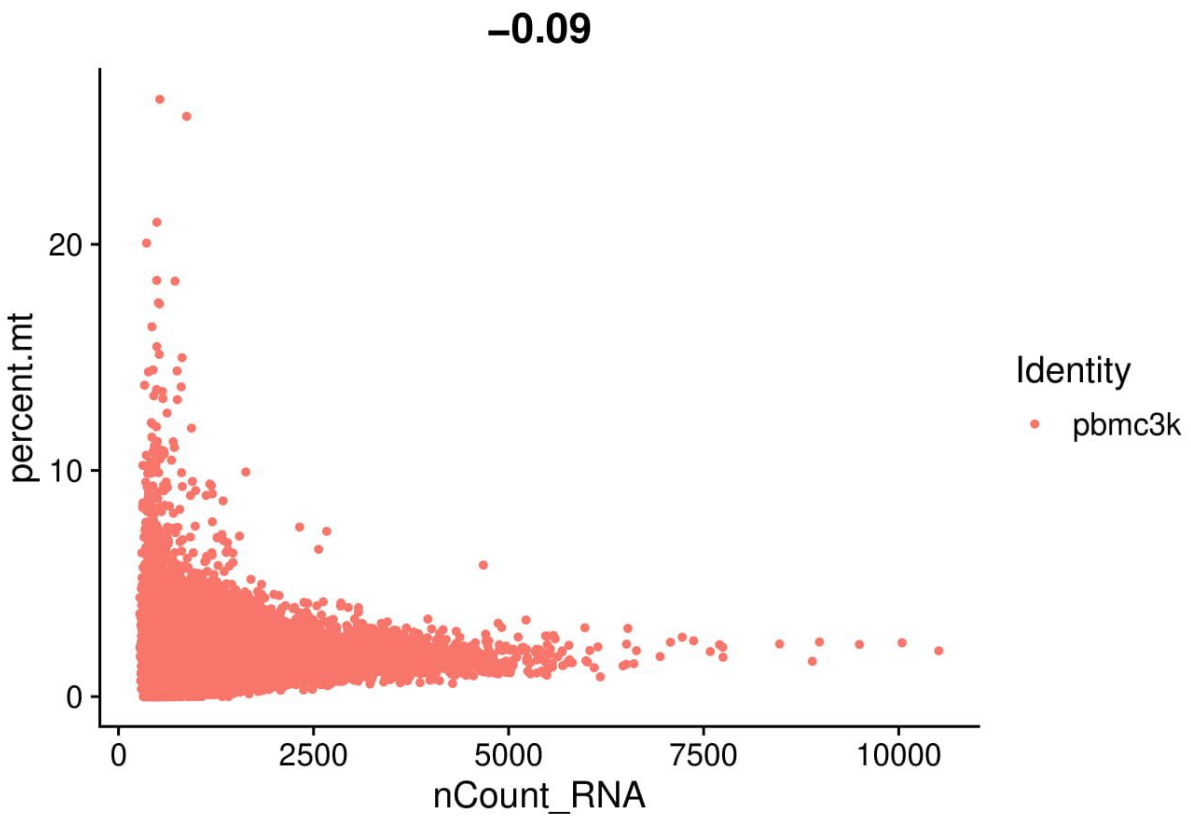
It's really hard to see anything, right?

```
VlnPlot(pbmc, features = c("nFeature_RNA", "nCount_RNA", "percent.mt"), ncol = 3, pt.size = 0)
```

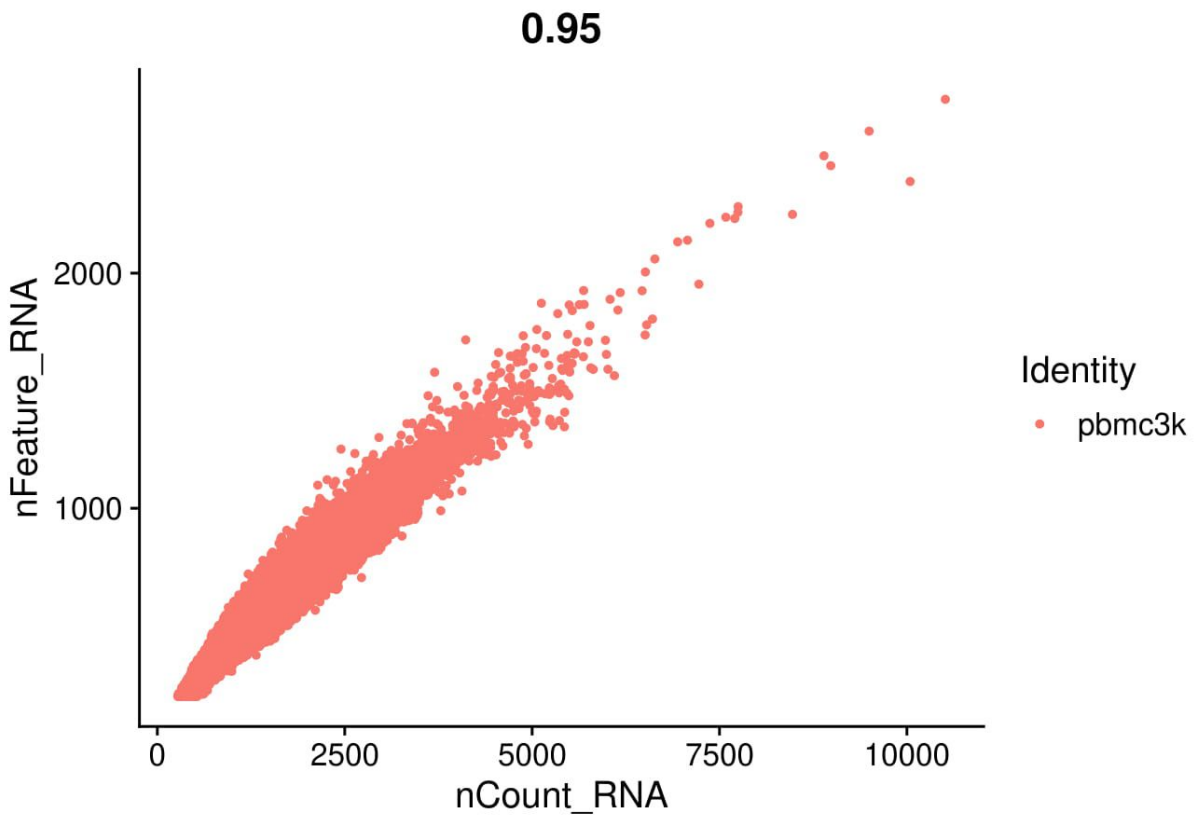


FeatureScatter is typically used to visualize feature-feature relationships, but can be used for anything calculated by the object, i.e. columns in object metadata, PC scores etc.

```
FeatureScatter(pbmc, feature1 = "nCount_RNA", feature2 = "percent.mt")
```




```
FeatureScatter(pbmc, feature1 = "nCount_RNA", feature2 = "nFeature_RNA")
```



Our data is a bit messy, so we need to filter and normalize it.

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

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

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

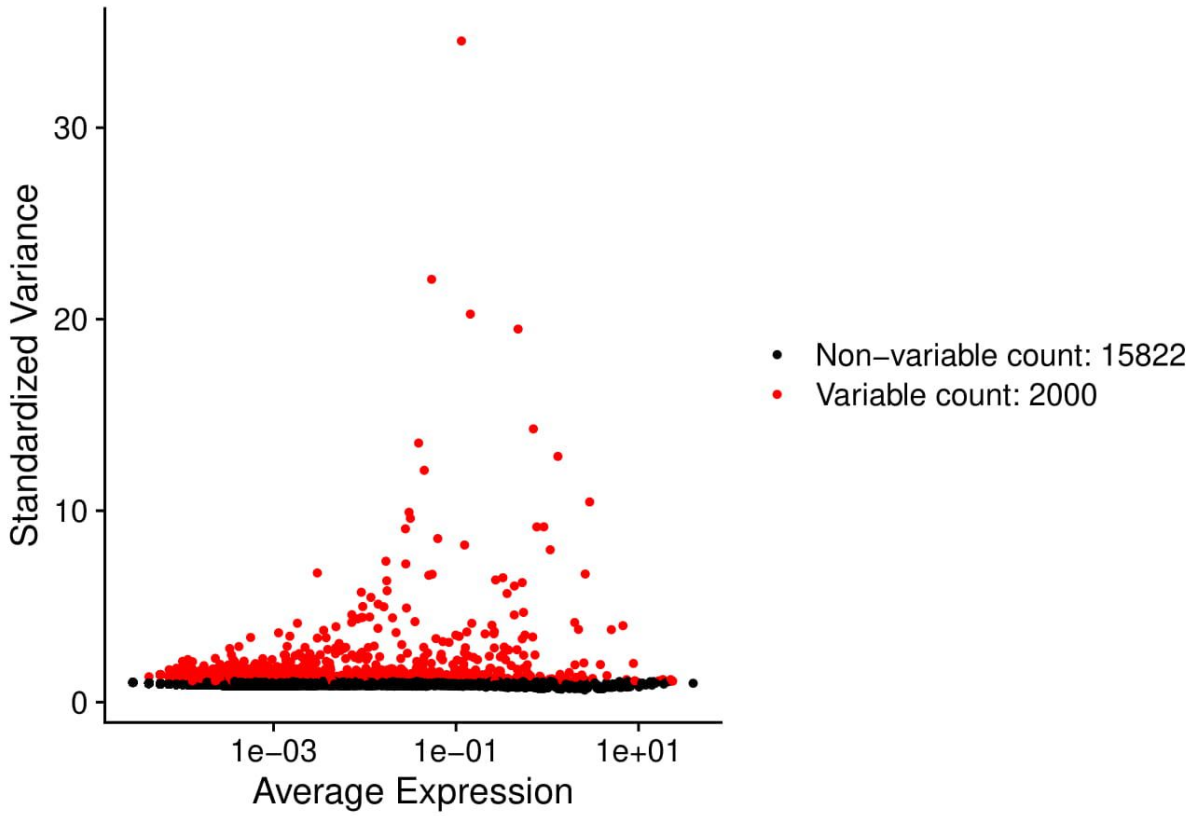
Identify the 10 most highly variable genes.

```
top10 <- head(VariableFeatures(pbmc), 10)
top10
```

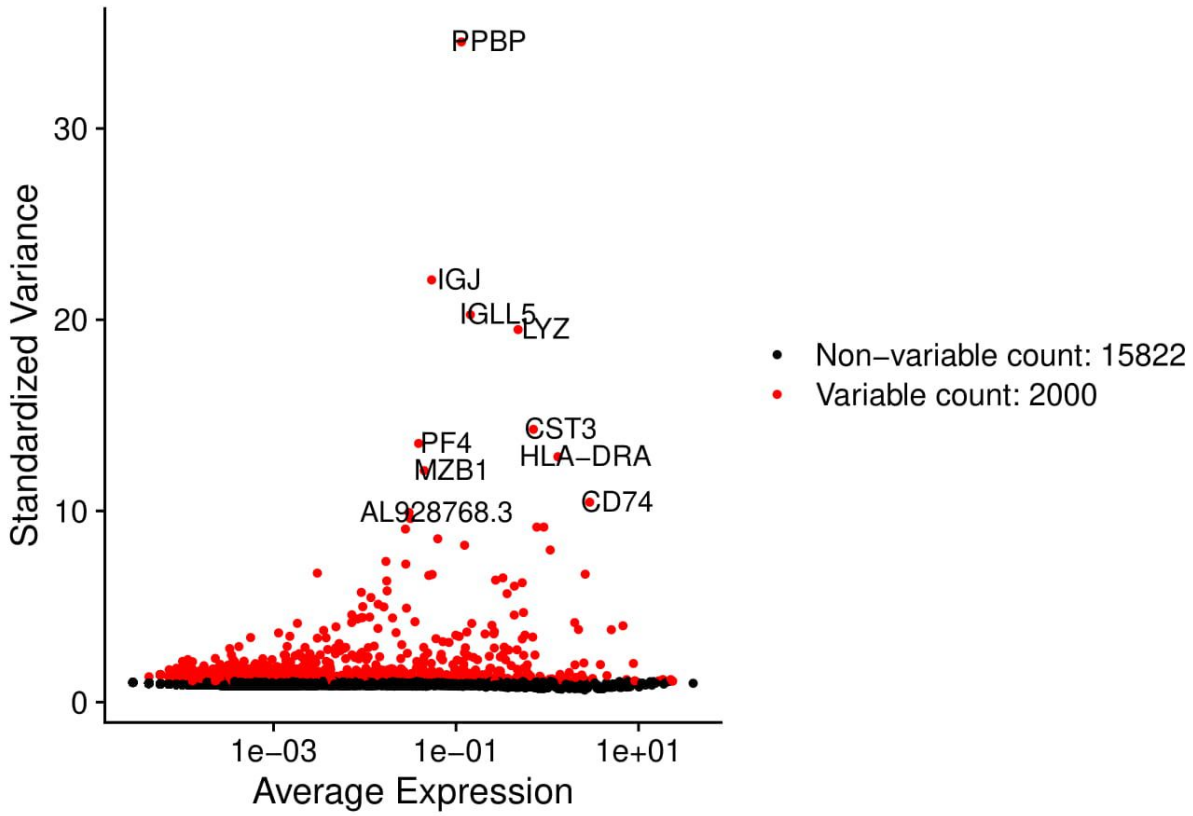
```
##  [1] "PPBP"       "IGJ"        "IGLL5"      "LYZ"        "CST3"
##  [6] "PF4"        "HLA-DRA"    "MZB1"       "CD74"       "AL928768.3"
```

Plot variable features with and without labels.

```
plot1 <- VariableFeaturePlot(pbmc)
plot1
```




```
LabelPoints(plot = plot1, points = top10, repel = F)
```



Scale the data.

```
pbmc <- ScaleData(pbmc)
```

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

Do PCA.

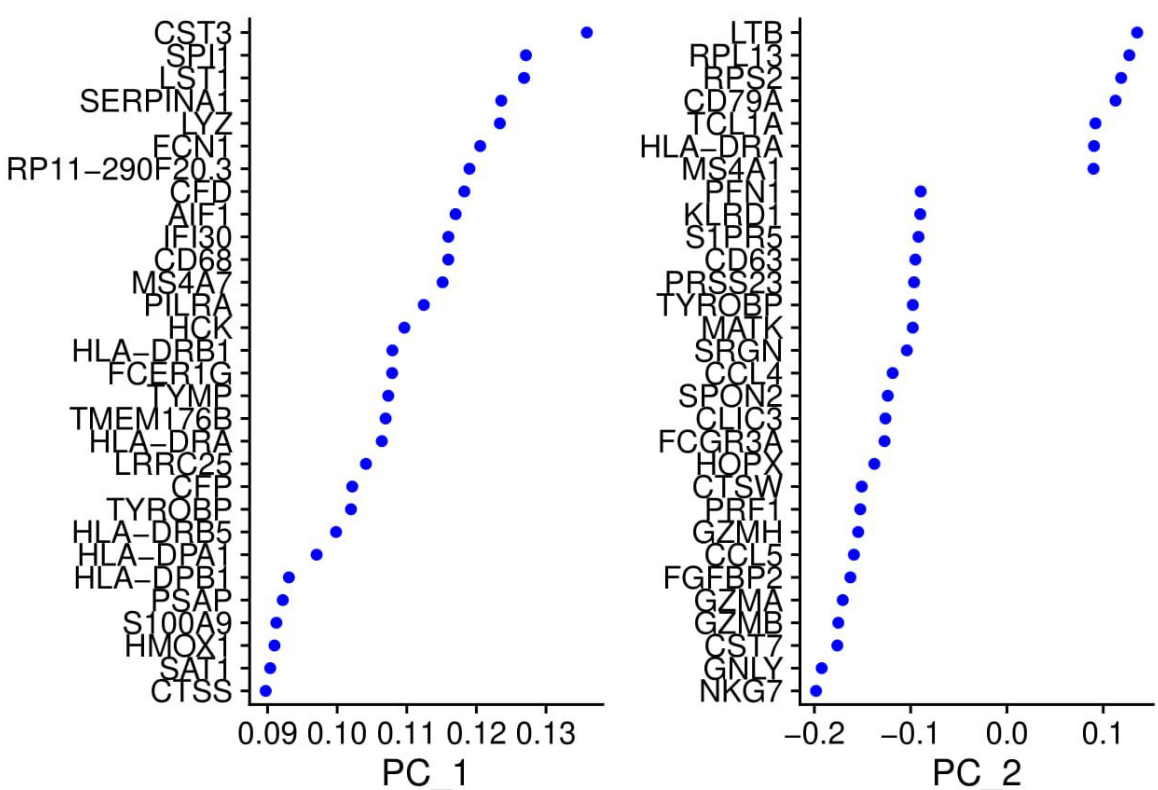
```
pbmc <- RunPCA(pbmc, features = VariableFeatures(object = pbmc))
```

```
## PC_ 1
## Positive:  CST3, SPI1, LST1, SERPINA1, LYZ, FCN1, RP11-290F20.3, CFD, AIF1, IFI30
##     CD68, MS4A7, PILRA, HCK, HLA-DRB1, FCER1G, TYMP, TMEM176B, HLA-DRA, LRRC25
##     CFP, TYROBP, HLA-DRB5, HLA-DPA1, HLA-DPB1, PSAP, S100A9, HMOX1, SAT1, CTSS
## Negative:  RPS6, RPL13, LTB, IL32, RPS2, CD7, IL7R, CCR7, JUN, CD8B
##     CTSW, AQP3, CCL5, CD8A, GZMK, RP11-291B21.2, PASK, FKBP11, GZMA, JUNB
##     NGFRAP1, MYC, CST7, HIST1H1D, DUSP2, LYAR, KLRG1, XBP1, CRIP2, NKG7
## PC_ 2
## Positive:  LTB, RPL13, RPS2, CD79A, TCL1A, HLA-DRA, MS4A1, CD79B, RPLP1, RPS6
##     LINC00926, HLA-DQA1, VPREB3, FCER2, HLA-DQA2, JUNB, HLA-DOB, SPIB, HLA-DMB, HLA-DMA
##     LY86, HVCN1, BANK1, FCRLA, EAF2, BLK, CCR7, HLA-DQB1, CD22, IGLL5
## Negative:  NKG7, GNLY, CST7, GZMB, GZMA, FGFBP2, CCL5, GZMH, PRF1, CTSW
##     HOPX, FCGR3A, CLIC3, SPON2, CCL4, SRGN, MATK, TYROBP, PRSS23, CD63
##     S1PR5, KLRD1, PFN1, IGFBP7, GPR56, RHOC, TMSB4X, IFITM2, ABI3, ID2
## PC_ 3
```

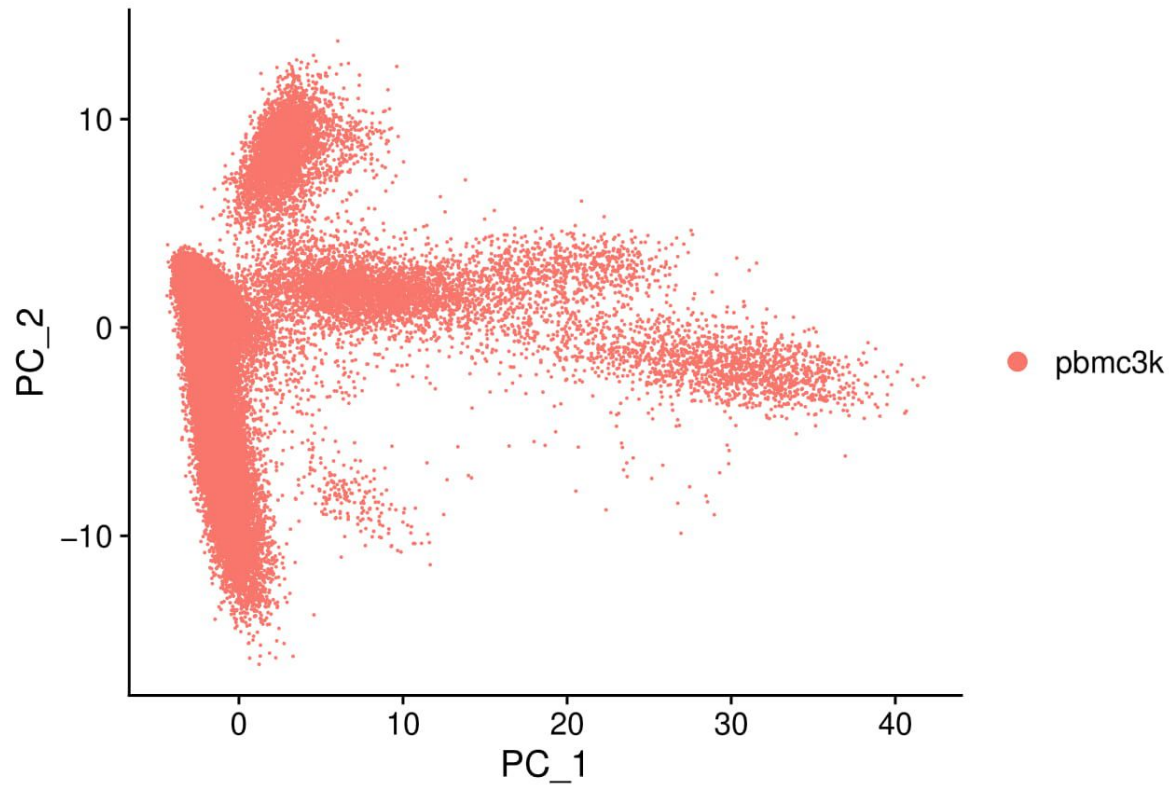
```
## Positive:  AIF1, RPL13, JUNB, CFD, RP11-290F20.3, SERPINA1, RPS2, FCN1, LST1, TMEM176B
##     S100A9, IL7R, PILRA, MS4A7, RPS6, S100A6, S100A8, IFI30, TYMP, CFP
##     HES4, LILRB2, TMSB10, CDKN1C, LRRC25, S100A11, CD68, GPBAR1, HMOX1, C1QA
## Negative:  CD79A, MS4A1, TCL1A, CD79B, HLA-DRA, HLA-DQA1, HLA-DPA1, HLA-DPB1, CD74, HLA-DQA2
##     LINC00926, HLA-DRB1, HLA-DRB5, VPREB3, FCER2, SPIB, HLA-DOB, BANK1, HLA-DMB, PDLIM1
##     HLA-DQB1, FCRLA, HLA-DMA, HVCN1, GZMB, BLK, IRF8, GNLY, NKG7, IGLL5
## PC_ 4
## Positive:  RPS2, RPL13, RPS6, TMSB10, NKG7, GNLY, CYBA, RPLP1, CST7, GZMB
##     GZMA, FGFBP2, GZMH, TYROBP, PRF1, FCGR3A, CD74, HOPX, CTSW, HLA-DRB1
##     HLA-DPB1, HLA-DRB5, HLA-DPA1, CLIC3, CCL4, HLA-DRA, SPON2, HLA-DQA1, HLA-DQA2, CD79A
## Negative:  PPBP, PF4, SDPR, GNG11, CLU, NRGN, TUBB1, HIST1H2AC, GP9, ACRBP
##     SPARC, TREML1, CMTM5, NCOA4, RUFY1, ITGA2B, PTCRA, MPP1, AP001189.4, MYL9
##     SNCA, AC147651.3, CLDN5, CD9, MAP3K7CL, RGS18, TSC22D1, TMEM40, PGRMC1, PARVB
## PC_ 5
## Positive:  MS4A7, RP11-290F20.3, VMO1, HMOX1, CDKN1C, FCGR3A, HES4, LTB, CKB, CD79B
##     ICAM4, PILRA, C1QA, LILRA3, SIGLEC10, LRRC25, CXCL16, LYN, ACTB, TESC
##     LYPD2, OAS1, GPBAR1, CD68, IFITM3, JUNB, CTSL, LILRB2, CD79A, CUX1
## Negative:  S100A8, S100A9, LGALS2, FCER1A, CD14, CST3, CLEC10A, LYZ, GPX1, MS4A6A
##     GRN, TYROBP, BLVRB, LGALS1, ALDH2, TYMP, CSF3R, S100A4, S100A6, CD1C
##     GSTP1, VCAN, TMSB10, CPVL, RPLP1, GSN, CEBPD, ENHO, S100A12, CAPG
```

Examine and visualize PCA results a few different ways

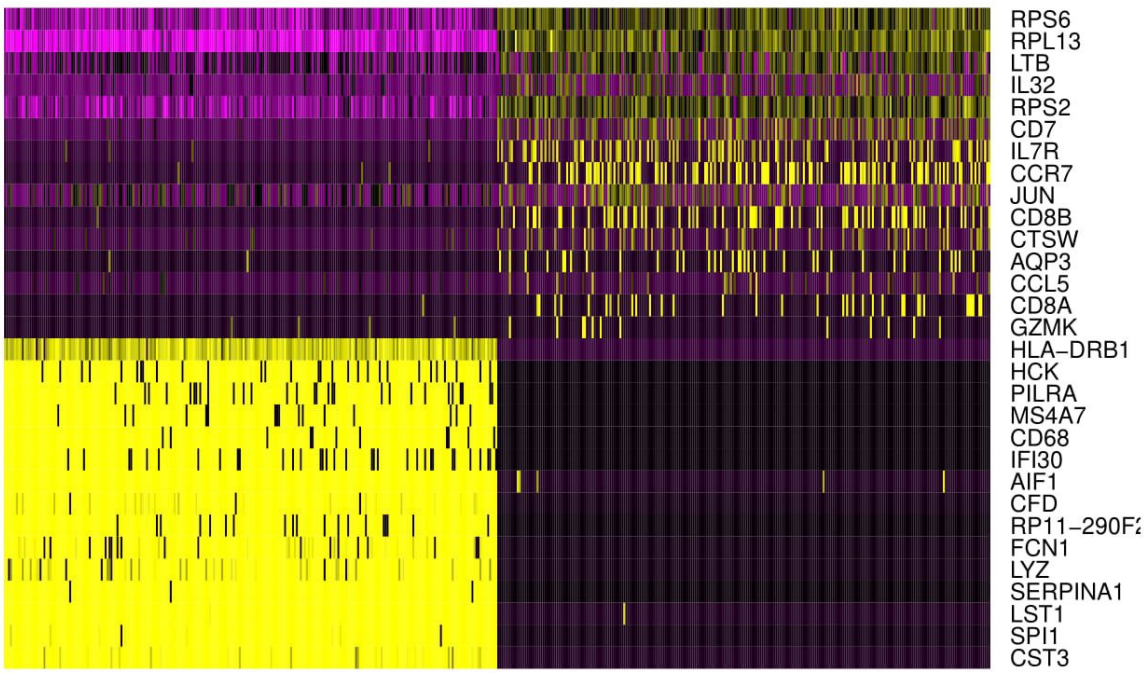
```
VizDimLoadings(pbmc, dims = 1:2, reduction = "pca")
```



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

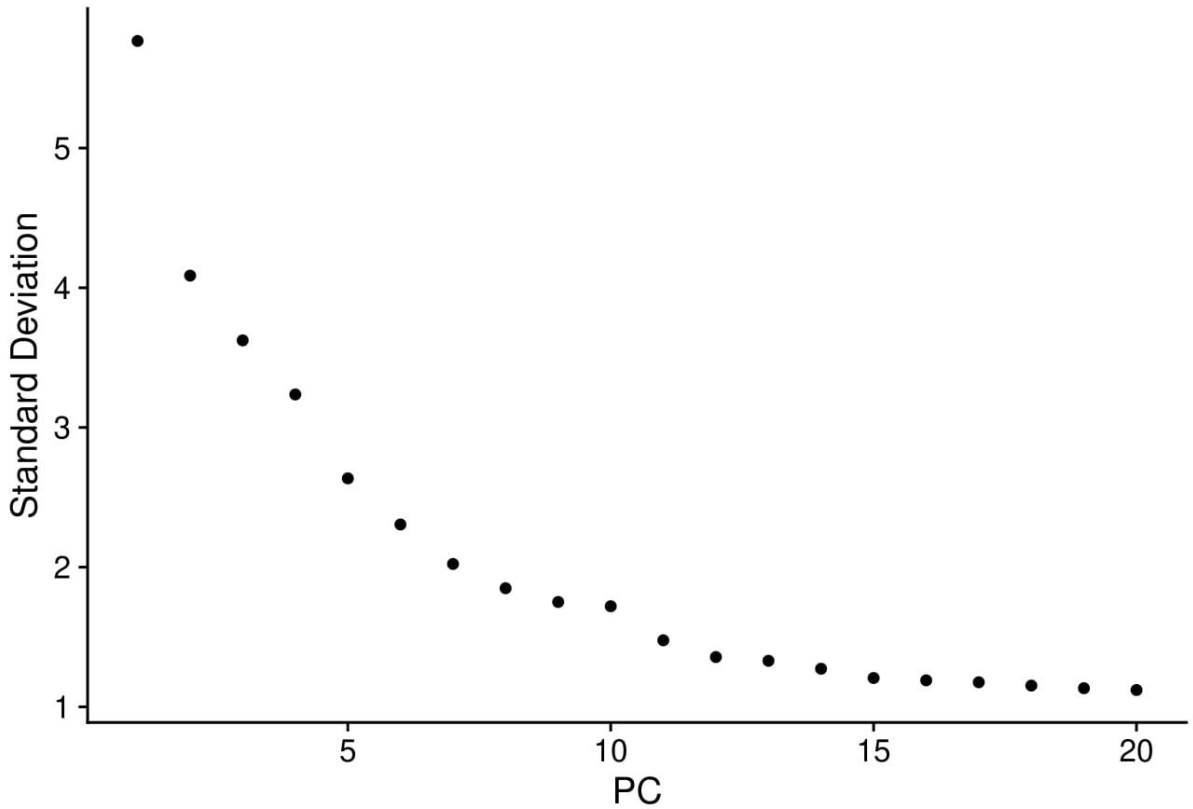



```
DimHeatmap(pbmc, dims = 1, cells = 500, balanced = TRUE)
```



We can see two elbow points - around 8. and 9. PC, and around 12. and 13.

```
ElbowPlot(pbmc)
```



Extract first **12** PCs as we chose second elbow point.

Clustering

We will use shared nearest neighbors method, because it's easier to use along with other Seurat functionalities than basic k-means one.

```
pbmc <- FindNeighbors(pbmc, dims = 1:12)
```

```
## Computing nearest neighbor graph

## Computing SNN
```

NOTE: as we found 14 communities (clusters), we will stick with this number instead of proposed 10 clusters. Additional clusters are probably small, so authors of our reference paper didn't consider them as that important, but we believe that these clusters could consist of important (however rare) features.

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

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 69567
## Number of edges: 1900651
```

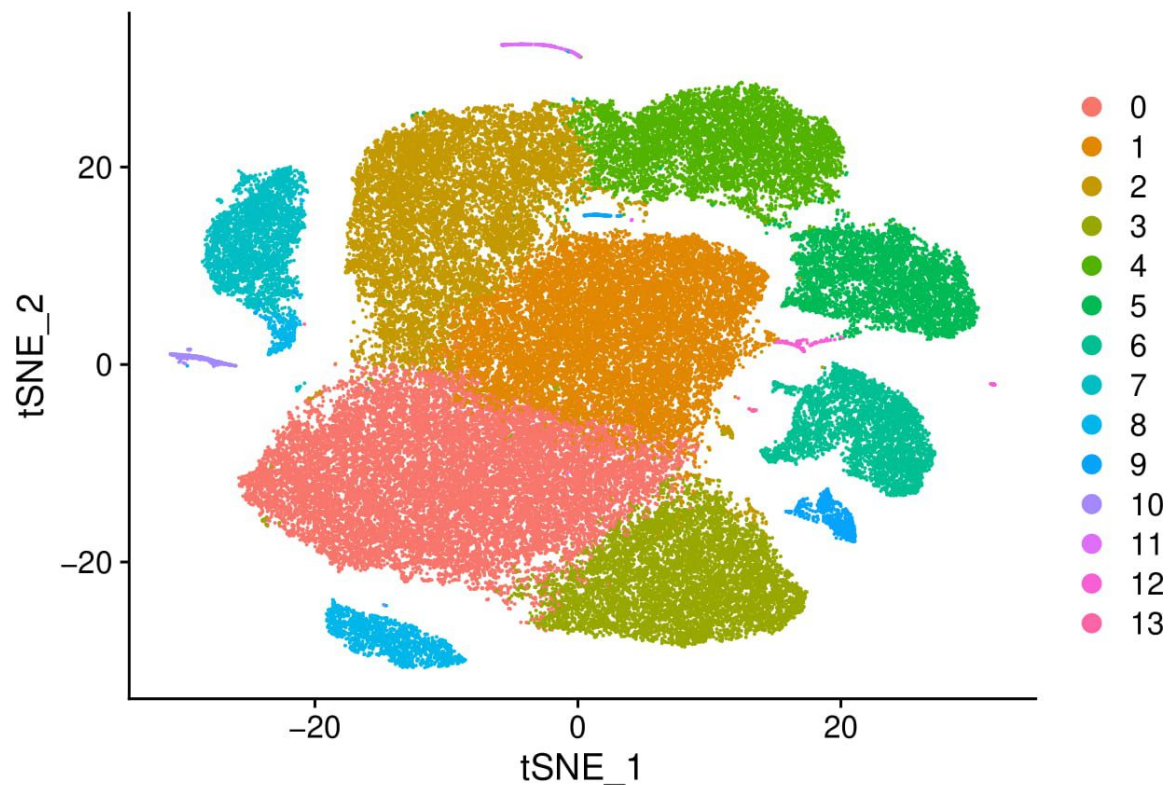
```
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.9096
## Number of communities: 14
## Elapsed time: 27 seconds
```

```
pbmc <- RunTSNE(object = pbmc, reduction = 'pca', dims = 1:12)
```

Problem 1

Reproduced Figure 3b on Zheng et al. 2017. Important chosen parameters: * **12** PCs (not 10), * **14** clusters found by **shared nearest neighbors** method (not 9, and not using k-means), * regularization done using **t-SNE**.

```
DimPlot(pbmc, reduction = 'tsne')
```



Problem 2

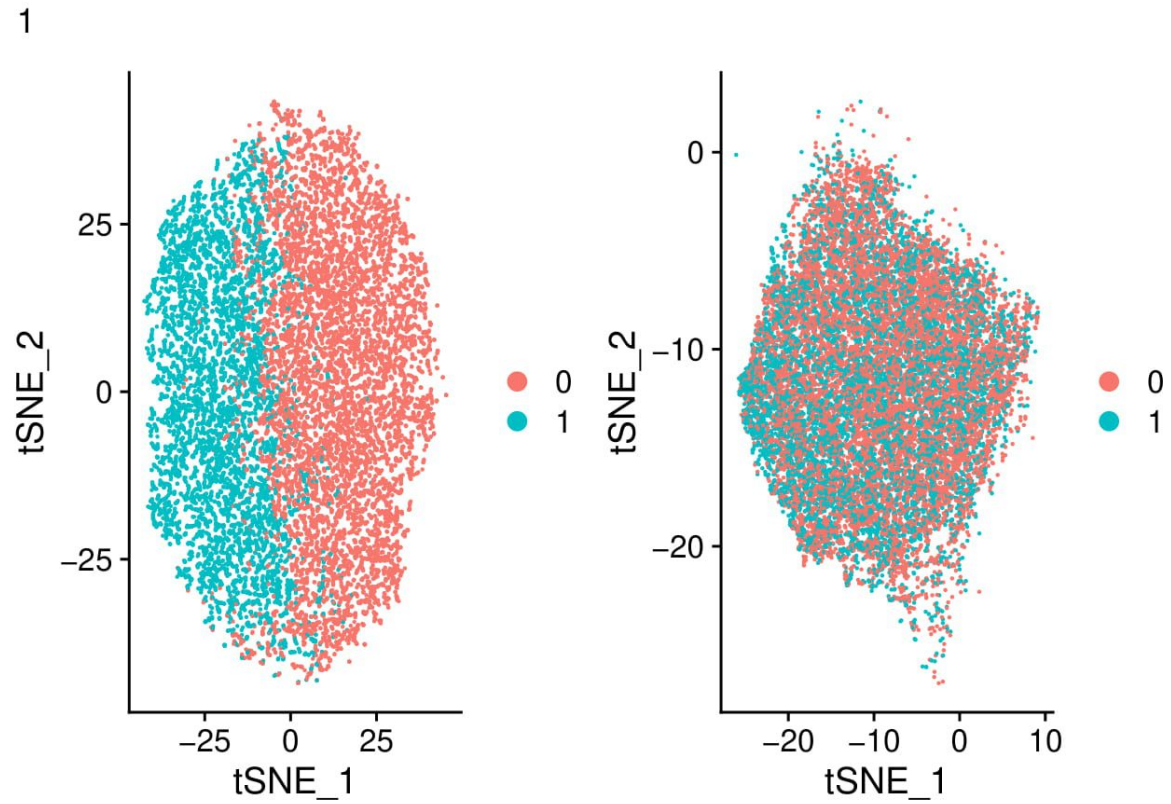
2.1 Our workflow: * subset data according to the cluster from 1st clustering, * normalize, find variable features, scale, run PCA, do clustering and t-SNE decomposition - so we repeat our main workflow on the subset of the whole data, * as we don't know, how many variables to choose (because some clusters could be much smaller/larger than others), we choose **mvp** method for computing variable features, * as making elbow plots in one for loop and choosing number of PCs in another would take a lot of time, we set number of PCs to **8** for each cluster. It's smaller than previously used 12, so it could cooperate with smaller clusters,

but, hopefully, large enough to cooperate with bigger ones as well, * we decrease **resolution = 0.3**, so we get a smaller number of subclusters, because more clusters != less clear plot with whole data in point 2.2, * we compute t-SNE reduction as well with **perplexity = 20**, because default number (30) was too big for smaller clusters.

Finally, we will make two plots for each cluster: 1. **left** - with t-SNE projection computed on subset data; it should be more informative of our subclusters, 2. **right** - with fragment of t-SNE projection computed on whole data; it should give us an overall idea, where in the whole dataset is our subset.

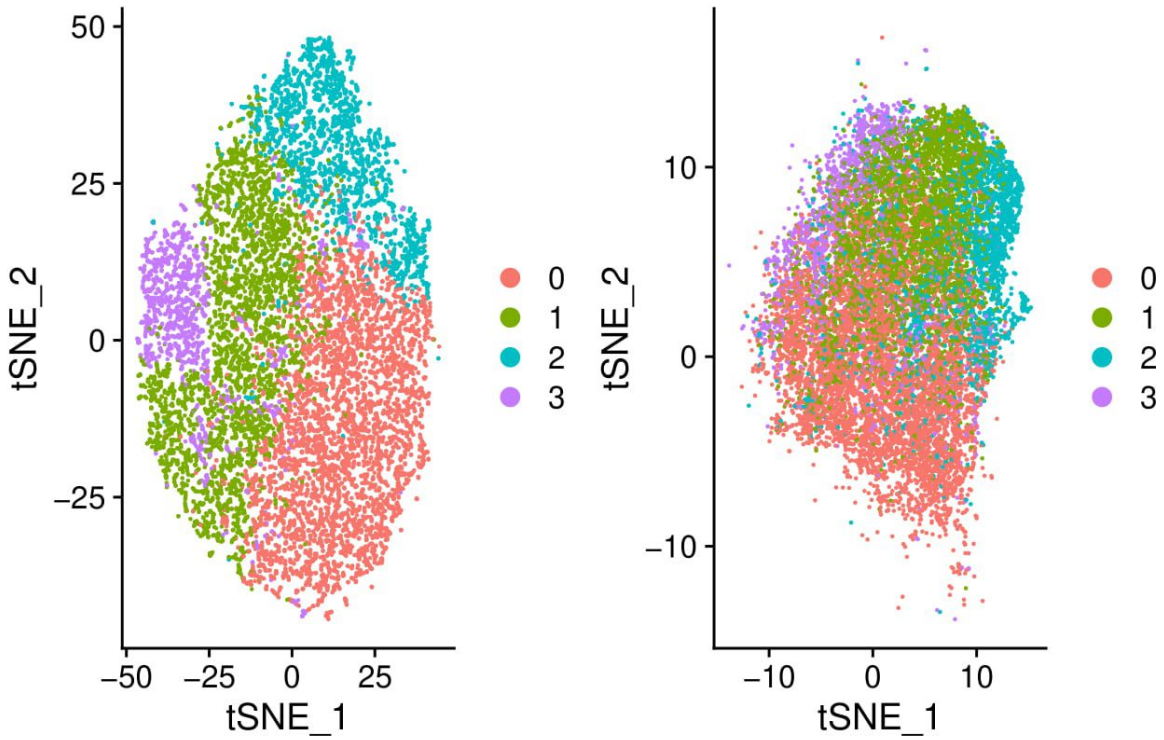
```
subsets <- vector(mode = "list", length = 14)

for (i in 1:14) {
  subset <- subset(pbmc, idents = c(i - 1))

  subset <- NormalizeData(subset, normalization.method = "LogNormalize", scale.factor = 10000)
  subset <- FindVariableFeatures(subset, selection.method = "mvp")
  subset <- ScaleData(subset)
  subset <- RunPCA(subset, features = VariableFeatures(object = subset))

  subset <- FindNeighbors(subset, dims = 1:8)
  subset <- FindClusters(subset, resolution = 0.3)

  subsets[i] <- subset

  subset_ <- RunTSNE(object = subset, reduction = 'pca', dims = 1:8, perplexity = 20)
  print(DimPlot(subset_, reduction = 'tsne') + DimPlot(subset, reduction = 'tsne') + plot_annotation(ti
}
```

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 16607
## Number of edges: 393402
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.7470
## Number of communities: 2
## Elapsed time: 6 seconds
```

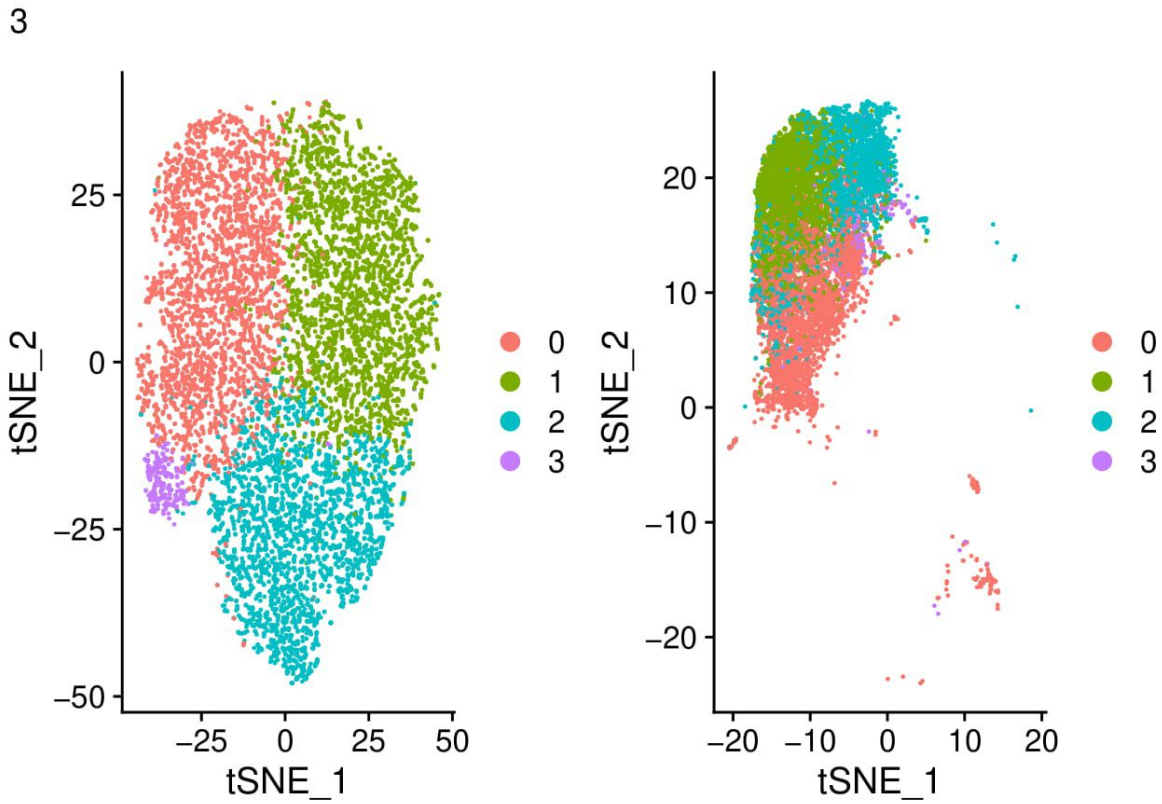



```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 13684
## Number of edges: 351759
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.7974
## Number of communities: 4
## Elapsed time: 3 seconds
```

2

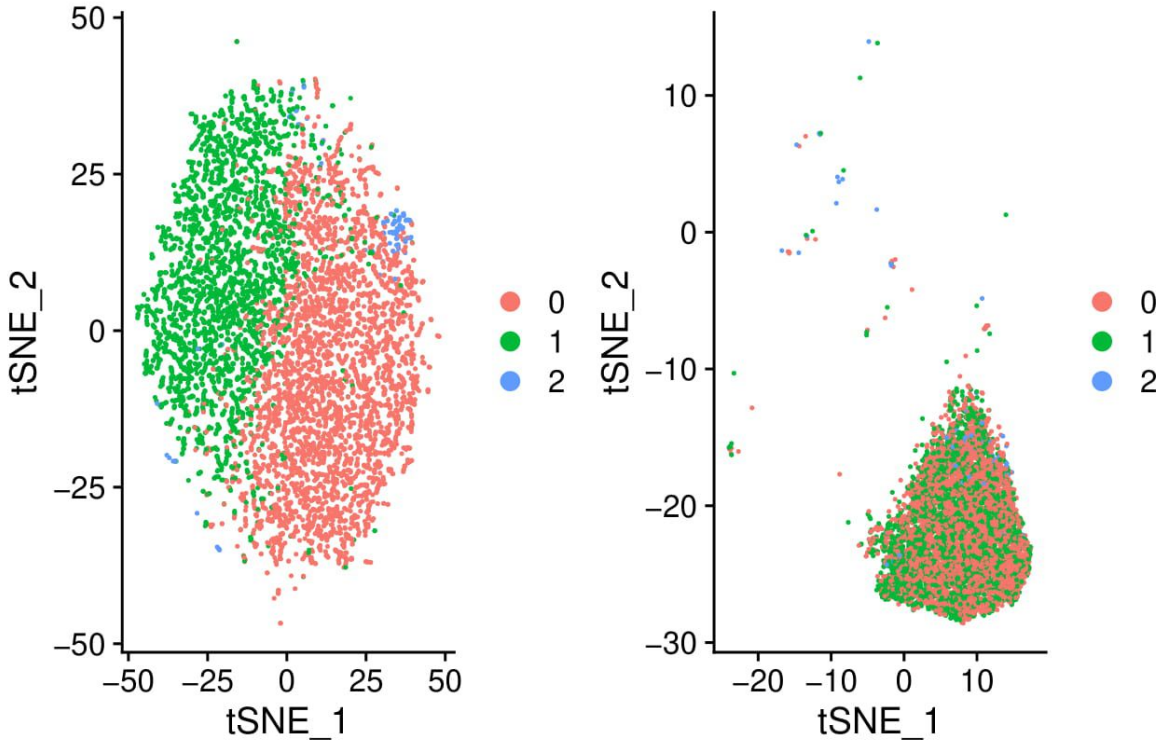



```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10183
## Number of edges: 280331
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.8348
## Number of communities: 4
## Elapsed time: 2 seconds
```



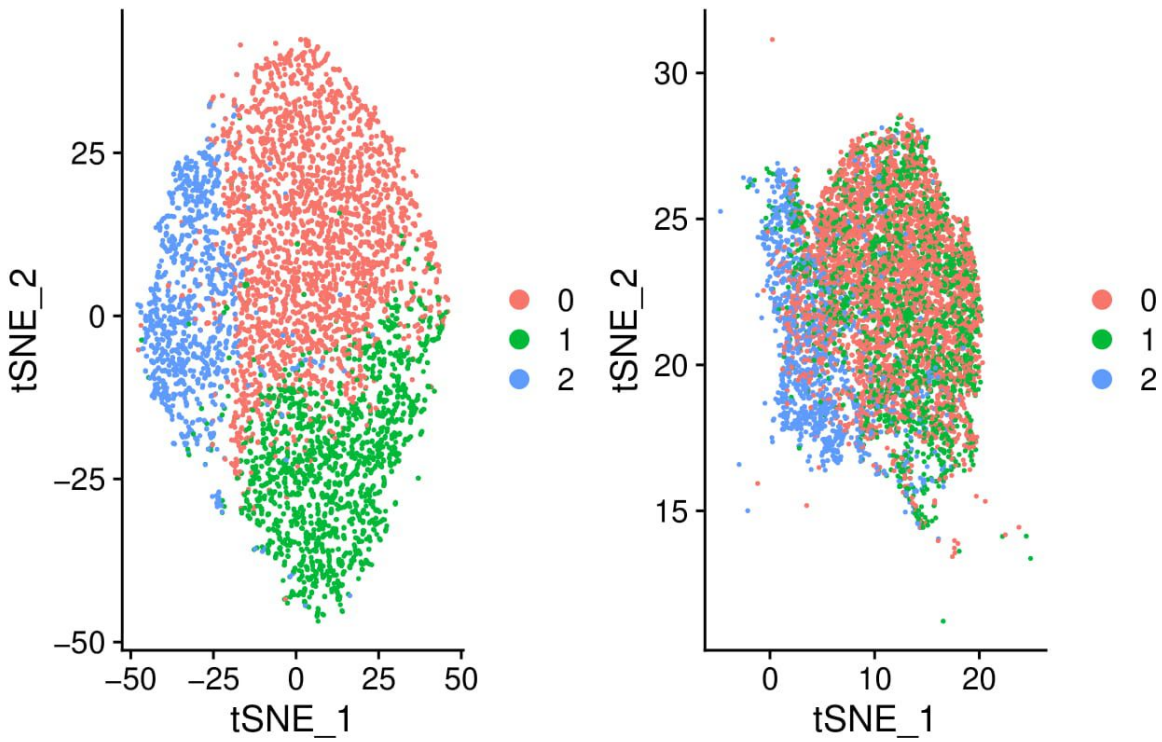

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 7354
## Number of edges: 189569
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.7427
## Number of communities: 3
## Elapsed time: 1 seconds
```

4



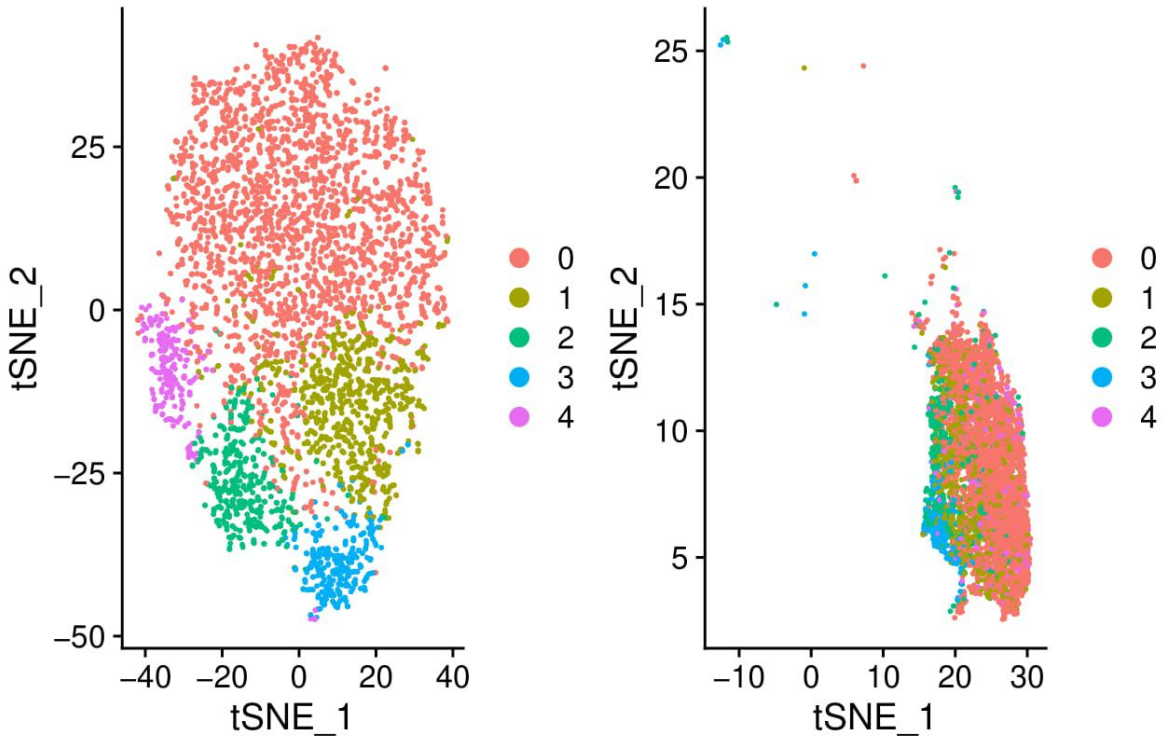

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 6589
## Number of edges: 178858
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.7698
## Number of communities: 3
## Elapsed time: 1 seconds
```

5



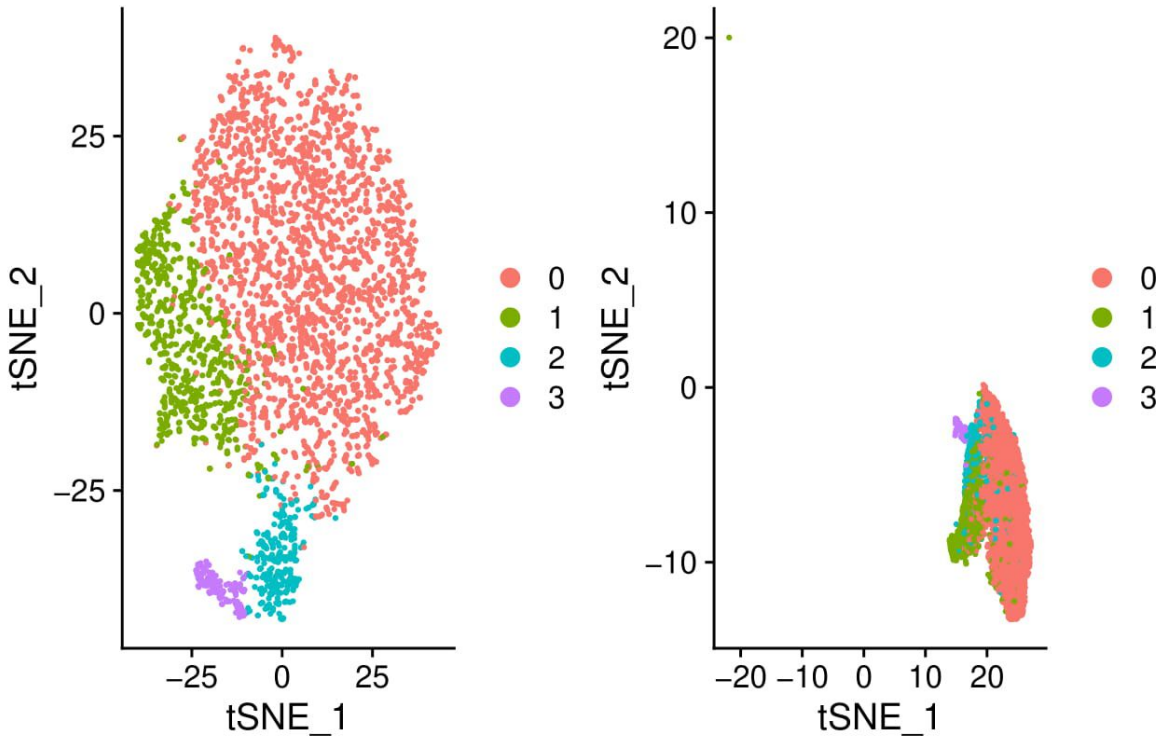

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 4615
## Number of edges: 135881
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.7945
## Number of communities: 5
## Elapsed time: 0 seconds
```

6



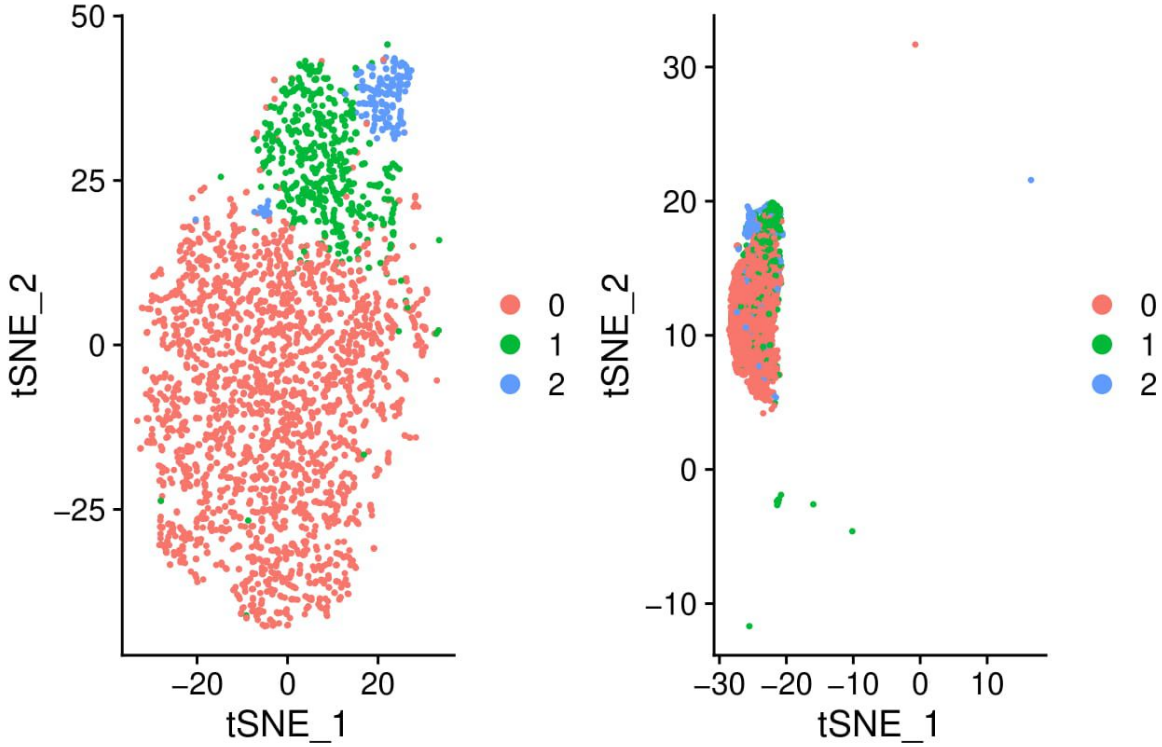

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 3871
## Number of edges: 117370
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.8114
## Number of communities: 4
## Elapsed time: 0 seconds
```

7



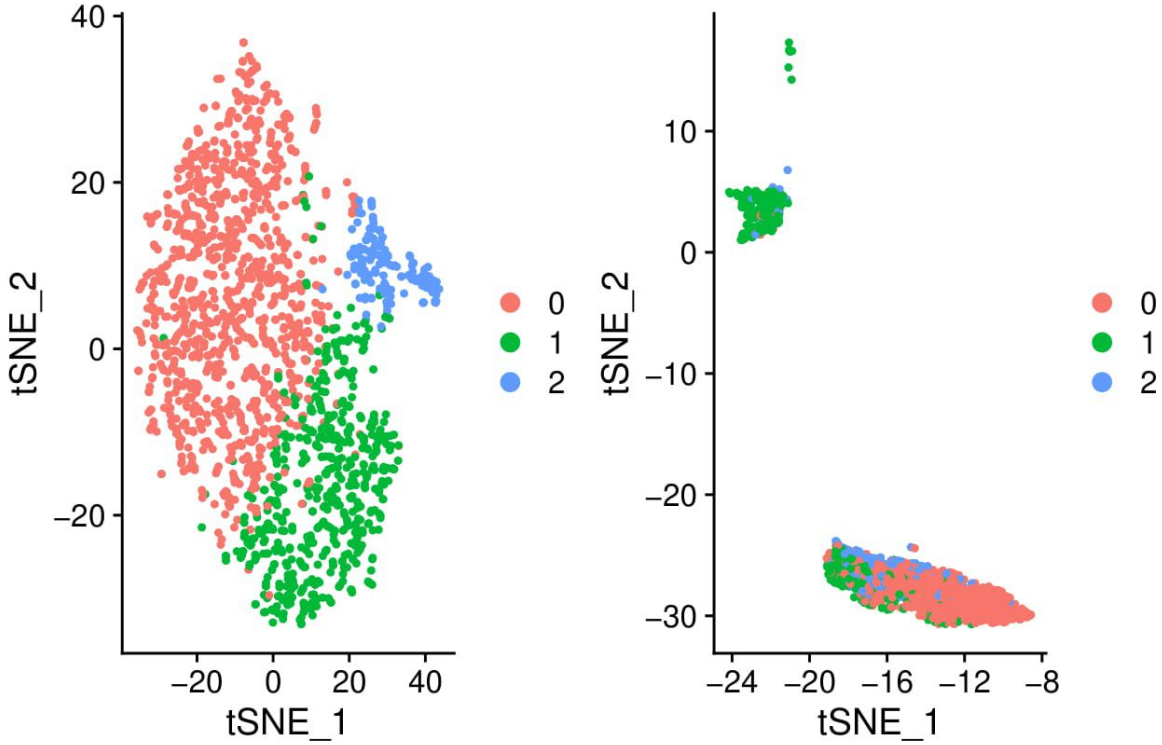

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 3125
## Number of edges: 95641
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.7710
## Number of communities: 3
## Elapsed time: 0 seconds
```

8

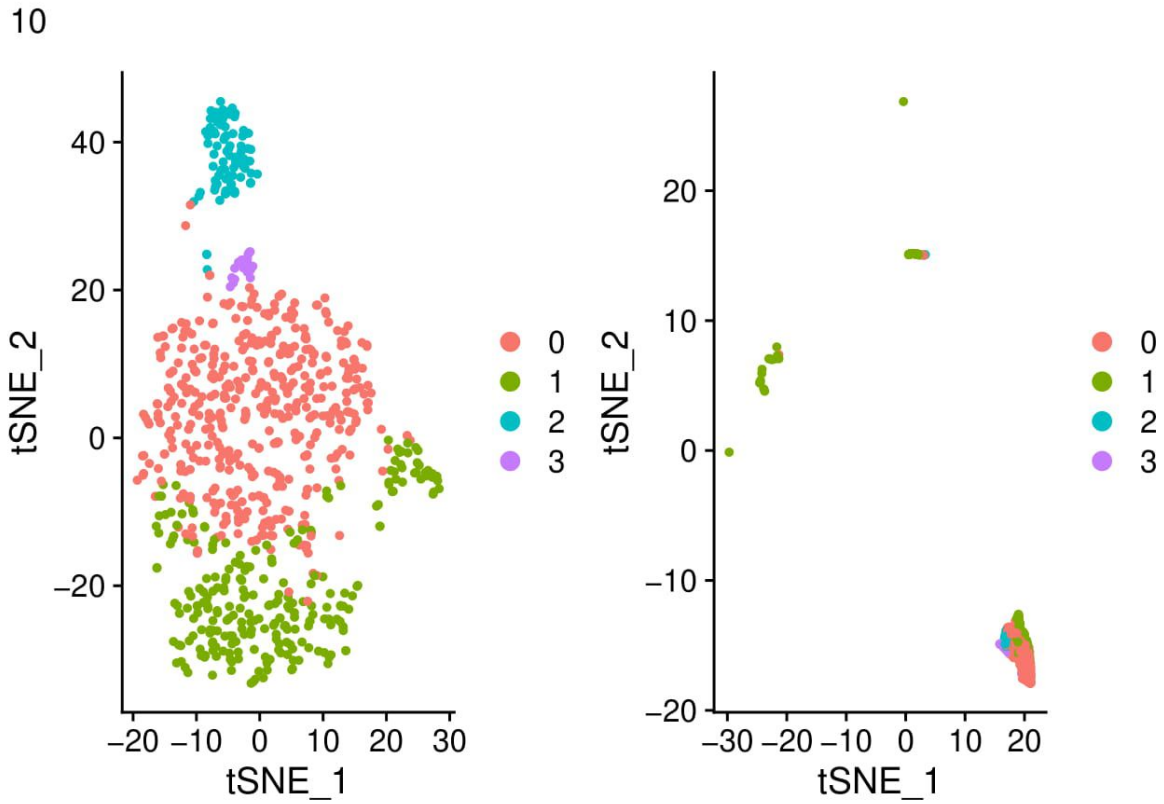



```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 1855
## Number of edges: 59983
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.7904
## Number of communities: 3
## Elapsed time: 0 seconds
```

9

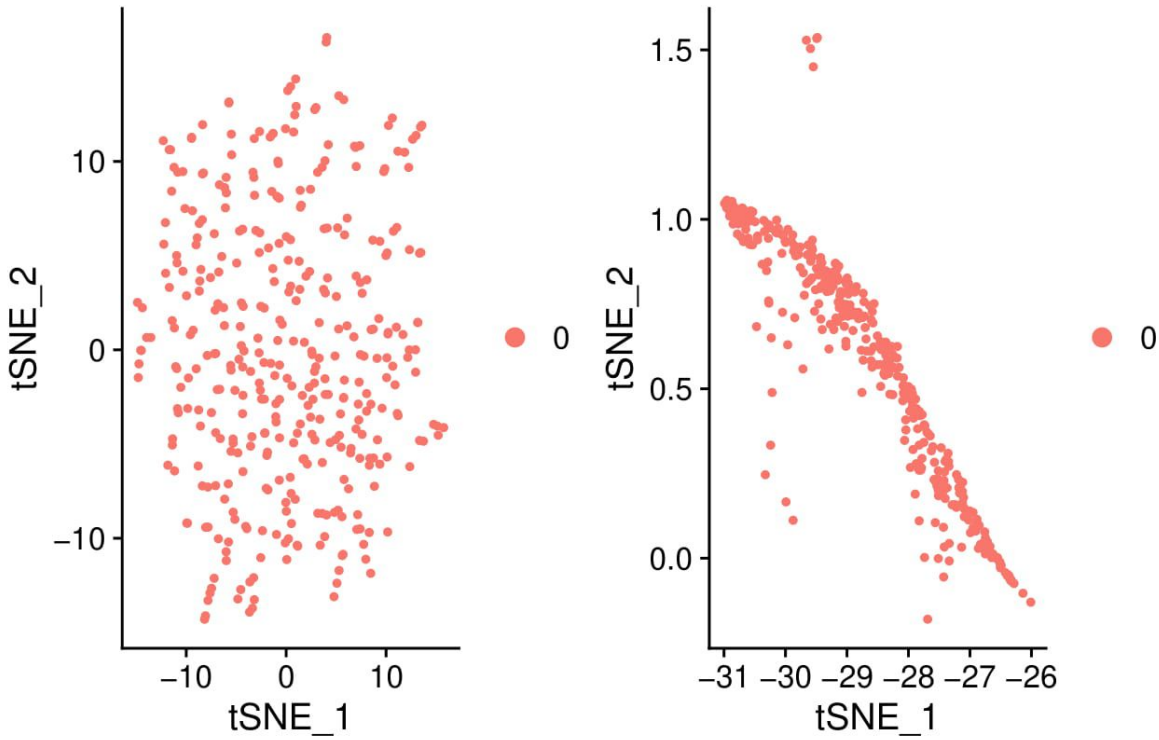



```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 792
## Number of edges: 28370
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.8087
## Number of communities: 4
## Elapsed time: 0 seconds
```



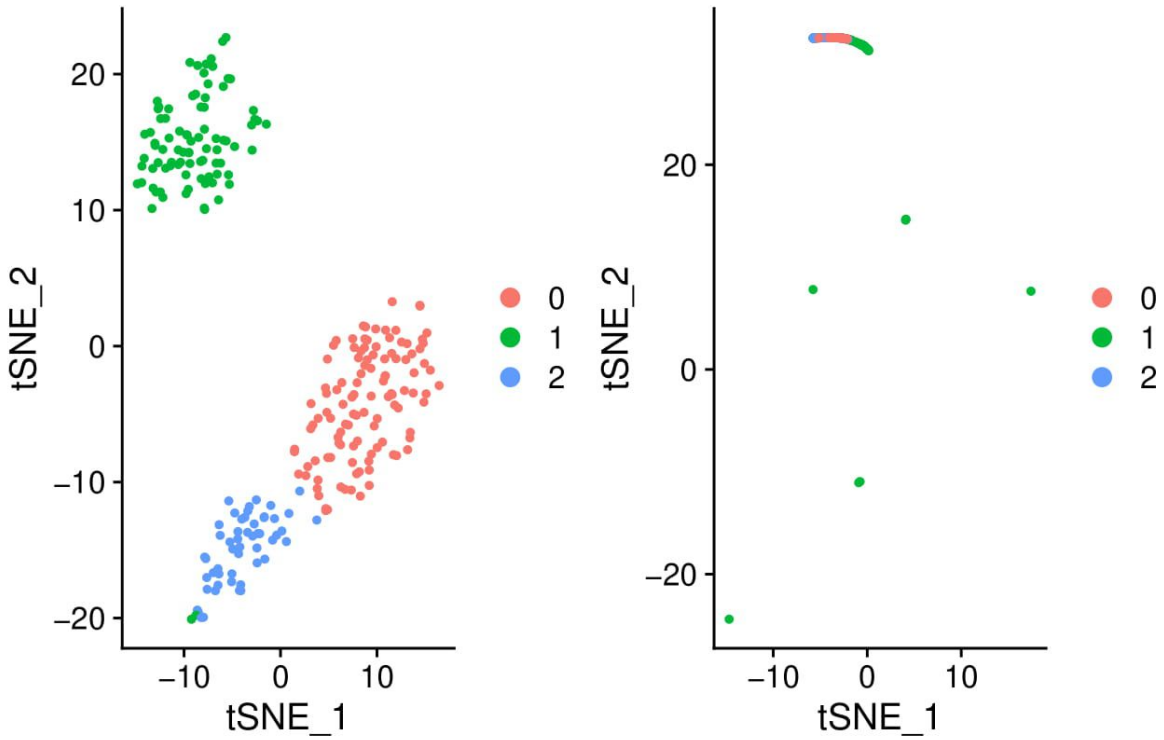

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 375
## Number of edges: 16243
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.7000
## Number of communities: 1
## Elapsed time: 0 seconds
```

11

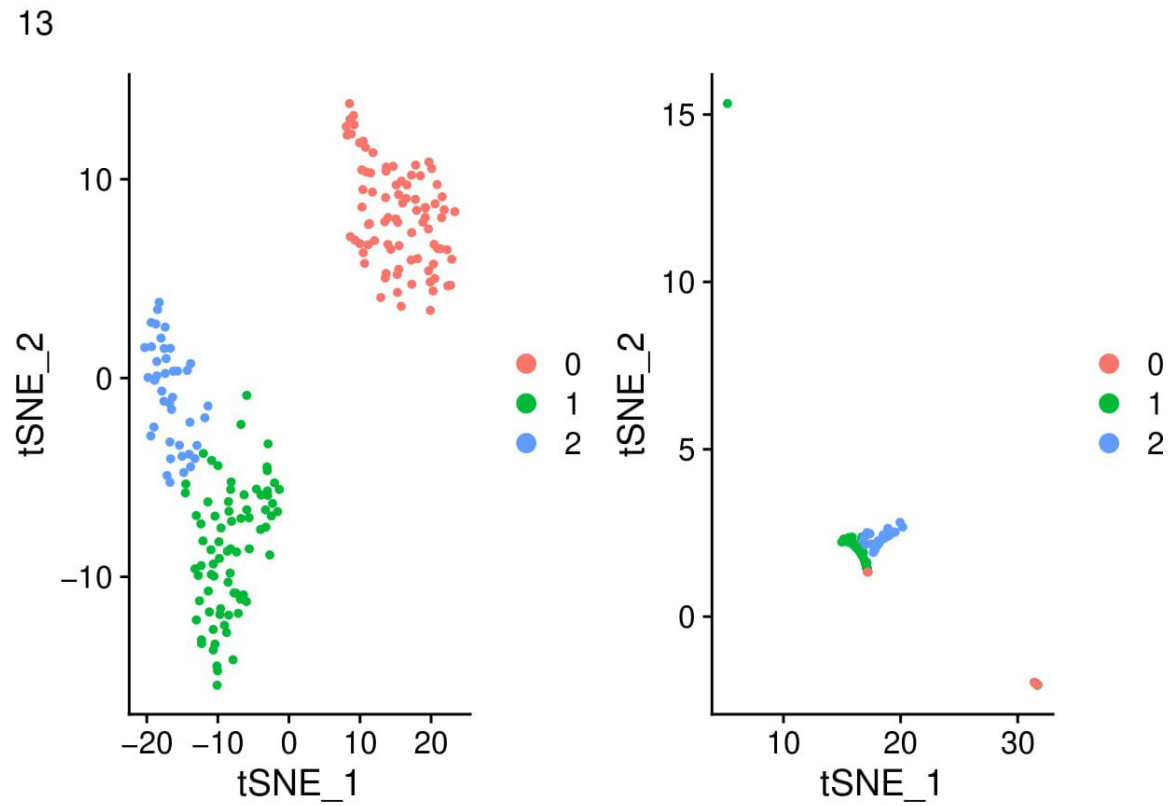



```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 251
## Number of edges: 8609
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.8658
## Number of communities: 3
## Elapsed time: 0 seconds
```

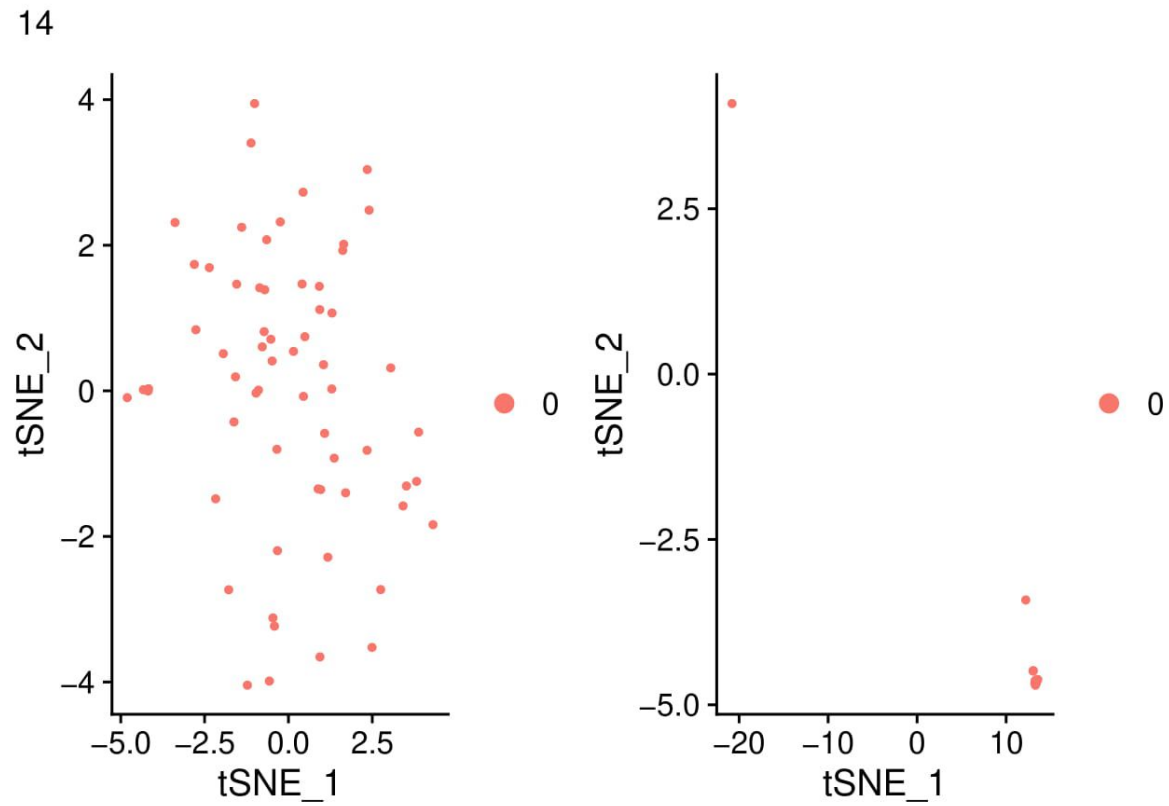
12




```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 203
## Number of edges: 5759
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.8490
## Number of communities: 3
## Elapsed time: 0 seconds
```




```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 63
## Number of edges: 1922
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.7000
## Number of communities: 1
## Elapsed time: 0 seconds
```



2.2

We need to change cluster ids, so we have no repetitions.

```
cluster_ids <- vector(mode = "list", length = 14)
feature_names <- vector(mode = "list", length = 14)

for (i in 1:14) {
  if (i == 1) {
    max_id <- 0
  } else {
    max_id <- max(cluster_ids[[i - 1]])
  }

  subset_ids <- as.numeric(Idents(subsets[[i]]))
  feature_names[[i]] <- names(Idents(subsets[[i]]))
  max_id_ <- max_id + max(subset_ids) + 1
  cluster_ids[[i]] <- subset_ids + max_id
  max_id <- max_id_
}
```

```
print(unique(cluster_ids[[1]]))
```

```
## [1] 1 2
```

```
print(unique(cluster_ids[[2]]))
```

```
## [1] 4 6 5 3
```

```
print(unique(cluster_ids[[14]]))
```

```
## [1] 43
```

We need to extract whole t-SNE data.

```
tsne_data <-pbmc@reductions$tsne@cell.embeddings
head(tsne_data)
```

```
##                       tSNE_1     tSNE_2
## AAACATACACCCAA-1 -13.441264  15.655646
## AAACATACCCCTCA-1  14.665141 -26.185281
## AAACATACCGGAGA-1   5.724358  10.373027
## AAACATACTAACCG-1  23.690490  -5.852502
## AAACATACTCTTCA-1 -11.047767 -16.235428
## AAACATACTGGATC-1 -14.794032 -15.122961
```

```
mylist <- list()
cluster_ids2 <- unlist(cluster_ids)

for (i in 1:dim(tsne_data)[1]) {
  mylist[[i]] <- c(tsne_data[i, 'tSNE_1'], tsne_data[i, 'tSNE_2'], cluster_ids2[i])
}
df <- do.call("rbind", mylist)
```

```
colnames(df) <- c('tsne1', 'tsne2', 'cluster')
rownames(df) <- unlist(feature_names)
df <- as.data.frame(df)
```

```
head(df)
```

```
##                        tsne1      tsne2 cluster
## AAACATACTCTTCA-1 -13.441264  15.655646       1
## AAACATACTGGATC-1  14.665141 -26.185281       2
## AAACATTGGTTCAG-1   5.724358  10.373027       1
## AAACCGTGTCGCTC-1  23.690490  -5.852502       1
## AAACCGTGTTTCAC-1 -11.047767 -16.235428       2
## AAACGCTGAGCCAT-1 -14.794032 -15.122961       2
```

```
df <- df[with(df, order(cluster)), ]
```

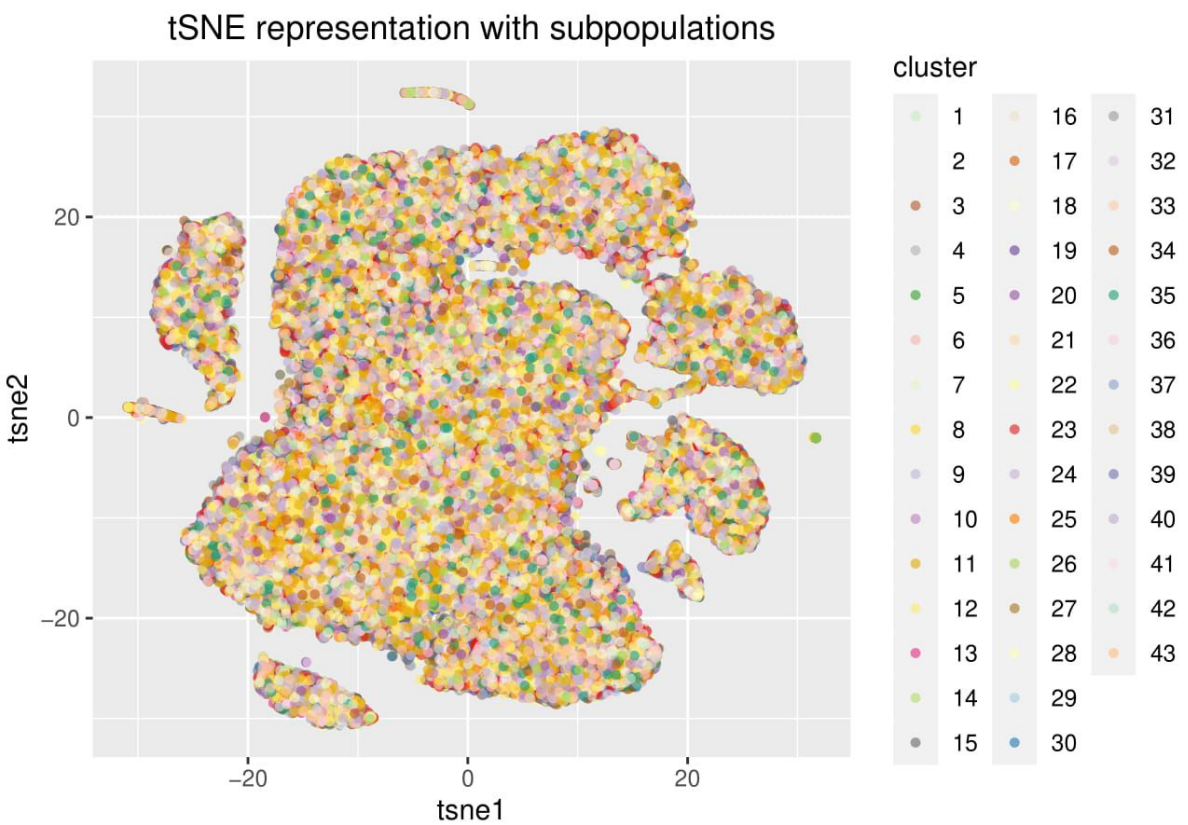
```
library(RColorBrewer)
n <- 50
qual_col_pals = brewer.pal.info[brewer.pal.info$category == 'qual',]
col_vector = unlist(mapply(brewer.pal, qual_col_pals$maxcolors, rownames(qual_col_pals)))
col_vector <- unique(sample(col_vector, n))
```

```
df$color <- apply(df, MARGIN = 1, FUN = function(row) {
    return (col_vector[as.numeric(row[3])])
  })
```

```
library(ggplot2)

plot <- ggplot(df) +
    geom_point(aes(x = tsne1, y = tsne2, color = color), pch = 16, alpha = 0.6) +
    scale_colour_manual(name = "cluster",
                        labels = unique(df$cluster),
                        values = unique(df$color)) +
    ggtitle("tSNE representation with subpopulations") +
    theme(plot.title = element_text(hjust = 0.5))
```

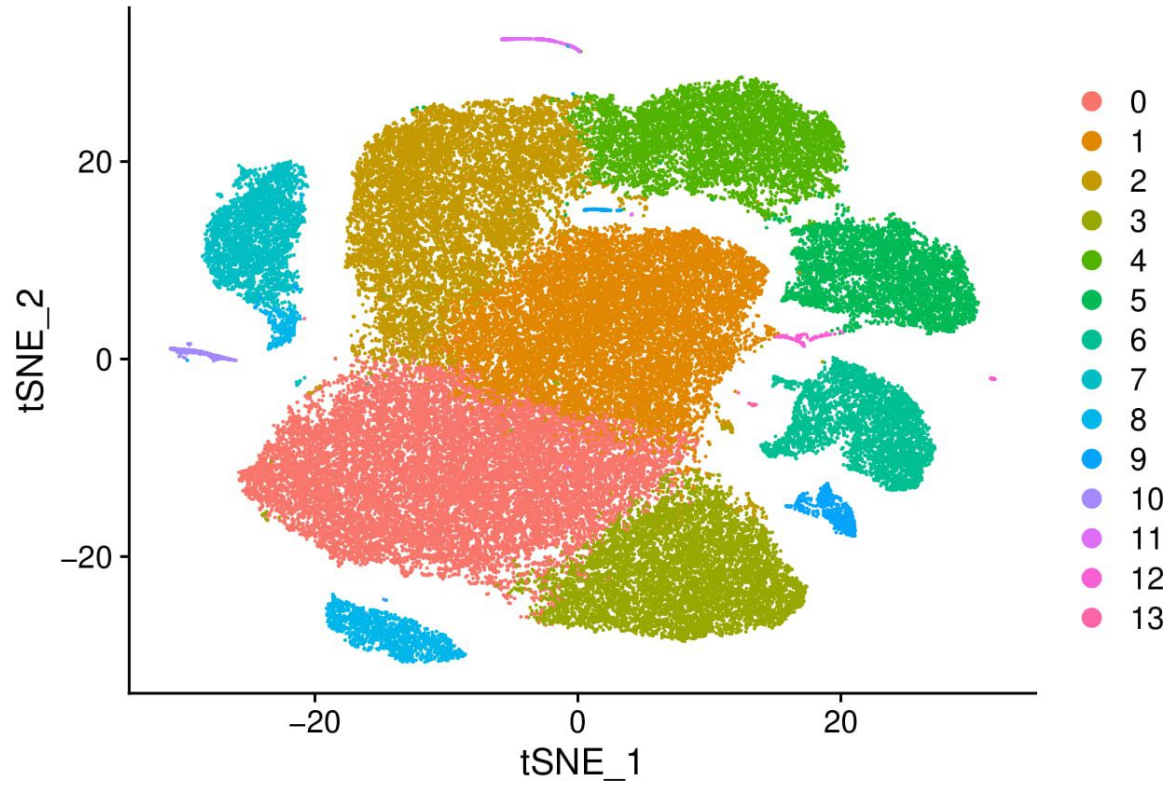
Well, it's hard to see anything, because we have **43** (sic!) clusters. But we know from the single plots (2.1) that our subclusters looked good.

```
plot
```



We can compare this plot to the first level of clustering.

```
DimPlot(pbmc, reduction = 'tsne')
```



Appendix: failed attempts of using k-means (it's not compatible with Seurat's format)

```
#library(caret)
library(cluster)
pca <- pbmc@reductions$pca@feature.loadings[, 1:12]
pca[1:5, ]
```

Find an optimal number of cluster using silhouette score. **Silhouette score** is a metrics grading goodness of the clustering without known ground truth, so when true cluster labels are unknown. It computes a score for each observation (each gene in our case). Score close to 1 means that a gene is well-clustered, score near 0 means that gene is somewhere between two clusters, and a negative score means that a gene should probably belong to another cluster. We will compute score based on **euclidean** distance between computed PCs.

```
silhouette_score <- function(k){
  km <- kmeans(pca, centers = k)
  # compute a silhouette score for each clustering, based on euclidean distance matrix computed on PCs
  ss <- silhouette(km$cluster, dist(pca))
  return (c(mean(ss[, 3]), km$cluster))
}
```

Unfortunately, this k-means implementation is not stable - results depend on initialization of first cluster centers. That's why we will set a random seed (to ensure reproducibility), but be aware that you could get another optimal k (number of clusters) with another random seed.

```r
set.seed(32)

k <- 2:16
results <- sapply(k, silhouette_score)
plot(k, type = 'b', results[1, ], xlab = 'Number of clusters', ylab = 'Average Silhouette Scores', frame
```

With chosen random seed, optimal number of clusters is equal to **13** (it's where we can see a peak in silhouette score). **Because of our chosen metrics, we will use 13 clusters from now on (not 10 like suggested in the homework's description).**

```r
# different enumeration
basic_clusters <- results[2:2001, 13 - 1]
```

Now we can visualize our data using t-SNE.

```r
pbmc <- RunTSNE(object = pbmc, features = pbmc@reductions$pca@feature.loadings, dims = 1:12)
```

```r
DimPlot(pbmc, reduction = 'tsne')
```

```r
library(ggplot2)

tsne_data <- as.data.frame(pbmc@reductions$tsne@cell.embeddings)
tsne_data

ggplot(tsne_data, aes(reductions$tsne@cell.embeddings[, 1], reductions$tsne@cell.embeddings[, 2]))
```

Cluster data.

```r
km <- kmeans(pbmc@reductions$pca@cell.embeddings[, 1:12], centers = 13)
km$cluster
#ss <- mean(silhouette(km$cluster, dist(pbmc@reductions$pca@cell.embeddings[, 1:12]))[, 3])
```

```r
library(tsne)
tsne_data <- tsne(pca, initial_dims = 1:12, k = 2)
```

Look at cluster IDs of the first 5 cells

```r
plot(tsne_data)
```