STAT646-Midterm

Jaeyoung Cho

Question 1. Part 1: Segerstolpe et al.

```
##          : SeuratObject

##          : sp

##
##          : 'SeuratObject'

## The following object is masked from 'package:base':
##
##     intersect

##
##          : 'dplyr'

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

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

Setup for Seger data with lines from midterm_kickstart

```
##
##   H1   H2   H3   H4   H5   H6 T2D1 T2D2 T2D3 T2D4
##   96  352  383  383  383  383  383  383  384  384
```

Hint 1: Reordering the cells

```
col.order <- cell.meta$Source.Name
count <- count[, col.order]

all(colnames(count)==cell.meta$Source.Name)
```

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

The cells in both count matrix and cell.meta were reordered in alphabetical order so that they match each other.

Removing the Duplicates

Next, we remove the duplicates in count matrix, so that we can successfully create a Seurat object.

```
count_uniq <- count[!duplicated(rownames(count)), ]
dim(count_uniq)
```

```
## [1] 25526  3514
```

```
all(colnames(count_uniq)==cell.meta$Source.Name)
```

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

Creating a Seurat object

```
Seger <- CreateSeuratObject(counts = count_uniq, project = "Segerstolpe", meta.data = cell.meta)
```

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

```
## Warning: Data is of class matrix. Coercing to dgCMatrix.
```

```
dim(Seger)
```

```
## [1] 25526  3514
```

We have successfully created a Seurat object, and the description of it does not show any significant sign of error.

Hint 2: Selecting only the healthy individuals

Next, we leave only the healthy individuals and remove the T2D patients. From running the table() function on cell.meta object we have noticed that we can distinguish T2D patients based on whether we can find 'T2D' in it.

```
Seger <- Seger[, grep("T2D", colnames(Seger), invert = TRUE)]
dim(Seger)
```

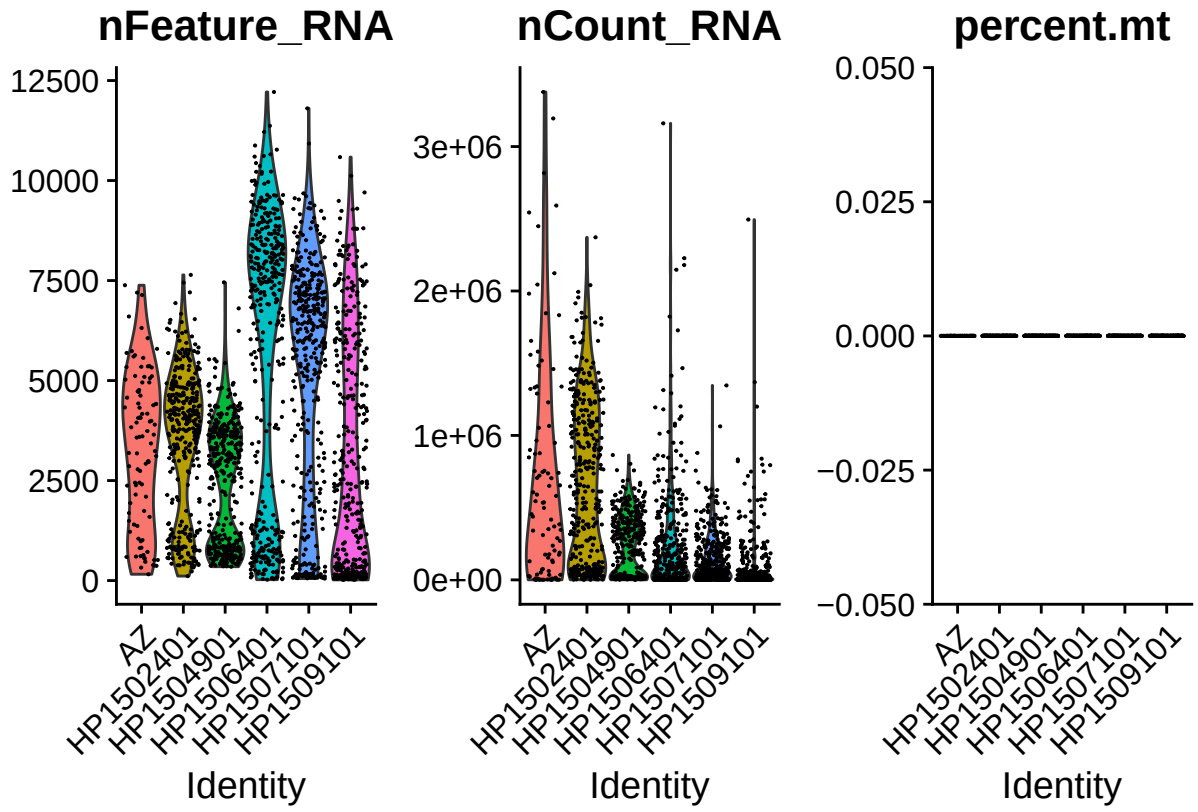
```
## [1] 25526  1980
```

QC for Seurat Object - Removing outliers

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

VlnPlot(Seger, 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 in SingleExIPlot(type = type, data = data[, x, drop = FALSE], idents =
## idents, : All cells have the same value of percent.mt.
```



While we take the standard procedure of setting QC thresholds to remove outliers, mt percentage does not display specific pattern and are all equal to zero. Nonetheless, from the violin plot, we can set threshold for number of features, and for this project, we take a conservative approach and set the lower bound as 1000 and upper bound as 11000.

```
Seger <- subset(Seger, subset = nFeature_RNA > 1000 & nFeature_RNA < 11000)
dim(Seger)
```

```
## [1] 25526  1483
```

Normalization & dimension reduction

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

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

```
Seger <- FindVariableFeatures(Seger, selection.method = 'vst', nfeatures = 2000)
```

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

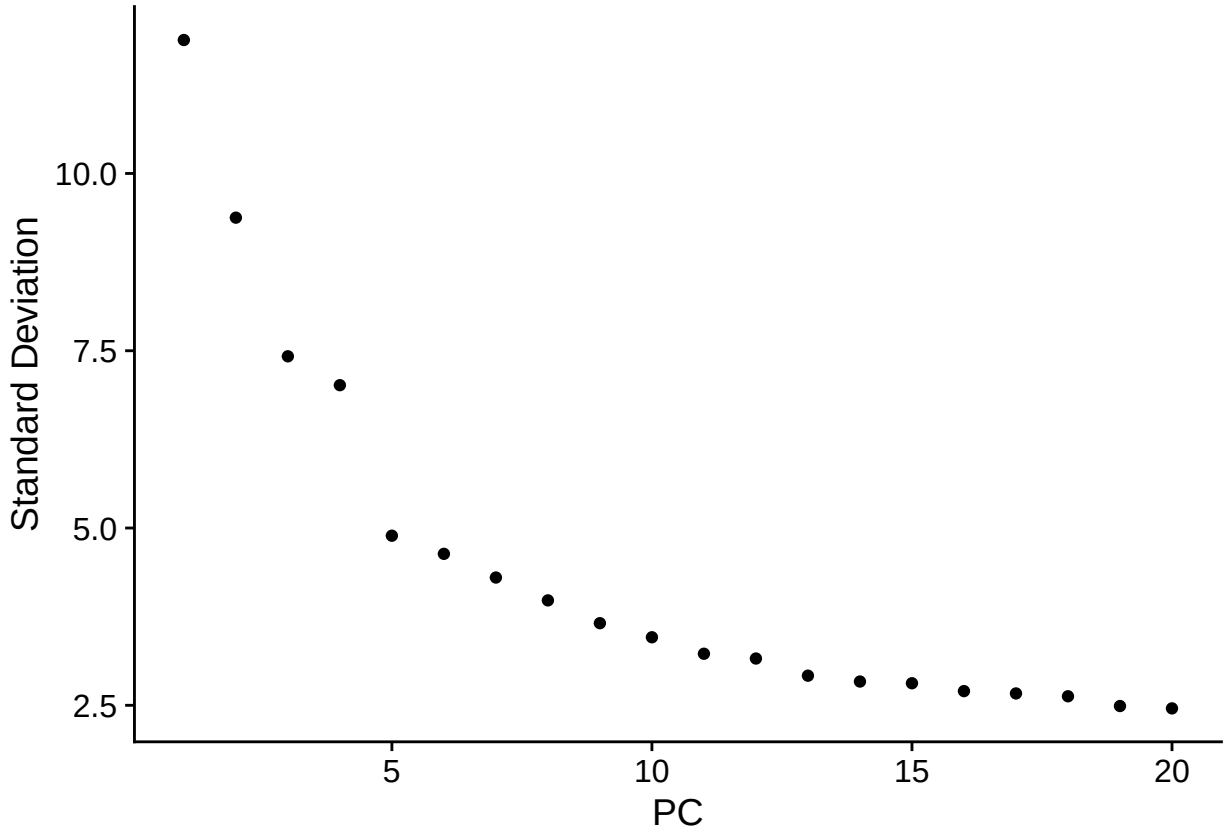
```
Seger <- ScaleData(Seger, features = rownames(Seger))
```

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

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

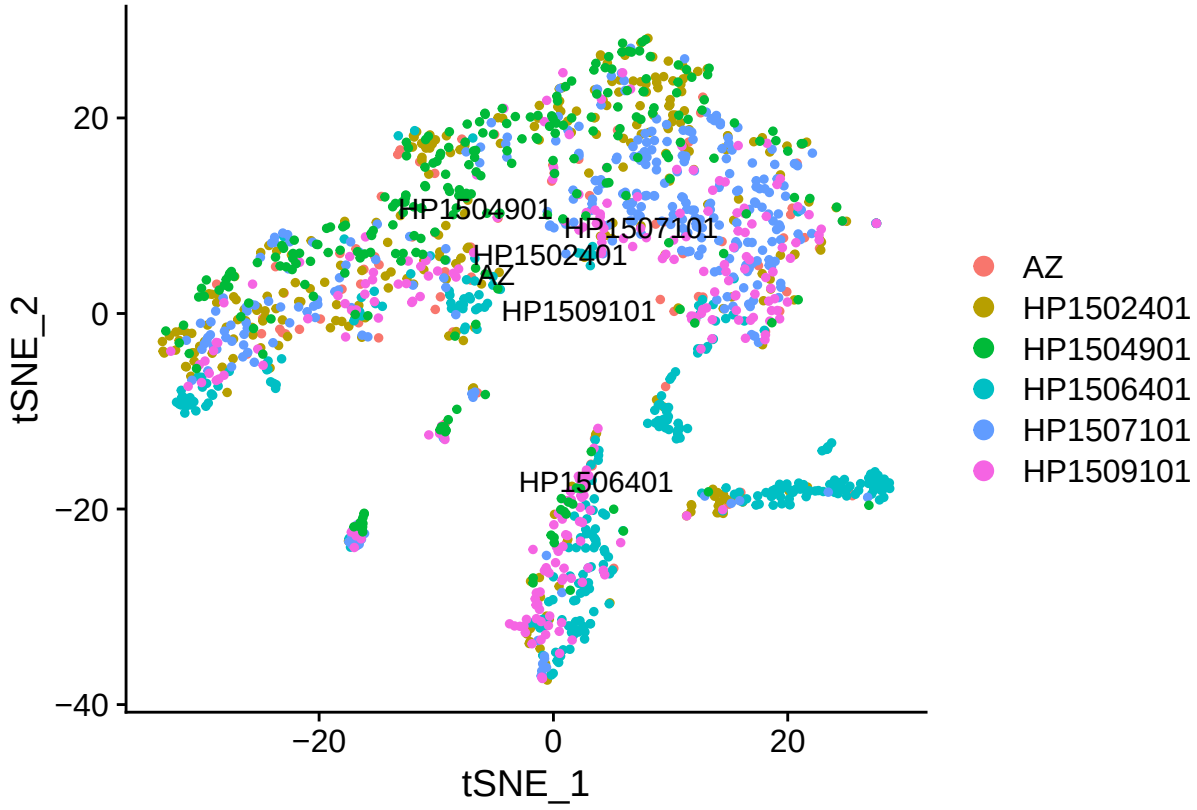
```
## PC_ 1
## Positive:  IFITM3, SERPING1, CTSH, RHOC, PRSS8, CDC42EP1, KRT7, TACSTD2, DHRS3, LCN2
##     SDC4, ANXA4, CLDN1, TMSB4X, ZFP36L1, LGALS3, CD24, S100A11, SERPINA3, CFB
##     LAD1, LITAF, KRT18, CLDN10, TM4SF1, SAT1, SERPINA5, KRT19, GPRC5B, RAMP1
## Negative:  PEMT, G6PC2, CRYBA2, SCGB2A1, HEPACAM2, RFX6, PARM1, SLC38A4, RGS4, PCP4
##     FAM159B, LOXL4, EDIL3, CRH, PLCE1, FAM105A, FAP, RASD1, SERPINE2, MLLT11
##     PCSK1, NROB1, SLC7A14, C1QL1, DHRS2, SPTSSB, IGFBPL1, PDK4, CACNB2, VWDE
## PC_ 2
## Positive:  KRT8, CD24, CLDN4, ELF3, LCN2, GATM, SDC4, PRSS8, SERPINA3, RAB11FIP1
##     CFB, TACSTD2, KRT7, REG1A, MUC1, KRT18, GSTA1, AMBP, CLDN3, CD74
##     ANPEP, SPINK1, CD44, GPX2, ANXA4, KIAA1522, CLDN1, LAD1, LYZ, ATP1A1
## Negative:  SPARC, COL4A1, IFITM1, NID1, LRRC32, PXDN, COL1A2, BGN, TMEM204, TIMP3
##     COL3A1, PDGFRB, LAMA4, PTRF, COL5A1, COL6A3, COL1A1, ENG, COL15A1, F2R
##     IGFBP4, SFRP2, THY1, AEBP1, CYGB, MMP2, CDH11, MFGE8, ANGPTL2, COL4A2
## PC_ 3
## Positive:  CFTR, TINAGL1, CMTM7, SPP1, TSPAN8, ALDH1A3, VTCN1, PPAP2C, CCND1, KRT19
##     MMP7, IGFBP7, PMEPA1, HSD17B2, CEACAM7, CA2, FUT3, SERPINA6, APCDD1, SLC4A4
##     SERPINA1, ANXA9, ANXA2, TFPI2, ONECUT2, DAB2, PROM1, TRPV6, C10orf54, LGALS4
## Negative:  CTRB1, CTRC, PNLIPRP2, CPA2, CTRB2, PNLIPRP1, BCAT1, PNLIP, CELA2A, REG1B
##     PLA2G1B, PRSS1, PRSS3, RARRES2, CELA3A, CELA3B, KLK1, CPB1, PDIA2, REG3G
##     DPEP1, REG3A, CEL, MT1H, SERPINI2, GP2, SPINK1, FAM129A, ALB, MT1G
## PC_ 4
## Positive:  COL3A1, COL1A2, COL6A3, COL1A1, COL5A1, PDGFRB, SFRP2, BGN, LUM, PCOLCE
##     NOTCH3, CYGB, THBS2, LAMC3, CDH11, MXRA8, FN1, AEBP1, COL6A1, MMP11
##     EDNRA, DCN, CD248, PLXDC1, COL5A2, MYL9, MSC, ISLR, VSTM4, SPON2
## Negative:  PODXL, PLVAP, RGCC, FLT1, PCAT19, ELTD1, KDR, ERG, ESAM, S1PR1
##     CLEC14A, ECSCR, MMP1, CXCR4, VWF, MYCT1, MMRN2, FAM101B, TM4SF18, GPR4
##     DYSF, RBP5, F2RL3, PRDM1, IFI27, ACVRL1, CD36, ESM1, CALCRL, CD93
## PC_ 5
## Positive:  HADH, IAPP, RBP4, ADCYAP1, PDX1, CASR, PCSK1, NPTX2, PFKFB2, MEG3
##     BMP5, IGSF1, GPM6A, FAM159B, PRSS23, FFAR4, RGS16, DLK1, GSN, CYYR1
##     ELMO1, VAT1L, GCGR, TMEM37, ASB9, TSPAN1, RASD1, CAPN13, G6PC2, RPH3A
## Negative:  TOP2A, BIRC5, PBK, CDC20, TPX2, CENPF, UBE2C, NUSAP1, CDK1, PTTG1
##     MKI67, CDCA8, ANLN, HMMR, ASPM, PRC1, CENPW, PLK1, CKS2, BUB1
##     CCNB2, SPAG5, CENPE, NDC80, CCNB1, KIF20A, CDCA3, AURKB, CENPA, DEPDC1
```

```
ElbowPlot(Seger)
```

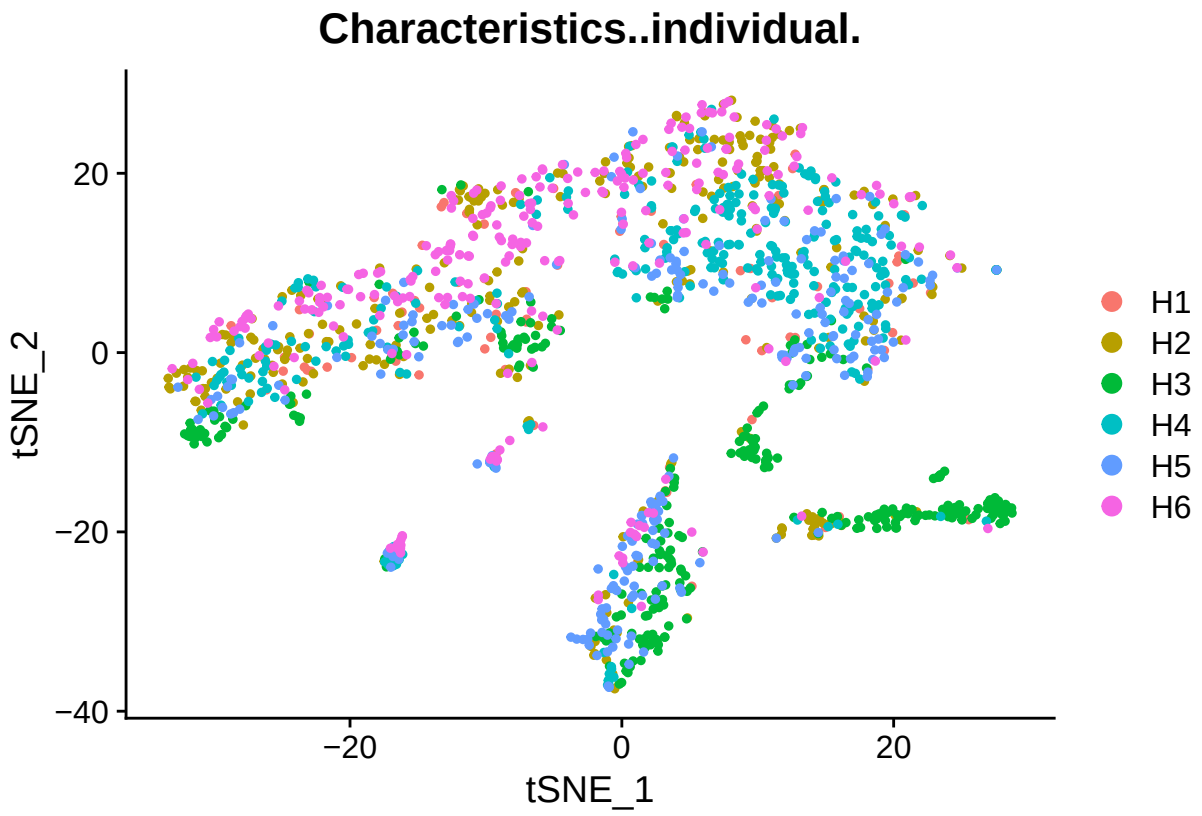


From the elbow plot, we first conduct tSNE and carry out visualizations without any additional information.

```
Seger <- RunTSNE(Seger, dims = 1:5)
p1=DimPlot(Seger, reduction = "tsne", label = TRUE)
p1 #tSNE plot
```




```
p2=DimPlot(Seger, reduction = "tsne", group.by='Characteristics..individual.')
p2
```



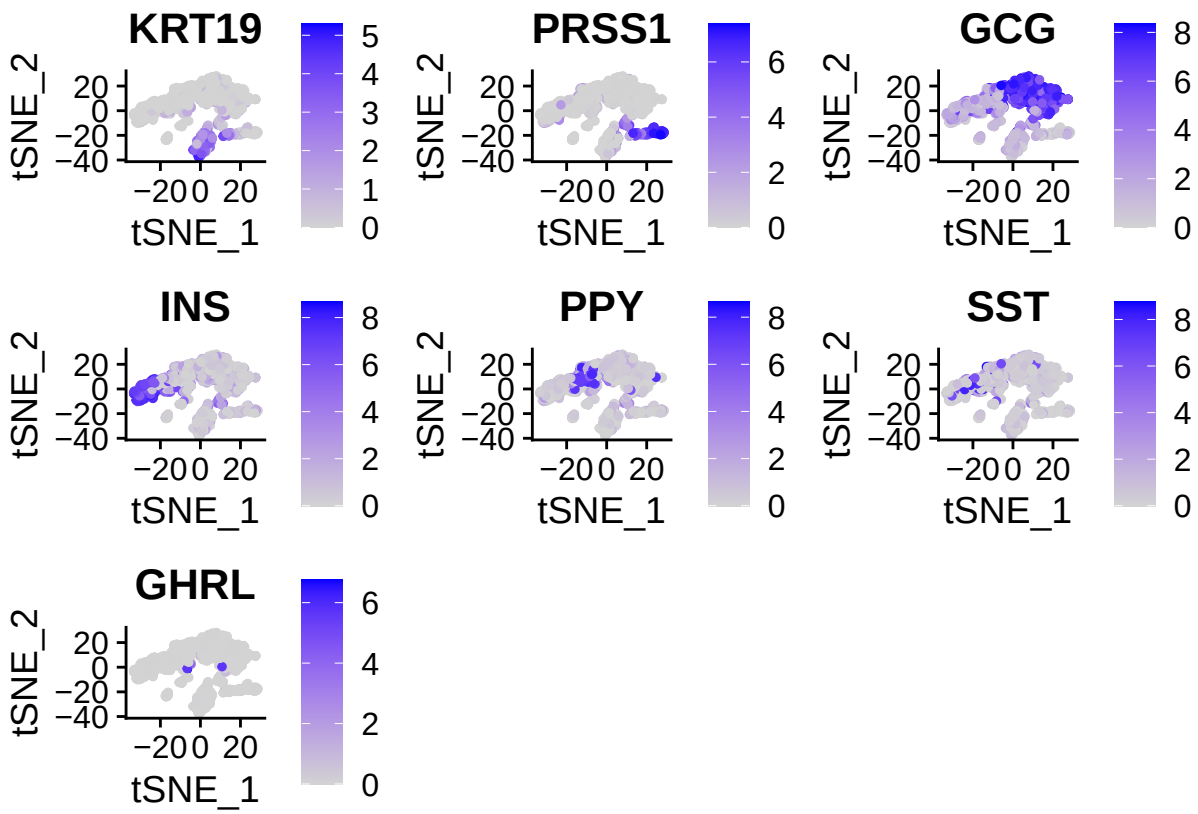
The first tSNE plot does not suggest any significant evidence that the cells cluster by individuals.

The second plot is just for sanity check purposes, where we can confirm that the 6 categories in the first plot refer to individuals, with just different variable names. The first tSNE plot has automatically inferred to the six individuals without any specification or argument provided.

Visualization of expression with marker genes

Finally, we take the provided marker genes and create UMAP plots to determine whether the cells cluster by discrete cell types in human pancreas.

```
FeaturePlot(Seger, features = c("KRT19","PRSS1","GCG","INS","PPY","SST","GHRL"))
```



It is more reasonable to conclude that the cells do cluster by discrete cell types in human pancreas.

Q1. Part 2: Baron et al.

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

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

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

```
## [1]  8569 20128
```

```
count = t(count)
all(colnames(count)==cell.meta$cell)
```

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

```
Baron <- CreateSeuratObject(counts = count, project = "Baron", min.cells = 3, min.features = 200, meta.
```

```
## Warning: Data is of class matrix. Coercing to dgCMatrix.
```

```
Baron
```

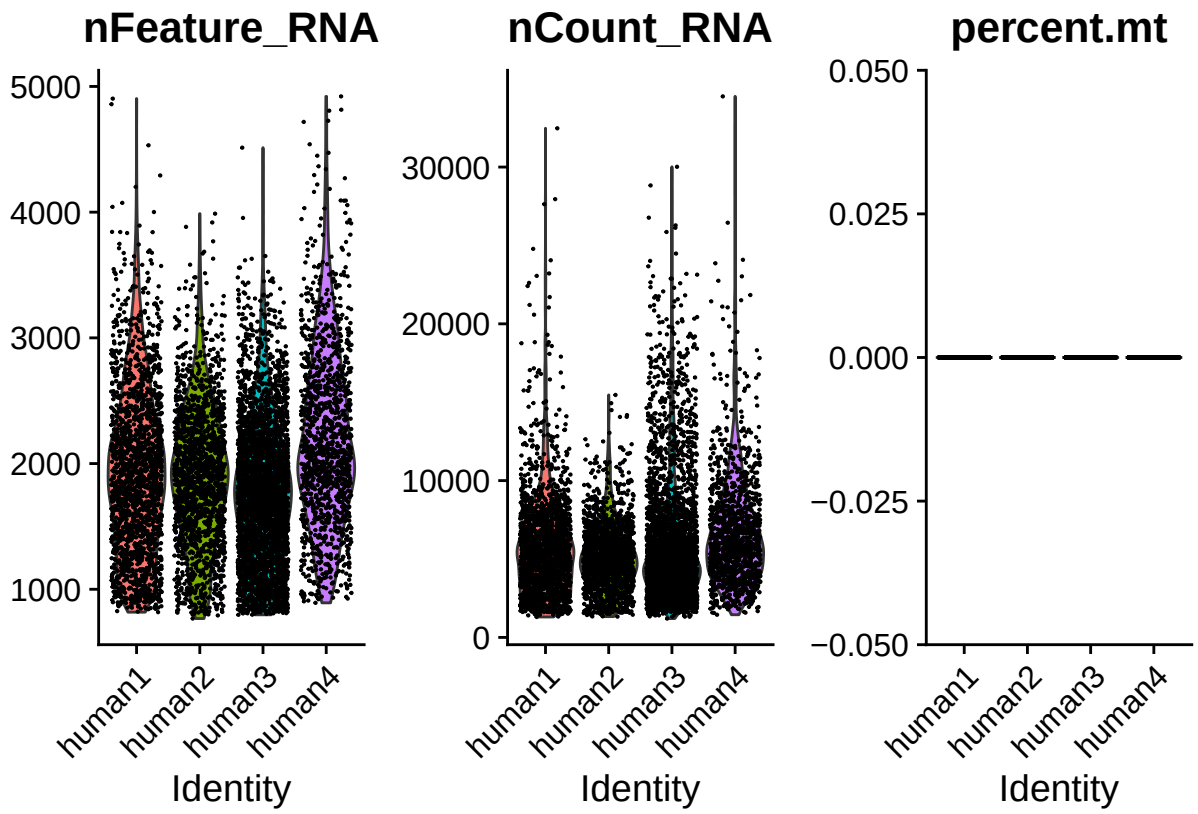
```
## An object of class Seurat
## 16359 features across 8569 samples within 1 assay
## Active assay: RNA (16359 features, 0 variable features)
##  1 layer present: counts
```

For this case, we would have to transpose the count matrix to match the dimension with metadata. After simple sanity check, we proceed to create Seurat object.

```
Baron[["percent.mt"]] <- PercentageFeatureSet(Baron, pattern = "^mt-")
VlnPlot(Baron, 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 in SingleExIPlot(type = type, data = data[, x, drop = FALSE], idents =
## idents, : All cells have the same value of percent.mt.
```




```
Baron <- subset(Baron, subset = nFeature_RNA > 600 & nFeature_RNA < 3800) #removing outliers
```

Here, we have decided the QC thresholds based on violin plot and removed outliers.

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

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

```
Baron <- FindVariableFeatures(Baron, selection.method = 'vst', nfeatures = 2000)
```

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

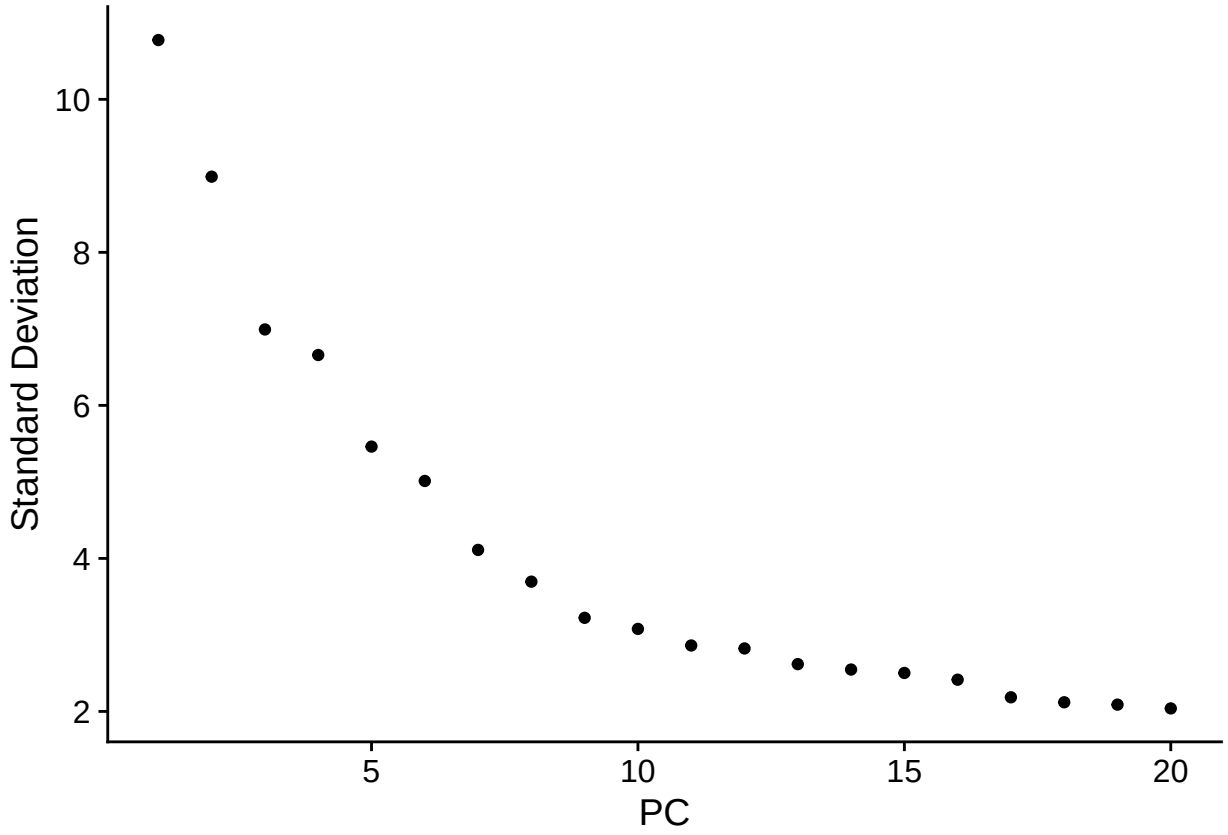
```
Baron <- ScaleData(Baron, features = rownames(Baron))
```

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

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

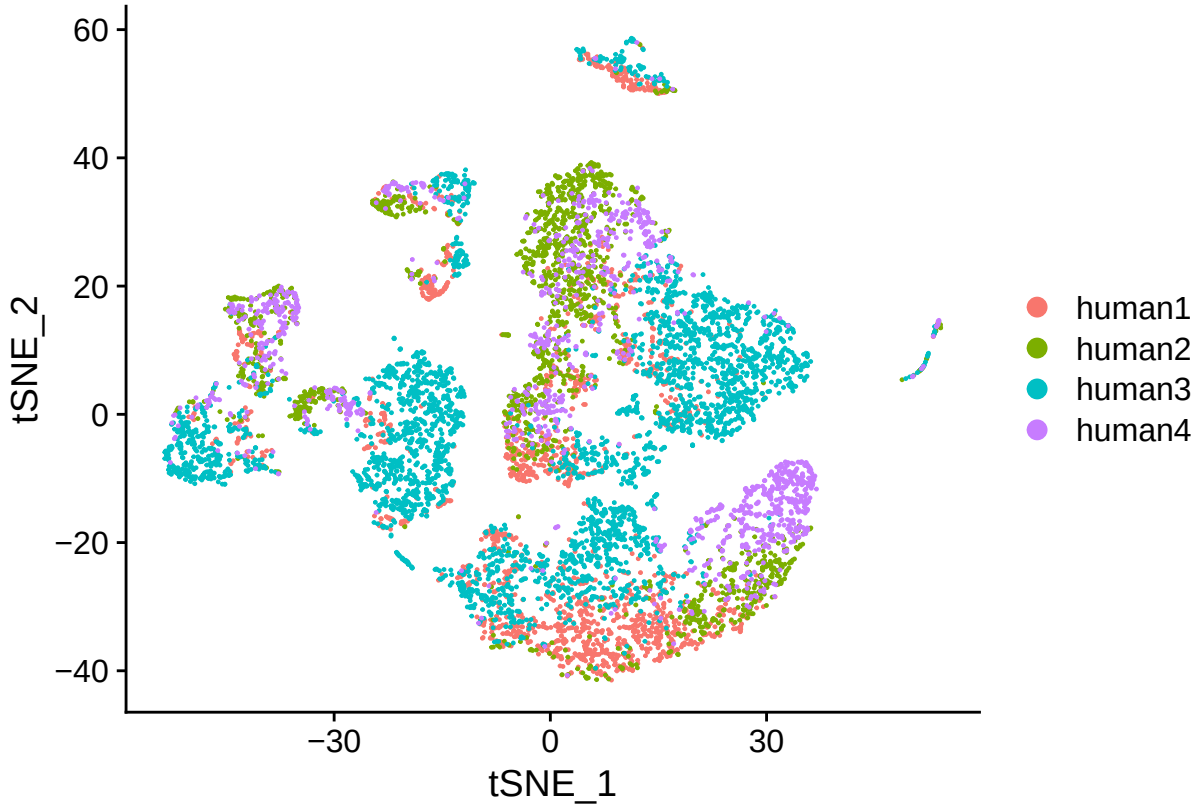
```
## PC_ 1
## Positive:  IFITM3, PMEPA1, TMSB10, COL18A1, ZFP36L1, IFITM2, CDC42EP1, SOX4, FLNA, MMP14
##     SERPING1, YBX3, ITGA5, MSN, RHOC, TMSB4X, MYH9, S100A16, LGALS1, TACSTD2
##     S100A11, HSPG2, LGALS3, SERPINH1, IL32, KRT7, CYR61, LITAF, PFKFB3, CEBPB
## Negative:  CHGA, CHGB, SCG2, PPP1R1A, VGF, PCSK2, GAD2, UCHL1, SLC30A8, ERO1B
##     NEUROD1, SYT7, ABCC8, C1QL1, GNG4, PCSK1, C2CD4A, UCN3, PAPSS2, IAPP
##     GCH1, PCP4, TMEM176B, STX1A, CFC1, MAFB, TMEM176A, INSM1, PAK3, C2CD4B
## PC_ 2
## Positive:  SPARC, COL6A2, IGFBP4, BGN, COL4A1, PDGFRB, COL3A1, PXDN, ENG, COL1A2
##     COL15A1, CYGB, COL6A3, COL1A1, AEBP1, COL6A1, COL5A1, TIMP1, EMILIN1, ITGA1
##     COL5A3, TIMP3, CRLF1, MMP2, HTRA3, COL5A2, CRISPLD2, LRRC32, MFGE8, FMOD
## Negative:  SERPINA3, CD24, KRT8, KRT18, SDC4, TACSTD2, PDZK1IP1, CD44, SPINK1, PRSS8
##     KRT7, IL32, CTRB1, LCN2, PRSS2, CLDN4, CTRB2, REG1A, PRSS1, CPA2
##     CPA1, PLA2G1B, CELA3A, MUC1, CELA3B, CPB1, KLK1, CTRC, ELF3, GATM
## PC_ 3
## Positive:  CELA3B, CTRC, CPA2, PNLIPRP1, CLPS, PLA2G1B, CELA2A, PRSS1, CELA3A, CPA1
##     KLK1, CTRB1, SYCN, CPB1, PRSS2, PNLIP, CTRB2, PNLIPRP2, GP2, REG1B
##     CUZD1, REG1A, PRSS3, CTRL, CELA2B, SPINK1, REG1P, MGST1, FOXD2, PLTP
## Negative:  CFTR, MMP7, TINAGL1, KRT19, SLC4A4, ANXA4, LGALS4, SERPINA5, PPAP2C, AGRN
##     SERPINA1, S100A14, CD74, TSPAN8, CTSH, VWA1, ONECUT2, SFRP5, AQP1, MALL
##     TSPAN15, VCAM1, GPRC5B, KRT23, FXYD2, TRPV6, TNFAIP2, ALDH1A3, RHPN2, CX3CL1
## PC_ 4
## Positive:  COL1A1, COL1A2, COL6A3, COL3A1, CRLF1, PDGFRB, COL5A1, BGN, NOTCH3, SFRP2
##     COL6A1, FMOD, THBS2, COL5A2, FN1, EMILIN1, CYGB, VCAN, ADAMTS12, SPON2
##     C1S, COL5A3, MXRA8, LUM, LAMC3, VSTM4, COL6A2, PRRX1, TPM2, CRISPLD2
## Negative:  PLVAP, PECAM1, CD93, FLT1, PODXL, KDR, SOX18, RGCC, ACVRL1, PCAT19
##     ROBO4, VWF, ESM1, TIE1, ESAM, ADGRL4, ECSCR, SEMA3F, S1PR1, PASK
##     CXCR4, CLDN5, DYSF, NOTCH4, ERG, ANGPT2, CDH5, BCL6B, IFI27, PTPRB
## PC_ 5
## Positive:  TYROBP, LAPTM5, SPI1, FCER1G, CD53, ACP5, SDS, C1QC, HLA.DRA, ITGB2
##     LCP1, ALOX5AP, LSP1, CD300A, C1QB, CSF1R, APOE, RGS1, HLA.DMB, FGL2
##     TNFRSF1B, SRGN, C1QA, MYO1F, MPEG1, HCK, HLA.DPA1, HLA.DRB1, NCKAP1L, PLA2G7
## Negative:  PRSS23, COL4A1, PXDN, HSPG2, PLVAP, COL18A1, ESAM, FLT1, PODXL, KDR
##     IGFBP7, STC1, WWTR1, VWF, ROBO4, SPARC, SOX18, PCAT19, MCAM, SERPINE1
##     LAMA4, ESM1, ETS1, PMEPA1, ECSCR, ADGRF5, ANGPTL2, ADGRL4, SEMA3F, PASK
```

```
ElbowPlot(Baron)
```



Moving on to clustering, we carry out the first visualization, by creating a tSNE plot.

```
Baron <- RunTSNE(Baron, dims = 1:10)
p1=DimPlot(Baron, reduction = "tsne")
p1 #tSNE plot
```

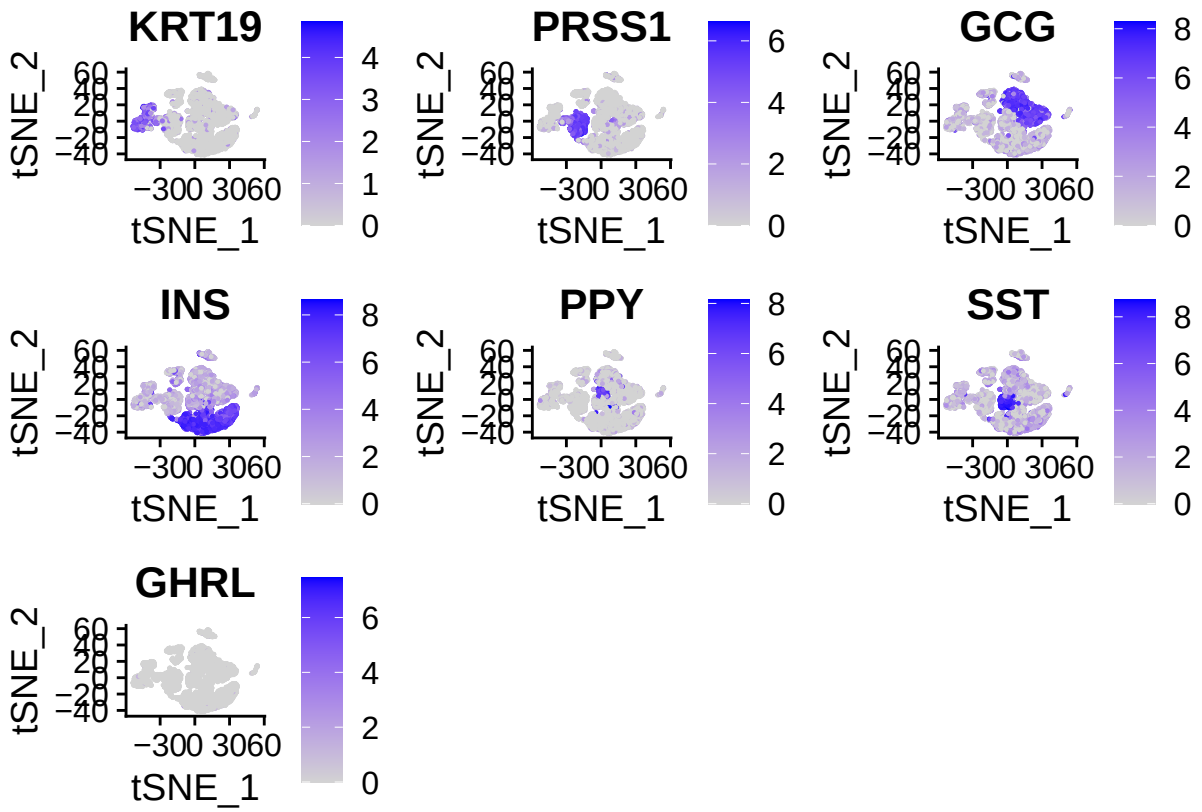


Again, the tSNE plot does not convince that the cells cluster by donors.

Visualization of expression with marker genes

Lastly, we utilize the marker genes to visualize their expression profiles across cells.

```
FeaturePlot(Baron, features = c("KRT19","PRSS1","GCG","INS","PPY","SST","GHRL"))
```



By comparing the second plot to the first, we can conclude that the cells cluster by discrete cell types in human pancreas for Baron dataset as well.

Question 2. Part 1: Baron et al.

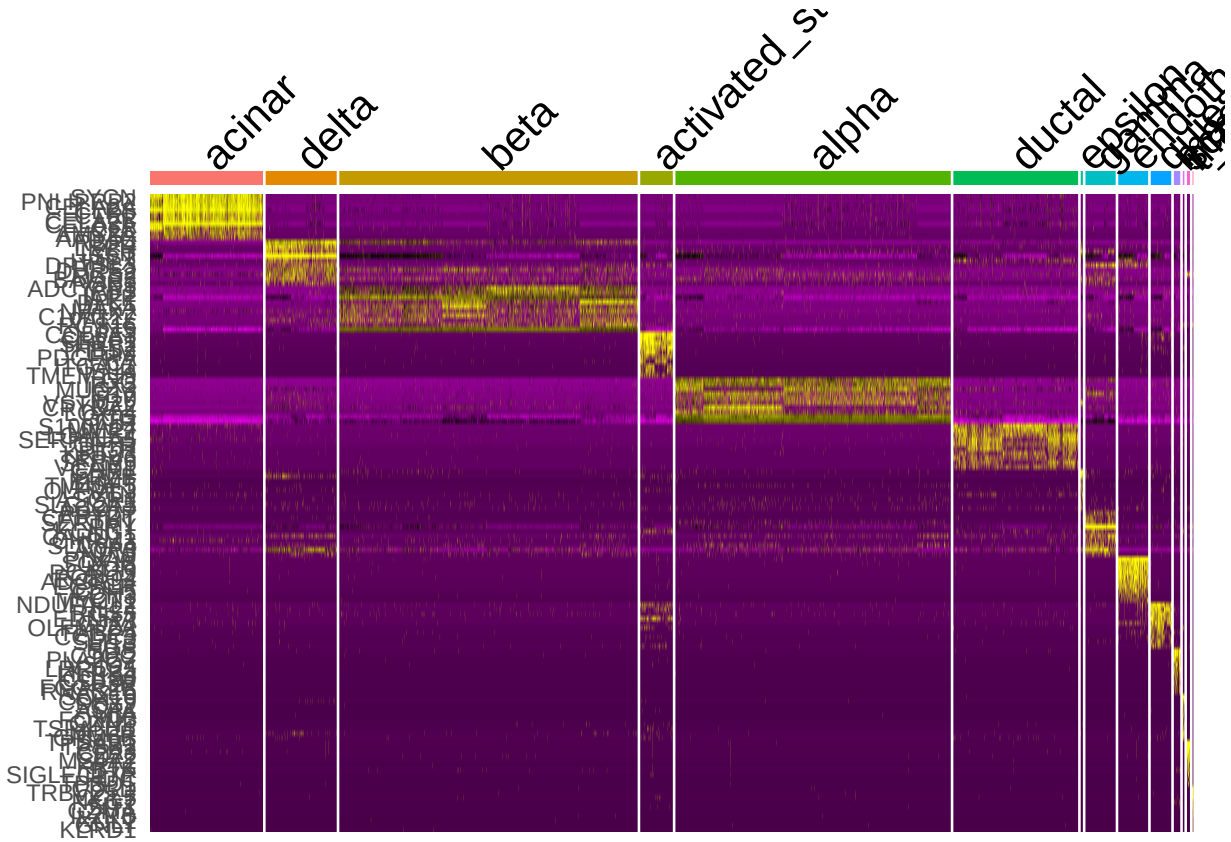
For question 2, we first try to identify and visualize cell-type-specific gene expressions in a heatmap for Baron et al. dataset. It can be easily carried out using cell type information within the object's metadata.

```
Idents(Baron)=Baron$celltype
Baron.markers <- FindAllMarkers(Baron, only.pos = TRUE, min.pct = 0.25, logfc.threshold = 0.25)
```

```
## Calculating cluster acinar

## 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 delta
```

```
## Calculating cluster beta

## Calculating cluster activated_stellate

## Calculating cluster alpha

## Calculating cluster ductal

## Calculating cluster epsilon

## Calculating cluster gamma

## Calculating cluster endothelial

## Calculating cluster quiescent_stellate

## Calculating cluster macrophage

## Calculating cluster schwann

## Calculating cluster mast

## Calculating cluster t_cell
```

```
Baron.markers %>%
  group_by(cluster) %>%
  top_n(n = 10, wt = avg_log2FC) -> top10
DoHeatmap(Baron, features = top10$gene) + NoLegend()
```



Question 2. Part 2: Segerstolpe et al.

Next, we move on to apply the same procedure to Segerstolpe et al. dataset, but we will first need to annotate the cell types.

```
anchors <- FindTransferAnchors(
  reference = Baron,
  query = Seger,
  dims = 1:10,
  reference.reduction = "pca",
  recompute.residuals = FALSE
)
```

```
## Projecting cell embeddings

## Finding neighborhoods

## Finding anchors

##  Found 3851 anchors
```

```
Seger <- MapQuery(
  anchorset = anchors,
  query = Seger,
  reference = Baron,
  refdata = list(
    celltype = "celltype"
  ),
  reference.reduction = "pca"
)
```

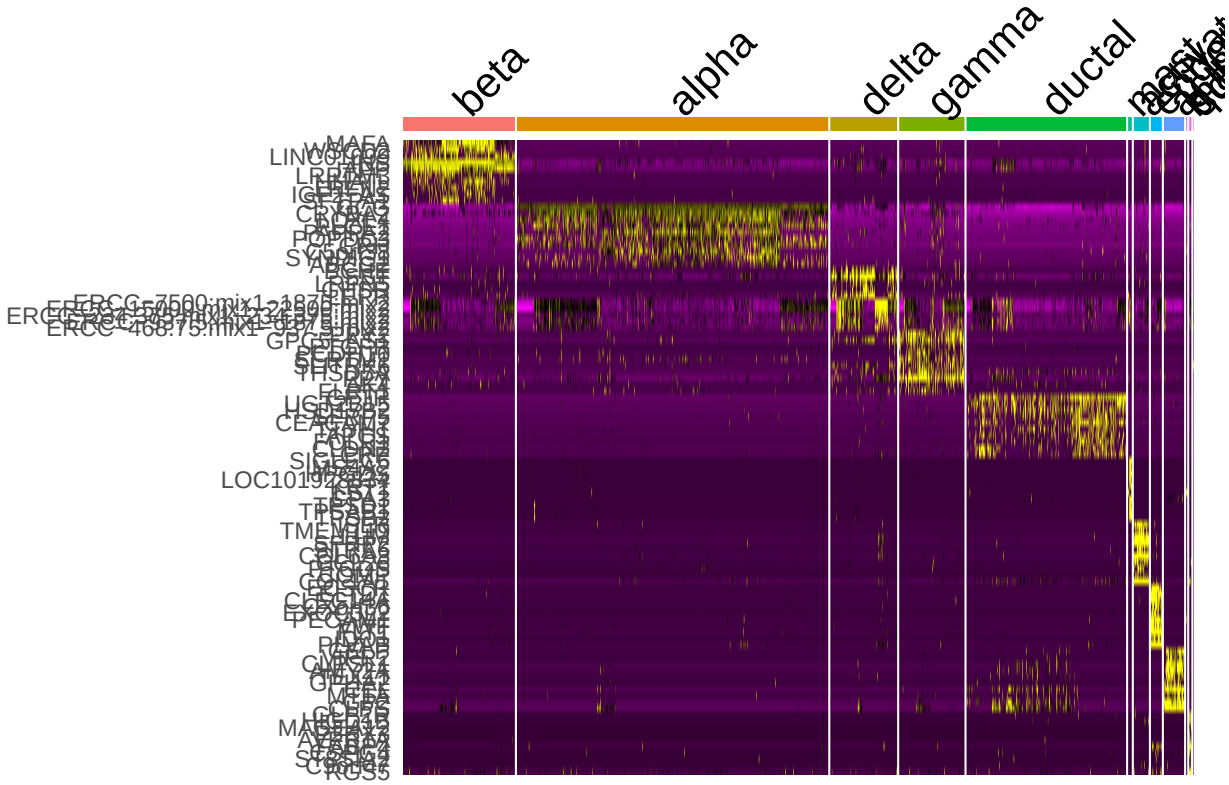
```
## Finding integration vectors

## Finding integration vector weights

## Predicting cell labels

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

##
## Integrating dataset 2 with reference dataset

## Finding integration vectors

## Integrating data
```

Now, for Segerstolpe et al. dataset, we can refer to newly added 'predicted cell type' info within the object and produce a heatmap to visualize cell-type-specific gene expressions, with top 10 genes per cell type.

```
Idents(Seger)=Seger$predicted.celltype
Seger.markers <- FindAllMarkers(Seger, only.pos = TRUE, min.pct = 0.25, logfc.threshold = 0.25)
```

```
## Calculating cluster beta

## Calculating cluster alpha

## Calculating cluster delta

## Calculating cluster gamma

## Calculating cluster ductal

## Calculating cluster mast

## Calculating cluster activated_stellate

## Calculating cluster endothelial

## Calculating cluster acinar

## Calculating cluster macrophage

## Calculating cluster quiescent_stellate

## Calculating cluster schwann
```

```
Seger.markers %>%
  group_by(cluster) %>%
  top_n(n = 10, wt = avg_log2FC) -> top10
DoHeatmap(Seger, features = top10$gene) + NoLegend()
```



Question 3

Combining two datasets

In order to conduct a joint analysis of the two datasets, we first create 'datasource' label for each object so that we can distinguish the two sources that has different protocol/technology. Then, we find integration anchors and create an integrated dataset.

```
Seger$datasource='Seger'
Baron$datasource='Baron'
Seger$celltype=Seger$predicted.celltype

Panc.anchors <- FindIntegrationAnchors(object.list = list(Seger, Baron), dims = 1:10)
```

```
## Computing 2000 integration features
```

```
## Scaling features for provided objects
```

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

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

```
## Finding all pairwise anchors

## Running CCA

## Merging objects

## Finding neighborhoods

## Finding anchors

##  Found 5554 anchors

## Filtering anchors

##  Retained 1890 anchors
```

```
Combined <- IntegrateData(anchorset = Panc.anchors, dims = 1:10)
```

```
## Merging dataset 1 into 2

## Extracting anchors for merged samples

## Finding integration vectors

## Finding integration vector weights

## Integrating data
```

Q3. Part 1: Clustering without data integration

With the combined dataset, we first carry out standard clustering process without data integration.

```
DefaultAssay(Combined) <- "RNA" # Without integration (i.e., concatenating cells)
Combined <- ScaleData(Combined, verbose = FALSE)
Combined <- RunPCA(Combined, npcs = 30, verbose = FALSE)
Combined <- RunUMAP(Combined, reduction = "pca", dims = 1:20)
```

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

## 15:47:38 UMAP embedding parameters a = 0.9922 b = 1.112

## 15:47:38 Read 10004 rows and found 20 numeric columns

## 15:47:38 Using Annoy for neighbor search, n_neighbors = 30
```

```
## 15:47:38 Building Annoy index with metric = cosine, n_trees = 50

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

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

## **************************************************|
## 15:47:39 Writing NN index file to temp file C:\Users\PC1\AppData\Local\Temp\RtmpQpXrkS\fileaaa815fa6
## 15:47:39 Searching Annoy index using 1 thread, search_k = 3000
## 15:47:41 Annoy recall = 100%
## 15:47:42 Commencing smooth kNN distance calibration using 1 thread with target n_neighbors = 30
## 15:47:42 Initializing from normalized Laplacian + noise (using RSpectra)
## 15:47:43 Commencing optimization for 200 epochs, with 418162 positive edges
## 15:47:52 Optimization finished
```

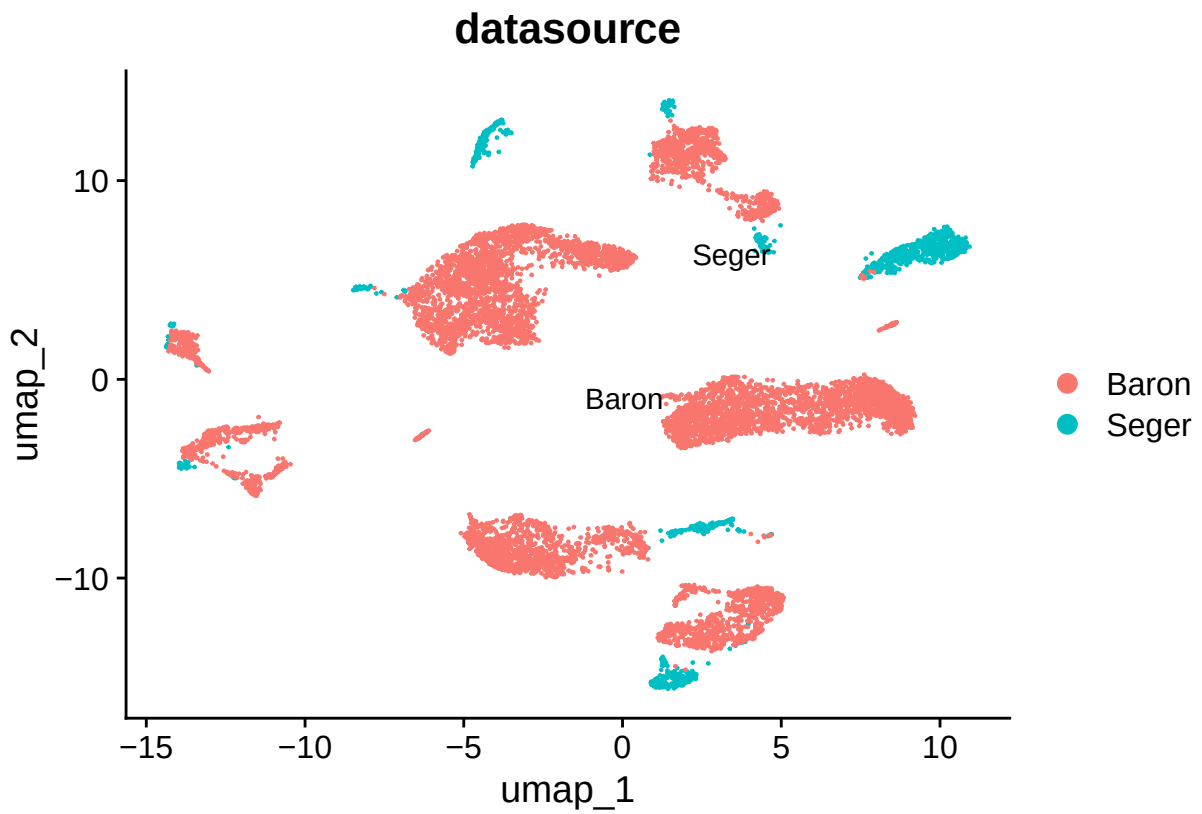
```
Combined <- FindNeighbors(Combined, reduction = "pca", dims = 1:20)
```

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

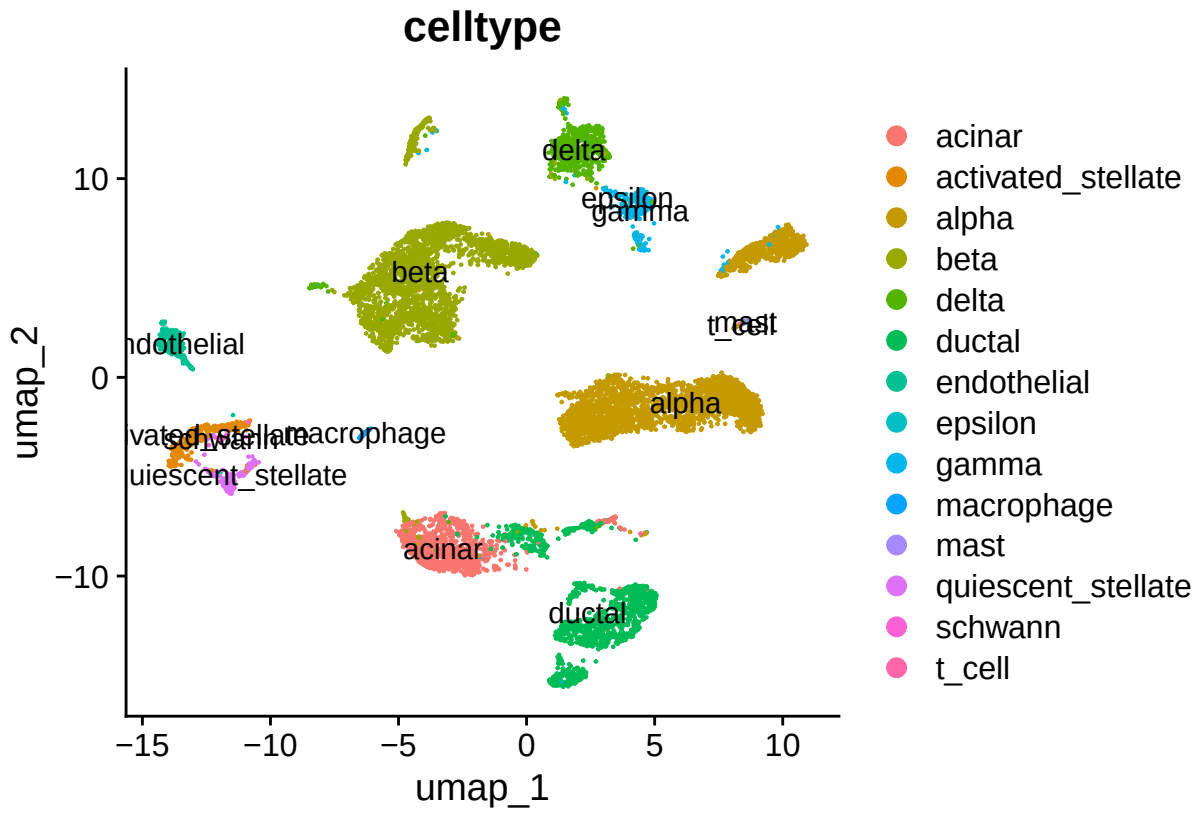
```
Combined <- FindClusters(Combined, resolution = 0.5)
```

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10004
## Number of edges: 360424
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.9412
## Number of communities: 20
## Elapsed time: 0 seconds
```

```
p3 <- DimPlot(Combined, reduction = "umap", group.by = "datasource", label=TRUE)
p4 <- DimPlot(Combined, reduction = "umap", group.by='celltype', label=TRUE)
p3
```


datasource
umap_2
umap_1
10
0
−10
−15
−10
−5
0
5
10
Seger
Baron
Baron
Seger

```
p4
```



The first plot suggests that cells tend to cluster by protocol/technology (labeled as 'datasource') without data integration. The second plot is visualization based on cell types.

Q3. Part 2: Clustering after data integration by CCA alignment

```
DefaultAssay(Combined) <- "integrated" # Without integration (i.e., concatenating cells)
Combined <- ScaleData(Combined, verbose = FALSE)
Combined <- RunPCA(Combined, npcs = 30, verbose = FALSE)
Combined <- RunUMAP(Combined, reduction = "pca", dims = 1:20)
```

```
## 15:48:01 UMAP embedding parameters a = 0.9922 b = 1.112

## 15:48:01 Read 10004 rows and found 20 numeric columns

## 15:48:01 Using Annoy for neighbor search, n_neighbors = 30

## 15:48:01 Building Annoy index with metric = cosine, n_trees = 50

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

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

```
## **************************************************|
## 15:48:02 Writing NN index file to temp file C:\Users\PC1\AppData\Local\Temp\RtmpQpXrkS\fileaaa8195f13
## 15:48:02 Searching Annoy index using 1 thread, search_k = 3000
## 15:48:05 Annoy recall = 100%
## 15:48:05 Commencing smooth kNN distance calibration using 1 thread with target n_neighbors = 30
## 15:48:06 Initializing from normalized Laplacian + noise (using RSpectra)
## 15:48:06 Commencing optimization for 200 epochs, with 427388 positive edges
## 15:48:16 Optimization finished
```

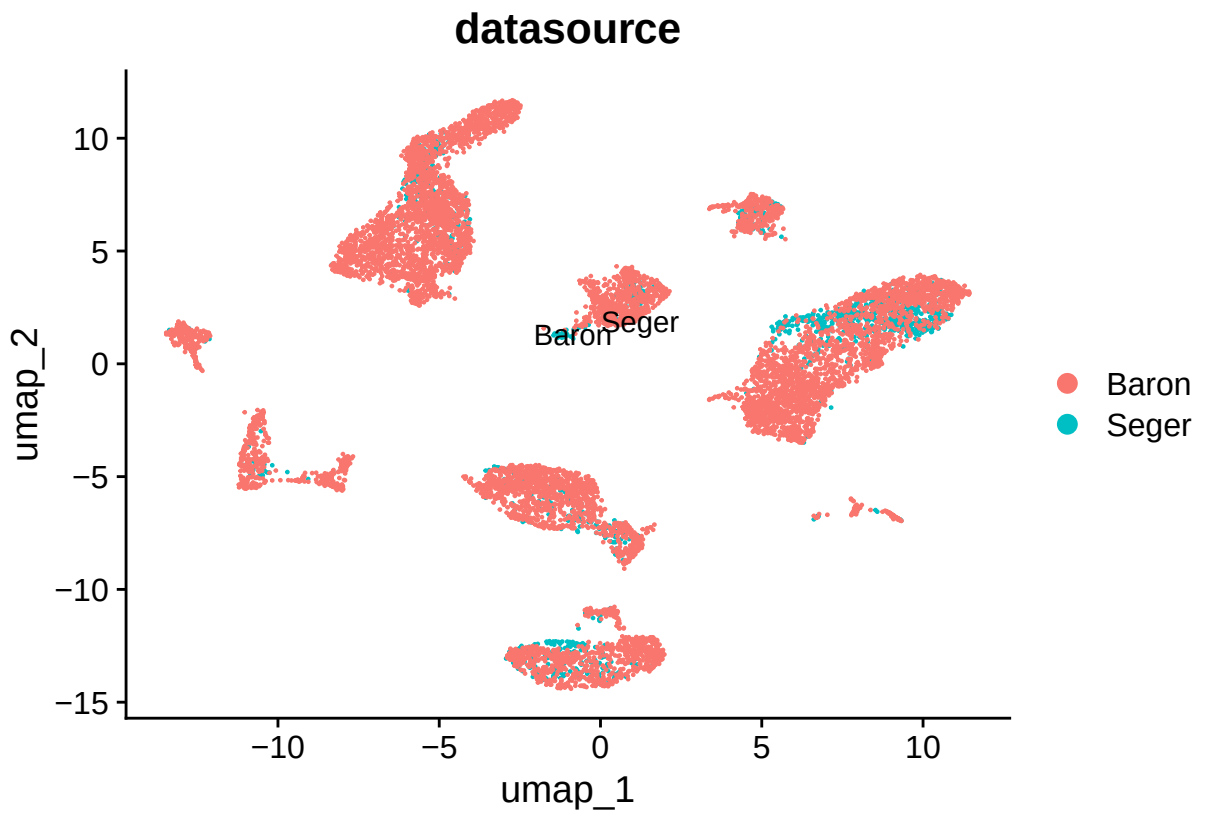
```
Combined <- FindNeighbors(Combined, reduction = "pca", dims = 1:20)
```

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

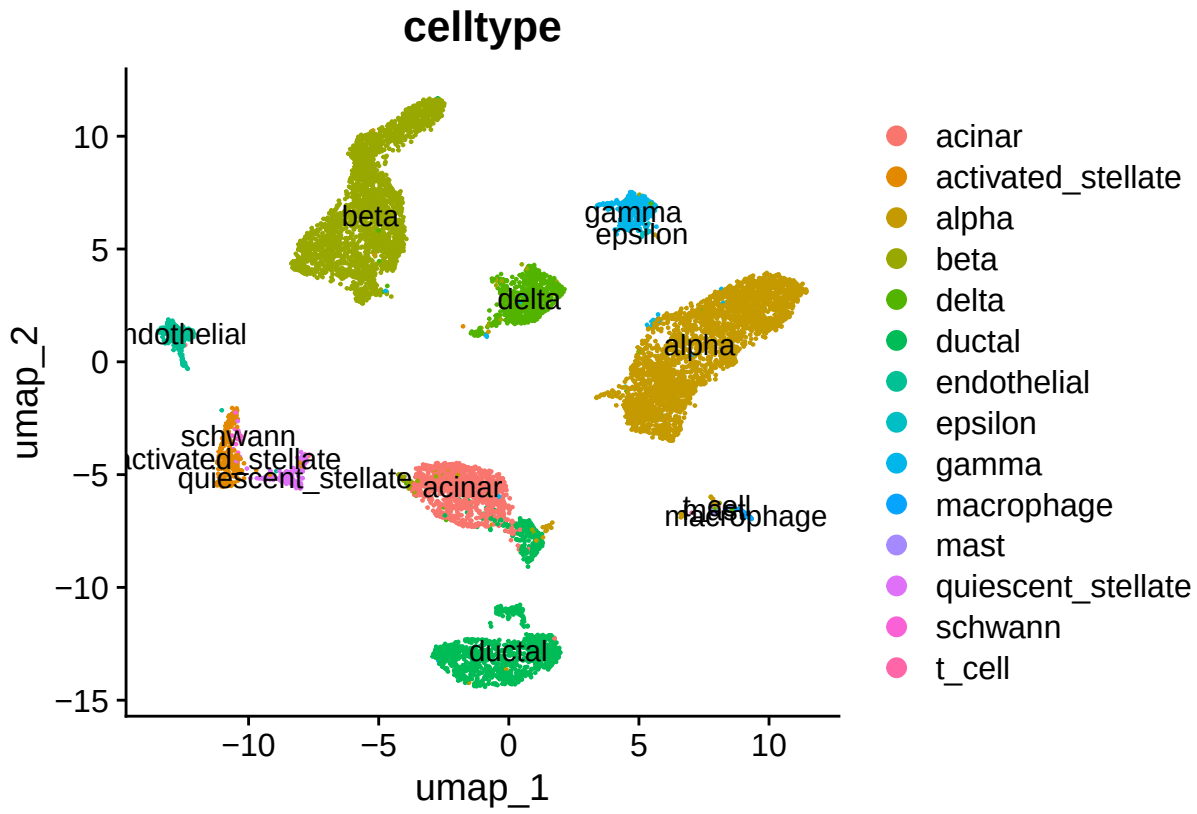
```
Combined <- FindClusters(Combined, resolution = 0.5)
```

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 10004
## Number of edges: 373107
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.9277
## Number of communities: 15
## Elapsed time: 0 seconds
```

```
p5 <- DimPlot(Combined, reduction = "umap", group.by = "datasource", label=TRUE)
p6 <- DimPlot(Combined, reduction = "umap", group.by='celltype', label=TRUE)
p5
```


datasource
Baron
Seger
umap_2
umap_1
10
5
0
−5
−10
−15
−10
−5
0
5
10
Baron
Seger

```
p6
```



The first plot suggests that the integration has resulted in a better mixture of cells, so that it is less clustered based on protocol/technology. Next is the post-integration UMAP plot.