SingelCellRNAseq_Homework

XiaoyanWen

8/11/2021

Single Cell RNA-seq Workflow using Seurat For this assignment we will analyze a publicly available scRNA-seq dataset using Seurat, a popular R package for analyzing scRNA-seq. There are several different workflows included in the package, but for now we will focus on clustering the cells based on gene expression. We will follow the worflow provided here Seurat - Guided Clustering Tutorial

There are two main goals of this assignment:

How many different cell types do we have in our dataset? Which genes are useful in identifying one cell type from another. The dataset we will be using is from a dataset looking at pancreatic ductal adenocarcinoma GSE111672. The study looks at two different samples, however for this assignment, we will only focus on one.

Moncada R, Barkley D, Wagner F, Chiodin M et al. Integrating microarray-based spatial transcriptomics and single-cell RNA-seq reveals tissue architecture in pancreatic ductal adenocarcinomas. Nat Biotechnol 2020 Mar;38(3):333-342. PMID: 31932730 Loading the data Step 0: Please install and load the following packages

```
library(dplyr)
```

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

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

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

```
library(Seurat)
```

```
## Attaching SeuratObject
```

```
library(patchwork)
```

Step 1: Load the TSV file

```
GSM3036909 <- read.delim("GSM3036909.tsv")
```

Step 2: Create a Seurat object. Call the object pdac1. You set the project argument in the CreateSeuratObject the same. Here we will also request the same criteria as mentioned in the workflow: min.cells=3 and min.features=200.

```
pdac1 <- CreateSeuratObject(counts = GSM3036909, min.cells = 3, min.features = 200)
```

```
## Warning in storage.mode(from) <- "double": NAs introduced by coercion
```

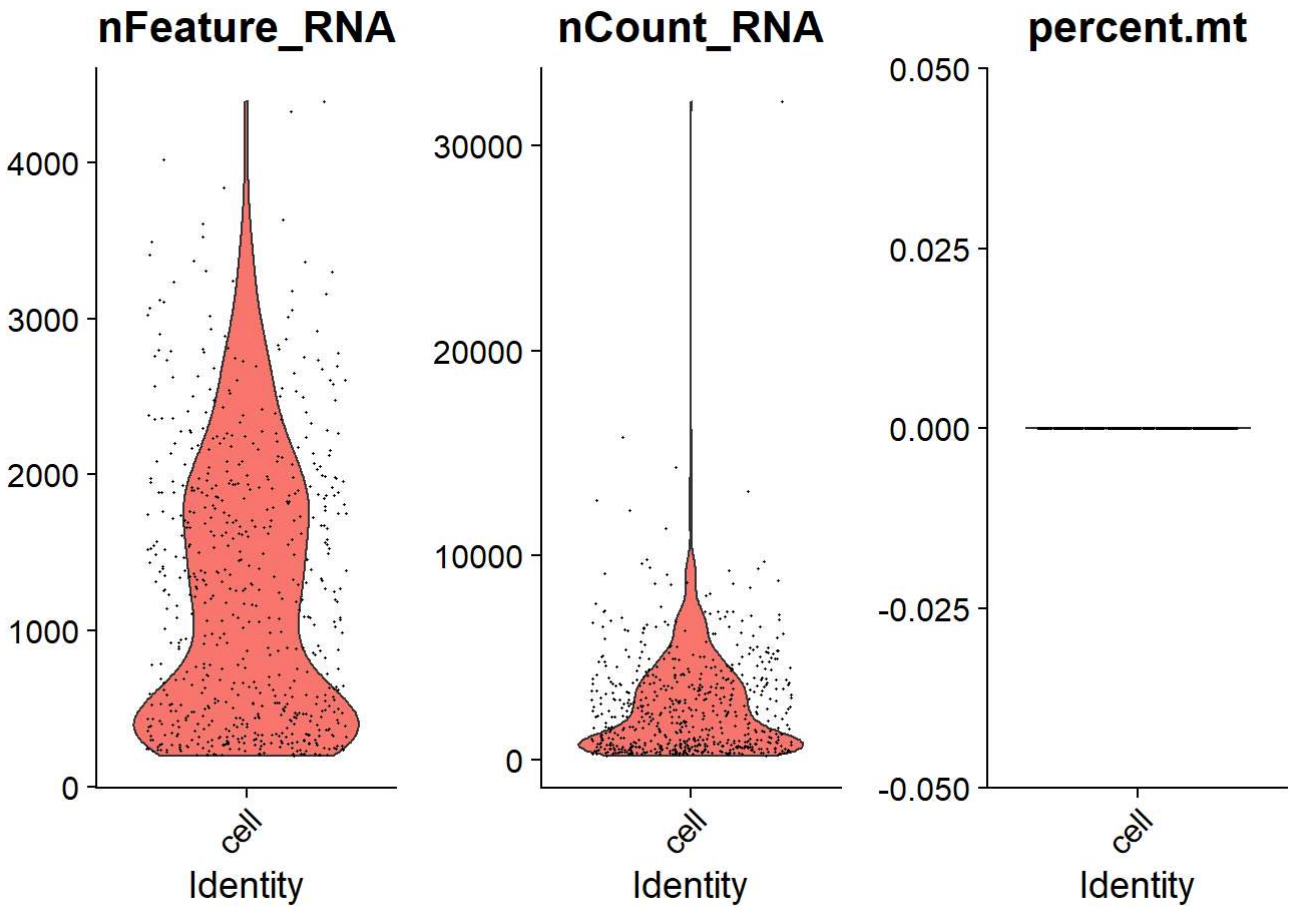
Quality control Step 3: Label the Mitochondrial genes We don't want to use cells that have too many mitochondrial genes, so we create a new column to help us summarize how many mitochondrial genes were identified in the different cells.

```
pdac1[["percent.mt"]] <- PercentageFeatureSet(pdac1, pattern = "^MT-")
```

Step 4: Visualize the distribution Use the VlnPlot function to view the number of counts, number of features, and the percent mitochondrial genes.

```
VlnPlot(pdac1, features = c("nFeature_RNA", "nCount_RNA", "percent.mt"), ncol = 3)
```

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



Step 5: Filter data Only keep the cells that have greater than 200 and less than 2500 unique features and the percent mitochondrial genes is less than 5.

```
pdac1 <- subset(pdac1, subset = nFeature_RNA > 200 & nFeature_RNA < 2500 & percent.mt < 5)
```

Normalization Step 6: Normalize data Taking the log of the data, makes the data more normal distributed. Normalize data using the LogNormaliza method with a scale factor of 10,000

```
pdac1 <- NormalizeData(pdac1, normalization.method = "LogNormalize", scale.factor = 10000)
```

Step 6: Calculate gene variation Find the 2000 most variable genes using the FindVariableFeatures command using the vst method.

```
pdac1 <- FindVariableFeatures(pdac1, selection.method = "vst", nfeatures = 2000)
```

PCA Step 7: Scale data Scaling the data normalizes the standard deviation and centers the data. This is an important step before performing PCA.

```
all.genes <- rownames(pdac1)
pdac1 <- ScaleData(pdac1, features = all.genes)
```

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

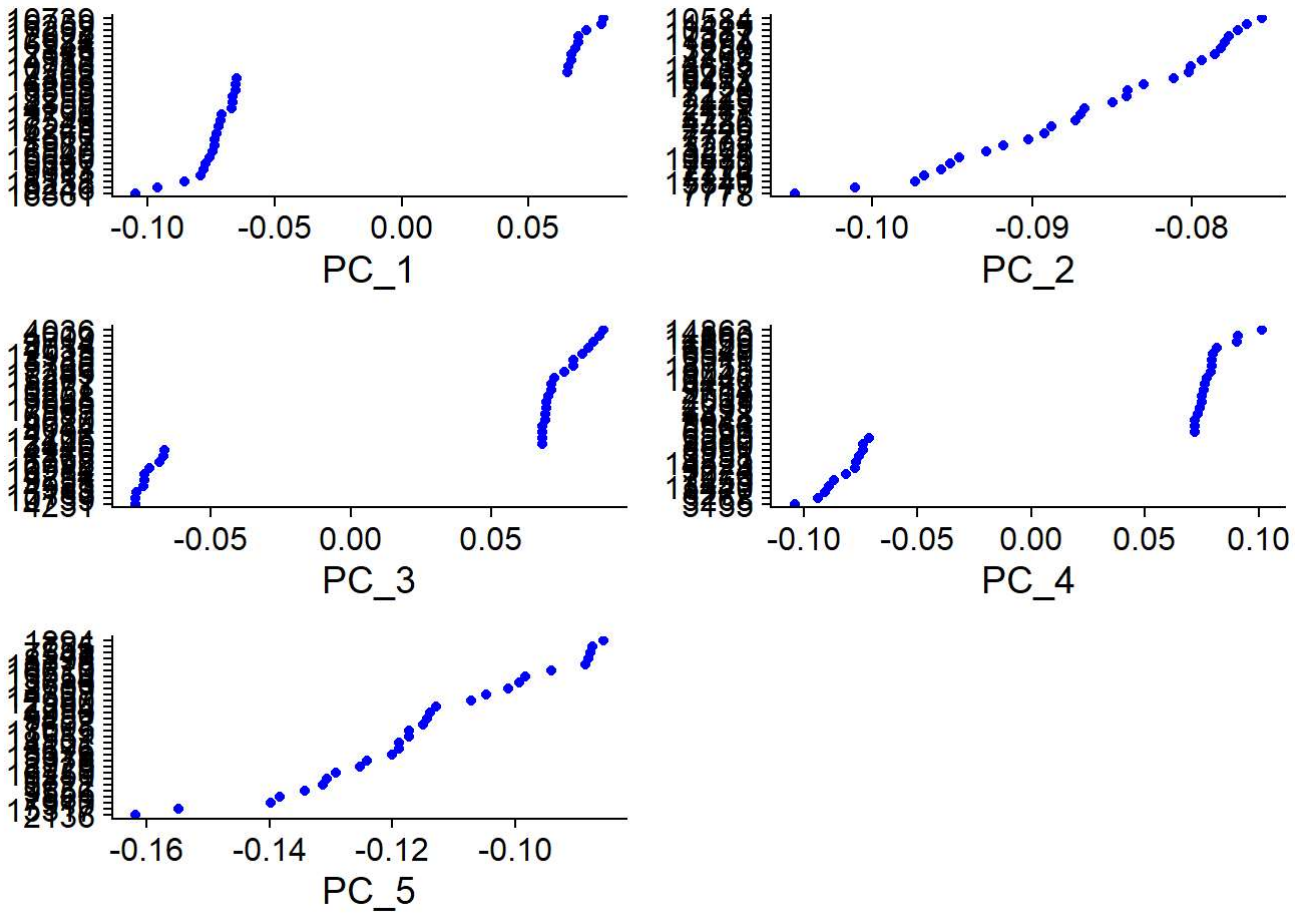
Step 8: PCA Run PCA

```
pdac1 <- RunPCA(pdac1, features = VariableFeatures(object = pdac1))
```

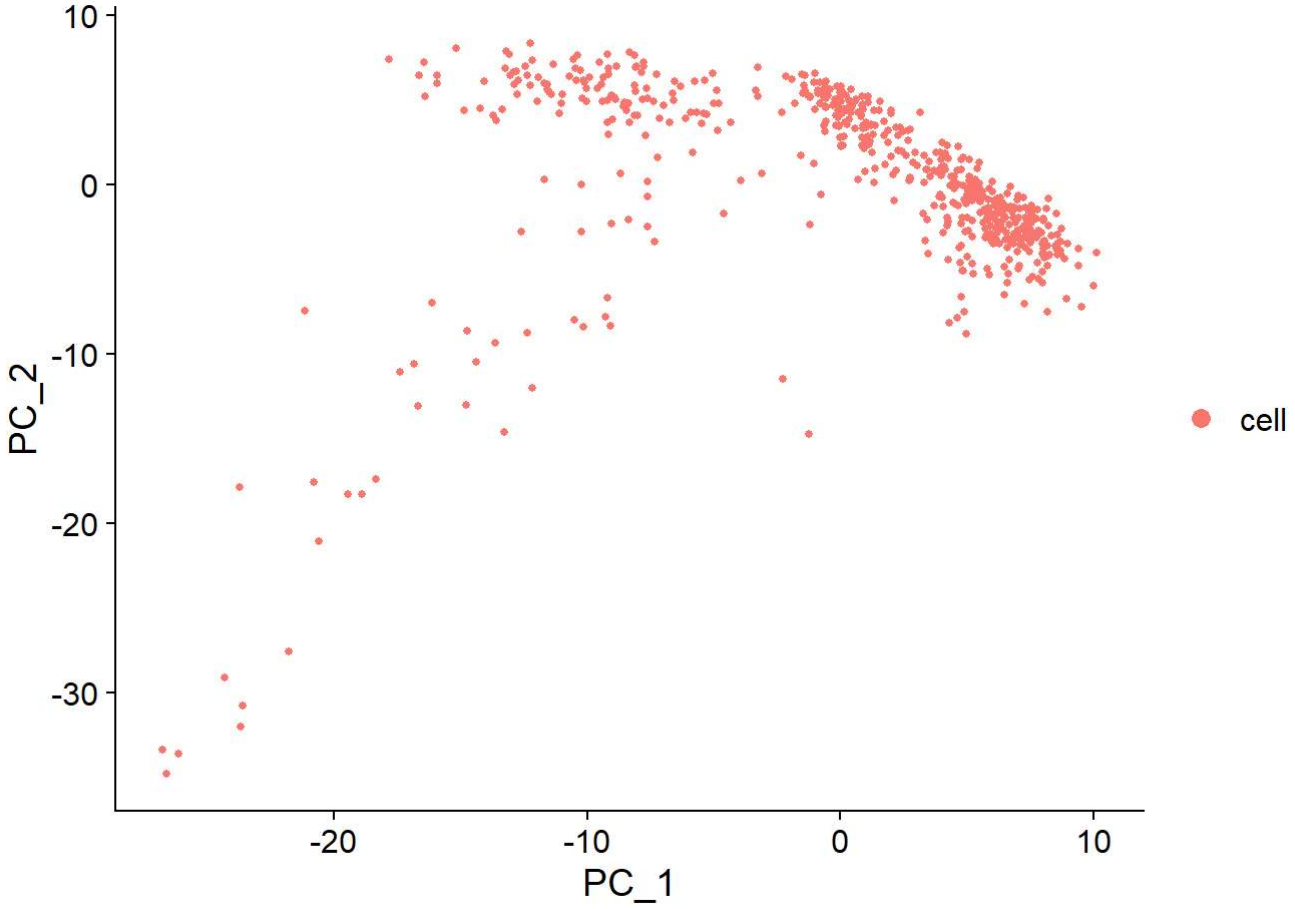
```
## PC_ 1
## Positive:  10739, 16259, 17298, 12924, 16984, 848, 17119, 4983, 10736, 7262
##     16289, 10370, 2092, 16770, 17117, 1378, 5227, 6798, 14234, 1418
##     4231, 15868, 14850, 5207, 5634, 18411, 4246, 10733, 926, 2568
## Negative:  16361, 9246, 18258, 1111, 9683, 9307, 15640, 6540, 1967, 1513
##     14860, 16249, 7549, 13794, 4106, 14859, 3259, 8503, 6269, 14333
##     1620, 6271, 4291, 4483, 18665, 14905, 6263, 17673, 18877, 875
## PC_ 2
## Positive:  6649, 2011, 4481, 16446, 7543, 4478, 14860, 7542, 14859, 8253
##     3133, 3134, 4291, 3135, 13440, 4480, 3904, 1111, 10372, 8277
##     3825, 13513, 6541, 6317, 3132, 17872, 3214, 12955, 9556, 3268
## Negative:  7778, 7777, 15840, 7776, 7774, 7770, 10589, 3271, 3202, 7773
##     7772, 2446, 4286, 7771, 2447, 2449, 7775, 7220, 17134, 16471
##     6267, 13635, 4413, 7767, 3250, 11394, 7277, 10587, 1415, 10584
## PC_ 3
## Positive:  4036, 4009, 9714, 4035, 12925, 3135, 5766, 17245, 15201, 8277
##     10251, 3268, 9045, 17905, 6587, 4670, 9044, 4192, 12406, 13440
##     7459, 13105, 14851, 7070, 1999, 18665, 14853, 10246, 14446, 3224
## Negative:  4231, 10739, 15183, 4983, 9758, 14234, 16984, 6798, 1378, 4246
##     17119, 16082, 15289, 16289, 16259, 10736, 6320, 6540, 5971, 18411
##     1076, 14860, 17117, 14859, 1967, 2627, 2568, 4291, 6271, 4478
## PC_ 4
## Positive:  14863, 14860, 14859, 6649, 6540, 2011, 13755, 9045, 16446, 4481
##     9714, 4009, 4035, 4291, 4478, 9044, 5566, 6593, 8259, 1967
##     17905, 6272, 10251, 7738, 15201, 1447, 3287, 4192, 10372, 8253
## Negative:  3135, 3268, 8277, 13440, 7459, 7070, 3224, 14323, 3258, 8991
##     8990, 3280, 19630, 3214, 5766, 9292, 7460, 15452, 16275, 8989
##     18939, 102, 4425, 11249, 18938, 3247, 3246, 7212, 5243, 3267
## PC_ 5
## Positive:  3135, 3268, 8277, 7459, 13440, 7070, 3224, 3133, 16524, 8991
##     8990, 19630, 3280, 13794, 9307, 16289, 4097, 5227, 8482, 9683
##     16361, 18939, 8995, 15452, 18938, 4483, 7212, 3258, 3134, 4425
## Negative:  2136, 15317, 7910, 1881, 9556, 9722, 1311, 16250, 8729, 2978
##     15318, 3676, 4101, 11551, 18015, 7608, 9452, 4950, 7294, 12990
##     14327, 8709, 3689, 13754, 16610, 13716, 8898, 7548, 755, 1894
```

Step 9: Visualize data using VizDimLoadings and DimPlot functions. Can you tell from the PCA analyis, the number of cell types that are present?

```
VizDimLoadings(pdac1,  reduction = "pca")
```

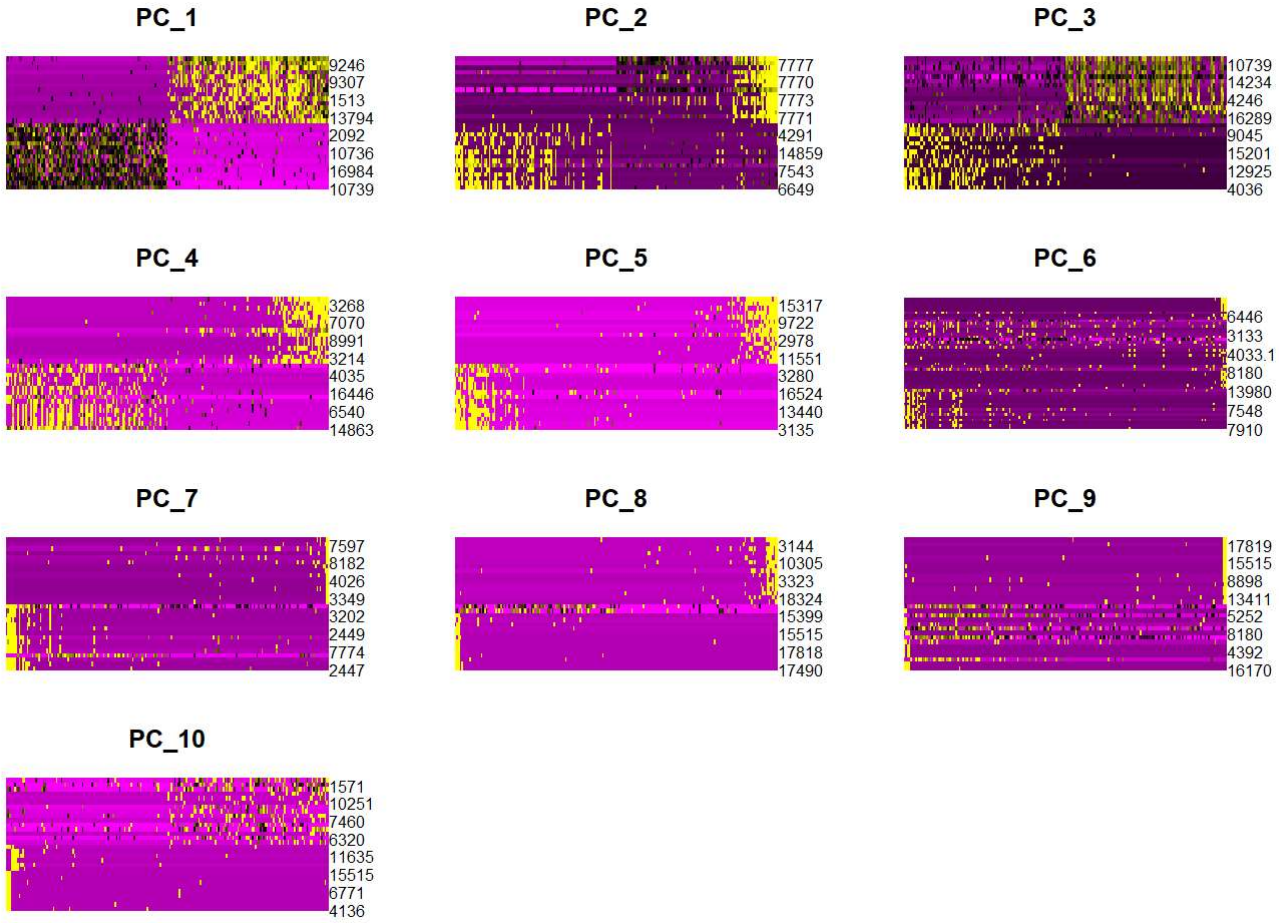



```
DimPlot(pdac1, reduction = "pca")
```



Step 10: PCA heatmaps Another way to visualize the variation explained by the PC is creating heatmaps. Create heatmaps of the first 10 dimensions and include 200 cells.

```
DimHeatmap(pdac1, dims = 1:10, cells = 200, balanced = TRUE)
```



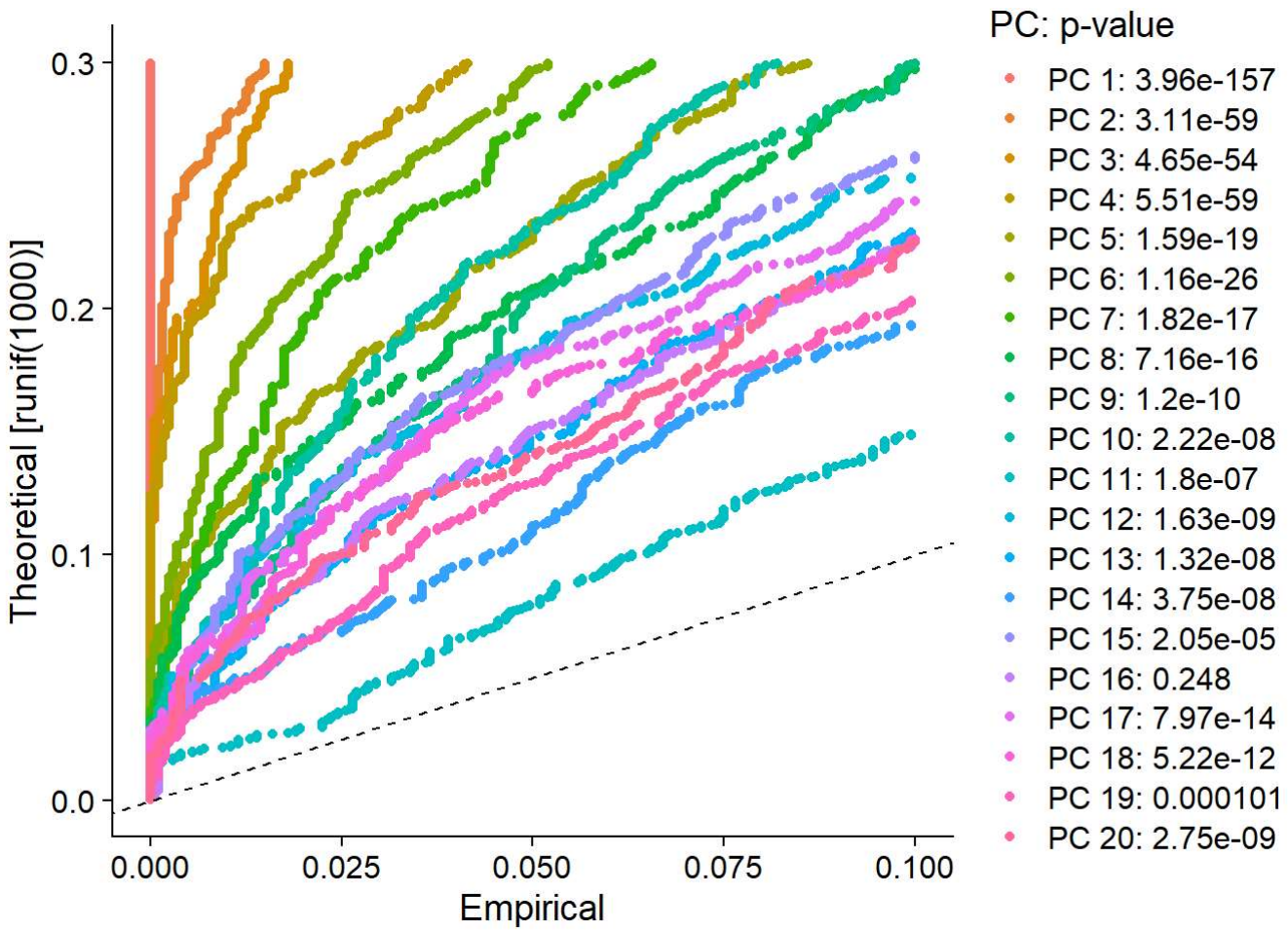
Step 11: Dimensionality To make this more quantitative, let's see when does the variation reach the lowest amount of variation explained. Use the JackStraw method with 100 replicates and score the first 20 dimensions.

```
pdac1 <- JackStraw(pdac1, num.replicate = 100)
pdac1 <- ScoreJackStraw(pdac1, dims = 1:20)
```

Plot the results for the first 20 dimensions.

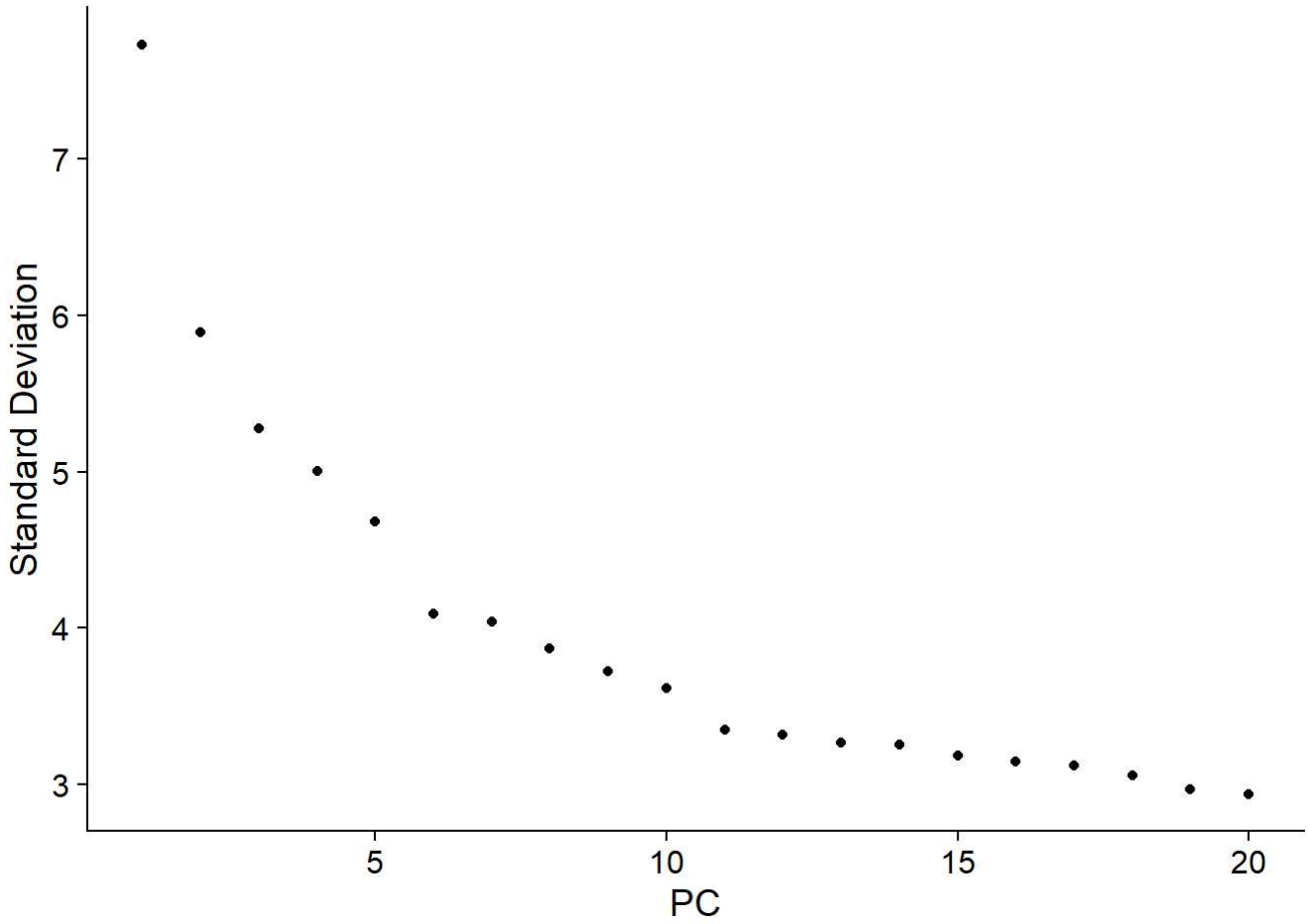
```
JackStrawPlot(pdac1, dims = 1:20)
```

```
## Warning: Removed 29571 rows containing missing values (geom_point).
```



Use the elbow plot

```
ElbowPlot(pdac1)
```



Step 12: Clustering. Now we will group together the cells based on where they are located in the different dimensions. Use the FindNeighbors function using the first 9 dimensions.

And then identify the clusters usng the FindClusters function.

```
pdac1 <- FindNeighbors(pdac1, dims = 1:9)
```

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

```
## Computing SNN
```

```
pdac1 <- FindClusters(pdac1, resolution = 0.5)
```

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 569
## Number of edges: 14274
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.8645
## Number of communities: 8
## Elapsed time: 0 seconds
```

tsne/umap Step 13: Perform a UMAP analysis using the first 9 dimensions using RunUMAP and then visualize it using DimPlot.

```
pdac1 <- RunUMAP(pdac1, dims = 1:9)
```

```
## 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
```

```
## 13:42:13 UMAP embedding parameters a = 0.9922 b = 1.112
```

```
## 13:42:13 Read 569 rows and found 9 numeric columns
```

```
## 13:42:13 Using Annoy for neighbor search, n_neighbors = 30
```

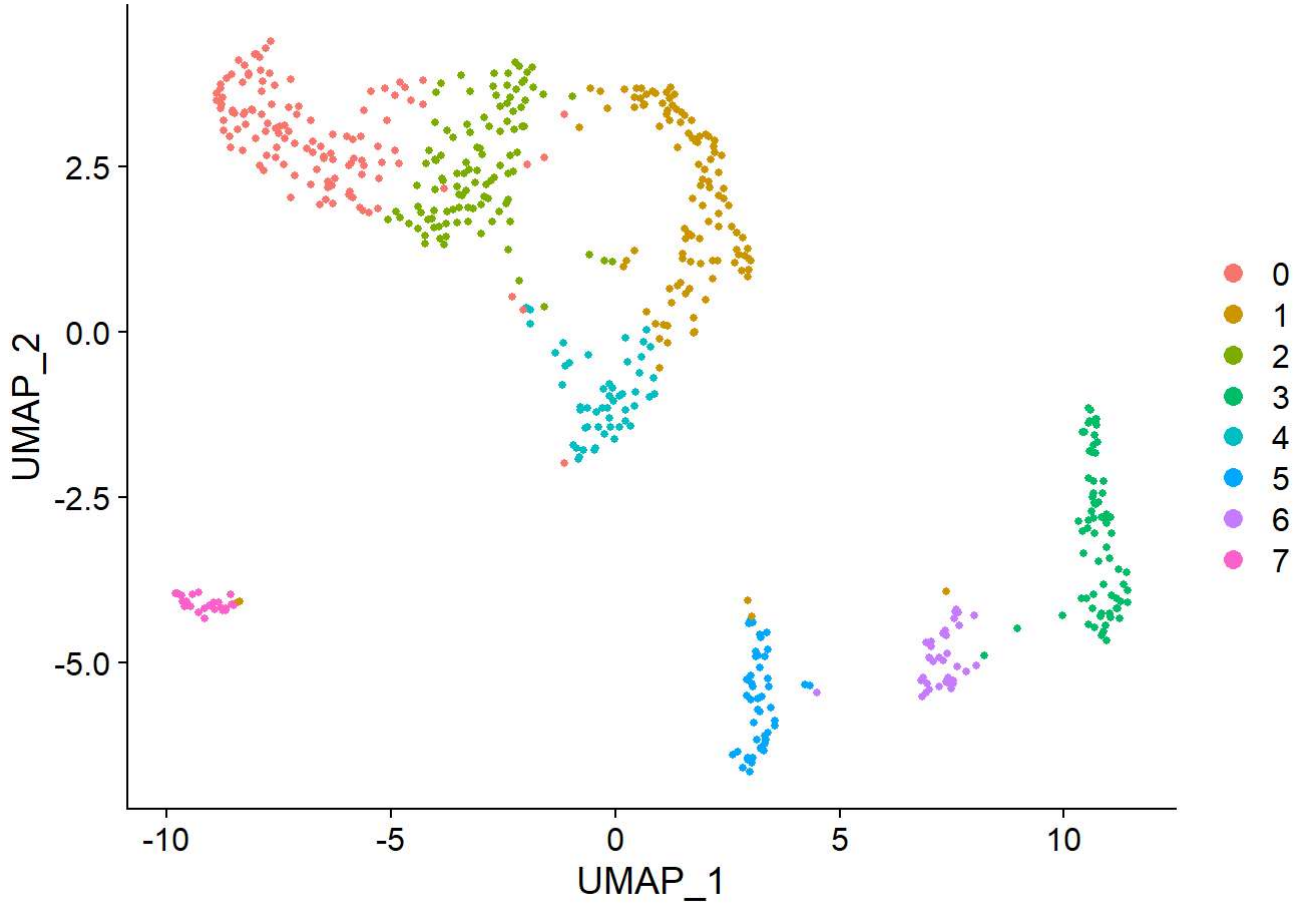
```
## 13:42:13 Building Annoy index with metric = cosine, n_trees = 50
```

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

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

```
## **************************************************|
## 13:42:13 Writing NN index file to temp file C:\Users\wen_x\AppData\Local\Temp\RtmpOuXhut\file
2eec6e1250ea
## 13:42:13 Searching Annoy index using 1 thread, search_k = 3000
## 13:42:13 Annoy recall = 100%
## 13:42:14 Commencing smooth kNN distance calibration using 1 thread
## 13:42:14 Initializing from normalized Laplacian + noise
## 13:42:14 Commencing optimization for 500 epochs, with 20700 positive edges
## 13:42:15 Optimization finished
```

```
DimPlot(pdac1, reduction = "umap")
```



How many clusters do you get? How many possible mistakes do you see? answer: there shows 8 clusters and > 16 mistakes.

Step 14: Identify the markers that compare each cluster agains all. Report only positively markers. Use the FindAllMarkers for this.

```
pdac1.markers <- FindAllMarkers(pdac1, only.pos = TRUE)
```

```
## Calculating cluster 0
```

```
## Calculating cluster 1
```

```
## Calculating cluster 2
```

```
## Calculating cluster 3
```

```
## Calculating cluster 4
```

```
## Calculating cluster 5
```

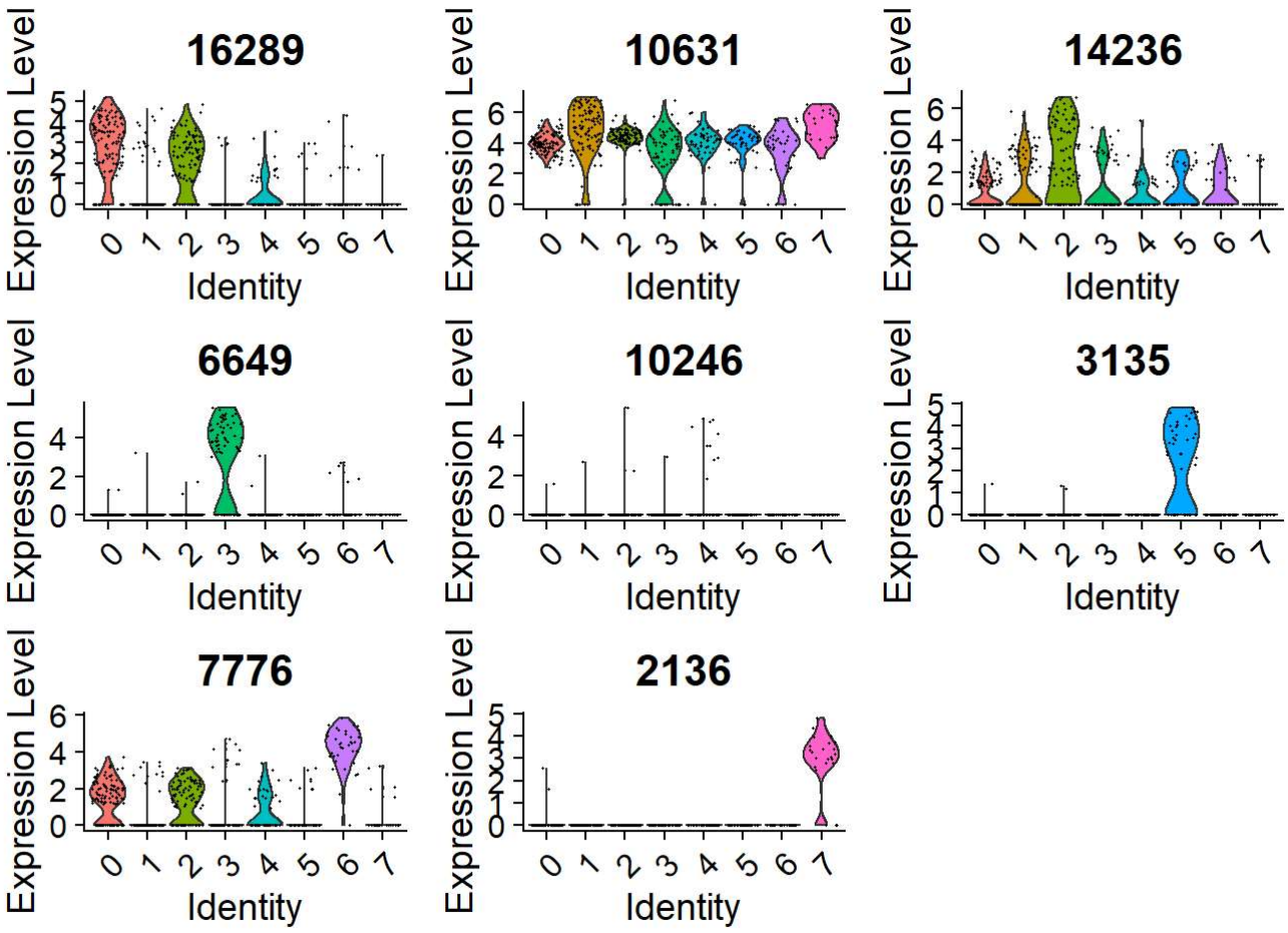
```
## Calculating cluster 6
```

```
## Calculating cluster 7
```

```
pdac1.markers %>%
  group_by(cluster) %>%
  top_n(n = 1, wt = avg_log2FC) -> top1_clusterMarkers
```

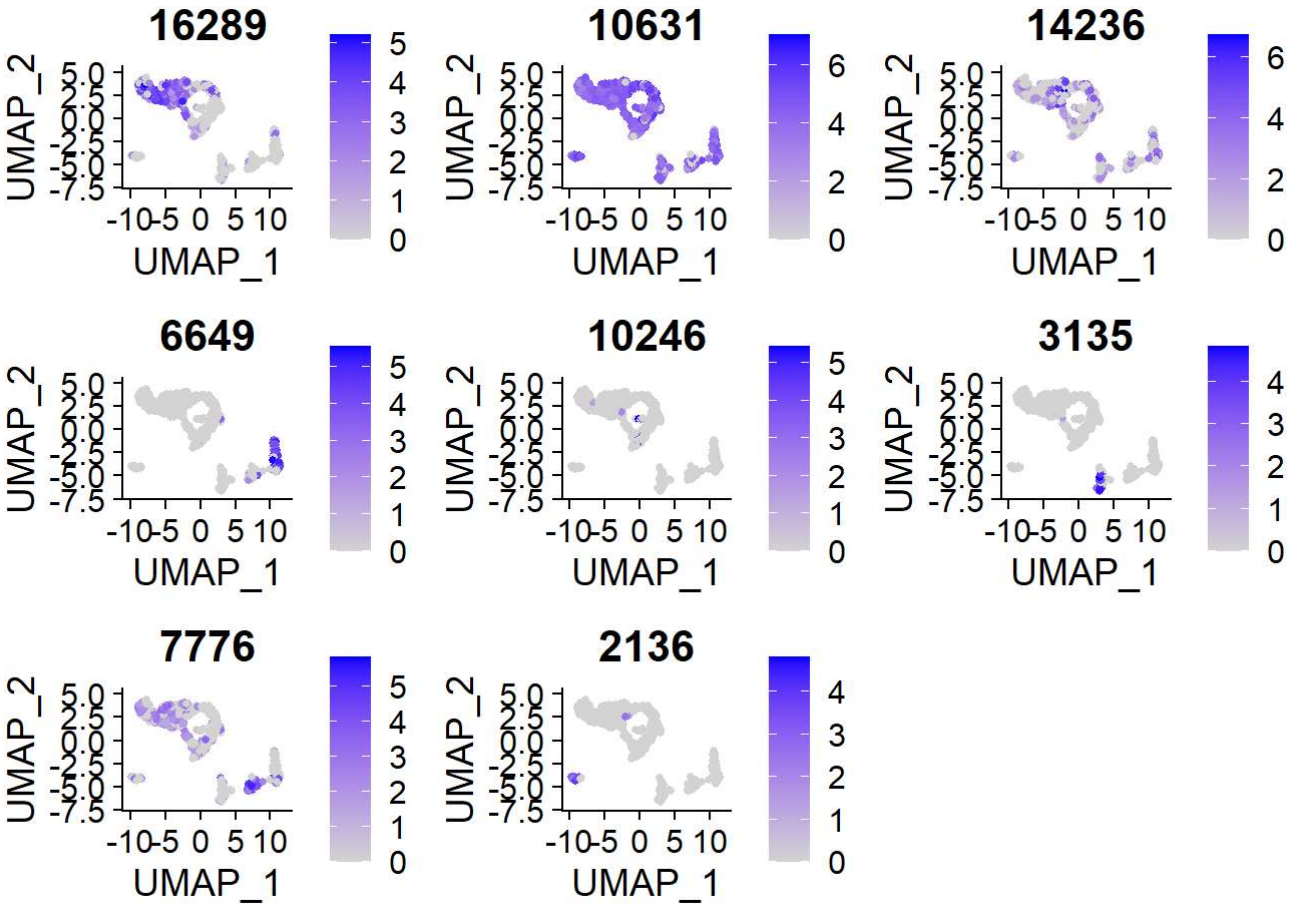
Step 15: Create a violin plot using one feature from each cluster.

```
VlnPlot(pdac1, features = top1_clusterMarkers$gene)
```



Step 16: Create a feature plot using the same features as before

```
FeaturePlot(pdac1, features = top1_clusterMarkers$gene)
```


16289
10631
14236
6649
10246
3135
7776
2136
UMAP_1
UMAP_2