Transcriptomics Homework 5

Charlee Cobb

2023-04-24

#Step 1: Load the TSV file

```
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)

tsv <- 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 = tsv, project = "pdac1", min.cells = 3, min.features = 200)
```

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

```
pdac1
```

```
## An object of class Seurat
## 13248 features across 633 samples within 1 assay
## Active assay: RNA (13248 features, 0 variable features)
```

#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.

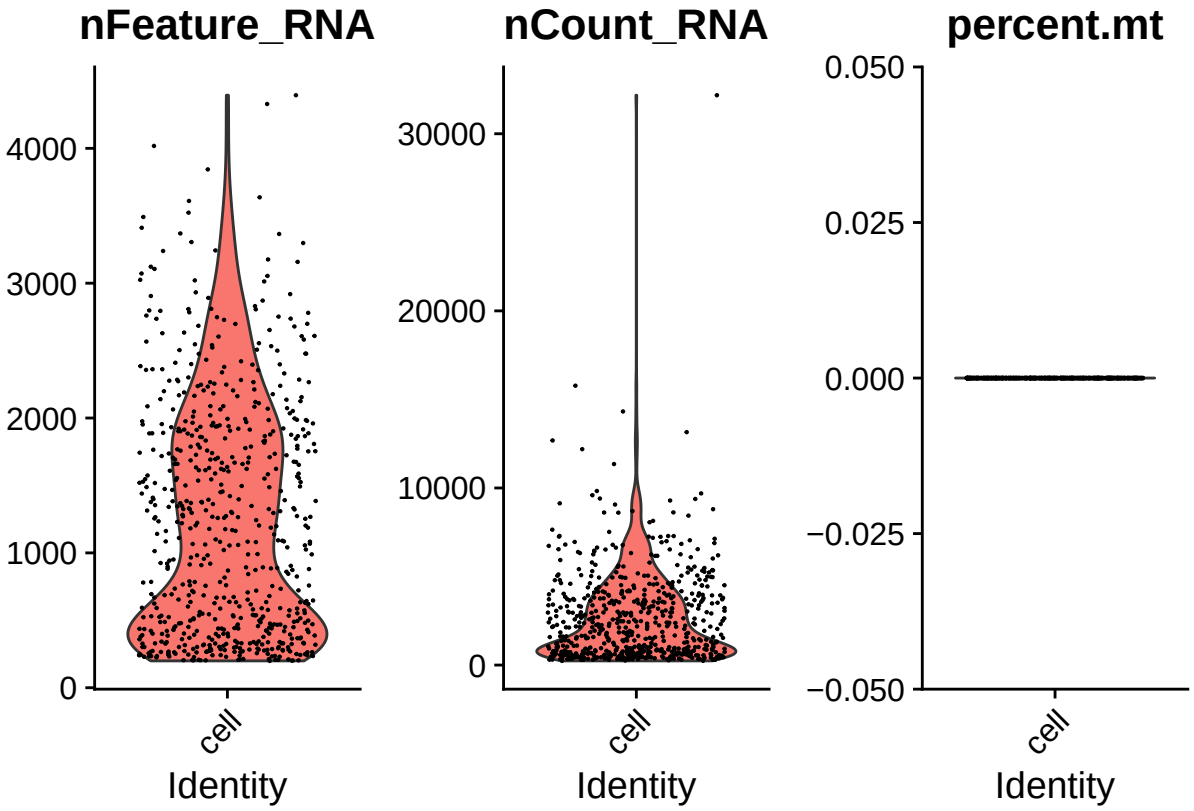
```
pdac1[["percent.mt"]] <- PercentageFeatureSet(pdac1, pattern = "^MT-")
head(pdac1$percent.mt)
```

```
## cell_091.133 cell_177.113 cell_289.088 cell_205.268 cell_162.063 cell_183.039
##            0            0            0            0            0            0
```

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

```
VlnPlot(object = 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 <- subset(x = 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 LogNormalize method with a scale factor of 10,000

```
pdac1_subset <- NormalizeData(object = pdac1_subset, normalization.method = "LogNormalize", scale.facto
```

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

```
pdac1_subset <- FindVariableFeatures(object = pdac1_subset,selection.method = 'vst', nfeatures = 2000)

# Identify the 10 most highly variable genes
top10 <- head(x = VariableFeatures(object = pdac1_subset), 10)

# plot variable features with and without labels
plot1 <- VariableFeaturePlot(object = pdac1_subset)
plot2 <- LabelPoints(plot = plot1, points = top10, repel = TRUE)
```

```
## When using repel, set xnudge and ynudge to 0 for optimal results
```

```
CombinePlots(plots = list(plot1, plot2))
```

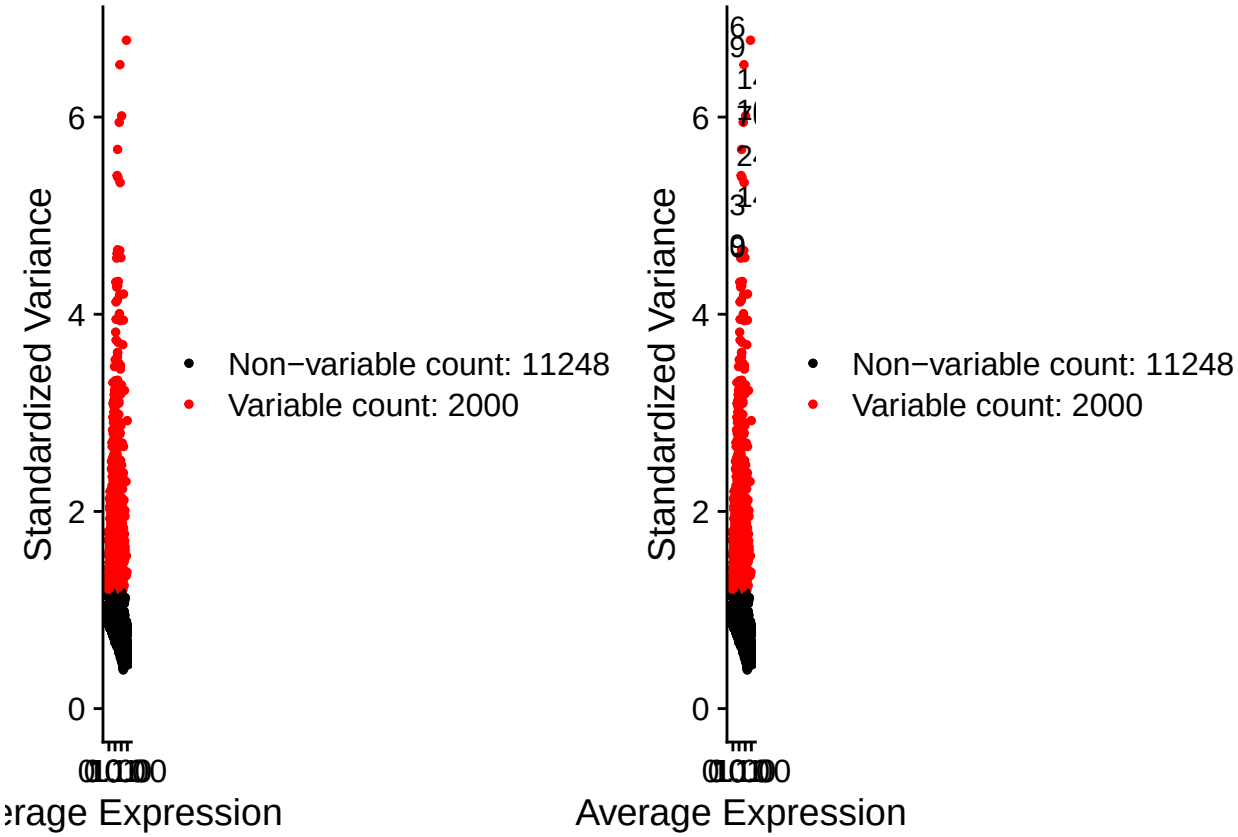
```
## Warning: CombinePlots is being deprecated. Plots should now be combined using
## the patchwork system.
```

```
## Warning: Transformation introduced infinite values in continuous x-axis
```

```
## Warning: Removed 21 rows containing missing values (geom_point).
```

```
## Warning: Transformation introduced infinite values in continuous x-axis
```

```
## Warning: Removed 21 rows containing missing values (geom_point).
```



#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(x = pdac1_subset)
pdac1_subset <- ScaleData(object = pdac1_subset, features = all.genes)
```

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

```
pdac1_subset <- ScaleData(object = pdac1_subset)
```

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

#Step 8: PCA #Step 9: Visualize data using VizDimLoadings and DimPlot functions. Can you tell from the PCA analyis, the number of cell types that are present? Not clearly, approximately 500 cell types

```
pdac1_subset <- RunPCA(object = pdac1_subset, features = VariableFeatures(object = pdac1_subset))
```

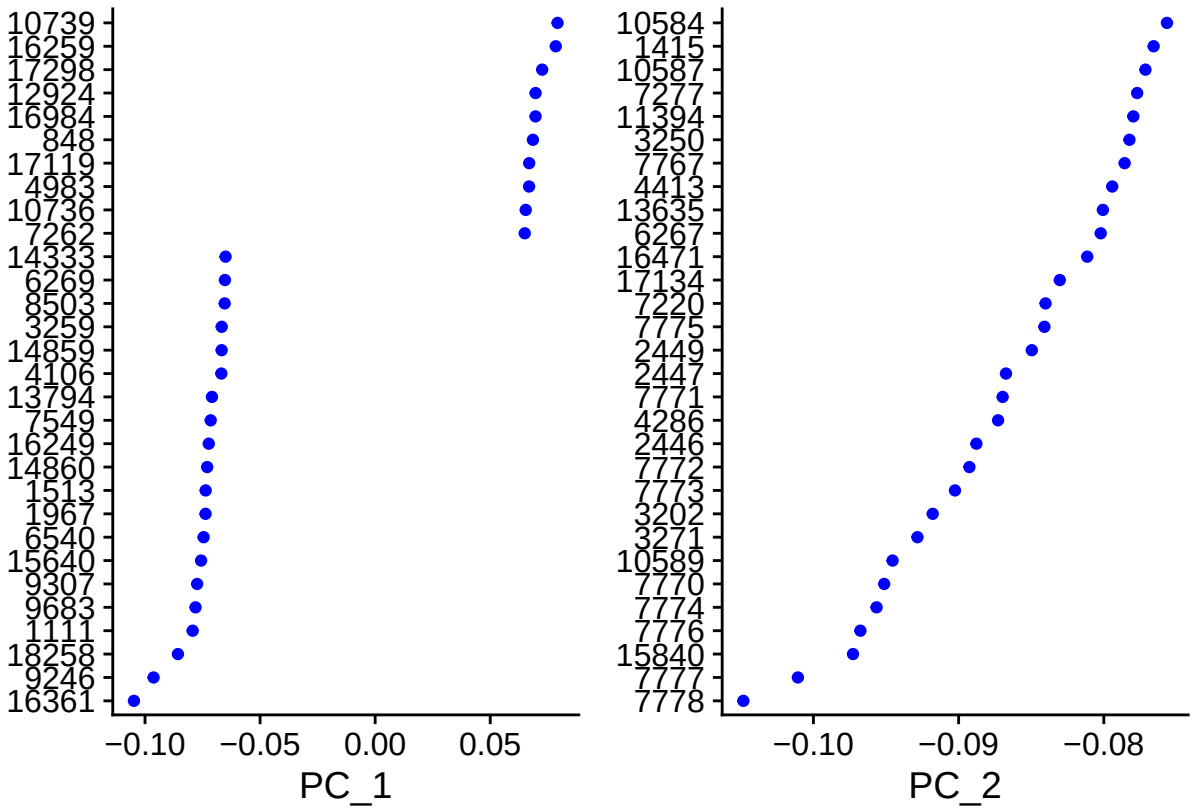
```
## PC_ 1
## Positive:  10739, 16259, 17298, 12924, 16984, 848, 17119, 4983, 10736, 7262
##     16289, 10370, 2092, 16770, 17117, 1378, 5227, 6798, 14234, 1418
##     4231, 15868, 14850, 5207, 5634, 18411, 4246, 10733, 926, 2568
```

```
## 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:  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
## Negative:  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
```

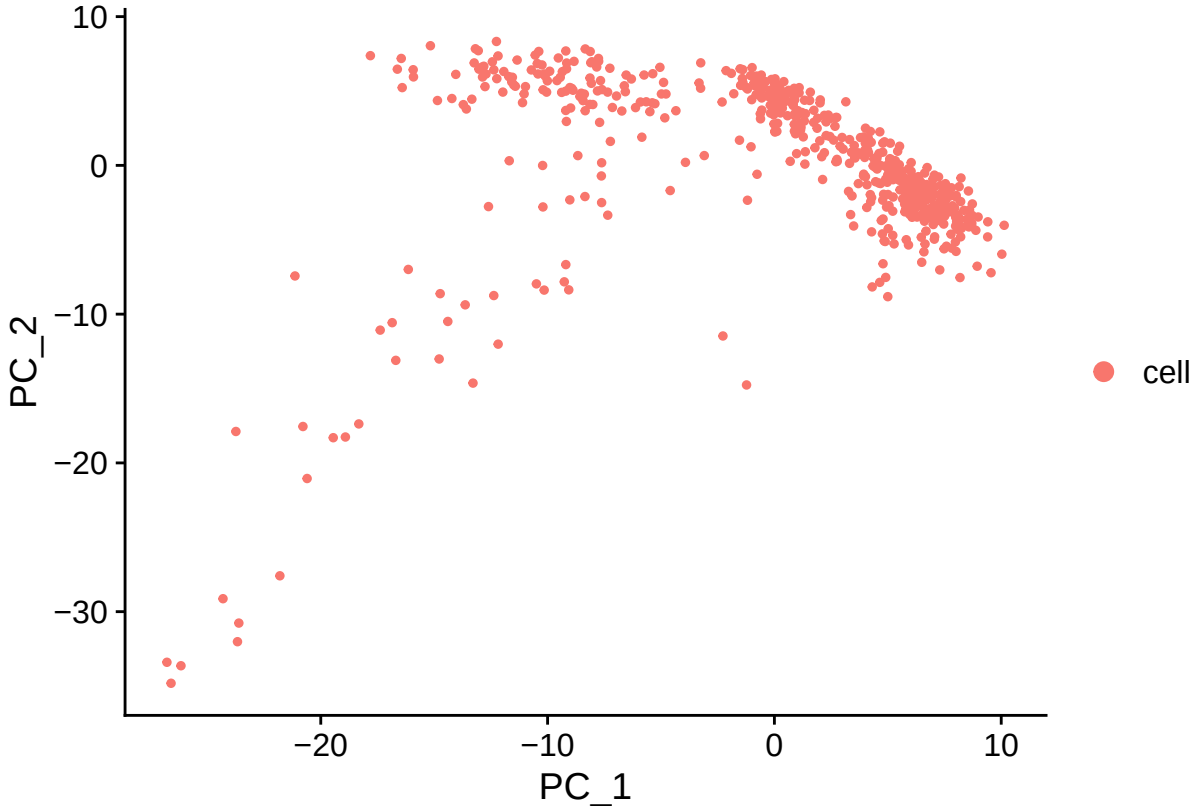
```
print(x = pdac1_subset[['pca']], dims = 1:5, nfeatures = 5)
```

```
## PC_ 1
## Positive:  10739, 16259, 17298, 12924, 16984
## Negative:  16361, 9246, 18258, 1111, 9683
## PC_ 2
## Positive:  6649, 2011, 4481, 16446, 7543
## Negative:  7778, 7777, 15840, 7776, 7774
## PC_ 3
## Positive:  4036, 4009, 9714, 4035, 12925
## Negative:  4231, 10739, 15183, 4983, 9758
## PC_ 4
## Positive:  14863, 14860, 14859, 6649, 6540
## Negative:  3135, 3268, 8277, 13440, 7459
## PC_ 5
## Positive:  2136, 15317, 7910, 1881, 9556
## Negative:  3135, 3268, 8277, 7459, 13440
```

```
VizDimLoadings(object = pdac1_subset, dims = 1:2, reduction = 'pca')
```



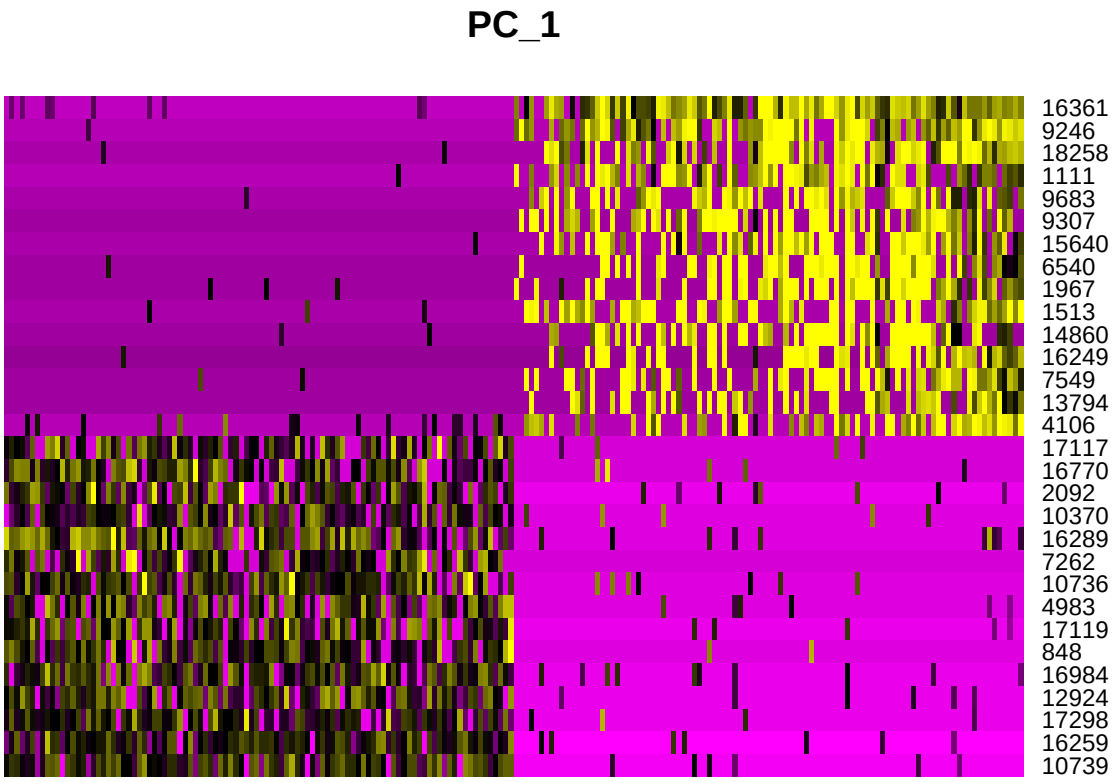

```
DimPlot(object = pdac1_subset, reduction = 'pca')
```



#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(object = pdac1_subset, dims = 1, cells = 200, balanced = TRUE)
```



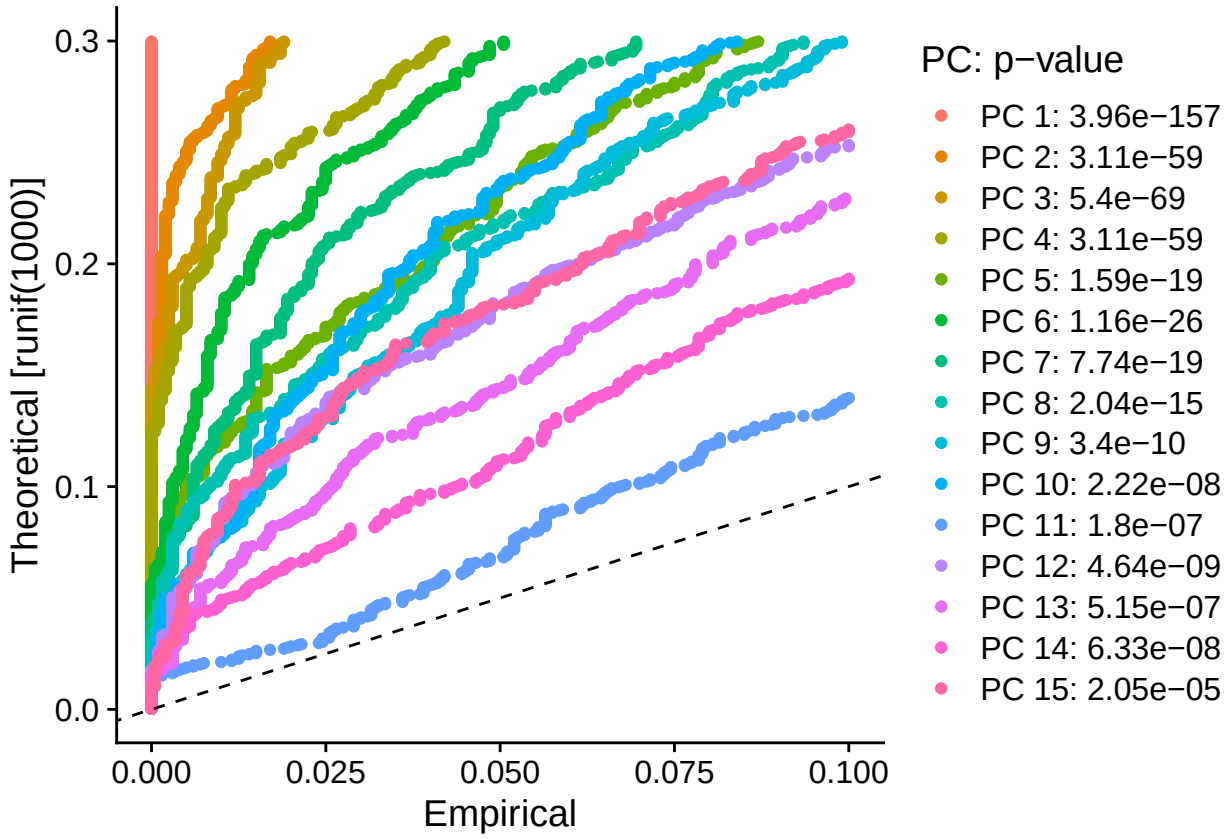
#Step 11: Dimensionality To make this more quantitative, let's see when does the variation reach the lowest amount of variation explained. Use the JackStraw method with 100 replicates and score the first 20 dimensions.

```
pdac1_subset <- JackStraw(object = pdac1_subset, num.replicate = 100)
pdac1_subset <- ScoreJackStraw(object = pdac1_subset, dims = 1:20)
```

Plot the results for the first 20 dimensions.

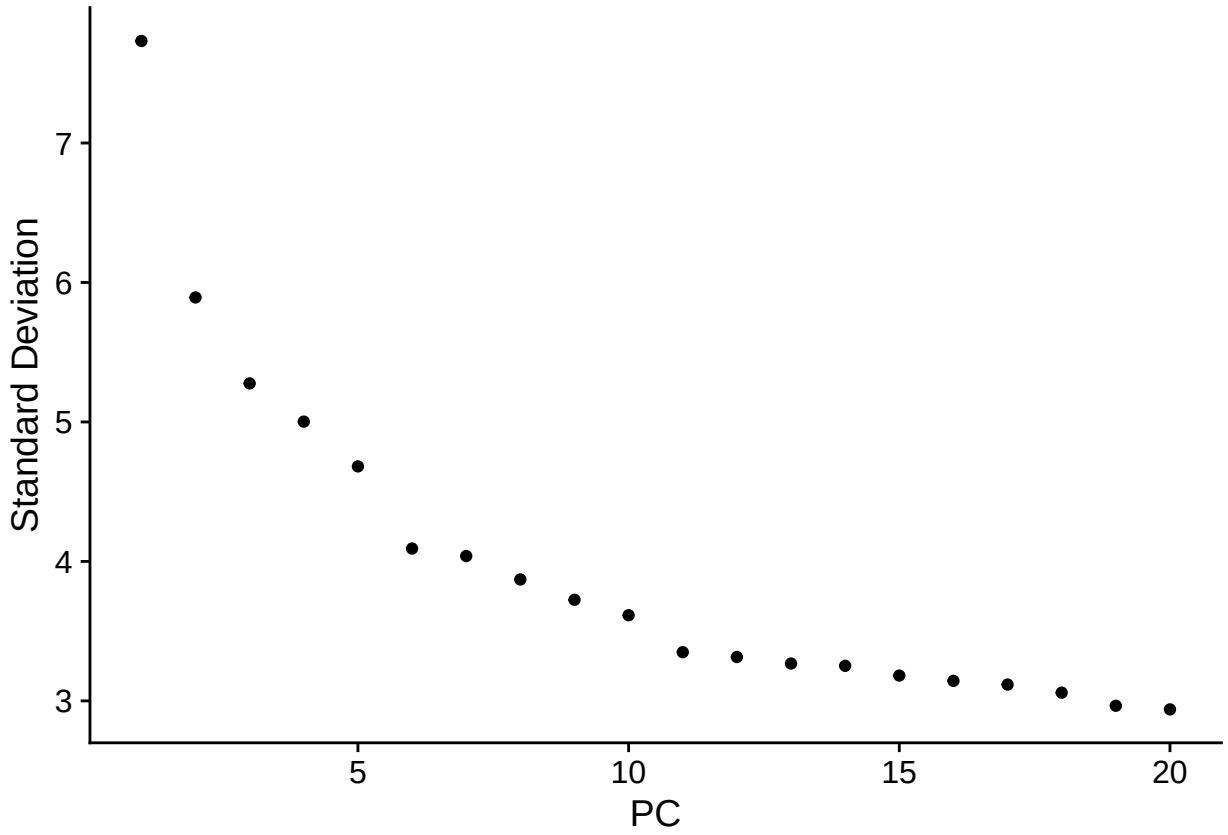
```
JackStrawPlot(object = pdac1_subset, dims = 1:15)
```

```
## Warning: Removed 21846 rows containing missing values (geom_point).
```



Use the elbow plot

```
ElbowPlot(object = pdac1_subset)
```



#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.

```
pdac1_subset <- FindNeighbors(object = pdac1_subset, dims = 1:10)
```

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

```
## Computing SNN
```

```
pdac1_subset <- FindClusters(object = pdac1_subset, resolution = 0.5)
```

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 569
## Number of edges: 14355
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.8635
## Number of communities: 7
## Elapsed time: 0 seconds
```

And then identify the clusters using the FindClusters function.

```
head(x = Idents(object = pdac1_subset), 5)
```

```
## cell_091.133 cell_229.285 cell_359.174 cell_143.259 cell_013.161
##            1            0            5            3            5
## Levels: 0 1 2 3 4 5 6
```

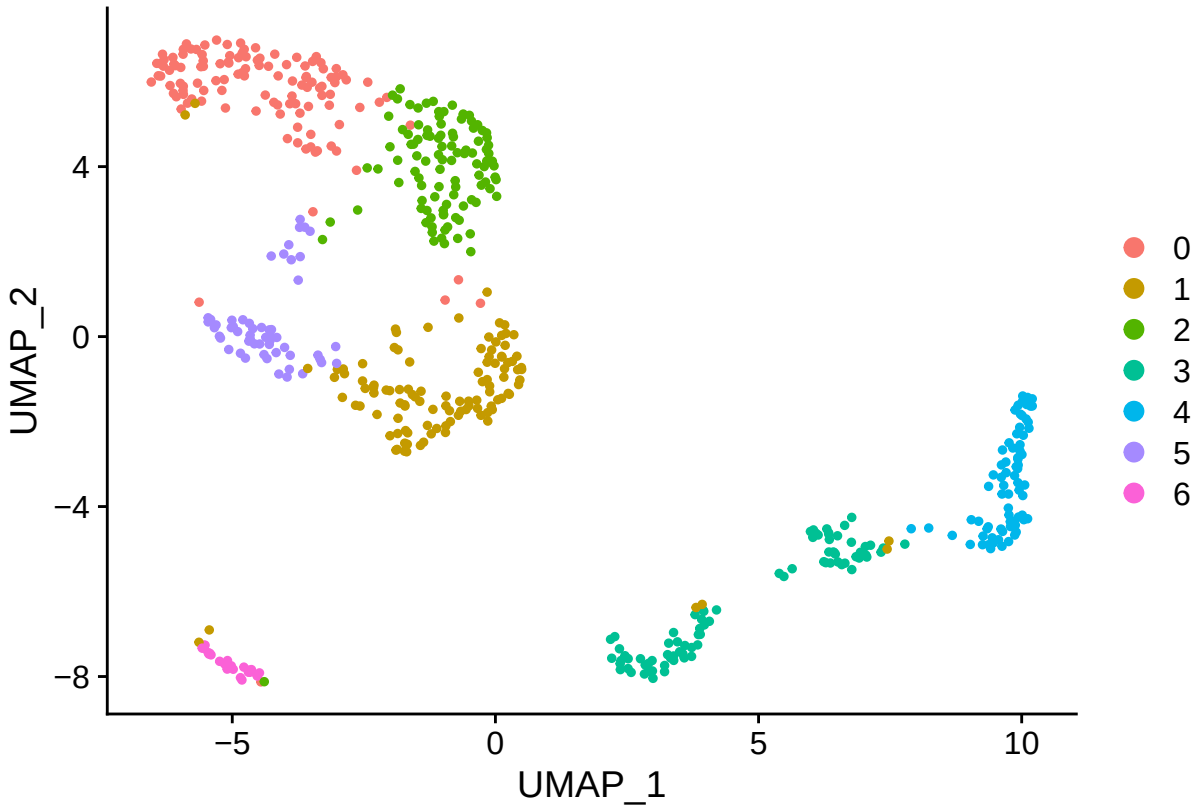
#tsne/umap #Step 13: Perform a UMAP analysis using the first 9 dimensions using RunUMAP and then visualize it using DimPlot. How many clusters do you get? How many possible mistakes do you see? - About

6 distinct clusters, maybe around 8 misplaced cells?

```
pdac1_subset <- RunUMAP(object = pdac1_subset, dims = 1:10)
```

```
## Warning: The default method for RunUMAP has changed from calling Python UMAP via reticulate to the R
## To use Python UMAP via reticulate, set umap.method to 'umap-learn' and metric to 'correlation'
## This message will be shown once per session
## 21:09:36 UMAP embedding parameters a = 0.9922 b = 1.112
## 21:09:36 Read 569 rows and found 10 numeric columns
## 21:09:36 Using Annoy for neighbor search, n_neighbors = 30
## 21:09:36 Building Annoy index with metric = cosine, n_trees = 50
## 0%   10   20   30   40   50   60   70   80   90   100%
## [----|----|----|----|----|----|----|----|----|----|
## **************************************************|
## 21:09:36 Writing NN index file to temp file /tmp/RtmpwR5LK5/file30bc6362cbc670
## 21:09:36 Searching Annoy index using 1 thread, search_k = 3000
## 21:09:36 Annoy recall = 100%
## 21:09:36 Commencing smooth kNN distance calibration using 1 thread
## 21:09:36 Initializing from normalized Laplacian + noise
## 21:09:36 Commencing optimization for 500 epochs, with 20638 positive edges
## 21:09:37 Optimization finished
```

```
DimPlot(object = pdac1_subset, reduction = 'umap')
```



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

```
cluster1.markers <- FindMarkers(object = pdac1_subset, ident.1 = 0, logfc.threshold = 0.25, test.use =
head(cluster1.markers)
```

```
##       myAUC avg_diff power avg_log2FC pct.1 pct.2
## 3438  0.863 1.216005 0.726   1.754324 0.983 0.380
## 17303 0.858 1.194031 0.716   1.722623 0.958 0.384
## 5225  0.858 1.164801 0.716   1.680453 0.966 0.371
## 9047  0.853 1.024254 0.706   1.477686 0.992 0.493
## 10727 0.847 1.041779 0.694   1.502969 0.966 0.400
## 16289 0.842 1.632769 0.684   2.355587 0.882 0.302
```

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

```
#VlnPlot(object = pdac1_subset, features = cluster1.markers)
```

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

```
#FeaturePlot(object = pdac1_subset, features = c("MS4A1", "GNLY", "CD3E", "CD14", "FCER1A", "FCGR3A", "
```