# Single Cell RNA-seq

Research Informatics Core

March 1, 2023

## Contents

# 1 Morning

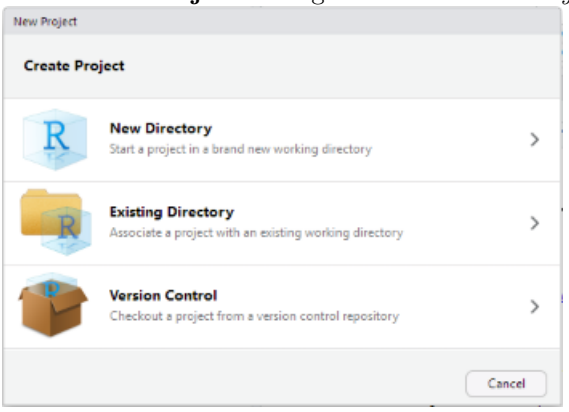
## REFRESHER: Using R Studio

### *TIPS*

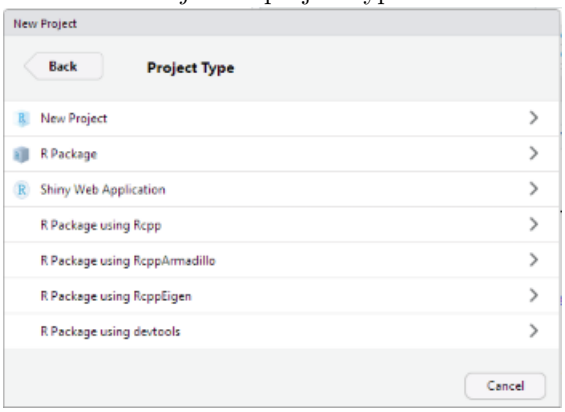
- Use Tab to auto complete
- Use up arrow to get previous command

#### 1.0.1 Create a new project in R Studio

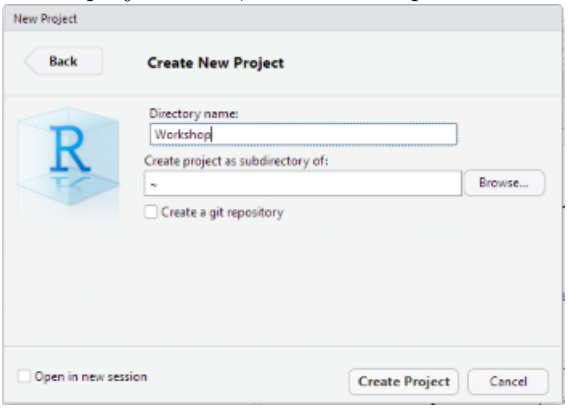
1. From *File* menu select *New Project...*
2. From **New Project** Dialog select *New Directory*




3. Select *New Project* as project type.



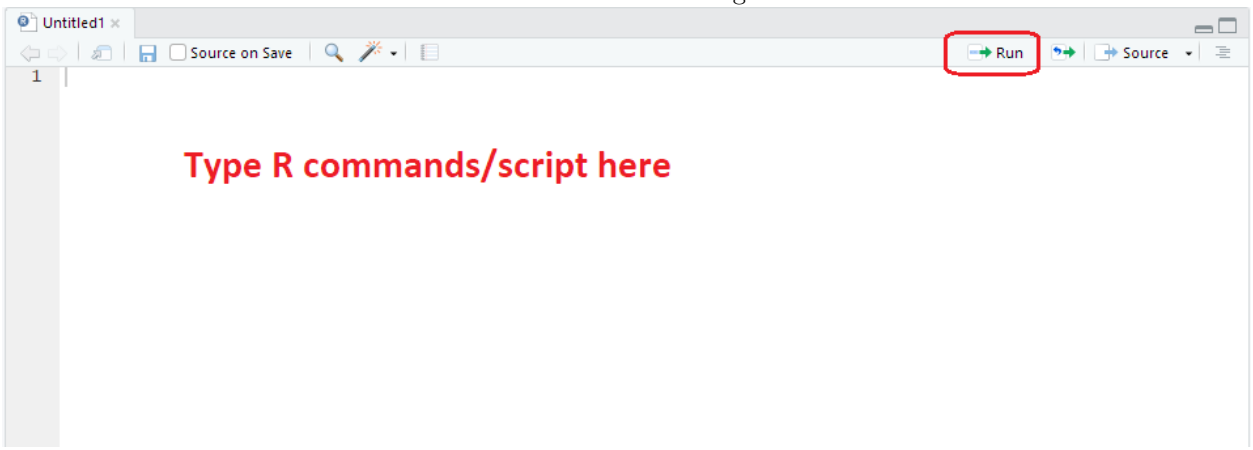

4. Give a project name, i.e. "Workshop" and click **Create Project** button.



### 1.0.2 Create a new script.

1. From the *File* menu select *New File* > *R Script*
2. Save file as "my_Rscript.R"

Write commands in code editor of R Studio and run them using icon **Run** in R Studio.



## PREPARATION: Installing R packages

We need several R packages for our analysis today. Some are installed through bioconductor, some from CRAN through install.packages.

- While installing packages, you may get messages about updating other packages. This is generally optional: you can select not to (enter 'n' in the prompt), but at some point it's a good idea to update.

### Bioconductor

- If you have not already done so, install BiocManager for Bioconductor tools:

```
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
```

### Seurat

- Seurat requires the multtest package from Bioconductor, but it is installed from CRAN:

```
BiocManager::install('multtest', update=F)
install.packages('Seurat')
```

### Other CRAN packages

- Install Matrix, dplyr, reshape2, and fossil from CRAN:

```
install.packages('Matrix')
install.packages('dplyr')
install.packages('reshape2')
install.packages('fossil')
```

### Bioconductor packages

- Install ComplexHeatmap and monocle from Bioconductor:

```
BiocManager::install("ComplexHeatmap", update=F)
BiocManager::install("monocle", update=F)
```

### Check installations

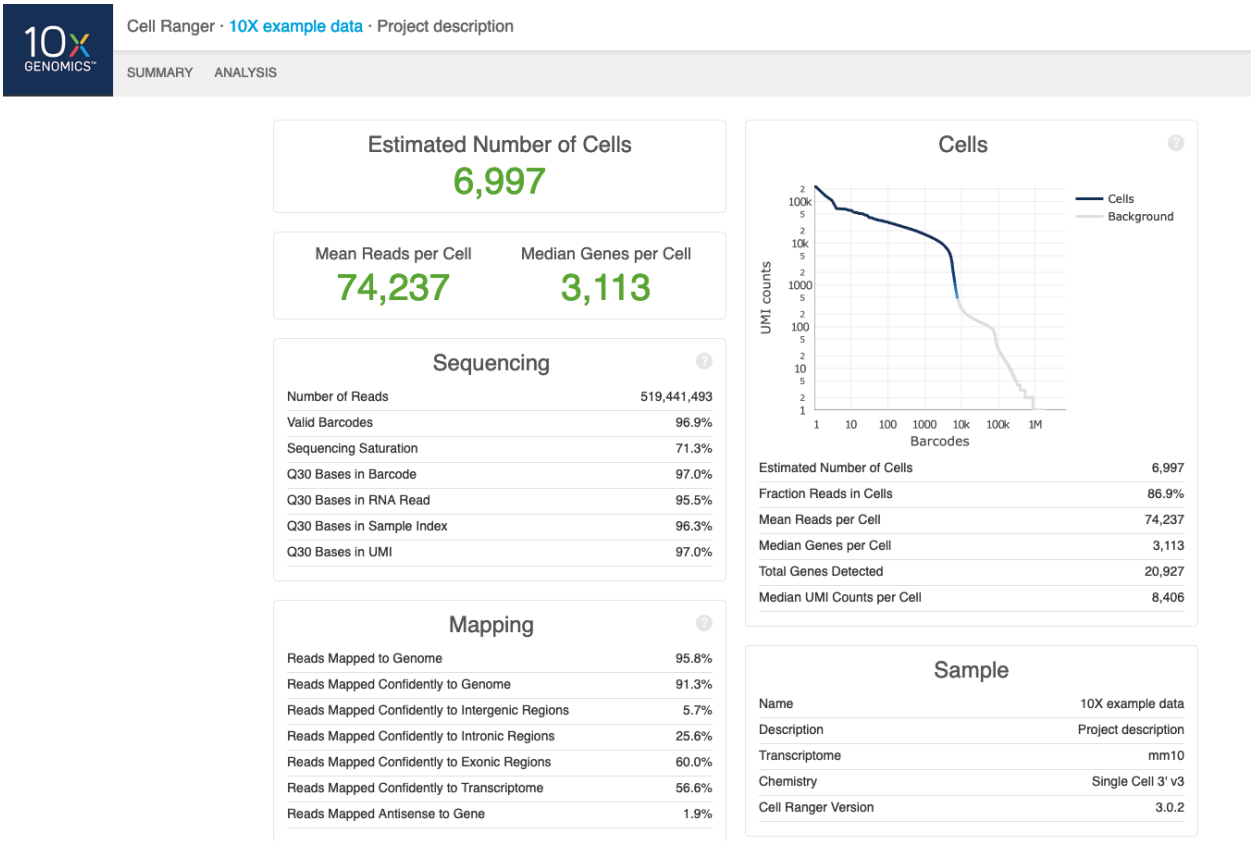
- Try to load each package to confirm that the installation was successful:

```
library(Seurat)
library(Matrix)
library(dplyr)
library(reshape2)
library(fossil)
library(ComplexHeatmap)
library(monocle)
```

## 1.1 View CellRanger report

- Look at example quantification and demultiplexing report from CellRanger (for 10X)
- Note these statistics:
  - Number of cells, and associated barcode coverage plot
  - Median UMI counts per cell
  - Median genes per cell
  - Mapping percent to genome
  - Mapping percent to transcriptome
  - Barcode identification percent
  - Sequencing saturation

Open this link in a web browser: https://wd.cri.uic.edu/sc_rna/10X_example_report.html



## 1.2 Read single-cell data into R

Practice with inDrop data (full matrix) and 10X data (sparse matrix)

### 1.2.1 inDrop data

1. Load the libraries first

```
library(Seurat)
library(Matrix)
library(dplyr)
```

2. Read in inDrop data first with regular read.table function. We can read the inDrop matrix directly from the URL.

```
counts_indrop <- read.table("https://wd.cri.uic.edu/sc_rna/inDrop_counts.txt",
   header=T,row.names=1,sep="\t")
```

3. Convert to sparse matrix with the Matrix package

```
sparse_indrop <- Matrix(as.matrix(counts_indrop),sparse=T)
```

4. (*OPTIONAL*) You can compare the sizes of the original and sparse matrix

```
format(object.size(counts_indrop), units="auto")
```

```
[1] "46.8 Mb"
```

```
format(object.size(sparse_indrop), units="auto")
```

```
[1] "13.4 Mb"
```

5. Create the Seurat object from the sparse matrix.

```
seurat_indrop <- CreateSeuratObject(counts=sparse_indrop, project="indrop_sample_data")
seurat_indrop
```

```
An object of class Seurat
9435 features across 1275 samples within 1 assay
Active assay: RNA (9435 features, 0 variable features)
```

### 1.2.2 10X data

- First download this file from a web browser: https://wd.cri.uic.edu/sc_rna/10X_test.zip
- Then unzip it on your computer.
- **NOTE: we're assuming you have downloaded it to your ~/Downloads folder.** Please change the path in the first command below as necessary based on where you downloaded the data.
- When we use the Read10X function, we give it a folder name, and it expects to find 3 files in that folder:
  1. matrix.mtx.gz (gene expression matrix in sparse format)
  2. features.tsv.gz (list of gene names, usually with 2 columns, Ensembl ID and gene symbol)
  3. barcodes.tsv.gz (list of cell barcodes)
- If you want to do the analysis with respect to gene symbols, you'd use 'gene.column=2' below. But some gene symbols will be duplicated. Best to use IDs as the primary identifier, and we'll show how to bring gene symbols back in at Exercise 2.1.

1. Read the 10X data.
   - If you are using a **macOS** computer and unzipped the file in your `Downloads` folder, execute the following.

```
sparse_10X <- Read10X("~/Downloads/10X_test/filtered_feature_bc_matrix/",
                      gene.column=1)
```

   - If you are using a **Windows** computer and unzipped the file in your `Downloads` folder, execute the following.

```
sparse_10X <- Read10X("~/../Downloads/10X_test/filtered_feature_bc_matrix/",
                      gene.column=1)
```

2. Create the Seurat object

```
seurat_10X <- CreateSeuratObject(counts=sparse_10X, project="10X_sample_data")
```

3. View the created Seurat object

```
seurat_10X
```

```
An object of class Seurat
31053 features across 1301 samples within 1 assay
Active assay: RNA (31053 features, 0 variable features)
```

## 1.3 Quality control and filtering

### 1.3.1 Download the data

- Now we will analyze 2 samples captured on a 10X. Please download zip files with the data first:
  1. https://wd.cri.uic.edu/sc_rna/V035_F031_subsample.zip
  2. https://wd.cri.uic.edu/sc_rna/V039_F093_subsample.zip
- Again unzip these on your computer. **Note the folder these are downloaded to. The exercises below assume they are in your ~/Downloads folder.**

*ABOUT*: These are two samples from a recently published study, Martos SN, Campbell MR, Lozoya OA, Wang X *et al.* Single-cell analyses identify dysfunctional CD16+ CD8 T cells in smokers. *Cell Rep Med* 2020 Jul 21;1(4). PMID: 33163982.

- Peripheral blood cells from smoking and nonsmoking adults were extracted, captures run on 10X.
- GEO accession GSE138867, the samples we're looking at are GSM4120733/V035_F031 (smoker), GSM4120734/V039_F093 (non-smoker).
- The cells in these samples have been sub-sampled to 20% of the original total, to save on computational time during the workshop.

### 1.3.2 Read the data into R

1. Read the 10X files
   - If you are using a **macOS** computer and unzipped the file in your `Downloads` folder, execute the following.

```
sample1 <- Read10X("~/Downloads/V035_F031_subsample", gene.column=1)
sample2 <- Read10X("~/Downloads/V039_F093_subsample", gene.column=1)
```

   - If you are using a **Windows** computer and unzipped the file in your `Downloads` folder, execute the following.

```
sample1 <- Read10X("~/../Downloads/V035_F031_subsample", gene.column=1)
sample2 <- Read10X("~/../Downloads/V039_F093_subsample", gene.column=1)
```

2. Create the Seurat objects

```
seurat1 <- CreateSeuratObject(counts=sample1, project="sample1")
seurat1
```

```
An object of class Seurat
32738 features across 1629 samples within 1 assay
Active assay: RNA (32738 features, 0 variable features)
```

```
seurat2 <- CreateSeuratObject(counts=sample2, project="sample2")
seurat2
```

```
An object of class Seurat
32738 features across 1334 samples within 1 assay
Active assay: RNA (32738 features, 0 variable features)
```

3. Combine the seurat objects together to make one master data set. The `add.cell.ids` parameter specifies a prefix to add to the cell IDs when combining the data.

```
sc_data <- merge(seurat1, y=seurat2, add.cell.ids=c("s1","s2"))
sc_data
```

```
An object of class Seurat
32738 features across 2963 samples within 1 assay
Active assay: RNA (32738 features, 0 variable features)
```

4. **NOTE**! `orig.ident` is how we're keeping track of which file came from which sample

```
head(sc_data$orig.ident)
```

```
s1_AAACCTGAGCACCGTC-1 s1_AAACCTGAGCGTGAAC-1 s1_AAACCTGGTAGAGCTG-1
            "sample1"             "sample1"             "sample1"
s1_AAACCTGGTCTAGCGC-1 s1_AAACCTGGTGTGGCTC-1 s1_AAACGGGAGTGTACGG-1
            "sample1"             "sample1"             "sample1"
```

5. Count how many cells are in each sample using `table()` function

```
table(sc_data$orig.ident)
```

```
sample1 sample2
   1629    1334
```

### 1.3.3 Basic quality control checks

- Check mitochondrial counts per cell: high levels of mitochondrial expression are an indication of a dying cell, as mitochrondrial contents spill into the cytoplasm.
- Also check the distribution of UMI counts and genes expressed per cell.

1. Read in a list of the coordinates for each gene so, that we know which genes are on chrM

```
genes <- read.table("https://wd.cri.uic.edu/sc_rna/hg19_genes.bed")
dim(genes)
```

```
[1] 43319     6
```

```
head(genes)
```

```
     V1        V2        V3              V4 V5 V6
1  chrX  99883667  99894988 ENSG00000000003  0  -
2  chrX  99839799  99854882 ENSG00000000005  0  +
3 chr20  49551404  49575092 ENSG00000000419  0  -
4  chr1 169818772 169863408 ENSG00000000457  0  -
5  chr1 169631245 169823221 ENSG00000000460  0  +
6  chr1  27938575  27961788 ENSG00000000938  0  -
```

2. Make a list of the mitochondrial genes

```
mt.genes <- as.character(genes[genes[,1]=="chrM", 4])
length(mt.genes)
```

```
[1] 13
```

3. Compute percent mitochondrial expression per cell

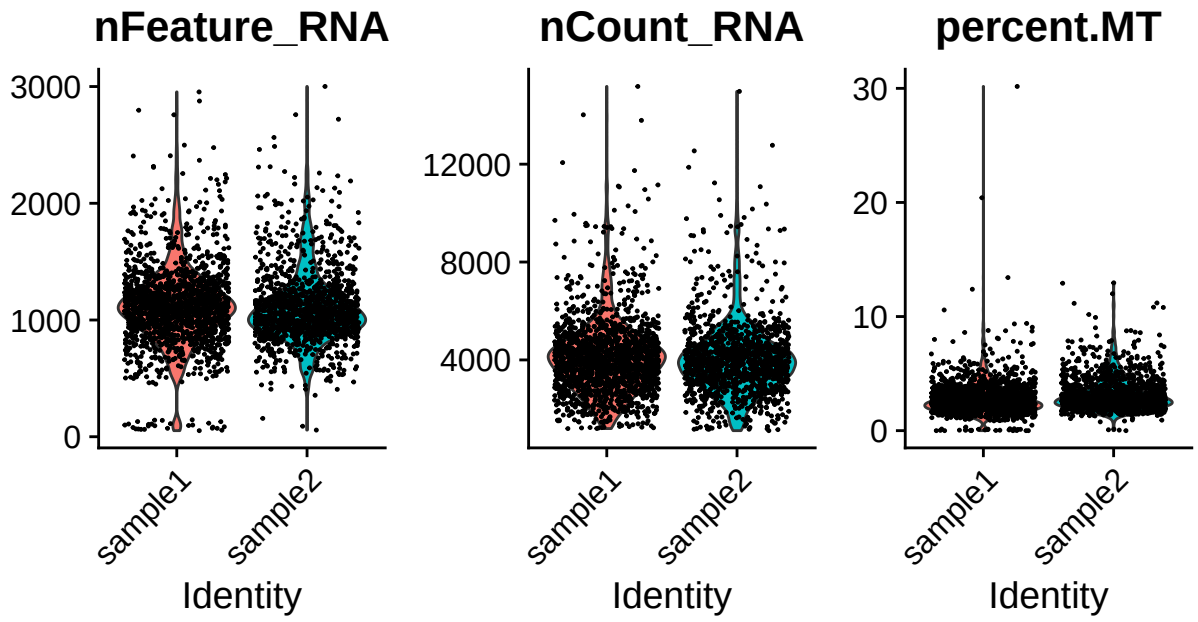
```
sc_data[["percent.MT"]] <- PercentageFeatureSet(sc_data, features=mt.genes)
```

NOTE*: There are a few ways to get a list of the mitochondrial gene IDs for your genome.*

- You can download BED files similar to this one from UCSC genome browser's Table Browser `https://genome.ucsc.edu/cgi-bin/hgTables`, just make sure to get the one that matches the gene IDs in your file.
- You can also download or request a copy of the GTF file used to quantify your expression data; GTF is a different format, but can also be used to get a list of gene IDs and the chromosomes they are on.
- Finally, you can also ask your local bioinformatics core for assistance!

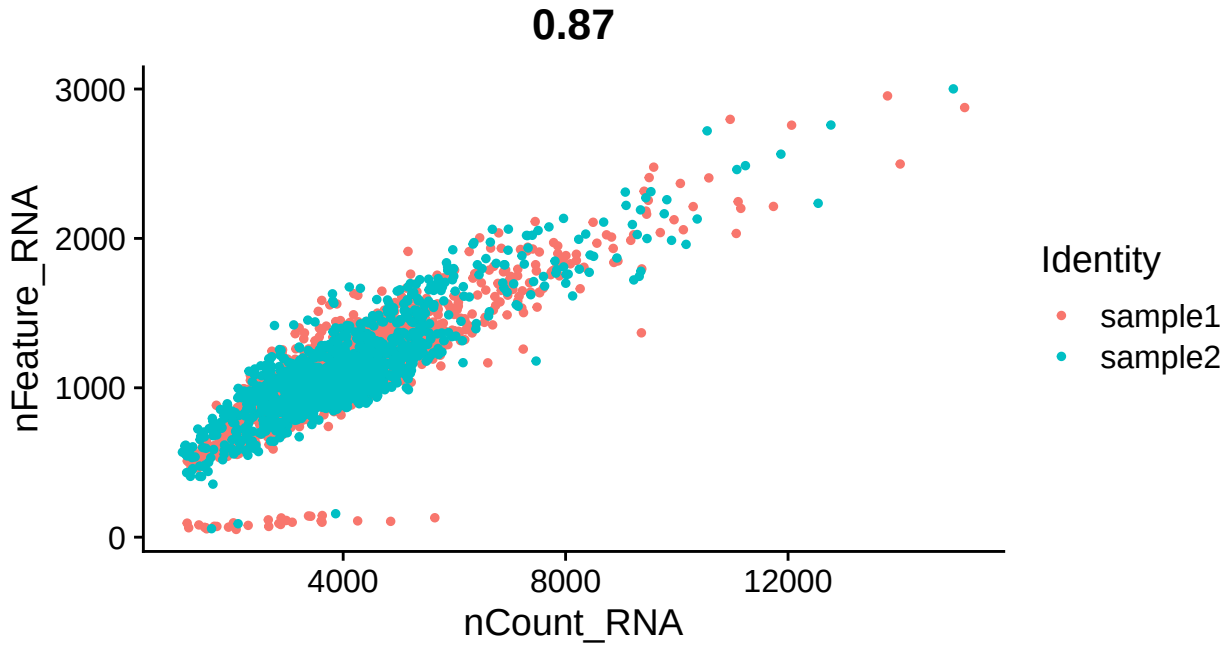
4. Make plots of the distribution of counts, genes, and mitochondrial expression

```
VlnPlot(sc_data, features = c("nFeature_RNA","nCount_RNA","percent.MT"), pt.size=0.2)
```



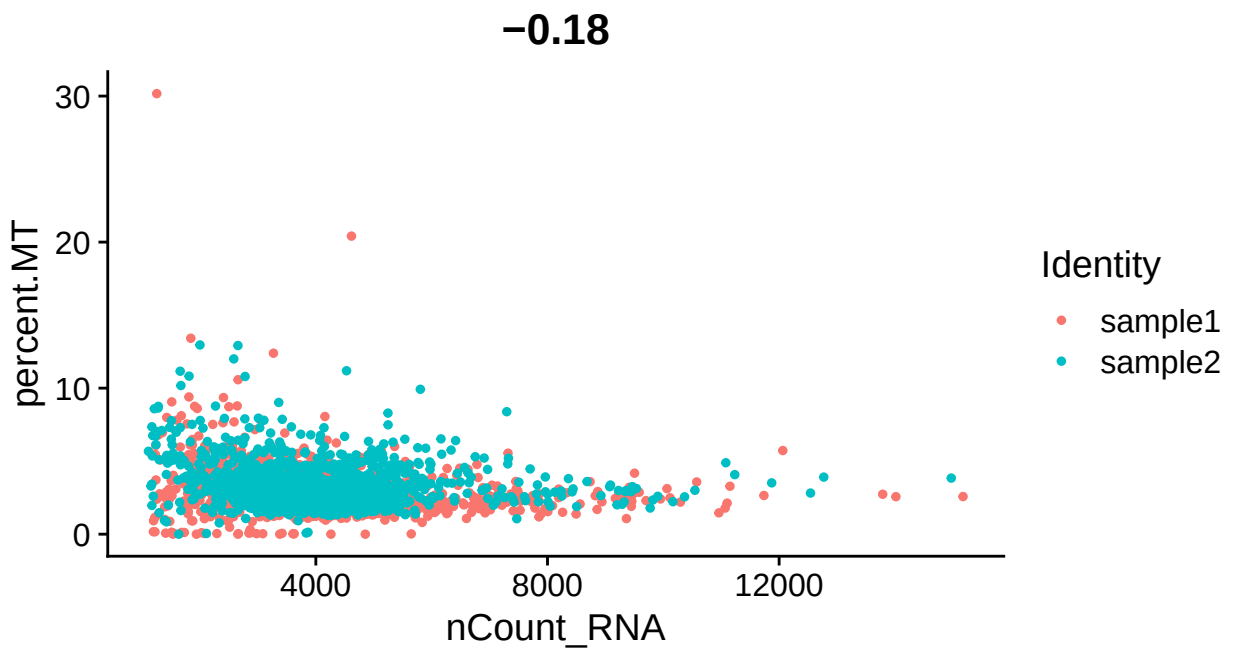

5. Make scatterplot of number of UMI counts vs. number of genes

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



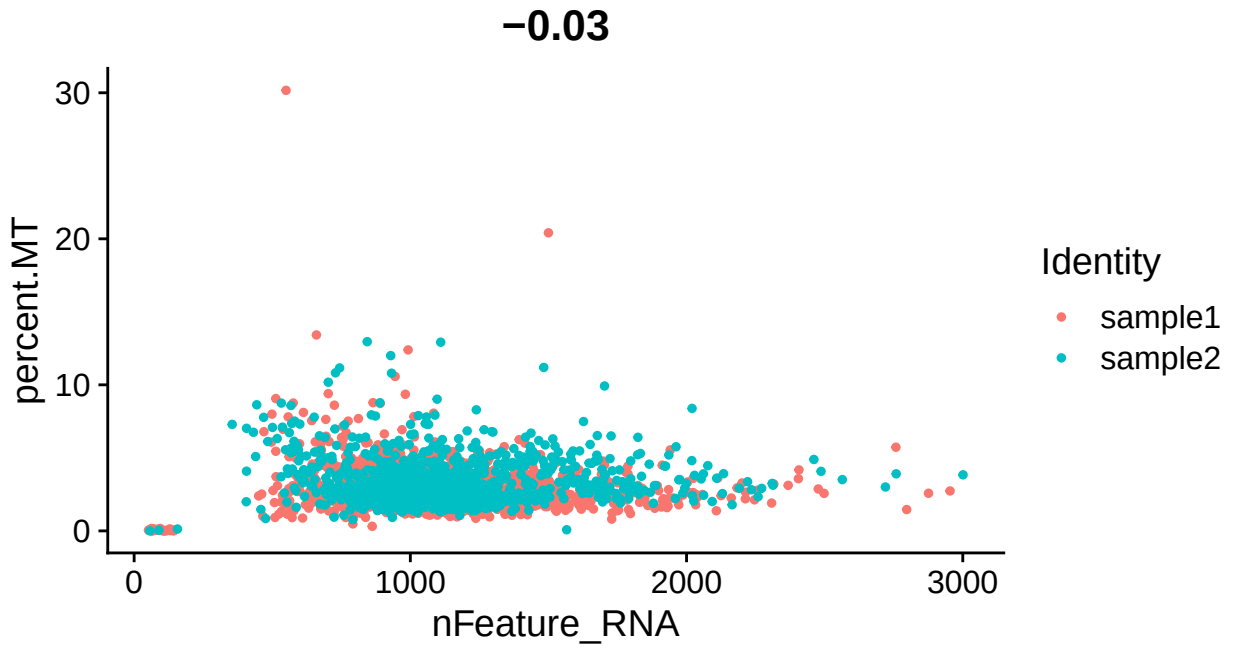
6. Make scatterplot of number of UMI counts vs % mitochondrial expression

```
FeatureScatter(sc_data, feature1="nCount_RNA", feature2="percent.MT")
```




7. Make scatterplot of number of genes vs. pecent mitochondrial expression

```
FeatureScatter(sc_data, feature1="nFeature_RNA", feature2="percent.MT")
```



### 1.3.4 Filtering cells for downstream analysis

Based on the violin plots and scatterplots, decide on a reasonable threshold for filtering out the "bad" cells.

- We want to remove cells with very low expression, or high levels of mitochondrial expression, as those have less reliable signals or may be dying.
- We may want to remove cells with very high expression as possible doublets. But, they could also just be high-expressing cells.
- We're usually looking for bimodal distributions in the violin plots, which would indicate a second population of low-quality cells, and then what threshold would separate the modes of the distributions.

For today, our passing cells must have:

- Number of genes (nFeature) per cell greater than 500
- UMI counts (nCount) per cell greater than 2000
- Less than 10% mitochondrial content

```
sc_subset <- subset(sc_data,
                    nFeature_RNA>500 & nCount_RNA>2000 & percent.MT < 10)
sc_subset
```

```
An object of class Seurat
32738 features across 2791 samples within 1 assay
Active assay: RNA (32738 features, 0 variable features)
```

### 1.3.5 Check what fraction of cells we kept from each sample, using the `table()` function

1. Get the counts of cell for each sample before filtering

```
orig.counts <- table(sc_data$orig.ident)
```

2. Get the counts of cells for each sample after filtering.

```
subset.counts <- table(sc_subset$orig.ident)
```

3. Create a data.frame of the original counts, filtered counts and fraction of cells retained.

```
cell_stats <- cbind(orig.counts, subset.counts, subset.counts/orig.counts)
colnames(cell_stats) <- c("Starting Cells","Retained Cells","Fraction")
cell_stats
```

```
        Starting Cells Retained Cells  Fraction
sample1           1629           1523 0.9349294
sample2           1334           1268 0.9505247
```

## 1.4 Gene feature selection and scaling

- Normalize gene expression
- Find variable genes
- Scale (z-score) genes
- Run PCA

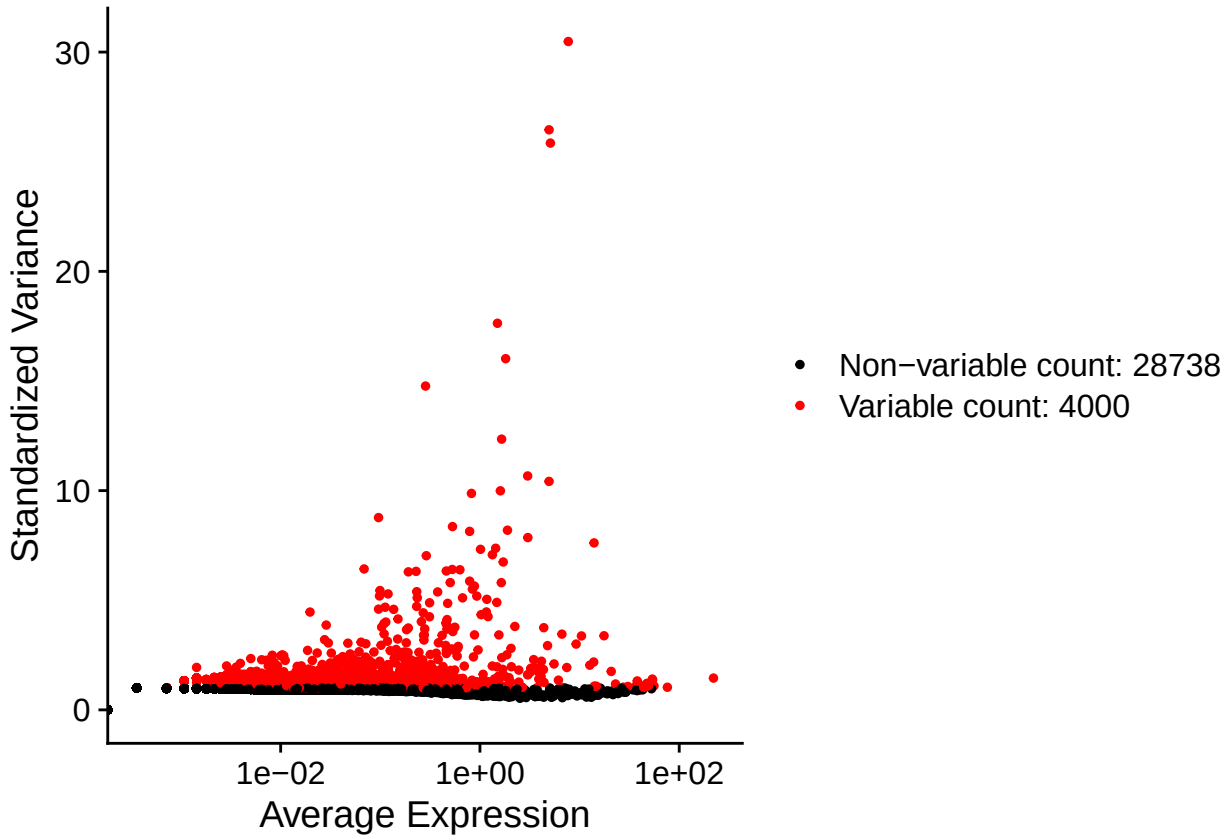
1. Normalized the data

```
sc_subset <- NormalizeData(sc_subset)
```

2. Find variable genes. In practice, you may want to vary the number of features you select here based on the variable features plot.

```
sc_subset <- FindVariableFeatures(sc_subset, nfeatures=4000)
VariableFeaturePlot(sc_subset)
```

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




3. Scale (z-score) the data. ***NOTE*** the `ScaleData` function will only scale the features that are identified as variable via the `FindVariableFeatures()` function.

```
sc_subset <- ScaleData(sc_subset)
```

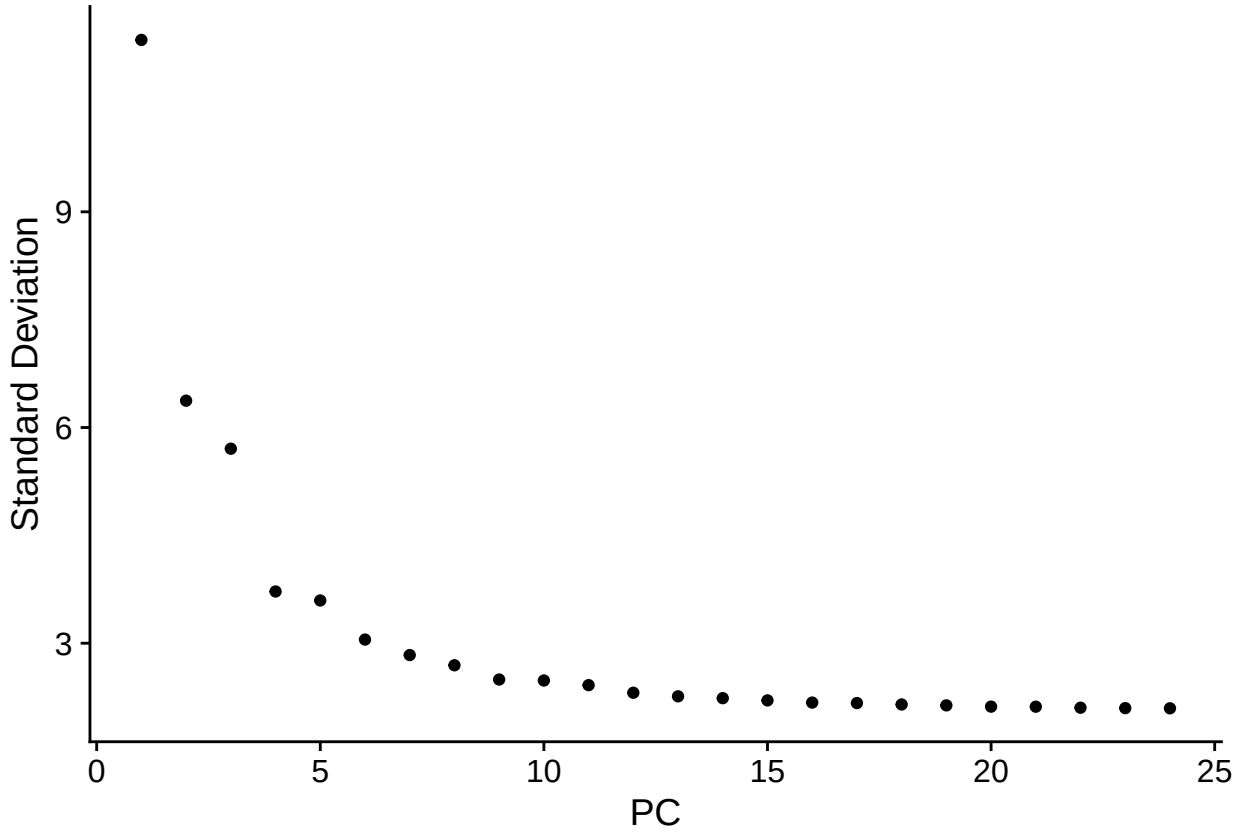
```
Centering and scaling data matrix
```

4. Run the PCA. `npcs = 50` is the default. You may want to increase for more complex data sets. Also set "`verbose=F`" otherwise Seurat will print out a number of statistics to the console.

```
sc_subset <- RunPCA(sc_subset, features=VariableFeatures(object=sc_subset),
                    npcs = 50, verbose=F)
```

5. Generate the elbow (scree) plot of the PCA results. For this example we will only plot the top 24 PCs.

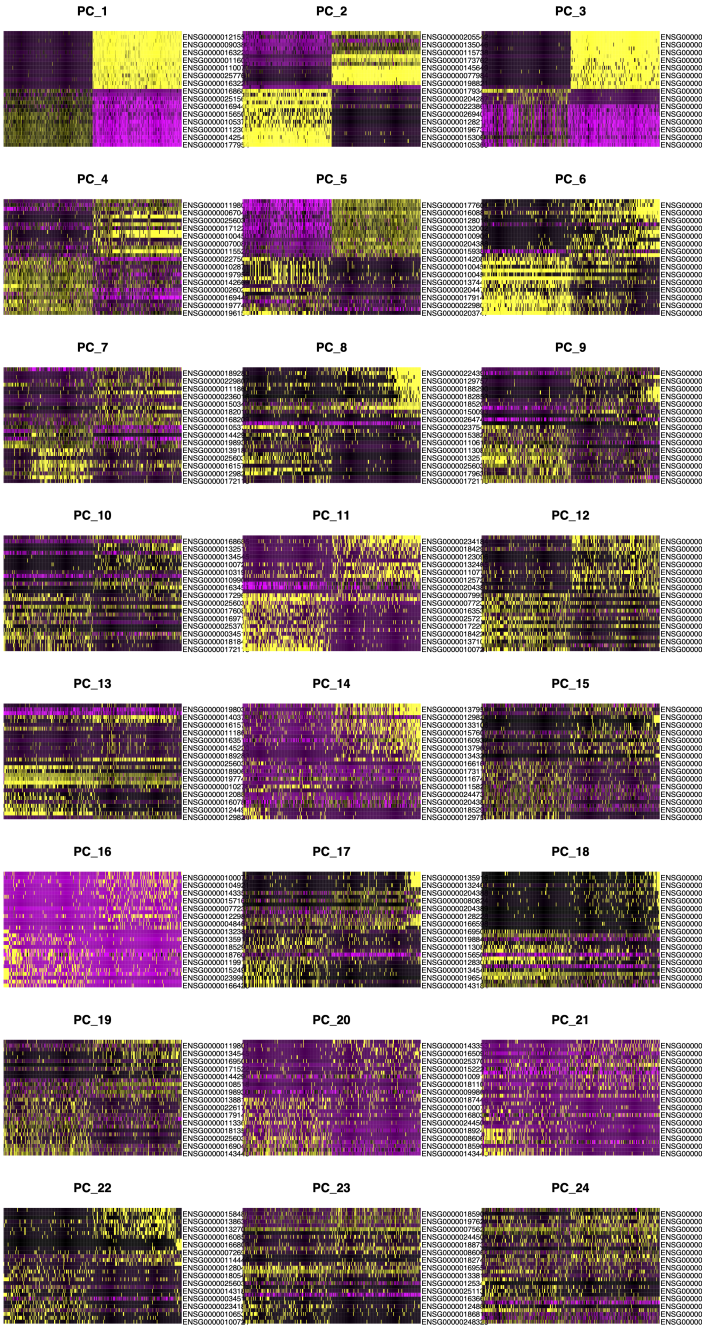
```
ElbowPlot(sc_subset, ndims=24)
```



6. Make a heatmap of signals from the top 24 PCs. To decide how many we want to keep we're plotting the top 300 cells based on their weightings for each PC, with an even mix of positive and negative associations (`balanced=T`). We want to examine each PC's heatmap to see at what point the "signal" starts to go away.

**NOTE**: If you get warnings about figure margins too large, try making the plot pane in your R studio bigger

```
DimHeatmap(sc_subset, dims=1:24, cells=300, balanced=T)
```



### 1.4.1 JackStraw (*AT HOME EXERCISE*)

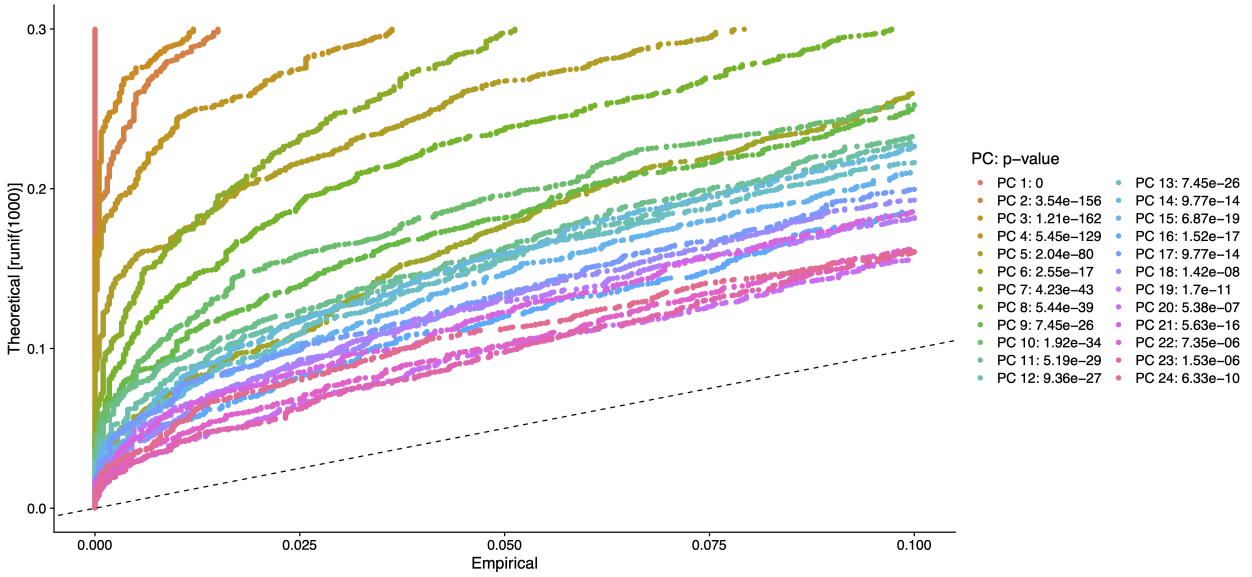
**DO NOT RUN THESE STEPS TODAY** as they are computationally demanding (this exercise took ~9 minutes on my computer). We will review the commands and results in the exercise handout.

JackStraw is a method for computing p-values for principal components. As implemented in Seurat, the steps are:

- `JackStraw()` computes projected PCA scores for a random permutation of a subset of the data, and compares these values to the observed PCA scores for the genes. This allows the computation of a p-value for the association of each gene with each PC.
- `ScoreJackStraw()` computes p-values for the PCs based on the distribution of p-values obtained for all genes against that PC from the JackStraw step.
- `JackStrawPlot()` makes a diagnostic plot we can use to evaluate the signal level and significance of each PC.

**NOTE:** The selection of the number of PCs to use (dims parameter) is different for the different functions. For JackStraw() you just give a number (e.g., 24), and for the other two you give a range (e.g., 1:24).

```
sc_subset = JackStraw(sc_subset, num.replicate = 100, dims = 24)
sc_subset = ScoreJackStraw(sc_subset, dims = 1:24)
JackStrawPlot(sc_subset, dims = 1:24)
```




These are Q-Q plots: PC curves that are more diverged from the black dashed diagonal line have a higher level of signal. P-values are given in the legend.

- You don't necessarily want to set a strict p<0.05 threshold, as this may still retain a lot of PCs with minimal (but statistically significant) levels of signal.
  - A PC can be significant but still capture variability primarily from a tiny fraction of cells, and therefore be more likely to be noise.
- What's more helpful is to compare the exponent magnitude of the PCs as you go down the list.
  - PCs 1-13 all have p-values <1e-20.
  - The last p-value <1e-10 is PC 21.

## 1.5 Clustering

### 1.5.1 Run clustering on the top PCs to filter out noise

- Based on the PCA visualizations, we'll take the top 20 PCs, and cluster on those data.
- Then visualizations and differential expression across clusters.

1. First run `FindNeighbors()`. For this exercise we will use the first 20 PCs to determine the distance between the cells and thus who are the "neighbors"

```
pca.dims = 1:20
sc_subset <- FindNeighbors(sc_subset, dims=pca.dims)
```

```
Computing nearest neighbor graph
```

```
Computing SNN
```

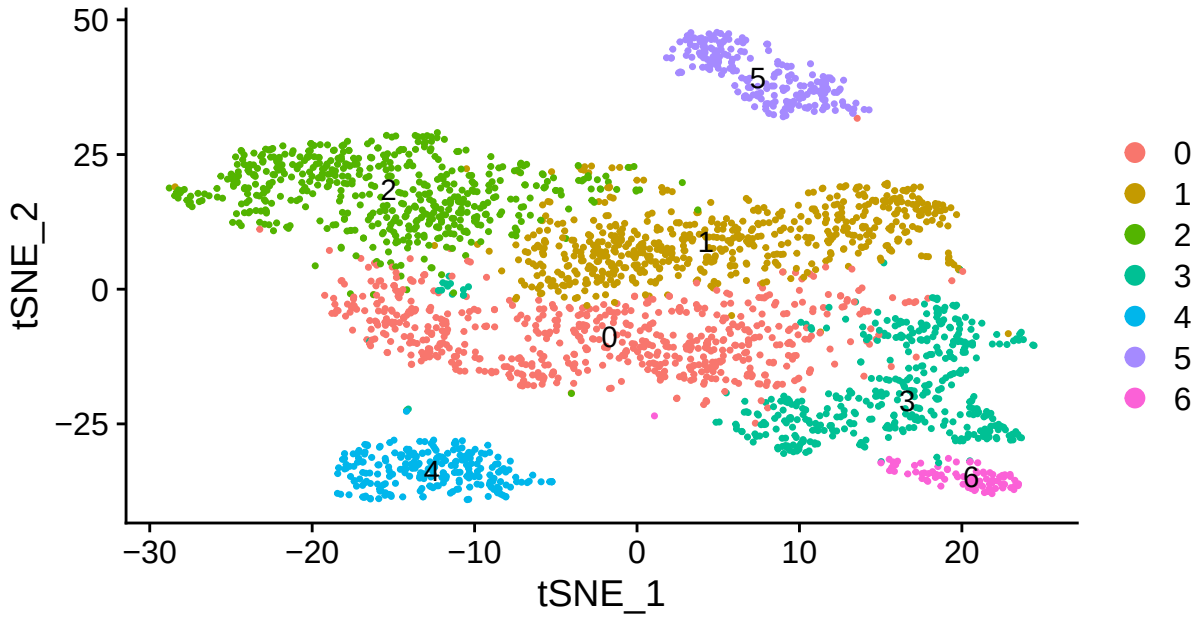
2. Find cell clusters with a resolution 0.5. The larger the number the finer the resolution. Optimal cluster resolution will likely have too be determined empirically.

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

```
Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck

Number of nodes: 2791
Number of edges: 110384

Running Louvain algorithm...
Maximum modularity in 10 random starts: 0.8599
Number of communities: 7
Elapsed time: 0 seconds
```

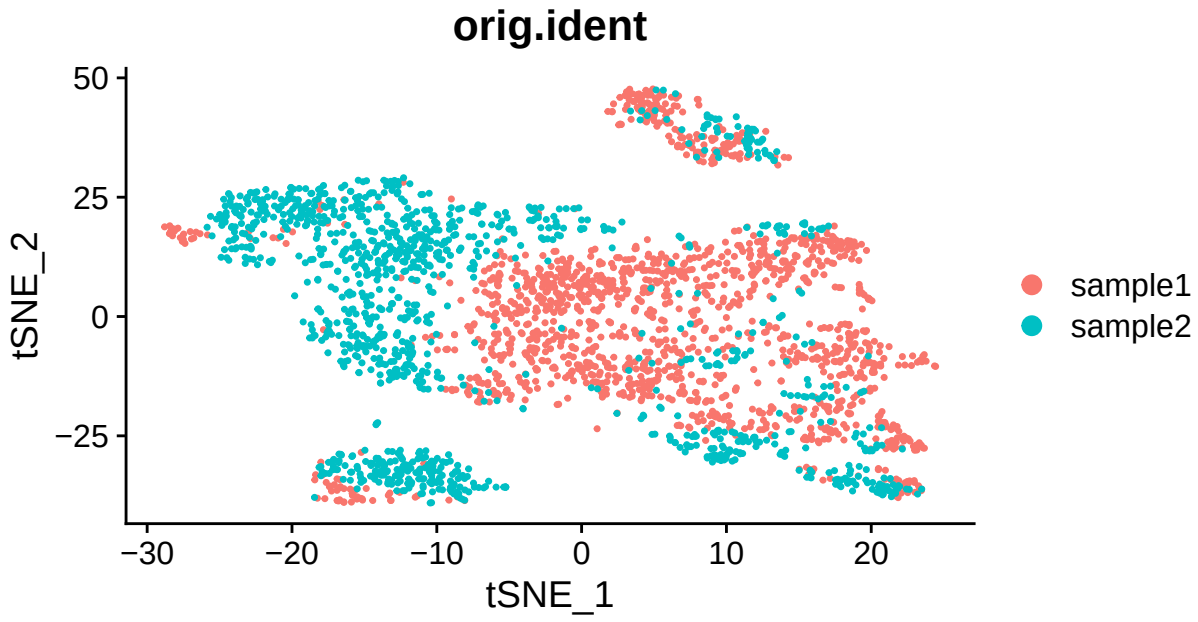
3. Generate tSNE plot. *NOTE* you can look at `?RunTSNE` for how to change perplexity.

```
sc_subset <- RunTSNE(sc_subset, dims=pca.dims)
DimPlot(sc_subset, reduction='tsne', label=T)
```




4. We can also make the tSNE plot and color based on original identity, i.e. sample.

```
DimPlot(sc_subset, reduction='tsne', group.by="orig.ident")
```




### 1.5.2 Differential expression with respect to clusters

- This calculation may take a few minutes, so start it running (it took ~1 minute on my computer).

- **We'll return to the lecture while it goes.** The ROC test is usually faster than Wilcox.

```
roc_stats <- FindAllMarkers(sc_subset, test.use="roc")
```

```
Calculating cluster 0
Calculating cluster 1
Calculating cluster 2
Calculating cluster 3
Calculating cluster 4
Calculating cluster 5
Calculating cluster 6
```

#### 1.5.3 Save the object to an R data file so we can use it later

- Save our Seurat object in case we want to close R

```
saveRDS(sc_subset, file="sc_subset.rds")
```

#### 1.5.4 Other ways to run differential expression (*AT HOME EXERCISE*):

- You can use the function **FindMarkers** (instead of `FindAllMarkers`) to test specific subsets of clusters.
- You can use the **group.by** parameter to test based on, e.g., `orig.ident` instead of cluster

1. If you wanted to test between a specific pair of clusters.

```
cluster1vs2 <- FindMarkers(sc_subset, ident.1 = 1, ident.2 = 2)
```

2. If you wanted to test between one cluster vs the combination of two other clusters.

```
cluster1vs2and3 <- FindMarkers(sc_subset, ident.1 = 1, ident.2 = c(2,3))
```

3. If you wanted to run stats with Wilcox instead of ROC

```
wilcox_stats <- FindAllMarkers(sc_subset,
                               test.use="wilcox",
                               group.by="seurat_clusters")
```

4. If you wanted to test between samples instead of clusters, you need to set the "Idents" for the cells first the value you give, such as "orig.ident", needs to match a column in the `sc_subset@meta.data` data frame.

```
Idents(sc_subset) = "orig.ident"
sample_stats <- FindAllMarkers(sc_subset, test.use="wilcox")
```

# 2 Afternoon

*If you closed R, then reload the* ***sc_subset*** *Seurat object. Otherwise skip this step.*

```
library(Seurat)
sc_subset <- readRDS("sc_subset.rds")
```

*If you closed R and forgot to save your Seurat object, you can read it from our server. Otherwise skip this step.* ***NOTE*** *Use the* ***url()*** *function when reading an RDS from the web.*

```
library(Seurat)
sc_subset <- readRDS(url("https://wd.cri.uic.edu/sc_rna/sc_subset.rds"))
```

## 2.1 More explorations of clustering results in Seurat

*If your differential stats calculation didn't finish, you can read it from our server for the next steps. If it did finish, skip this and the following command.*

```
roc_stats <- read.table("https://wd.cri.uic.edu/sc_rna/diff_stats.txt",
   header=T, row.names=1, sep="\t", stringsAsFactors=F)
```

*Save the stats from last step into a text file.*

```
write.table(roc_stats, file="diff_stats.txt", col.names=NA, quote=F, sep="\t")
```

### 2.1.1 Cluster abundance

- The table of clusters vs. samples can be used to compare the frequency of different cell sub-populations across samples.
- **To test this rigorously, you need to collect replicate samples to measure variability.** You could then compare counts per sample for each cluster using edgeR.

1. Compute the size of the clusters

```
table(sc_subset$seurat_clusters)
```

```
  0   1   2   3   4   5   6
641 601 549 429 243 233  95
```

2. Compute the number of samples in each cluster. We will provide two separate vectors (`seurat_clusters` is the cluster ID and `orig.ident` is the sample ID for each cell). The `dnn` parameter will allow us to provide a nicer name for the output from `table()`.

```
cluster_abundance <- table(sc_subset$seurat_clusters,
                           sc_subset$orig.ident,
                           dnn=c("cluster", "group"))
cluster_abundance
```

```
       group
cluster sample1 sample2
      0     410     231
      1     536      65
      2      39     510
      3     286     143
      4      49     194
      5     169      64
      6      34      61
```

3. See the counts as a percent of total. **NOTE** R does operations column-wise, so we need to transpose to divide correctly

```
cluster_percent <- t(t(cluster_abundance)/colSums(cluster_abundance))
cluster_percent
```

```
       group
cluster    sample1    sample2
      0 0.26920552 0.18217666
      1 0.35193697 0.05126183
      2 0.02560735 0.40220820
      3 0.18778726 0.11277603
      4 0.03217334 0.15299685
      5 0.11096520 0.05047319
      6 0.02232436 0.04810726
```

### 2.1.2 Top genes per cluster

1. Compute the number of genes with `AUC > 0.7` for each cluster.

```
head(roc_stats)
```

```
                myAUC   avg_diff power  avg_logFC pct.1 pct.2 cluster
ENSG00000227507 0.735  0.5847424 0.470  0.5847424 0.986 0.817       0
ENSG00000008517 0.734  0.5826307 0.468  0.5826307 0.959 0.704       0
ENSG00000160789 0.712  1.0074135 0.424  1.0074135 0.528 0.120       0
ENSG00000168685 0.702  0.5610427 0.404  0.5610427 0.863 0.551       0
ENSG00000197728 0.786  0.4001011 0.572  0.4001011 1.000 0.997       1
ENSG00000234745 0.222 -0.4425595 0.556 -0.4425595 1.000 1.000       1
                           gene
ENSG00000227507 ENSG00000227507
ENSG00000008517 ENSG00000008517
ENSG00000160789 ENSG00000160789
ENSG00000168685 ENSG00000168685
ENSG00000197728 ENSG00000197728
ENSG00000234745 ENSG00000234745
```

```
table(roc_stats[roc_stats$myAUC>0.7, "cluster"])
```

```
  0   1   2   3   4   5   6
  4   7  14  27 258  44   7
```

2. Get top genes per cluster. For this example we will get the top 5 genes for cluster 5.

```
cluster5_stats <- subset(roc_stats, cluster == 5)
top5_cluster5 <- head(cluster5_stats[order(cluster5_stats[,1], decreasing=T),], 5)
top5_cluster5
```

```
                  myAUC avg_diff power avg_logFC pct.1 pct.2 cluster
ENSG00000019582.1 0.996 2.872648 0.992  2.872648 1.000 0.534       5
ENSG00000105369   0.993 2.685255 0.986  2.685255 0.987 0.032       5
ENSG00000223865.1 0.981 2.310003 0.962  2.310003 0.996 0.253       5
ENSG00000007312   0.978 2.250470 0.956  2.250470 0.970 0.089       5
ENSG00000204287.1 0.972 2.426704 0.944  2.426704 0.996 0.199       5
                             gene
ENSG00000019582.1 ENSG00000019582
ENSG00000105369   ENSG00000105369
```

```
ENSG00000223865.1 ENSG00000223865
ENSG00000007312   ENSG00000007312
ENSG00000204287.1 ENSG00000204287
```

3. Alternative way to do it with the `dplyr` package. This example will get the top 10 genes for all clusters. The `group_by()` function will group the data.frame rows by the `cluster` column and the `top_n()` function will return the top n (10) rows ranked by the column `myAUC`. **NOTE** The `%>%` is a "pipe" function that pass the output from the each function to the next function.

```
library(dplyr)
top10 <- roc_stats %>% group_by(cluster) %>% top_n(n=10, wt=myAUC)
top10
```

```
# A tibble: 64 x 8
# Groups:   cluster [7]
   myAUC avg_diff power avg_logFC pct.1 pct.2 cluster gene
   <dbl>    <dbl> <dbl>     <dbl> <dbl> <dbl>   <int> <chr>
 1 0.735    0.585 0.47      0.585 0.986 0.817       0 ENSG00000227507
 2 0.734    0.583 0.468     0.583 0.959 0.704       0 ENSG00000008517
 3 0.712    1.01  0.424     1.01  0.528 0.12        0 ENSG00000160789
 4 0.702    0.561 0.404     0.561 0.863 0.551       0 ENSG00000168685
 5 0.786    0.400 0.572     0.400 1     0.997       1 ENSG00000197728
 6 0.776    0.314 0.552     0.314 1     1           1 ENSG00000071082
 7 0.764    0.274 0.528     0.274 1     1           1 ENSG00000164587
 8 0.247   -1.36  0.506    -1.36  0.615 0.826       1 ENSG00000196154
 9 0.751    0.263 0.502     0.263 1     1           1 ENSG00000145425
10 0.748    0.305 0.496     0.305 1     0.999       1 ENSG00000198034
# ... with 54 more rows
```

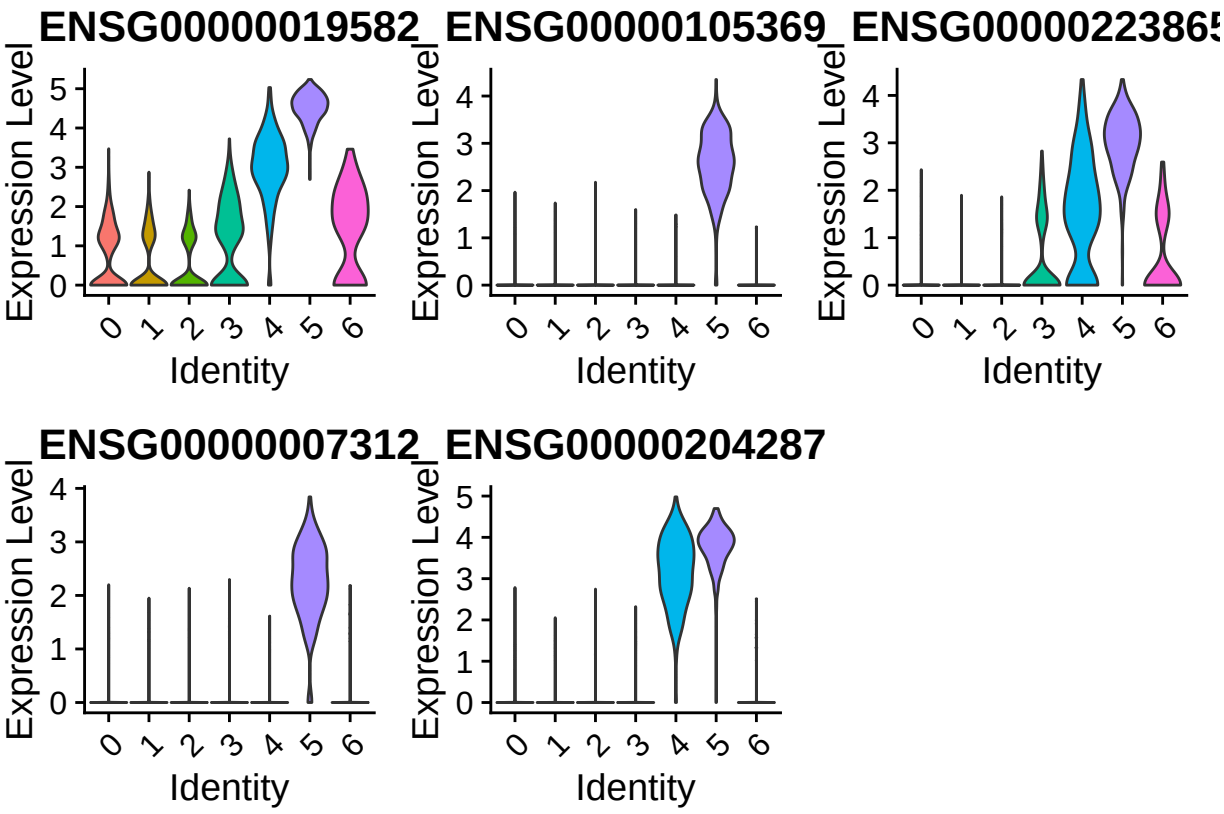
4. Look at the top 10 from cluster 5. Can also look at the R object for `top10` in R Studio.

```
head(subset(top10, cluster==5))
```

```
# A tibble: 6 x 8
# Groups:   cluster [1]
  myAUC avg_diff power avg_logFC pct.1 pct.2 cluster gene
  <dbl>    <dbl> <dbl>     <dbl> <dbl> <dbl>   <int> <chr>
1 0.996     2.87 0.992      2.87 1     0.534       5 ENSG00000019582
2 0.993     2.69 0.986      2.69 0.987 0.032       5 ENSG00000105369
3 0.981     2.31 0.962      2.31 0.996 0.253       5 ENSG00000223865
4 0.978     2.25 0.956      2.25 0.97  0.089       5 ENSG00000007312
5 0.972     2.43 0.944      2.43 0.996 0.199       5 ENSG00000204287
6 0.958     2.30 0.916      2.30 0.923 0.026       5 ENSG00000156738
```

- Visualizations for top genes in cluster 5 in side-by-side violin plot

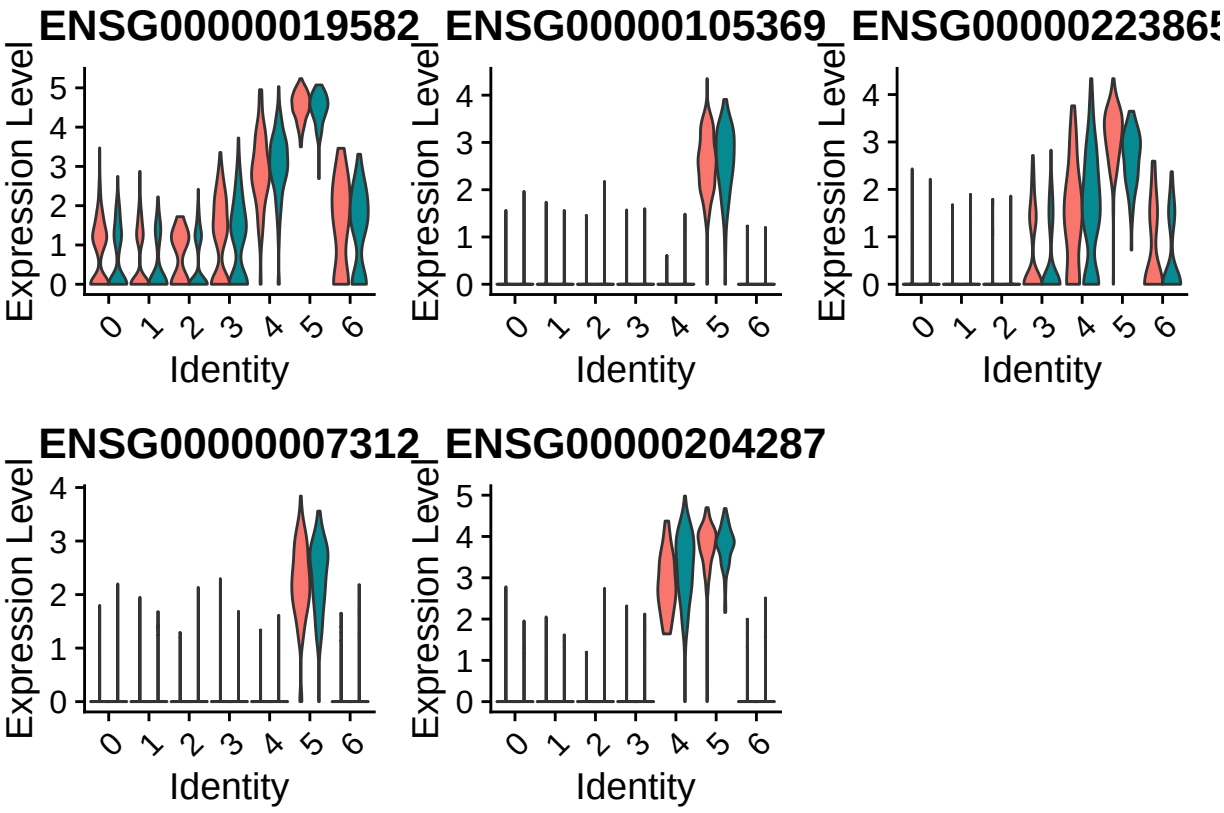
```
VlnPlot(sc_subset, features=top5_cluster5$gene, pt.size=F)
```



- Violin plot of top 5 genes for cluster 5 comparing sample1/sample2 differences

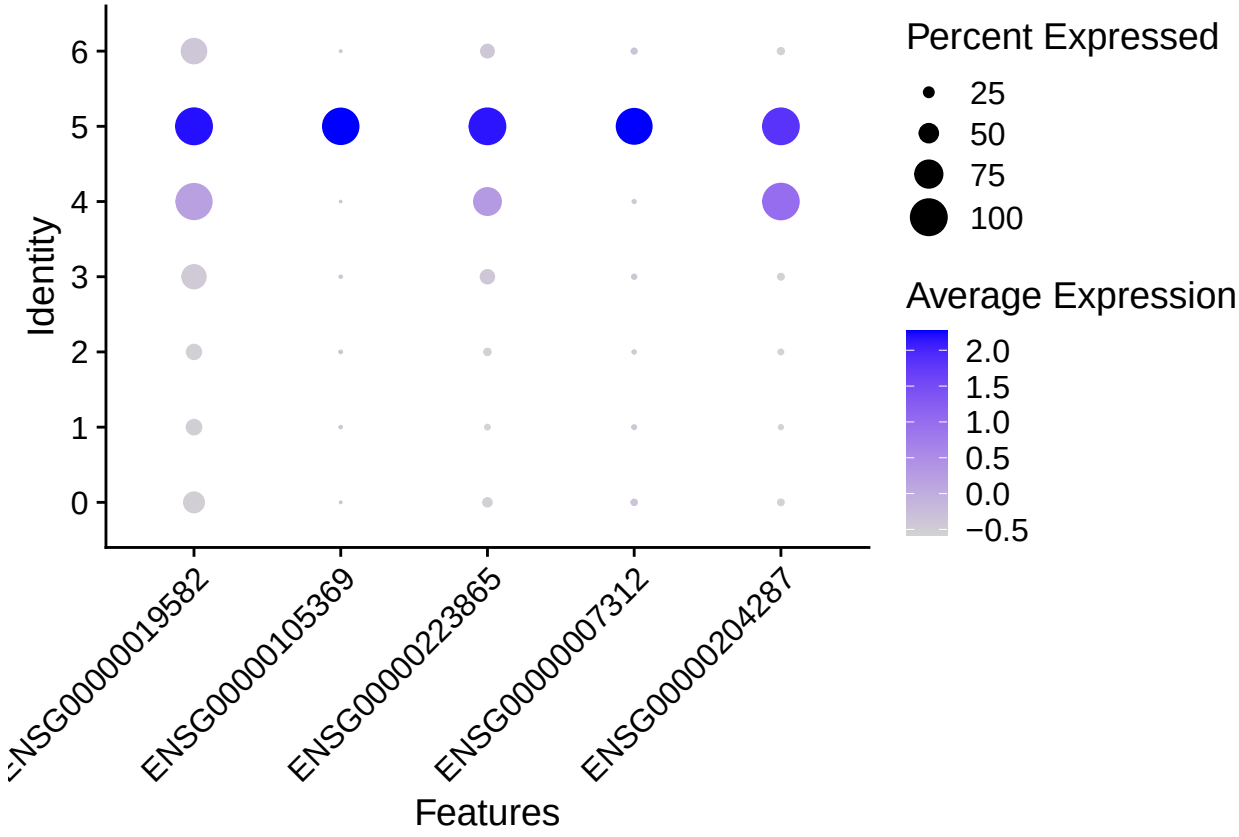
```
VlnPlot(sc_subset, features=top5_cluster5$gene, split.by = "orig.ident", pt.size=F)
```

```
The default behaviour of split.by has changed.
Separate violin plots are now plotted side-by-side.
To restore the old behaviour of a single split violin,
set split.plot = TRUE.

This message will be shown once per session.
```



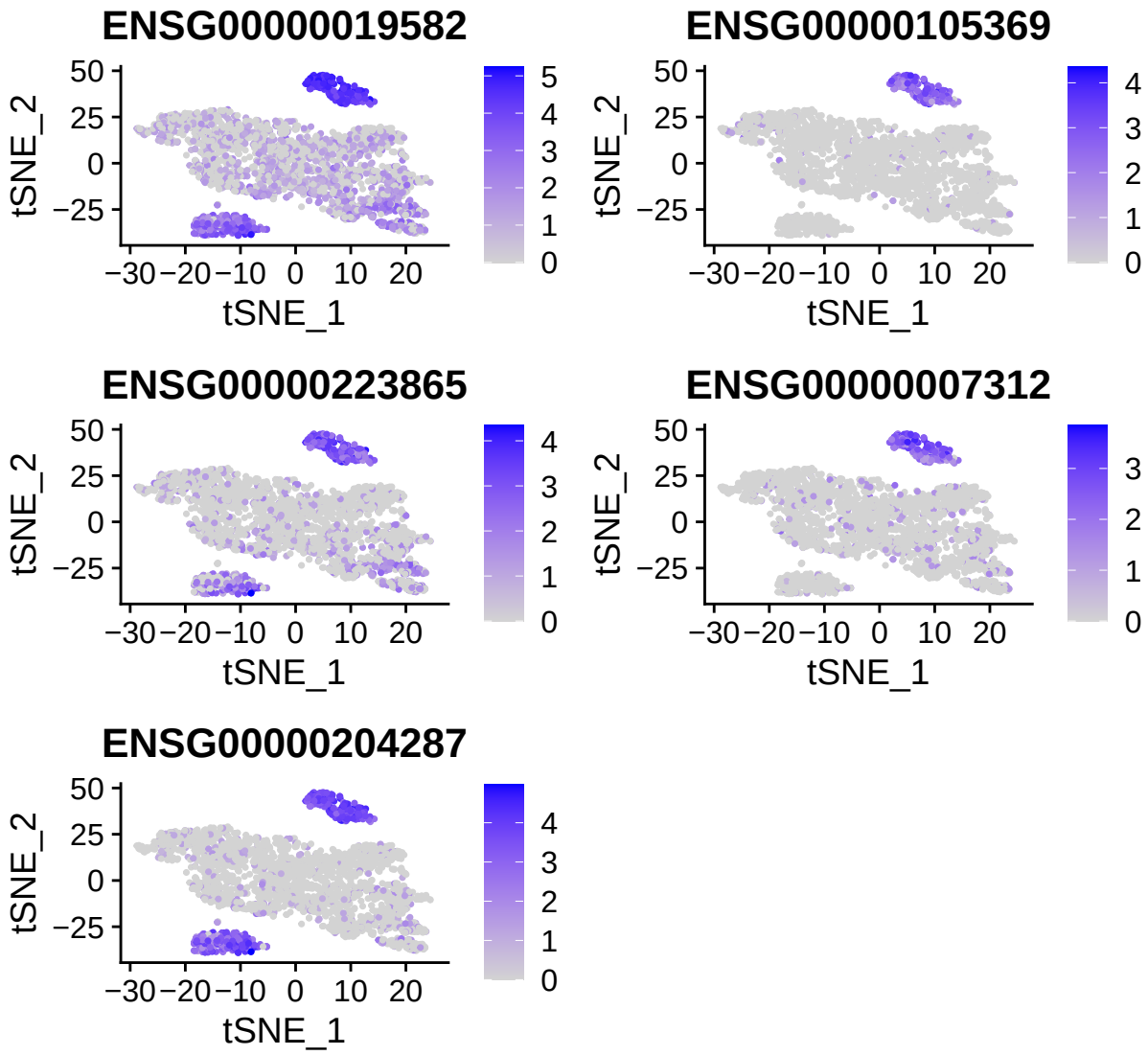
- Dotplot of top 5 genes for cluster 5

```
DotPlot(sc_subset,features=top5_cluster5$gene) + RotatedAxis()
```



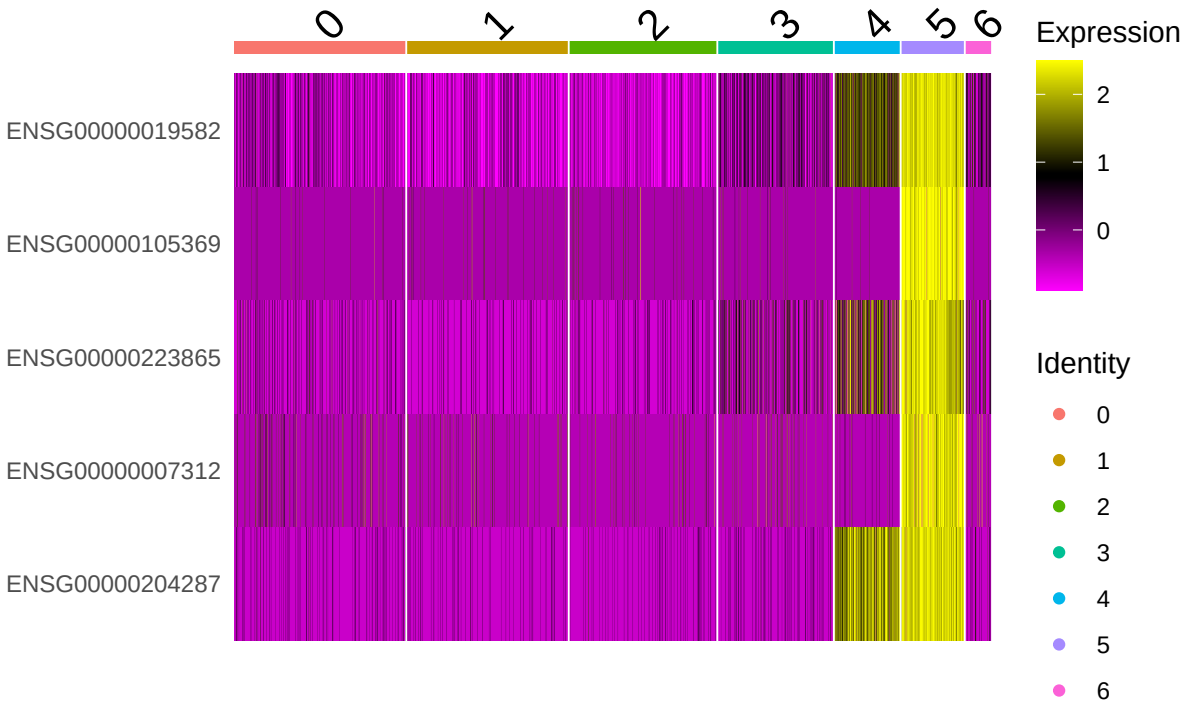
- tSNE plot colored by expression levels of top 5 genes for cluster 5

```
FeaturePlot(sc_subset, features=top5_cluster5$gene, reduction='tsne')
```



- Heatmap of top 5 genes for cluster 5.

```
DoHeatmap(sc_subset, features=top5_cluster5$gene)
```

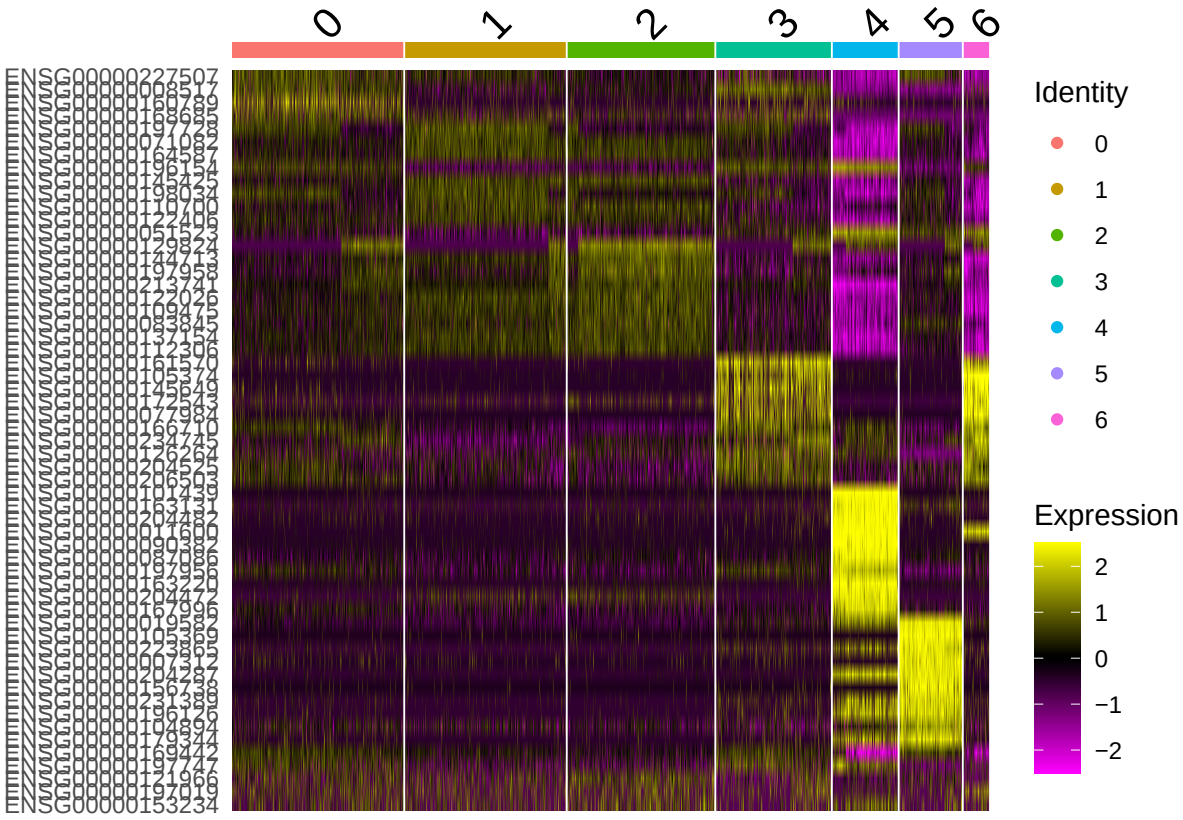


- Heatmap of top genes for all clusters. **NOTE**! By default, Seurat will only scale the variable features identified by `FindVariableFeatures()`. We will rescale the object to make sure all the features to be plotted are scaled.

```
sc_top10 <- ScaleData(sc_subset, features=top10$gene)
```

```
Centering and scaling data matrix
```

```
DoHeatmap(sc_top10, features=top10$gene)
```

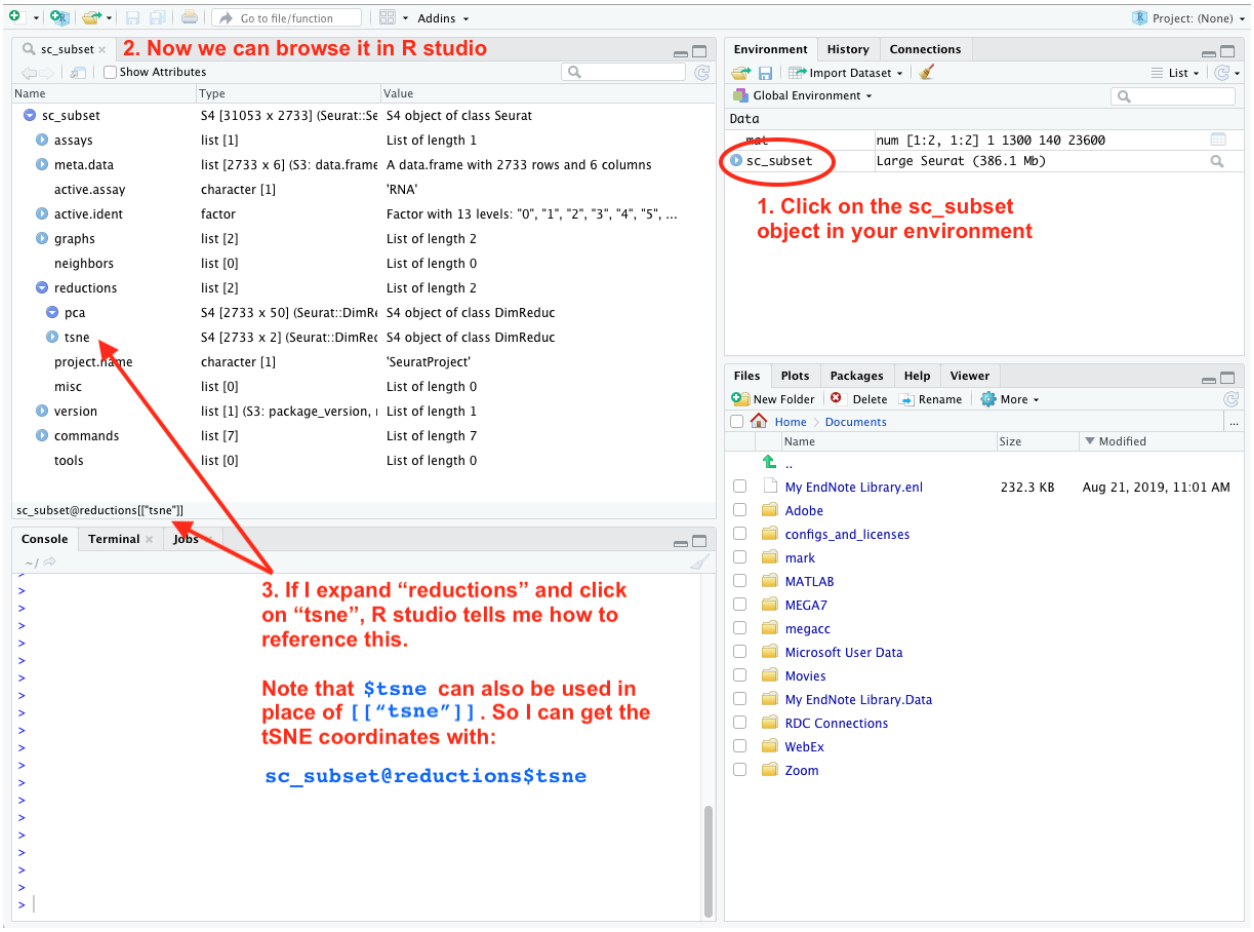


## 2.2 Exploring the Seurat object

### 2.2.1 Let's take a look at some of the parts of the Seurat package

Browse the Seurat R object through R Studio

- Find the "sc_subset" object in your enviroment and click on it.
- Browse through the object in R studio to see the structure.
- Notice how R studio will tell you how to reference the parts of the object if you click on them.




### 2.2.2 Find vectors/dataframes for different results from the Seurat object

- Get the original sample IDs for each "cell"

```
head(sc_subset@meta.data$orig.ident)
```

```
[1] "sample1" "sample1" "sample1" "sample1" "sample1" "sample1"
```

- Get the current cluster assignments - 2 different ways

```
head(sc_subset@meta.data$seurat_clusters)
```

```
[1] 5 0 1 3 0 3
Levels: 0 1 2 3 4 5 6
```

```
head(sc_subset@meta.data$RNA_snn_res.0.5)
```

```
[1] 5 0 1 3 0 3
Levels: 0 1 2 3 4 5 6
```

- Get the normalized, scaled expression, first 5 rows and columns

```
sc_subset@assays$RNA@scale.data[1:5,1:5]
```

```
                s1_AAACCTGAGCACCGTC-1 s1_AAACCTGGTAGAGCTG-1
ENSG00000228327           -0.09095396           -0.09095396
ENSG00000188290           -0.14042283           -0.14042283
ENSG00000187608           -0.57919340           -0.57919340
ENSG00000237330           -0.01892867           -0.01892867
ENSG00000272141           -0.01892867           -0.01892867
                s1_AAACCTGGTCTAGCGC-1 s1_AAACCTGGTGTGGCTC-1
ENSG00000228327           -0.09095396           -0.09095396
ENSG00000188290           -0.14042283           -0.14042283
ENSG00000187608            0.80624558           -0.57919340
ENSG00000237330           -0.01892867           -0.01892867
ENSG00000272141           -0.01892867           -0.01892867
                s1_AAACGGGGTGGTACAG-1
ENSG00000228327           -0.09095396
ENSG00000188290           -0.14042283
ENSG00000187608            2.85856629
ENSG00000237330           -0.01892867
ENSG00000272141           -0.01892867
```

- List the "Variable" genes, as identified by `FindVariableFeatures()`

```
head(sc_subset@assays$RNA@var.features)
```

```
[1] "ENSG00000163220" "ENSG00000143546" "ENSG00000090382" "ENSG00000115523"
[5] "ENSG00000169429" "ENSG00000254709"
```

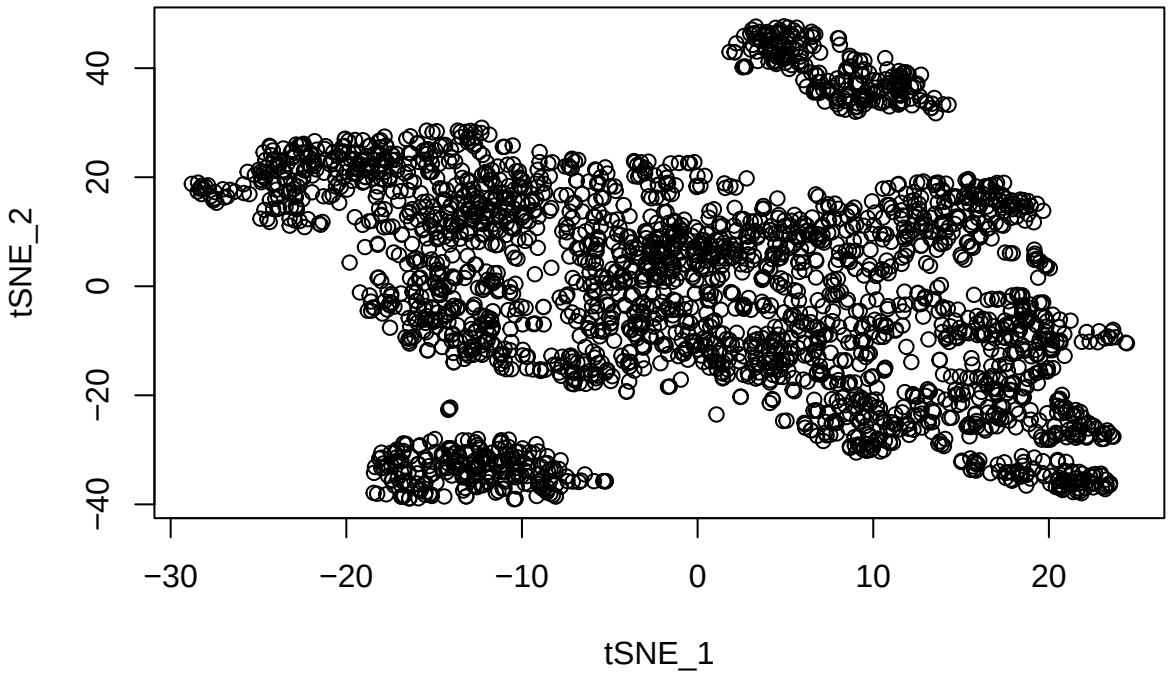
- Get the tSNE coordinates for each cell

```
head(sc_subset@reductions$tsne@cell.embeddings)
```

```
                        tSNE_1     tSNE_2
s1_AAACCTGAGCACCGTC-1  7.948768  38.126300
s1_AAACCTGGTAGAGCTG-1  9.917789  -8.467060
s1_AAACCTGGTCTAGCGC-1 13.015861   4.240164
s1_AAACCTGGTGTGGCTC-1 20.655254 -10.385203
s1_AAACGGGGTGGTACAG-1  7.958638 -17.873703
s1_AAACGGGTCACCCGAG-1 12.736358 -23.365006
```

- Try making our own tSNE plot. Not as nice as Seurat's (which uses ggplot2) but, just to demonstrate that these are the right coordinates

```
plot(sc_subset@reductions$tsne@cell.embeddings)
```




### 2.2.3 Get the marker list for cluster 3, including gene symbols

*If you closed R, read your roc_stats table back in, either from your computer (if you saved it earlier), or from our server.*

- From your computer

```
roc_stats <- read.table("diff_stats.txt",header=T,row.names=1,sep="\t",stringsAsFactors=F)
```

- From our workshop data on GitHub.

```
roc_stats <- read.table("https://wd.cri.uic.edu/sc_rna/diff_stats.txt",
   header=T,row.names=1,sep="\t",stringsAsFactors=F)
```

Next we'll read in the list of gene ID & gene symbol from one of the original CellRanger feature files.

- Note that this file is gzipped, but R's read.table can still read it.
- Then we can combine data frames in R.

1. Read in gene lists.

```
names <- read.table("~/Downloads/V035_F031_subsample/features.tsv.gz",
   sep="\t",row.names=1)
```

2. Take a peek at the gene lists.

```
dim(names)
```

```
[1] 32738     1
```

```
head(names)
```

```
                        V2
ENSG00000243485   MIR1302-10
ENSG00000237613      FAM138A
ENSG00000186092        OR4F5
ENSG00000238009 RP11-34P13.7
ENSG00000239945 RP11-34P13.8
ENSG00000237683   AL627309.1
```

3. Get the differentially expressed genes for cluster 3 with `AUC > 0.8`

```
cluster3_markers <- subset(roc_stats, cluster == 3 & myAUC > 0.8)[,"gene"]
length(cluster3_markers)
```

```
[1] 10
```

```
cluster3_markers
```

```
 [1] "ENSG00000161570" "ENSG00000105374" "ENSG00000145649" "ENSG00000172543"
 [5] "ENSG00000077984" "ENSG00000166710" "ENSG00000234745" "ENSG00000126264"
 [9] "ENSG00000204525" "ENSG00000206503"
```

4. Get the corresponding gene symbols from the names data frame

```
cluster3_symbols <- names[cluster3_markers, 1]
cluster3_symbols
```

```
 [1] "CCL5"  "NKG7"  "GZMA"  "CTSW"  "CST7"  "B2M"   "HLA-B" "HCST"  "HLA-C"
[10] "HLA-A"
```

## 2.3 Cluster comparisons

### 2.3.1 First, let's rerun clustering with two other resolutions

We can use the same object, as each resolution will get saved into its own slot.

```
sc_subset <- FindClusters(sc_subset, resolution=0.1)
```

```
Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck

Number of nodes: 2791
Number of edges: 110384

Running Louvain algorithm...
Maximum modularity in 10 random starts: 0.9439
Number of communities: 4
Elapsed time: 0 seconds
```

```
sc_subset <- FindClusters(sc_subset, resolution=1)
```

```
Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck

Number of nodes: 2791
Number of edges: 110384

Running Louvain algorithm...
Maximum modularity in 10 random starts: 0.8027
Number of communities: 12
Elapsed time: 0 seconds
```

- Check how many clusters we got for each one. Remember that higher resolution would mean more clusters

```
table(sc_subset@meta.data$RNA_snn_res.0.1)
```

```
   0    1    2    3
1165 1150  243  233
```

```
table(sc_subset@meta.data$RNA_snn_res.0.5)
```

```
  0   1   2   3   4   5   6
641 601 549 429 243 233  95
```

```
table(sc_subset@meta.data$RNA_snn_res.1)
```

```
  0   1   2   3   4   5   6   7   8   9  10  11
413 380 294 283 251 243 229 227 143 124 109  95
```

### 2.3.2 Compare the similarity of the clustering results at a high level with the adjusted Rand index

- The fossil package was written with archaeological data sets in mind, but has a useful implementation of the rand and adjusted rand indices

1. Load the `fossil` library

```
library(fossil)
```

2. Make separate vectors from the different clustering results for convenience

```
clust0.1 <- sc_subset@meta.data$RNA_snn_res.0.1
clust0.5 <- sc_subset@meta.data$RNA_snn_res.0.5
clust1 <- sc_subset@meta.data$RNA_snn_res.1
```

3. Compute the adjusted rand index for each. Not surprisingly, resolutions 0.1 and 1 are the least similar

```
adj.rand.index(clust0.1, clust0.5)
```

```
[1] 0.6428863
```

```
adj.rand.index(clust0.1, clust1)
```

```
[1] 0.4831814
```

```
adj.rand.index(clust0.5, clust1)
```

```
[1] 0.7179299
```

**Exercise for home, skip for today**

*NOTE: If you ended up with a large number of clustering results and you wanted to compute pair-wise similarities across all of them using the adjusted rand index, you would approach this by setting up a data frame with all of the clustering results in it across the columns, then running a loop over all pairs of columns.:*

```
# building a data frame for our 3 clustering results
cluster.df <- data.frame(clust0.1, clust0.5, clust1)
# define a function to make a similarity matrix
cluster_similarity <- function( clust_df ){
   # number of columns we're comparing
   columns <- ncol(clust_df)
   # set up a similarity matrix
   rand.sim <- matrix( nrow=columns, ncol=columns )
   # set all values to be 1 first
   # this will keep the diagonal entries 1 after we do the other calculations
   rand.sim[,] <- 1
   # loop over all pairs of columns
   for( i in 1:(columns-1) ){
      for( j in (i+1):columns ){
         # compute the similarity and store it at positions i,j and j,i
         sim <- adj.rand.index( clust_df[,i], clust_df[,j] )
         rand.sim[i,j] <- sim
         rand.sim[j,i] <- sim
      }
   }
   return(rand.sim)
}
# run the function
cluster.sim <- cluster_similarity(cluster.df)
```

### 2.3.3 Detailed comparison between clustering at resolutions 0.25 and 1 using the overlap index

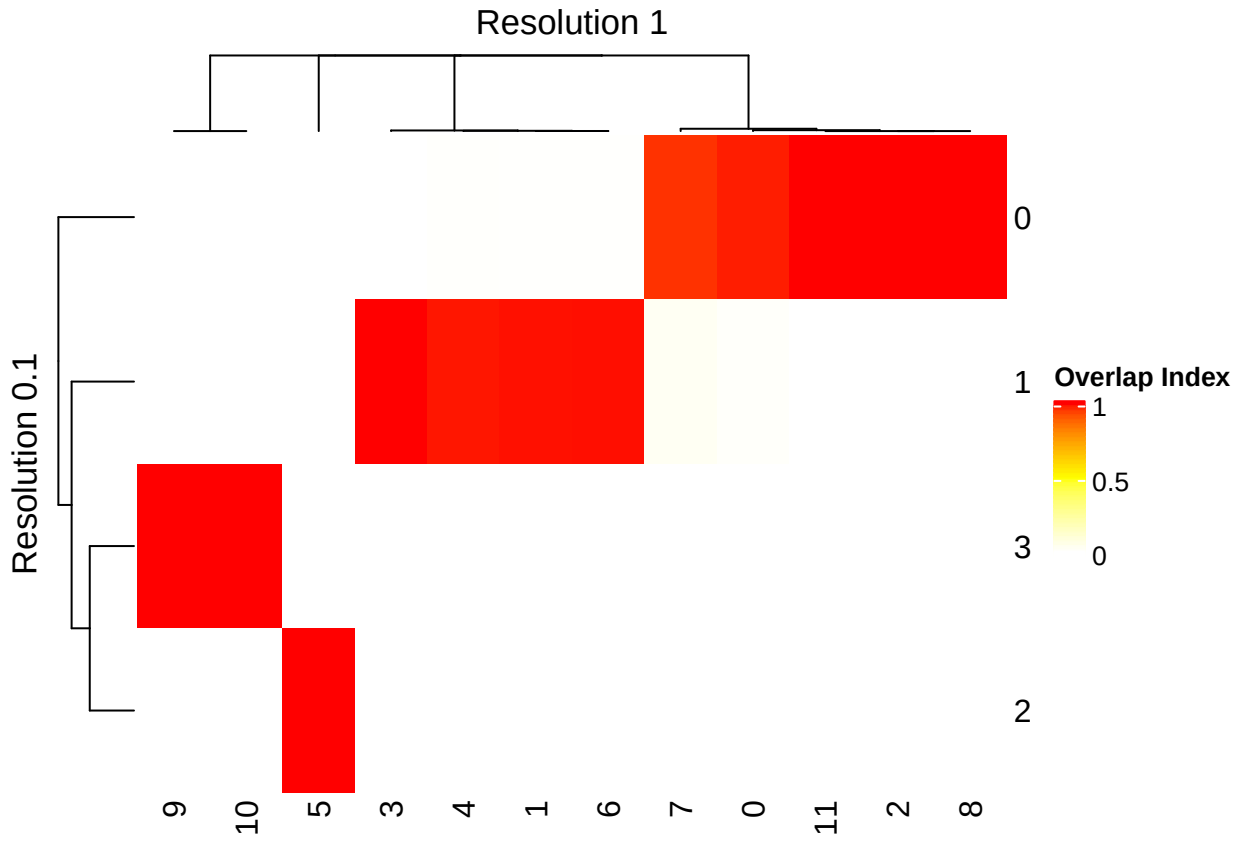
- The goal is to compare the actual sets of cells in each clustering result, so that we can see which clusters between the results are related to each other
- We'll write a function to do this so that we can re-use it in another analysis if wanted
- In the function, we need to write a double loop over both sets of clusters. You don't need to type the comment lines (#), but they are there to explain our steps along the way.

```
# function for overlap index
overlap_index <- function( cluster1, cluster2 ){
   # make factor vectors from clusters
   c1 = as.factor(cluster1)
   c2 = as.factor(cluster2)
   # define a set of "names" for our cells, which we'll compare between clusters
   # the names are arbitraty, so we'll just number them
   names = 1:length(c1)
   # make distance matrix to store our calculations in
   my_dist = matrix(nrow=length(levels(c1)),ncol=length(levels(c2)))
   rownames(my_dist) = levels(c1)
   colnames(my_dist) = levels(c2)
   for(i in 1:length(levels(c1))){
      for(j in 1:length(levels(c2))){
         # get the list of cells in each cluster
         c1.sub = names[c1==levels(c1)[i]]
         c2.sub = names[c2==levels(c2)[j]]
         # get the intersection and min cluster size
         int = length(intersect(c1.sub,c2.sub))
         minsize = min(length(c1.sub),length(c2.sub))
         # overlap index
         overlap = int/minsize
         # store in matrix
         my_dist[i,j] = overlap
      }
   }
   return(my_dist)
}
# now run the function
res_0.1vs1 <- overlap_index(clust0.1, clust1)
res_0.1vs1
```

```
           0           1 2 3           4 5           6        7 8 9 10 11
0 0.98789346 0.005263158 1 0 0.007968127 0 0.004366812 0.969163 1 0  0  1
1 0.01210654 0.994736842 0 1 0.992031873 0 0.995633188 0.030837 0 0  0  0
2 0.00000000 0.000000000 0 0 0.000000000 1 0.000000000 0.000000 0 0  0  0
3 0.00000000 0.000000000 0 0 0.000000000 0 0.000000000 0.000000 0 1  1  0
```

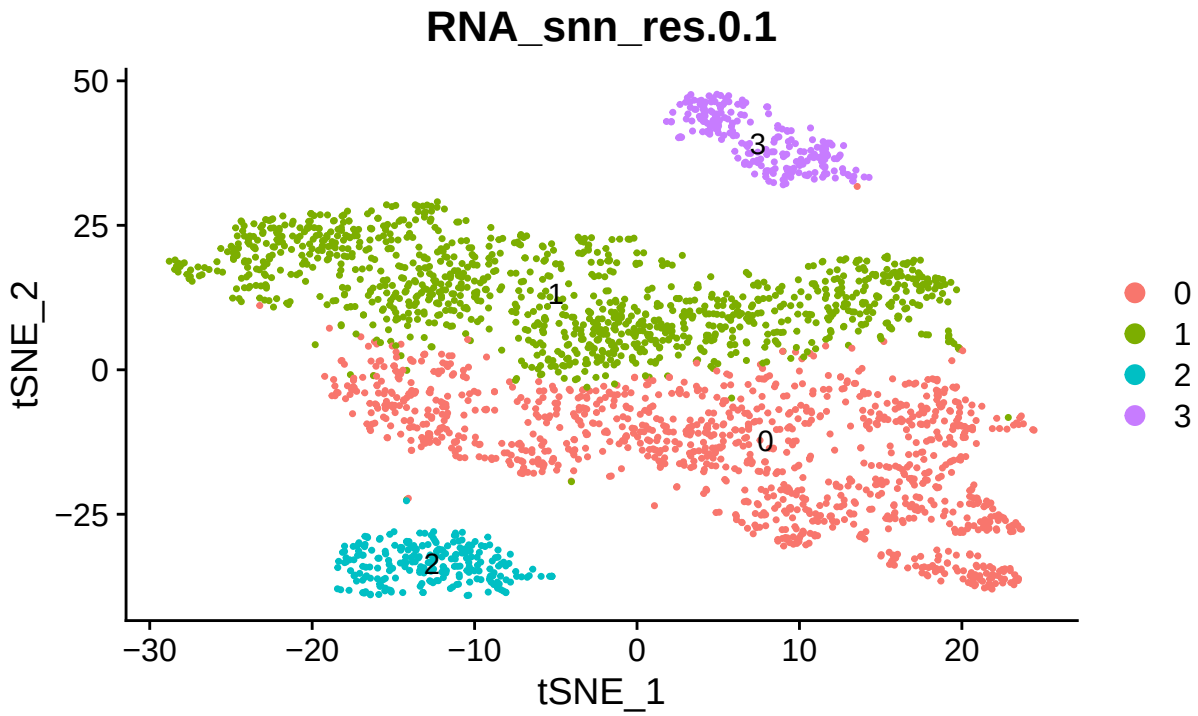
- Plot values in a heatmap. In the heatmap we can see...
  - Cluster 2 in res=0.1 becomes cluster 5 in res=1
  - Cluster 1 in res=0.1 is split into clusters 4, 3, 1 and 7 in res=1

```
library(ComplexHeatmap)
Heatmap(res_0.1vs1,
        col = c("white","yellow","red"),
        name = "Overlap Index",
        column_title = "Resolution 1",
        row_title = "Resolution 0.1")
```

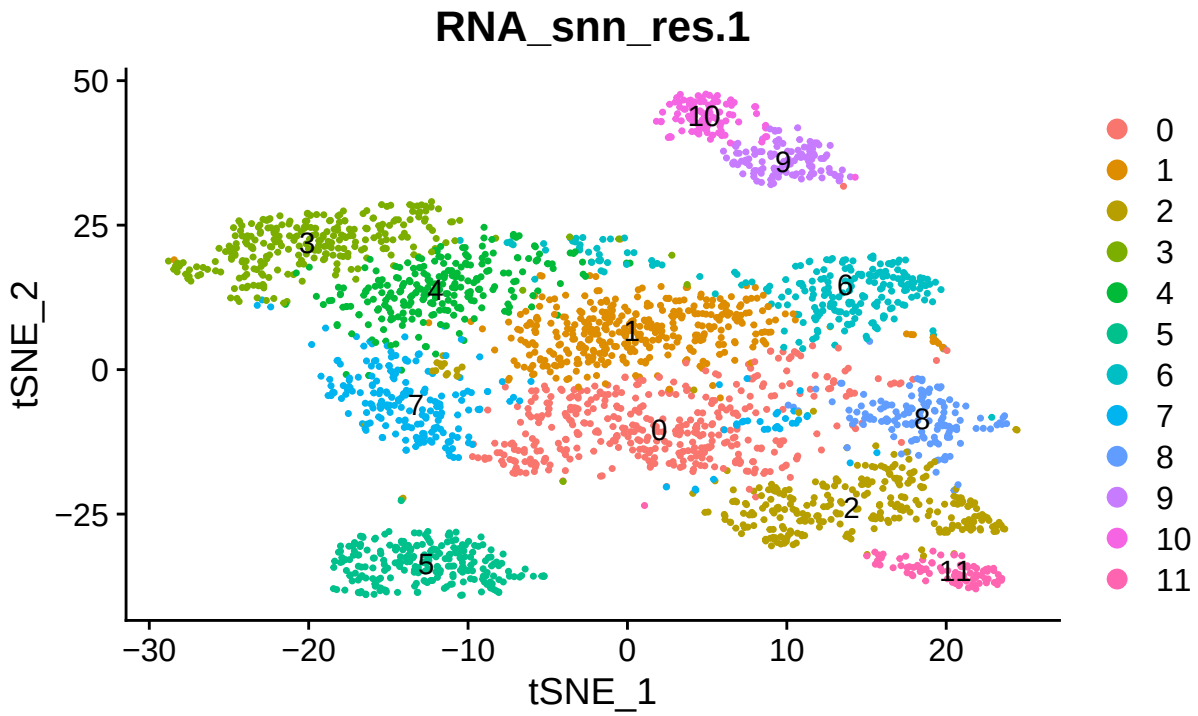


- Also try tSNE plots for a side-by-side comparison

```
DimPlot(sc_subset, reduction='tsne', group.by="RNA_snn_res.0.1", label=T)
```




```
DimPlot(sc_subset, reduction='tsne', group.by="RNA_snn_res.1", label=T)
```



## 2.4 Putative cell type identification

We will try two strategies for this:

1. Look at marker expression per cluster.
2. Look at gene expression correlations with "gold standard" cell type data sets.

### 2.4.1 Marker expression

We have made a small list of a few CD markers and which cell type(s) they are associated with for common immune cell types. In practice, you would want to derive your own cell marker list based on the types of cells you anticipate finding in your data set, the level of specificity you're looking for in defining cell types (e.g., generic T cell, vs T helper, T reg, T responder, etc.), and what markers you consider to be specific and informative for those cell types.

1. Read in the marker list

```
marker_list <- read.table("https://wd.cri.uic.edu/sc_rna/human-blood-markers.txt",
   sep="\t", header=T)
marker_list
```

```
   Marker       Cell.Type
1    CD14        Monocyte
2   CD163        Monocyte
3   CD244        Monocyte
4    CD28          B cell
5    CD38          B cell
6    CD86          B cell
7    CD19 B cell, NK cell
8     CD2         NK cell
9    CD3D NK cell, T cell
10   CD3E NK cell, T cell
11   CD3G NK cell, T cell
12   CD8A          T cell
```

2. Convert between ensembl IDs and gene symbols

3. Convert between ensembl IDs and gene symbols

```
names <- read.table("~/Downloads/V035_F031_subsample/features.tsv.gz",
   sep="\t", row.names=1)
```

4. Use `make.unique()` to deal with duplicate gene symbols

```
symbols_to_ids <- rownames(names)
names(symbols_to_ids) <- make.unique(names[,1])
marker_list$ID <- symbols_to_ids[marker_list$Marker]
```

5. Check we matched all of them

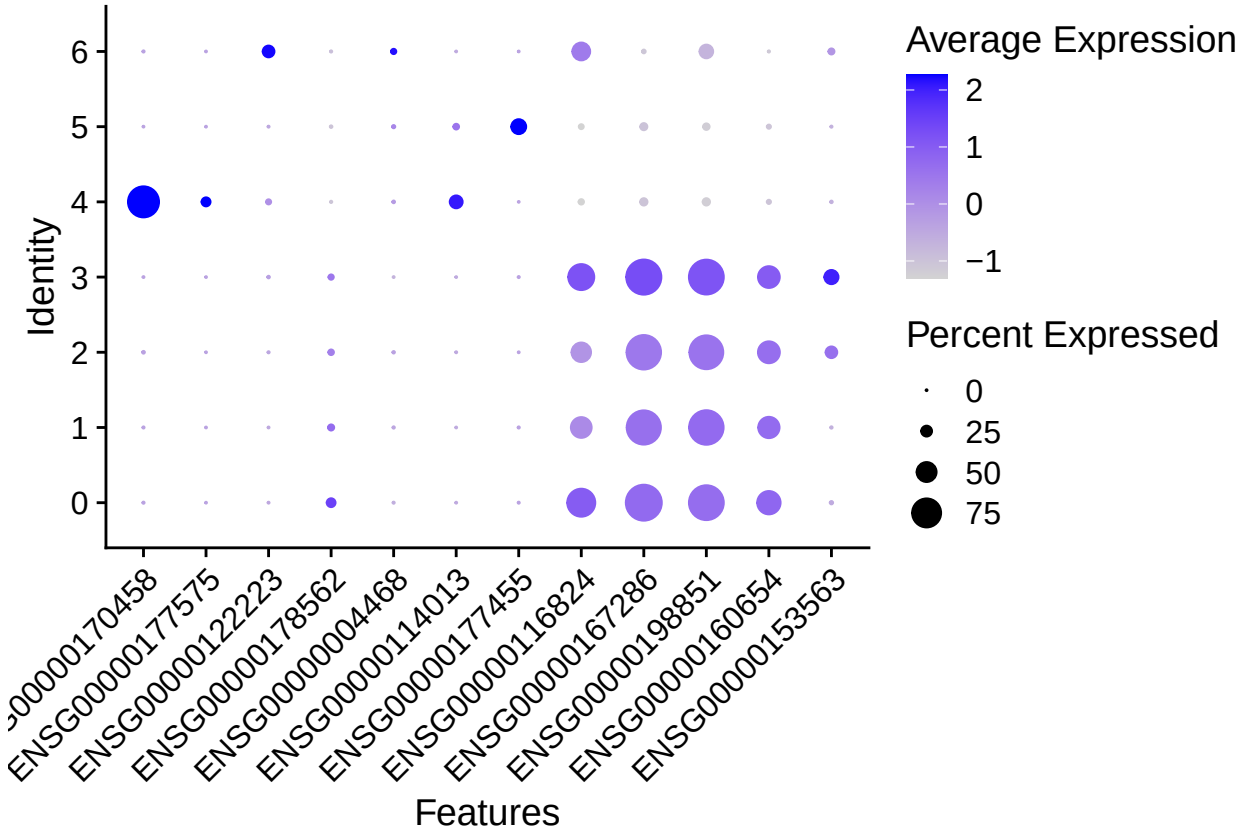
```
marker_list
```

```
   Marker       Cell.Type              ID
1    CD14        Monocyte ENSG00000170458
2   CD163        Monocyte ENSG00000177575
3   CD244        Monocyte ENSG00000122223
4    CD28          B cell ENSG00000178562
5    CD38          B cell ENSG00000004468
6    CD86          B cell ENSG00000114013
7    CD19 B cell, NK cell ENSG00000177455
```

```
8     CD2         NK cell ENSG00000116824
9    CD3D NK cell, T cell ENSG00000167286
10   CD3E NK cell, T cell ENSG00000198851
11   CD3G NK cell, T cell ENSG00000160654
12   CD8A          T cell ENSG00000153563
```

Use DotPlot from Seurat to look at expression levels per cluster.

```
DotPlot(sc_subset, features=marker_list$ID, group.by="RNA_snn_res.0.5" ) + RotatedAxis()
```




The DotPlot would be more useful if we could label the x-axis with gene symbols and cell type descriptions. Let's do that.

1. Add Ensembl IDs to marker list rownames

```
rownames(marker_list) = marker_list$ID
```

2. Add a description column that combines the gene symbol and cell type

```
marker_list$description = paste(marker_list$Cell.Type, marker_list$Marker, sep=": ")
marker_list
```

```
                Marker       Cell.Type              ID          description
ENSG00000170458   CD14        Monocyte ENSG00000170458       Monocyte: CD14
ENSG00000177575  CD163        Monocyte ENSG00000177575      Monocyte: CD163
ENSG00000122223  CD244        Monocyte ENSG00000122223      Monocyte: CD244
ENSG00000178562   CD28          B cell ENSG00000178562         B cell: CD28
ENSG00000004468   CD38          B cell ENSG00000004468         B cell: CD38
ENSG00000114013   CD86          B cell ENSG00000114013         B cell: CD86
```

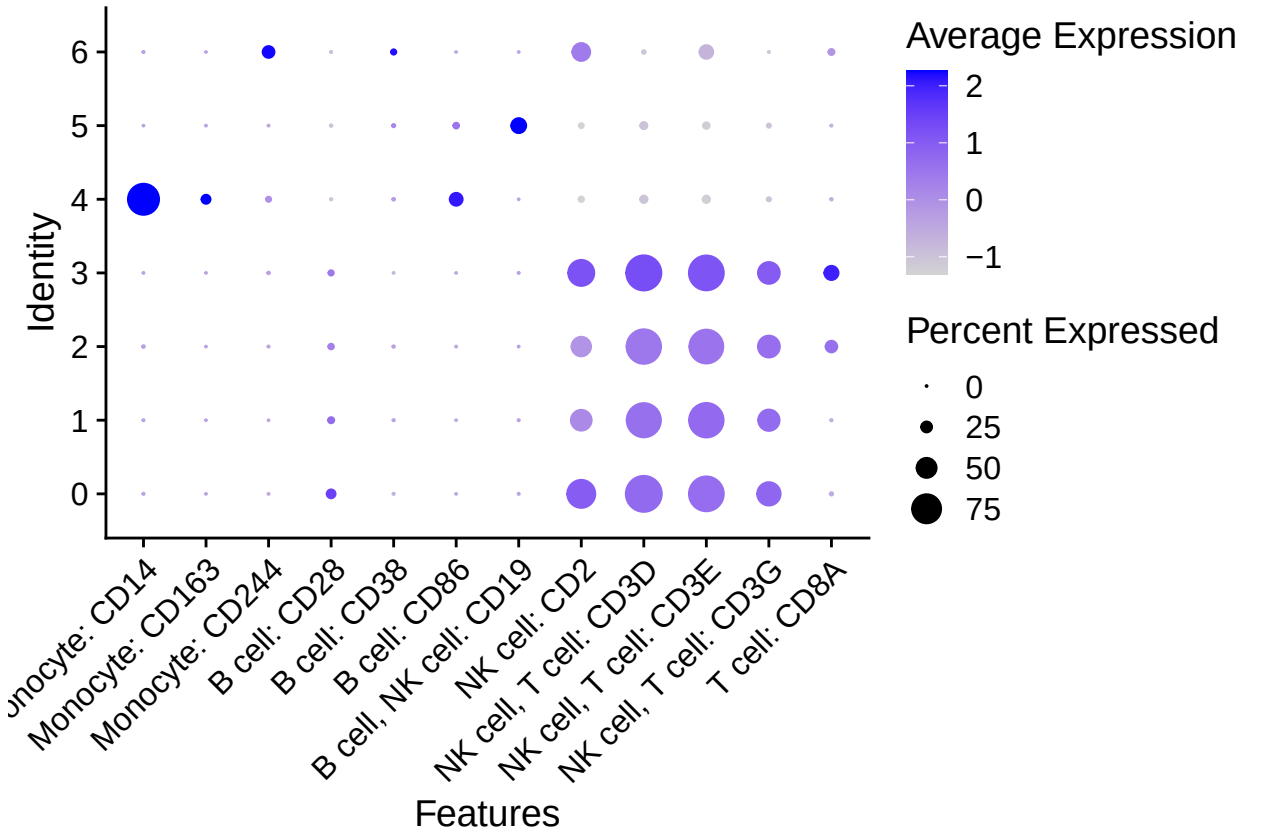
```
ENSG00000177455   CD19 B cell, NK cell ENSG00000177455 B cell, NK cell: CD19
ENSG00000116824    CD2         NK cell ENSG00000116824          NK cell: CD2
ENSG00000167286   CD3D NK cell, T cell ENSG00000167286 NK cell, T cell: CD3D
ENSG00000198851   CD3E NK cell, T cell ENSG00000198851 NK cell, T cell: CD3E
ENSG00000160654   CD3G NK cell, T cell ENSG00000160654 NK cell, T cell: CD3G
ENSG00000153563   CD8A          T cell ENSG00000153563          T cell: CD8A
```

3. Add custom x labels to our plot. The `labels()` parameter in scale_x_discrete lets us change x-tic labels. We're making a function that will return the description column of marker_list, given the Ensembl ID

```
library(ggplot2)
```

```
Warning: package 'ggplot2' was built under R version 4.1.3
```

```
DotPlot(sc_subset, features=marker_list$ID, group.by="RNA_snn_res.0.5" ) +
  RotatedAxis() +
  scale_x_discrete(labels=function(x) marker_list[x,"description"])
```




At this point, you will need to examine the expression of markers across clusters manually decide on cell types.

**NOTE!** For the exercise using gene expression correlations, see the the section titled *Putative cell type annotation: Gene expression correlations* In the **Extra (Take Home) Exercises** section at the end of this document.

## 2.5 Incorporating Feature Barcoding (FBC) Data

### 2.5.1 Loading 10X data into Seurat

**NOTE: DO *NOT* PERFORM THE FOLLOWING COMMANDS**

This is an example of the R commands to load 10X data with both gene expression (GEX) and feature barcoding (FBC) data into Seurat. For this workshop we will be providing a loaded, filtered, and clustered R data object (.rds file). So, skip ahead to *Basic visualization of FBC data* for the actual exercise.

1. Load the matrix files using the standard `Read10X` function.

```
data_w_fbc <- Read10X("Sample_A_matrix/", gene.column=1)
```

```
10X data contains more than one type and is being returned as a list containing matrices of each type.
```

*NOTE*: When the data are loaded by Read10X, it will create a R `list` with elements for both the gene expression (GEX) data and the feature barcoding (FBC) data. You can use the following command to view the elements in this list. This is not necessary if you already know the type of FBC data included.

```
names(data_w_fbc)
```

```
[1] "Gene Expression"  "Antibody Capture"
```

2. Load the Gene Expression (GEX) data into a new Seurat object. The GEX data will be stored in the "RNA" assay in the Seurat object.

```
sc_w_fbc <- CreateSeuratObject(counts = data_w_fbc[['Gene Expression']])
```

3. Add the FBC data as an additional assay in the Seurat object. For this example we will load into the "Protein" assay. *NOTE*: The FBC data does not need to loaded into an assay named "Protein". The name of this assay can be anything (except RNA) that makes sense for your data.

```
sc_w_fbc[["Protein"]] <- CreateAssayObject(counts = data_w_fbc[["Antibody Capture"]])
sc_w_fbc
```

```
An object of class Seurat
32295 features across 594 samples within 2 assays
Active assay: RNA (32285 features, 0 variable features)
 1 other assay present: Protein
```

### 2.5.2 Basic visualization of FBC data

For this exercise we will be providing a loaded, filtered, and clustered R data object. The original data are an example dataset from 10X (https://www.10xgenomics.com/resources/datasets/human-pbmc-from-a-healthy-donor-1-k-cells-v-2-2-standard-4-0-0).

For this Suerat object we had performed the following.

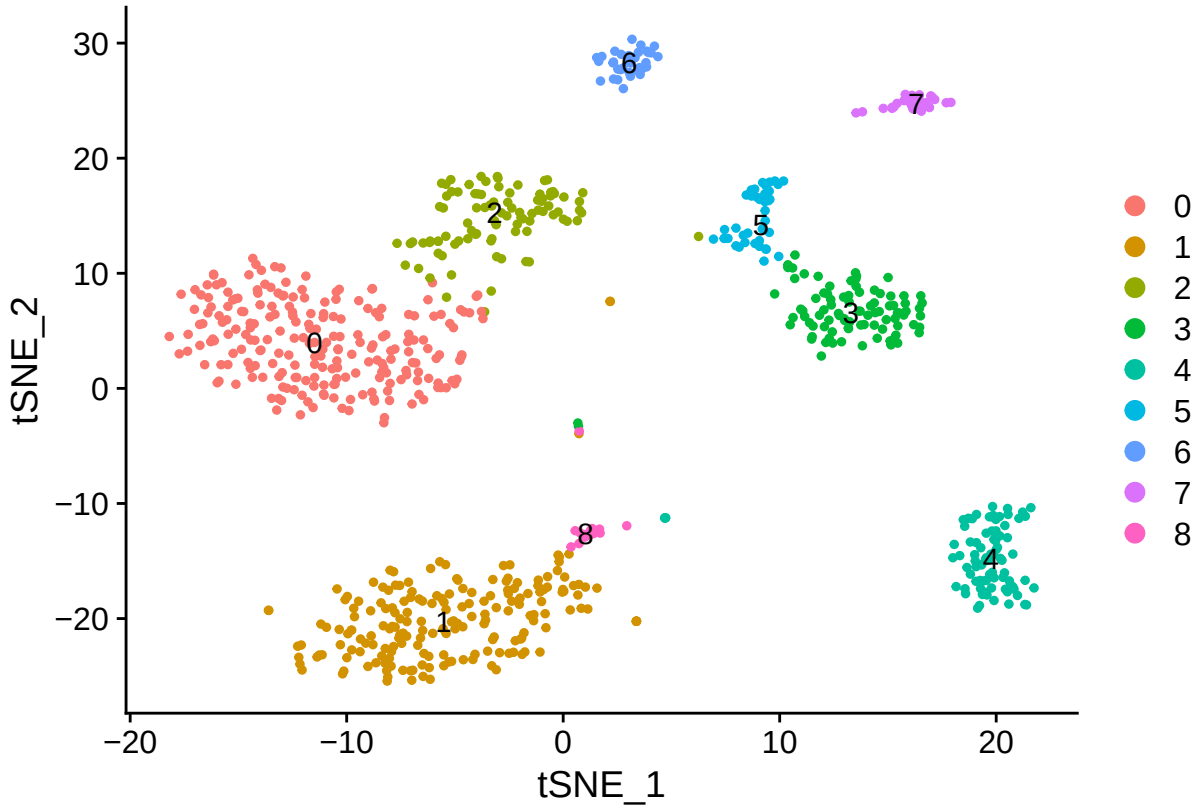
- Load the 10X GEX with FBC data into Seurat using the commands listed above. FBC data were loaded in to the "Protein" assay.
- Filter the cells using the thresholds `nFeature_RNA > 1000 & nCount_RNA > 3000 & percent.MT < 10`.
- Normalized the data using `NormalizeData()`.
- Selected the top 4000 features using `FindVariableFeatures()`.
- Computed PCA object with up to 50 dimensions using `RunPCA()`.
- Executed `FindNeighbors()` using the first 10 PCA dimensions.
- Computed clusters with `FindClusters()` with a resolution of 1.

1. Load the pre-made RDS file.

```
sc_w_fbc <- readRDS(url("https://wd.cri.uic.edu/sc_rna/sc_fbc.rds"))
```

2. Generate basic tSNE plot.

```
DimPlot(sc_w_fbc, reduction='tsne', label=T)
```




3. List the features included in the FBC (Protein) assay.

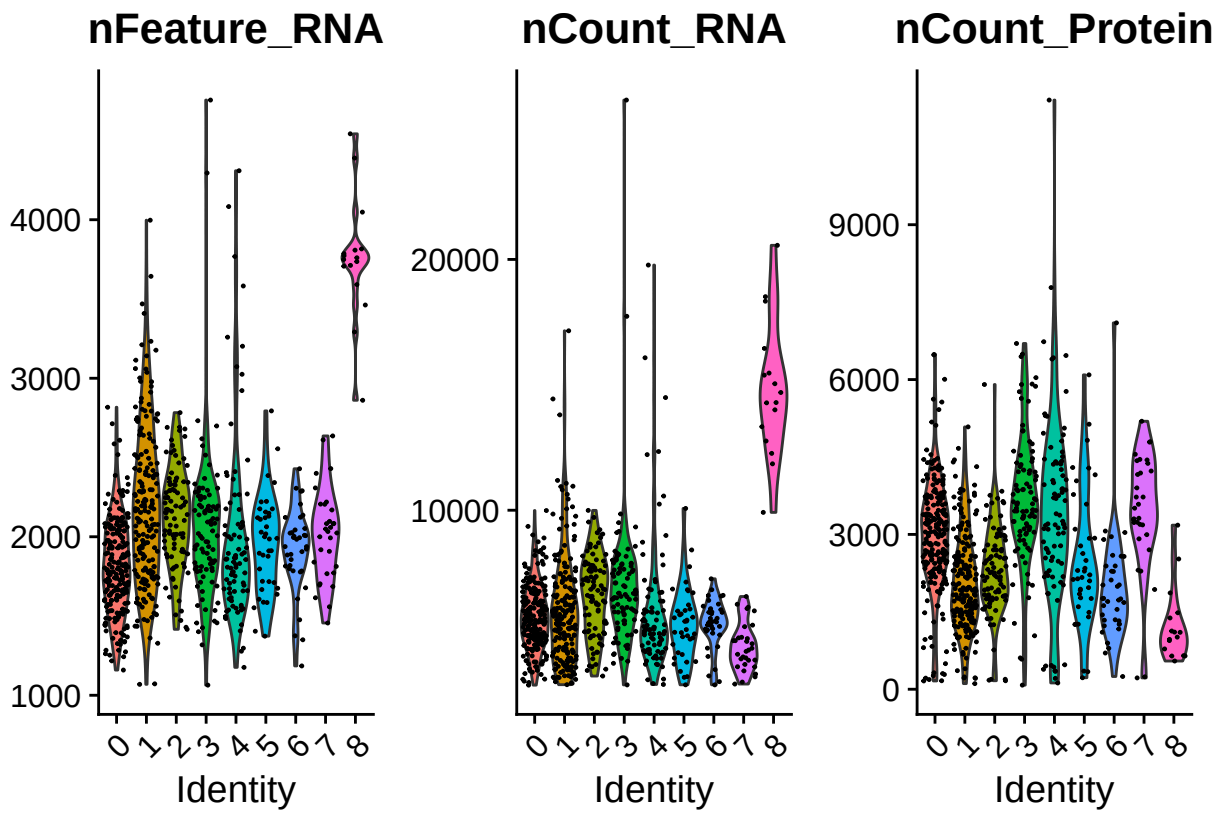
```
row.names(sc_w_fbc[["Protein"]])
```

```
 [1] "CD3"          "CD19"         "CD45RA"       "CD4"          "CD8a"
 [6] "CD14"         "CD16"         "CD56"         "CD25"         "CD45RO"
[11] "PD-1"         "TIGIT"        "IgG1"         "IgG2a"        "IgG2b"
[16] "CD127"        "CD15"         "CD197-(CCR7)" "HLA-DR"
```

4

. Generate violin plots including FBC data and grouped by cluster.

```
VlnPlot(sc_w_fbc,
    features = c("nFeature_RNA","nCount_RNA", "nCount_Protein"),
    pt.size=0.2)
```




5. Normalized the FBC data using a center log ratio (CLR) normalization.

```
sc_w_fbc <- NormalizeData(sc_w_fbc, assay="Protein", normalization.method="CLR")
```

```
Normalizing across features
```

6. Select “Protein” as the default assay for Feature and Dot plots

```
DefaultAssay(sc_w_fbc)
```

```
[1] "RNA"
```

```
DefaultAssay(sc_w_fbc) <- "Protein"
DefaultAssay(sc_w_fbc)
```

```
[1] "Protein"
```

7. Create tSNE plots colored by FBC levels
    a. First get a list of the antibody features.

```
row.names(sc_w_fbc[["Protein"]])
```

```
 [1] "CD3"          "CD19"         "CD45RA"       "CD4"          "CD8a"
 [6] "CD14"         "CD16"         "CD56"         "CD25"         "CD45RO"
```

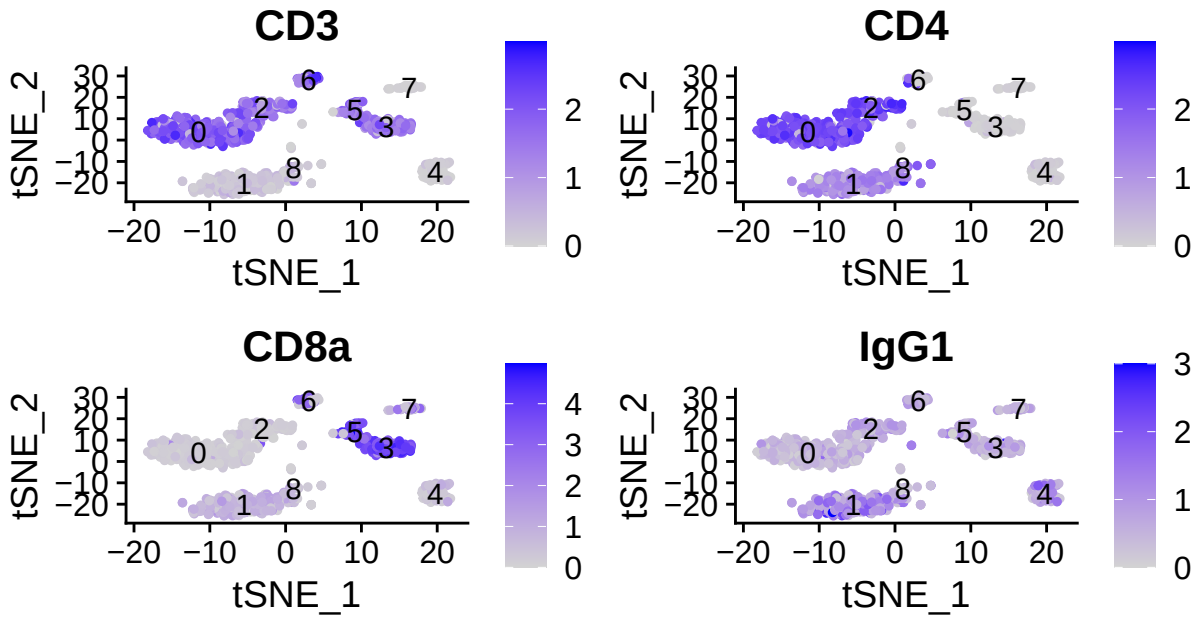
```
[11] "PD-1"         "TIGIT"        "IgG1"          "IgG2a"        "IgG2b"
[16] "CD127"        "CD15"         "CD197-(CCR7)" "HLA-DR"
```

b. Use the following four antibody tags for the plots

```
sel_ab <- c("CD3", "CD4", "CD8a", "IgG1")
```

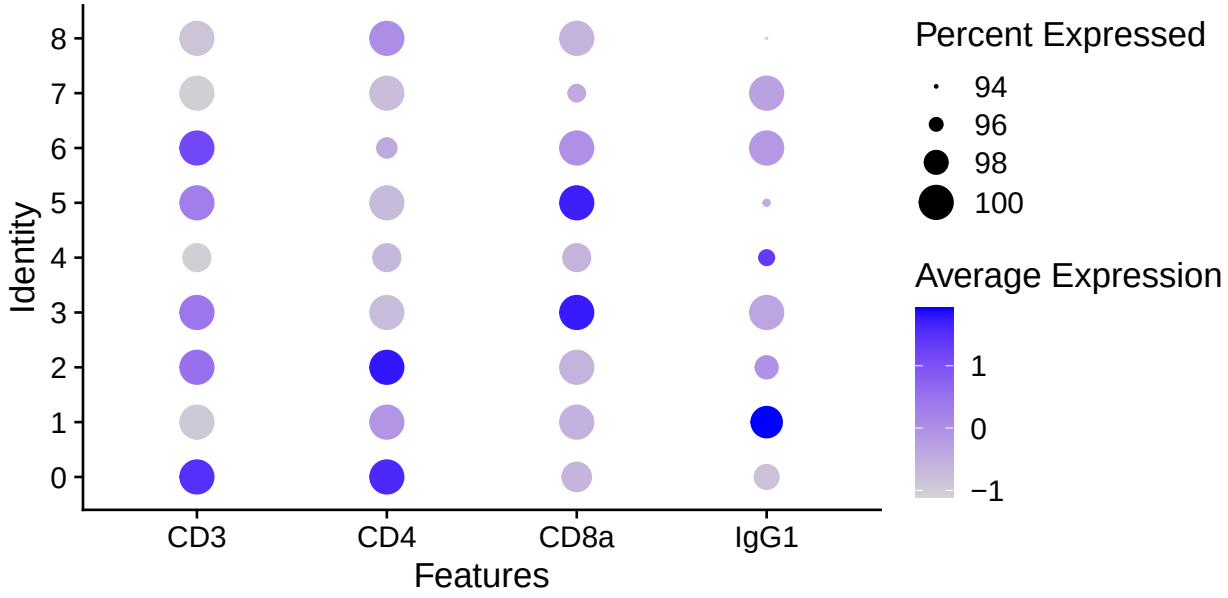
c. Generate the tSNE plot colored by the selected antibody tags

```
FeaturePlot(sc_w_fbc, features=sel_ab, label=T)
```



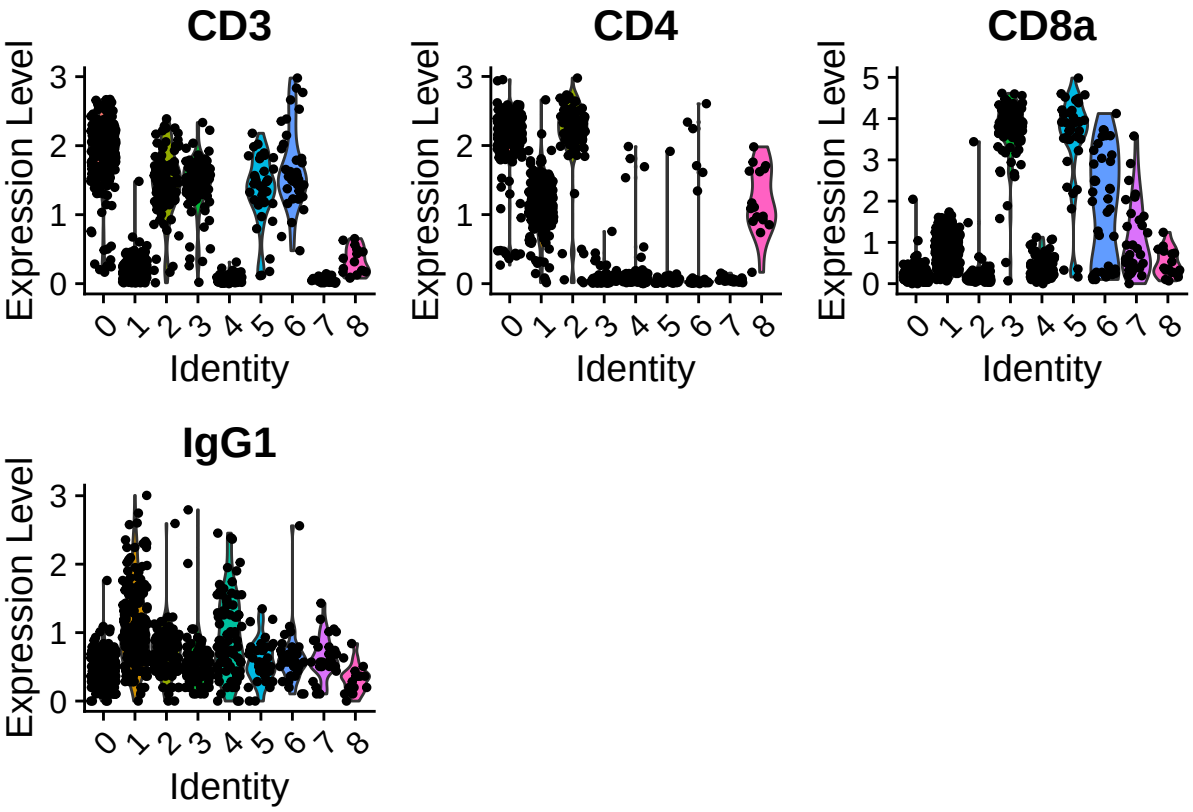

8. Create dot plot of FBC features

```
DotPlot(sc_w_fbc, features=sel_ab)
```



9. Create violin plot of features

```
VlnPlot(sc_w_fbc, features=sel_ab)
```



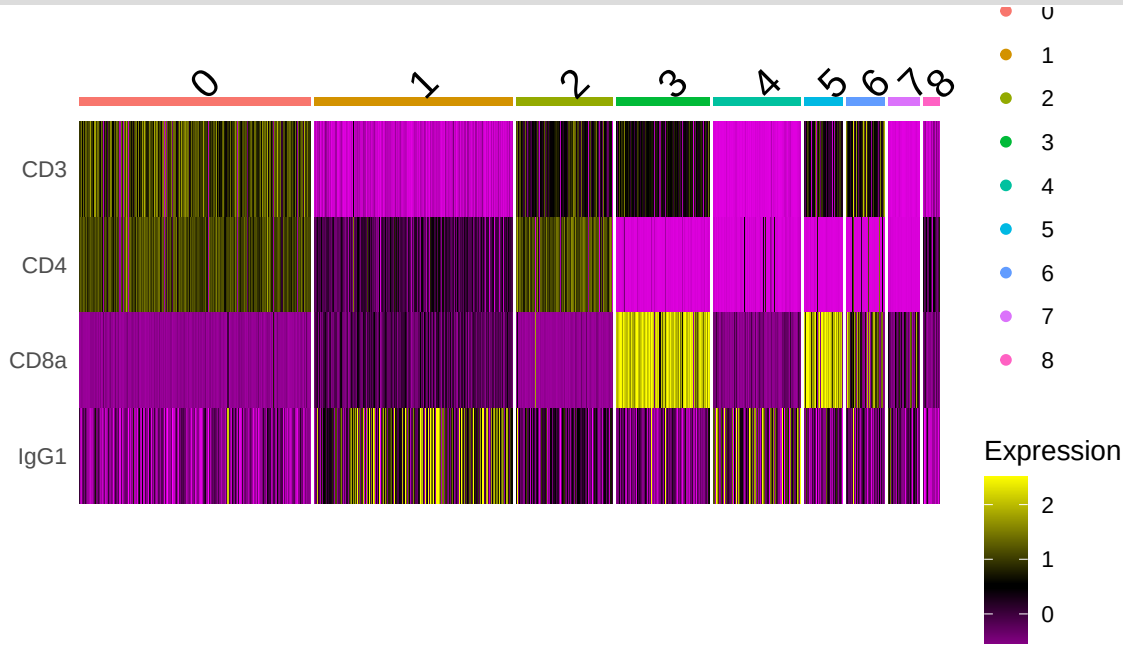

10. Create heatmap of features
    a. First need to scale the data for the heatmap

```
sc_w_fbc <- ScaleData(sc_w_fbc, assay="Protein")
```

```
Centering and scaling data matrix
```

b. Create a heatmap of the selected antibody tags

```
DoHeatmap(sc_w_fbc, features=sel_ab)
```




c. Create a heatmap of all antibody tags

```
DoHeatmap(sc_w_fbc, features=row.names(sc_w_fbc[["Protein"]]))
```

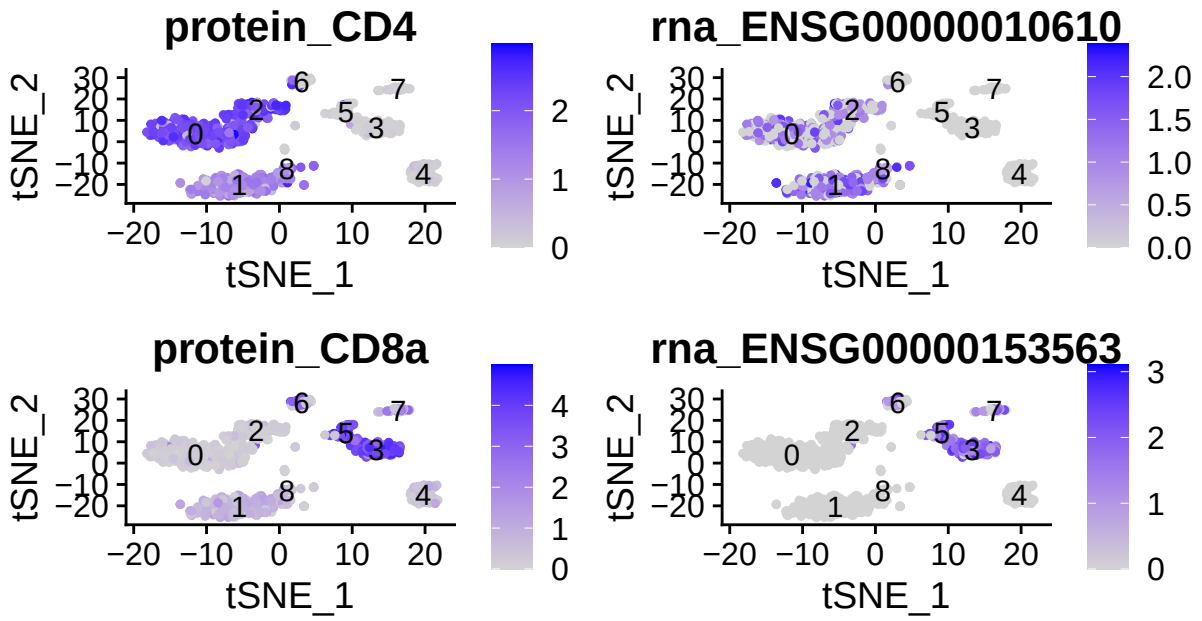
#### 2.5.2.1 Plot both RNA (gene expression) and FBC features together

1. Select the FBC (protein) and RNA features for CD4 and CD8a. In these data the RNA features are identified by Ensembl ID. CD4 is ENSG00000010610 and CD8a is ENSG00000153563.

```
sel_features <- c("protein_CD4", "rna_ENSG00000010610",
                  "protein_CD8a", "rna_ENSG00000153563")
```

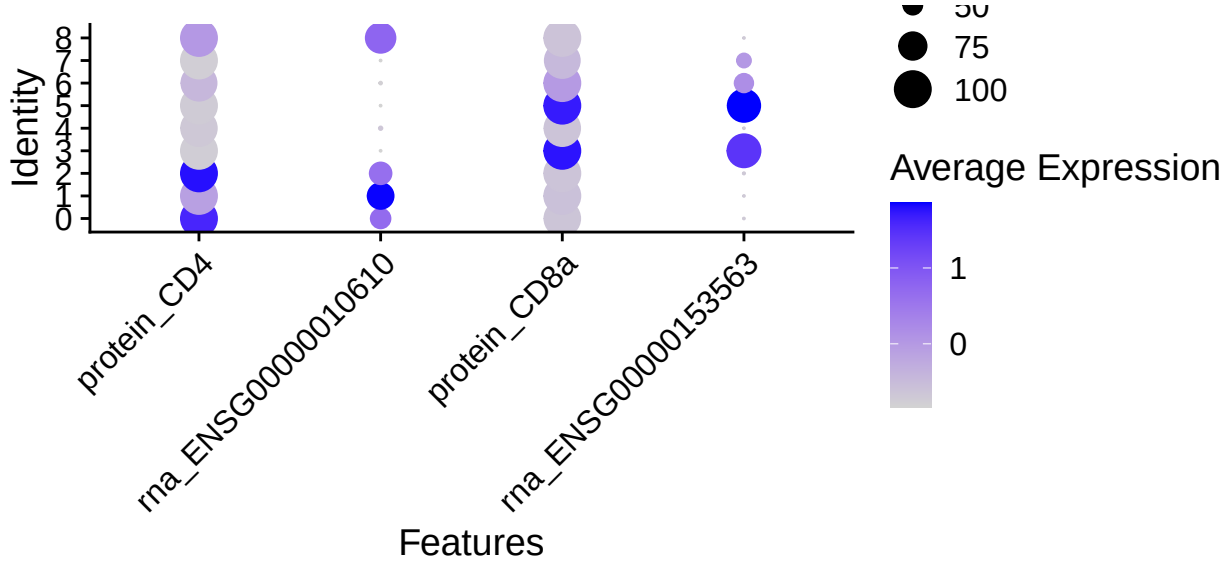
2. Generate feature plots.

```
FeaturePlot(sc_w_fbc, features = sel_features, label=T)
```



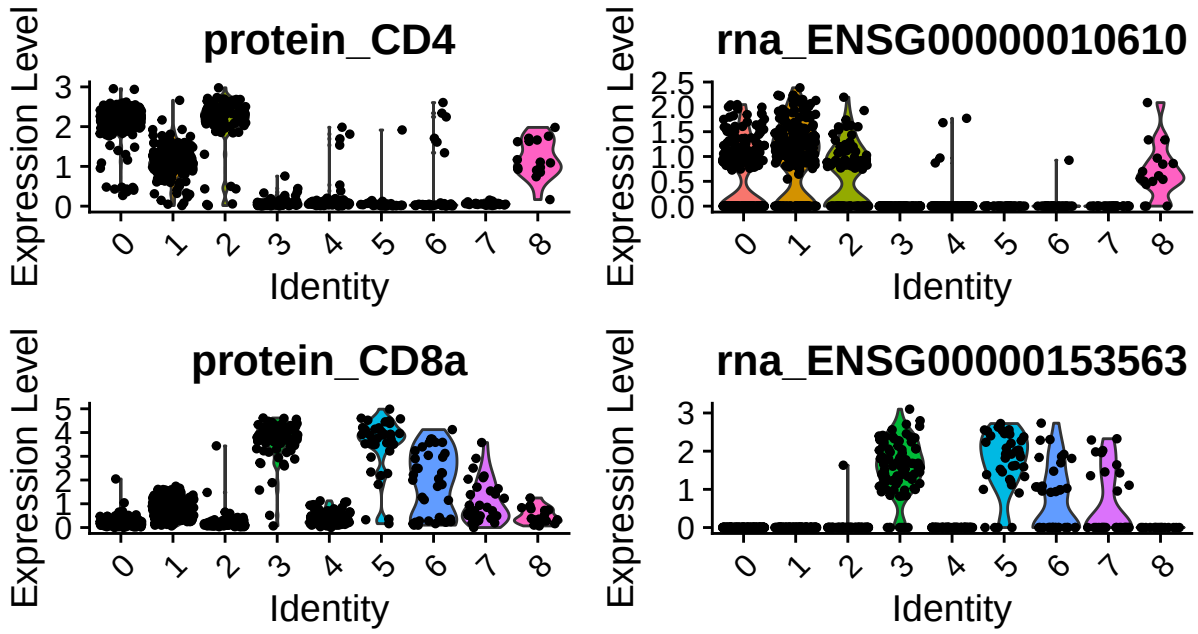

3. Generate dot plots.

```
DotPlot(sc_w_fbc, features = sel_features) + RotatedAxis()
```




4. Generate violin plots.

```
VlnPlot(sc_w_fbc, features = sel_features, ncol=2)
```




### 2.5.3 Computing basic statistics of features

#### 2.5.3.1 Generate cell counts for each cluster with antibody counts above a threshold. (Can use violin plots to determine cutoff)

1. Compute "presence" of each antibody tag if more than 1 normalized (center log ratio) counts. *NOTE*: In real datasets, you may need to be more sophisticated in choosing the threshold. You may even need to set different thresholds for different antibody tags.

```
fbc_presence <- t(sc_w_fbc[["Protein"]]@data > 1)
```

2. Get cluster IDs for each cell and merge with the FBC presence table.

```
cluster_ids <- data.frame(Cluster=sc_w_fbc$seurat_clusters)

fbc_presence <- merge(cluster_ids, fbc_presence, by.x=0, by.y=0)
```

3. Compute cell counts for presence/absence for an antibody, e.g. CD4

```
cd4_counts <- table(fbc_presence[,c("Cluster", "CD4")])

cd4_counts
```

```
       CD4
Cluster FALSE TRUE
      0    10  213
      1    55  137
      2     6   87
      3    91    0
      4    80    4
      5    37    1
      6    31    6
      7    31    0
```

```
      8     7     9
```

#### 2.5.3.2 (*OPTIONAL*) Compile a table of CD4+ cell counts and a fraction of cells in each cluster.

1. Convert the compute table to a data.frame

```
cd4_counts_df <- as.data.frame(cd4_counts)
```

2. Create a side by side (CD4 presence/absence) data.frame

```
cd4_counts <- merge(subset(cd4_counts_df, CD4 == "TRUE", select=c(Cluster, Freq)),
                    subset(cd4_counts_df, CD4 == "FALSE", select=c(Cluster, Freq)),
                    by.x=1, by.y=1)
```

3. Rename the columns for convenience

```
colnames(cd4_counts)[2:3] <- c("Present", "Absent")
```

4. Compute the fraction of cells that are CD4+

```
cd4_counts$Fraction <- cd4_counts$Present / ( cd4_counts$Present + cd4_counts$Absent )
cd4_counts
```

```
  Cluster Present Absent   Fraction
1       0     213     10 0.95515695
2       1     137     55 0.71354167
3       2      87      6 0.93548387
4       3       0     91 0.00000000
5       4       4     80 0.04761905
6       5       1     37 0.02631579
7       6       6     31 0.16216216
8       7       0     31 0.00000000
9       8       9      7 0.56250000
```

## 2.6 Incorporating V(D)J data

Information about the format of the `filtered_contig_annotations.csv` file can be found at https://support.10xgenomics.com/single-cell-vdj/software/pipelines/latest/output/annotation#contig. For this exercise, the following contig annotations were provided.

| Column | Description |
|---|---|
| barcode | Cell-barcode for this contig. |
| is_cell | True or False value indicating whether the barcode was called as a cell. |
| contig_id | Unique identifier for this contig. |
| high_confidence | True or False value indicating whether the contig was called as high-confidence (unlikely to be a chimeric sequence or some other artifact). |
| length | The contig sequence length in nucleotides. |
| chain | The chain associated with this contig; for example, TRA, TRB, IGK, IGL, or IGH. A value of "Multi" indicates that segments from multiple chains were present. |
| v_gene | The highest-scoring V segment, for example, TRAV1-1. |
| d_gene | The highest-scoring D segment, for example, TRBD1. |
| j_gene | The highest-scoring J segment, for example, TRAJ1-1. |
| c_gene | The highest-scoring C segment, for example, TRAC. |
| full_length | If the contig was declared as full-length. |
| productive | If the contig was declared as productive. |
| cdr3 | The predicted CDR3 amino acid sequence. |
| cdr3_nt | The predicted CDR3 nucleotide sequence. |
| reads | The number of reads aligned to this contig. |
| umis | The number of distinct UMIs aligned to this contig. |
| raw_clonotype_id | The ID of the clonotype to which this cell barcode was assigned. |
| raw_consensus_id | The ID of the consensus sequence to which this contig was assigned. |

### 2.6.1 Get basic TCR chain statistics for each cluster.

1. Load the contig annotations into R.

```
sc_vdj_contigs <- read.csv("https://wd.cri.uic.edu/sc_rna/tcr_contigs.csv")
```

2. Clean up the barcode IDs. Seurat will strip the trailing numbers from the cell barcodes, if present.

```
sc_vdj_contigs$barcode <- sub('-[0-9]+$', '', sc_vdj_contigs$barcode)
```

3. Compute the chain counts for each cell.

```
vdj_chain_counts <- as.data.frame(table(sc_vdj_contigs[, c("barcode", "chain")]))
```

4. Add presence/absence for the chains and take a peek at the results.

```
vdj_chain_counts$presence <- vdj_chain_counts$Freq > 0
head(vdj_chain_counts)
```

```
           barcode chain Freq presence
1 AAACGGGTCGCTGATA Multi    0    FALSE
2 AAAGCAAAGTATGACA Multi    0    FALSE
3 AAATGCCCACTTAAGC Multi    0    FALSE
4 AACACGTCAACAACCT Multi    0    FALSE
5 AACACGTGTTATCCGA Multi    0    FALSE
6 AACACGTGTTATCGGT Multi    0    FALSE
```

5. Get the cluster IDs for each cell as a data.frame and take a peek at the results

```
cluster_ids <- data.frame(cluster=Idents(sc_w_fbc))
head(cluster_ids)
```

```
                 cluster
AAACCTGCAGCCTGTG       6
AAACGGGTCGCTGATA       2
AAAGCAAAGTATGACA       2
AAATGCCCACTTAAGC       0
AACACGTCAACAACCT       0
AACACGTCAAGCGATG       1
```

6. Compile chain annotations per cell. Please note, this will require the use of `dplyr`
   a. Load the `dplyr` library

```
library(dplyr)
```

   b. Then use dplyr to create a chain "assignment" for each cell. Basically concatenate the name of the chains detected in each cell into a comma separated list

```
vdj_cell_anno <- subset(vdj_chain_counts, presence) %>%
        group_by(barcode) %>%
        summarize(chains=paste(chain, collapse=","))
```

   c. Take a peek at the results

```
head(vdj_cell_anno)
```

```
# A tibble: 6 x 2
  barcode          chains
  <fct>            <chr>
1 AAACGGGTCGCTGATA TRA,TRB
2 AAAGCAAAGTATGACA TRA,TRB
3 AAATGCCCACTTAAGC TRA,TRB
4 AACACGTCAACAACCT TRA,TRB
5 AACACGTGTTATCCGA TRA,TRB
6 AACACGTGTTATCGGT TRA,TRB
```

7. Merge the cell chain annotations and cluster IDs

```
vdj_cell_chain_cluster <- merge(vdj_cell_anno, cluster_ids, by.x=1, by.y=0, all.y=T)

# Set any NAs to "NONE" for easy visualization later
vdj_cell_chain_cluster$chains[ is.na(vdj_cell_chain_cluster$chains) ] <- "NONE"

# Take a peek at the results
head(vdj_cell_chain_cluster)
```

```
           barcode  chains cluster
1 AAACGGGTCGCTGATA TRA,TRB       2
2 AAAGCAAAGTATGACA TRA,TRB       2
3 AAATGCCCACTTAAGC TRA,TRB       0
4 AACACGTCAACAACCT TRA,TRB       0
5 AACACGTGTTATCCGA TRA,TRB       2
6 AACACGTGTTATCGGT TRA,TRB       3
```

8. Compute the cell/chain counts for each cluster.

```
table(vdj_cell_chain_cluster[,c("cluster","chains")])
```

```
       chains
cluster Multi,TRA,TRB Multi,TRB NONE TRA TRA,TRB TRB
```

```
0              0         0    3   3     204  13
1              0         0  191   0       1   0
2              0         1    2   1      88   1
3              2         1    2   1      80   5
4              0         0   84   0       0   0
5              1         0    4   0      33   0
6              0         0    8   0      25   4
7              0         0   31   0       0   0
8              0         0   16   0       0   0
```

### 2.6.2 Visualize TCR chain presence/absence on tSNE plot.

*NOTE*: This exercise will extract the tSNE plot data from the Seurat object and create a custom plot. It is also possible to add the TCR data to the Seurat object as a secondary assay, akin to the FBC data, and use the Seurat tools.

1. Get the tSNE coordinates as a data.frame.

```
tsne_coords <- as.data.frame(Embeddings(sc_w_fbc, reduction="tsne"))

# Take a peek at the results
head(tsne_coords)
```

```
                     tSNE_1     tSNE_2
AAACCTGCAGCCTGTG   3.4724126  29.196388
AAACGGGTCGCTGATA  -3.9342305  16.898720
AAAGCAAAGTATGACA  -0.5865363  16.724405
AAATGCCCACTTAAGC  -6.4543106   4.206299
AACACGTCAACAACCT -11.4623578   2.765037
AACACGTCAAGCGATG  -5.6095553 -18.254272
```

2. Merge the tSNE coordinates and chain information.

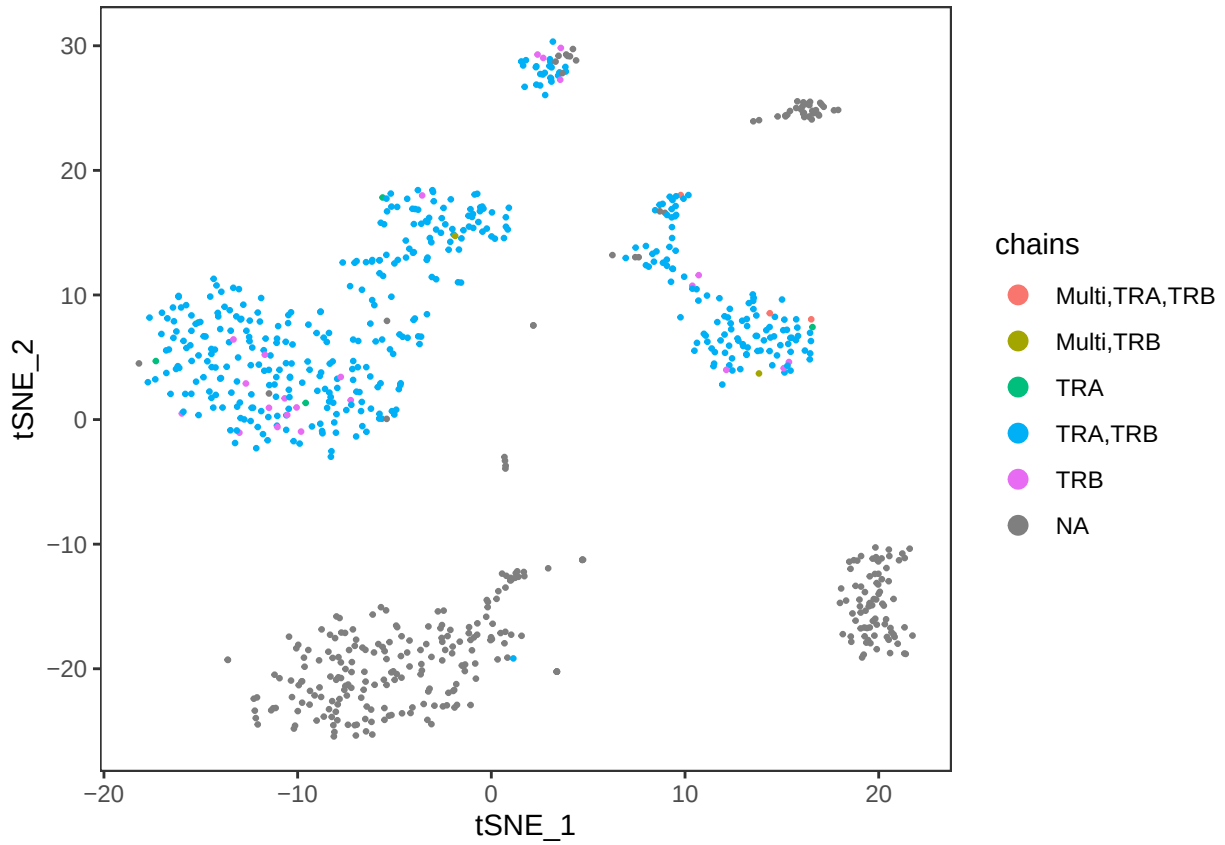
```
plot_data <- merge(vdj_cell_chain_cluster, tsne_coords, by.x=1, by.y=0)

# Take a peek at the results
head(plot_data)
```

```
           barcode  chains cluster      tSNE_1     tSNE_2
1 AAACCTGCAGCCTGTG    NONE       6   3.4724126  29.196388
2 AAACGGGTCGCTGATA TRA,TRB       2  -3.9342305  16.898720
3 AAAGCAAAGTATGACA TRA,TRB       2  -0.5865363  16.724405
4 AAATGCCCACTTAAGC TRA,TRB       0  -6.4543106   4.206299
5 AACACGTCAACAACCT TRA,TRB       0 -11.4623578   2.765037
6 AACACGTCAAGCGATG    NONE       1  -5.6095553 -18.254272
```

3. Create the basic plot, tSNE plot colored by chain information. A few quick notes.
   - If we set the cells with "NONE" for the chains to `NA` then ggplot2 will plot these with a gray color.
   - The `theme(panel.grid = element.blank())` will remove the grid lines on the plot (more like the tSNE plot that Seurat will create)
   - The `guides(color=guide_legend(override.aes = list(size=3)))` will set the size of the plot character in the legend to be a different size than the plot characters used in the plot.

```
library(ggplot2)

plot_data$chains[ plot_data$chains == "NONE" ] <- NA

ggplot(plot_data, aes(x=tSNE_1, y=tSNE_2, color=chains)) +
```

```
        geom_point(size=0.5) +
        theme_bw() + theme(panel.grid = element_blank()) +
        guides(color=guide_legend(override.aes = list(size=3)))
```




**NOTE**: Additional exercise of incorporating V(D)J data into Seurat is found in *Extra (Take Home) Exercises*.

# 3 Extra (Take Home) Exercises

## 3.1 Pseudotime

### 3.1.1 Load the libraries and data (if necessary)

1. Load the `Seurat` (needed to manipulate the previously create Seurat object) and `monocle` for the pseudotime analysis

```
# load both libraries, if not done already
library(Seurat)
library(monocle)
```

```
Warning: package 'VGAM' was built under R version 4.1.3
```

```
Warning: package 'DDRTree' was built under R version 4.1.3
```

```
Warning: package 'irlba' was built under R version 4.1.3
```

2. Load the previously analyzed Seurat object, if needed.

```
sc_subset <- readRDS("sc_subset.rds")
```

### 3.1.2 Get Monocle object from Seurat

Monocle has a function called "importCDS" that is supposed to import from Seurat. But it doesn't work. Fun. We have created a fixed version of the function, which you can source from public-data.

*Pro tip: If you want to see the code of a function, just type its name and hit enter. E.g., type "importCDS" to see the code for this function in Monocle2. After you've sourced our R script, you can type "importCDS2" to see our version.*

```
# source the R script
source("https://wd.cri.uic.edu/sc_rna/importCDS2.R")
# use the importCDS2 function (our function) instead of importCDS (monocle function)
monocle_data <- importCDS2(sc_subset)
monocle_data
```

```
CellDataSet (storageMode: environment)
assayData: 32738 features, 2791 samples
  element names: exprs
protocolData: none
phenoData
  sampleNames: s1_AAACCTGAGCACCGTC-1 s1_AAACCTGGTAGAGCTG-1 ...
    s2_TTTGTCATCCTCAATT-1 (2791 total)
  varLabels: orig.ident nCount_RNA ... Size_Factor (7 total)
  varMetadata: labelDescription
featureData
  featureNames: ENSG00000243485 ENSG00000237613 ... ENSG00000215611
    (32738 total)
  fvarLabels: gene_short_name
  fvarMetadata: labelDescription
experimentData: use 'experimentData(object)'
Annotation:
```

### 3.1.3 Start processing in Monocle

We need to tell Monocle which genes to base the analysis on. Monocle has a set of steps to do this, similar to how we picked the variable genes before. But we can just use the variable genes we've already selected.

1. Estimate size factors and pick which genes to use

```
monocle_data <- estimateSizeFactors(monocle_data)
monocle_data <- setOrderingFilter(monocle_data, sc_subset@assays$RNA@var.features)
```

2. Reduce dimensions - this step may take awhile

```
monocle_data <- reduceDimension(monocle_data, max_components=2, method='DDRTree')
```

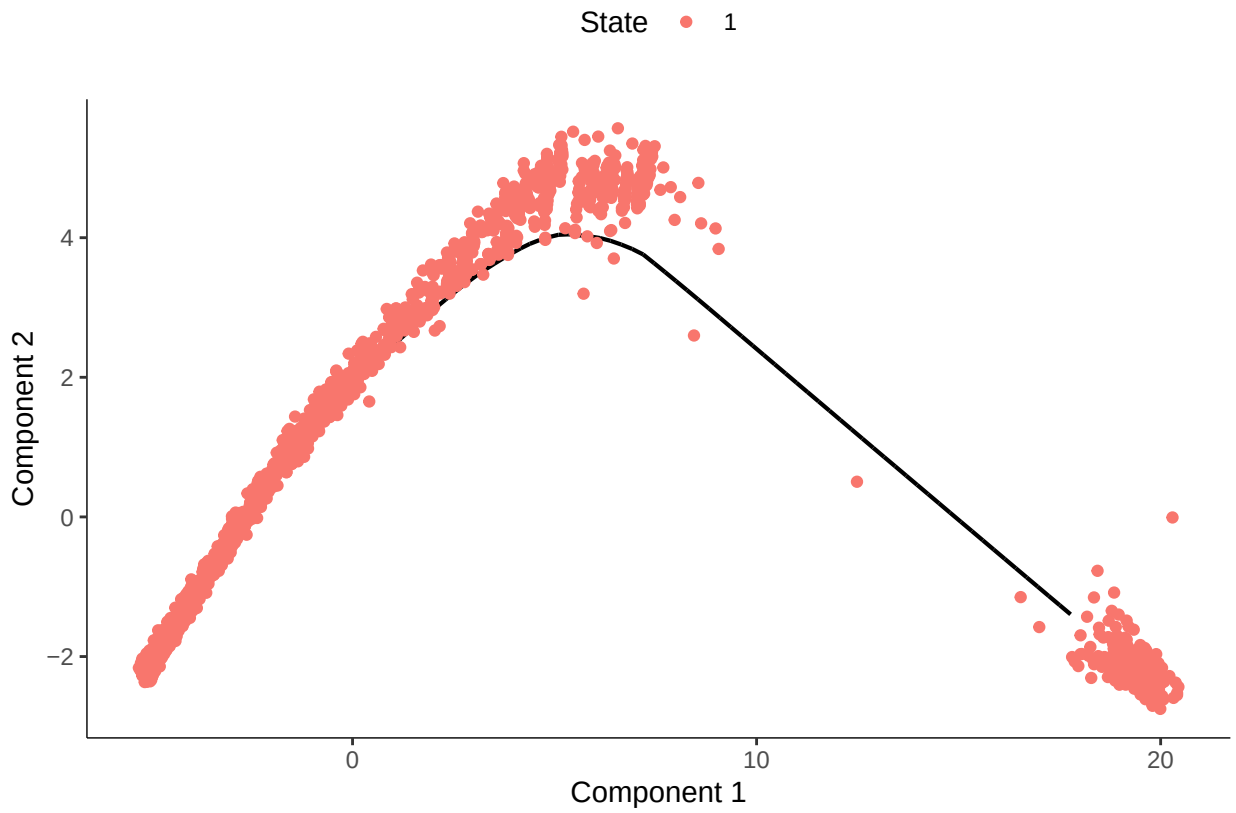
3. Infer the cell ordering (pseudotime). This command may take a minute to run, be patient. It took ~30 seconds on my computer.

```
monocle_data <- orderCells(monocle_data)
```

#### 3.1.4 Try different visualizations

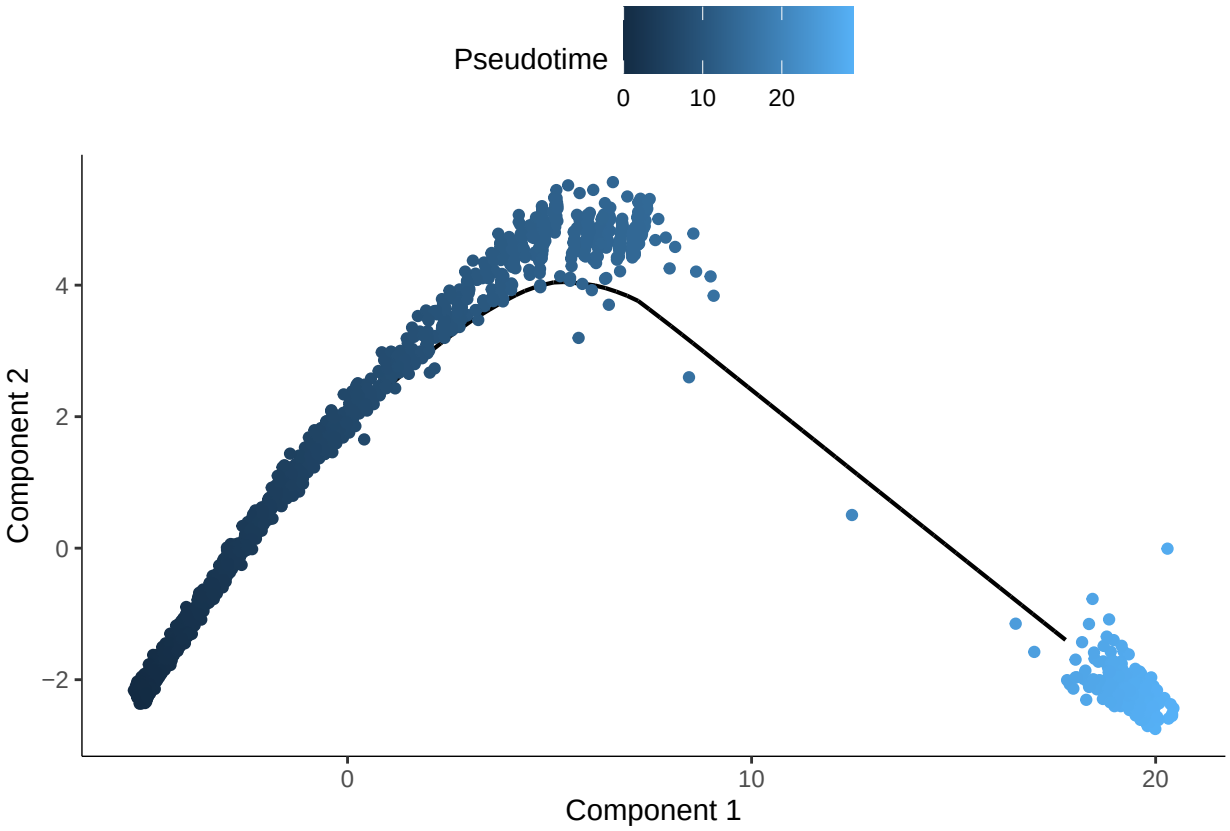
- Default coloring is by "state", the segment of the tree we're in this tree doesn't have any branches, so there is only 1 state

```
plot_cell_trajectory(monocle_data)
```



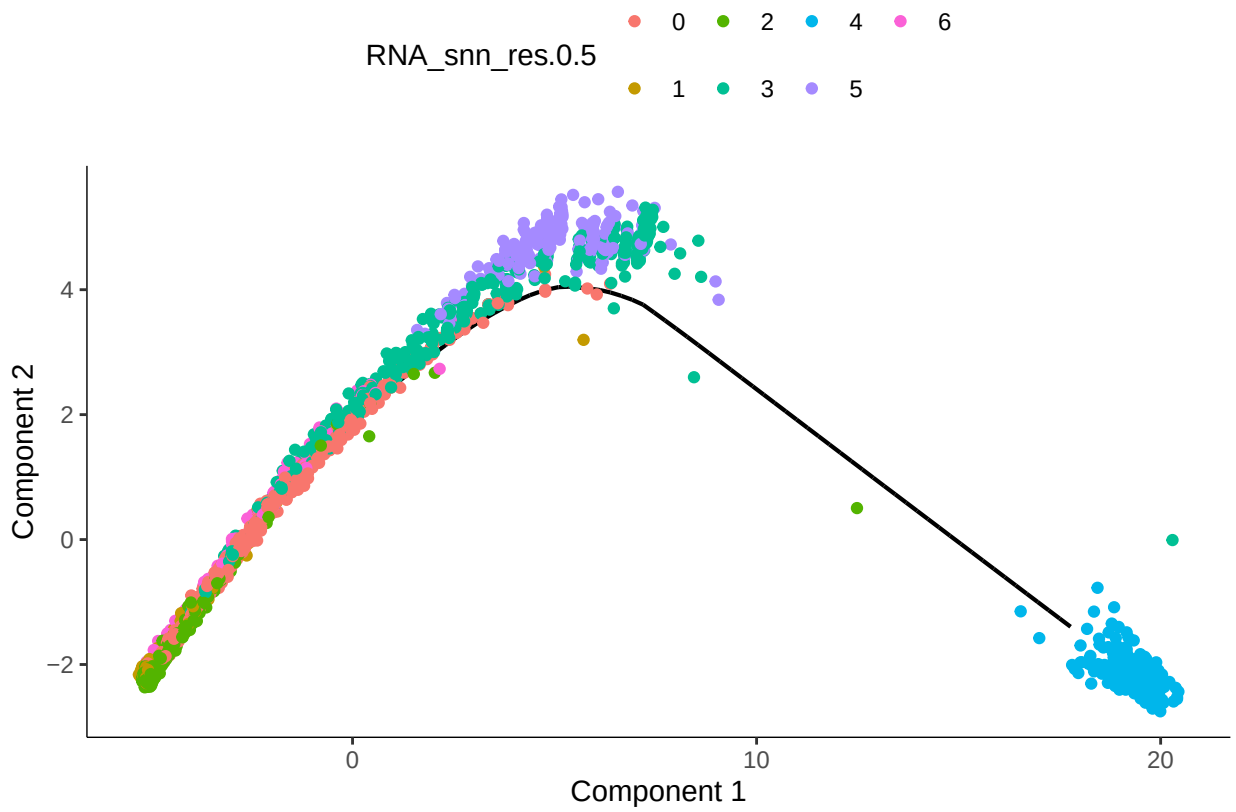
- Can also color by the pseudotime

```
plot_cell_trajectory(monocle_data, color_by = "Pseudotime")
```



- Color by seurat cluster

```
plot_cell_trajectory(monocle_data, color_by = "RNA_snn_res.0.5")
```




### 3.1.5 Further exploration of the data.

The `pData` function from the monocle package will return a data.frame with the computed psuedotime values. As the original Seurat data were also imported, e.g. clusters, original idents.

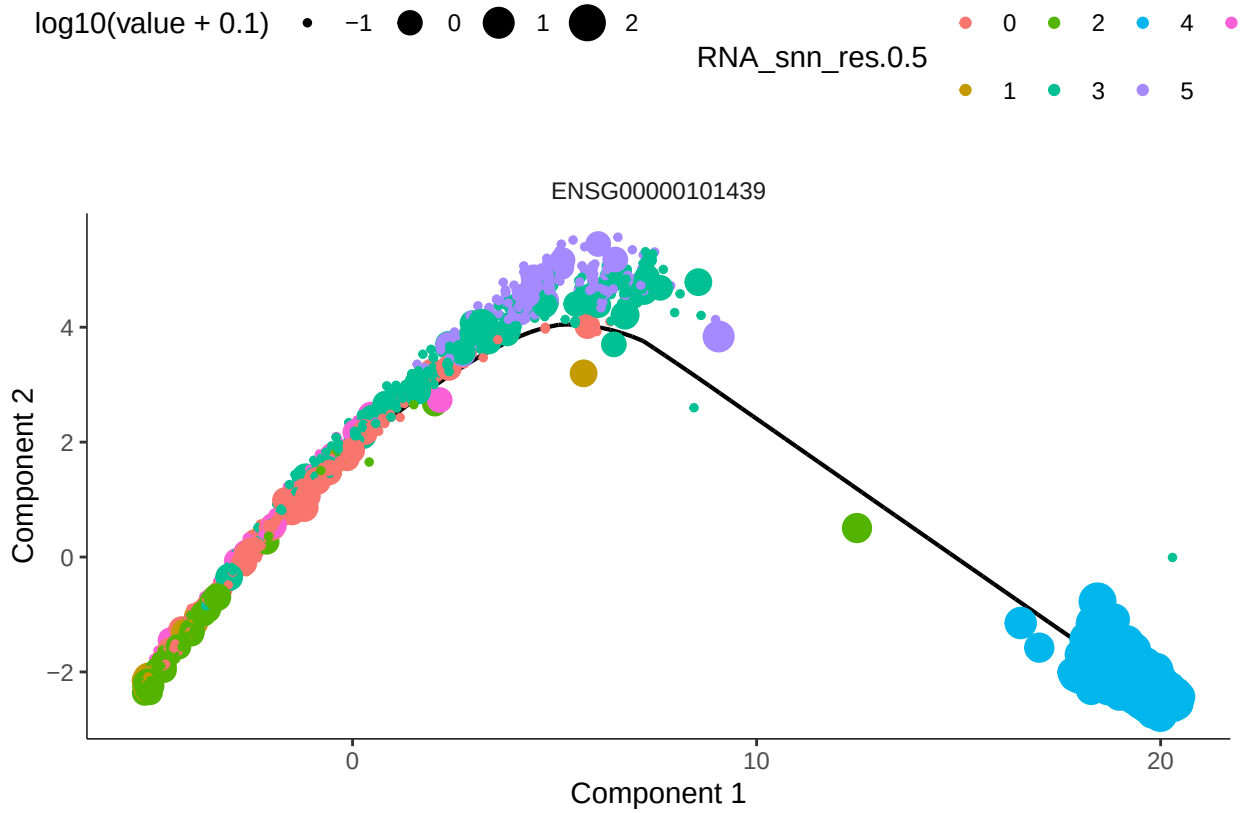
```
head(pData(monocle_data))
```

```
                     orig.ident nCount_RNA nFeature_RNA percent.MT
s1_AAACCTGAGCACCGTC-1    sample1       4236         1105   3.257790
s1_AAACCTGGTAGAGCTG-1    sample1       3116          941   3.498074
s1_AAACCTGGTCTAGCGC-1    sample1       6187         1322   1.729433
s1_AAACCTGGTGTGGCTC-1    sample1       2230          740   3.766816
s1_AAACGGGGTGGTACAG-1    sample1       3038          999   2.797893
s1_AAACGGGTCACCCGAG-1    sample1       3153         1002   4.947669
                     RNA_snn_res.0.5 seurat_clusters RNA_snn_res.0.1
s1_AAACCTGAGCACCGTC-1               5               9               3
s1_AAACCTGGTAGAGCTG-1               0               6               0
s1_AAACCTGGTCTAGCGC-1               1               7               1
s1_AAACCTGGTGTGGCTC-1               6               8               0
s1_AAACGGGGTGGTACAG-1               0               0               0
s1_AAACGGGTCACCCGAG-1               3               2               0
                     RNA_snn_res.1 Size_Factor Pseudotime State
s1_AAACCTGAGCACCGTC-1             9   1.0609141 11.6279862     1
s1_AAACCTGGTAGAGCTG-1             6   0.7804080  4.3312184     1
```

```
s1_AAACCTGGTCTAGCGC-1              7   1.5495457  0.5021657     1
s1_AAACCTGGTGTGGCTC-1              8   0.5585077  4.9794399     1
s1_AAACGGGGTGGTACAG-1              0   0.7608728  8.5888506     1
s1_AAACGGGTCACCCGAG-1              2   0.7896747  5.1233028     1
```

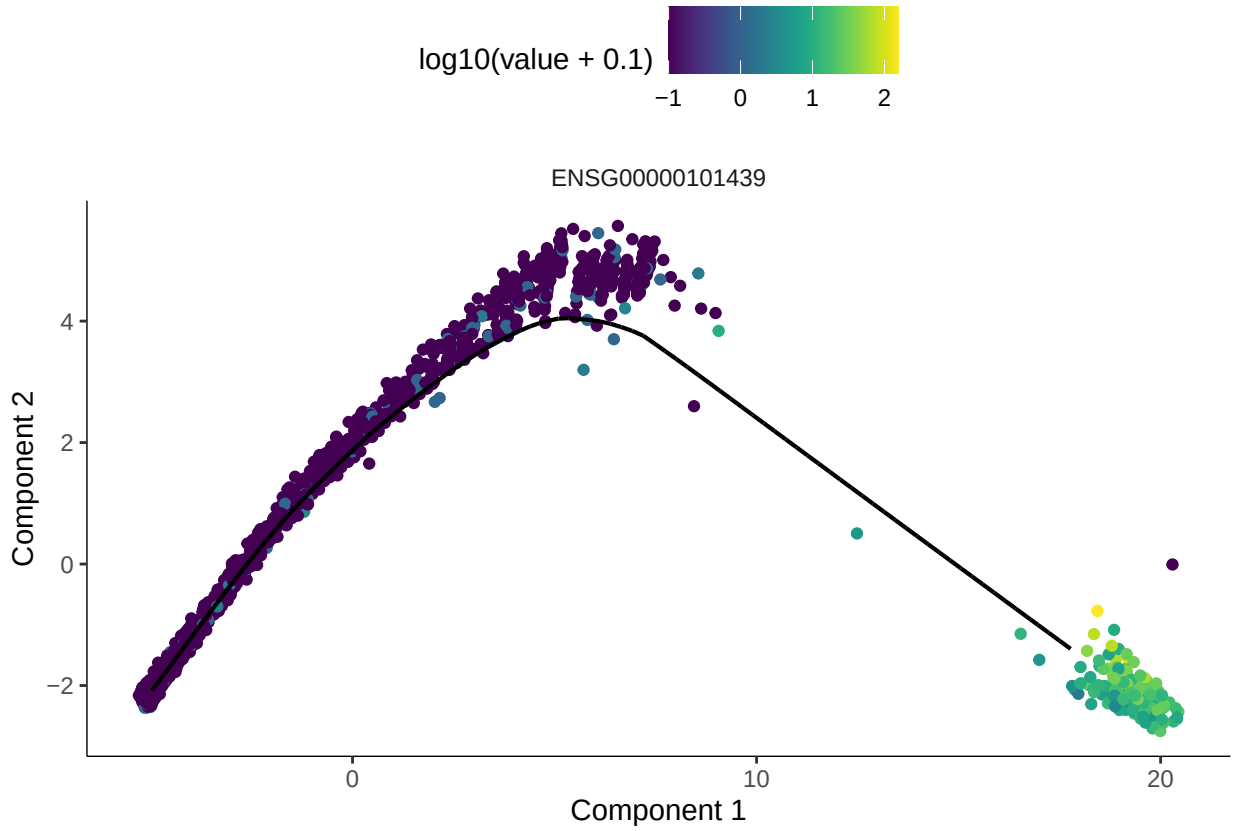
- Can plot time vs gene expression.

```
# Get the top marker gene for cluster 4
topgene = rownames(roc_stats[roc_stats$cluster==4,])[1]
# plot using gene expression as size of dot
plot_cell_trajectory(monocle_data, color_by = "RNA_snn_res.0.5", markers = topgene)
```



- Can plot pseudotime vs gene expression using color

```
plot_cell_trajectory(monocle_data, markers = topgene, use_color_gradient=T)
```




- One can extract plot coordinates from trajectory plot. Note that monocle_data@reducedDimW has "whitened" cell coordinates in the reduced dimension space. analogous to the loadings from PCA

```
head(t(monocle_data@reducedDimS))
```

```
                          [,1]       [,2]
s1_AAACCTGAGCACCGTC-1  4.288248  4.7812101
s1_AAACCTGGTAGAGCTG-1 -1.986593  0.6107187
s1_AAACCTGGTCTAGCGC-1 -4.873036 -1.9120832
s1_AAACCTGGTGTGGCTC-1 -1.545207  1.1164150
s1_AAACGGGGTGGTACAG-1  1.588916  2.9565131
s1_AAACGGGTCACCCGAG-1 -1.426535  1.2003225
```

## 3.2 Statistics with Pseudotime

### 3.2.1 Look for genes that are highly correlated with pseudotime

- We'll test just a subset of genes them to save computational time.
- We're comparing two continuous variables, so a correlation or regression analysis is appropriate.

**First we can use the built-in regression test from Monocle, which smooths the data using a spline fit.**

1. Get a list of genes to test. Usually you might test all the genes, or just the variable features. This example will test just the variable features.

```
to_test <- sc_subset@assays$RNA@var.features
```

2. Subset the monocole data to only include the specified genes

```
monocle_subset <- monocle_data[to_test,]
```

3. Perform differential gene test. `sm.ns` fits a spline fit through the expression values to smooth them a bit. `differentialGeneTest` is then running a regression analysis against the smoothed expression.

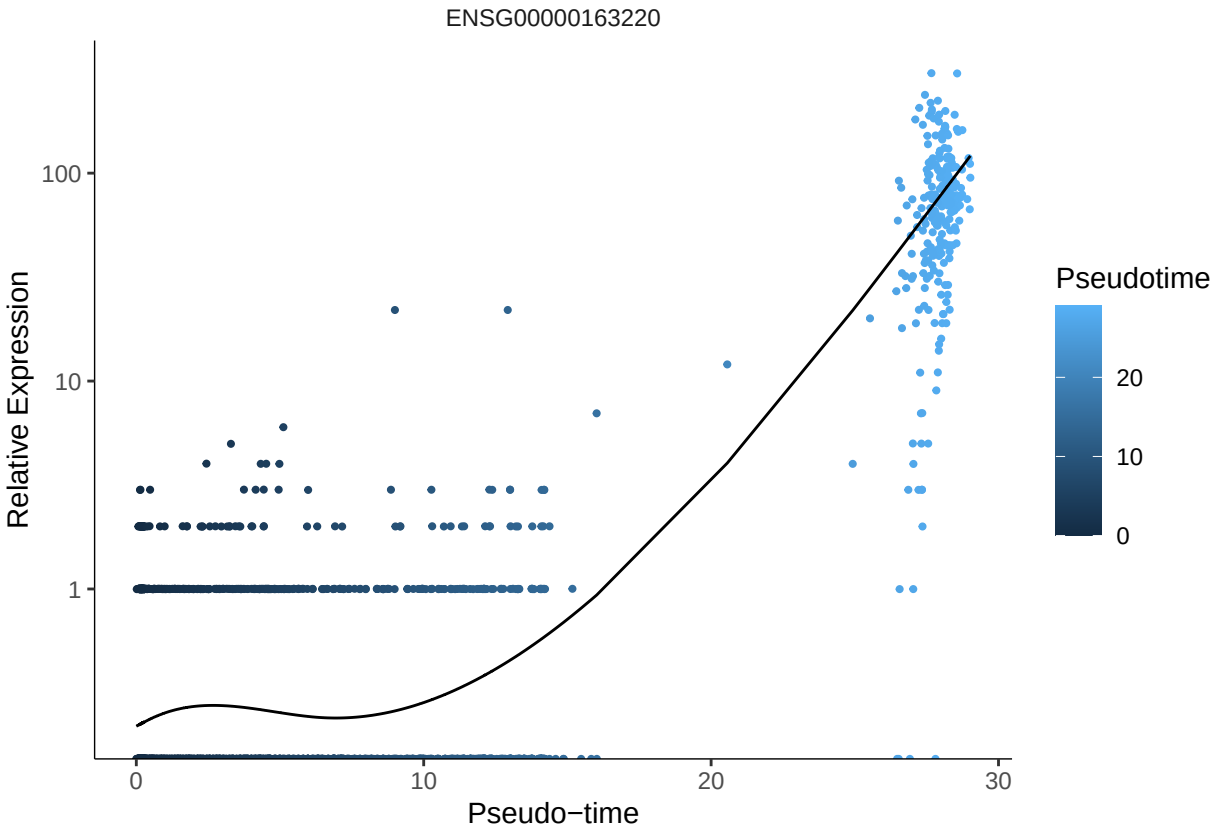
```
pseudotime_test <- differentialGeneTest(monocle_subset,
   fullModelFormulaStr="~sm.ns(Pseudotime)")
head(pseudotime_test)
```

```
                status           family         pval         qval
ENSG00000163220     OK negbinomial.size 0.000000e+00 0.000000e+00
ENSG00000143546     OK negbinomial.size 0.000000e+00 0.000000e+00
ENSG00000090382     OK negbinomial.size 0.000000e+00 0.000000e+00
ENSG00000115523     OK negbinomial.size 0.000000e+00 0.000000e+00
ENSG00000169429     OK negbinomial.size 0.000000e+00 0.000000e+00
ENSG00000254709     OK negbinomial.size 9.291718e-93 1.579987e-91
                gene_short_name use_for_ordering
ENSG00000163220 ENSG00000163220             TRUE
ENSG00000143546 ENSG00000143546             TRUE
ENSG00000090382 ENSG00000090382             TRUE
ENSG00000115523 ENSG00000115523             TRUE
ENSG00000169429 ENSG00000169429             TRUE
ENSG00000254709 ENSG00000254709             TRUE
```

4. Plot genes. This will plot the first gene in the data.

```
plot_genes_in_pseudotime(monocle_subset[1,], color_by = "Pseudotime")
```

```
Warning: Transformation introduced infinite values in continuous y-axis
Transformation introduced infinite values in continuous y-axis
```

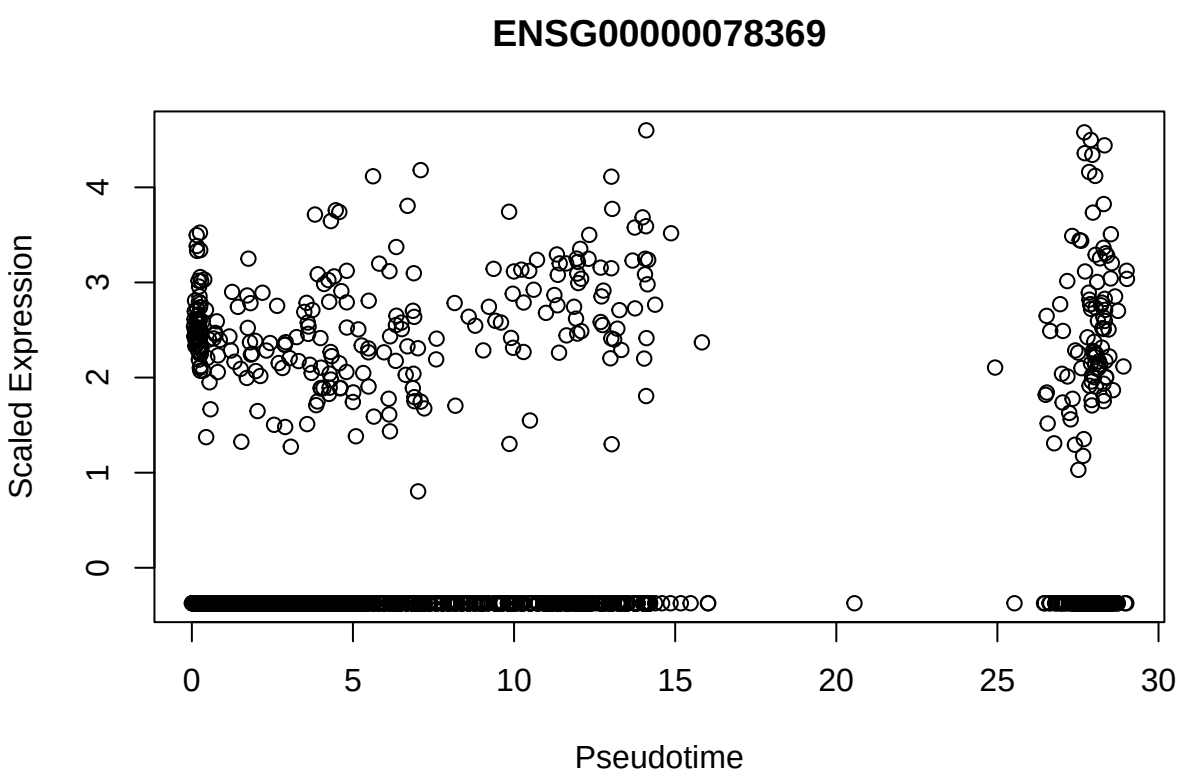


**Alternatively, we can do the test with a Spearman correlation.**

```
pseudotime <- pData(monocle_data)$Pseudotime
# for time purposes, we'll do the first 20 genes again
expression <- sc_subset@assays$RNA@scale.data[1:20,]
# run test with apply statement, making our own short function to return the
# correlation coefficient and p-value from cor.test
# we're going to transpose the result because otherwise the genes are going
# along the columns, and keep it as a data frame
spearman.corrs <- data.frame(t(apply(expression,1,function(x){
   r=cor.test(x,pseudotime,method="spearman"); return(c(r$estimate, r$p.value))})))
colnames(spearman.corrs) = c("rho","p.value")
# add FDR correction
q.value <- p.adjust(spearman.corrs$p.value,method="fdr")
spearman.corrs <- cbind(spearman.corrs,q.value)
# sort by p-value
spearman.corrs <- spearman.corrs[order(spearman.corrs$p.value),]
# see what the final table looks like
head(spearman.corrs)
```

```
                     rho      p.value      q.value
ENSG00000078369 0.19452634 3.344815e-25 6.689631e-24
ENSG00000187608 0.15802841 4.552872e-17 4.552872e-16
ENSG00000189409 0.12201670 9.968879e-11 6.645919e-10
ENSG00000186891 0.10912077 7.491225e-09 3.745612e-08
ENSG00000188290 0.07139697 1.599919e-04 6.399675e-04
ENSG00000186827 0.06902791 2.627936e-04 8.759787e-04
```

```
# plot the top gene
top_gene <- rownames(spearman.corrs)[1]
plot(pseudotime, expression[top_gene,],xlab="Pseudotime",ylab="Scaled Expression",
   main=top_gene)
```

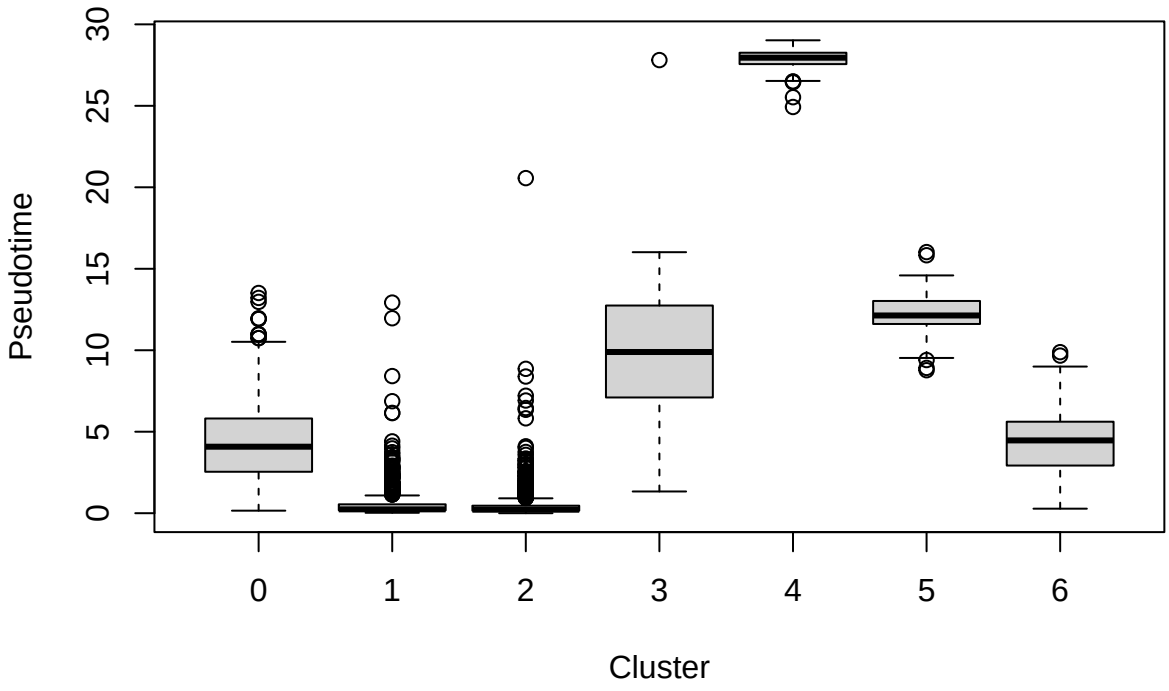



**Note that if we had another pseudotime result, i.e., another analysis that assigned a pseudotime to each cell, we could use cor or cor.test to compare it to this result in much the same way.**

### 3.2.2 Compare our cell clusters with pseudotime

- We're comparing categorical and continuous variables, so a Kruskal-Wallis/ANOVA type test is appropriate.

```
clusters <- sc_subset@meta.data$RNA_snn_res.0.5
pseudotime <- pData(monocle_data)$Pseudotime
cluster_vs_time <- data.frame(clusters, pseudotime)
boxplot(pseudotime ~ clusters, data=cluster_vs_time, xlab="Cluster",ylab="Pseudotime")
```




There is clearly a strong association of cluster vs time. Further tests may be helpful if you want to test the significance of which clusters come earlier or later with respect to pseudotime. For now, we can do an overall test using Kruskal Wallis.

```
kruskal.test(pseudotime ~ clusters)
```

```
    Kruskal-Wallis rank sum test

data:  pseudotime by clusters
Kruskal-Wallis chi-squared = 2304.3, df = 6, p-value < 2.2e-16
```

## 3.3 CytoTRACE

CytoTRACE (Cellular (Cyto) Trajectory Reconstruction Analysis using gene Counts and Expression) is a computational method that predicts the differentiation state of cells from single-cell RNA-sequencing data. More details on this package can be found at https://cytotrace.stanford.edu/

### 3.3.1 Installing CytoTRACE

1. Download CytoTRACE installation source from https://cytotrace.stanford.edu/. **NOTE**: For the latest version go to the CytoTRACE website and then click on *Install R Package* in the sidebar menu. In the installation instructions there will be a link to the current version of CytoTRACE. At the time of writing this exercise, the current version is 0.3.3.

If you are using a Linux or macOS system, you can download the installation package using the following command.

```
wget https://cytotrace.stanford.edu/CytoTRACE_0.3.3.tar.gz
```

2. Execute the following commands in `R`. Modify the `PATH/TO/DOWNLOAD` to reflect the directory where you download the CytoTRACE installation package.

```
if ( ! requireNamespace("devtools"))
  install.packages("devtools")

devtools::install_local("PATH/TO/DOWNLOAD/CytoTRACE_0.3.3.tar.gz")
```

**NOTE**: If CytoTRACE could not be installed because `ERROR: dependency 'sva' is not available for package 'CytoTRACE'`, install "sva" using `BiocManager`.

```
if (! requireNamespace("BiocManager"))
  install.packages("BiocManager")

BiocManager::install("sva", update=F)
```

3. (*OPTIONAL*) If you plan to use CytoTRACE to analyze across multiple batches/datasets you will need to install the Python packages `scanoramaCT` and `numpy`. If you are using a Linux or macOS system, you can execute the following commands to install these packages

```
pip install scanoramaCT
pip install numpy
```

### 3.3.2 Running CytoTRACE

1. Load the Seurat (if not already loaded) and CytoTRACE libraries

```
library(Seurat)
library(CytoTRACE)
```

```
Welcome to the CytoTRACE R package, a tool for the unbiased prediction of differentiation states in scR
```

2. Load the desired Seurat object ***If*** it is NOT currently loaded.

```
sc_obj <- readRDS("sc_subset.rds")
```

3. (*OPTIONAL*) You can subset your Seurat object to clusters of interest, for example, clusters 1 and 2.

```
sc_obj.1_2 <- subset(sc_obj, subset = seurat_clusters == 1 | seurat_clusters == 2)
```

4. Get counts matrix of your Seurat object. Either the whole object or the subset from the previous step. In this example, we are just going to use the subset.

```
sc_counts <- as.matrix(sc_obj@assays$RNA@counts)
```

5. Run CytoTRACE analysis (check methods for what enableFast does if you want to enable it)

```
cyto_results <- CytoTRACE(sc_counts, enableFast=F)
```

```
The number of cells in your dataset is less than 3,000. Fast mode has been disabled.

Warning in CytoTRACE(sc_counts, enableFast = F): 14919 genes have zero
expression in the matrix and were filtered

CytoTRACE will be run on 1 sub-sample(s) of approximately 2791 cells each using 1 / 1 core(s)

Pre-processing data and generating similarity matrix...

Calculating gene counts signature...

Smoothing values with NNLS regression and diffusion...

Calculating genes associated with CytoTRACE...

Done
```

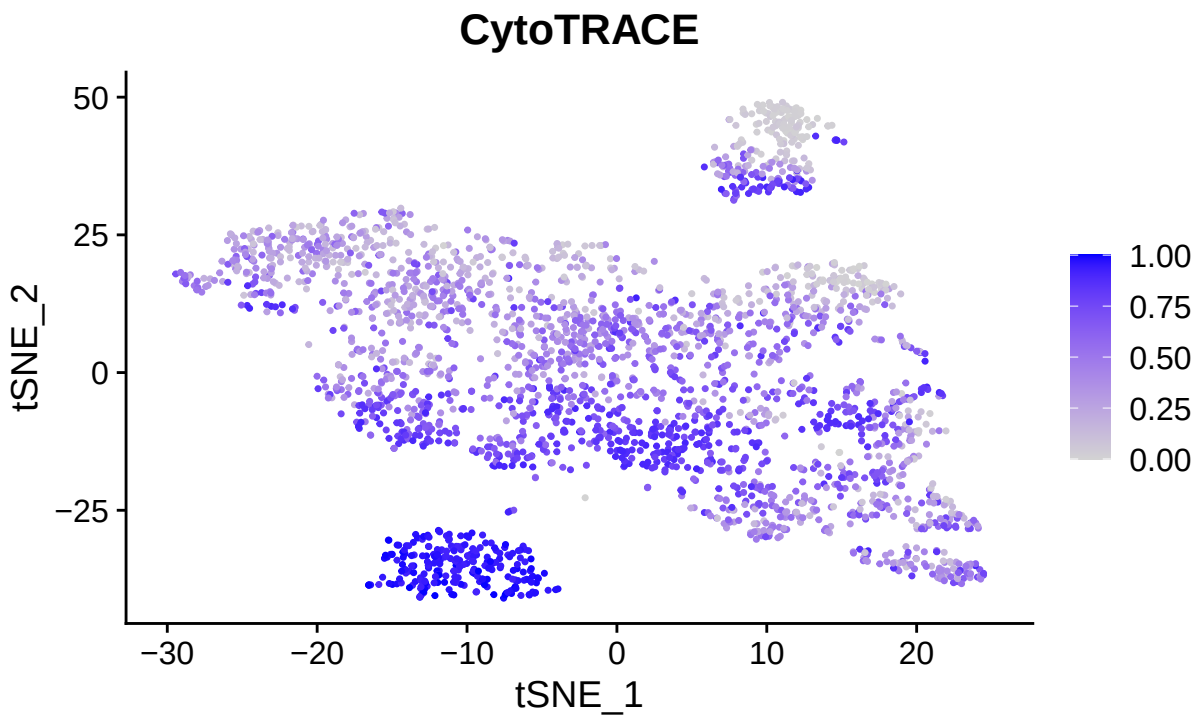
### 3.3.3 Analyzing the CytoTRACE results

For this exercise, we will add the results from CytoTRACE back into the Seurat object. This will allow us to use a number of the existing Seurat functions to analyze and visualize the CytoTRACE results.

```
sc_obj@meta.data$CytoTRACE <- cyto_results$CytoTRACE
```

**Visualize the CytoTRACE time variable on a tSNE plot**

```
FeaturePlot(sc_obj, reduction='tsne', features='CytoTRACE')
```



**Differential expression with respect to CytoTRACE time**

1. Remove genes with $< 25\%$ of cells expressing

```
# Get the count of cells expressing the gene.
# The code sum(x>0) is a shortcut to get a count of items greater than zero (0)
counts_sum = apply(sc_counts, 1, function(x) sum(x>0) )
# Get the genes in which the count is greater than 25% of the number of cells.
genes_keep = counts_sum > ( 0.25 * ncol(sc_counts) )
# Subset the gene counts data to just those genes.
genes_data = as.matrix(sc_obj@assays$RNA@data)[genes_keep,]
# Take a peek at the number of genes (rows) in the filtered data.
dim(genes_data)
```

```
[1] 1083 2791
```

2. Run a correlation analysis on the normalized expression levels and CytoTRACE time. For this example, we will use the Kendall's Tau test as it can better handle if there are ties in the data.

**NOTE** If you have a large dataset, Kendall's Tau test may not scale well and it may be better to use Spearman or Pearson correlation tests.

```
# Use an apply function to run the Kendall's Tau test (correlation) for each row.
# We transpose the results as the apply function will combine the output of each
# iteration column wise and we want to have it by row and save as a data.frame
cytotrace_corr <- data.frame(t(apply(genes_data, 1, function (x) {
  res <- cor.test(x, sc_obj@meta.data$CytoTRACE, method="kendall")
  return(c(res$estimate, res$p.value))
})))

# Set the column names
colnames(cytotrace_corr) <- c("estimate", "PValue")

# Add FDR corrected PValue
cytotrace_corr$QValue <- p.adjust(cytotrace_corr$PValue, method="BH")

# Sort by absolute value of the correlation estimate
cytotrace_corr <- cytotrace_corr[order(abs(cytotrace_corr$estimate), decreasing = T), ]

# Peek at the first few results
head(cytotrace_corr)
```

```
                  estimate        PValue        QValue
ENSG00000196154  0.4167914 6.208672e-230 6.723992e-227
ENSG00000163191  0.4057103 5.801650e-181 2.094396e-178
ENSG00000026025  0.3727315 3.409813e-188 1.846414e-185
ENSG00000100097  0.3725142 4.550332e-152 6.160013e-150
ENSG00000251562 -0.3608868 1.178563e-179 3.190959e-177
ENSG00000182718  0.3496557 7.936114e-130 7.813465e-128
```

***Differential time with respect to cluster or sample***

1. Compile the basic CytoTRACE time stats for each cluster and sample. In this example, we will compute the mean and standard deviation of the CytoTRACE time for each cluster in each sample.

```
library(dplyr)
sc_obj@meta.data %>%
  group_by(seurat_clusters, orig.ident) %>%
```

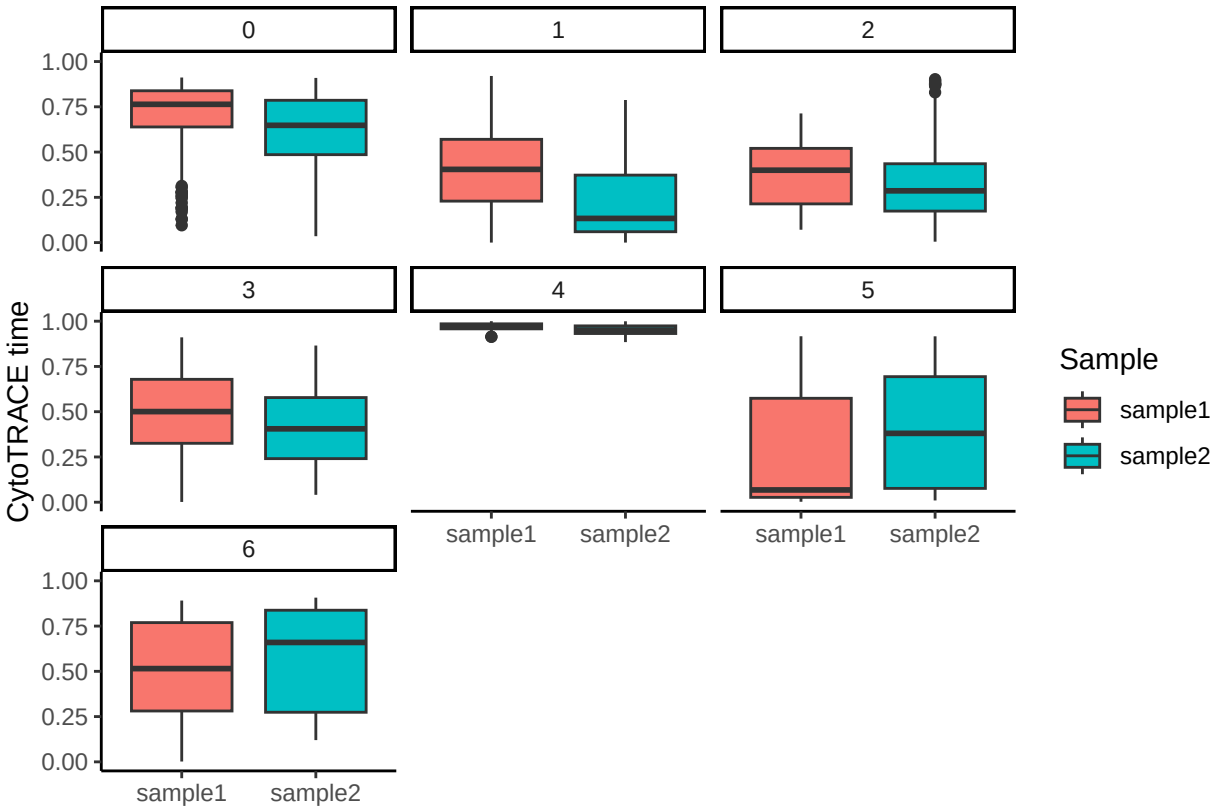
```
  summarize(mean=mean(CytoTRACE), sd=sd(CytoTRACE))
```

```
`summarise()` has grouped output by 'seurat_clusters'. You can override using
the `.groups` argument.
```

```
# A tibble: 14 x 4
# Groups:   seurat_clusters [7]
   seurat_clusters orig.ident  mean     sd
   <fct>           <chr>      <dbl>  <dbl>
 1 0               sample1    0.718 0.165
 2 0               sample2    0.613 0.206
 3 1               sample1    0.401 0.220
 4 1               sample2    0.216 0.216
 5 2               sample1    0.381 0.184
 6 2               sample2    0.325 0.191
 7 3               sample1    0.488 0.228
 8 3               sample2    0.416 0.208
 9 4               sample1    0.970 0.0217
10 4               sample2    0.953 0.0255
11 5               sample1    0.274 0.326
12 5               sample2    0.412 0.316
13 6               sample1    0.509 0.277
14 6               sample2    0.553 0.304
```

2. Generate a box plot of the CytoTRACE times for each cluster and sample

```
library(ggplot2)
ggplot(sc_obj@meta.data, aes(x=orig.ident, y= CytoTRACE, fill=orig.ident)) +
  theme_classic() +
  geom_boxplot() +
  facet_wrap(~ seurat_clusters) +
  labs(x="", y="CytoTRACE time", fill="Sample")
```




3. Perform statistical test(s) of CytoTRACE time as compared with cluster or sample.
   a. Kruskall-Wallis test of CytoTRACE time vs. cluster

```
kruskal.test(CytoTRACE ~ seurat_clusters, data = sc_obj@meta.data)
```

```
    Kruskal-Wallis rank sum test

data:  CytoTRACE by seurat_clusters
Kruskal-Wallis chi-squared = 1256, df = 6, p-value < 2.2e-16
```

   b. Compute differences between the samples for each cluster

```
sc_obj@meta.data %>% group_by(seurat_clusters) %>%
  do(w=wilcox.test(CytoTRACE ~ orig.ident, data=.)) %>%
  summarize(ClusterID=seurat_clusters, PValue=w$p.value)
```

```
# A tibble: 7 x 2
  ClusterID   PValue
  <fct>        <dbl>
1 0         7.14e-11
2 1         6.95e-10
3 2         4.15e- 2
4 3         8.74e- 4
5 4         3.67e- 5
6 5         1.05e- 3
7 6         5.62e- 1
```

## 3.4 Putative cell type annotation: Gene expression correlations

As our gold standard, we will use data from the Human Primary Cell Atlas (HPCA).

- We have prepared a table of log-scaled expression and cell type labels.
- These files are adapted from data available in the *celldex* BioConductor R package (https://bioconductor.org/packages/release/data/experiment/html/celldex.html).
- Notes:
  - We have 713 transcriptomes from 36 cell types.
  - We want to correlate average expression per cell type to average expression per cluster.
  - HPCA data identifiers are gene symbols, so we will need to convert identifiers.

1. Read in hpca data and labels

```
hpca.data <- read.table("https://wd.cri.uic.edu/sc_rna/hpca.logcounts.txt",
   row.names=1, header=T, sep="\t")
hpca.labels <- read.table("https://wd.cri.uic.edu/sc_rna/hpca.labels.txt",
   row.names=1, header=T, sep="\t")
```

2. Ensure columns in the data table are in the same order as the labels

```
hpca.data[1:5,1:5]
```

```
         GSM112490 GSM112491 GSM112540 GSM112541 GSM112661
A1BG        7.3651    6.9673    6.9751    7.0509    6.7073
A1BG-AS1    6.3622    6.0945    6.3750    6.3942    5.8412
A1CF        3.7219    3.6959    3.6518    3.6352    3.6508
A2M        10.1296   11.6344    9.6626    9.9476   11.4929
A2M-AS1     4.0154    3.7543    4.2488    4.1022    3.7049
```

```
head(hpca.labels)
```

```
           Cell.type
GSM112490         DC
GSM112491         DC
GSM112540         DC
GSM112541         DC
GSM112661         DC
GSM112664         DC
```

```
dim(hpca.data)
```

```
[1] 19363   713
```

3. Write a function to compute a "signature" for all samples/cells of a given type. we will use this for both the hpca cell types and the clusters.

```
compute_signatures <- function( norm, labels ){
  # get list of clusters
  unique_labels <- levels(as.factor(labels))
  # compute average expression for one label
  label_signature = function(label, normdata, all_labels){
    subset_data <- normdata[,all_labels==label,drop=F]
    signature <- rowSums(subset_data) / ncol(subset_data)
    return(signature)
  }
  # use sapply to compute for all labels
  signatures <- sapply(unique_labels, function(x) label_signature(x, norm, labels))
  colnames(signatures) <- unique_labels

  return(signatures)
}
# compute signatures for hpca data
hpca.signatures <- compute_signatures( hpca.data, hpca.labels[,1] )
hpca.signatures[1:5,1:5]
```

```
          Astrocyte   B_cell       BM BM & Prog. Chondrocytes
A1BG        7.41125 6.618992 6.896686     5.9231     7.215975
A1BG-AS1    5.05435 5.786815 5.578643     5.9708     5.151450
A1CF        3.62705 3.943215 3.789871     3.6468     3.764775
A2M        12.96550 4.603104 5.114671     4.3913     6.123663
A2M-AS1     4.62190 4.007696 5.015486     4.1059     4.131000
```

```
# compute signatures for clusters at 0.5 resolution
# NOTE: use as.matrix to convert sparse matrix to regular one
sc_norm <- as.matrix(sc_subset@assays$RNA@data)
sc_clusters <- sc_subset@meta.data$RNA_snn_res.0.5
cluster.signatures <- compute_signatures( sc_norm, sc_clusters )
cluster.signatures[1:5,1:5]
```

```
                         0 1           2 3           4
ENSG00000243485 0.000000000 0 0.000000000 0 0.000000000
ENSG00000237613 0.000000000 0 0.000000000 0 0.000000000
ENSG00000186092 0.000000000 0 0.000000000 0 0.000000000
ENSG00000238009 0.001973418 0 0.003153293 0 0.004975362
ENSG00000239945 0.000000000 0 0.000000000 0 0.000000000
```

We now have signatures for both clusters and cell types, but we need to match the identifiers. We will convert Ensembl IDs to gene symbols.

```
# convert ensembl IDs to gene symbols
names <- read.table("~/Downloads/V035_F031_subsample/features.tsv.gz",
   sep="\t", row.names=1)
```

```
# make sure to use all upper-case for the gene symbols
rownames(cluster.signatures) <- toupper(make.unique(names[rownames(cluster.signatures),1]))
# check how many match
common_genes <- intersect(rownames(hpca.signatures), rownames(cluster.signatures))
dim(hpca.signatures)
```

```
[1] 19363    36
```

```
dim(cluster.signatures)
```

```
[1] 32738     7
```

```
length(common_genes)
```

```
[1] 16826
```

```
# subset signatures to the common genes
hpca.signatures.common <- hpca.signatures[common_genes,]
cluster.signatures.common <- cluster.signatures[common_genes,]

# run a correlation between all clusters and all cell types
correlations <- cor(cluster.signatures.common, hpca.signatures.common)
dim(correlations)
```

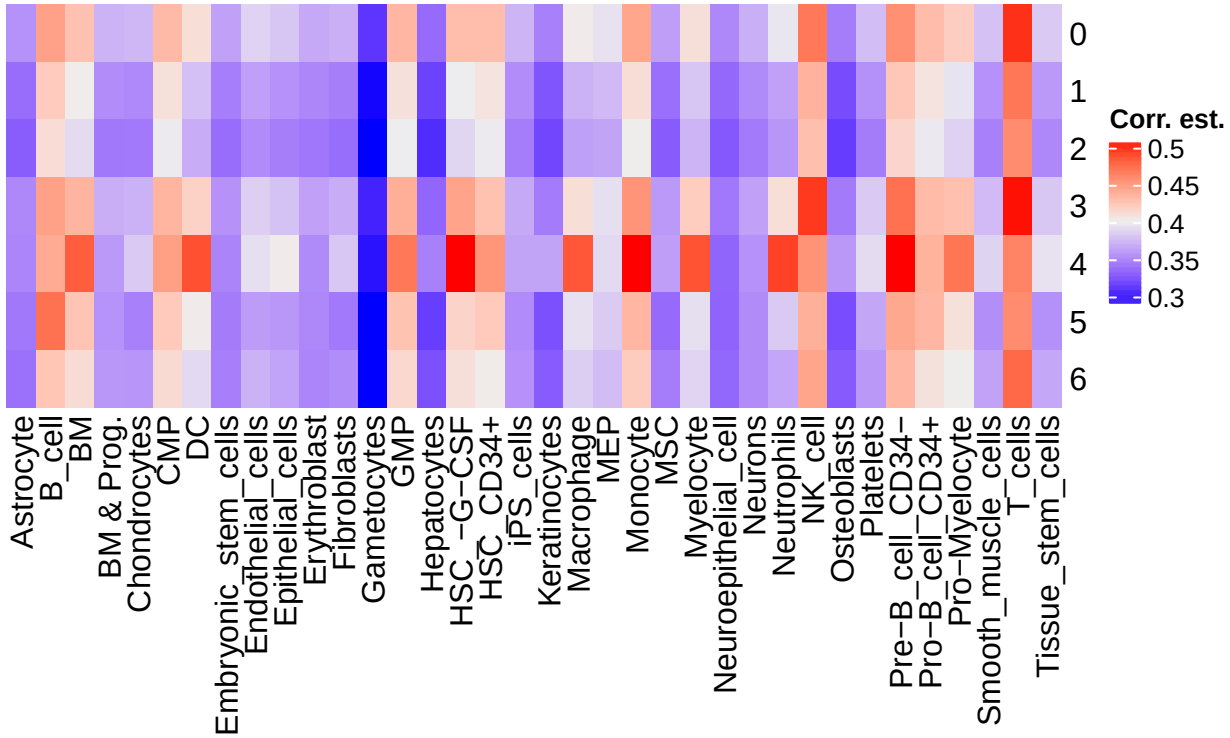
```
[1]  7 36
```

```
correlations[1:5,1:5]
```

```
  Astrocyte    B_cell        BM BM & Prog. Chondrocytes
0 0.3527475 0.4458172 0.4259881  0.3695857    0.3717114
1 0.3341783 0.4205231 0.4002821  0.3502637    0.3482544
2 0.3261378 0.4101080 0.3897959  0.3395300    0.3404228
3 0.3477782 0.4460850 0.4349242  0.3671684    0.3689765
4 0.3463954 0.4409660 0.4817135  0.3565039    0.3811244
```

Can plot a heatmap of the correlation matrix.

```
library(ComplexHeatmap)
Heatmap(correlations, name="Corr. est.", cluster_columns=F, cluster_rows=F)
```



You can examine the correlation matrix directly to see how closely different clusters correspond to each cell type, but it may be useful to find the best match for each cluster.

- To be thorough, we will also find the second-best match, and store the correlation coefficients for each so we can see how strong, and how unique, the best match is.

```
# make data frame to store closest and second closest match
cluster_list = colnames(cluster.signatures.common)
cluster_ids = data.frame(matrix(ncol=4,nrow=length(cluster_list)))
rownames(cluster_ids) = cluster_list
colnames(cluster_ids) = c("Best match","Best match score","Second best match",
  "Second best match score")
# loop over all clusters, find best and second best
for( i in 1:length(cluster_list) ){
  clust_scores = correlations[i,]
  clust_scores = clust_scores[order(clust_scores,decreasing=T)]
  cluster_ids[i,1] = names(clust_scores)[1]
  cluster_ids[i,2] = clust_scores[1]
  cluster_ids[i,3] = names(clust_scores)[2]
  cluster_ids[i,4] = clust_scores[2]
}
cluster_ids
```

```
  Best match Best match score Second best match Second best match score
0    T_cells        0.4986527           NK_cell                0.4685410
1    T_cells        0.4693142           NK_cell                0.4350938
2    T_cells        0.4581638           NK_cell                0.4279256
3    T_cells        0.5046618           NK_cell                0.4960100
4   Monocyte        0.5422486  Pre-B_cell_CD34-                0.5239103
5     B_cell        0.4721249           T_cells                0.4580311
6    T_cells        0.4764946           NK_cell                0.4439992
```

You could also save the above code in a function for reuse in other projects.

## 3.5 Diversity analyses of scRNAseq data

### 3.5.1 Alpha diversity analyses

Alpha diversity is a measure of diversity within an entity. In the case of scRNA seq data we will consider the cell the entity and we will measure the diversity of gene expression within each cell. We will then compare the gene expression diversity of the cells from different clusters to see if there is a higher "diversity" of gene expression in different clusters.

#### 3.5.1.1 Load the libraries and data (if necessary)

1. Load the `Seurat` (needed to manipulate the previously create Seurat object) and `vegan` for the diversity analysis

```
library(Seurat)
library(vegan)
```

```
Warning: package 'permute' was built under R version 4.1.3
```

```
Warning: package 'lattice' was built under R version 4.1.3
```

2. Load the previously analyzed Seurat object, if needed.

```
sc_subset <- readRDS("sc_subset.rds")
```

#### 3.5.1.2 Perform the diversity analysis

1. Compute the diversity index for each cell. In this instance we will use the "Shannon" index as a measure of diversity. We will save the diversity indices with the cluster assignment for each cell in a data.frame. For the `diversity` function we need to set `MARGIN=2` to tell the function to compute the diversity index for each column (cells in the `data` slot). The default is to compute on each row.

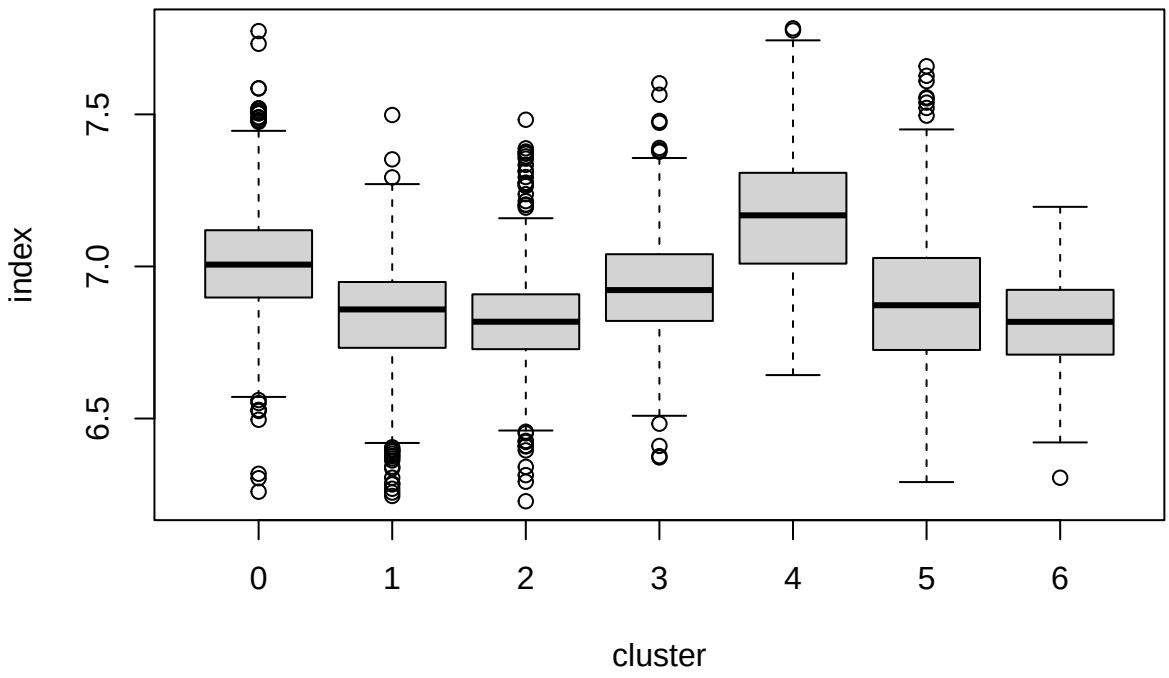
```
alpha_div <- data.frame(cluster=sc_subset$seurat_clusters,
                        index=diversity(as.matrix(sc_subset@assays$RNA@data), MARGIN=2))
```

2. Use `dplyr` to compute the average diversity for each cluster

```
library(dplyr)

alpha_div_cluster <- alpha_div %>% group_by(cluster) %>%
                        summarize(mean=mean(index),
                                  sd=sd(index))
```

3. Generate a box plot of the diversity indices as a function of cluster

```
boxplot(index ~ cluster, data=alpha_div)
```




4. Compute pairwise comparisons of diversity between each cluster. **NOTE**: Due to the often large number of cells in each cluster, it may be better to utilize other statistical tests to determine if the diversity of gene expression is different between the clusters.

```
# Setup the variables to hold the results
cluster_a <- character()
cluster_b <- character()
PValue <- numeric()

# Get the list of clusters to compare
clusters <- alpha_div_cluster$cluster

# Iterate over all possible pairwise comparisons
# and test for differences using Wilcox/Mann-Whitney test
for ( i in 1:(length(clusters) - 1) ) {
  cluster_a_values <- alpha_div$index[ alpha_div$cluster == clusters[i]]
  for ( j in (i+1):length(clusters) ) {
    cluster_b_values <- alpha_div$index[ alpha_div$cluster == clusters[j]]
    cluster_a <- c(cluster_a, clusters[i])
    cluster_b <- c(cluster_b, clusters[j])
    mw_res <- wilcox.test(cluster_a_values, cluster_b_values)
    PValue <- c(PValue, mw_res$p.value)
  }
}

# Create a data.frame from the results
alpha_div_results <- data.frame(ClusterA=cluster_a,
```

```
                                ClusterB=cluster_b,
                                PValue=PValue,
                                QValue=p.adjust(PValue, method="BH"))
alpha_div_results
```

```
   ClusterA ClusterB       PValue       QValue
1         1        2 2.285629e-57 1.199955e-56
2         1        3 6.133542e-71 1.288044e-69
3         1        4 1.367407e-12 2.208889e-12
4         1        5 2.115186e-17 4.441890e-17
5         1        6 6.634908e-15 1.266664e-14
6         1        7 4.410275e-26 1.157697e-25
7         2        3 5.119676e-03 5.972955e-03
8         2        4 1.046020e-13 1.830534e-13
9         2        5 6.967394e-64 4.877176e-63
10        2        6 8.092890e-03 8.944773e-03
11        2        7 1.792077e-01 1.881681e-01
12        3        4 1.296798e-22 3.025863e-22
13        3        5 2.639327e-66 2.771293e-65
14        3        6 1.418734e-04 1.986227e-04
15        3        7 9.896245e-01 9.896245e-01
16        4        5 7.171902e-32 2.510166e-31
17        4        6 4.819145e-03 5.972955e-03
18        4        7 8.598982e-10 1.289847e-09
19        5        6 3.787473e-27 1.136242e-26
20        5        7 7.810219e-36 3.280292e-35
21        6        7 4.851899e-03 5.972955e-03
```

### 3.5.2 Beta-diversity analyses

Beta diversity is a measure of dissimilarity between different entities. In the case of single cell RNAseq we will consider each cell to be an entity and compare the overall gene expression for each cell. From there, we will utilize the ANOSIM R statistic to give a sense of how dissimilar the clusters are from each other.

#### 3.5.2.1 Load the libraries and data (if necessary)

1. Load the `Seurat` (needed to manipulate the previously create Seurat object) and `vegan` for the diversity analysis

```
library(Seurat)
library(vegan)
```

2. Load the previously analyzed Seurat object, if needed.

```
sc_subset <- readRDS("sc_subset.rds")
```

#### 3.5.2.2 Perform the diversity analysis

1. Compute the beta diversity (dissimilarity) indices. For this example, we will use a Euclidean distance on the scaled data of the variable features. We will need to transpose the scaled data matrix as the `vegdist` function will compute the dissimilarity indices on the rows of the matrix.

**NOTE**: This can take a long time depending on the number of variable genes selected. Instead you may want to try computing the Euclidean distance on the average expression of the cluster rather than the individuals cells.

```
bd_dist <- vegdist(t(as.matrix(sc_subset@assays$RNA@scale.data)),
                   method = "euclidean")
```

2. Use the ANOSIM test to compute the `R`, i.e. measure of separation, between the clusters. **NOTE**: This can also take a long time depending on the number of cells in each cluster.

```
# Setup the variables to hold the results
cluster_a <- character()
cluster_b <- character()
RValue <- numeric()

# Get the list of clusters to compare
clusters <- unique(sc_subset$seurat_clusters)

# Iterate over all possible pairwise comparisons
# and test for differences using ANOSIM test
for ( i in 1:(length(clusters) - 1) ) {
  cluster_a_cells <- which(sc_subset$seurat_clusters == clusters[i])
  for ( j in (i+1):length(clusters) ) {
    cluster_b_cells <- which(sc_subset$seurat_clusters == clusters[j])
    these_cells <- c(cluster_a_cells, cluster_b_cells)
    this_dist <- as.matrix(bd_dist)[these_cells, these_cells]
    anosim_res <- anosim(this_dist, sc_subset$seurat_clusters[these_cells])
    cluster_a <- c(cluster_a, clusters[i])
    cluster_b <- c(cluster_b, clusters[j])
    RValue <- c(RValue, anosim_res$statistic)
  }
}

# Create a data.frame from the results
```

```
beta_div_results <- data.frame(ClusterA=cluster_a, ClusterB=cluster_b,
                               RValue=RValue)
beta_div_results
```

```
   ClusterA ClusterB      RValue
1         6        1  0.74742937
2         6        2  0.76075476
3         6        7 -0.12520250
4         6        4  0.31345898
5         6        5  0.43552517
6         6        3  0.72675274
7         1        2  0.03987599
8         1        7  0.06126019
9         1        4  0.34167159
10        1        5  0.77760817
11        1        3  0.02996233
12        2        7  0.33884749
13        2        4  0.43284327
14        2        5  0.74993412
15        2        3  0.02569004
16        7        4 -0.24773243
17        7        5 -0.28736012
18        7        3  0.42577682
19        4        5  0.57039388
20        4        3  0.39404293
21        5        3  0.70468212
```

#### 3.5.2.3 Generate a heatmap of R values

1. Convert the data frame to a matrix. We will use `reshape2` to do this, but can use `tidyr` or other methods

```
library(reshape2)
r_matrix <- dcast(rbind(beta_div_results,
                        data.frame(ClusterA=beta_div_results$ClusterB,
                                   ClusterB=beta_div_results$ClusterA,
                                   RValue=beta_div_results$RValue)),
                  ClusterA ~ ClusterB,
                  value.var="RValue",
                  fill=0)
```

2. Set the row names to be the clusters. Then drop the first column (cluster IDs) and convert to a matrix.

```
row.names(r_matrix) <- r_matrix[,1]
r_matrix <- as.matrix(r_matrix[,-1])
```

3. Any values that are less than zero (0) can be considered zero

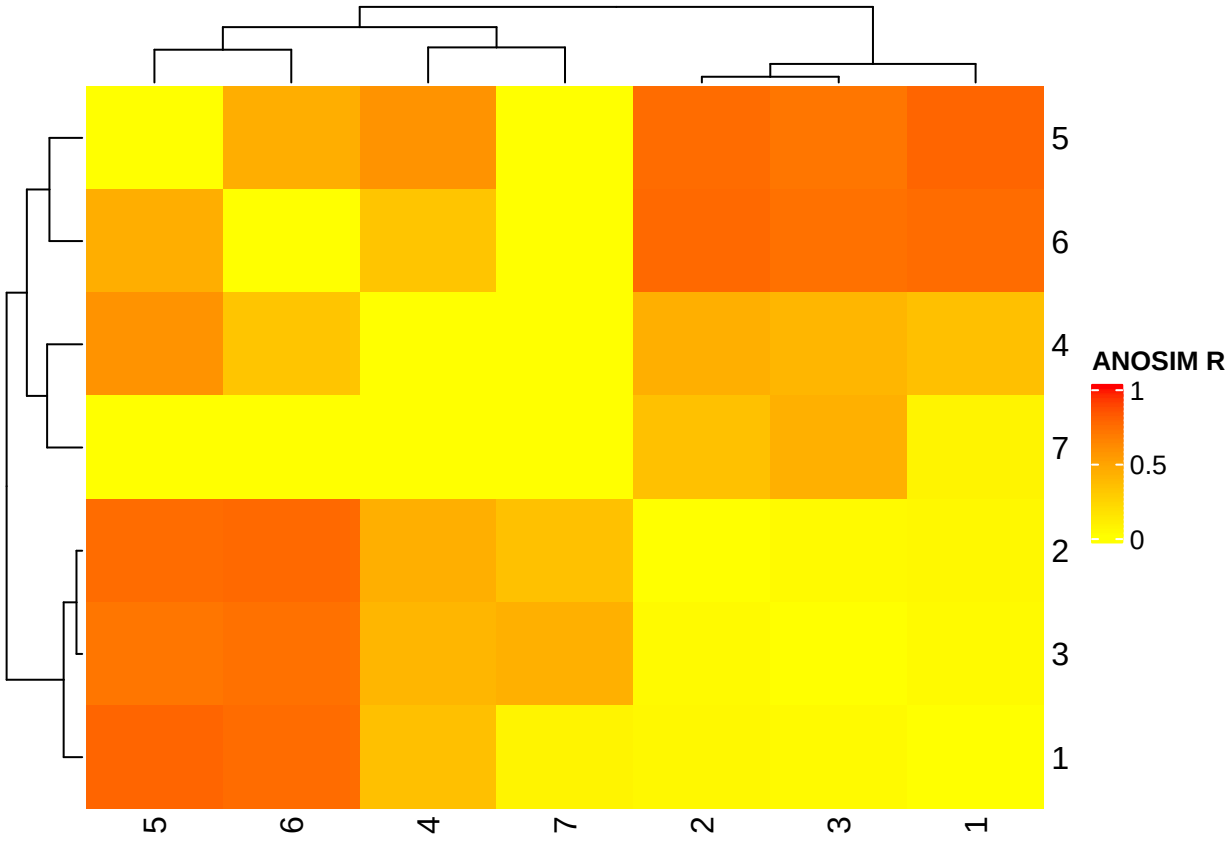
```
r_matrix[r_matrix < 0] <- 0
```

4. Load library `ComplexHeatmap`

```
library(ComplexHeatmap)
library(circlize)
```

4. Generate the heatmap

```
Heatmap(r_matrix,
        name="ANOSIM R",
        col=colorRamp2(c(1,0), colors=c("red", "yellow")))
```



## 3.6 Incorporate TCR data into Seurat object

### 3.6.1 Convert and load the TCR data into the Seurat object

NOTE: The first five (5) steps are the same as in exercise 2.5.1.

1. Load the contig annotations into R.

```
sc_vdj_contigs <- read.csv("https://wd.cri.uic.edu/sc_rna/tcr_contigs.csv")
```

2. Clean up the barcode IDs. Seurat will strip the trailing numbers from the cell barcodes, if present.

```
sc_vdj_contigs$barcode <- sub('-[0-9]+$', '', sc_vdj_contigs$barcode)
```

3. Compute the chain counts for each cell.

```
# First get a list of the detected chains
table(sc_vdj_contigs$chain)
```

```
Multi   TRA   TRB
    6   578   614
```

```
# Then compute the chain counts per cell
vdj_chain_counts <- as.data.frame(table(sc_vdj_contigs[, c("barcode", "chain")]))
```

4. Add presence/absence for the chains.

```
vdj_chain_counts$presence <- vdj_chain_counts$Freq > 0

# Take a peek at the results
head(vdj_chain_counts)
```

```
           barcode chain Freq presence
1 AAACGGGTCGCTGATA Multi    0    FALSE
2 AAAGCAAAGTATGACA Multi    0    FALSE
3 AAATGCCCACTTAAGC Multi    0    FALSE
4 AACACGTCAACAACCT Multi    0    FALSE
5 AACACGTGTTATCCGA Multi    0    FALSE
6 AACACGTGTTATCGGT Multi    0    FALSE
```

5. Get the cluster IDs for each cell as a data.frame.

```
cluster_ids <- data.frame(cluster=Idents(sc_w_fbc))

# Take a peek at the results
head(cluster_ids)
```

```
                cluster
AAACCTGCAGCCTGTG       6
AAACGGGTCGCTGATA       2
AAAGCAAAGTATGACA       2
AAATGCCCACTTAAGC       0
AACACGTCAACAACCT       0
AACACGTCAAGCGATG       1
```

6. Convert the chain counts into "wide" format. Will we use the `dcast` function in the `reshape2` library.

```
library(reshape2)
```

```
vdj_cell_chains <- dcast(vdj_chain_counts[, c("barcode", "chain", "Freq")],
                         barcode ~ chain)
```

```
Using Freq as value column: use value.var to override.
```

```
dim(vdj_cell_chains)
```

```
[1] 482   4
```

```
head(vdj_cell_chains)
```

```
           barcode Multi TRA TRB
1 AAACGGGTCGCTGATA     0   1   1
2 AAAGCAAAGTATGACA     0   1   1
3 AAATGCCCACTTAAGC     0   1   1
4 AACACGTCAACAACCT     0   2   3
5 AACACGTGTTATCCGA     0   2   1
6 AACACGTGTTATCGGT     0   1   1
```

7. Merge with the cell cluster IDs. This is primarily to add rows for all the cells.

```
vdj_cell_chains <- merge(cluster_ids, vdj_cell_chains,
                         by.x=0, by.y=1, all.x=T)

dim(vdj_cell_chains)
```

```
[1] 805   5
```

8. Set the row names as the cell IDs (barcodes).

```
row.names(vdj_cell_chains) <- vdj_cell_chains[,1]
```

9. Drop the first two columns, i.e. barcode and cluster ID, and convert to matrix

```
vdj_cell_chains <- as.matrix(vdj_cell_chains[, c(-1, -2)])

head(vdj_cell_chains)
```

```
                 Multi TRA TRB
AAACCTGCAGCCTGTG    NA  NA  NA
AAACGGGTCGCTGATA     0   1   1
AAAGCAAAGTATGACA     0   1   1
AAATGCCCACTTAAGC     0   1   1
AACACGTCAACAACCT     0   2   3
AACACGTCAAGCGATG    NA  NA  NA
```

10. Set any `NA` values to be zero (no chains)

```
vdj_cell_chains[is.na(vdj_cell_chains)] <- 0
```

11. Add the data to the Seurat object.

*NOTE*: the `CreateAssayObject` expects a counts table with features (chains) on the rows and cell on the columns. So, we will need to transpose the matrix when we create the new assay object.

```
sc_w_fbc[["TCR"]] <- CreateAssayObject(counts=t(vdj_cell_chains))
```

12. (*OPTIONAL*) Save the modified Seurat object to an `.rds` file for repeated use.

```
saveRDS(sc_w_fbc, file="seurat_obj_w_tcr.rds")
```

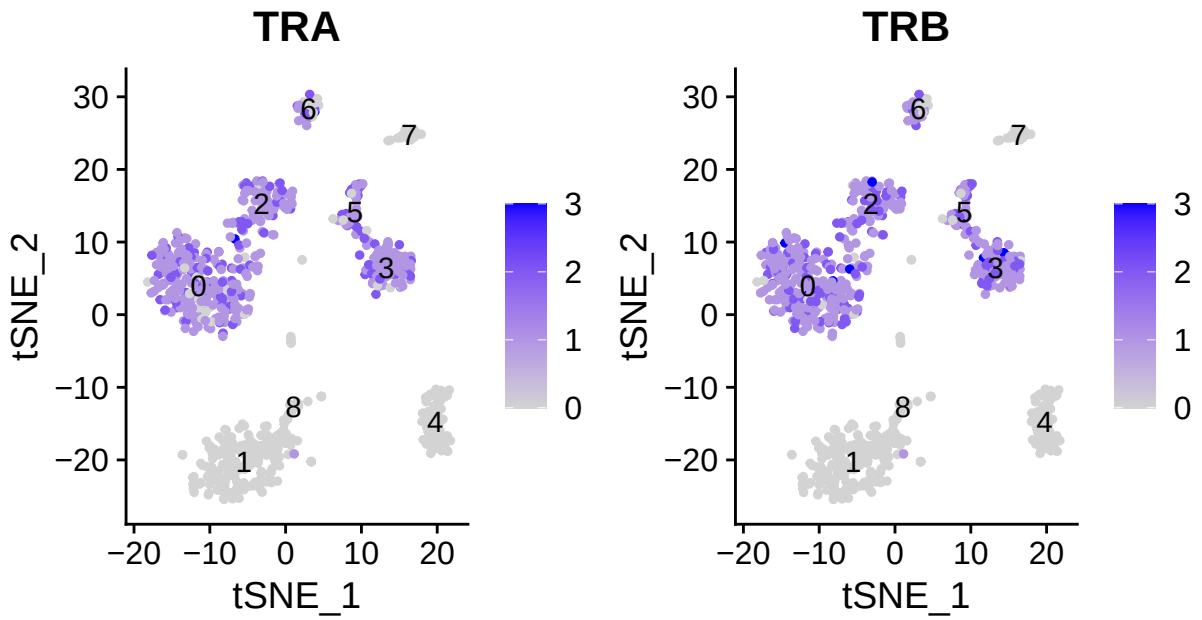
### 3.6.2 Create TCR feature plot in Seurat.

1. Select the TCR assay as the default assay for plotting.

```
DefaultAssay(sc_w_fbc) <- "TCR"
```

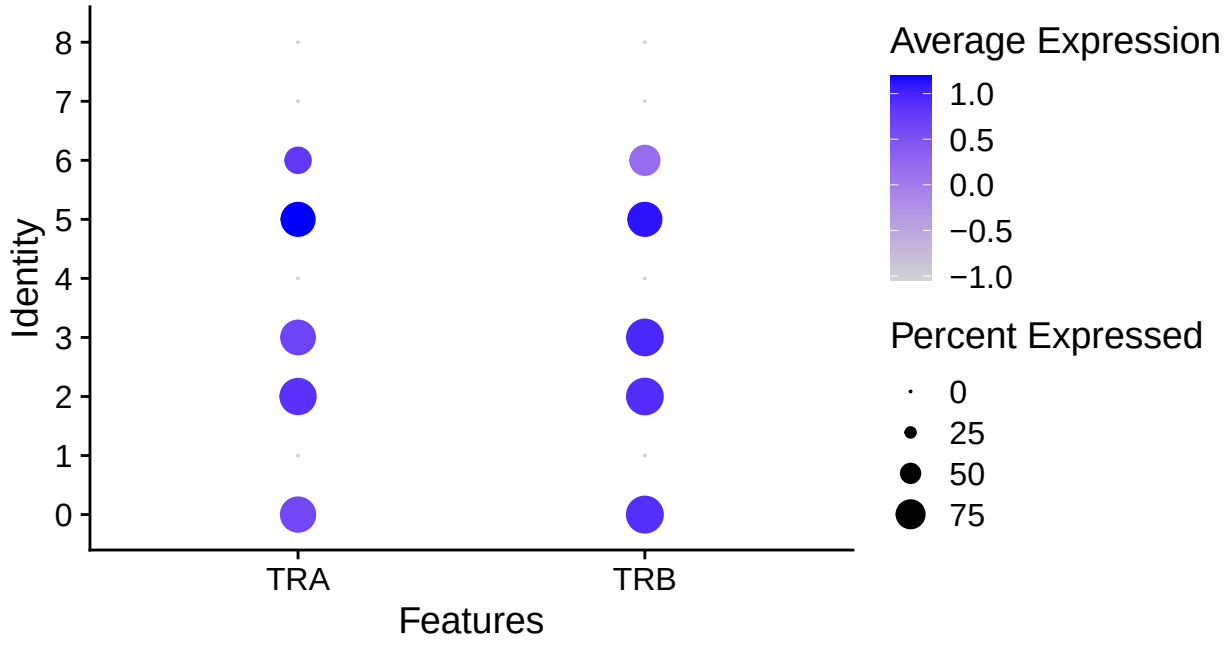
2. Create a tSNE feature plot using the TRA and TRB chain counts.

```
FeaturePlot(sc_w_fbc, features = c("TRA", "TRB"), label=T)
```



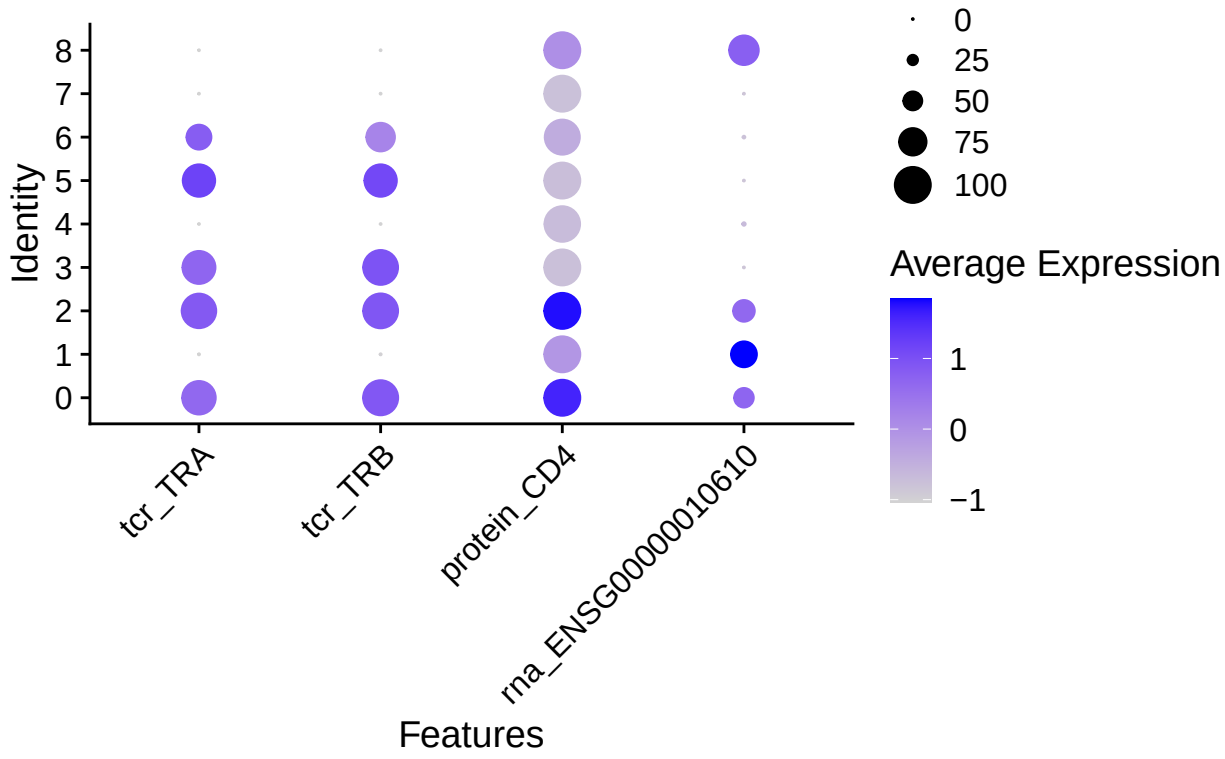

3. Create a Seurat dot plot using the TRA and TRB chain counts.

```
DotPlot(sc_w_fbc, features=c("TRA", "TRB"))
```



4. Can also make plot using data from multiple assays. Just add a prefix of the assay name in lowercase.

```
DotPlot(sc_w_fbc, features=c("tcr_TRA", "tcr_TRB", "protein_CD4", "rna_ENSG00000010610")) +
  RotatedAxis()
```



## 3.7 Diversity analysis of V(D)J data (EXAMPLE ONLY)

This example will use the library `vegan` to compute diversity of V(D)J+C annotations as compared with different biological samples.

**NOTE**: A non-publically available dataset was used for this example as the previous V(D)J dataset contained only a single sample. ***Thus, this is "exercise" is demonstration only.***

The `vegan` package is available on CRAN and can be installed via `install.packages`.

```
install.packages('vegan')
```

1. If needed, load the Seurat object.

```
sc_obj <- readRDS("sc_obj.rds")
```

2. Load the contig annotations into R.

```
sc_vdj_contigs <- read.delim("filtered_contig_annotations.txt")
```

3. Clean up the barcode IDs. Seurat will strip the trailing numbers from the cell barcodes, if present.

```
sc_vdj_contigs$barcode <- sub('-[0-9]+$', '', sc_vdj_contigs$barcode)
```

4. Get the cluster IDs and original sample IDs for each cell as a data.frame.

```
cluster_ids <- data.frame(cluster=Idents(sc_obj), sample=sc_obj$orig.ident)
head(cluster_ids)
```

```
                          cluster     sample
Sample_001_AAACCTGAGCTGTCTA      14 Sample_001
Sample_001_AAACCTGCAAAGGAAG      29 Sample_001
Sample_001_AAACCTGGTGAGGGTT       9 Sample_001
Sample_001_AAACGGGGTGAACCTT      12 Sample_001
Sample_001_AAACGGGGTTGATTCG       8 Sample_001
Sample_001_AAACGGGTCCTATTCA      13 Sample_001
```

5. Get the selected V(D)J data from the loaded contig data. The following are number of options to extract V(D)J data.
   - Get all CDR3 amino acid sequences

```
vdj_data <- subset(sc_vdj_contigs, select=c(barcode, cdr3))
head(vdj_data)
```

```
                    barcode            cdr3
1 Sample_001_AAACCTGAGCTGTCTA CASSDLGAGTGQLYF
2 Sample_001_AAACCTGAGCTGTCTA CAVRDPPGNTRKLIF
3 Sample_001_AAACCTGGTGAGGGTT  CASRSGTGDYEQYF
4 Sample_001_AACGTTGAGTAGCCGA    CALSESGGKLTL
5 Sample_001_AACGTTGAGTAGCCGA CASSLKTGGYAEQFF
6 Sample_001_AACTTTCAGGGAAACA CASSLKTGGYAEQFF
```

   - Get all CDR3 amino acid sequences for TRA chains

```
vdj_data <- subset(sc_vdj_contigs, chain=="TRA", select=c(barcode, cdr3))
head(vdj_data)
```

```
                     barcode            cdr3
2  Sample_001_AAACCTGAGCTGTCTA CAVRDPPGNTRKLIF
4  Sample_001_AACGTTGAGTAGCCGA    CALSESGGKLTL
7  Sample_001_AACTTTCAGGGAAACA    CALSESGGKLTL
9  Sample_001_AACTTTCAGTTACCCA CALSDLDYSNNRLTL
```

```
11 Sample_001_AACTTTCCATTCTTAC CAASMRGSALGRLHF
14 Sample_001_AAGGAGCGTAAACGCG   CAVRASSGQKLVF
```

- Get combined V(D)J+C annotations.

```
vdj_data <- data.frame(barcode=sc_vdj_contigs$barcode,
        vdj_annotation=paste(sc_vdj_contigs$v_gene, sc_vdj_contigs$d_gene,
            sc_vdj_contigs$j_gene, sc_vdj_contigs$c_gene, sep="."))
head(vdj_data)
```

```
                      barcode             vdj_annotation
1 Sample_001_AAACCTGAGCTGTCTA    TRBV13-1..TRBJ2-2.TRBC2
2 Sample_001_AAACCTGAGCTGTCTA         TRAV1..TRAJ37.TRAC
3 Sample_001_AAACCTGGTGAGGGTT TRBV19.TRBD1.TRBJ2-7.TRBC2
4 Sample_001_AACGTTGAGTAGCCGA      TRAV6N-7..TRAJ44.TRAC
5 Sample_001_AACGTTGAGTAGCCGA      TRBV29..TRBJ2-1.TRBC2
6 Sample_001_AACTTTCAGGGAAACA      TRBV29..TRBJ2-1.TRBC2
```

- Get combined V(D)J+C annotations for TRA chains.

```
tra_data <- subset(sc_vdj_contigs, chain=="TRA")
vdj_data <- data.frame(barcode=tra_data$barcode,
        vdj_annotation=paste(tra_data$v_gene, tra_data$d_gene,
            tra_data$j_gene, tra_data$c_gene, sep="."))
head(vdj_data)
```

```
                      barcode        vdj_annotation
1 Sample_001_AAACCTGAGCTGTCTA    TRAV1..TRAJ37.TRAC
2 Sample_001_AACGTTGAGTAGCCGA TRAV6N-7..TRAJ44.TRAC
3 Sample_001_AACTTTCAGGGAAACA TRAV6N-7..TRAJ44.TRAC
4 Sample_001_AACTTTCAGTTACCCA   TRAV6-6..TRAJ7.TRAC
5 Sample_001_AACTTTCCATTCTTAC  TRAV9-1..TRAJ18.TRAC
6 Sample_001_AAGGAGCGTAAACGCG    TRAV1..TRAJ16.TRAC
```

6. Merge the cluster and sample information with V(D)J data. You could use any of the above subsets of the V(D)J data for this analysis. For this example we will use the previous example of V(D)J+C annotations for all chains.

```
vdj_data <- merge(cluster_ids, vdj_data, by.x=0, by.y=1)

head(vdj_data)
```

```
                   Row.names cluster     sample             vdj_annotation
1 Sample_001_AAACCTGAGCTGTCTA      14 Sample_001    TRBV13-1..TRBJ2-2.TRBC2
2 Sample_001_AAACCTGAGCTGTCTA      14 Sample_001         TRAV1..TRAJ37.TRAC
3 Sample_001_AAACCTGGTGAGGGTT       9 Sample_001 TRBV19.TRBD1.TRBJ2-7.TRBC2
4 Sample_001_AACGTTGAGTAGCCGA       4 Sample_001      TRAV6N-7..TRAJ44.TRAC
5 Sample_001_AACGTTGAGTAGCCGA       4 Sample_001      TRBV29..TRBJ2-1.TRBC2
6 Sample_001_AACTTTCAGGGAAACA       4 Sample_001      TRAV6N-7..TRAJ44.TRAC
```

7. Compute counts for each annotation and sample. For this example, we are using the V(D)J+C annotations data.

```
library(dplyr)

vdj_counts <- vdj_data %>% group_by(sample, vdj_annotation) %>% summarize(counts=n())
```

```
`summarise()` has grouped output by 'sample'. You can override using the
`.groups` argument.
```

```
head(vdj_counts)
```

```
# A tibble: 6 x 3
# Groups:   sample [1]
  sample     vdj_annotation       counts
  <chr>      <chr>                 <int>
1 Sample_001 TRAV1..TRAJ16.TRAC        3
2 Sample_001 TRAV1..TRAJ37.TRAC        1
3 Sample_001 TRAV10D..TRAJ26.TRAC      1
4 Sample_001 TRAV10D..TRAJ37.TRAC      1
5 Sample_001 TRAV10D..TRAJ45.TRAC      1
6 Sample_001 TRAV10N..TRAJ7.TRAC       1
```

8. Convert counts data.frame to a matrix with samples on the rows and V(D)J+C annotations on columns.

```
library(reshape2)

vdj_counts_mat <- dcast(vdj_counts, sample ~ vdj_annotation, value.var="counts", fill=0)

head(vdj_counts_mat)[,1:4]
```

```
      sample TRAV1..TRAJ12.TRAC TRAV1..TRAJ16.TRAC TRAV1..TRAJ17.TRAC
1 Sample_001                  0                  3                  0
2 Sample_002                  0                  0                  0
3 Sample_003                  0                  0                  0
4 Sample_004                  0                  0                  0
5 Sample_005                  0                  0                  0
6 Sample_006                  0                  0                  0
```

```
# The first column has the sample IDs.
# So, set as the row names and drop the first column when converting to matrix.
row.names(vdj_counts_mat) <- vdj_counts_mat[,1]
vdj_counts_mat <- as.matrix(vdj_counts_mat[,-1, drop=F])

head(vdj_counts_mat)[,1:3]
```

```
           TRAV1..TRAJ12.TRAC TRAV1..TRAJ16.TRAC TRAV1..TRAJ17.TRAC
Sample_001                  0                  3                  0
Sample_002                  0                  0                  0
Sample_003                  0                  0                  0
Sample_004                  0                  0                  0
Sample_005                  0                  0                  0
Sample_006                  0                  0                  0
```

9. Compute the diversity indices for each sample. Will compute Shannon entropy (S) and richness, a.k.a. number of unique entities (N).

```
library(vegan)
vdj_diversity <- data.frame(sample=row.names(vdj_counts_mat),
            S=diversity(vdj_counts_mat, index="shannon"),
            N=specnumber(vdj_counts_mat))

head(vdj_diversity)
```

```
              sample        S   N
Sample_001 Sample_001 4.640368 156
Sample_002 Sample_002 4.087640  90
```

```
Sample_003 Sample_003 5.001865 220
Sample_004 Sample_004 4.956640 292
Sample_005 Sample_005 5.119334 299
Sample_006 Sample_006 4.551837 199
```

Once the diversity indices are generated these can be compared with the experimental factors for your biological samples using basic statistical tests, e.g. Kruskal-Wallis or Mann-Whitney. The diversity indices could also be plotted using dot/box plots as a function of particular experimental factors.

### 3.7.1 Comparing and visualization of diversity indices.

For these example we used the following mapping information

```
mapping <- read.delim("mapping.txt")

head(mapping)
```

```
      Sample Group
1 Sample_001     A
2 Sample_002     B
3 Sample_003     D
4 Sample_004     C
5 Sample_005     C
6 Sample_006     C
```

- Merge the mapping with the compute diversity indices.

```
vdj_diversity <- merge(vdj_diversity, mapping, by.x=1, by.y=1)

# Take a peek at the results
head(vdj_diversity)
```

```
      sample        S   N Group
1 Sample_001 4.640368 156     A
2 Sample_002 4.087640  90     B
3 Sample_003 5.001865 220     D
4 Sample_004 4.956640 292     C
5 Sample_005 5.119334 299     C
6 Sample_006 4.551837 199     C
```

- Perform statistical comparison using Kruskal-Wallis test

```
# Comparing difference in computed Shannon entropy (S) with group
kruskal.test(S ~ Group, vdj_diversity)
```

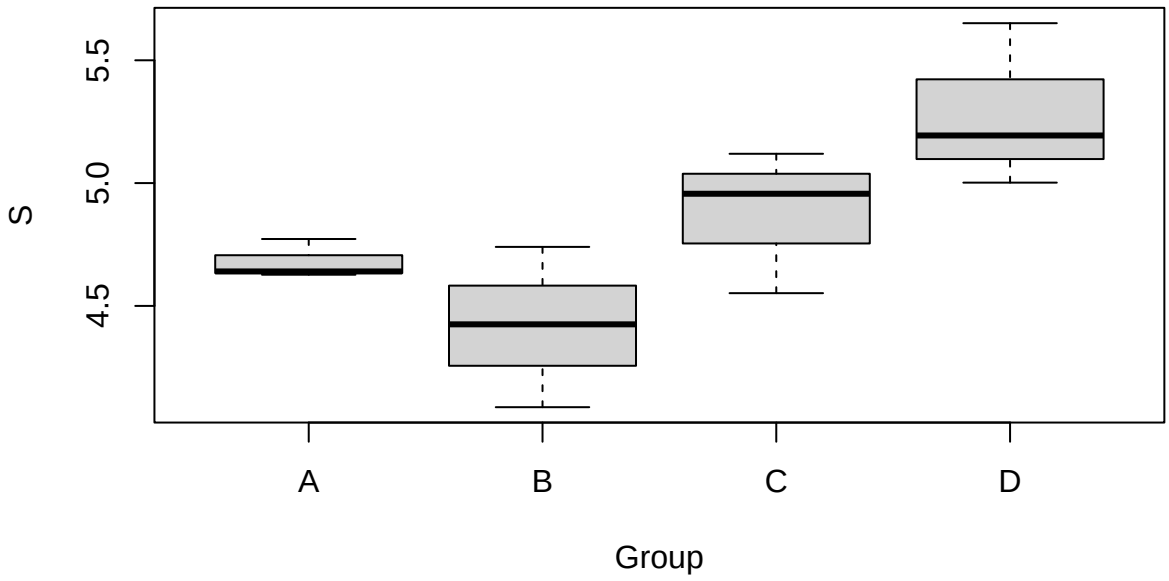
```

    Kruskal-Wallis rank sum test

data:  S by Group
Kruskal-Wallis chi-squared = 7.2051, df = 3, p-value = 0.06564
```

```
# Comparing difference in richness (N) with group
kruskal.test(N ~ Group, vdj_diversity)
```

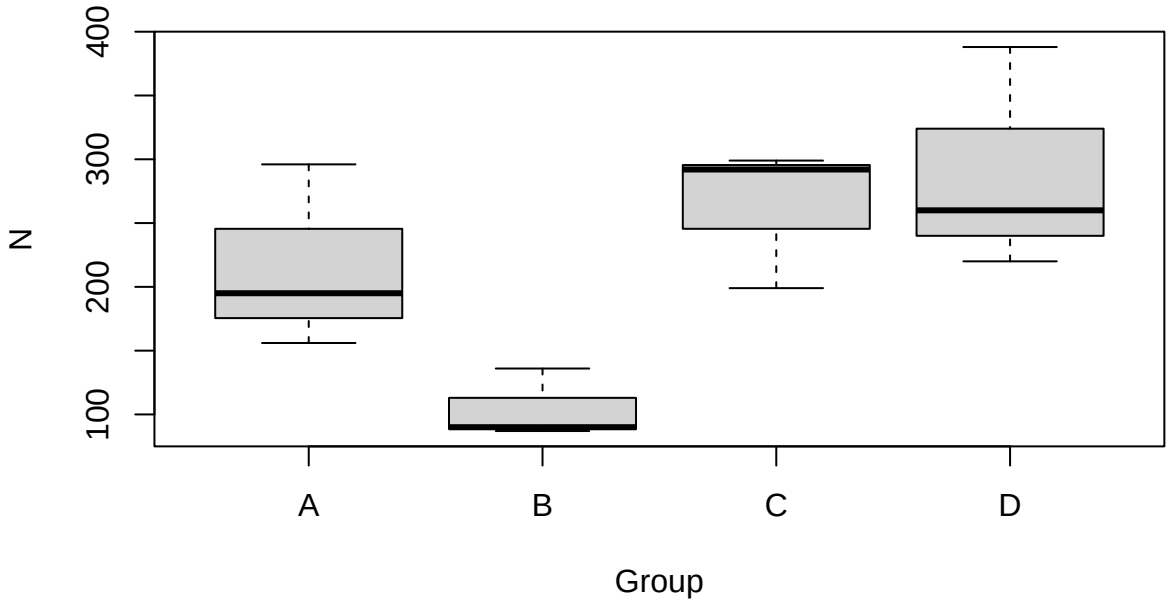
```

    Kruskal-Wallis rank sum test

data:  N by Group
Kruskal-Wallis chi-squared = 7.2051, df = 3, p-value = 0.06564
```

- Visualization of diversity indices

```
boxplot(S ~ Group, vdj_diversity)
```

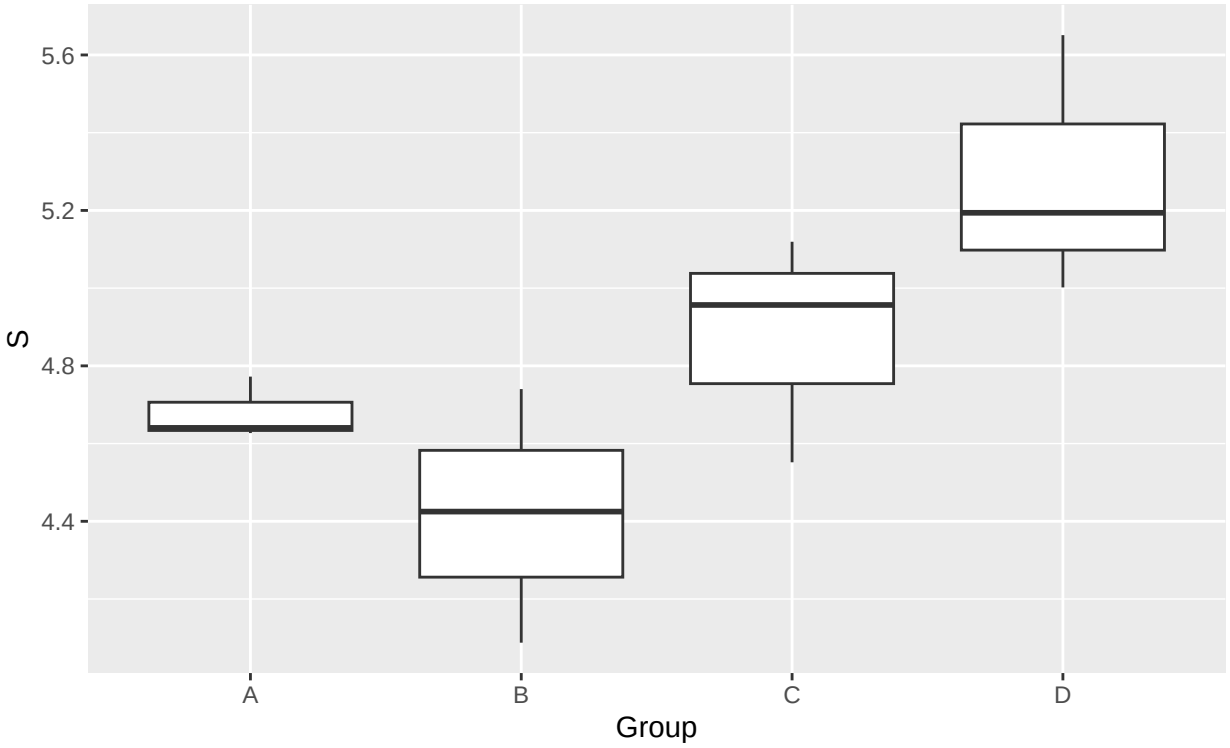



```
boxplot(N ~ Group, vdj_diversity)
```

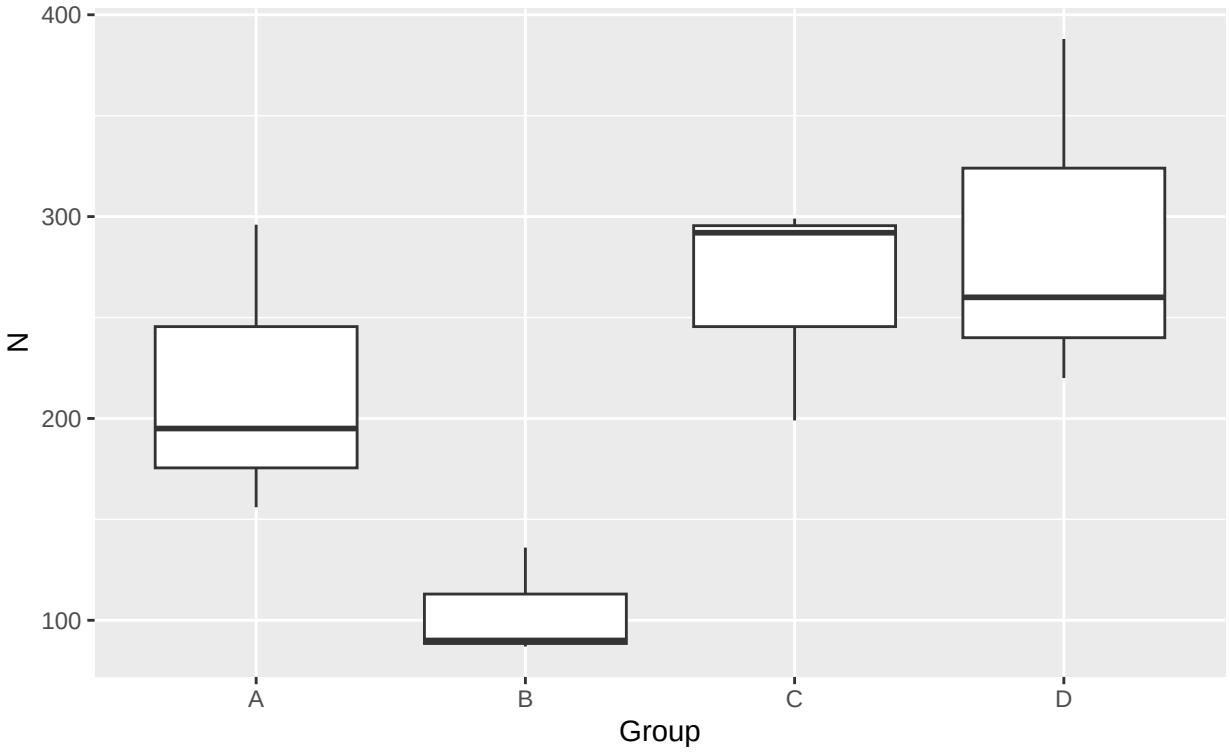


- Using `ggplot2`

```
library(ggplot2)

ggplot(vdj_diversity, aes(x=Group, y=S)) + geom_boxplot()
```




```
ggplot(vdj_diversity, aes(x=Group, y=N)) + geom_boxplot()
```




**NOTE**: It may be important to subsample/rarefy the counts data to account for any diversity indices that may be due to the fact that some samples had higher cell counts than others. The following is an example of subsample the data to 500 counts per sample.

```
vdj_counts_200 <- rrarefy(vdj_counts_mat, 200)
```

```
Warning in rrarefy(vdj_counts_mat, 200): some row sums < 'sample' and are not
rarefied
```

```
# Filter out any samples that were not subsampled to a depth of 200 (have less than 200 counts)
vdj_counts_200 <- vdj_counts_200[ rowSums(vdj_counts_200) == 200, ]

# Compute the diversity indices for each sample
# Will compute Shannon entropy (S) and richness, a.k.a. number of unique entities (N)
vdj_diversity <- data.frame(sample=row.names(vdj_counts_200),
            S=diversity(vdj_counts_200, index="shannon"),
            N=specnumber(vdj_counts_200))

head(vdj_diversity)
```

```
              sample        S   N
Sample_001 Sample_001 4.450887 110
Sample_003 Sample_003 4.736687 137
Sample_004 Sample_004 4.391094 131
Sample_005 Sample_005 4.550198 123
Sample_006 Sample_006 4.144116 102
Sample_008 Sample_008 4.159743 103
```