# Erson Analysis Vignette

**Sebastian Reyes**

**2025-05-16**

# 0. Knitting this Document

```
# this chunk knits the document
# if all necessary objects are not loaded into the environment, clear it and run the chu
nk below prior to knitting
rmarkdown::render("Erson_Analysis_Vignette.Rmd", envir = globalenv())
```

# 1. Introduction

The below framework outlines a workflow I designed to streamline the steps outlined in this Seurat vignette (https://satijalab.org/seurat/articles/pbmc3k_tutorial) for the purposes needed by the Erson Lab in the summer of 2024–when I made this. Really, the entire framework below is a reframing of this vignette to better suit the specific needs of the Erson Lab—from the perspective of my specific project, anyway.

To explain the “custom functions” created below: many of these functions are used repeatedly with different parameters during the exploratory analyses I conducted on the Erson Lab’s samples. Making these custom functions quickened the workflow and allowed parameters to more easily be edited.

That said, reading the aforementioned Seurat vignette (https://satijalab.org/seurat/articles/pbmc3k_tutorial) would provide the reader with the vast majority of the information contained in the below outline.

# 2. Installing Packages Needed

```
install.packages("dplyr")
install.packages("Seurat")
install.packages("patchwork")
install.packages("tibble")
install.packages("SingleR")
install.packages("SCINA")
BiocManager::install("celldex")
install.packages("ggplot2")
```

# 3. Loading Packages Needed

```
library(dplyr)
library(Seurat)
library(patchwork)
library(tibble)
library(SingleR)
library(SCINA)
library(celldex)
library(ggplot2)
```

# 4. All Custom Functions Used

## CreateSeuratObject

### Explanation:

- More easily creates a Seurat object based on file directory
- **Parameters**:
  - fileDirectory: file path (must be h5 file for this specific function)
  - objectName: self-explanatory
  - projectName: self-explanatory

```
CreateSeuratObject <- function(fileDirectory, objectName, projectName) {
  rawCounts <- Read10X_h5(fileDirectory)
  seurat_obj <- Seurat::CreateSeuratObject(counts = rawCounts,
                                           project = projectName,
                                           min.cells = 3,
                                           min.features = 200)
  assign(objectName, seurat_obj, envir = .GlobalEnv)
  return(objectName)
}
```

## AddMitoRna

### Explanation:

- Appends mitochondrial RNA data to Seurat object on a cell-by-cell basis

```
AddMitoRna <- function(seuratObject){
  seuratObject[["percent.mt"]] <- Seurat::PercentageFeatureSet(seuratObject, pattern =
"^MT-")
  return(seuratObject)
}
```

## MakeVlnPlot

### Explanation:

- More easily creates violin plots for QC purposes
    - nFeature_RNA = gives the number of unique features (genes) in a given cell, each point represents one cell
    - nCount_RNA = gives the total number of RNA molecules in a cell, each point represents one cell
    - percent.mt = percentage of mitochondrial RNA, each point represents one cell

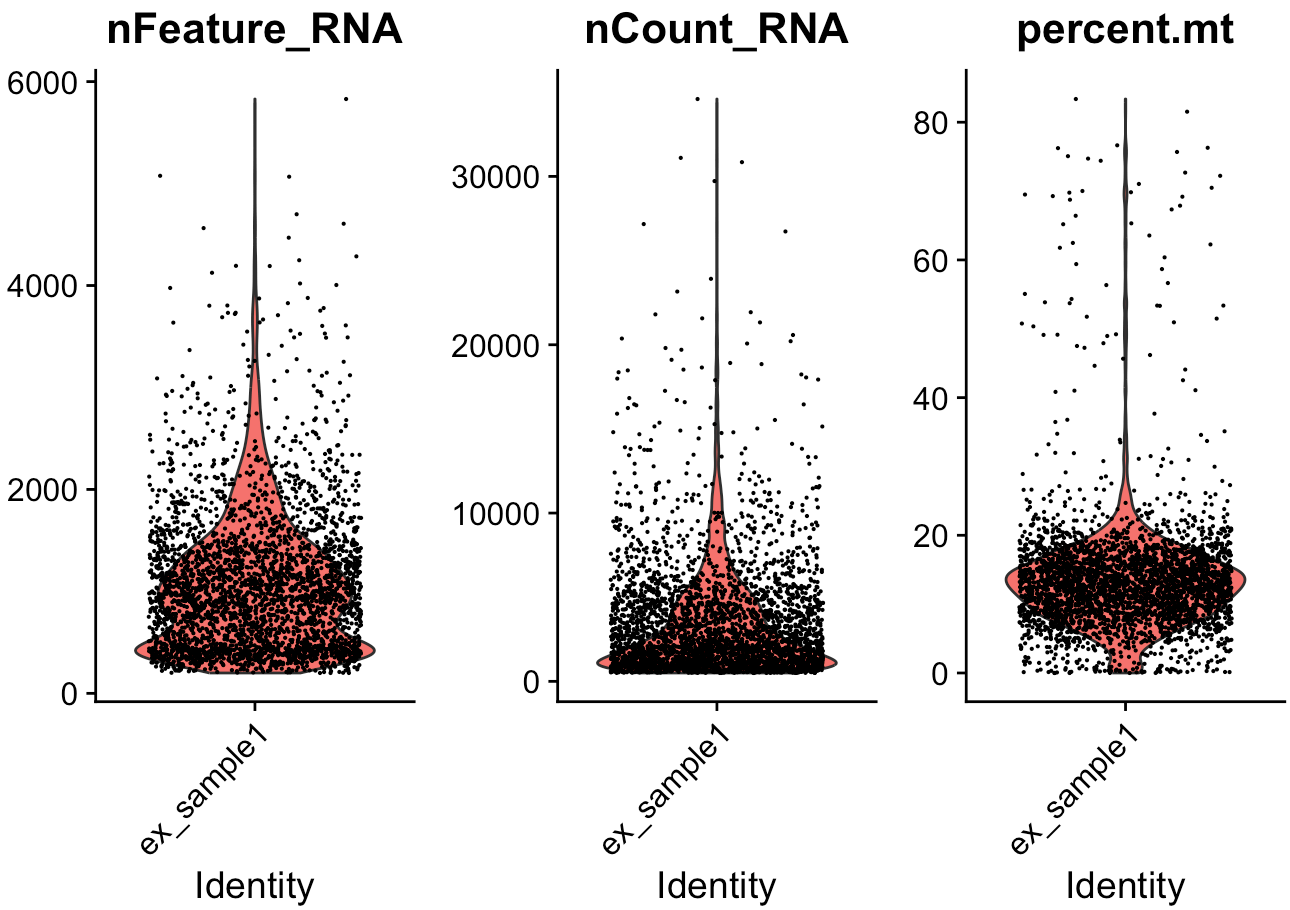
```
MakeVlnPlot <- function(seuratObject) {
  print(VlnPlot(seuratObject,
                features = c("nFeature_RNA",
                             "nCount_RNA",
                             "percent.mt"),
                ncol = 3))
  return(seuratObject)
}
```

## Example:

```
# Must be run previously:
CreateSeuratObject("DATA1.H5",
                   "ex_sample1",
                   "ex_sample1")
ex_sample1 <- AddMitoRna(ex_sample1)
```

```
MakeVlnPlot(ex_sample1)
```

```
## Warning: Default search for "data" layer in "RNA" assay yielded no results; utilizing
"counts" layer instead.
```




```
## An object of class Seurat
## 18197 features across 3301 samples within 1 assay
## Active assay: RNA (18197 features, 0 variable features)
##  1 layer present: counts
```

# QCSeurat

### Explanation:

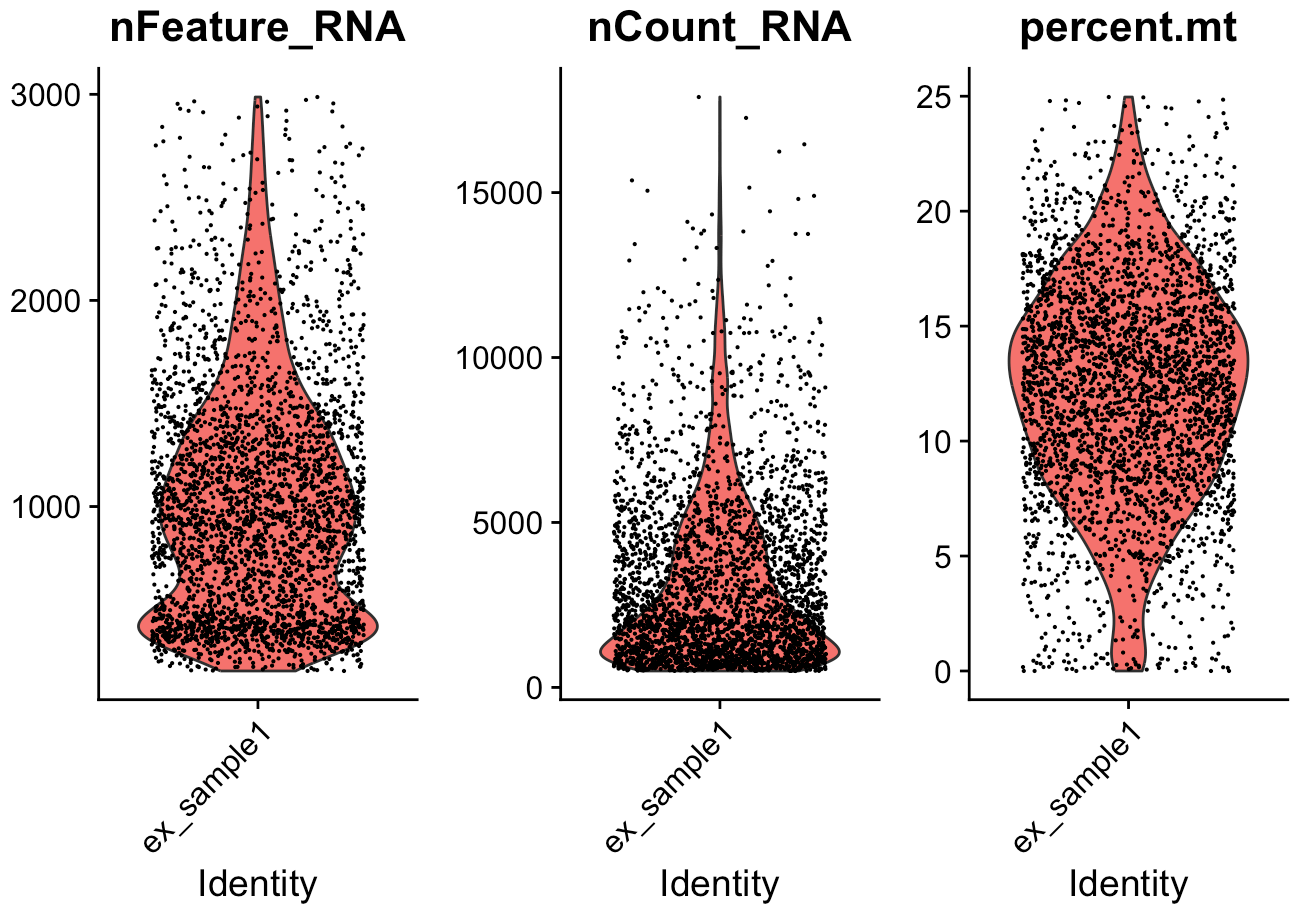
- Streamlines the quality control process, allows for low-quality cells to be eliminated based on feature count and mitochondrial RNA percentage
- Parameters usually decided based on violin plots to minimize outliers/maximize viable retained data
- **Parameters:**
  - seuratObject = seuratObject to be trimmed
  - minFeature = minimum number of genes a cell must contain to remain included in data
  - maxFeature = maximum number of genes a cell can contain to remain included in data
  - percentMT = maximum percentage of mitochondrial RNA which a cell can contain to remain included in data

```
QCSeurat <- function(seuratObject, minFeature, maxFeature, percentMT) {
  seuratObject <- subset(seuratObject, subset = nFeature_RNA > minFeature & nFeature_RNA
< maxFeature & percent.mt < percentMT)
  return(seuratObject)
}
```

## Example:

```
# Violin plot prior to QC
MakeVlnPlot(ex_sample2)
```

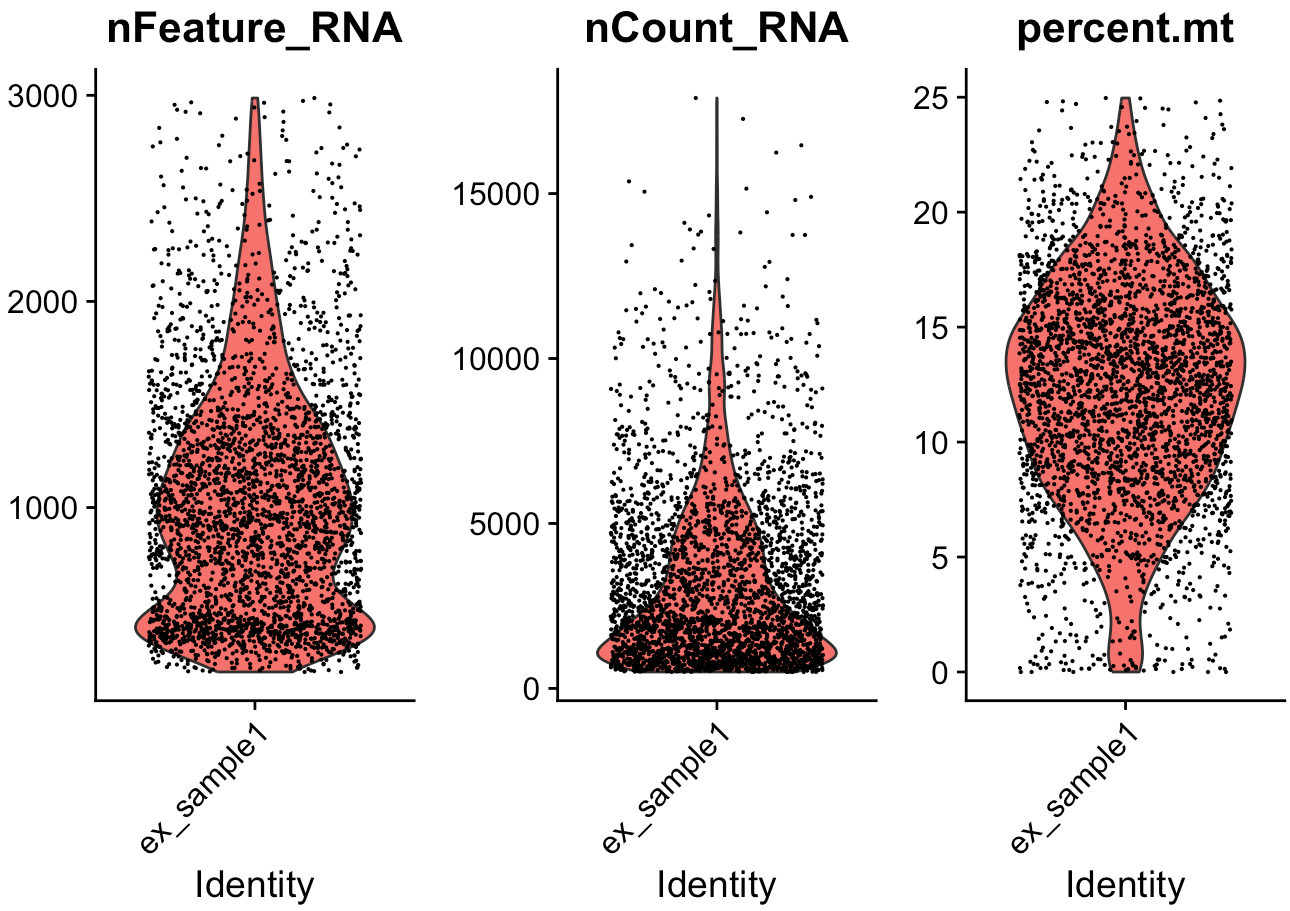
```
## Warning: Default search for "data" layer in "RNA" assay yielded no results; utilizing
"counts" layer instead.
```




```
## An object of class Seurat
## 18197 features across 3121 samples within 1 assay
## Active assay: RNA (18197 features, 0 variable features)
##  1 layer present: counts
```

```
ex_sample2 <- QCSeurat(ex_sample2, 200, 3000, 25)

#Violin plot following QC
MakeVlnPlot(ex_sample2)
```

```
## Warning: Default search for "data" layer in "RNA" assay yielded no results; utilizing
"counts" layer instead.
```




```
## An object of class Seurat
## 18197 features across 3121 samples within 1 assay
## Active assay: RNA (18197 features, 0 variable features)
##  1 layer present: counts
```

# NormalizeAndVariable

### Explanation:

- Normalizes data, finds genes within Seurat object that display variable expression patterns, and creates a plot highlighting the top 10 genes with the highest degree of variability in their expression.

```
NormalizeAndVariable <- function(seuratObject) {
  seuratObject <- NormalizeData(seuratObject)
  seuratObject <- FindVariableFeatures(seuratObject, selection.method = "vst", nfeatures
= 2000)
  top10 <- head(VariableFeatures(seuratObject), 5)
  plot1 <- VariableFeaturePlot(seuratObject)
  plot2 <- LabelPoints(plot = plot1, points = top10, repel = TRUE)
  print(plot2)
  return(seuratObject)
}
```

## Example:

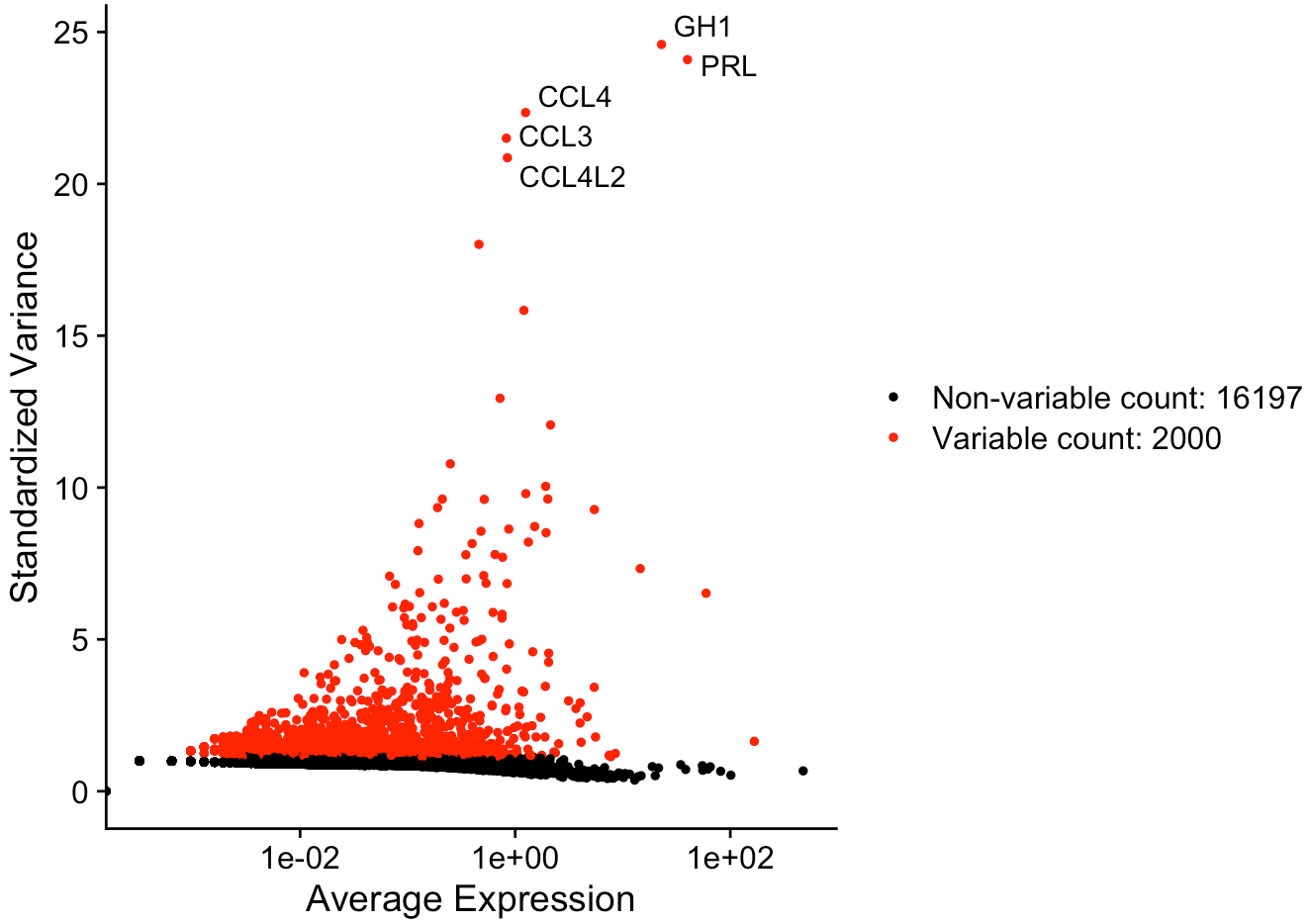
```
ex_sample3 <- NormalizeAndVariable(ex_sample3)
```

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

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

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

```
## Warning in scale_x_log10(): log-10 transformation introduced infinite values.
```



# Scale

## Explanation:

- Scales the Seurat object as per the workflow outlined in the Seurat vignette

```
Scale <- function(seuratObject) {
  seuratObject <- ScaleData(seuratObject, features = rownames(seuratObject))
  seuratObject <- ScaleData(seuratObject, vars.to.regress = "percent.mt",)
  return(seuratObject)
}
```

# PcaElbow

## Explanation:

- Runs a principal component analysis and creates an elbow plot based on it

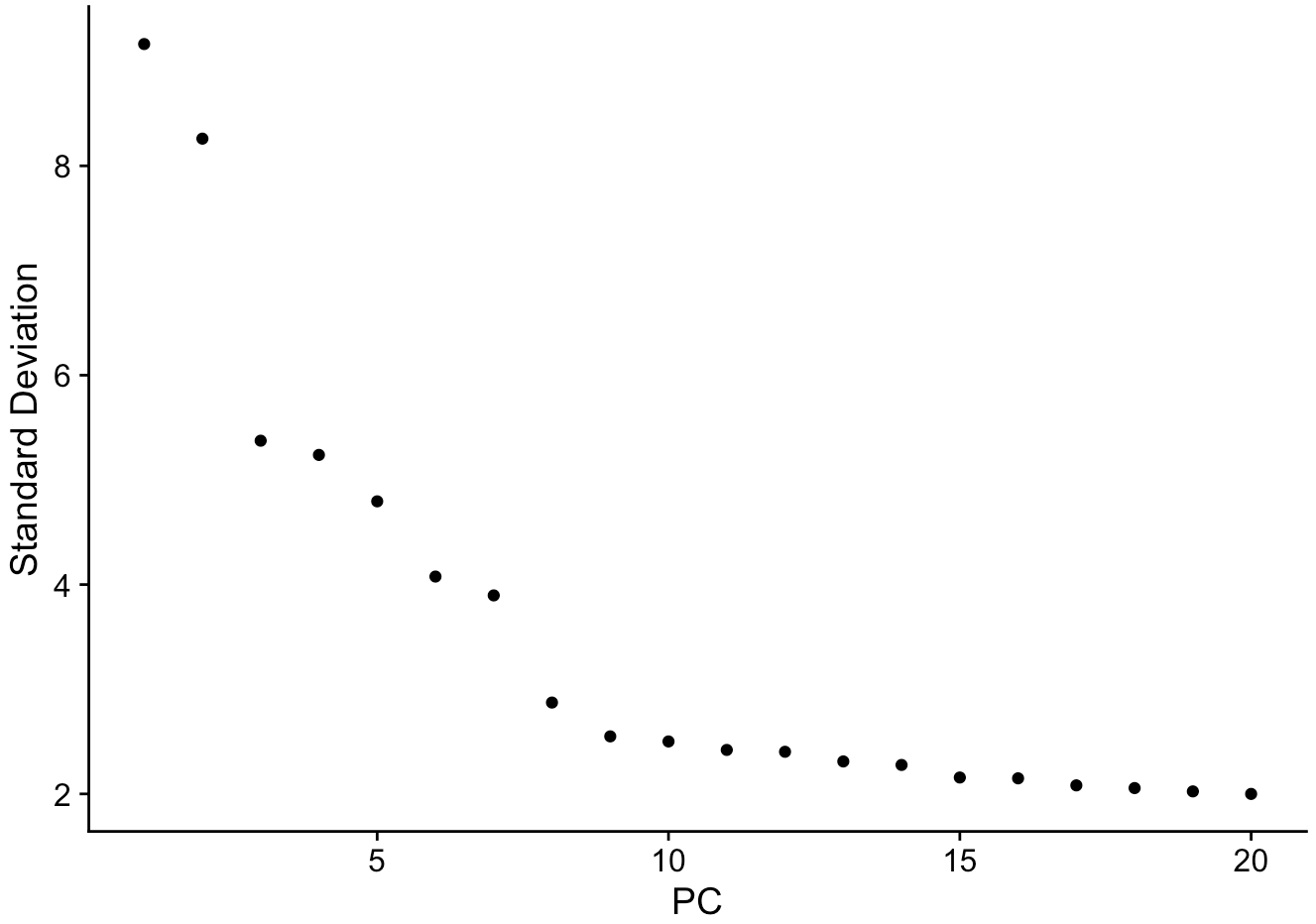
```
PcaElbow <- function(seuratObject) {
  seuratObject <- RunPCA(seuratObject, features = VariableFeatures(seuratObject))
  print(ElbowPlot(object = seuratObject))
  return(seuratObject)
}
```

## Example:

```
# Must be run previously
ex_sample4 <- Scale(ex_sample4)
```

```
ex_sample4 <- PcaElbow(ex_sample4)
```

```
## PC_ 1
## Positive:  IL1B, LYZ, PLAUR, CTSS, IER3, CXCL8, HLA-DRA, FCN1, OLR1, SRGN
##     S100A9, VCAN, BCL2A1, TYROBP, AIF1, EREG, FCER1G, NLRP3, SOD2, CXCL2
##     CD74, FGL2, CCL20, S100A8, MS4A7, VIM, THBS1, G0S2, MNDA, NFKBIA
## Negative:  CHGB, GNAS, SCG2, MEG3, PEG10, NNAT, SEC11C, MIR7-3HG, CCND1, RPS24
##     CALY, RAB3B, SHTN1, HSPA12A, KRT18, TMEM136, CITED1, DLK1, ASCL1, PTN
##     PEG3, POLR3A, PRDX4, KRT8, ARG2, ORAOV1, CYFIP2, POU1F1, MAP1B, INSM1
## PC_ 2
## Positive:  TIMP3, SPARCL1, IGFBP4, IGFBP5, GNG11, IFITM3, A2M, IFI27, PLVAP, SPARC
##     ABCG2, ADGRF5, KDR, EMCN, RAMP2, CAVIN2, SLCO2A1, PLPP3, STC1, ADGRL4
##     IGFBP7, EPAS1, PLAT, HYAL2, PCAT19, CD9, IGFBP6, ESAM, PODXL, TGM2
## Negative:  LYZ, IL1B, CTSS, BCL2A1, S100A9, TYROBP, FCN1, S100A8, EREG, SAMSN1
##     FCER1G, AIF1, PTPRC, CD44, OLR1, NLRP3, CCL20, S100A12, MNDA, VCAN
##     LCP1, PLAUR, C5AR1, FGL2, MS4A7, G0S2, CLEC7A, AC020656.1, CSTA, CYBB
## PC_ 3
## Positive:  CCND1, SHTN1, RAB3B, RPS24, HSPA12A, TMEM136, ASCL1, INSR, ORAOV1, PMAIP1
##     SPP1, POLR3A, TBX19, RCN1, ARG2, PCSK1, SCG2, GAL, GADD45B, PPP1R17
##     CYP3A4, RGS16, CREG1, INSM1, FILIP1, PRDX4, CHGB, TRDN, OSBPL1A, NEB
## Negative:  TMSB4X, NKG7, B2M, CCL5, IL32, KLRB1, GZMA, CD52, GZMB, CST7
##     GNLY, CD3D, CXCR4, HCST, TRBC2, S100A4, GZMH, ARL4C, CD2, KLRD1
##     ZFP36L2, CD7, FGFBP2, CTSW, IL7R, PTPRC, TRBC1, CD247, GZMM, S100A6
## PC_ 4
## Positive:  ABCG2, PLVAP, KDR, SLCO2A1, IFI27, EMCN, RAMP2, ADGRL4, CA4, HYAL2
##     PCAT19, PODXL, HLA-B, ITM2A, TMEM88, ESAM, DNASE1L3, PTPRB, CLEC14A, ITGA6
##     RGCC, CD34, PLPP3, EGFL7, TM4SF18, ROBO4, MMRN2, SLC9A3R2, CYP26B1, NRP2
## Negative:  DCN, MGP, BGN, RGS5, TAGLN, FBLN1, IGFBP7, IGFBP2, CCL2, STEAP4
##     MYL9, CRISPLD2, SERPING1, COL1A2, C1S, GPC3, LUM, NTRK3, NR2F1, APOD
##     APOE, TPM2, TWIST1, IGFBP6, CALD1, C1R, LAMC3, PCOLCE, RARRES2, COL6A1
## PC_ 5
## Positive:  CXCR4, TMSB4X, NKG7, TMSB10, CCL5, RPS24, IL32, GNLY, CD52, GZMA
##     GZMB, HLA-B, CST7, KLRB1, CD3D, HCST, ZFP36L2, RAB3B, GZMH, S100A4
##     TRBC2, PTPRC, CD2, ARL4C, CTSW, CD7, CCND1, FGFBP2, KLRD1, ASCL1
## Negative:  MEG3, DLK1, LINC00632, MIR7-3HG, POU1F1, HTATSF1, ARC, EGR3, GNAS, NNAT
##     EGR1, SCGN, ARHGAP36, PITX1, CITED1, CADM1, TCEAL5, TAGLN3, ALDH1A1, PEG3
##     TCEAL6, NR4A1, FOSB, C14orf132, CADPS, EGR4, GH1, VMP1, PRL, ENPP1
```




# MakeHeatmap

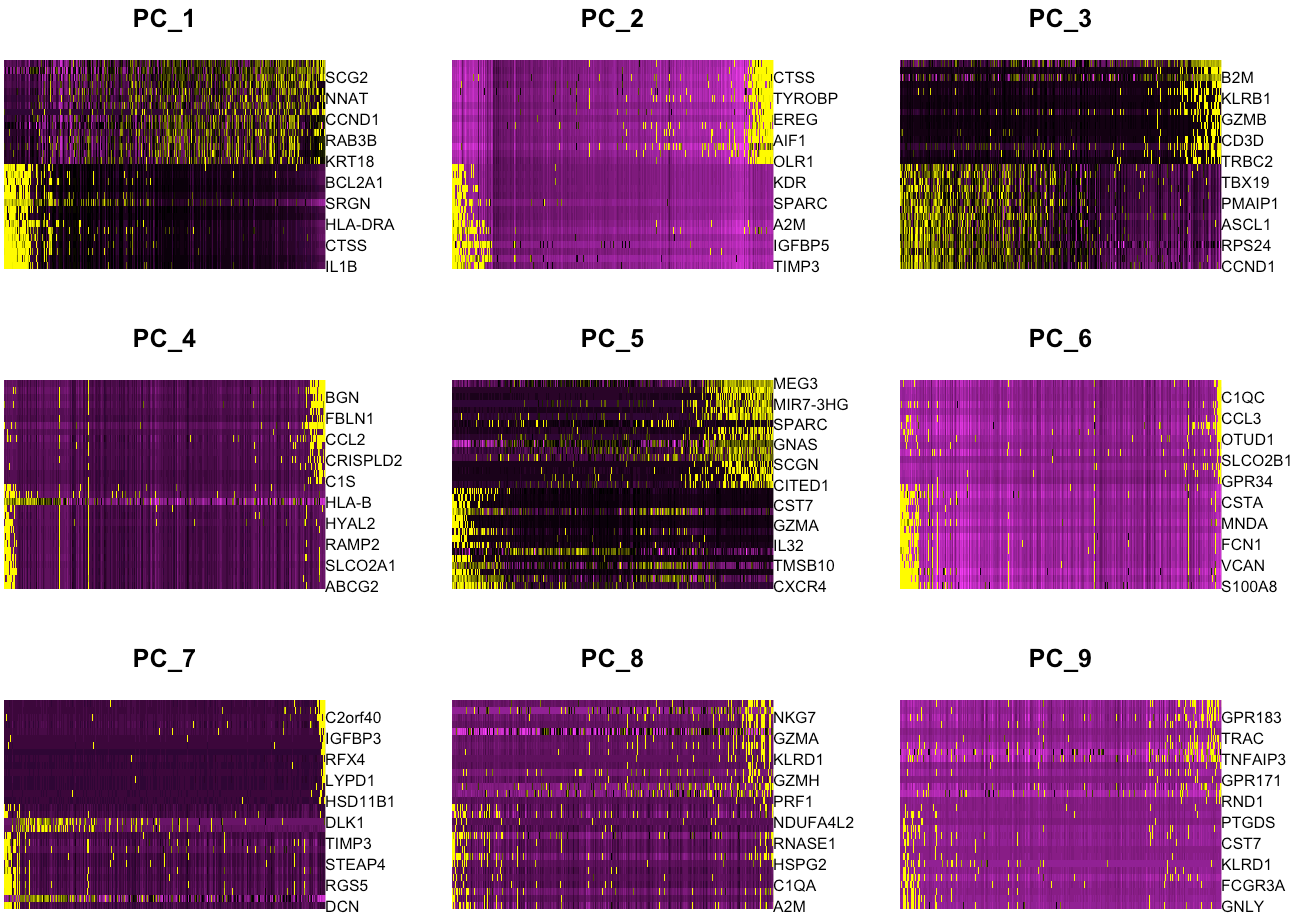
## Explanation

- Creates one heatmap for every principal component included in `PcNumberVector`
    - `PcNumberVector` should be a numeric vector

```
MakeHeatmap <- function(seuratObject, PcNumberVector){
  DimHeatmap(seuratObject, dims = PcNumberVector, balanced = TRUE)
  return(seuratObject)
}
```

## Example:

```
# Note: PcaElbow must be run first, as it attaches PCA data to the Seurat object
MakeHeatmap(ex_sample5, 1:9)
```




```
## An object of class Seurat
## 18197 features across 3121 samples within 1 assay
## Active assay: RNA (18197 features, 2000 variable features)
##  3 layers present: counts, data, scale.data
##  1 dimensional reduction calculated: pca
```

# FindCellClusters

## Explanation:

- Finds clusters of cells based off principal component behavior
- **Parameters**:
    - seuratObject = self-explanatory
    - PCs = a numeric vector containing the principal components to be used
    - resolution = a value between 0 and 1; defines the granularity of the clusters
- Cluster assignments may be found via the dataframe `seuratObject@meta.data` , column = `seurat_clusters`

```
FindCellClusters <- function(seuratObject, PCs, resolution) {
  seuratObject2 <- FindNeighbors(seuratObject, dims = PCs)
  seuratObject3 <- FindClusters(seuratObject2, resolution = resolution)
  return(seuratObject3)
}
```

### Example:

```
# Below code finds 11 clusters
ex_sample6 <- FindCellClusters(ex_sample6, 1:7, 0.5)
```

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

```
## Computing SNN
```

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 3121
## Number of edges: 91787
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.9101
## Number of communities: 13
## Elapsed time: 0 seconds
```

# MakeUMAP

### Explanation:

- Runs UMAP + creates a UMAP plot based on the specified principal components
- **Parameters:**
    - seuratObject = self-explanatory
    - PCs = a numeric vector containing the principal components to be used

```
MakeUMAP <- function(seuratObject1, PCs) {
  seuratObject2 <- RunUMAP(seuratObject1, dims = PCs)
  print(DimPlot(seuratObject2, reduction = "umap"))
  return(seuratObject2)
}
```

### Example:

```
ex_sample7 <- MakeUMAP(ex_sample7, 1:7)
```

```
## 15:44:03 UMAP embedding parameters a = 0.9922 b = 1.112
```

```
## 15:44:03 Read 3121 rows and found 7 numeric columns
```

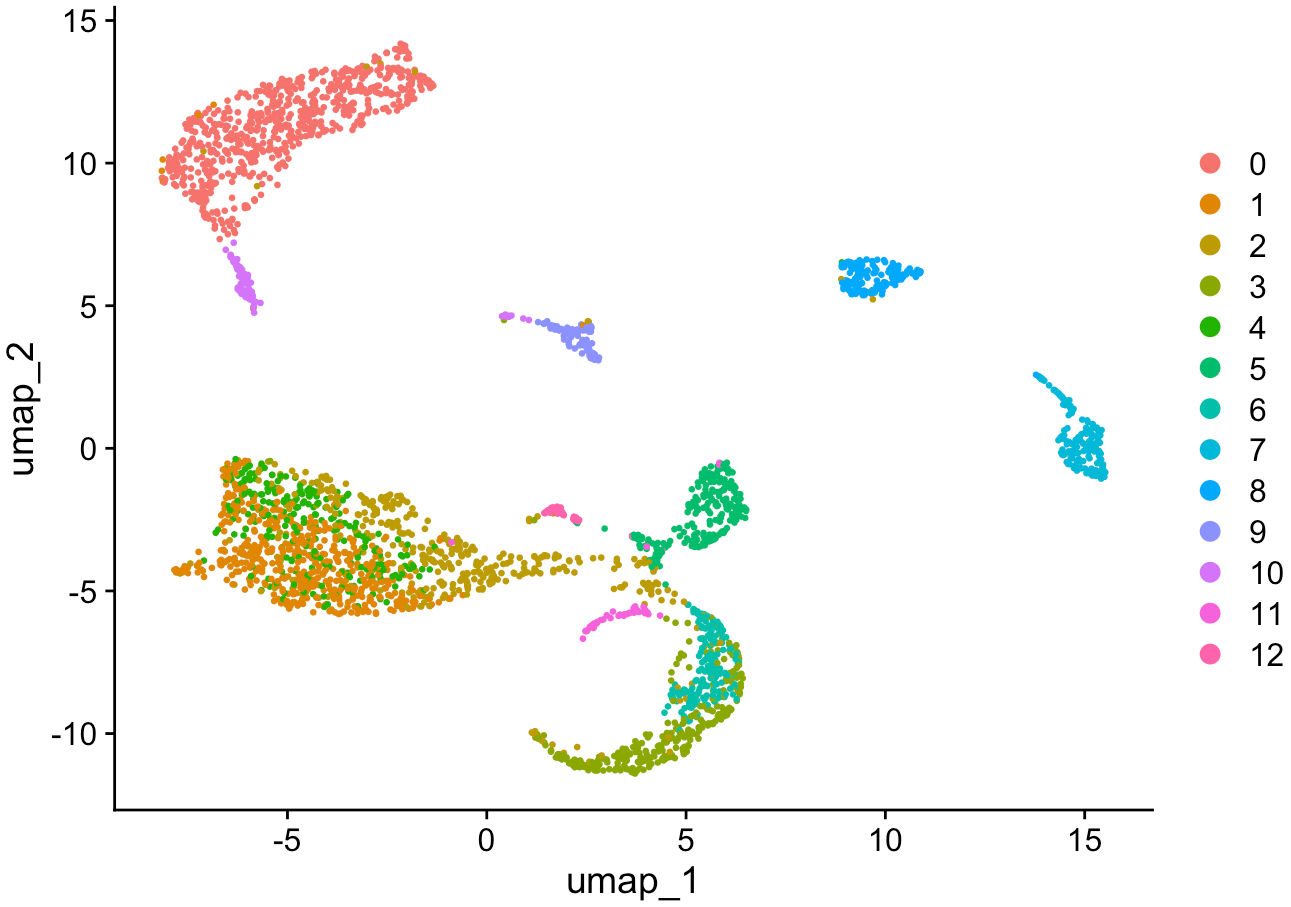
```
## 15:44:03 Using Annoy for neighbor search, n_neighbors = 30
```

```
## 15:44:03 Building Annoy index with metric = cosine, n_trees = 50
```

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

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

```
## **************************************************|
## 15:44:03 Writing NN index file to temp file /var/folders/xs/84n7_bk52l98qmnzc7ndzlmr0
000gn/T//RtmphMbDkA/file2ce4c281533
## 15:44:03 Searching Annoy index using 1 thread, search_k = 3000
## 15:44:04 Annoy recall = 100%
## 15:44:04 Commencing smooth kNN distance calibration using 1 thread with target n_neig
hbors = 30
## 15:44:05 Initializing from normalized Laplacian + noise (using RSpectra)
## 15:44:05 Commencing optimization for 500 epochs, with 123504 positive edges
## 15:44:08 Optimization finished
```



# FindPosMarkers

## Explanation:

- Finds markers that are upregulated within previously established clusters

```
FindPosMarkers <- function(seuratObject){
  posMarkers <- FindAllMarkers(seuratObject, only.pos = TRUE)
  posMarkers %>%
    group_by(cluster) %>%
    dplyr::filter(avg_log2FC >1)
  return(seuratObject)
}
```

# Top5Markers

## Explanation:

- Prints out the top 5 most upregulated genetic markers on a cluster-by-cluster basis

```
Top5Markers <- function(seuratObject) {
  posMarkers <- FindAllMarkers(seuratObject, only.pos = TRUE)
  separatedMarkers <- posMarkers %>%
    dplyr::filter(avg_log2FC >1) %>%
    group_by(cluster) %>%
    slice_head(n = 5)
  print(separatedMarkers, n = Inf)
}
```

## Example:

```
# the actual output is 65 rows
head(Top5Markers(ex_sample8))
```

| p_val <dbl> | avg_log2FC <dbl> | pct.1 <dbl> | pct.2 <dbl> | p_val_adj <dbl> | cluster <fctr> | gene <chr> |
|---|---|---|---|---|---|---|
| 0.000000e… | 5.220663 | 0.793 | 0.033 | 0.000000e… | 0 | DLK1 |
| 0.000000e… | 7.499710 | 0.644 | 0.011 | 0.000000e… | 0 | LINC00632 |
| 0.000000e… | 3.948433 | 0.970 | 0.433 | 0.000000e… | 0 | MEG3 |
| 1.365985e-… | 5.079439 | 0.588 | 0.025 | 2.485682e-… | 0 | POU1F1 |
| 6.449769e-… | 3.402770 | 0.781 | 0.150 | 1.173664e-… | 0 | MIR7-3HG |
| 6.356813e-… | 2.291421 | 0.979 | 0.342 | 1.156749e-… | 1 | CCND1 |

6 rows

# SingleHPCA

## Explanation:

- Annotates each individual cell based on its expression profile's closest match within the Human Primary Cell Atlas reference dataset; based on the use of SingleR

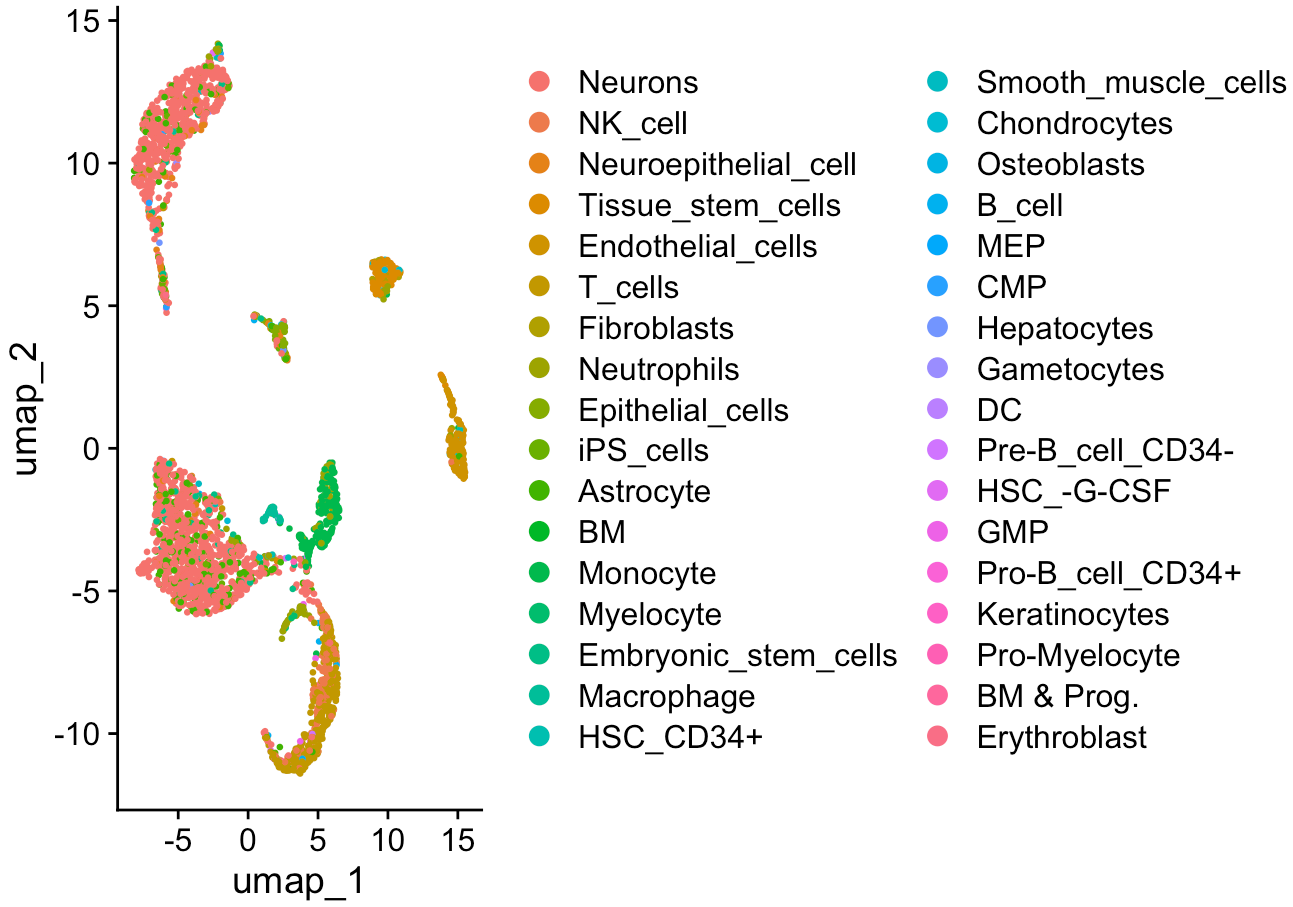
- HPCA dataset is more generic than the below neuroendocrine or immune cell datasets, but is good for establishing an initial tissue composition overview
- Cell identities may be found via the dataframe `seuratObject@meta.data` , column = `HPCACellLabels`

```
SingleHPCA <- function(seuratObject){
  HpcaLabels <- SingleR(test = as.SingleCellExperiment(seuratObject),
                        ref =    HumanPrimaryCellAtlasData(),
                        labels = HumanPrimaryCellAtlasData()$label.main)
  seuratObject1 <- AddMetaData(seuratObject,
                               metadata = HpcaLabels@listData[["labels"]],
                               col.name = "HPCACellLabels")
  return(seuratObject1)
}
```

## Example:

```
ex_sample9 <- SingleHPCA(ex_sample9)
```

```
Idents(ex_sample9) <- ex_sample9@meta.data$HPCACellLabels
DimPlot(ex_sample9, reduction = "umap")
```



# SingleNeuro

- **Explanation:** Annotates each individual cell its expression profile's closest match within the dataset linked from the paper linked here (https://www.nature.com/articles/s41467-020-19012-4), via the use of SingleR
    - Reference dataset is of neuroendocrine cells
    - Reference dataset file is in the GitHub repository and should automatically be utilized when calling this function

## Reference Dataset

```
#Just run entire chunk

NeuroMeta <- read.csv("Reference Datasets/NeuroMeta.csv.gz")

NeuroExpression <- read.csv("Reference Datasets/NeuroExpression.csv.gz")

columns_to_keep <- colnames(NeuroExpression) %in% NeuroMeta$X | colnames(NeuroExpressio
n) == "X"

NeuroExpressionFiltered <- NeuroExpression[, columns_to_keep]

rm(NeuroExpression)

rownames(NeuroExpressionFiltered) <- NeuroExpressionFiltered[,1]
NeuroExpressionFiltered <- NeuroExpressionFiltered[,-1]

NeuroExpressionFiltered <- NeuroExpressionFiltered %>%
  mutate(across(everything(), as.numeric))

NeuroIDs <- as.character(NeuroMeta$cell_type)
names(NeuroIDs) <- colnames(NeuroExpressionFiltered)

rm(NeuroMeta)
NeuroExpressionFiltered <- as.matrix(NeuroExpressionFiltered)
```

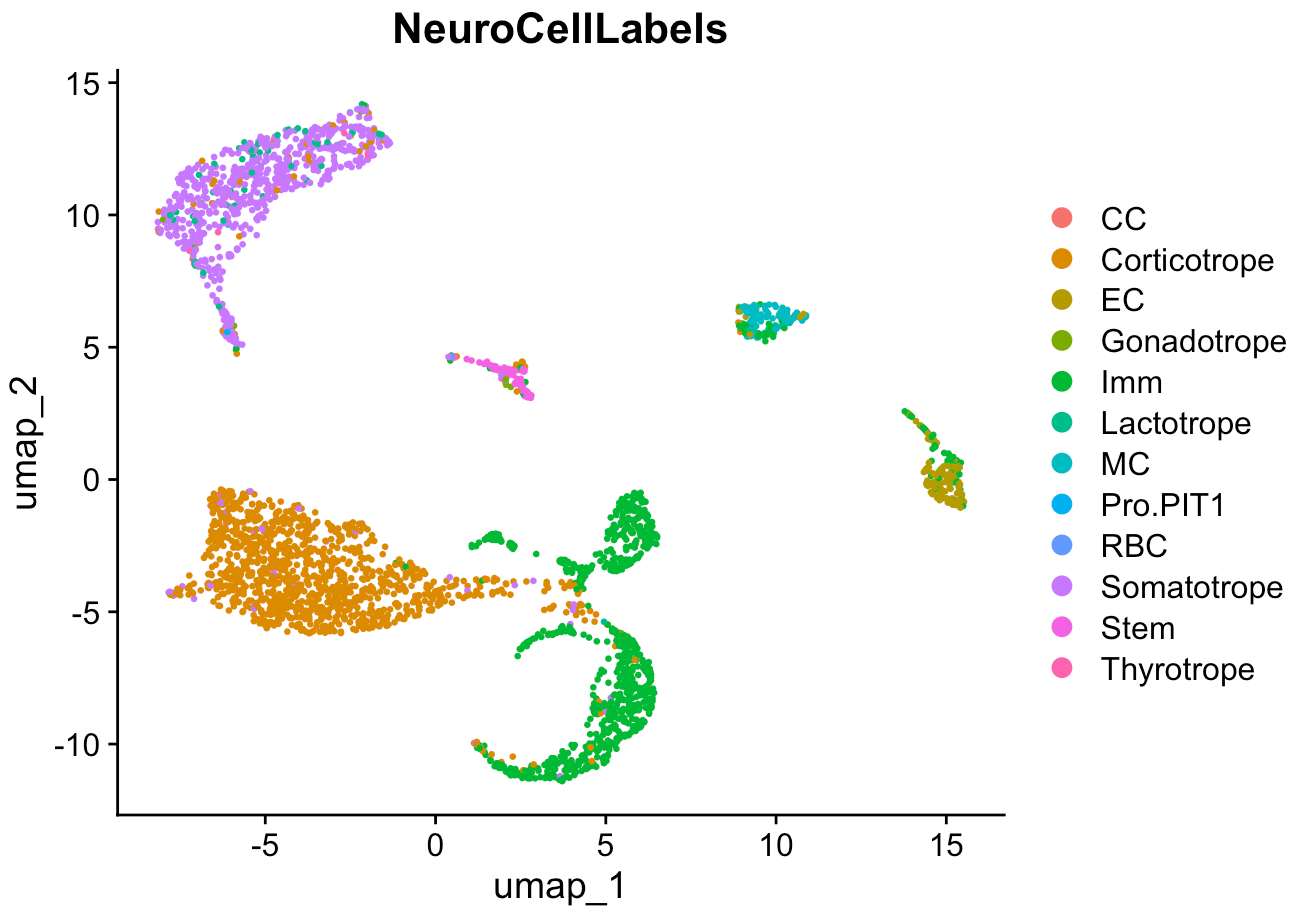
## Function

```
SingleNeuro <- function(seuratObject) {
  # Function to run SingleR
  NeuroClusterLabels <- SingleR(test = as.SingleCellExperiment(seuratObject),
                                ref = NeuroExpressionFiltered,
                                labels = NeuroIDs)

  #To add SingleR output to Seurat Object
  seuratObject1 <- AddMetaData(seuratObject, metadata = NeuroClusterLabels@listData[["la
bels"]], col.name = "NeuroCellLabels")

  #To group everything by Neuro Cell Type
  Idents(seuratObject1) <- "NeuroCellLabels"
  return(seuratObject1)
}
```

## Example:

```
ex_sample10 <- SingleNeuro(ex_sample10)
```

```
DimPlot(ex_sample10,
        reduction = "umap",
        group.by = "NeuroCellLabels")
```



# SingleImmune

## Explanation:

- Annotates each cell identified based on its expression profiles' closest match within the dataset from the paper linked here (https://www.science.org/doi/10.1126/science.abl5197), via the use of SCINA (https://pmc.ncbi.nlm.nih.gov/articles/PMC6678337/).
- Cell identities may be found via the dataframe `seuratObject@meta.data` , column = `ImmuneCellLabels`
- **Note:** Notice that some of the cells from the previous SingleNeuro() function were labeled "Imm". These are typically the only cells that I ran this function on, so as to not mischaracterize previously identified neuroendocrine cells.
- **Note:** SCINA is used in place of SingleR due to the different structure of the reference dataset.
- **Note:** The function is named `Single Immune` purely for syntax consistency, as the function serves the same effective purpose as the aforementioned `SingleHPCA` despite using SCINA instead of SingleR.
    - Reference dataset is of immune cells
    - Reference dataset file is in the GitHub repository and should automatically be utilized when calling this function

## Reference Dataset

```
ImmuneRef <- read.csv("Reference Datasets/ImmuneRef.csv")
ImmuneRef_list <- lapply(1:39, function(i) {
  genes <- ImmuneRef[, i]
  genes <- genes[!is.na(genes)]
  return(genes)
})

names(ImmuneRef_list) <- colnames(ImmuneRef)
ImmuneRef <- ImmuneRef_list
rm(ImmuneRef_list)
```

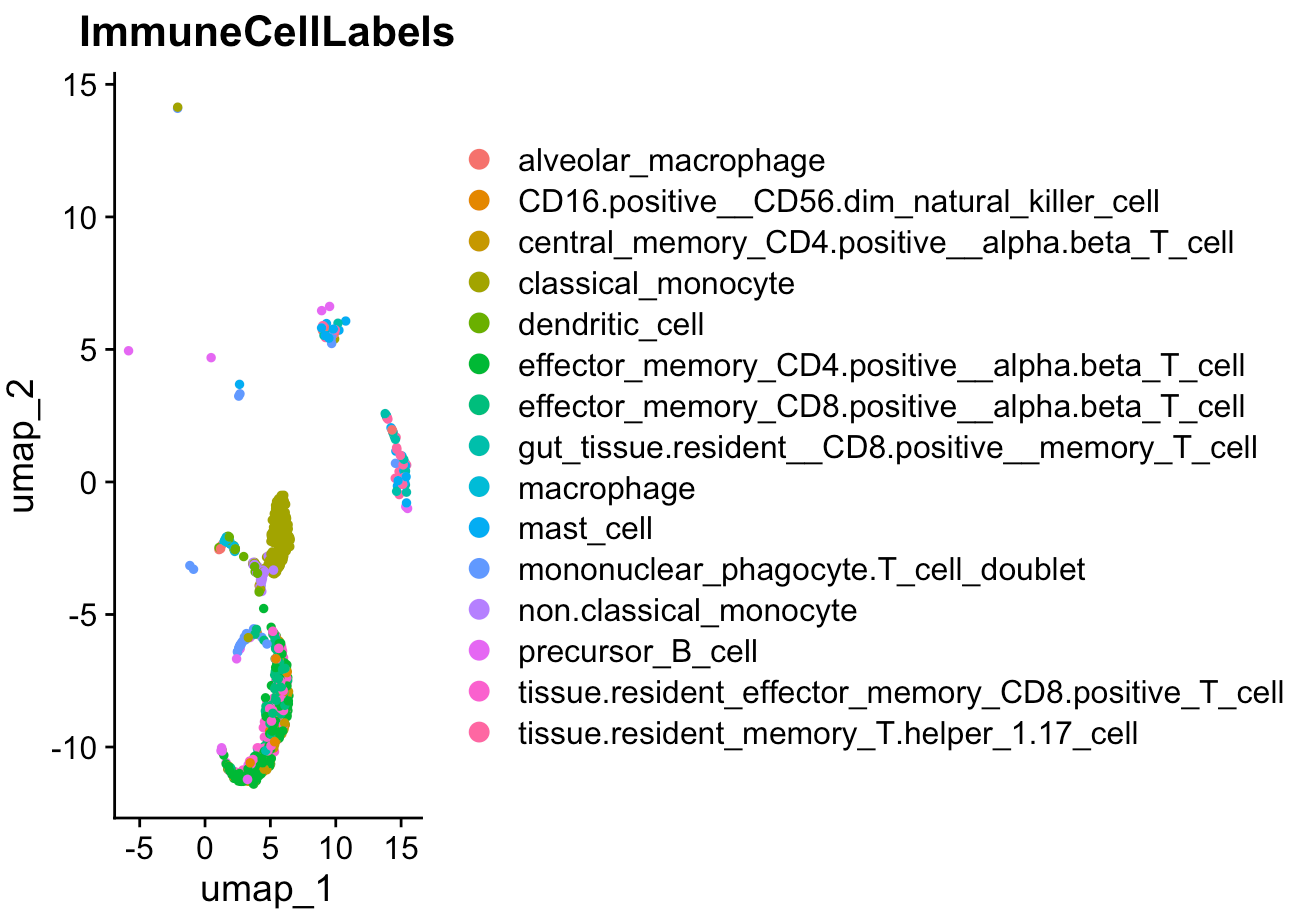
## Function

```
SingleImmune <- function(seuratObject) {

SeuratObjectExpressionData <- as.matrix(GetAssayData(seuratObject, assay = "RNA", layer
= "data"))

#NOTE: significant overlap of genetic signatures between cell types--had to set rm_overl
ap to FALSE

results <- SCINA(SeuratObjectExpressionData, ImmuneRef, rm_overlap = FALSE)
seuratObject1 <- AddMetaData(seuratObject, metadata = results$cell_labels, col.name = "I
mmuneCellLabels")
Idents(seuratObject1) <- "ImmuneCellLabels"
return(seuratObject1)
}
```

## Example:

```
#notice that we are leveraging our SingleNeuro characterizations
ex_sample10 <- SingleImmune(ex_sample10)
ex_sample10imm <- subset(ex_sample10, NeuroCellLabels == "Imm")

# this dataset results in a VERY large number of cell IDs, so for legibility, we are get
ting rid of any cell ID with less than 5 cells

CategoryCounts <- ex_sample10imm@meta.data %>%
  group_by(ImmuneCellLabels) %>%
  tally()

ValidCategories <- CategoryCounts %>%
  filter(n >= 10) %>%
  pull(ImmuneCellLabels)
```

```
DimPlot(subset(ex_sample10imm,  ImmuneCellLabels %in% ValidCategories),
        reduction = "umap",
        group.by = "ImmuneCellLabels")
```



# SingleUnbiasedNeuro

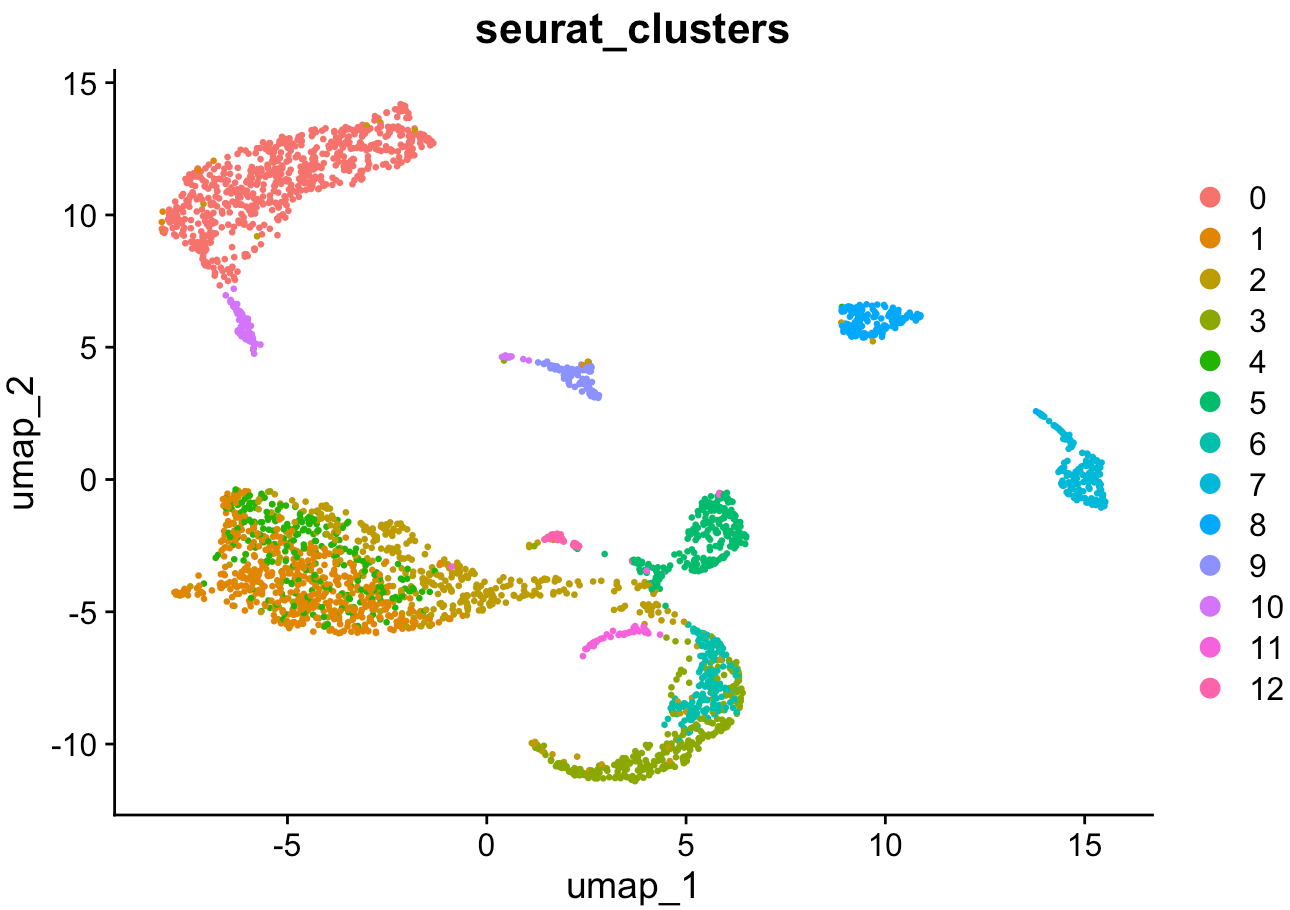
- **Explanation:** Annotates each individual cluster previously identified via principal component analysis (aka via `FindCellClusters`) based on its collective expression profile's closest match within the dataset linked from the paper linked here (https://www.nature.com/articles/s41467-020-19012-4), via the use of SingleR
    - Reference dataset is of neuroendocrine cells (same as `SingleNeuro`)
    - Reference dataset file is in the GitHub repository and should automatically be utilized when calling this function
    - Annotations can be referenced via `seuratObject@meta.data$UnbiasedNeuroCellLabels`

```
SingleUnbiasedNeuro <- function(seuratObject){

  # Create a table of `seurat_clusters` vs `NeuroCellLabels`
  cluster_label_counts <- table(seuratObject@meta.data$seurat_clusters,
                                seuratObject@meta.data$NeuroCellLabels)

  # Identify the most abundant label for each cluster
  most_abundant_labels <- apply(cluster_label_counts, 1, function(x) {
    names(x)[which.max(x)] # Returns the label with the highest count
  })

  # Relabel `seurat_clusters` with the most abundant `NeuroCellLabels`
  seuratObject@meta.data$UnbiasedNeuroCellLabels <- factor(seuratObject@meta.data$seurat
_clusters, levels = names(most_abundant_labels), labels = most_abundant_labels)
  return(seuratObject)
}
```

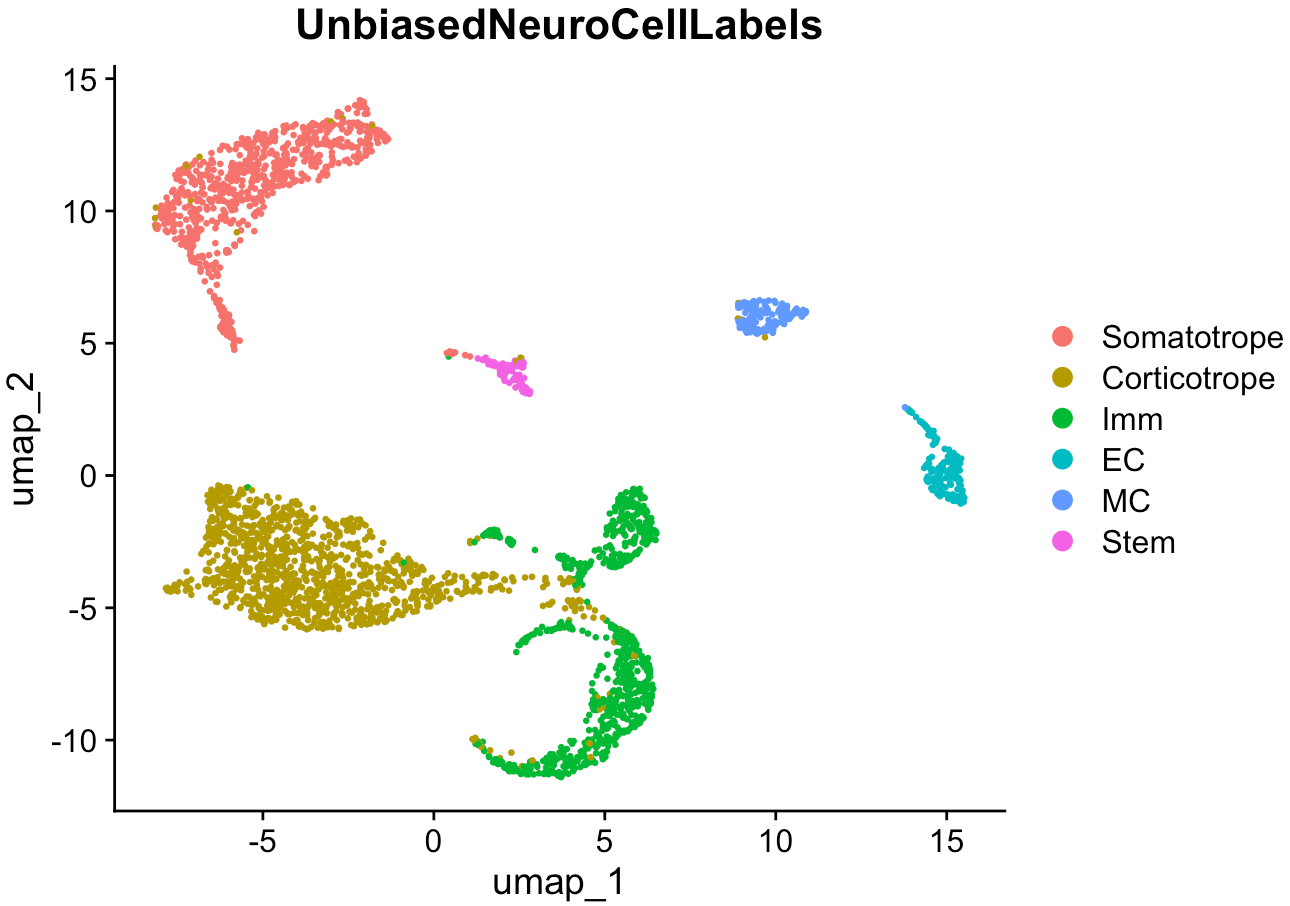
## Example:

```
ex_sample10 <- SingleUnbiasedNeuro(ex_sample10)
```

```
# notice that it is each cluster within "seurat_clusters" that is relabeled
DimPlot(ex_sample10,
        reduction = "umap",
        group.by = "seurat_clusters")
```




```
DimPlot(ex_sample10,
        reduction = "umap",
        group.by = "UnbiasedNeuroCellLabels")
```




## SingleUnbiasedImmune

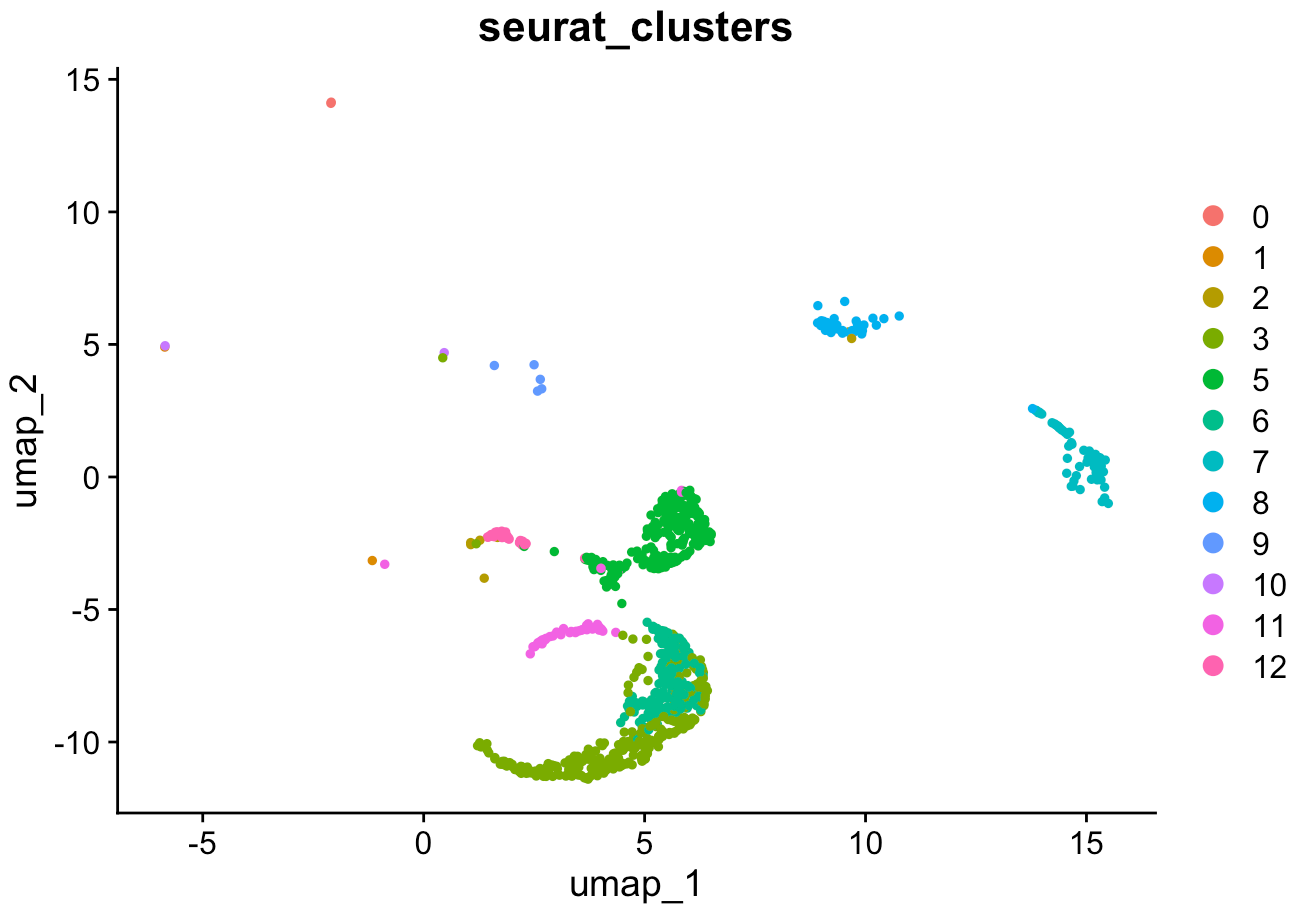
- **Explanation:** Annotates each individual cluster previously identified via principle component analysis (aka via `FindCellClusters`) based on its expression profiles' closest match within the dataset from the paper linked here (https://www.science.org/doi/10.1126/science.abl5197), via the use of SCINA (https://pmc.ncbi.nlm.nih.gov/articles/PMC6678337/).
- **Note:** SCINA is used in place of SingleR due to the different structure of the reference dataset.
- **Note:** The function is named *`Single`*`UnbiasedImmune` purely for syntax consistency, as the function serves the same effective purpose as the aforementioned `SingleUnbiasedNeuro`, despite using SCINA instead of SingleR.
    - Reference dataset is of immune cells
    - Reference dataset file is in the GitHub repository and should automatically be utilized when calling this function

```
SingleUnbiasedImmune <- function(seuratObject){
  cluster_label_counts <- table(seuratObject@meta.data$seurat_clusters,
                                seuratObject@meta.data$ImmuneCellLabels)

  most_abundant_labels <- apply(cluster_label_counts, 1, function(x) {
    names(x)[which.max(x)]
  })

  seuratObject@meta.data$UnbiasedImmuneCellLabels <- factor(seuratObject@meta.data$seura
t_clusters, levels = names(most_abundant_labels), labels = most_abundant_labels)

  return(seuratObject)
}
```

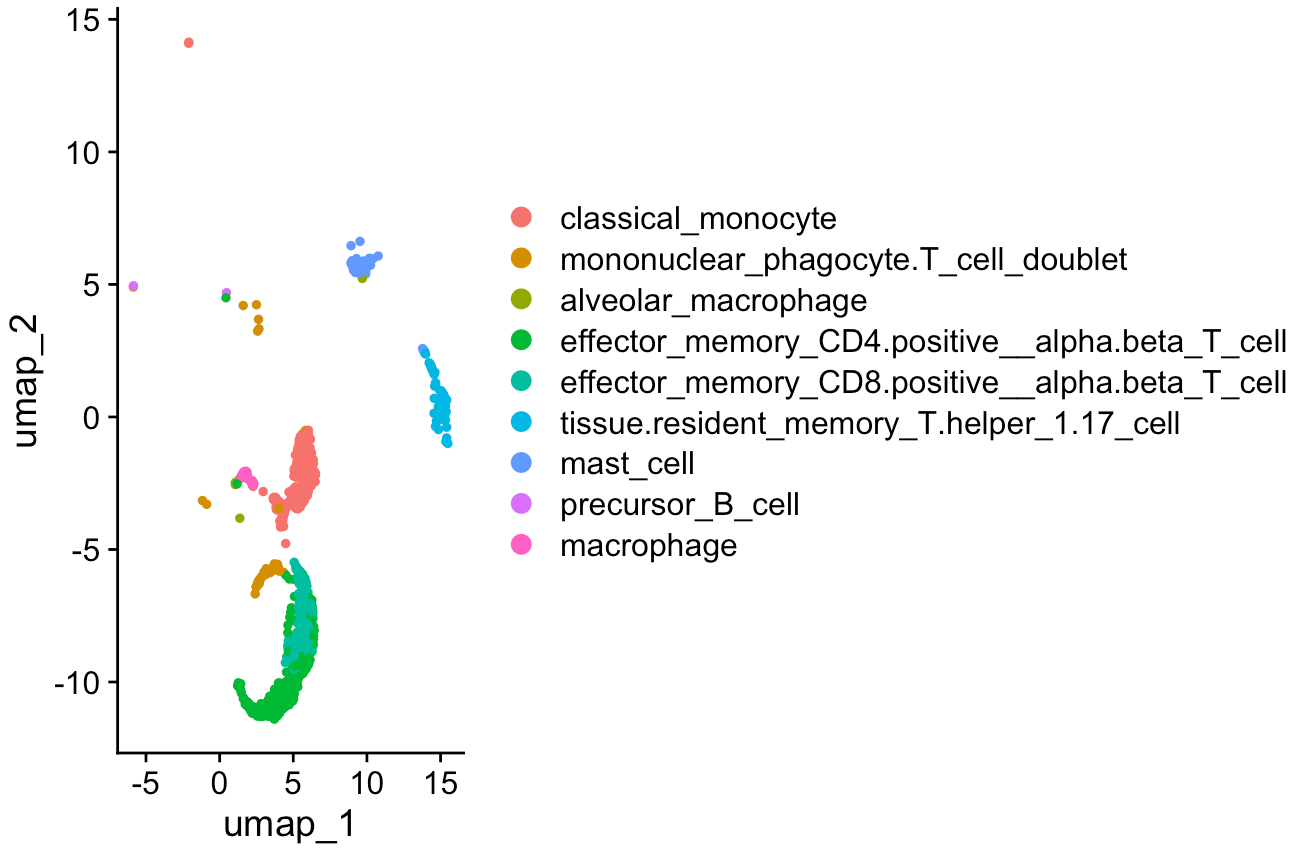
## Example:

```
# notice that we are using the Seurat object that is subsetted to only have immune cells
(identified via SingleNeuro)
ex_sample10imm <- SingleUnbiasedImmune(ex_sample10imm)
```

```
# notice that it is each cluster within "seurat_clusters" that is relabeled
DimPlot(ex_sample10imm,
        reduction = "umap",
        group.by = "seurat_clusters")
```



```
DimPlot(ex_sample10imm,
        reduction = "umap",
        group.by = "UnbiasedImmuneCellLabels")
```



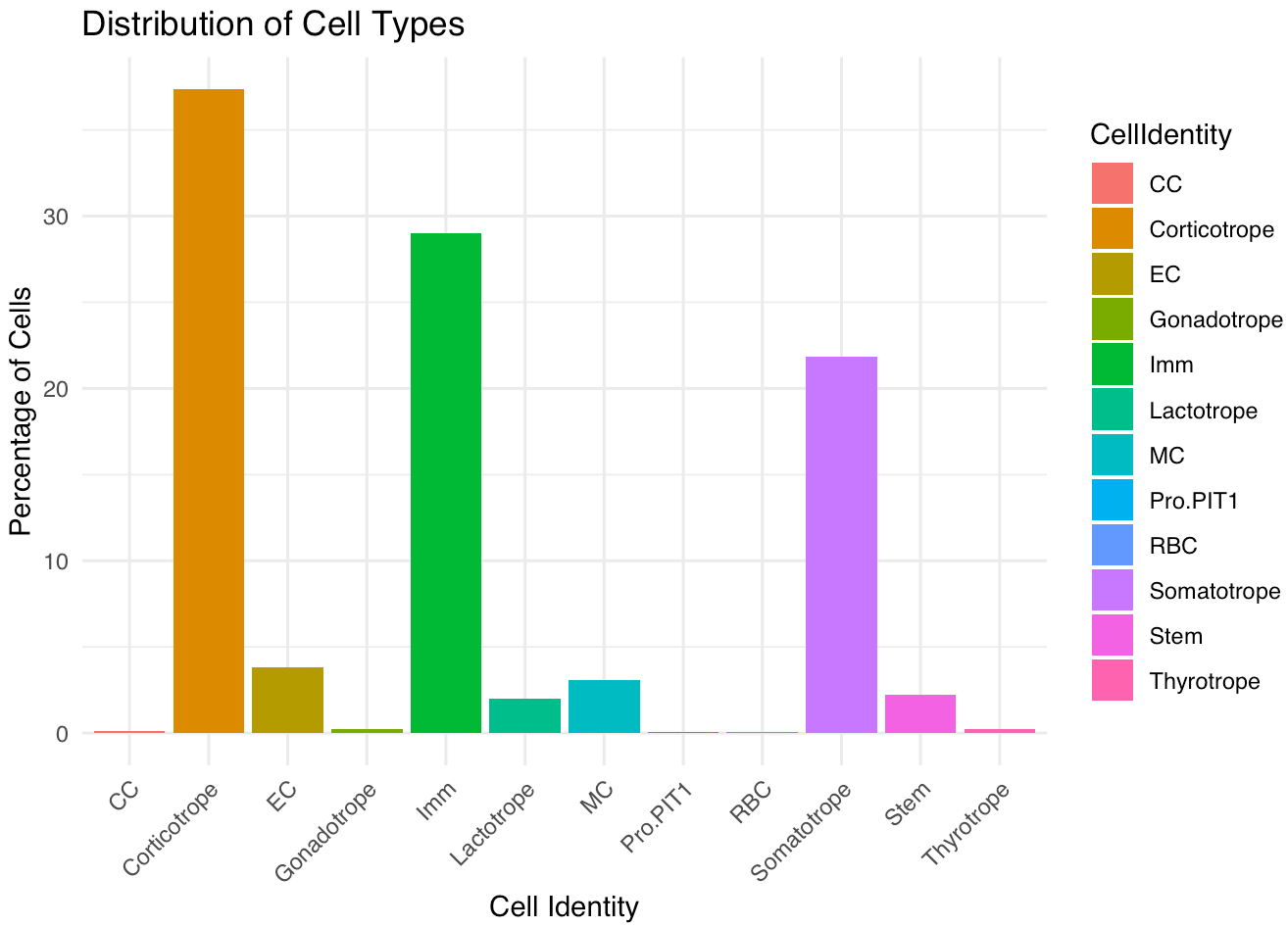

# BarPlotIdPercent

- **Explanation:** Creates a barplot detailing the percentage each cell identity makes up of the total sample
    - **Note:** the `Identity` parameter needs to be a character string in quotation marks for the function to work properly
- **Parameters:**
    - **Identity**: the cell annotation that will be used to make the barplot
        - e.g. `"NeuroCellLabels"` or `"seurat_clusters"`

```
#To create an inclusive barplot based on cell identities
BarPlotIdPercent <- function(seuratObject, Identity){

  TotalCells <- dim(seuratObject@meta.data)[1]

  cellIdentities <- seuratObject@meta.data[[Identity]]

  cellCounts <- table(cellIdentities)

  cellCountsDF <- as.data.frame(cellCounts)

  colnames(cellCountsDF) <- c("CellIdentity", "Count")

  ggplot(cellCountsDF, aes(x = CellIdentity,
                           y = Count/TotalCells*100,
                           fill = CellIdentity)) +
    geom_bar(stat = "identity") +
    theme_minimal() +
    labs(title = "Distribution of Cell Types", x = "Cell Identity", y = "Percentage of C
ells") +
    theme(axis.text.x = element_text(angle = 45, hjust = 1)) +
    theme(text = element_text("sans"))
}
```

## Example:

```
BarPlotIdPercent(ex_sample10, "NeuroCellLabels")
```


Distribution of Cell Types
Percentage of Cells
30
20
10
0
CC
Corticotrope
EC
Gonadotrope
Imm
Lactotrope
MC
Pro.PIT1
RBC
Somatotrope
Stem
Thyrotrope
Cell Identity
CellIdentity
CC
Corticotrope
EC
Gonadotrope
Imm
Lactotrope
MC
Pro.PIT1
RBC
Somatotrope
Stem
Thyrotrope