Seurat - Clustering Tutorial

Dennis Dimitri Krutkin

2023-06-17

```r
#Load necessary libraries
library(dplyr)
```

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

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

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

```r
library(Seurat)
```

```
## Attaching SeuratObject
```

```r
library(patchwork)
```

Import 10X dataset

```r
# Load the PBMC data set:
pbmc.data = Read10X(data.dir = "./filtered_gene_bc_matrices/hg19/")

# Initialize the Seurat object with the raw (non-normalized data):
pbmc = CreateSeuratObject(counts = pbmc.data, project = "pbmc3k", min.cells = 3, min.features = 200)
```

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

```r
pbmc
```

```
## An object of class Seurat
## 13714 features across 2700 samples within 1 assay
## Active assay: RNA (13714 features, 0 variable features)
```

Examine the loaded dataset

```r
# Examine a 3 genes in the first 25 cells:
pbmc.data[c("CD3D", "TCL1A", "MS4A1"), 1:25]
```

```
## 3 x 25 sparse Matrix of class "dgCMatrix"
```

```
##    [[ suppressing 25 column names 'AAACATACAACCAC-1', 'AAACATTGAGCTAC-1', 'AAACATTGATCAGC-1' ...
]]
```

```
##
## CD3D   4 . 10 . . 1 2 3 1 . . 2 7 1 . . 1 3 . 2  3 . . . .
## TCL1A  . .  . . . . . . 1 . . . . . . . . . . .  . 1 . . .
## MS4A1  . 6  . . . . . . 1 1 1 . . . . . . . . . . 36 1 2 . .
```

The "." values in the matrix represent 0s (no molecules detected).

Most values in a scRNA-seq matrix are 0 - Seurat uses a sparse-matrix representation whenever possible. This results in significant memory and speed savings for Drop-seq/inDrop/10x data.

```r
#Check the full size of object:
dense.size <- object.size(as.matrix(pbmc.data))
dense.size
```

```
## 709591472 bytes
```

```r
#Check the sparse-represented size:
sparse.size <- object.size(pbmc.data)
sparse.size
```

```
## 29905192 bytes
```

```r
#Check the ratio of full-size:sparse-size:
dense.size/sparse.size
```

```
## 23.7 bytes
```

Standard pre-processing workflow

The steps below demonstrate the standard pre-processing workflow for scRNA-seq data using Seurat.

The steps include selection and filtration of cells based on QC metrics, data normalization and scaling, and the detection of highly variable features.

Quality Control and selecting cells to further analyze

In Seurat, QC metrics can be explored and cells can be filtered according to any user-defined criteria.

Some QC metrics used by the community include:

- The number of unique genes detected in each cell
  - Low-quality cells or empty droplets often have very few genes
  - Cell doublets or multiplets may have unusually high gene counts
- The total number of molecules detected within a cell, which correlate strongly with unique genes
- The percentage of reads which map to the mitochondrial genome
  - Poor quality and dying cells frequently show extensive mitochondrial contamination
  - The PercentageFeatureSet() function calculates the percentage of counts originating from a set of features to calculate mitochondrial QC metrics
  - Genes starting with MT- are pooled as a set of mitochondrial genes

```r
#Add QC stats for mitochondrial genes to the object metadata:
pbmc[["percent.mt"]] = PercentageFeatureSet(pbmc, pattern = "^MT-")
#pbmc[["percent.mt"]]
```

Where are QC metrics stored in Seurat?

- The number of unique genes and total molecules are automatically calculated during CreateSeuratObject()
  - They are stored in the object metadata

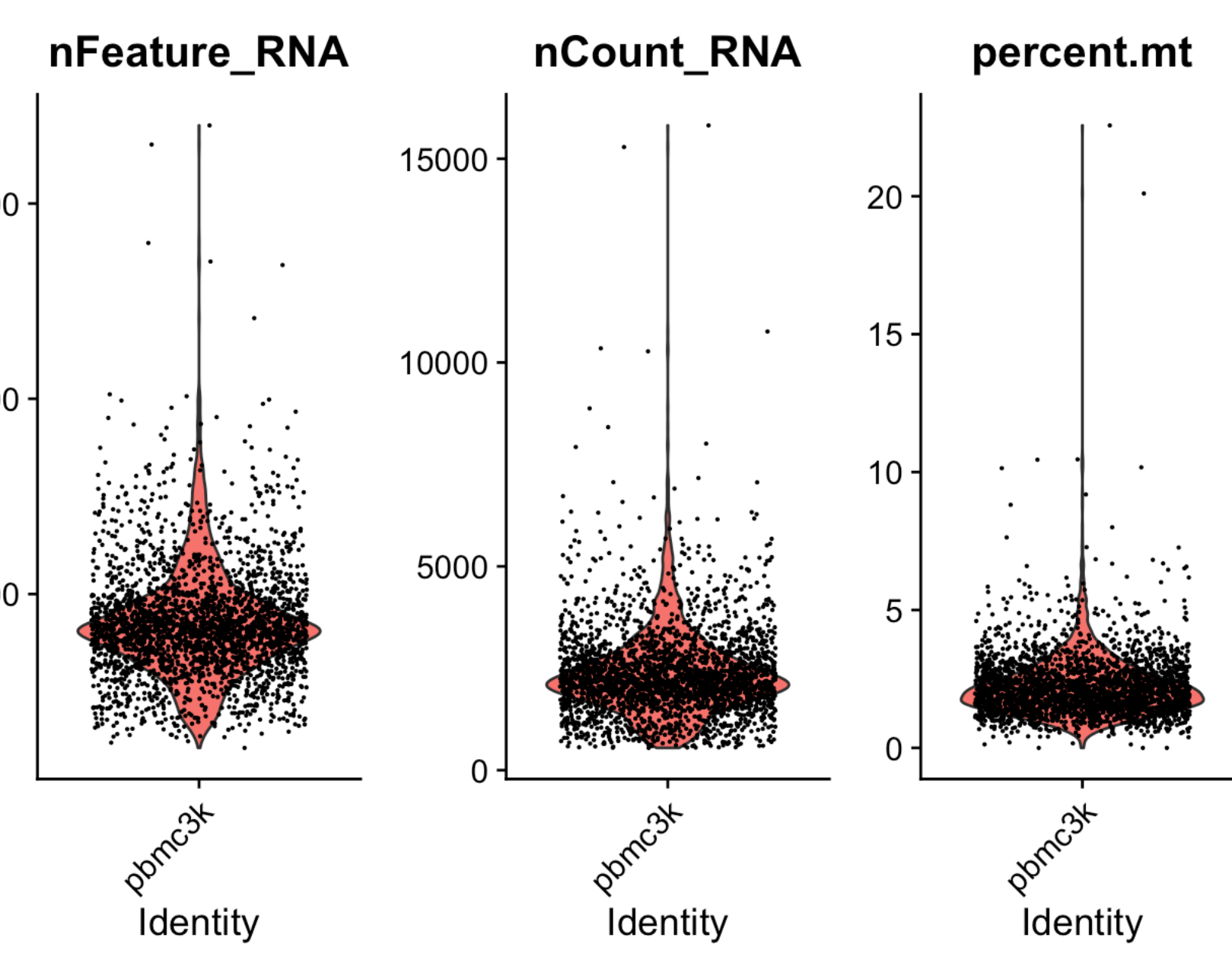
```r
#Show QC metrics for the first 5 cells:
head(pbmc@meta.data, 5)
```

```
##                  orig.ident nCount_RNA nFeature_RNA percent.mt
## AAACATACAACCAC-1     pbmc3k       2419          779  3.0177759
## AAACATTGAGCTAC-1     pbmc3k       4903         1352  3.7935958
## AAACATTGATCAGC-1     pbmc3k       3147         1129  0.8897363
## AAACCGTGCTTCCG-1     pbmc3k       2639          960  1.7430845
## AAACCGTGTATGCG-1     pbmc3k        980          521  1.2244898
```

Below, QC metrics are visualized and then used to filter cells

- Cells are filtered that have unique feature counts over 2,500 or less than 200
- Cells are filtered that have >5% mitochondrial counts

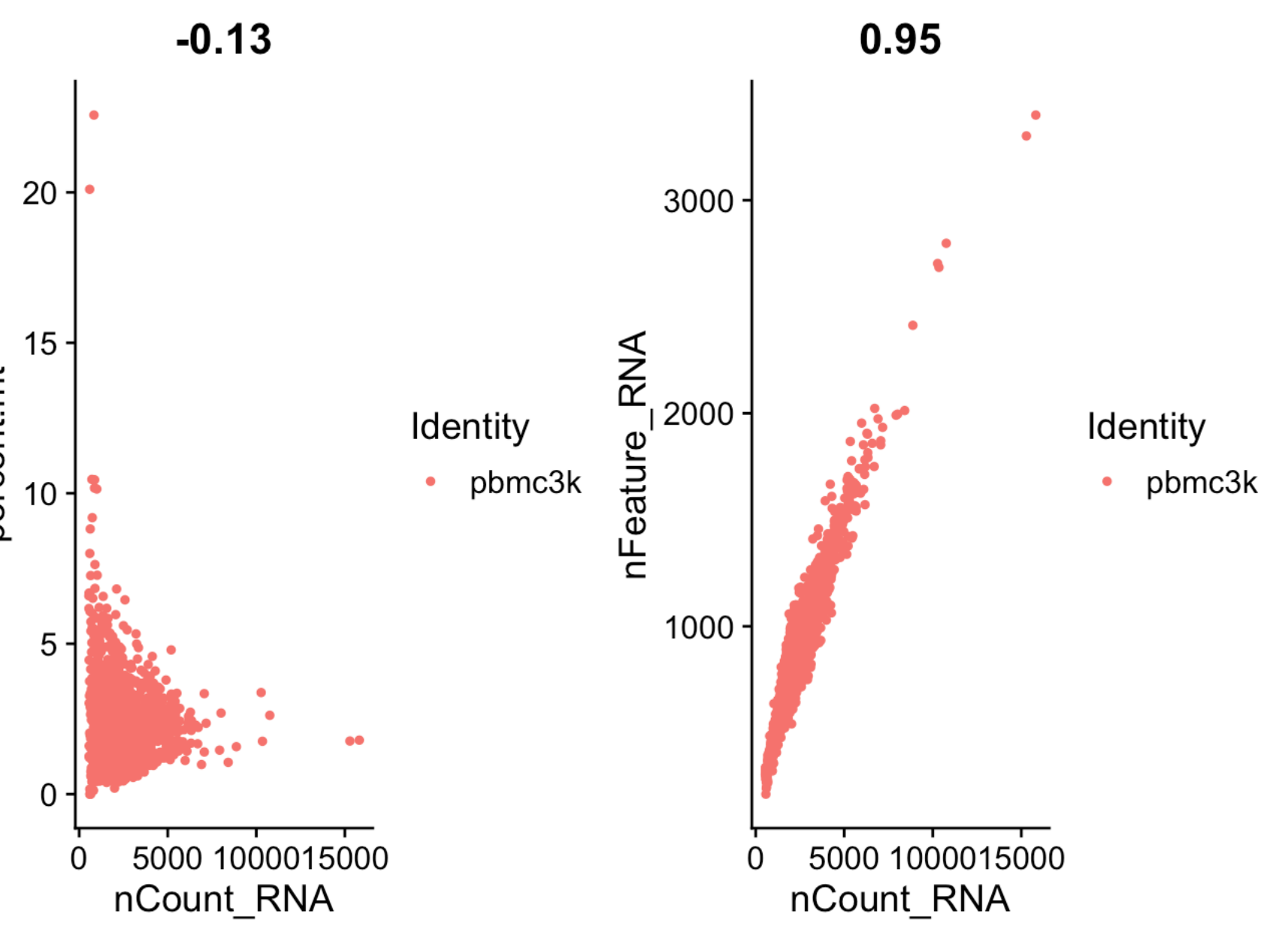
```r
#Visualize QC metrics as violin plots:
VlnPlot(pbmc, features = c("nFeature_RNA", "nCount_RNA", "percent.mt"), ncol = 3)
```




```r
#FeatureScatter is typically used to visualize feature-feature relationships

#It can be used for anything calculated by the object, such as columns in object metadata, PC scores, etc.

plot1 = FeatureScatter(pbmc, feature1 = "nCount_RNA", feature2 = "percent.mt")
plot2 = FeatureScatter(pbmc, feature1 = "nCount_RNA", feature2 = "nFeature_RNA")
plot1 + plot2
```




```r
#Filter the PBMC data set:
pbmc_filtered = subset(pbmc, subset = nFeature_RNA > 200 & nFeature_RNA < 2500 & percent.mt < 5)
```

Normalizing the data

After filtering unwanted cells from the dataset, the next step is to normalize the data set.

By default, a global-scaling normalization method - "LogNormalize" - normalizes the feature expression measurements for each cell by the total expression, multiplies this by a scale factor (10,000 by default), and log-transforms the result. Normalized values are stored in pbmc[["RNA"]]@data.

```r
pbmc_filtered = NormalizeData(pbmc_filtered, normalization.method = "LogNormalize", scale.factor = 10
000)

#The same result from the above command can be achieved with the statement below:
pbmc_filtered = NormalizeData(pbmc_filtered)
```

Identification of highly variable features (feature selection)

Next, a subset of features are calculated which have high cell-to-cell variation in the dataset (they are highly expressed in some cells and lowly expressed in others). Focusing on these genes in downstream analyses helps to highlight the biological signals in single-cell datasets.

Seurat directly models the mean-variance relationship inherent with single-cell data and it is implemented in the FindVariableFeatures() function. By default, 2,000 features are returned per dataset. These will be used in downstream analyses, like PCA.

```r
pbmc_filtered = FindVariableFeatures(pbmc_filtered, selection.method = "vst", nfeatures = 2000)

# Identifies the top 10 most highly variable genes:
top10 = head(VariableFeatures(pbmc_filtered), 10)

# Plot variable features with and without labels:
plot3 = VariableFeaturePlot(pbmc_filtered)
plot4 = LabelPoints(plot = plot3, points = top10, repel = TRUE)
```

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

```r
plot3
plot4
```