scRNAseq Analysis in R with Seurat

Monash Genomics and Bioinformatics Platform (MGBP)

Compiled: November 02, 2024

Contents

1 Getting started **5**
1.1 Summary 5

2 Set up **7**
2.1 Get the workshop material and data 7
2.2 Package Installation 7
2.3 Raw Data 8

3 Schedule **9**

4 Load data **11**
4.1 Setup the Seurat Object 11
4.2 The data set 11
4.3 Different ways of loading the data 12

I Single Cell Analysis **15**

5 QC Filtering **17**
5.1 QC and selecting cells for further analysis 17

6 Normalisation **25**

7 PCAs and UMAPs **27**
7.1 Identification of highly variable features (feature selection) 27
7.2 Scaling the data 28

8 Dimensionality reduction **31**
8.1 Perform linear dimensional reduction 31
8.2 Determine the 'dimensionality' of the dataset 36
8.3 Run non-linear dimensional reduction (UMAP/tSNE) 37
8.4 Save 38

9 Data set integration with Harmony **41**

10 Clustering **49**
10.1 Cluster cells 49
10.2 Choosing a cluster resolution 51

11 Cluster Markers **55**
11.1 Finding differentially expressed features (cluster biomarkers) . . 55
11.2 Use makers to label or find a cluster 66
11.3 Assigning cell type identity to clusters 67

II Futher Analysis **71**

12 SingleR **73**

13 Differential Expression **77**
13.1 Prefiltering 81
13.2 Default Wilcox test 84
13.3 Seurat Negative binomial 86
13.4 Pseudobulk 87

14 Cell cycle Assignment **91**

III Other Resources **97**

15 Resources **99**
15.1 Help and fruther Resources 100
15.2 Data 101
15.3 Analysis Tools 102
15.4 Preprocessing Tools 103

IV Seurat Object **105**

16 Structure **107**
16.1 Load an existing Seurat object 107
16.2 What's in there? 109

17 Acknowledgements **111**

18 Session info **113**

Chapter 1

Getting started

Instructors: Adele Barugahare, Paul Harrison & Laura Perlaza-Jimenez

1.1 Summary

This workshop, conducted by the Monash Genomics and Bioinformatics Platform, will cover how to extend analysis to contemporary third-party tools, Seurat and SingleR. We will be walking through the Single Cell analysis using seurat package and extend this to cell annotation using SingleR.

Important links:

- Installation and Setup instructions
- Slideshow introduction

1.1.1 Recommended Computer Requirements:

System Requirements:

Windows:

- Windows 8.1 (64-bit) or later
- 4GB RAM
- SSD storage highly recommended
- Updated video/display drivers recommended

macOS:

- macOS 10.15 (Catalina) or later
- 4GB RAM
- SSD storage highly recommended

Install latest versions of:

- R
- RStudio
- Seurat
- SeuratWrappers
- SingleR
- Celldex
- Harmony

Chapter 2

Set up

For this workshop you will have available a VM. However, if you which to use your local computer here are the set up instructions:

IMPORTANT: If you have an M1 Mac - Make sure you have gfortan.

2.1 Get the workshop material and data

In RStudio **create a new project**. This ensures all the files for this workshop are placed in their own folder.

Once you've created a new project, run the following R code

2.2 Package Installation

For this workshop, several packages need to be installed.

BiocManager likes to update installed packages, but we have disabled this in the R code below. If your installation fails, then you might need to turn updates on. Note that if there are a large number of packages that BiocManager wants to update it can take several hours.

These instructions have been tested with R version 4.4.1 and Bioconductor version 3.16.

```
## Install required packages for Seurat:
install.packages(c("Seurat", "dplyr", "remotes", "R.utils","harmony"))

## Install required Bioconductor packages
install.packages("BiocManager")
BiocManager::install(c('SingleR', 'celldex',
                       'BiocGenerics', 'DelayedArray', 'DelayedMatrixStats',
```

```
                        'limma', 'S4Vectors', 'SingleCellExperiment',
                        'SummarizedExperiment','edgeR' ),
                      update=FALSE)


## install clustree
install.packages("clustree")
install.packages("RColorBrewer")
```

2.3 Raw Data

```
# where are you? what folder are you working in

getwd()
#> [1] "/mnt/ceph/mbp/servers/bioinformatics-platform/home/lper0012/tasks/training/scR

## Download and untar the data
options(timeout=3600)
download.file(
    "https://bioinformatics.erc.monash.edu/home/lper0012/SingleCellWorkshopData/single

untar("data.tar")
```

Chapter 3

Schedule

Time	Content
10:00	Zoom opened for participants
	Welcome and Housekeeping
	Background on ScRNAseq + R tools
	Seurat Object
11:15	00:15 Breakout activity
	Load data
11:50	00:10 Break
	QC filtering
12:25	00:10 Breakout activity
	QC filtering continued
12:45	00:10 Breakout activity
	Normalisation
	Lunch Break
	PCAs and UMAPs
14:50	00:15 Breakout activity
	Clustering
15:30	00:15 Breakout activity
15:45	00:10 Break
	Cluster Markers
	SingleR
16:25	00:10 Break
	Differential Expression
	Main room Discussion on differential expression
17:00	Final remarks and conclusion

Chapter 4

Load data

4.1 Setup the Seurat Object

4.2 The data set

The dataset used in this workshop is a *modified* version derived from this study (see here). It has been adapted to introduce additional complexity for instructional purposes. *Please refrain from drawing any biological conclusions from this data as it does not represent real experimental results.*

This dataset represents human peripheral blood mononuclear cells (PBMCs), pooled from eight individual donors. Genetic differences among donors enable the identification of some cell doublets, enhancing data complexity. It includes two single-cell sequencing batches, one of which was stimulated with IFN-beta. Additionally, mitochondrial expression levels have been introduced to demonstrate how to interpret and apply mitochondrial thresholds for filtering purposes.

Note: What does the data look like?

What do the input files look like? It varies, but this is the output of the CellRanger pipleine, described here

```
analysis
    clustering
    diffexp
    pca
    tsne
    umap
cloupe.cloupe
filtered_feature_bc_matrix
    barcodes.tsv.gz
```

```
    features.tsv.gz
    matrix.mtx.gz
filtered_feature_bc_matrix.h5.  --> matrix to read in h5 format
metrics_summary.csv
molecule_info.h5
possorted_genome_bam.bam
possorted_genome_bam.bam.bai
raw_feature_bc_matrix
    barcodes.tsv.gz
    features.tsv.gz
    matrix.mtx.gz
raw_feature_bc_matrix.h5
web_summary.html
```

We start by reading in the data. There are several options for loading the data. The `Read10X()` function reads in the output of the cellranger pipeline from 10X, returning a unique molecular identified (UMI) count matrix. The values in this matrix represent the number of molecules for each feature (i.e. gene; row) that are detected in each cell (column).

We next use the count matrix to create a `Seurat` object. The object serves as a container that contains both data (like the count matrix) and analysis (like PCA, or clustering results) for a single-cell dataset. For a technical discussion of the `Seurat` object structure, check out the GitHub Wiki. For example, the count matrix is stored in `pbmc@assays$RNA@counts`.

```
library(dplyr)
library(ggplot2)
library(Seurat)
library(patchwork)
```

4.3 Different ways of loading the data

Example 1. Load your data using the path to the folder: `filtered_feature_bc_matrix` that is in the output folder of your cellranger run using the `Read10X` function.

```
# Load the PBMC dataset
pbmc.data <- Read10X(data.dir = "outs/filtered_feature_bc_matrix")
# Initialize the Seurat object with the raw (non-normalized data).
seurat_object <- CreateSeuratObject(counts = pbmc.data, min.cells = 3, min.features = 2
```

Example 2. Load your data directing the `ReadMtx` function to each of the relevant files in the `filtered_feature_bc_matrix` folder in the outputs from your cellranger run.

```
expression_matrix <- ReadMtx(
  mtx = "outs/filtered_feature_bc_matrix/count_matrix.mtx.gz", features = "outs/filtered_feature_
  cells = "outs/filtered_feature_bc_matrix/barcodes.tsv.gz"
)

seurat_object <- CreateSeuratObject(counts = expression_matrix)
```

Example 3. Load your data from the directing the `ReadMtx` function to each of the relevant files in the `filtered_feature_bc_matrix` folder in the outputs from your cellranger run.

```
pbmc.data <- Read10X_h5("data/filtered_feature_bc_matrix.h5")
metadata <- read.table("data/metadata.txt")



seurat_object <- CreateSeuratObject(counts = pbmc.data ,
                                    assay = "RNA", project = 'pbmc')
#> Warning: Feature names cannot have underscores ('_'),
#> replacing with dashes ('-')



# Use AddMetaData to add new meta data to object
seurat_object  <- AddMetaData(object = seurat_object, metadata = metadata)
```

What does data in a count matrix look like?

```
# Lets examine a few genes in the first thirty cells
pbmc.data[c("CD3D","TCL1A","MS4A1"), 1:30]
#> 3 x 30 sparse Matrix of class "dgCMatrix"
#>   [[ suppressing 30 column names 'AGGGCGCTATTTCC-1', 'GGAGACGATTCGTT-1', 'CACCGTTGTCGTAG-1' ..
#>
#> CD3D  . 2 . . . 5 . . . . 2 . . . 1 . . 1 1 . . . 1 . 1 3 .
#> TCL1A . . . . . . . . . . . . . . . . . . . . . . . 3 . . .
#> MS4A1 . . . . . . . . . . . . . . . . . . . . . . . 1 . . .
#>
#> CD3D  2 . .
#> TCL1A . . .
#> MS4A1 . . .
```

The . values in the matrix represent 0s (no molecules detected). Since most values in an 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.

```
dense.size <- object.size(as.matrix(pbmc.data))
#> Warning in asMethod(object): sparse->dense coercion:
#> allocating vector of size 1.3 GiB
dense.size
#> 1428252816 bytes
sparse.size <- object.size(pbmc.data)
sparse.size
#> 39615768 bytes
dense.size / sparse.size
#> 36.1 bytes
```

Part I

Single Cell Analysis

Chapter 5

QC Filtering

The steps below encompass the standard pre-processing workflow for scRNA-seq data in Seurat. These represent the selection and filtration of cells based on QC metrics, data normalization and scaling, and the detection of highly variable features.

5.1 QC and selecting cells for further analysis

Why do we need to do this?

Low quality cells can add noise to your results leading you to the wrong biological conclusions. Using only good quality cells helps you to avoid this. Reduce noise in the data by filtering out low quality cells such as dying or stressed cells (high mitochondrial expression) and cells with few features that can reflect empty droplets.

Seurat allows you to easily explore QC metrics and filter cells based on any user-defined criteria. A few QC metrics commonly used by the community include

- The number of unique genes detected in each cell.
  - Low-quality cells or empty droplets will often have very few genes
  - Cell doublets or multiplets may exhibit an aberrantly high gene count
- Similarly, the total number of molecules detected within a cell (correlates strongly with unique genes)
- The percentage of reads that map to the mitochondrial genome
  - Low-quality / dying cells often exhibit extensive mitochondrial contamination
  - We calculate mitochondrial QC metrics with the `PercentageFeatureSet()` function, which calculates the percentage of counts originating from

a set of features
  - We use the set of all genes starting with `MT-` as a set of mitochondrial genes

```
# The $ operator can add columns to object metadata.
# This is a great place to stash QC stats
seurat_object$percent.mt <- PercentageFeatureSet(seurat_object, pattern = "^MT-")
```

Challenge: The meta.data slot in the Seurat object

Where are QC metrics stored in Seurat?

- The number of unique genes and total molecules are automatically calculated during `CreateSeuratObject()`
  - You can find them stored in the object meta data

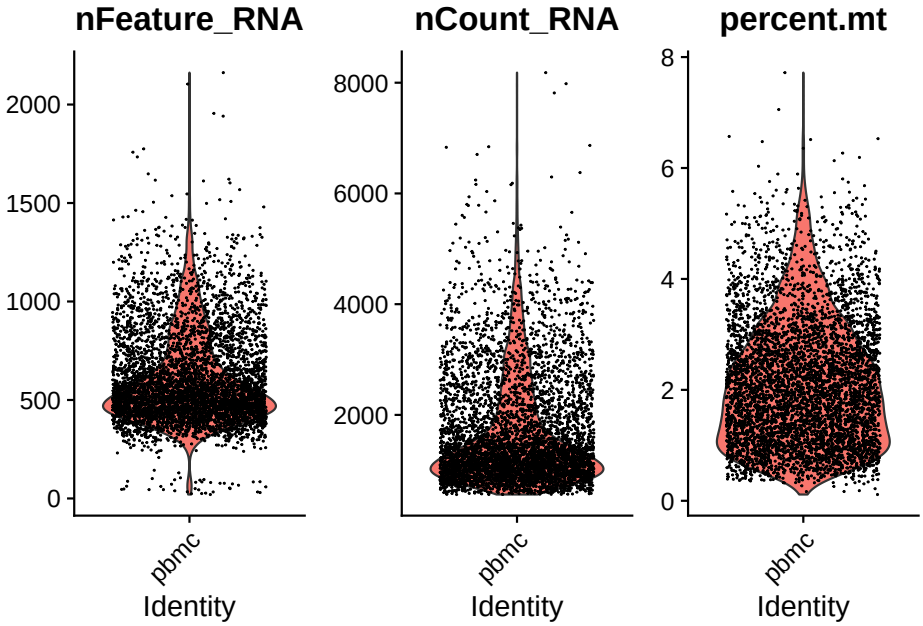
1. What do you notice has changed within the `meta.data` table now that we have calculated mitochondrial gene proportion?

2. Imagine that this is the first of several samples in our experiment. Add a `samplename` column to to the `meta.data` table.

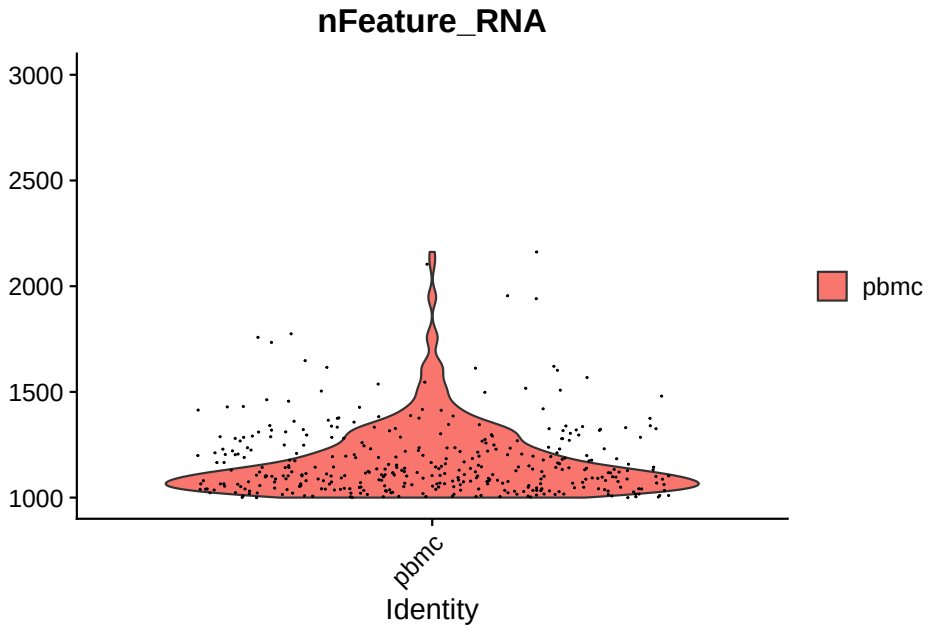
In the example below, we visualize QC metrics, and use these to filter cells.

- We filter cells that have unique feature counts over 2,500 or less than 200
- We filter cells that have >5% mitochondrial counts

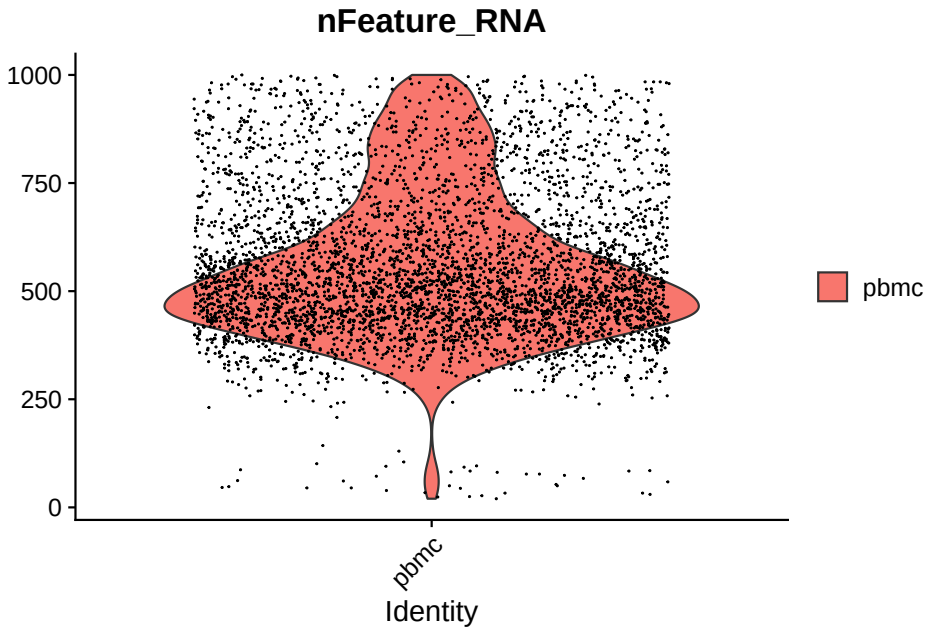
```
#Visualize QC metrics as a violin plot
VlnPlot(seurat_object, features = c("nFeature_RNA", "nCount_RNA", "percent.mt"), ncol =
#> Warning: Default search for "data" layer in "RNA" assay
#> yielded no results; utilizing "counts" layer instead.
```



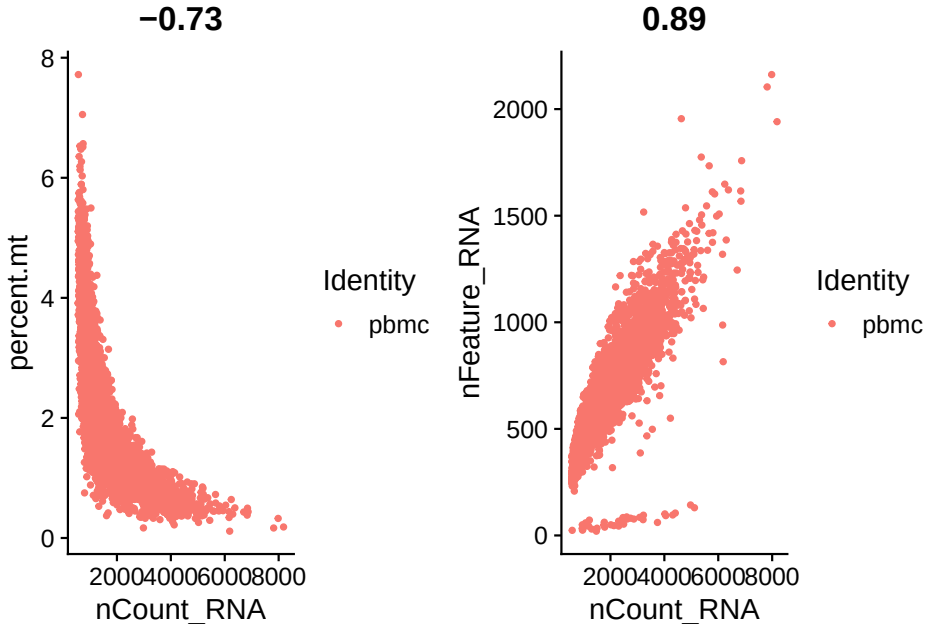

```
# visualize the max and min
VlnPlot(seurat_object, features = "nFeature_RNA")+ scale_y_continuous(limits = c(1000,3000))
#> Warning: Default search for "data" layer in "RNA" assay
#> yielded no results; utilizing "counts" layer instead.
#> Scale for y is already present.
#> Adding another scale for y, which will replace the existing
#> scale.
#> Warning: Removed 4624 rows containing non-finite outside the scale
#> range (`stat_ydensity()`).
#> Warning: Removed 4624 rows containing missing values or values
#> outside the scale range (`geom_point()`).
```




```
VlnPlot(seurat_object, features = "nFeature_RNA",y.max =1000)
#> Warning: Default search for "data" layer in "RNA" assay
#> yielded no results; utilizing "counts" layer instead.
#> Warning: Removed 376 rows containing non-finite outside the scale
#> range (`stat_ydensity()`).
#> Warning: Removed 376 rows containing missing values or values
#> outside the scale range (`geom_point()`).
```

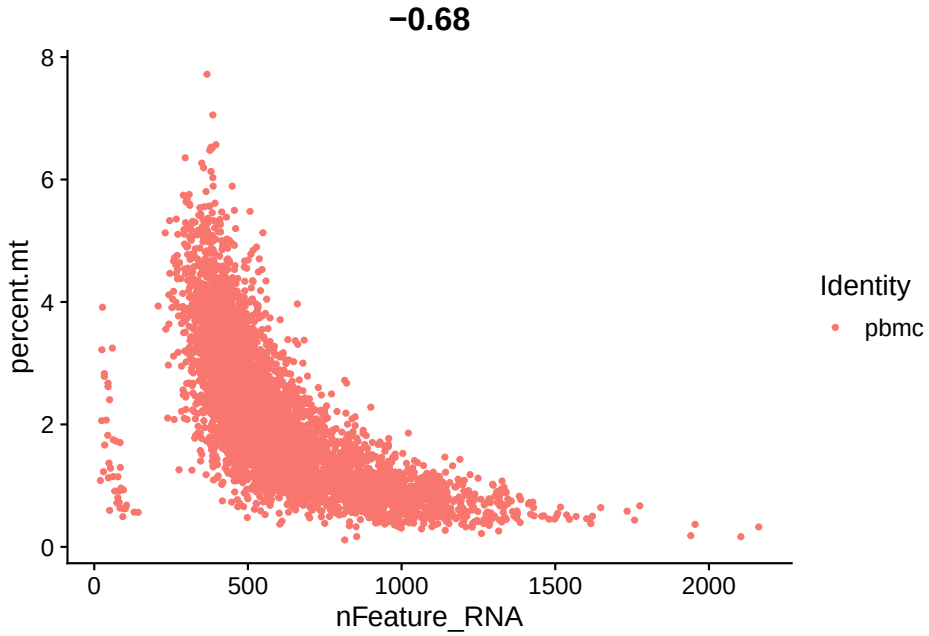



```
# FeatureScatter is typically used to visualize feature-feature relationships,
# but can be used for anything calculated by the object,
# i.e. columns in object metadata, PC scores etc.
plot1 <- FeatureScatter(seurat_object, feature1 = "nCount_RNA", feature2 = "percent.mt")
plot2 <- FeatureScatter(seurat_object, feature1 = "nCount_RNA", feature2 = "nFeature_RNA")
plot1 + plot2
```



Lets look at the number of features (genes) to the percent mitochondrial genes plot.

```
plot3 <- FeatureScatter(seurat_object, feature1 = "nFeature_RNA", feature2 = "percent.m
plot3
```



you can check different thresholds of mito percentage

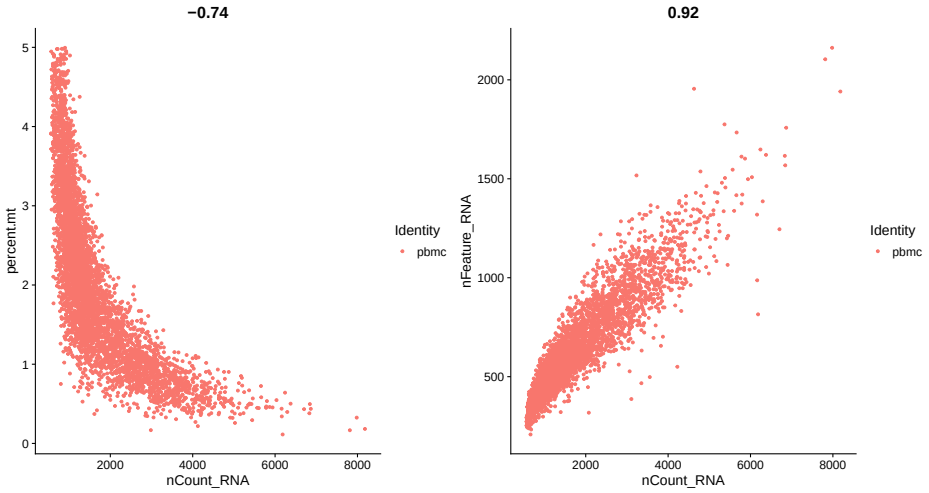
```
#Number of cells left after filters
# percentage of cell with less than 5% mito
round(sum(seurat_object$percent.mt < 5)/dim(seurat_object)[2]*100,2)
#> [1] 98.3

# percentage of cell with less than 2% mito
round((sum(seurat_object$percent.mt < 2)/dim(seurat_object)[2])*100,2)
#> [1] 51.58
```

Okay we are happy with our thresholds for mitochondrial percentage in cells, lets apply them and subset our data. This will remove the cells we think are of poor quality.

```
seurat_object <- subset(seurat_object, subset = nFeature_RNA > 200 & nFeature_RNA < 2500 & percer
```

Lets replot the feature scatters and see what they look like.

```
plot5 <- FeatureScatter(seurat_object, feature1 = "nCount_RNA", feature2 = "percent.mt")
plot6 <- FeatureScatter(seurat_object, feature1 = "nCount_RNA", feature2 = "nFeature_RNA")
plot5 + plot6
```



Challenge: Filter the cells

WE COULD PUT THE BLOOD GENES AS A CHALLENGE HERE

Chapter 6

Normalisation

Why do we need to do this?

The sequencing depth can be different per cell. This can bias the counts of expression showing higher numbers for more sequenced cells leading to the wrong biological conclusions. To correct this the feature counts are normalized.

After removing unwanted cells from the dataset, the next step is to normalize the data. By default, we employ a global-scaling normalization method "LogNormalize" that 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 `seurat_object$RNA@data`.

```
seurat_object <- NormalizeData(seurat_object, normalization.method = "LogNormalize", scale.factor
#> Normalizing layer: counts
```

For clarity, in this previous line of code (and in future commands), we provide the default values for certain parameters in the function call. However, this isn't required and the same behavior can be achieved with:

```
seurat_object <- NormalizeData(seurat_object)
```

There are other options for normalization such as SCTtransform which was popularized in 2019, however Log base normalization continued to be preferred as they perform better see here the for mo details.

Chapter 7

PCAs and UMAPs

7.1 Identification of highly variable features (feature selection)

Why do we need to do this?

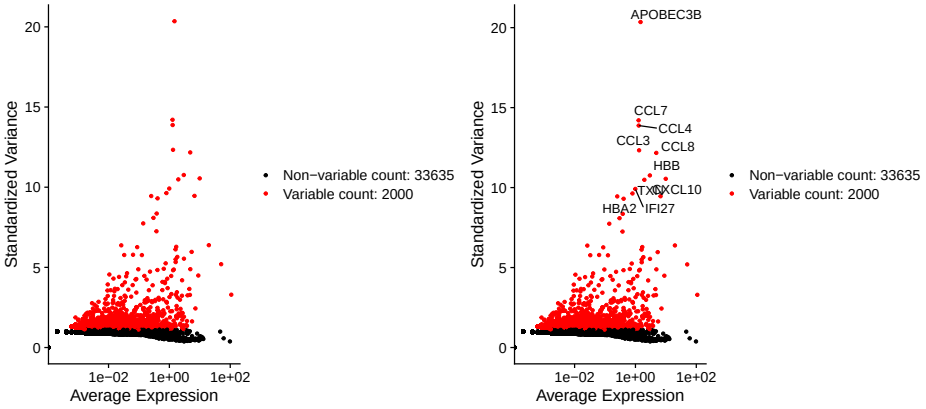
Identifying the most variable features allows retaining the real biological variability of the data and reduce noise in the data.

We next calculate a subset of features that exhibit high cell-to-cell variation in the dataset (i.e, they are highly expressed in some cells, and lowly expressed in others). We and others have found that focusing on these genes in downstream analysis helps to highlight biological signal in single-cell datasets.

Our procedure in Seurat is described in detail here, and improves on previous versions by directly modeling the mean-variance relationship inherent in single-cell data, and is implemented in the `FindVariableFeatures()` function. By default, we return 2,000 features per dataset. These will be used in downstream analysis, like PCA.

```
seurat_object <- FindVariableFeatures(seurat_object, selection.method = 'vst', nfeatures = 2000)
#> Finding variable features for layer counts
# Identify the 10 most highly variable genes
top10 <- head(VariableFeatures(seurat_object), 10)
# plot variable features with and without labels
plot1 <- VariableFeaturePlot(seurat_object)
plot2 <- LabelPoints(plot = plot1, points = top10, repel = TRUE)
#> When using repel, set xnudge and ynudge to 0 for optimal results
plot1 + plot2
```

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



7.2 Scaling the data

Why do we need to do this?

Highly expresed genes can overpower the signal of other less expresed genes with equal importance. Within the same cell the assumption is that the underlying RNA content is constant. Aditionally, If variables are provided in vars.to.regress, they are individually regressed against each feature, and the resulting residuals are then scaled and centered. This step allows controling for cell cycle and other factors that may bias your clustering.

Next, we apply a linear transformation ('scaling') that is a standard pre-processing step prior to dimensional reduction techniques like PCA. The `ScaleData()` function:

- Shifts the expression of each gene, so that the mean expression across cells is 0
- Scales the expression of each gene, so that the variance across cells is 1
  - This step gives equal weight in downstream analyses, so that highly-expressed genes do not dominate
- The results of this are stored in `seurat_object$RNA@scale.data`

```
all.genes <- rownames(seurat_object)
seurat_object <- ScaleData(seurat_object, features = all.genes)
#> Centering and scaling data matrix
```

This step takes too long! Can I make it faster?

Scaling is an essential step in the Seurat workflow, but only on genes that will be used as input to PCA. Therefore, the default in `ScaleData()` is only to perform scaling on the previously identified variable features (2,000 by default). To do this, omit the `features` argument in the previous function call, i.e.

```
# seurat_object <- ScaleData(seurat_object)
```

Your PCA and clustering results will be unaffected. However, Seurat heatmaps (produced as shown below with `DoHeatmap()`) require genes in the heatmap to be scaled, to make sure highly-expressed genes don't dominate the heatmap. To make sure we don't leave any genes out of the heatmap later, we are scaling all genes in this tutorial.

How can I remove unwanted sources of variation, as in Seurat v2?

In `Seurat v2` we also use the `ScaleData()` function to remove unwanted sources of variation from a single-cell dataset. For example, we could 'regress out' heterogeneity associated with (for example) cell cycle stage, or mitochondrial contamination. These features are still supported in `ScaleData()` in `Seurat v3`, i.e.:

```
# seurat_object <- ScaleData(seurat_object, vars.to.regress = 'percent.mt')
```

However, particularly for advanced users who would like to use this functionality, we strongly recommend the use of our new normalization workflow, `SCTransform()`. The method is described in our paper, with a separate vignette using Seurat v3 here. As with `ScaleData()`, the function `SCTransform()` also includes a `vars.to.regress` parameter.

Chapter 8

Dimensionality reduction

Why do we need to do this?

Imagine each gene represents a dimension - or an axis on a plot. We could plot the expression of two genes with a simple scatterplot. But a genome has thousands of genes - how do you collate all the information from each of those genes in a way that allows you to visualise it in a 2 dimensional image. This is where dimensionality reduction comes in, we calculate meta-features that contains combinations of the variation of different genes. From thousands of genes, we end up with 10s of meta-features

8.1 Perform linear dimensional reduction

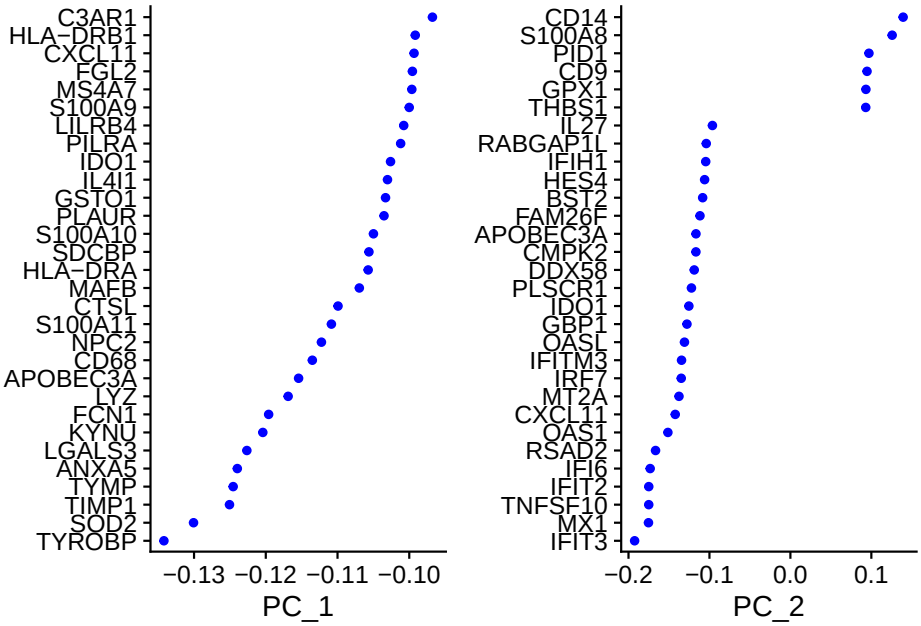
Next we perform PCA on the scaled data. By default, only the previously determined variable features are used as input, but can be defined using `features` argument if you wish to choose a different subset.

```
seurat_object <- RunPCA(seurat_object, features = VariableFeatures(object = seurat_object))
#> PC_ 1
#> Positive:  CCR7, TRAT1, CREM, ALOX5AP, NKG7, TSC22D3, CST7, PASK, GPR171, CD8A
#>     CD8B, ADTRP, SVIP, PRF1, MYC, NOP58, TTC39C, SESN3, CMTM8, C14orf1
#>     GRAP2, GZMA, TARBP1, KLRD1, CD320, GCHFR, AMICA1, TUBA4A, DDIT4, NOB1
#> Negative:  TYROBP, SOD2, TIMP1, TYMP, ANXA5, LGALS3, KYNU, FCN1, LYZ, APOBEC3A
#>     CD68, NPC2, S100A11, CTSL, MAFB, HLA-DRA, SDCBP, S100A10, PLAUR, GSTO1
#>     IL4I1, IDO1, PILRA, LILRB4, S100A9, MS4A7, FGL2, CXCL11, HLA-DRB1, C3AR1
#> PC_ 2
#> Positive:  CD14, S100A8, PID1, CD9, GPX1, THBS1, PLAUR, C19orf59, OSM, CTB-61M7.2
#>     MGST1, S100A9, GAPDH, C5AR1, SLC7A11, ATP6V0B, PPIF, CXCL2, TGFBI, PFN1
#>     LIMS1, OLR1, PLIN2, TIMP1, COTL1, CYP1B1, PDLIM7, SLC11A1, CYP27A1, CEBPB
```

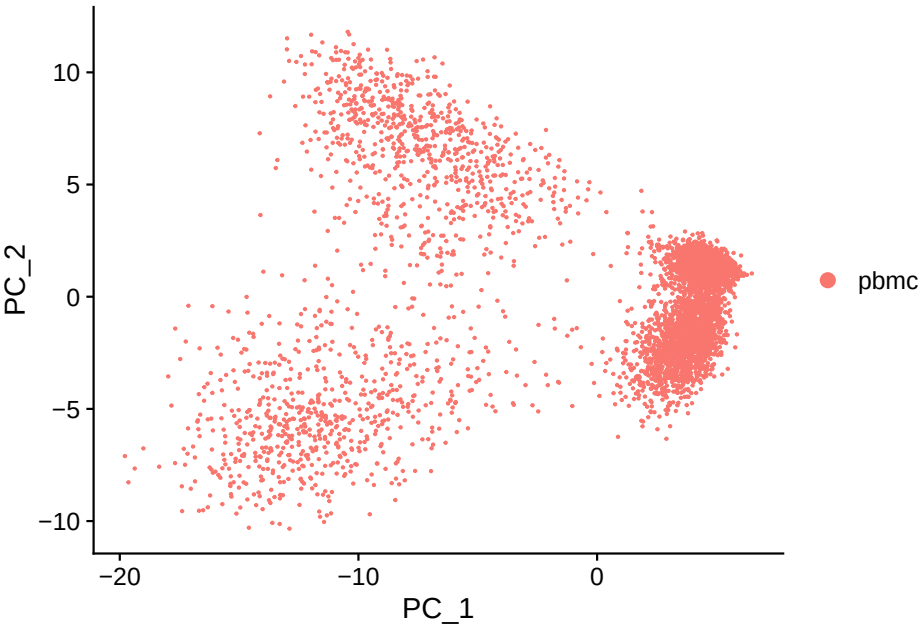
```
#> Negative:  IFIT3, MX1, TNFSF10, IFIT2, IFI6, RSAD2, OAS1, CXCL11, MT2A, IRF7
#>     IFITM3, OASL, GBP1, IDO1, PLSCR1, DDX58, CMPK2, APOBEC3A, FAM26F, BST2
#>     HES4, IFIH1, RABGAP1L, IL27, VAMP5, SERPING1, GMPR, SPATS2L, IRG1, IL4I1
#> PC_ 3
#> Positive:  HLA-DQA1, CD83, HLA-DQB1, HLA-DRB1, HLA-DRA, HLA-DMA, HERPUD1, HSP90AB1,
#>     PKIB, TCF4, FABP5, BANK1, HSPD1, CLIC2, CD79B, FSCN1, HSPH1, CMTM6
#>     SQLE, TNFRSF13B, CD40, ALDH2, LY9, NME1, CKS2, HSP90AA1, HAPLN3, IGLL5
#> Negative:  ANXA1, NKG7, PRF1, GZMA, KLRD1, MT2A, S100A8, S100A9, OASL, CD300E
#>     CST7, C3AR1, CD8A, TYROBP, CD14, S100A12, S100A6, FCGR3A, CTSL, FCN1
#>     MAFB, GCHFR, KLRC1, S100A11, IFI6, C5AR1, AQP9, FPR1, C19orf59, CD8B
#> PC_ 4
#> Positive:  CCR7, ADTRP, TRAT1, MYC, CMTM8, PASK, TARBP1, CTSL, SOCS3, S100A9
#>     S100A8, EMP3, SGTB, TSC22D3, FBLN7, SESN3, GBP1, NEXN, NPC2, MPRIP
#>     HSP90AB1, IL27, S100A12, CCR1, PPA1, FCN1, SOD2, RSAD2, GPR171, HSPD1
#> Negative:  NKG7, CST7, PRF1, KLRD1, GZMA, ID2, KLRC1, TNFRSF18, RAMP1, IGFBP7
#>     ALOX5AP, CD8A, GCHFR, GNG2, GAPDH, FCGR3A, XCL2, ANXA1, PRR5L, OASL
#>     RAB27A, HAVCR2, EIF4EBP1, PKIB, RHOC, GZMK, LINC00996, ADAM8, GSN, BST2
#> PC_ 5
#> Positive:  FCGR3A, MS4A4A, MS4A7, CXCL16, PPM1N, SMPDL3A, AIF1, SERPINA1, ADA, CDKN
#>     CH25H, C3AR1, PLAC8, IL3RA, PILRA, CFD, CLEC12A, VMP1, FGL2, VNN2
#>     FCGR3B, MTMR11, C1QA, MAPKAPK3, LILRB2, COTL1, FAM26F, FPR2, IFNGR2, IL15
#> Negative:  S100A9, MGST1, SLC7A11, S100A8, P2RY6, CCR5, LYZ, FPR3, RSAD2, FABP5
#>     SDS, EMP1, IFI6, CCR1, DHRS9, PRF1, CSTB, MX1, LILRB4, SPHK1
#>     TGFBI, ANXA1, IDO1, CXCL2, CCNA1, HSP90B1, NKG7, CMPK2, S100A12, LPXN
```

Seurat provides several useful ways of visualizing both cells and features that define the PCA, including `VizDimReduction()`, `DimPlot()`, and `DimHeatmap()`

```
# Examine and visualize PCA results a few different ways
print(seurat_object$pca, dims = 1:5, nfeatures = 5)
#> PC_ 1
#> Positive:  CCR7, TRAT1, CREM, ALOX5AP, NKG7
#> Negative:  TYROBP, SOD2, TIMP1, TYMP, ANXA5
#> PC_ 2
#> Positive:  CD14, S100A8, PID1, CD9, GPX1
#> Negative:  IFIT3, MX1, TNFSF10, IFIT2, IFI6
#> PC_ 3
#> Positive:  HLA-DQA1, CD83, HLA-DQB1, HLA-DRB1, HLA-DRA
#> Negative:  ANXA1, NKG7, PRF1, GZMA, KLRD1
#> PC_ 4
#> Positive:  CCR7, ADTRP, TRAT1, MYC, CMTM8
#> Negative:  NKG7, CST7, PRF1, KLRD1, GZMA
#> PC_ 5
#> Positive:  FCGR3A, MS4A4A, MS4A7, CXCL16, PPM1N
#> Negative:  S100A9, MGST1, SLC7A11, S100A8, P2RY6
VizDimLoadings(seurat_object, dims = 1:2, reduction = 'pca')
```



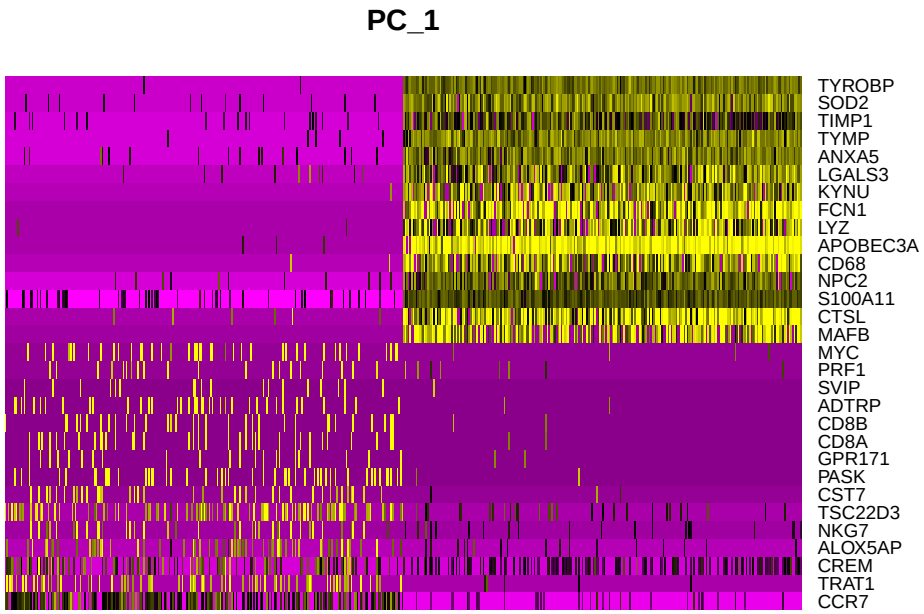

```
DimPlot(seurat_object, reduction = 'pca')
```



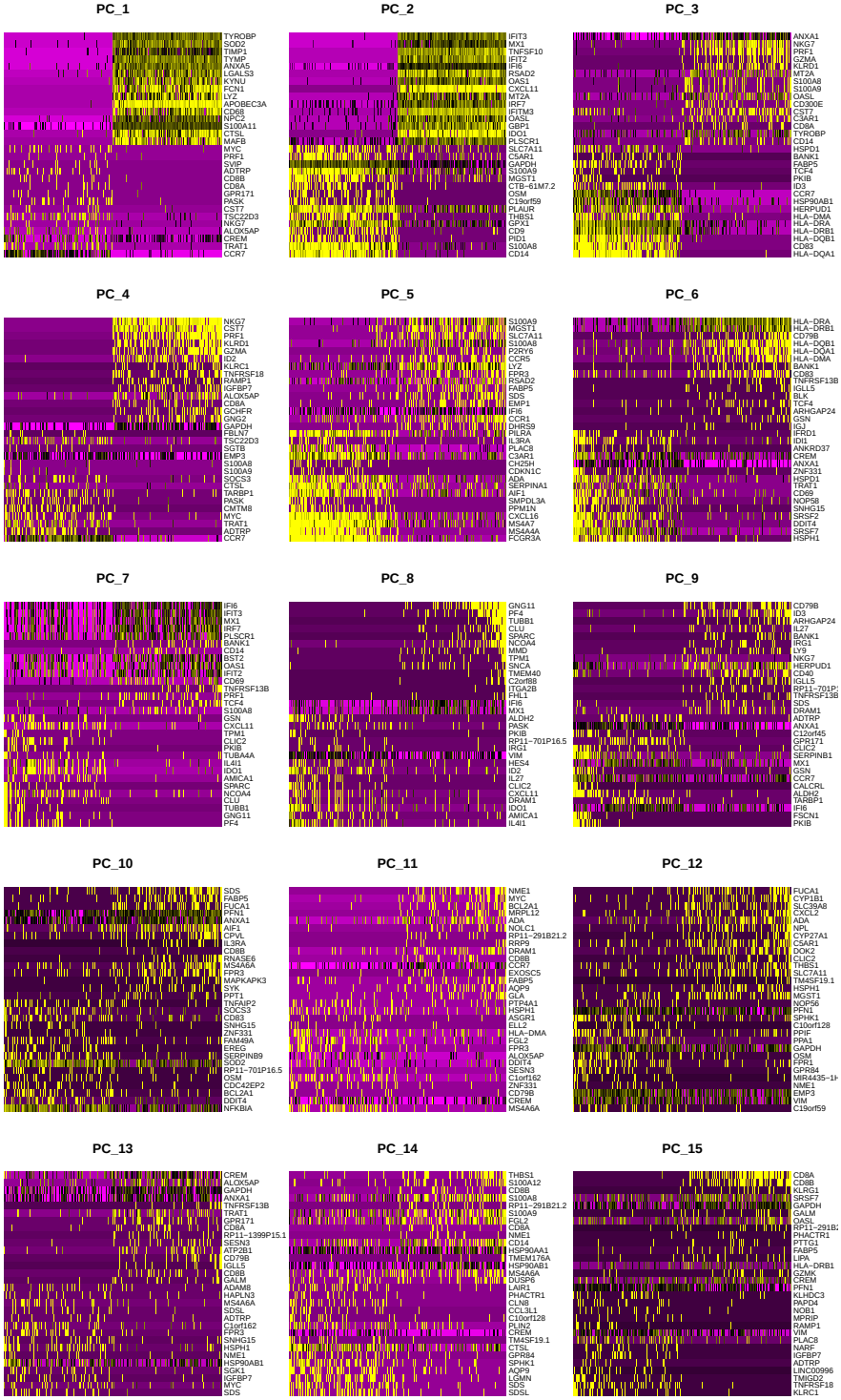
In particular `DimHeatmap()` allows for easy exploration of the primary sources of heterogeneity in a dataset, and can be useful when trying to decide which PCs to include for further downstream analyses. Both cells and features are ordered according to their PCA scores. Setting `cells` to a number plots the

'extreme' cells on both ends of the spectrum, which dramatically speeds plotting for large datasets. Though clearly a supervised analysis, we find this to be a valuable tool for exploring correlated feature sets.

```
DimHeatmap(seurat_object, dims = 1, cells = 500, balanced = TRUE)
```




```
DimHeatmap(seurat_object, dims = 1:15, cells = 500, balanced = TRUE)
```


PC_1
TYROBP
SOD2
TIMP1
TYMP
ANXA5
LGALS3
KYNU
FCN1
LYZ
APOBEC3A
CD68
NPC2
S100A11
CTSL
MAFB
MYC
PRF1
SVIP
ADTRP
CD8B
CD8A
GPR171
PASK
CST7
TSC22D3
NKG7
ALOX5AP
CREM
TRAT1
CCR7
PC_2
IFIT3
MX1
TNFSF10
IFIT2
IFI6
RSAD2
OAS1
CXCL11
MT2A
IRF7
IFITM3
OASL
GBP1
IDO1
PLSCR1
SLC7A11
CSAR1
GAPDH
S100A9
MGST1
CTB−61M7.2
OSM
C19orf59
PLAUR
THBS1
GPX1
CD9
PID1
S100A8
CD14
PC_3
ANXA1
NKG7
PRF1
GZMA
KLRD1
MT2A
S100A8
S100A9
OASL
CD300E
CST7
C3AR1
CD8A
TYROBP
CD14
HSPD1
BANK1
FABP5
TCF4
PKIB
ID3
CCR7
HSP90AB1
HERPUD1
HLA−DMA
HLA−DRA
HLA−DRB1
HLA−DQB1
CD83
HLA−DQA1
PC_4
NKG7
CST7
PRF1
KLRD1
GZMA
ID2
KLRC1
TNFRSF18
RAMP1
IGFBP7
ALOX5AP
CD8A
GCHFR
GNG2
GAPDH
FBLN7
TSC22D3
SGTB
EMP3
S100A8
S100A9
SOCS3
CTSL
TARBP1
PASK
CMTM8
MYC
TRAT1
ADTRP
CCR7
PC_5
S100A9
MGST1
SLC7A11
S100A8
P2RY6
CCR5
LYZ
FPR3
RSAD2
FABP5
SDS
EMP1
IFI6
CCR1
DHRS9
PLSA
IL3RA
PLAC8
C3AR1
CH25H
CDKN1C
ADA
SERPINA1
AIF1
SMPDL3A
PPM1N
CXCL16
MS4A7
MS4A4A
FCGR3A
PC_6
HLA−DRA
HLA−DRB1
CD79B
HLA−DQB1
HLA−DQA1
HLA−DMA
BANK1
CD83
TNFRSF13B
IGLL5
BLK
TCF4
ARHGAP24
GSN
IGJ
IFRD1
IDI1
ANKRD37
CREM
ANXA1
ZNF331
HSPD1
TRAT1
CD69
NOP58
SNHG15
SRSF2
DDIT4
SRSF7
HSPH1
PC_7
IFI6
IFIT3
MX1
IRF7
PLSCR1
BANK1
CD14
BST2
OAS1
IFIT2
CD69
TNFRSF13B
PRF1
TCF4
S100A8
GSN
CXCL11
TPM1
CLIC2
PKIB
TUBA4A
IL4I1
IDO1
AMICA1
SPARC
NCOA4
CLU
TUBB1
GNG11
PF4
PC_8
GNG11
PF4
TUBB1
CLU
SPARC
NCOA4
MMD
TPM1
SNCA
TMEM40
C2orf88
ITGA2B
FHL1
IFI6
MX1
ALDH2
PASK
PKIB
RP11−701P16.5
IRG1
VIM
HES4
ID2
IL27
CLIC2
CXCL11
DRAM1
IDO1
AMICA1
IL4I1
PC_9
CD79B
ID3
ARHGAP24
IL27
BANK1
IRG1
LY9
NKG7
HERPUD1
CD40
IGLL5
RP11−701P:
TNFRSF13B
SDS
DRAM1
ADTRP
ANXA1
C12orf45
GPR171
CLIC2
SERPINB1
MX1
GSN
CCR7
CALCRL
ALDH2
TARBP1
IFI6
FSCN1
PKIB
PC_10
SDS
FABP5
EUCA1
PFN1
ANXA1
AIF1
CPVL
IL3RA
CD8B
RNASE6
MS4A6A
FPR3
MAPKAPK3
SYK
PPT1
TNFAIP2
SOCS3
CD83
SNHG15
ZNF331
FAM49A
EREG
SERPINB9
SOD2
RP11−701P16.5
OSM
CDC42EP2
BCL2A1
DDIT4
NFKBIA
PC_11
NME1
MYC
BCL2A1
MRPL12
ADA
NOLC1
RP11−291B21.2
AQP9
DRAM1
CD8B
CCR7
EXOSC5
FABP5
AQP9
GLA
PTP4A1
HSPH1
ASGR1
ELL2
HLA−DMA
FGL2
FPR3
ALOX5AP
DDIT4
SESN3
C1orf162
ZNF331
CD79B
CREM
MS4A6A
PC_12
EUCA1
CYP1B1
SLC39A8
CXCL2
ADA
NPL
CYP27A1
CSAR1
DOK2
CLIC2
THBS1
SLC7A11
TMEM59.1
HSPH1
MGST1
NOP56
PFN1
SPHK1
C10orf128
PPIF
PPA1
GAPDH
OSM
PRF1
GPR84
MIR4435−1h
NME1
EMP3
VIM
C19orf59
PC_13
CREM
ALOX5AP
GAPDH
ANXA1
TNFRSF13B
TRAT1
GPR171
CD8A
RP11−1399P15.1
SESN3
ATP2B1
CD79B
IGLL5
CD8B
GALM
ADAM8
HAPLN3
MS4A6A
SDSL
ADTRP
C1orf162
FPR3
SNHG15
HSPH1
NME1
HSP90AB1
SGK1
IGFBP7
MYC
SDS
PC_14
THBS1
S100A12
CD8B
S100A8
RP11−291B21.2
S100A9
FGL2
CD8A
NME1
CD14
HSP90AA1
TMEM176A
HSP90AB1
MS4A6A
DUSP6
LAIR1
PHACTR1
CLN8
CCL3L1
C10orf128
PLIN2
CREM
TMEM59.1
CTSL
GPR84
SPHK1
AQP9
LGMN
SDS
SDSL
PC_15
CD8A
CD8B
KLRG1
SRSF7
GAPDH
GALM
OASL
RP11−291B:
PHACTR1
PTTG1
FABP5
LIPA
HLA−DRB1
GZMK
CREM
PFN1
KLHDC3
PAPD4
NOB1
MPRIP
RAMP1
VIM
PLAC8
NAREF
IGFBP7
ADTRP
LINC00996
TMIGD2
TNFRSF18
KLRC1

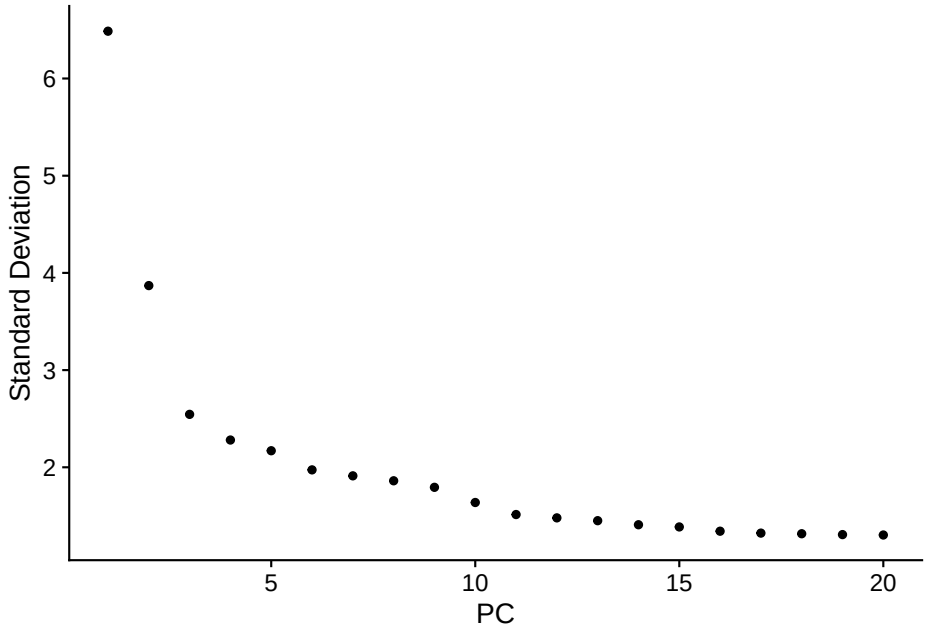
8.2 Determine the 'dimensionality' of the dataset

To overcome the extensive technical noise in any single feature for scRNA-seq data, Seurat clusters cells based on their PCA scores, with each PC essentially representing a 'metafeature' that combines information across a correlated feature set. The top principal components therefore represent a robust compression of the dataset. However, how many components should we choose to include? 10? 20? 100?

Note: The Seurat developers suggest using a JackStraw resampling test to determine this. See Macosko *et al*, and the original seurat_object3 vignette. We're going to use an Elbow Plot instead here, because its much quicker.

An alternative heuristic method generates an 'Elbow plot': a ranking of principle components based on the percentage of variance explained by each one (`ElbowPlot()` function). In this example, we can observe an 'elbow' around PC9-10, suggesting that the majority of true signal is captured in the first 10 PCs.

```
ElbowPlot(seurat_object)
```



Identifying the true dimensionality of a dataset – can be challenging/uncertain for the user. We therefore suggest these three approaches to consider. The first

is more supervised, exploring PCs to determine relevant sources of heterogeneity, and could be used in conjunction with GSEA for example. The second implements a statistical test based on a random null model, but is time-consuming for large datasets, and may not return a clear PC cutoff. The third is a heuristic that is commonly used, and can be calculated instantly. In this example, all three approaches yielded similar results, but we might have been justified in choosing anything between PC 7-12 as a cutoff.

We chose 10 here, but encourage users to consider the following:

- Dendritic cell and NK aficionados may recognize that genes strongly associated with PCs 12 and 13 define rare immune subsets (i.e. MZB1 is a marker for plasmacytoid DCs). However, these groups are so rare, they are difficult to distinguish from background noise for a dataset of this size without prior knowledge.
- We encourage users to repeat downstream analyses with a different number of PCs (10, 15, or even 50!). As you will observe, the results often do not differ dramatically.
- We advise users to err on the higher side when choosing this parameter. For example, performing downstream analyses with only 5 PCs does significantly and adversely affect results.

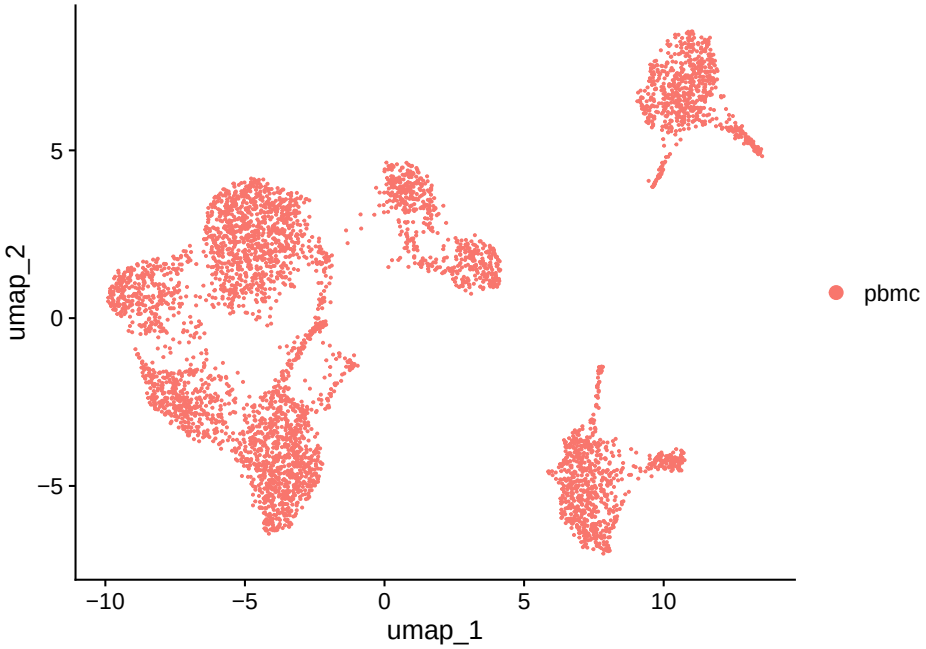
8.3 Run non-linear dimensional reduction (UMAP/tSNE)

Seurat offers several non-linear dimensional reduction techniques, such as tSNE and UMAP, to visualize and explore these datasets. The goal of these algorithms is to learn the underlying manifold of the data in order to place similar cells together in low-dimensional space. Cells within the graph-based clusters determined above should co-localize on these dimension reduction plots. As input to the UMAP and tSNE, we suggest using the same PCs as input to the clustering analysis.

```
# If you haven't installed UMAP, you can do so via reticulate::py_install(packages = "umap-learn"
seurat_object <- RunUMAP(seurat_object, dims = 1:10)
#> Warning: The default method for RunUMAP has changed from calling Python UMAP via reticulate to
#> To use Python UMAP via reticulate, set umap.method to 'umap-learn' and metric to 'correlation
#> This message will be shown once per session
#> 11:08:06 UMAP embedding parameters a = 0.9922 b = 1.112
#> 11:08:06 Read 4877 rows and found 10 numeric columns
#> 11:08:06 Using Annoy for neighbor search, n_neighbors = 30
#> 11:08:06 Building Annoy index with metric = cosine, n_trees = 50
#> 0%   10   20   30   40   50   60   70   80   90   100%
#> [----|----|----|----|----|----|----|----|----|----|
```

```
#> ************************************************|
#> 11:08:07 Writing NN index file to temp file /tmp/RtmplxpVRz/file25a1ccd17e2eb
#> 11:08:07 Searching Annoy index using 1 thread, search_k = 3000
#> 11:08:09 Annoy recall = 100%
#> 11:08:09 Commencing smooth kNN distance calibration using 1 thread with target n_ne
#> 11:08:10 Initializing from normalized Laplacian + noise (using RSpectra)
#> 11:08:10 Commencing optimization for 500 epochs, with 201470 positive edges
#> 11:08:16 Optimization finished
```

```
# note that you can set `label = TRUE` or use the LabelClusters function to help label
DimPlot(seurat_object, reduction = 'umap')
```



Challenge: PC genes

You can plot gene expression on the UMAP with the `FeaturePlot()` function.

Try out some genes that were highly weighted in the principal component analysis. How do they look?

8.4 Save

You can save the object at this point so that it can easily be loaded back in without having to rerun the computationally intensive steps performed above,

or easily shared with collaborators.

```
saveRDS(seurat_object, file = "seurat_object_tutorial_saved.rds")
```

Chapter 9

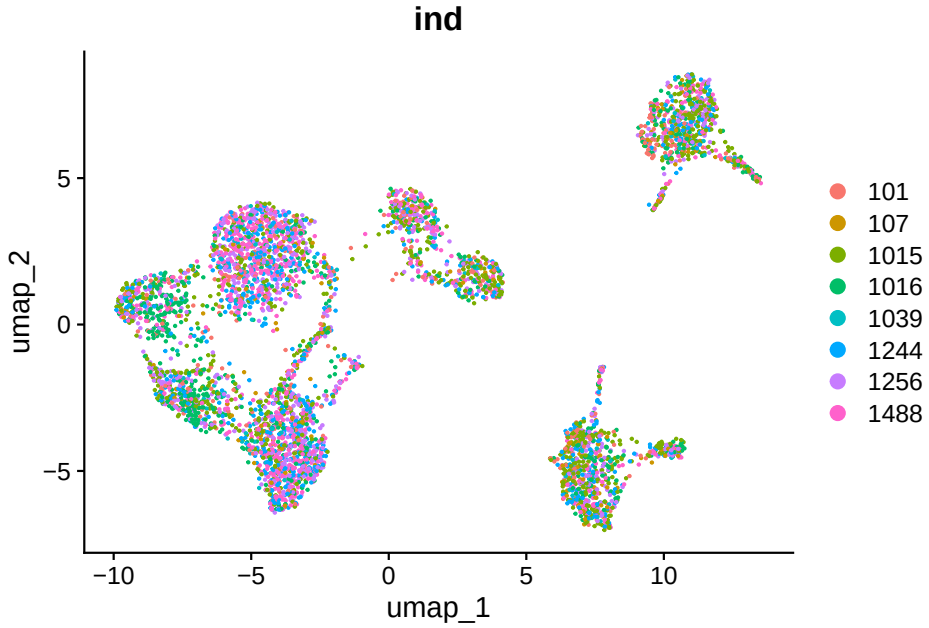
Data set integration with Harmony

Why do we need to do this?

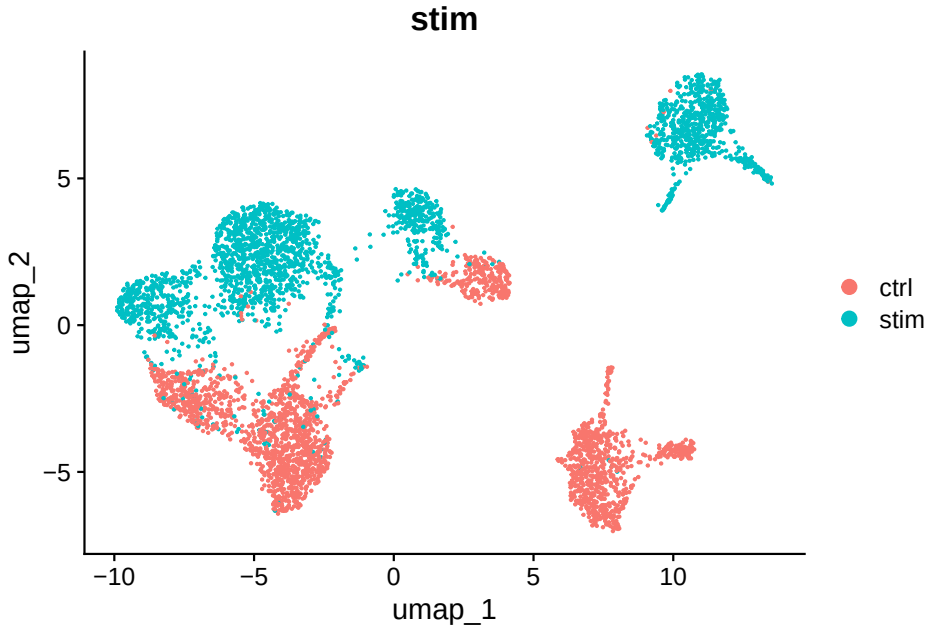
You can have data coming from different samples, batches or experiments and you will need to combine them.

- `ind` identifies a cell as coming from one of 8 individuals.
- `stim` identifies a cell as control or stimulated with IFN-beta.
- `cell` contains the cell types identified by the creators of this data set.
- `multiplets` classifies cells as singlet or doublet.

```
DimPlot(seurat_object, reduction="umap", group.by="ind")
```

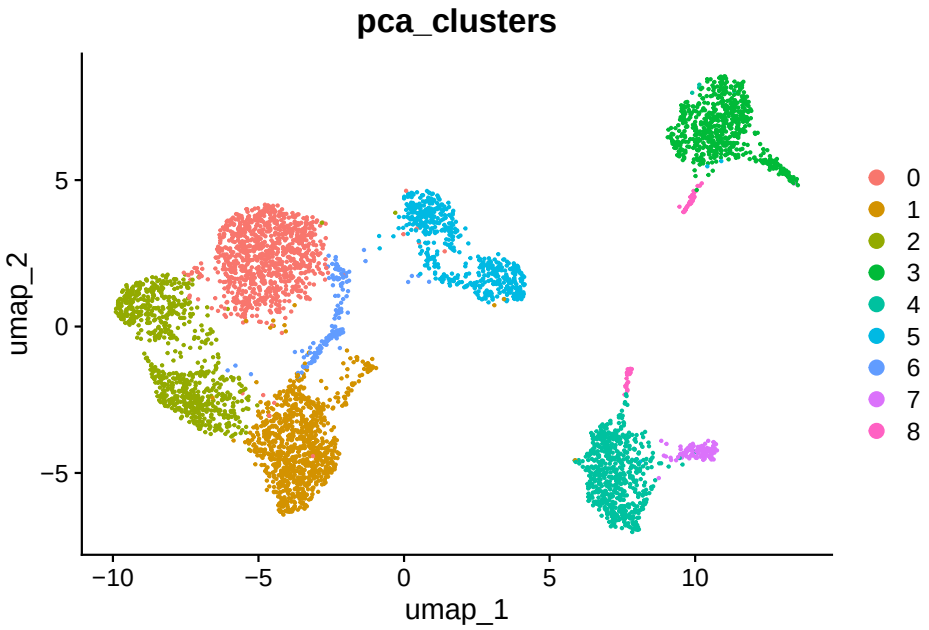



```
DimPlot(seurat_object, reduction="umap", group.by="stim")
```




```
seurat_object<- FindNeighbors(seurat_object, reduction="pca", dims=1:10)
#> Computing nearest neighbor graph
#> Computing SNN
```

```
seurat_object <- FindClusters(seurat_object, resolution=0.25)
#> Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
#>
#> Number of nodes: 4877
#> Number of edges: 171734
#>
#> Running Louvain algorithm...
#> Maximum modularity in 10 random starts: 0.9409
#> Number of communities: 9
#> Elapsed time: 1 seconds
seurat_object$pca_clusters <- seurat_object$seurat_clusters

DimPlot(seurat_object, reduction="umap", group.by="pca_clusters")
```



There is a big difference between unstimulated and stimulated cells. This has split cells of the same type into pairs of clusters. If the difference was simply uniform, we could regress it out (e.g. using `ScaleData(..., vars.to.regress="stim")`). However, as can be seen in the PCA plot, the difference is not uniform and we need to do something cleverer.

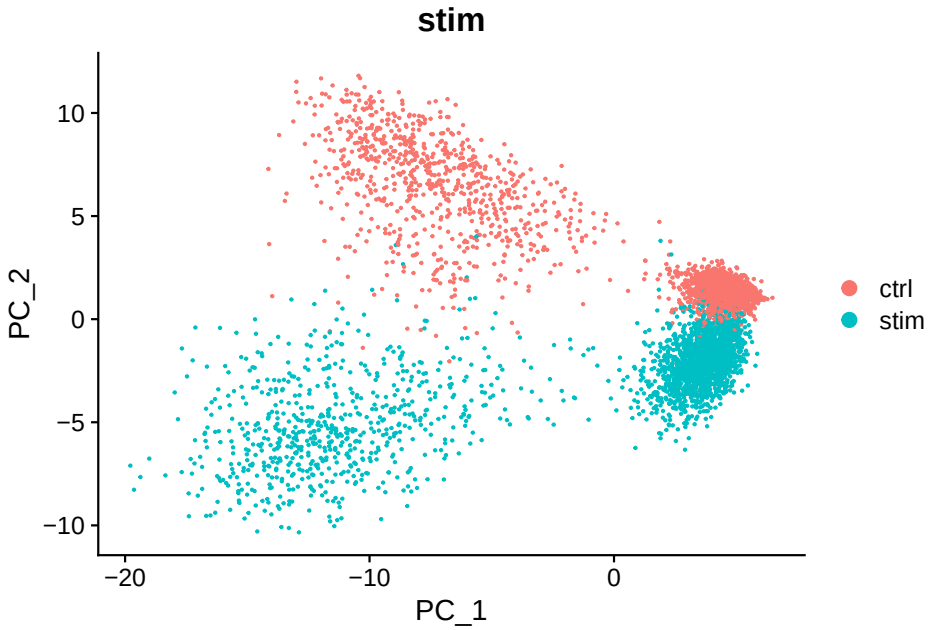
We will use Harmony, which can remove non-uniform effects. We will try to remove both the small differences between individuals and the large difference between the unstimulated and stimulated cells.

Harmony operates only on the PCA scores. The original gene expression levels remain unaltered.

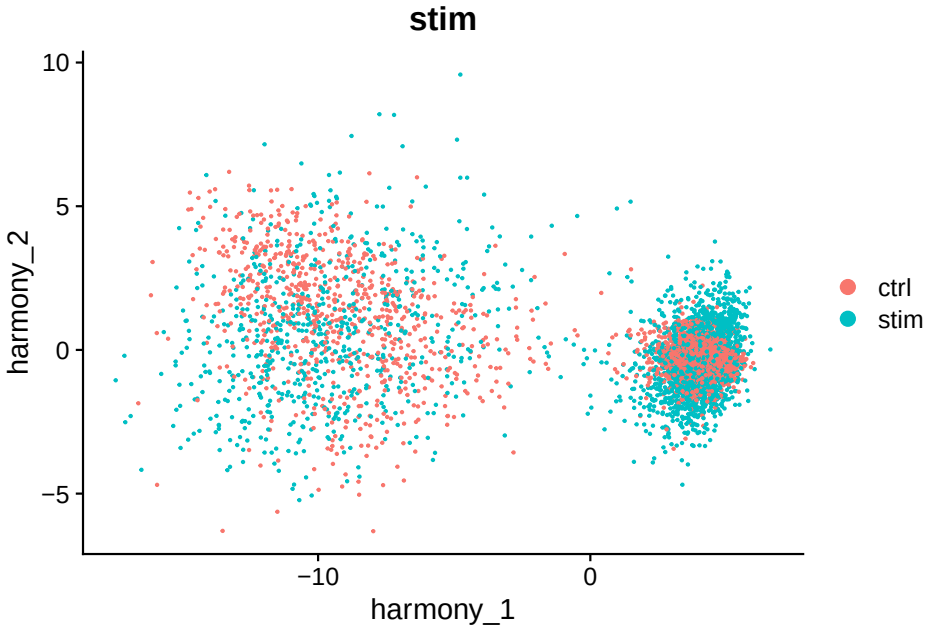
```
library(harmony)
#> Loading required package: Rcpp

seurat_object <- RunHarmony(seurat_object, c("stim", "ind"), reduction="pca",reduction
#> Transposing data matrix
#> Initializing state using k-means centroids initialization
#> Harmony 1/10
#> Harmony 2/10
#> Harmony 3/10
#> Harmony 4/10
#> Harmony converged after 4 iterations
```

This has added a new set of reduced dimensions to the Seurat object, `seurat_object$harmony` which is a modified version of the existing `seurat_object$pca` reduced dimensions. The PCA plot shows a large difference between 'ctrl' and 'stim', but this has been removed in the harmony reduction.

```
DimPlot(seurat_object, reduction="pca", group.by="stim")
```

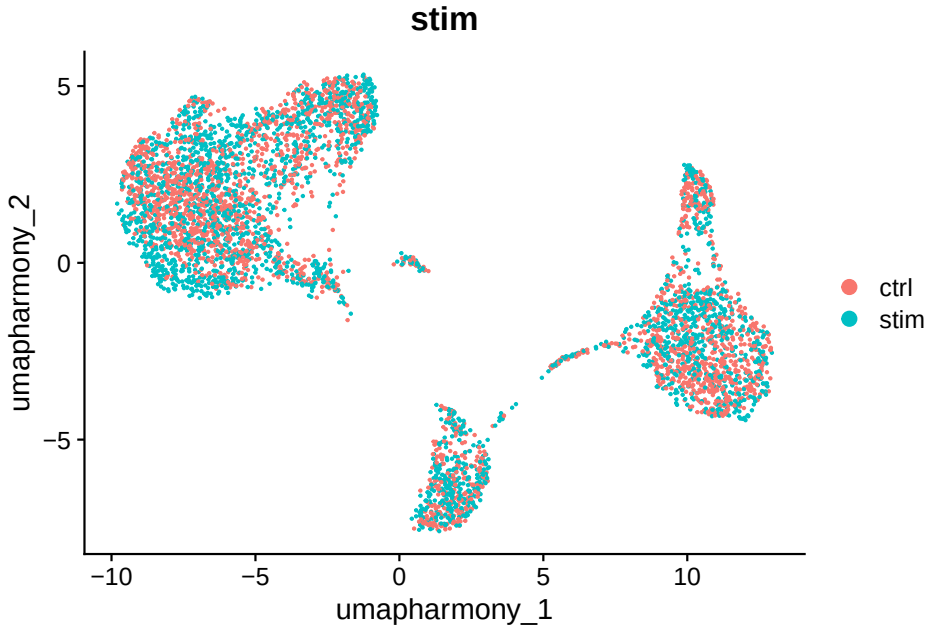



```
DimPlot(seurat_object, reduction="harmony", group.by="stim")
```

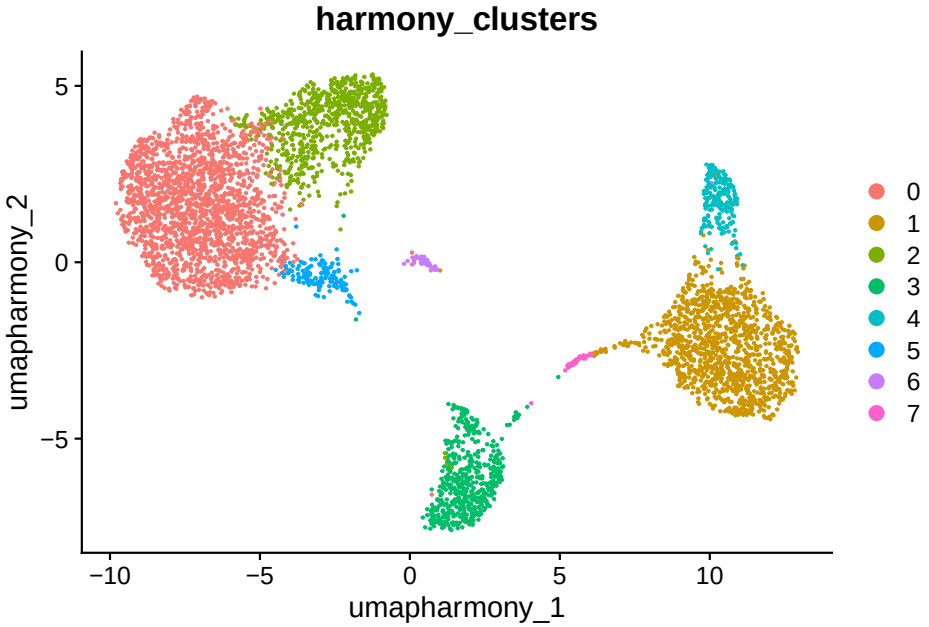


We can use `harmony` the same way we used the `pca` reduction to compute a UMAP layout or to find clusters.

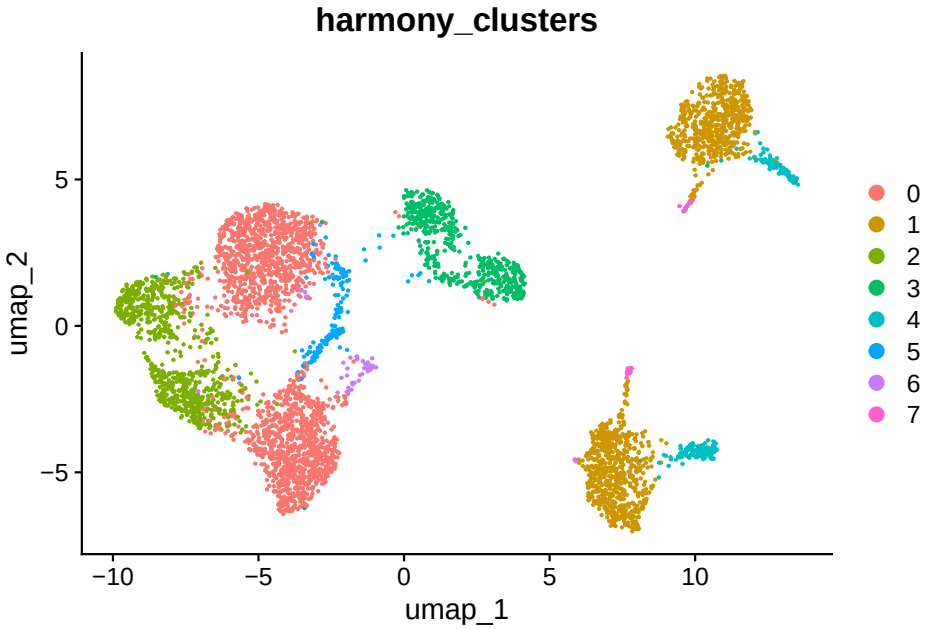
```
seurat_object <- RunUMAP(seurat_object, reduction="harmony", dims=1:10, reduction.name="umap_harm
#> 11:08:39 UMAP embedding parameters a = 0.9922 b = 1.112
#> 11:08:39 Read 4877 rows and found 10 numeric columns
#> 11:08:39 Using Annoy for neighbor search, n_neighbors = 30
#> 11:08:39 Building Annoy index with metric = cosine, n_trees = 50
#> 0%   10   20   30   40   50   60   70   80   90   100%
#> [----|----|----|----|----|----|----|----|----|----|
#> **************************************************|
#> 11:08:40 Writing NN index file to temp file /tmp/RtmplxpVRz/file25a1cc243672f0
#> 11:08:40 Searching Annoy index using 1 thread, search_k = 3000
#> 11:08:42 Annoy recall = 100%
#> 11:08:42 Commencing smooth kNN distance calibration using 1 thread with target n_neighbors = 3
#> 11:08:43 Initializing from normalized Laplacian + noise (using RSpectra)
#> 11:08:43 Commencing optimization for 500 epochs, with 203710 positive edges
#> 11:08:49 Optimization finished

DimPlot(seurat_object, reduction="umap_harmony", group.by="stim")
```




```
seurat_object <- FindNeighbors(seurat_object, reduction="harmony", dims=1:10)
#> Computing nearest neighbor graph
#> Computing SNN
seurat_object <- FindClusters(seurat_object, resolution=0.25)
#> Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
#>
#> Number of nodes: 4877
#> Number of edges: 169427
#>
#> Running Louvain algorithm...
#> Maximum modularity in 10 random starts: 0.9208
#> Number of communities: 8
#> Elapsed time: 1 seconds
seurat_object$harmony_clusters <- seurat_object$seurat_clusters

DimPlot(seurat_object, reduction="umap_harmony", group.by="harmony_clusters")
```




```
DimPlot(seurat_object, reduction="umap", group.by="harmony_clusters")
```



Having found a good set of clusters, we would usually perform differential expression analysis on the original data and include batches/runs/individuals as predictors in the linear model. In this example we could now compare unstimulated and stimulated cells within each cluster. A particularly nice statisti-

cal approach that is possible here would be to convert the counts to pseudo-bulk data for the eight individuals, and then apply a bulk RNA-Seq differential expression analysis method. However there is still the problem that unstimulated and stimulated cells were processed in separate batches.

Chapter 10

Clustering

Why do we need to do this?

Clustering the cells will allow you to visualise the variability of your data, can help to segregate cells into cell types.

10.1 Cluster cells

Seurat v3 applies a graph-based clustering approach, building upon initial strategies in (Macosko *et al*). Importantly, the *distance metric* which drives the clustering analysis (based on previously identified PCs) remains the same. However, our approach to partitioning the cellular distance matrix into clusters has dramatically improved. Our approach was heavily inspired by recent manuscripts which applied graph-based clustering approaches to scRNA-seq data [SNN-Cliq, Xu and Su, Bioinformatics, 2015] and CyTOF data [PhenoGraph, Levine *et al.*, Cell, 2015]. Briefly, these methods embed cells in a graph structure - for example a K-nearest neighbor (KNN) graph, with edges drawn between cells with similar feature expression patterns, and then attempt to partition this graph into highly interconnected 'quasi-cliques' or 'communities'.

As in PhenoGraph, we first construct a KNN graph based on the euclidean distance in PCA space, and refine the edge weights between any two cells based on the shared overlap in their local neighborhoods (Jaccard similarity). This step is performed using the `FindNeighbors()` function, and takes as input the previously defined dimensionality of the dataset (first 10 PCs).

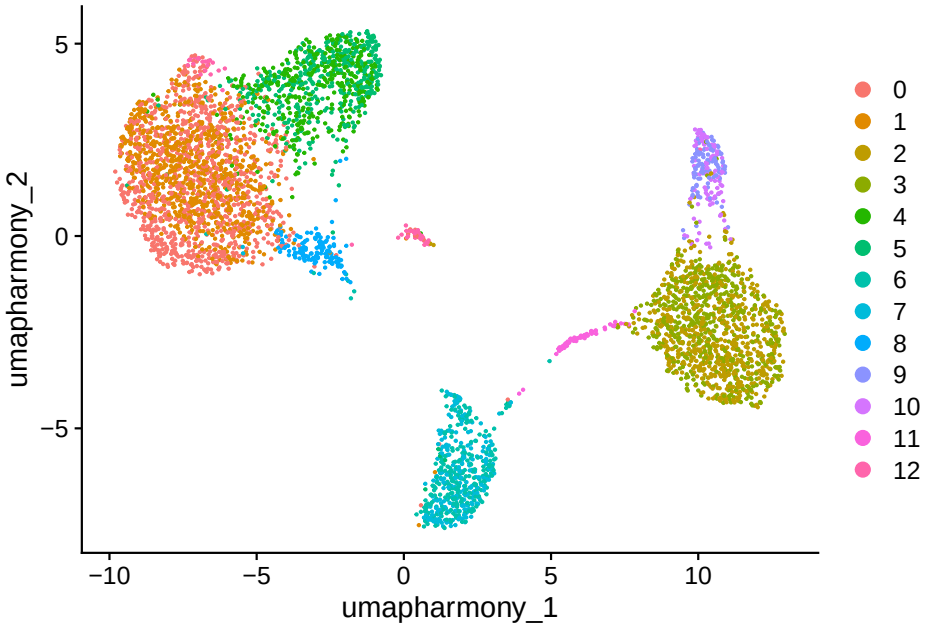
To cluster the cells, we next apply modularity optimization techniques such as the Louvain algorithm (default) or SLM [SLM, Blondel *et al.*, Journal of Statistical Mechanics], to iteratively group cells together, with the goal of optimizing the standard modularity function. The `FindClusters()` function implements

this procedure, and contains a resolution parameter that sets the 'granularity' of the downstream clustering, with increased values leading to a greater number of clusters. We find that setting this parameter between 0.4-1.2 typically returns good results for single-cell datasets of around 3K cells. Optimal resolution often increases for larger datasets. The clusters can be found using the `Idents()` function.

```
seurat_object <- FindNeighbors(seurat_object, dims = 1:10)
#> Computing nearest neighbor graph
#> Computing SNN
seurat_object <- FindClusters(seurat_object, resolution = 0.5)
#> Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
#>
#> Number of nodes: 4877
#> Number of edges: 171734
#>
#> Running Louvain algorithm...
#> Maximum modularity in 10 random starts: 0.9108
#> Number of communities: 13
#> Elapsed time: 0 seconds
# Look at cluster IDs of the first 5 cells
head(Idents(seurat_object), 5)
#> AGGGCGCTATTTCC-1 GGAGACGATTCGTT-1 CACCGTTGTCGTAG-1
#>               3               0               9
#> TATCGTACACGCAT-1 TACGAGACCTATTC-1
#>               6               0
#> Levels: 0 1 2 3 4 5 6 7 8 9 10 11 12
```

Check out the clusters.

```
DimPlot(seurat_object,reduction = "umap_harmony")
```




```
# Equivalent to
# DimPlot(seurat_object,reduction="umap", group.by="seurat_clusters")
# DimPlot(seurat_object,reduction="umap", group.by="RNA_snn_res.0.5")
```

Challenge: Try different cluster settings

Run `FindNeighbours` and `FindClusters` again, with a different number of dimensions or with a different resolution. Examine the resulting clusters using `DimPlot`.

To maintain the flow of this tutorial, please put the output of this exploration in a different variable, such as `seurat_object2`!

10.2 Choosing a cluster resolution

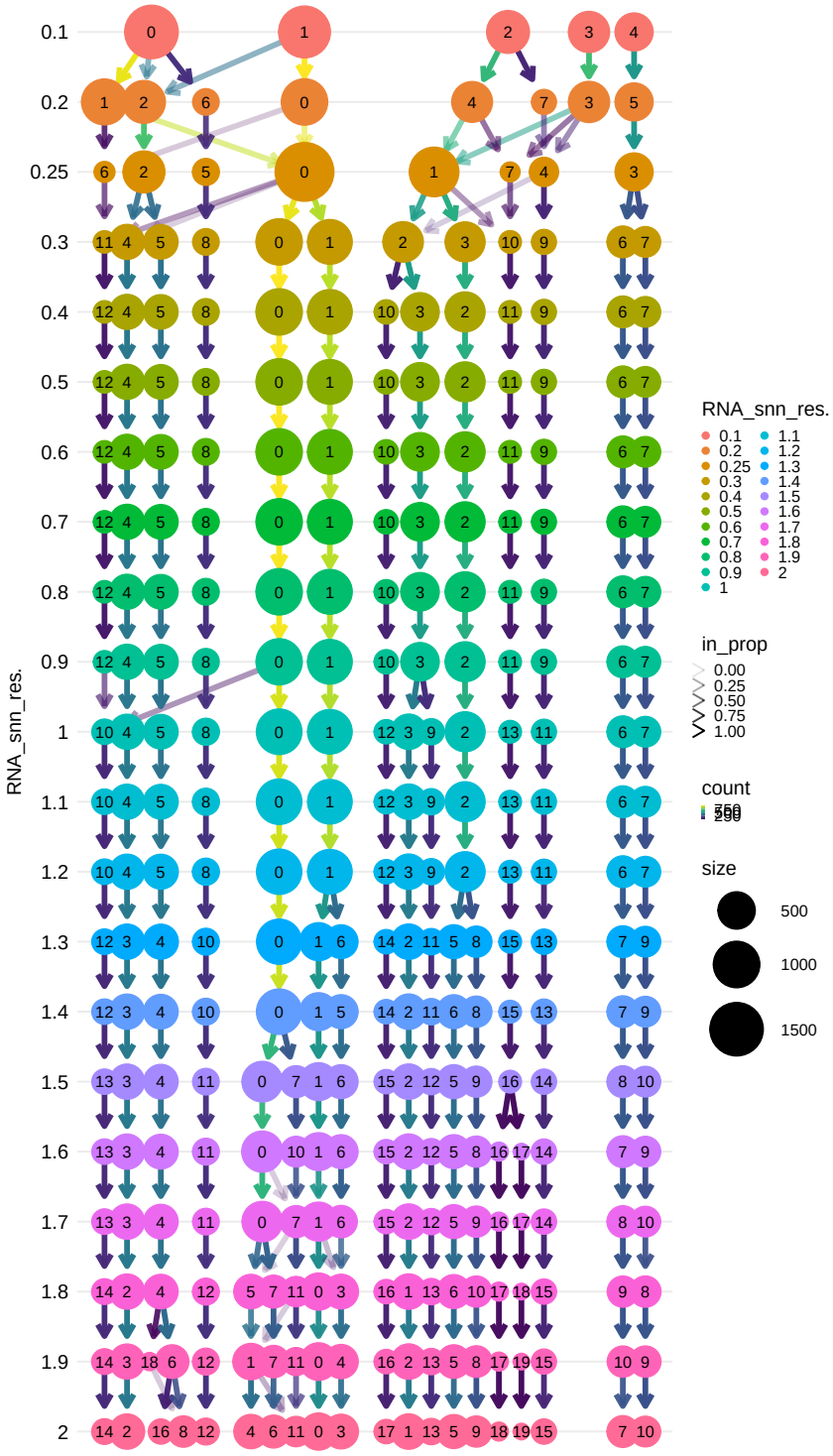
Its a good idea to try different resolutions when clustering to identify the variability of your data.

```
resolution = 2
seurat_object <- FindClusters(object = seurat_object, reduction = "umap_harmony", resolution = se
    dims = 1:10)

# the different clustering created
```

```
names(seurat_object@meta.data)

# Look at cluster IDs of the first 5 cells
head(Idents(seurat_object), 5)
```

Plot a clustree to decide how many clusters you have and what resolution capture them.

```
library(clustree)
#> Loading required package: ggraph
#>
#> Attaching package: 'ggraph'
#> The following object is masked from 'package:sp':
#>
#>     geometry
clustree(seurat_object, prefix = "RNA_snn_res.",show_axis=TRUE) + theme(legend.key.siz
```

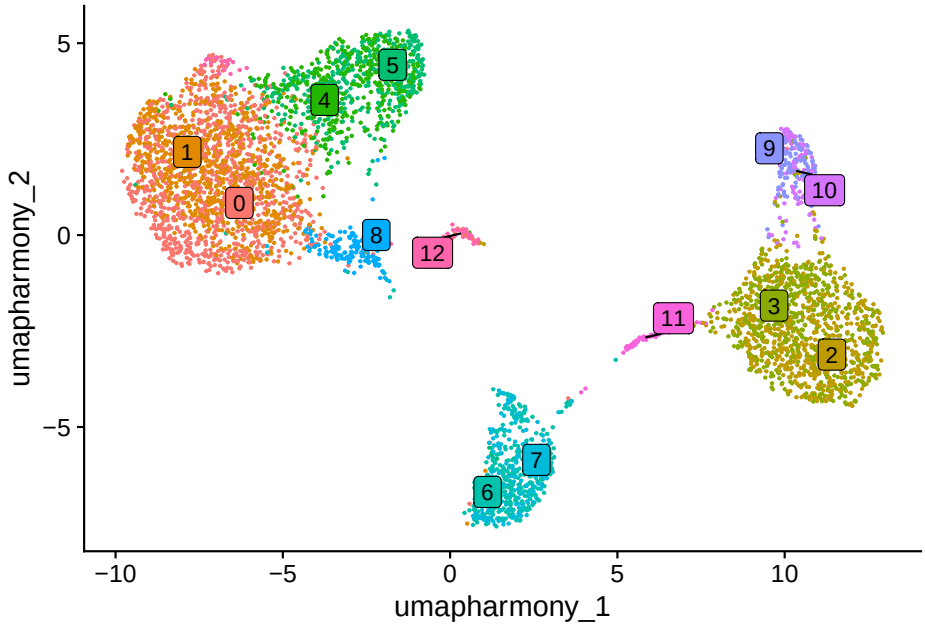

RNA_snn_res.
0.1
0.2
0.25
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
1.1
1.2
1.3
1.4
1.5
1.6
1.7
1.8
1.9
2
RNA_snn_res.
0.1
0.2
0.25
0.3
0.4
0.5
0.6
0.7
0.8
0.9
1
1.1
1.2
1.3
1.4
1.5
1.6
1.7
1.8
1.9
2
in_prop
0.00
0.25
0.50
0.75
1.00
count
size
500
1000
1500

Name cells with the corresponding cluster name at the resolution you pick. This case we are happy with 0.5.

```
# The name of the cluster is prefixed with 'RNA_snn_res' and the number of the resolut
Idents(seurat_object) <- seurat_object$RNA_snn_res.0.5
```

Plot the UMAP with colored clusters with Dimplot

```
DimPlot(seurat_object, reduction = "umap_harmony", label = TRUE, repel = TRUE, label.bo
```



Chapter 11

Cluster Markers

Why do we need to do this?

Single cell data helps to segragate cell types. Use markers to identify cell types. warning: In this example the cell types/markers are well known and making this step easy, but in reality this step needs the experts curation.

11.1 Finding differentially expressed features (cluster biomarkers)

Seurat can help you find markers that define clusters via differential expression. By default, it identifies positive and negative markers of a single cluster (specified in `ident.1`), compared to all other cells. `FindAllMarkers()` automates this process for all clusters, but you can also test groups of clusters vs. each other, or against all cells.

The `min.pct` argument requires a feature to be detected at a minimum percentage in either of the two groups of cells, and the thresh.test argument requires a feature to be differentially expressed (on average) by some amount between the two groups. You can set both of these to 0, but with a dramatic increase in time - since this will test a large number of features that are unlikely to be highly discriminatory. As another option to speed up these computations, `max.cells.per.ident` can be set. This will downsample each identity class to have no more cells than whatever this is set to. While there is generally going to be a loss in power, the speed increases can be significant and the most highly differentially expressed features will likely still rise to the top.

```
# find all markers of cluster 2
cluster2.markers <- FindMarkers(seurat_object, ident.1 = 2, min.pct = 0.25)
```

```
#> For a (much!) faster implementation of the Wilcoxon Rank Sum Test,
#> (default method for FindMarkers) please install the presto package
#> --------------------------------------------
#> install.packages('devtools')
#> devtools::install_github('immunogenomics/presto')
#> --------------------------------------------
#> After installation of presto, Seurat will automatically use the more
#> efficient implementation (no further action necessary).
#> This message will be shown once per session
head(cluster2.markers, n = 5)
#>          p_val avg_log2FC pct.1 pct.2 p_val_adj
#> IL8          0   4.906707 0.866 0.080         0
#> S100A8       0   4.888621 0.840 0.079         0
#> C15orf48     0   2.264979 0.971 0.221         0
#> S100A4       0   2.984441 0.927 0.243         0
#> CD14         0   4.446820 0.743 0.060         0
# find all markers distinguishing cluster 5 from clusters 0 and 3
cluster5.markers <- FindMarkers(seurat_object, ident.1 = 5, ident.2 = c(0, 3), min.pct
head(cluster5.markers, n = 5)
#>              p_val avg_log2FC pct.1 pct.2     p_val_adj
#> NKG7 4.579017e-283   6.232442 0.845 0.042 1.631733e-278
#> GZMB 2.084977e-248   6.690074 0.738 0.027 7.429816e-244
#> GNLY 2.943011e-231   6.321798 0.753 0.050 1.048742e-226
#> PRF1 2.376253e-223   6.073309 0.703 0.036 8.467777e-219
#> CCL5 2.223689e-204   4.779416 0.806 0.106 7.924115e-200
# find markers for every cluster compared to all remaining cells, report only the posi
seurat_object.markers <- FindAllMarkers(seurat_object, only.pos = TRUE, min.pct = 0.25
#> Calculating cluster 0
#> Calculating cluster 1
#> Calculating cluster 2
#> Calculating cluster 3
#> Calculating cluster 4
#> Calculating cluster 5
#> Calculating cluster 6
#> Calculating cluster 7
#> Calculating cluster 8
#> Calculating cluster 9
#> Calculating cluster 10
#> Calculating cluster 11
#> Calculating cluster 12
seurat_object.markers %>% group_by(cluster) %>% slice_max(n = 2, order_by = avg_log2FC)
#> # A tibble: 26 x 7
#> # Groups:   cluster [13]
#>        p_val avg_log2FC pct.1 pct.2 p_val_adj cluster gene
#>        <dbl>      <dbl> <dbl> <dbl>     <dbl> <fct>   <chr>
```

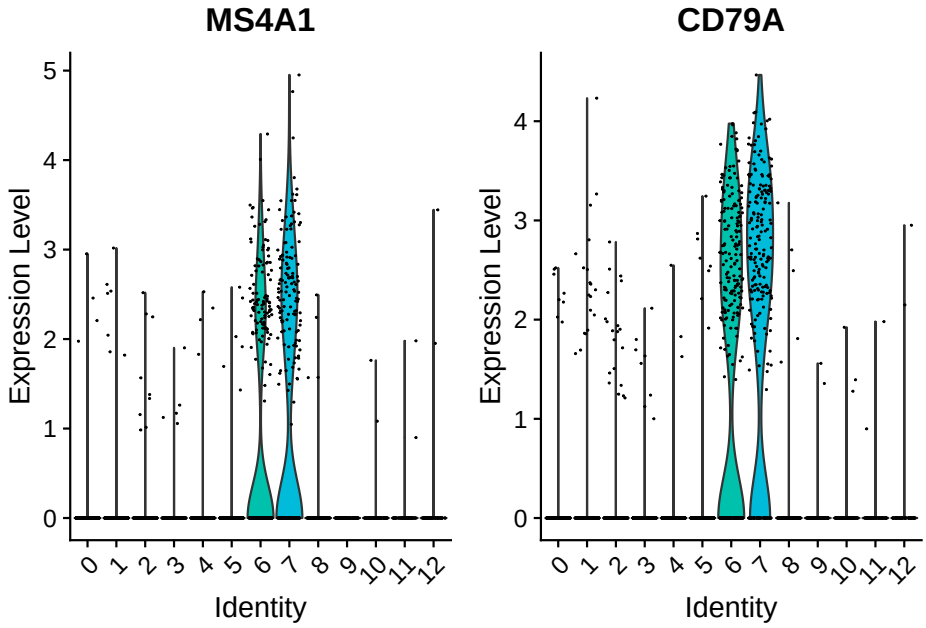
```
#>  1 2.02e-174       1.92 0.706 0.276 7.19e-170 0       SELL
#>  2 4.80e-218       1.77 0.873 0.419 1.71e-213 0       GIMAP7
#>  3 2.26e- 54       1.98 0.258 0.081 8.06e- 50 1       LEF1
#>  4 1.39e- 79       1.78 0.41  0.143 4.96e- 75 1       LCK
#>  5 0               6.24 0.484 0.01  0         2       CLEC5A
#>  6 0               5.76 0.454 0.014 0         2       VCAN
#>  7 0               7.25 0.899 0.068 0         3       CCL8
#>  8 7.57e-136       6.18 0.268 0.024 2.70e-131 3       APOBE~
#>  9 4.38e-177       4.29 0.407 0.039 1.56e-172 4       FGFBP2
#> 10 0               3.60 0.859 0.118 0         4       NKG7
#> # i 16 more rows
```

Seurat has several tests for differential expression which can be set with the test.use parameter (see our DE vignette for details). For example, the ROC test returns the 'classification power' `abs(AUC-0.5)*2` for any individual marker, ranging from 0 = random to 1 = perfect.

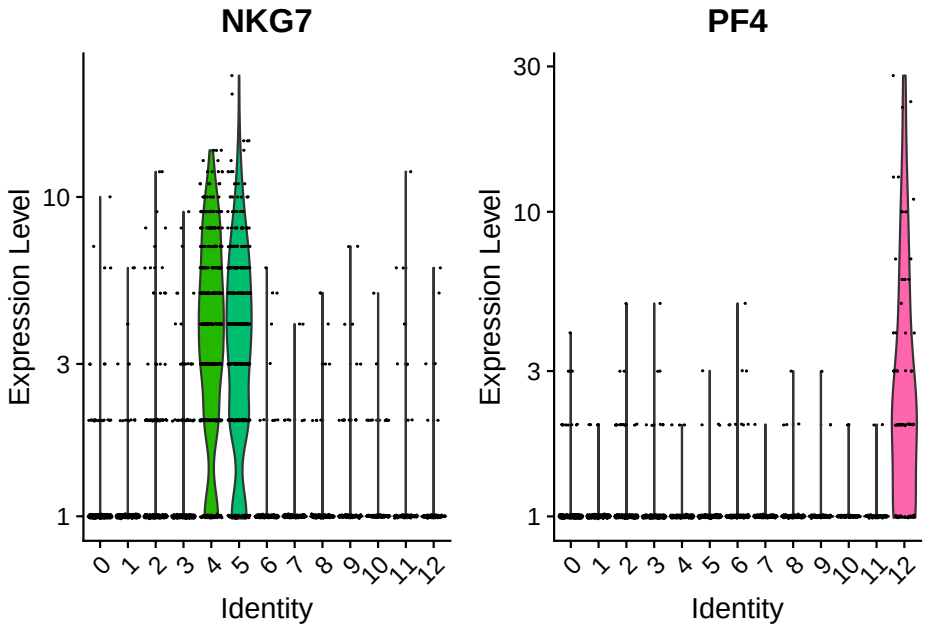
```
cluster0.markers <- FindMarkers(seurat_object, ident.1 = 0, logfc.threshold = 0.25, test.use = "r
```

We include several tools for visualizing marker expression. `VlnPlot()` (shows expression probability distributions across clusters), and `FeaturePlot()` (visualizes feature expression on a tSNE or PCA plot) are our most commonly used visualizations. We also suggest exploring `RidgePlot()`, `CellScatter()`, and `DotPlot()` as additional methods to view your dataset.

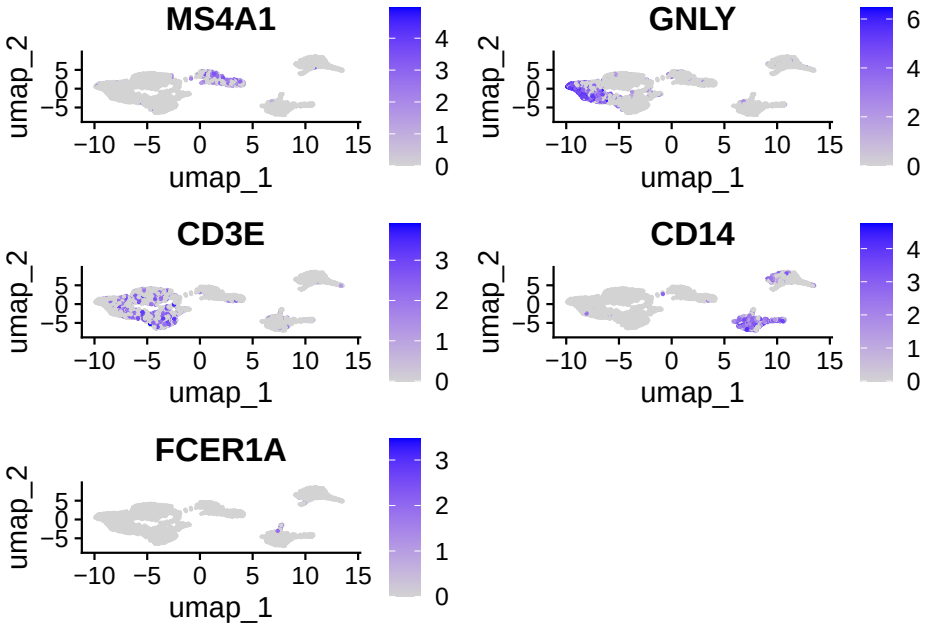
```
VlnPlot(seurat_object, features = c("MS4A1", "CD79A"))
```



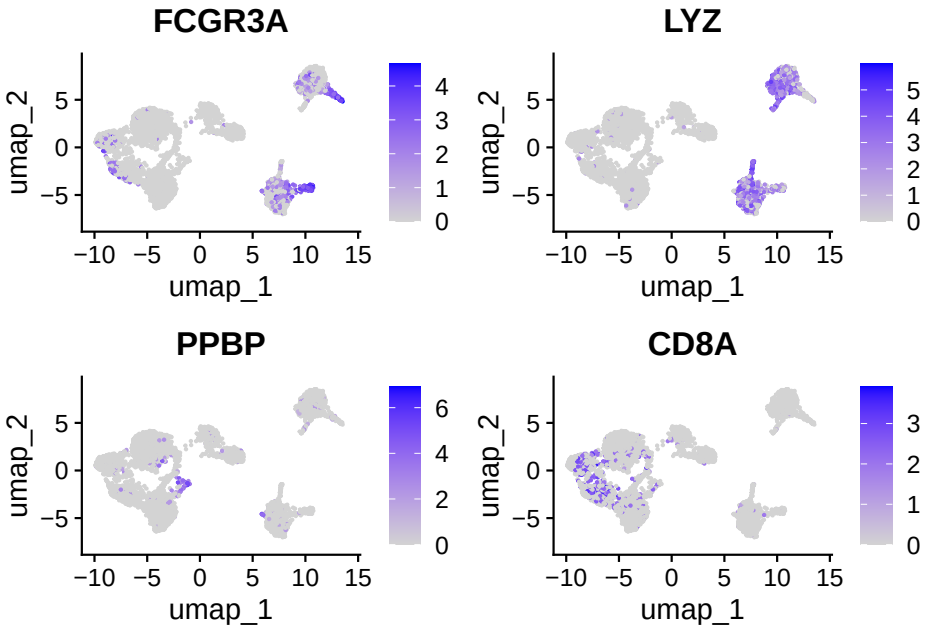
```
# you can plot raw counts as well
VlnPlot(seurat_object, features = c("NKG7", "PF4"), slot = 'counts', log = TRUE)
#> Warning: The `slot` argument of `VlnPlot()` is deprecated as of
#> Seurat 5.0.0.
#> i Please use the `layer` argument instead.
#> This warning is displayed once every 8 hours.
#> Call `lifecycle::last_lifecycle_warnings()` to see where
#> this warning was generated.
```




```
FeaturePlot(seurat_object, features = c("MS4A1", "GNLY", "CD3E", "CD14", "FCER1A"))
```



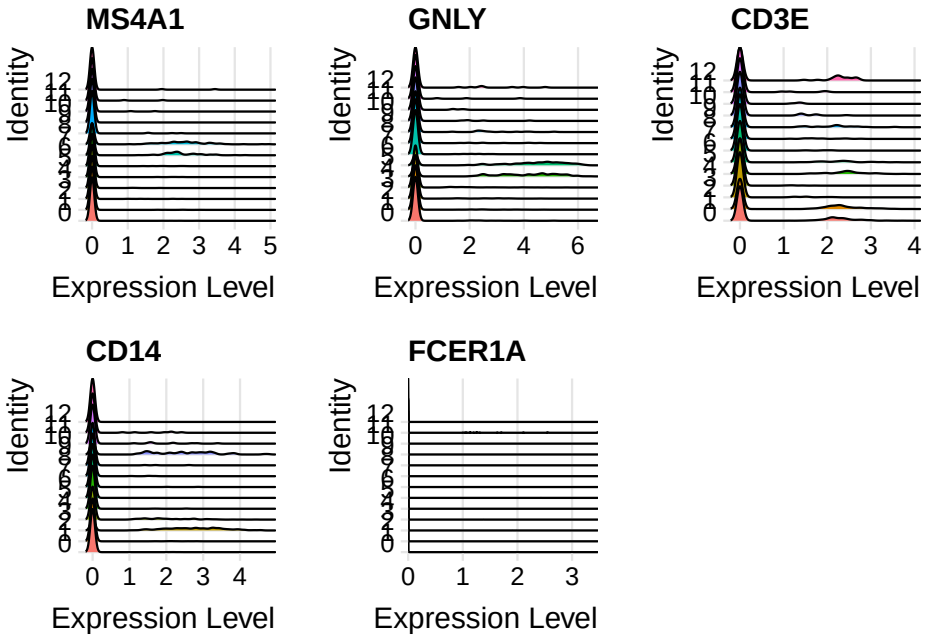

```
FeaturePlot(seurat_object, features = c("FCGR3A", "LYZ", "PPBP", "CD8A"))
```



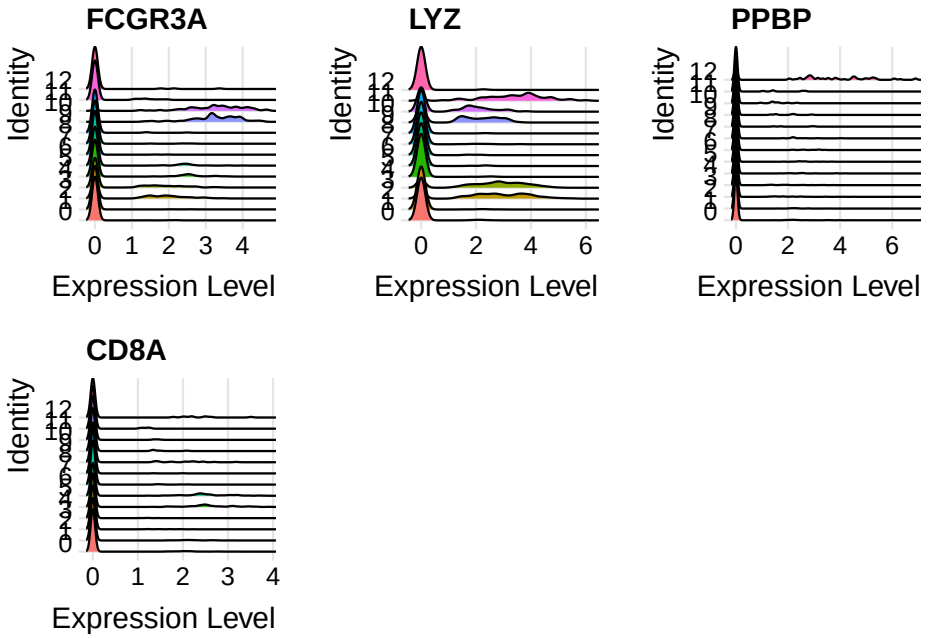
Other useful plots

These are ridgeplots, cell scatter plots and dotplots. Replace `FeaturePlot` with the other functions.

```
RidgePlot(seurat_object, features = c("MS4A1", "GNLY", "CD3E", "CD14", "FCER1A"))
#> Picking joint bandwidth of 0.0599
#> Picking joint bandwidth of 0.0898
#> Picking joint bandwidth of 0.0709
#> Picking joint bandwidth of 0.0628
#> Picking joint bandwidth of 3.07e-06
```




```
RidgePlot(seurat_object, features = c("FCGR3A", "LYZ", "PPBP", "CD8A"))
#> Picking joint bandwidth of 0.0788
#> Picking joint bandwidth of 0.152
#> Picking joint bandwidth of 0.0607
#> Picking joint bandwidth of 0.0492
```

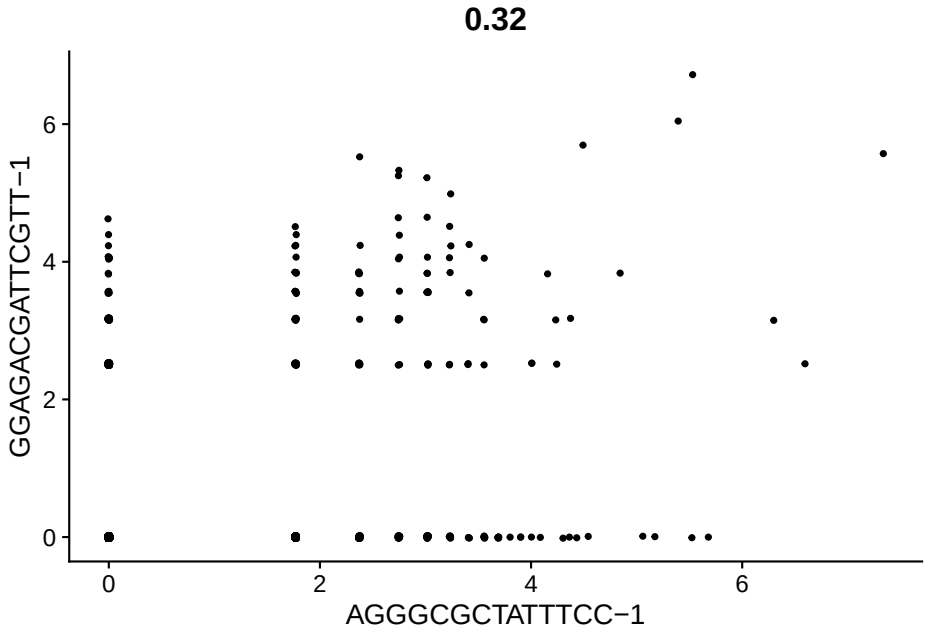


For CellScatter plots, will need the cell id of the cells you want to look at. You can get this from the cell metadata (`seurat_object@meta.data`).

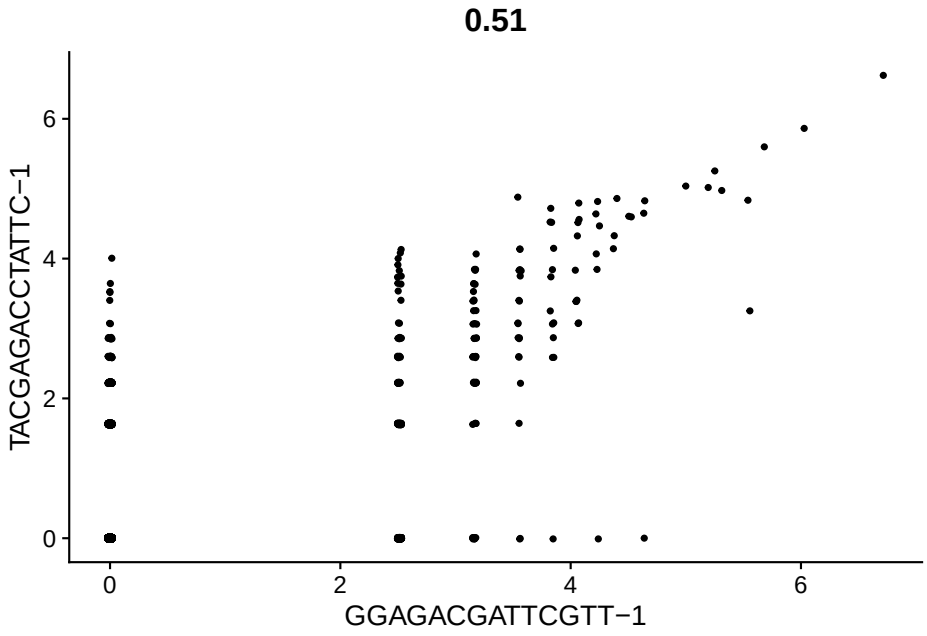
```
head( seurat_object@meta.data )
#>                    orig.ident nCount_RNA nFeature_RNA  ind
#> AGGGCGCTATTTCC-1 SeuratProject       2053          532 1256
#> GGAGACGATTCGTT-1 SeuratProject        881          392 1256
#> CACCGTTGTCGTAG-1 SeuratProject       3130         1005 1016
#> TATCGTACACGCAT-1 SeuratProject       1042          549 1488
#> TACGAGACCTATTC-1 SeuratProject       2425          777 1244
#> GTACTACTCATACG-1 SeuratProject       3951         1064 1256
#>                  stim               cell multiplets
#> AGGGCGCTATTTCC-1 stim   CD14+ Monocytes    singlet
#> GGAGACGATTCGTT-1 stim       CD4 T cells    singlet
#> CACCGTTGTCGTAG-1 ctrl FCGR3A+ Monocytes    singlet
#> TATCGTACACGCAT-1 stim           B cells    singlet
#> TACGAGACCTATTC-1 stim       CD4 T cells    singlet
#> GTACTACTCATACG-1 ctrl FCGR3A+ Monocytes    singlet
#>                  percent.mt RNA_snn_res.0.25
#> AGGGCGCTATTTCC-1  1.6336634                1
#> GGAGACGATTCGTT-1  4.8809524                0
#> CACCGTTGTCGTAG-1  1.0655473                4
#> TATCGTACACGCAT-1  3.0662710                3
#> TACGAGACCTATTC-1  1.0837849                0
#> GTACTACTCATACG-1  0.7137395                4
#>                  seurat_clusters pca_clusters
```

```
#> AGGGCGCTATTTCC-1               1             3
#> GGAGACGATTCGTT-1              14             0
#> CACCGTTGTCGTAG-1              15             7
#> TATCGTACACGCAT-1               7             5
#> TACGAGACCTATTC-1               6             0
#> GTACTACTCATACG-1              15             7
#>                    harmony_clusters RNA_snn_res.0.5
#> AGGGCGCTATTTCC-1                  1               3
#> GGAGACGATTCGTT-1                  0               0
#> CACCGTTGTCGTAG-1                  4               9
#> TATCGTACACGCAT-1                  3               6
#> TACGAGACCTATTC-1                  0               0
#> GTACTACTCATACG-1                  4               9
#>                    RNA_snn_res.0.1 RNA_snn_res.0.2
#> AGGGCGCTATTTCC-1                 3               3
#> GGAGACGATTCGTT-1                 1               0
#> CACCGTTGTCGTAG-1                 2               7
#> TATCGTACACGCAT-1                 4               5
#> TACGAGACCTATTC-1                 1               0
#> GTACTACTCATACG-1                 2               7
#>                    RNA_snn_res.0.3 RNA_snn_res.0.4
#> AGGGCGCTATTTCC-1                 2               3
#> GGAGACGATTCGTT-1                 0               0
#> CACCGTTGTCGTAG-1                 9               9
#> TATCGTACACGCAT-1                 6               6
#> TACGAGACCTATTC-1                 0               0
#> GTACTACTCATACG-1                 9               9
#>                    RNA_snn_res.0.6 RNA_snn_res.0.7
#> AGGGCGCTATTTCC-1                 3               3
#> GGAGACGATTCGTT-1                 0               0
#> CACCGTTGTCGTAG-1                 9               9
#> TATCGTACACGCAT-1                 6               6
#> TACGAGACCTATTC-1                 0               0
#> GTACTACTCATACG-1                 9               9
#>                    RNA_snn_res.0.8 RNA_snn_res.0.9
#> AGGGCGCTATTTCC-1                 3               3
#> GGAGACGATTCGTT-1                 0               0
#> CACCGTTGTCGTAG-1                 9               9
#> TATCGTACACGCAT-1                 6               6
#> TACGAGACCTATTC-1                 0               0
#> GTACTACTCATACG-1                 9               9
#>                    RNA_snn_res.1 RNA_snn_res.1.1
#> AGGGCGCTATTTCC-1               3              3
#> GGAGACGATTCGTT-1              10             10
#> CACCGTTGTCGTAG-1              11             11
```

```
#> TATCGTACACGCAT-1              6               6
#> TACGAGACCTATTC-1              0               0
#> GTACTACTCATACG-1             11              11
#>                   RNA_snn_res.1.2 RNA_snn_res.1.3
#> AGGGCGCTATTTCC-1                3               2
#> GGAGACGATTCGTT-1               10              12
#> CACCGTTGTCGTAG-1               11              13
#> TATCGTACACGCAT-1                6               7
#> TACGAGACCTATTC-1                0               0
#> GTACTACTCATACG-1               11              13
#>                   RNA_snn_res.1.4 RNA_snn_res.1.5
#> AGGGCGCTATTTCC-1                2               2
#> GGAGACGATTCGTT-1               12              13
#> CACCGTTGTCGTAG-1               13              14
#> TATCGTACACGCAT-1                7               8
#> TACGAGACCTATTC-1                0               0
#> GTACTACTCATACG-1               13              14
#>                   RNA_snn_res.1.6 RNA_snn_res.1.7
#> AGGGCGCTATTTCC-1                2               2
#> GGAGACGATTCGTT-1               13              13
#> CACCGTTGTCGTAG-1               14              14
#> TATCGTACACGCAT-1                7               8
#> TACGAGACCTATTC-1                0               0
#> GTACTACTCATACG-1               14              14
#>                   RNA_snn_res.1.8 RNA_snn_res.1.9
#> AGGGCGCTATTTCC-1                1               2
#> GGAGACGATTCGTT-1               14              14
#> CACCGTTGTCGTAG-1               15              15
#> TATCGTACACGCAT-1                9              10
#> TACGAGACCTATTC-1                7               7
#> GTACTACTCATACG-1               15              15
#>                   RNA_snn_res.2
#> AGGGCGCTATTTCC-1              1
#> GGAGACGATTCGTT-1             14
#> CACCGTTGTCGTAG-1             15
#> TATCGTACACGCAT-1              7
#> TACGAGACCTATTC-1              6
#> GTACTACTCATACG-1             15

CellScatter(seurat_object, cell1 = "AGGGCGCTATTTCC-1", cell2 = "GGAGACGATTCGTT-1")
```

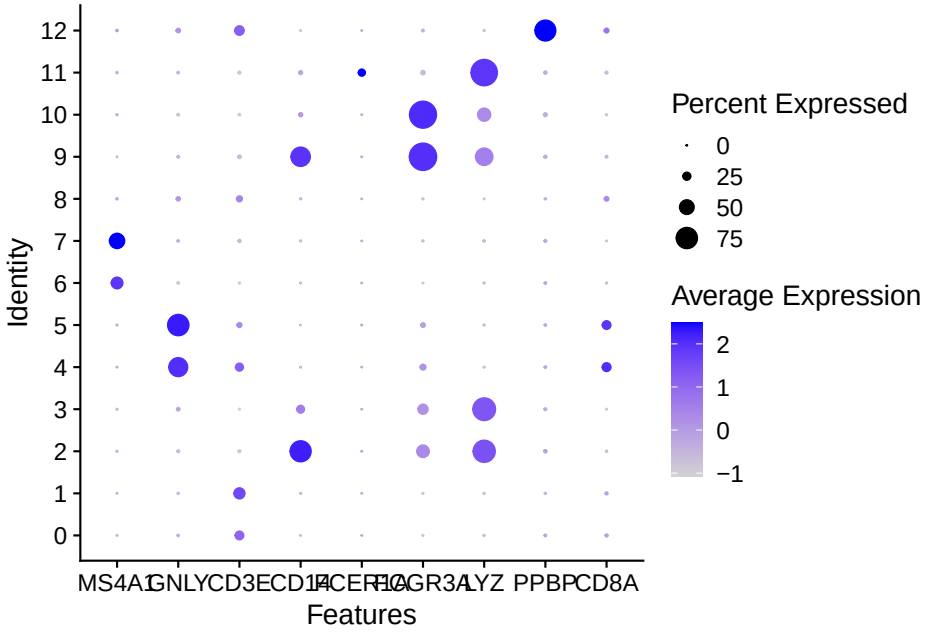



```
CellScatter(seurat_object, cell1 = "GGAGACGATTCGTT-1", cell2 = "TACGAGACCTATTC-1")
```



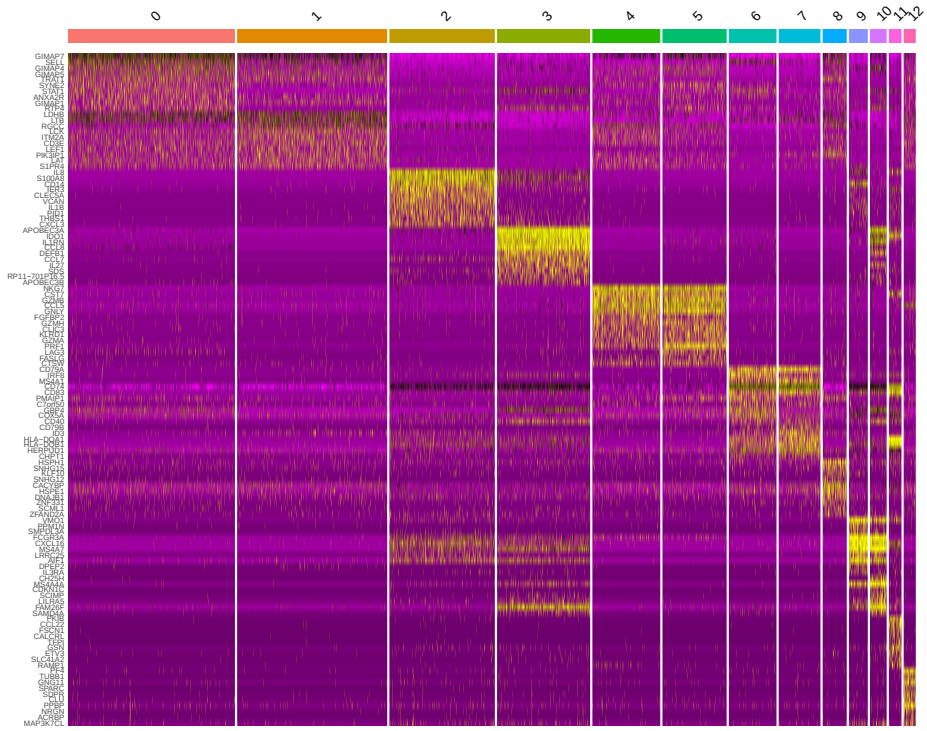
DotPlots

```
DotPlot(seurat_object, features = c("MS4A1", "GNLY", "CD3E", "CD14", "FCER1A", "FCGR3A
```



`DoHeatmap()` generates an expression heatmap for given cells and features. In this case, we are plotting the top 10 markers (or all markers if less than 10) for each cluster.

```
top10 <- seurat_object.markers %>% group_by(cluster) %>% top_n(n = 10, wt = avg_log2FC)
DoHeatmap(seurat_object, features = top10$gene) + NoLegend()
```

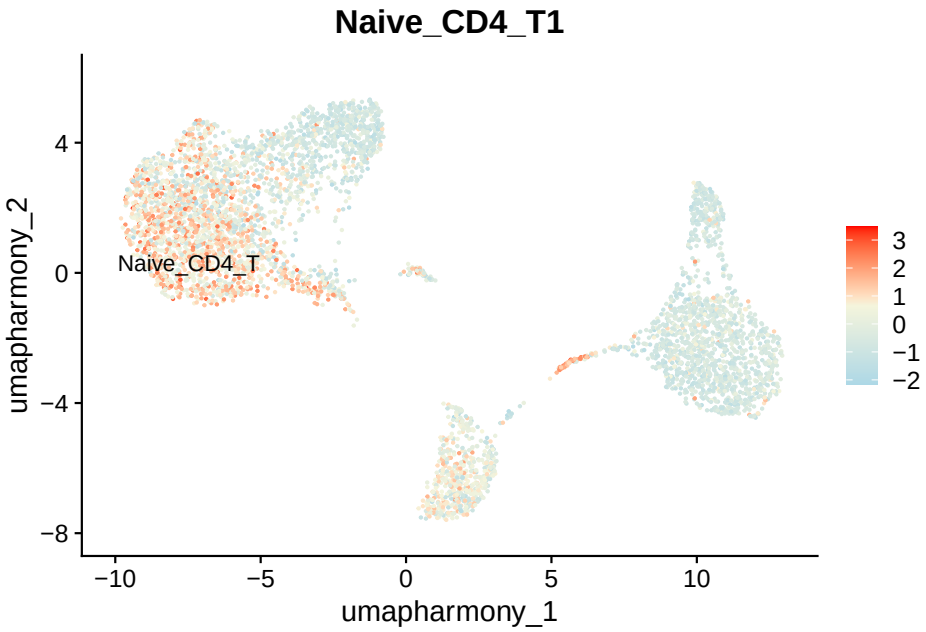


11.2 Use makers to label or find a cluster

If you know markers for your cell types, use AddModuleScore to label them.

```
genes_markers <- list(Naive_CD4_T = c("IL7R", "CCR7"))

seurat_object <- AddModuleScore(object = seurat_object, features = genes_markers, ctrl
    search = TRUE)


# notice the name of the cluster has a 1 at the end
names(seurat_object@meta.data)
#>  [1] "orig.ident"        "nCount_RNA"
#>  [3] "nFeature_RNA"      "ind"
#>  [5] "stim"              "cell"
#>  [7] "multiplets"        "percent.mt"
#>  [9] "RNA_snn_res.0.25" "seurat_clusters"
#> [11] "pca_clusters"      "harmony_clusters"
#> [13] "RNA_snn_res.0.5"  "RNA_snn_res.0.1"
#> [15] "RNA_snn_res.0.2"  "RNA_snn_res.0.3"
#> [17] "RNA_snn_res.0.4"  "RNA_snn_res.0.6"
#> [19] "RNA_snn_res.0.7"  "RNA_snn_res.0.8"
```

```
#> [21] "RNA_snn_res.0.9"  "RNA_snn_res.1"
#> [23] "RNA_snn_res.1.1"  "RNA_snn_res.1.2"
#> [25] "RNA_snn_res.1.3"  "RNA_snn_res.1.4"
#> [27] "RNA_snn_res.1.5"  "RNA_snn_res.1.6"
#> [29] "RNA_snn_res.1.7"  "RNA_snn_res.1.8"
#> [31] "RNA_snn_res.1.9"  "RNA_snn_res.2"
#> [33] "Naive_CD4_T1"

# label that cell type
seurat_object$cell_label = NA
seurat_object$cell_label[seurat_object$Naive_CD4_T1 > 1] = "Naive_CD4_T"
Idents(seurat_object) = seurat_object$cell_label

# plot
# Using a custom colour scale
FeaturePlot(seurat_object, features = "Naive_CD4_T1", label = TRUE, repel = TRUE, reduction = "um
#> Scale for colour is already present.
#> Adding another scale for colour, which will replace the
#> existing scale.
```

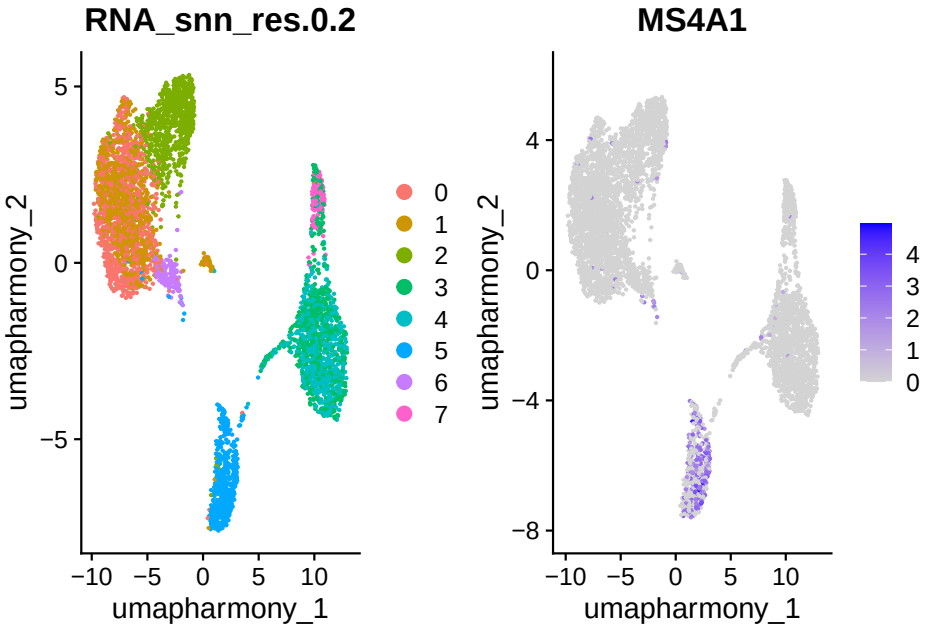


11.3 Assigning cell type identity to clusters

Fortunately in the case of this dataset, we can use canonical markers to easily match the unbiased clustering to known cell types:

Markers	Cell Type
IL7R, CCR7	Naive CD4+ T
CD14, LYZ	CD14+ Mono
IL7R, S100A4	Memory CD4+
MS4A1	B
CD8A	CD8+ T
FCGR3A, MS4A7	FCGR3A+ Mono
GNLY, NKG7	NK
FCER1A, CST3	DC
PPBP	Platelet

```
DimPlot(seurat_object,group.by = "RNA_snn_res.0.2",reduction = "umap_harmony")|FeatureP
```



Challenge: Match cluster numbers with cell labels

Use the markers provided and the resolution 0.2 to identity the cell labels

code ideas?

```
Idents(seurat_object) <- seurat_object$RNA_snn_res.0.2
# this is not the proper order. Make sure the labels are in the same order that the nu
new.cluster.ids <- c("Naive CD4 T","B cells","CD14+ Monocytes","CD4 T cells","CD8 T ce
names(new.cluster.ids) <- levels(seurat_object)
seurat_object <- RenameIdents(seurat_object, new.cluster.ids)
```

```
DimPlot(seurat_object, reduction = 'umap_harmony', label = TRUE, pt.size = 0.5) + NoLegend()
```

11.3.0.1 save the plots

11.3.0.2 save the seurat object

```
saveRDS(seurat_object, file = "seurat_object3k_final.rds")
```

Part II

Futher Analysis

Chapter 12

SingleR

```
#install.packages("BiocManager")
#BiocManager::install(c("SingleCellExperiment","SingleR","celldex"),ask=F)
library(SingleCellExperiment)
library(SingleR)
library(celldex)
```

In this workshop we have focused on the Seurat package. However, there is another whole ecosystem of R packages for single cell analysis within Bioconductor. We won't go into any detail on these packages in this workshop, but there is good material describing the object type online : OSCA.

For now, we'll just convert our Seurat object into an object called SingleCellExperiment. Some popular packages from Bioconductor that work with this type are Slingshot, Scran, Scater.

```
sce <- as.SingleCellExperiment(seurat_object)
sce
#> class: SingleCellExperiment
#> dim: 35635 4877
#> metadata(0):
#> assays(3): counts logcounts scaledata
#> rownames(35635): MIR1302-10 FAM138A ... MT-ND6 MT-CYB
#> rowData names(0):
#> colnames(4877): AGGGCGCTATTTCC-1 GGAGACGATTCGTT-1 ...
#>   ATGTTGCTAAAAGC-1 GATGACACTAGCGT-1
#> colData names(35): orig.ident nCount_RNA ...
#>   cell_label ident
#> reducedDimNames(4): PCA UMAP HARMONY UMAP_HARMONY
#> mainExpName: RNA
#> altExpNames(0):
```

We will now use a package called SingleR to label each cell. SingleR uses a reference data set of cell types with expression data to infer the best label for each cell. A convenient collection of cell type reference is in the `celldex` package which currently contains the follow sets:

```
ls('package:celldex')
#>  [1] "BlueprintEncodeData"
#>  [2] "DatabaseImmuneCellExpressionData"
#>  [3] "defineTextQuery"
#>  [4] "fetchLatestVersion"
#>  [5] "fetchMetadata"
#>  [6] "fetchReference"
#>  [7] "HumanPrimaryCellAtlasData"
#>  [8] "ImmGenData"
#>  [9] "listReferences"
#> [10] "listVersions"
#> [11] "MonacoImmuneData"
#> [12] "MouseRNAseqData"
#> [13] "NovershternHematopoieticData"
#> [14] "saveReference"
#> [15] "searchReferences"
#> [16] "surveyReferences"
```

In this example, we'll use the `HumanPrimaryCellAtlasData` set, which contains high-level, and fine-grained label types. Lets download the reference dataset

```
# This too is a sce object,
# colData is equivalent to seurat's metadata
ref.set <- celldex::HumanPrimaryCellAtlasData()
```

The "main" labels.

```
unique(ref.set$label.main)
#>  [1] "DC"                   "Smooth_muscle_cells"
#>  [3] "Epithelial_cells"     "B_cell"
#>  [5] "Neutrophils"          "T_cells"
#>  [7] "Monocyte"             "Erythroblast"
#>  [9] "BM & Prog."           "Endothelial_cells"
#> [11] "Gametocytes"          "Neurons"
#> [13] "Keratinocytes"        "HSC_-G-CSF"
#> [15] "Macrophage"           "NK_cell"
#> [17] "Embryonic_stem_cells" "Tissue_stem_cells"
#> [19] "Chondrocytes"         "Osteoblasts"
#> [21] "BM"                   "Platelets"
#> [23] "Fibroblasts"          "iPS_cells"
#> [25] "Hepatocytes"          "MSC"
#> [27] "Neuroepithelial_cell" "Astrocyte"
```

```
#> [29] "HSC_CD34+"              "CMP"
#> [31] "GMP"                    "MEP"
#> [33] "Myelocyte"              "Pre-B_cell_CD34-"
#> [35] "Pro-B_cell_CD34+"       "Pro-Myelocyte"
```

An example of the types of "fine" labels.

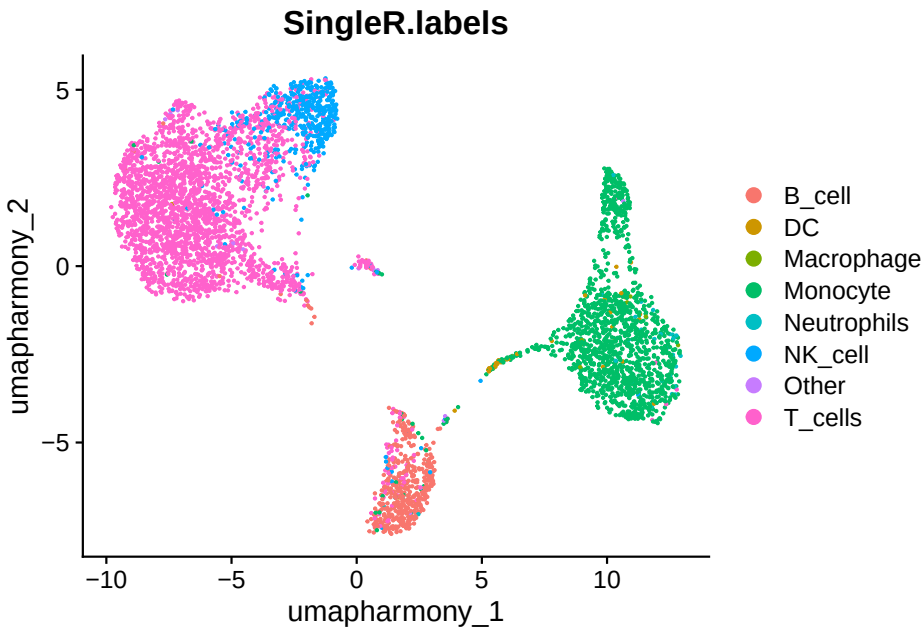
```
head(unique(ref.set$label.fine))
#> [1] "DC:monocyte-derived:immature"
#> [2] "DC:monocyte-derived:Galectin-1"
#> [3] "DC:monocyte-derived:LPS"
#> [4] "DC:monocyte-derived"
#> [5] "Smooth_muscle_cells:bronchial:vit_D"
#> [6] "Smooth_muscle_cells:bronchial"
```

Now we'll label our cells using the SingleCellExperiment object, with the above reference set.

```
pred.cnts <- SingleR::SingleR(test = sce, ref = ref.set, labels = ref.set$label.main)
```

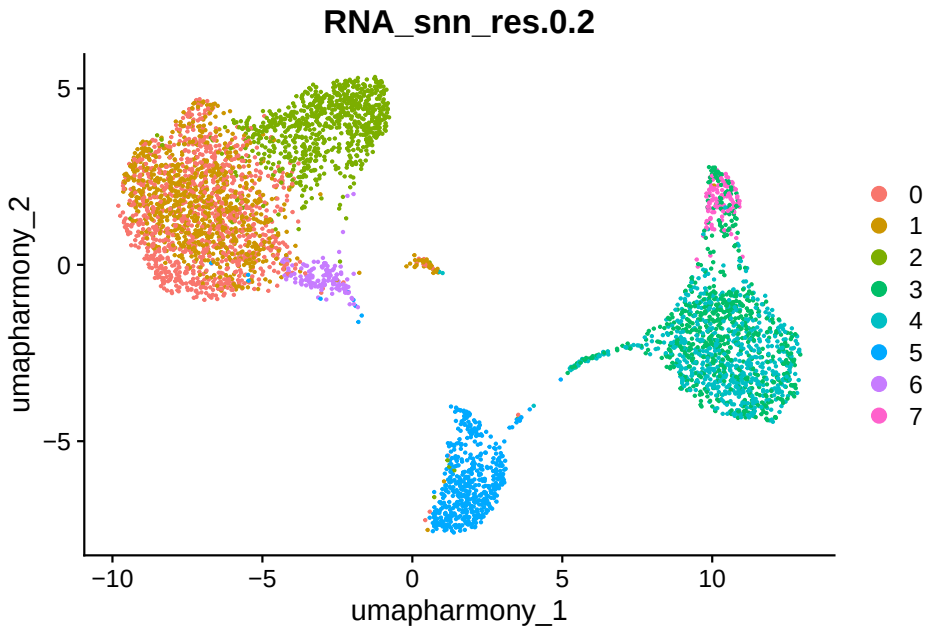
Keep any types that have more than 10 cells to the label, and put those labels back on our Seurat object and plot our on our umap.

```
lbls.keep <- table(pred.cnts$labels)>10
seurat_object$SingleR.labels <- ifelse(lbls.keep[pred.cnts$labels], pred.cnts$labels, 'Other')
DimPlot(seurat_object, reduction='umap_harmony', group.by='SingleR.labels')
```

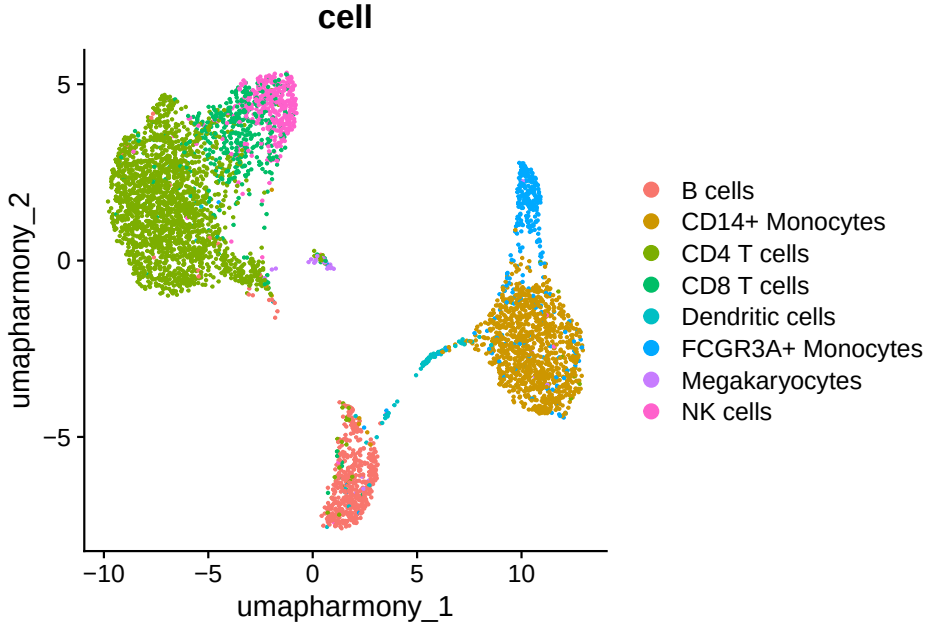


Compare cell labels by different annotation methods:

```
DimPlot(seurat_object,group.by = "RNA_snn_res.0.2",reduction = "umap_harmony")
```




```
DimPlot(seurat_object,group.by = "cell",reduction = "umap_harmony")
```



Chapter 13

Differential Expression

There are many different methods for calculating differential expression between groups in scRNAseq data. There are a number of review papers worth consulting on this topic.

There is the Seurat differential expression Vignette which walks through the variety implemented in Seurat.

There is also a good discussion of useing pseudobulk approaches which is worth checking out if youre planning differential expression analyses.

```
head(seurat_object@meta.data)
#>                     orig.ident nCount_RNA nFeature_RNA  ind
#> AGGGCGCTATTTCC-1 SeuratProject       2053          532 1256
#> GGAGACGATTCGTT-1 SeuratProject        881          392 1256
#> CACCGTTGTCGTAG-1 SeuratProject       3130         1005 1016
#> TATCGTACACGCAT-1 SeuratProject       1042          549 1488
#> TACGAGACCTATTC-1 SeuratProject       2425          777 1244
#> GTACTACTCATACG-1 SeuratProject       3951         1064 1256
#>                  stim               cell multiplets
#> AGGGCGCTATTTCC-1 stim   CD14+ Monocytes    singlet
#> GGAGACGATTCGTT-1 stim       CD4 T cells    singlet
#> CACCGTTGTCGTAG-1 ctrl FCGR3A+ Monocytes    singlet
#> TATCGTACACGCAT-1 stim           B cells    singlet
#> TACGAGACCTATTC-1 stim       CD4 T cells    singlet
#> GTACTACTCATACG-1 ctrl FCGR3A+ Monocytes    singlet
#>                  percent.mt RNA_snn_res.0.25
#> AGGGCGCTATTTCC-1  1.6336634                1
#> GGAGACGATTCGTT-1  4.8809524                0
#> CACCGTTGTCGTAG-1  1.0655473                4
#> TATCGTACACGCAT-1  3.0662710                3
#> TACGAGACCTATTC-1  1.0837849                0
```

```
#> GTACTACTCATACG-1  0.7137395               4
#>                  seurat_clusters pca_clusters
#> AGGGCGCTATTTCC-1               1            3
#> GGAGACGATTCGTT-1              14            0
#> CACCGTTGTCGTAG-1              15            7
#> TATCGTACACGCAT-1               7            5
#> TACGAGACCTATTC-1               6            0
#> GTACTACTCATACG-1              15            7
#>                  harmony_clusters RNA_snn_res.0.5
#> AGGGCGCTATTTCC-1                1               3
#> GGAGACGATTCGTT-1                0               0
#> CACCGTTGTCGTAG-1                4               9
#> TATCGTACACGCAT-1                3               6
#> TACGAGACCTATTC-1                0               0
#> GTACTACTCATACG-1                4               9
#>                  RNA_snn_res.0.1 RNA_snn_res.0.2
#> AGGGCGCTATTTCC-1               3               3
#> GGAGACGATTCGTT-1               1               0
#> CACCGTTGTCGTAG-1               2               7
#> TATCGTACACGCAT-1               4               5
#> TACGAGACCTATTC-1               1               0
#> GTACTACTCATACG-1               2               7
#>                  RNA_snn_res.0.3 RNA_snn_res.0.4
#> AGGGCGCTATTTCC-1               2               3
#> GGAGACGATTCGTT-1               0               0
#> CACCGTTGTCGTAG-1               9               9
#> TATCGTACACGCAT-1               6               6
#> TACGAGACCTATTC-1               0               0
#> GTACTACTCATACG-1               9               9
#>                  RNA_snn_res.0.6 RNA_snn_res.0.7
#> AGGGCGCTATTTCC-1               3               3
#> GGAGACGATTCGTT-1               0               0
#> CACCGTTGTCGTAG-1               9               9
#> TATCGTACACGCAT-1               6               6
#> TACGAGACCTATTC-1               0               0
#> GTACTACTCATACG-1               9               9
#>                  RNA_snn_res.0.8 RNA_snn_res.0.9
#> AGGGCGCTATTTCC-1               3               3
#> GGAGACGATTCGTT-1               0               0
#> CACCGTTGTCGTAG-1               9               9
#> TATCGTACACGCAT-1               6               6
#> TACGAGACCTATTC-1               0               0
#> GTACTACTCATACG-1               9               9
#>                  RNA_snn_res.1 RNA_snn_res.1.1
#> AGGGCGCTATTTCC-1             3               3
```

```
#> GGAGACGATTCGTT-1            10              10
#> CACCGTTGTCGTAG-1            11              11
#> TATCGTACACGCAT-1             6               6
#> TACGAGACCTATTC-1             0               0
#> GTACTACTCATACG-1            11              11
#>                  RNA_snn_res.1.2 RNA_snn_res.1.3
#> AGGGCGCTATTTCC-1               3               2
#> GGAGACGATTCGTT-1              10              12
#> CACCGTTGTCGTAG-1              11              13
#> TATCGTACACGCAT-1               6               7
#> TACGAGACCTATTC-1               0               0
#> GTACTACTCATACG-1              11              13
#>                  RNA_snn_res.1.4 RNA_snn_res.1.5
#> AGGGCGCTATTTCC-1               2               2
#> GGAGACGATTCGTT-1              12              13
#> CACCGTTGTCGTAG-1              13              14
#> TATCGTACACGCAT-1               7               8
#> TACGAGACCTATTC-1               0               0
#> GTACTACTCATACG-1              13              14
#>                  RNA_snn_res.1.6 RNA_snn_res.1.7
#> AGGGCGCTATTTCC-1               2               2
#> GGAGACGATTCGTT-1              13              13
#> CACCGTTGTCGTAG-1              14              14
#> TATCGTACACGCAT-1               7               8
#> TACGAGACCTATTC-1               0               0
#> GTACTACTCATACG-1              14              14
#>                  RNA_snn_res.1.8 RNA_snn_res.1.9
#> AGGGCGCTATTTCC-1               1               2
#> GGAGACGATTCGTT-1              14              14
#> CACCGTTGTCGTAG-1              15              15
#> TATCGTACACGCAT-1               9              10
#> TACGAGACCTATTC-1               7               7
#> GTACTACTCATACG-1              15              15
#>                  RNA_snn_res.2 Naive_CD4_T1  cell_label
#> AGGGCGCTATTTCC-1             1   -0.3540023        <NA>
#> GGAGACGATTCGTT-1            14    2.1376158 Naive_CD4_T
#> CACCGTTGTCGTAG-1            15   -1.1487836        <NA>
#> TATCGTACACGCAT-1             7    0.5557941        <NA>
#> TACGAGACCTATTC-1             6    1.4250065 Naive_CD4_T
#> GTACTACTCATACG-1            15   -0.2082793        <NA>
#>                  SingleR.labels
#> AGGGCGCTATTTCC-1       Monocyte
#> GGAGACGATTCGTT-1        T_cells
#> CACCGTTGTCGTAG-1       Monocyte
#> TATCGTACACGCAT-1    Neutrophils
```

```
#> TACGAGACCTATTC-1        T_cells
#> GTACTACTCATACG-1       Monocyte
```

How cells from each condition do we have?

```
table(seurat_object$stim)
#>
#> ctrl stim
#> 2378 2499
```

How many cells per individuals per group?

```
table(seurat_object$ind, seurat_object$stim)
#>
#>        ctrl stim
#>   101   175  226
#>   107   115  106
#>   1015  503  489
#>   1016  335  348
#>   1039   87  100
#>   1244  373  311
#>   1256  384  385
#>   1488  406  534
```

And for each sample, how many of each cell type has been classified?

```
table(paste(seurat_object$ind,seurat_object$stim), seurat_object$cell)
#>
#>             B cells CD14+ Monocytes CD4 T cells CD8 T cells
#>   101 ctrl       23              47          60          15
#>   101 stim       30              52          83          17
#>   1015 ctrl      79             149         142          44
#>   1015 stim      68             149         148          21
#>   1016 ctrl      21              71          82         107
#>   1016 stim      29              65          65         109
#>   1039 ctrl       7              32          36           6
#>   1039 stim       6              26          48           5
#>   107 ctrl        9              50          32           6
#>   107 stim        9              35          43           1
#>   1244 ctrl      23              85         202           8
#>   1244 stim      18              58         191           4
#>   1256 ctrl      32              80         176          26
#>   1256 stim      42              70         194          25
#>   1488 ctrl      36              59         246          13
#>   1488 stim      59              59         319          15
#>
#>             Dendritic cells FCGR3A+ Monocytes
```

```
#>   101 ctrl                4                  11
#>   101 stim                6                  23
#>   1015 ctrl               3                  49
#>   1015 stim              17                  43
#>   1016 ctrl               4                  21
#>   1016 stim               2                  32
#>   1039 ctrl               1                   3
#>   1039 stim               1                   6
#>   107 ctrl                3                  12
#>   107 stim                2                   5
#>   1244 ctrl               8                  19
#>   1244 stim               6                   4
#>   1256 ctrl               6                  20
#>   1256 stim               3                  11
#>   1488 ctrl               8                  25
#>   1488 stim              12                  28
#>
#>              Megakaryocytes NK cells
#>   101 ctrl                4       11
#>   101 stim                1       14
#>   1015 ctrl               5       32
#>   1015 stim               5       38
#>   1016 ctrl               3       26
#>   1016 stim               1       45
#>   1039 ctrl               1        1
#>   1039 stim               3        5
#>   107 ctrl                0        3
#>   107 stim                0       11
#>   1244 ctrl               3       25
#>   1244 stim               4       26
#>   1256 ctrl               1       43
#>   1256 stim               7       33
#>   1488 ctrl               4       15
#>   1488 stim               6       36
```

13.1 Prefiltering

Why do we need to do this?

If expression is below a certain level, it will be almost impossible to see any differential expression.

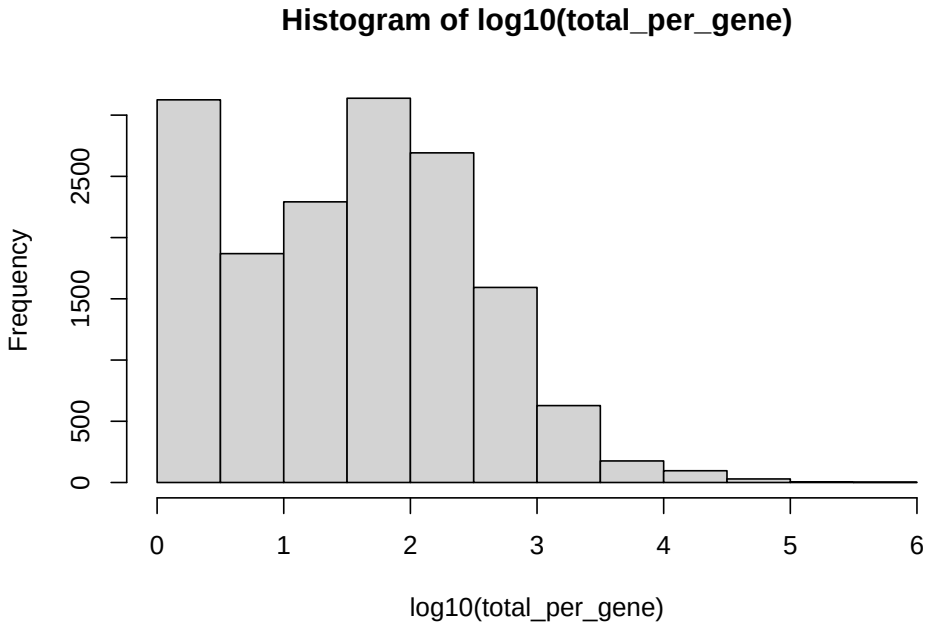
When doing differential expression, you generally ignore genes with low expression. In single cell datasets, there are many genes like this. Filtering here to make our dataset smaller so it runs quicker, and there is less aggressive correction for multiple hypotheses.

How many genes before filtering?

```
seurat_object
#> An object of class Seurat
#> 35635 features across 4877 samples within 1 assay
#> Active assay: RNA (35635 features, 2000 variable features)
#>  3 layers present: counts, data, scale.data
#>  4 dimensional reductions calculated: pca, umap, harmony, umap_harmony
```

How many copies of each gene are there?

```
total_per_gene <- rowSums(GetAssayData(seurat_object, assay='RNA', slot='counts'))
#> Warning: The `slot` argument of `GetAssayData()` is deprecated as of
#> SeuratObject 5.0.0.
#> i Please use the `layer` argument instead.
#> This warning is displayed once every 8 hours.
#> Call `lifecycle::last_lifecycle_warnings()` to see where
#> this warning was generated.
hist(log10(total_per_gene))
```



Lets keep only those genes with at least 50 copies across the entire experiment.

```
seurat_object<- seurat_object[total_per_gene >= 50, ]
```

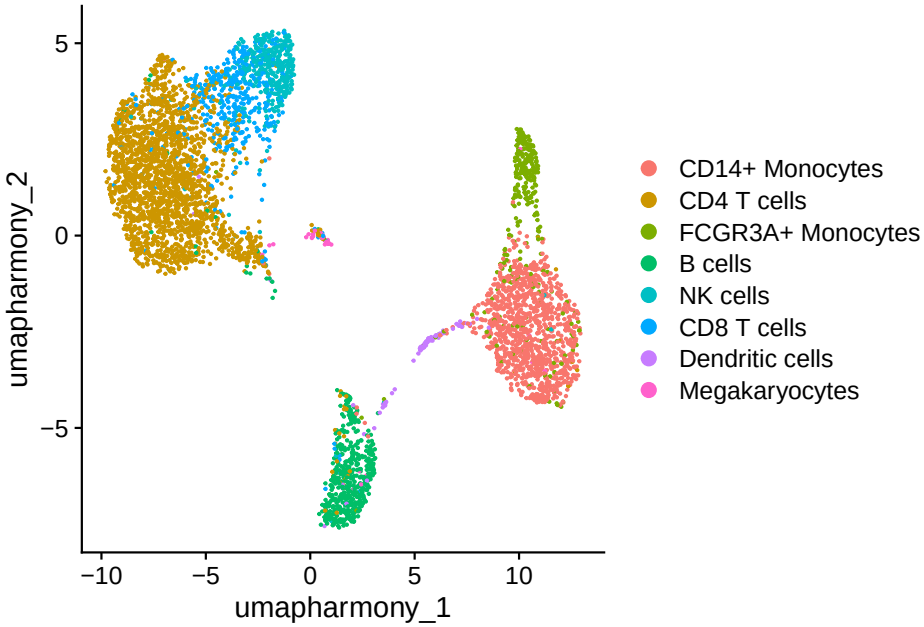
How many genes after filtering?

```
seurat_object
#> An object of class Seurat
#> 7200 features across 4877 samples within 1 assay
#> Active assay: RNA (7200 features, 1236 variable features)
#>  3 layers present: counts, data, scale.data
#>  4 dimensional reductions calculated: pca, umap, harmony, umap_harmony
```

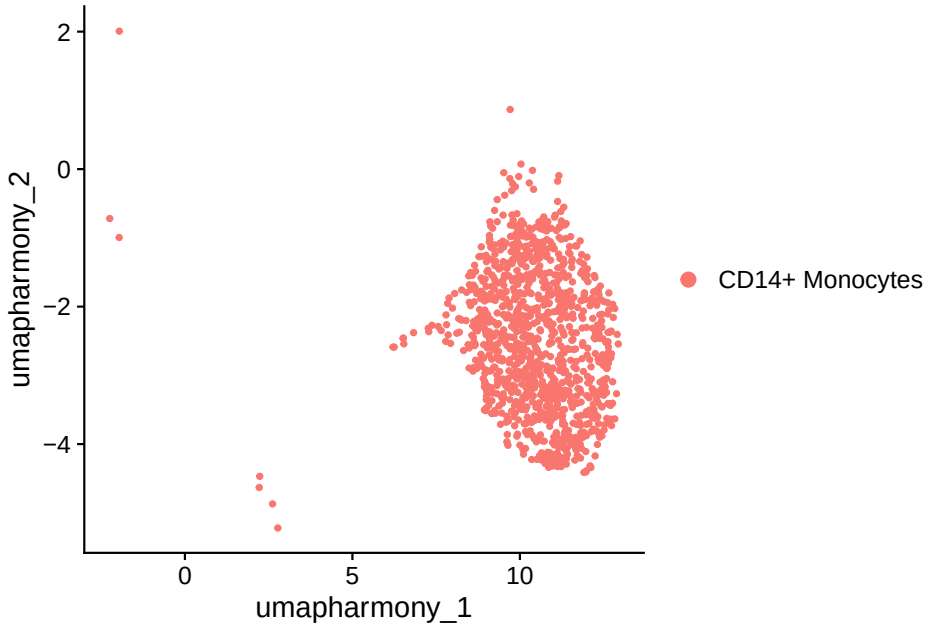
We might like to see the effect of IFN-beta stimulation on each cell type individually. For the purposes of this workshop, just going to test one cell type; CD14+ Monocytes

An easy way is to subset the object.

```
# Set idents to 'cell' column.
Idents(seurat_object) <- seurat_object$cell
DimPlot(seurat_object, reduction = "umap_harmony")
```




```
seurat_object_celltype <- seurat_object[, seurat_object$cell == "CD14+ Monocytes" ]
DimPlot(seurat_object_celltype, reduction = "umap_harmony")
```



13.2 Default Wilcox test

To run this test, we change the Idents to the factor(column) we want to test. In this case, that's 'stim'.

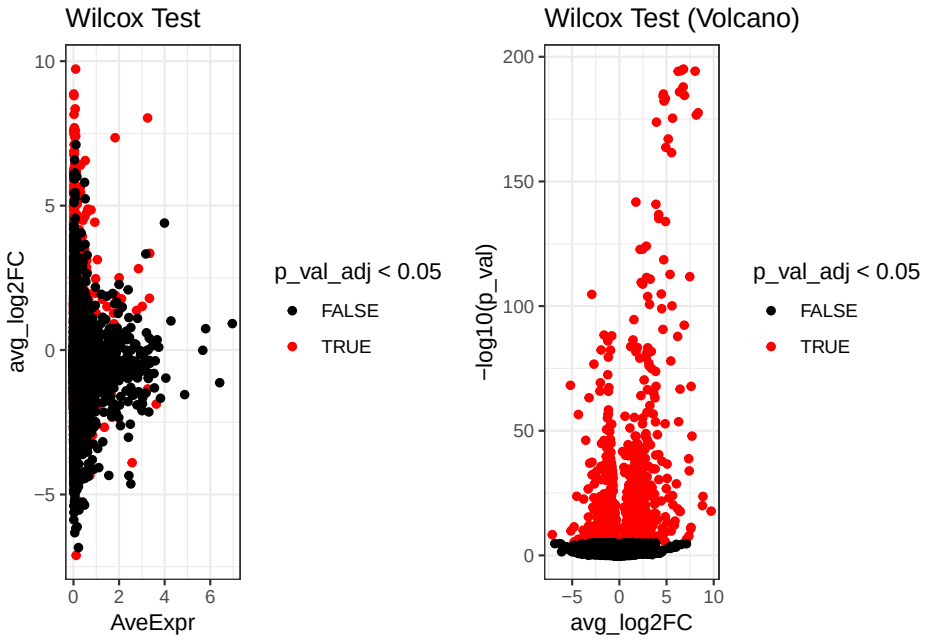
```
# Change Ident to Condition
Idents(seurat_object_celltype) <- seurat_object_celltype$stim

# default, wilcox test
de_result_wilcox <- FindMarkers(seurat_object_celltype,
            ident.1 = 'stim',
            ident.2 = 'ctrl',
            logfc.threshold = 0, # Give me ALL results
            min.pct = 0
            )

# Add average expression for plotting
de_result_wilcox$AveExpr<- rowMeans(seurat_object_celltype[rownames(de_result_wilcox),]
```

Look at the top differentially expressed genes.

```
head(de_result_wilcox)
#>                  p_val avg_log2FC pct.1 pct.2     p_val_adj
#> RSAD2   8.471603e-196   6.797508 0.979 0.044 6.099554e-192
#> TNFSF10 3.399691e-195   6.559853 0.982 0.056 2.447777e-191
#> CXCL10  5.809758e-195   8.034949 0.975 0.038 4.183025e-191
```

```
#> IFIT3   7.214495e-195   6.234703 0.982 0.051 5.194437e-191
#> IFIT1   1.224398e-188   6.764699 0.953 0.026 8.815662e-185
#> ISG15   1.228738e-186   6.377243 1.000 0.323 8.846914e-183
#>            AveExpr
#> RSAD2   0.04196286
#> TNFSF10 0.51655832
#> CXCL10  3.25184139
#> IFIT3   0.02032398
#> IFIT1   0.01254584
#> ISG15   0.14165156
```

```
p1 <- ggplot(de_result_wilcox, aes(x=AveExpr, y=avg_log2FC, col=p_val_adj < 0.05)) +
  geom_point() +
  scale_colour_manual(values=c('TRUE'="red",'FALSE'="black")) +
  theme_bw() +
  ggtitle("Wilcox Test")


p2 <- ggplot(de_result_wilcox, aes(x=avg_log2FC, y=-log10(p_val), col=p_val_adj < 0.05)) +
  geom_point() +
  scale_colour_manual(values=c('TRUE'="red",'FALSE'="black")) +
  theme_bw() +
  ggtitle("Wilcox Test (Volcano)")

p1 + p2
```



13.3 Seurat Negative binomial

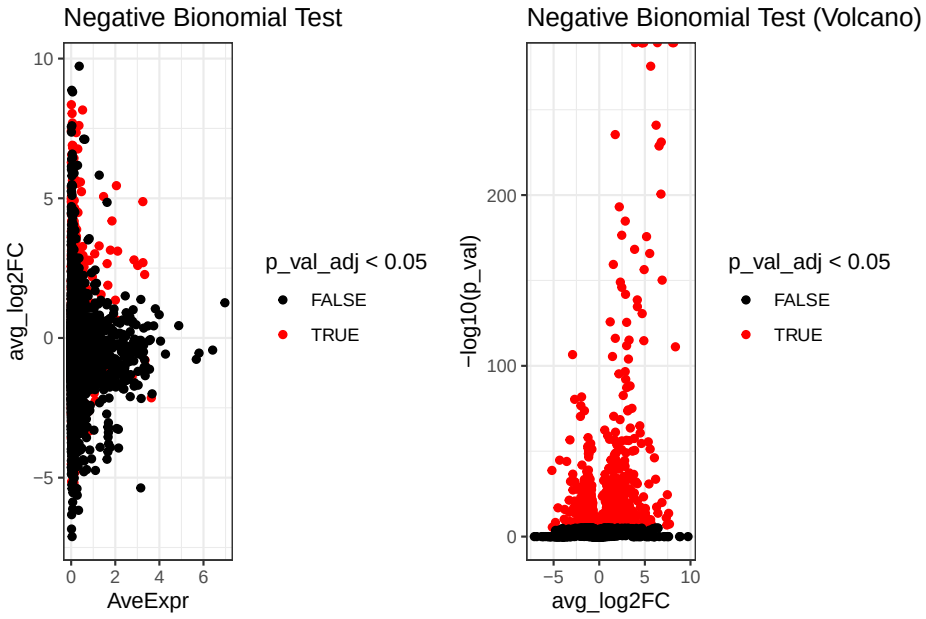
Negative binonial test is run almost the same way - just need to specify it under 'test.use'

```
# Change Ident to Condition
Idents(seurat_object_celltype) <- seurat_object_celltype$stim

# default, wilcox test
de_result_negbinom <- FindMarkers(seurat_object_celltype,
            test.use="negbinom", # Choose a different test.
            ident.1 = 'stim',
            ident.2 = 'ctrl',
            logfc.threshold = 0, # Give me ALL results
            min.pct = 0
)

# Add average expression for plotting
de_result_negbinom$AveExpr<- rowMeans(seurat_object_celltype[rownames(de_result_negbino
```

Look at the top differentially expressed genes.

```
head(de_result_negbinom)
#>           p_val avg_log2FC pct.1 pct.2 p_val_adj    AveExpr
#> CXCL10        0   8.034949 0.975 0.038         0 0.04196286
#> CCL8          0   8.160755 0.912 0.023         0 0.51655832
#> LY6E          0   4.880093 0.979 0.134         0 3.25184139
#> APOBEC3A      0   4.745713 0.996 0.262         0 0.02032398
#> IFITM3        0   4.675898 0.998 0.271         0 0.01254584
#> IFI6          0   3.935122 0.982 0.257         0 0.14165156
```

```
p1 <- ggplot(de_result_negbinom, aes(x=AveExpr, y=avg_log2FC, col=p_val_adj < 0.05)) +
  geom_point() +
  scale_colour_manual(values=c('TRUE'="red",'FALSE'="black")) +
  theme_bw() +
  ggtitle("Negative Bionomial Test")


p2 <- ggplot(de_result_negbinom, aes(x=avg_log2FC, y=-log10(p_val), col=p_val_adj < 0.0
  geom_point() +
  scale_colour_manual(values=c('TRUE'="red",'FALSE'="black")) +
  theme_bw() +
  ggtitle("Negative Bionomial Test (Volcano)")

p1 + p2
```



13.4 Pseudobulk

Pseudobulk analysis is an option where you have biological replicates. It is essentially pooling the individual cell counts and treating your expreiment like a bulk RNAseq.

First, you need to build a pseudobulk matrix - the `AggregateExpression()` function can do this, once you set the 'Idents' of your seurat object to your grouping factor (here, thats a combination of individual+treatment called 'sample', instead of the 'stim' treatment column).

```
# Tools for bulk differential expression
library(limma)
#>
#> Attaching package: 'limma'
#> The following object is masked from 'package:BiocGenerics':
#>
#>     plotMA
library(edgeR)
#>
#> Attaching package: 'edgeR'
#> The following object is masked from 'package:SingleCellExperiment':
#>
#>     cpm
```

```
# Change idents to ind for grouping.
seurat_object_celltype$sample <- factor(paste(seurat_object_celltype$stim, seurat_objec
Idents(seurat_object_celltype) <- seurat_object_celltype$sample

# THen pool together counts in those groups
# AggregateExperssion returns a list of matricies - one for each assay requested (even
pseudobulk_matrix_list <- AggregateExpression( seurat_object_celltype,  slot = 'counts
#> Names of identity class contain underscores ('_'), replacing with dashes ('-')
#> This message is displayed once every 8 hours.
pseudobulk_matrix      <- pseudobulk_matrix_list[['RNA']]
colnames(pseudobulk_matrix) <- as.character(colnames(pseudobulk_matrix)) # Changes col
pseudobulk_matrix[1:5,1:4]
#> 5 x 4 sparse Matrix of class "dgCMatrix"
#>          ctrl-101 ctrl-1015 ctrl-1016 ctrl-1039
#> NOC2L           2         7         .         .
#> HES4            .         3         2         1
#> ISG15          31       185       234        41
#> TNFRSF18        .         3         4         2
#> TNFRSF4         .         2         .         .
```

Now it looks like a bulk RNAseq experiment, so treat it like one.

We can use the popular `limma` package for differential expression. Here is one tutorial, and the hefty reference manual is hosted by bioconductor.

In brief, this code below constructs a linear model for this experiment that accounts for the variation in individuals and treatment. It then tests for differential expression between 'stim' and 'ctrl' groups.

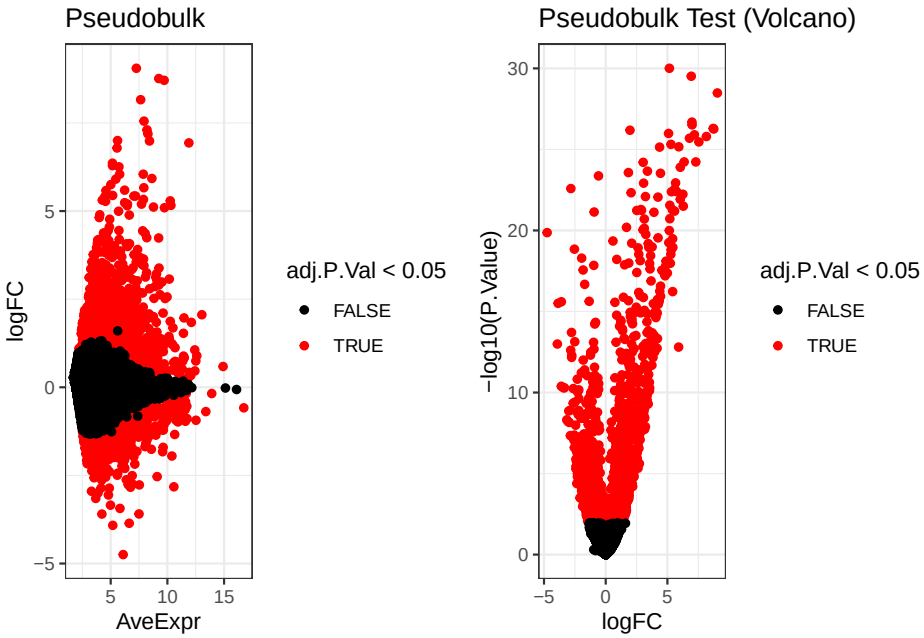
```
dge <- DGEList(pseudobulk_matrix)
dge <- calcNormFactors(dge)

# Remove _ or - and everything after it - yeilds stim group
stim <- gsub("[-_].*","",colnames(pseudobulk_matrix))

# Removing everything before the _ or - for the individual, then converting those nume
ind  <- as.character(gsub(".*[-_]","",colnames(pseudobulk_matrix)))

design <- model.matrix( ~0 + stim + ind)
vm  <- voom(dge, design = design, plot = FALSE)
fit <- lmFit(vm, design = design)

contrasts <- makeContrasts(stimstim - stimctrl, levels=coef(fit))
fit <- contrasts.fit(fit, contrasts)
fit <- eBayes(fit)

de_result_pseudobulk <- topTable(fit, n = Inf, adjust.method = "BH")
```

```
de_result_pseudobulk <- arrange(de_result_pseudobulk , adj.P.Val)
```

Look at the significantly differentially expressed genes:

```
head(de_result_pseudobulk)
#>           logFC   AveExpr        t      P.Value
#> ISG20  5.161336 10.313033 35.72209 9.838370e-31
#> ISG15  6.932960 11.897783 34.63778 3.049704e-30
#> CXCL11 9.050139  7.263356 32.46019 3.282727e-29
#> IFIT3  6.987552  8.423467 28.97126 2.052443e-27
#> HERC5  6.999781  5.604886 28.65574 3.050588e-27
#> CXCL10 8.708460  9.715253 28.25406 5.081902e-27
#>           adj.P.Val        B
#> ISG20  7.083626e-27 60.13277
#> ISG15  1.097894e-26 59.02083
#> CXCL11 7.878545e-26 55.45779
#> IFIT3  3.694397e-24 52.01990
#> HERC5  4.392847e-24 51.18902
#> CXCL10 5.629458e-24 51.32019
```

```
p1 <- ggplot(de_result_pseudobulk, aes(x=AveExpr, y=logFC, col=adj.P.Val < 0.05)) +
  geom_point() +
  scale_colour_manual(values=c('TRUE'="red",'FALSE'="black")) +
  theme_bw() +
  ggtitle("Pseudobulk")


p2 <- ggplot(de_result_pseudobulk, aes(x=logFC, y=-log10(P.Value), col=adj.P.Val < 0.05)) +
  geom_point() +
  scale_colour_manual(values=c('TRUE'="red",'FALSE'="black")) +
  theme_bw() +
  ggtitle("Pseudobulk Test (Volcano)")

p1 + p2
```



Discussion

These methods give different results. How would you decide which to use? How could you check an individual gene?

Chapter 14

Cell cycle Assignment

In some datasets, the phase of cell cycle that a cell is in (G1/G2M/S) can account for alot of the observed transcriptomic variation. There may be clustering by phase, or separation in the UMAP by phase.

Seurat provides a simple method for assigning cell cycle state to each cell. Other methods are available.

More information about assigning cell cycle states to cells is in the cell cycle vignette

```
# A list of cell cycle markers, from Tirosh et al, 2015, is loaded with Seurat.  We can
# segregate this list into markers of G2/M phase and markers of S phase
s.genes   <- cc.genes$s.genes
g2m.genes <- cc.genes$g2m.genes

# Use those lists with the cell cycle scoring function in Seurat.
seurat_object <- CellCycleScoring(seurat_object, s.features = s.genes, g2m.features = g2m.genes)
#> Warning: The following features are not present in the
#> object: TYMS, MCM2, MCM4, UNG, GINS2, CDCA7, DTL, UHRF1,
#> MLF1IP, HELLS, RAD51AP1, GMNN, WDR76, CCNE2, ATAD2, RAD51,
#> RRM2, CDC45, CDC6, EXO1, DSCC1, BLM, CASP8AP2, CLSPN,
#> POLA1, CHAF1B, BRIP1, E2F8, not searching for symbol
#> synonyms
#> Warning: The following features are not present in the
#> object: CDK1, NUSAP1, UBE2C, BIRC5, TPX2, TOP2A, NDC80,
#> NUF2, MKI67, FAM64A, CCNB2, CKAP2L, AURKB, BUB1, KIF11,
#> GTSE1, HJURP, CDCA3, CDC20, TTK, CDC25C, KIF2C, NCAPD2,
#> DLGAP5, CDCA2, CDCA8, ECT2, KIF23, HMMR, AURKA, PSRC1,
#> ANLN, CENPE, NEK2, GAS2L3, CENPA, not searching for symbol
#> synonyms
```

Which adds S.Score, G2M.Score and Phase calls to the metadata.

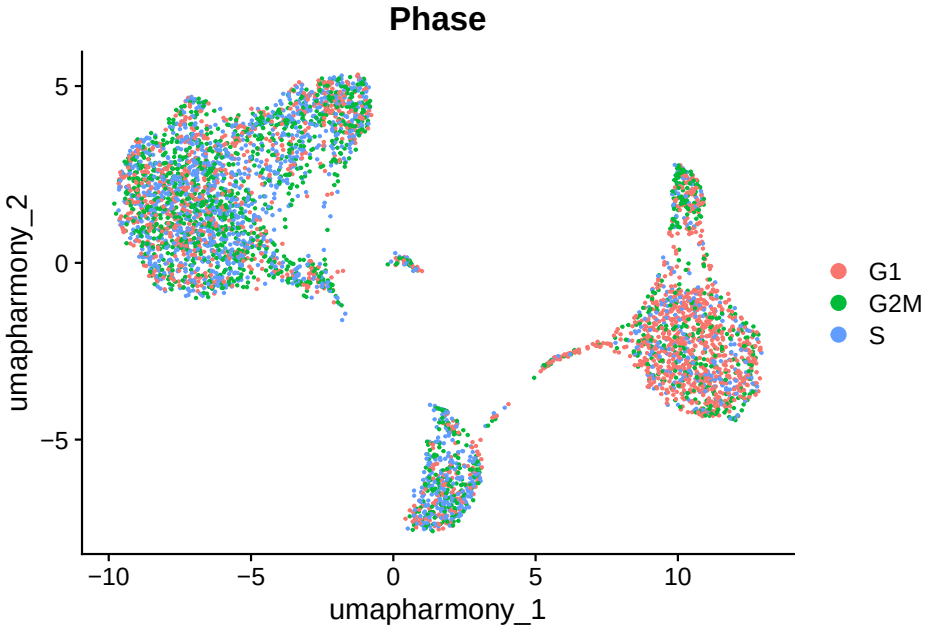
```
head(seurat_object@meta.data)
#>                     orig.ident nCount_RNA nFeature_RNA  ind
#> AGGGCGCTATTTCC-1 SeuratProject       2053          532 1256
#> GGAGACGATTCGTT-1 SeuratProject        881          392 1256
#> CACCGTTGTCGTAG-1 SeuratProject       3130         1005 1016
#> TATCGTACACGCAT-1 SeuratProject       1042          549 1488
#> TACGAGACCTATTC-1 SeuratProject       2425          777 1244
#> GTACTACTCATACG-1 SeuratProject       3951         1064 1256
#>                  stim              cell multiplets
#> AGGGCGCTATTTCC-1 stim   CD14+ Monocytes    singlet
#> GGAGACGATTCGTT-1 stim       CD4 T cells    singlet
#> CACCGTTGTCGTAG-1 ctrl FCGR3A+ Monocytes    singlet
#> TATCGTACACGCAT-1 stim           B cells    singlet
#> TACGAGACCTATTC-1 stim       CD4 T cells    singlet
#> GTACTACTCATACG-1 ctrl FCGR3A+ Monocytes    singlet
#>                  percent.mt RNA_snn_res.0.25
#> AGGGCGCTATTTCC-1  1.6336634                1
#> GGAGACGATTCGTT-1  4.8809524                0
#> CACCGTTGTCGTAG-1  1.0655473                4
#> TATCGTACACGCAT-1  3.0662710                3
#> TACGAGACCTATTC-1  1.0837849                0
#> GTACTACTCATACG-1  0.7137395                4
#>                  seurat_clusters pca_clusters
#> AGGGCGCTATTTCC-1               1            3
#> GGAGACGATTCGTT-1              14            0
#> CACCGTTGTCGTAG-1              15            7
#> TATCGTACACGCAT-1               7            5
#> TACGAGACCTATTC-1               6            0
#> GTACTACTCATACG-1              15            7
#>                  harmony_clusters RNA_snn_res.0.5
#> AGGGCGCTATTTCC-1                1               3
#> GGAGACGATTCGTT-1                0               0
#> CACCGTTGTCGTAG-1                4               9
#> TATCGTACACGCAT-1                3               6
#> TACGAGACCTATTC-1                0               0
#> GTACTACTCATACG-1                4               9
#>                  RNA_snn_res.0.1 RNA_snn_res.0.2
#> AGGGCGCTATTTCC-1               3               3
#> GGAGACGATTCGTT-1               1               0
#> CACCGTTGTCGTAG-1               2               7
#> TATCGTACACGCAT-1               4               5
#> TACGAGACCTATTC-1               1               0
#> GTACTACTCATACG-1               2               7
#>                  RNA_snn_res.0.3 RNA_snn_res.0.4
```

```
#> AGGGCGCTATTTCC-1               2               3
#> GGAGACGATTCGTT-1               0               0
#> CACCGTTGTCGTAG-1               9               9
#> TATCGTACACGCAT-1               6               6
#> TACGAGACCTATTC-1               0               0
#> GTACTACTCATACG-1               9               9
#>                   RNA_snn_res.0.6 RNA_snn_res.0.7
#> AGGGCGCTATTTCC-1               3               3
#> GGAGACGATTCGTT-1               0               0
#> CACCGTTGTCGTAG-1               9               9
#> TATCGTACACGCAT-1               6               6
#> TACGAGACCTATTC-1               0               0
#> GTACTACTCATACG-1               9               9
#>                   RNA_snn_res.0.8 RNA_snn_res.0.9
#> AGGGCGCTATTTCC-1               3               3
#> GGAGACGATTCGTT-1               0               0
#> CACCGTTGTCGTAG-1               9               9
#> TATCGTACACGCAT-1               6               6
#> TACGAGACCTATTC-1               0               0
#> GTACTACTCATACG-1               9               9
#>                   RNA_snn_res.1 RNA_snn_res.1.1
#> AGGGCGCTATTTCC-1              3               3
#> GGAGACGATTCGTT-1             10              10
#> CACCGTTGTCGTAG-1             11              11
#> TATCGTACACGCAT-1              6               6
#> TACGAGACCTATTC-1              0               0
#> GTACTACTCATACG-1             11              11
#>                   RNA_snn_res.1.2 RNA_snn_res.1.3
#> AGGGCGCTATTTCC-1               3               2
#> GGAGACGATTCGTT-1              10              12
#> CACCGTTGTCGTAG-1              11              13
#> TATCGTACACGCAT-1               6               7
#> TACGAGACCTATTC-1               0               0
#> GTACTACTCATACG-1              11              13
#>                   RNA_snn_res.1.4 RNA_snn_res.1.5
#> AGGGCGCTATTTCC-1               2               2
#> GGAGACGATTCGTT-1              12              13
#> CACCGTTGTCGTAG-1              13              14
#> TATCGTACACGCAT-1               7               8
#> TACGAGACCTATTC-1               0               0
#> GTACTACTCATACG-1              13              14
#>                   RNA_snn_res.1.6 RNA_snn_res.1.7
#> AGGGCGCTATTTCC-1               2               2
#> GGAGACGATTCGTT-1              13              13
#> CACCGTTGTCGTAG-1              14              14
```

```
#> TATCGTACACGCAT-1               7               8
#> TACGAGACCTATTC-1               0               0
#> GTACTACTCATACG-1              14              14
#>                  RNA_snn_res.1.8 RNA_snn_res.1.9
#> AGGGCGCTATTTCC-1               1               2
#> GGAGACGATTCGTT-1              14              14
#> CACCGTTGTCGTAG-1              15              15
#> TATCGTACACGCAT-1               9              10
#> TACGAGACCTATTC-1               7               7
#> GTACTACTCATACG-1              15              15
#>                  RNA_snn_res.2 Naive_CD4_T1  cell_label
#> AGGGCGCTATTTCC-1             1   -0.3540023        <NA>
#> GGAGACGATTCGTT-1            14    2.1376158 Naive_CD4_T
#> CACCGTTGTCGTAG-1            15   -1.1487836        <NA>
#> TATCGTACACGCAT-1             7    0.5557941        <NA>
#> TACGAGACCTATTC-1             6    1.4250065 Naive_CD4_T
#> GTACTACTCATACG-1            15   -0.2082793        <NA>
#>                  SingleR.labels     S.Score   G2M.Score
#> AGGGCGCTATTTCC-1       Monocyte -0.06808986 -0.08878508
#> GGAGACGATTCGTT-1        T_cells  0.09975063 -0.06812238
#> CACCGTTGTCGTAG-1       Monocyte -0.12770457  0.16052598
#> TATCGTACACGCAT-1    Neutrophils  0.08685571 -0.08316351
#> TACGAGACCTATTC-1        T_cells -0.02572695  0.02074759
#> GTACTACTCATACG-1       Monocyte -0.07294007 -0.01517046
#>                  Phase
#> AGGGCGCTATTTCC-1    G1
#> GGAGACGATTCGTT-1     S
#> CACCGTTGTCGTAG-1   G2M
#> TATCGTACACGCAT-1     S
#> TACGAGACCTATTC-1   G2M
#> GTACTACTCATACG-1    G1
```

We can then check the cell phase on the UMAP. In this dataset, phase isn't driving the clustering, and would not require any further handling.

```
DimPlot(seurat_object, reduction = 'umap_harmony', group.by = "Phase")
```



Where a bias *is* present, your course of action depends on the task at hand. It might involve 'regressing out' the cell cycle variation when scaling data `ScaleData(kang, vars.to.regress="Phase")`, omitting cell-cycle dominated clusters, or just accounting for it in your differential expression calculations.

If you are working with non-human data, you will need to convert these gene lists, or find new cell cycle associated genes in your species.

Part III

Other Resources

Chapter 15

Resources

Useful resources for next steps.

15.0.1 Suggested Further Reading Material

- Orchestrating Single Cell Analysis with Bioconductor - this book teaches single cell analysis with the bioconductor ecosystem of packages rather than Seurat. Regardless of your preference for Bioconductor or Seurat, it provides an excellent grounding and further depth and rationale behind each step of a single cell analysis.
- Seurat tutorials for gene expression, spatial & multimodal analysis
- Getting started with Signac - the sibling package to Seurat for scATAC analysis
- Monocle documentationn for trajectories
- Cell Annotation with SingleR
- VDJ analysis with Immcantation

15.0.2 Useful links arising from the discussion during the previous workshop

- 10x Genomics link to ribosomal protein expression
- 10x Genomics link to mitochondrial gene expression
- scRNA Tools, catalogue of tools for scRNA Seq analysis

15.0.2.1 Data interpretation

- Interactive website explaining UMAP and comparision to t-SNE.
- OSCA, dimensionality reduction interpretation
- A simple description of what PCA and UMAP do, with a 3D example.

15.0.2.2 Data tools and visualisation

- scTransform Vignette
- Link to the workflowr library
- iSEE Bioconductor library, interactive explorer
- ShinyCell makes interactive Shiny app from Seurat output
- iCellR interactive data explorer
- Diffusion maps for single cell instead of umaps *Projections of a high-dimensional dataset with an animated scatter-plot

15.0.2.3 Papers

- Doublet cell detection method benchmarking paper.
- From Louvain to Leiden: guaranteeing well-connected communities

15.0.2.4 Reference data and databases

- Gene tissue expression database
- ImmGen Database and Explorer
- Single Cell Study Portal from The Broad
- Common ref data for cell indexing
- Azimuth is a Seurat-friendly reference-based annotation tool
- Celaref, cell reference annotation tool

15.1 Help and fruther Resources

Seurat Vignettes

https://satijalab.org/seurat/index.html

There are a good many Seurat vigettes for different aspects of the Seurat package. E.g.

- Guided Clustering tutorial : We've just worked through this
- Differential expression : An Exploration of differential expression methods within Seurat
- Data integration : Seurat's data integration is a popular method to combine different datasets into one joint analysis.

Seurat Cheatsheet

https://satijalab.org/seurat/articles/essential_commands.html

A useful resource for asking; How can I do 'X' with my seurat object?

OSCA

https://bioconductor.org/books/release/OSCA/

An comprehensive resource for analysis approaches for single cell data. This uses the SingleCellExperiment bioconductor ecosystem, but alot of the same principle still apply.

This includes a good discussion of useing pseudobulk approaches, worth checking out for differential expression analyses.

MBP training Reading list

https://monashbioinformaticsplatform.github.io/Single-Cell-Workshop/

A workshop page for a previous workshop (upon which this one is based) run by Monash Bioinformatics Platform - down the bottom there is an extensive list of useful single cell links and resources.

Biocommons Single Cell Omics

https://www.biocommons.org.au/single-cell-omics

Join the single cell omics community resources being setup by biocommons.

15.2 Data

Demo 10X data

https://www.10xgenomics.com/resources/datasets

10X genomics have quite a few example datasets availble for download (including PBMC3k). This is a useful resource if you want to see what the 'raw' data looks like for a particular technology.

GEO

https://www.ncbi.nlm.nih.gov/geo/

Many papers publish their raw single cell data in GEO. Formats vary, but often you can find the counts matrix. # (PART) Other resources {-}

Seurat data

https://github.com/satijalab/seurat-data

Package for obtaining a few datasets as seurat objects.

15.3 Analysis Tools

A handful of the many tools that might be worth checking out for next steps.

Cyclone

https://pubmed.ncbi.nlm.nih.gov/26142758/

Part of the scran package, cyclone is a(nother) method for determining cell phase. Doco

Harmony

https://portals.broadinstitute.org/harmony/articles/quickstart.html

Method for integration of multiple single cell datasets.

SingleR

http://bioconductor.org/books/release/SingleRBook/

There is extensive documentation for the singleR package in the 'singleR' book.

Scrublet

https://github.com/swolock/scrublet

A python based tool for doublet detection. One of many tools in this space.

ScVelo

https://scvelo.readthedocs.io/

A package for single cell RNA velocity analysis, useful for developmental/pseudotime trajectories. Python/scanpy based.

Monocle

https://cole-trapnell-lab.github.io/monocle3/

A package for single cell developmental//pseudotime trajectory analysis.

TidySeurat

https://stemangiola.github.io/tidyseurat/

For fans of tidyverse-everything, there's tidyseurat. Example workflow here

15.4 Preprocessing Tools

Tooks that process raw sequencing data into counts matricies

Cell Ranger

https://support.10xgenomics.com/single-cell-gene-expression/software/pipelines/latest/what-is-cell-ranger

CellRanger is the 10X tool that takes raw fastq sequence files and produces the counts matricies that are the starting point for today's analysis. It only works for 10X data.

STARSolo

STAR is an aligner (which is actually used within cell ranger). STARSolo is a tool for producing counts matricies, and is configurable enough for use with multiple technologies.

https://github.com/alexdobin/STAR/blob/master/docs/STARsolo.md

Part IV

Seurat Object

Chapter 16

Structure

16.1 Load an existing Seurat object

The data we're working with today is a small dataset of about 5000 PBMCs (peripheral blood mononuclear cells) from a healthy donor.

First, load Seurat package.

```
library(Seurat)
```

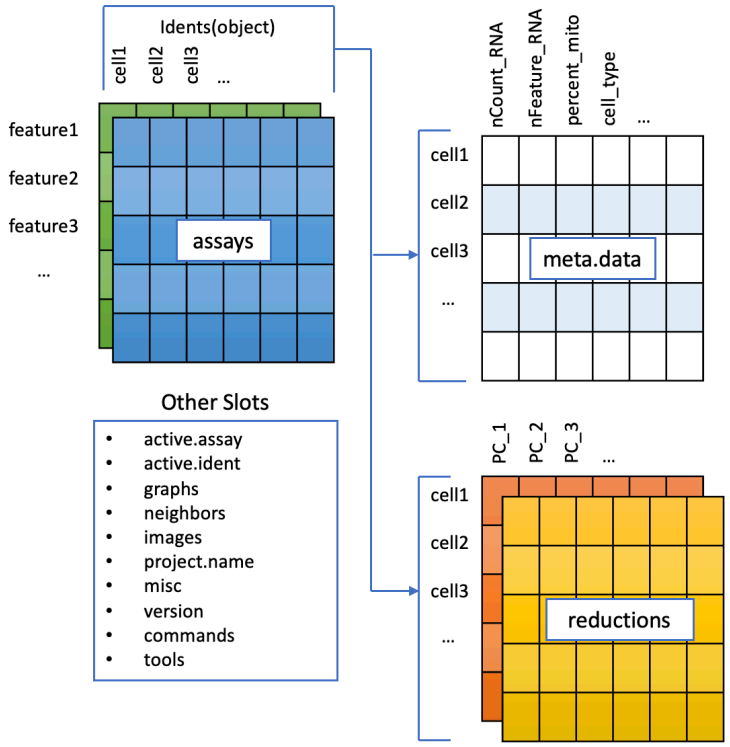
And here's the one we prepared earlier. Seurat objects are usually saved as '.rds' files, which is an R format for storing binary data (not-text or human-readable). The functions `readRDS()` can load it.

```
seurat_object <- readRDS("data/kang2018.rds")
```

Discussion: The Seurat Object in R

Lets take a look at the seurat object we have just created in R, `seurat_object`

To accomodate the complexity of data arising from a single cell RNA seq experiment, the seurat object keeps this as a container of multiple data tables that are linked.



The functions in seurat can access parts of the data object for analysis and visualization, we will cover this later on.

There are a couple of concepts to discuss here.

Class

These are essentially data containers in R as a class, and can accessed as a variable in the R environment.

Classes are pre-defined and can contain multiple data tables and metadata. For Seurat, there are three types.

- Seurat - the main data class, contains all the data.
- Assay - found within the Seurat object. Depending on the experiment a cell could have data on RNA, ATAC etc measured
- DimReduc - for PCA and UMAP

Slots

Slots are parts within a class that contain specific data. These can be lists, data tables and vectors and can be accessed with conventional R methods.

Data Access

Many of the functions in Seurat operate on the data class and slots within them

seamlessly. There maybe occasion to access these separately to `hack` them, however this is an advanced analysis method.

The ways to access the slots can be through methods for the class (functions) or with standard R accessor nomenclature.

Examples of accessing a Seurat object

The `assays` slot in `seurat_object` can be accessed with `seurat_object@assays`.

The `RNA` assay can be accessed from this with `seurat_object@assays$RNA`.

We often want to access assays, so Seurat nicely gives us a shortcut `seurat_object$RNA`. You may sometimes see an alternative notation `seurat_object[["RNA"]]`.

In general, slots that are always in an object are accessed with `@` and things that may be different in different data sets are accessed with `$`.

Have a go

Use `str` to look at the structure of the Seurat object `seurat_object`.

What is in the `meta.data` slot within your Seurat object currently? What type of data is contained here?

Where is our count data within the Seurat object?

16.2 What's in there?

Some of the most important information for working with Seurat objects is in the metadata. This is cell level information - each row is one cell, identified by its barcode. Extra information gets added to this table as analysis progresses.

```
head(seurat_object@meta.data)
#>                     orig.ident nCount_RNA nFeature_RNA  ind
#> AGGGCGCTATTTCC-1 SeuratProject       2020          523 1256
#> GGAGACGATTCGTT-1 SeuratProject        840          381 1256
#> CACCGTTGTCGTAG-1 SeuratProject       3097          995 1016
#> TATCGTACACGCAT-1 SeuratProject       1011          540 1488
#> TGACGCCTTGCTTT-1 SeuratProject        570          367  101
#> TACGAGACCTATTC-1 SeuratProject       2399          770 1244
#>                  stim              cell multiplets
#> AGGGCGCTATTTCC-1 stim   CD14+ Monocytes    singlet
#> GGAGACGATTCGTT-1 stim       CD4 T cells    singlet
#> CACCGTTGTCGTAG-1 ctrl FCGR3A+ Monocytes    singlet
#> TATCGTACACGCAT-1 stim           B cells    singlet
#> TGACGCCTTGCTTT-1 ctrl       CD4 T cells       ambs
#> TACGAGACCTATTC-1 stim       CD4 T cells    singlet
```

That doesn't have any gene expression though, that's stored in an 'Assay'. The Assay structure has some nuances (see discussion below), but there are functions that get the assay data out for you.

By default this object will return the normalised data (from the only assay in this object, called RNA). Every '.' is a zero.

```
GetAssayData(seurat_object)[1:15,1:2]
#> 15 x 2 sparse Matrix of class "dgCMatrix"
#>               AGGGCGCTATTTCC-1 GGAGACGATTCGTT-1
#> MIR1302-10                   .                .
#> FAM138A                      .                .
#> OR4F5                        .                .
#> RP11-34P13.7                 .                .
#> RP11-34P13.8                 .                .
#> AL627309.1                   .                .
#> RP11-34P13.14                .                .
#> RP11-34P13.9                 .                .
#> AP006222.2                   .                .
#> RP4-669L17.10                .                .
#> OR4F29                       .                .
#> RP4-669L17.2                 .                .
#> RP5-857K21.15                .                .
#> RP5-857K21.1                 .                .
#> RP5-857K21.2                 .                .
```

But the raw counts data is accessible too.

```
GetAssayData(seurat_object, slot='counts')[1:15,1:2]
#> 15 x 2 sparse Matrix of class "dgCMatrix"
#>               AGGGCGCTATTTCC-1 GGAGACGATTCGTT-1
#> MIR1302-10                   .                .
#> FAM138A                      .                .
#> OR4F5                        .                .
#> RP11-34P13.7                 .                .
#> RP11-34P13.8                 .                .
#> AL627309.1                   .                .
#> RP11-34P13.14                .                .
#> RP11-34P13.9                 .                .
#> AP006222.2                   .                .
#> RP4-669L17.10                .                .
#> OR4F29                       .                .
#> RP4-669L17.2                 .                .
#> RP5-857K21.15                .                .
#> RP5-857K21.1                 .                .
#> RP5-857K21.2                 .                .
```

Chapter 17

Acknowledgements

This material is mostly based on the Seurat introductory tutorial.

It also draws from material for a workshop provided by QCIF developed by Sarah Williams and Ahmed Mehdi for the Australian Biocommons.

Chapter 18

Session info

```
library(pander)
demo.Rmd_session <- sessionInfo()
pander(demo.Rmd_session)
```

R version 4.4.0 (2024-04-24)

Platform: x86_64-pc-linux-gnu

locale: *LC_CTYPE=en_AU.UTF-8*, *LC_NUMERIC=C*, *LC_TIME=en_AU.UTF-8*, *LC_COLLATE=en_AU.UTF-8*, *LC_MONETARY=en_AU.UTF-8*, *LC_MESSAGES=en_AU.UTF-8*, *LC_PAPER=en_AU.UTF-8*, *LC_NAME=C*, *LC_ADDRESS=C*, *LC_TELEPHONE=C*, *LC_MEASUREMENT=en_AU.UTF-8* and *LC_IDENTIFICATION=C*

attached base packages: *stats4*, *stats*, *graphics*, *grDevices*, *utils*, *datasets*, *methods* and *base*

other attached packages: *pander(v.0.6.5)*, *edgeR(v.4.2.2)*, *limma(v.3.60.6)*, *celldex(v.1.14.0)*, *SingleR(v.2.6.0)*, *SingleCellExperiment(v.1.26.0)*, *SummarizedExperiment(v.1.34.0)*, *Biobase(v.2.64.0)*, *GenomicRanges(v.1.56.2)*, *GenomeInfoDb(v.1.40.1)*, *IRanges(v.2.38.1)*, *S4Vectors(v.0.42.1)*, *BiocGenerics(v.0.50.0)*, *MatrixGenerics(v.1.16.0)*, *matrixStats(v.1.4.1)*, *clustree(v.0.5.1)*, *ggraph(v.2.2.1)*, *harmony(v.1.2.1)*, *Rcpp(v.1.0.13)*, *patchwork(v.1.3.0)*, *Seurat(v.5.1.0)*, *SeuratObject(v.5.0.2)*, *sp(v.2.1-4)*, *ggplot2(v.3.5.1)* and *dplyr(v.1.1.4)*

loaded via a namespace (and not attached): *spatstat.sparse(v.3.1-0)*, *httr(v.1.4.7)*, *RColorBrewer(v.1.1-3)*, *alabaster.base(v.1.4.2)*, *tools(v.4.4.0)*, *sctransform(v.0.4.1)*, *backports(v.1.5.0)*, *utf8(v.1.2.4)*, *R6(v.2.5.1)*, *HDF5Array(v.1.32.1)*, *lazyeval(v.0.2.2)*, *uwot(v.0.2.2)*, *rhdf5filters(v.1.16.0)*, *withr(v.3.0.2)*, *gridExtra(v.2.3)*, *progressr(v.0.15.0)*, *cli(v.3.6.3)*, *textshaping(v.0.4.0)*, *spatstat.explore(v.3.3-*

3), *fastDummies(v.1.7.4)*, *alabaster.se(v.1.4.1)*, *labeling(v.0.4.3)*, *spatstat.data(v.3.1-2)*, *ggridges(v.0.5.6)*, *pbapply(v.1.7-2)*, *systemfonts(v.1.1.0)*, *parallelly(v.1.38.0)*, *rstudioapi(v.0.17.1)*, *RSQLite(v.2.3.7)*, *generics(v.0.1.3)*, *ica(v.1.0-3)*, *spatstat.random(v.3.3-2)*, *Matrix(v.1.7-0)*, *ggbeeswarm(v.0.7.2)*, *fansi(v.1.0.6)*, *abind(v.1.4-8)*, *lifecycle(v.1.0.4)*, *yaml(v.2.3.10)*, *rhdf5(v.2.48.0)*, *SparseArray(v.1.4.8)*, *BiocFileCache(v.2.12.0)*, *Rtsne(v.0.17)*, *grid(v.4.4.0)*, *blob(v.1.2.4)*, *promises(v.1.3.0)*, *ExperimentHub(v.2.12.0)*, *crayon(v.1.5.3)*, *miniUI(v.0.1.1.1)*, *lattice(v.0.22-6)*, *beachmat(v.2.20.0)*, *cowplot(v.1.1.3)*, *KEGGREST(v.1.44.1)*, *pillar(v.1.9.0)*, *knitr(v.1.48)*, *future.apply(v.1.11.3)*, *codetools(v.0.2-20)*, *leiden(v.0.4.3.1)*, *glue(v.1.8.0)*, *spatstat.univar(v.3.0-1)*, *data.table(v.1.16.2)*, *gypsum(v.1.0.1)*, *vctrs(v.0.6.5)*, *png(v.0.1-8)*, *spam(v.2.11-0)*, *gtable(v.0.3.6)*, *cachem(v.1.1.0)*, *xfun(v.0.48)*, *S4Arrays(v.1.4.1)*, *mime(v.0.12)*, *tidygraph(v.1.3.1)*, *survival(v.3.5-8)*, *tinytex(v.0.53)*, *statmod(v.1.5.0)*, *fitdistrplus(v.1.2-1)*, *ROCR(v.1.0-11)*, *nlme(v.3.1-164)*, *bit64(v.4.5.2)*, *alabaster.ranges(v.1.4.2)*, *filelock(v.1.0.3)*, *RcppAnnoy(v.0.0.22)*, *irlba(v.2.3.5.1)*, *vipor(v.0.4.7)*, *KernSmooth(v.2.23-22)*, *colorspace(v.2.1-1)*, *DBI(v.1.2.3)*, *ggrastr(v.1.0.2)*, *tidyselect(v.1.2.1)*, *bit(v.4.5.0)*, *compiler(v.4.4.0)*, *curl(v.5.2.3)*, *httr2(v.1.0.5)*, *hdf5r(v.1.3.11)*, *DelayedArray(v.0.30.1)*, *plotly(v.4.10.4)*, *bookdown(v.0.41)*, *checkmate(v.2.3.2)*, *scales(v.1.3.0)*, *lmtest(v.0.9-40)*, *rappdirs(v.0.3.3)*, *stringr(v.1.5.1)*, *digest(v.0.6.37)*, *goftest(v.1.2-3)*, *spatstat.utils(v.3.1-0)*, *alabaster.matrix(v.1.4.2)*, *rmarkdown(v.2.28)*, *XVector(v.0.44.0)*, *RhpcBLASctl(v.0.23-42)*, *htmltools(v.0.5.8.1)*, *pkgconfig(v.2.0.3)*, *sparseMatrixStats(v.1.16.0)*, *highr(v.0.11)*, *dbplyr(v.2.5.0)*, *fastmap(v.1.2.0)*, *rlang(v.1.1.4)*, *htmlwidgets(v.1.6.4)*, *UCSC.utils(v.1.0.0)*, *shiny(v.1.9.1)*, *DelayedMatrixStats(v.1.26.0)*, *farver(v.2.1.2)*, *zoo(v.1.8-12)*, *jsonlite(v.1.8.9)*, *BiocParallel(v.1.38.0)*, *BiocSingular(v.1.20.0)*, *magrittr(v.2.0.3)*, *GenomeInfoDbData(v.1.2.12)*, *dotCall64(v.1.2)*, *Rhdf5lib(v.1.26.0)*, *munsell(v.0.5.1)*, *viridis(v.0.6.5)*, *reticulate(v.1.39.0)*, *alabaster.schemas(v.1.4.0)*, *stringi(v.1.8.4)*, *zlibbioc(v.1.50.0)*, *MASS(v.7.3-60.2)*, *AnnotationHub(v.3.12.0)*, *plyr(v.1.8.9)*, *parallel(v.4.4.0)*, *listenv(v.0.9.1)*, *ggrepel(v.0.9.6)*, *deldir(v.2.0-4)*, *Biostrings(v.2.72.1)*, *graphlayouts(v.1.2.0)*, *splines(v.4.4.0)*, *tensor(v.1.5)*, *locfit(v.1.5-9.10)*, *igraph(v.2.1.1)*, *spatstat.geom(v.3.3-3)*, *RcppHNSW(v.0.6.0)*, *reshape2(v.1.4.4)*, *ScaledMatrix(v.1.12.0)*, *BiocVersion(v.3.19.1)*, *evaluate(v.1.0.1)*, *BiocManager(v.1.30.25)*, *tweenr(v.2.0.3)*, *httpuv(v.1.6.15)*, *RANN(v.2.6.2)*, *tidyr(v.1.3.1)*, *purrr(v.1.0.2)*, *polyclip(v.1.10-7)*, *future(v.1.34.0)*, *scattermore(v.1.2)*, *ggforce(v.0.4.2)*, *rsvd(v.1.0.5)*, *xtable(v.1.8-4)*, *RSpectra(v.0.16-2)*, *later(v.1.3.2)*, *viridisLite(v.0.4.2)*, *ragg(v.1.3.3)*, *tibble(v.3.2.1)*, *memoise(v.2.0.1)*, *beeswarm(v.0.4.0)*, *AnnotationDbi(v.1.66.0)*, *cluster(v.2.1.6)* and *globals(v.0.16.3)*