scRNA-seq Tutorial

Leah Rosen, Donnacha Fitzgerald, Constantin Ahlmann-Eltze

04/03/2021

This tutorial roughly follows the Seurat pbms3k_tutorial: https://satijalab.org/seurat/v3.2/pbmc3k_tutorial.html with some of my own additions. For more Seurat tutorials see: https://satijalab.org/seurat/vignettes.html

We will be using some tidyverse and data.table packages. A nice blogpost outlining these is: https://wetlandscapes.com/blog/a-comparison-of-r-dialects/

Installs

This chunk of code will install the packages. You will only need to run it once.

```
#install.packages("Seurat")
#install.packages("devtools")
#devtools::install_github('satijalab/seurat-data')
#InstallData("pbmc3k")
#install.packages("gridExtra")
#install.packages("data.table")
#install.packages("ggplot2")
#if (!requireNamespace("BiocManager", quietly = TRUE))
#    install.packages("BiocManager")
#BiocManager::install("SingleCellExperiment")
#BiocManager::install("glmGamPoi")
#BiocManager::install("scds")
#install.packages("useful")
```

Imports

There are 3 main frameworks for scRNA-seq data analysis

In R:

- Seurat
  - Simple, but somewhat restrictive
  - Does not always contain the best method
- OSCA: A collection of bioconductor packages including scran, SingleCellExperiment, and batchelor
  - A lot harder to get comfortable with
  - More flexible

In python:

- Scanpy

It is useful to switch between frameworks to use the "best" method. Today we'll be mainly using Seurat, but switch to OSCA for doublet detection

First, we want to import the packages we will be using.

```
library(Seurat)
library(SeuratData)
library(gridExtra)
library(tidyverse)
library(data.table)
library(SingleCellExperiment)
library(scds)
library(useful)
```

Overview of a Typical scRNA-seq Pipeline

A typical scRNA-seq pipeline involves 4 preprocessing steps

1. Quality Control
2. Variance Stabilising Transformation
3. Highly Variable Gene Selection
4. Optional: Scaling

We will discuss these various steps today.

Further steps that we can do with the data are:

- Clustering
- Visualisation
- Celltype Assignment. 2 Approaches:
  - Marker Gene Selection
  - Reference-based
- Data integration/Batch Effect Correction

Load data

We will be working with a toy dataset from 10X Genomics.

10X Genomics

- Most widely used sequencing technology today, due to good throughput-coverage ratio
- Up to 10,000 cells per library
- ~3,500 genes per cell
- ~20,000 UMIs per cell
  - UMI - "unique molecular identifier"
  - Each poly-adenylated RNA molecule is given a unique barcode (UMI)
  - UMI control for PCR amplification bias

This dataset:

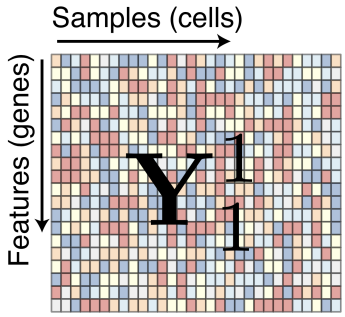


Figure 1: A count matrix

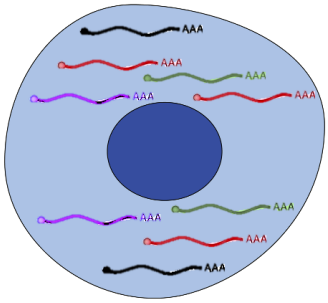


Figure 2: A schematic of poly-adenylated mRNA of different colours (i.e. genes) in a cell

- 2,700 Cells
- Species: Human
- Sample: Peripheral Blood Mononucleated Cells (PBMCs)
- Note: This is a toy dataset, down-sampled so everything runs quickly

Note: getting the error: "no slot of name"images" for this object of class "Seurat"" is totally normal. Don't worry if you get that.

```
data("pbmc3k") # Or, alternatively:
# pbmc.data <- Read10X(data.dir = "filtered_gene_bc_matrices/hg19/")
# pbmc3k <- CreateSeuratObject(counts = pbmc.data, project = "pbmc3k", min.cells = 3, min.features = 20
pbmc3k
```

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

As you can see, we have now loaded a dataset with 2700 samples (i.e. cells) across 13714 features (i.e. genes). We have one "active assay", meaning we have RNA data. We can access this assay and look at the count matrix (the "head" function means that we are only displaying the first 6 rows):

```
corner(as.matrix(pbmc3k@assays$RNA@counts))
```

```
##               AAACATACAACCAC AAACATTGAGCTAC AAACATTGATCAGC AAACCGTGCTTCCG
## AL627309.1                0              0             0              0
## AP006222.2                0              0             0              0
## RP11-206L10.2             0              0             0              0
## RP11-206L10.9             0              0             0              0
```

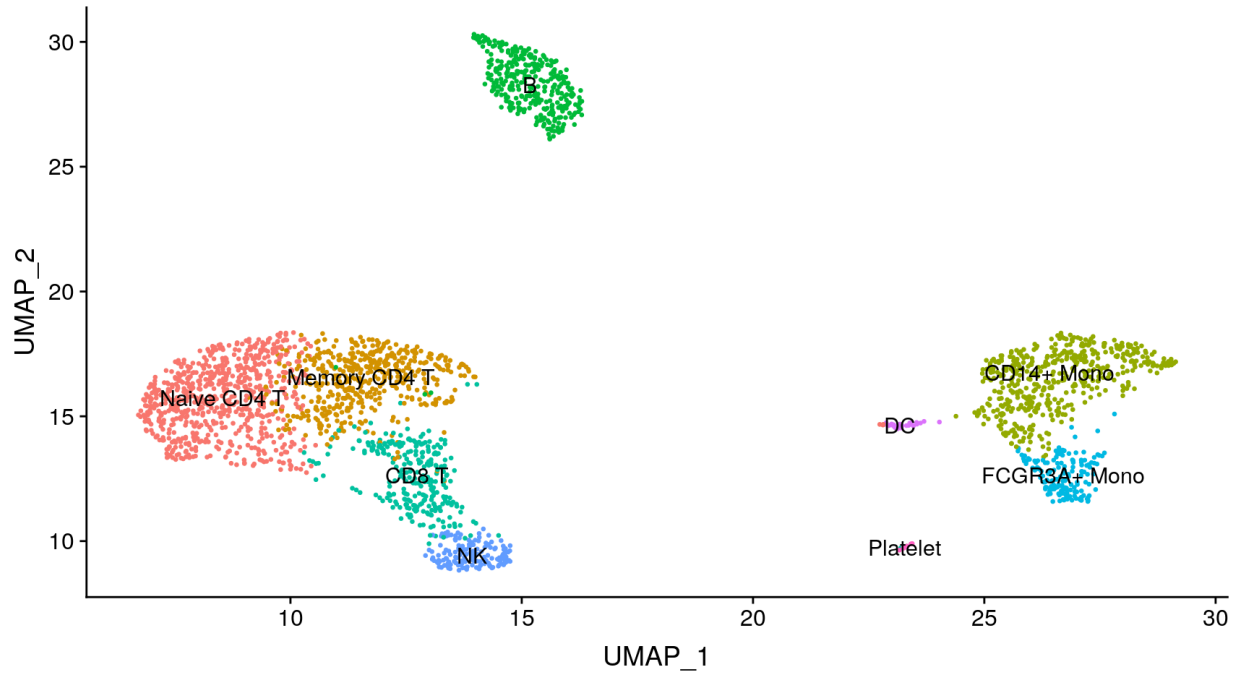


Figure 3: Seurat's UMAP visualisation and celltype annotation

```
## LINC00115                 0             0             0             0
##                AAACCGTGTATGCG
## AL627309.1                 0
## AP006222.2                 0
## RP11-206L10.2              0
## RP11-206L10.9              0
## LINC00115                  0
```

We also have a metadata file that gives us some information about each cell (again, we're displaying the first 6 cells)

```
head(pbmc3k@meta.data)
```

```
##                orig.ident nCount_RNA nFeature_RNA seurat_annotations
## AAACATACAACCAC     pbmc3k       2419          779       Memory CD4 T
## AAACATTGAGCTAC     pbmc3k       4903         1352                  B
## AAACATTGATCAGC     pbmc3k       3147         1129       Memory CD4 T
## AAACCGTGCTTCCG     pbmc3k       2639          960         CD14+ Mono
## AAACCGTGTATGCG     pbmc3k        980          521                 NK
## AAACGCACTGGTAC     pbmc3k       2163          781       Memory CD4 T
```

Quality Control the data

What do we need to check for?

- Doublets: These cells have many UMIs, many genes
- Stripped nuclei: These cells have few UMIs, few genes, low mitochondrial proportions

- Low quality cells (e.g., due to inefficient reverse transcription or PCR amplification): These cells have few UMIs, few genes, high mitochondrial proportions

Why do we need to remove these cells? Artefacts could obscure true underlying biology

Therefore 5 things we check when QCing a cell are:

- The number of transcripts captured (automatically saved as nCount_RNA in the metadata table)
- The number of unique genes captured (automatically saved as nFeature_RNA in the metadata table)
- The percentage of the counts that come from mitochondrial genes (we will need to calculate this)
- The percentage of the counts that come from ribosomal transcripts (we will need to calculate this), I think these are captured due to cryptic polyadenylation (but don't quote me on this)
- Whether a cell is a doublet

1. Calculate mitochondrial and ribosomal percentage

```
# calculate percent counts from mitochondrial genes
pbmc3k[["percent.mt"]] <- PercentageFeatureSet(pbmc3k, pattern = "MT-")


# calculate percent counts from mitochondrial genes:
ribo.genes <- c(grep(pattern = "^RPL", x = rownames(pbmc3k), value = TRUE),grep(pattern = "^RPS", x = r
pbmc3k[["percent.ribo"]] <- PercentageFeatureSet(pbmc3k, features = ribo.genes)
```

Note that the pattern can depend on your core or your organism. E.g. if you are given the genes as Ensembl IDs, you will first have to convert them to symbols. Or, if you're working in mouse, the genes aren't all caps, so you'll have to use "mt-" and "^Rpl", "^Rps".

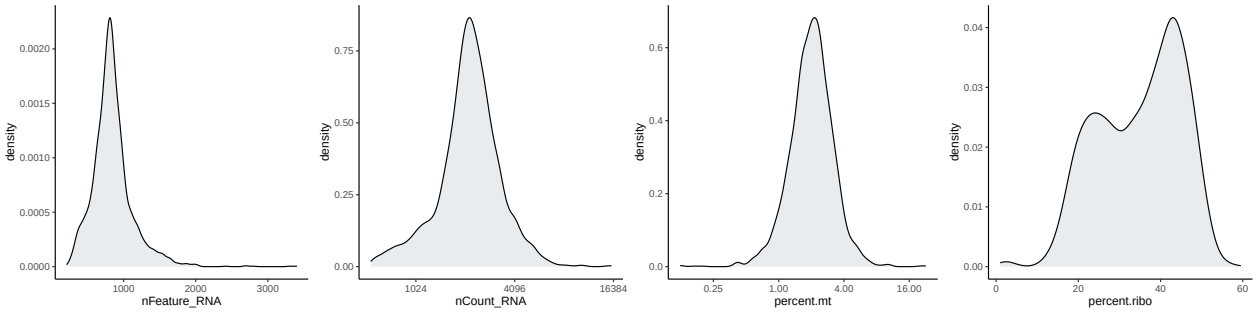
2. Plot QC metrics (number of genes, number of UMIs, percent mito, percent ribo)

```
pbmc3k@meta.data$barcode <- rownames(pbmc3k@meta.data)
md <- pbmc3k@meta.data

p1 <- (ggplot(md, aes(x=nFeature_RNA)) +
  geom_density(fill="#e9ecef", position = 'identity') +
  labs(fill="") +
  theme_classic())

p2 <- (ggplot(md, aes(x=nCount_RNA)) +
  geom_density(fill="#e9ecef", position = 'identity') +
  labs(fill="") +
  scale_x_continuous(trans='log2') +
  theme_classic())

p3 <- (ggplot(md, aes(x=percent.mt)) +
  geom_density(fill="#e9ecef", position = 'identity') +
  labs(fill="") +
  scale_x_continuous(trans='log2') +
  theme_classic())

p4 <- (ggplot(md, aes(x=percent.ribo)) +
  geom_density(fill="#e9ecef", position = 'identity') +
```

```
  labs(fill="") +
  theme_classic())

grid.arrange(p1, p2, p3, p4, ncol=4)
```

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

```
## Warning: Removed 3 rows containing non-finite values (stat_density).
```



3. Threshold the cells

Note that using a fixed threshold is the easiest method, but it isn't necessarily the best. There are methods for "adaptive thresholds".

```
min_nFeature_RNA <- 500
min_nCount_RNA <- 1250
min_percent.mt <- 0.5
max_percent.mt <- 5

md$barcode <- row.names(md)
md <- as.data.table(pbmc3k@meta.data)
md$pass_QC <- TRUE
md[nFeature_RNA < min_nFeature_RNA, pass_QC := FALSE]
md[nCount_RNA < min_nCount_RNA, pass_QC := FALSE]
md[percent.mt < min_percent.mt, pass_QC := FALSE]
md[percent.mt > max_percent.mt, pass_QC := FALSE]
```

4. Plot QC metrics for removed vs. retained cells

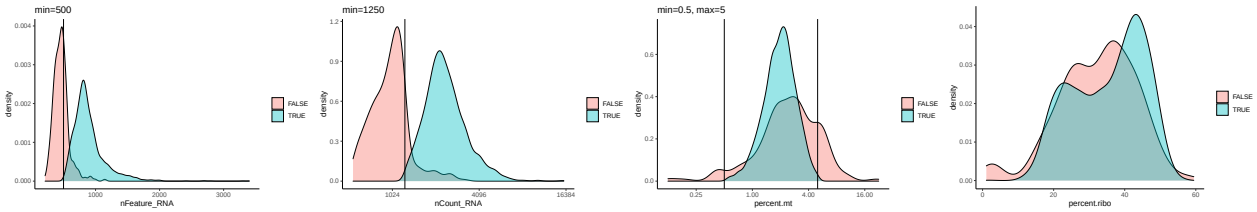
Let's plot the QC metrics splitting cells depending on whether they pass QC:

```
p1 <- (ggplot(md, aes(x=nFeature_RNA, fill=pass_QC)) +
  geom_density(alpha=0.4, position = 'identity') +
  labs(fill="") +
  geom_vline(xintercept=min_nFeature_RNA) +
  ggtitle(paste0("min=", min_nFeature_RNA)) +
  theme_classic())

p2 <- (ggplot(md, aes(x=nCount_RNA, fill=pass_QC)) +
  geom_density(alpha=0.4, position = 'identity') +
  labs(fill="") +
  geom_vline(xintercept=min_nCount_RNA) +
  ggtitle(paste0("min=", min_nCount_RNA)) +
  scale_x_continuous(trans='log2') +
  theme_classic())
```

```
p3 <- (ggplot(md, aes(x=percent.mt, fill=pass_QC)) +
  geom_density(alpha=0.4, position = 'identity') +
  labs(fill="") +
  geom_vline(xintercept=min_percent.mt) +
  geom_vline(xintercept=max_percent.mt) +
  ggtitle(paste0("min=", min_percent.mt, ", max=", max_percent.mt)) +
  scale_x_continuous(trans='log2') +
  theme_classic())

p4 <- (ggplot(md, aes(x=percent.ribo, fill=pass_QC)) +
  geom_density(alpha=0.4, position = 'identity') +
  labs(fill="") +
  theme_classic())

grid.arrange(p1, p2, p3, p4, ncol=4)
```

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

```
## Warning: Removed 3 rows containing non-finite values (stat_density).
```



5. Doublet Detection

Theory:

- Simply thresholding cells with many UMIs and genes isn't enough
- Many methods exist for detecting doublets in silico, most take one of two approaches:
  - Co-expression of genes that usually don't co-express
  - Simulation of doublets by combining random cells
- New technologies allow for identifying doublets in the data
- Based on data from my group, many undetected doublets
- *in silico* methods especially struggle with intra-cluster doublets

As far as I know, Seurat has no good inbuilt method for doing this. A good comparison of methods is this paper. When benchmarking methods myself, I found their hybrid method worked best. Their method takes the data in a different format, so we need to convert the data from Seurat to SingleCellExperiment

```
pbmc3k <- subset(pbmc3k, subset = nFeature_RNA>min_nFeature_RNA & nCount_RNA>min_nCount_RNA & percent.m

pbmc3k_sce <- as.SingleCellExperiment(pbmc3k)
pbmc3k_sce <- cxds_bcds_hybrid(pbmc3k_sce, estNdbl=TRUE)
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
## [09:08:29] WARNING: amalgamation/../src/learner.cc:1061: Starting in XGBoost 1.3.0, the default evalu
```

```
## Setting levels: control = 0, case = 1
## Setting direction: controls < cases
```

```
## Setting levels: control = 0, case = 1
```

```
## Setting direction: controls < cases
```

```
pbmc3k <- AddMetaData(
  object = pbmc3k,
  metadata = pbmc3k_sce$hybrid_call,
  col.name = "doublet"
)

pbmc3k <- subset(pbmc3k, subset = doublet==FALSE)
```

a side note

On my own data, I don't find this to be enough. When I was able to have more of a ground truth doublet calling (using cell hashing and genotyping) I found that this only got me to a TPR of 41% (on differentating cell line data), and 50% (on differentiating mESC data). Therefore, I remove any cells that are similar to called doublets, and thereby was able to get my TPR to 82% and 79%, respectively. This step completely overkills on this toy dataset, but I'll include my code here, in case you want to try it in the future:

```
pbmc3k <- NormalizeData(pbmc3k)
pbmc3k <- FindVariableFeatures(pbmc3k)
pbmc3k <- ScaleData(pbmc3k)
pbmc3k <- RunPCA(pbmc3k, verbose=FALSE)
pbmc3k <- FindNeighbors(pbmc3k, k.param=10, reduction = "pca")

md <- pbmc3k@meta.data

tmp <- data.table(hybrid_map = rowSums(pbmc3k@graphs$RNA_nn[,md[hybrid_call==TRUE]$cell])/10,
                  barcode = rownames(pbmc3k@graphs$RNA_snn))
md <- merge(md, tmp, by = "barcode", all.x=TRUE)
md[hybrid_call==TRUE, hybrid_map := NA]
md[,hybrid_call2 := TRUE]
md[hybrid_map<=((sum(md$hybrid_call==TRUE))/(dim(md)[1])), hybrid_call2 := FALSE]
rownames(md) <- md$cell
```

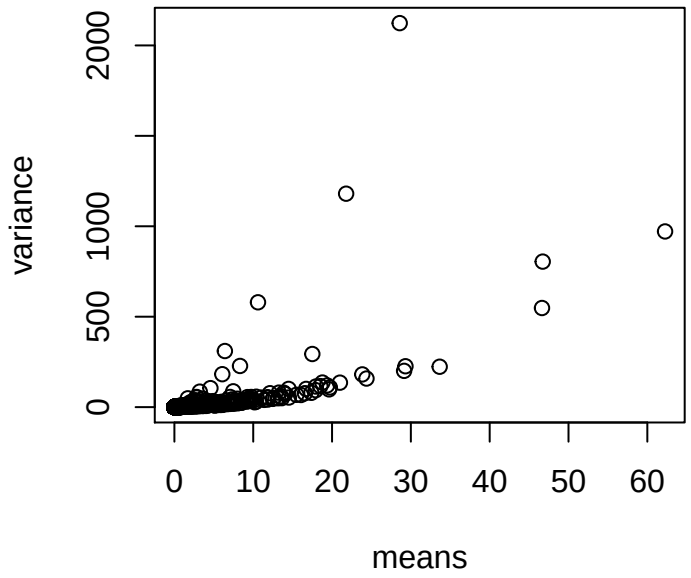
Discussion

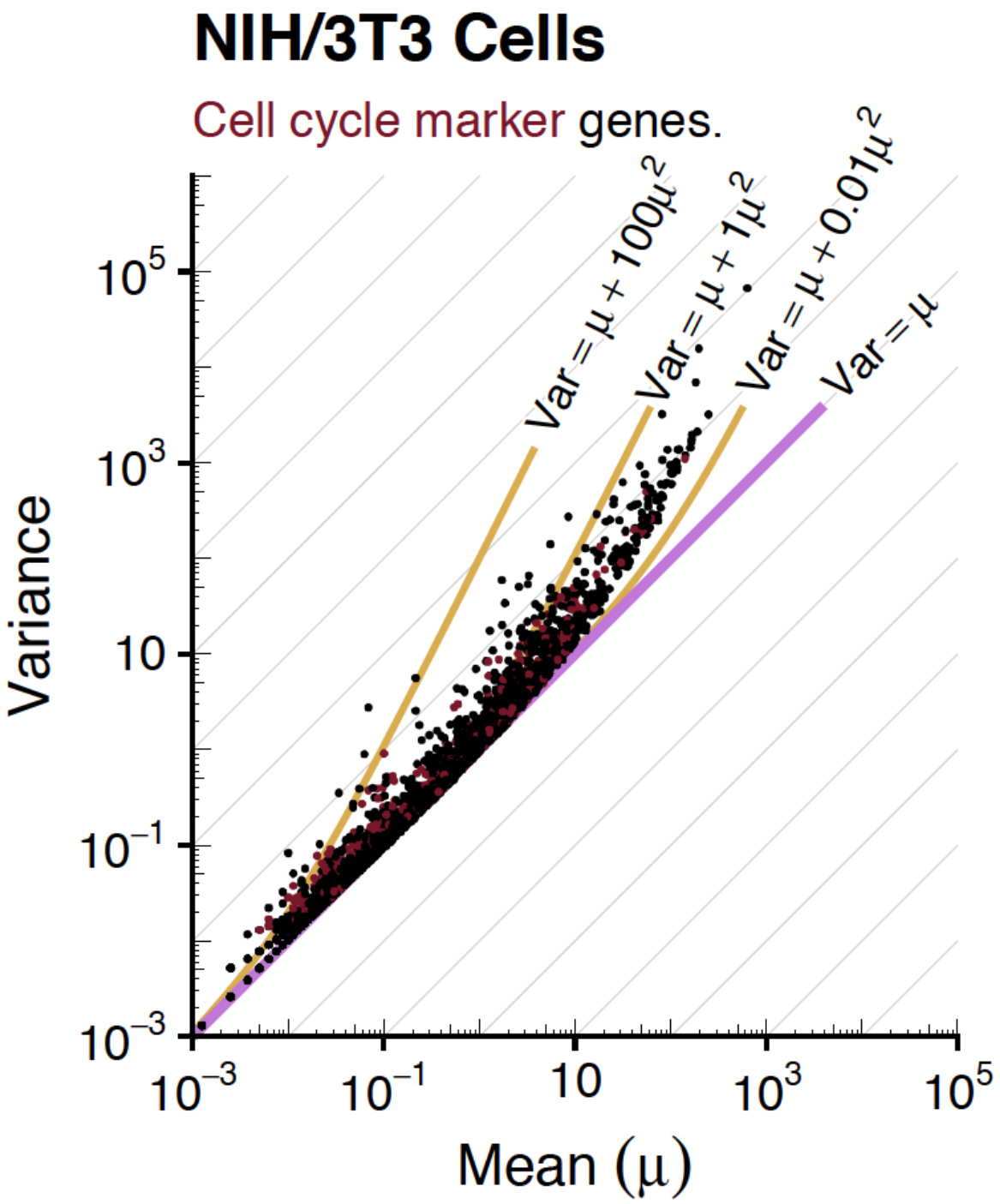
- How to choose thresholds?
- How do we know that we're not removing biologically meaningful cells? (e.g. a very transcriptionally inactive population)

Variance Stabilising Transformation

- Single cell data has a mean-variance relationship, as shown below: Genes with high means, vary a lot regardless of whether they are biologically meaningful in your system

```
means <- sparseMatrixStats::rowMeans2(pbmc3k@assays$RNA@counts)
variance <- sparseMatrixStats::rowVars(pbmc3k@assays$RNA@counts)
plot(means, variance)
```

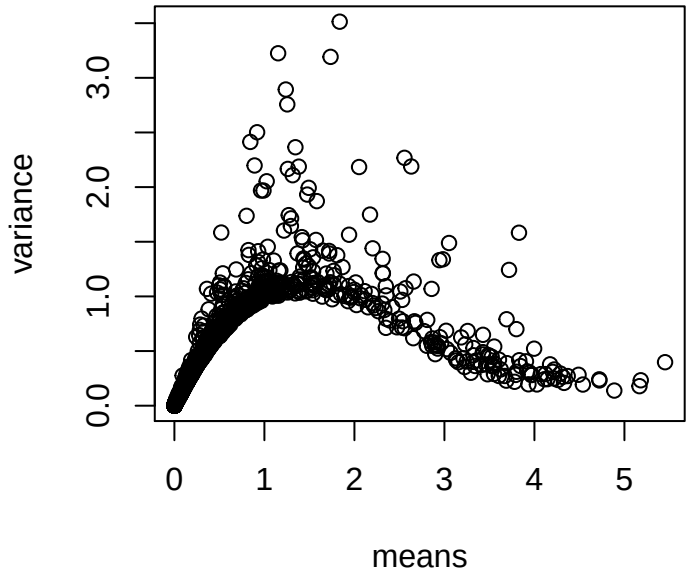

variance
0
500
1000
2000
0
10
20
30
40
50
60
means



- The mean-variance relation matches a Gamma-Poisson distribution
- The statistically most robust approach to handle such data are generalized linear models (GLMs), for example implemented by `DESeq2`, `edgeR`, or `glmGamPoi`. However they can be difficult to fit and exclude the application of many general purpose methods for clustering, dimension reduction and classification
- The alternative is to try to *normalize* the data. This mainly implies that the mean and variance become independent (homo-skedastic). That is why this step is also called **variance stabilizing transformation** (VST).

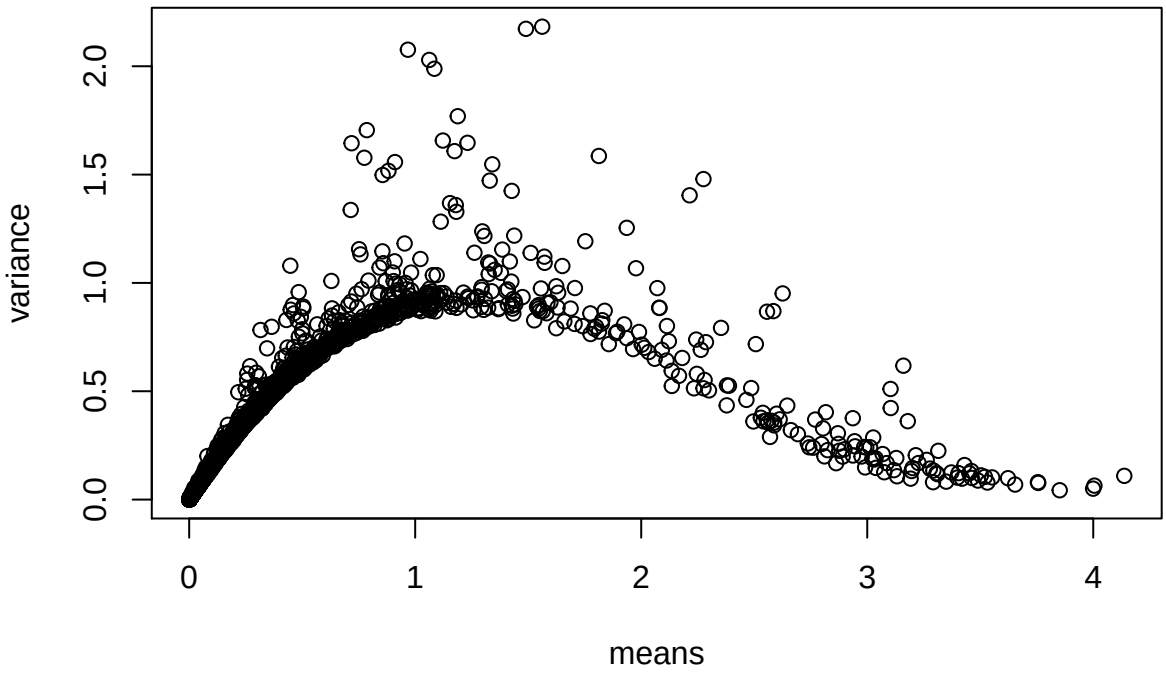
- The most basic method is log normalisation:
  - Kind of stabilises the variance
  - but not really...

```
pbmc3k <- NormalizeData(pbmc3k, normalization.method = "LogNormalize")

means <- sparseMatrixStats::rowMeans2(as.matrix(pbmc3k@assays$RNA@data))
variance <- sparseMatrixStats::rowVars(pbmc3k@assays$RNA@data)
plot(means, variance)
```



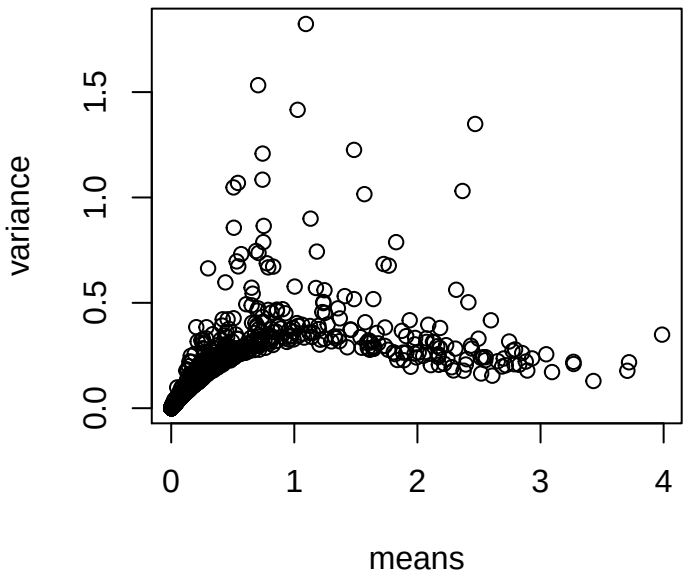
- Alternative VSTs are based on the delta-method
- Two popular alternatives
  - Ascombe's transformation: $2\sqrt{x}$
  - DESeq2's transformation: $\frac{1}{\sqrt{\alpha}}\text{asinh}(\sqrt{\alpha}x)$
- Not actually much more successful because all struggle to stabilize small counts (and always will see Warton (2018))

```
# Fix it to kind of reasonable value
alpha <- 0.1
stab_data <- 1 / sqrt(alpha) * asinh(sqrt(alpha) * pbmc3k@assays$RNA@data)

means <- sparseMatrixStats::rowMeans2(stab_data)
variance <- sparseMatrixStats::rowVars(stab_data)
plot(means, variance)
```



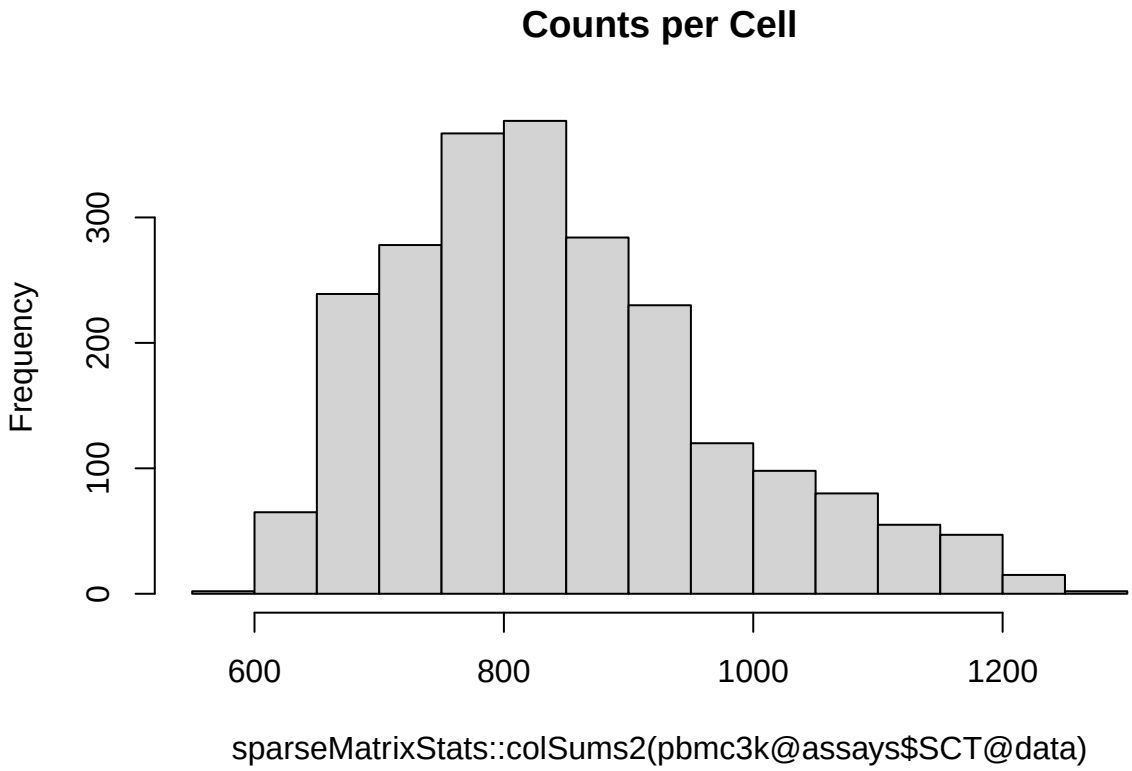
- Third apporach: `sctransform`
  - Based on pearson residuals
  - Trade's off stabilization at small means for potentially reduced sensitivity for real biological differences
  - Worked quite well in a recent benchmark (Germain et al. 2020)

```
pbmc3k <- SCTransform(pbmc3k, method="glmGamPoi",verbose=FALSE)

means <- sparseMatrixStats::rowMeans2(pbmc3k@assays$SCT@data)
variance <- sparseMatrixStats::rowVars(pbmc3k@assays$SCT@data)
plot(means, variance)
```



- Scaling

```
hist(sparseMatrixStats::colSums2(pbmc3k@assays$SCT@data), main = "Counts per Cell")
```



- Goal: remove library size-dependent differences in variance
- Seurat also scales each cell's counts by a scale factor before normalisation
- scran uses more sophisticated methods of estimating scale factor
- sctransform treats them a bit weirdly, but you can just suppose that they are handled as size facto
  - For somebody else's take on the topic see: https://ltla.github.io/SingleCellThoughts/general/transformation.html

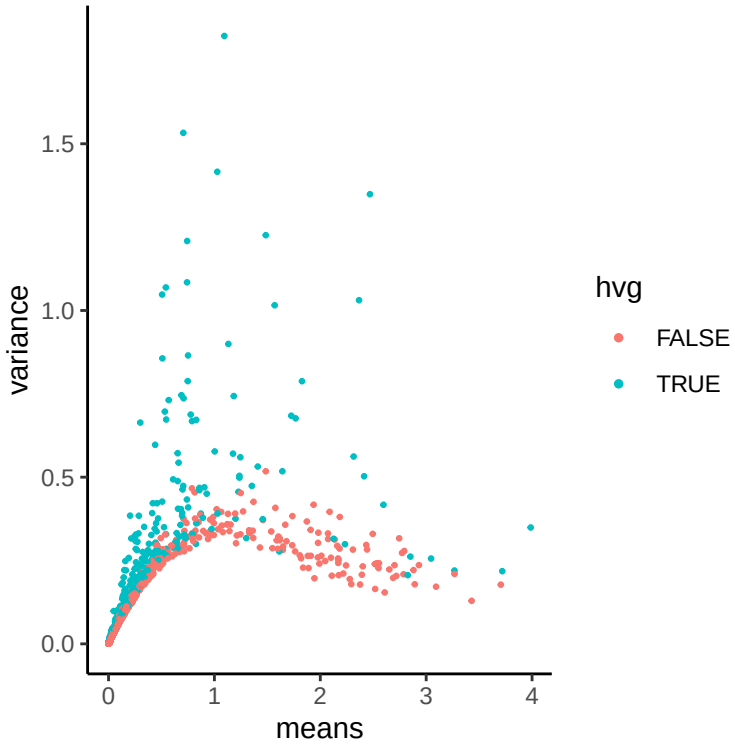
Discussion

- Where does the VST work well?
- Which genes does the VST work poorly for?
- When may we want to use count data instead?
- Are the data still sparse? What's the benefit of sparse data?

Highly Variable Gene Selection

- ~30,000 protein-coding genes in mice and men
- Most genes don't vary systematically between cells
- Variance in most genes is technical noise –> introduces noise but we don't gain any information

```
pbmc3k <- FindVariableFeatures(pbmc3k)

to.plot <- data.table(means=rowMeans(pbmc3k@assays$SCT@data),
                      variance=rowVars(as.matrix(pbmc3k@assays$SCT@data)),
                      hvg=(rownames(pbmc3k@assays$SCT@data) %in% VariableFeatures(pbmc3k)))

p <- ggplot(data=to.plot, mapping = aes(x=means, y=variance, colour=hvg)) +
  geom_point(size=0.5, alpha=1) +
  guides(colour = guide_legend(override.aes = list(size=1))) +
  theme_classic()
print(p)
```



Discussion

- How do you choose HVGs?
- How many HVGs should we use?

Scaling the Data

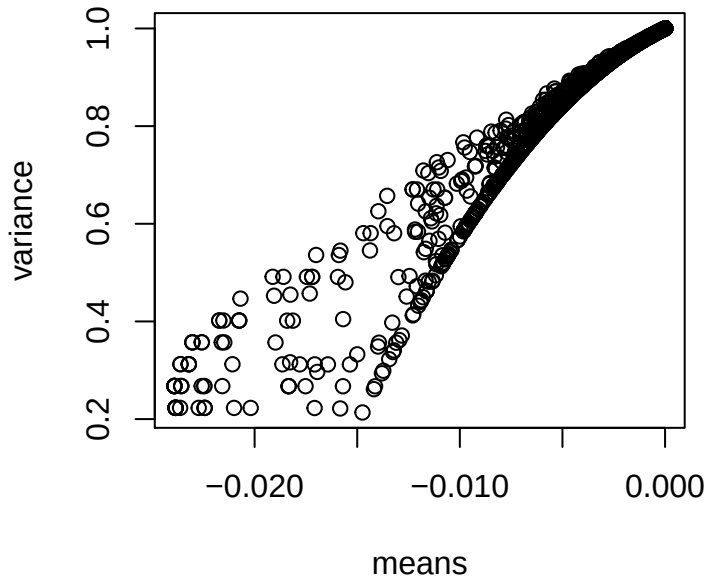
- Standardisation: All features (genes) have mean of 0 and unit variance
- Pro: Ensures that high-variance genes don't dominate
- Cons:
  - No weighting in how highly variable a gene is, and amplifies more minor variance
  - Blurs boundaries between subpopulations (especially when the gene has >2 states)
  - Removes log-fold changes
  - Biases towards number of differentially expressed genes (vs. amplitude of difference in expression)

- Personally, I think the cons outweigh the pros, but Seurat forces us to use it. To avoid it, Donnacha has a hack.
- Further discussion: https://ltla.github.io/SingleCellThoughts/general/standardization.html

```
pbmc3k <- ScaleData(pbmc3k)
```

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

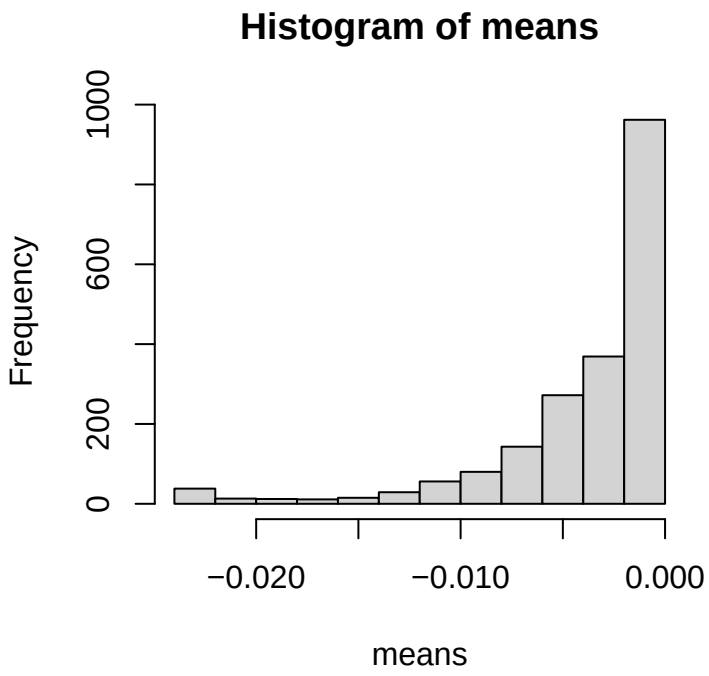
```
#pbmc3k@assays$SCT@scale.data <- pbmc3k@assays$SCT@data@x

means <- rowMeans(pbmc3k@assays$SCT@scale.data)
variance <- rowVars(pbmc3k@assays$SCT@scale.data)
p1 <- plot(means, variance)
```




```
print(p1)
```

```
## NULL
```

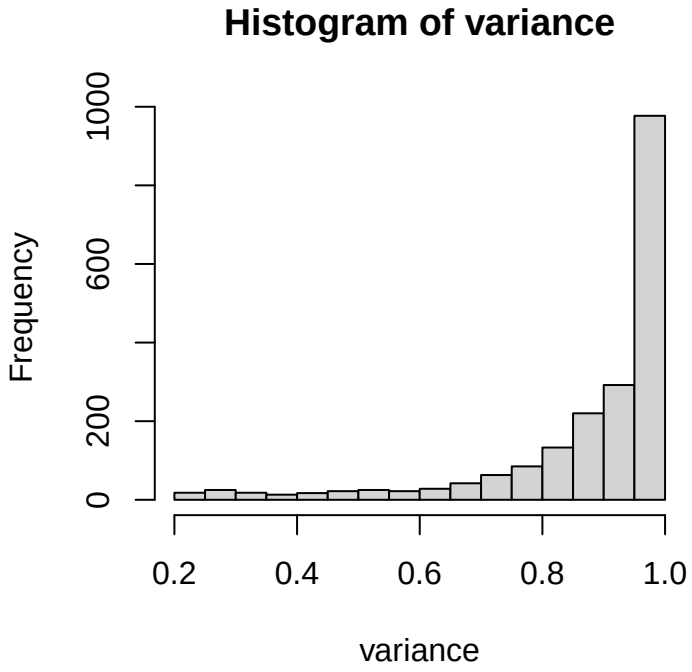
```
p2 <- hist(means)
```




```
print(p2)
```

```
## $breaks
##  [1] -2.400000e-02 -2.200000e-02 -2.000000e-02 -1.800000e-02 -1.600000e-02
##  [6] -1.400000e-02 -1.200000e-02 -1.000000e-02 -8.000000e-03 -6.000000e-03
## [11] -4.000000e-03 -2.000000e-03  2.252501e-14
##
## $counts
##  [1]  38  13  12  11  15  29  56  80 143 272 369 962
##
## $density
##  [1]   9.50   3.25   3.00   2.75   3.75   7.25  14.00  20.00  35.75  68.00
## [11]  92.25 240.50
##
## $mids
##  [1] -0.023 -0.021 -0.019 -0.017 -0.015 -0.013 -0.011 -0.009 -0.007 -0.005
## [11] -0.003 -0.001
##
## $xname
## [1] "means"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
```

```
p2 <- hist(variance)
```




```
print(p2)
```

```
## $breaks
##  [1] 0.20 0.25 0.30 0.35 0.40 0.45 0.50 0.55 0.60 0.65 0.70 0.75 0.80 0.85 0.90
## [16] 0.95 1.00
##
## $counts
##  [1]  18  25  18  13  17  22  25  22  28  42  63  85 133 220 292 977
##
## $density
##  [1] 0.18 0.25 0.18 0.13 0.17 0.22 0.25 0.22 0.28 0.42 0.63 0.85 1.33 2.20 2.92
## [16] 9.77
##
## $mids
##  [1] 0.225 0.275 0.325 0.375 0.425 0.475 0.525 0.575 0.625 0.675 0.725 0.775
## [13] 0.825 0.875 0.925 0.975
##
## $xname
## [1] "variance"
##
## $equidist
## [1] TRUE
##
## attr(,"class")
## [1] "histogram"
```

Clustering

A Side-Note on PCA

- We do a lot of calculations in "PCA" space
- Instead of working with each gene individually, they are combined into principle components (PCs)

- PCs are ordered by how much variance they explain. We only use the top 10-50 PCs –> removes noise

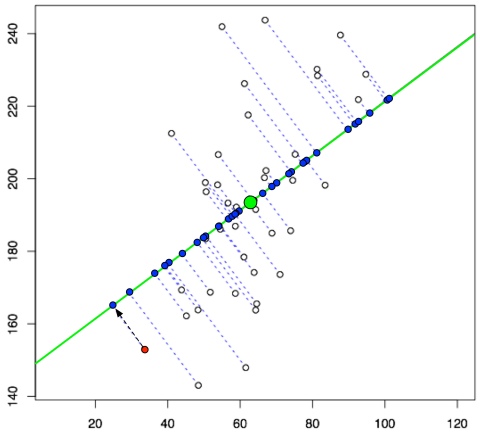


Figure 4: Projection of datapoints onto a PC

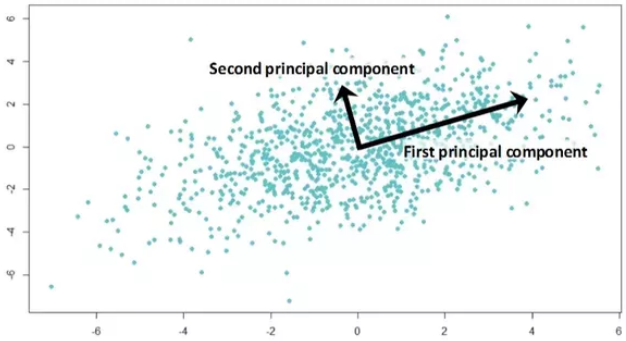


Figure 5: The 2 PCs for 2D data

Back to Clustering

Why?

- Sparsity of each individual cell -> More power in group
- We want to investigate cell types

How?

- Reduce dimensionality using PCA first
- Graph based
- Popularised by Seurat
- Several algorithms
- Build graph using knn or snn

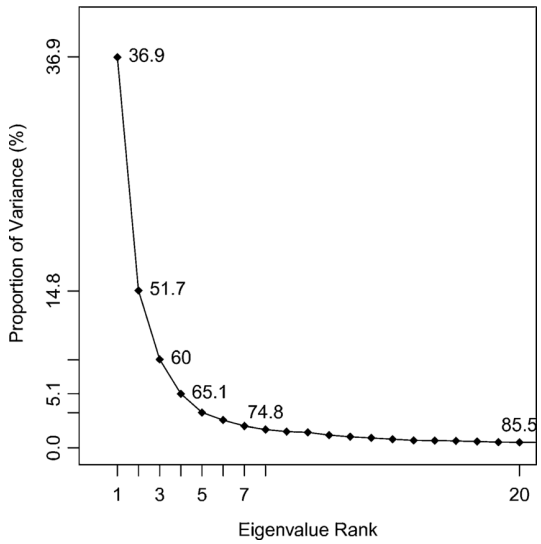


Figure 6: An elbow plot showing decreasing amounts of variance explained per PC

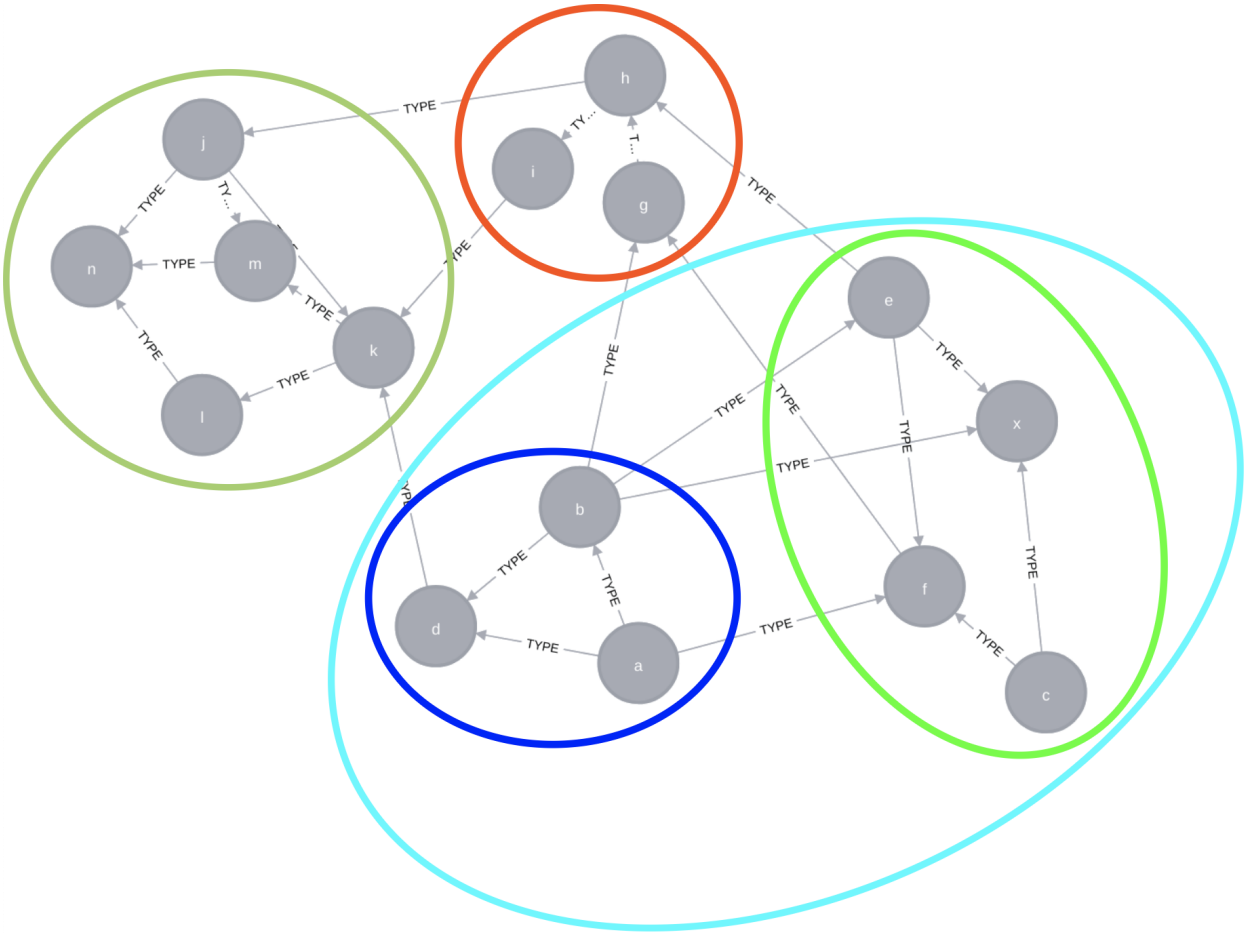


Figure 7: Schematic of Louvain Clustering, a commonly-used graph-based algorithm for clustering single cell data

```
pbmc3k <- RunPCA(pbmc3k, verbose=FALSE)
pbmc3k <- FindNeighbors(pbmc3k, k.param=10, reduction = "pca")
```

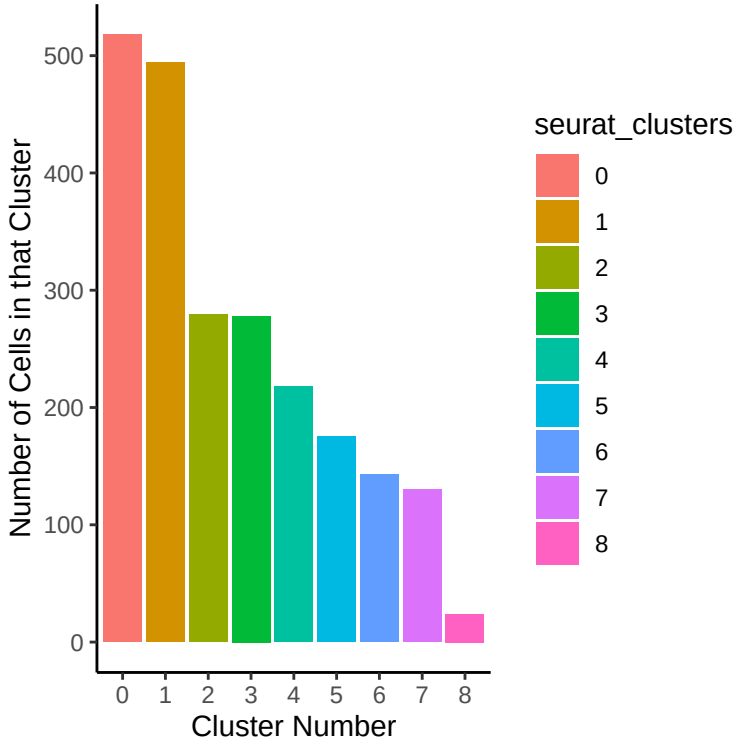
```
## Computing nearest neighbor graph
```

```
## Computing SNN
```

```
pbmc3k <- FindClusters(pbmc3k, resolution = 0.5)
```

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 2259
## Number of edges: 32790
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.8895
## Number of communities: 9
## Elapsed time: 0 seconds
```

```
p <- ggplot(data=as.data.table(pbmc3k@meta.data)[,.N,by="seurat_clusters"], mapping = aes(x=seurat_clus
  geom_bar(stat="identity") +
  labs(x="Cluster Number", y="Number of Cells in that Cluster") +
  theme_classic()
print(p)
```



Visualisation

We like seeing our data, but scRNA-seq data is high dimensional and humans struggle beyond 2. Therefore, we want to reduce our data to 2 dimensions:

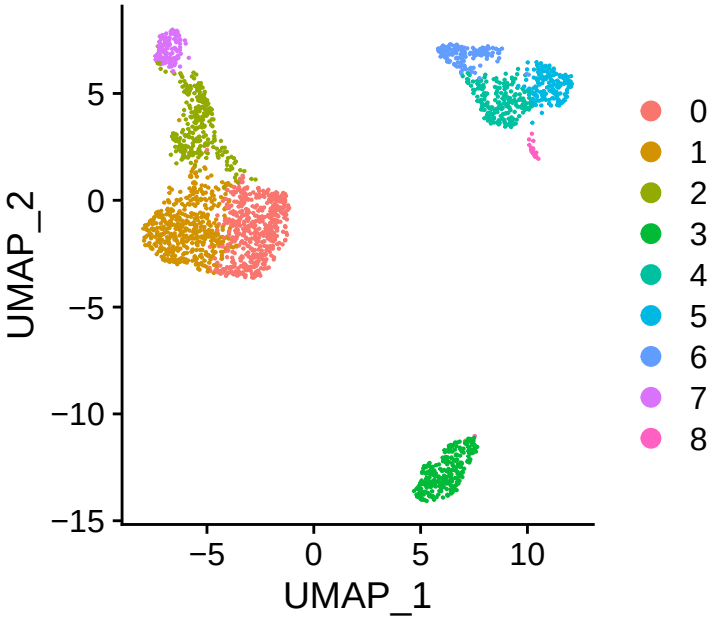
- Could use PCA

- Could use t-SNE: Maintains local distances
- Could use UMAP: Fits a manifold
- State of the art: PCA followed by UMAP

```
pbmc3k <- RunUMAP(pbmc3k, dims = 1:10)
```

```
## Warning: The default method for RunUMAP has changed from calling Python UMAP via reticulate to the R
## To use Python UMAP via reticulate, set umap.method to 'umap-learn' and metric to 'correlation'
## This message will be shown once per session
## 09:09:06 UMAP embedding parameters a = 0.9922 b = 1.112
## 09:09:06 Read 2259 rows and found 10 numeric columns
## 09:09:06 Using Annoy for neighbor search, n_neighbors = 30
## 09:09:06 Building Annoy index with metric = cosine, n_trees = 50
## 0%   10   20   30   40   50   60   70   80   90   100%
## [----|----|----|----|----|----|----|----|----|----|
## **************************************************|
## 09:09:06 Writing NN index file to temp file /var/folders/yf/3g1jwnkj1814bn89hknd60tjsrdw67/T//Rtmp9rl
## 09:09:06 Searching Annoy index using 1 thread, search_k = 3000
## 09:09:06 Annoy recall = 100%
## 09:09:07 Commencing smooth kNN distance calibration using 1 thread
## 09:09:08 Initializing from normalized Laplacian + noise
## 09:09:08 Commencing optimization for 500 epochs, with 88254 positive edges
## 09:09:12 Optimization finished
```

```
DimPlot(pbmc3k, reduction = "umap", pt.size = 0.1, shuffle = TRUE) + coord_fixed()
```



Discussion

- How much information can we get from the UMAP? And when should we be careful?
- How can we evaluate if the UMAP is doing a good job?

Fun with UMAPs: UMAP zoo: https://duhaime.s3.amazonaws.com/apps/umap-zoo/index.html

Celltype Annotation

Method 1: Using Marker Genes

```
cluster1.markers <- FindMarkers(pbmc3k, ident.1 = 1, min.pct = 0.25)
```

```
## For a more efficient implementation of the Wilcoxon Rank Sum Test,
## (default method for FindMarkers) please install the limma package
## --------------------------------------------
## install.packages('BiocManager')
## BiocManager::install('limma')
## --------------------------------------------
## After installation of limma, Seurat will automatically use the more
## efficient implementation (no further action necessary).
## This message will be shown once per session
```

```
head(cluster1.markers, n = 5)
```

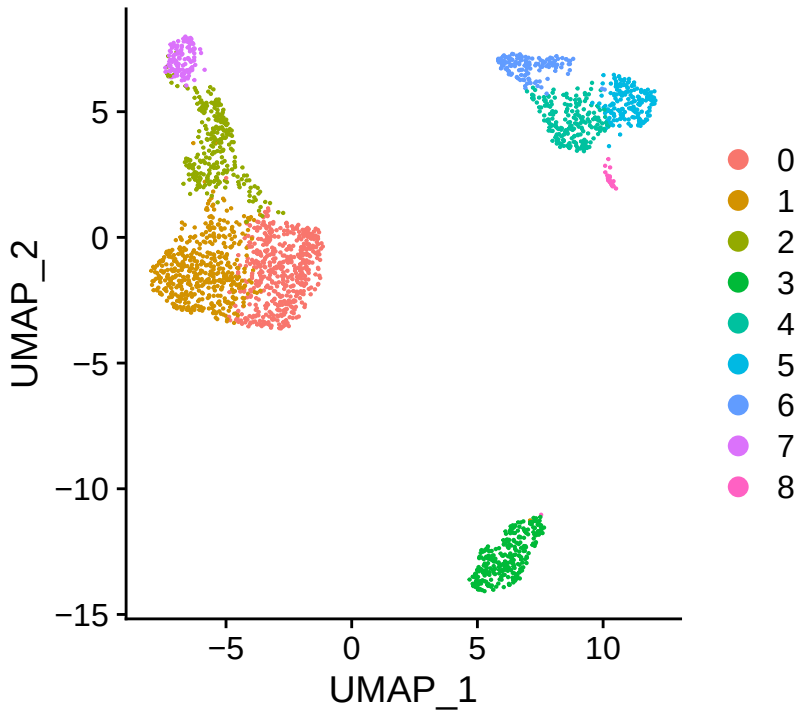
```
##             p_val avg_log2FC pct.1 pct.2    p_val_adj
## LTB  6.526961e-89  1.1648795 0.976 0.601 7.957671e-85
## IL7R 1.507776e-84  0.8808368 0.769 0.312 1.838280e-80
## IL32 2.993918e-84  0.8732298 0.935 0.450 3.650185e-80
## LDHB 7.361523e-70  0.7249483 0.953 0.594 8.975169e-66
## CD3D 3.475667e-63  0.6177126 0.905 0.425 4.237534e-59
```

Luckily, in this dataset we know the cell types very well and have known marker genes. From the vignette:

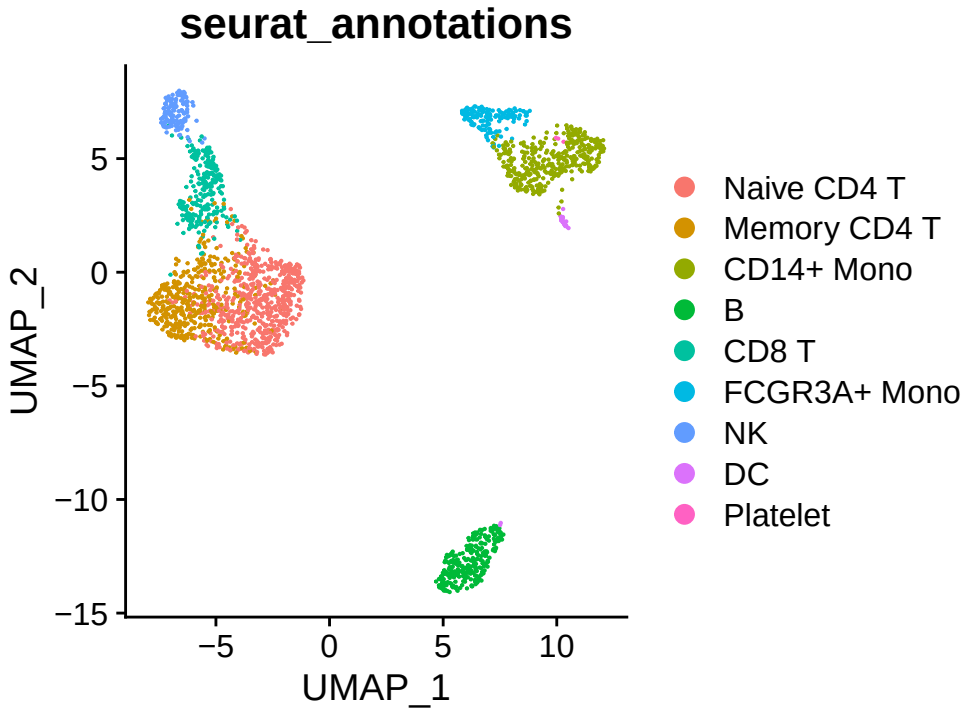
Markers	Cell Type
IL7R, CCR7	Naive CD4+ T
IL7R, S100A4	Memory CD4+
CD14, LYZ	CD14+ Mono
MS4A1	B
CD8A	CD8+ T
FCGR3A, MS4A7 FCGR3A+	Mono
GNLY, NKG7	NK
FCER1A, CST3	DC
PPBP	Platelet

Based on this, Seurat has already annotated the cells. Let's visualise. Do they agree with our clusters? Why not?

```
DimPlot(pbmc3k, reduction = "umap", pt.size = 0.1, shuffle = TRUE) + coord_fixed()
```




```
DimPlot(pbmc3k, reduction = "umap", group.by = "seurat_annotations", pt.size = 0.1, shuffle = TRUE) + c
```



Differential Expression Testing

Clusters are just groups of cells which have roughly similar gene expression profiles to each other. Clusters can represent:

- Cell type, or subtype (we often want this)
- Cell cycle phase

- Doublets (remove)
- Stripped nuclei/dead cells (remove)
- Batch/sample (can "correct")
- A combination of the above

To interpret the meaning of these clusters, we must put our detective hats on and find out what genes/gene patterns define them. There are several metrics for which genes best characterize each cluster. Here, we discuss two different approaches:

A: Difference in average expression

A popular approach is to compare differences in average expression for each gene in one cluster vs all other clusters. The significance of these differences can be assessed, for example, with the Wilcoxon Rank Sum test. However, the large number of cells often leads to exaggerated p-values and fold-change is heavily influenced by outliers, or genes which may only characterize a small portion of a cluster.

```
# Wilcox test for differentially expressed genes. May take 20-30s.
FCmarkers <- FindAllMarkers(pbmc3k, logfc.threshold = 0.25, test.use = "wilcox", 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
```

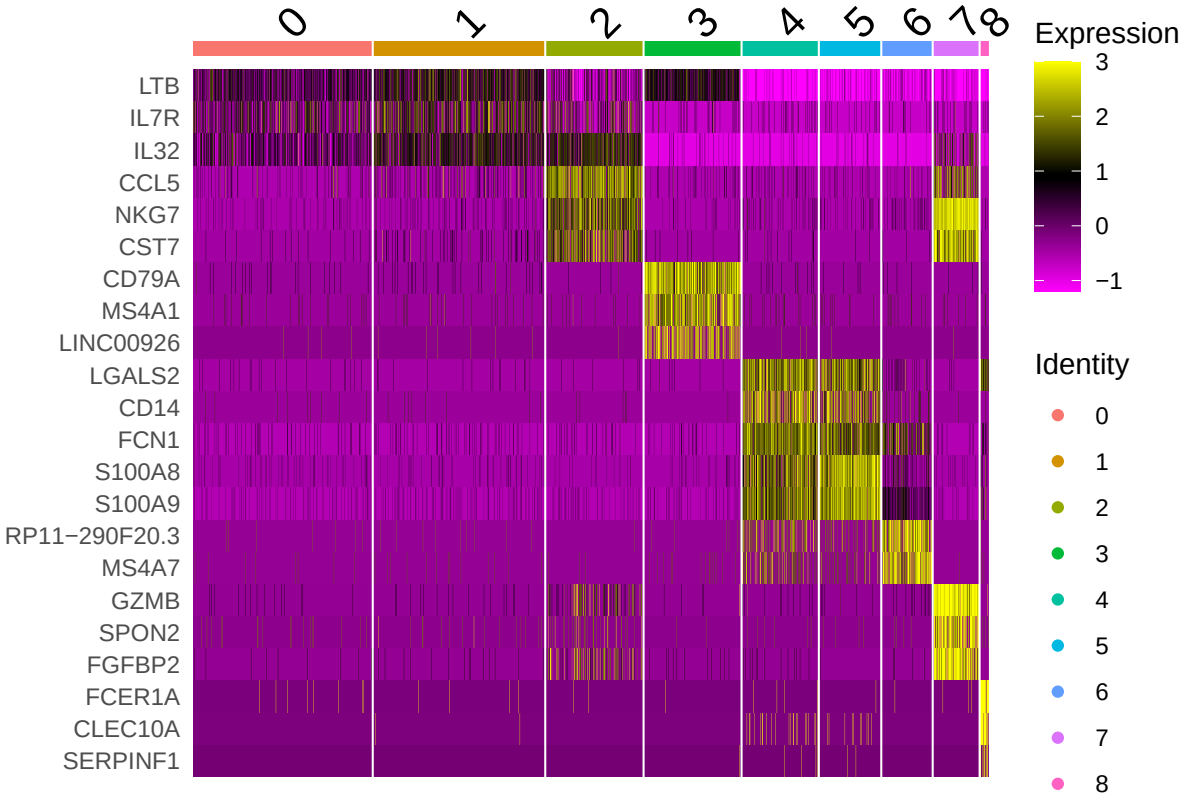
```
FCmarkers %>% group_by(cluster) %>% top_n(n = 10, wt = -p_val_adj)
```

```
## # A tibble: 90 x 7
## # Groups:   cluster [9]
##        p_val avg_log2FC pct.1 pct.2 p_val_adj cluster gene
##        <dbl>      <dbl> <dbl> <dbl>     <dbl> <fct>   <chr>
##  1 5.74e-121      0.618 1     0.999 7.00e-117 0       RPL32
##  2 1.21e-118      0.694 1     0.998 1.48e-114 0       RPS27
##  3 3.81e-117      0.670 1     1     4.64e-113 0       RPS12
##  4 7.25e-112      0.661 1     1     8.84e-108 0       RPS6
##  5 6.67e-107     -1.78  0.647 0.883 8.13e-103 0       S100A4
##  6 3.68e-106      0.584 1     0.999 4.49e-102 0       RPS14
##  7 4.19e-105     -1.29  0.641 0.913 5.11e-101 0       CYBA
##  8 2.25e-102      0.686 1     0.982 2.75e- 98 0       RPL31
##  9 1.85e- 97      0.665 1     0.987 2.26e- 93 0       RPS25
## 10 4.09e- 96      0.552 1     1     4.98e- 92 0       RPL13
## # ... with 80 more rows
```

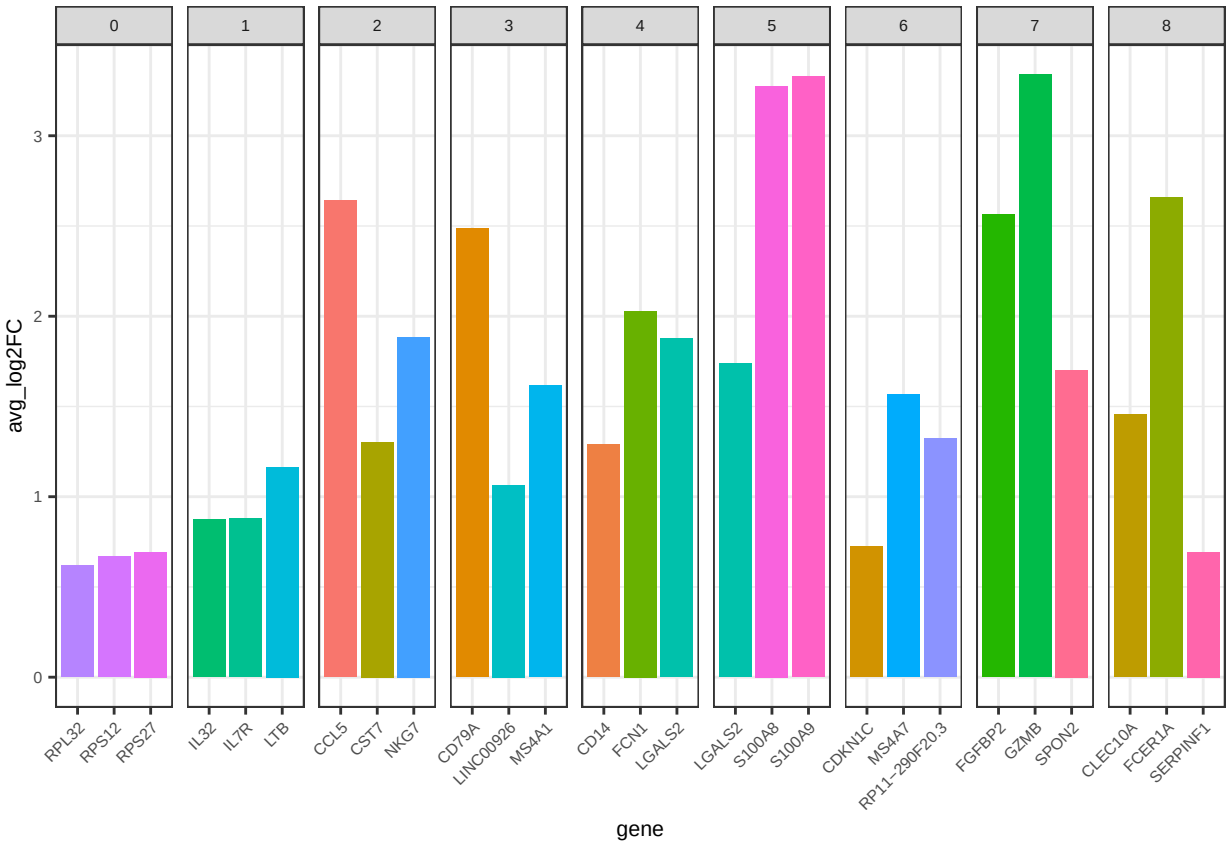
Note that both positively and negatively differentially expressed genes are identified. avg_log2FC shows the fold-change in mean expression, whereas p_val_adj indicates the significance.

```
# Marker heatmap
topFCmarkers <- FCmarkers %>% group_by(cluster) %>% top_n(n = 3, wt = -p_val_adj)
DoHeatmap(pbmc3k, features = topFCmarkers$gene, slot = 'scale.data', disp.max = 3)
```

```
## Warning in DoHeatmap(pbmc3k, features = topFCmarkers$gene, slot =
## "scale.data", : The following features were omitted as they were not found in
## the scale.data slot for the SCT assay: CDKN1C, RPS12, RPS27, RPL32
```




```
# Log fold change of main cluster markers
ggplot(topFCmarkers, aes(x = gene, y = avg_log2FC, fill = gene)) +
  geom_col() +
  theme_bw() +
    facet_grid(. ~ cluster, scales = "free") +
  RotatedAxis() +
  NoLegend() +
  theme(text = element_text(size=8))
```



B: ROC analysis

Aside from testing differences in average expression, which is prone to identifying markers which may not represent the cluster's full population, another approach to identifying cluster markers is to ask which genes can be used to best classify cells in each cluster from all remaining cells. For this, we borrow a simple measurement from supervised learning: the area under the ROC curve (TPR vs FPR).

```
# Test area under the ROC curve for markers for each cluster. May take 20-30s.
ROCmarkers <- FindAllMarkers(pbmc3k, logfc.threshold = 0.25, test.use = "roc", 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
```

```
ROCmarkers %>% group_by(cluster) %>% top_n(n = 10, wt = myAUC)
```

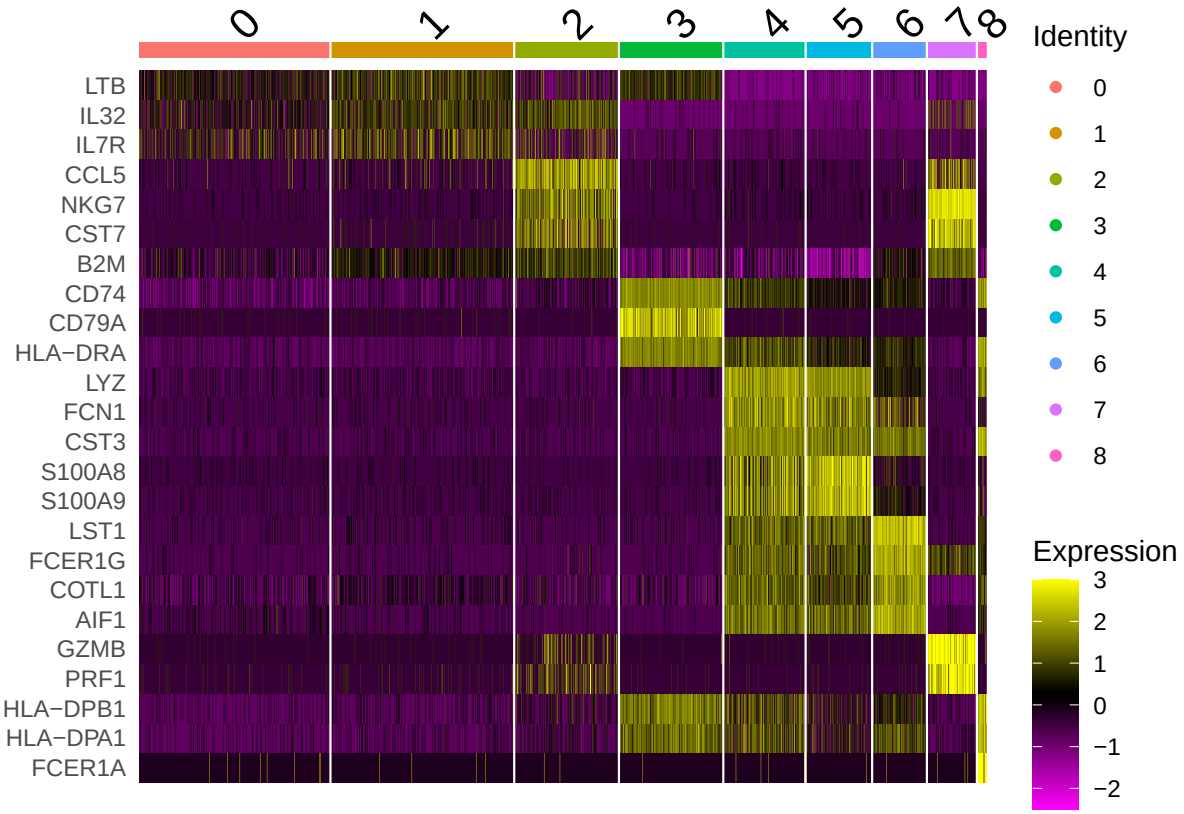
```
## # A tibble: 92 x 8
## # Groups:   cluster [9]
##    myAUC avg_diff power avg_log2FC pct.1 pct.2 cluster gene
```

```
##    <dbl>    <dbl> <dbl>      <dbl> <dbl> <dbl> <fct>   <chr>
##  1 0.838    0.428 0.676      0.618     1 0.999 0       RPL32
##  2 0.834    0.481 0.668      0.694     1 0.998 0       RPS27
##  3 0.832    0.464 0.664      0.670     1 1     0       RPS12
##  4 0.825    0.458 0.650      0.661     1 1     0       RPS6
##  5 0.816    0.405 0.632      0.584     1 0.999 0       RPS14
##  6 0.81     0.475 0.62       0.686     1 0.982 0       RPL31
##  7 0.802    0.461 0.604      0.665     1 0.987 0       RPS25
##  8 0.801    0.383 0.602      0.552     1 1     0       RPL13
##  9 0.8      0.527 0.6        0.760     1 0.98  0       RPS3A
## 10 0.793    0.407 0.586      0.587     1 0.999 0       RPL3
## # ... with 82 more rows
```

An AUC of 1 represents a perfect positive classifier (positive only on all cells in the cluster), while an AUC of 0 represents a perfect negative classifier (negative/absent only on all cells in the cluster). An AUC of 0.5 means the gene has no classification power.

```
# Marker heatmap
topROCmarkers <- ROCmarkers %>% group_by(cluster) %>% top_n(n = 3, wt = myAUC)
DoHeatmap(pbmc3k, features = topROCmarkers$gene, slot = 'scale.data', disp.max = 3)
```

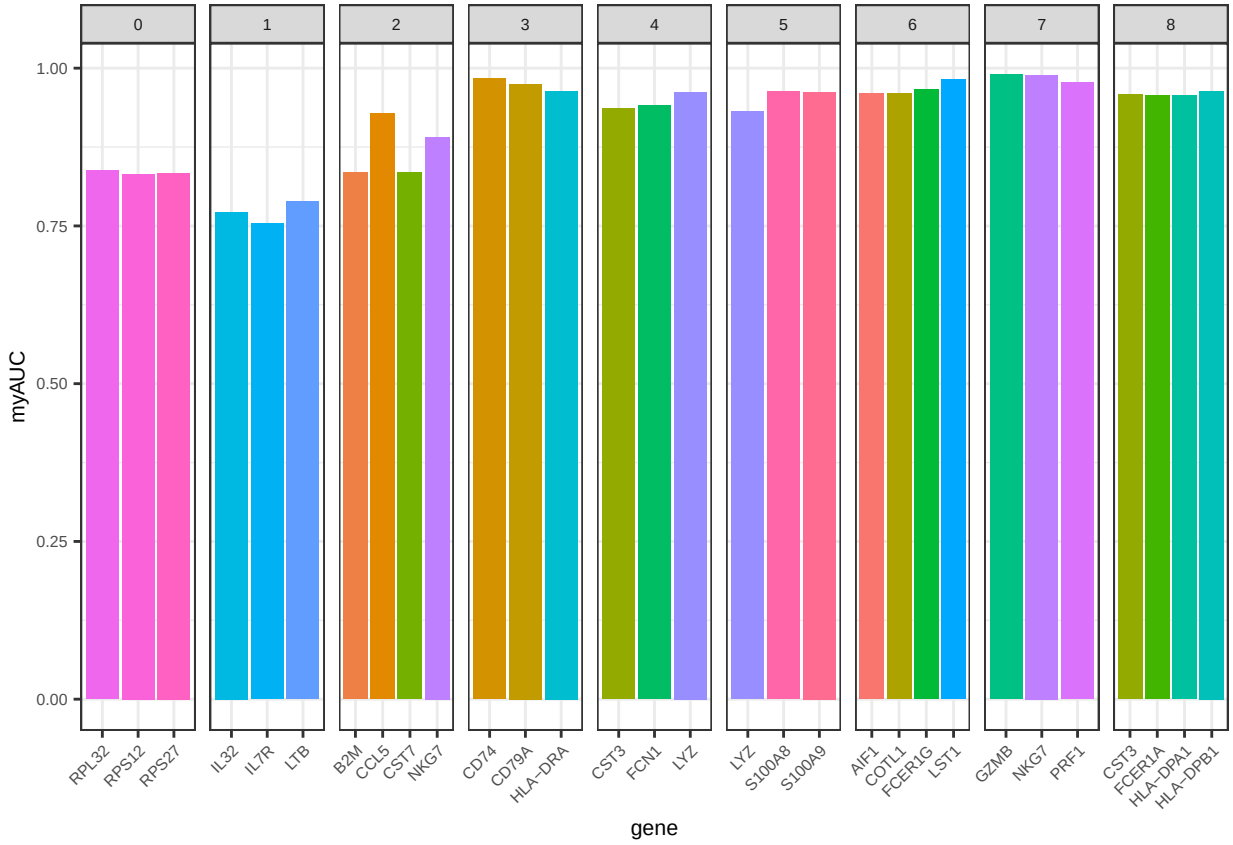
```
## Warning in DoHeatmap(pbmc3k, features = topROCmarkers$gene, slot =
## "scale.data", : The following features were omitted as they were not found in
## the scale.data slot for the SCT assay: RPS12, RPS27, RPL32
```



Note minor differences in the markers identified by ROC vs Wilcox

```
# Area under the ROC curve of main cluster markers
ggplot(topROCmarkers, aes(x = gene, y = myAUC, fill = gene)) +
  geom_col() +
```

```
  theme_bw() +
    facet_grid(. ~ cluster, scales = "free") +
  RotatedAxis() +
  NoLegend() +
  theme(text = element_text(size=8))
```



Note that for each cluster, there are moderate-to-strong positive classifiers. See if you can use these to guess the identity of each cluster (no, not Colonel Mustard. . .)

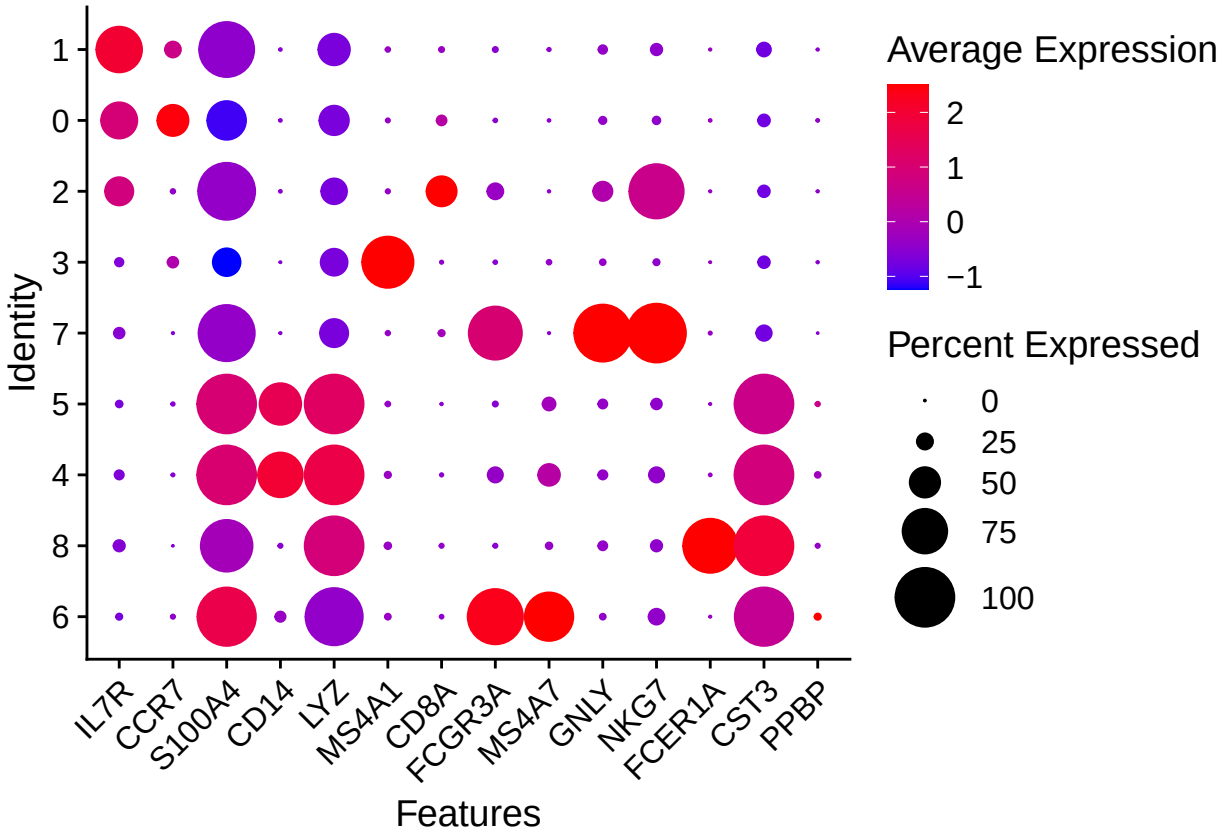
Which method you use to identify cluster markers also depends on your goals. . .

Exploring Known Celltype Markers

Since we already know the major PBMC cell types and their markers, we can directly use these to characterize our clusters.

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

```
markers.to.plot <- c("IL7R", "CCR7", "S100A4", "CD14", "LYZ", "MS4A1", "CD8A", "FCGR3A", "MS4A7", "GNLY
DotPlot(pbmc3k, features = markers.to.plot, cols = c("blue", "red"), dot.scale = 10, cluster.idents = T
```



Now try assigning the clusters to cell types...

Method 2: Using a Reference Atlas

Another method is to use previously annotated, similar datasets. Such high-quality, annotated datasets are known as "atlases". Big projects, perhaps most notably the human cell atlas, exist to create such atlases that can be used to annotate and compare future datasets. Usually, these atlases are healthy/unperturbed/*in vivo* samples, which you can map your diseased/perturbed/(in my case) organoid samples to.

Assigning celltypes using a reference atlas, requires you to integrate your data with the reference atlas. There are many ways to do this, and I recommend Luecken *et al.*, 2020 for a good overview and analysis. Different methods can give drastically different results, and even how you preprocess the data can affect results.

In short:

- Batch effects are technical variance in gene expression that exist between samples. These can be large or small:
  - Small: e.g. replicates processed by the same researcher on different days
  - Large: e.g. samples processed with different technologies in different species in different labs
  - ==> Depending on how similar your data are to the atlas, your batch effects could vary in size
- Data integration methods attempt to estimate the batch effect for each cell, and apply a correction vector to each
  - However, it's very hard to disentangle technical and biological variance in high dimensions

  - Some methods assume more, others less technical variance
  - Advice: Be critical: Some methods may not remove batch effects sufficiently, while others may remove too much

- Miscellaneous advice:
  - These methods always give you a celltype annotation! It's very hard to say if it's a good annotation or not. Look at marker genes to check (but this is a very hard question).
  - Just because your two samples don't overlap well on a UMAP after integration, doesn't mean that they mapped poorly (though it's certainly something to be on the lookout for)
  - Integrate the atlas first, and then map your samples to the atlas
  - When integrating the atlas, integrate within a timepoint, and then across timepoints
  - Integrate each of your samples with the atlas separately

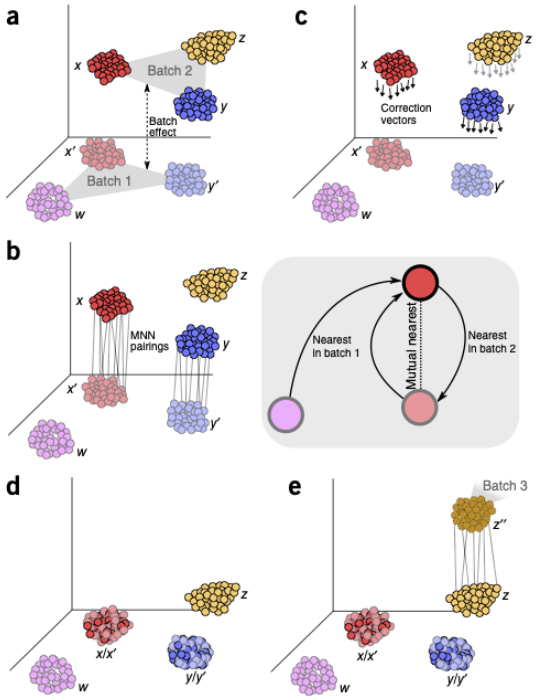


Figure 8: Schematic of how MNN (Haghverdi et al., 2018), a method for batch effect correction, works. This illustrates the underlying principle nicely.

Batch effect correction

Similar to reference mapping, if you are analysing separate samples, you may have batch effects between samples.

Do you have batch effects?

- Were your samples generated from the same pool of cells on the same day, just in separate 10X lanes? Then you probably don't have to batch effect correct (though some people still do)
- Otherwise: YES!

Apart from this, the same methods as above are used, and thus the same advice still applies:

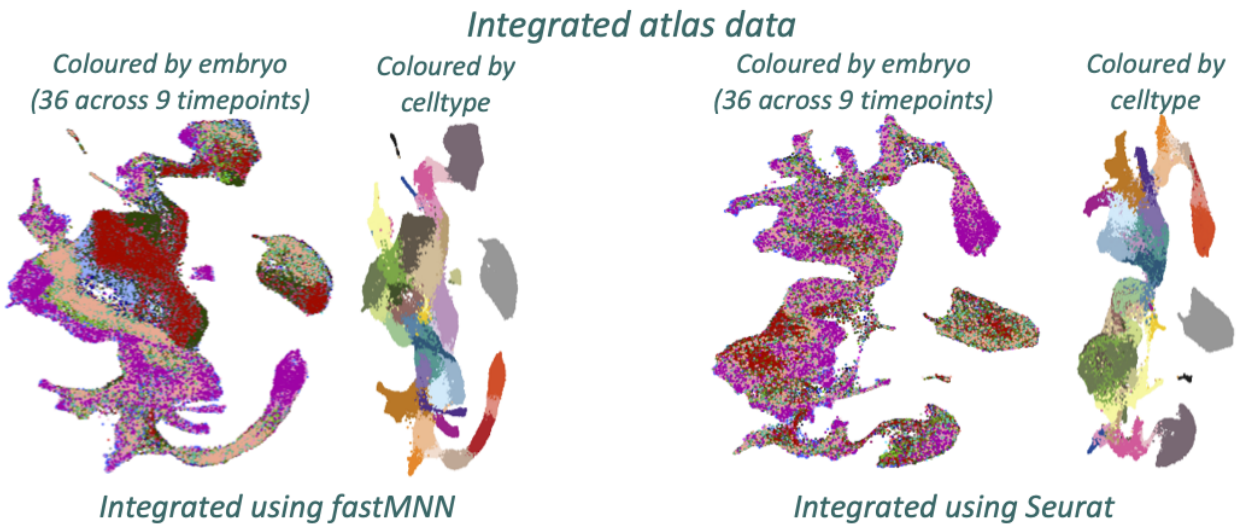


Figure 9: Integrating an atlas of mouse gastrulation (Pijuan-Sala et al., 2019) with two methods: fastMNN (left) and Seurat (right). Seurat removes more variance than MNN, merging batches across timepoints, and blurring celltypes (especially epiblast, primitive streak, and brain).

- Try several methods
- Check if meaningful biological variance is being removed ("over-integration")