Introduction to Single-cell RNA-seq Analysis - 2

by Jiajia Li

Research School of Biology

6 Aug 2025

Australian National University

Learning Objectives of Today

- Normalise and scale data
- Find highly variable features
- Principal component analysis
- Cell clustering

Normalise and Scale data

In single-cell analysis, particularly in the Seurat package, `NormalizeData` and `ScaleData` are two distinct but **sequential steps** that prepare your data for downstream analysis like PCA and clustering.

- `**NormalizeData**` makes gene expression levels comparable **between cells**.
- `**ScaleData**` makes gene expression levels comparable **between genes**.

`NormalizeData`: Correcting for library size

This step is to remove technical differences between cells, primarily the **sequencing depth** (or library size).

A cell that was sequenced more deeply (how?) will have higher counts for all its genes, **which doesn't mean it's biologically different.**

Two identical cells, but during the droplet formation:

- Cell A is captured perfectly: most of its mRNA is reverse-transcribed to cDNA.
- Cell B is poorly captured: much of its mRNA is lost.

- Cell A gets 20,000 reads.
- Cell B gets 8,000 reads.

The difference isn't biological, it's technical.

Normalisation

By default, `**NormalizeData()**` performs a "LogNormalize" transformation. This process involves three simple steps for each cell.

1. **Calculate relative counts**: The count for each gene is divided by the total number of counts in that cell.
2. **Scale the data**: The result is multiplied by a **scale factor** (the default is 10000). This creates values like "counts per 10000" or CP10K.
3. **Log-transform the data**: A natural log transformation is applied to the scaled data after adding a pseudo count of 1. This is often written as `**log1p**`.

$$\text{Normalized Value} = \ln\left(1 + \frac{\text{Gene Count}}{\text{Total Counts}} \times \text{Scale Factor}\right)$$

Normalisation

```
# normalise
merged <- NormalizeData(merged, assay = "RNA",
                        normalization.method = "LogNormalize",
                        scale.factor = 10000)
```

```
> merged
An object of class Seurat
18187 features across 23185 samples within 1 assay
Active assay: RNA (18187 features, 0 variable features)
 2 layers present: counts, data
```

Now we have 2 layers for our Seurat object, one is the previous count matrix, then the other one is called "**data**". Let's have a look.

The number has changed. Now they are normalised counts.

After normalisation, you can compare the expression of **Gene A in Cell 1** to the expression of **Gene A in Cell 2**.

Normalisation

```
> merged[['RNA']]$data
18187 x 23185 sparse Matrix of class "dgCMatrix"
  [[ suppressing 34 column names 'Rep1_ICBdT_AAACCTGAGCCAACAG-1', 'Rep1_ICBdT_AAACCTGAG
CCTTGAT-1', 'Rep1_ICBdT_AAACCTGAGTACCGGA-1' ... ]]
  [[ suppressing 34 column names 'Rep1_ICBdT_AAACCTGAGCCAACAG-1', 'Rep1_ICBdT_AAACCTGAG
CCTTGAT-1', 'Rep1_ICBdT_AAACCTGAGTACCGGA-1' ... ]]

Xkr4          .         .         .         .         .         .         .         .
Sox17         .         .         .         .         .         .         .         .
Mrpl15        .         .         .         .         .         .         1.323243  .
Lypla1        0.8990924 .         .         .         1.045510  .         1.873518  1.109667
Tcea1         0.6788359 1.165797  1.2108295 .         .         0.6474508 .         .
Rgs20         .         .         .         .         .         .         .         .
Atp6v1h       .         .         0.5796551 .         .         1.5352865 .         .
Rb1cc1        0.6788359 1.165797  .         .         1.326351  .         .         .
4732440D04Rik .         .         .         .         .         .         .         .
St18          .         .         .         .         .         .         .         .
```

The number has changed. Now they are normalised counts.

`ScaleData`: standardising gene expression

This step ensures that highly expressed genes don't dominate downstream analyses, which are often sensitive to the magnitude of values.

For each gene, `ScaleData` shifts its expression so that the mean expression across all cells is 0. Then, it scales the expression so that the variance across all cell is 1.

This is known as a **Z-score transformation.** It gives all genes **equal weight** in downstream analyses like PCA.

Without scaling, the genes with the highest variance (often the most highly expressed genes) would become the main principal components, potentially masking the contributions from genes with lower but still significant biological variance.

`ScaleData`: standardising gene expression

```
# scale data
merged <- ScaleData(merged, verbose = TRUE)
```

```
> merged
An object of class Seurat
18187 features across 23185 samples within 1 assay
Active assay: RNA (18187 features, 0 variable features)
 3 layers present: counts, data, scale.data
```

The scaled data is stored in the layer "**scale.data**".

Find variable features

This step is to identify genes that are highly variable across cells, which are more likely to be biologically meaningful and define the differences between cell types.

This process helps to reduce the dimensionality of the data and focus on the most informative signals, improving downstream analyses like clustering.

Why find variable features?

Not all genes are equally informative. Many genes, like housekeeping genes, have similar expression levels across all cells and contribute **little** to understanding **cellular heterogeneity**.

Conversely, genes that show high variability between cells are often those that drive biological differences, such as **cell type-specific markers**.

Find variable features

```
# find variable genes
merged <- FindVariableFeatures(merged, assay = "RNA",
                               selection.method = "vst",
                               nfeatures = 2000,
                               mean.cutoff = c(0.1, 8),
                               dispersion.cutoff = c(1,Inf))
```

- `**selection.method**`: Variance Stabilising Transformation which is the default setting.
- `**nfeatures**`: keep top 2000 variable genes, which is the default setting and a good place to start. This can be changed later.
- `**mean.cutoff**`: filter genes that are too highly/lowly expressed. A default setting.
- `**dispersion.cutoff**`: filter genes that have no variability.

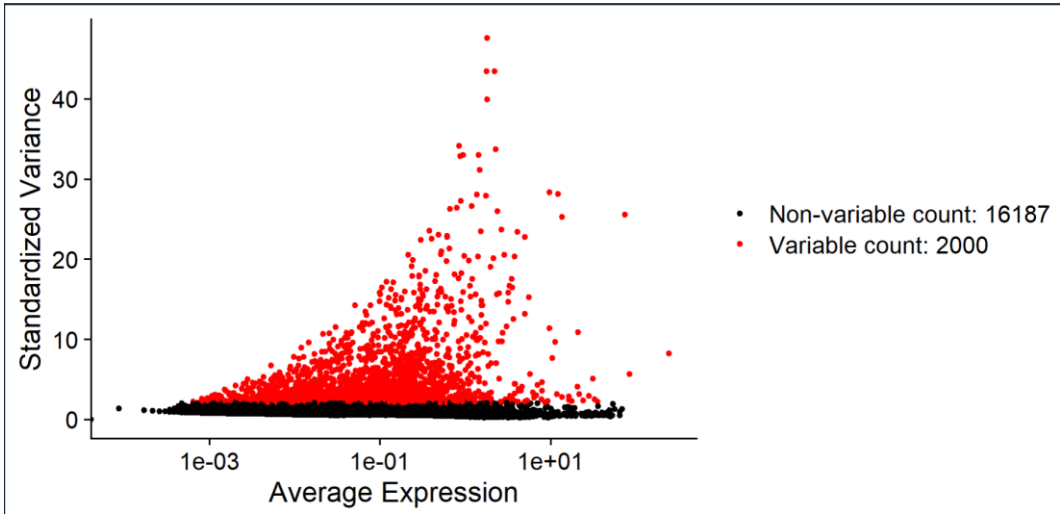
```
# top 10 highly variable genes
top10 <- head(VariableFeatures(merged), 10)
```

```
> top10
 [1] "Jchain" "Ighg1"  "Igkc"   "Sprr2d" "Mzb1"   "Slpi"   "Sprr2f" "Sprr2h" "Saa3"
[10] "Cxcl3"
```

Visualise highly variable genes

The `VariableFeaturePlot` function creates a scatter plot that helps to visualise the genes selected by the `FindVariableFeatures`.

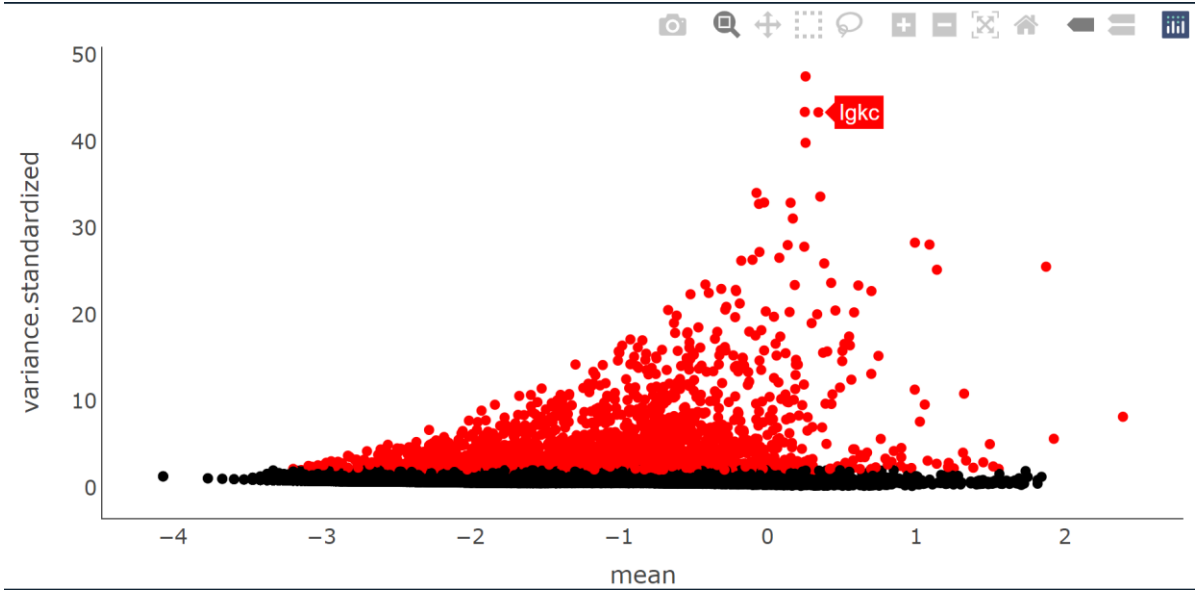
```
# visualise selected genes
VariableFeaturePlot(object=merged)
```



Visualise highly variable genes

Use `HoverLocator` to get quick information from the scatterplot by hovering over points.

```
# visualise selected genes
HoverLocator(VariableFeaturePlot(object=merged))
```



Principal Component Analysis (PCA)

Principal Component Analysis (PCA) is a powerful statistical technique used for **dimensionality reduction**, **data visualisation**, and **feature extraction**.

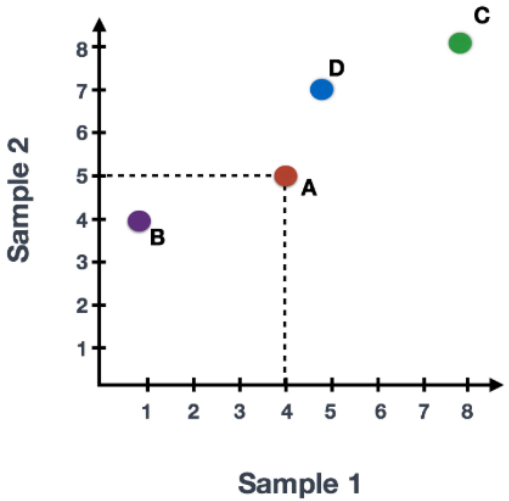
It transforms a dataset with possibly correlated features into a set of linearly uncorrelated features called **principal components**.

We will briefly go over PCA in this lesson (adapted from hbctraining and StatQuest's video).

Principal Component Analysis (PCA)

What are PCs?

For a two-dimensional data, where your data points can be plotted on a standard X-Y plane.

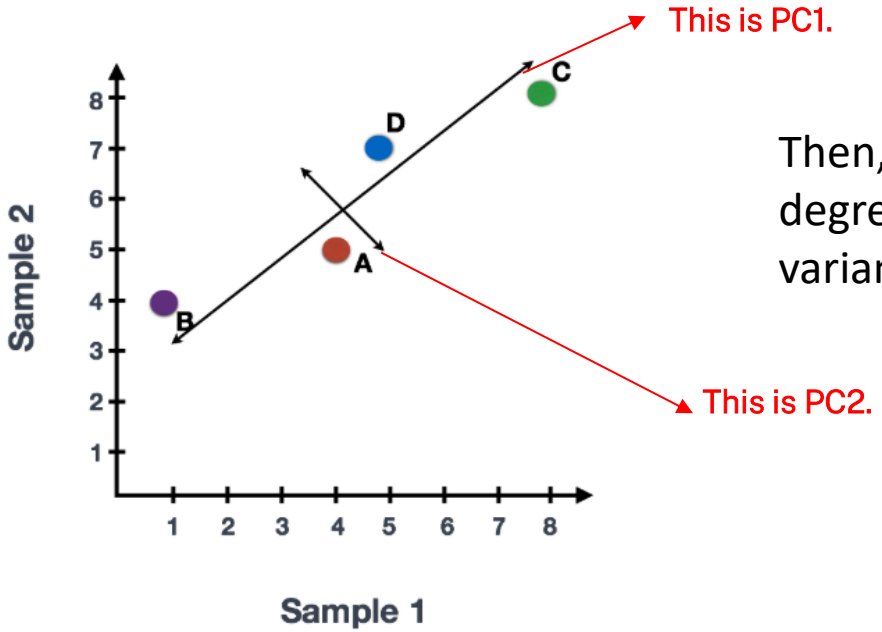


	Sample 1	Sample 2
Gene A	4	5
Gene B	1	4
Gene C	8	8
Gene D	5	7

Think of sample 1, sample 2 as cell1, cell2.

Principal Component Analysis (PCA)

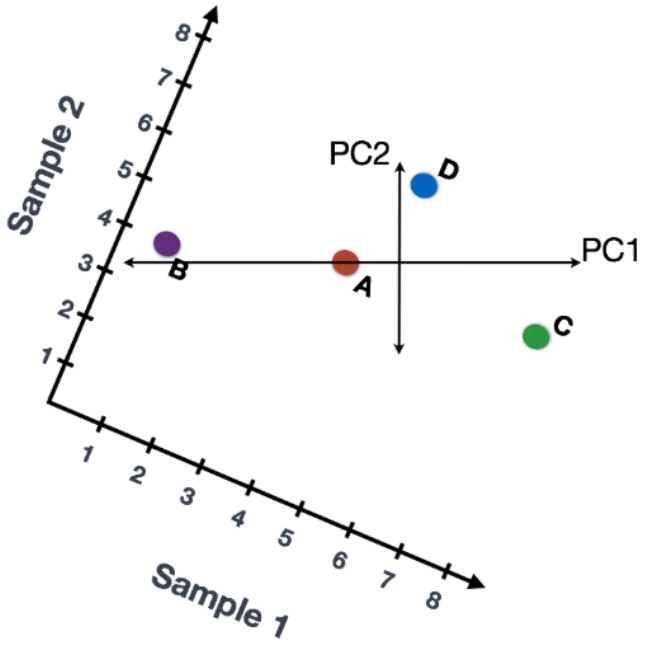
You could draw a line through the data points in the direction representing the most variance.



Then, if you draw another line **perpendicular** (90 degrees) to PC1, it describes the second-most variance, called PC2.

Principal Component Analysis (PCA)

After finding PC1 and PC2, we will use them as the new coordinates.



Then, we would have a new set of coordinates for our genes. We call it "**influence**".

Gene A's influence on PC2 **seems** to be zero.

	Sample 1	Sample 2	Influence on PC1	Influence on PC2
Gene A	4	5	-2	0.5
Gene B	1	4	-10	1
Gene C	8	8	8	-5
Gene D	5	7	1	6

Principal Component Analysis (PCA)

Once the influence has been determined, the score for each sample is calculated using the following equation.

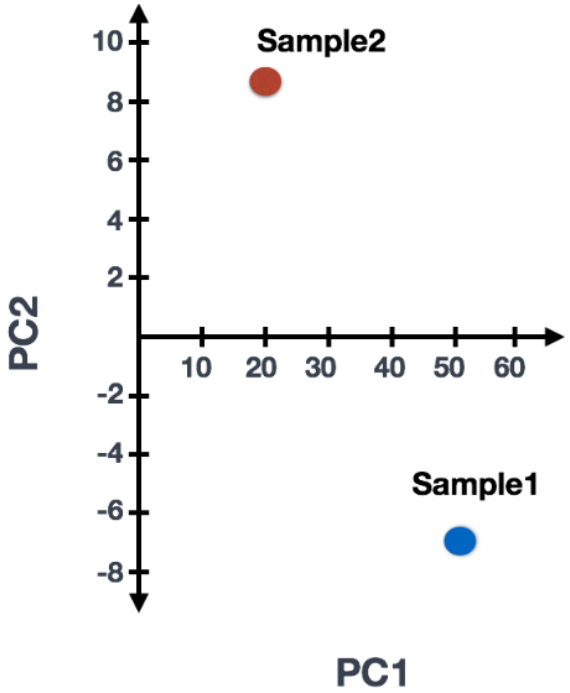
Sample1_PC1_Score = (GeneA_read_count * influence) + (GeneB_read_count * influence) + …
Sample1_PC2_Score = …

Sample2_PC1_Score = …
Sample2_PC2_Score = …

	PC1	PC2
Sample1	51	-7
Sample2	21	8.5

Principal Component Analysis (PCA)

Once we have the PC scores for all samples. We could plot them.



If we have more samples, we can add more dots to this plot.

The dots close to each other are **more similar** than the dots are further away from each other.

But if we have 3 samples, our data points would be living in 3-dimensional space, and we could have 3 PCs.

For single-cell data, each cell is a sample, and we could have **thousands of PCs**.

Calculating PCs

Not quite!

The number of PCs depend on the shape of our data. If you remember, we have selected the top 2000 highly variable genes.

The dimension of our data has down to **2000 genes x 23185 cells**.

The number of PCs we have would be 2000.

Although we could technically compute all possible principal components, **we don't**.

Calculating PCs

Most of the time, calculating the **first 50 PCs** is enough.

```
# Run PCA
merged <- RunPCA(merged, npcs = 50, assay = "RNA")
```

```
> merged
An object of class Seurat
18187 features across 23185 samples within 1 assay
Active assay: RNA (18187 features, 2000 variable features)
 3 layers present: counts, data, scale.data
 1 dimensional reduction calculated: pca
>
```

This adds PCA calculations to our Seurat object.

Calculating PCs

Why only the first 50 PCs?

The first PCs usually capture the **most variance** in the data, which tends to reflect **biologically meaningful variation**, such as:

- Cell type differences
- Cell cycle stages
- Stress responses

Later PCs often capture **technical noise** or very subtle variations that may not be biologically informative.

First 50 PCs

Balance between information and overfitting.

Including **too many PCs** can:

- Introduce noise
- Lead to overfitting in downstream

Using **too few PCs** might miss subtle but important signals.

Around **30-50 PCs** is often a good balance empirically found in many datasets.

Calculating PCs

To access the PCA table:

```
> merged[["pca"]]
A dimensional reduction object with key PC_
 Number of dimensions: 50
 Number of cells: 23185
 Projected dimensional reduction calculated:  FALSE
 Jackstraw run: FALSE
 Computed using assay: RNA
>
```

```
> Embeddings(merged, reduction = "pca")
                                      PC_1          PC_2          PC_3          PC_4
Rep1_ICBdT_AAACCTGAGCCAACAG-1   3.83517341 -8.115711e-01  9.892141e+00  2.259431e+00
Rep1_ICBdT_AAACCTGAGCCTTGAT-1   3.66390674 -2.208585e+00 -1.244708e+00  1.200532e+00
Rep1_ICBdT_AAACCTGAGTACCGGA-1 -65.63461428 -4.609580e+01 -9.346435e+00  7.999386e+00
Rep1_ICBdT_AAACCTGCACGGCCAT-1   4.13973608 -8.617120e-01 -3.006448e-01  4.993232e-01
Rep1_ICBdT_AAACCTGCACGGTAAG-1   3.90591611 -3.761172e-01  1.351412e+00  1.034904e+00
Rep1_ICBdT_AAACCTGCATGCCACG-1 -67.30210318 -4.567717e+01 -1.158631e+01  3.440012e+00
Rep1_ICBdT_AAACCTGGTCTTGTCC-1   4.31030709 -1.299729e+00  7.158373e-01  7.437302e-01
Rep1_ICBdT_AAACCTGGTTCCGTCT-1  -2.83886020  8.670165e-02 -1.378686e+00 -2.033373e+01
Rep1_ICBdT_AAACCTGTCTCATTCA-1 -67.28732906 -4.686328e+01 -7.824346e+00  6.042013e+00
Rep1_ICBdT_AAACGGGAGCAATATG-1   2.97871720  4.762028e-02  8.068692e-01  3.928423e-01
Rep1_ICBdT_AAACGGGAGCGCTCCA-1  -1.30507863  4.182009e-02 -9.466520e-01 -1.450336e+01
Rep1_ICBdT_AAACGGGCACTTAACG-1  -4.16141189  5.120773e-01 -2.053729e+00 -2.520738e+01
```

Elbow Plot

An elbow plot is a diagnostic tool used in PCA to help determine the **optimal number of PCs** to retain for analysis.

An elbow plot shows:

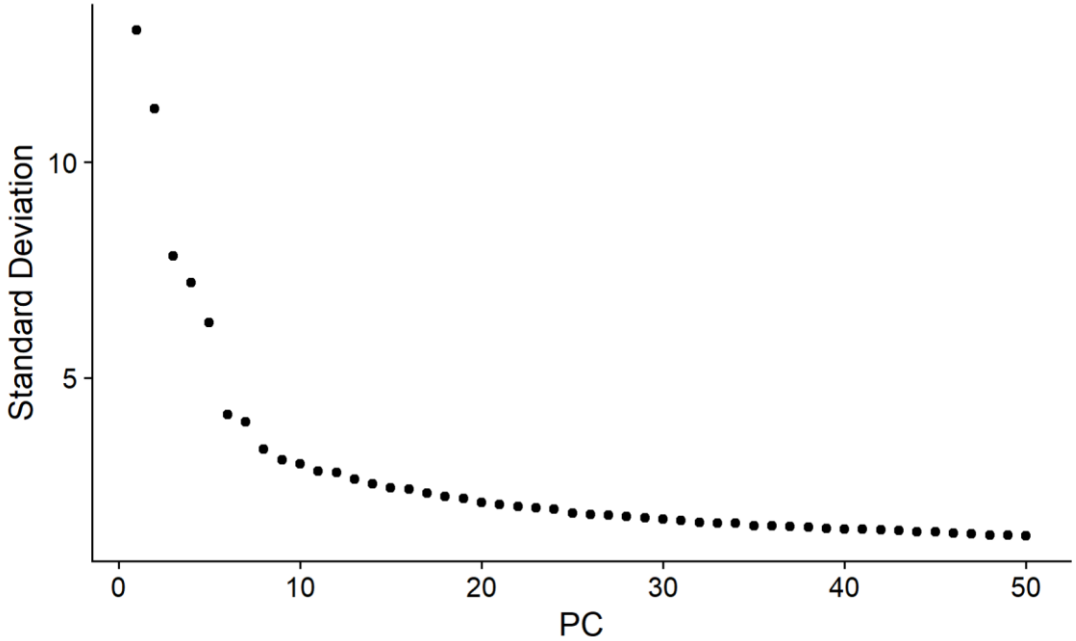
- **X-axis**: PC number (e.g., 1 to 50)
- **Y-axis**: Standard deviation (SD) of each PC (which correlates with the variance explained)
  - Higher SD -> more variation is captured by that PC
  - Lower SD -> less variation

The plot typically shows a steep drop followed by a plateau, like a bent arm - hence the name "elbow".

Elbow Plot

To create an elbow plot:

```
# create elbow plot
elbow <- ElbowPlot(merged, ndims = 50)
elbow
```



We can see that after **10 PCs**, the drop in SD **begins to flatten**.

Save the Elbow Plot

To save the plot to a file:

```
# create elbow plot
elbow <- ElbowPlot(merged, ndims = 50)
jpeg("figures/Elbow.jpg", width = 8, height = 6, units = 'in', res = 150)
print(elbow)
dev.off()
```

Dimension Heatmaps

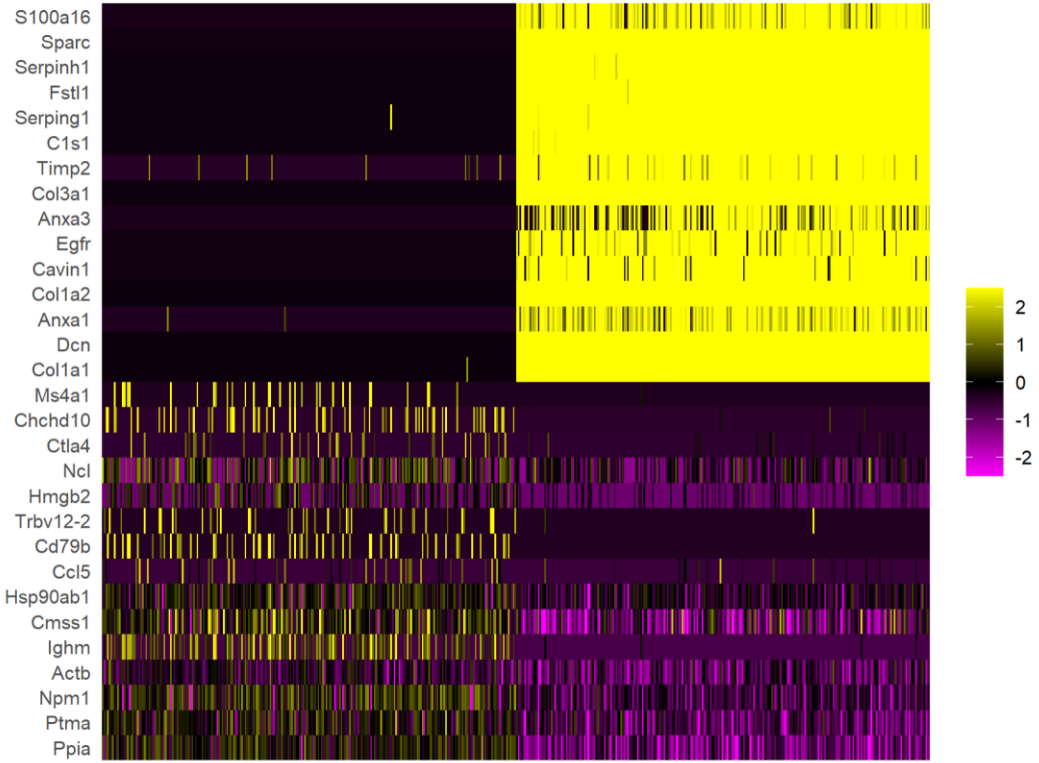
Dimension heatmaps are useful for interpreting the **biological meaning** of individual principal components (PCs).

To create a dimension heatmap for PC1:

```
# dimension heatmap for PC1
DimHeatmap(merged, dims = 1, cells = 500, nfeatures = 30,
           balanced = TRUE, fast = FALSE)
```

- `**dims = 1**`: plot PC1.
- `**cells = 500**`: plot top 500 cells.
- `**nfeatures = 30**`: plot top 30 genes.
- `**balanced = **`: plot an equal number of genes/cells with both + and - scores.
- `**fast = **`: use ggplot2 to generate figure, with a legend.

Dimension Heatmaps - PC1

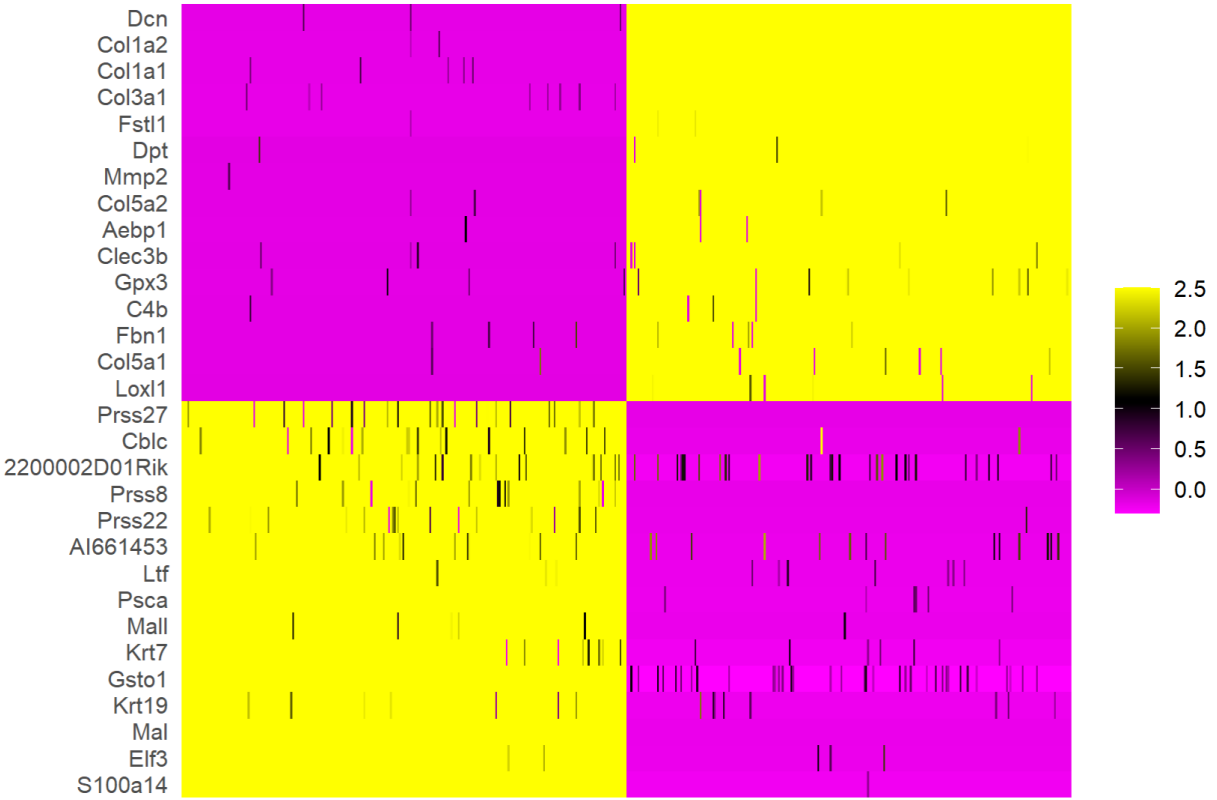


Each column is a cell.
Each row is a gene.

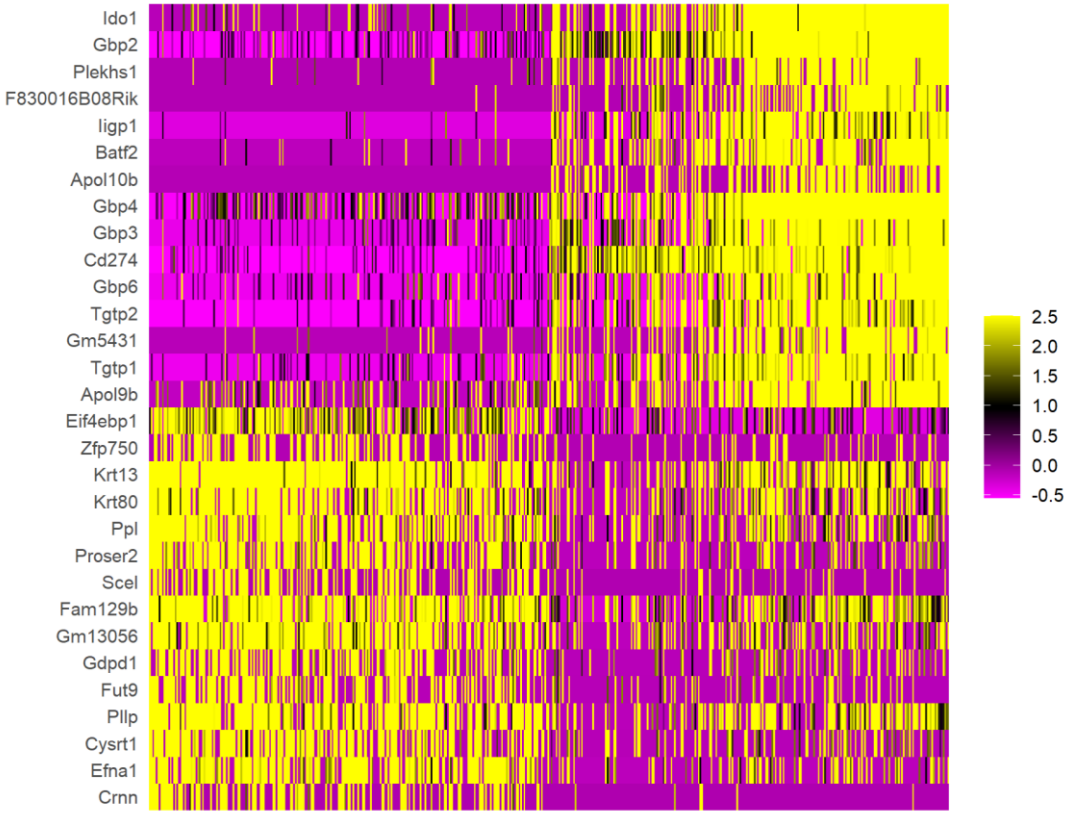
When inspecting PC heatmaps, we want to see **boundaries** or **patterns**.

When the **pattern starts to fade**, the PCs afterwards capture noise rather than biological variation.

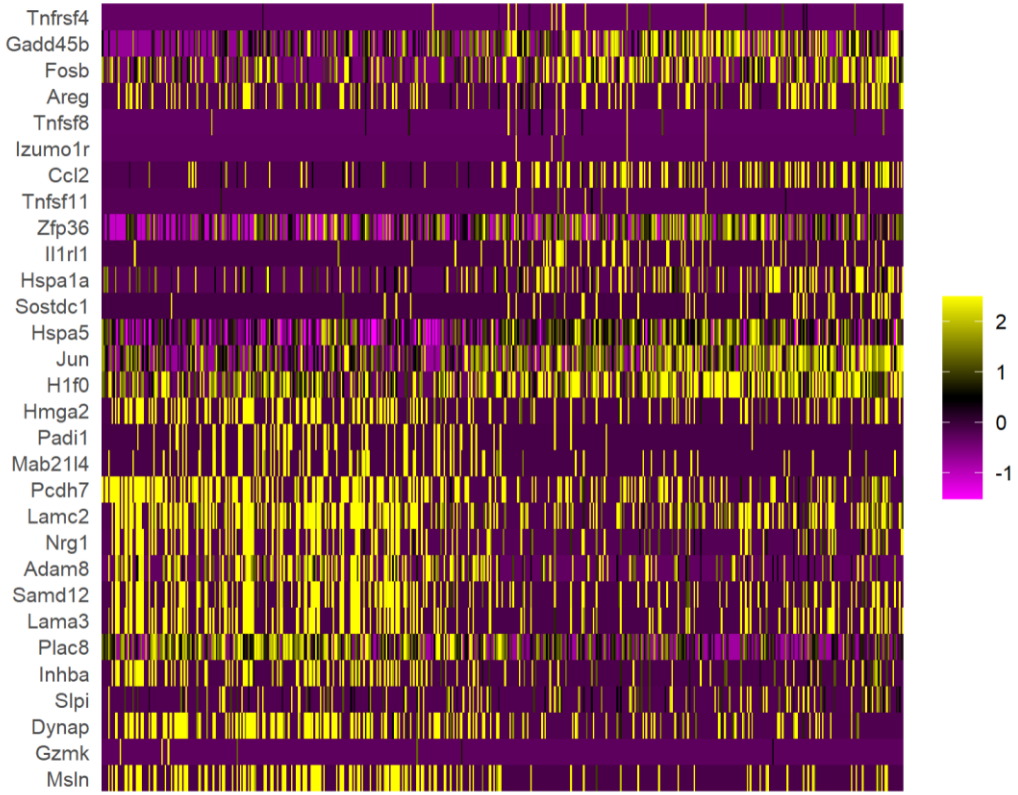
Dimension Heatmaps - PC2



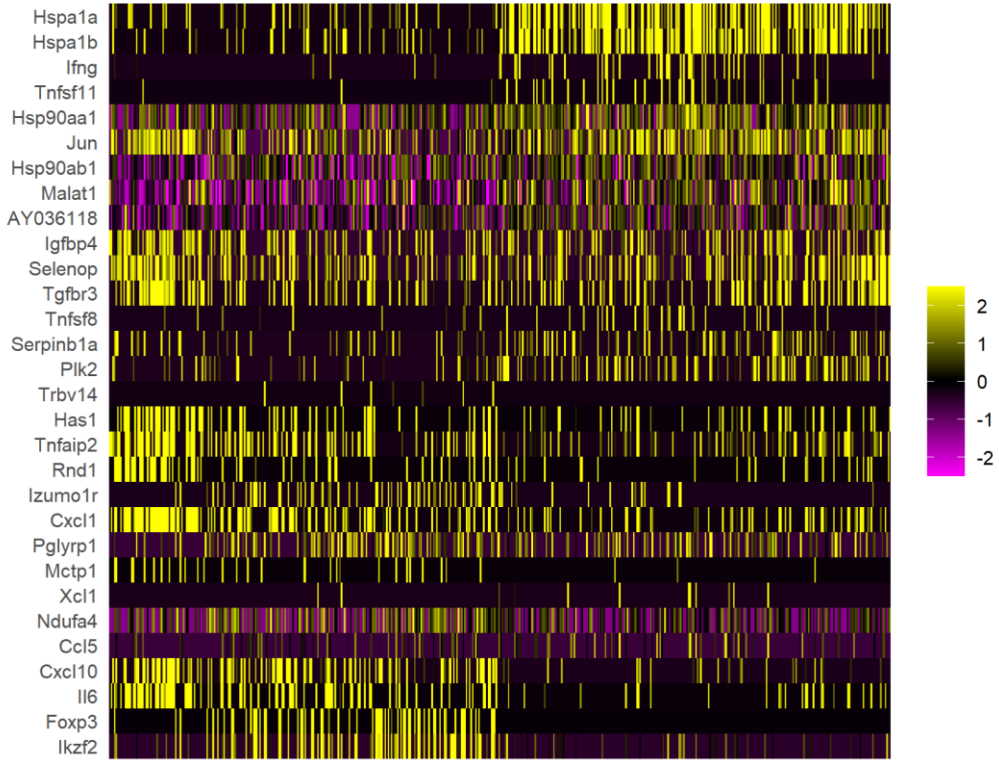
Dimension Heatmaps - PC10



Dimension Heatmaps - PC25



Dimension Heatmaps - PC50



Dimension Heatmaps

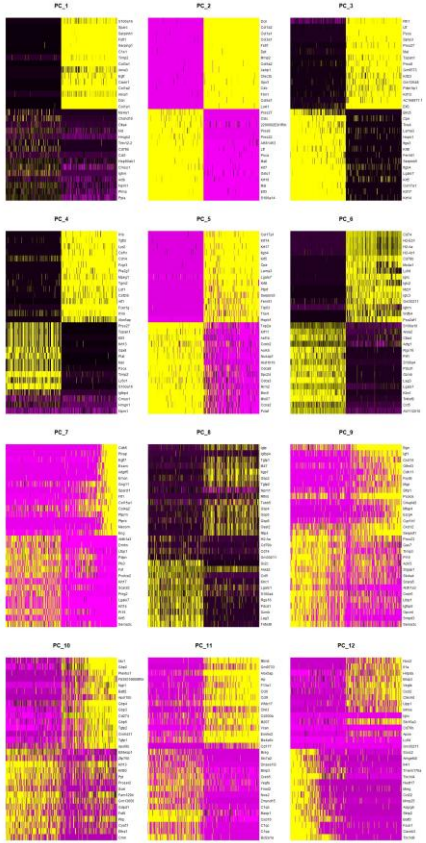
We can also plot multiple heatmaps together for easier comparison.

Let's plot heatmaps of PC1 to PC12, and save it as a figure:

```
jpeg("figures/DimHm1_12.jpg", width = 10, height = 20, units = 'in', res = 150)
DimHeatmap(merged, dims = 1:12, balanced = TRUE, cells = 500)
dev.off()
```

Dimension Heatmaps

Result:

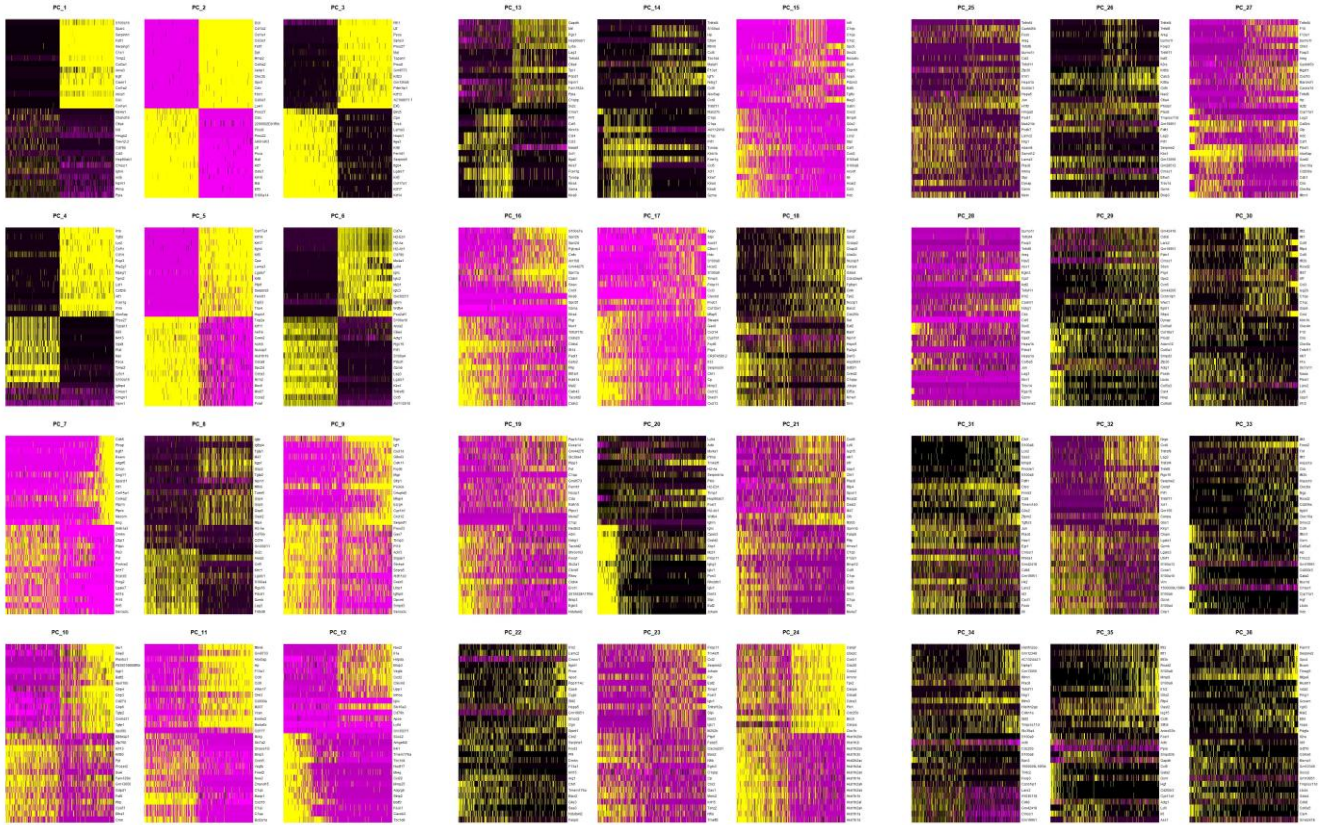


Please do the same for:

- **PC13 to PC24**
- **PC25 to PC36**

Dimension Heatmaps

Result:



Choosing the number of PCs to use

Using the elbow plot and heatmap, we can make an informed decision on how many PCs we should use for **cell clustering**.

There is one we can use to deduce our PC cutoff. We can keep PCs have a **standard deviation above 2**.

To find PCs have SD above 2:

```
> merged@reductions$pca@stdev > 2
 [1]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE
[15]  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE  TRUE FALSE FALSE FALSE FALSE FALSE FALSE
[29] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
[43] FALSE FALSE FALSE FALSE FALSE FALSE FALSE FALSE
```

Choosing the number of PCs to use

Count how many TRUEs we have:

```
> length(which(merged@reductions$pca@stdev > 2))
[1] 22
```

The first 22 PCs have a SD above 2.

So, we can choose `**PCs = 22**`.

This is **a great place to start**, and by **adding or removing PCs**, we can compare the differences in result.

When in doubt, slightly more PCs is better than not enough.

Cell Clustering - FindNeighbor()

Cell clustering is the process of grouping individual cells with similar gene expression profiles. Each cluster ideally corresponds to a specific **cell type**, **cell state**, or **functional group**.

The first step in clustering is the `**FindNeighbors()**` function, which computes the `**k.param**` nearest neighbours for a given dataset.

`**FindNeighbors()**` function construct a graph that captures the relationships (similarity) between cells based on their gene expression profiles - specifically, a **K-nearest neighbour (KNN)** graph.

```
PC <- 22
merged <- FindNeighbors(merged, dims = 1:PC)
```

Cell Clustering - FindClusters()

Then, we run `**FindClusters()**` function.

This groups cells into clusters by running a community detection algorithm on the shared nearest neighbour (SNN) graph created by `FindNeighbors()`.

```
merged <- FindClusters(merged, resolution = 1.2,
                       cluster.name = 'seurat_clusters_res1.2')
```

- `**resolution = 1.2**`: Value of the resolution parameter, use a value above/below 1.0 if you want to obtain a larger/smaller number of communities.
- `**cluster.name =**`: give this cluster a name.

Cell Clustering

```
> merged <- FindClusters(merged, resolution = 1.2,
+                        cluster.name = 'seurat_clusters_res1.2')
Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck

Number of nodes: 23185
Number of edges: 786013

Running Louvain algorithm...
0%   10   20   30   40   50   60   70   80   90   100%
[----|----|----|----|----|----|----|----|----|----|
**************************************************|
Maximum modularity in 10 random starts: 0.8501
Number of communities: 25
Elapsed time: 3 seconds
```

We have 25 cell clusters.

Cell Clustering - RunUMAP()

Then, we can run `**RunUMAP()**` to perform UMAP (Uniform Manifold Approximation and Projection) - a popular non-linear dimensionality reduction technique for visualising high-dimensional data in 2D or 3D.

This projects cells into a 2D space such that **similar cells are close together** and dissimilar ones are further apart.

```
merged <- RunUMAP(merged, dims = 1:PC)
```

```
> merged
An object of class Seurat
18187 features across 23185 samples within 1 assay
Active assay: RNA (18187 features, 2000 variable features)
 3 layers present: counts, data, scale.data
 2 dimensional reductions calculated: pca, umap
```

Plotting - DimPlot()

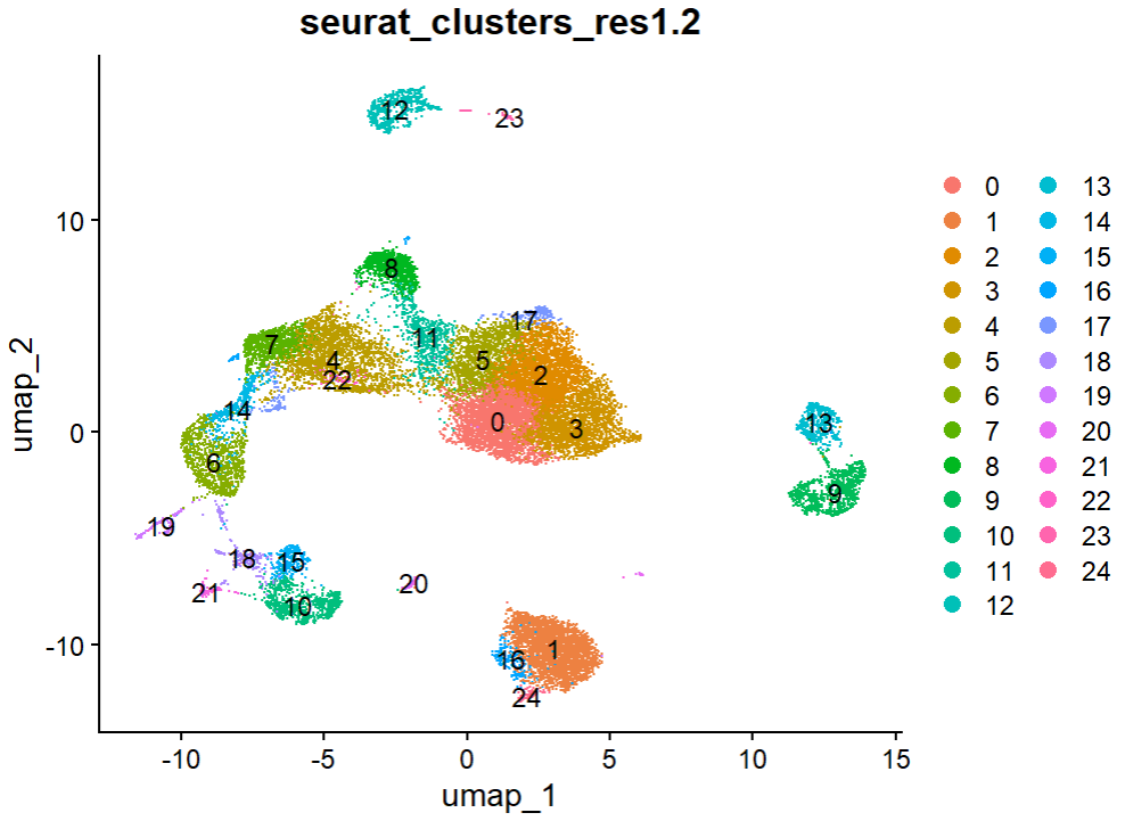
Then, we can use the `**DimPlot()**` function to visualise the clusters.

```
DimPlot(merged, label = TRUE, group.by = 'seurat_clusters_res1.2')
```

To save the plot:

```
jpeg("figures/UMAP.jpg", width = 5, height = 4, units = 'in', res = 150)
DimPlot(merged, label = TRUE, group.by = 'seurat_clusters_res1.2')
dev.off()
```

Plotting - DimPlot()



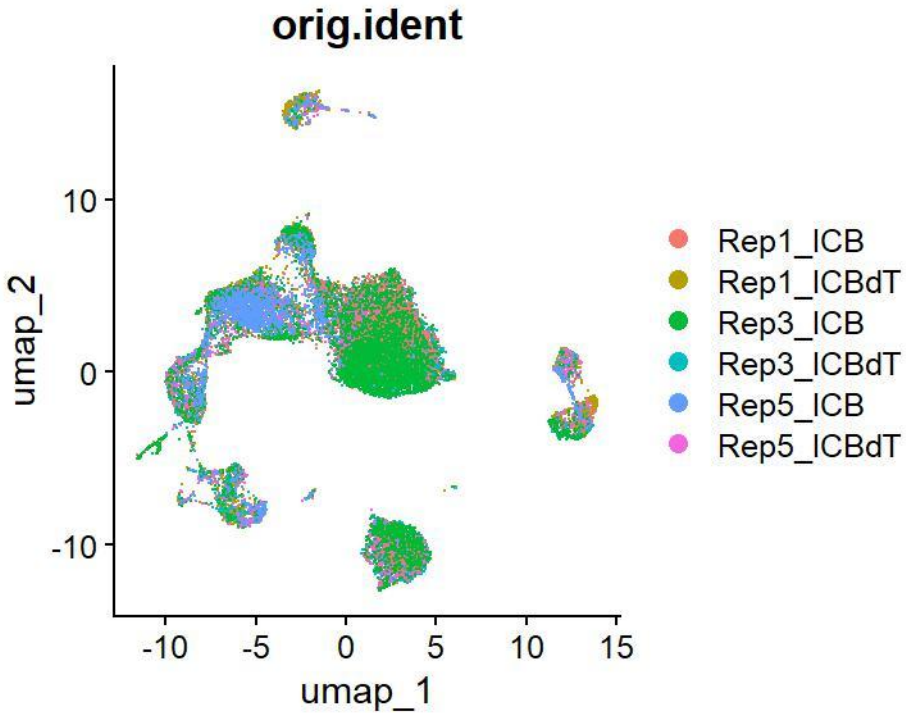
Colour cells by sample

Now, let's colour our cells by sample. This is to make sure that **no sample is clustered by itself** - an indicator of batch effects.

We want similar cells to be clustered together because they have similar gene expression. We do not want our cells to cluster together because of technical reasons (e.g., different sequencing batch, different replicate, etc).

```
jpeg("figures/UMAP_by_sample.jpg", width = 5, height = 4, units = 'in',
     res = 150)
DimPlot(merged, label = FALSE, group.by = "orig.ident")
dev.off()
```

Colour cells by sample

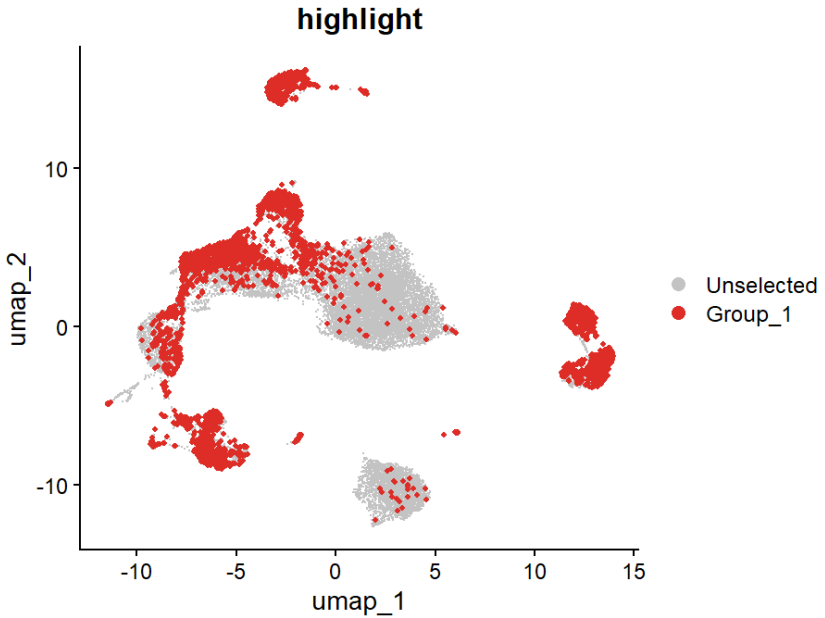


It looks like all samples mix well.

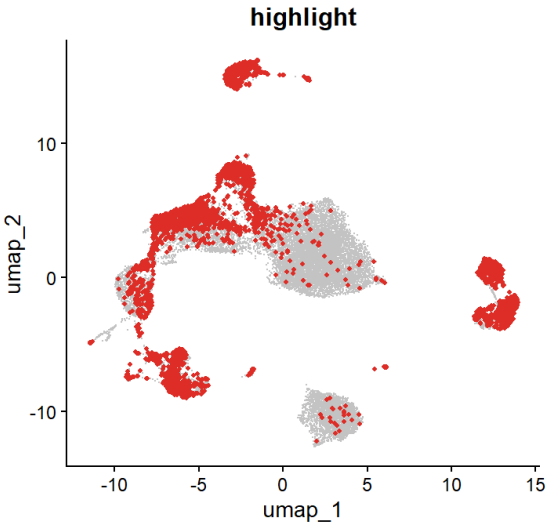
We can plot further with only highlighting one sample.

Highlight - Rep1_ICBdT

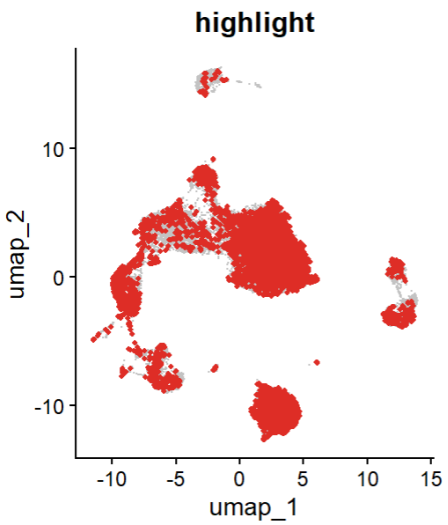
```
highlighted_cells <- WhichCells(merged, expression = orig.ident == "Rep1_ICBdT")

DimPlot(merged, reduction = 'umap', group.by = 'orig.ident',
        cells.highlight = highlighted_cells)
```



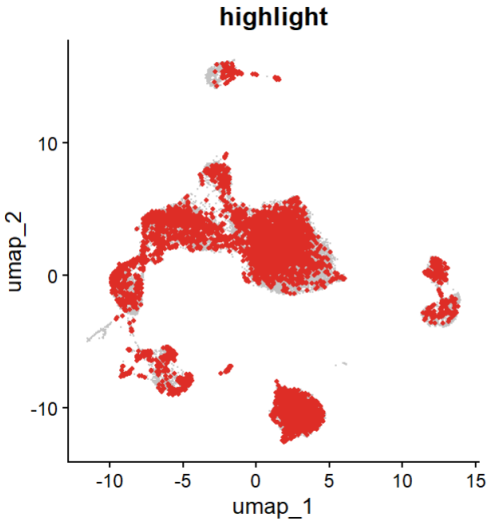
Highlight - Rep1,3,5_ICBdT



Rep1_ICBdT

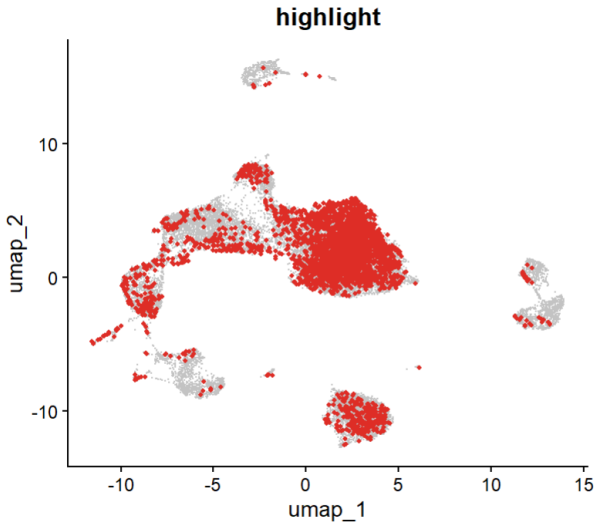


Rep3_ICBdT

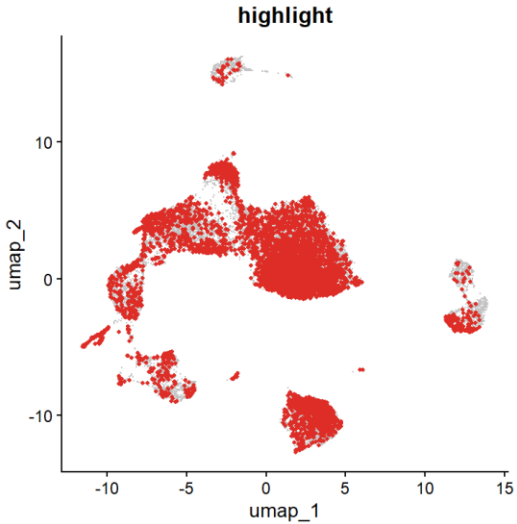


Rep5_ICBdT

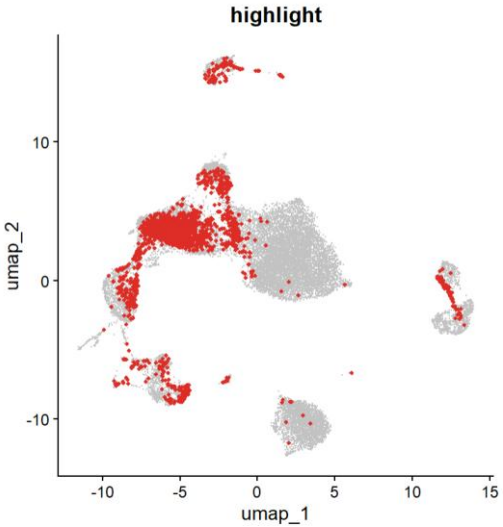
Highlight - Rep1,3,5_ICB



Rep1_ICB



Rep3_ICB



Rep5_ICB

Explore clustering resolution

When running FindClusters(), we can specify the resolution. Higher resolution calls more clusters.

Let's try a few resolutions and compare them.

```
merged <- FindClusters(merged, resolution = 0.8,
                       cluster.name = 'seurat_clusters_res0.8')
merged <- FindClusters(merged, resolution = 0.5,
                       cluster.name = 'seurat_clusters_res0.5')
```

Explore clustering resolution

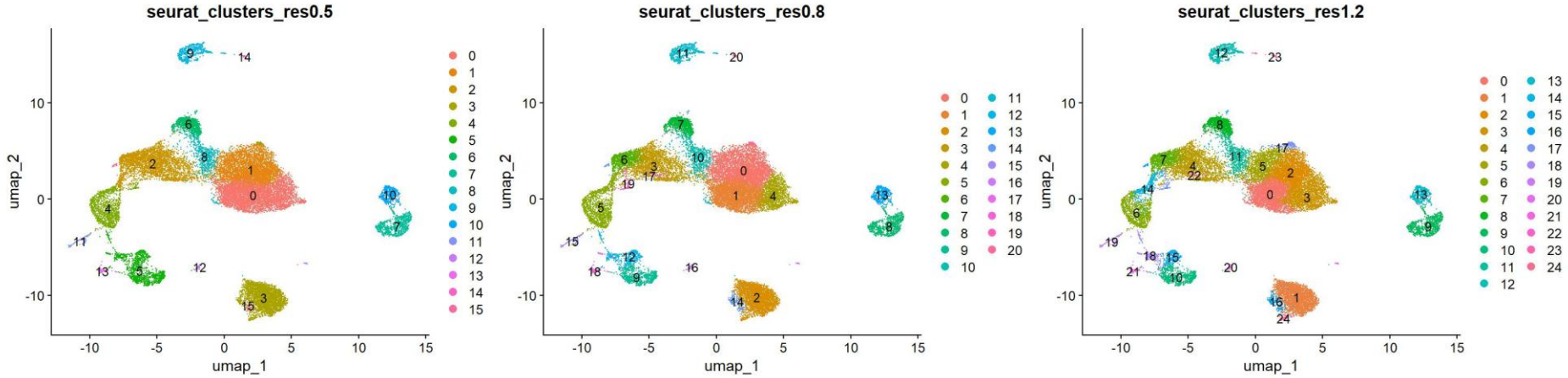
When running **FindClusters()**, we can specify the resolution. Higher resolution calls more clusters.

Let's try a few resolutions and compare them.

```
merged <- FindClusters(merged, resolution = 0.8,
                       cluster.name = 'seurat_clusters_res0.8')
merged <- FindClusters(merged, resolution = 0.5,
                       cluster.name = 'seurat_clusters_res0.5')
```

```
jpeg("figures/UMAP_compare_res.jpg", width = 20, height = 5, units = 'in', res = 150)
DimPlot(merged, label = TRUE, group.by = 'seurat_clusters_res0.5') +
  DimPlot(merged, label = TRUE, group.by = 'seurat_clusters_res0.8') +
  DimPlot(merged, label = TRUE, group.by = 'seurat_clusters_res1.2')
dev.off()
```

Explore clustering resolution



The shape of the UMAP **doesn't change** if you change the cluster resolution.

The shape of the UMAP is determined by **the number of PCs** used to create the UMAP.

Cell cycle scoring

Cell cycle scoring is an important step in single-cell RNA-seq analysis when you're interested in **identifying and accounting for the cell cycle phase** of each cell.

This is useful because cell cycle-related gene expression can **affect clustering**, making it look like there are different cell types when it's actually just a difference in cell cycle states.

It's the process of:

1. Measuring the expression of known **cell cycle-related genes** in each cell.
2. Scoring each cell based on how strongly it expresses genes associated with:
    - G1 phase
    - S phase (DNA synthesis)
    - G2/M phase (cell division)
3. Assigning a phase label (G1, S, or G2M) to each cell.

Cell cycle scoring

Seurat has a built-in list of cell cycle related genes called `cc.genes`.

```
> cc.genes
$s.genes
 [1] "MCM5"     "PCNA"     "TYMS"     "FEN1"     "MCM2"     "MCM4"     "RRM1"
 [8] "UNG"      "GINS2"    "MCM6"     "CDCA7"    "DTL"      "PRIM1"    "UHRF1"
[15] "MLF1IP"   "HELLS"    "RFC2"     "RPA2"     "NASP"     "RAD51AP1" "GMNN"
[22] "WDR76"    "SLBP"     "CCNE2"    "UBR7"     "POLD3"    "MSH2"     "ATAD2"
[29] "RAD51"    "RRM2"     "CDC45"    "CDC6"     "EXO1"     "TIPIN"    "DSCC1"
[36] "BLM"      "CASP8AP2" "USP1"     "CLSPN"    "POLA1"    "CHAF1B"   "BRIP1"
[43] "E2F8"

$g2m.genes
 [1] "HMGB2"    "CDK1"     "NUSAP1"   "UBE2C"    "BIRC5"    "TPX2"     "TOP2A"    "NDC80"
 [9] "CKS2"     "NUF2"     "CKS1B"    "MKI67"    "TMPO"     "CENPF"    "TACC3"    "FAM64A"
[17] "SMC4"     "CCNB2"    "CKAP2L"   "CKAP2"    "AURKB"    "BUB1"     "KIF11"    "ANP32E"
[25] "TUBB4B"   "GTSE1"    "KIF20B"   "HJURP"    "CDCA3"    "HN1"      "CDC20"    "TTK"
[33] "CDC25C"   "KIF2C"    "RANGAP1"  "NCAPD2"   "DLGAP5"   "CDCA2"    "CDCA8"    "ECT2"
```

Cell cycle scoring

`**cc.genes**` has two vectors:

- `**$s.genes**`: which is a vector of genes associated with S-phase.
- `**$g2m.genes**`: a vector of genes associated with G2M-phase.

But these genes are from human, we need to covert it to mouse genes. To do this, we can use the `**gorth()**` function from the `**gprofiler2**` package.

```
library(gprofiler2)

s.genes <- gorth(cc.genes.updated.2019$s.genes, source_organism = "hsapiens",
                 target_organism = "mmusculus")$ortholog_name
g2m.genes <- gorth(cc.genes.updated.2019$g2m.genes, source_organism = "hsapiens",
                   target_organism = "mmusculus")$ortholog_name
```

Cell cycle scoring

The `**cc.genes.updated.2019**` is also a list of cell cycle-related genes, but an updated version in 2019.

If you run `**cc.genes.updated.2019$s.genes**`, you still get a list of gene names that are associated with S-phase.

```
> cc.genes.updated.2019$s.genes
 [1] "MCM5"    "PCNA"    "TYMS"    "FEN1"    "MCM7"    "MCM4"    "RRM1"
 [8] "UNG"     "GINS2"   "MCM6"    "CDCA7"   "DTL"     "PRIM1"   "UHRF1"
[15] "CENPU"   "HELLS"   "RFC2"    "POLR1B"  "NASP"    "RAD51AP1" "GMNN"
[22] "WDR76"   "SLBP"    "CCNE2"   "UBR7"    "POLD3"   "MSH2"    "ATAD2"
[29] "RAD51"   "RRM2"    "CDC45"   "CDC6"    "EXO1"    "TIPIN"   "DSCC1"
[36] "BLM"     "CASP8AP2" "USP1"   "CLSPN"   "POLA1"   "CHAF1B"  "MRPL36"
[43] "E2F8"
```

Cell cycle scoring

After we get the list of gene names that are associated with cell cycle. We can run the `**CellCycleScoring()**` function from Seurat.

```
merged <- CellCycleScoring(object=merged, s.features=s.genes,
                           g2m.features=g2m.genes)
```

The CellCycleScoring() function will **calculate a score** for each cell based on the given genes and assign the cell a phase.

This function will add the columns `**S.Score**`, `**G2M.Score**`, and `**Phase**` to the `**meta.data**` table.

After calculation, we can take a look of our meta.data table.

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

Cell cycle scoring

```
                          seurat_clusters_res0.8 seurat_clusters_res0.5     S.Score
Rep1_ICBdT_AAACCTGAGCCAACAG-1                      5                      4  0.61857647
Rep1_ICBdT_AAACCTGAGCCTTGAT-1                      2                      3 -0.06225445
Rep1_ICBdT_AAACCTGAGTACCGGA-1                     11                      9 -0.14015383
Rep1_ICBdT_AAACCTGCACGGCCAT-1                      7                      6 -0.05258375
Rep1_ICBdT_AAACCTGCACGGTAAG-1                      3                      2 -0.10176717
Rep1_ICBdT_AAACCTGCATGCCACG-1                     11                      9 -0.08794336
                            G2M.Score Phase
Rep1_ICBdT_AAACCTGAGCCAACAG-1  0.19639961     S
Rep1_ICBdT_AAACCTGAGCCTTGAT-1 -0.13230705    G1
Rep1_ICBdT_AAACCTGAGTACCGGA-1 -0.17780932    G1
Rep1_ICBdT_AAACCTGCACGGCCAT-1 -0.06400598    G1
Rep1_ICBdT_AAACCTGCACGGTAAG-1 -0.06637093    G1
Rep1_ICBdT_AAACCTGCATGCCACG-1 -0.21230015    G1
>
```

The code will print out all columns in the meta.data table. If you scroll down to the end, you will see the added three columns: S.Score, G2M.Score, and Phase.

Cell cycle scoring

Why don't we use the G1-phase related genes to calculate the cell cycle score?

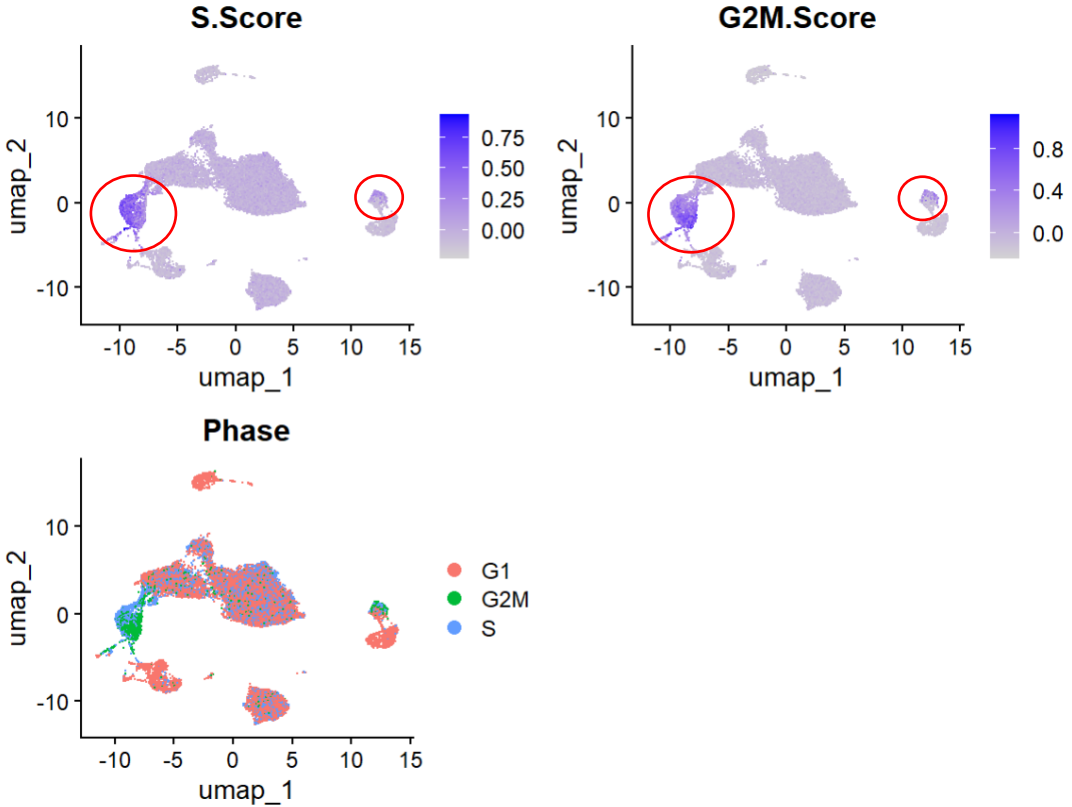
Because G1 phase is considered the "default" or "resting" state, and **it doesn't have a distinct, defined gene expression signature** the way S and G2/M phases does.

If a cell does not express high levels of S or G2/M phase genes, it's inferred to be in G1.

After cell cycle scoring and assigning phases, we can visualise the results.

```
FeaturePlot(merged, features = c("S.Score", "G2M.Score")) +
  DimPlot(merged, group.by = "Phase")
```

Cell cycle scoring

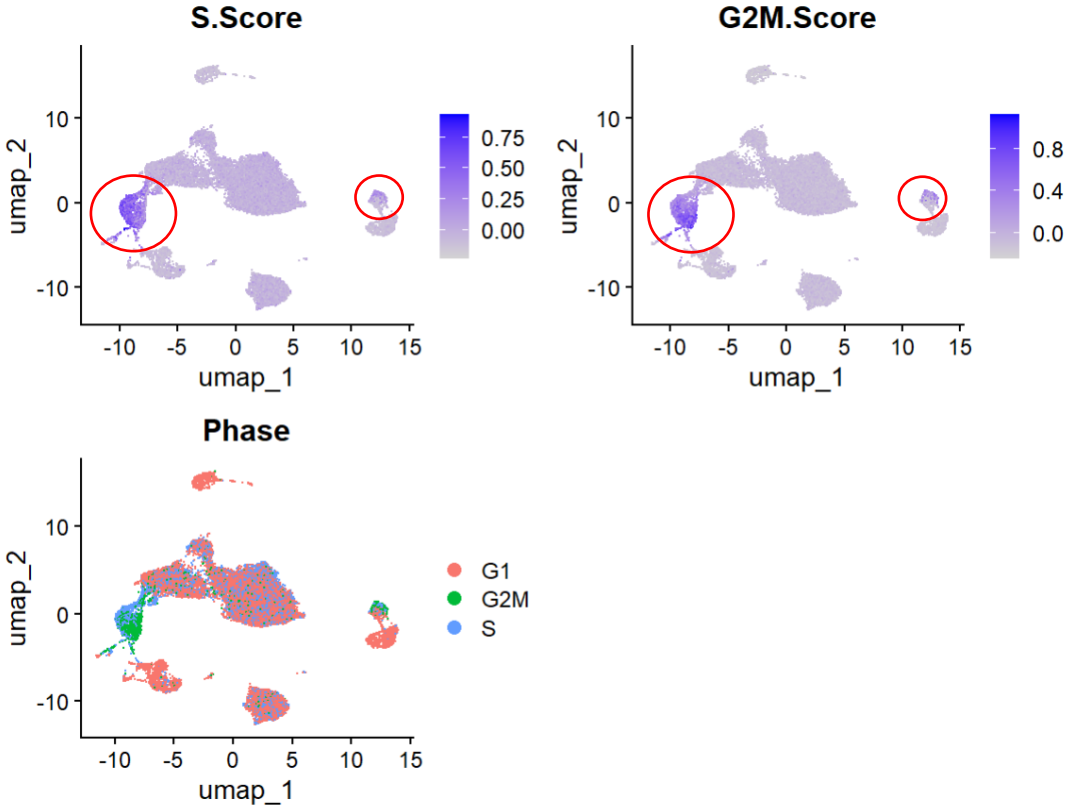


We can see that only clusters in the red circle were driven by cell cycle-related genes.

All other clusters are driven by other genes.

So, our data looks good.

Cell cycle scoring



The interpretation of cell cycle scoring is **largely context** and **experiment dependent**.

In some cases, if you are not expecting large differences in cell cycle in your samples, this may be an additional QC metric to consider.

Save Seurat object

Finally, we can save our object for further analysis:

```
saveRDS(merged, file = "rep135_clustered.rds")
```

Later, we can read in the "**rep135_clustered.rds**" file to continue our analysis without running everything again.

Thank you

Contact us

Jiajia Li
Research School of Biology

RN Robertson Building, 46 Sullivan's Creek Rd
The Australian National University
Canberra ACT 2600

E jiajia.li1@anu.edu.au

Australian National University

TEQSA PROVIDER ID: PRV12002 (AUSTRALIAN UNIVERSITY)
CRICOS PROVIDER CODE: 00120C