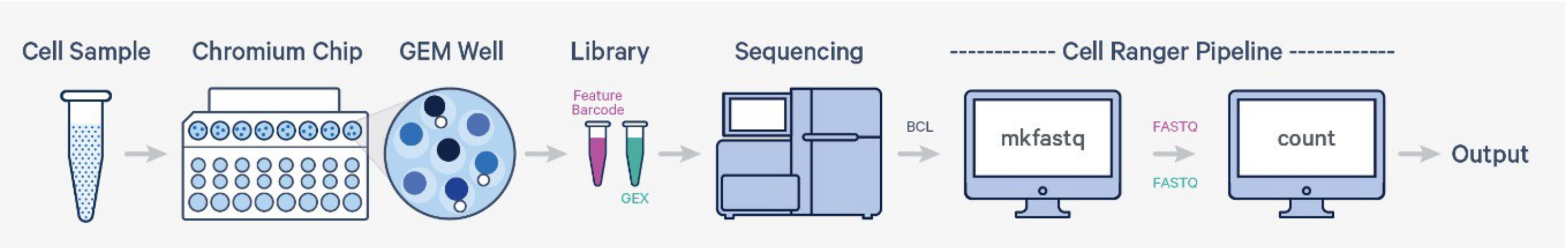
Single-cell RNA-seq

Based on:

https://github.com/hbctraining/scRNA-seq_online/tree/master/lessons

and the CERMO-FC Single-cell workshop



What is scRNA-seq?

Across human tissues there is an incredible diversity of cell types, states, and interactions. To better understand these tissues and the cell types present, **single-cell RNA-seq (scRNA-seq) offers a glimpse into what genes are being expressed at the level of individual cells.**

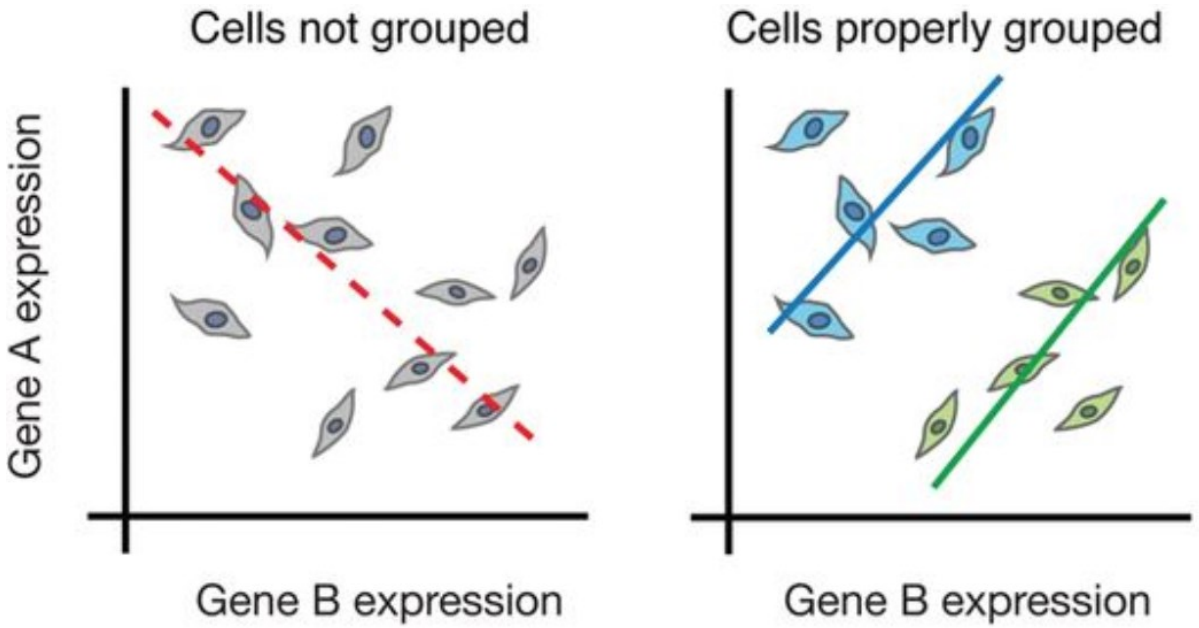
Why use scRNA-seq?

- Explore which cell types are present in a tissue
- Identify **unknown/rare cell types** or states
- Elucidate the **changes in gene expression** during differentiation processes or **across time or states**
- Identify **genes that are differentially expressed in particular cell type between conditions** (e.g. treatment or disease)
- Explore changes in expression among a cell type while incorporating spatial, regulatory, and/or protein information

Comparaison with bulk RNA-seq

- Transcriptome analysis using bulk RNA-seq:
  - compares the averages of cellular expression.
  - is the best choice if looking for disease biomarkers
  - not expecting or not concerned with a lot of cellular heterogeneity
  - can explore differences in gene expression between conditions (control, treated)
- Transcriptome analysis using scRNA-seq:
  - studying heterogeneity, lineage tracing study and stochastic gene expression

Challenges of scRNA-seq

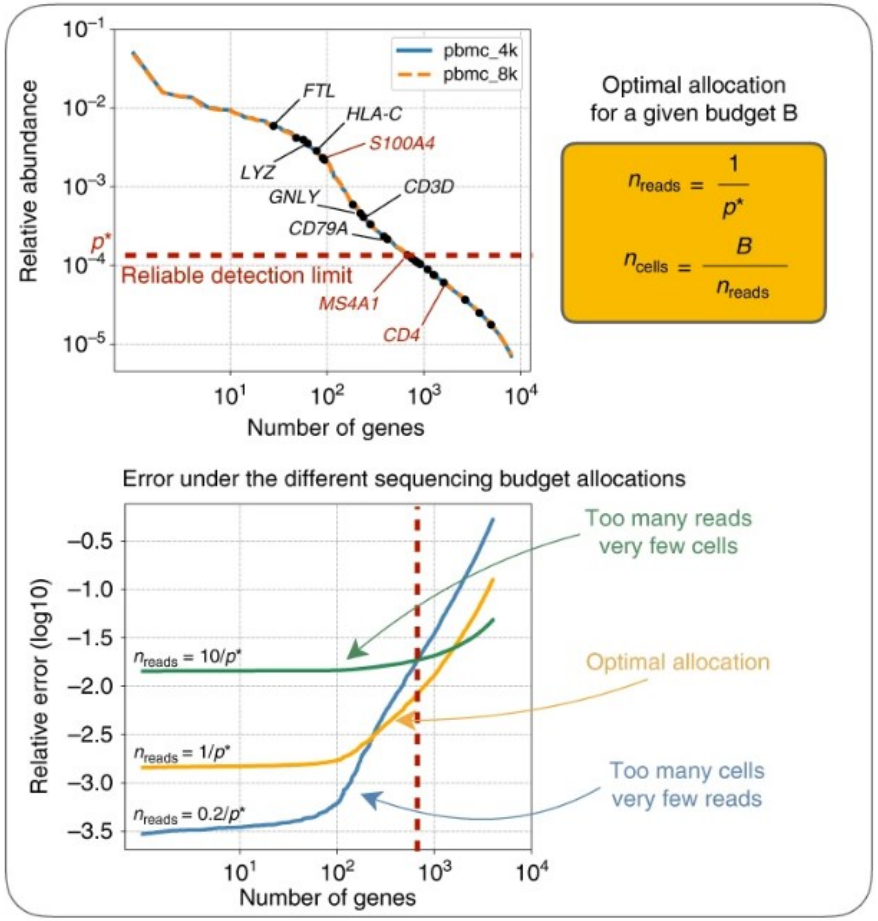


- While bulk RNA-seq can explore differences in gene expression between conditions (e.g. treatment or disease), the differences at the cellular level are not adequately captured.
- For instance, in the images below, if analyzed **in bulk (left)** we would not detect the correct association between the expression of gene A and gene B. However, if we properly group the cells by cell type or cell state, we can see the correct correlation between the genes **(right, scRNA-seq)**.

$$ Sample generation and library preparation are more expensive $$

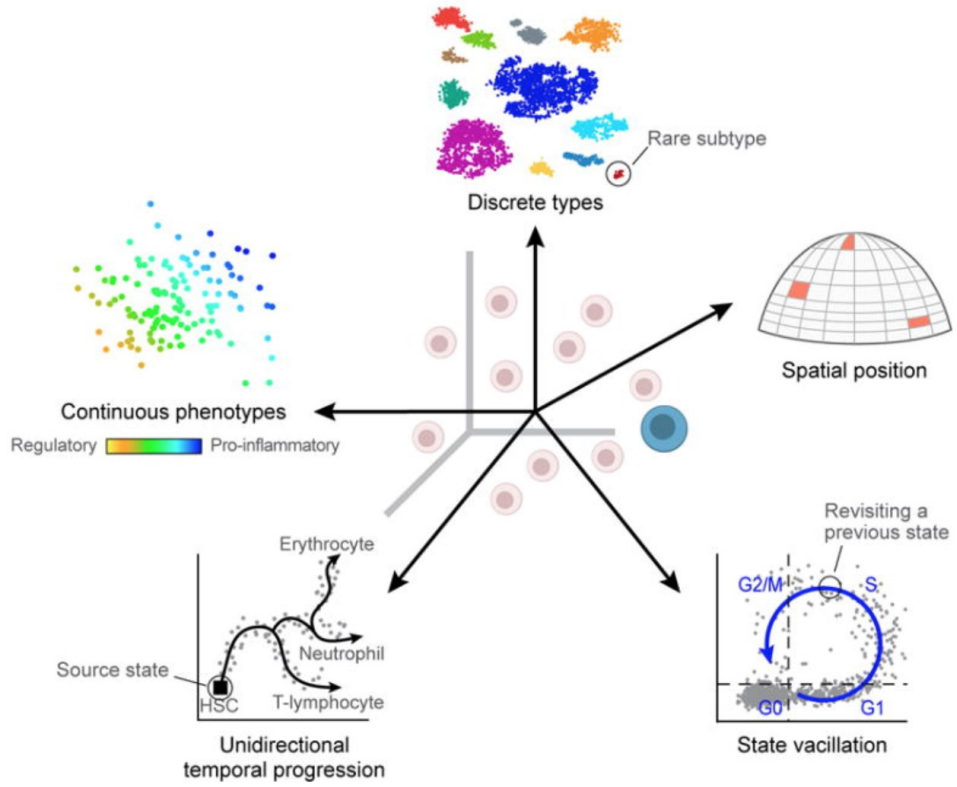
Complexity of analysis of scRNA-seq data:

- A large volume of data
  - Expression data represent **tens or hundreds of thousands of reads for thousands of cells** = + memory, + storage, + time
- Low depth of sequencing per cell
  - The droplet-based methods **detect only 10-50% of the transcriptome per cell**
  - In a particular cell, a **zero count** could either mean that the gene was **not being expressed** or the transcripts were just **not detected**.



Zhang, M.J., Ntranos, V. & Tse, D. Determining sequencing depth in a single-cell RNA-seq experiment. *Nat Commun* **11,** 774 (2020). https://doi.org/10.1038/s41467-020-14482-y

Complexity of analysis of scRNA-seq data:



Wagner, A., Regev, A. & Yosef, N. Revealing the vectors of cellular identity with single-cell genomics. *Nat Biotechnol* **34,** 1145–1160 (2016). https://doi.org/10.1038/nbt.3711

- <u>Biological variability across cells/samples</u>
  - **Transcriptional bursting**: Gene transcription is not turned on all of the time for all genes. Time of harvest will determine whether gene is on or off in each cell.
  - **Varying rates of RNA processing**: Different RNAs are processed at different rates.
  - **Continuous or discrete cell**: Continuous phenotypes are variable in gene expression and separating the continuous from the discrete can sometimes be difficult.
  - **Environmental stimuli**: The local environment of the cell can influence the gene expression depending on spatial position, signaling molecules, etc.
  - **Temporal changes**: Fundamental fluxuating cellular processes, such as cell cycle, can affect the gene expression profiles of individual cells.

Complexity of analysis of scRNA-seq data:

- Technical variability across cells/samples
  - **Cell-specific capture efficiency**: Different cells will have differing numbers of transcripts captured resulting in differences in sequencing depth (e.g., 10-50% of transcriptome).
  - **Library quality**: Degraded RNA, low viability/dying cells, lots of free-floating RNA, poorly dissociated cells, and inaccurate quantitation of cells can result in low quality metrics
  - **Amplification bias**: During the amplification step of library preparation, not all transcripts are amplified to the same level.
  - **Batch effects**: Batch effects are a significant issue for scRNA-Seq analyses, since you can see significant differences in expression due solely to the batch effect.

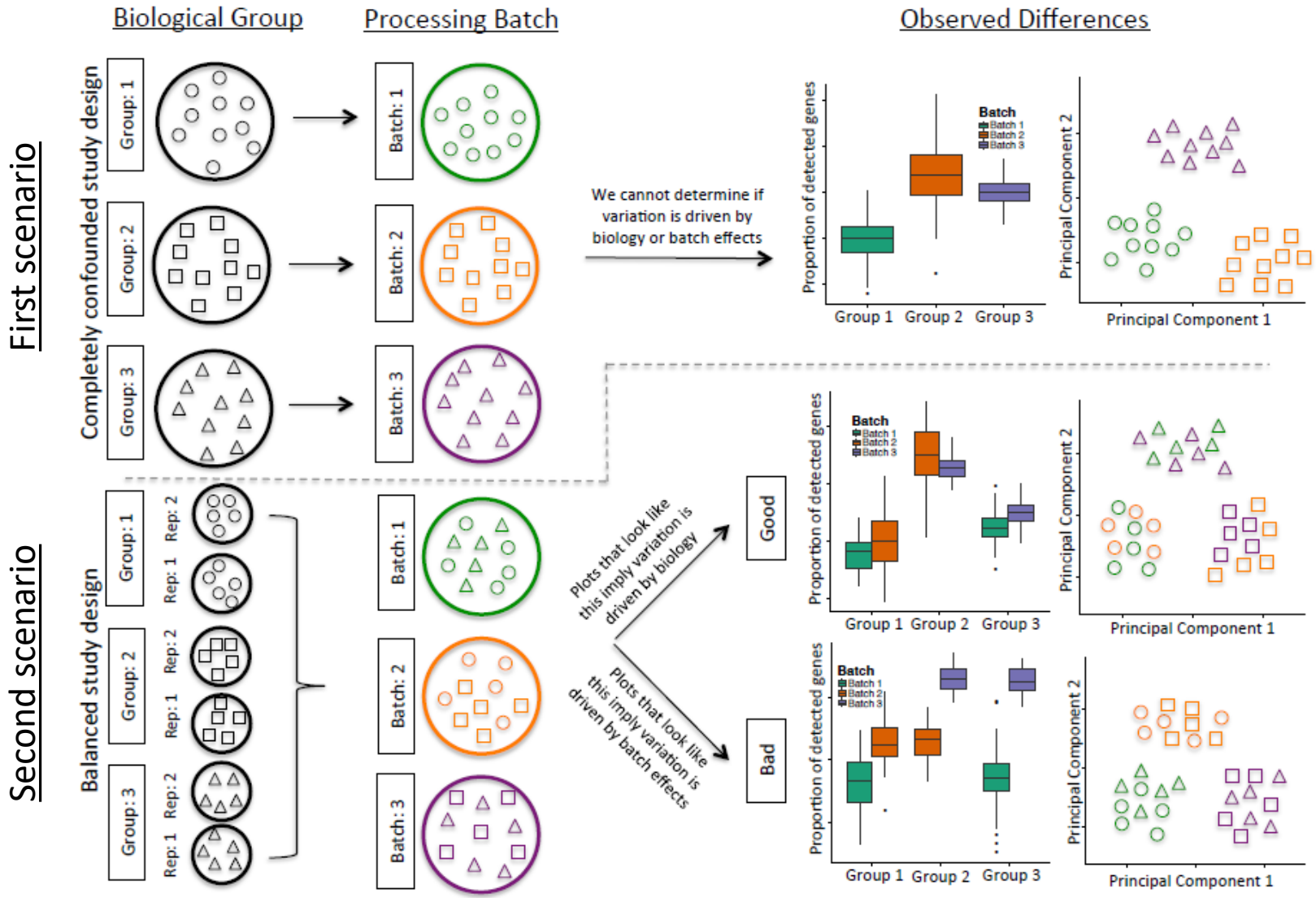
Technical sources of variation can result in **gene expression between cells being more similar/different based on technical sources instead of biological cell types/states.**

Batch effects

How to know whether you have batches?

- Were all RNA isolations performed on the same day?
- Were all library preparations performed on the same day?
- Did the same person perform the RNA isolation/library preparation for all samples?
- Did you use the same reagents for all samples?
- Did you perform the RNA isolation/library preparation in the exact location?

If *any* above answer is "No", then you have batches!



Hicks, S. C., Teng, M., & Irizarry, R. A. (2015). On the widespread and critical impact of systematic bias and batch effects in single-cell RNA-Seq data. *BioRxiv*, *10*, 025528. https://doi.org/10.1101/025528

Best practices regarding batches:

- **Do NOT confound** your experiment by batch: First scenario.
- **DO** split replicates of the different sample groups across batches. The more replicates the better (definitely more than 2), if doing DE across conditions or making conclusions at the population level. If using inDrops, which prepares a single library at a time, *alternate the sample groups* (e.g. don't prepare all control libraries first, then prepare all treatment libraries). Second scenario.
- **DO** include batch information in your **experimental metadata**. During the analysis, we can regress out variation due to batch or integrate across batches, so it doesn’t affect our results if we have that information.

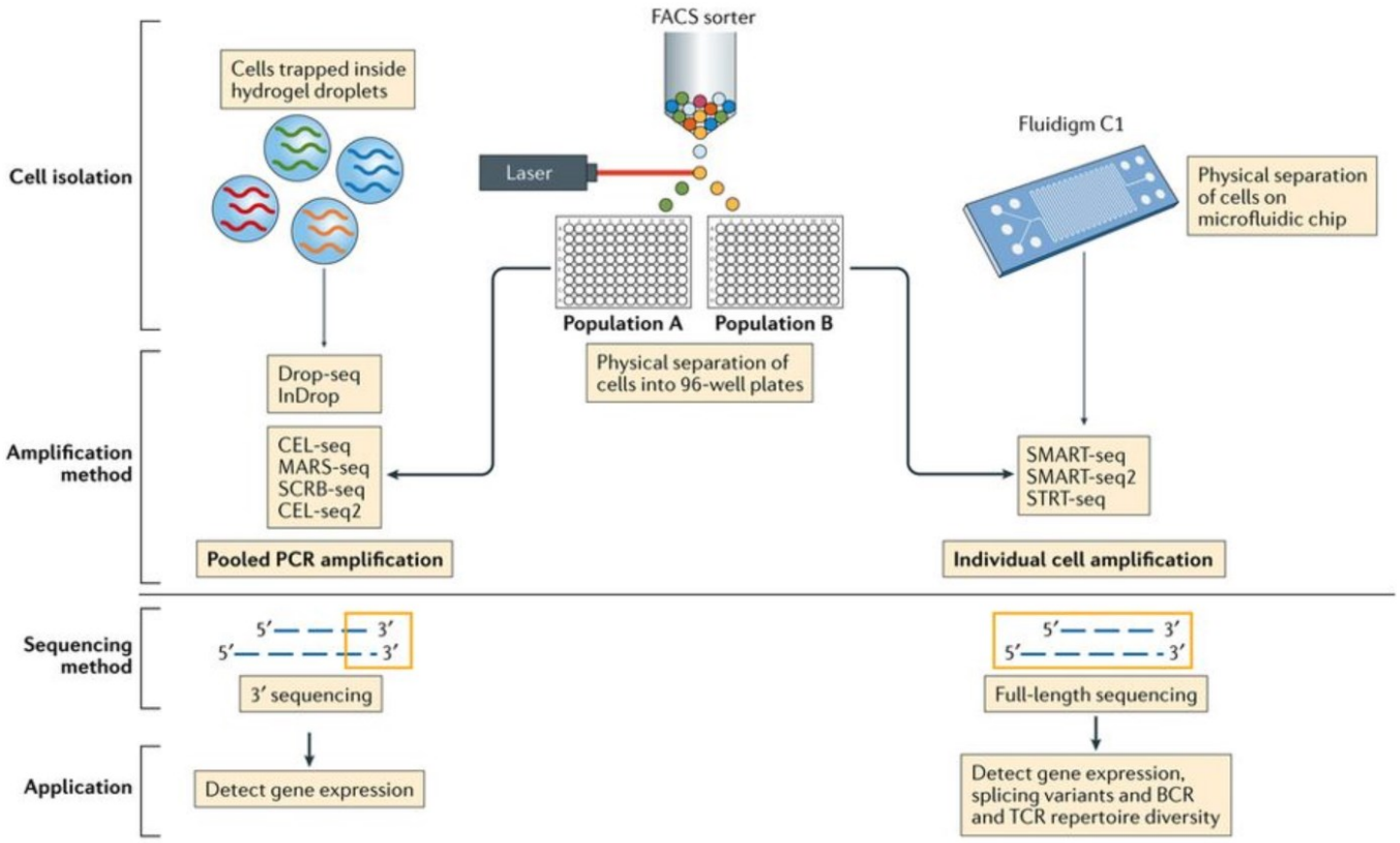
Conclusions / Overall recommendations

- Do not perform single-cell RNA-seq unless it is necessary for the experimental question of interest. **Could you answer the question using bulk sequencing, which is simpler and less costly?** Perhaps FACS sorting the samples could allow for bulk analysis?
- **Understand the details of the experimental question** you wish to address. The recommended library preparation method and analysis workflow can vary based on the specific experiment.
- **Avoid technical sources of variability**, if possible:
  - Discuss experimental design with experts prior to the initiation of the experiment
  - Isolate RNA from samples at same time
  - Prepare libraries at same time or alternate sample groups to avoid batch confounding
  - Do not confound sample groups by sex, age or batch

Raw data to count matrix

Library preparation

- The RNA sequences (also referred to as reads or tags), will be derived either
  - from the 3' ends (or 5' ends) of the transcripts (**10X Genomics**, CEL-seq2, Drop-seq, inDrops) or
  - from full-length transcripts (Smart-seq).

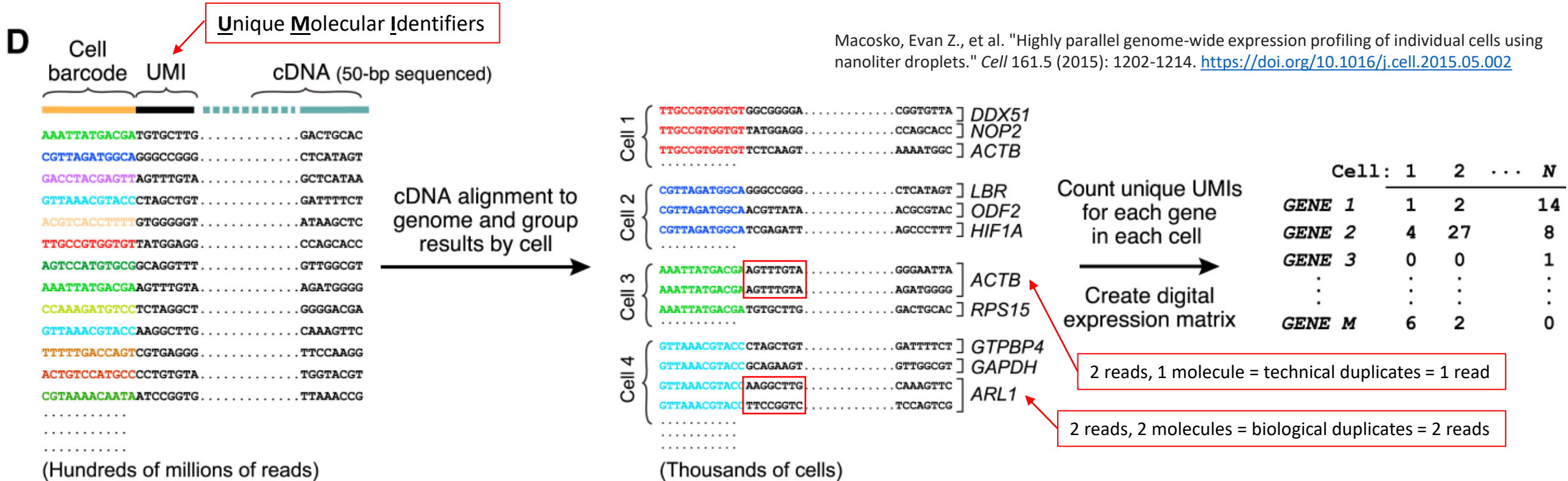


Papalexi, E., Satija, R. Single-cell RNA sequencing to explore immune cell heterogeneity. *Nat Rev Immunol* **18,** 35–45 (2018). https://doi.org/10.1038/nri.2017.76

3'-end reads (includes all droplet-based methods)

For the analysis of scRNA-seq data, it is helpful to understand **what information is present in each of the reads** and how we use it moving forward through the analysis.

- Reads with **different UMIs** mapping to the same transcript were derived from **different molecules** and are biological duplicates - each read should be counted.
- Reads with the **same UMI** originated from the **same molecule** and are technical duplicates - the UMIs should be collapsed to be counted as a single read.
- In image below, the reads for ACTB should be collapsed and counted as a single read, while the reads for ARL1 should each be counted.



Macosko, Evan Z., et al. "Highly parallel genome-wide expression profiling of individual cells using nanoliter droplets." *Cell* 161.5 (2015): 1202-1214. https://doi.org/10.1016/j.cell.2015.05.002

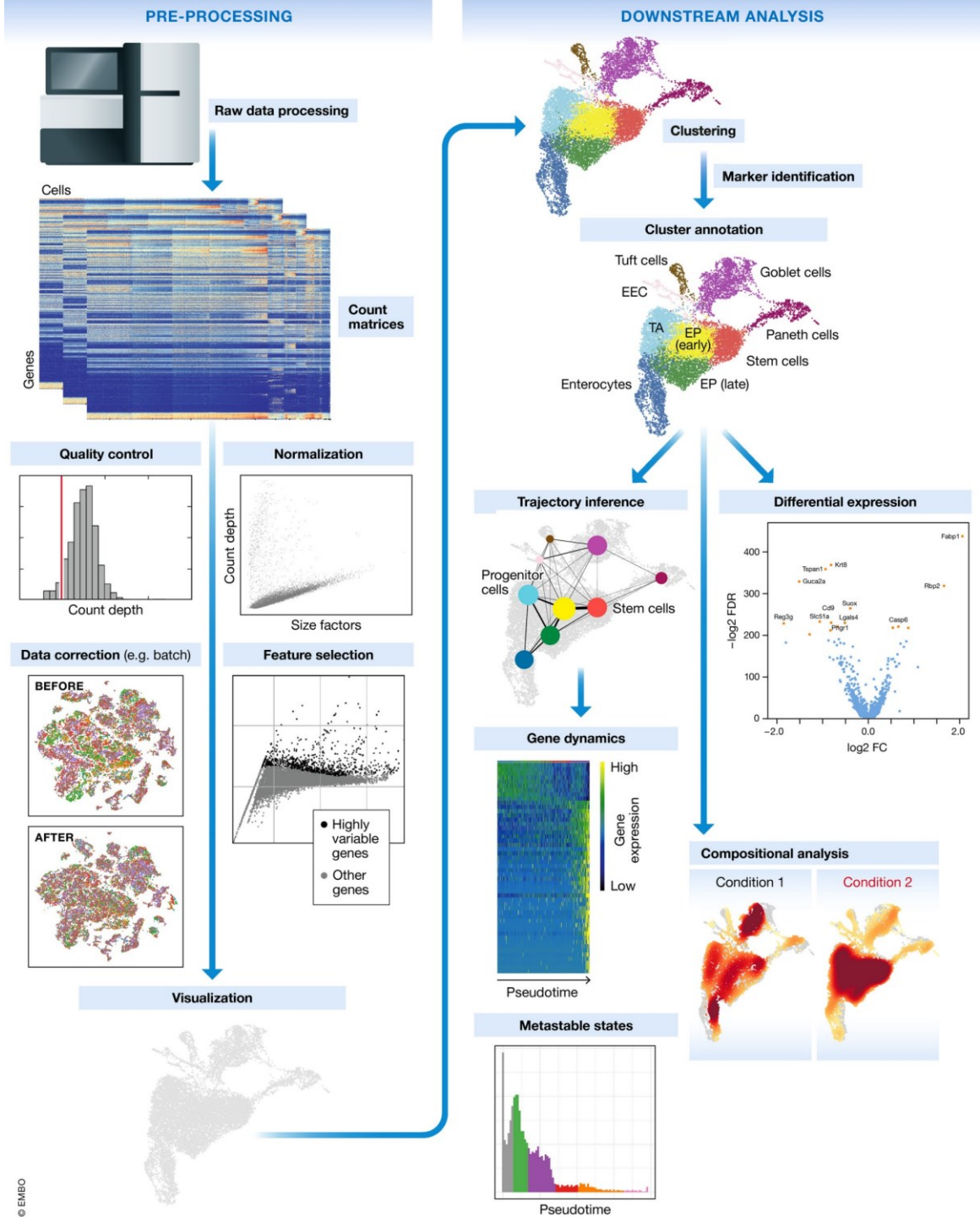
Workflow

1. **Generation of the count matrix** (method-specific steps): formating reads, demultiplexing samples, mapping and quantification
2. **Quality control of the raw counts**: filtering of poor quality cells
3. **Clustering of filtered counts**: clustering cells based on similarities in transcriptional activity (cell types = different clusters)
4. **Marker identification and cluster annotation**: identifying gene markers for each cluster and annotating known cell type clusters

Optional downstream steps

Conclusions about a ***population*** based on a single sample per condition are not trustworthy.

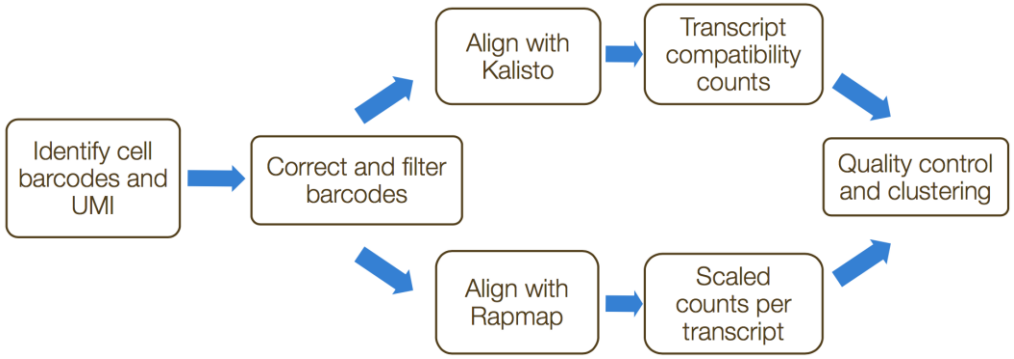
BIOLOGICAL REPLICATES ARE STILL NEEDED!



Luecken, MD and Theis, FJ. Current best practices in single-cell RNA-seq analysis: a tutorial, Mol Syst Biol 2019. https://doi.org/10.15252/msb.20188746

Generation of count matrix (not Cell Ranger)

1. Formatting reads and filtering noisy cellular barcodes
   - Known cellular barcodes used in the library preparation method should be known, and unknown barcodes would be dropped, while allowing for an acceptable number of mismatches to the known cellular barcodes.
2. Demultiplexing the samples
   - If sequencing more than a single sample: We would need to parse the reads to determine the sample barcode associated with each cell.
3. Mapping/pseudo-mapping to transcriptome
   - To determine which gene the read originated from, the reads are aligned using traditional (STAR) or light-weight methods (Kallisto/RapMap).
4. Collapsing UMIs and quantification of reads
   - The resulting output is a cell by gene matrix of counts



Generation of count matrix (Cell Ranger)

If using 10X Genomics library preparation method, then the Cell Ranger pipeline would be used for all of the above steps.

- **cellranger mkfastq** demultiplexes raw base call (BCL) files generated by Illumina sequencers into FASTQ files. It is a wrapper around Illumina's bcl2fastq, with additional features that are specific to 10x Genomics libraries and a simplified sample sheet format.
- **cellranger count** takes FASTQ files from `cellranger mkfastq` and performs alignment, filtering, barcode counting, and UMI counting. It uses the Chromium cellular barcodes to generate feature-barcode matrices, determine clusters, and perform gene expression analysis. The `count` pipeline can take input from multiple sequencing runs on the same GEM well. `cellranger count` also processes Feature Barcode data alongside Gene Expression reads.
- **cellranger multi** is used to analyze Cell Multiplexing and Fixed RNA Profiling data. It takes FASTQ files from `cellranger mkfastq` and performs alignment, filtering, barcode counting, and UMI counting. It uses the Chromium cellular barcodes to generate feature-barcode matrices, determine clusters, and perform gene expression analysis. The `cellranger multi` pipeline also supports the analysis of Feature Barcode data.
- **cellranger aggr** aggregates outputs from multiple runs of `cellranger count` or `cellranger multi`, normalizing those runs to the same sequencing depth and then recomputing the feature-barcode matrices and analysis on the combined data. The `aggr` pipeline can be used to combine data from multiple samples into an experiment-wide feature-barcode matrix and analysis.
- **cellranger reanalyze** takes feature-barcode matrices produced by `cellranger count`, `cellranger multi`, or `cellranger aggr` and reruns the dimensionality reduction, clustering, and gene expression algorithms using tunable parameter settings.

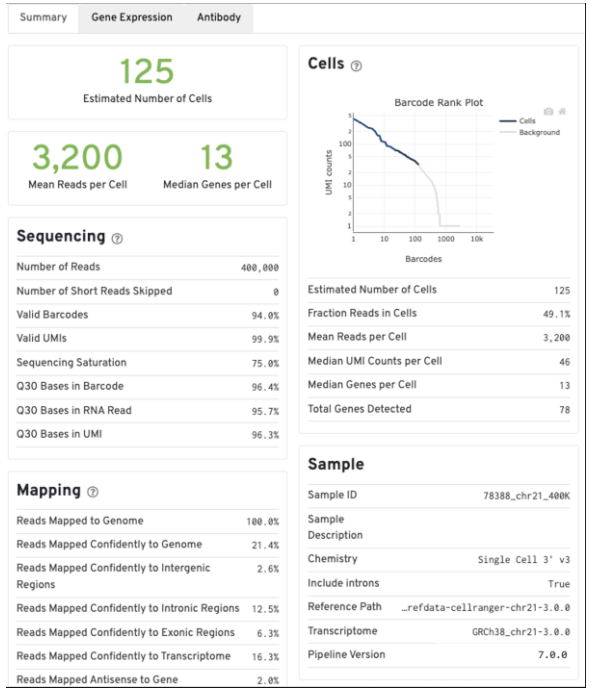
Cell Ranger output

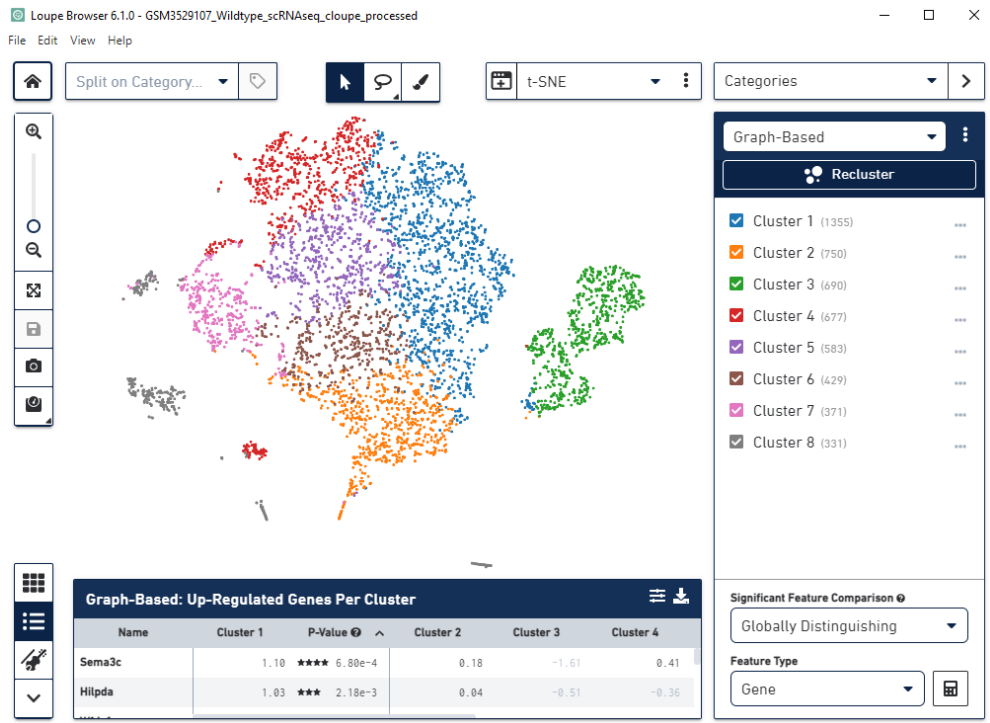
- Web report (.html)
  - Interactive report with summary metrics

- Loupe Browser file (.cloupe)
  - Readable by **Loupe Browser**
  - (free desktop application by 10X genomics)

- For each sample:
  - barcodes.tsv
  - features.tsv
  - matrix.mtx

Will be seen later





Example Dataset

Example dataset

- The data used to test their algorithm is comprised of pooled Peripheral Blood Mononuclear Cells (PBMCs) taken from eight lupus patients, split into control and interferon beta-treated (stimulated) conditions.
- **Raw data**
  - we downloaded the BAM files from the SRA (SRP102802). These BAM files were converted back to FASTQ files, then run through Cell Ranger to obtain the count data that we will be using.
  - **NOTE:** The count data for this dataset is also freely available from 10X Genomics and is used in the Seurat tutorial.
- **Metadata**
  - In addition to the raw data, we also need to collect **information about the data**; this is known as **metadata**.

Good practice

You should know:

- Where the data is come from:
  - Nucleus RNA / cell RNA
  - Cell-line / tissue / organ
  - Fresh / frozen / fixed cells
  - Immune cells (specific protocols)
- Preparation steps (communicate with the biologist (prior to the study design and in the process of the analysis)
  - Number of samples and replicas
  - Cell sorting/count
  - Ask biologists to remove dead cells
  - Sequencing parameters (20K – 50K reads/cell)

- The libraries were prepared using ? method
- The samples were sequenced on ?
- Samples from ? Individual ?
- ? and ? cells were identified (after removing doublets) for control and stimulated, respectively
- ? observations in the process

Back to the dataset

- The libraries were prepared using **10X Genomics** version 2 chemistry
- The samples were sequenced on the **Illumina NextSeq 500**
- PBMC samples from **eight individual lupus patients** were separated into two aliquots each.
- **12,138 and 12,167 cells** were identified (after removing doublets) for **control and stimulated** pooled samples, respectively.
- Since the samples are PBMCs, **we will expect immune cells**, such as:
  - B cells
  - T cells
  - NK cells
  - monocytes
  - macrophages
  - possibly megakaryocytes

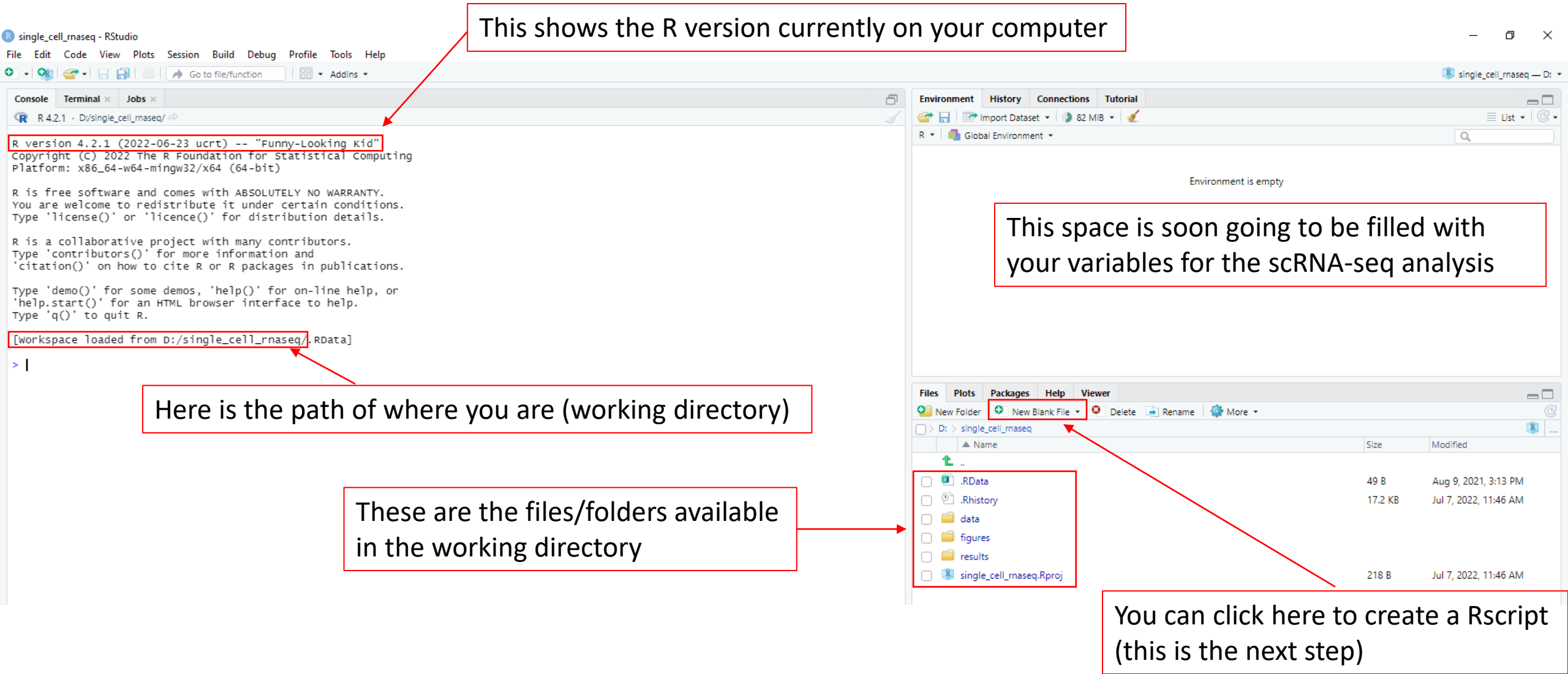
Time to work!

Using Rstudio and multiple packages

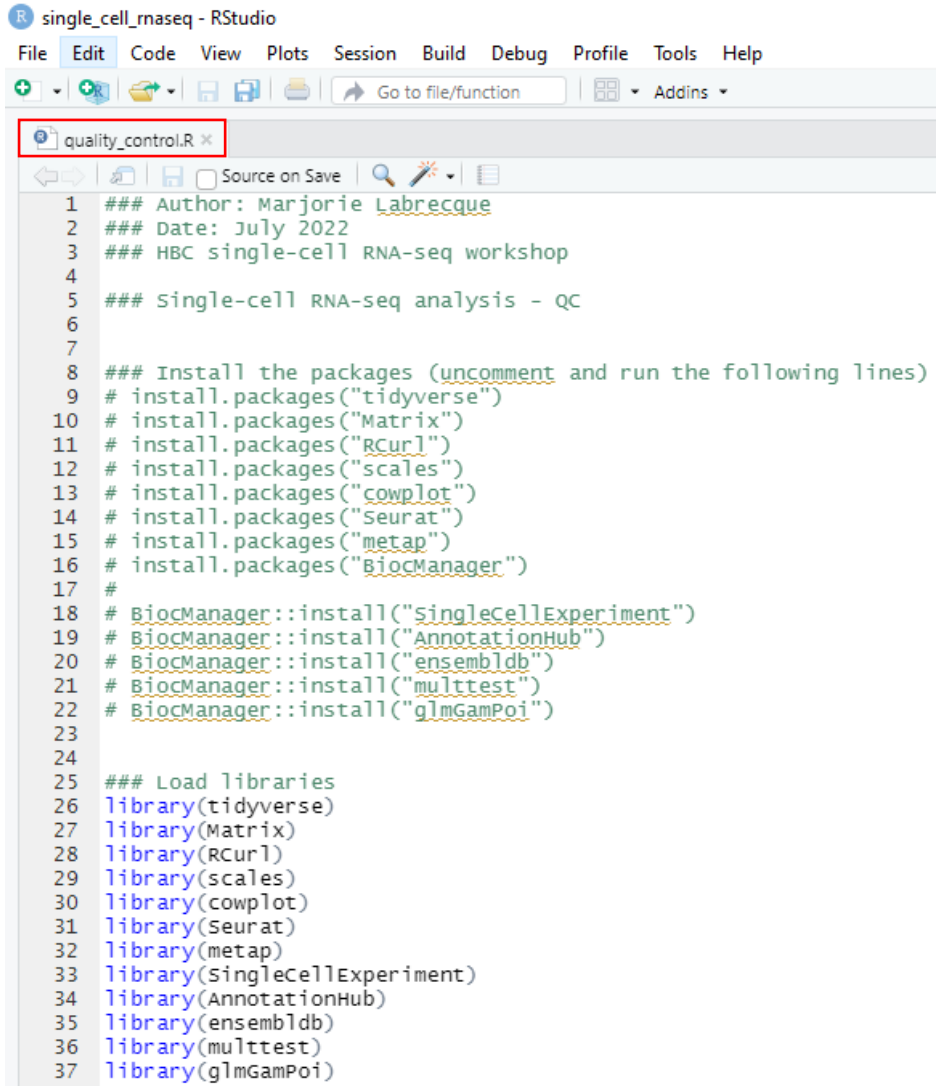
R and RStudio

1. Install R:
   - For macOS: https://cran.rstudio.com/bin/macosx/
   - For Windows: https://cran.rstudio.com/bin/windows/
2. Install Rstudio Desktop:
   - https://www.rstudio.com/products/rstudio/download/#download
3. Download the dataset:
   - https://www.dropbox.com/s/we1gmyb9c8jej2u/single_cell_rnaseq.zip?dl=1
4. Unzip the *single_cell_rnaseq.zip* file
5. Move the *single_cell_rnaseq* folder to your working directory
6. Click on the *single_cell_rnaseq.Rproj* file

Your Rstudio should look like this:



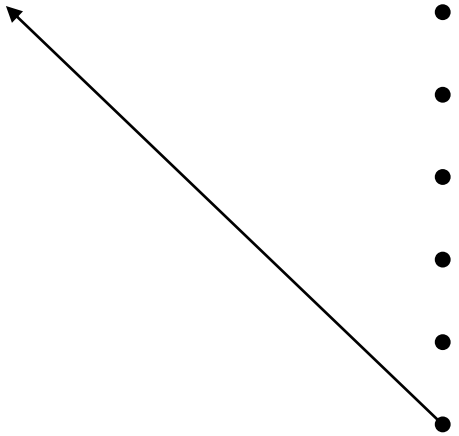
First script



- You can add some information of what the script is for
- Next you can install the packages (on the next slide) and load the libraries

Packages

1. Install the 5 packages listed below from Bioconductor using the the BiocManager::install() function.
   - SingleCellExperiment
   - AnnotationHub
   - ensembldb
   - multtest
   - glmGamPoi



1. Install the 8 packages listed below from CRAN using the install.packages() function.
   - tidyverse
   - Matrix
   - RCurl
   - scales
   - cowplot
   - BiocManager
   - Seurat
   - metap

Raw data / Cell Ranger output

For each sample: 3 files:

- **cell IDs**, representing all quantified cells (***barcodes.tsv***)
  - This is a text file which contains all cellular barcodes present for that sample. Barcodes are listed in the order of data presented in the matrix file (i.e. these are the column names).
- **gene IDs**, representing all genes quantified (***features.tsv***)
  - This is a text file which contains the identifiers of the quantified genes. The source of the identifier can vary depending on what reference (i.e. Ensembl, NCBI, UCSC) you use in the quantification methods, but most often these are official gene symbols. The order of these genes corresponds to the order of the rows in the matrix file (i.e. these are the row names).
- **matrix of counts** per gene for every cell (***matrix.mtx***)
  - This is a text file which contains a matrix of count values. The rows are associated with the gene IDs above and columns correspond to the cellular barcodes. Note that there are many zero values in this matrix.

Function **Read10X()**: This function is from the **Seurat package** and will use the **Cell Ranger output directory as input**, directly. With this method individual files do not need to be loaded in, instead the function will load and combine them into a sparse matrix. **We will be using this function to load in our data!**

Loading the data (one sample)

```
# How to read in 10X data for a single sample (output is a sparse matrix)
ctrl_counts <- Read10X(data.dir = "data/ctrl_raw_feature_bc_matrix")
# Turn count matrix into a Seurat object (output is a Seurat object)
ctrl <- CreateSeuratObject(counts = ctrl_counts, min.features = 100)
# Explore the metadata
head(ctrl@meta.data)
```

Columns of metadata:
orig.ident: this often contains the sample identity if known, but will default to "SeuratProject"
nCount_RNA: number of UMIs per cell
nFeature_RNA: number of genes detected per cell

The min.features argument specifies the minimum number of genes that need to be detected per cell. This argument will filter out poor quality cells that likely just have random barcodes encapsulated without any cell present. Usually, cells with less than 100 genes detected are not considered for analysis.

Loading the data (in a loop)

```
# Create a Seurat object for each sample
for (file in c("ctrl_raw_feature_bc_matrix",
"stim_raw_feature_bc_matrix")){
    seurat_data <- Read10X(data.dir = paste0("data/", file))
    seurat_obj <- CreateSeuratObject(counts = seurat_data,
                                     min.features = 100,
                                     project = file)
    assign(file, seurat_obj)
}
```

Specify the sample names

Get the correct file in the folder

Assign each sample to the correct Seurat object.
Creates one object for each sample name
Can access the metadata after:
head(ctrl_raw_feature_bc_matrix@meta.data)
head(stim_raw_feature_bc_matrix@meta.data)

Merging samples

- This will make it easier to run the QC steps for both sample groups together and enable us to easily compare the data quality for all the samples.

- 2 samples:

```
merged_seurat <- merge(x = ctrl_raw_feature_bc_matrix,
                       y = stim_raw_feature_bc_matrix,
                       add.cell.id = c("ctrl", "stim"))
```

- 3+ samples (add more to the y):

```
merged_seurat <- merge(x = ctrl_raw_feature_bc_matrix,
                       y = c(stim1_raw_feature_bc_matrix, stim2_raw_feature_bc_matrix, stim3_raw_feature_bc_matrix),
                       add.cell.id = c("ctrl", "stim1", "stim2", "stim3"))
```

```
# Check that the merged object has the appropriate sample-specific prefixes
head(merged_seurat@meta.data)
tail(merged_seurat@meta.data)
```

Quality Control

Quality metrics

In order to create the appropriate plots for the quality control analysis, we need to calculate some additional metrics. These include:

- **number of genes detected per UMI:** this metric with give us an idea of the complexity of our dataset (more genes detected per UMI, more complex our data)
  - This value is quite easy to calculate, as we take the log10 of the number of genes detected per cell and the log10 of the number of UMIs per cell, then divide the log10 number of genes by the log10 number of UMIs.

  # Add number of genes per UMI for each cell to metadata

  merged_seurat$log10GenesPerUMI <- log10(merged_seurat$nFeature_RNA) / log10(merged_seurat$nCount_RNA)

- **mitochondrial ratio:** this metric will give us a percentage of cell reads originating from the mitochondrial genes
  - For each cell, the function takes the sum of counts across all genes (features) belonging to the "Mt-" set, and then divides by the count sum for all genes (features). This value is multiplied by 100 to obtain a percentage value.

  # Compute percent mito ratio

  merged_seurat$mitoRatio <- PercentageFeatureSet(object = merged_seurat, pattern = "^MT-")

  merged_seurat$mitoRatio <- merged_seurat@meta.data$mitoRatio / 100 *#gives a ratio instead of percentage*

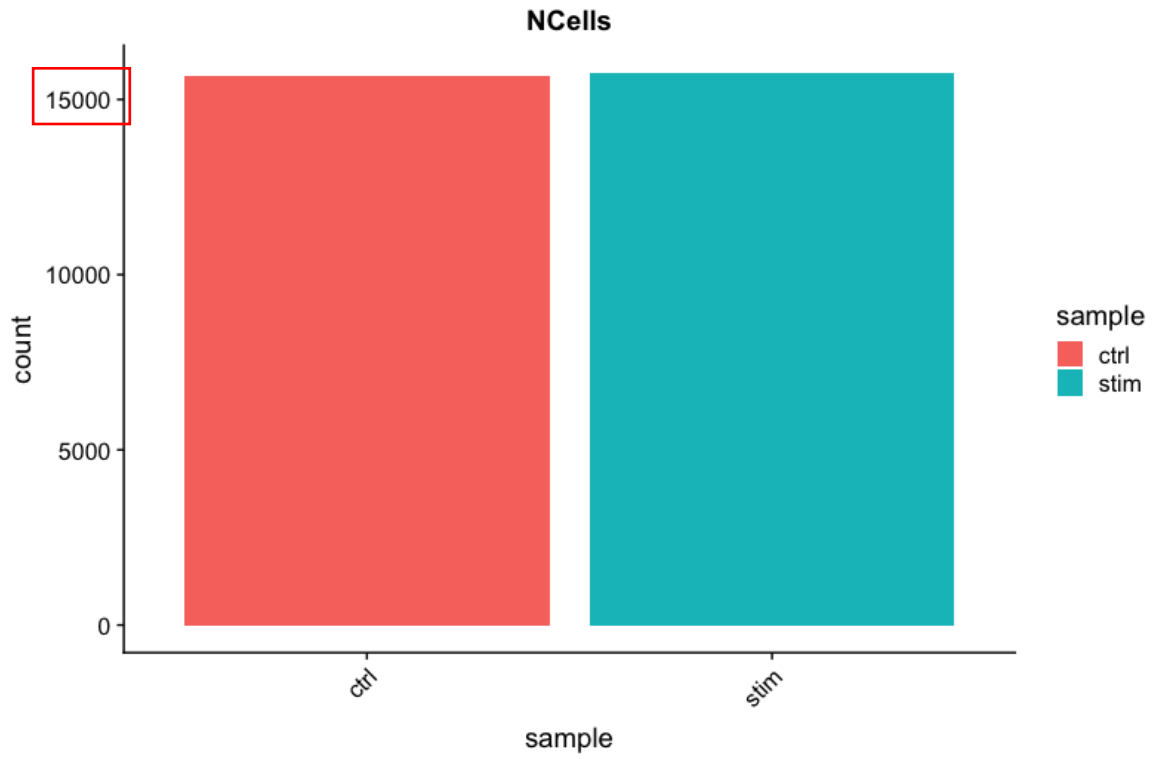
Arrange the metadata

```
# Create metadata dataframe
metadata <- merged_seurat@meta.data
# Add cell IDs to metadata
metadata$cells <- rownames(metadata)
# Create sample column
metadata$sample <- NA
metadata$sample[which(str_detect(metadata$cells, "^ctrl_"))] <- "ctrl"
metadata$sample[which(str_detect(metadata$cells, "^stim_"))] <- "stim"
# Rename columns
metadata <- metadata %>%
      dplyr::rename(seq_folder = orig.ident,
                    nUMI = nCount_RNA,
                    nGene = nFeature_RNA)
# Add metadata back to Seurat object
merged_seurat@meta.data <- metadata
# See resulting merged metadata
View(merged_seurat@meta.data)
# Create .RData object to load at any time
save(merged_seurat, file="data/merged_filtered_seurat.RData")
```

Assessing the quality metrics (Cell counts)

- The cell counts are determined by the number of unique cellular barcodes detected. For this experiment, between 12,000 -13,000 cells are expected.

```
# Visualize the number of cell counts per sample
metadata %>%
            ggplot(aes(x=sample, fill=sample)) +
            geom_bar() +
            theme_classic() +
            theme(axis.text.x = element_text(angle = 45,
vjust = 1, hjust=1)) +
            theme(plot.title = element_text(hjust=0.5,
face="bold")) +
            ggtitle("NCells")
```

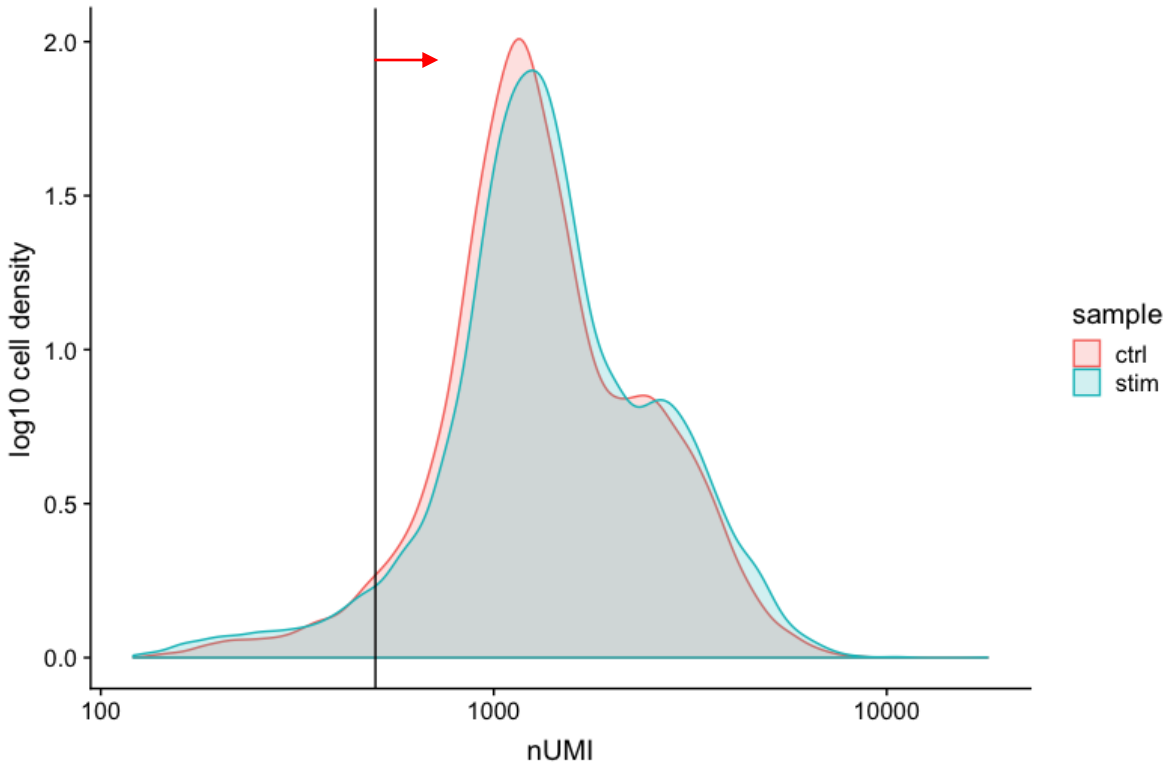


We see over 15,000 cells per sample, which is quite a bit more 13,000 expected. It is clear that we likely have some junk 'cells' present.

Assessing the quality metrics (UMI counts per cell)

- The UMI counts per cell should generally be above 500, that is the low end of what we expect. If UMI counts are between 500-1000 counts, it is usable, but the cells probably should have been sequenced more deeply.

```
# Visualize the number UMIs/transcripts per cell
metadata %>%
        ggplot(aes(color=sample, x=nUMI, fill= sample)) +
        geom_density(alpha = 0.2) +
        scale_x_log10() +
        theme_classic() +
        ylab("Cell density") +
        geom_vline(xintercept = 500)
```

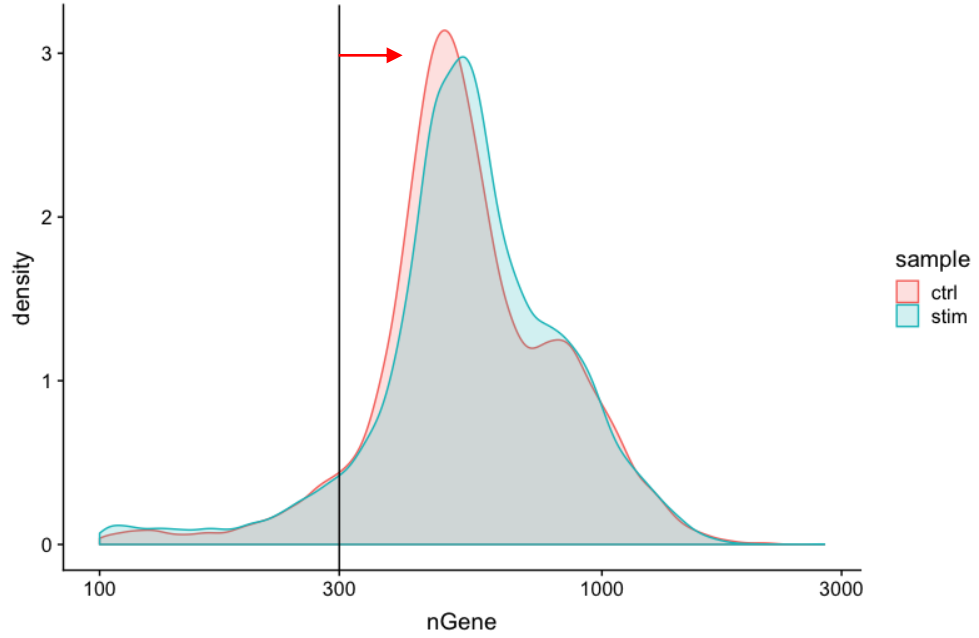


We can see that majority of our cells in both samples have 1000 UMIs or greater, which is great.

Assessing the quality metrics (Genes detected per cell)

- We have similar expectations for gene detection as for UMI detection, although it may be a bit lower than UMIs. For high quality data, **the proportional histogram should contain a single large peak that represents cells that were encapsulated**. If we see a small shoulder to the left of the major peak (not present in our data), or a bimodal distribution of the cells, that can indicate a couple of things. It might be that there are a set of cells that failed for some reason. It could also be that there are biologically different types of cells (i.e. quiescent cell populations, less complex cells of interest), and/or one type is much smaller than the other (i.e. cells with high counts may be cells that are larger in size). Therefore, this threshold should be assessed with other metrics that we describe in this lesson.

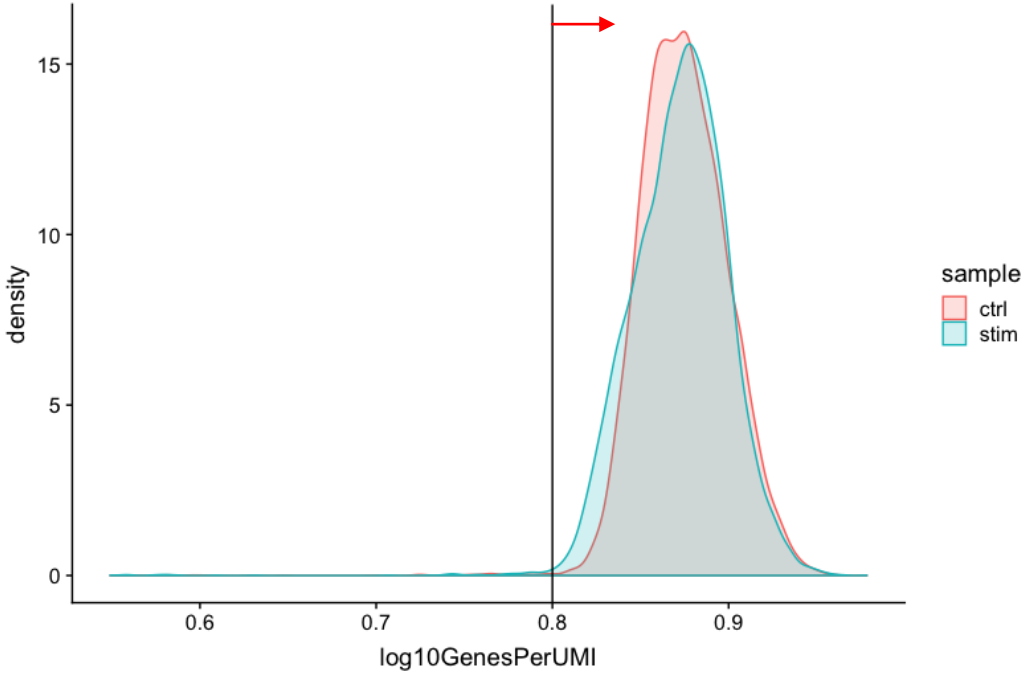
```
# Visualize the distribution of genes detected per cell via
histogram
metadata %>%
        ggplot(aes(color=sample, x=nGene, fill= sample)) +
        geom_density(alpha = 0.2) +
        theme_classic() +
        scale_x_log10() +
        geom_vline(xintercept = 300)
```



Assessing the quality metrics (Complexity)

- We can evaluate each cell in terms of how complex the RNA species are by using a measure called the novelty score. The novelty score is computed by taking the ratio of nGenes over nUMI. If there are many captured transcripts (high nUMI) and a low number of genes detected in a cell, this likely means that you only captured a low number of genes and simply sequenced transcripts from those lower number of genes over and over again. These low complexity (low novelty) cells could represent a specific cell type (i.e. red blood cells which lack a typical transcriptome) or could be due to an artifact or contamination. Generally, we expect the novelty score to be above 0.80 for good quality cells.

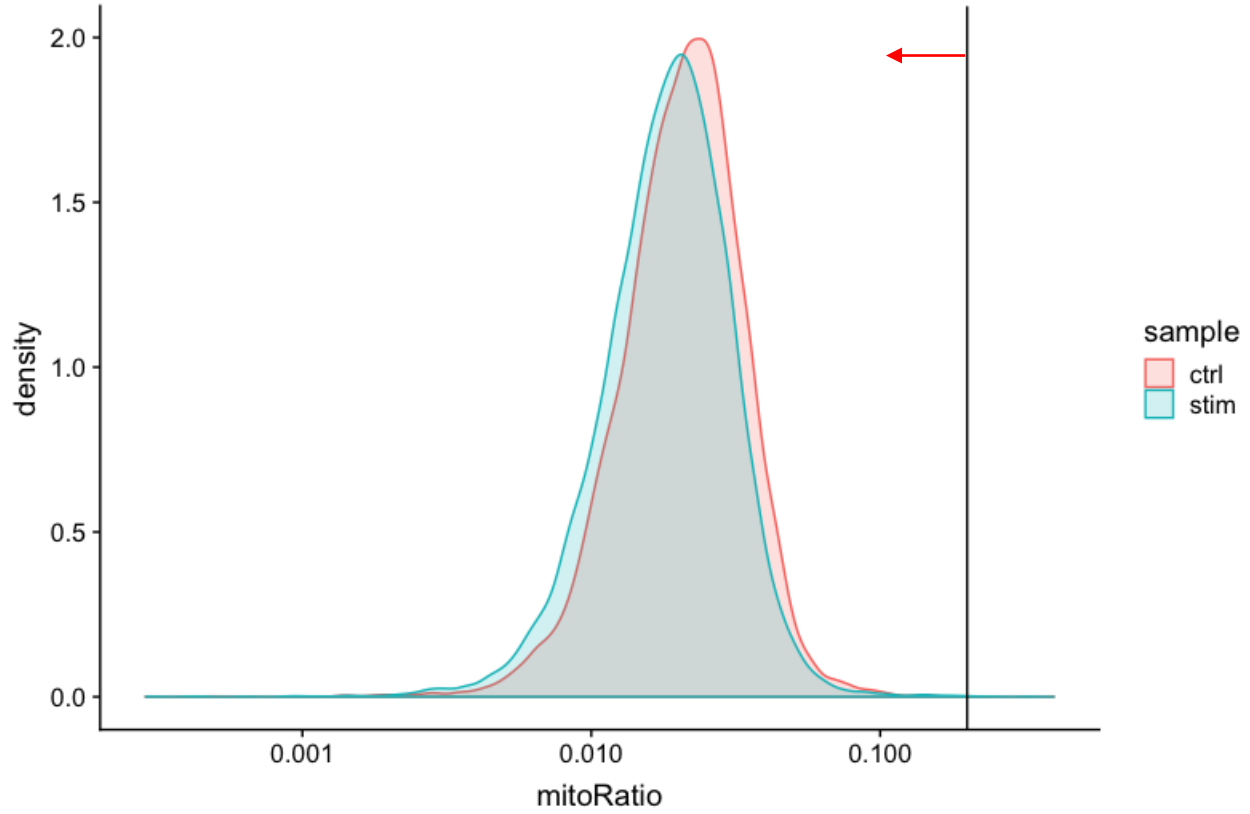
```
# Visualize the overall complexity of the gene expression by visualizing the genes detected per UMI (novelty score)
metadata %>%
        ggplot(aes(x=log10GenesPerUMI, color = sample, fill=sample)) +
        geom_density(alpha = 0.2) +
        theme_classic() +
        geom_vline(xintercept = 0.8)
```



Assessing the quality metrics (Mitochondrial counts ratio)

- This metric can identify whether there is a large amount of mitochondrial contamination from dead or dying cells. We define poor quality samples for mitochondrial counts as cells which surpass the 0.2 mitochondrial ratio mark, unless of course you are expecting this in your sample.

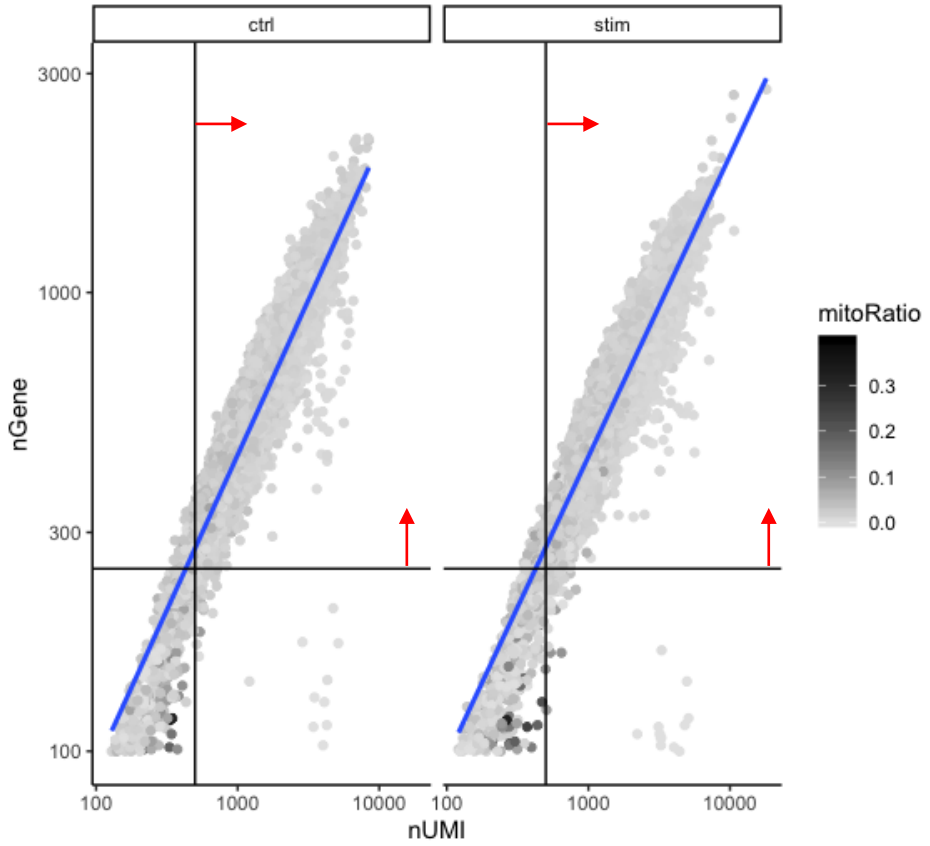
```
# Visualize the distribution of mitochondrial gene expression detected per cell
metadata %>%
        ggplot(aes(color=sample, x=mitoRatio, fill=sample)) +
        geom_density(alpha = 0.2) +
        scale_x_log10() +
        theme_classic() +
        geom_vline(xintercept = 0.2)
```



Joint filtering effects

- Considering any of these QC metrics in isolation can lead to misinterpretation of cellular signals. For example, cells with a comparatively high fraction of mitochondrial counts may be involved in respiratory processes and may be cells that you would like to keep. Likewise, other metrics can have other biological interpretations. A general rule of thumb when performing QC is to set thresholds for individual metrics to be as permissive as possible, and always consider the joint effects of these metrics. In this way, you reduce the risk of filtering out any viable cell populations.

```
# Visualize the correlation between genes detected and number of UMIs and
determine whether strong presence of cells with low numbers of genes/UMIs
metadata %>%
        ggplot(aes(x=nUMI, y=nGene, color=mitoRatio)) +
        geom_point() +
        scale_colour_gradient(low = "gray90", high = "black") +
        stat_smooth(method=lm) +
        scale_x_log10() +
        scale_y_log10() +
        theme_classic() +
        geom_vline(xintercept = 500) +
        geom_hline(yintercept = 250) +
        facet_wrap(~sample)
```



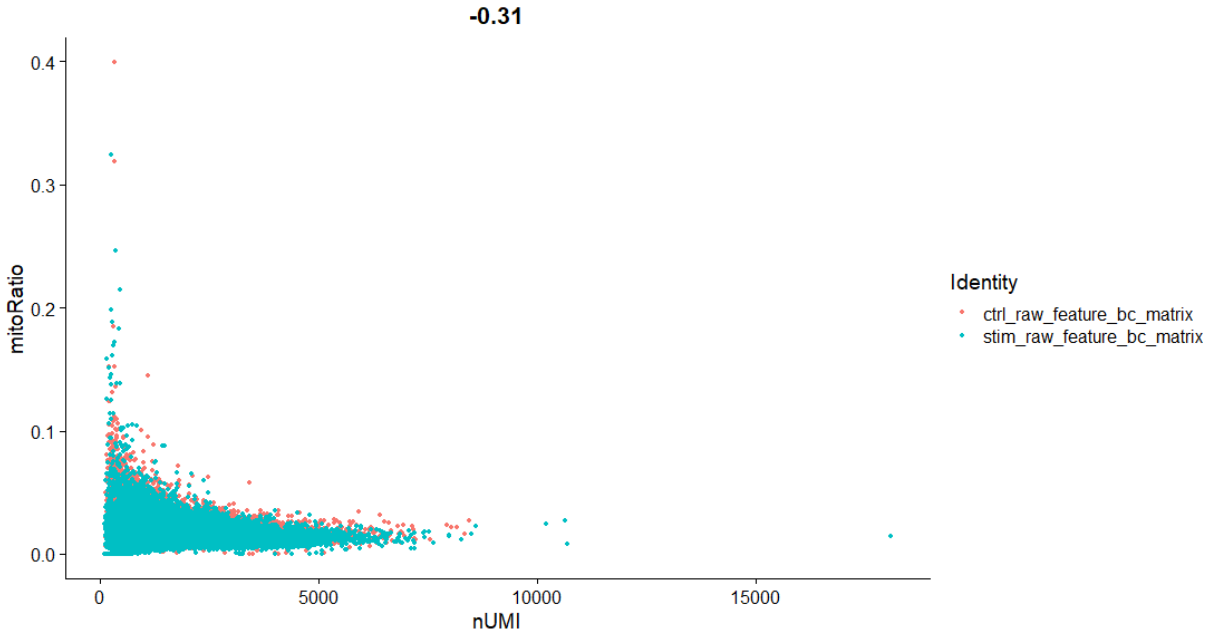
Cells that are **poor quality are likely to have low genes and UMIs per cell**, and correspond to the data points in the bottom left quadrant of the plot. With this plot we also evaluate the **slope of the line**, and any scatter of data points in the **bottom right hand quadrant** of the plot. These cells have a high number of UMIs but only a few number of genes. These could be dying cells, but also could represent a population of a low complexity cell type (i.e red blood cells).

Mitochondrial read fractions are only high in particularly low count cells with few detected genes (darker colored data points). This could be indicative of damaged/dying cells whose cytoplasmic mRNA has leaked out through a broken membrane, and thus, only mRNA located in the mitochondria is still conserved. We can see from the plot, that these cells are filtered out by our count and gene number thresholds.

Other figures

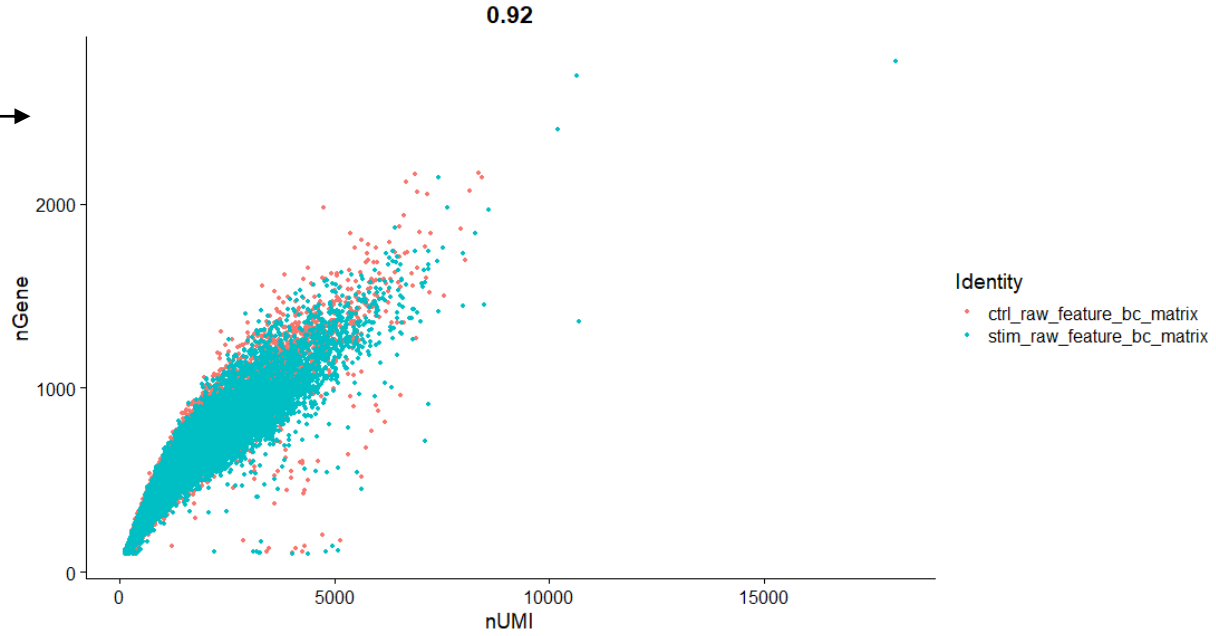
- mitoRatio compared to nUMI

```
plot1 <- FeatureScatter(merged_seurat, feature1 = “nUMI", feature2 = "mitoRatio",raster=FALSE)

print(plot1)
```



- nGene compared to nUMI

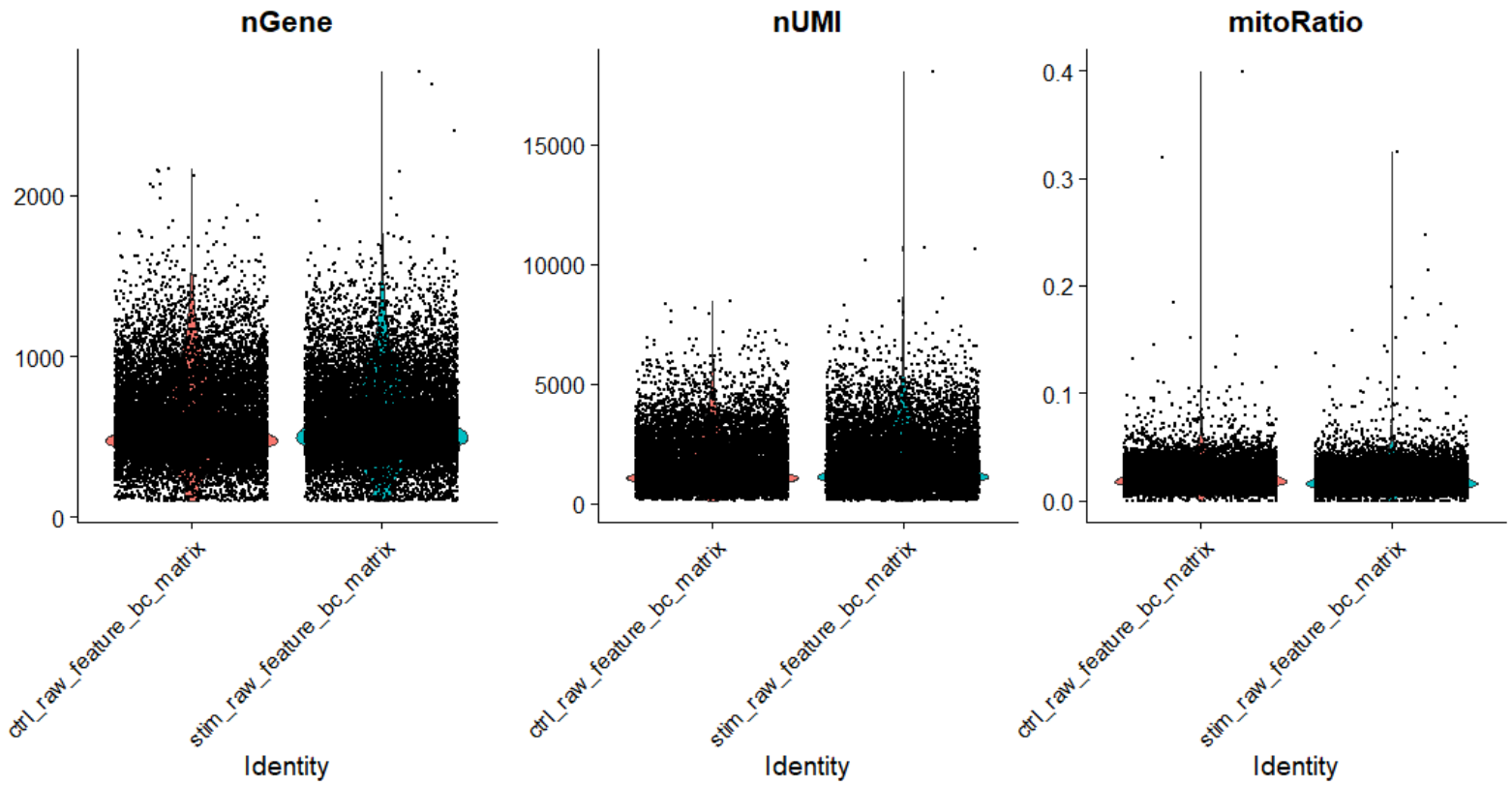
```
plot2 <- FeatureScatter(merged_seurat, feature1 = "nUMI", feature2 = "nGene", raster=FALSE)

print(plot2)
```



Other figures (continued)

- nGene, nUMI and mitoRatio

print(VlnPlot(merged_seurat, features = c("nGene", "nUMI", "mitoRatio"), ncol = 3))



Filtering

Cell-level filtering

- Now that we have visualized the various metrics, we can decide on the thresholds to apply which will result in the removal of low quality cells. Often the recommendations mentioned earlier are a rough guideline, and the specific experiment needs to inform the exact thresholds chosen. We will use the following thresholds:

  - nUMI > 500
  - nGene > 250
  - log10GenesPerUMI > 0.8
  - mitoRatio < 0.2

****Depends on the experiment****

```
# Filter out low quality cells using selected thresholds - these will change with experiment
filtered_seurat <- subset(x = merged_seurat,
                          subset= (nUMI >= 500) &
                            (nGene >= 250) &
                            (log10GenesPerUMI > 0.80) &
                            (mitoRatio < 0.20))
```

Gene-level filtering

- Within our data we will have many genes with zero counts. These genes can dramatically reduce the average expression for a cell and so we will remove them from our data. We will start by identifying which genes have a zero count in each cell:

```
# Extract counts
counts <- GetAssayData(object = filtered_seurat, slot = "counts")
# Output a logical matrix specifying for each gene on whether or not there are more than zero counts per cell
nonzero <- counts > 0
```

- For our data we choose to keep only genes which are expressed in 10 or more cells. By using this filter, genes which have zero counts in all cells will effectively be removed.

```
# Sums all TRUE values and returns TRUE if more than 10 TRUE values per gene
keep_genes <- Matrix::rowSums(nonzero) >= 10

# Only keeping those genes expressed in more than 10 cells
filtered_counts <- counts[keep_genes, ]
```

Checking the quality after filtering

- Finally, take those filtered counts and create a new Seurat object for downstream analysis.

```
# Reassign to filtered Seurat object

filtered_seurat <- CreateSeuratObject(filtered_counts, meta.data = filtered_seurat@meta.data)

# Save filtered subset to new metadata

metadata_clean <- filtered_seurat@meta.data
```

After performing the filtering, it's recommended to look back over the metrics to make sure that your data matches your expectations and is good for downstream analysis.

↳ Rerun all the plots with filtered_seurat instead of merged_seurat and metadata_clean instead of metadata

When everything is looking good

- Based on these QC metrics we would identify any failed samples and move forward with our filtered cells. Often, we iterate through the QC metrics using different filtering criteria; it is not necessarily a linear process. When satisfied with the filtering criteria, we would save our filtered cell object for clustering and marker identification.

```
# Create .RData object to load at any time
save(filtered_seurat, file="data/seurat_filtered.RData")
```

To explore “bad” quality data: https://github.com/hbctraining/scRNA-seq_online/blob/master/lessons/QC_bad_data.md

Count normalization

Count normalization

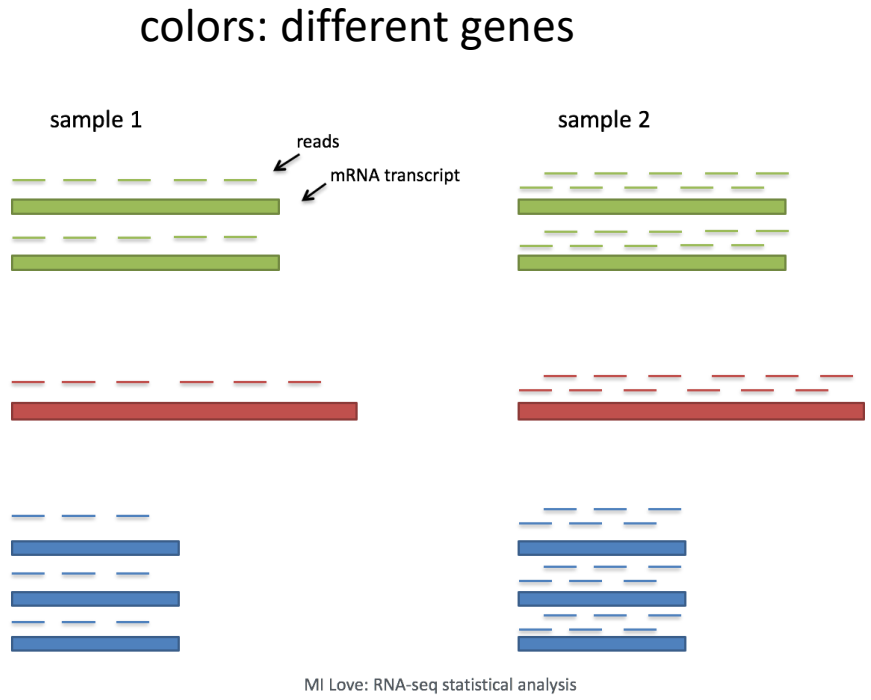
- Is essential to make accurate comparisons of gene expression between cells (or samples).
- The counts of mapped reads for each gene is proportional to the expression of RNA ("interesting") in addition to many other factors ("uninteresting"). Normalization is the process of scaling raw count values to account for the "uninteresting" factors. In this way the expression levels are more comparable between and/or within cells.

- The main factors often considered during normalization are:

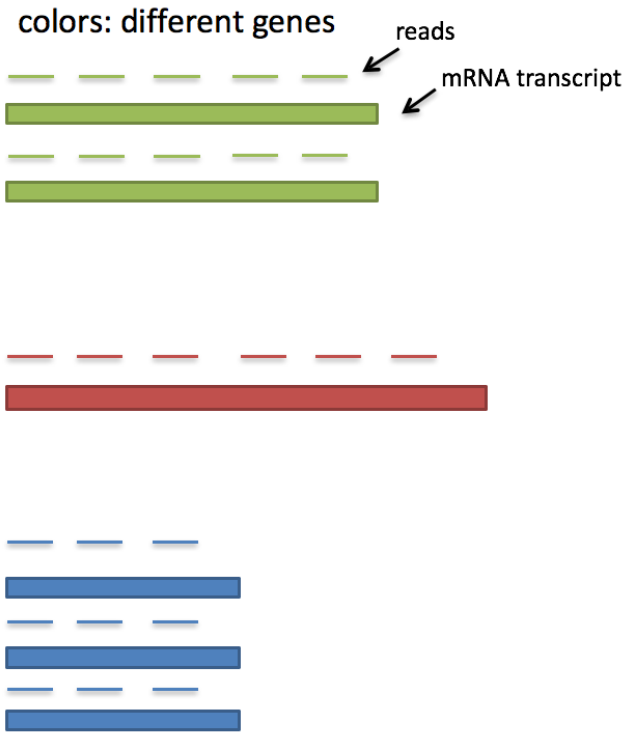
1. Sequencing depth

- Accounting for sequencing depth is necessary for **comparison of gene expression between cells**. In the example below, each gene appears to have doubled in expression in sample 2, however this is a consequence of sample 2 having twice the sequencing depth.

- Each cell in scRNA-seq will have a differing number of reads associated with it. So to accurately compare expression between cells, it is necessary to normalize for sequencing depth.



2. Gene length



Slide adapted from MI Love: RNA-seq statistical analysis

- Accounting for gene length is necessary for **comparing expression between different genes within the same cell**. The number of reads mapped to a longer gene can appear to have equal count/expression as a shorter gene that is more highly expressed.
- In scRNA-seq analysis, we will be comparing the expression of different genes within the cells to cluster the cells. If using a **3' or 5' droplet-based method (e.g. 10X Genomics), the length of the gene will not affect the analysis** because only the 5' or 3' end of the transcript is sequenced. However, **if using full-length sequencing, the transcript length should be accounted for** (e.g. Smart-seq).

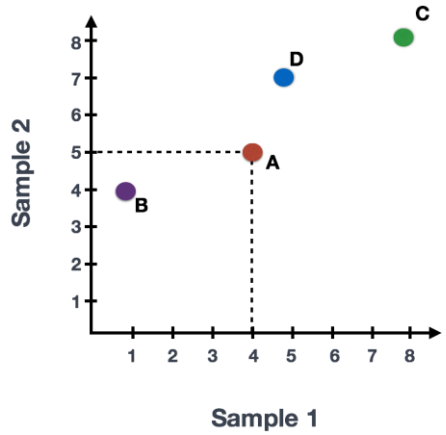
Principal Component Analysis (PCA)

What is PCA?

- Principal Component Analysis (PCA) is a technique used to emphasize variation as well as similarity, and to bring out strong patterns in a dataset; it is one of the methods used for "dimensionality reduction".
- For full explanation, see video by StatQuests and Josh Starmer:
  - https://www.youtube.com/watch?v=FgakZw6K1QQ

Basic explanation with a simple example

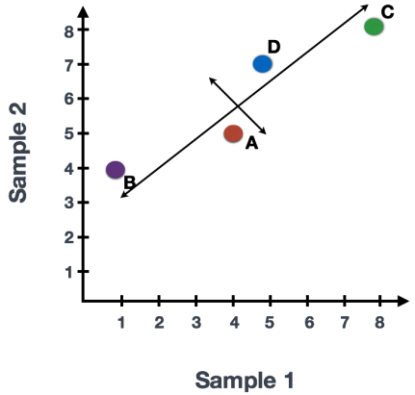
Let's say you had quantified the expression of four genes in **two samples (or cells)**, you could plot the expression values of those genes with one sample represented on the x-axis and the other sample on the y-axis as shown below:



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

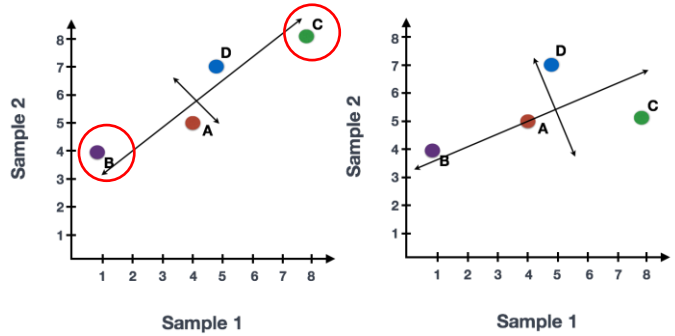
You could draw a line through the data in the direction representing the most variation, which is on the **diagonal** in this example. The maximum variation in the dataset is between the genes that make up the two endpoints of this line.

We also see the genes vary somewhat above and below the line. We could draw another line through the data representing the second most amount of variation in the data, since this plot is in 2D (2 axes).

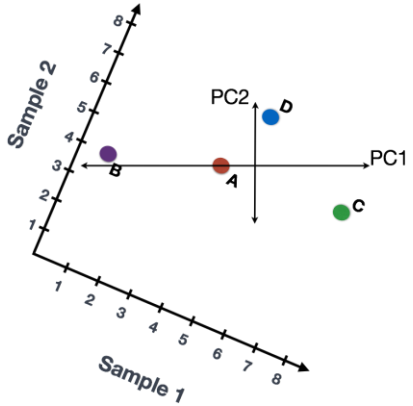


The genes near the ends of each line would be those with the highest variation; these genes have the greatest influence on the direction of the line, mathematically.

For example, a small change in the value of Gene C would greatly change the direction of the longer line, whereas a small change in Gene A or Gene D would have little affect on it.



We could also rotate the entire plot and view the lines representing the variation as left-to-right and up-and-down. We see most of the variation in the data is left-to-right (longer line) and the second most variation in the data is up-and-down (shorter line). You can now think of these lines as the axes that represent the variation. These axes are essentially the "Principal Components", with PC1 representing the most variation in the data and PC2 representing the second most variation in the data.

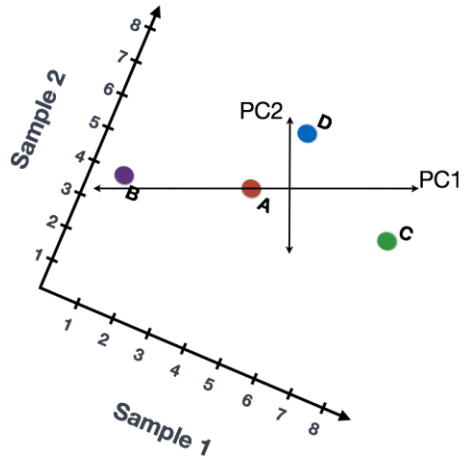


Now, what if we had three samples/cells, then we would have an extra direction in which we could have variation (3D). Therefore, if we have N samples/cells we would have N-directions of variation or N principal components (PCs)! Once these PCs have been calculated, the PC that deals with the largest variation in the dataset is designated PC1, and the next one is designated PC2 and so on.

Once the PCs have been determined for a dataset, we have to figure out how each sample/cell fits back into that context to enable us to visualize the similarities/dissimilarities in an intuitive manner. The question here is "what is sample_X's score for a given PC based on the gene expression in sample_X?". This is the actual step where the dimenionality is reduced, since you plot PC scores for each sample/cell on the final PCA plot.

PC scores are calculated for all sample-PC pairs as described in the steps and schematic below:

(1) First, each gene is assigned an "influence" score based on how much it influenced each PC. Genes that did not have any influence on a given PC get scores near zero, while genes with more influence receive larger scores. Genes on the ends of a PC line will have a larger influence, so they would receive larger scores but with opposite signs.



	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

(2) Once the influence has been determined, the score for each sample is calculated using the following equation:

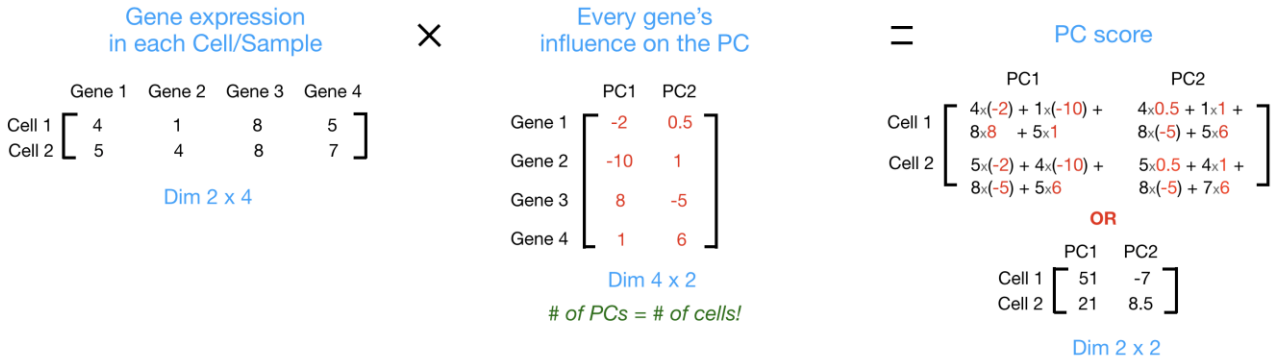
Sample1 PC1 score = (read count * influence) + ... for all genes

For our 2-sample example, the following is how the scores would be calculated:

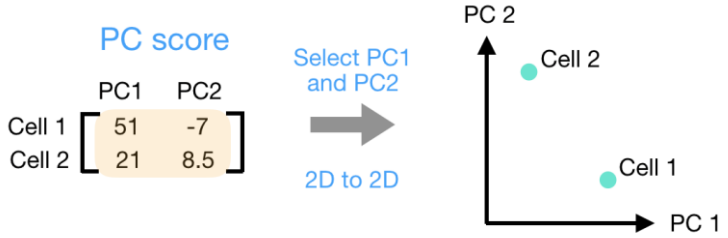
Sample1
PC1 score = (4 * -2) + (1 * -10) + (8 * 8) + (5 * 1) = 51
PC2 score = (4 * 0.5) + (1 * 1) + (8 * -5) + (5 * 6) = -7

Sample2
PC1 score = (5 * -2) + (4 * -10) + (8 * 8) + (7 * 1) = 21
PC2 score = (5 * 0.5) + (4 * 1) + (8 * -5) + (7 * 6) = 8.5

Here is a schematic that goes over the first 2 steps:

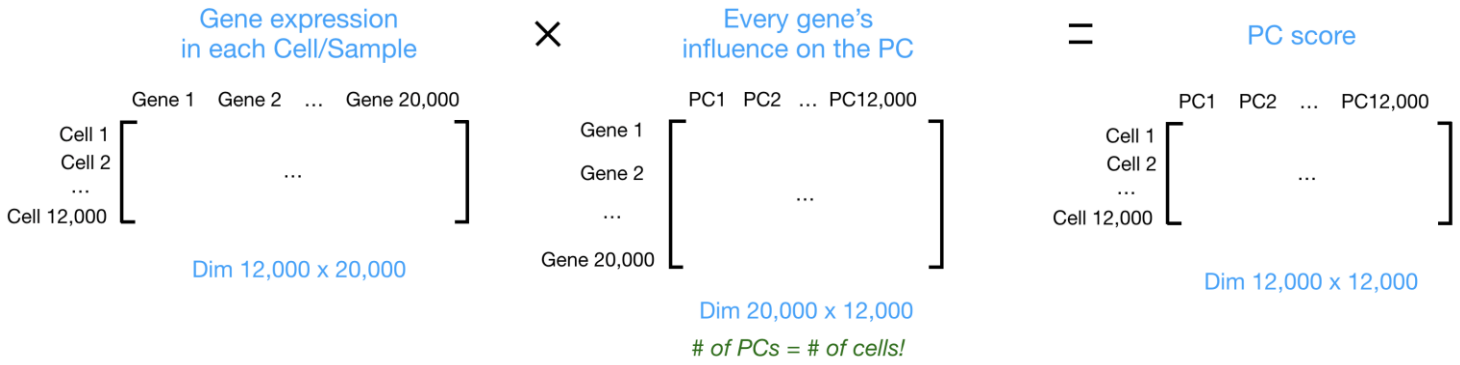


(3) Once these scores are calculated for all the PCs, they can be plotted on a simple scatter plot. Below is the plot for the example here, going from the 2D matrix to a 2D plot:

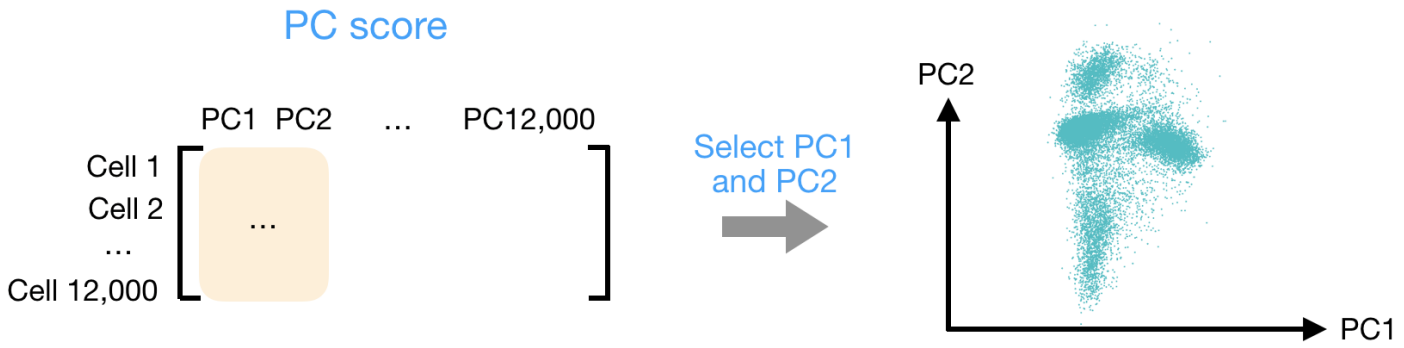


Example with scRNA-seq data

- Let's say you are working with a single-cell RNA-seq dataset with 12,000 cells and you have quantified the expression of 20,000 genes.

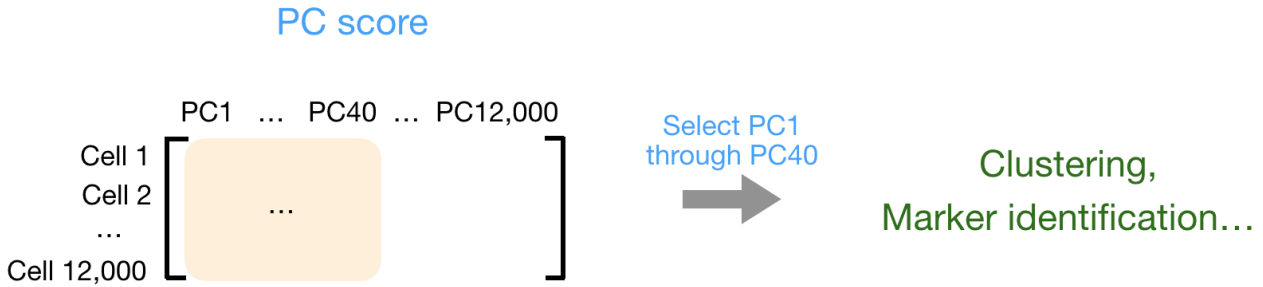


- After the PC scores have been calculated, you are looking at a matrix of 12,000 x 12,000 that represents the information about relative gene expression in all the cells. You can select the PC1 and PC2 columns and plot that in a 2D way.

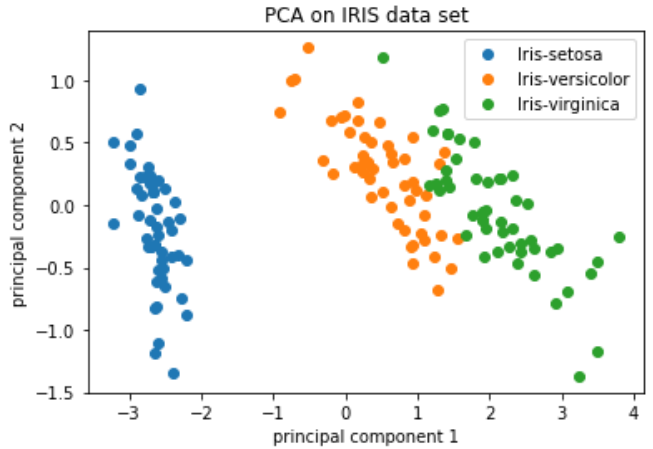


Example with scRNA-seq data (continued)

- You can also use the PC scores from the first 40 PCs for downstream analysis like clustering, marker identification etc., since these represent the majority of the variation in the data. We will be talking a lot more about this later in this workshop.



Note: For datasets with a larger number of samples or cells, only the PC1 and PC2 scores for each sample/cell are usually plotted or used for visualization. Since these PCs explain the most variation in the dataset, the expectation is that the samples/cells that are more similar to each other will cluster together with PC1 and PC2. See a real-life example below:



Back to RStudio!

To make a new script

Creating new script and loading data

- Click on “+ New Blank File” and create the SCT_integration_analysis.R file
- Load the libraries:

```
library(Seurat)
library(tidyverse)
library(RCurl)
library(cowplot)
```

- Load the data:

```
load("data/seurat_filtered.RData")
```

Normalize the counts

- The most common biological data correction is to remove the effects of the cell cycle on the transcriptome. This data correction can be performed by a simple linear regression against a cell cycle score which is what we will demonstrate below.
- The raw counts are not comparable between cells and we can't use them as is for our exploratory analysis. So we will perform a rough normalization by dividing by total counts per cell and taking the natural log. This normalization is solely for the purpose of exploring the sources of variation in our data.

Normalize the counts with Seurat

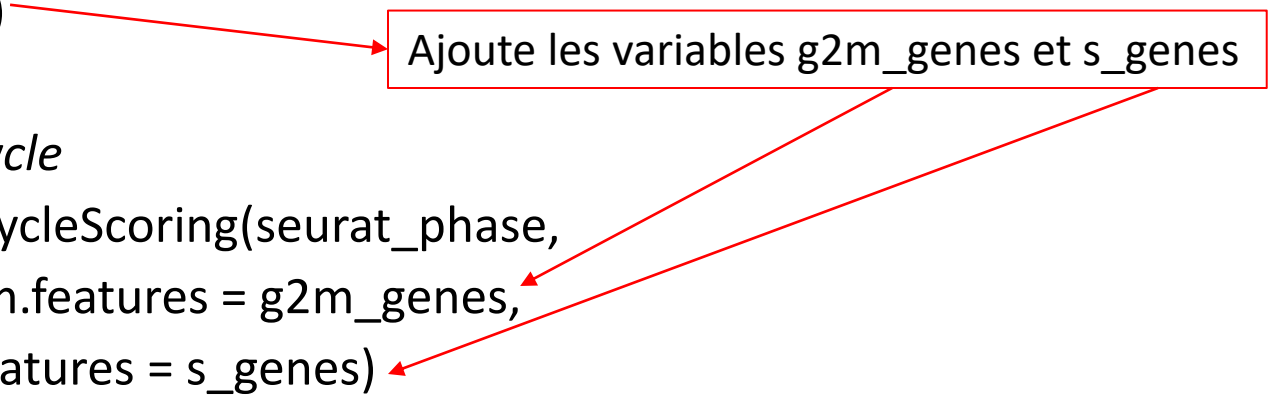
- Seurat recently introduced a new normalization method called sctransform, which simultaneously performs variance stabilization and regresses out unwanted variation. This is the normalization method that we are implementing in our workflow.

```
# Normalize the counts
seurat_phase <- NormalizeData(filtered_seurat)
```

Evaluating effects of cell cycle

- To assign each cell a score based on its expression of G2/M and S phase markers, we can use the Seuart function CellCycleScoring(). This function calculates cell cycle phase scores based on canonical markers that required as input.

```
# Load cell cycle markers
load("data/cycle.rda")

# Score cells for cell cycle
seurat_phase <- CellCycleScoring(seurat_phase,
                                 g2m.features = g2m_genes,
                                 s.features = s_genes)

# View cell cycle scores and phases assigned to cells
View(seurat_phase@meta.data)
```



*** if you are not working with human data, we have additional materials detailing how to acquire cell cycle markers for other organisms of interest ***

Evaluating effects of cell cycle (continued)

After scoring the cells for cell cycle, we would like to determine whether cell cycle is a major source of variation in our dataset using PCA. To perform PCA, we need to first choose the most variable features, then scale the data. Since highly expressed genes exhibit the highest amount of variation and we don't want our 'highly variable genes' only to reflect high expression, we need to scale the data to scale variation with expression level. The Seurat ScaleData() function will scale the data by:

- adjusting the expression of each gene to give a mean expression across cells to be 0
- scaling expression of each gene to give a variance across cells to be 1

```
# Identify the most variable genes
seurat_phase <- FindVariableFeatures(seurat_phase,
                selection.method = "vst",
                nfeatures = 2000,
                verbose = FALSE)

# Scale the counts
seurat_phase <- ScaleData(seurat_phase)
```

NOTE: For the selection.method and nfeatures arguments the values specified are the default settings. Therefore, you do not necessarily need to include these in your code. We have included it here for transparency and inform you what you are using.

Cell cycle PCA

 # Perform PCA

seurat_phase <- RunPCA(seurat_phase)

 # Plot the PCA colored by cell cycle phase

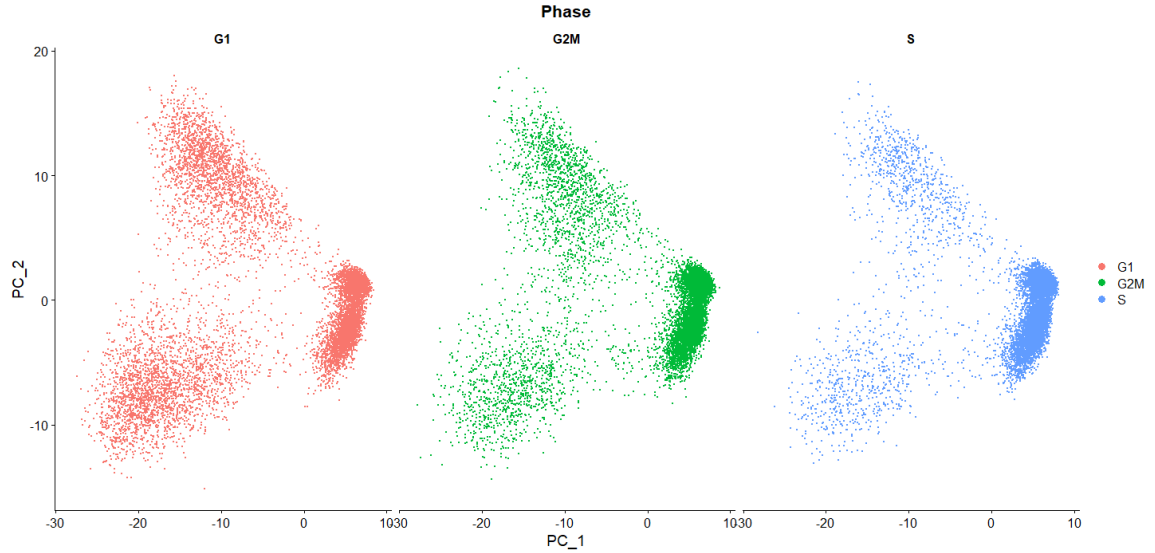
DimPlot(seurat_phase,

reduction = "pca",

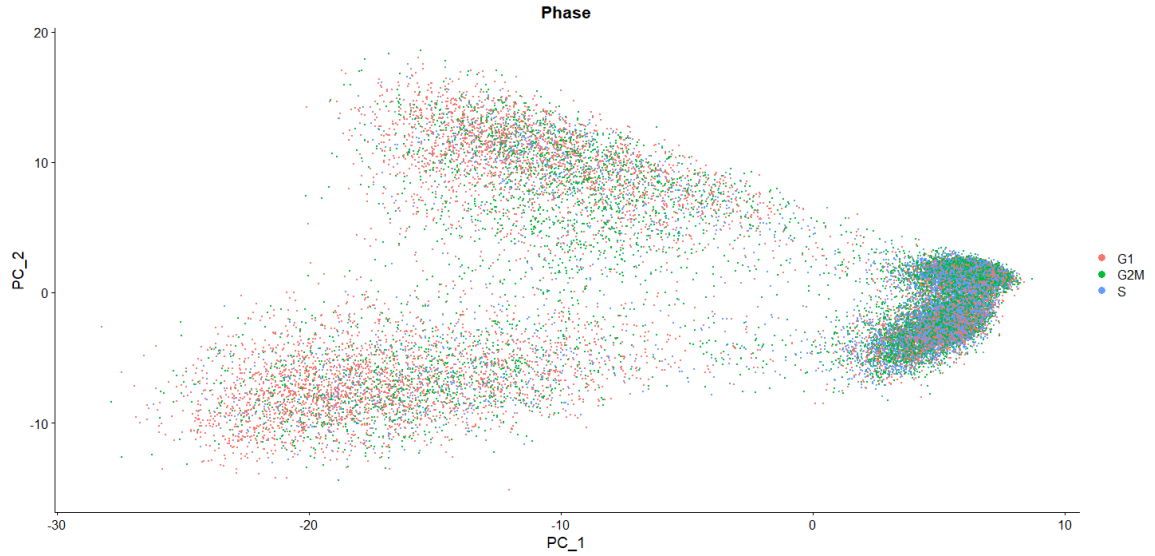
group.by= "Phase",

split.by = "Phase")

With split.by = “Phase”



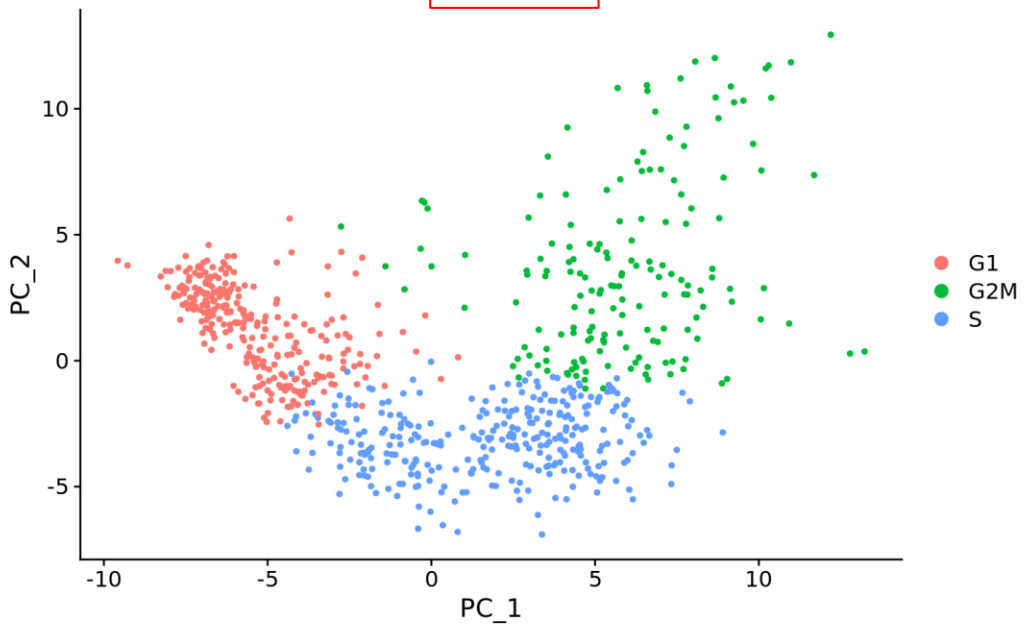
Without split.by = “Phase”



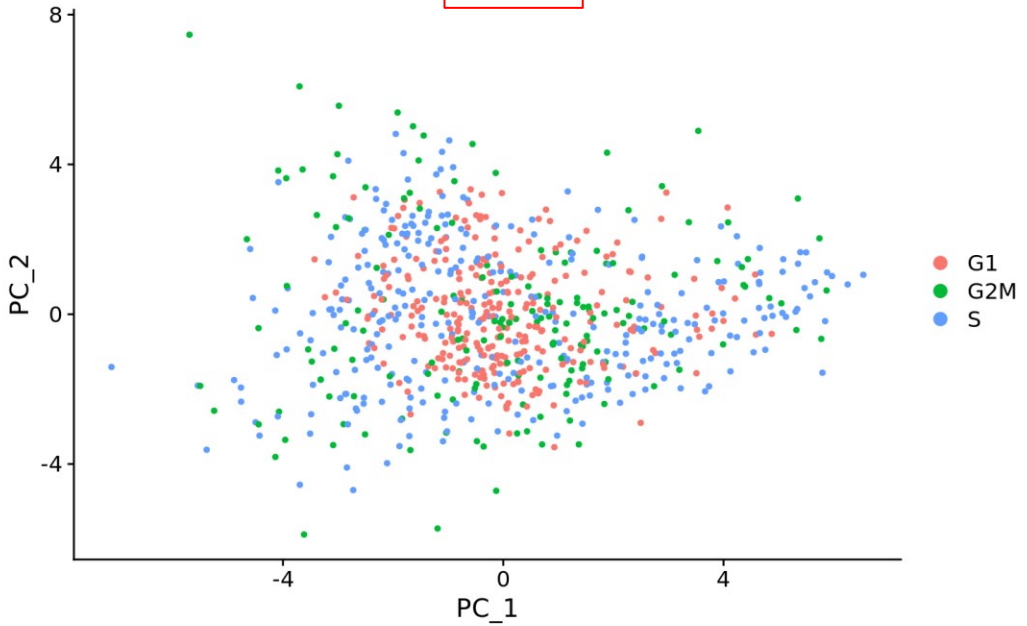
We do not see large differences due to cell cycle phase. Based on this plot, <u>we would not regress out the variation due to cell cycle</u>.

Example when cell cycle should be regressed out

Before



After



Regression

marrow <- ScaleData(marrow, **vars.to.regress = c("S.Score", "G2M.Score")**, features = rownames(marrow))

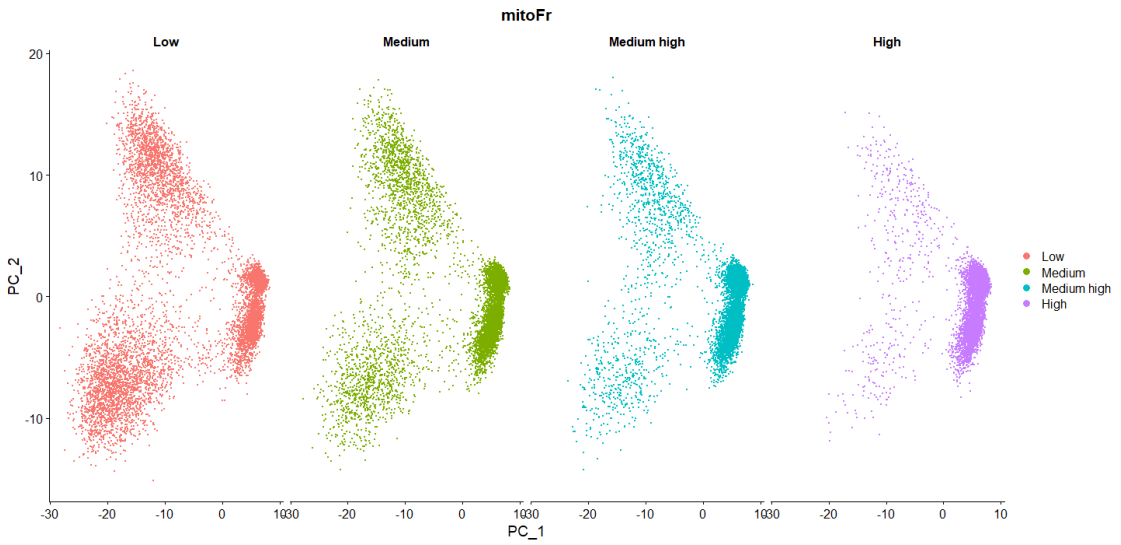
Evaluating effects of mitochodrial expression

- Mitochondrial expression is another factor which can greatly influence clustering. Oftentimes, it is useful to regress out variation due to mitochondrial expression. However, if the differences in mitochondrial gene expression represent a biological phenomenon that may help to distinguish cell clusters, then we advise not regressing this out.

1. First, turn the mitochondrial ratio variable into a new categorical variable based on quartiles (using the code below):

```
# Check quartile values
summary(seurat_phase@meta.data$mitoRatio)
# Turn mitoRatio into categorical factor vector based on quartile values
seurat_phase@meta.data$mitoFr <- cut(seurat_phase@meta.data$mitoRatio,
                breaks=c(-Inf, 0.0144, 0.0199, 0.0267, Inf),
                labels=c("Low","Medium","Medium high", "High"))
```

1. Next, plot the PCA similar to how we did with cell cycle regression.
   - Use the new **mitoFr** variable to split cells and color them accordingly.
2. Evaluate the PCA plot generated in #2.
   - Determine whether or not you observe an effect.
   - Describe what you see.
   - Would you regress out mitochndrial fraction as a source of unwanted variation?



SCTransform

- Now we can use the sctransform method as a more accurate method of normalizing, estimating the variance of the raw filtered data, and identifying the most variable genes. The sctransform method models the UMI counts using a regularized negative binomial model to remove the variation due to sequencing depth (total nUMIs per cell), while adjusting the variance based on pooling information across genes with similar abundances (similar to some bulk RNA-seq methods).
- Sctransform automatically accounts for cellular sequencing depth by regressing out sequencing depth (nUMIs). However, if there are other sources of uninteresting variation identified in the data during the exploration steps, we can also include these. We observed little to no effect due to cell cycle phase and so we chose not to regress this out of our data. **We observed some effect of mitochondrial expression and so we choose to regress this out from the data.**

Separating the conditions

- Since we have two samples in our dataset (from two conditions), we want to keep them as separate objects and transform them as that is what is required for integration. We will first split the cells in seurat_phase object into "Control" and "Stimulated":

 # Split seurat object by condition to perform cell cycle scoring and SCT on all samples

```
split_seurat <- SplitObject(seurat_phase, split.by = "sample")


split_seurat <- split_seurat[c("ctrl", "stim")]
```

SCTransform on all samples

- NOTE: Before we run this for loop, we know that the output can generate large R objects/variables in terms of memory. If we have a large dataset, then we might need to adjust the limit for allowable object sizes within R (Default is 500 * 1024 ^ 2 = 500 Mb) using the following code: options(future.globals.maxSize = 4000 * 1024^2)

```
for (i in 1:length(split_seurat)) {
    split_seurat[[i]] <- SCTransform(split_seurat[[i]], vars.to.regress = c("mitoRatio"))
}
```

- NOTE: By default, after normalizing, adjusting the variance, and regressing out uninteresting sources of variation, SCTransform will rank the genes by residual variance and output the 3000 most variant genes. If the dataset has larger cell numbers, then it may be beneficial to adjust this parameter higher using the variable.features.n argument.

Assays

 # Check which assays are stored in objects

split_seurat$ctrl@assays

Exercise:

1. Are the same assays available for the "stim" samples within the split_seurat object? What is the code you used to check that?
2. Any observations for the genes or features listed under "First 10 features:" and the "Top 10 variable features:" for "ctrl" versus "stim"?

Saving the split_seurat object

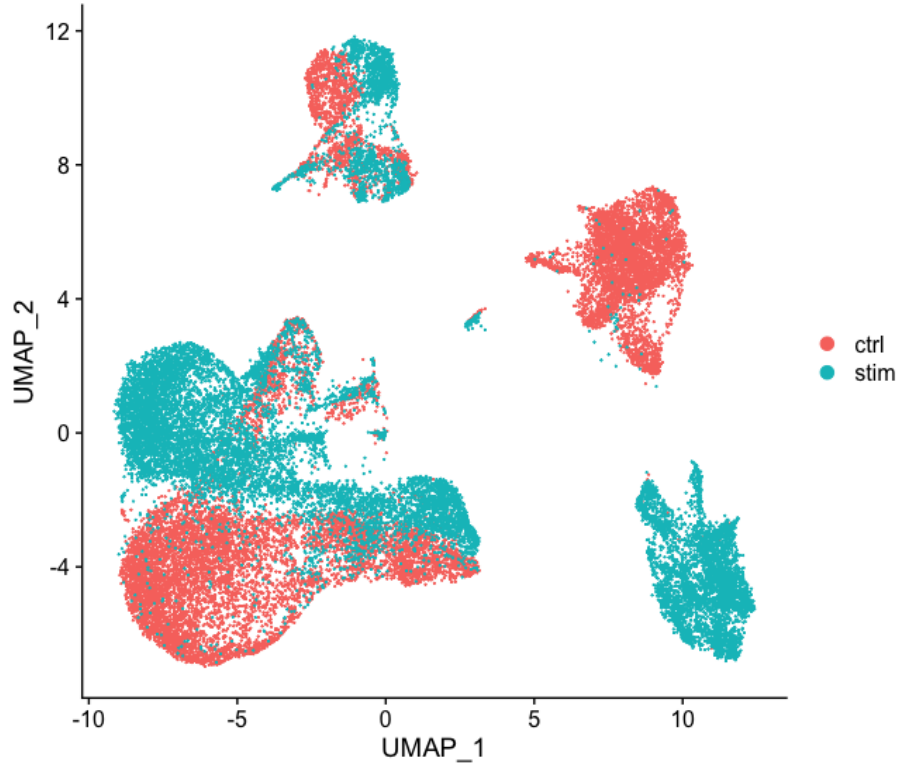
- It can take a while to get back to this stage especially when working with large datasets, it is best practice to save the object as an easily loadable file locally.

```
# Save the split seurat object
saveRDS(split_seurat, "data/split_seurat.rds")

# Load the split seurat object into the environment
split_seurat <- readRDS("data/split_seurat.rds")
```

Integration

To integrate or not to integrate?

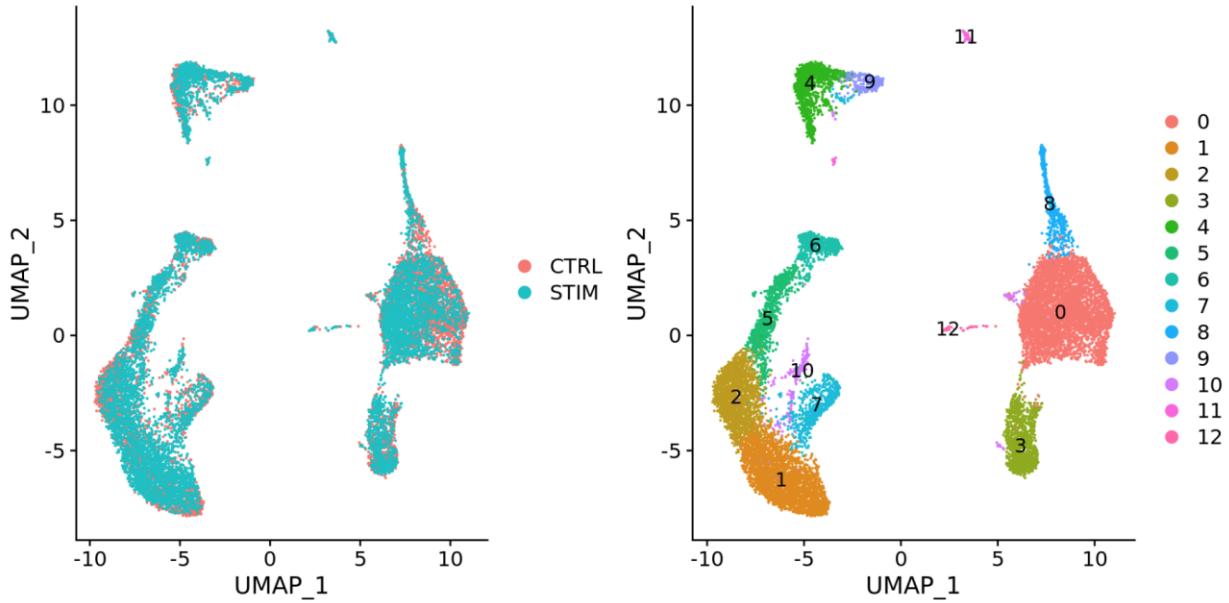
- Generally, we always **look at our clustering without integration** before deciding whether we need to perform any alignment. Do not just always perform integration because you think there might be differences - explore the data. If we had performed the normalization on both conditions together in a Seurat object and visualized the similarity between cells, we would have seen condition-specific clustering:



Condition-specific clustering of the cells indicates that we need to integrate the cells across conditions to ensure that cells of the same cell type cluster together.

Importance of clustering

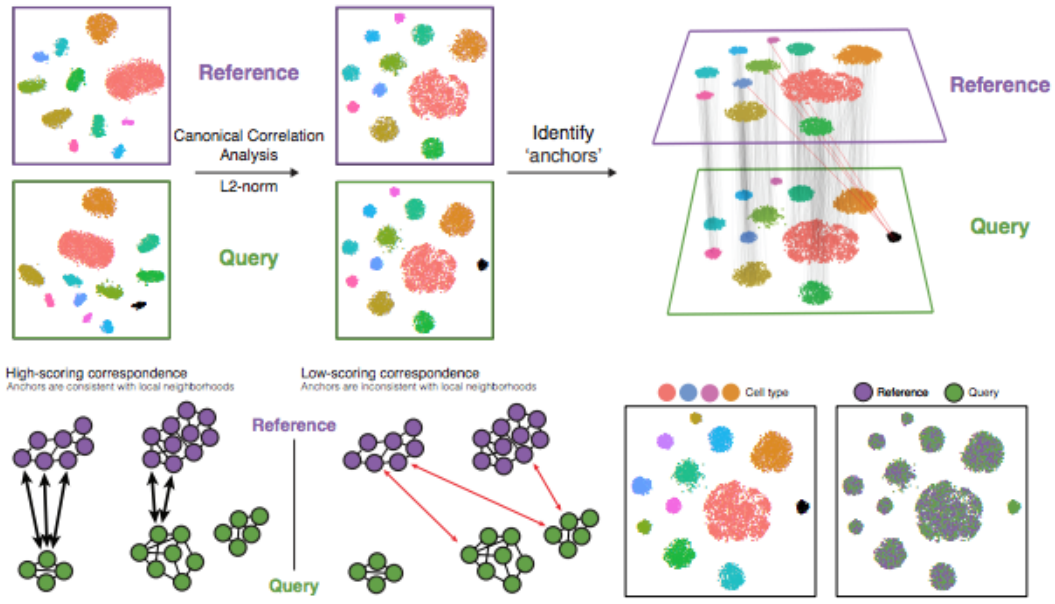
- We want to identify cell types which are present in all samples/conditions/modalities within our dataset, and therefore would like to observe a representation of cells from both samples/conditions/modalities in every cluster. This will enable more interpretable results downstream (i.e. DE analysis, ligand-receptor analysis.
- Integrating = "harmonizing" the groups to **overlay cells that are similar or have a "common set of biological features" between groups**. For example, we could integrate across:
- Example for different conditions:



When integration is not needed

- Seurat has a vignette for how to run through the workflow **without integration**. The workflow is fairly similar to this workflow, but the samples would not necessarily be split in the beginning and integration would not be performed.
- **It can help to first run conditions individually if unsure what clusters to expect** or expecting some different cell types between conditions (e.g. tumor and control samples), then run them together to see whether there are condition-specific clusters for cell types present in both conditions. Oftentimes, when clustering cells from multiple conditions there are condition-specific clusters and integration can help ensure the same cell types cluster together.

Integration steps



Stuart T and Butler A, et al. Comprehensive integration of single cell data, bioRxiv 2018. https://doi.org/10.1101/460147

1. Perform canonical correlation analysis (CCA):
   - CCA identifies shared sources of variation between the conditions/groups. It is a form of PCA, in that it **identifies the greatest sources of variation** in the data, but only if it is **shared or conserved** across the conditions/groups (using the 3000 most variant genes from each sample).
   - This step roughly aligns the cells using the greatest shared sources of variation.
2. Identify anchors or mutual nearest neighbors (MNNs) across datasets (sometimes incorrect anchors are identified):
   - MNNs can be thought of as 'best buddies'. For each cell in one condition:
     - The cell's closest neighbor in the other condition is identified based on gene expression values - it's 'best buddy'.
     - The reciprocal analysis is performed, and if the two cells are 'best buddies' in both directions, then those cells will be marked as anchors to 'anchor' the two datasets together.
3. Filter anchors to remove incorrect anchors:
   - Assess the similarity between anchor pairs by the overlap in their local neighborhoods (incorrect anchors will have low scores) - do the adjacent cells have 'best buddies' that are adjacent to each other?
4. Integrate the conditions/datasets:
   - Use anchors and corresponding scores to transform the cell expression values, allowing for the integration of the conditions/datasets (different samples, conditions, datasets, modalities)

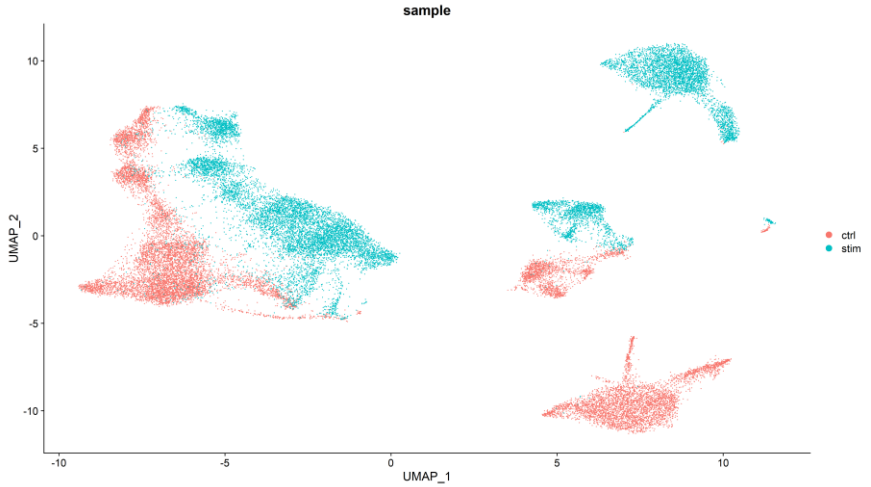
If cell types are present in one dataset, but not the other, then the cells will still appear as a separate sample-specific cluster.

Back to RStudio

For the integration

Check the plots before integration

```
# Run PCA
before_intrgration <- RunPCA(object = seurat_phase)

# Plot PCA
PCAPlot(before_intrgration, split.by = "sample")

# Run UMAP
before_intrgration <- RunUMAP(before_intrgration,
                              dims = 1:40,
                              reduction = "pca")
# Plot UMAP
DimPlot(before_intrgration, group.by = "sample")
```



Integration

- Now, using our SCTransform object as input, let's perform the integration across conditions.
- First, we need to specify that we want to use all of the 3000 most variable genes identified by SCTransform for the integration. By default, this function only selects the top 2000 genes.

```
# Select the most variable features to use for integration
integ_features <- SelectIntegrationFeatures(object.list = split_seurat,
                                            nfeatures = 3000)
```

Integration (continued)

1. **Now, we need to prepare the SCTransform object for integration.**

```
# Prepare the SCT list object for integration
split_seurat <- PrepSCTIntegration(object.list = split_seurat,
                                   anchor.features = integ_features)
```

2. **Now, we are going to perform CCA, find the best buddies or anchors and filter incorrect anchors. For our dataset, this will take up to 15 minutes to run.**

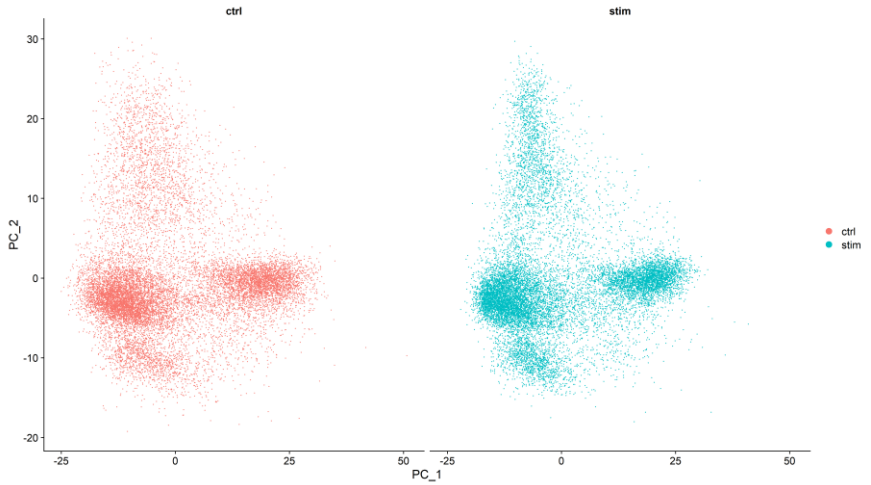
```
# Find best buddies - can take a while to run
integ_anchors <- FindIntegrationAnchors(object.list = split_seurat,
                                        normalization.method = "SCT",
                                        anchor.features = integ_features)
```

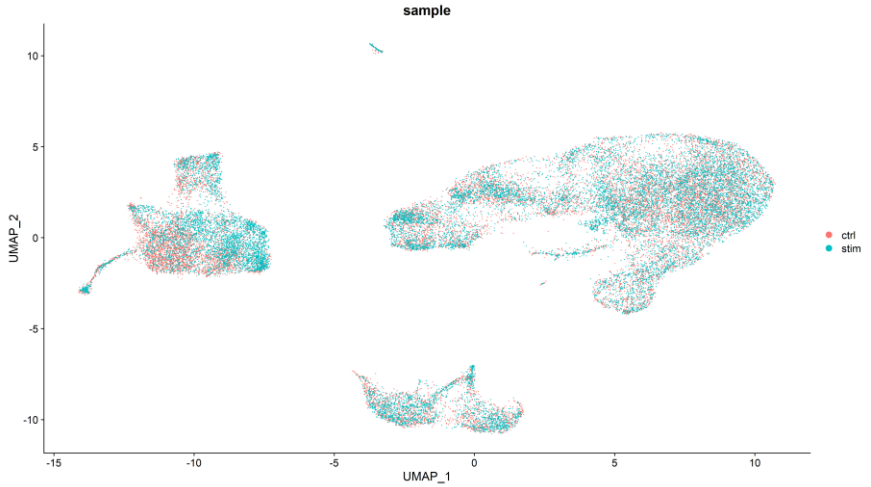
3. **Finally, we can integrate across conditions.**

```
# Integrate across conditions
seurat_integrated <- IntegrateData(anchorset = integ_anchors,
                                   normalization.method = "SCT")
```

Recheck the plots

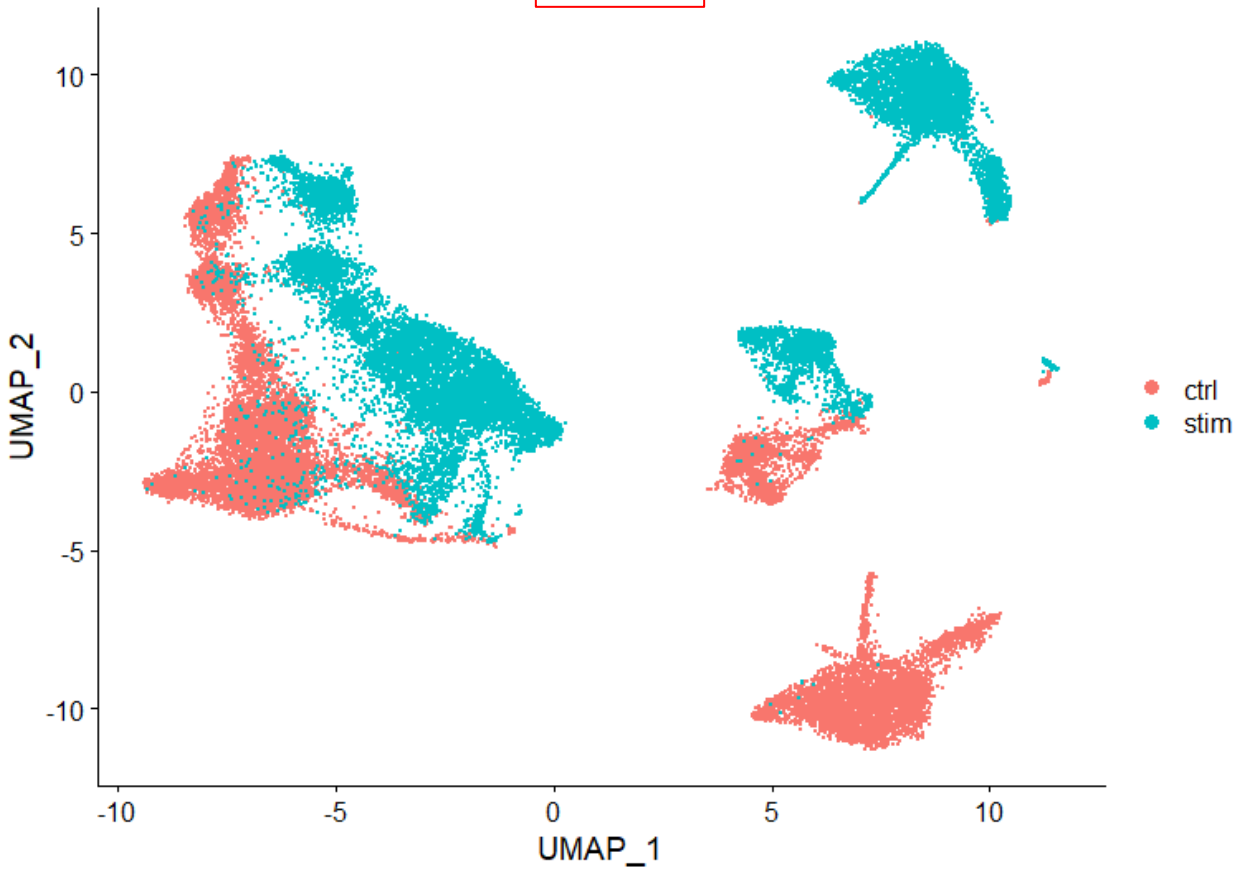
```
# Run PCA
seurat_integrated <- RunPCA(object = seurat_integrated)

# Plot PCA
PCAPlot(seurat_integrated, split.by = "sample")

# Run UMAP
seurat_integrated <- RunUMAP(seurat_integrated,
                             dims = 1:40,
                             reduction = "pca")

# Plot UMAP
DimPlot(seurat_integrated, group.by = "sample")
```



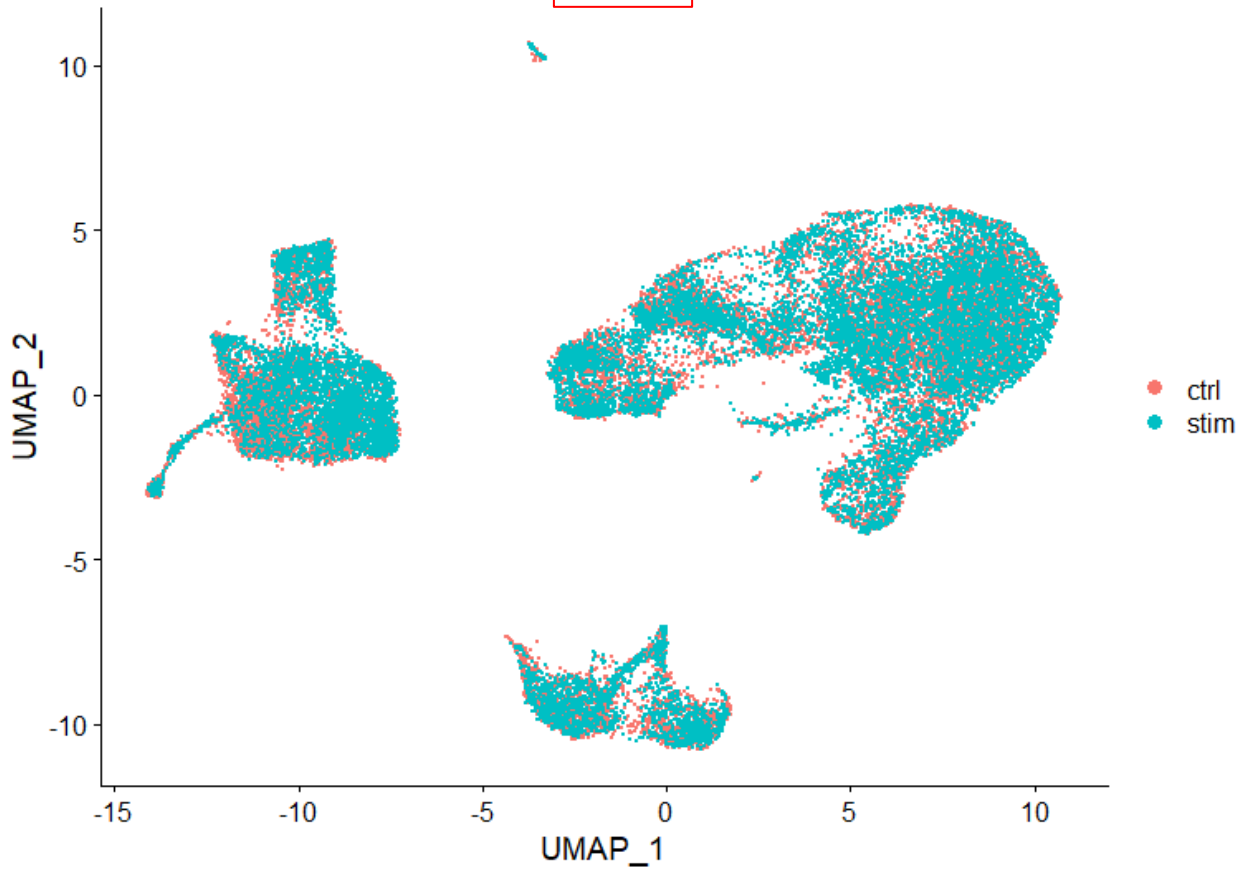


Comparison

Before



After

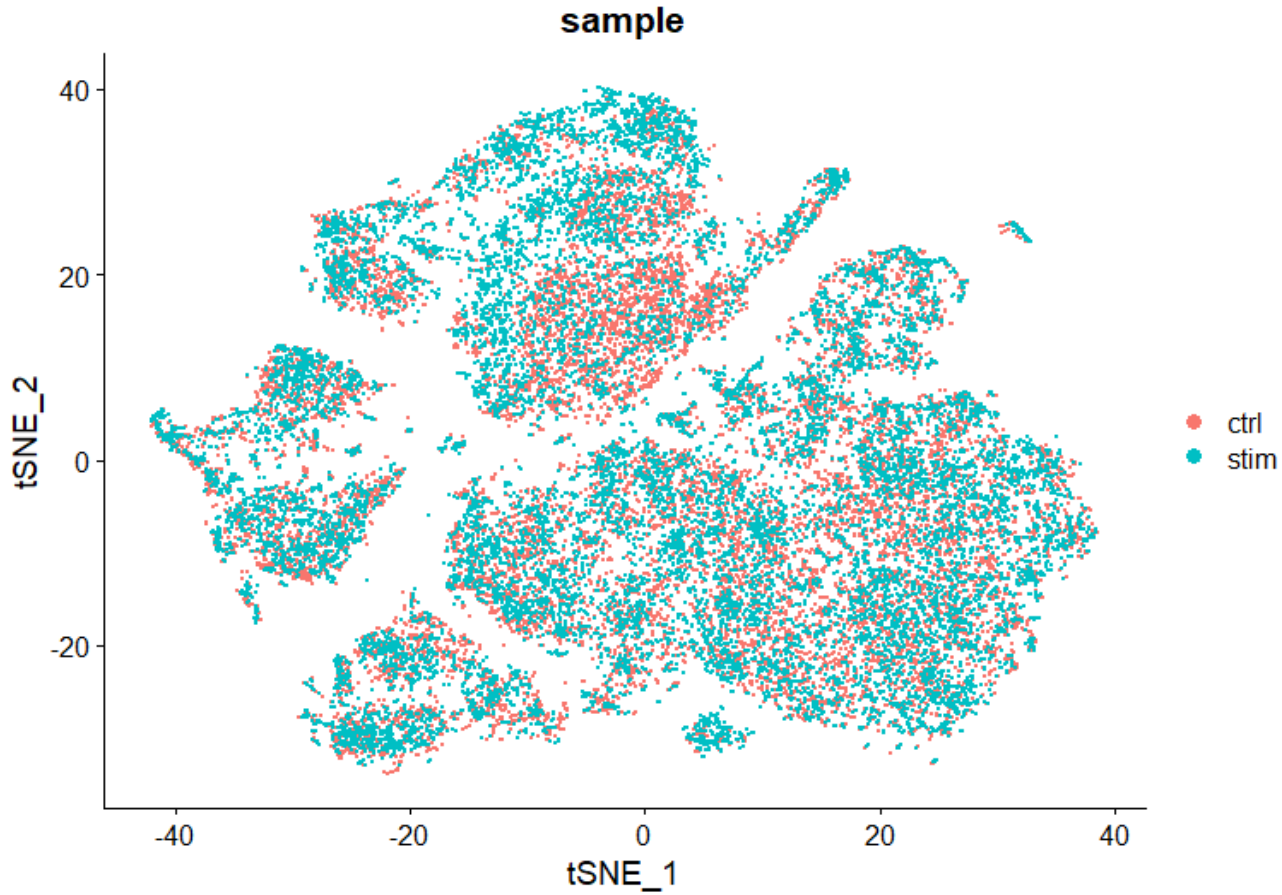


It is clear that this dataset benefitted from the integration! → saveRDS(seurat_integrated, "results/integrated_seurat.rds")

Other type of visualization (tSNE)

```
seurat_integrated <-
RunTSNE(seurat_integrated,
reduction = "pca", dims =
1:40, perplexity = 30,
max_iter = 1000, theta = 0.5,
eta = 200, num_threads = 0)
```

```
DimPlot(seurat_integrated,
reduction = "tsne", group.by
= "orig.ident")
```



Clustering

Create a new Rscript file (clustering.R)

New script (clustering.R)

```
# Load libraries
library(Seurat)
library(tidyverse)
library(RCurl)
library(cowplot)

# Load the integrated seurat object into the environment
seurat_integrated <- readRDS("results/integrated_seurat.rds")
```

Identify significant PCs

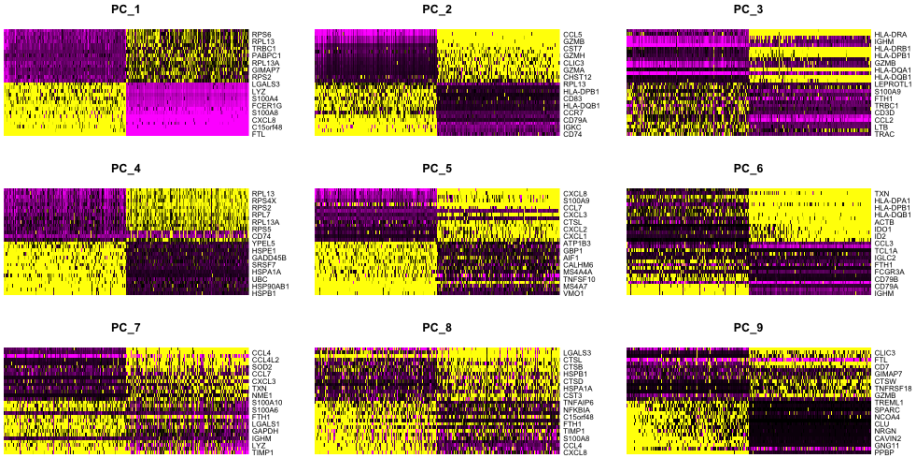
- To overcome the extensive technical noise in the expression of any single gene for scRNA-seq data, **Seurat assigns cells to clusters based on their PCA scores derived from the expression of the integrated most variable genes**, with each PC essentially representing a "metagene" that combines information across a correlated gene set. **Determining how many PCs to include in the clustering step is therefore important to ensure that we are capturing the majority of the variation**, or cell types, present in our dataset.

- It is useful to explore the PCs prior to deciding which PCs to include for the downstream clustering analysis.

Exploring the PCs (heatmap)

- One way of exploring the PCs is using a **heatmap** to visualize the most variant genes for select PCs with the **genes and cells ordered by PCA scores**. The idea here is to look at the PCs and determine whether the genes driving them make sense for differentiating the different cell types.
- The cells argument specifies the number of cells with the most negative or postive PCA scores to use for the plotting. The idea is that we are looking for a PC where the heatmap starts to look more "fuzzy", i.e. where the distinctions between the groups of genes is not so distinct.

```
# Explore heatmap of PCs
DimHeatmap(seurat_integrated,
           dims = 1:9, cells = 500, balanced = TRUE)
```



Solution

```
# Printing out the most variable genes driving PCs
print(x = seurat_integrated[["pca"]],
      dims = 1:10,
      nfeatures = 5)
```

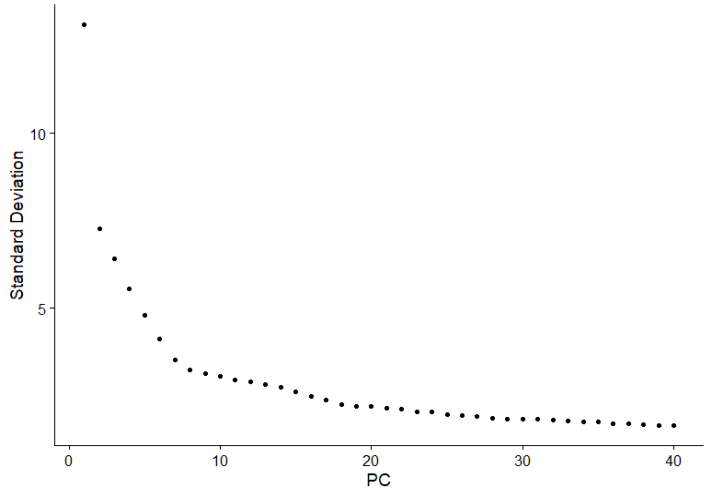
```
PC_ 1
Positive:  FTL, TIMP1, C15orf48, FTH1, CXCL8
Negative:  RPL3, RPS6, RPS18, RPL13, TRAC
PC_ 2
Positive:  CD74, IGHM, IGKC, HLA-DRA, CD79A
Negative:  GNLY, CCL5, NKG7, GZMB, FGFBP2
PC_ 3
Positive:  TRAC, PABPC1, LTB, GIMAP7, CCL2
Negative:  CD74, HLA-DRA, IGKC, IGHM, GNLY
PC_ 4
Positive:  HSPB1, CACYBP, HSP90AB1, HSPA8, UBC
Negative:  RPL3, RPL13, RPL10, RPS4X, RPS18
PC_ 5
Positive:  VMO1, FCGR3A, MS4A7, TIMP1, TNFSF10
Negative:  CCL2, CXCL8, S100A8, S100A9, FTL
```

This method can be slow and hard to visualize individual genes if we would like to explore a large number of PCs.

Exploring the PCs (elbow plot)

- The elbow plot is another helpful way to determine how many PCs to use for clustering so that we are capturing majority of the variation in the data. The elbow plot visualizes the standard deviation of each PC, and we are looking for where the standard deviations begins to plateau. Essentially, where the elbow appears is usually the threshold for identifying the majority of the variation. However, this method can be quite subjective.

```
# Plot the elbow plot
ElbowPlot(object = seurat_integrated,
          ndims = 40)
```

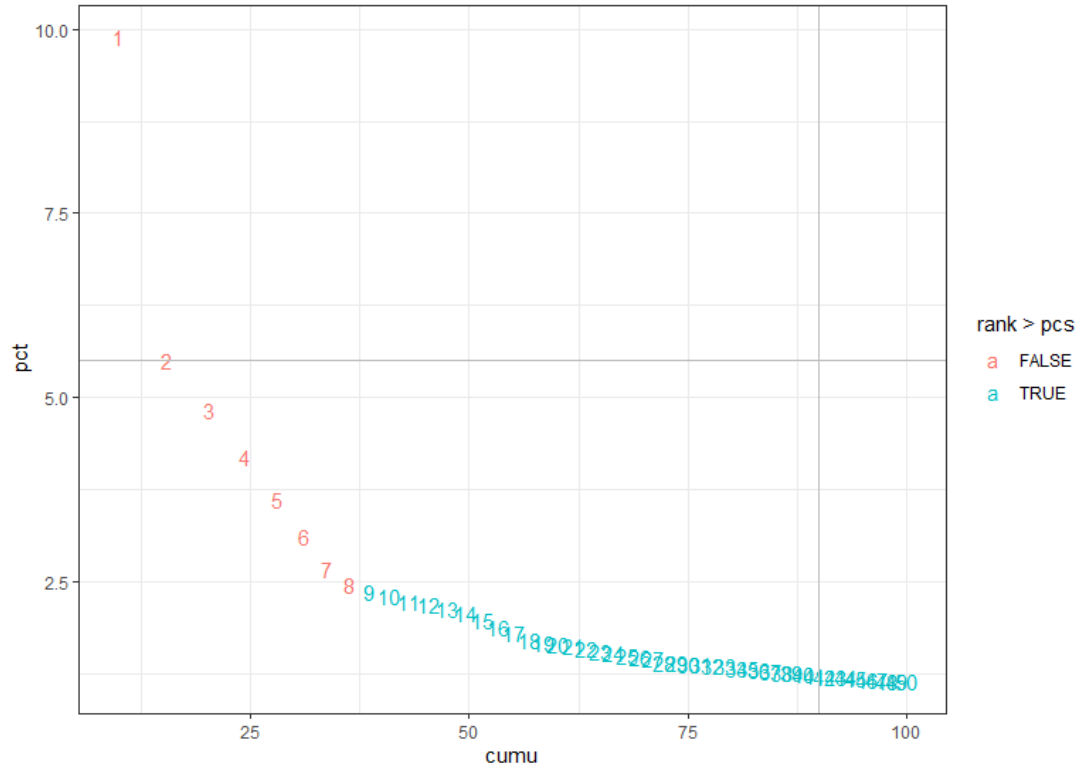


Based on this plot, we could roughly determine the majority of the variation by where the elbow occurs around PC8 - PC10, or one could argue that it should be when the data points start to get close to the X-axis, PC30 or so. This gives us a very rough idea of the number of PCs needed to be included. *Optional: Using a quantitative method to determine how many PCs to use.*

Optional: Quantitative approach

```
# Determine percent of variation associated with each PC

pct <- seurat_integrated[["pca"]]@stdev /
sum(seurat_integrated[["pca"]]@stdev) * 100


# Calculate cumulative percents for each PC

cumu <- cumsum(pct)


# Determine which PC exhibits cumulative percent greater than
90% and % variation associated with the PC as less than 5

co1 <- which(cumu > 90 & pct < 5)[1]


# Determine the difference between variation of PC and
subsequent PC

co2 <- sort(which((pct[1:length(pct) - 1] - pct[2:length(pct)]) > 0.1),
decreasing = T)[1] + 1


# last point where change of % of variation is more than 0.1%.

co2


# Minimum of the two calculation

pcs <- min(co1, co2)
```

```
# Create a dataframe with values

plot_df <- data.frame(pct = pct,

        cumu = cumu,

        rank = 1:length(pct))


# Elbow plot to visualize

 ggplot(plot_df, aes(cumu, pct, label = rank, color = rank > pcs)) +

 geom_text() +

 geom_vline(xintercept = 90, color = "grey") +

 geom_hline(yintercept = min(pct[pct > 5]), color = "grey") +

 theme_bw()
```



New vs old methods

- While the above **2 methods (heatmap and elbow plot) were used a lot more with older methods from Seurat for normalization** and identification of variable genes, they are no longer as important as they used to be. This is because **the SCTransform method is more accurate than older methods**.

Why is selection of PCs more important for older methods?

- The older methods incorporated some technical sources of variation into some of the higher PCs, so selection of PCs was more important. SCTransform estimates the variance better and does not frequently include these sources of technical variation in the higher PCs.

- **In theory, with SCTransform, the more PCs we choose the more variation is accounted for when performing the clustering**, but it takes a lot longer to perform the clustering. Therefore, for this analysis, **we will use the first 40 PCs to generate the clusters**.

Clustering

- Seurat uses a graph-based clustering approach, which embeds cells in a graph structure, using a K-nearest neighbor (KNN) graph (by default), with edges drawn between cells with similar gene expression patterns. Then, it attempts to partition this graph into highly interconnected ‘quasi-cliques’ or ‘communities’ [Seurat - Guided Clustering Tutorial]. A nice in-depth description of clustering methods is provided in the SVI Bioinformatics and Cellular Genomics Lab course.

- We will use the FindClusters() function to perform the graph-based clustering. The resolution is an important argument that sets the "granularity" of the downstream clustering and will need to be **optimized for every individual experiment**. For datasets of **3,000 - 5,000 cells, the resolution set between 0.4-1.4 generally yields good clustering. Increased resolution values lead to a greater number of clusters, which is often required for larger datasets.**
- The FindClusters() function allows us to enter a series of resolutions and will calculate the "granularity" of the clustering. This is very helpful for testing which resolution works for moving forward without having to run the function for each resolution.

Clustering (continued)

```
# Determine the K-nearest neighbor graph
seurat_integrated <- FindNeighbors(object = seurat_integrated,
                                   dims = 1:40)
# Determine the clusters for various resolutions
seurat_integrated <- FindClusters(object = seurat_integrated,
                                  resolution = c(0.4, 0.6, 0.8, 1.0, 1.4))
```

- **If we look at the metadata of our Seurat object(seurat_integrated@meta.data), there is a separate column for each of the different resolutions calculated.**

```
# Explore resolutions
seurat_integrated@meta.data %>%
    View()
```

- **To choose a resolution to start with, we often pick something in the middle of the range like 0.6 or 0.8.**

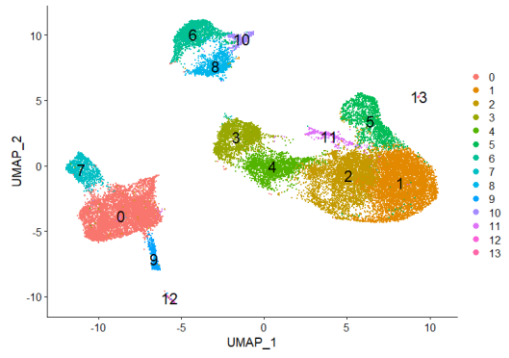
```
# Assign identity of clusters
Idents(object = seurat_integrated) <- "integrated_snn_res.0.4 "
# Plot the UMAP
DimPlot(seurat_integrated,
        reduction = "umap",
        label = TRUE,
        label.size = 6)
```

Check the UMAP for all the resolution: 0.4, 0.6, 0.8, 1, 1.4

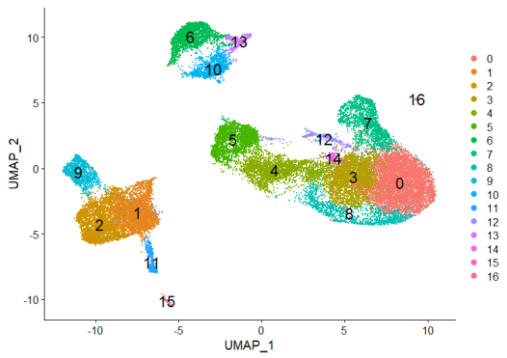
Comparing the different resolutions

Resolution:

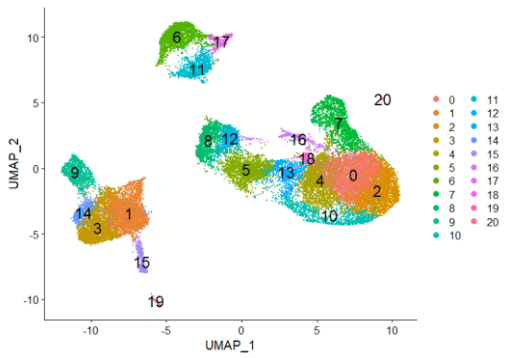
0,4



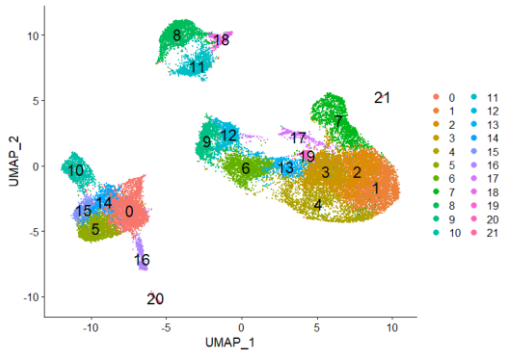
0,6



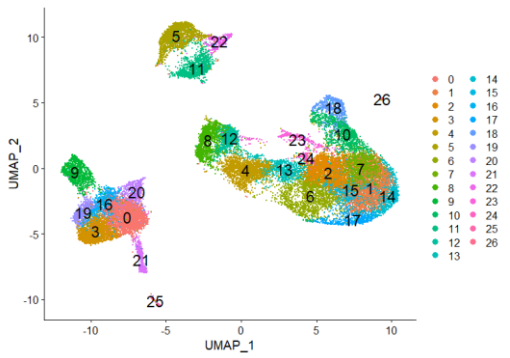
0,8



1,0

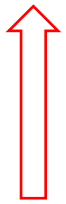


1,4



Number of clusters:

13	16	20	21	26



Differences in clustering

- It is possible that there is some variability in the way your clusters look compared to the image in this lesson. In particular you may see a difference in the labeling of clusters. **This is an unfortunate consequence of slight variations in the versions of packages (mostly Seurat dependencies).**

If your clusters do look different from what we have in the lesson, please follow the instructions provided below.

- Inside your data folder you will see a folder called additional_data. It contains the seurat_integrated object that we have created for the class.
- Load in the object to your R session and overwrite the existing one:

load("data/additional_data/seurat_integrated.RData.bz")

We will now **continue with the 0.8 resolution** to check the quality control metrics and known markers for the anticipated cell types. Plot the UMAP again to make sure your image now (or still) matches what you see in the lesson:

```
# Assign identity of clusters
Idents(object = seurat_integrated) <-
"integrated_snn_res.0.8"

# Plot the UMAP
DimPlot(seurat_integrated,
        reduction = "umap",
        label = TRUE,
        label.size = 6)
```

Exploration of quality control metrics

- To determine whether our clusters might be **due to artifacts such as cell cycle phase or mitochondrial expression**, it can be useful to explore these metrics visually to see if any clusters exhibit enrichment or are different from the other clusters. However, if enrichment or differences are observed for particular clusters it may not be worrisome if it can be explained by the cell type.

- To explore and visualize the various quality metrics, we will use the versatile DimPlot() and FeaturePlot() functions from Seurat.

Segregation of clusters by sample

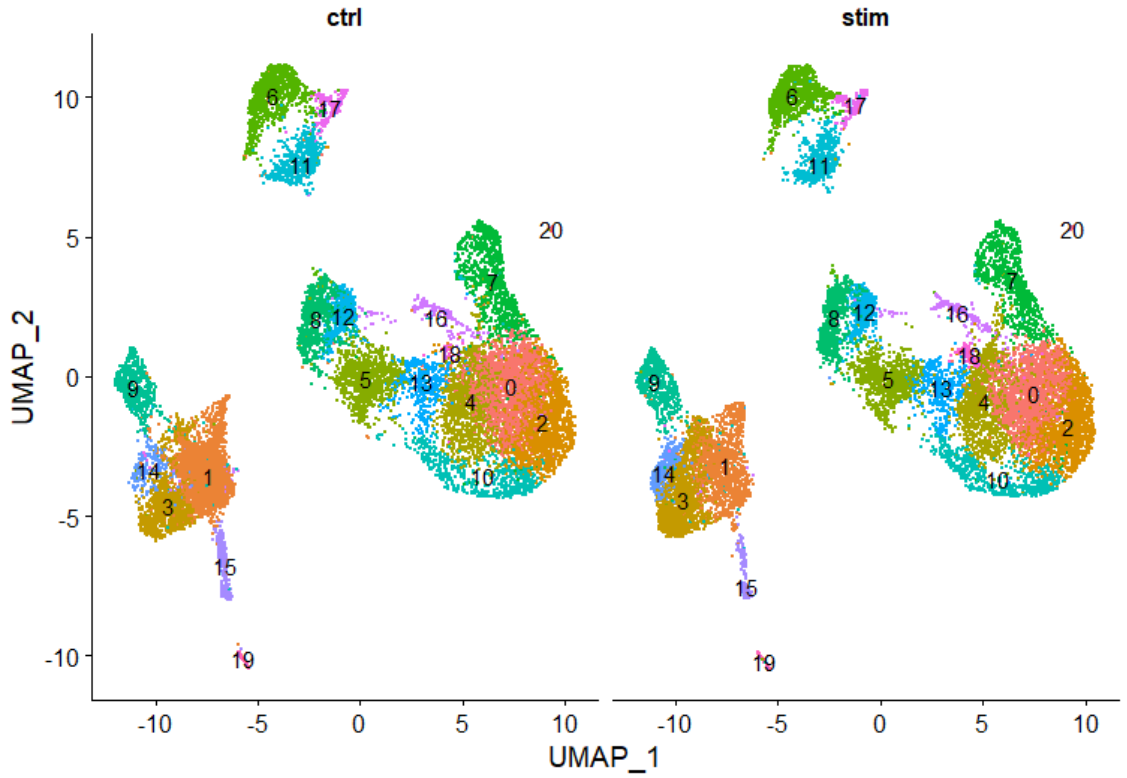
```
# Extract identity and sample information from seurat object to determine the number of cells per cluster per sample
n_cells <- FetchData(seurat_integrated,
                     vars = c("ident", "orig.ident")) %>%
      dplyr::count(ident, orig.ident) %>%
      tidyr::spread(ident, n)
# View table
View(n_cells)
```

orig.ident	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
ctrl	1839	2248	1582	1007	1178	938	910	902	741	609	580	567	407	451	201	227	183	183	36	46	12
stim	1786	1189	1567	1628	1147	976	912	899	724	624	606	579	507	430	408	186	196	181	147	78	12

Segregation of clusters by sample (UMAP)

 # UMAP of cells in each cluster by sample

```
DimPlot(seurat_integrated,
        label = TRUE,
        split.by = "sample")  + NoLegend()
```

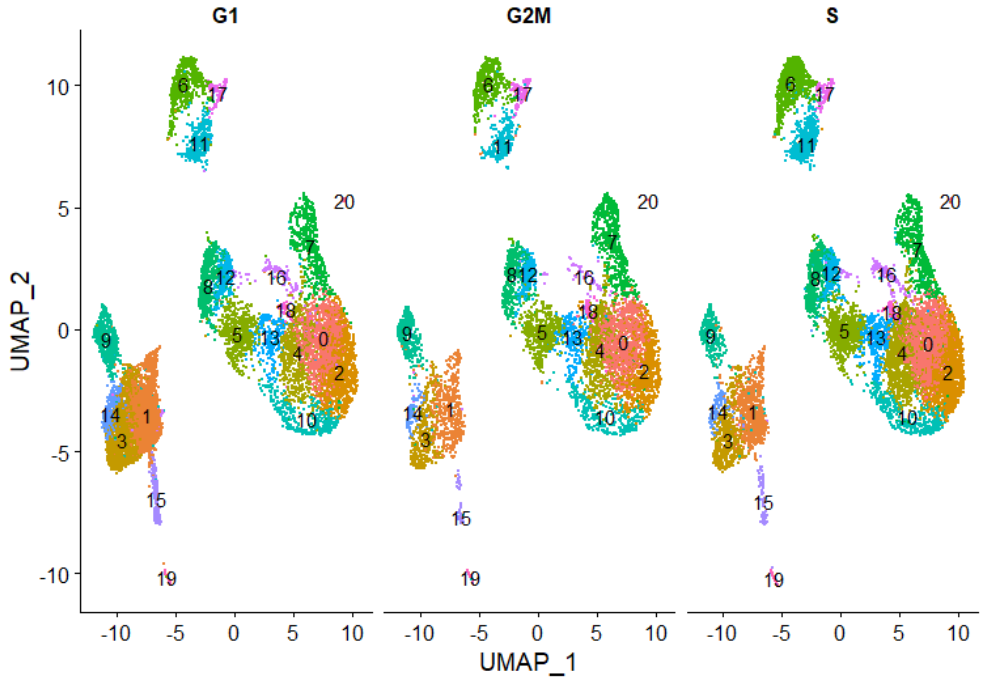


Generally, we expect to see the majority of the cell type clusters to be present in all conditions; however, depending on the experiment we might expect to see some condition-specific cell types present. **These clusters look pretty similar between conditions**, which is good since we expected similar cell types to be present in both control and stimulated conditions.

Segregation of clusters by cell cycle phase

- Next, we can explore whether the cells cluster by the different cell cycle phases. We did not regress out variation due to cell cycle phase when we performed the SCTransform normalization and regression of uninteresting sources of variation. If our cell clusters showed large differences in cell cycle expression, this would be an indication we would want to re-run the SCTransform and add the S.Score and G2M.Score to our variables to regress, then re-run the rest of the steps.

```
# Explore whether clusters segregate by cell cycle phase
DimPlot(seurat_integrated,
        label = TRUE,
        split.by = "Phase")  + NoLegend()
```

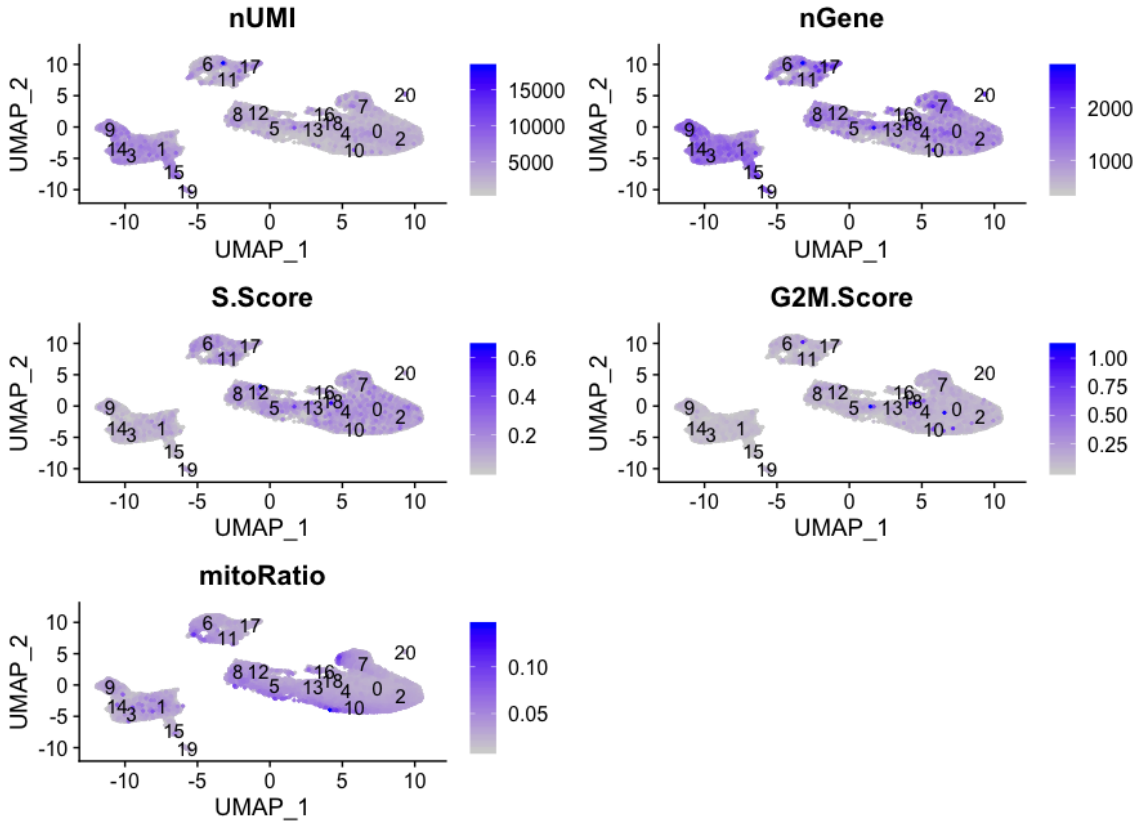


We do not see much clustering by cell cycle score, so **we can proceed with the QC.**

Segregation of clusters by various sources of uninteresting variation

- Next we will explore additional metrics, such as the number of UMIs and genes per cell, S-phase and G2M-phase markers, and mitochondrial gene expression by UMAP. Looking at the individual S and G2M scores can give us additional information to checking the phase as we did previously.

```
# Determine metrics to plot present in seurat_integrated@meta.data
metrics <-  c("nUMI", "nGene", "S.Score", "G2M.Score", "mitoRatio")

FeaturePlot(seurat_integrated,
        reduction = "umap",
        features = metrics,
        pt.size = 0.4,
        order = TRUE,
*       min.cutoff = 'q10',
        label = TRUE)
```



The metrics seem to be relatively even across the clusters, with the exception of the **nUMIs and nGene exhibiting higher values in clusters 3, 9, 14, and 15, and, perhaps, cluster 17**. We will keep an eye on these clusters to see whether the cell types may explain the increase.

If we see differences corresponding to any of these metrics at this point in time, then we will often note them and then **decide after identifying the cell type identities whether to take any further action.**

Back to RNA assay

 # Select the RNA counts slot to be the default assay

DefaultAssay(seurat_integrated) <- "RNA"

 # Normalize RNA data for visualization purposes

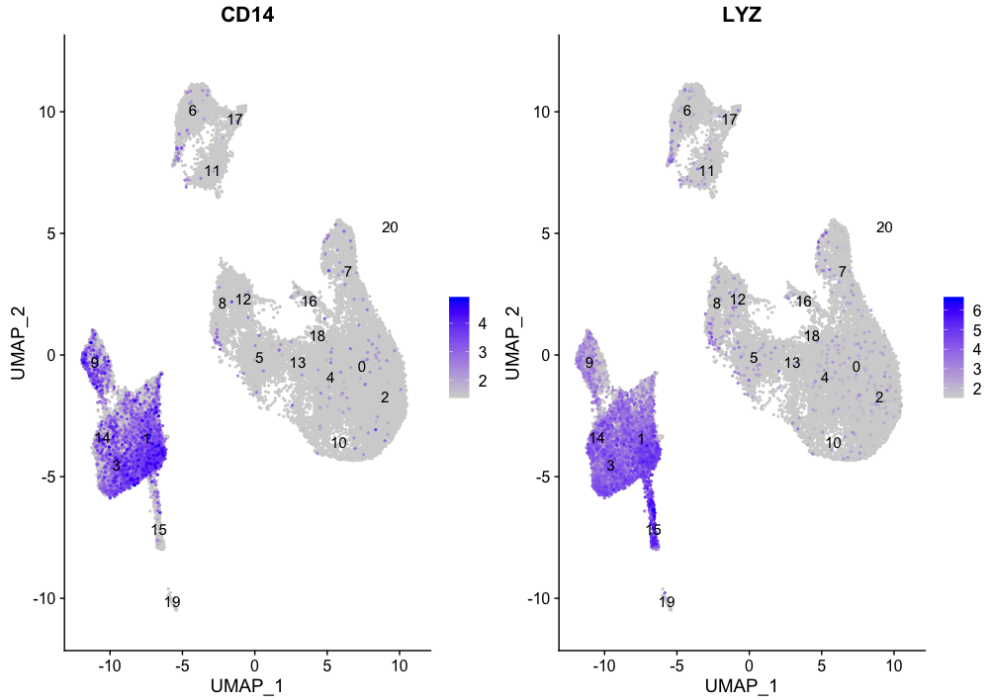
seurat_integrated <- NormalizeData(seurat_integrated, verbose = FALSE)

- The FeaturePlot() function from seurat makes it easy to visualize a handful of genes using the gene IDs stored in the Seurat object. We can easily explore the expression of known gene markers on top of our UMAP visualizations. Let's go through and determine the identities of the clusters. To access the normalized expression levels of all genes, we can use the normalized count data stored in the RNA assay slot.

Cell types and known markers

Cell Type	Marker
CD14+ monocytes	CD14, LYZ
FCGR3A+ monocytes	FCGR3A, MS4A7
Conventional dendritic cells	FCER1A, CST3
Plasmacytoid dendritic cells	IL3RA, GZMB, SERPINF1, ITM2C
B cells	CD79A, MS4A1
T cells	CD3D
CD4+ T cells	CD3D, IL7R, CCR7
CD8+ T cells	CD3D, CD8A
NK cells	GNLY, NKG7
Megakaryocytes	PPBP
Erythrocytes	HBB, HBA2

FeaturePlot(seurat_integrated, reduction = "umap", features = c("CD14", "LYZ"), order = TRUE, min.cutoff = 'q10', label = TRUE)



CD14+ monocytes appear to correspond to **clusters 1, 3, and 14**. We wouldn't include clusters 9 and 15 because they do not highly express both of these markers.

Doing this for every categories

Exercice

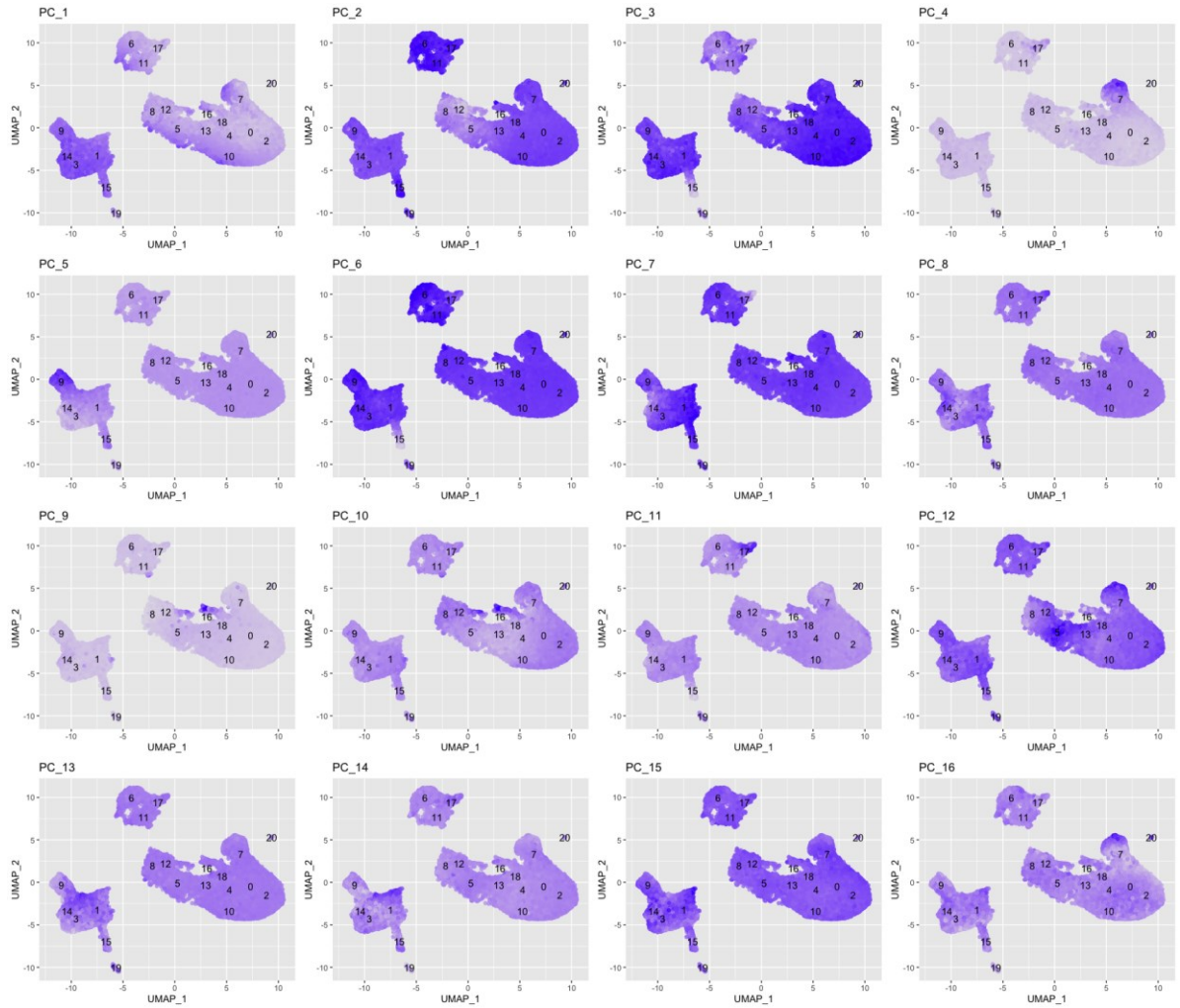
- Hypothesize the clusters corresponding to each of the different clusters in the table:

Cell Type	Clusters
CD14+ monocytes	1, 3, 14
FCGR3A+ monocytes	9
Conventional dendritic cells	15
Plasmacytoid dendritic cells	19
Marcrophages	-
B cells	?
T cells	?
CD4+ T cells	?
CD8+ T cells	?
NK cells	?
Megakaryocytes	?
Erythrocytes	?
Unknown	?

NOTE: **If any cluster appears to contain two separate cell types, it's helpful to increase our clustering resolution to properly subset the clusters**. Alternatively, if we still can't separate out the clusters using increased resolution, then it's possible that we had used too few principal components such that we are just not separating out these cell types of interest. To inform our choice of PCs, we could look at our **PC gene expression overlapping the UMAP plots and determine whether our cell populations are separating by the PCs included.**

Exploration of the PCs driving the different clusters

```
# Defining the information in the seurat object of interest
columns <- c(paste0("PC_", 1:16),
             "ident",
             "UMAP_1", "UMAP_2")
# Extracting this data from the seurat object
pc_data <- FetchData(seurat_integrated,
                     vars = columns)
# Extract the UMAP coordinates for the first 10 cells
Seurat_integrated@reductions$umap@cell.embeddings[1:10, 1:2]
# Adding cluster label to center of cluster on UMAP
umap_label <- FetchData(seurat_integrated,
                        vars = c("ident", "UMAP_1", "UMAP_2"))  %>%
  group_by(ident) %>%
  summarise(x=mean(UMAP_1), y=mean(UMAP_2))
# Plotting a UMAP plot for each of the PCs
map(paste0("PC_", 1:16), function(pc){
       ggplot(pc_data,
              aes(UMAP_1, UMAP_2)) +
               geom_point(aes_string(color=pc), alpha = 0.7) +
               scale_color_gradient(guide = FALSE, low = "grey90", high = "blue")  +
               geom_text(data=umap_label, aes(label=ident, x, y)) +
               ggtitle(pc) }) %>%
      plot_grid(plotlist = .)
```



We can see how the clusters are represented by the different PCs. For instance, the genes driving PC_2 exhibit higher expression in clusters 6, 11, and 17 (maybe a bit higher in 15, too). We could look back at our genes driving this PC to get an idea of what the cell types might be

Exploring known cell type markers

 # Examine PCA results

```
print(seurat_integrated[["pca"]], dims = 1:5,
nfeatures = 5)
```

```
PC_ 1
Positive:  FTL, TIMP1, C15orf48, FTH1, CXCL8
Negative:  RPL3, RPS6, RPS18, RPL13, TRAC
PC_ 2
Positive:  CD74, IGHM, IGKC, HLA-DRA, CD79A
Negative:  GNLY, CCL5, NKG7, GZMB, FGFBP2
PC_ 3
Positive:  TRAC, PABPC1, LTB, GIMAP7, CCL2
Negative:  CD74, HLA-DRA, IGKC, IGHM, GNLY
PC_ 4
Positive:  HSPB1, CACYBP, HSP90AB1, HSPA8, UBC
Negative:  RPL3, RPL13, RPL10, RPS4X, RPS18
PC_ 5
Positive:  VMO1, FCGR3A, MS4A7, TIMP1, TNFSF10
Negative:  CCL2, CXCL8, S100A8, S100A9, FTL
```

With the CD79A and CD74 genes and the HLA genes as positive markers of PC_2, **we can hypothesize that clusters 6, 11, and 17 correspond to B cells.** This just hints at what the clusters identity could be, with the identities of the clusters being determined through a combination of the PCs.

Unanswered questions

1. What are the cell type identities of clusters 7 and 20?
2. Do the clusters corresponding to the same cell types have biologically meaningful differences? Are there subpopulations of these cell types?
3. Can we acquire higher confidence in these cell type identities by identifying other marker genes for these clusters?

Marker identification analysis can help us address all of these questions!!

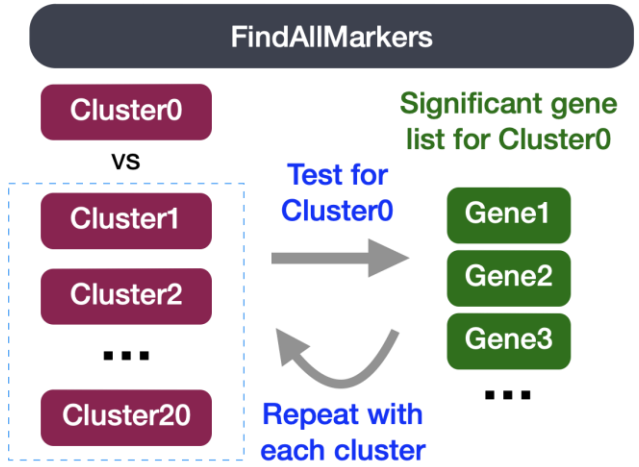
Marker identification

Different types of marker identification

1. **Identification of all markers for each cluster**: this analysis compares each cluster against all others and outputs the genes that are differentially expressed/present.
   - Useful for identifying unknown clusters and improving confidence in hypothesized cell types.
2. **Identification of conserved markers for each cluster**: This analysis looks for genes that are differentially expressed/present within each condition first, and then reports those genes that are conserved in the cluster across all conditions. These genes can help to figure out the identity for the cluster.
   - Useful with more than one condition to identify cell type markers that are conserved across conditions.
3. **Marker identification between specific clusters**: this analysis explores differentially expressed genes between specific clusters.
   - Useful for determining differences in gene expression between clusters that appear to be representing the same celltype (i.e with markers that are similar) from the above analyses.

1. Identification of all markers for each cluster

This type of analysis is typically **recommended for when evaluating a single sample group/condition**. With the FindAllMarkers() function we are comparing each cluster against all other clusters to identify potential marker genes. The cells in each cluster are treated as replicates, and essentially a differential expression analysis is performed with some statistical test.



<u>The FindAllMarkers() function has three important arguments which provide thresholds for determining whether a gene is a marker:</u>

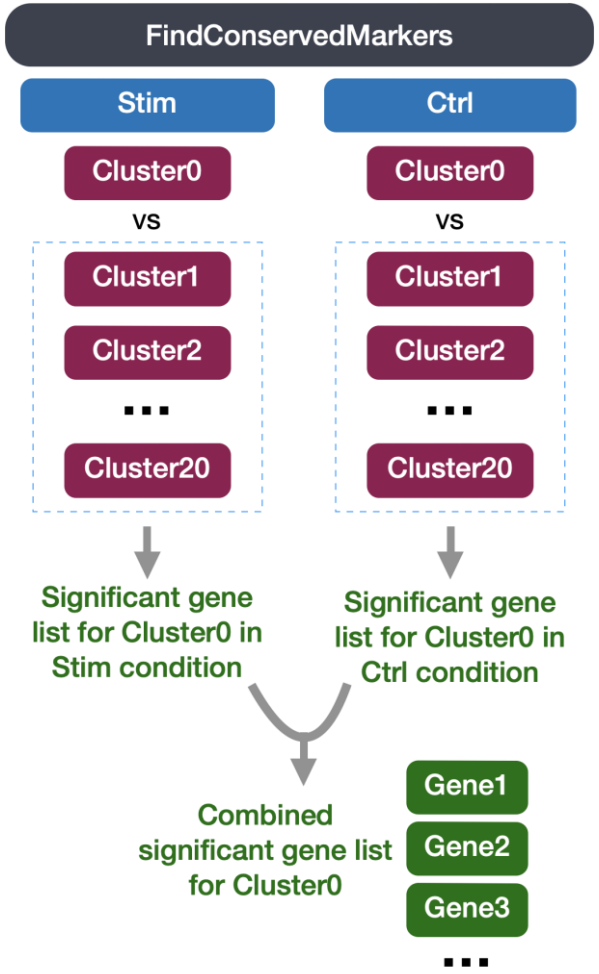
- **logfc.threshold**: minimum log2 fold change for average expression of gene in cluster relative to the average expression in all other clusters combined. Default is 0.25.
  - <u>Cons:</u>
    - could miss those cell markers that are expressed in a small fraction of cells within the cluster of interest, but not in the other clusters, if the average logfc doesn't meet the threshold
  - could return a lot of metabolic/ribosomal genes due to slight differences in metabolic output by different cell types, which are not as useful to distinguish cell type identities
- **min.diff.pct**: minimum percent difference between the percent of cells expressing the gene in the cluster and the percent of cells expressing gene in all other clusters combined.
  - <u>Cons:</u> could miss those cell markers that are expressed in all cells, but are highly up-regulated in this specific cell type
- **min.pct**: only test genes that are detected in a minimum fraction of cells in either of the two populations. Meant to speed up the function by not testing genes that are very infrequently expressed. Default is 0.1.
  - <u>Cons:</u> if set to a very high value could incur many false negatives due to the fact that not all genes are detected in all cells (even if it is expressed)

1. Identification of all markers for each cluster (continued)

- You could use any combination of these arguments depending on how stringent/lenient you want to be. Also, by default this function will return to you genes that exhibit both positive and negative expression changes. Typically, we add an argument only.pos to opt for keeping only the positive changes. The code to find markers for each cluster is shown below. **We will not run this code.**

```
## DO NOT RUN THIS CODE ##


# Find markers for every cluster compared to all remaining cells, report only the positive ones
# markers <- FindAllMarkers(object = seurat_integrated,
                            only.pos = TRUE,
                            logfc.threshold = 0.25)
```

2. Identification of conserved markers in all conditions



Since we have samples representing different conditions in our dataset, **our best option is to find conserved markers**. This function internally separates out cells by sample group/condition, and then performs differential gene expression testing for a single specified cluster against all other clusters (or a second cluster, if specified). Gene-level p-values are computed for each condition and then combined across groups using meta-analysis methods from the MetaDE R package.

2. Identification of conserved markers in all conditions (continued)

- Before we start our marker identification, we will explicitly set our default assay, we want to use the normalized data, but not the integrated data.

```
DefaultAssay(seurat_integrated) <- "RNA"

## DO NOT RUN ##

# FindConservedMarkers(seurat_integrated,
                       ident.1 = cluster,
                       grouping.var = "sample",
                       only.pos = TRUE,
                       min.diff.pct = 0.25,
                       min.pct = 0.25,
                       logfc.threshold = 0.25)
```

ident.1: this function only evaluates one cluster at a time; here you would specify the cluster of interest.

grouping.var: the variable (column header) in your metadata which specifies the separation of cells into groups

See FindAllMarkers() function

2. Identification of conserved markers in all conditions (Back to our analysis)

- For our analysis we will be fairly lenient and use only the log fold change threshold greater than 0.25. We will also specify to return only the positive markers for each cluster.

```
# For cluster 0
cluster0_conserved_markers <- FindConservedMarkers(seurat_integrated,
                              ident.1 = 0,
                              grouping.var = "sample",
                              only.pos = TRUE,
                              logfc.threshold = 0.25)
```

2. Identification of conserved markers in all conditions (Adding gene annotation)

- **Get annotations** **

```
annotations <- read.csv("data/annotation.csv")
```

- **Merge this annotation file with our results from the FindConservedMarkers():**

```
# Combine markers with gene descriptions
cluster0_ann_markers <- cluster0_conserved_markers %>%
                rownames_to_column(var="gene") %>%
                left_join(y = unique(annotations[, c("gene_name", "description")]),
                        by = c("gene" = "gene_name"))

View(cluster0_ann_markers)
```

2. Identification of conserved markers in all conditions (Last 2 slides in a loop)

- The function FindConservedMarkers() accepts a single cluster at a time, and we could run this function as many times as we have clusters. However, this is not very efficient. Instead we will first create a function to find the conserved markers including all the parameters we want to include. We will also add a few lines of code to modify the output. Our function will:

1. Run the FindConservedMarkers() function
2. Transfer row names to a column using rownames_to_column() function
3. Merge in annotations
4. Create the column of cluster IDs using the cbind() function

```
# Create function to get conserved markers for any given cluster
get_conserved <- function(cluster){
  FindConservedMarkers(seurat_integrated,
                       ident.1 = cluster,
                       grouping.var = "sample",
                       only.pos = TRUE) %>%
    rownames_to_column(var = "gene") %>%
    left_join(y = unique(annotations[, c("gene_name", "description")]),
              by = c("gene" = "gene_name")) %>%
    cbind(cluster_id = cluster, .)
  }
```

Now that we have this function created we can use it as an argument to the appropriate map function. We want the output of the map family of functions to be a dataframe with each cluster output bound together by rows, we will use the map_dfr() function.

```
# Iterate function across desired clusters
conserved_markers <- map_dfr(c(7,20), get_conserved)
```

Now, let's try this function to find the conserved markers for the clusters that were unidentified celltypes: **cluster7 and cluster 20.**

2. Identification of conserved markers in all conditions (Output)

- gene: gene symbol
- condition_p_val: p-value not adjusted for multiple test correction for condition
- condition_avg_logFC: average log fold change for condition. Positive values indicate that the gene is more highly expressed in the cluster.
- condition_pct.1: percentage of cells where the gene is detected in the cluster for condition
- condition_pct.2: percentage of cells where the gene is detected on average in the other clusters for condition
- condition_p_val_adj: adjusted p-value for condition, based on bonferroni correction using all genes in the dataset, used to determine significance
- max_pval: largest p value of p value calculated by each group/condition
- minimump_p_val: combined p value

When looking at the output, we suggest looking for markers with large differences in expression between pct.1 and pct.2 and larger fold changes. For instance, **if pct.1 = 0.90 and pct.2 = 0.80, it may not be as exciting of a marker. However, if pct.2 = 0.1 instead, the bigger difference would be more convincing**. Also, of interest is if the **majority of cells expressing the marker is in my cluster of interest**. If **pct.1 is low, such as 0.3, it may not be as interesting**. Both of these are also possible parameters to include when running the function, as described above.

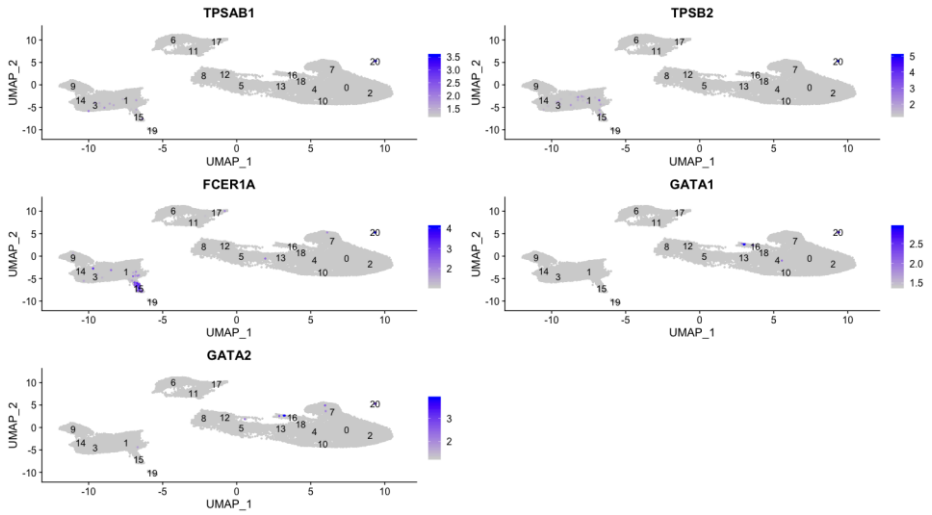
2. Identification of conserved markers in all conditions (Evaluating marker genes)

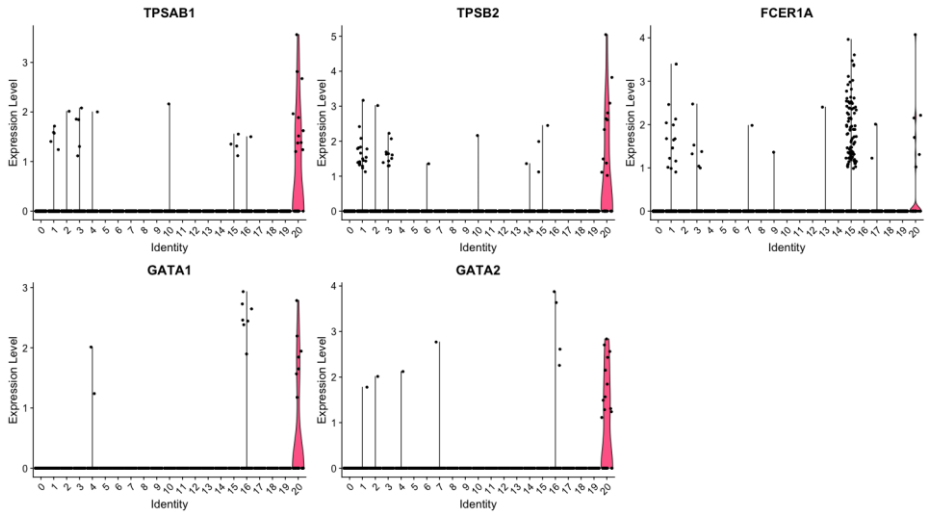
- We would like to use these gene lists to see of we can identify which celltypes these clusters identify with. Let's take a look at the top genes for each of the clusters and see if that gives us any hints. We can view the top 10 markers by average fold change across the two groups, for each cluster for a quick perusal:

```
# Extract top 10 markers per cluster
top10 <- conserved_markers %>%
  mutate(avg_fc = (ctrl_avg_log2FC + stim_avg_log2FC) /2) %>%
  group_by(cluster_id) %>%
  top_n(n = 10, wt = avg_fc)

# Visualize top 10 markers
# per cluster
View(top10)
```

	cluster_id	gene	stim_p_val	stim_avg_logFC	stim_pct.1	stim_pct.2	stim_p_val_adj	ctrl_p_val	ctrl_avg_logFC	ctrl_pct.1	ctrl_pct.2	ctrl_p_val_adj	max_pval	minimump_p_val	description
1	7	HSPH1	2.141001e-171	1.6350694	0.368	0.092	3.011318e-167	1.542393e-277	1.9741014	0.409	0.074	2.169376e-273	2.141001e-171	3.084786e-277	heat shock protein family H (Hsp110)
2	7	SRSF2	3.191808e-274	1.7482952	0.583	0.163	4.489279e-270	1.130148e-260	1.7199432	0.595	0.183	1.589553e-256	1.130148e-260	6.383617e-274	serine and arginine rich splicing fact
3	7	CACYBP	1.592172e-176	1.7705710	0.576	0.233	2.239390e-172	1.625804e-221	1.9658233	0.614	0.233	2.286694e-217	1.592172e-176	3.251608e-221	calcyclin binding protein [Source:HGN
4	7	RSRC2	2.781634e-186	1.4267216	0.405	0.102	3.912368e-182	2.103779e-177	1.4772857	0.411	0.110	2.958965e-173	2.103779e-177	5.563268e-186	arginine and serine rich coiled-coil 2
5	7	SRSF7	1.023796e-157	1.4358684	0.670	0.362	1.439969e-153	2.714803e-173	1.5435843	0.655	0.332	3.818371e-169	1.023796e-157	5.429607e-173	serine and arginine rich splicing fact
6	7	HSPA1B	3.038596e-37	1.4794885	0.141	0.048	4.273786e-33	3.135881e-148	1.8503230	0.155	0.017	4.410617e-144	3.038596e-37	6.271763e-148	heat shock protein family A (Hsp70)
7	7	HSPB1	1.280720e-12	1.0037800	0.349	0.279	1.801332e-08	3.301056e-146	2.3316820	0.420	0.142	4.642935e-142	1.280720e-12	6.602112e-146	heat shock protein family B (small) m
8	7	DNAJB1	1.282078e-85	1.4717877	0.251	0.075	1.803243e-81	4.952534e-141	1.7763258	0.307	0.075	6.965739e-137	1.282078e-85	9.905068e-141	DnaJ heat shock protein family (Hsp4
9	7	HSPE1	1.850008e-90	1.4485231	0.420	0.186	2.602036e-86	6.027633e-140	1.6666549	0.445	0.159	8.477866e-136	1.850008e-90	1.205527e-139	heat shock protein family E (Hsp10)
10	7	GADD45B	3.294581e-114	1.7212106	0.495	0.223	4.633828e-110	6.514622e-128	1.7449959	0.411	0.143	9.162816e-124	3.294581e-114	1.302924e-127	growth arrest and DNA damage indu
11	20	HBD	0.000000e+00	2.3767693	0.417	0.001	0.000000e+00	7.263101e-136	1.1152851	0.167	0.000	1.021555e-131	7.263101e-136	0.000000e+00	hemoglobin subunit delta [Source:HC
12	20	TPSB2	0.000000e+00	1.7607060	0.333	0.000	0.000000e+00	0.000000e+00	2.9627604	0.583	0.002	0.000000e+00	0.000000e+00	0.000000e+00	tryptase beta 2 (gene/pseudogene) [S
13	20	PRSS57	0.000000e+00	1.4620049	0.667	0.001	0.000000e+00	0.000000e+00	1.2949321	0.583	0.000	0.000000e+00	0.000000e+00	0.000000e+00	serine protease 57 [Source:HGNC Syr
14	20	GATA2	0.000000e+00	1.4589872	0.417	0.000	0.000000e+00	0.000000e+00	1.5080986	0.583	0.000	0.000000e+00	0.000000e+00	0.000000e+00	GATA binding protein 2 [Source:HGN
15	20	TPSAB1	0.000000e+00	0.9337225	0.417	0.000	0.000000e+00	0.000000e+00	1.9744595	0.500	0.001	0.000000e+00	0.000000e+00	0.000000e+00	tryptase alpha/beta 1 [Source:HGNC
16	20	FCER1A	9.608603e-60	1.8805174	0.250	0.003	1.351450e-55	3.168113e-30	0.7597692	0.250	0.005	4.455951e-26	3.168113e-30	1.921721e-59	Fc fragment of IgE receptor Ia [Source
17	20	TSC22D1	7.866117e-25	1.3327658	0.500	0.026	1.106369e-20	1.048548e-46	1.4604168	0.750	0.030	1.474782e-42	7.866117e-25	2.097095e-46	TSC22 domain family member 1 [Sou
18	20	CAT	1.070471e-22	1.3234859	0.583	0.038	1.505617e-18	3.916753e-33	1.8295099	0.917	0.065	5.508913e-29	1.070471e-22	7.833505e-33	catalase [Source:HGNC Symbol;Acc:H
19	20	SERPINB1	2.627577e-10	1.4668306	1.000	0.277	3.695687e-06	7.523932e-17	2.0501998	1.000	0.167	1.058241e-12	2.627577e-10	1.504786e-16	serpin family B member 1 [Source:HC
20	20	EIF3L	3.384988e-15	1.4013667	0.917	0.138	4.760986e-11	1.583196e-09	1.1595958	1.000	0.264	2.226765e-05	1.583196e-09	6.769976e-15	eukaryotic translation initiation facto

2. Identification of conserved markers in all conditions (Visualizing marker genes)





 # Plot interesting marker gene expression for cluster 20

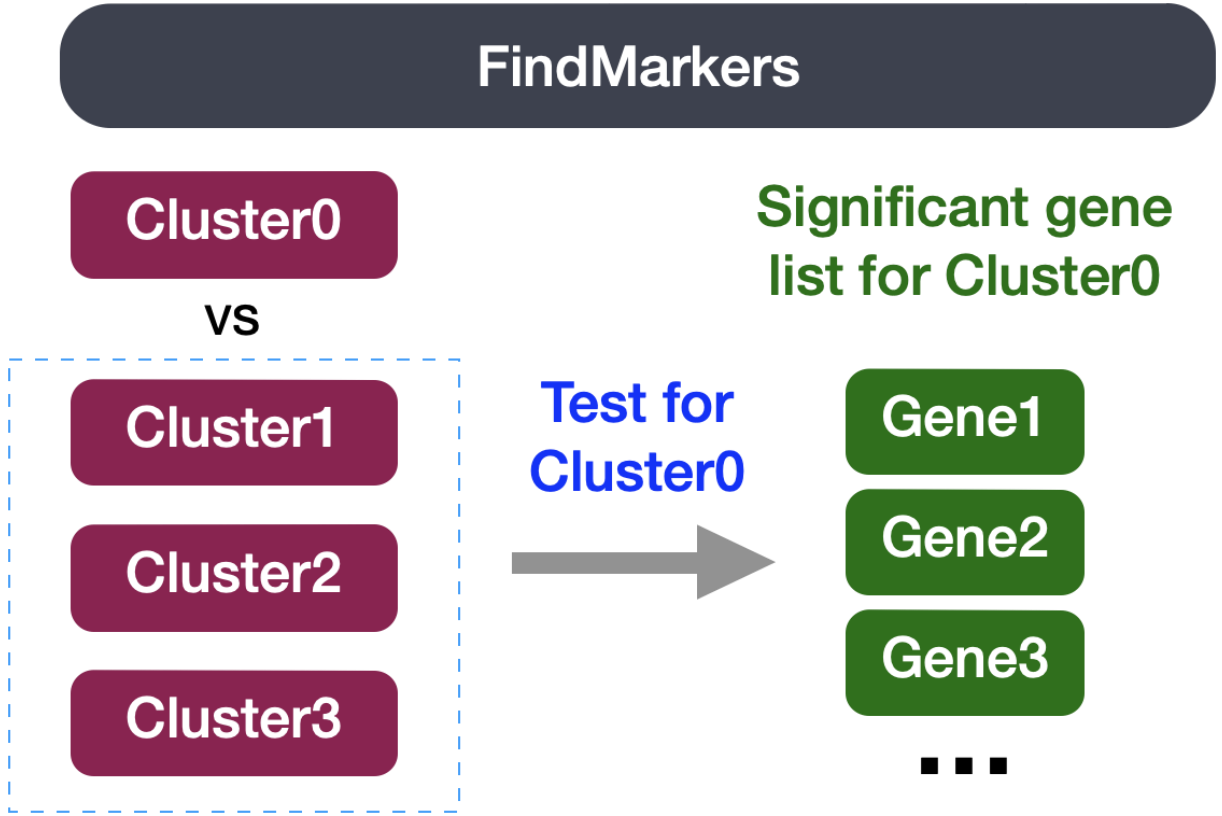
```
FeaturePlot(object = seurat_integrated,
                    features = c("TPSAB1", "TPSB2", "FCER1A", "GATA1", "GATA2"),
                    order = TRUE,
                    min.cutoff = 'q10',
                    label = TRUE,
                    repel = TRUE)
```

 # Violin plot for cluster 20

```
VlnPlot(object = seurat_integrated,
        features = c("TPSAB1", "TPSB2", "FCER1A", "GATA1", "GATA2"))
```

3. Identifying gene markers for each cluster

The last set of questions we had regarding the analysis involved **whether the clusters corresponding to the same cell types have biologically meaningful differences**. Sometimes the list of markers returned don't sufficiently separate some of the clusters. For instance, we had previously identified clusters 0, 2, 4, 10, and 18 as CD4+ Tcells, but are there biologically relevant differences between these clusters of cells? We can use the FindMarkers() function to determine the genes that are differentially expressed between two specific clusters.



3. Identifying gene markers for each cluster (In our analysis)

- We can try all combinations of comparisons, but we'll start with cluster 2 versus all other CD4+ T cell clusters:

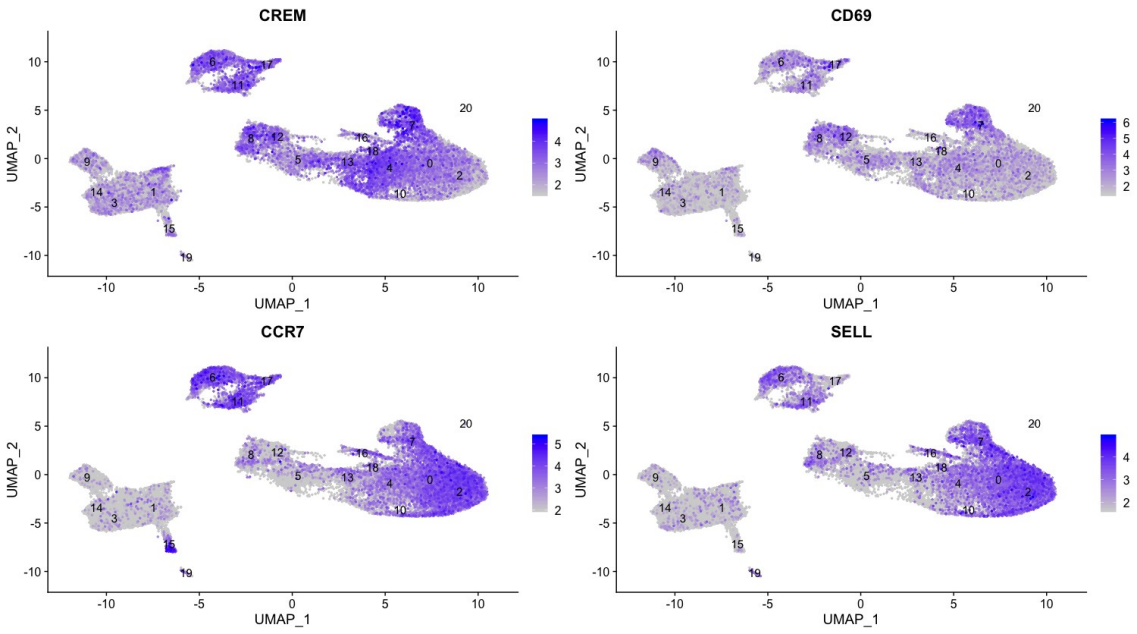
```
# Determine differentiating markers for CD4+ T cell
cd4_tcells <- FindMarkers(seurat_integrated,
                          ident.1 = 2,
                          ident.2 = c(0,4,10,18))

# Add gene symbols to the DE table
cd4_tcells <- cd4_tcells %>%
  rownames_to_column(var = "gene") %>%
  left_join(y = unique(annotations[, c("gene_name", "description")]),
            by = c("gene" = "gene_name"))

# Reorder columns and sort by padj
cd4_tcells <- cd4_tcells[, c(1, 3:5,2,6:7)]
cd4_tcells <- cd4_tcells %>%
  dplyr::arrange(p_val_adj)

# View data
View(cd4_tcells)
```

	gene	avg_logFC	pct.1	pct.2	p_val	p_val_adj	description
1	CREM	2.6397664	0.226	0.496	1.314107e-179	3.942320e-176	cAMP responsive element modulator [Source:HGNC Sy..
2	CD9	9.5800560	0.016	0.112	1.916164e-169	5.748492e-166	CD9 molecule [Source:HGNC Symbol;Acc:HGNC:1709]
3	SELL	9.0965860	0.682	0.438	9.627819e-158	2.888346e-154	selectin L [Source:HGNC Symbol;Acc:HGNC:10720]
4	TPST1	0.6578474	0.024	0.134	3.320298e-142	9.960893e-139	tyrosylprotein sulfotransferase 1 [Source:HGNC Symb...
5	AIF1	3.8619303	0.043	0.125	1.763972e-129	5.291916e-126	allograft inflammatory factor 1 [Source:HGNC Symbol;..
6	AARSD1	7.8978238	0.078	0.202	1.884181e-129	5.652543e-126	alanyl-tRNA synthetase domain containing 1 [Source:...
7	OSM	4.2415932	0.003	0.117	1.894438e-125	5.683313e-122	oncostatin M [Source:HGNC Symbol;Acc:HGNC:8506]
8	FCGBP	0.8675799	0.017	0.219	3.804900e-125	1.141470e-121	Fc fragment of IgG binding protein [Source:HGNC Sy...
9	PTX3	1.1851374	0.003	0.152	6.319115e-124	1.895734e-120	pentraxin 3 [Source:HGNC Symbol;Acc:HGNC:9692]
10	ATXN7L3B	1.8629754	0.070	0.207	1.464097e-116	4.392292e-113	ataxin 7 like 3B [Source:HGNC Symbol;Acc:HGNC:379...
11	MAP1A	0.9964565	0.026	0.136	6.454661e-112	1.936398e-108	microtubule associated protein 1A [Source:HGNC Sym..
12	SLC25A4	1.1214033	0.088	0.166	1.995140e-110	5.985420e-107	solute carrier family 25 member 4 [Source:HGNC Sym...
13	FABP5	0.8636861	0.029	0.105	2.549302e-109	7.647906e-106	fatty acid binding protein 5 [Source:HGNC Symbol;Acc..
14	SUCLG2-AS1	1.1462363	0.004	0.133	1.424875e-105	4.274625e-102	SUCLG2 antisense RNA 1 (head to head) [Source:HGN...
15	PAXIP1-AS2	5.3370444	0.077	0.154	7.381585e-105	2.214475e-101	PAXIP1 antisense RNA 2 [Source:HGNC Symbol;Acc:H...
16	LST1	1.8430015	0.018	0.106	2.619578e-103	7.858735e-100	leukocyte specific transcript 1 [Source:HGNC Symbol;...

3. Identifying gene markers for each cluster (Results)

- Of these top genes the CREM gene stands out as a marker of activation. We know that another marker of activation is CD69, and markers of naive or memory cells include the SELL and CCR7 genes. Interestingly, the SELL gene is also at the top of the list. Let's explore activation status a bit visually using these new cell state markers:

Cell State	Marker
Naive T cells	CCR7, SELL
Activated T cells	CREM, CD69




```
# Plot gene markers of activated and naive/memory T cells
FeaturePlot(seurat_integrated,
        reduction = "umap",
        features = c("CREM", "CD69", "CCR7", "SELL"),
        label = TRUE,
        order = TRUE,
        min.cutoff = 'q10',
                repel = TRUE
        )
```

As markers for the naive and activated states both showed up in the marker list, it is helpful to visualize expression. Based on these plots it seems as though clusters 0 and 2 are reliably the naive T cells. However, for the activated T cells it is hard to tell. We might say that clusters 4 and 18 are activated T cells, but the CD69 expression is not as apparent as CREM. We will label the naive cells and leave the remaining clusters labeled as CD4+ T cells.

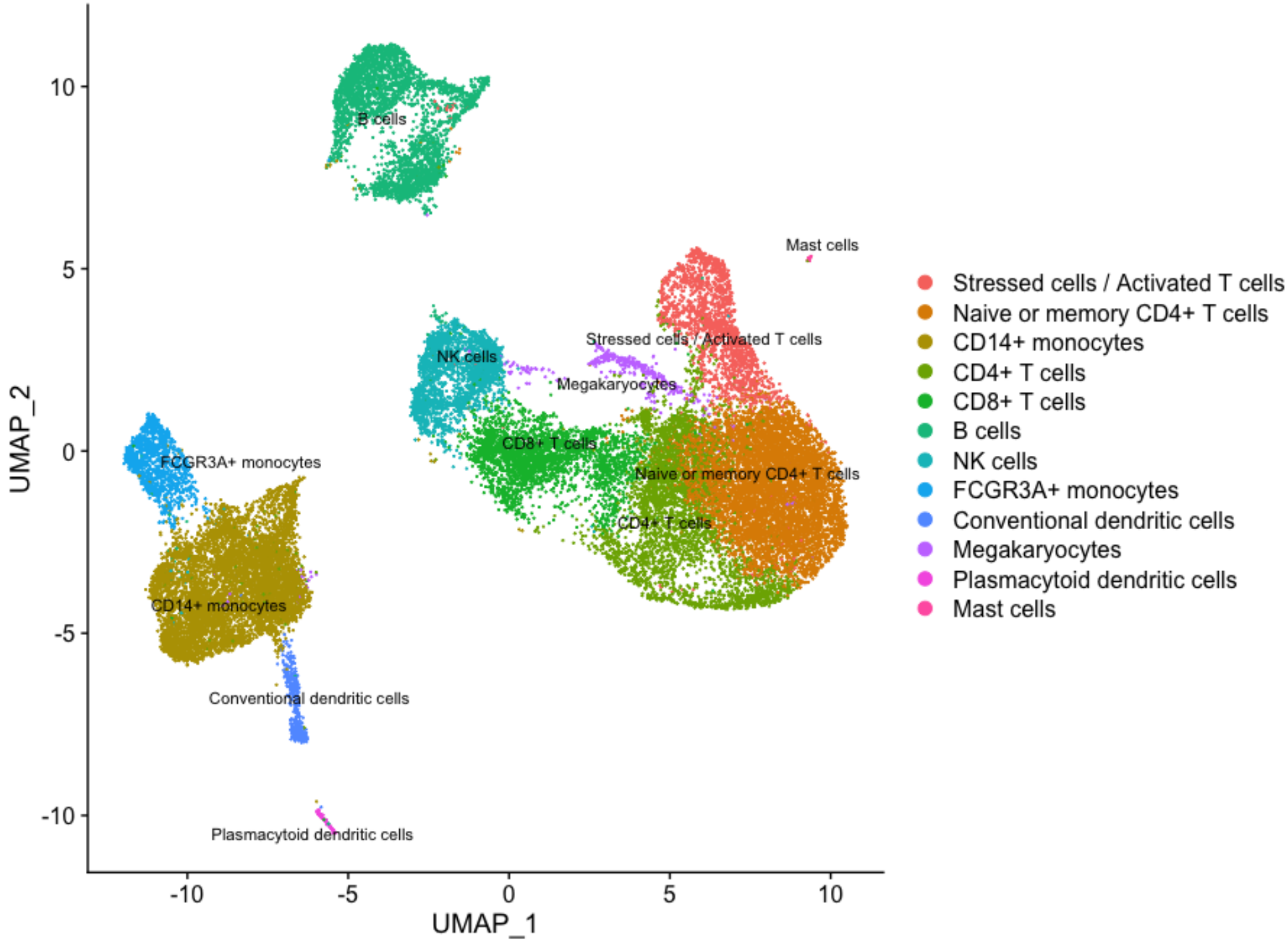
Summary

We can then reassign the identity of the clusters to these cell types:

```
# Rename all identities
seurat_integrated <- RenameIdents(object = seurat_integrated,
                                  "0" = "Naive or memory CD4+ T cells",
                                  "1" = "CD14+ monocytes",
                                  "2" = "Naive or memory CD4+ T cells",
                                  "3" = "CD14+ monocytes",
                                  "4" = "CD4+ T cells",
                                  "5" = "CD8+ T cells",
                                  "6" = "B cells",
                                  "7" = "Stressed cells / Activated T cells",
                                  "8" = "NK cells",
                                  "9" = "FCGR3A+ monocytes",
                                  "10" = "CD4+ T cells",
                                  "11" = "B cells",
                                  "12" = "NK cells",
                                  "13" = "CD8+ T cells",
                                  "14" = "CD14+ monocytes",
                                  "15" = "Conventional dendritic cells",
                                 "16" = "Megakaryocytes",
                                 "17" = "B cells",
                                 "18" = "CD4+ T cells",
                                 "19" = "Plasmacytoid dendritic cells",
                                 "20" = "Mast cells")
```

```
# Plot the UMAP
DimPlot(object = seurat_integrated,
        reduction = "umap",
        label = TRUE,
        label.size = 3,
        repel = TRUE)
```

Cluster ID	Cell Type
0	Naive or memory CD4+ T cells
1	CD14+ monocytes
2	Naive or memory CD4+ T cells
3	CD14+ monocytes
4	CD4+ T cells
5	CD8+ T cells
6	B cells
7	Stressed cells / Activated T cells
8	NK cells
9	FCGR3A+ monocytes
10	CD4+ T cells
11	B cells
12	NK cells
13	CD8+ T cells
14	CD14+ monocytes
15	Conventional dendritic cells
16	Megakaryocytes
17	B cells
18	CD4+ T cells
19	Plasmacytoid dendritic cells
20	Mast cells

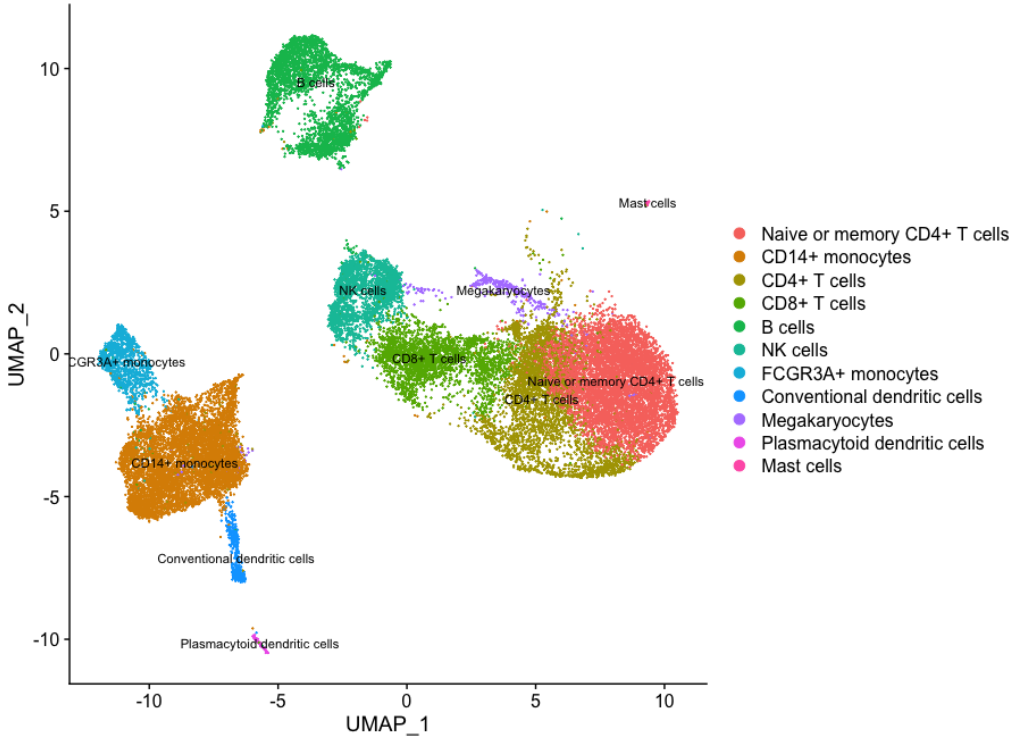

B cells
Mast cells
Stressed cells / Activated T cells
NK cells
Megakaryocytes
CD8+ T cells
Naive or memory CD4+ T cells
CD4+ T cells
FCGR3A+ monocytes
CD14+ monocytes
Conventional dendritic cells
Plasmacytoid dendritic cells
UMAP_2
UMAP_1
10
5
0
-5
-10
-10
-5
0
5
10
Stressed cells / Activated T cells
Naive or memory CD4+ T cells
CD14+ monocytes
CD4+ T cells
CD8+ T cells
B cells
NK cells
FCGR3A+ monocytes
Conventional dendritic cells
Megakaryocytes
Plasmacytoid dendritic cells
Mast cells

Get subset plot

- If we wanted to remove the potentially stressed cells, we could use the subset() function:

```
# Remove the stressed or dying cells
seurat_subset_labeled <- subset(seurat_integrated,
                                idents = "Stressed cells / Activated T cells", invert = TRUE)
```

```
# Re-visualize the clusters
DimPlot(object = seurat_subset_labeled,
        reduction = "umap",
        label = TRUE,
        label.size = 3,
            repel = TRUE)
```



Saving the projet

```
# Save final R object
write_rds(seurat_integrated,
          file = "results/seurat_labelled.rds")

# Create and save a text file with sessionInfo
sink("sessionInfo_scrnaseq_July2022.txt")
sessionInfo()
sink()
```

What's next?

- Experimentally validate intriguing markers for our identified cell types.
- Explore a subset of the cell types to discover subclusters of cells as described here
- Perform differential expression analysis between conditions ctrl and stim
  - Biological replicates are necessary to proceed with this analysis, and we have additional materials to help walk through this analysis.
- Trajectory analysis, or lineage tracing, could be performed if trying to determine the progression between cell types or cell states. For example, we could explore any of the following using this type of analysis:
  - Differentiation processes
  - Expression changes over time
  - Cell state changes in expression