Tutorial of single-cell RNA-ATAC multiomic sequencing data analysis in R

Compiled by Zhisong He, Barbara Treutlein

Updated on 2024-04-22

Table of Content

Introduction

Besides single-cell RNA-sequencing (scRNA-seq), there are nowadays more single-cell genomic technologies being developed to characterize different molecular signatures of single cells. Among them, single-cell assay for transposase-accessible chromatin using sequencing (scATAC-seq) is probably the most commonly used one. The ATAC-seq technology probes open chromatin with hyperactive mutant Tn5 Transposase that inserts sequencing adapters into open regions of the genome, to identify genomic regions of the measured sample which allows the binding of trans-regulatory factors, e.g. transcriptional factors. Those regions are likely the important regulatory genomic regions such as promoters or enhancers. On top of that, the scATAC-seq technology provides such chromatin accessibility information of single cells, which provides the epigenomic information of different cell types/states in a tissue that cannot be captured by scRNA-seq which only measures the transcriptome.

One difficulty when analyzing scATAC-seq data, though, is to annotate different cell populations identified in the data, as in most of the time, the definition of a cell type or state is based on the expression or certain gene markers, which is not directly measured by scATAC-seq experiments. There are computational methods developed that try to predict the transcriptomic outcomes given the chromatin accessibility information, and then annotate cells based

on such prediction. One simple but good example is the gene activity score used in `Signac` , the sister package of `Seurat` but specifically for scATAC-seq data analysis. This method sums the ATAC fragments intersecting the gene body and promoter region as the putative "gene activity", and uses it as the inferred gene expression for annotating cells as well as the potential integration with scRNA-seq data. This method generally works but definitely imperfect. Also, lacking the information of both transcriptome and epigenome for the same cell makes it hard to answer some interesting questions, such as whether there is any "lagging" of expression change relative to the epigenomic changes which may provide causal information between the two modalities.

Such an issue is then largely solved, by the technologies which can measure both the transcriptome and open chromatin profiles of the same cells. In 2020, 10x Genomics released their comercial single-cell RNA-ATAC multiome platform, allowing simultaneous measure of gene expression and chromatin accessibility profile in the same cells. There are also other techniques with different mechanisms. One good example is sci-CAR developed by Jay Shendure's lab in University of Washington, which uses a pooled barcode method to jointly capture and then dissect the transcriptomic and chromatin accessibility profiles of the same cells.

No matter with which method, once we get not only one modality of the data but two for each cell, there are much more that we can do, by not only having double the amount of analysis to do because of doubling of the data, but also jointly analyzing the two modalities together for some more interesting insight. In this tutorial, we will briefly talk about how we can analyze the single-cell RNA-ATAC multiomic sequencing data (scMultiome) in R using `Seurat` and `Signac` , the sister package of Seurat but mostly for scATAC-seq data analysis. We will talk about the basic analysis pipeline of the two modalities separately, the basic joint analysis for clustering and visualization, the more advanced joint analysis to look for potential important transcription regulators, and also to reconstruct gene regulatory network using both modalities.

If you are a beginner of single-cell transcriptomic data analysis, we recommend another tutorial we prepared for scRNA-seq data analysis. Besides, we have another tutorial also for scRNA-seq data analysis but focusing on comparison between different conditions.

Preparation

In this tutorial, we assume the data being generated with the 10x multiome platform, and have been preprocessed (base calling, sample demultiplexing, mapping, peak calling, read counting) using the 10x Genomic analysis pipeline Cell Ranger ARC. As a part of the tutorial, we provide one example data set, which is a scMultiome data of the blood vessle organoids cultured for seven days. This data is a part of the data presented in the paper "Fate and state transitions during human blood vessel organoid development".

This tutorial contains three sections. In the first section, we will quickly go through the analysis on transcriptome, and then mention the analysis we can do based on the chromatin accessibility profiles. In the second section, we will start to do the bimodal integrative analysis to look for potential regulators in the system. In the third section, we will introduce Pando, the algorithm to reconstruct gene regulatory network (GRN) using both of the data modalities, and apply it to the data to reconstruct the GRN responsible for cell type signature maintenance of the system.

Section 1. Mono-modal data analysis of the scMultiome data

Step 0. Import the required packages

As mentioned, we use `Seurat` and `Signac` packages to do most of the analysis in R. As the first step of the analysis, we import the two packages, so as the `dplyr` package which provides the pipe operator function as well as some other convenience functions to manipulate data frames.

```
library(Seurat)
library(Signac)
library(Matrix)
library(dplyr)
```

Step 1. Load the data and create the Seurat object

One major difference between the scRNA-seq and scMultiome (and scATAC-seq as well) data is that for transcriptome, when the same annotation is used, the genes represented in the count matrix are the same regardless of data sets. For the ATAC-seq assay, however, there is no such a comprehensive annotation of possible open chromatin or regulatory regions like genes. Therefore, before doing read counting, there is the peak calling step to look for genomic regions with significantly more accumulative reads covered than the surrounding background regions. As mentioned above, this step is a part of the Cell Ranger ARC pipeline, and it usually returns tens of thousands of peaks for one sample. After that, it counts the number of reads or fragments overlapping with each of those regions to generate an ATAC fragment count matrix. After running Cell Ranger ARC, one would therefore obtain two different count matrices with the same number of columns (cells) but different numbers of rows (genes or peaks), one for transcriptome and one for chromatin accessibility profiles. Both matrices are stored as separated .mtx files, but also together in one .h5 file. Here we directly read the .h5 file using the `Read10X_h5` function in the `Seurat` package, which returns a list of two sparse matrices, one for each assay. Note that in the example data there are two samples, and one .h5 file for each of them.

```
counts_NC8 <- Read10X_h5("data/NC8/filtered_feature_bc_matrix.h5")
counts_H9 <- Read10X_h5("data/H9/filtered_feature_bc_matrix.h5")
```

For the RNA assay, the count matrix is already sufficient. For the ATAC assay, on the other hand, while it is also possible to just use the count matrix, it is required to add extra information in order to allow more ATAC-based analysis. Those information includes the gene annotations (so that the relative location of a peak to the annotated genes are known) and the ATAC fragment table. The former one can be easily retrieved from Ensembl using the R package `AnnotationHub`. The latter one is a part of the Cell Ranger ARC output (atac_fragments.tsv.gz), which is the intermediate data summarizing the mapping result of the ATAC data by collapsing identical reads with the same cellular barcode, merging overlapping reads from the same cell into fragments, and discarding any information which is not needed to identify and quantify open chromatin regions (e.g. the sequence of the reads).

Firstly let's retrieve the gene annotation.

```
library(AnnotationHub)
ah <- AnnotationHub()
ensdbs <- query(ah, c("EnsDb.Hsapiens"))

ensdb_id <- ensdbs$ah_id[grep(paste0(" 98 EnsDb"), ensdbs$title)]
ensdb <- ensdbs[[ensdb_id]]

seqlevelsStyle(ensdb) <- "UCSC"
annotations <- GetGRangesFromEnsDb(ensdb = ensdb)
genome(annotations) <- "hg38"
```

Here we use the human gene annotation in Ensembl v98 database. This is also the version of annotation used for the RNA assay. The chromosome name is set to have the UCSC style so that they are called "chr1", "chr2" etc. rather than "1", "2" etc. so that they match with the name used in the reference genome used by the default Cell Ranger ARC pipeline.

Next, we create the Seurat object for each sample, using the expression count matrix to generate the RNA assay, and the ATAC fragment count matrix as well as the above retrieved annotation and the link to the fragment table file to generate the ATAC assay.

```
seurat_NC8 <- CreateSeuratObject(counts = counts_NC8$`Gene Expression`,
                                 assay = "RNA",
                                 project = "BVO_NC8")
seurat_NC8[['ATAC']] <- CreateChromatinAssay(counts = counts_NC8$`Peaks`,
                                             annotation = annotations,
                                             fragments =
"data/NC8/atac_fragments.tsv.gz",
                                             sep = c(":", "-"),
                                             genome = 'hg38')

seurat_H9 <- CreateSeuratObject(counts = counts_H9$`Gene Expression`,
                                assay = "RNA",
                                project = "BVO_H9")
seurat_H9[['ATAC']] <- CreateChromatinAssay(counts = counts_H9$`Peaks`,
                                            annotation = annotations,
                                            fragments =
"data/H9/atac_fragments.tsv.gz",
                                            sep = c(":", "-"),
                                            genome = 'hg38')
```

Afterwards, we need to combine the two biological replicate samples for the following analysis. Technically to combine two Seurat object we just can use the `merge` function in the `Seurat` package.

```
seurat <- merge(seurat_NC8, seurat_H9)
```

This would just work all fine for the RNA assay. For the ATAC assay, however, this is not enough. As mentioned above, peak calling is required in the preprocessing to identify regions with detected accessibility in the sample. This step, however, is applied to different samples separately, and therefore generates different peak list for different samples. This simply makes the merging failed.

To properly combine the ATAC assay, the critical steps are to firstly obtain a unified peak list for all samples, and then redo fragment counting based on the new peak list. To do that, there are several options.

1. Cell Ranger ARC has the command `aggr` which is designed to aggregate multiple 10x multiome experiments. One part of its pipeline is to redo peak calling by aggregating the fragment tables of all samples. Afterwards, we can direct read in the .h5 file in the aggregated sample output folder, as well as the aggregated fragment file.

2. We can union the peak lists of different samples called by Cell Ranger ARC to get the unified peak list, and then do the fragment counting again for all cells in all samples. Here the union means also to merge peaks from different samples which are partially overlapping with each other. `Signac` uses an `GRanges` object, which is a commonly used class defined in the `GenomicRanges` package as a critical part of Bioconductor for genomic analysis. The package also has methods to manipulate the GRanges objects in the way we need. Here, we can

use the `reduce` function which collapses all the overlapping regions into one so that there is no overlap between any pair of region in the final list.

```r
peaks <- reduce(unlist(as(c(seurat_NC8@assays$ATAC@ranges,
                            seurat_H9@assays$ATAC@ranges),
                          "GRangesList")))
peakwidths <- width(peaks)
peaks <- peaks[peakwidths < 10000 & peakwidths > 20]

counts_atac_merged <- FeatureMatrix(seurat@assays$ATAC@fragments,
                                    features = peaks,
                                    cells = colnames(seurat))
seurat[['ATAC']] <- CreateChromatinAssay(counts_atac_merged,
                                         fragments = seurat@assays$ATAC@fragments,
                                         annotation =
seurat@assays$ATAC@annotation,
                                         sep = c(":","-"),
                                         genome = "hg38")
```

3. We can redo the peak calling step. In `Signac` there is a wrapper function called `CallPeaks`. It calls MACS (Model-based Analysis of ChIP-Seq), the commonly used peak calling algorithm that originally designed for ChIP-seq data, to de novo call peaks. By default, it pools all the fragments of all cells to do it; but it also has the option to call peaks given a grouping column in the metadata, so that it calls peaks for each group separately first and then combine the called peaks of different groups to get the unified peak list.

```r
peaks <- CallPeaks(seurat,
                   assay="ATAC",
                   group.by="orig.ident",
                   macs2.path="/usr/bin/macs2")
counts_atac <- FeatureMatrix(seurat@assays$ATAC@fragments,
                             features = peaks,
                             cells = colnames(seurat))
seurat[['ATAC']] <- CreateChromatinAssay(counts_atac,
                                         fragments = seurat@assays$ATAC@fragments,
                                         annotation =
seurat@assays$ATAC@annotation,
                                         genome = 'hg38')
```

Once we get the new unified peak list and its count matrix, we can further subset the peaks to just focus on those at the standard chromosomes. This step is not a must but recommended as it makes some following analysis that requires calculation of local background GC content much easier.

```r
library(BSgenome.Hsapiens.UCSC.hg38)
standard_chroms <- standardChromosomes(BSgenome.Hsapiens.UCSC.hg38)
idx_standard_chroms <- which(as.character(seqnames(granges(seurat[['ATAC']]))) %in%
standard_chroms)
seurat[["ATAC"]] <- subset(seurat[["ATAC"]],
                           features = rownames(seurat[["ATAC"]])
[idx_standard_chroms])
seqlevels(seurat[['ATAC']]@ranges) <-
intersect(seqlevels(granges(seurat[['ATAC']])),

    unique(seqnames(granges(seurat[['ATAC']]))))
```

Step 2. Quality control

Now we have the Seurat object with the data stored. The next step is to do quality control, excluding cells which are not measured properly or for any reason with bad quality. There are several commonly used metrics for this.

For the RNA assay, what we would check are similar to the typical scRNA-seq. We try to filter out cells with too few or too many detected genes or transcripts (UMIs), as well as those with too high mitochondrial transcript percentage, just as mentioned in the scRNA-seq data analysis tutorial.

For the ATAC assay, we would also look at the number of detected peaks or detected fragments, similar to the RNA assay. On top of that, we would also exclude cells with too weak fragment enrichment around the transcriptional start sites (TSS). We can also quantify the approximate ratio of ATAC fragments with a strong nucleosome banding pattern (those with fragment lengths around a single nucleosome) that unlikely represent real accessible genomic regions, in relative to the nucleosome-free fragments, and then discard cells with too high of such ratio.

```r
seurat <- PercentageFeatureSet(seurat, pattern = "^MT-", col.name = "percent.mt",
assay = "RNA")
seurat <- NucleosomeSignal(seurat, assay = "ATAC")
seurat <- TSSEnrichment(seurat, assay = "ATAC")

VlnPlot(seurat,
        features = c("nFeature_RNA",
                     "percent.mt",
                     "nFeature_ATAC",
                     "TSS.enrichment",
                     "nucleosome_signal"),
        ncol = 5,
        pt.size = 0)
```
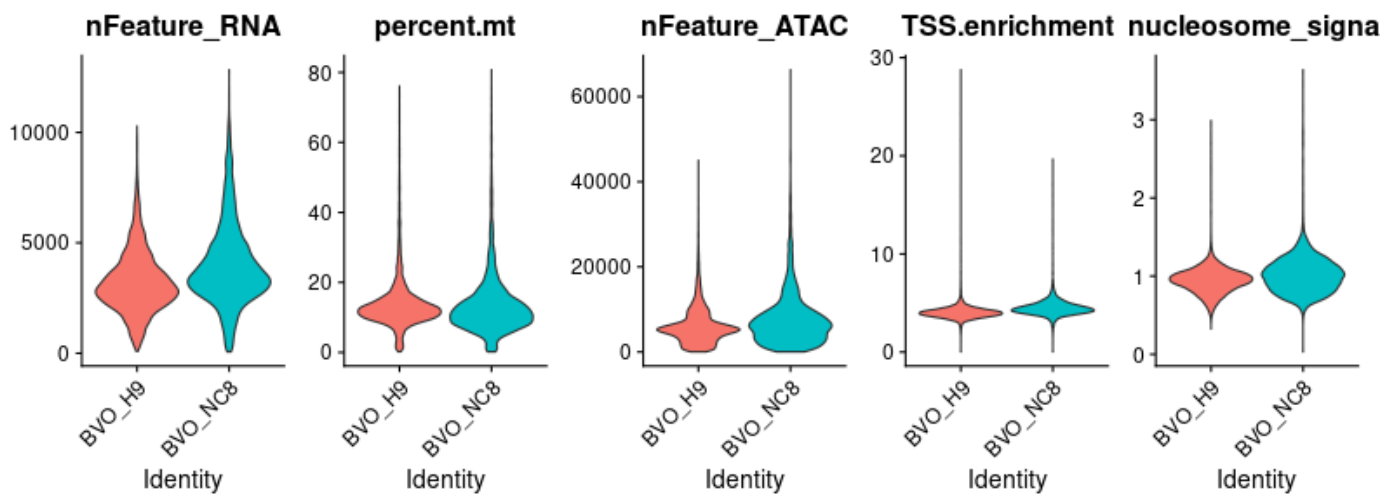
Based on the distributions, we can manually set the cutoffs for each metric to exclude the outlier cells.

```
seurat <- subset(seurat,
    subset = nFeature_RNA > 1000 &
      nFeature_RNA < 7500 &
      percent.mt < 30 &
      nFeature_ATAC > 1000 &
      nFeature_ATAC < 30000 &
      TSS.enrichment > 1 &
      nucleosome_signal < 2
)
```

## Step 3. Analysis on the RNA assay

Now we can start to do analysis for the RNA and ATAC assay separately, and we start with the RNA assay. Here we don't go into the details of all the steps. For those who would like to know more details, please refer to the scRNA-seq data analysis tutorial we prepared in parallel. In brief, what we will do include data normalization, highly variable genes identification, data scaling, principal component analysis (PCA), and then generate the UMAP embedding. Then, we can do feature plots for several markers of cell types we expect to see in the blood vessel organoids, including endothelial cells (PDGFB+, CLDN5+) and mural cells (PDGFRB+, COL5A1+).

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

seurat <- NormalizeData(seurat) %>%
    FindVariableFeatures(nfeatures = 3000) %>%
    CellCycleScoring(s.features = cc.genes.updated.2019$s.genes,
                     g2m.features = cc.genes.updated.2019$g2m.genes) %>%
    ScaleData() %>%
    RunPCA(npcs = 50) %>%
    RunUMAP(dims = 1:20, reduction.name = "umap_rna", reduction.key = "UMAPRNA_")

p1 <- DimPlot(seurat, group.by = "orig.ident", reduction = "umap_rna") & NoAxes()
p2 <- FeaturePlot(seurat,
                  c("PDGFB","CLDN5","PDGFRB","COL5A1"),
                  reduction = "umap_rna") & NoAxes() & NoLegend()
p1 | p2
```
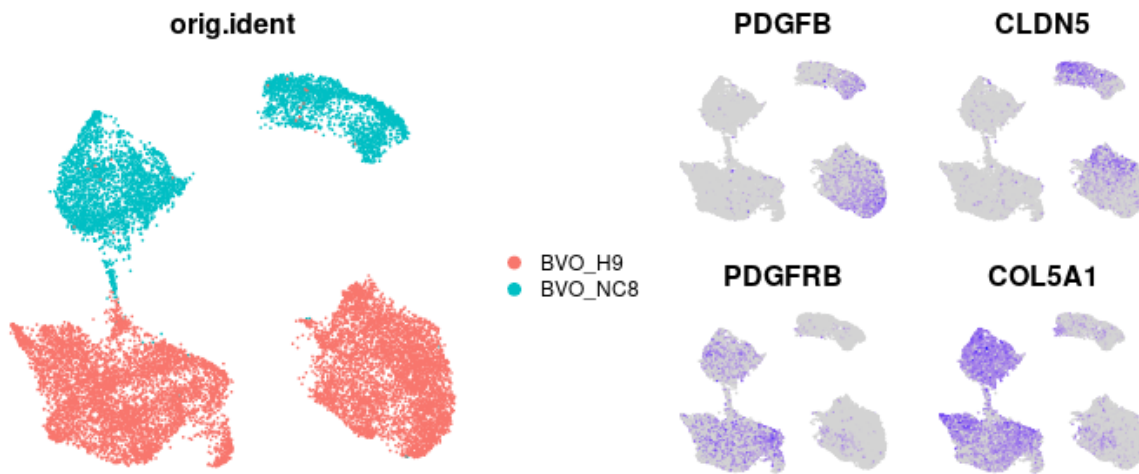
P.S. In the example script here, we use the typical procedure of log-normalization plus vst variable gene identification method plus z-transformation for data scaling. It is worth mentioning that there is the alternative way of using `SCTransform`. Indeed, there is the opinion that SCTransform may work better as the 10x multiome technology captures the RNA in the nucleus only (so it is actually single-nucleus RNA-seq), and therefore have generally lower detected transcript number than typical scRNA-seq experiment; while principally speaking SCTransform should work better than the classical normalization in such a scenario. In reality, they might not provide very different results, but this is something to keep in mind which also worths a try when you find the result of one way looking weird.

There seems to be pretty strong batch effect between the two biological replicates, and we shall therefore try to do data integration. There are quite many integration methods developed for scRNA-seq data, some of which have been introduced in the other tutorial. Usually the suggestion would be to try several and then choose one based on how well different samples are mixed and whether the feature plots make sense when checking the marker expression of some expected cell types; but here we will directly use CSS which was also used in the original paper introducing this data set.

```r
library(simspec)
seurat <- cluster_sim_spectrum(seurat,
                               label_tag = "orig.ident",
                               cluster_resolution = 0.6,
                               reduction.name = "css_rna",
                               reduction.key = "CSSRNA_")
seurat <- RunUMAP(seurat,
                  reduction = "css_rna",
                  dims = 1:ncol(Embeddings(seurat,"css_rna")),
                  reduction.name = "umap_css_rna",
                  reduction.key = "UMAPCSSRNA_")

p1 <- DimPlot(seurat, group.by = "orig.ident", reduction = "umap_css_rna") &
NoAxes()
p2 <- FeaturePlot(seurat,
                  c("PDGFB","CLDN5","PDGFRB","COL5A1"),
                  reduction = "umap_css_rna") & NoAxes() & NoLegend()
p1 | p2
```
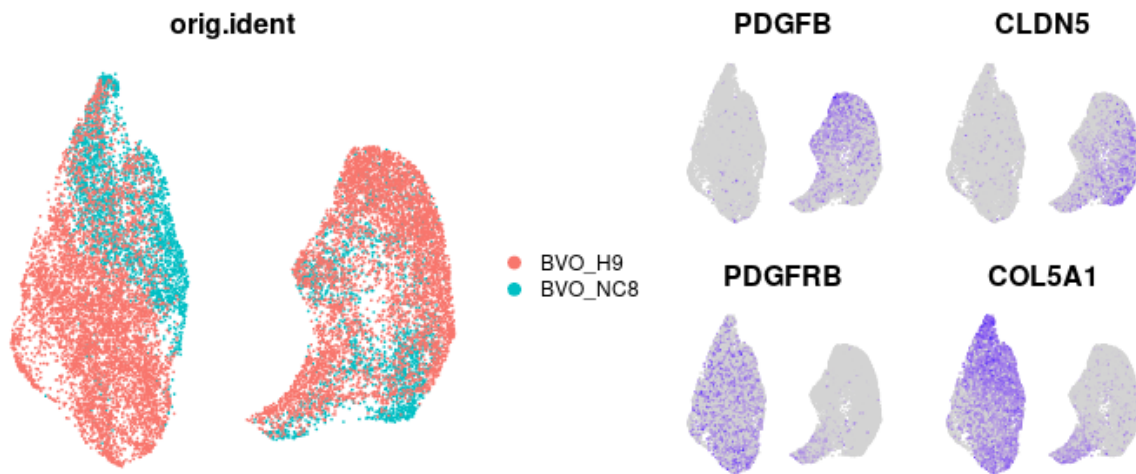
The integration works pretty well based on the marker expression. Afterwards we can do clustering and cluster marker identification, just as in the typical scRNA-seq data analysis.

```r
seurat <- FindNeighbors(seurat,
                        reduction = "css_rna",
                        dims = 1:ncol(Embeddings(seurat,"css_rna"))) %>%
  FindClusters(resolution = 0.2)

DE_cl_rna <- presto::wilcoxauc(seurat, "RNA_snn_res.0.2")
top_markers <- DE_cl_rna %>%
  filter(logFC > log(1.2) &
         auc > 0.7 &
         padj < 0.01 &
         pct_in - pct_out > 30 &
         pct_out < 30) %>%
  group_by(group) %>%
  top_n(1, wt = auc)

p1 <- DimPlot(seurat,
              group.by="RNA_snn_res.0.2",
              reduction="umap_css_rna", label=T) & NoAxes() & NoLegend()
p2 <- FeaturePlot(seurat,
                  features = unique(top_markers$feature),
                  reduction="umap_css_rna",
                  order = T,
                  ncol=3) & NoAxes() & NoLegend()
(p1 | p2) + patchwork::plot_layout(widths = c(2,3))
```
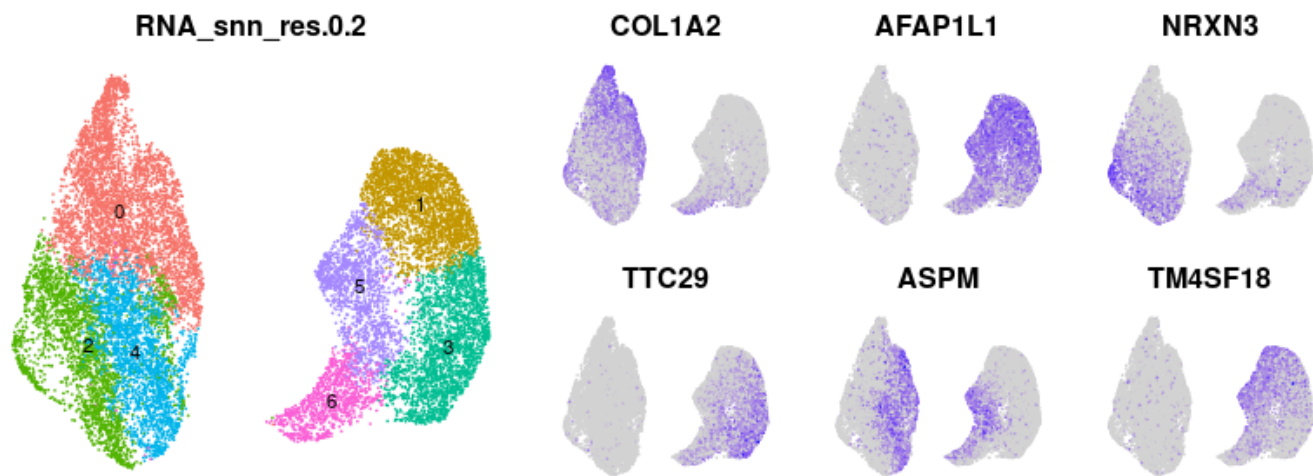
Step 4. Analysis on the ATAC assay

The general procedure of analyzing an ATAC assay is actually quite similar to the RNA counterpart. The steps include feature selection, data normalization, linear dimension reduction, non-linear dimension reduction for visualization, clustering and annotation, and marker identification. On the other hand, there are detailed differences between the analysis of the two modalities, mostly because the methods being used can be quite different.

Step 4.1. Feature selection

The first difference between analysis of RNA and ATAC assays is how to select features for further heterogeneity analysis. For the RNA assay, commonly used methods involve `vst` and `SCTransform`, and they select genes mostly based on the expression variantion of each gene across all the cells. This mechanism relies on the fact that the expression levels of a gene in different cells can be largely seen as cotinuous values. For the ATAC assay, however, technically this may be still true but conceptually this is no longer valid as one cell (at least for a normal human cell) contains only two copies of DNA, and a certain genomic region in one cell is either accessible or inaccessible in each of the two copies. This makes the accessibility levels rather discrete than continuous. Therefore, instead of using the variantion-based feature selection methods, the even simpler method is usually used for ATAC assay, which is to select peaks being detected in sufficient number of cells in the data. Here the number can be a fixed numeric number, or the percentile of peaks in terms of how commonly detected it is across cells. The selected features are therefore called "top features" instead of "variable features". Here let's select all the peaks with fragment detected in at least 50 cells.

```
DefaultAssay(seurat) <- "ATAC"
seurat <- FindTopFeatures(seurat, min.cutoff = 50)
```

Step 4.2. Normalization

Next, we need to do the data normalization, and it also comes the second difference between analyzing RNA and ATAC data. For RNA assay, the normalization step relies on certain assumptions, for instance, that every cell should contain the same amount of RNA (the size normalization used in log-normalization method). Those assumptions, however, doesn't match with the nature of ATAC data. Therefore, a different method called TF-IDF, short for **T**erm **F**requency–**I**nverse **D**ocument **F**requency, is used for ATAC data normalization instead. This is a statistic often used in searches of information retrieval, text mining, and user modeling. The calculation of TF-IDF includes the calculation of TF and IDF, and is implemented in the function `RunTFIDF` in the `Signac` package. The TF value is proportional to the frequency of a term (i.e. the number of fragments in a peak) in a document (i.e. a cell), and normalized by the total frequency of the term in all documents. The IDF value is the inverse of the proportion of documents that contains this term (i.e. the proportion of cells with any fragment within the peak detected). By

default, the TF-IDF is then defined as TF times IDF followed by log-transformation (`method = 1`). There are also other options, for instance, TF times log-transformed IDF (`method = 2`), and log-transformed TF times log-transformed IDF (`method = 3`).

```
seurat <- RunTFIDF(seurat, method = 1)
```

Another reason of using TF-IDF for ATAC-seq data is that its scales of different peaks, regardless the size of the peaks, should be comparable, and therefore the further data scaling becomes unnecessary. Computationally this is very important. From the definition it is clear that for peaks with no fragment detected in a cell, its TF-IDF value remains zero. Therefore, the TF-IDF matrix can be still represented by a sparse matrix, and then is directly used as the input for the next analysis without the need to generating a scaled data matrix which is likely a high-dimensional dense matrix that needs huge amount of memory to store.

Step 4.3. Linear dimension reduction

Similar to doing PCA for linear dimension reduction for the RNA assay, a linear dimension reduction is also needed for the ATAC assay to denoise the data as well as to reduce the computational complexity. In the Signac pipeline, instead of doing PCA with `RunPCA` , singular value decomposition (SVD), implemented as `RunSVD` , is used to do dimension reduction for ATAC data. As you may have known, PCA and SVD are very much related. In fact, the calculation of PCA is usually done via SVD (see here for more explanations). If you check the source code of `RunPCA` and `RunSVD` functions, you would probably realize easily that the key part of them two are exactly the same. What actually make the two behave a bit differently is that by default, `RunSVD` scales each component of the dimension reduction embeddings to mean 0 and SD 1, while no such scaling is applied in `RunPCA` .

```
seurat <- RunSVD(seurat, n = 50)
```

Step 4.4. Non-linear dimension reduction with UMAP for visualization

Once we get the SVD result, we can then generate the UMAP embedding given the SVD dimension reduction result as the input. This is very similar to the same step for scRNA-seq, just need to keep in mind to set `reduction = "lsi"` to tell the `RunUMAP` function (or `RunTSNE` function if you prefer t-SNE for visualization) to use the SVD result.

And similar to scRNA-seq, we also need to decide which components of the SVD results to use. By default `RunSVD` calculates the first 50 components (`n = 50`), but usually we don't use them all. Similar to the way we select the top components of PCA for the scRNA-seq data, we also try to select the first SVD components for the following analysis. Here the elbow plot (`ElbowPlot`), which shows the explained variance per component, may help. However, what makes it a bit different is that the first SVD component, which explains the most variance of the data, is usually highly correlated with total counts of the ATAC fragments per cell, and therefore represents more technical variance than biological variance. In this case, we would exclude the first SVD component and start from the second one. And since the first SVD component usually explains a lot more variance than the rest, it unfortunately makes the elbow plot less informative than when it is used to check the principal components for scRNA-seq data.

To confirm the correlation with the coverage, we can plot the correlation between each component and total count number per cell using the function `DepthCor` .

```
p1 <- ElbowPlot(seurat, ndims = 30, reduction="lsi")
p2 <- DepthCor(seurat, n = 30)
p1 | p2
```
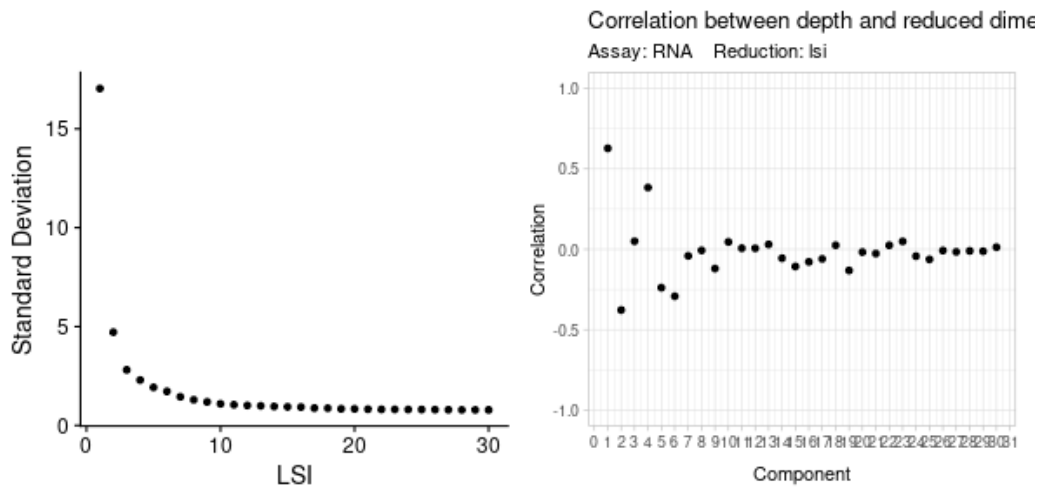
Next we use the 2nd to the 30th SVD components to generate the UMAP embedding of the ATAC assay. And since this is a scMultiome data set with gene expression information, we can also do feature plot of the marker genes mentioned above but using the ATAC-based UMAP.

```r
seurat <- RunUMAP(seurat,
                  reduction = "lsi",
                  dims = 2:30,
                  reduction.name = "umap_atac",
                  reduction.key = "UMAPATAC_")
p1 <- DimPlot(seurat,
              group.by = "orig.ident",
              reduction = "umap_atac") & NoAxes()
p2 <- FeaturePlot(seurat,
                  c("PDGFB","CLDN5","PDGFRB","COL5A1"),
                  reduction = "umap_atac") & NoAxes() & NoLegend()
p1 | p2
```
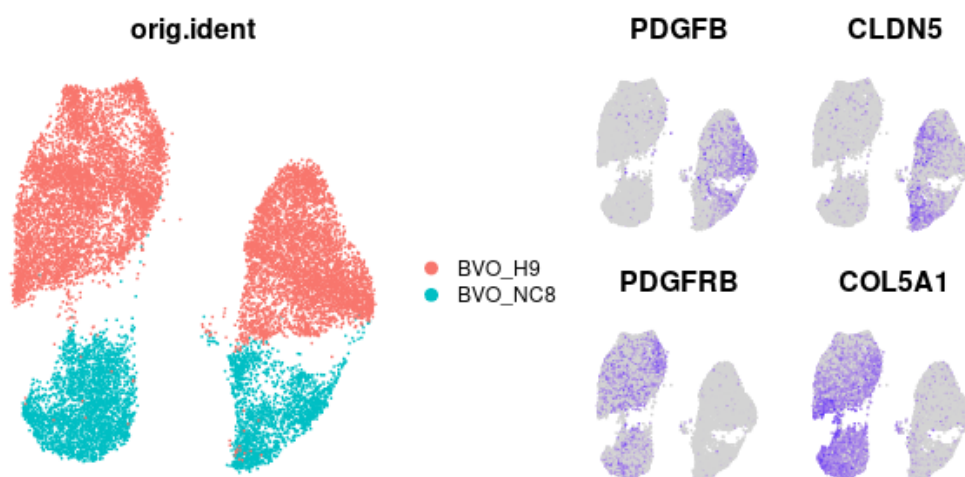

So similar to the RNA assay, the two biological replicates are separated completely, although both contains mural cells and endothelial cells in general. This implies potential batch effect just as the RNA assay.

Step 4.O. What if we don't have the RNA information

The RNA information in the multiome data is great because it is much easier to annotate cell types/states by gene expression. However, what if we have scATAC-seq rather than scMultiome data? In that case is there a way to "predict" the gene expression based on the ATAC fragments? Although not perfect, there is indeed some possible solutions.

The way that's the easiest is to use the `GeneActivity` function in `Signac` . What it does essentially is to count the number of fragments per cell that are overlapping with the annotated gene body of each gene. This fragment counts per gene per cell can be then normalized using the log-normalization method and then used as the proxy of gene expression in this cell. We can also apply this strategy in our scMultiome data and compare this predicted gene activity with the measured gene expression levels.

```
gene_act <- GeneActivity(seurat)
seurat[['RNA_inferred']] <- CreateAssayObject(gene_act) %>% NormalizeData()

DefaultAssay(seurat) <- "RNA_inferred"
beach_colscheme <-
colorRampPalette(c("#cdcdcd","#edf8b1","#7fcdbb","#41b6c4","#1d91c0","#225ea8","#0c
2c84"))
p3 <- FeaturePlot(seurat,
                  c("PDGFB","CLDN5","PDGFRB","COL5A1"),
                  reduction = "umap_atac",
                  cols = beach_colscheme(30)) & NoAxes() & NoLegend()
p1 | p2 | p3
```
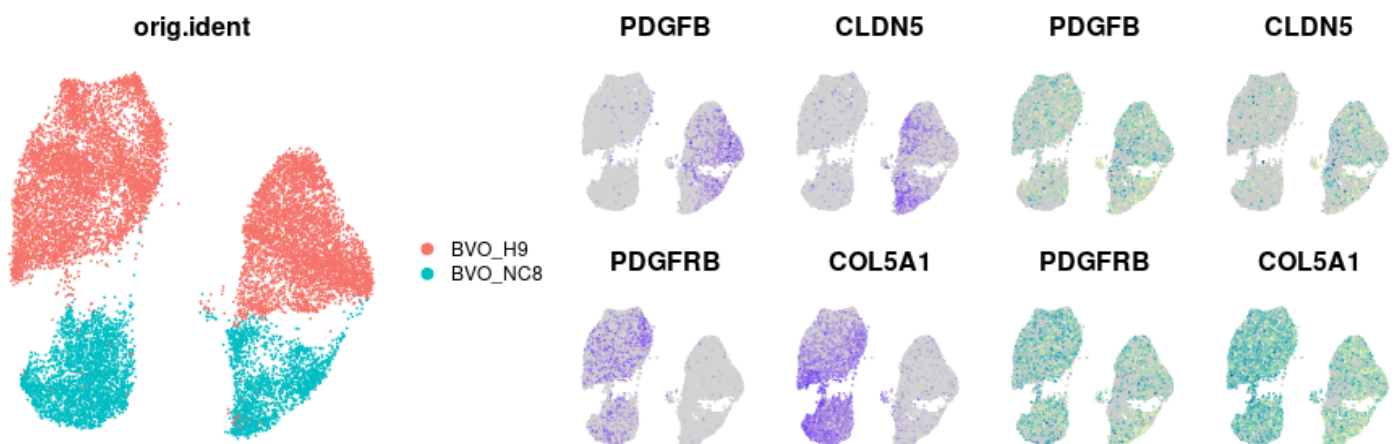

Although the signal is much less clear than the real RNA assay, we can still see the cell populations on the left have higher predicted activity of PDGFRB and COL5A1, while the right ones have higher predicted CLDN5 activity. This is consistent with the real gene expression information.

Step 4.5. Data integration of the ATAC assay

As we see the potential batch effect also in the ATAC assay, we shall then consider doing data integration or batch effect correction for the ATAC assay just like what we do for the RNA assay. While most of the data integration methods were developed for scRNA-seq data, many of them are also applicable to the ATAC assay. Those methods include but not limit to Seurat, Harmony and CSS. The way of using those integration methods is also quite similar to when applying them to the RNA assay.

For CSS and Harmony, just to make sure that it should be the SVD embedding (by default called "lsi") instead of the PCA embedding (by default called "pca") being used as the input.

```r
DefaultAssay(seurat) <- "ATAC"

library(simspec)
seurat <- cluster_sim_spectrum(seurat,
                               label_tag = "orig.ident",
                               use_dr = "lsi",
                               dims_use = 2:30,
                               cluster_resolution = 0.6,
                               reduction.name = "css_atac",
                               reduction.key = "CSSATAC_")
seurat <- RunUMAP(seurat,
                  reduction = "css_atac",
                  dims = 1:ncol(Embeddings(seurat,"css_atac")),
                  reduction.name = "umap_css_atac",
                  reduction.key = "UMAPCSSATAC_")

library(harmony)
seurat <- RunHarmony(seurat,
                     group.by.vars = "orig.ident",
                     reduction = "lsi",
                     dims.use = 2:30,
                     max.iter.harmony = 50,
                     reduction.save = "harmony_atac")
seurat <- RunUMAP(seurat,
                  reduction = "harmony_atac",
                  dims = 1:ncol(Embeddings(seurat,"harmony_atac")),
                  reduction.name = "umap_harmony_atac",
                  reduction.key = "UMAPHARMONYATAC_")
```

For Seurat it is a bit different. The first step is still to identify the anchors, although using the reduction method "rlsi" instead of the default value "cca" for the RNA assay. The second step is different. Instead of using the `IntegrateData` function in the RNA assay to impute the integrated gene expression, here it will be the SVD embedding being integrated to generate a new integrated embeddings. This step relies on the function `IntegrateEmbeddings`.

```r
integration.anchors <- FindIntegrationAnchors(
  object.list = SplitObject(seurat, "orig.ident"),
  anchor.features = rownames(seurat),
  reduction = "rlsi",
  dims = 2:30
)
seurat_integrated <- IntegrateEmbeddings(
  anchorset = integration.anchors,
  reductions = seurat[["lsi"]],
  new.reduction.name = "integrated_lsi",
  dims.to.integrate = 1:30
)
seurat[['integrated_lsi_atac']] <- CreateDimReducObject(
  Embeddings(seurat_integrated, "integrated_lsi")[colnames(seurat),],
key="INTEGRATEDLSIATAC_", assay="ATAC"
)
seurat <- RunUMAP(seurat,
                  reduction = "integrated_lsi_atac",
                  dims = 2:30,
                  reduction.name = "umap_seurat_atac",
                  reduction.key = "UMAPSEURATATAC_")
```

Now we can visualize and compare the results

```r
DefaultAssay(seurat) <- "RNA"
Reduce("|", lapply(c("umap_atac",
                     "umap_css_atac",
                     "umap_harmony_atac",
                     "umap_seurat_atac"), function(dr){
  p1 <- DimPlot(seurat,
                group.by = "orig.ident",
                reduction = dr) & NoAxes()
  p2 <- FeaturePlot(seurat,
                    c("PDGFB","CLDN5","PDGFRB","COL5A1"),
                    reduction = dr) & NoAxes() & NoLegend()
  p1 / p2
}))
```
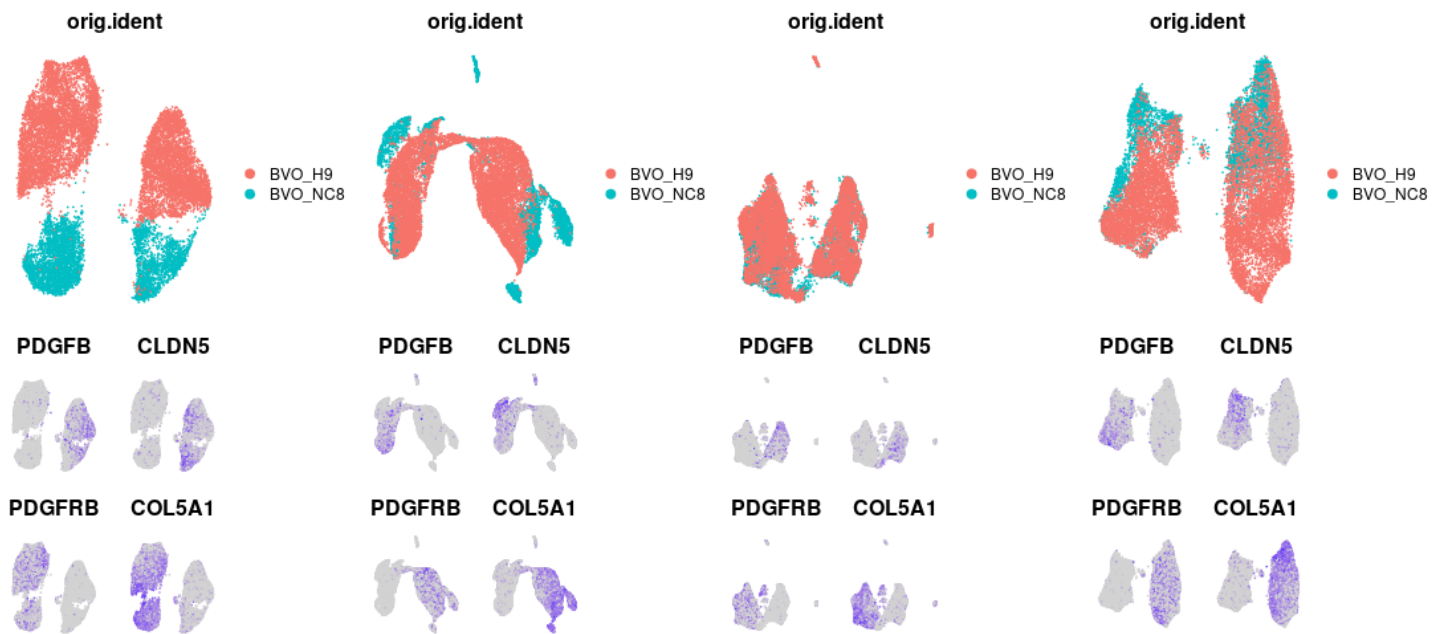
Generally speaking, all the three integration methods work to reduce the difference between the two replicates. In the following analysis, the CSS-integrated one would be used to be in line with the original publication.

With or without the data being integrated, we can also do clustering based on the ATAC data using the same procedure as for the RNA assay. Similarly, we can also do marker identification using the `FindMarkers` function in `Seurat` (recommended to use `test.use = 'LR'`, which also allows additional confounding covariates being considered by setting the `latent.vars` parameter), or the `wilcoxauc` function in `presto` for the fast Wilcoxon test. Here we don't show the details.

Section 2. Bi-modal integrative analysis of the RNA-ATAC scMultiome data

Since we have both the RNA and ATAC information in the scMultiome experiment, and they may have different capacity discriminating different cell populations, it would be good to try if we can integrate the information from both sides for the heterogeneity analysis. What's more, combining the RNA and ATAC information may also allow us to look deeper into the transcriptional regulation, for example, to identify potential transcription factors which are important to define cell type identities. In this section, we will briefly show how these analysis can be done with `Seurat` and `Signac` in R.

Step 1. Weighted nearest neighbor analysis

In `Seurat`, generating the UMAP embedding and doing clustering by taking both RNA and ATAC modalities into consideration is achievable via the weighted nearest neighbor graph method. The idea is to generate a k-nearest neighbor (kNN) network of cells, but this network is generated based on the information of both modalities. On the other hand, instead of simply averaging the two distances or unioning the kNN of each modality, it firstly calculates weights about how much each modality should contribute in defining neighbors for each cell. In brief, it does within-modal and cross-modal prediction by averaging the data of one modality (not the raw data, but the dimension reduced embedding) of its kNNs of the same (within-modal) or the other (cross-modal) modality. One modality gets a higher score in a cell if its within-modal prediction is better than the cross-modal prediction, and the scores of the two modalities are then normalized to be with sum of one. The method details can be seen in one of the Seurat methodology paper (Hao et al. 2021, Cell).

Therefore, the first step to do is to reconstruct the weighted nearest neighbor graph.

```
seurat <- FindMultiModalNeighbors(seurat,
                                  reduction.list = list("css_rna", "css_atac"),
                                  dims.list =
list(1:ncol(Embeddings(seurat,"css_rna")),

 1:ncol(Embeddings(seurat,"css_atac"))),
                                  modality.weight.name =
c("RNA.weight","ATAC.weight"),

                                  verbose = TRUE)
```

The `FindMultiModalNeighbors` function generates a neighbor network called "weighted.nn" by default and stores it at the `neighbors` slot of the Seurat object. It also generates two neighbor graphs called "wknn" and "wsnn" by default which are both stored at the `graphs` slot, just like the graphs generated by the `FindNeighbors` function when doing mono-modal analysis. The "weighted.nn" object can be then used as the input to generate the UMAP embedding, while the "wsnn" graph can be used for clustering. These operations are based on the two dimension reduction representation, each for one modality. In this example, we use the two CSS-integrated dimension reduction representations (`css_rna` and `css_atac`). When no integration is done, it should be then `pca` for the RNA assay and `lsi` for the ATAC assay, if the single-modal analysis is done with the default setting. Or when other integration methods are used, or the dimention reduction representations are named differently, just change here accordingly.

```
seurat <- RunUMAP(seurat, nn.name = "weighted.nn", assay = "RNA")
seurat <- FindClusters(seurat, graph.name = "wsnn", resolution = 0.2)

p1 <- UMAPPlot(seurat, group.by = "orig.ident") & NoAxes()
p2 <- UMAPPlot(seurat, group.by = "wsnn_res.0.2", label=T) & NoAxes()
p3 <- FeaturePlot(seurat,
                  c("PDGFB","CLDN5","PDGFRB","COL5A1"),
                  reduction = "umap") & NoAxes() & NoLegend()
p1 | p2 | p3
```
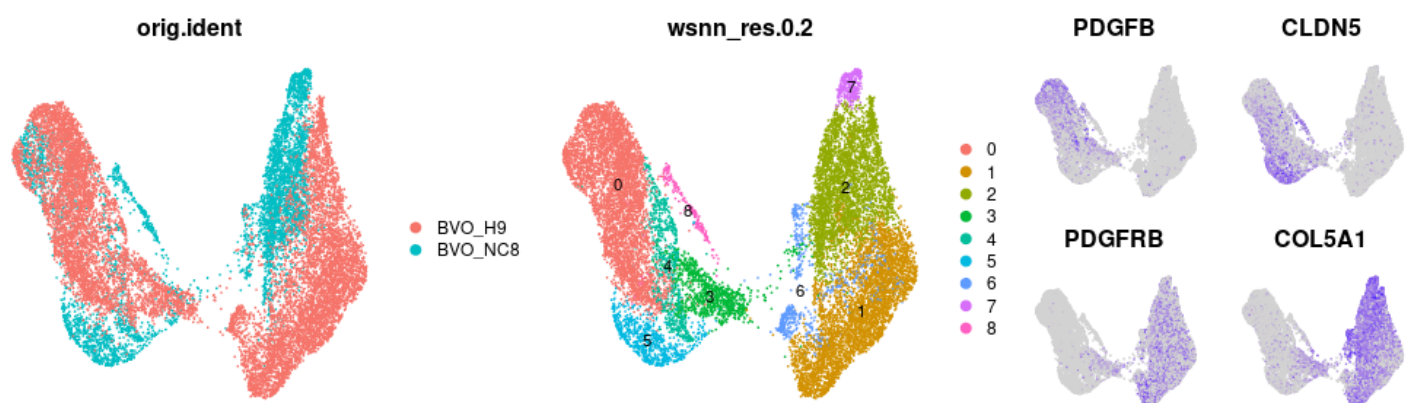

Next, we do a rough annotation of mural cells and endothelial cells in the data. The following analysis will focus on that level.

```
seurat$celltype <- setNames(rep(c("endothelial","mural","intermediate"), c(4,4,1)),
                            c(c(0,4,5,8),c(1,2,6,7),3))
[as.character(seurat$wsnn_res.0.2)]

p1 <- UMAPPlot(seurat, group.by = "celltype", label=T) & NoAxes()
p2 <- FeaturePlot(seurat,
                  c("PDGFB","CLDN5","PDGFRB","COL5A1"),
                  order=T,
                  reduction = "umap") & NoAxes() & NoLegend()
p1 | p2
```
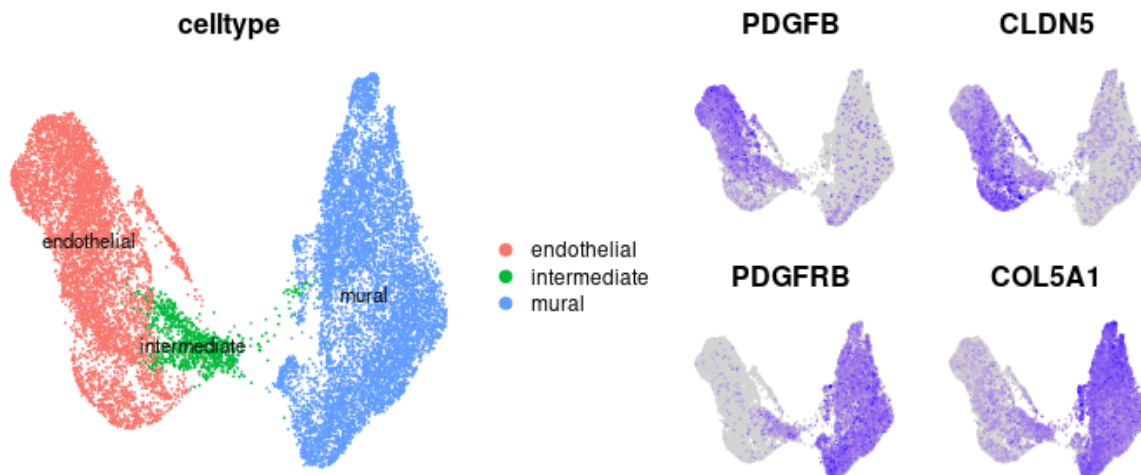

P.S. It is not always the best idea to generate UMAP representation and doing clustering using both modalities together. It really depends on the system and the question that one tries to answer. For instance, different cell lines may have their own genetic background and therefore some line-specific chromatin accessibility patterns that are independent from the cell types/states. In that case, if our focus is on the cell state transition and we realize that the unbiased ATAC analysis doesn't really provide much more information than the RNA assay only, doing UMAP embedding and clustering on RNA assay can also be a better option than the RNA-ATAC integrative counterpart.

Step 2. Cell type gene/peak marker identification and visualization of the chromatin accessibility profiles

With the cell type annotation we made, we can now apply differential expression analysis to compare each cell type versus the rest to identify gene markers for the cell type. Similarly, we can apply differential accessibility analysis to identify chromatin peaks which are accessible in a cell-type-specific manner. As mentioned above, both can be done via the `FindMarkers` function in `Seurat` (maybe with varied `test` parameters), or to be fast to use the `wilcoxauc` function in `presto` which relies on the non-parametric Wilcoxon test and the AUC values. Here we use `wilcoxauc` in this example script.

```r
library(presto)

DefaultAssay(seurat) <- "RNA"
DE_ct <- wilcoxauc(seurat, "celltype", seurat_assay = "RNA")
top_markers_ct <- DE_ct %>%
  filter(abs(logFC) > log(1.2) &
           padj < 0.01 &
           auc > 0.65 &
           pct_in - pct_out > 30 &
           pct_out < 20) %>%
  group_by(group) %>%
  top_n(10, wt = auc)


top_markers_ct
```

```
# A tibble: 20 x 10
# Groups:   group [2]
   feature     group      avgExpr logFC statistic   auc  pval  padj pct_in pct_out
   <chr>       <chr>        <dbl> <dbl>     <dbl> <dbl> <dbl> <dbl>  <dbl>   <dbl>
 1 GJA4        endothel…     1.44  1.23  51203790. 0.881     0     0   86.1    19.9
 2 RASGRP3     endothel…     1.84  1.66  53891328. 0.927     0     0   91.5    13.9
 3 DYSF        endothel…     1.34  1.18  51665930. 0.889     0     0   85.7    15.0
 4 AFAP1L1     endothel…     1.16  1.04  50454884. 0.868     0     0   80.7    12.5
 5 PTCHD4      endothel…     1.48  1.28  50265490. 0.865     0     0   82.8    15.6
 6 CSGALNACT1  endothel…     1.35  1.19  49954466. 0.859     0     0   80.1    14.2
 7 COL15A1     endothel…     1.37  1.24  51154980. 0.880     0     0   82.6    11.8
 8 MMRN2       endothel…     1.05  0.925 50117202. 0.862     0     0   80.8    13.2
 9 CDH5        endothel…     1.09  0.969 50567216. 0.870     0     0   81.5    12.8
10 PECAM1      endothel…     1.18  1.07  50964218  0.877     0     0   82.1    11.6
11 EPHB2       mural        0.931 0.813 46560530  0.773     0     0   62.9    13.1
12 PRRX1       mural         1.22  1.05  49455776. 0.821     0     0   73.7    17.7
13 COL6A3      mural         1.51  1.33  50756998. 0.843     0     0   77.1    17.4
14 KCNIP4      mural         2.46  2.18  50619454  0.841     0     0   76.4    16.8
15 PDGFRA      mural        0.991 0.863 47465138. 0.788     0     0   66.2    14.6
16 COL1A2      mural         1.08  0.925 45634088. 0.758     0     0   61.4    16.1
17 PCOLCE      mural         1.01  0.885 48950574. 0.813     0     0   70.9    15.5
18 SDC2        mural         1.04  0.883 48397369  0.804     0     0   70.6    17.6
19 NTRK2       mural        0.954 0.855 46458874. 0.772     0     0   61.1    10.1
20 COL1A1      mural         1.66  1.46  52758774  0.876     0     0   83.2    18.9
```

```r
DefaultAssay(seurat) <- "ATAC"
DA_ct <- wilcoxauc(seurat, "celltype", seurat_assay = "ATAC")
top_peaks_ct <- DA_ct %>%
  filter(abs(logFC) > log(1.1) &
           padj < 0.01 &
           auc > 0.55) %>%
  group_by(group) %>%
  top_n(100, wt = auc)

marker_peak_ct %>% top_n(5, wt=auc)
```

```
# A tibble: 15 x 10
# Groups:   group [3]
   feature       group   avgExpr logFC statistic   auc       pval       padj pct_in
   <chr>         <chr>     <dbl> <dbl>     <dbl> <dbl>      <dbl>      <dbl>  <dbl>
 1 chr1-348426… endoth…    0.789 0.691 39907049  0.687 0          0          42.7
 2 chr5-149288… endoth…    0.615 0.538 36367686. 0.626 0          0          29.0
 3 chr9-136673… endoth…    0.621 0.531 36325084. 0.625 0          0          29.5
 4 chr17-70218… endoth…    0.760 0.537 38520466. 0.663 0          0          46.3
 5 chr17-42319… endoth…    0.615 0.545 36235217  0.623 0          0          28.1
 6 chr5-141636… interm…    0.633 0.178  7735404. 0.573 1.34e- 19 5.48e- 16   48.2
 7 chr17-74987… interm…    0.622 0.165  7711786  0.571 1.31e- 18 2.92e- 15   49.0
 8 chr19-88316… interm…    0.620 0.238  7846815  0.581 3.51e- 30 7.92e- 25   39.8
 9 chr19-48362… interm…    0.647 0.176  7744355  0.573 3.36e- 19 9.99e- 16   51.5
10 chrX-241487… interm…    0.635 0.170  7717278  0.571 1.06e- 18 2.47e- 15   49.5
11 chr8-734192… mural      0.546 0.427 36218550. 0.602 7.22e-238 7.27e-235   26.5
12 chr16-85354… mural      0.547 0.439 36218620  0.602 4.47e-244 5.28e-241   26.0
13 chr16-85473… mural      0.576 0.426 36241576. 0.602 2.84e-219 2.01e-216   28.5
14 chr17-50209… mural      0.635 0.446 37388758. 0.621 2.68e-253 3.83e-250   35.6
15 chr20-38108… mural      0.593 0.490 37175143  0.617 4.62e-301 1.74e-297   29.1
# … with 1 more variable: pct_out <dbl>
```

Since we can obtain the RNA and ATAC profiles for the same cells, we can try to link a gene with its nearby peaks, which are its potential cis-regulatory elements. In `Signac`, the `LinkPeaks` function implements exactly that. What it does is to calculate Pearson correlation between the expression of a gene across cells, and the accessibility of every nearby peak. This coefficient is then compared to the expected correlation given the GC content, general accessibility and peak length to derive the p-value. Peaks with significant p-values are then considered as the linked peaks to the gene. This analysis is pretty time-consuming due to the large amount of linear regression model to fit (one per gene). Therefore here we limit it to the top cell type markers we just identified.

```
library(BSgenome.Hsapiens.UCSC.hg38)

seurat <- RegionStats(seurat,
                      genome = BSgenome.Hsapiens.UCSC.hg38)
seurat <- LinkPeaks(seurat,
                    peak.assay = "ATAC",
                    expression.assay = "RNA",
                    genes.use = top_markers_ct$feature)
```

P.S. The `RegionStats` function here is to calculate the GC content of peaks with the providing genome information in the `BSgenome.Hsapiens.UCSC.hg38` object. The function expects all the chromosomes in the ATAC assay should be included in the provided genome. Because the contig sequences are named differently in the Cell Ranger ARC reference than the standard UCSC-style hg38 genome, it would very likely return an error if ATAC peaks in those non-standard chromosomes are included. This is the reason why at the beginning we only keep the peaks in the standard chromosomes.

With the markers identified, we can of course do some feature plots for them, and we can also do more than just feature plots. You probably still remember the ATAC fragment file information we provided when creating the ATAC assay in the Seurat object. With that we can check the chromatin accessibility patterns in different cell types for a given region (can be a peak or a gene). Now let's check some of the top marker genes and peaks.

First let's check around the identified top marker genes RASGRP3 for endothelial cells and COL6A3 for mural cells. Since we've calculated the links between those marker genes and their nearby peaks, such information can also be shown in the same plot.

```r
DefaultAssay(seurat) <- "ATAC"
p1 <- CoveragePlot(seurat,
                   region = "RASGRP3",
                   features = "RASGRP3",
                   group.by = "celltype",
                   extend.upstream = 1000,
                   extend.downstream = 1000)
p2 <- CoveragePlot(seurat,
                   region = "COL6A3",
                   features = "COL6A3",
                   group.by = "celltype",
                   extend.upstream = 1000,
                   extend.downstream = 1000)
patchwork::wrap_plots(p1, p2, ncol = 1)
```
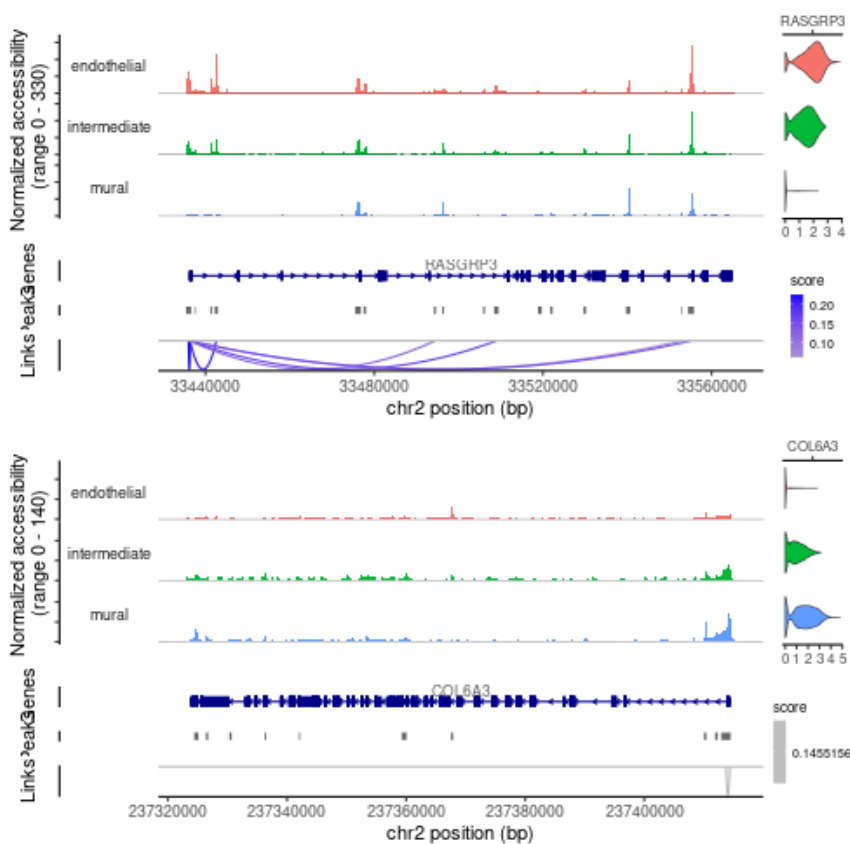


```r
p1 <- CoveragePlot(seurat,
                   region = "chr1-34842652-34844740",
                   group.by = "celltype",
                   extend.upstream = 1000,
                   extend.downstream = 1000)
p2 <- CoveragePlot(seurat,
                   region = "chr17-50209360-50211102",
                   group.by = "celltype",
                   extend.upstream = 1000,
                   extend.downstream = 1000)
patchwork::wrap_plots(p1, p2, ncol = 1)
```
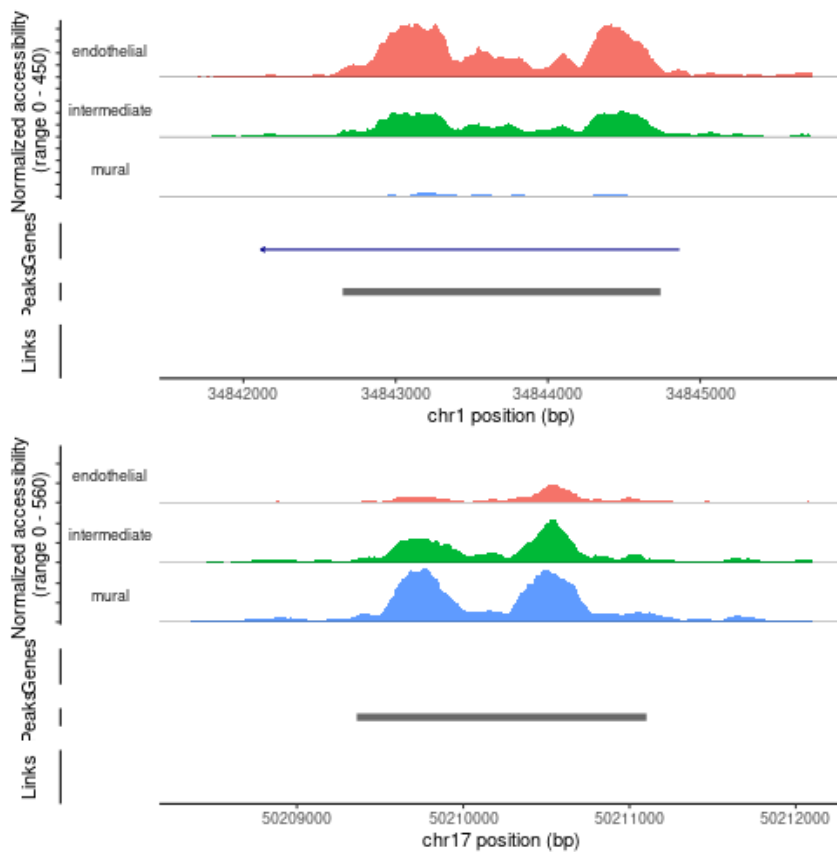
Step 3. TF binding motif enrichment analysis

The previous analysis can help us to identify putative cis-regulatory elements that are critical for regulating cell type identity or cell state transitions. This is usually achieved via the binding of certain trans-regulators, e.g. TFs, to those open chromatin regions. The binding of most TFs have strong sequence specificity, which can be summarized into sequence motifs, i.e. the TF binding motifs. If certain TFs play important roles in those regulations, it is very likely that the cell-type-specific peaks are regions enriched for the binding, with their genomic sequences enriched for the corresponding TF binding motifs. In this case, by checking the motifs enriched in those cell-type-specific peaks, we may be then able to identify TFs responsible for establishment and maintanence of cell type identity.

To do that, we also need a database of TF binding motifs. Indeed, there are several commonly used databases, including the commercial ones like TRANSFAC and open source ones like JASPAR. By scanning for sequences matching those motifs, we can predict possible binding sites of the TFs with binding motif information in the databases, and then check their enrichment in the peak list of interest in relative to the other peaks.

To do the motif scanning, we can use the `AddMotifs` function in `Signac`, given the list of binding motifs in the format of `PFMatrixList` defined in the `TFBSTools` package. We can retrieve the JASPAR database, and then convert it to the format in need.

```r
library(TFBSTools)
library(JASPAR2020)

pfm <- getMatrixSet(
  x = JASPAR2020,
  opts = list(collection = "CORE", tax_group = 'vertebrates', all_versions = FALSE)
)
df_pfm <- data.frame(t(sapply(pfm, function(x)
  c(id=x@ID, name=x@name,
symbol=ifelse(!is.null(x@tags$symbol),x@tags$symbol,NA)))))

seurat <- AddMotifs(seurat, genome = BSgenome.Hsapiens.UCSC.hg38, pfm = pfm)
```

P.S. The `AddMotifs` function is a bit time-consuming, as it needs to look through the sequence of all peaks in the peak list of the ATAC assay given the genome information in the `BSgenome.Hsapiens.UCSC.hg38` object. This is another reason why at the beginning we only keep the peaks in the standard chromosomes, as the sequences of peaks at the non-standard chromosome will be likely missing due to the different naming style of those contigs.

Now we can use the `FindMotifs` function in `Signac` to identify enriched TF binding motifs in a provided peak list, for instance, the cell-type-specific accessible peaks. Similar to the functional enrichment analysis for inferring possible gene set function, the motif enrichment analysis also requires a background set. By default, it randomly selects 40000 peaks from the whole peak list. Alternatively, we can define the background peak list based on their accessibility in certain cell types, for instance, as well as peaks with matched GC content as the peak list of interest.

```r
open_peaks <- AccessiblePeaks(seurat)
peaks_matched <- MatchRegionStats(meta.feature =
seurat[['ATAC']]@meta.features[open_peaks, ],
                                  query.feature =
seurat[['ATAC']]@meta.features[marker_peak_ct$feature, ],
                                  n = 50000)

motif_enrichment_endo <- FindMotifs(seurat,
                                    features =
top_peaks_ct$feature[top_peaks_ct$group == "endothelial"],
                                    background = peaks_matched) %>%
  mutate(symbol = setNames(ifelse(is.na(df_pfm$symbol), df_pfm$name,
df_pfm$symbol), df_pfm$id)[motif]) %>%
  mutate(padj = p.adjust(pvalue, method="BH"))
enriched_motif_endo <- motif_enrichment_endo %>%
  filter(padj < 0.01 & fold.enrichment > 3) %>%
  top_n(-4, wt = padj)

motif_enrichment_mural <- FindMotifs(seurat,
                                     features =
top_peaks_ct$feature[top_peaks_ct$group == "mural"],
                                     background = peaks_matched) %>%
  mutate(symbol = setNames(ifelse(is.na(df_pfm$symbol), df_pfm$name,
df_pfm$symbol), df_pfm$id)[motif]) %>%
  mutate(padj = p.adjust(pvalue, method="BH"))
enriched_motif_mural <- motif_enrichment_mural %>%
  filter(padj < 0.01 & fold.enrichment > 3) %>%
  top_n(-4, wt = padj)
```
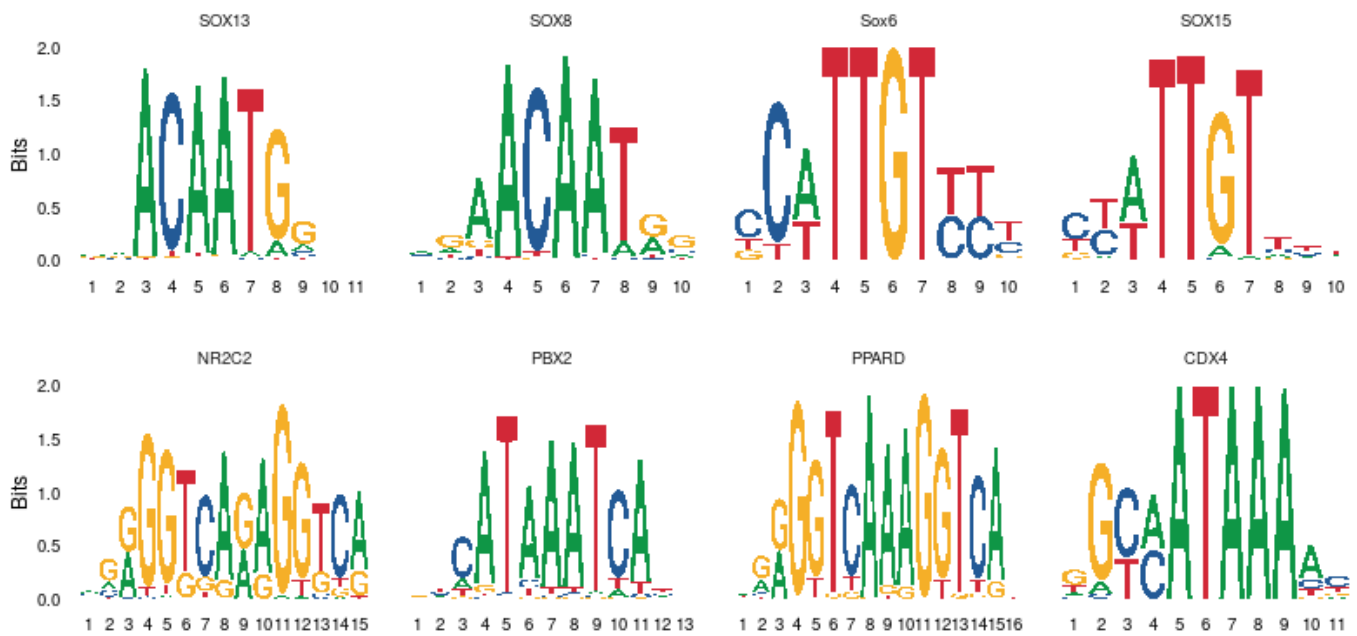
Now we can visualize the top-4 overrepresented TF binding motifs in endothelial cells (top) and mural cells (botton)

```r
p1 <- MotifPlot(seurat, motifs = enriched_motif_endo$motif[1:4], ncol=4)
p2 <- MotifPlot(seurat, motifs = enriched_motif_mural$motif[1:4], ncol=4)
p1 / p2
```

We can also check the expression of the corresponding TFs with feature plots.

```r
DefaultAssay(seurat) <- "RNA"
p1 <- FeaturePlot(seurat,
                  c("SOX13","SOX8","SOX6","SOX15"),
                  reduction="umap",
                  order=T,
                  ncol=4) & NoAxes() & NoLegend()
p2 <- FeaturePlot(seurat,
                  c("NR2C2","PBX2","PPARD","CDX4"),
                  reduction="umap",
                  order=T,
                  ncol=4) & NoAxes() & NoLegend()
p1 / p2
```
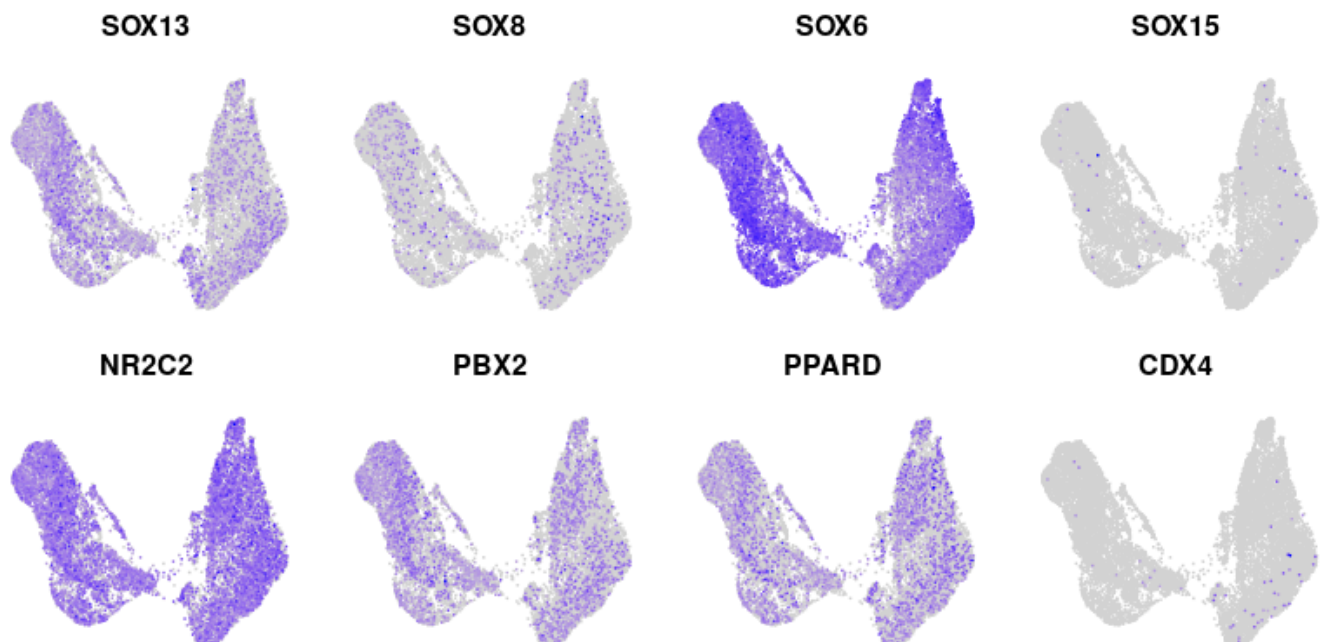
Unfortunately the expression of these eight TFs don't look very cell type specific. There are different possibilities here. It could be that these enrichments are not biological relevant and their corresponding TFs are not the critical factors in defining the endothelial and mural cell identities. The other possibility is that some ubiquitously expressed TFs can bind to cell-type-specific enhancers and therefore regulate different downstream target genes in different cell types. One needs to check in more details, and also check more motifs instead of just the top-4.

Step 4. ChromVAR: another motif enrichment analysis

Another way of motif enrichment analysis which doesn't require the input of a specific list of peaks of interest is the chromVAR analysis, developed by the Greenleaf lab in Stanford University. This method estimates motif activity scores for each motif on cell level. In brief, for each motif in each cell, it counts the number of ATAC fragments mapped to those peaks, and normalizes by the expected numbers based on the average of all cells to get the raw accessibility score. Next, a background peak set is created with comparable GC content and average accessibility as the motif peaks, and to obtain similar raw accessibility score for the background. The background accessibility score is then used to normalize the accessibility score of the motif to get the corrected accessibility score of the motif in each cell. More details can be seen in the chromVAR paper.

In `Signac` , a wrapper function to run chromVAR on the ATAC assay is implemented as `RunChromVAR` . It requires the `chromVAR` package being installed (via Bioconductor with `BiocManager::install("chromVAR")`).

```
seurat <- RunChromVAR(seurat, genome = BSgenome.Hsapiens.UCSC.hg38)
```

This creates a new assay in the Seurat object, namely "chromvar" by default. It contains the JASPAR motifs we added to the Seurat object above as the features, and the chromVAR estimated accessibility scores of those motifs in all cells. We can then do the differential activity analysis between cell types to identify motifs that show cell type specific enrichment. Similarly, this can be done with `FindMarkers` in `Seurat` , `wilcoxauc` in `presto` or other statistical tests.

```
DefaultAssay(seurat) <- "chromvar"
DA_motifs_ct <- wilcoxauc(seurat, group_by = "celltype", seurat_assay = "chromvar")
%>%
  mutate(symbol = setNames(ifelse(is.na(df_pfm$symbol), df_pfm$name,
df_pfm$symbol),
                           df_pfm$id)[feature])

enriched_motifs_ct <- DA_motifs_ct %>%
  filter(padj < 0.01 & auc > 0.7) %>%
  group_by(group)
top_motifs_ct <- top_n(enriched_motifs_ct, 3, wt=auc)
```

We can use feature plots to visualize the enrichment profiles directly.

```
bluered_colscheme <-
colorRampPalette(rev(c("#d73027","#f46d43","#fdae61","#fee090","#e0f3f8","#abd9e9",
"#74add1","#4575b4")))
FeaturePlot(seurat,
            features = top_motifs_ct$feature,
            cols = bluered_colscheme(30),
            reduction = "umap",
            ncol = 3) & NoAxes() & NoLegend()
```

MA0077.1 **MA1152.1** **MA0868.2**

MA1473.1 **MA0878.2** **MA1640.1**

This analysis result can be further looked into together with the DE analysis on TFs between cell types. We can obtain the list of TFs from the animalTFDB database. Here we provide the list as a part of the tutorial as well. What we can do then is to further filter the marker TFs requiring their motifs being enriched in the same cell type.

```r
tfs <- read.table("extdata/Homo_sapiens_TF.txt", sep="\t", header=T)

tf_motifs_ct <- enriched_motifs_ct %>%
  filter(symbol %in% tfs$Symbol)
marker_tfs_ct <- DE_ct %>%
  filter(feature %in% tfs$Symbol &
           abs(logFC) > log(1.2) &
           padj < 0.01 &
           auc > 0.65 &
           pct_in - pct_out > 20) %>%
  inner_join(tf_motifs_ct,
             by = c("feature" = "symbol"),
             suffix = c("_tf","_motif")) %>%
  filter(group_tf == group_motif)

top_tfs_ct <- group_by(marker_tfs_ct, group_tf) %>%
  top_n(3, wt = auc_motif)

beach_colscheme <-
colorRampPalette(c("#cdcdcd","#edf8b1","#7fcdbb","#41b6c4","#1d91c0","#225ea8","#0c
2c84"))
DefaultAssay(seurat) <- "RNA"
p1 <- FeaturePlot(seurat,
                  top_tfs_ct$feature,
                  reduction = "umap",
                  order=T,
                  cols=beach_colscheme(30),
                  ncol=6) & NoAxes() & NoLegend()
DefaultAssay(seurat) <- "chromvar"
p2 <- FeaturePlot(seurat,
                  top_tfs_ct$feature_motif,
                  reduction = "umap",
                  order=T,
                  cols=bluered_colscheme(30),
                  ncol=6) & NoAxes() & NoLegend()
p1 / p2
```
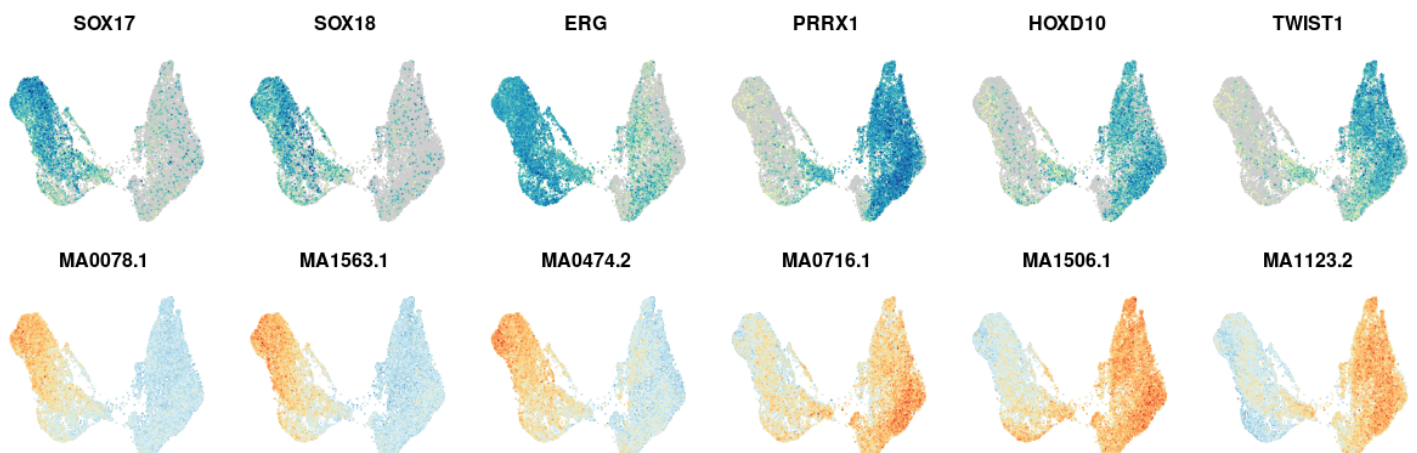
Section 3. Gene regulatory network reconstruction

In the previous section, we show some examples of how we can integrate the RNA and ATAC assays to identify cell type heterogeneity in the data, as well as to look for putative cis- and trans-regulatory elements that could be important for cell type/state identity establishment, maintenance and transition. Meanwhile, there is another way of looking at the data, that is, instead of focusing on different cell populations, we focus on relationship between genes, or more specifically, to identify the regulatory relationship of different TFs interacting with different cis-regulatory elements in order to regulate the transcription of different target genes. This is so-call the gene regulatory network (GRN) analysis.

How to accurately and quantitatively reconstruct a GRN is a tough question and still remained to be answered in a nice way. Still, people have spent lots of effort and developed many tools aiming to solve the problem at least partially. One of the first attempts is to check expression correlation between different genes, and therefore to reconstruct a co-expression network followed by co-expression modules identification. Each of the co-expression modules contains genes that are supposed to be regulated together, whose regulators can be hopefully identified via TF binding motif enrichment analysis, for instance. One typical example of this type of methods is WGCNA, short for Weighted Gene Coexpression Network Analysis, which is widely used in bulk transcriptome data analysis.

Another strategy is to consider the expression of one gene to be a function of the expression of its regulators (mostly TFs). The solution is then to fit a linear or non-linear regression model using the target gene expression in different samples as the response, while the expression of all possible regulators in samples as covariates. This is usually followed by statistical test to determine whether any covariate contributes to the target gene expression significantly, or includes regularization for feature selection. Examples of this type of method include GENIE3 and GRNBoost2. Many of those methods have also been shown being applicable in single-cell transcriptomic data.

Although those methods can provide very useful information of possible regulatory mechanisms,they purely rely on expression of TFs and putative target genes, ignoring the biology that the target gene needs to contain the binding motif of a certain TF at its cis-regulatory elements, either its promoter or enhancers for the TF to regulate. To further improve and push the analysis closer to biology, people started to build up pipelines which takes into account both expression and TF binding site prediction. SCENIC, developed by the Aerts lab in KU Leuven Center for Human Genetics and the VIB Center for Brain and Disease Research that also developed GRNBoost2, implements a filtering step after inferring the raw network with GRNBoost2, excluding the inferred regulations that lack of corresponding TF binding motif at the target gene's promoter region. More information can be seen in its paper. CellOracle implements this in a different way, by firstly looking for putative TFs of a gene via motif searching at its putative promoter and enhancers based on public data or additional ATAC-seq data, for instance, and then use a regularized linear regression model that only contains TFs with predicted binding motifs to further infer the relationship between TFs and the target gene quantitatively. More information can be found in its preprint.

Now with the scMultiome data, we have the chance to even quantitatively involve the chromatin accessibility profile to the model to infer GRN. That is the rationale behind Pando. It firstly scans for TF binding motifs at each peak in the ATAC assay of the scMultiome data, with further filtering based on sequence conservation and public regulatory element databases (e.g. ENCODE), and then for each gene, build a linear regression model considering not only the expression of those TFs with any predicted binding sites that may regulate the gene, but also its interaction with the peak accessibility that the putative TF binding site is located. More details can be read in the biorxiv preprint. In the following part of the tutorial, we will briefly show how to run Pando in the scMultiome data to identify potential targets of TFs.

P.S. Pando had a major upgrade to change its defined data structure, which was to avoid possible effect by any major change to the data structure of Seurat objects in the future. The following tutorial session has therefore been updated to adapt to the changes.

Pando can be installed via `devtools::install_github("quadbio/Pando")`. Once the installation is done we can import the package into the R environment.

```r
library(Pando)
```

Once Pando is imported, we need to initialize a Pando-compatible Seurat object. This step also insert the sequence conservation information in the form of phastCons scores, which is derived from mult-species whole genome alignment and ranged between 0 and 1, with 1 meaning identical in all genomes.

```r
grn <- initiate_grn(seurat,
                    regions=phastConsElements20Mammals.UCSC.hg38,
                    rna_assay = "RNA", peak_assay = "ATAC")
```

Next we need to incorporate the TF binding motif information into the object. We can use the JASPAR database as mentioned above, or Pando also contains an extended database that contains not only the information in JASPAR, but also some extra motif information from CIS-BP database as well as the sequence-similarity-based transferred motifs for TFs without any annotated motif information in either JASPAR or CIS-BP. Here we directly use the Pando-extended database.

```r
grn <- find_motifs(grn,
                   pfm = Pando::motifs,
                   motif_tfs = Pando::motif2tf,
                   genome = BSgenome.Hsapiens.UCSC.hg38)
```

This function calls the `AddMotifs` function in `Signac` to scan for TF binding motifs at the peaks.

Now we can start to run Pando. Pando supports any multithreads parallelization method in R that implements the `foreach` function. Here we use the `doParallel` package to run `Pando` with 20 cores. Do modify this number if it is too much for you. If you don't need parallelization at all, skip the first two lines and directly run `infer_grn`.

```r
library(doParallel)
registerDoParallel(20)
grn <- infer_grn(grn,
                 peak_to_gene_method='Signac',
                 parallel = T,
                 tf_cor = 0.05,
                 method="glm",
                 family="gaussian",
                 scale=F,
                 verbose=T)
```

The important parameters of the `infer_grn` function includes `tf_cor` which pre-filter only TFs with sufficient correlation with the gene in the model. `peak_cor` does similar pre-filtering but for the peaks. `peak_to_gene_method`, `upstream`, `downstream`, and `extend` together determine which peaks are linked to a gene. The `method` parameter claims the regression model to use. More information can be seen in the function manual.

This step will take a while. Once it is done, we can extract the TF-peak-target trios with significant p-values.

```
coef_grn <- coef(grn_object) %>%
  filter(padj < 0.01)

coef_grn
```

```
# A tibble: 10,177 x 9
   tf      target  region       term  estimate std_err statistic     pval     padj
   <chr>   <chr>   <chr>        <chr>    <dbl>   <dbl>     <dbl>    <dbl>    <dbl>
 1 TRPS1   NLGN4X  chrX-58106…  chrX… -0.0335  0.00901    -3.71 2.05e- 4 3.56e- 3
 2 MECOM   STS     chrX-71095…  chrX…  0.0152  0.00169     8.96 3.63e-19 2.89e-17
 3 PBX3    ANOS1   chrX-87322…  chrX…  0.00829 0.00183     4.54 5.67e- 6 1.51e- 4
 4 ZNF800  SHROOM2 chrX-97658…  chrX…  0.0591  0.0141      4.19 2.79e- 5 6.29e- 4
 5 ZNF800  SHROOM2 chrX-97869…  ZNF8…  0.0551  0.0130      4.23 2.32e- 5 5.33e- 4
 6 PRDM16  SHROOM2 chrX-97890…  chrX…  0.0729  0.0211      3.46 5.48e- 4 8.09e- 3
 7 SOX4    SHROOM2 chrX-98567…  chrX…  0.0271  0.00634     4.27 1.96e- 5 4.60e- 4
 8 TEAD1   SHROOM2 chrX-99117…  chrX…  0.0230  0.00593     3.88 1.05e- 4 1.99e- 3
 9 ZFPM2   MID1    chrX-10618…  ZFPM…  0.0380  0.00910     4.17 3.00e- 5 6.69e- 4
10 GATA4   MID1    chrX-10619…  chrX…  0.0753  0.0157      4.79 1.73e- 6 5.12e- 5
# … with 10,167 more rows
```

With this information we can now generate the list of regulons, which are genes that are co-regulated positively or negatively by the same TF. Afterwards, we can use the `AddModuleScore` in `Seurat` to calculate the regulon activity score for each regulon in each cell. This information can be then incorporated into the Seurat object as an extra assay (in the example we call it 'regulon').

```
grn_object <- find_modules(grn_object,
                           p_thresh = 0.01)
regulons <- NetworkModules(grn_object)

positive_regulons <- regulons@features[['genes_pos']]
positive_regulons <- positive_regulons[lengths(positive_regulons) > 10]
negative_regulons <- regulons@features[['genes_neg']]
negative_regulons <- negative_regulons[lengths(negative_regulons) > 10]

DefaultAssay(seurat) <- "RNA"
mod_act_pos <- AddModuleScore(seurat,
                              features = positive_regulons,
                              name = "regulon_")@meta.data
mod_act_pos <- mod_act_pos[,grep("^regulon_", colnames(mod_act_pos))] %>%
  setNames(paste0(names(positive_regulons),"(+)"))
mod_act_neg <- AddModuleScore(seurat,
                              features = negative_regulons,
                              name = "regulon_")@meta.data
mod_act_neg <- mod_act_neg[,grep("^regulon_", colnames(mod_act_neg))] %>%
  setNames(paste0(names(negative_regulons),"(-)"))

seurat[['regulon']] <- CreateAssayObject(data = t(cbind(mod_act_pos, mod_act_neg)))
```

Now we can do the feature plots for the same TFs we found combining DE and chromVAR analysis, and their positive and negative regulons.

```r
DefaultAssay(seurat) <- "RNA"
p1 <- FeaturePlot(seurat,
                  top_tfs_ct$feature,
                  reduction = "umap",
                  cols = beach_colscheme(30),
                  order = T,
                  ncol = 6) & NoAxes() & NoLegend()
DefaultAssay(seurat) <- "regulon"
p2 <- FeaturePlot(seurat,
                  features = c(intersect(paste0(top_tfs_ct$feature,"(+)"),
rownames(seurat)),
                              intersect(paste0(top_tfs_ct$feature,"(-)"),
rownames(seurat))),
                  reduction = "umap",
                  cols = bluered_colscheme(30),
                  order = T,
                  ncol = 6) & NoAxes() & NoLegend()
(p1 / p2) + patchwork::plot_layout(height = c(1,2))
```
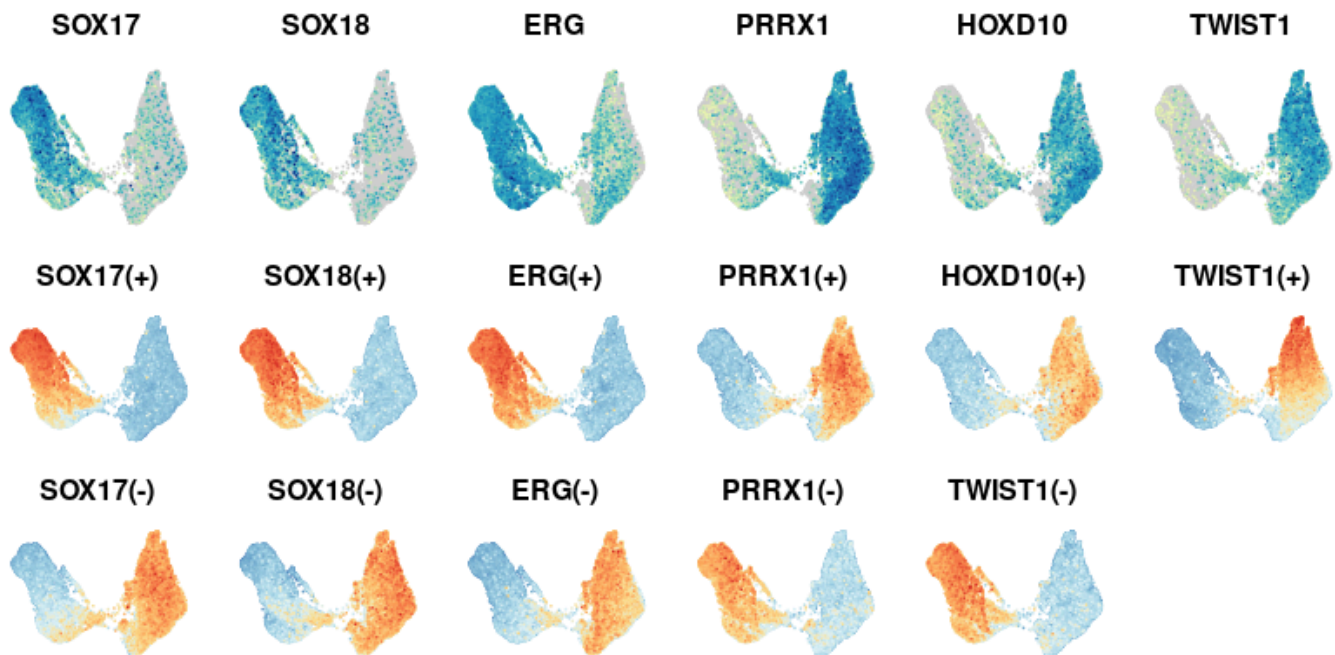

There are much more analysis, particularly those graph-based analysis, that can be done to the GRN, such as centrality analysis to identify TFs which may trigger huge regulatory cascades. One can also visualize the TF cross-regulatory network.