BIOST 2154 Lab 6: Signac

Wenzhuo Lin

2025-10-15

Contents

In this lab section, I will talk about Signac package. The content is primarily adapted from Signac tutorial. And the dataset we analyzed today is a single-cell ATAC-seq dataset of human peripheral blood mononuclear cells (PBMCs) provided by 10x Genomics. All the datasets equiped at crc and the absolute path for it is `/ix1/biost2154-2025f/lab/lab6`.

And load the packages we would use.

```
library(Signac)
library(Seurat)
library(GenomicRanges)
library(ggplot2)
library(patchwork)
```

If there are any missing packages, please find the command to install the missing ones.

```
## Seurat, ggplot2, patchwork
install.packages(c("Seurat", "ggplot2", "patchwork"))

## GenomicRanges
if (!requireNamespace("BiocManager", quietly = TRUE))
    install.packages("BiocManager")
BiocManager::install("GenomicRanges")

## Signac
setRepositories(ind=1:3) # needed to automatically install Bioconductor dependencies
install.packages("Signac")
```

Pre-processing workflow

When pre-processing chromatin data, Signac uses information from two related input files, both of which can be created using CellRanger:

- **Peak/Cell matrix**. This is analogous to the gene expression count matrix used to analyze single-cell RNA-seq. However, instead of genes, each row of the matrix represents a region of the genome (a peak), that is predicted to represent a region of open chromatin. Each value in the matrix represents the number of Tn5 integration sites for each single barcode that map within each peak. You can find more detail from 10X Genomics.
- **Fragment file**. This represents a full list of all unique fragments across all single cells. It is a substantially larger file, is slower to work with, and is stored on-disk (instead of in memory). However, the advantage of retaining this file is that it contains all fragments associated with each single cell, as opposed to only fragments that map to peaks. You can find more detail from 10X Genomics.

We start by creating a `Seurat` object using the peak/cell matrix and cell metadata generated by cellranger-atac, and store the path to the fragment file on disk in the Seurat object:

```
counts <- Read10X_h5(filename = "/ix1/biost2154-2025f/lab/lab6/10k_pbmc_ATACv2_nextgem_Chromium_Controll
metadata <- read.csv(
  file = "/ix1/biost2154-2025f/lab/lab6/10k_pbmc_ATACv2_nextgem_Chromium_Controller_singlecell.csv",
  header = TRUE,
  row.names = 1
)
```

```
chrom_assay <- CreateChromatinAssay(
  counts = counts,
  sep = c(":", "-"),
  fragments = "/ix1/biost2154-2025f/lab/lab6/10k_pbmc_ATACv2_nextgem_Chromium_Controller_fragments.tsv.g
  min.cells = 1500, # 10
```

```
  min.features = 5000 # 200
)
```

```
## Computing hash
```

```
pbmc <- CreateSeuratObject(
  counts = chrom_assay,
  assay = "peaks",
  meta.data = metadata
)
```

```
pbmc
```

```
## An object of class Seurat
## 18657 features across 9657 samples within 1 assay
## Active assay: peaks (18657 features, 0 variable features)
##  2 layers present: counts, data
```

The ATAC-seq data is stored using a custom assay, the `ChromatinAssay`. This enables some specialized functions for analysing genomic single-cell assays such as scATAC-seq. By printing the assay we can see some of the additional information that can be contained in the `ChromatinAssay`, including motif information, gene annotations, and genome information.

```
pbmc[['peaks']]
```

```
## ChromatinAssay data with 18657 features for 9657 cells
## Variable features: 0
## Genome:
## Annotation present: FALSE
## Motifs present: FALSE
## Fragment files: 1
```

Question: How to read count data in scRNA-seq in Seurat?

Then we can call granges to see the genomic ranges associated with each feature in the object.

```
granges(pbmc)
```

```
## GRanges object with 18657 ranges and 0 metadata columns:
##             seqnames        ranges strand
##                <Rle>     <IRanges>  <Rle>
##       [1]       chr1 778263-779184      *
##       [2]       chr1 827056-827941      *
##       [3]       chr1 869458-870366      *
##       [4]       chr1 904344-905188      *
##       [5]       chr1 958863-959759      *
##       ...        ...           ...    ...
##   [18653] GL000195.1   32220-33058      *
##   [18654] GL000219.1   39972-40871      *
##   [18655] GL000219.1   42166-43046      *
##   [18656] GL000219.1  99241-100144      *
##   [18657] KI270713.1   21459-22358      *
##   -------
##   seqinfo: 35 sequences from an unspecified genome; no seqlengths
```

We then remove the features that correspond to chromosome scaffolds e.g. (KI270713.1) or other sequences instead of the (22+2) standard chromosomes.

```
peaks.keep <- seqnames(granges(pbmc)) %in% standardChromosomes(granges(pbmc))
pbmc <- pbmc[as.vector(peaks.keep), ]
```

We can also add gene annotations to the `pbmc` object for the human genome. This will allow downstream functions to pull the gene annotation information directly from the object.

Multiple patches are released for each genome assembly. When dealing with mapped data (such as the 10x Genomics files we will be using), it is advisable to use the annotations from the same assembly patch that was used to perform the mapping.

Checkpoint 1

You can read the `pbmc` object from `/ix1/biost2154-2025f/lab/lab6/pbmc_checkpoint1.rds` if you fail to construct the object from raw data.

```
library(AnnotationHub)
ah <- AnnotationHub()

# Search for the Ensembl 98 EnsDb for Homo sapiens on AnnotationHub
query(ah, "EnsDb.Hsapiens.v98")
```

```
## AnnotationHub with 1 record
## # snapshotDate(): 2025-04-08
## # names(): AH75011
## # $dataprovider: Ensembl
## # $species: Homo sapiens
## # $rdataclass: EnsDb
## # $rdatadateadded: 2019-05-02
## # $title: Ensembl 98 EnsDb for Homo sapiens
## # $description: Gene and protein annotations for Homo sapiens based on Ensem...
## # $taxonomyid: 9606
## # $genome: GRCh38
## # $sourcetype: ensembl
## # $sourceurl: http://www.ensembl.org
## # $sourcesize: NA
## # $tags: c("98", "AHEnsDbs", "Annotation", "EnsDb", "Ensembl", "Gene",
## #   "Protein", "Transcript")
## # retrieve record with 'object[["AH75011"]]'
```

Then read the record.

```
ensdb_v98 <- ah[["AH75011"]]
```

```
## loading from cache
```

```
## require("ensembldb")
```

```
# extract gene annotations from EnsDb
annotations <- GetGRangesFromEnsDb(ensdb = ensdb_v98)

# change to UCSC style since the data was mapped to hg38
seqlevels(annotations) <- paste0('chr', seqlevels(annotations))
genome(annotations) <- "hg38"
```

Checkpoint annotation

You can read the `annotations` object from `/ix1/biost2154-2025f/lab/lab6/annotations.rds`. If you begin your job from here, please also load the `pbmc` object from `/ix1/biost2154-2025f/lab/lab6/pbmc_checkpoint1.rds`.

```
# add the gene information to the object
Annotation(pbmc) <- annotations
```

Quality Control

We then compute some QC metrics for the scATAC-seq experiment.

- **Nucleosome banding pattern**: The histogram of DNA fragment sizes (determined from the paired-end sequencing reads) should exhibit a strong nucleosome banding pattern corresponding to the length of DNA wrapped around a single nucleosome. We calculate this per single cell, and quantify the approximate ratio of mononucleosomal to nucleosome-free fragments (stored as nucleosome_signal)
- **Transcriptional start site (TSS) enrichment score**: The ENCODE project has defined an ATAC-seq targeting score based on the ratio of fragments centered at the TSS to fragments in TSS-flanking regions. Poor ATAC-seq experiments typically will have a low TSS enrichment score. We can compute this metric for each cell with the `TSSEnrichment()` function, and the results are stored in metadata under the column name `TSS.enrichment`.
- **Total number of fragments in peaks**: A measure of cellular sequencing depth / complexity. Cells with very few reads may need to be excluded due to low sequencing depth. Cells with extremely high levels may represent doublets, nuclei clumps, or other artefacts.
- **Fraction of fragments in peaks**: Represents the fraction of all fragments that fall within ATAC-seq peaks. Cells with low values (i.e. <15-20%) often represent low-quality cells or technical artifacts that should be removed. Note that this value can be sensitive to the set of peaks used.
- **Ratio reads in genomic blacklist regions**: The ENCODE project has provided a list of blacklist regions, representing reads which are often associated with artefactual signal. Cells with a high proportion of reads mapping to these areas (compared to reads mapping to peaks) often represent technical artifacts and should be removed. The `FractionCountsInRegion()` function can be used to calculate the fraction of all counts within a given set of regions per cell. We can use this function and the blacklist regions to find the fraction of blacklist counts per cell.

```
# compute nucleosome signal score per cell
pbmc <- NucleosomeSignal(object = pbmc)

# compute TSS enrichment score per cell
pbmc <- TSSEnrichment(object = pbmc)
```

```
## Extracting TSS positions

## Extracting fragments at TSSs

##
## Computing TSS enrichment score
```

```
# add fraction of reads in peaks
pbmc$pct_reads_in_peaks <- pbmc$peak_region_fragments / pbmc$passed_filters * 100

# add blacklist ratio
pbmc$blacklist_ratio <- FractionCountsInRegion(
  object = pbmc,
  assay = 'peaks',
  regions = blacklist_hg38_unified
)
```

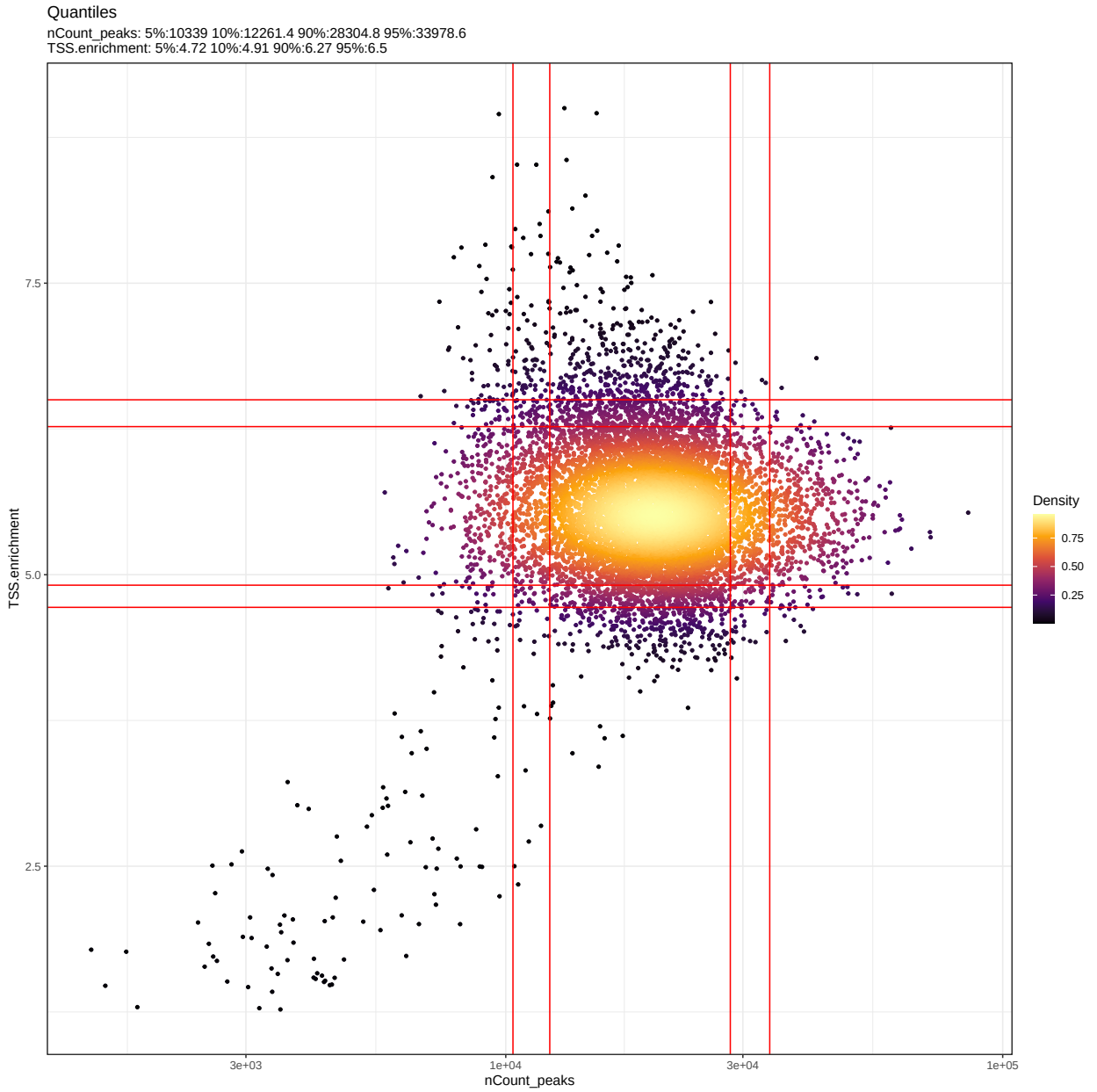
Checkpoint 2

You can read the `pbmc` object from `/ix1/biost2154-2025f/lab/lab6/pbmc_checkpoint2.rds`.

The relationship between variables stored in the object metadata can be visualized using the `DensityScatter()` function. This can also be used to quickly find suitable cutoff values for differ-

ent QC metrics by setting `quantiles=TRUE`:

```
DensityScatter(pbmc, x = 'nCount_peaks', y = 'TSS.enrichment', log_x = TRUE, quantiles = TRUE)
```



Finally we remove cells that are outliers for these QC metrics. The exact QC thresholds can be adjusted.

```
pbmc <- subset(
  x = pbmc,
  subset = nCount_peaks > 9000 &
    nCount_peaks < 100000 &
    pct_reads_in_peaks > 40 &
    blacklist_ratio < 0.01 &
    nucleosome_signal < 4 &
    TSS.enrichment > 4
)
```

```
pbmc
```

```
## An object of class Seurat
## 18649 features across 9367 samples within 1 assay
## Active assay: peaks (18649 features, 0 variable features)
##  2 layers present: counts, data
```

Normalization

- **Normalization**: Signac performs term frequency-inverse document frequency (TF-IDF) normalization. This is a two-step normalization procedure, that both normalizes across cells to correct for differences in cellular sequencing depth, and across peaks to give higher values to more rare peaks.
- **Feature selection**: The low dynamic range of scATAC-seq data makes it challenging to perform variable feature selection. Instead, we can choose to use only the top n% of features (peaks) for dimensional reduction, or remove features present in less than n cells with the `FindTopFeatures()` function. Here we will use all features, though we have seen very similar results when using only a subset of features (try setting `min.cutoff` to 'q75' to use the top 25% all peaks), with faster runtimes. Features used for dimensional reduction are automatically set as `VariableFeatures()` for the Seurat object by this function.
- **Dimension reduction**: We next run singular value decomposition (SVD) on the TD-IDF matrix, using the features (peaks) selected above. This returns a reduced dimension representation of the object.

```
pbmc <- RunTFIDF(pbmc)
```

```
## Performing TF-IDF normalization
```

```
pbmc <- FindTopFeatures(pbmc, min.cutoff = 'q50')
pbmc <- RunSVD(pbmc)
```

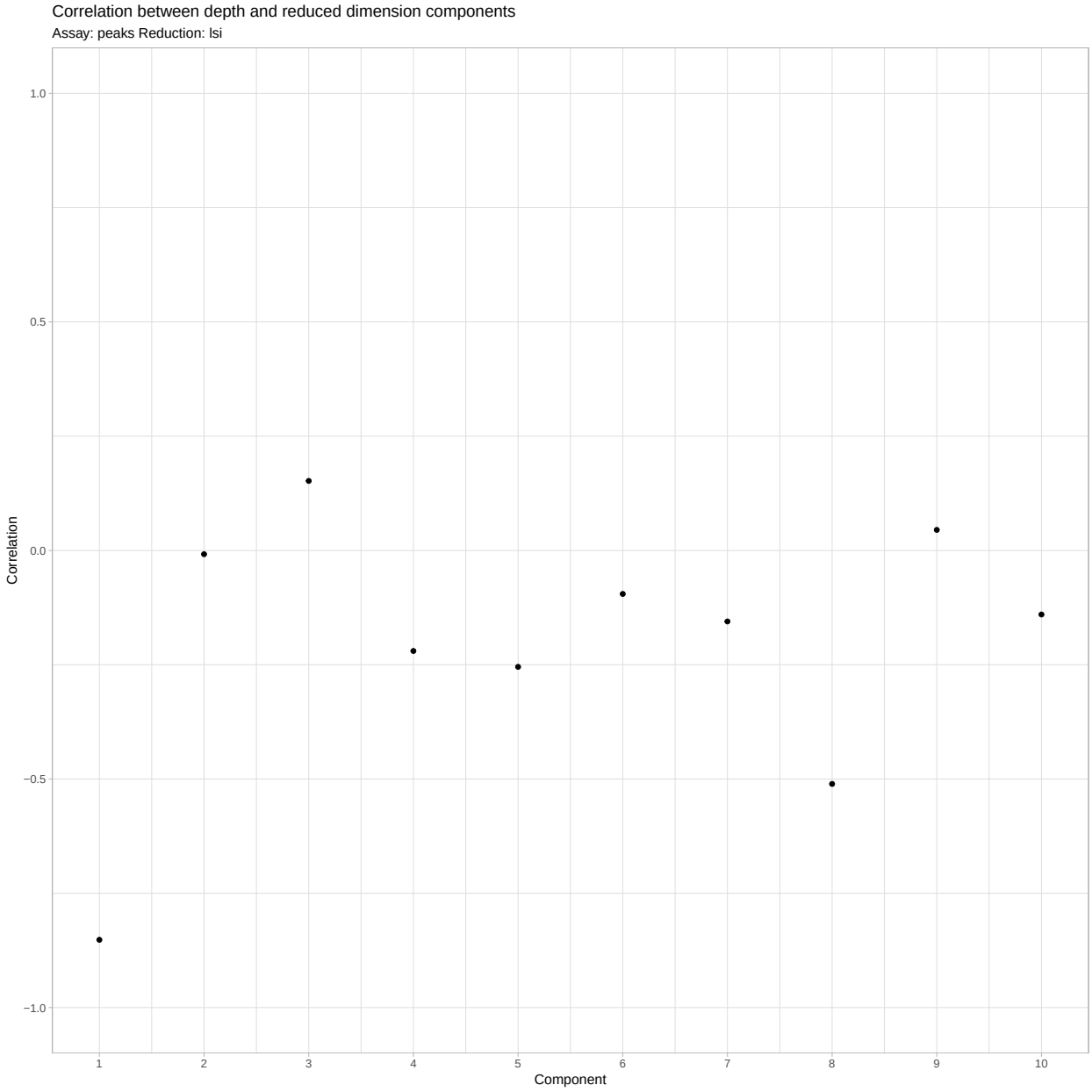
```
## Running SVD
```

```
## Scaling cell embeddings
```

Inspect results

The first LSI component often captures sequencing depth (technical variation) rather than biological variation. If this is the case, the component should be removed from downstream analysis. We can assess the correlation between each LSI component and sequencing depth using the `DepthCor()` function:

```
DepthCor(pbmc)
```

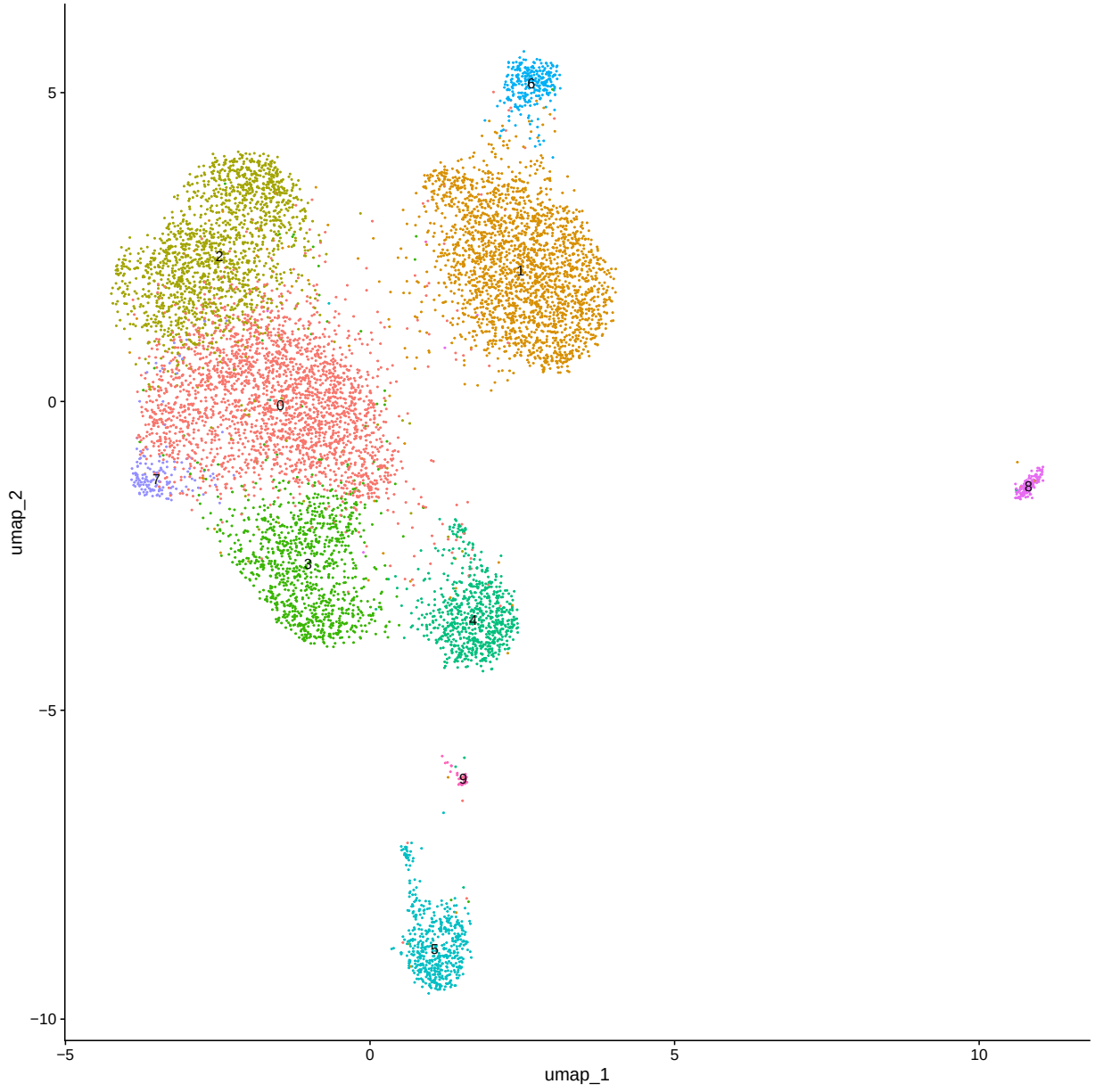


Here we see there is a very strong correlation between the first LSI component and the total number of counts for the cell. We will perform downstream steps without this component as we don't want to group cells together based on their total sequencing depth, but rather by their patterns of accessibility at cell-type-specific peaks.

Non-linear Dimension Reduction, Clustering

Now that the cells are embedded in a low-dimensional space we can use methods commonly applied for the analysis of scRNA-seq data to perform graph-based clustering and non-linear dimension reduction for visualization.

```
pbmc <- RunUMAP(object = pbmc, reduction = 'lsi', dims = 2:30)
pbmc <- FindNeighbors(object = pbmc, reduction = 'lsi', dims = 2:30)
pbmc <- FindClusters(object = pbmc, verbose = FALSE, algorithm = 3)
DimPlot(object = pbmc, label = TRUE) + NoLegend()
```



Gene Activity

The UMAP visualization reveals the presence of multiple cell groups in human blood. However, annotating and interpreting clusters is more challenging in scATAC-seq data as much less is known about the functional roles of non-coding genomic regions than is known about protein coding regions (genes).

We can try to quantify the activity of each gene in the genome by assessing the chromatin accessibility associated with the gene, and create a new gene activity assay derived from the scATAC-seq data. We will use a simple approach of summing the fragments intersecting the gene body and promoter region.

To create a gene activity matrix, we extract gene coordinates and extend them to include the 2 kb upstream region (as promoter accessibility is often correlated with gene expression). We then count the number of fragments for each cell that map to each of these regions, using the using the `FeatureMatrix()` function. These steps are automatically performed by the `GeneActivity()` function:

```
gene.activities <- GeneActivity(pbmc)
```

```
## Extracting gene coordinates
```

```
## Extracting reads overlapping genomic regions
```

Gene Activities

You can read the `gene.activities` object from `/ix1/biost2154-2025f/lab/lab6/gene.activities.rds`.

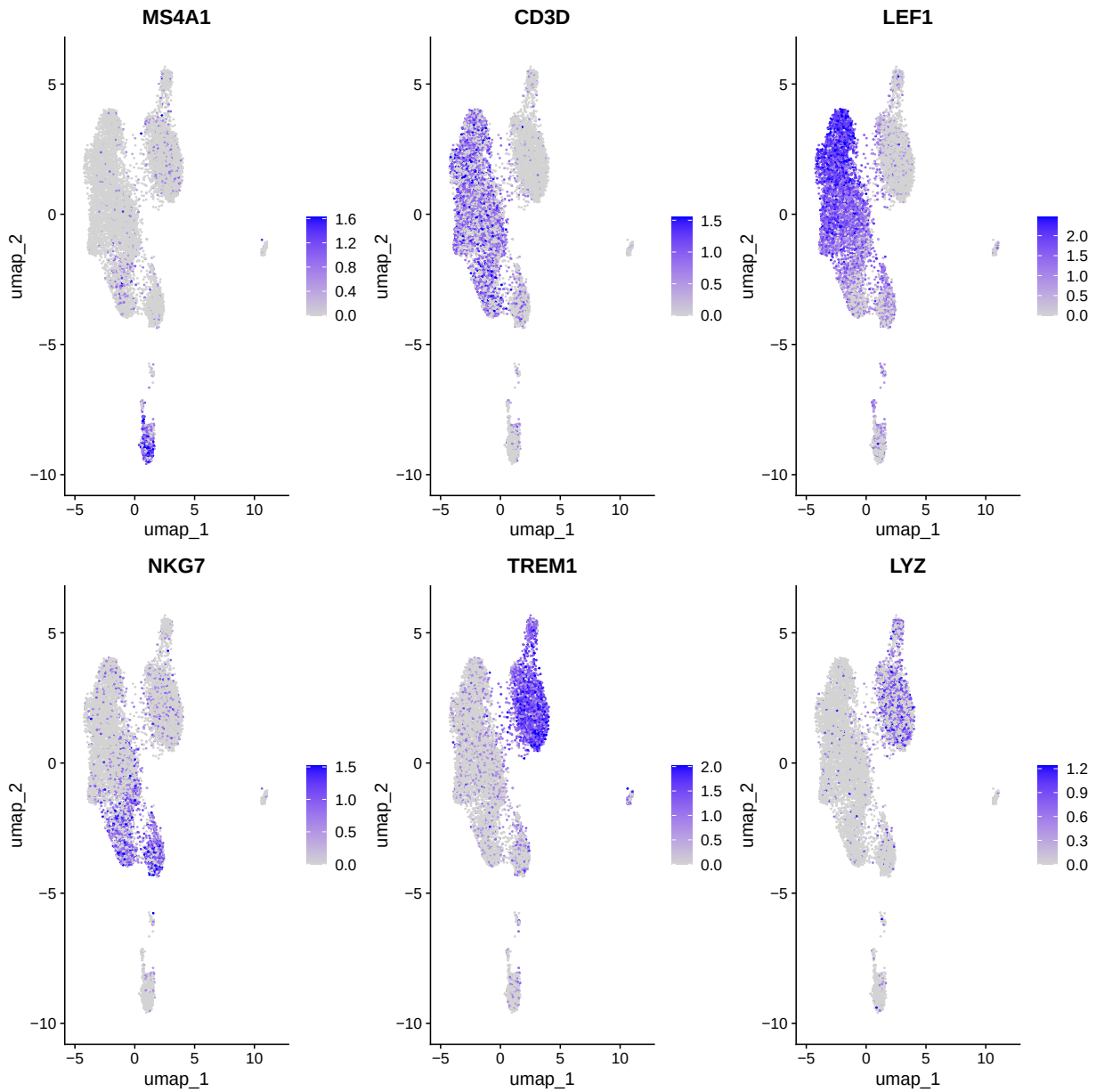
```
# add the gene activity matrix to the Seurat object as a new assay and normalize it
pbmc[['RNA']] <- CreateAssayObject(counts = gene.activities)
pbmc <- NormalizeData(
  object = pbmc,
  assay = 'RNA',
  normalization.method = 'LogNormalize',
  scale.factor = median(pbmc$nCount_RNA)
)
DefaultAssay(pbmc) <- 'RNA'
```

Checkpoint 3

You can read the annotated `pbmc` object from `/ix1/biost2154-2025f/lab/lab6/pbmc_checkpoint3.rds`.

Now we can visualize the activities of canonical marker genes to help interpret our ATAC-seq clusters. Note that the activities will be much noisier than scRNA-seq measurements. This is because they represent measurements from sparse chromatin data, and because they assume a general correspondence between gene body/promoter accessibility and gene expression which may not always be the case. Nonetheless, we can begin to discern populations of monocytes, B, T, and NK cells based on these gene activity profiles.

```
FeaturePlot(
  object = pbmc,
  features = c('MS4A1', 'CD3D', 'LEF1', 'NKG7', 'TREM1', 'LYZ'),
  pt.size = 0.1,
  max.cutoff = 'q95',
  ncol = 3
)
```


MS4A1
CD3D
LEF1
NKG7
TREM1
LYZ
umap_1
umap_2
5
0
−5
−10
−5
0
5
10
1.6
1.2
0.8
0.4
0.0
1.5
1.0
0.5
0.0
2.0
1.5
1.0
0.5
0.0
1.5
1.0
0.5
0.0
2.0
1.5
1.0
0.5
0.0
1.2
0.9
0.6
0.3
0.0

Integrating with scRNA-seq data

To help interpret the scATAC-seq data, we can classify cells based on an scRNA-seq experiment from the same biological system (human PBMC). We utilize methods for cross-modality integration and label transfer. We aim to identify shared correlation patterns in the gene activity matrix and scRNA-seq dataset to identify matched biological states across the two modalities. This procedure returns a classification score for each cell for each scRNA-seq-defined cluster label.

```
pbmc_rna <- readRDS("/ix1/biost2154-2025f/lab/lab6/pbmc_10k_v3.rds")
pbmc_rna <- UpdateSeuratObject(pbmc_rna)
```

```
## Validating object structure
## Updating object slots
## Ensuring keys are in the proper structure
## Updating matrix keys for DimReduc 'pca'
## Updating matrix keys for DimReduc 'tsne'
## Updating matrix keys for DimReduc 'umap'
## Ensuring keys are in the proper structure
## Ensuring feature names don't have underscores or pipes
## Updating slots in RNA
## Updating slots in RNA_nn
## Setting default assay of RNA_nn to RNA
## Updating slots in RNA_snn
## Setting default assay of RNA_snn to RNA
## Updating slots in pca
## Updating slots in tsne
## Setting tsne DimReduc to global
## Updating slots in umap
## Setting umap DimReduc to global
## Setting assay used for NormalizeData.RNA to RNA
## Setting assay used for FindVariableFeatures.RNA to RNA
## Setting assay used for ScaleData.RNA to RNA
## Setting assay used for RunPCA.RNA to RNA
## Setting assay used for RunTSNE.pca to RNA
## Setting assay used for FindNeighbors.RNA.pca to RNA
## No assay information could be found for FindClusters
## Setting assay used for RunUMAP.RNA.pca to RNA
## Validating object structure for Assay 'RNA'
## Validating object structure for Graph 'RNA_nn'
## Validating object structure for Graph 'RNA_snn'
## Validating object structure for DimReduc 'pca'
```

```
## Validating object structure for DimReduc 'tsne'

## Validating object structure for DimReduc 'umap'

## Object representation is consistent with the most current Seurat version
```

```
transfer.anchors <- FindTransferAnchors(
  reference = pbmc_rna,
  query = pbmc,
  reduction = 'cca'
)
```

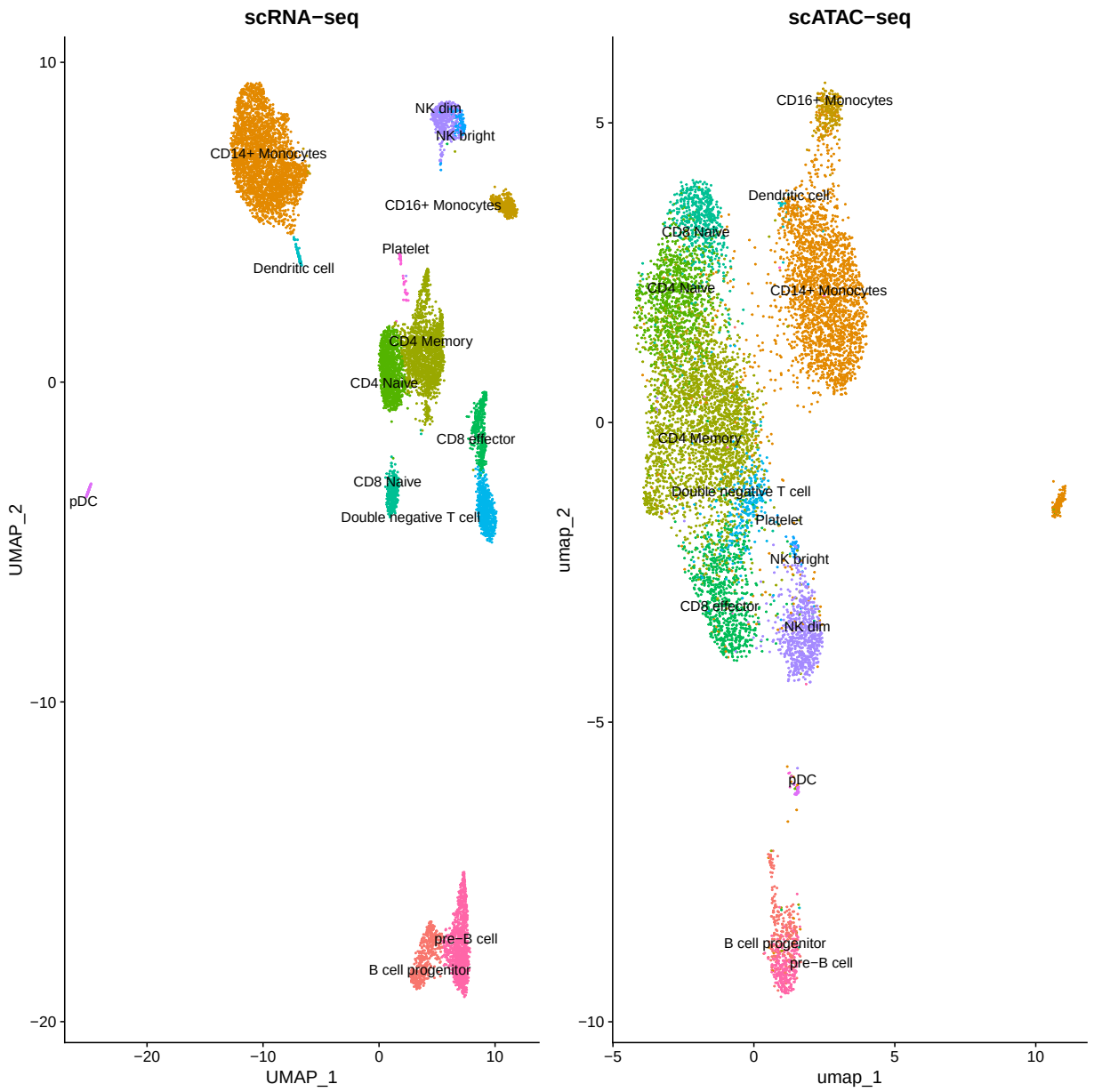
```
## Running CCA

## Merging objects

## Finding neighborhoods

## Finding anchors

##  Found 18932 anchors
```

```
predicted.labels <- TransferData(
  anchorset = transfer.anchors,
  refdata = pbmc_rna$celltype,
  weight.reduction = pbmc[['lsi']],
  dims = 2:30
)
```

```
## Finding integration vectors

## Finding integration vector weights

## Predicting cell labels
```

```
pbmc <- AddMetaData(object = pbmc, metadata = predicted.labels)
```

```
plot1 <- DimPlot(
  object = pbmc_rna,
  group.by = 'celltype',
  label = TRUE,
  repel = TRUE) + NoLegend() + ggtitle('scRNA-seq')

plot2 <- DimPlot(
  object = pbmc,
  group.by = 'predicted.id',
  label = TRUE,
  repel = TRUE) + NoLegend() + ggtitle('scATAC-seq')

plot1 + plot2
```



Checkpoint 4

You can read the annotated `pbmc` object from `/ix1/biost2154-2025f/lab/lab6/pbmc_checkpoint4.rds`.

You can see that the scRNA-based classifications are consistent with the UMAP visualization that was computed using the scATAC-seq data. Notice, however, that a small population of cells are predicted to be platelets in the scATAC-seq dataset. This is unexpected as platelets are not nucleated and should not be detected by scATAC-seq. It is possible that the cells predicted to be platelets could instead be the platelet-precursors megakaryocytes, which largely reside in the bone marrow but are rarely found in the peripheral blood of healthy patients, such as the individual these PBMCs were drawn from. Given the already extreme rarity of megakaryocytes within normal bone marrow ($< 0.1\%$), this scenario seems unlikely.

For downstream analysis we will thus remove the extremely rare cell states that were predicted, retaining only cell annotations with >20 cells total.

```r
predicted_id_counts <- table(pbmc$predicted.id)

# Identify the predicted.id values that have more than 20 cells
major_predicted_ids <- names(predicted_id_counts[predicted_id_counts > 20])
pbmc <- pbmc[, pbmc$predicted.id %in% major_predicted_ids]
# change cell identities to the per-cell predicted labels
Idents(pbmc) <- pbmc$predicted.id
```

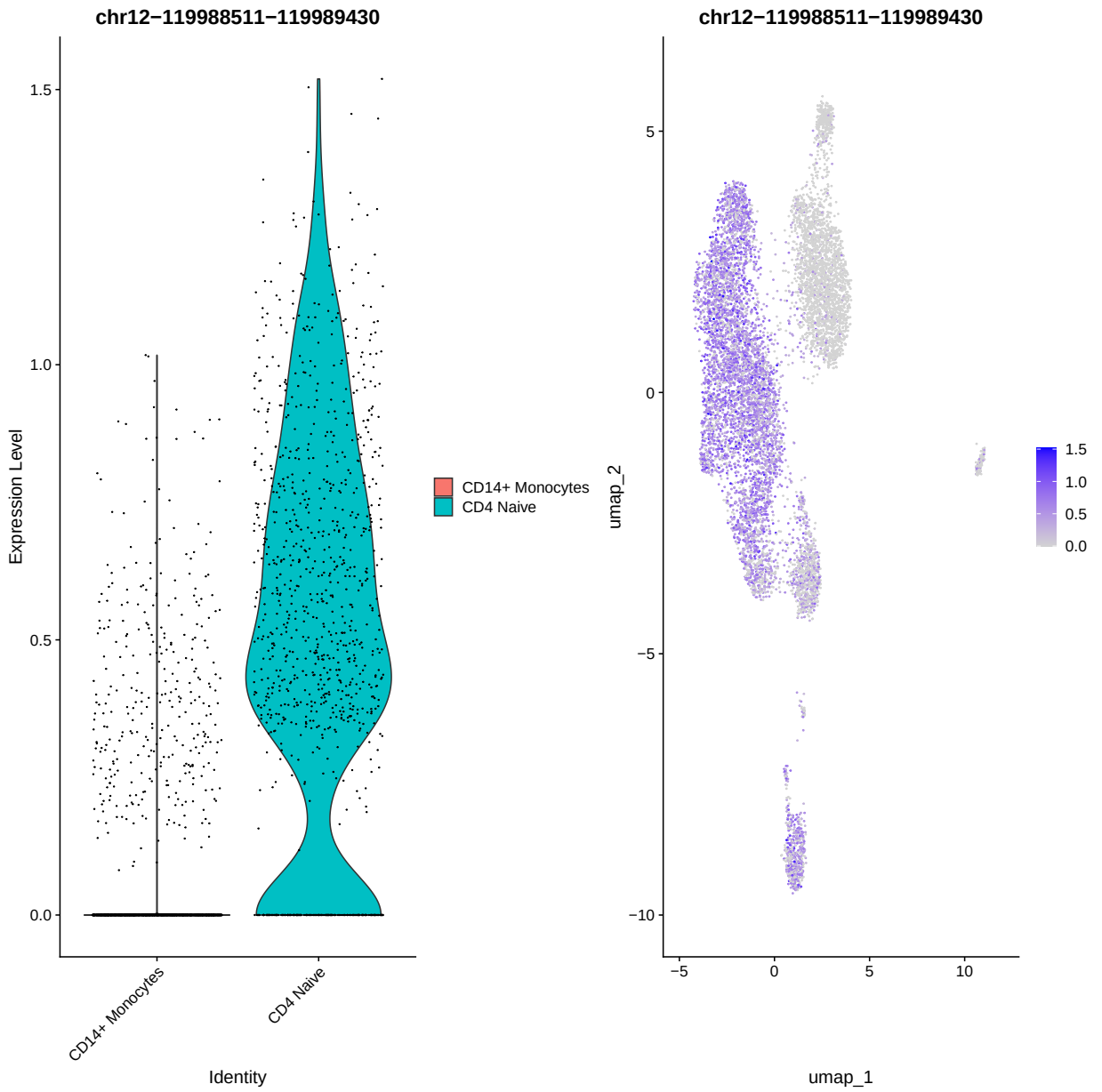
Find differentially accessible peaks between cell types

To find differentially accessible regions between clusters of cells, we can perform a differential accessibility (DA) test. A simple approach is to perform a Wilcoxon rank sum test.

```
# change back to working with peaks instead of gene activities
DefaultAssay(pbmc) <- 'peaks'

# wilcox is the default option for test.use
da_peaks <- FindMarkers(
  object = pbmc,
  ident.1 = "CD4 Naive",
  ident.2 = "CD14+ Monocytes",
  test.use = 'wilcox',
  min.pct = 0.1
)

head(da_peaks)
```

```
##                                 p_val avg_log2FC pct.1 pct.2     p_val_adj
## chr12-119988511-119989430  0.000000e+00   3.329073 0.798 0.127  0.000000e+00
## chr17-82126458-82127377    0.000000e+00   4.163995 0.711 0.066  0.000000e+00
## chr7-142808530-142809435   0.000000e+00   3.156111 0.753 0.122  0.000000e+00
## chr7-138752283-138753197   0.000000e+00   3.837290 0.687 0.078  0.000000e+00
## chr4-152100248-152101142   0.000000e+00   4.139164 0.634 0.060  0.000000e+00
## chr6-13302533-13303459    7.321348e-311  -6.523875 0.014 0.779 1.365358e-306
```

```
plot1 <- VlnPlot(
  object = pbmc,
  features = rownames(da_peaks)[1],
  pt.size = 0.1,
  idents = c("CD4 Naive","CD14+ Monocytes")
)
plot2 <- FeaturePlot(
  object = pbmc,
  features = rownames(da_peaks)[1],
  pt.size = 0.1
)

plot1 | plot2
```



Peak coordinates can be difficult to interpret alone. We can find the closest gene to each of these peaks using the `ClosestFeature()` function.

```
open_cd4naive <- rownames(da_peaks[da_peaks$avg_log2FC > 3, ])
open_cd14mono <- rownames(da_peaks[da_peaks$avg_log2FC < -3, ])

closest_genes_cd4naive <- ClosestFeature(pbmc, regions = open_cd4naive)
closest_genes_cd14mono <- ClosestFeature(pbmc, regions = open_cd14mono)
head(closest_genes_cd4naive)
```

```
##                          tx_id gene_name         gene_id   gene_biotype type
## ENSE00002206071 ENST00000397558    BICDL1 ENSG00000135127 protein_coding exon
## ENST00000665763 ENST00000665763    CCDC57 ENSG00000176155 protein_coding  gap
## ENST00000632998 ENST00000632998     PRSS2 ENSG00000275896 protein_coding  utr
## ENST00000645515 ENST00000645515  ATP6V0A4 ENSG00000105929 protein_coding  cds
```

```
## ENST00000603548 ENST00000603548     FBXW7 ENSG00000109670 protein_coding  utr
## ENST00000545320 ENST00000545320       CD6 ENSG00000013725 protein_coding  gap
##                            closest_region              query_region distance
## ENSE00002206071 chr12-119989869-119990297 chr12-119988511-119989430      438
## ENST00000665763   chr17-82101867-82127691   chr17-82126458-82127377        0
## ENST00000632998  chr7-142774509-142774564  chr7-142808530-142809435    33965
## ENST00000645515  chr7-138752625-138752837  chr7-138752283-138753197        0
## ENST00000603548  chr4-152320544-152322880  chr4-152100248-152101142   219401
## ENST00000545320   chr11-60971915-60987907   chr11-60985909-60986801        0
```

```
head(closest_genes_cd14mono)
```

```
##                           tx_id gene_name         gene_id   gene_biotype type
## ENST00000606214 ENST00000606214    TBC1D7 ENSG00000145979 protein_coding  gap
## ENST00000448962 ENST00000448962      RPS9 ENSG00000170889 protein_coding  gap
## ENST00000592988 ENST00000592988     AFMID ENSG00000183077 protein_coding  gap
## ENST00000336600 ENST00000336600  C6orf223 ENSG00000181577 protein_coding  utr
## ENSE00002618192 ENST00000569518    VCPKMT ENSG00000100483 protein_coding exon
## ENSE00001136869 ENST00000264710     RAB10 ENSG00000084733 protein_coding exon
##                          closest_region            query_region distance
## ENST00000606214  chr6-13267836-13305061  chr6-13302533-13303459        0
## ENST00000448962 chr19-54201610-54231740 chr19-54207815-54208728        0
## ENST00000592988 chr17-78191061-78202498 chr17-78198651-78199583        0
## ENST00000336600  chr6-44003127-44007612  chr6-44058439-44059230    50826
## ENSE00002618192 chr14-50108632-50109713 chr14-50038381-50039286    69345
## ENSE00001136869  chr2-26034084-26034735  chr2-26008700-26009572    24511
```

We could follow up with this result by doing gene ontology enrichment analysis on the gene sets.

Plotting genomic regions

We can plot the frequency of Tn5 integration across regions of the genome for cells grouped by cluster, cell type, or any other metadata stored in the object for any genomic region using the `CoveragePlot()` function. These represent pseudo-bulk accessibility tracks, where signal from all cells within a group have been averaged together to visualize the DNA accessibility in a region. Alongside these accessibility tracks we can visualize other important information including gene annotation, peak coordinates, and genomic links (if they're stored in the object).

For plotting purposes, it's nice to have related cell types grouped together. We can automatically sort the plotting order according to similarities across the annotated cell types by running the `SortIdents()` function:

```
pbmc <- SortIdents(pbmc)
```

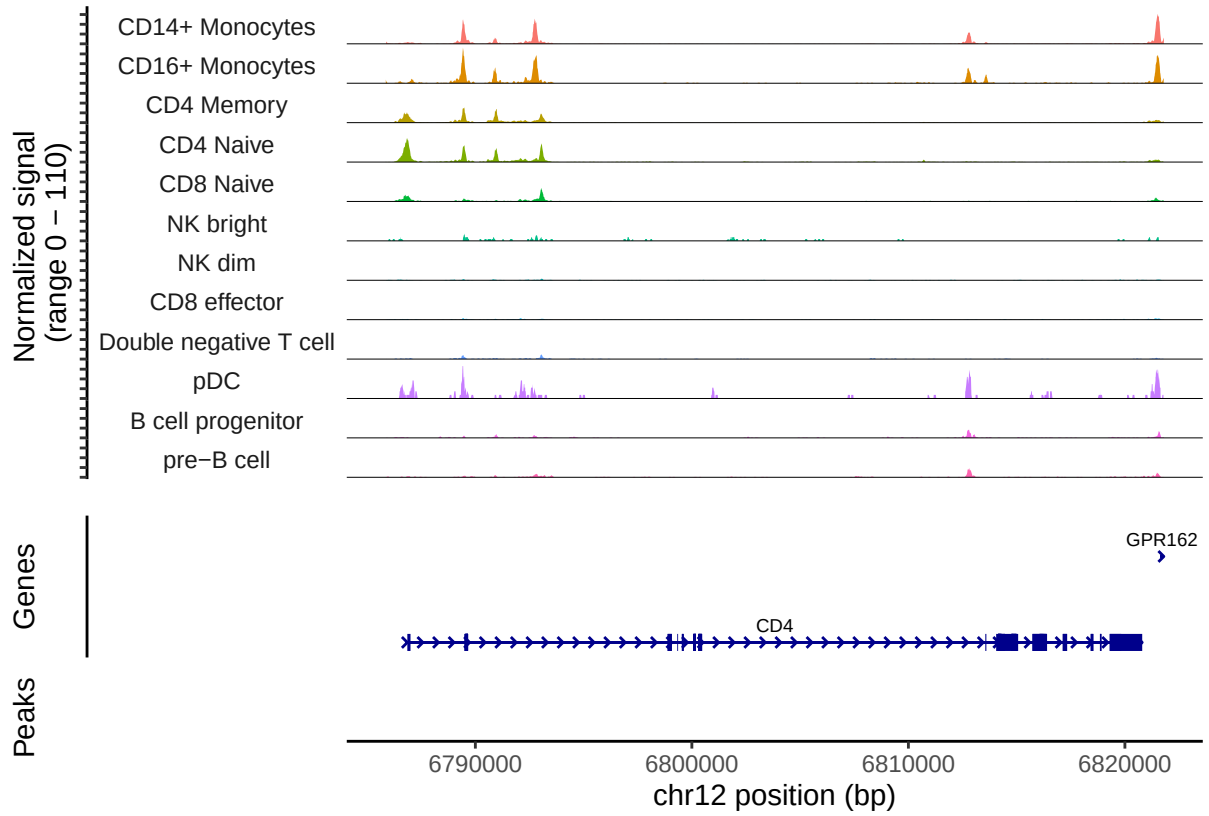
```
## Creating pseudobulk profiles for 12 variables across 18649 features
```

```
## Computing euclidean distance between pseudobulk profiles
```

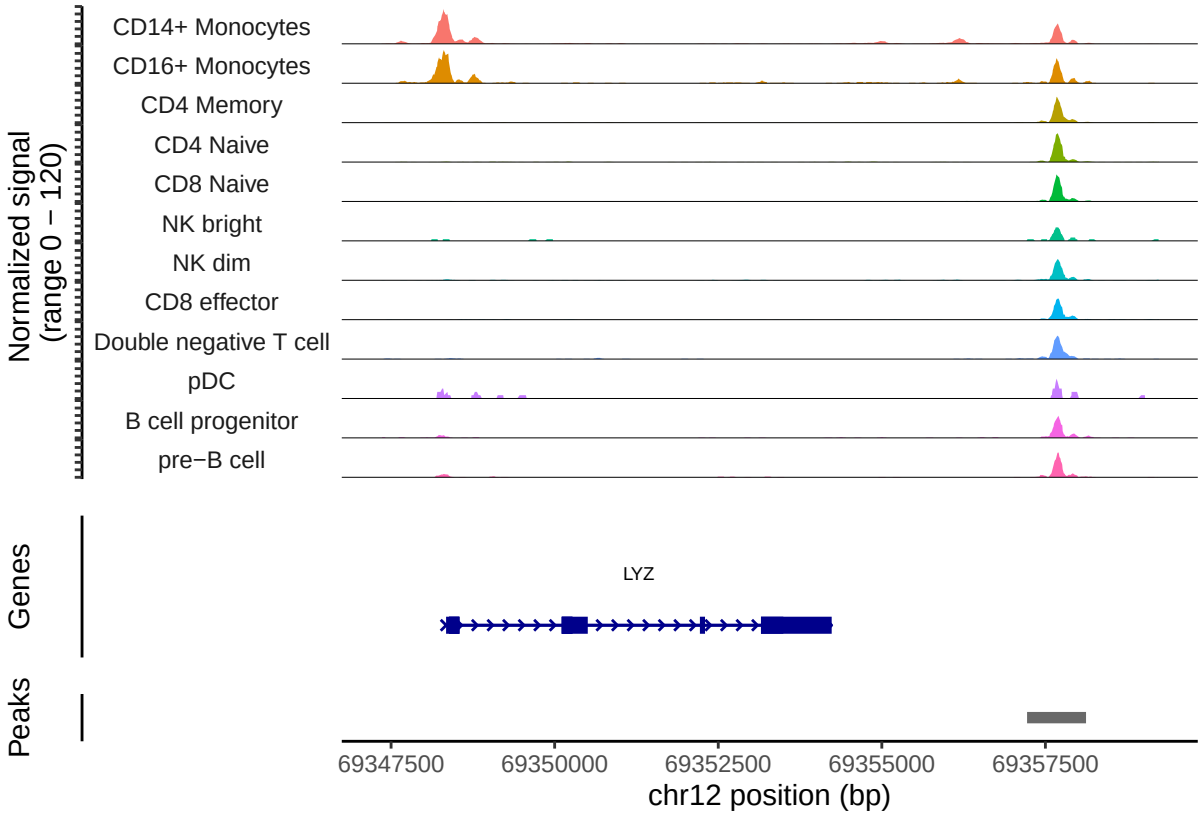
```
## Clustering distance matrix
```

We can then visualize the DA peaks open in CD4 naive cells and CD14 monocytes, near some key marker genes for these cell types, CD4 and LYZ respectively. Here we'll highlight the DA peaks regions in grey.

```
# find DA peaks overlapping gene of interest
regions_highlight <- subsetByOverlaps(StringToGRanges(open_cd4naive), LookupGeneCoords(pbmc, "CD4"))

CoveragePlot(
  object = pbmc,
  region = "CD4",
  region.highlight = regions_highlight,
  extend.upstream = 1000,
  extend.downstream = 1000
)
```




```
regions_highlight <- subsetByOverlaps(StringToGRanges(open_cd14mono), LookupGeneCoords(pbmc, "LYZ"))

CoveragePlot(
  object = pbmc,
  region = "LYZ",
  region.highlight = regions_highlight,
  extend.upstream = 1000,
  extend.downstream = 5000
)
```



Next Lab

In the next lab I will talk about spatial transcriptomics in Seurat. Please install these packges and data:

```
devtools::install_github('satijalab/seurat-data')
SeuratData::InstallData("stxBrain")
```

Questions

If you have any questions or run into issues related to the class, please contact us or go to office hours:

Instructor: jbwang@pitt.edu, Monday 4-5pm in A732

TA: wel235@pitt.edu, Wednesday 1-2pm in A724A