Importing the data into R and Signac+Seurat, initial QC and clustering

Ivan Berest and Christian Arnold

Contents

Goals

This vignette will teach you how to read the processed 10x data into R and `Seurat`, one of the most commonly used and most powerful packages for single-cell data analysis in R. You will learn learn about data QC, filtering, normalization, dimensionality reduction, creating embeddings such as UMAP, clustering, and more generally how to work with `Seurat` objects in R and what pitfalls scATAC-seq data have and how to circumvent them.

References

This vignette is a modified version of this vignette, with modified descriptions from the ArchR website as well as the clustree vignette.

ATAC-Seq recap

The most fundamental component of any ATAC-seq experiment is a *fragment*. In ATAC-seq, a fragment refers to a sequenceable DNA molecule created by two transposition events. Each end of that fragment is sequenced using paired-end sequencing. The inferred single-base position of the start and end of the fragment is adjusted based on the *insertion* offset of Tn5. As reported previously, Tn5 transposase binds to DNA as a homodimer with 9-bp of DNA between the two Tn5 molecules. Because of this, each Tn5 homodimer binding event creates two insertions, separated by 9 bp. Thus, the actual central point of the *accessible* site is in the very center of the Tn5 dimer, not the location of each Tn5 insertion. To account for this, one usually applies an offset to the individual Tn5 insertions, adjusting plus-stranded insertion events by +4 bp and minus-stranded insertion events by -5 bp. This is consistent with the convention put forth during the original description of ATAC-seq. This correction is already performed by 10x in the `fragments` file, see here for details.

Thus, after importing the fragment file into R, *fragments* refers to a table or genomic ranges object containing the chromosome, offset-adjusted single-base chromosome start position, offset-adjusted single-base chromosome end position, and unique cellular barcode ID corresponding to each sequenced fragment. Similarly, *insertions* refer to the offset-adjusted single-base position at the very center of an accessible site.

Load required R libraries

Before we start with any analysis, it is good practice to load all required libraries. This will also immediately identify libraries that may be missing. Note that for this course, we pre-installed all libraries for you. When you run your own analysis, you have to check which libraries are already available, and which are not. We use `suppressPackageStartupMessages` here to suppress the output messages of the various packages for reasons of brevity.

When using functions that sample pseudorandom numbers, each time you execute them you will obtain a different result. For the purpose of this vignette, this is not what we want. We therefore set a particular seed value(here: 1990) so that the result will always be the same. For more information, check out this webpage that explains this general concept in more detail.

```
suppressPackageStartupMessages({
library(Signac)
library(Seurat)
library(GenomeInfoDb)
library(tidyverse)
library(patchwork)
library(AnnotationHub)
library(clustree)
library(hdf5r)
library(biovizBase)
library(scDblFinder)
})
set.seed(1990)
```

Load 10x data into R and setup output directories

We are now ready to load our 10x data (the `Cell Ranger ATAC` output) into R! All files we need for this are located in the `outs` folder, and we just have to provide the locale path to it when importing it into R.

When pre-processing chromatin data, `Signac` uses information from two related input files, both of which have been created with `CellRanger ATAC (cellranger-atac)`:

- **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 (i.e. a cell) that map within each peak. You can find more detail on the 10X Website.
- **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. More information about the fragment file can be found on the 10x Genomics website or on the sinto website.

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

```
# Specify the path to the "outs" folder from cellranger-atac here
# Make sure to have a trailing slash here
outFolder="/mnt/data/cellranger/outs/"
outFolder="/g/zaugg/zaugg_shared/Courses_and_Teaching/ATAC-Seq_Course/2024/testdata/test_lifminus/outs/
peakCounts <- Read10X_h5(filename = paste0(outFolder, "filtered_peak_bc_matrix.h5"))

# The barcode metadata we also have to import
metadata <- read.csv(
  file = paste0(outFolder, "singlecell.csv"),
  header = TRUE, row.names = 1)

# Alternatively, we can use the tidyverse, but we need to create rownames
# after importing the table. Not executed here
# metadata <- read_csv(file = paste0(outFolder, "singlecell.csv"))
# %>% column_to_rownames("barcode")

# We can now create our ChromatinAssay object from a count matrix
# We have to use mm10 here because the 10x reference is
# also "only" for mm10 and not yet for mm39
chrom_assay <- CreateChromatinAssay(
  counts = peakCounts,
  sep = c(":", "-"),
  genome = 'mm10',
  fragments = paste0(outFolder,'fragments.tsv.gz'),
  min.cells = 10,
  min.features = 100
)
```

```
## Computing hash
```

```
# We can now create a Seurat object from the ChromatinAssay object
seu <- CreateSeuratObject(
  counts = chrom_assay,
  project = "mouse_ES_LIF",
  assay = "ATAC",
  meta.data = metadata
)
```

```
# Set default array so we dont have to specify array = "ATAC" in subsequent steps
DefaultAssay(seu) = "ATAC"

# Delete objects and variables we do not need anymore to save space
rm(chrom_assay,peakCounts,metadata)
```

As you can see, the ATAC-seq data is stored using a custom assay named `ATAC`, an assay of class `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.

Explore Seurat object

To familiarize ourselves with our Seurat object, we here execute a few commands that explore the object and what is stored in there. For more information and a general overview of what is possible, see, for example, here

For example, we can call `granges` on a Seurat object with a `ChromatinAssay` set as the active assay (or on a `ChromatinAssay`) to see the genomic ranges associated with each feature in the object. See the object interaction vignette for more information about the `ChromatinAssay` class.

```
# Print general object information by typing the object name
seu
```

```
## An object of class Seurat
## 275782 features across 19741 samples within 1 assay
## Active assay: ATAC (275782 features, 0 variable features)
##  2 layers present: counts, data
```

```
# See the counts, and how the are stored
GetAssayData(seu, slot = "counts")[1:20,1:20]
```

```
## Warning: The ‘slot‘ argument of ‘GetAssayData()‘ is deprecated as of SeuratObject 5.0.0.
## i Please use the ‘layer‘ argument instead.
## This warning is displayed once every 8 hours.
## Call ‘lifecycle::last_lifecycle_warnings()‘ to see where this warning was
## generated.
```

```
## 20 x 20 sparse Matrix of class "dgCMatrix"
```

```
##   [[ suppressing 20 column names ’AAACGAAAGAACGACC-1’, ’AAACGAAAGACACAAT-1’, ’AAACGAAAGACCCATT-1’ ..
```

```
##
## chr1-3035418-3036317 . . . . . .
## chr1-3046141-3047094 . . . . . .
## chr1-3062361-3063374 . . . . . .
## chr1-3191291-3192291 . . . . . .
## chr1-3263525-3264378 . . . . . .
## chr1-3267966-3268954 . . . . . .
## chr1-3321191-3322300 . . . . . .
## chr1-3325013-3325907 . . . . . .
```

```
## chr1-3327962-3328870 . . . . . .
## chr1-3340421-3341327 . . . . . 2 . . . .
## chr1-3343380-3343935 . 2 . . . . . .
## chr1-3344039-3344659 . . 2 . . . . . .
## chr1-3353146-3354027 . . . . . 2 .
## chr1-3360669-3361599 . . . . . .
## chr1-3377173-3378064 . . . . . .
## chr1-3395275-3396191 . . . . . .
## chr1-3398485-3399513 . . . . . .
## chr1-3399747-3400704 . . . . . .
## chr1-3421368-3422283 . . . . . .
## chr1-3423083-3423849 . . . . . .
```

```
# Print the Assay information
seu[["ATAC"]]
```

```
## ChromatinAssay data with 275782 features for 19741 cells
## Variable features: 0
## Genome: mm10
## Annotation present: FALSE
## Motifs present: FALSE
## Fragment files: 1
```

```
# Check the peaks, we make use of GenomicRanges here
granges(seu)
```

```
## GRanges object with 275782 ranges and 0 metadata columns:
##               seqnames          ranges strand
##                  <Rle>       <IRanges>  <Rle>
##        [1]       chr1 3035418-3036317      *
##        [2]       chr1 3046141-3047094      *
##        [3]       chr1 3062361-3063374      *
##        [4]       chr1 3191291-3192291      *
##        [5]       chr1 3263525-3264378      *
##        ...        ...             ...    ...
##   [275778] GL456216.1     43787-44669      *
##   [275779] GL456216.1     47168-48061      *
##   [275780] GL456216.1     48813-49713      *
##   [275781] GL456216.1     50753-51651      *
##   [275782] JH584295.1       1099-1976      *
##   -------
##   seqinfo: 26 sequences from an unspecified genome; no seqlengths
```

Add genome annotation to Seurat object

Next, we add annotation to our `Seurat` object. This will allow downstream functions to pull the gene annotation information directly from the object.

First, let's retrieve all available genome annotations:

```
# downloads database to the cache
annHub <- AnnotationHub(ask = FALSE)
```

```
## snapshotDate(): 2022-10-31
```

```
## Query for all available EnsDb mouse databases
annVersions = query(annHub, pattern = c("Mmusculus","EnsDb"))
```

Now, we can choose one of them and select them as annotation for our `Seurat` object.

```
# Now we chose an appropriate annotation version that matches our genome:
# Here, mm10 corresponds to GRCm38
which(annVersions$genome == "GRCm38")
```

```
##  [1]  1  2  3  4  5  6  7  8  9 10 11 12 13 14 15 16
```

```
# Pick the newest, here with index 16
annVersionName = names(annVersions)[16]

# Retrieving ENSEMBL 102 release with ID AH89211  for mm10
adb.use <-  annVersions[[annVersionName]]
```

```
## loading from cache
```

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

```
adb.use
```

```
## EnsDb for Ensembl:
## |Backend: SQLite
## |Db type: EnsDb
## |Type of Gene ID: Ensembl Gene ID
## |Supporting package: ensembldb
## |Db created by: ensembldb package from Bioconductor
## |script_version: 0.3.6
## |Creation time: Sun Dec 20 12:15:16 2020
## |ensembl_version: 102
## |ensembl_host: localhost
## |Organism: Mus musculus
## |taxonomy_id: 10090
## |genome_build: GRCm38
## |DBSCHEMAVERSION: 2.1
## | No. of genes: 56305.
## | No. of transcripts: 144778.
## |Protein data available.
```

```
# use only standard chromosomes for annotation
annotations <- suppressMessages(suppressWarnings(
    GetGRangesFromEnsDb(adb.use, standard.chromosomes = TRUE)))

# change to UCSC style since the data was mapped to mm10
seqlevelsStyle(annotations) <- 'UCSC'
Annotation(seu[["ATAC"]]) <- annotations
```

Initial data filtering: Filtering peaks

First, we will filter out some peaks that are located outside of the regular chromosomes. usually, this only affects a very small number of peaks, but it is nevertheless a good idea to do so! This also teaches you how to filter a `Seurat` object!

```
# Number of rows (here, this corresponds to peaks) before any filtering
nrow(seu)
```

```
## [1] 275782
```

```
genome(seu@assays$ATAC@ranges) <- "mm10"

# Get a list of all main chromosomes from the annotation
main.chroms <- levels(seqnames(annotations))

# Determine for each peak whether it is located on a main chromosome (TRUE) or not (FALSE)
keep.peaks <- which(as.character(seqnames(granges(seu[["ATAC"]]))) %in% main.chroms)

# Filter the object and discard peaks
seu[["ATAC"]] <- subset(seu[["ATAC"]], features = rownames(seu[["ATAC"]])[keep.peaks])
```

```
## Warning: Different cells and/or features from existing assay ATAC
```

```
seu
```

```
## An object of class Seurat
## 275719 features across 19741 samples within 1 assay
## Active assay: ATAC (275719 features, 0 variable features)
##  2 layers present: counts, data
```

How many peaks we have now, can you compare this to the number of peaks before the filtering?

Detection and handling of doublets/multiplets

Doublet detection is a crucial step in the analysis of single-cell data. Doublets are technical artifacts that occur when two or more cells are mistakenly encapsulated into one reaction volume, leading to the generation of artificial cells that can confound downstream analyses. Identifying and removing doublets is important to ensure accurate interpretation of scRNA-seq data and to avoid spurious biological conclusions.

`CellRanger ATAC` currently does not have a method for computationally classifying whether a barcode contains more than one cell in single cell gene expression data from a single species. Thus, we need to use other tools for doublet detection. Multiple tools exist, but most have originally been developed for scRNA-seq data. These methods aim to distinguish real cells from doublets by analyzing gene expression patterns. One such method is DoubletFinder, which predicts doublets based on the proximity of real cells in gene expression space to artificial doublets created by averaging the transcriptional profiles of randomly chosen cell pairs. `DoubletFinder` uses only gene expression data and has been shown to be effective in identifying doublets.

Another method is `scDblFinder`, which has originally also been developed for doublet detection in scRNA-seq data. It uses a cluster-based approach and selects the top most expressed features to reduce the dataset before detecting doublets. However, methods that were built / optimized for scRNA-seq may not perform

well for scATAC-seq because scRNA-seq are typically limited to a relatively small set of features that are most informative (typically, a few thousands): genes with a higher expression. In contrast, scATACseq data is considerably more sparse, with most reads being spread across hundreds of thousands of regions. Selecting a subset of genes therefore is not very suitable.

`scDblFinder` specifically developed and implemented doublet detection methods for scATAC-seq, and we here present and use one of them. For the other available methods for scATAC-data, see the Vignette for more details. Default options will not work well for scATAC-seq, but using the options `aggregateFeatures = TRUE` as well as a (relatively small) value for `nfeatures` supposedly works well for many datasets:

```
ATAC.sce <- as.SingleCellExperiment(seu, assay = "ATAC")
ATAC.sce <- scDblFinder(ATAC.sce, artificialDoublets = 1, aggregateFeatures = TRUE,
                        nfeatures = 25, processing = "normFeatures")
```

```
## Aggregating features...

## Warning: Quick-TRANSfer stage steps exceeded maximum (= 13785950)

## Warning: Quick-TRANSfer stage steps exceeded maximum (= 13785950)

## Creating ~19741 artificial doublets...

## Dimensional reduction

## Evaluating kNN...

## Training model...

## iter=0, 1553 cells excluded from training.

## iter=1, 2569 cells excluded from training.

## iter=2, 2921 cells excluded from training.

## Threshold found:0.533

## 2005 (10.2%) doublets called
```

```
seu$ATAC_doublets = ATAC.sce$scDblFinder.class
rm(ATAC.sce)
```

How many doublets were found? Is this percentage lower or higher what you initially expected? Discuss with your group!

Quality checks (QC)

We can now compute some QC metrics for the scATAC-seq experiment. Strict QC of scATAC-seq data is essential to remove the contribution of low-quality cells. We currently suggest the following metrics below to assess data quality. As with scRNA-seq, the expected range of values for these parameters will vary depending on your biological system, cell viability, and other factors.

- **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 (see https://www.encodeproject.org/data-standards/terms/). 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. ENCODE blacklist regions for the most commonly used organisms including mouse are included in the `Signac` package.

Note that the last three metrics can be obtained from the output of `Cell Ranger ATAC` (which is stored in the object metadata), but can also be calculated for non-10x datasets using `Signac`.

The number of unique nuclear fragments (i.e. not mapping to mitochondrial DNA). The signal-to-background ratio. Low signal-to-background ratio is often attributed to dead or dying cells which have de-chromatinzed DNA which allows for random transposition genome-wide. The fragment size distribution. Due to nucleosomal periodicity, we expect to see depletion of fragments that are the length of DNA wrapped around a nucleosome (~147 bp). The first metric, unique nuclear fragments, is straightforward - cells with very few usable fragments will not provide enough data to make useful interpretations and should therefore be excluded.

The second metric, signal-to-background ratio, is calculated as the TSS enrichment score. Traditional bulk ATAC-seq analysis has used this TSS enrichment score as part of a standard workflow for determination of signal-to-background (for example, the ENCODE project). We and others have found the TSS enrichment to be representative across the majority of cell types tested in both bulk ATAC-seq and scATAC-seq. The idea behind the TSS enrichment score metric is that ATAC-seq data is universally enriched at gene TSS regions compared to other genomic regions, due to large protein complexes that bind to promoters. By looking at per-basepair accessibility centered at these TSS regions, we see a local enrichment relative to flanking regions (1900-2000 bp distal in both directions). The ratio between the peak of this enrichment (centered at the TSS) relative to these flanking regions represents the TSS enrichment score.

Traditionally, the per-base-pair accessibility is computed for each bulk ATAC-seq sample and then this profile is used to determine the TSS enrichment score. Performing this operation on a per-cell basis in scATAC-seq is relatively slow and computationally expensive. To accurately approximate the TSS enrichment score

per single cell, we count the average accessibility within a 50-bp region centered at each single-base TSS position and divide this by the average accessibility of the TSS flanking positions (+/- 1900 – 2000 bp). This approximation was highly correlated (R > 0.99) with the original method and values were extremely close in magnitude.

The third metric, fragment size distribution, is generally less important but always good to manually inspect. Because of the patterned way that DNA wraps around nucleosomes, we expect to see a nucleosomal periodicity in the distribution of fragment sizes in our data. These hills and valleys appear because fragments must span 0, 1, 2, etc. nucleosomes (Tn5 cannot cut DNA that is tightly wrapped around a nucleosome.

Compute QC metrics

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

# TSS enrichment (takes a few minutes)
seu <- TSSEnrichment(object = seu, fast = FALSE)
```

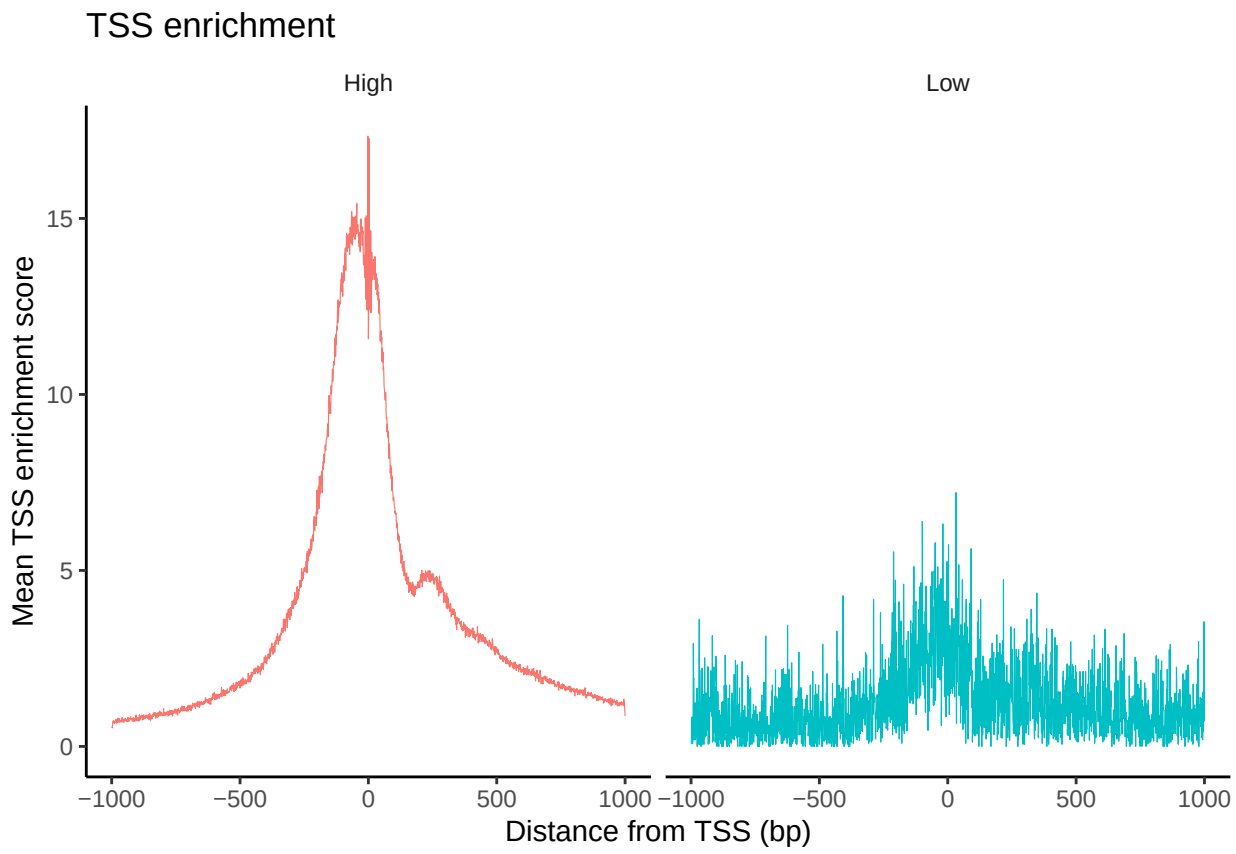
```
## Extracting TSS positions

## Finding + strand cut sites

## Finding - strand cut sites

## Computing mean insertion frequency in flanking regions

## Normalizing TSS score
```

```
# add blacklist ratio and fraction of reads in peaks
seu$pct_reads_in_peaks <- seu$peak_region_fragments / seu$passed_filters * 100

# Set the fraction of reads per cell located in blacklist regions
seu$blacklist_fraction <- FractionCountsInRegion(object = seu, regions = blacklist_mm10)
```

Visualize TSS enrichment

We can inspect the TSS enrichment scores by grouping the cells based on the score and plotting the accessibility signal over all TSS sites, here stratified by whether or not the TSS enrichment is above 2 or not.

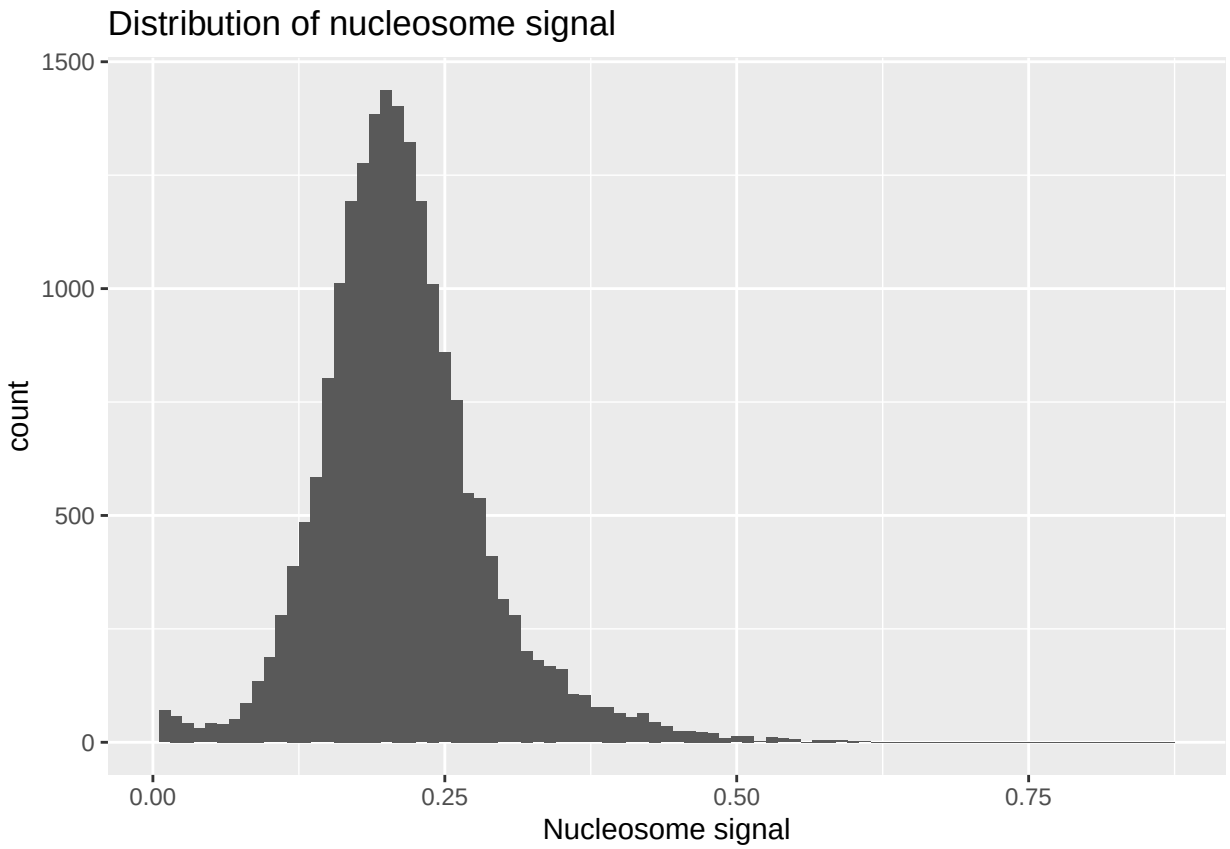
```
seu$high.tss <- ifelse(seu$TSS.enrichment > 2, 'High', 'Low')
TSSPlot(seu, group.by = 'high.tss') + NoLegend()
```



Visualize nucleosome signal

We can also look at the fragment length periodicity for all the cells, and group by cells with high or low nucleosomal signal strength. You can see that cells that are outliers for the mononucleosomal / nucleosome-free ratio (based on the plots above) have different nucleosomal banding patterns. The remaining cells exhibit a pattern that is typical for a successful ATAC-seq experiment.

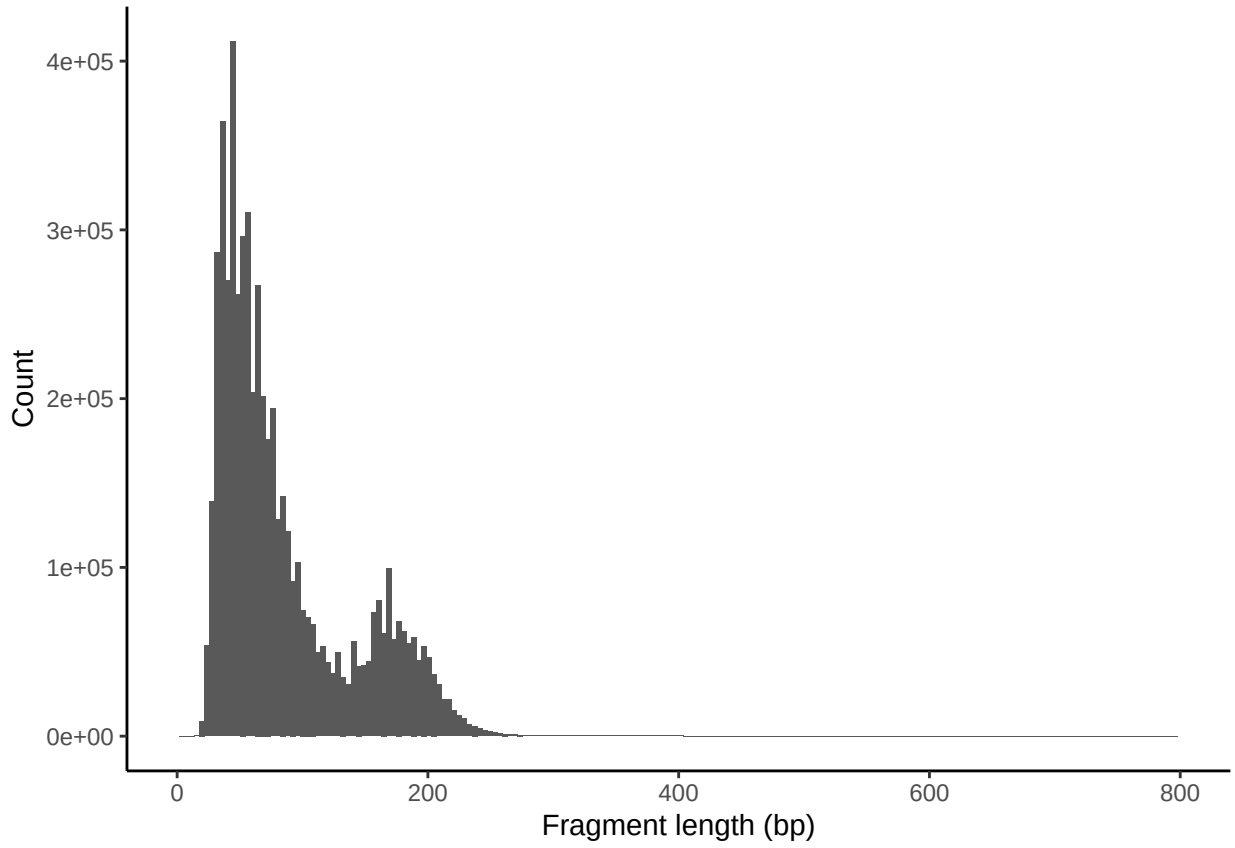
```
# We have to leave the ggplot() empty because the Seurat object seu is not
# a data frame technically.
ggplot() + geom_histogram(aes(x = seu$nucleosome_signal), binwidth = 0.01) +
  ggtitle("Distribution of nucleosome signal") + xlab("Nucleosome signal")
```




```
# This threshold we can set to a reasonable value. Play around with it!
thr = 1.5

seu$nucleosome_group <- ifelse(seu$nucleosome_signal > thr,
                               paste0('NS > ',thr),
                               paste0('NS < ',thr))

# Plot a 10M region from chr1 as example, this size already seems large enough
FragmentHistogram(object = seu, group.by = 'nucleosome_group',
                  region = "chr1-0-100000000")
```

```
## Warning: Removed 99 rows containing non-finite values (‘stat_bin()‘).
```

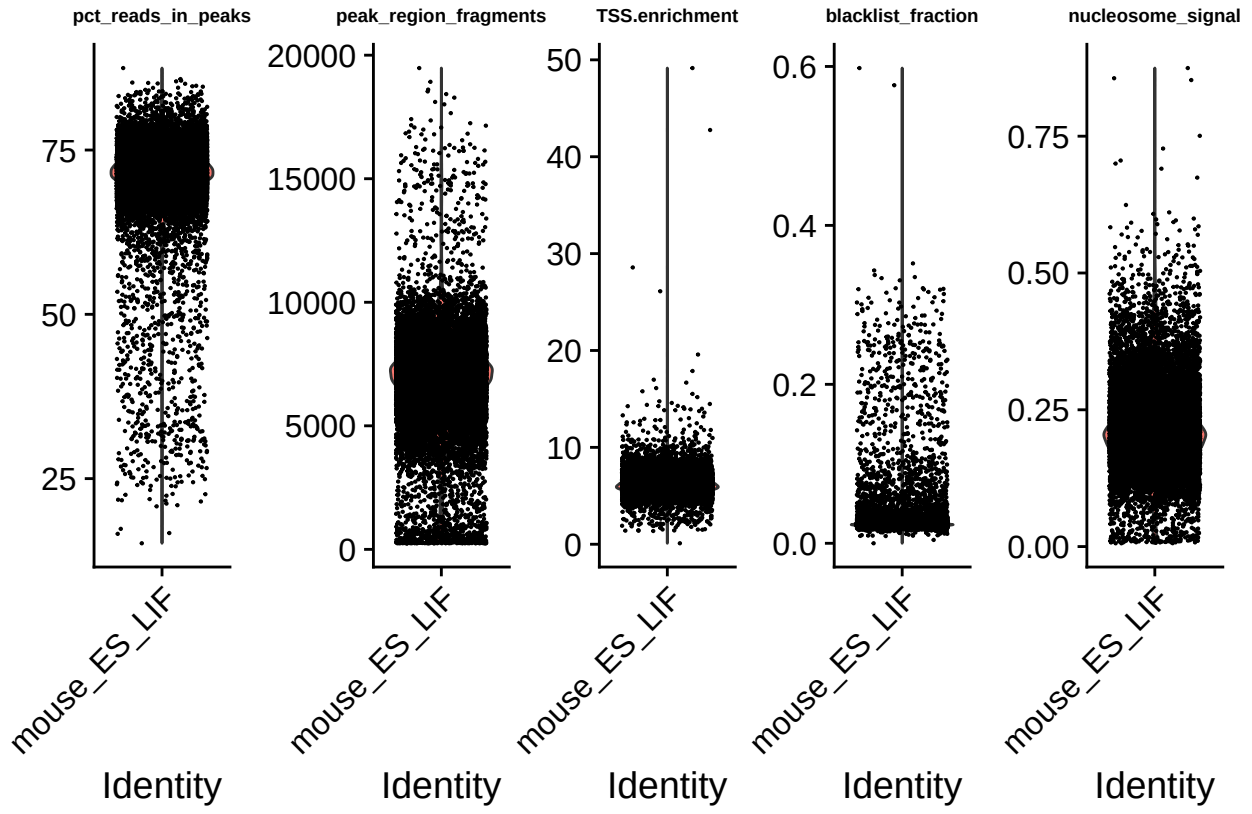
```
## Warning: Removed 2 rows containing missing values (‘geom_bar()‘).
```



Summary visualization of multiple quality metrics stats

We can also plot various features (the ones we manually defined above) at the same time in a Violin plot:

```
VlnPlot(
  object = seu,
  features = c('pct_reads_in_peaks', 'peak_region_fragments', 'TSS.enrichment',
               'blacklist_fraction', 'nucleosome_signal'),
  pt.size = 0.1,
  ncol = 5
) & theme(plot.title = element_text(size = 7))
```



Final filtering

After initial QC, we are now ready to filter our object and include only cells that pass our QC filters. That is, we remove cells that are outliers for the QC metrics that we defined above or that have been identified as doublets. Note that setting thresholds for filters is in a way very subjective and you may want to test different thresholds in order to judge their suitability and whether the applied stringency is sufficient for downstream analyses.

```
seu.s <- subset(
  x = seu,
  subset = peak_region_fragments > 1000 &
    peak_region_fragments < 15000 &
    pct_reads_in_peaks > 40 &
    blacklist_fraction < 0.05 &
    nucleosome_signal < 1.5 &
    TSS.enrichment > 2 &
    ATAC_doublets == "singlet"
)

seu.s
```

```
## An object of class Seurat
## 275719 features across 16488 samples within 1 assay
## Active assay: ATAC (275719 features, 0 variable features)
##  2 layers present: counts, data
```

As you can see, this retained most of the cells but not all!

Save unfiltered object to disk

It is time to save our original Seurat object to disk! We do so in a special format, called `rds`, which we can use later to quickly load our Seurat object back into R using the function `readRDS()`. In fact, in all other vignettes that we will use in the course, this will be our first step: Loading the preprocessed Seurat object into R using the `readRDS()` function!

```
# If we are sure that we do not use the unfiltered seu object anymore,
# we can delete it here, or save it to disk
saveRDS(seu, file = paste0(outFolder,"obj.rds"))

# Let us not delete it here to allow changing the filtering above
# if you need to save memory, delete the (unfiltered) Seurat oject
rm(seu)
```

Normalization and dimensionality reduction

- 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, as we do for scRNA-seq. 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 all features, though we note that we see 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 (for users who are more familiar with scRNA-seq, you can think of this as analogous to the output of PCA).

The combined steps of TF-IDF followed by SVD are known as latent semantic indexing (LSI), and were first introduced for the analysis of scATAC-seq data by Cusanovich et al. 2015.

You may have heard about `scTransform` for scRNA-seq data, wondering whether we can also apply this to scATAC-Seq data. The short answer: no. The reason is as follows: `scTransform`expects the data to follow a negative binomial or poisson distribution, which is not the case for scATAC data. It is also trying to normalize the data in a way that accounts for the strong mean-variance relationship in scRNA-seq, which is not something that we observe in scATAC due to the much lower dynamic range. From a practical perspective also, we want to preserve the sparsity of the data when normalizing scATAC-seq due to the large number of features we have. If we performed a normalization that lost the sparsity, we would have a very large dense matrix that will require a huge amount of memory to hold. As you can see, each data type requires its own specific normalization that incorporates the data-type specifics.

In the following, we give more background on dimensionality reduction with scATAC-seq data, which we took from the archR website. In general, dimensionality reduction with scATAC-seq is challenging due to the sparsity of the data. In scATAC-seq, a particular site can either be accessible on one allele, both alleles, or no alleles. Even in higher-quality scATAC-seq data, the majority of accessible regions are not transposed and this leads to many loci having 0 accessibility alleles. Moreover, when we

see (for example) three Tn5 insertions within a single peak region in a single cell, the sparsity of the data prevents us from confidently determining that this site in this cell is actually three times more accessible than another cell that only has one insertion in the same site. For this reason a lot of analytical strategies work on a binarized scATAC-seq data matrix. This binarized matrix still ends up being mostly 0s because transposition is rare. However, it is important to note that a 0 in scATAC-seq could mean "non-accessible" or "not sampled" and these two inferences are very different from a biological standpoint. Because of this, the 1s have information and the 0s do not. This low information content is what makes our scATAC-seq data sparse.

If you were to perform a standard dimensionality reduction, like *Principal Component Analysis* (PCA), on this sparse insertion counts matrix and plot the top two principal components (PCs), you would not obtain the desired result because the sparsity causes high inter-cell similarity at all of the 0 positions. To get around this issue, we use a layered dimensionality reduction approach. First, we use *Latent Semantic Indexing* (LSI), an approach from natural language processing that was originally designed to assess document similarity based on word counts. This solution was created for natural language processing because the data is sparse and noisy (many different words and many low frequency words). LSI was first introduced for scATAC-seq by *Cusanovich et al.* (Science 2015). In the case of scATAC-seq, different samples are the documents and different regions/peaks are the words. First, we calculate the term frequency by depth normalization per single cell. These values are then normalized by the inverse document frequency which weights features by how often they occur to identify features that are more "specific" rather than commonly accessible. The resultant *term frequency-inverse document frequency* (TF-IDF) matrix reflects how important a word (aka region/peak) is to a document (aka sample). Then, through a technique called *singular value decomposition* (SVD), the most valuable information across samples is identified and represented in a lower dimensional space. LSI allows you to reduce the dimensionality of the sparse insertion counts matrix from many thousands to tens or hundreds. Then, a more conventional dimensionality reduction technique, such as *Uniform Manifold Approximation and Projection* (UMAP) or *t-distributed stochastic neighbor embedding* (t-SNE) (see below) can be used to visualize the data. Let's create a two-dimensional *embedding* of the data!

```
seu.s <- seu.s %>%
        RunTFIDF(method = 1) %>%
        FindTopFeatures(min.cutoff = "q25") %>%
        RunSVD(reduction.key = 'LSI_', reduction.name = 'lsi')
```

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

```
## Running SVD
```

```
## Scaling cell embeddings
```

Examining the LSI components and removing components

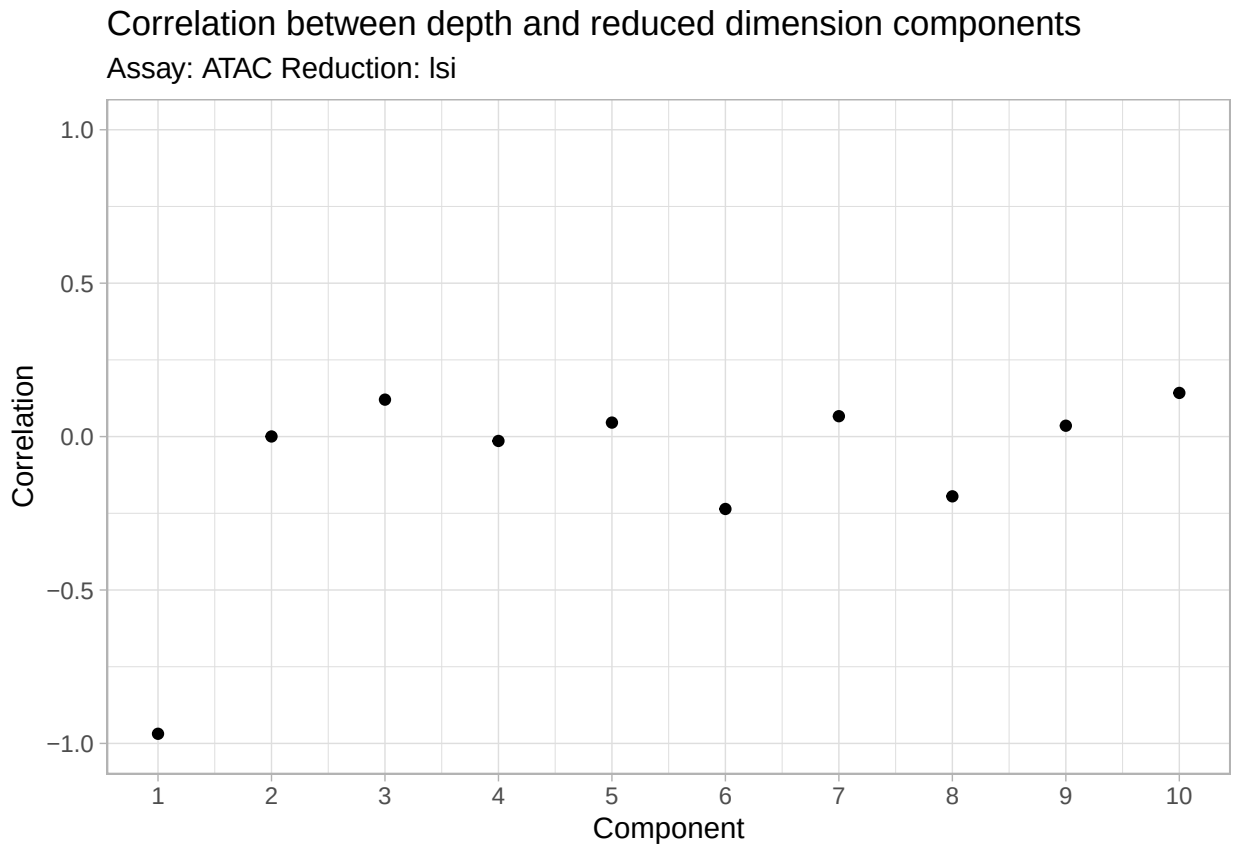
The first LSI component often captures sequencing depth (technical variation) rather than biological variation. If this is the case, the component should be removed from downstream analysis. We can assess the correlation between each LSI component and sequencing depth using the `DepthCor()` function:

```
p1 = DepthCor(seu.s, reduction = "lsi")
p1
```

```
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## width unknown for character 0x9
```

```
## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
```

```
## width unknown for character 0x9

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## width unknown for character 0x9

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## width unknown for character 0x9

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## width unknown for character 0x9

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## width unknown for character 0x9

## Warning in grid.Call(C_textBounds, as.graphicsAnnot(x$label), x$x, x$y, : font
## width unknown for character 0x9

## Warning in grid.Call.graphics(C_text, as.graphicsAnnot(x$label), x$x, x$y, :
## font width unknown for character 0x9
```



Here we see there is a very strong (negative) correlation between the first LSI component and the total number of counts for the cell, so we will perform downstream steps without this component.

Non-linear dimension reduction for visualization and 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. The functions `RunUMAP()`, `FindNeighbors()`, and `FindClusters()` all come from the Seurat package.

Visualize embeddings: UMAP

Th following description is taken and slightly modified from here. In general, embeddings such as *Uniform Manifold Approximation and Projection* (UMAP) or *t-distributed stochastic neighbor embedding* (t-SNE), are used to visualize single cells in reduced dimension space. They each have distinct advantages and disadvantages. We call them *embeddings* because they are strictly used to visualize the clusters and are not used to identify clusters which is done in an LSI sub-space as mentioned before. The primary difference between UMAP and t-SNE is the interpretation of the distance between cells or clusters. t-SNE is designed to preserve the local structure in the data while UMAP is designed to preserve both the local and most of the global structure in the data. In theory, this means that the distance between two clusters is not informative in t-SNE but is informative in UMAP. For example, t-SNE does not allow you to say that Cluster A is more similar to Cluster B than it is to Cluster C based on the observation that Cluster A is located closer to Cluster B than Cluster C on the t-SNE. UMAP, on the other hand, is designed to permit this type of comparison, though it is worth noting that UMAP is a new enough method that this is still being flushed out in the literature.

It is important to note that neither t-SNE nor UMAP are naturally deterministic (same input always gives exactly the same output). However, t-SNE shows much more randomness across multiple replicates of the same input than does UMAP. Moreover, UMAP as implemented in the `uwot` package is effectively deterministic when using the same random seed. The choice of whether to use UMAP or t-SNE is nuanced but from our experience, UMAP works very well for a diverse set of applications and this is our standard choice for scATAC-seq data. UMAP also performs faster than t-SNE. Perhaps most importantly, with UMAP you can create an embedding and project new samples into that embedding and this is not possible with t-SNE because the fitting and prediction of data happens simultaneously.

Regardless of which method you choose, the input parameters can have drastic effects on the resulting embedding. Because of this, it is important to understand the various input parameters and to tweak these to best meet the needs of your own data.

Here, due to the aforementioned advantages of UMAP, we will focus on UMAPs only and ignore t-SNE. Feel free to also produce t-SNE plots if you are interested!

```
# First, we explicitly define some parameters for creating the UMAP and
# subsequent functions related to clustering

# Exclude the first LSI component due to the strong correlation, see the plot above
dims_sel = 2:20
nn = 30
res = 0.5
clusters_alg = 1

# We are now ready to run it!
seu.s <- seu.s %>%
            RunUMAP(dims = dims_sel,
                    n.neighbors = nn, verbose = FALSE,
                     # a = 0.9, b = 0.6,
                    metric = "cosine",
                    reduction = "lsi") %>%
```

```
            FindNeighbors(dims = dims_sel,
                          graph.name = "ATAC_snn",reduction = "lsi") %>%
            FindClusters(resolution = res, algorithm = clusters_alg,
                         graph.name = "ATAC_snn")
```

```
## Warning: The default method for RunUMAP has changed from calling Python UMAP via reticulate to the R
## To use Python UMAP via reticulate, set umap.method to 'umap-learn' and metric to 'correlation'
## This message will be shown once per session
```

```
## Found more than one class "dist" in cache; using the first, from namespace 'BiocGenerics'
```

```
## Also defined by 'spam'
```

```
## Found more than one class "dist" in cache; using the first, from namespace 'BiocGenerics'
```

```
## Also defined by 'spam'
```

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

```
## Computing SNN
```

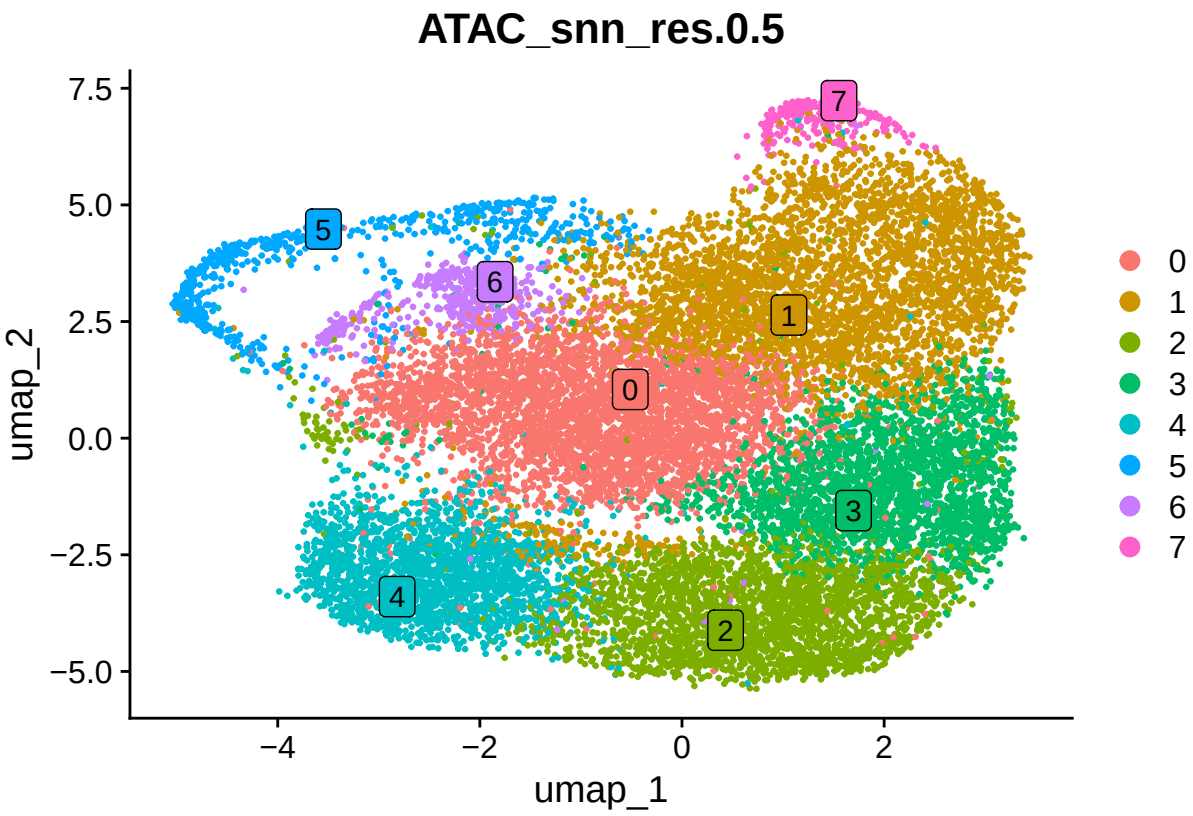
```
## Only one graph name supplied, storing nearest-neighbor graph only
```

```
## Modularity Optimizer version 1.3.0 by Ludo Waltman and Nees Jan van Eck
##
## Number of nodes: 16488
## Number of edges: 155115
##
## Running Louvain algorithm...
## Maximum modularity in 10 random starts: 0.7406
## Number of communities: 16
## Elapsed time: 1 seconds
```

```
## 8 singletons identified. 8 final clusters.
```

Visualize clusters in embedded space

```
res = 0.5

DimPlot(object = seu.s, pt.size = 0.5,
            group.by = paste0("ATAC_snn_res.",res),
            label = TRUE, repel = TRUE, label.box = TRUE)
```



Cluster stability: Generating a cluster tree

In what follows, we will visualize clusters and cluster stability using a so called clustering tree as implemented by the `clustree` package in R. In general, clustering analyses are mainly used to group similar samples. The most challenging choice is often the number of clusters to use. This is usually a priori controlled by a parameter provided to the clustering algorithm, such as k for k-means clustering.

Statistics designed to help you make this choice typically either compare two clusterings or score a single clustering. A clustering tree is different in that it visualises the relationships between at a range of resolutions.

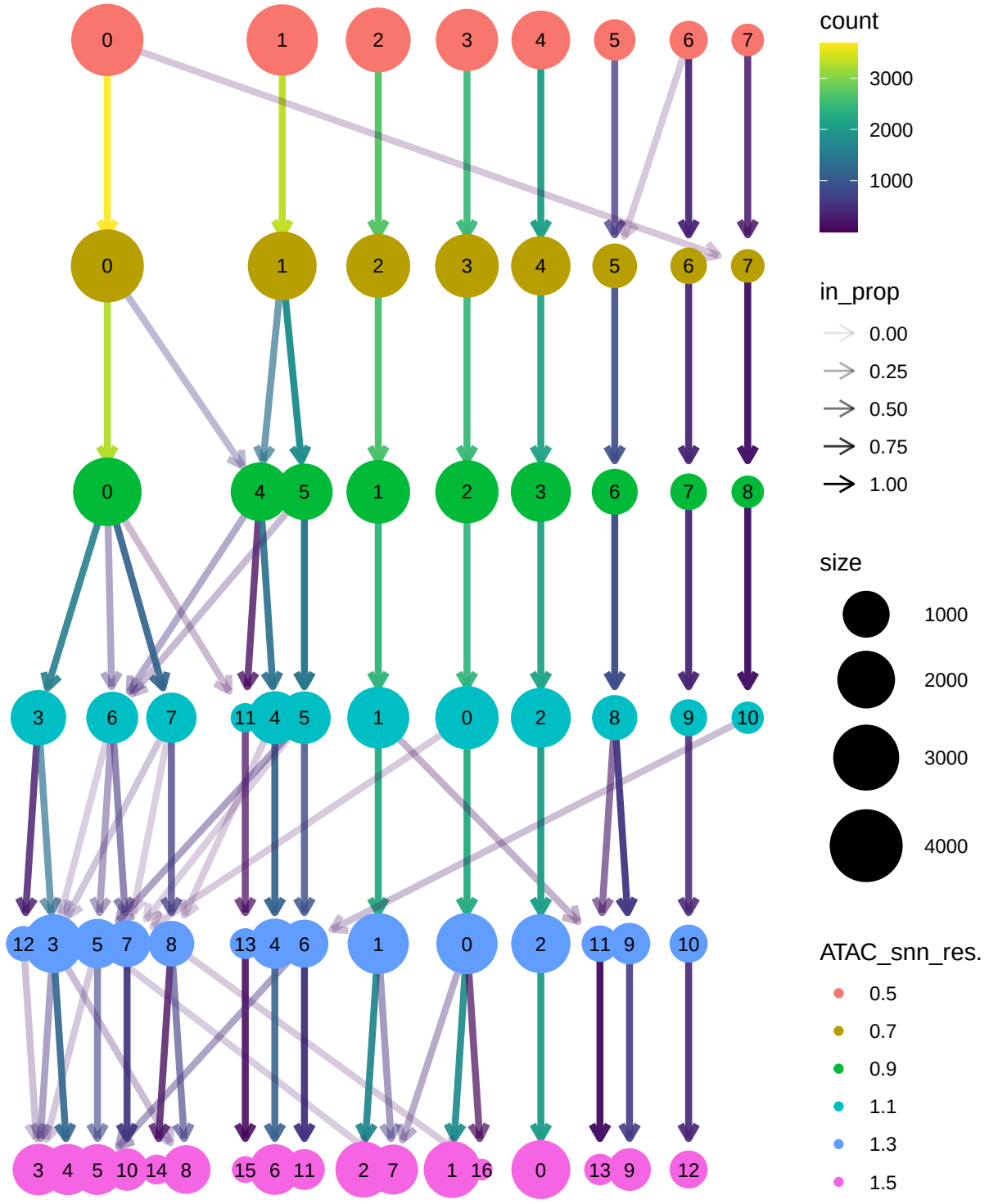
To do so, we need to look at how cells move as the clustering resolution is increased. Each cluster forms a node in the tree and edges are constructed by considering the cells in a cluster at a lower resolution (say k=2) that end up in a cluster at the next highest resolution (say k=3). By connecting clusters in this way we can see how clusters are related to each other, which are clearly distinct and which are unstable.

Let's try this approach out to get a better understanding of our clusters here! First, we need to calculate different clusterings with varying resolutions, which can be done via the `resolution` parameter in the `FindClusters` function.

```
clusters_alg = 1
for (res in seq(0.5,1.6,0.2)){
  # In each iteration, a new metadata is added to the Seurat object with the basename ATAC_snn
  seu.s = FindClusters(seu.s, resolution = res, algorithm = clusters_alg,
                       graph.name = "ATAC_snn", verbose = FALSE)
}
```

We are now ready to plot our cluster tree!

```
clustree(seu.s)
```



Lastly, we need to rerun `FindClusters` one last time to make sure we use the 0.5 resolution (that we had in the object before but that was overwritten by our code chunk to prepare the `clustree` function) for all subsequent analysis:

```r
# Let's make sure the 0.5 resolution is the one that we keep for further analyses,
# not the last one (1.5) from the chunk above
seu.s = FindClusters(seu.s, resolution = 0.5, algorithm = 1, graph.name = "ATAC_snn", verbose = FALSE)
```

Save filtered object to disk

We reached the end of the first vignette and the first set of analyses. Finally, we can now save our filtered Seurat object to disk, in analogy to what we did above with the original, unfiltered `Seurat` object.

```r
saveRDS(seu.s, file = paste0(outFolder,"obj.filt.rds"))
```

Further reading

10x Genomics: What is Cell Ranger ATAC?

scATAC-seq data analysis tools and papers

Beginner's Guide to Understanding Single-Cell ATAC-Seq

scATAC.Explorer: A Collection of Single-cell ATAC Sequencing Datasets and Corresponding Metadata for R/Bioconductor

ShinyArchR.UiO: user-friendly,integrative and open-source tool for visualization of single-cell ATAC-seq data using ArchR

Efficient pre-processing of Single-cell ATAC-seq data

Dimensionality Reduction for scATAC Data

CellWalkR: R package for integrating and visualizing single-cell and bulk data to resolve regulatory elements

scDblFinder R package for doublet detection

Grandi, F.C., Modi, H., Kampman, L. and Corces, M.R., 2022. Chromatin accessibility profiling by ATAC-seq. Nature Protocols, pp.1-35. [2] Yu, W., Uzun, Y., Zhu, Q., Chen, C. and Tan, K., 2020. scATAC-pro: a comprehensive workbench for single-cell chromatin accessibility sequencing data. Genome biology, 21(1), pp.1-17.

Granja, J.M., Corces, M.R., Pierce, S.E., Bagdatli, S.T., Choudhry, H., Chang, H.Y. and Greenleaf, W.J., 2021. ArchR is a scalable software package for integrative single-cell chromatin accessibility analysis. Nature genetics, 53(3), pp.403-411.

Hao, Y., Hao, S., Andersen-Nissen, E., Mauck III, W.M., Zheng, S., Butler, A., Lee, M.J., Wilk, A.J., Darby, C., Zager, M. and Hoffman, P., 2021. Integrated analysis of multimodal single-cell data. Cell, 184(13), pp.3573-3587.

Ji, Z., Zhou, W., Hou, W. and Ji, H., 2020. Single-cell ATAC-seq signal extraction and enhancement with SCATE. Genome biology, 21(1), pp.1-36.

Baek, S. and Lee, I., 2020. Single-cell ATAC sequencing analysis: From data preprocessing to hypothesis generation. Computational and structural biotechnology journal, 18, pp.1429-1439.

A single-cell atlas of chromatin accessibility in the human genome

Comprehensive analysis of single cell ATAC-seq data with SnapATAC

Interesting blog post regarding normalization and dimensionality reduction of scATAC data

Session info

It is good practice to print the so-called session info at the end of an R script, which prints all loaded libraries, their versions etc. This can be helpful for reproducibility and recapitulating which package versions have been used to produce the results obtained above.

```
sessionInfo()
```

```
## R version 4.2.2 (2022-10-31)
## Platform: x86_64-pc-linux-gnu (64-bit)
## Running under: Rocky Linux 8.8 (Green Obsidian)
##
## Matrix products: default
## BLAS/LAPACK: /g/easybuild/x86_64/Rocky/8/haswell/software/FlexiBLAS/3.2.1-GCC-12.2.0/lib64/libflexibl
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8       LC_NUMERIC=C
##  [3] LC_TIME=en_US.UTF-8        LC_COLLATE=en_US.UTF-8
##  [5] LC_MONETARY=en_US.UTF-8    LC_MESSAGES=en_US.UTF-8
##  [7] LC_PAPER=en_US.UTF-8       LC_NAME=C
##  [9] LC_ADDRESS=C               LC_TELEPHONE=C
## [11] LC_MEASUREMENT=en_US.UTF-8 LC_IDENTIFICATION=C
##
## attached base packages:
## [1] stats4    stats     graphics  grDevices utils     datasets  methods
## [8] base
##
## other attached packages:
##  [1] ensembldb_2.22.0            AnnotationFilter_1.22.0
##  [3] GenomicFeatures_1.50.4      AnnotationDbi_1.60.2
##  [5] scDblFinder_1.12.0          SingleCellExperiment_1.20.1
##  [7] SummarizedExperiment_1.28.0 Biobase_2.58.0
##  [9] GenomicRanges_1.50.2        MatrixGenerics_1.10.0
## [11] matrixStats_0.63.0          biovizBase_1.46.0
## [13] hdf5r_1.3.8                 clustree_0.5.0
## [15] ggraph_2.1.0                AnnotationHub_3.6.0
## [17] BiocFileCache_2.6.1         dbplyr_2.3.1
## [19] patchwork_1.1.2             lubridate_1.9.2
## [21] forcats_1.0.0               stringr_1.5.0
## [23] dplyr_1.1.0                 purrr_1.0.1
## [25] readr_2.1.4                 tidyr_1.3.0
## [27] tibble_3.2.0                ggplot2_3.4.1
## [29] tidyverse_2.0.0             GenomeInfoDb_1.34.9
## [31] IRanges_2.32.0              S4Vectors_0.36.2
## [33] BiocGenerics_0.44.0         Seurat_5.0.2
## [35] SeuratObject_5.0.1          sp_1.6-0
## [37] Signac_1.12.9007
##
## loaded via a namespace (and not attached):
##   [1] rappdirs_0.3.3               rtracklayer_1.58.0
##   [3] scattermore_1.2              bit64_4.0.5
##   [5] knitr_1.42                   irlba_2.3.5.1
##   [7] DelayedArray_0.24.0          data.table_1.14.8
##   [9] rpart_4.1.19                 KEGGREST_1.38.0
```

```
##  [11] RCurl_1.98-1.10              generics_0.1.3
##  [13] ScaledMatrix_1.6.0           cowplot_1.1.1
##  [15] RSQLite_2.3.0                RANN_2.6.1
##  [17] future_1.32.0                bit_4.0.5
##  [19] tzdb_0.3.0                   spatstat.data_3.0-1
##  [21] xml2_1.3.3                   httpuv_1.6.9
##  [23] viridis_0.6.2                xfun_0.37
##  [25] hms_1.1.2                    evaluate_0.20
##  [27] promises_1.2.0.1             fansi_1.0.4
##  [29] restfulr_0.0.15              progress_1.2.2
##  [31] igraph_1.4.1                 DBI_1.1.3
##  [33] htmlwidgets_1.6.1            spatstat.geom_3.1-0
##  [35] ellipsis_0.3.2               RSpectra_0.16-1
##  [37] backports_1.4.1              sparseMatrixStats_1.10.0
##  [39] biomaRt_2.54.0               deldir_1.0-6
##  [41] vctrs_0.6.0                  ROCR_1.0-11
##  [43] abind_1.4-5                  cachem_1.0.7
##  [45] withr_2.5.0                  ggforce_0.4.1
##  [47] BSgenome_1.66.3              progressr_0.13.0
##  [49] checkmate_2.1.0              sctransform_0.4.1
##  [51] GenomicAlignments_1.34.1     scran_1.26.2
##  [53] prettyunits_1.1.1            goftest_1.2-3
##  [55] cluster_2.1.4                dotCall64_1.0-2
##  [57] lazyeval_0.2.2               crayon_1.5.2
##  [59] spatstat.explore_3.1-0       labeling_0.4.2
##  [61] edgeR_3.40.2                 pkgconfig_2.0.3
##  [63] tweenr_2.0.2                 vipor_0.4.5
##  [65] nlme_3.1-162                 ProtGenerics_1.30.0
##  [67] nnet_7.3-18                  rlang_1.1.0
##  [69] globals_0.16.2               lifecycle_1.0.3
##  [71] miniUI_0.1.1.1               filelock_1.0.2
##  [73] fastDummies_1.6.3            rsvd_1.0.5
##  [75] dichromat_2.0-0.1            ggrastr_1.0.2
##  [77] polyclip_1.10-4              RcppHNSW_0.4.1
##  [79] lmtest_0.9-40                Matrix_1.6-5
##  [81] zoo_1.8-11                   beeswarm_0.4.0
##  [83] base64enc_0.1-3              ggridges_0.5.4
##  [85] png_0.1-8                    viridisLite_0.4.1
##  [87] rjson_0.2.21                 bitops_1.0-7
##  [89] KernSmooth_2.23-20           spam_2.9-1
##  [91] Biostrings_2.66.0            DelayedMatrixStats_1.20.0
##  [93] blob_1.2.4                   parallelly_1.34.0
##  [95] spatstat.random_3.1-4        beachmat_2.14.0
##  [97] scales_1.2.1                 memoise_2.0.1
##  [99] magrittr_2.0.3               plyr_1.8.8
## [101] ica_1.0-3                    zlibbioc_1.44.0
## [103] compiler_4.2.2               dqrng_0.3.0
## [105] BiocIO_1.8.0                 RColorBrewer_1.1-3
## [107] fitdistrplus_1.1-8           Rsamtools_2.14.0
## [109] cli_3.6.0                    XVector_0.38.0
## [111] listenv_0.9.0                pbapply_1.7-0
## [113] htmlTable_2.4.1              Formula_1.2-5
## [115] MASS_7.3-58.3                tidyselect_1.2.0
## [117] stringi_1.7.12               highr_0.10
```

```
## [119] yaml_2.3.7                    locfit_1.5-9.7
## [121] BiocSingular_1.14.0           ggrepel_0.9.3
## [123] grid_4.2.2                    VariantAnnotation_1.44.1
## [125] fastmatch_1.1-3               tools_4.2.2
## [127] timechange_0.2.0              future.apply_1.10.0
## [129] parallel_4.2.2                rstudioapi_0.14
## [131] bluster_1.8.0                 foreign_0.8-84
## [133] metapod_1.6.0                 gridExtra_2.3
## [135] farver_2.1.1                  Rtsne_0.16
## [137] digest_0.6.31                 BiocManager_1.30.20
## [139] shiny_1.7.4                   Rcpp_1.0.10
## [141] scuttle_1.8.4                 BiocVersion_3.16.0
## [143] later_1.3.0                   RcppAnnoy_0.0.21
## [145] httr_1.4.5                    colorspace_2.1-0
## [147] XML_3.99-0.13                 tensor_1.5
## [149] reticulate_1.28               splines_4.2.2
## [151] statmod_1.5.0                 uwot_0.1.16
## [153] RcppRoll_0.3.0                spatstat.utils_3.0-2
## [155] scater_1.26.1                 graphlayouts_0.8.4
## [157] xgboost_1.7.3.1               plotly_4.10.1
## [159] xtable_1.8-4                  jsonlite_1.8.4
## [161] tidygraph_1.2.3               R6_2.5.1
## [163] Hmisc_5.0-1                   pillar_1.8.1
## [165] htmltools_0.5.4               mime_0.12
## [167] glue_1.6.2                    fastmap_1.1.1
## [169] BiocParallel_1.32.5           BiocNeighbors_1.16.0
## [171] interactiveDisplayBase_1.36.0 codetools_0.2-19
## [173] utf8_1.2.3                    lattice_0.20-45
## [175] spatstat.sparse_3.0-1         ggbeeswarm_0.7.1
## [177] curl_5.0.0                    leiden_0.4.3
## [179] limma_3.54.2                  survival_3.5-5
## [181] rmarkdown_2.20                munsell_0.5.0
## [183] GenomeInfoDbData_1.2.9        reshape2_1.4.4
## [185] gtable_0.3.1
```