

# Multiome tutorial - MultiAssayExperiment

**Xiaosai Yao**

3 June 2023

## Package

epiregulon 1.0.25

## Contents

1	Introduction . . . . .	2
2	Installation . . . . .	2
3	Data preparation . . . . .	2
4	Quick start . . . . .	4
4.1	Retrieve bulk TF ChIP-seq binding sites . . . . .	4
4.2	Link ATAC-seq peaks to target genes . . . . .	4
4.3	Add TF motif binding to peaks . . . . .	5
4.4	Generate regulons . . . . .	6
4.5	Network pruning (highly recommended) . . . . .	6
4.6	Add Weights . . . . .	7
4.7	(Optional) Annotate with TF motifs . . . . .	8
4.8	Calculate TF activity . . . . .	8
4.9	Perform differential activity . . . . .	9
4.10	Visualize the results . . . . .	9
4.11	Geneset enrichment . . . . .	13
4.12	Network analysis . . . . .	15
4.13	Differential networks . . . . .	16
5	Session Info . . . . .	20

## 1 Introduction

---

This tutorial walks through the same dataset used in the “multiome tutorial - archR workflow”. This is a dataset generated by infecting LNCaP cells with NKX2-1 and GATA6 to examine the effects of these TFs on AR activity.

## 2 Installation

---

Epiregulon is currently available on R/dev

```
library(epiregulon)
```

Alternatively, you could install from github

```
devtools::install_github(repo = 'xiaosaiyao/epiregulon')
```

```
library(epiregulon)
```

## 3 Data preparation

---

Single cell preprocessing needs to be performed by user's favorite methods prior to using Epiregulon. The following components are required: 1. Peak matrix from scATAC-seq 2. Gene expression matrix from either paired or unpaired scRNA-seq. RNA-seq integration needs to be performed for unpaired dataset. 3. Dimensionality reduction matrix from with either single modalities or joint scRNA-seq and scATAC-seq

Multiome data can now be conveniently processed by `initiate.archr` and then `gp.sa.archr` to obtain peak matrices. Finally, the archR project can be uploaded into DatasetDB as a `MultiAssayExperiment` object using `maw.archr::importArchr` or `maw.archr::create.mae.with.multiple.sces.from.archr`

```
# load the MAE object
library(scMultiome)
#> Loading required package: AnnotationHub
#> Loading required package: BiocFileCache
#> Loading required package: dbplyr
#>
#> Attaching package: 'AnnotationHub'
#> The following object is masked from 'package:Biobase':
#>
#>     cache
#> Loading required package: ExperimentHub
#> Loading required package: MultiAssayExperiment
#>
#> Attaching package: 'scMultiome'
#> The following object is masked from 'package:scRNaseq':
#>
#>     listDatasets
```

## Multiome tutorial - MultiAssayExperiment

```
mae <- scMultiome::reprogramSeq()
#> snapshotDate(): 2023-04-24
#> see ?scMultiome and browseVignettes('scMultiome') for documentation
#> loading from cache

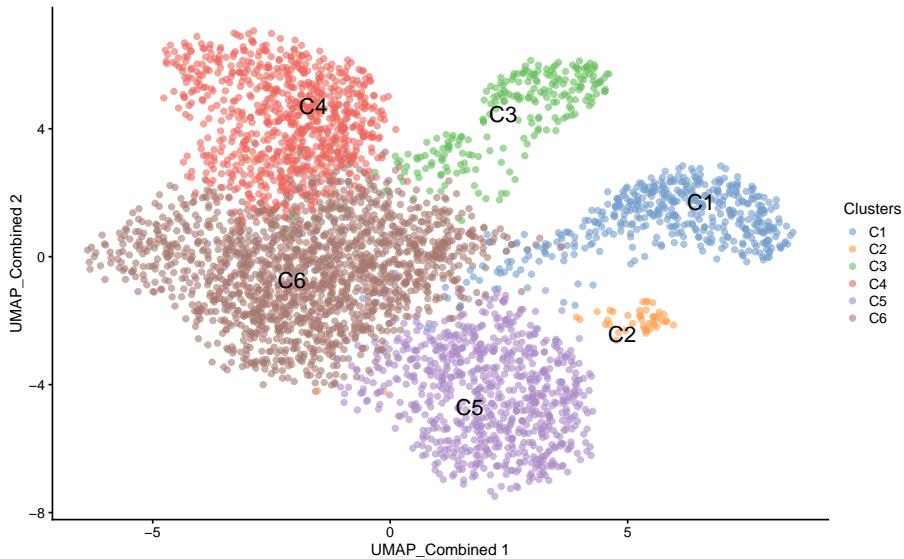
# peak matrix
PeakMatrix <- mae[["PeakMatrix"]]

# expression matrix
GeneExpressionMatrix <- mae[["GeneExpressionMatrix"]]
rownames(GeneExpressionMatrix) <- rowData(GeneExpressionMatrix)$name
assay(GeneExpressionMatrix) <- as(log2(assay(GeneExpressionMatrix)+1), "CsparseMatrix")

# dimensional reduction matrix
reducedDimMatrix <- reducedDim(mae[['TileMatrix500']], "LSI_ATAC")
```

Visualize singleCellExperiment by UMAP

```
# transfer UMAP_combined from TileMatrix to GeneExpressionMatrix
reducedDim(GeneExpressionMatrix, "UMAP_Combined") <- reducedDim(mae[['TileMatrix500']], "UMAP_Combined")
scater::plotReducedDim(GeneExpressionMatrix,
                       dimred = "UMAP_Combined",
                       text_by = "Clusters",
                       colour_by = "Clusters")
```



## 4 Quick start

---

### 4.1 Retrieve bulk TF ChIP-seq binding sites

First, we retrieve the information of TF binding sites collected from Cistrome and ENCODE ChIP-seq, which are hosted on Genomitory. Currently, human genomes HG19 and HG38 and mouse mm10 are available.

```
grl <- getTFMotifInfo(genome = "hg38")
#> snapshotDate(): 2023-04-24
#> see ?scMultiome and browseVignettes('scMultiome') for documentation
#> loading from cache
head(grl)
#> GRangesList object of length 6:
#> $`5-hmC`
#> GRanges object with 24048 ranges and 0 metadata columns:
#>   seqnames      ranges strand
#>   <Rle>      <IRanges>  <Rle>
#> [1] chr1    10000-10685 *
#> [2] chr1    13362-13694 *
#> [3] chr1    29631-29989 *
#> [4] chr1    40454-40754 *
#> [5] chr1    135395-135871 *
#> ...
#> [24044] chrY 56864377-56864627 *
#> [24045] chrY 56876124-56876182 *
#> [24046] chrM     84-2450 *
#> [24047] chrM     13613-14955 *
#> [24048] chrM     15134-16490 *
#> -----
#> seqinfo: 25 sequences from an unspecified genome; no seqlengths
#>
#> ...
#> <5 more elements>
```

### 4.2 Link ATAC-seq peaks to target genes

Next, we compute peak to gene correlations using a custom algorithm that has similar performance to ArchR's P2G function.

```
set.seed(1010)
p2g <- calculateP2G(peakMatrix = PeakMatrix,
                      expMatrix = GeneExpressionMatrix,
                      reducedDim = reducedDimMatrix)
#> Using epiregulon to compute peak to gene links...
#> performing k means clustering to form metacells
#> Computing correlation

p2g
```

```
#> DataFrame with 25693 rows and 8 columns
#>   idxATAC      chr    start     end   idxRNA target
#>   <integer> <character> <integer> <integer> <integer> <array>
#> 1       1      chr1  817121  817621      19 FAM41C
#> 2       6      chr1  869650  870150      14 AL669831.2
#> 3      10      chr1  920987  921487      19 FAM41C
#> 4      22      chr1  960317  960817      19 FAM41C
#> 5      22      chr1  960317  960817      28 PERM1
#> ...
#> 25689  126586      chrX 155071227 155071727  36422 MTCP1
#> 25690  126590      chrX 155228844 155229344  36426 CLIC2
#> 25691  126592      chrX 155334445 155334945  36426 CLIC2
#> 25692  126596      chrX 155820104 155820604  36436 VAMP7
#> 25693  126599      chrX 155897986 155898486  36436 VAMP7
#>   Correlation distance
#>   <matrix> <integer>
#> 1 0.505457  50578
#> 2 0.614009 108540
#> 3 0.662441  50587
#> 4 0.635411  89917
#> 5 0.614227 18210
#> ...
#> 25689 0.504839 0
#> 25690 0.830640 103268
#> 25691 0.529991 0
#> 25692 0.629908 58739
#> 25693 0.505036 16442
```

## 4.3 Add TF motif binding to peaks

The next step is to add the TF binding information by overlapping regions of the peak matrix with the bulk chip-seq database loaded in 2. The user can supply either an archR project path and this function will retrieve the peak matrix, or a peakMatrix in the form of a Granges object or RangedSummarizedExperiment.

```
overlap <- addTFMotifInfo(grl = grl, p2g = p2g, peakMatrix = PeakMatrix)
#> Computing overlap...
#> Success!
head(overlap)
#>   idxATAC idxTF      tf
#> 1       1      2 5-mC
#> 2       1     22 AML1-ETO
#> 3       1     25 AR
#> 4       1     49 ATF1
#> 5       1     50 ATF2
#> 6       1     51 ATF3
```

## 4.4 Generate regulons

A long format dataframe, representing the inferred regulons, is then generated. The dataframe consists of three columns:

- tf (transcription factor)
- target gene
- peak to gene correlation between tf and target gene

```
regulon <- getRegulon(p2g = p2g, overlap = overlap, aggregate = FALSE)
regulon
#> DataFrame with 3187407 rows and 11 columns
#>   idxATAC      chr    start     end   idxRNA target
#>   <integer> <character> <integer> <integer> <integer> <character>
#> 1       1      chr1    817121   817621      19 FAM41C
#> 2       1      chr1    817121   817621      19 FAM41C
#> 3       1      chr1    817121   817621      19 FAM41C
#> 4       1      chr1    817121   817621      19 FAM41C
#> 5       1      chr1    817121   817621      19 FAM41C
#> ...
#> 3187403 126599      chrX 155897986 155898486 36436 VAMP7
#> 3187404 126599      chrX 155897986 155898486 36436 VAMP7
#> 3187405 126599      chrX 155897986 155898486 36436 VAMP7
#> 3187406 126599      chrX 155897986 155898486 36436 VAMP7
#> 3187407 126599      chrX 155897986 155898486 36436 VAMP7
#>   all distance   idxTF      tf    corr
#>   <numeric> <integer> <integer> <character> <matrix>
#> 1 0.505457    50578      2      5-mC
#> 2 0.505457    50578     22 AML1-ETO
#> 3 0.505457    50578     25      AR
#> 4 0.505457    50578     49      ATF1
#> 5 0.505457    50578     50      ATF2
#> ...
#> 3187403 0.505036   16442     436 HOXB13
#> 3187404 0.505036   16442     669 NANOG
#> 3187405 0.505036   16442     762 ONECUT2
#> 3187406 0.505036   16442    1044 SUMO2
#> 3187407 0.505036   16442    1116 TLE3
```

## 4.5 Network pruning (highly recommended)

Epiregulon prunes the network by performing tests of independence on the observed number of cells jointly expressing transcription factor (TF), regulatory element (RE) and target gene (TG) vs the expected number of cells if TF/RE and TG are independently expressed. We implement two tests, the binomial test and the chi-square test. In the binomial test, the expected probability is  $P(\text{TF}, \text{RE}) * P(\text{TG})$ , and the number of trials is the total number of cells, and the observed successes is the number of cells jointly expressing all three elements. In the chi-square test, the expected probability for having all 3 elements active is also  $P(\text{TF},$

## Multiome tutorial - MultiAssayExperiment

$P(\text{RE}) * P(\text{TG})$  and the probability otherwise is  $1 - P(\text{TF}, \text{RE}) * P(\text{TG})$ . The observed cell count for the active category is the number of cells jointly expressing all three elements, and the cell count for the inactive category is  $n - n_{\text{triple}}$ .

We calculate cluster-specific p-values if users supply cluster labels. This is useful if we are interested in cluster-specific networks. The pruned regulons can then be used to visualize differential networks for transcription factors of interest. See section on differential networks.

```
pruned.regulon <- pruneRegulon(expMatrix = GeneExpressionMatrix,
                                exp_assay = "logcounts",
                                peakMatrix = PeakMatrix,
                                peak_assay = "counts",
                                test = "chi.sq",
                                regulon,
                                clusters = GeneExpressionMatrix$Clusters,
                                prune_value = "pval",
                                regulon_cutoff = 0.05
                               )

pruned.regulon
```

## 4.6 Add Weights

While the 'pruneRegulon' function provides statistics on the joint occurrence of TF-RE-TG, we would like to further estimate the strength of regulation. Biologically, this can be interpreted as the magnitude of gene expression changes induced by transcription factor activity. Epiregulon estimates the regulatory potential using one of the four measures: 1) correlation between TF and target gene expression, 2) mutual information between the TF and target gene expression, 3) Wilcoxon test statistics of target gene expression in cells jointly expressing all 3 elements vs cells that do not, or 4) log 2 fold difference of target gene expression in cells jointly expressing all 3 elements vs cells that do not.

Three measures (correlation, Wilcoxon statistics and log 2 fold difference) give both the magnitude and directionality of changes whereas mutational information is always positive. The correlation and mutual information statistics are computed on the grouped pseudobulks by user-supplied cluster labels, whereas the Wilcoxon and log fold change group cells based on the joint expression of TF, RE and TG in each single cell.

```
regulon.w <- addWeights(regulon = pruned.regulon,
                         expMatrix = GeneExpressionMatrix,
                         exp_assay = "logcounts",
                         peakMatrix = PeakMatrix,
                         peak_assay = "counts",
                         clusters = GeneExpressionMatrix$Clusters,
                         block_factor = NULL,
                         tf_re.merge = TRUE,
                         method = "corr")

regulon.w
```

## 4.7 (Optional) Annotate with TF motifs

So far the gene regulatory network was constructed from TF ChIP-seq exclusively. Some users would prefer to further annotate the regulatory elements with the presence of motifs. We provide an option to annotate peaks with motifs from the Cisbp database. If no motifs are present for this particular factor (as in the case of co-factors or chromatin modifiers), we return NA. If motifs are available for a factor and the RE contains a motif, we return 1. If motifs are available and the RE does not contain a motif, we return 0.

If the user has already performed motif annotation with ArchR, we could also retrieve the results directly. See `?addMotifScore`

```
regulon.w.motif <- addMotifScore(regulon = regulon.w,
                                    peaks = rowRanges(PeakMatrix),
                                    species = "human",
                                    genome = "hg38")
#> annotating peaks with motifs
#>
#> Attaching package: 'Biostrings'
#> The following object is masked from 'package:base':
#>
#>     strsplit
#>
#> Attaching package: 'rtracklayer'
#> The following object is masked from 'package:AnnotationHub':
#>
#>     hubUrl
# if desired, set weight to 0 if no motif is found
regulon.w.motif$weight[regulon.w.motif$motif == 0] <- 0
```

## 4.8 Calculate TF activity

Finally, the activities for a specific TF in each cell are computed by averaging expressions of target genes linked to the TF weighted by the test statistics of choice, chosen from either correlation, mutual information, Wilcoxon test statistics or log fold change.

$$y = \frac{1}{n} \sum_{i=1}^n x_i * weights_i$$

where  $y$  is the activity of a TF for a cell  $n$  is the total number of targets for a TF  $x_i$  is the log count expression of target i where  $i \in \{1, 2, \dots, n\}$   $weights_i$  is the weight of TF and target i

```
score.combine <- calculateActivity(expMatrix = GeneExpressionMatrix,
                                    regulon = regulon.w,
                                    mode = "weight",
                                    method = "weightedMean",
                                    exp_assay = "logcounts",
                                    normalize = FALSE)
#> calculating TF activity from regulon using weightedmean
```

```
#> aggregating regulons...
#> creating weight matrix...
#> calculating activity scores...
#> normalize by the number of targets...
```

### 4.9 Perform differential activity

```
markers <- findDifferentialActivity(activity_matrix = score.combine,
                                       groups = GeneExpressionMatrix$hash_assignment,
                                       pval.type = "some",
                                       direction = "up",
                                       test.type = "t")
```

Take the top TFs

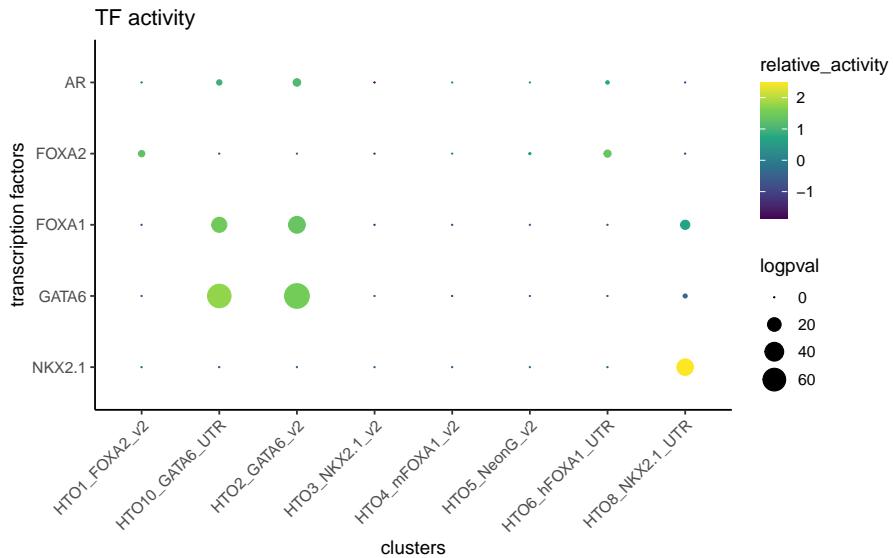
```
markers.sig <- getSigGenes(markers, topgenes = 5 )
#> Using a logFC cutoff of 0 for class HT01_FOXA2_v2
#> Using a logFC cutoff of 0.2 for class HT010_GATA6_UTR
#> Using a logFC cutoff of 0.2 for class HT02_GATA6_v2
#> Using a logFC cutoff of 0 for class HT03_NKX2.1_v2
#> Using a logFC cutoff of 0 for class HT04_mFOXA1_v2
#> Using a logFC cutoff of 0 for class HT05_NeonG_v2
#> Using a logFC cutoff of 0 for class HT06_hFOXA1_UTR
#> Using a logFC cutoff of 0 for class HT08_NKX2.1_UTR
```

### 4.10 Visualize the results

First visualize the known differential TFs by bubble plot

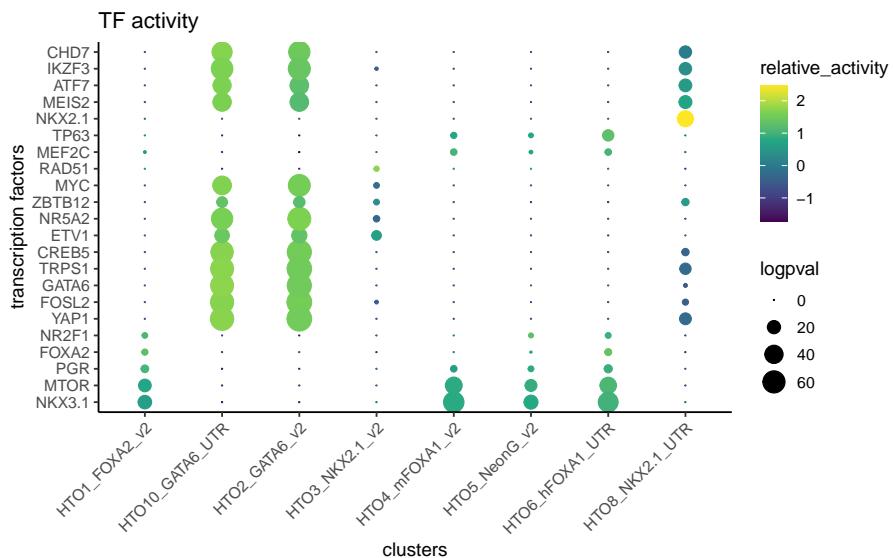
```
plotBubble(activity_matrix = score.combine,
           tf = c("NKX2-1", "GATA6", "FOXA1", "FOXA2", "AR"),
           clusters = GeneExpressionMatrix$hash_assignment)
```

## Multiome tutorial - MultiAssayExperiment



Then visualize the most differential TFs by clusters

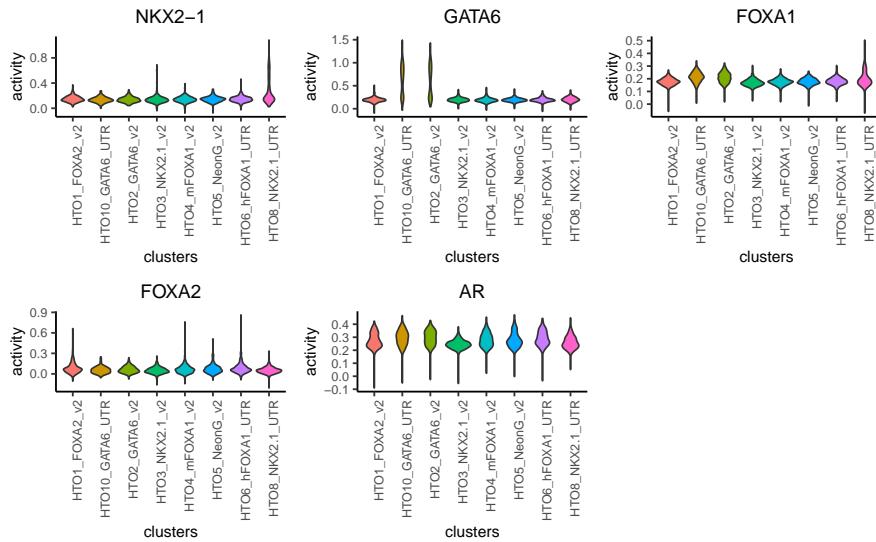
```
plotBubble(activity_matrix = score.combine,
           tf = markers.sig$tf,
           clusters = GeneExpressionMatrix$hash_assignment)
```



Visualize the known differential TFs by violin plot. Note there is no activity calculated for SOX2 because the expression of SOX2 is 0 in all cells.

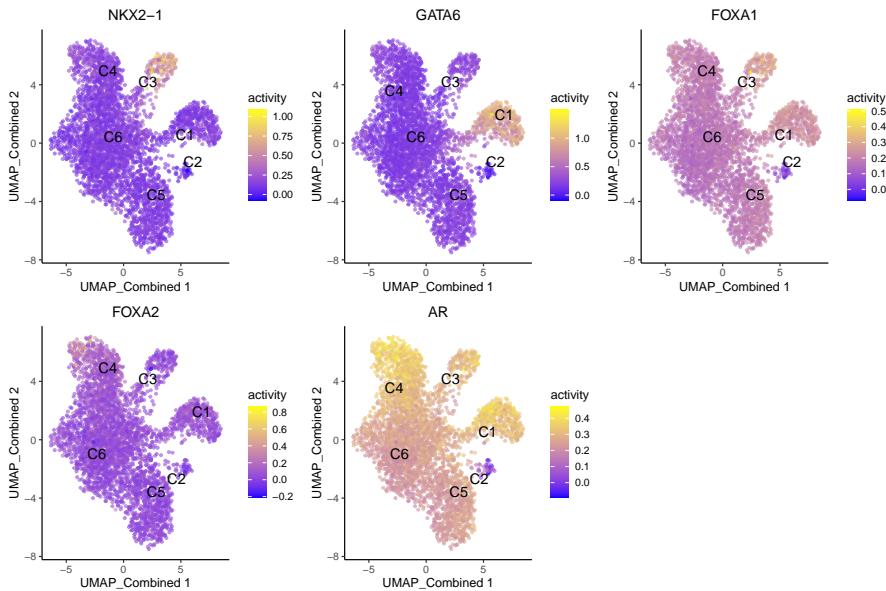
```
plotActivityViolin(activity_matrix = score.combine,
                   tf = c("NKX2-1", "GATA6", "FOXA1", "FOXA2", "AR"),
                   clusters = GeneExpressionMatrix$hash_assignment)
```

## Multiome tutorial - MultiAssayExperiment



Visualize the known differential TFs by UMAP

```
plotActivityDim(sce = GeneExpressionMatrix,
                activity_matrix = score.combine,
                tf = c("NKX2-1", "GATA6", "FOXA1", "FOXA2", "AR"),
                dimtype = "UMAP_Combined",
                label = "Clusters",
                point_size = 1,
                ncol = 3)
```

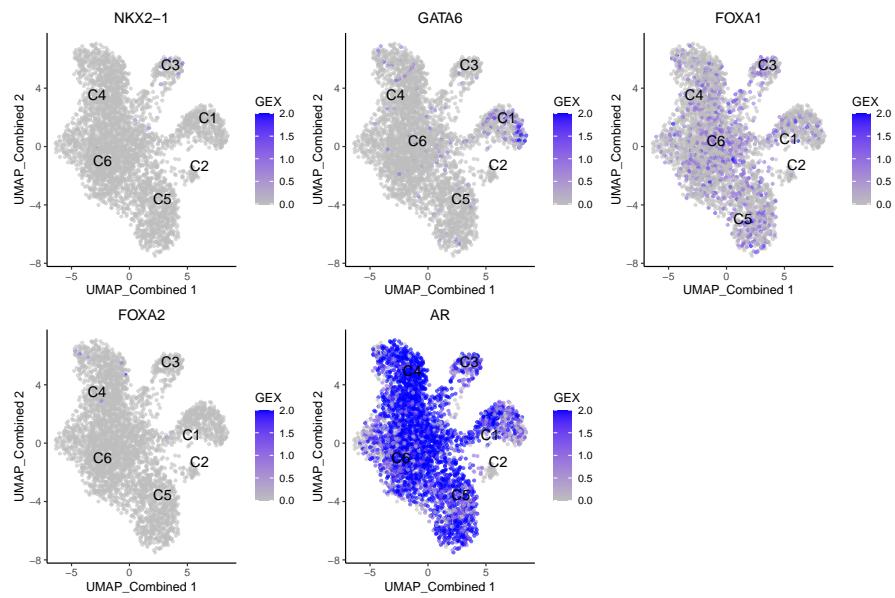


In contrast, the gene expression of the TFs is very sparse

```
plotActivityDim(sce = GeneExpressionMatrix,
                activity_matrix = counts(GeneExpressionMatrix),
```

## Multiome tutorial - MultiAssayExperiment

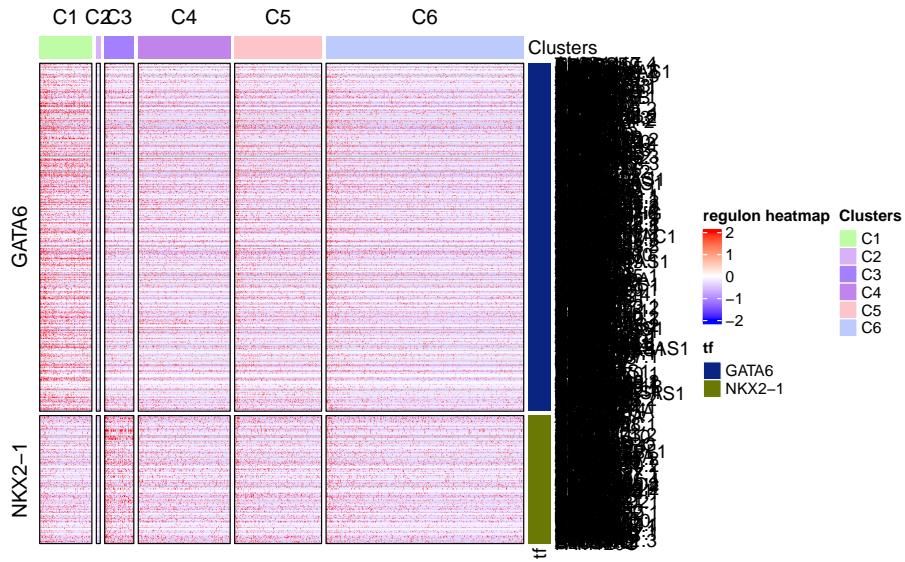
```
tf = c("NKX2-1", "GATA6", "FOXA1", "FOXA2", "AR"),
dimtype = "UMAP_Combined",
label = "Clusters",
point_size = 1,
ncol = 3,
limit = c(0,2),
colors = c("grey", "blue"),
legend.label = "GEX")
```



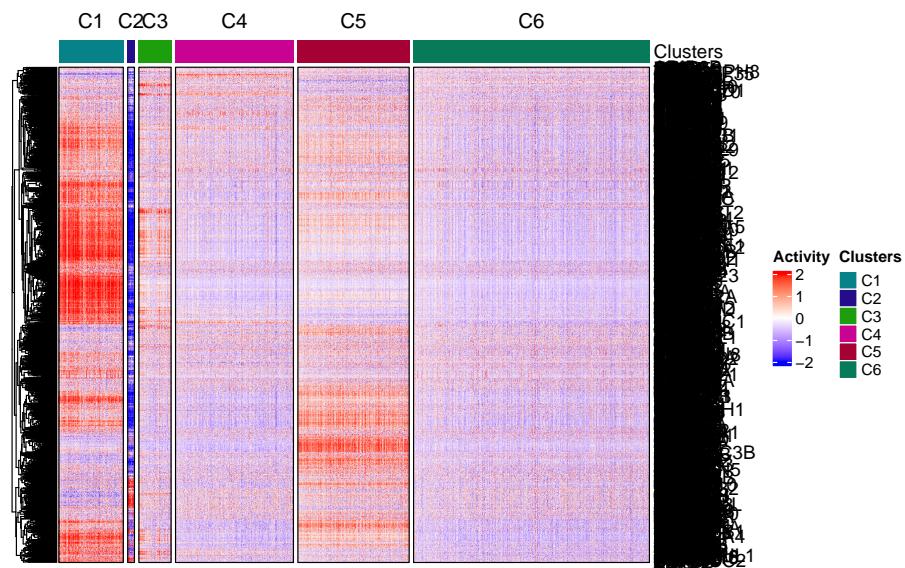
Visualize the gene expression of the regulons by heatmap

```
plotHeatmapRegulon(sce=GeneExpressionMatrix,
                     tfs=c("GATA6", "NKX2-1"),
                     regulon=regulon.w,
                     regulon_cutoff=0.1,
                     downsample=1000,
                     cell_attributes="Clusters",
                     col_gap="Clusters",
                     exprs_values="counts",
                     name="regulon heatmap")
```

## Multiome tutorial - MultiAssayExperiment



```
plotHeatmapActivity(activity=score.combine,
                     sce=GeneExpressionMatrix,
                     tfs=rownames(score.combine),
                     downsample=5000,
                     cell_attributes="Clusters",
                     col_gap="Clusters",
                     name = "Activity")
```



## 4.11 Geneset enrichment

Sometimes we are interested to know what pathways are enriched in the regulon of a particular TF. We can perform geneset enrichment using the enricher function from [clusterProfiler](#).

## Multiome tutorial - MultiAssayExperiment

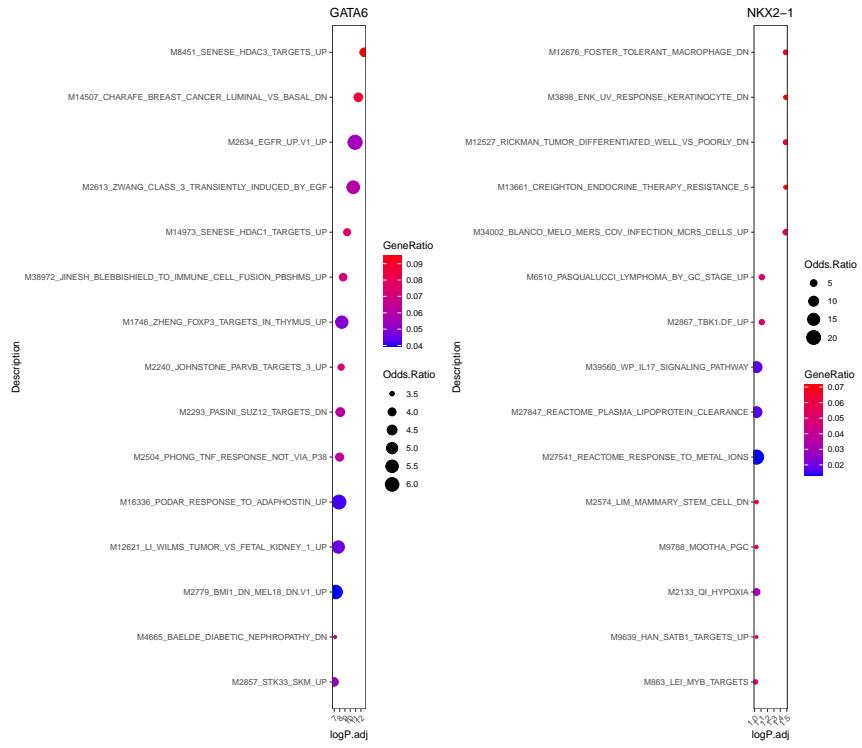
```
#retrieve genesets
H <- EnrichmentBrowser::getGenesets(org = "hsa",
                                      db = "msigdb",
                                      cat = "H",
                                      gene.id.type = "SYMBOL" )
#> Using cached version from 2023-06-04 00:42:26
C2 <- EnrichmentBrowser::getGenesets(org = "hsa",
                                      db = "msigdb",
                                      cat = "C2",
                                      gene.id.type = "SYMBOL" )
#> Using cached version from 2023-06-03 18:20:06
C6 <- EnrichmentBrowser::getGenesets(org = "hsa",
                                      db = "msigdb",
                                      cat = "C6",
                                      gene.id.type = "SYMBOL" )
#> Using cached version from 2023-06-04 00:42:30

#combine genesets and convert genesets to be compatible with enricher
gs <- c(H, C2, C6)
gs.list <- do.call(rbind,lapply(names(gs), function(x)
  {data.frame(gs=x, genes=gs[[x]])}))

enrichresults <- regulonEnrich(TF = c("GATA6", "NKX2-1"),
                                 regulon = regulon.w,
                                 weight = "weight",
                                 weight_cutoff = 0,
                                 genesets = gs.list)
#> GATA6
#> NKX2-1

#plot results
enrichPlot(results = enrichresults )
```

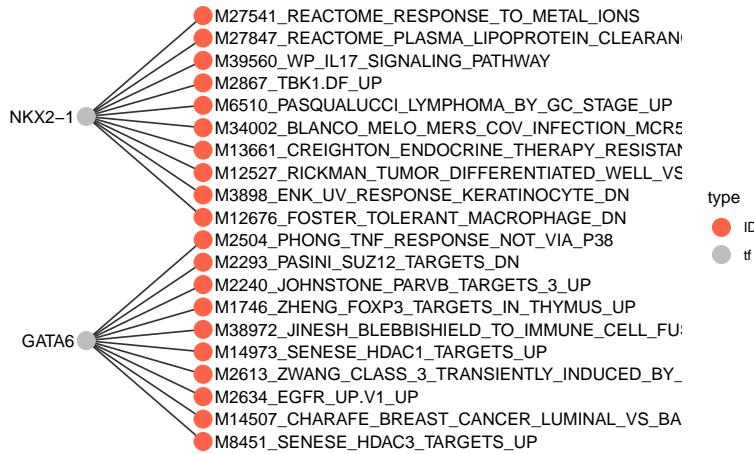
## Multiome tutorial - MultiAssayExperiment



## 4.12 Network analysis

We can visualize the genesets as a network

```
plotGseaNetwork(tf = names(enrichresults),
                enrichresults = enrichresults,
                p.adj_cutoff = 0.1,
                ntop_pathways = 10)
```

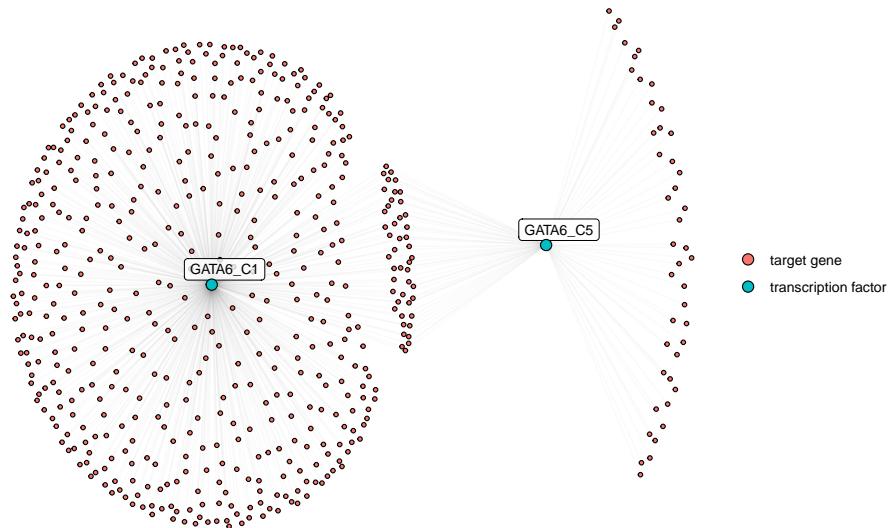


## 4.13 Differential networks

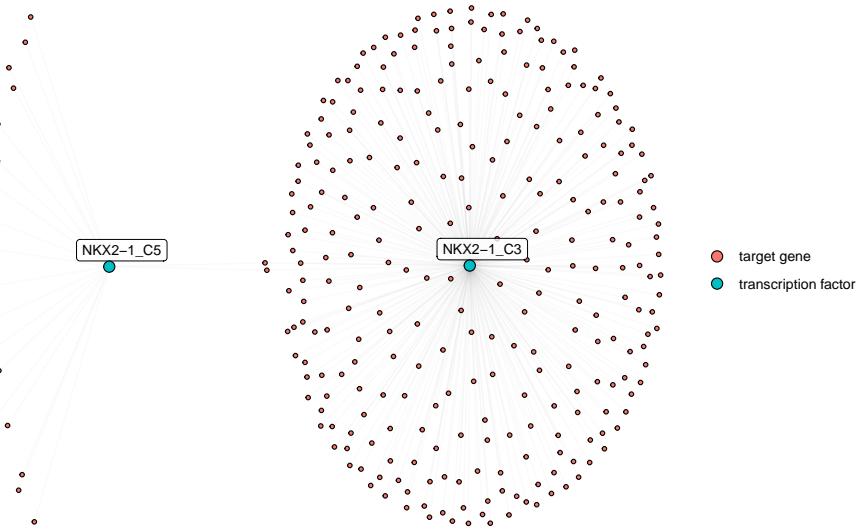
We are interested in understanding the differential networks between two conditions and determining which transcription factors account for the differences in the topology of networks. The pruned regulons with cluster-specific test statistics computed by `pruneRegulon` can be used to generate cluster-specific networks based on user-defined cutoffs and to visualize differential networks for transcription factors of interest. In this dataset, the GATA6 gene was only expressed in cluster 1 (C1) and NKX2-1 was only expressed in cluster 3 (C3). If we visualize the target genes of GATA6, we can see that C1 has many more target genes of GATA6 compared to C5, a cluster that does not express GATA6. Similarly, NKX2-1 target genes are confined to C3 which is the only cluster that exogenously expresses NKX2-1.

```
plotDiffNetwork(pruned.regulon,
               cutoff = 1,
               tf = c("GATA6"),
               weight = "stats",
               clusters = c("C1", "C5"),
               layout = "stress")
#> Building graph using weight as edge weights
```

## Multiome tutorial - MultiAssayExperiment



```
plotDiffNetwork(pruned.regulon,
                cutoff = 1,
                tf = c("NKX2-1"),
                weight = "stats",
                clusters = c("C3", "C5"),
                layout = "stress")
#> Building graph using weight as edge weights
```

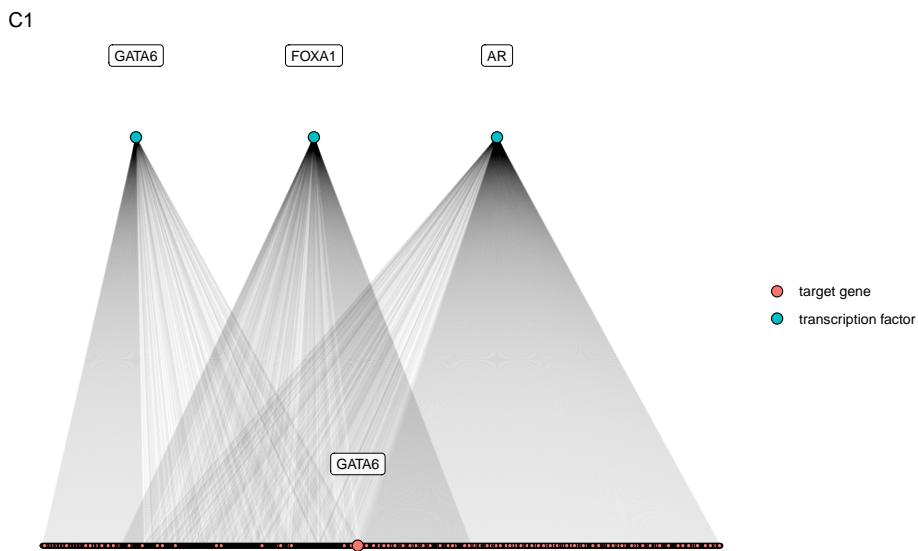


We can also visualize how transcription factors relate to other transcription factors in each cluster.

```
selected <- which(pruned.regulon$stats[, "C1"] > 1 &
                    pruned.regulon$tf %in% c("GATA6", "FOXA1", "AR"))
C1_network <- buildGraph(pruned.regulon[selected],)
```

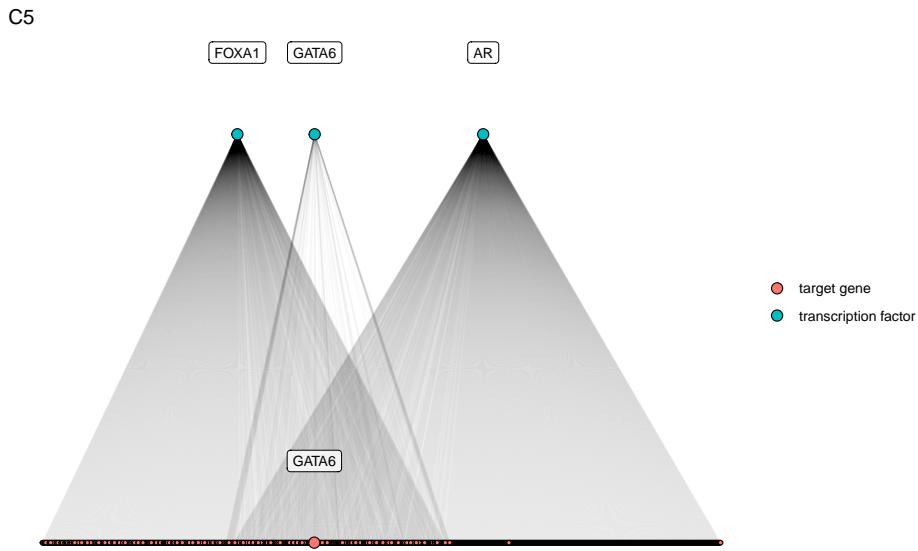
## Multiome tutorial - MultiAssayExperiment

```
weights = "stats",
cluster = "C1")
#> Building graph using stats as edge weights
selected <- which(pruned.regulon$stats[, "C5"] > 1 &
                     pruned.regulon$tf %in% c("GATA6", "FOXA1", "AR"))
C5_network <- buildGraph(pruned.regulon[selected, ],
                           weights = "stats",
                           cluster = "C5")
#> Building graph using stats as edge weights
plotEpiRegulonNetwork(C1_network,
                      layout = "sugiyama",
                      tfs_to_highlight = c("GATA6", "FOXA1", "AR")) +
ggplot2::ggtitle ("C1")
```



```
plotEpiRegulonNetwork(C5_network,
                      layout = "sugiyama",
                      tfs_to_highlight = c("GATA6", "FOXA1", "AR")) +
ggplot2::ggtitle ("C5")
```

## Multiome tutorial - MultiAssayExperiment

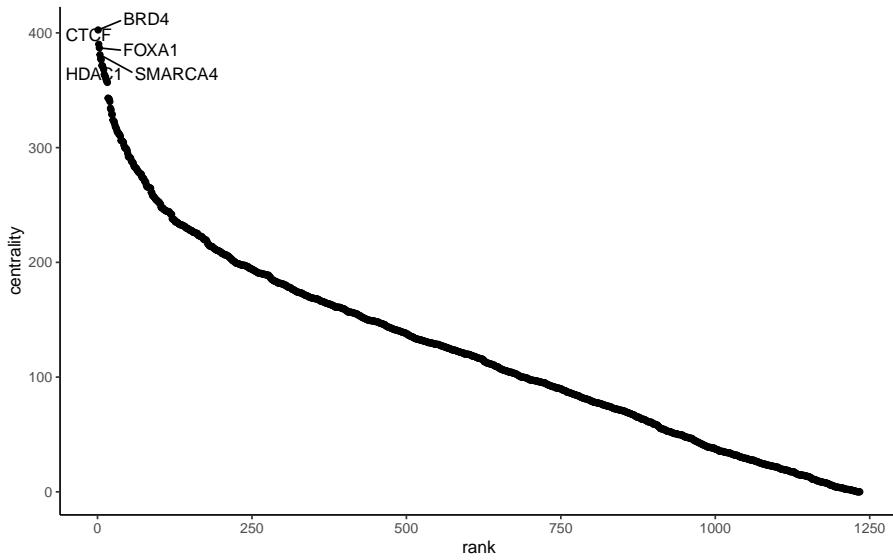


To systematically examine the differential network topology between two clusters, we perform an edge subtraction between two graphs, using weights computed by `pruneRegulon`. We then calculate the degree centrality of the weighted differential graphs and if desired, normalize the differential centrality against the total number of edges. The default normalization function is `sqrt` as it preserves both the difference in the number of edges (but scaled by `sqrt`) and the differences in the weights. If the user only wants to examine the differences in the averaged weights, the `FUN` argument can be changed to `identity`. Finally, we rank the transcription factors by (normalized) differential centrality.

```
# rank by differential centrality
C1_network <- buildGraph(pruned.regulon, weights = "stats", cluster="C1")
#> Replacement of na values for weights with 0
#> Building graph using stats as edge weights
C5_network <- buildGraph(pruned.regulon, weights = "stats", cluster="C5")
#> Replacement of na values for weights with 0
#> Building graph using stats as edge weights

diff_graph <- buildDiffGraph(C1_network, C5_network)
diff_graph <- addCentrality(diff_graph)
diff_graph <- normalizeCentrality(diff_graph)
rank_table <- rankTfs(diff_graph)

library(ggplot2)
ggplot(rank_table, aes(x = rank, y = centrality)) +
  geom_point() +
  ggrepel::geom_text_repel(data = head(rank_table, 5), aes(label = tf)) +
  theme_classic()
```



## 5 Session Info

```
sessionInfo()
#> R Under development (unstable) (2022-11-21 r83371)
#> Platform: x86_64-pc-linux-gnu (64-bit)
#> Running under: Ubuntu 18.04.6 LTS
#>
#> Matrix products: default
#> BLAS: /usr/local/lib/R/lib/libRblas.so
#> LAPACK: /usr/local/lib/R/lib/libRlapack.so
#>
#> 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
#>
#> time zone: Etc/UTC
#> tzcode source: system (glibc)
#>
#> attached base packages:
#> [1] stats4      stats       graphics    grDevices   utils       datasets    methods
#> [8] base
#>
#> other attached packages:
#> [1] BSgenome.Hsapiens.UCSC.hg38_1.4.5 BSgenome_1.67.4
#> [3] rtracklayer_1.59.1                  Biostrings_2.67.2
#> [5] XVector_0.39.0                     scMultiome_0.99.15
```

## Multome tutorial - MultiAssayExperiment

```
#> [7] MultiAssayExperiment_1.25.11      ExperimentHub_2.7.1
#> [9] AnnotationHub_3.7.4            BiocFileCache_2.7.2
#> [11] dbplyr_2.3.2                 org.Mm.eg.db_3.17.0
#> [13] AnnotationDbi_1.61.2        msigdbr_7.5.1
#> [15] epiregulon_1.0.25          scater_1.27.9
#> [17] ggplot2_3.4.2              scuttle_1.9.4
#> [19] scRNASeq_2.13.0           SingleCellExperiment_1.21.1
#> [21] SummarizedExperiment_1.29.1 Biobase_2.59.0
#> [23] GenomicRanges_1.51.4       GenomeInfoDb_1.35.17
#> [25] IRanges_2.33.1            S4Vectors_0.37.7
#> [27] BiocGenerics_0.45.3       MatrixGenerics_1.11.1
#> [29] matrixStats_0.63.0        dorothaea_1.11.0
#> [31] BiocStyle_2.27.2
#>
#> loaded via a namespace (and not attached):
#> [1] ProtGenerics_1.31.0          GSVA_1.47.3
#> [3] bitops_1.0-7                DirichletMultinomial_1.41.0
#> [5] enrichplot_1.19.2          TFBSTools_1.37.0
#> [7] HDO.db_0.99.1             httr_1.4.5
#> [9] RColorBrewer_1.1-3         doParallel_1.0.17
#> [11] Rgraphviz_2.43.0          tools_4.3.0
#> [13] backports_1.4.1          utf8_1.2.2
#> [15] R6_2.5.1                 HDF5Array_1.27.0
#> [17] lazyeval_0.2.2           uwot_0.1.14
#> [19] rhdf5filters_1.11.2       GetoptLong_1.0.5
#> [21] withr_2.5.0              prettyunits_1.1.1
#> [23] gridExtra_2.3            cli_3.6.1
#> [25] Cairo_1.6-0              scatterpie_0.1.9
#> [27] labeling_0.4.2           KEGGgraph_1.59.3
#> [29] readr_2.1.4              yulab.utils_0.0.6
#> [31] Rsamtools_2.15.3         gson_0.1.0
#> [33] R.utils_2.12.2           DOSE_3.25.0
#> [35] bcellViper_1.35.0        limma_3.55.10
#> [37] RSQLite_2.3.1            FNN_1.1.3.2
#> [39] gridGraphics_0.5-1        generics_0.1.3
#> [41] shape_1.4.6              BiocIO_1.9.2
#> [43] gtools_3.9.4             dplyr_1.1.2
#> [45] GO.db_3.17.0            Matrix_1.5-3
#> [47] ggbeeswarm_0.7.1         fansi_1.0.3
#> [49] R.methodsS3_1.8.2        lifecycle_1.0.3
#> [51] yaml_2.3.6               edgeR_3.41.9
#> [53] rhdf5_2.43.5            qvalue_2.31.1
#> [55] grid_4.3.0                blob_1.2.4
#> [57] promises_1.2.0.1         dqrng_0.3.0
#> [59] crayon_1.5.2            lattice_0.20-45
#> [61] beachmat_2.15.2          cowplot_1.1.1
#> [63] GenomicFeatures_1.51.4    annotate_1.77.0
#> [65] KEGGREST_1.39.0          magick_2.7.4
#> [67] pillar_1.9.0             knitr_1.42
#> [69] ComplexHeatmap_2.15.3   metapod_1.7.0
#> [71] fgsea_1.25.2            rjson_0.2.21
```

## Multome tutorial - MultiAssayExperiment

```
#> [73] codetools_0.2-18          fastmatch_1.1-3
#> [75] glue_1.6.2              ggfun_0.0.9
#> [77] downloader_0.4           data.table_1.14.8
#> [79] treeio_1.23.1           vctrs_0.6.2
#> [81] png_0.1-8               powerRlaw_0.70.6
#> [83] gtable_0.3.3            cachem_1.0.6
#> [85] xfun_0.39               mime_0.12
#> [87] tidygraph_1.2.3          pracma_2.4.2
#> [89] iterators_1.0.14         tinytex_0.42
#> [91] statmod_1.5.0           bluster_1.9.1
#> [93] interactiveDisplayBase_1.37.0 ellipsis_0.3.2
#> [95] nlme_3.1-162            ArchR_1.0.3
#> [97] ggtree_3.7.2            bit64_4.0.5
#> [99] progress_1.2.2          filelock_1.0.2
#> [101] irlba_2.3.5.1          viper_0.4.5
#> [103] seqLogo_1.65.0          colorspace_2.1-0
#> [105] DBI_1.1.3              tidyselect_1.2.0
#> [107] bit_4.0.5              compiler_4.3.0
#> [109] curl_4.3.3             graph_1.77.3
#> [111] BiocNeighbors_1.17.1    BSgenome.Mmusculus.UCSC.mm10_1.4.3
#> [113] xml2_1.3.3              DelayedArray_0.25.0
#> [115] shadowtext_0.1.2        bookdown_0.33
#> [117] caTools_1.18.2          checkmate_2.1.0
#> [119] scales_1.2.1            rappdirs_0.3.3
#> [121] stringr_1.5.0           digest_0.6.30
#> [123] motifmatchr_1.21.0      rmarkdown_2.21
#> [125] BSgenome.Hsapiens.UCSC.hg19_1.4.3 htmltools_0.5.5
#> [127] pkgconfig_2.0.3          sparseMatrixStats_1.11.1
#> [129] fastmap_1.1.0           ensemblldb_2.23.2
#> [131] rlang_1.1.0              GlobalOptions_0.1.2
#> [133] shiny_1.7.4              DelayedMatrixStats_1.21.0
#> [135] farver_2.1.1            jsonlite_1.8.4
#> [137] BiocParallel_1.33.12     R.oo_1.25.0
#> [139] GOSemSim_2.25.0          BiocSingular_1.15.0
#> [141] RCurl_1.98-1.12         magrittr_2.0.3
#> [143] ggplotify_0.1.0          GenomeInfoDbData_1.2.10
#> [145] patchwork_1.1.2          Rhdf5lib_1.21.1
#> [147] munsell_0.5.0            Rcpp_1.0.10
#> [149] ape_5.7-1               babelgene_22.9
#> [151] viridis_0.6.2            EnrichmentBrowser_2.29.2
#> [153] stringi_1.7.8            ggraph_2.1.0
#> [155] MASS_7.3-58.1            zlibbioc_1.45.0
#> [157] plyr_1.8.8               parallel_4.3.0
#> [159] ggrepel_0.9.3            CNEr_1.35.0
#> [161] graphlayouts_0.8.4         splines_4.3.0
#> [163] hms_1.1.3                circlize_0.4.15
#> [165] locfit_1.5-9.7           igraph_1.4.2
#> [167] reshape2_1.4.4            biomaRt_2.55.4
#> [169] ScaledMatrix_1.7.1       TFMPvalue_0.0.9
#> [171] BiocVersion_3.17.1       XML_3.99-0.14
#> [173] evaluate_0.20             scran_1.27.3
```

## Multiome tutorial - MultiAssayExperiment

```
#> [175] BiocManager_1.30.20          tzdb_0.3.0
#> [177] tweenr_2.0.2                foreach_1.5.2
#> [179] httpuv_1.6.9               polyclip_1.10-4
#> [181] tidyrr_1.3.0                purrr_1.0.1
#> [183] clue_0.3-64                ggforce_0.4.1
#> [185] rsvd_1.0.5                 xtable_1.8-4
#> [187] restfulr_0.0.15            AnnotationFilter_1.23.0
#> [189] tidytree_0.4.2              later_1.3.0
#> [191] viridisLite_0.4.1           tibble_3.2.1
#> [193] aplot_0.1.10                clusterProfiler_4.7.1
#> [195] memoise_2.0.1               beeswarm_0.4.0
#> [197] GenomicAlignments_1.35.1   cluster_2.1.4
#> [199] GSEABase_1.61.2
```