Hematopoeisis tutorial - MAE

Xiaosai Yao

25 July 2023

Package

epiregulon 1.0.27

Contents

1 Introduction

In this vignette, we used a dataset from the ArchR tutorial. Prior to using epiregulon, this dataset has been fully preprocessed in ArchR, and converted to a `MultiAssayExperiment` using `epireglon::archr2MAE`. The MAE object was uploaded to `scMultiome` for full reproducibility. In this dataset, scRNAseq and scATACseq were unpaired and integrated by the `ArchR::addGeneIntegrationMatrix` function.

2 Installation

Epiregulon is currently available on R/dev

```
#library(epiregulon)
devtools::load_all("/gstore/project/lineage/xiaosai/epiregulon")
## Warning: replacing previous import 'GenomicRanges::union' by 'igraph::union'
## when loading 'epiregulon'
```

If you would like to install from gitlab,

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

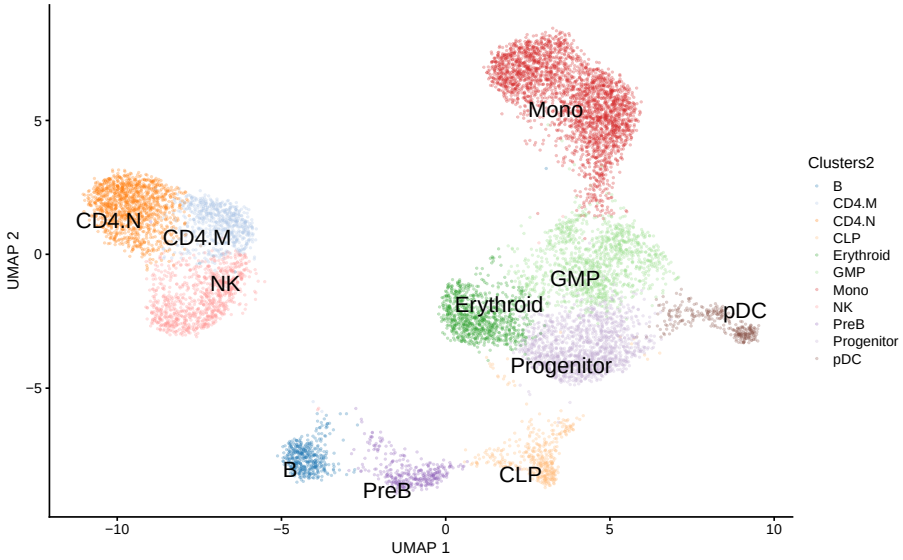
3 Data preparation

Download the example dataset from *scMultiome* package

```
mae <- scMultiome::hematopoiesis()
## snapshotDate(): 2023-07-24
## see ?scMultiome and browseVignettes('scMultiome') for documentation
## loading from cache

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

# Load expression matrix
GeneExpressionMatrix <- mae[["GeneIntegrationMatrix"]]

# Add gene symbols to rownames
rownames(GeneExpressionMatrix) <- rowData(GeneExpressionMatrix)$name

# Transfer dimensionality reduction matrix to GeneExpression
reducedDim(GeneExpressionMatrix, "IterativeLSI") <-
  reducedDim(mae[['TileMatrix500']], "IterativeLSI")
reducedDim(GeneExpressionMatrix, "UMAP") <-
  reducedDim(mae[['TileMatrix500']], "UMAP")
```

Visualize the data

```
scater::plotReducedDim(GeneExpressionMatrix,
                       dimred = "UMAP",
                       text_by = "Clusters2",
                       colour_by = "Clusters2",
                       point_size = 0.3,
                       point_alpha = 0.3)
```



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. Currently, human genomes hg19 and hg38 and mouse genome mm10 are available

```
grl <- getTFMotifInfo(genome = "hg19")
## snapshotDate(): 2023-07-24
## see ?scMultiome and browseVignettes('scMultiome') for documentation
## loading from cache
head(grl)
## GRangesList object of length 6:
## $`5-hmC`
## GRanges object with 22860 ranges and 0 metadata columns:
##           seqnames        ranges strand
##              <Rle>     <IRanges>  <Rle>
##       [1]     chr1   10001-10685      *
##       [2]     chr1   13362-13694      *
##       [3]     chr1   29631-29989      *
##       [4]     chr1   40454-40754      *
##       [5]     chr1 135395-135871      *
```

```
##       ...      ...           ...    ...
##   [22856]     chrM   15303-15326      *
##   [22857]     chrM   15328-16172      *
##   [22858]     chrM   16174-16183      *
##   [22859]     chrM   16186-16224      *
##   [22860]     chrM   16226-16492      *
##   -------
##   seqinfo: 25 sequences from an unspecified genome; no seqlengths
##
## ...
## <5 more elements>
```

4.2 Link ATACseq peaks to target genes

Next, we compute peak to gene correlations using a custom algorithm that has similar performance to ArchR's P2G function. Wherever possible, use a multidimensional dimensionality reduction matrix such as LSI or PCA instead of UMAP or TSNE since the former provides a more accurate estimate of cell similarity.

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

p2g
## DataFrame with 23711 rows and 8 columns
##         idxATAC         chr     start       end    idxRNA    target Correlation
##       <integer> <character> <integer> <integer> <integer>   <array>    <matrix>
## 1             7        chr1    801002    801502         2 LINC00115    0.864244
## 2             8        chr1    805039    805539         6    KLHL17    0.625471
## 3             9        chr1    845326    845826        10      AGRN    0.545993
## 4            10        chr1    846428    846928        10      AGRN    0.646209
## 5            13        chr1    856263    856763        10      AGRN    0.549411
## ...         ...         ...       ...       ...       ...       ...         ...
## 23707    146403       chr22  51021154  51021654     12089      ARSA    0.655996
## 23708    146412       chr22  51110826  51111326     12090    SHANK3    0.560404
## 23709    146417       chr22  51143606  51144106     12090    SHANK3    0.500026
## 23710    146421       chr22  51213512  51214012     12090    SHANK3    0.504567
## 23711    146421       chr22  51213512  51214012     12091       ACR    0.557113
##        distance
##       <integer>
## 1         36099
## 2         88427
## 3        107676
## 4        106574
```

```
## 5         96739
## ...         ...
## 23707     44747
## 23708         0
## 23709     30336
## 23710    100242
## 23711     36660
```

4.3 Add TF motif binding to peaks

The next step is to add the TF motif binding information by overlapping the regions of the peak matrix with the bulk chip-seq database.

```
overlap <- addTFMotifInfo(grl = grl, p2g = p2g, peakMatrix = PeakMatrix)
## Computing overlap...
## Success!
head(overlap)
##      idxATAC idxTF      tf
## 1018       7    35    ARNT
## 1019       7    50    ATF2
## 1020       7    55    ATF7
## 1021       7    76    BCL6
## 1022       7    80    BCOR
## 1023       7    82 BHLHE40
```

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, overlap, aggregate=FALSE)
head(regulon)
## DataFrame with 6 rows and 11 columns
##     idxATAC         chr     start       end    idxRNA      target       all
##   <integer> <character> <integer> <integer> <integer> <character> <numeric>
## 1         7        chr1    801002    801502         2   LINC00115  0.864244
## 2         7        chr1    801002    801502         2   LINC00115  0.864244
## 3         7        chr1    801002    801502         2   LINC00115  0.864244
## 4         7        chr1    801002    801502         2   LINC00115  0.864244
## 5         7        chr1    801002    801502         2   LINC00115  0.864244
## 6         7        chr1    801002    801502         2   LINC00115  0.864244
##    distance     idxTF          tf     corr
##   <integer> <integer> <character> <matrix>
## 1     36099        35        ARNT
## 2     36099        50        ATF2
```

```
## 3     36099        55        ATF7
## 4     36099        76        BCL6
## 5     36099        80        BCOR
## 6     36099        82     BHLHE40
```

4.5 Prune network

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(TF, RE) * P(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(TF, RE) * P(TG) and the probability otherwise is 1- P(TF, RE) * P(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_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 = "counts",
                               peakMatrix = PeakMatrix,
                               peak_assay = "counts",
                               regulon = regulon,
                               prune_value = "pval",
                               regulon_cutoff = 0.05,
                               clusters = GeneExpressionMatrix$Clusters2,
                               exp_cutoff = NULL)
## pruning network with chi.sq tests using a regulon cutoff of pval<0.05
## pruning regulons
```

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 grouped pseudobulks by

user-supplied cluster labels and yield a single weight across all clusters per each TF-RE-target triplet. In contrast, Wilcoxon and log fold change methods group cells based on the joint expression of TF, RE and TG in each single cell or in cell aggregates. Cell aggregation uses a default value of 10 cells and can help overcome sparsity and speed up computation. If cluster labels are provided, we can obtain weights of individual clusters and all cells combined. In this example, we apply Wilcoxon test on cell aggregates of 10 cells.

```
regulon.w <- addWeights(regulon = pruned.regulon,
                        expMatrix  = GeneExpressionMatrix,
                        exp_assay  = "counts",
                        peakMatrix = PeakMatrix,
                        peak_assay = "counts",
                        clusters = GeneExpressionMatrix$Clusters2,
                        aggregateCells = TRUE,
                        method = "wilcox",
                        useDim = "IterativeLSI")
## adding weights using wilcoxon...
## performing pseudobulk using an average of 10 cells

regulon.w
## DataFrame with 2388059 rows and 15 columns
##           idxATAC         chr     start       end    idxRNA      target
##         <integer> <character> <integer> <integer> <integer> <character>
## 1             463        chr1   3709928   3710428        66       SMIM1
## 2             732        chr1   8021367   8021867        95        UTS2
## 3             733        chr1   8021996   8022496        95        UTS2
## 4             828        chr1   8907982   8908482       100        RERE
## 5             891        chr1   9223922   9224422       107        H6PD
## ...           ...         ...       ...       ...       ...         ...
## 2388055    145987       chr22  46646470  46646970     12045       PPARA
## 2388056    146314       chr22  50732210  50732710     12066       PANX2
## 2388057    146374       chr22  50963929  50964429     12078        LMF2
## 2388058    146403       chr22  51021154  51021654     12078        LMF2
## 2388059    146403       chr22  51021154  51021654     12089        ARSA
##                all  distance     idxTF          tf     corr
##         <numeric> <integer> <integer> <character> <matrix>
## 1        0.550803     20394         5        ADNP
## 2        0.611979     46072         5        ADNP
## 3        0.543859     46701         5        ADNP
## 4        0.594452     28282         5        ADNP
## 5        0.584584     68440         5        ADNP
## ...           ...       ...       ...         ...      ...
## 2388055  0.547093     99771      1557        ZXDC
## 2388056  0.566345    122850      1557        ZXDC
## 2388057  0.712879     15793      1557        ZXDC
## 2388058  0.549414     73018      1557        ZXDC
## 2388059  0.655996     44747      1557        ZXDC
##                                        pval
##                                    <matrix>
## 1       2.88106e-04:        NaN:0.4032360:...
## 2       2.71837e-29:3.79230e-06:0.0117417:...
```

```
## 3       4.45015e-06:6.06719e-03:0.2144717:...
## 4       1.19087e-02:3.13907e-01:      NaN:...
## 5       1.25791e-05:        NaN:      NaN:...
## ...                                       ...
## 2388055     0.242104230:0.800758:0.304168:...
## 2388056     0.044675386:0.884161:     NaN:...
## 2388057     0.833286943:0.390888:0.725549:...
## 2388058     0.000372059:0.308149:0.868956:...
## 2388059     0.026174461:0.017739:0.698380:...
##                                     stats                    qval
##                                   <matrix>                <matrix>
## 1           13.14618:     NaN:0.698655:... 1.00000e+00:NaN:  1:...
## 2          126.24453:21.36706:6.349467:... 8.46867e-23:  1:  1:...
## 3           21.06047: 7.53024:1.540982:... 1.00000e+00:  1:  1:...
## 4            6.32441: 1.01417:     NaN:... 1.00000e+00:  1:NaN:...
## 5           19.07332:     NaN:     NaN:... 1.00000e+00:NaN:NaN:...
## ...                                     ...                      ...
## 2388055  1.3682997:0.0636888:1.0558312:...              1:1:  1:...
## 2388056  4.0308515:0.0212275:      NaN:...              1:1:NaN:...
## 2388057  0.0443051:0.7361788:0.1232378:...              1:1:  1:...
## 2388058 12.6675377:1.0385922:0.0272198:...              1:1:  1:...
## 2388059  4.9444569:5.6217414:0.1501623:...              1:1:  1:...
##                                   weight
##                                 <matrix>
## 1         0.0961176: 0.2100846:0.109080:...
## 2         0.1482138: 0.0403265:0.399392:...
## 3         0.1273540: 0.1511035:0.261575:...
## 4         0.2610927: 0.1209382:0.000000:...
## 5         0.3200770:-0.2609719:0.218160:...
## ...                                     ...
## 2388055 0.0395375:-0.1218877: 0.2181602:...
## 2388056 0.1320526:-0.0888343:-0.0165295:...
## 2388057 0.1071509: 0.1843265: 0.2748304:...
## 2388058 0.1280652: 0.2734060: 0.2748304:...
## 2388059 0.1225161:-0.2815069: 0.2644594:...

saveRDS(regulon, "/gstore/project/lineage/manuscript/epiregulon/OUTPUT/regulon.rds")
saveRDS(regulon.w, "/gstore/project/lineage/manuscript/epiregulon/OUTPUT/regulon.w.rds")
```

4.7 Calculate TF activity

Finally, the activities for a specific TF in each cell are computed by averaging the weighted expressions of target genes linked to the TF weighted.

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

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 $\{1,2,\ldots,n\}$ $weight_i$ is the weight of TF and target i

```
score.combine <- calculateActivity(expMatrix = GeneExpressionMatrix,
                                   regulon = regulon.w,
                                   mode = "weight",
                                   method = "weightedMean",
                                   exp_assay = "counts")
## calculating TF activity from regulon using weightedmean
## Warning in calculateActivity(expMatrix = GeneExpressionMatrix, regulon =
## regulon.w, : The weight column contains multiple subcolumns but no cluster
## information was provided. Using first column to compute activity...
## aggregating regulons...
## creating weight matrix...
## calculating activity scores...
## normalize by the number of targets...
head(score.combine[1:5,1:5])
## 5 x 5 sparse Matrix of class "dgCMatrix"
##       scATAC_BMMC_R1#TTATGTCAGTGATTAG-1 scATAC_BMMC_R1#AAGATAGTCACCGCGA-1
## ADNP                         0.2686148                         0.2784902
## AEBP2                        0.1205601                         0.3048003
## AFF1                         0.3440949                         0.3945086
## AFF4                         0.3121704                         0.4393103
## AGO1                         0.2467518                         0.3474437
##       scATAC_BMMC_R1#GCATTGAAGATTCCGT-1 scATAC_BMMC_R1#TATGTTCAGGGTTCCC-1
## ADNP                         0.2116217                         0.2529460
## AEBP2                        0.1088684                         0.1509700
## AFF1                         0.3767829                         0.3679830
## AFF4                         0.2683644                         0.3419842
## AGO1                         0.1698227                         0.2694617
##       scATAC_BMMC_R1#AGTTACGAGAACGTCG-1
## ADNP                         0.2958698
## AEBP2                        0.1374880
## AFF1                         0.3319492
## AFF4                         0.3181495
## AGO1                         0.2588457
```

4.8 Differential TF activity test

We can next determine which TFs exhibit differential activities across cell clusters/groups via the `findDifferentialActivity` function. This function depends on `findMarkers` function from *scran* package.

```
markers  <- findDifferentialActivity(activity_matrix = score.combine,
                                     groups = GeneExpressionMatrix$Clusters2,
                                     pval.type = "some",
                                     direction = "up",
                                     test.type = "t")
```

getSigGenes compiles the different test results into a single dataframe and enables user to supply their desired cutoffs for significance and variable to order by.

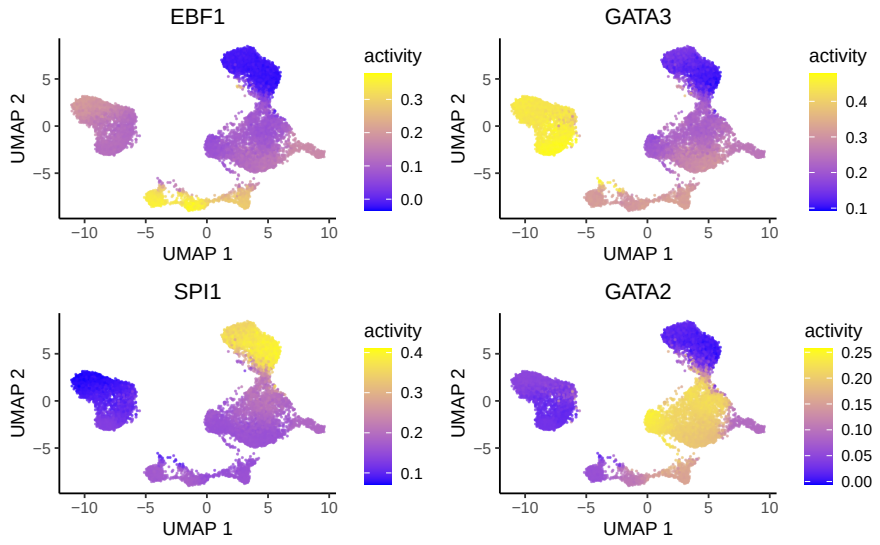
```
markers.sig <- getSigGenes(markers, topgenes = 3 )
## Using a logFC cutoff of 0.1 for class B
## Using a logFC cutoff of 0.3 for class CD4.M
## Using a logFC cutoff of 0.2 for class CD4.N
## Using a logFC cutoff of 0.1 for class CLP
## Using a logFC cutoff of 0.1 for class Erythroid
## Using a logFC cutoff of 0.1 for class GMP
## Using a logFC cutoff of 0.4 for class Mono
## Using a logFC cutoff of 0.3 for class NK
## Using a logFC cutoff of 0.1 for class PreB
## Using a logFC cutoff of 0.1 for class Progenitor
## Using a logFC cutoff of 0.1 for class pDC
```

4.9 Visualizing TF activities

Epiregulon also provides multiple options for visualizing the inferred TF activities by reduced dimensional space

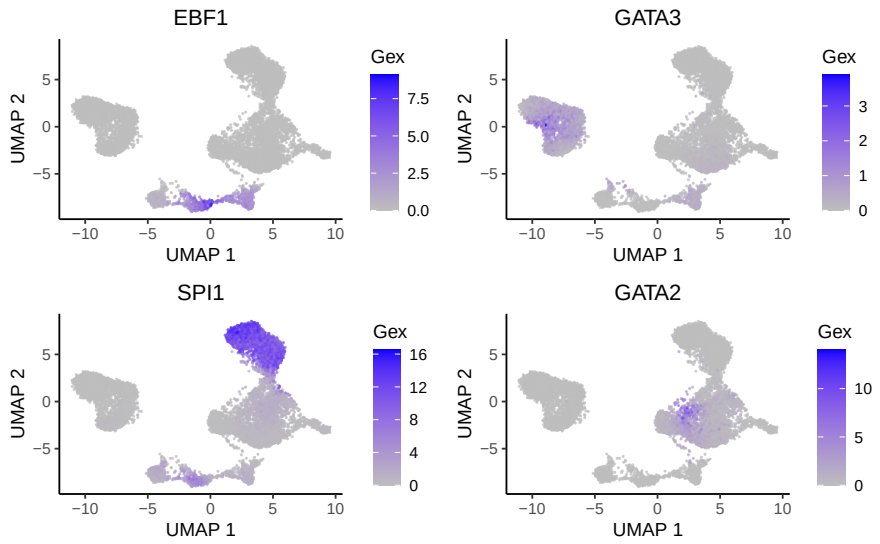
tSNE or UMAP plots:

```
tfs_interest <- c("EBF1","GATA3", "SPI1", "GATA2")
plotActivityDim(sce = GeneExpressionMatrix,
                activity_matrix = score.combine[tfs_interest,],
                tf = tfs_interest,
                dimtype = "UMAP",
                nrow=2,
                ncol=2,
                point_size=0.1,
                rasterise = TRUE)
```



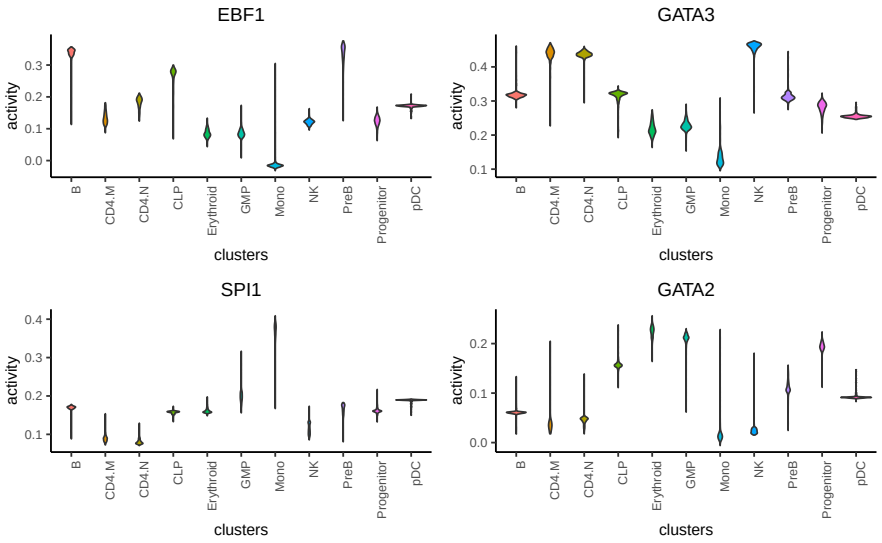
We can compare the activity with gene expression of the same TFs.

```
#options(ggrastr.default.dpi=300)
plotActivityDim(sce = GeneExpressionMatrix,
                activity_matrix = counts(GeneExpressionMatrix)[tfs_interest,],
                tf = tfs_interest,
                dimtype = "UMAP",
                nrow=2,
                ncol=2,
                legend.label = "Gex",
                colors = c("grey","blue"),
                #limit = c(0,2),
                point_size=0.1,
                rasterise = TRUE)
```



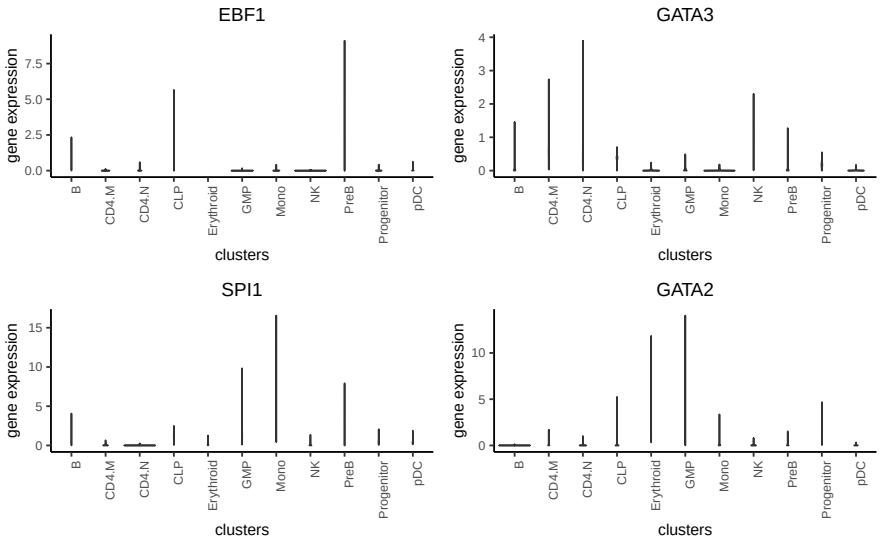
We can also plot violin plot to visualize TF activity.

```
plotActivityViolin(activity_matrix = score.combine,
                   tf = tfs_interest,
                   clusters = GeneExpressionMatrix$Clusters2,
                   nrow=2,
                   ncol=2)
```



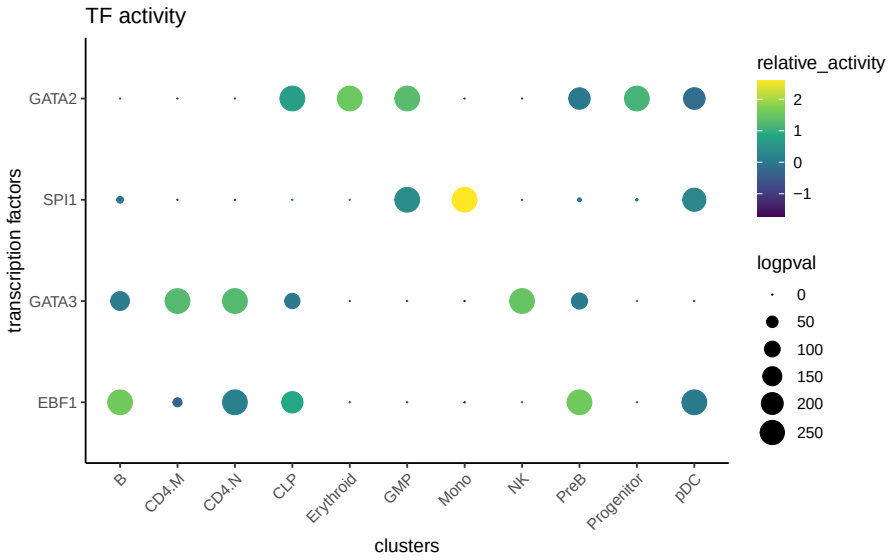
We plot violin plot to visualize TF gene expression.

```
plotActivityViolin(activity_matrix = counts(GeneExpressionMatrix)[tfs_interest,],
                   tf = tfs_interest,
                   clusters = GeneExpressionMatrix$Clusters2,
                   nrow=2,
                   ncol=2,
                   legend.label = "gene expression")
```



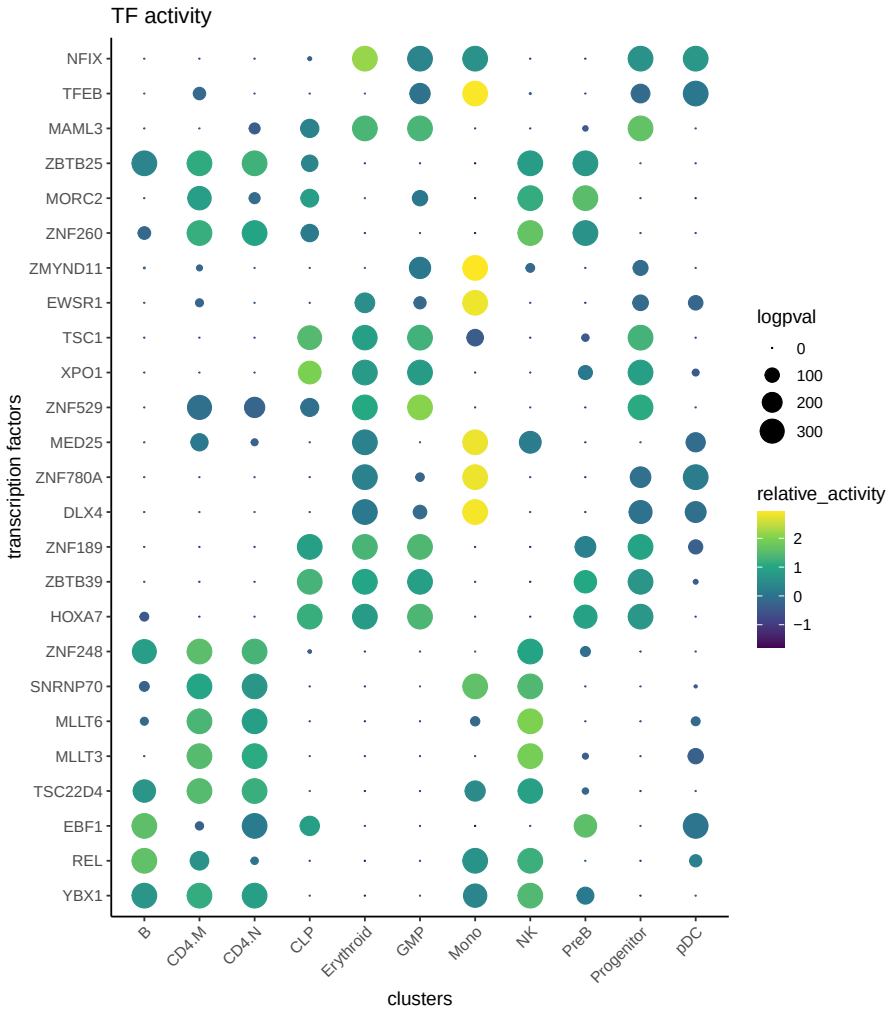
We can visualize the different TFs in a bubble plot:

```
plotBubble(activity_matrix = score.combine,
           tf = tfs_interest,
           GeneExpressionMatrix$Clusters2,
           bubblesize = "FDR")
```



We visualize the top differential TFs based on activity.

```
plotBubble(activity_matrix = score.combine,
           tf = markers.sig$tf,
           GeneExpressionMatrix$Clusters2,
           bubblesize = "FDR")
```



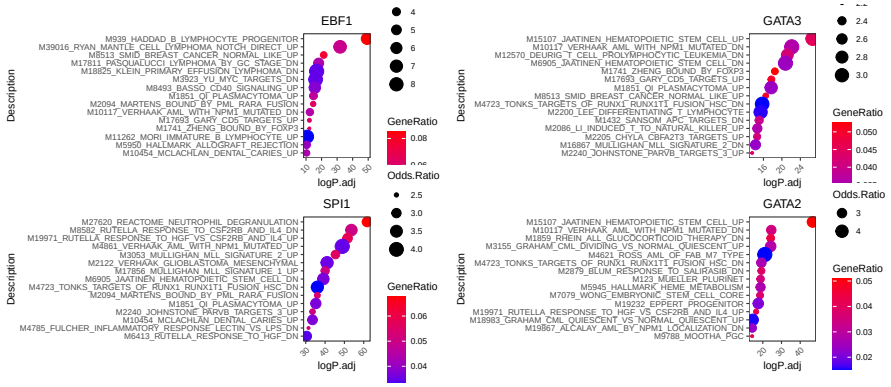
4.10 Geneset enrichment

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.

Here we first download Hallmark and C2 signatures from hallmark and then perform gene set enrichment of the known lineage factors. As expected, EBF1 is consistent with a B cell lineage factor, GATA3 and RUNX3 with lymphoid lineage and SPI1 with myeloid lineage.

```
#retrieve genesets
H <- EnrichmentBrowser::getGenesets(org = "hsa",
                                    db = "msigdb",
                                    cat = "H",
                                    gene.id.type = "SYMBOL" )
## Using cached version from 2023-07-20 20:46:00
C2 <- EnrichmentBrowser::getGenesets(org = "hsa",
                                     db = "msigdb",
                                     cat = "C2",
```

```
                                        gene.id.type = "SYMBOL" )
## Using cached version from 2023-07-22 00:04:45


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

enrichresults <- regulonEnrich(TF = tfs_interest,
                               regulon = regulon.w,
                               weight = "weight",
                               weight_cutoff = 0,
                               genesets = gs.list)
## EBF1
## GATA3
## SPI1
## GATA2

#plot results
enrichPlot(results = enrichresults, ncol=2)
```

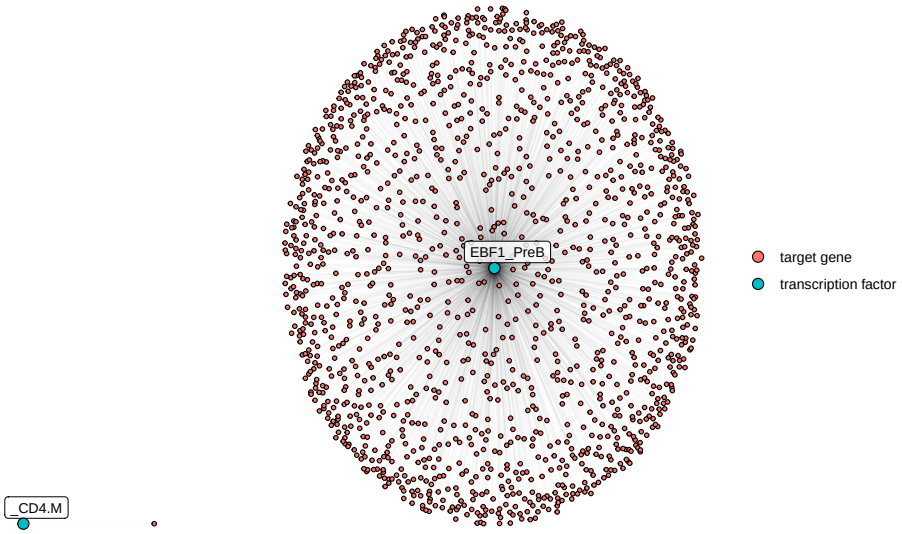


5 Differential Network analysis

In addition to looking at the summed TF activity, a second approach to investigate differential TF activity is to compare and contrast target genes or network topology. In this example, we know that EBF1 is a B cell lineage factor. If we plot the differential network of EBF1 using the regulon with cluster-specific weights, we can see that EBF1 has many more targets in PreB cells than it has in CD4 memory cells.

```
plotDiffNetwork(regulon.w,
                cutoff = 0,
                tf = c("EBF1"),
                weight = "weight",
                clusters = c("PreB","CD4.M"),
                layout = "stress")
```

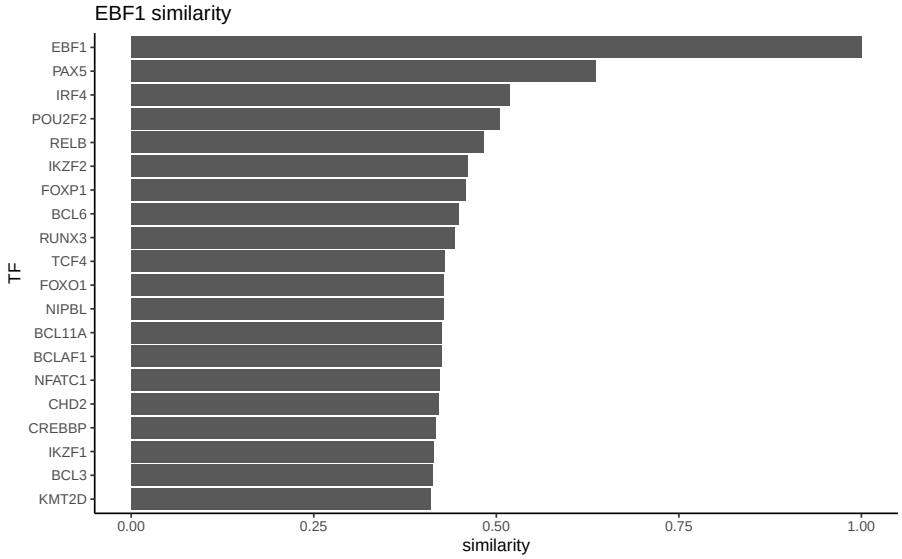
```
## Replacement of na values for weights with 0
## Building graph using weight as edge weights
```



Sometimes, we are interested to identify interaction partners of the TFs of interest. This can be achieved by comparing the overlap of the targets genes for all the TFs and identify the most similar TFs by Jaccard similarity. To illustrate this function, we take a look at the top most similar 20 TFs to EBF1, and we successfully identify PAX5 as the most similar TF. Both PAX5 and EBF1 are important factors for B cell development (https://www.nature.com/articles/ni.2641).

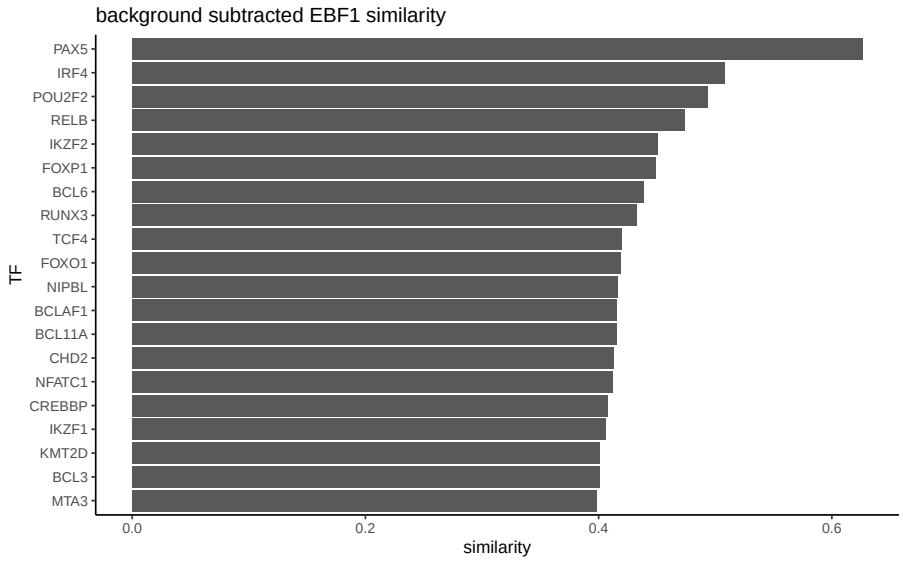
```
# construct a graph of the preB cells
preB_network <- buildGraph(regulon.w, weights = "weight", cluster="PreB")
## Building graph using weight as edge weights

# compute a similarity matrix of all TFs
similarity_score <- calculateJaccardSimilarity(preB_network)

# Focus on EBF1
similarity_score_EBF1 <- similarity_score[, "EBF1"]
similarity_df <- data.frame(similarity = head(sort(similarity_score_EBF1,
                                                   decreasing = TRUE),20),
                            TF = names(head(sort(similarity_score_EBF1,
                                                 decreasing = TRUE),20)))

similarity_df$TF <- factor(similarity_df$TF, levels = rev(unique(similarity_df$TF)))

# plot top TFs most similar to EBF1
topTFplot <- ggplot(similarity_df, aes(x=TF, y=similarity)) +
  geom_bar(stat="identity") +
  coord_flip() +
  ggtitle("EBF1 similarity") +
  theme_classic()
```

```
print(topTFplot)
```



In order to convince ourselves that our differential network is statistically significant, we permute the edges and obtain a background graph from averaging many iterations. Here, we plot the differential network graph subtracted by permuted graphs.

```
# create a permuted graph by rewiring the edges 100 times
permute_matrix <- permuteGraph(preB_network, "EBF1", 100, p=1)
permute_matrix <- permute_matrix[names(similarity_score_EBF1),]
diff_matrix <- similarity_score_EBF1-rowMeans(permute_matrix)

diff_matrix_df <- data.frame(similarity = head(sort(diff_matrix,
                                                    decreasing = TRUE),20),
                             TF = names(head(sort(diff_matrix,
                                                  decreasing = TRUE),20)))

diff_matrix_df$TF <- factor(diff_matrix_df$TF, levels = rev(unique(diff_matrix_df$TF)))

# plot top TFs most similar to EBF1
topTFplot <- ggplot(diff_matrix_df, aes(x=TF, y=similarity)) +
            geom_bar(stat="identity") +
            coord_flip() +
            ggtitle("background subtracted EBF1 similarity ") +
            theme_classic()
print(topTFplot)
```

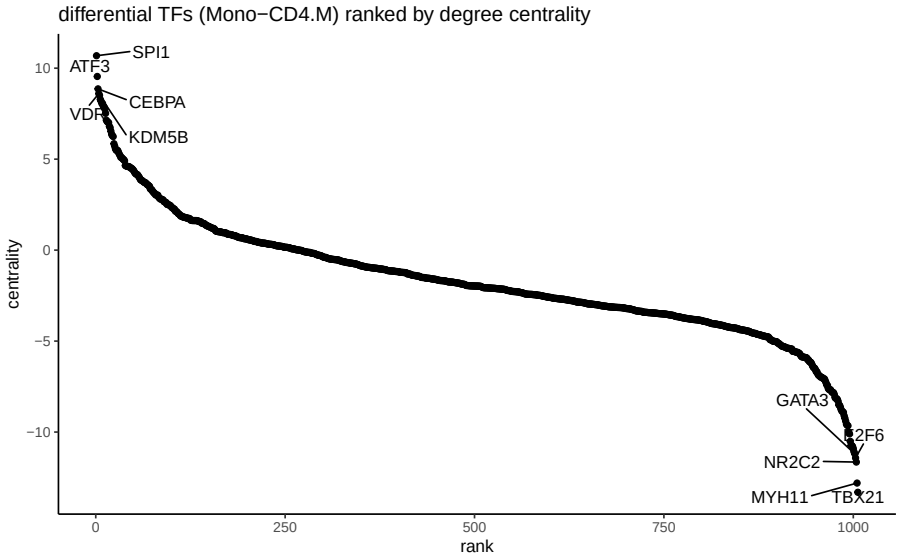



```
# obtain empirical p-values
p_matrix <- rowMeans(apply(permute_matrix, 2, function(x) {x > similarity_score_EBF1}))
p_matrix[names(head(sort(diff_matrix,decreasing = TRUE),20))]
##   PAX5   IRF4 POU2F2   RELB  IKZF2  FOXP1   BCL6  RUNX3   TCF4  FOXO1  NIPBL
##      0      0      0      0      0      0      0      0      0      0      0
## BCLAF1 BCL11A   CHD2 NFATC1 CREBBP  IKZF1  KMT2D   BCL3   MTA3
##      0      0      0      0      0      0      0      0      0
```

Next, we are interested to compare the networks of two cell types, in this case, CD4 memory cells (CD4.M) vs Monocytes (mono) cells. We build an edge subtracted graph and then calculate the degree centrality of the subtracted graph. We normalize centrality using the default square root function. The top 5 most positive TFs represent lineage factors more active in NK cells whereas the bottom 5 TFs present lineage factors enriched in CD4. We successfully identified the myeloid factor SPI1 to be associated with monocytes and Th1 factor TBX21 to be associated with CD4 cells.

```
# construct a graph of the CD4.M and NK cells respectively
CD4.M_network <- buildGraph(regulon.w, weights = "weight", cluster="CD4.M")
## Building graph using weight as edge weights
Mono_network <- buildGraph(regulon.w, weights = "weight", cluster="Mono")
## Building graph using weight as edge weights

# construct a difference graph
diff_graph <- buildDiffGraph( Mono_network,CD4.M_network, abs_diff = FALSE)
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 = rbind(head(rank_table, 5),
                                          tail(rank_table,5)),
```

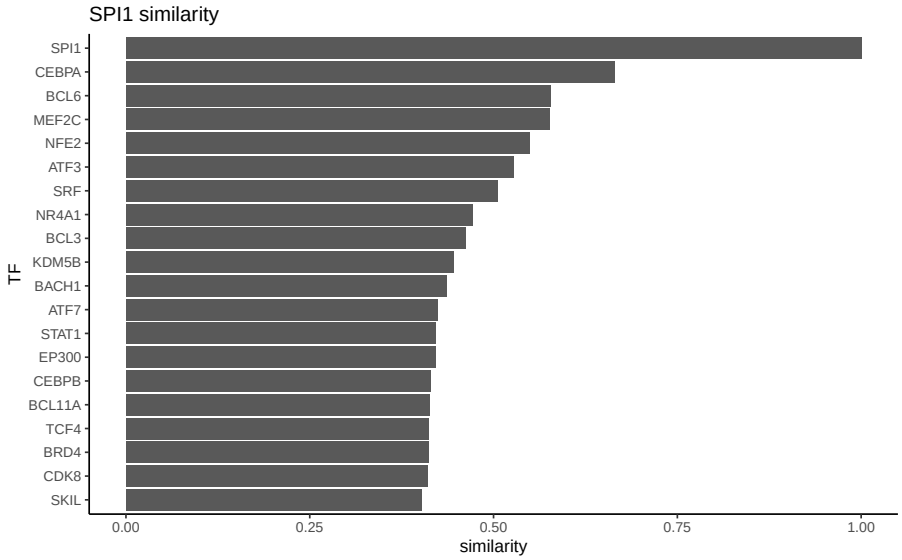
```
                                  aes(label = tf),
                                  nudge_x = 0, nudge_y = 0, box.padding = 0.5) +
      theme_classic() + ggtitle ("differential TFs (Mono-CD4.M) ranked by degree centrality")
```



We can further explore interacting factors with the myeloid factor SPI1 using the same Jaccard similarity approach. We found CEBPA as the most similar TF as SPI1. SPI1 and CEBPA are known to be important for differentiation into myeloid cells (https://www.cell.com/cell-reports/pdfExtended/S2211-1247(18)30745-9).

```
diff_graph_filter <- subgraph.edges(diff_graph,
                                    E(diff_graph)[E(diff_graph)$weight>0],
                                    del=T)


# compute a similarity matrix of all TFs
similarity_score <- calculateJaccardSimilarity(diff_graph_filter)

# Focus on SPI1
similarity_score_SPI1 <- similarity_score[, "SPI1"]
similarity_df <- data.frame(similarity = head(sort(similarity_score_SPI1,
                                                   decreasing = TRUE),20),
                            TF = names(head(sort(similarity_score_SPI1,
                                                 decreasing = TRUE),20)))

similarity_df$TF <- factor(similarity_df$TF,
                           levels = rev(unique(similarity_df$TF)))

# plot top TFs most similar to SPI1
topTFplot <- ggplot(similarity_df, aes(x=TF, y=similarity)) +
  geom_bar(stat="identity") +
  coord_flip() +
  ggtitle("SPI1 similarity") +
```

```
  theme_classic()

print(topTFplot)
```



6 Session Info

```
sessionInfo()
## R version 4.3.0 (2023-04-21)
## 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;  LAPACK version 3.11.0
##
## locale:
##  [1] LC_CTYPE=en_US.UTF-8 LC_NUMERIC=C         LC_TIME=C
##  [4] LC_COLLATE=C         LC_MONETARY=C        LC_MESSAGES=C
##  [7] LC_PAPER=C           LC_NAME=C            LC_ADDRESS=C
## [10] LC_TELEPHONE=C       LC_MEASUREMENT=C     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] epiregulon_1.0.27           msigdbr_7.5.1
##  [3] scMultiome_1.1.0            MultiAssayExperiment_1.27.0
##  [5] ExperimentHub_2.9.1         AnnotationHub_3.9.1
##  [7] BiocFileCache_2.9.1         dbplyr_2.3.3
##  [9] BiocStyle_2.29.1            ggrepel_0.9.3
## [11] ggplot2_3.4.2               batchtoolsSSH_0.3.6
## [13] checkmate_2.2.0             dorothea_1.13.0
## [15] testthat_3.1.8              SingleCellExperiment_1.23.0
## [17] SummarizedExperiment_1.31.1 GenomicRanges_1.53.1
## [19] GenomeInfoDb_1.37.2         MatrixGenerics_1.13.0
## [21] matrixStats_1.0.0           annotate_1.79.0
## [23] XML_3.99-0.14               org.Hs.eg.db_3.17.0
## [25] AnnotationDbi_1.63.2        IRanges_2.35.2
## [27] S4Vectors_0.39.1            Biobase_2.61.0
## [29] BiocGenerics_0.47.0         ggplotify_0.1.1
## [31] gridExtra_2.3               yardstick_1.2.0
##
## loaded via a namespace (and not attached):
##   [1] R.methodsS3_1.8.2                 GSEABase_1.63.0
##   [3] progress_1.2.2                    urlchecker_1.0.1
##   [5] poweRlaw_0.70.6                   Biostrings_2.69.2
##   [7] HDF5Array_1.29.3                  vctrs_0.6.2
##   [9] digest_0.6.31                     png_0.1-8
##  [11] shape_1.4.6                       bcellViper_1.37.0
##  [13] MASS_7.3-60                       reshape2_1.4.4
##  [15] httpuv_1.6.11                     foreach_1.5.2
##  [17] qvalue_2.33.0                     withr_2.5.0
##  [19] ggrastr_1.0.2                     xfun_0.39
##  [21] ggfun_0.1.1                       ellipsis_0.3.2
##  [23] memoise_2.0.1                     ggbeeswarm_0.7.2
##  [25] clusterProfiler_4.9.2             gson_0.1.0
##  [27] profvis_0.3.8                     tidytree_0.4.4
##  [29] GlobalOptions_0.1.2               gtools_3.9.4
##  [31] KEGGgraph_1.61.0                  entropy_1.3.1
##  [33] R.oo_1.25.0                       prettyunits_1.1.1
##  [35] KEGGREST_1.41.0                   promises_1.2.0.1
##  [37] httr_1.4.6                        downloader_0.4
##  [39] restfulr_0.0.15                   rhdf5filters_1.13.5
##  [41] ps_1.7.5                          rhdf5_2.45.1
##  [43] rstudioapi_0.14                   miniUI_0.1.1.1
##  [45] generics_0.1.3                    DOSE_3.27.2
##  [47] processx_3.8.1                    babelgene_22.9
##  [49] curl_5.0.0                        zlibbioc_1.47.0
##  [51] ScaledMatrix_1.9.1                ggraph_2.1.0
##  [53] polyclip_1.10-4                   GenomeInfoDbData_1.2.10
##  [55] SparseArray_1.1.10                interactiveDisplayBase_1.39.0
##  [57] xtable_1.8-4                      stringr_1.5.0
##  [59] desc_1.4.2                        pracma_2.4.2
##  [61] doParallel_1.0.17                 evaluate_0.21
##  [63] S4Arrays_1.1.4                    hms_1.1.3
##  [65] bookdown_0.34                     irlba_2.3.5.1
```

```
##  [67] colorspace_2.1-0                 filelock_1.0.2
##  [69] magrittr_2.0.3                   readr_2.1.4
##  [71] Rgraphviz_2.45.0                 later_1.3.1
##  [73] viridis_0.6.4                    ggtree_3.9.0
##  [75] lattice_0.21-8                   scuttle_1.11.0
##  [77] shadowtext_0.1.2                 cowplot_1.1.1
##  [79] pillar_1.9.0                     nlme_3.1-162
##  [81] iterators_1.0.14                 caTools_1.18.2
##  [83] compiler_4.3.0                   beachmat_2.17.13
##  [85] debugme_1.1.0                    stringi_1.7.12
##  [87] devtools_2.4.5                   GenomicAlignments_1.37.0
##  [89] MPO.db_0.99.7                    plyr_1.8.8
##  [91] crayon_1.5.2                     BiocIO_1.11.0
##  [93] scater_1.29.0                    gridGraphics_0.5-1
##  [95] locfit_1.5-9.8                   graphlayouts_1.0.0
##  [97] bit_4.0.5                        dplyr_1.1.2
##  [99] fastmatch_1.1-3                  codetools_0.2-19
## [101] BiocSingular_1.17.1              chromVARmotifs_0.2.0
## [103] GetoptLong_1.0.5                 mime_0.12
## [105] splines_4.3.0                    circlize_0.4.15
## [107] Rcpp_1.0.11                      sparseMatrixStats_1.13.0
## [109] HDO.db_0.99.1                    EnrichmentBrowser_2.31.2
## [111] brew_1.0-8                       knitr_1.43
## [113] blob_1.2.4                       utf8_1.2.3
## [115] clue_0.3-64                      BiocVersion_3.18.0
## [117] seqLogo_1.67.0                   fs_1.6.2
## [119] batchtools_0.9.17                DelayedMatrixStats_1.23.0
## [121] pkgbuild_1.4.0                   GSVA_1.49.2
## [123] tibble_3.2.1                     Matrix_1.6-0
## [125] statmod_1.5.0                    callr_3.7.3
## [127] tzdb_0.4.0                       tweenr_2.0.2
## [129] pkgconfig_2.0.3                  BSgenome.Hsapiens.UCSC.hg19_1.4.3
## [131] pheatmap_1.0.12                  tools_4.3.0
## [133] cachem_1.0.8                     BSgenome.Mmusculus.UCSC.mm10_1.4.3
## [135] RSQLite_2.3.1                    viridisLite_0.4.2
## [137] DBI_1.1.3                        fastmap_1.1.1
## [139] rmarkdown_2.23                   scales_1.2.1
## [141] grid_4.3.0                       usethis_2.1.6
## [143] Rsamtools_2.17.0                 patchwork_1.1.2
## [145] BiocManager_1.30.21.1            graph_1.79.0
## [147] farver_2.1.1                     tidygraph_1.2.3
## [149] scatterpie_0.2.1                 yaml_2.3.7
## [151] rtracklayer_1.61.0               cli_3.6.1
## [153] purrr_1.0.1                      motifmatchr_1.23.0
## [155] lifecycle_1.0.3                  bluster_1.11.3
## [157] sessioninfo_1.2.2                backports_1.4.1
## [159] BSgenome.Hsapiens.UCSC.hg38_1.4.5  BiocParallel_1.35.3
## [161] gtable_0.3.3                     rjson_0.2.21
## [163] parallel_4.3.0                   ape_5.7-1
## [165] limma_3.57.6                     jsonlite_1.8.7
## [167] edgeR_3.43.7                     TFBSTools_1.39.0
```

```
## [169] bitops_1.0-7                    HPO.db_0.99.2
## [171] bit64_4.0.5                     brio_1.1.3
## [173] yulab.utils_0.0.6               BiocNeighbors_1.19.0
## [175] CNEr_1.37.0                     metapod_1.9.0
## [177] GOSemSim_2.27.2                 dqrng_0.3.0
## [179] R.utils_2.12.2                  lazyeval_0.2.2
## [181] shiny_1.7.4.1                   htmltools_0.5.5
## [183] enrichplot_1.21.1               GO.db_3.17.0
## [185] rappdirs_0.3.3                  glue_1.6.2
## [187] TFMPvalue_0.0.9                 XVector_0.41.1
## [189] RCurl_1.98-1.12                 rprojroot_2.0.3
## [191] treeio_1.25.1                   scran_1.29.0
## [193] BSgenome_1.69.0                 AUCell_1.23.0
## [195] igraph_1.5.0.1                  R6_2.5.1
## [197] tidyr_1.3.0                     labeling_0.4.2
## [199] cluster_2.1.4                   pkgload_1.3.2
## [201] Rhdf5lib_1.23.0                 ArchR_1.0.3
## [203] aplot_0.1.10                    DirichletMultinomial_1.43.0
## [205] DelayedArray_0.27.9             tidyselect_1.2.0
## [207] vipor_0.4.5                     ggforce_0.4.1
## [209] rsvd_1.0.5                      munsell_0.5.0
## [211] data.table_1.14.8               htmlwidgets_1.6.2
## [213] fgsea_1.27.0                    ComplexHeatmap_2.17.0
## [215] RColorBrewer_1.1-3              rlang_1.1.1
## [217] remotes_2.4.2                   Cairo_1.6-0
## [219] fansi_1.0.4                     base64url_1.4
## [221] beeswarm_0.4.0
```