MultiRNAflow: An R package for integrated analysis of temporal RNA-seq data with multiple biological conditions

Loubaton Rodolphe [1], ***Champagnat Nicolas*** [2], ***Vallois Pierre*** [2], ***and Vallat Laurent*** [3,4]

[1]Université Clermont Auvergne, INRAE, VetAgro Sup, UMR-1213 Herbivores, F-63122 Saint-Genès-Champanelle, France
[2]Université de Lorraine, CNRS, Inria, IECL, F-54000 Nancy, France
[3]Université de Strasbourg, CNRS, UMR-7242 Biotechnology and cell signaling, F-67400 Illkirch, France
[4]Department of molecular genetic of cancers, Strasbourg University Hospital, F-67200 Strasbourg, France

October 31, 2025

Abstract

The dynamic transcriptional mechanisms that govern eukaryotic cell function can now be analyzed by RNA sequencing (RNAseq). However, the packages currently available for the analysis of raw sequencing data do not provide automatic analysis of complex experimental designs with multiple biological conditions and multiple analysis time-points.
The MultiRNAflow suite combines several packages in a unified framework allowing exploratory and supervised statistical analysis of temporal data for multiple biological conditions.

To cite this package

Rodolphe Loubaton et al. "MultiRNAflow: integrated analysis of temporal RNA-seq data with multiple biological conditions". In: *Bioinformatics* 40.5 (May 2024), btae315. DOI: https://doi.org/10.1093/bioinformatics/btae315

Contents

1 Introduction

1.1 Context

In eukaryotic cells, genes are expressed in the form of RNA molecules during the transcriptional process which are then translated into proteins with a cellular function. In resting cells, at the steady state, transcription is affected by stochastic phenomena generating a transcriptional noise within cells. After modification of the cellular environment (cellular stress, receptor activation), hundreds of genes are activated, inducing a dynamic temporal transcriptional response allowing an adapted response of the cells to the initial modification of the environment [1]. Alterations in these temporal transcriptional responses are at the origin of pathologies (*e.g.* cancer) and are extensively studied by biologists [2] through sometimes complex experimental designs.

Recent technological developments now make it possible to quantify the transcription of all genes in the genome by sequencing RNA molecules (RNAseq). These analysis generate raw count data whose properties (discrete data) are different from the fluorescence intensity data (continuous data) generated by previous microarray techniques. These data are presented as a transcriptome table reporting, for each sample, the number of reads for each gene, obtained from the alignment of collected RNA transcripts to a reference genome. The raw count of a gene (or transcript) corresponds to the number of reads mapped to the RNA sequence of this gene (or transcript).

The number of reads of a gene depends on the length of the gene and experimental or systematic errors may occur during sequencing (uncertainty on sequencing depth, effective library sizes...), which require the transformation of the original raw counts.
The first method developed consists to compute count per million (CPM). Although the previous method corrects the effective library sizes problem, it should not be used for comparison between samples [3, 4], particularly if samples belong to different biological conditions and/or time points.
New methods of normalization have been developed in order to be able to compare gene expression between samples which belong to different biological conditions and/or time points. These methods of normalization, ensure that differences between samples are only due to their membership to different biological conditions and/or time points. The most used R packages for normalization are DESeq2 [5] and EdgeR [6].

Another motivation of normalization is to deduce a number of RNA transcripts of genes. This requires the so called reads per kilobase of transcripts per million reads mapped (RPKM) [7] or transcripts per million (TPM) [8, 4], as these methods correct both the effective library sizes and the dependence of the number of reads of a gene with its length.

The two goals of normalization of raw counts data are 1) to allow for unsupervised analysis of data (within samples or between samples) and 2) to compare gene reads in different biological condition and/or time points, to detect so-called **Differentially Expressed** (DE) genes.

1.2 Context

Several R packages propose tools to normalize data, realize unsupervised analysis and find DE genes, such as IDEAL [9], RNASeqR [10], SeqGSEA
[11] and RNAflow [12]. These packages use DESeq2 [5] and/or EdgedR [6] in order to realize the normalization and DE analysis. All of them can detect DE genes in samples belonging to different biological conditions, although RNASeqR is limited to only two biological conditions. Some of them also perform GO enrichment analysis. However, these packages were not designed to deal with temporal data, although they could be adapted to this situation. None of them offer a unified and automatized framework to analyze RNA-seq data with both several time points and more than two biological conditions. Furthermore, these packages do not allow to automatically select subsets of genes that can be relevant for GO enrichment analysis, such as genes which are specific to a given biological condition and/or to a given time, or genes with particular DE patterns.

1.3 Our R package MultiRNAflow

The MultiRNAflow [13] suite gathers in a unified framework methodological tools found in various existing packages allowing to perform:

1. Exploratory (unsupervised) analysis of the data.
2. Statistical (supervised) analysis of dynamic transcriptional expression (DE genes), based on DESeq2 package [5].
3. Functional and GO analysis of subsets of genes automatically selected by the package, such as specific genes or genes with a given DE temporal pattern.

The package automates a commonly used workflow of analysis for studying complex biological phenomena (used *e.g.* in [14]).

1.4 Supported dataset

The package supports transcriptional RNAseq raw count data (and can be adapted to single cell RNAseq) from an experimental design with multiple conditions and/or multiple times. The experimental design supported by our packages assumes that there is a reference time noted t_0, distinct from the other times noted t_1 to t_n, which corresponds to a set of reference measurements to which the others are to be compared (*e.g.* as in [14], where t_0 corresponds to the basal state of the cell before activation of a cell receptor, and the experiments at times t_1 to t_n measure gene expression at different times after activation of the receptor). Our package is not designed to analyze experimental designs requiring to compare measurements between any pairs of times.
The package provides numerous graphical outputs that can be selected by the user. To illustrate these outputs, we gather in Figure 1 a selection of graphics obtained from the dataset of [15], which analyzes the role of invalidation of Bmal1 and Cry1/2 genes on murine transcriptional dynamics. The experimental map contains 4 biological conditions (Bmal1 wild type (wt), Bmal1 knock-out (ko), Cry1/2 wt and Cry1/2 ko) and 6 time points each ($t_0 = 0$h, $t_1 = 4$h, $t_2 = 8$h, $t_3 = 12$h, $t_4 = 16$h and $t_5 = 20$h), with 4 replicates (Figure 1.A).

The dataset is a table of raw counts where lines correspond to genes and columns correspond to samples. Each sample shows the raw counts of an individual sequencing, corresponding to a biological condition, an individual sampling in this biological condition and a time.

In this document, we illustrate the use of our package on four examples of datasets (see section Dataset used as examples in the package) corresponding to four cases:

- **Case 1**. Samples belong to different biological conditions.
- **Case 2**. Measures were realized at different time points.
- **Case 3**. Samples belong to two biological conditions and measures were realized at different time points.
- **Case 4**. Samples belong to different biological conditions and measures were realized at different time points.

1.5 Steps of the algorithm

The package MultiRNAflow realizes the following steps:

- Normalization, realized with the R package DESeq2 [5].
- Exploratory data unsupervised analysis which includes
  - Visualization of individual patterns using factorial analysis with the R package FactoMineR [16].
  - Visualization of biological conditions and/or temporal clusters with the R package ComplexHeatmap [17]
  - Visualization of groups of genes with similar temporal behavior with the R package Mfuzz [18, 19]
- Statistical supervised analysis of the transcriptional response of different groups of individuals over time with the R package DESeq2 [5], which includes
  - Temporal statistical DE analysis
  - Statistical DE analysis by biological condition
  - Combination of temporal and biological condition statistical DE analysis
  - Gene Ontology (GO) enrichment analysis using the R package gprofiler2 [20] (Figure 1.K), and automatic generation of outputs that can be implemented in DAVID [21], Webgestalt [22] (Figure 1.L), GSEA [23] (Figure 1.M), gProfiler [24], Panther [25], ShinyGO [26], Enrichr [27] and GOrilla [28]. for further analysis using these databases.

Below, we give a short description of each of these steps before a full description of the package outputs for four examples of datasets. Figure1 illustrates the short description below. It gathers a selection of graphs produced by our package for the Example of MultiRNAflow in case 4, several time points and more than two biological conditions: Mouse dataset 2 [15] corresponding to **case 4**.

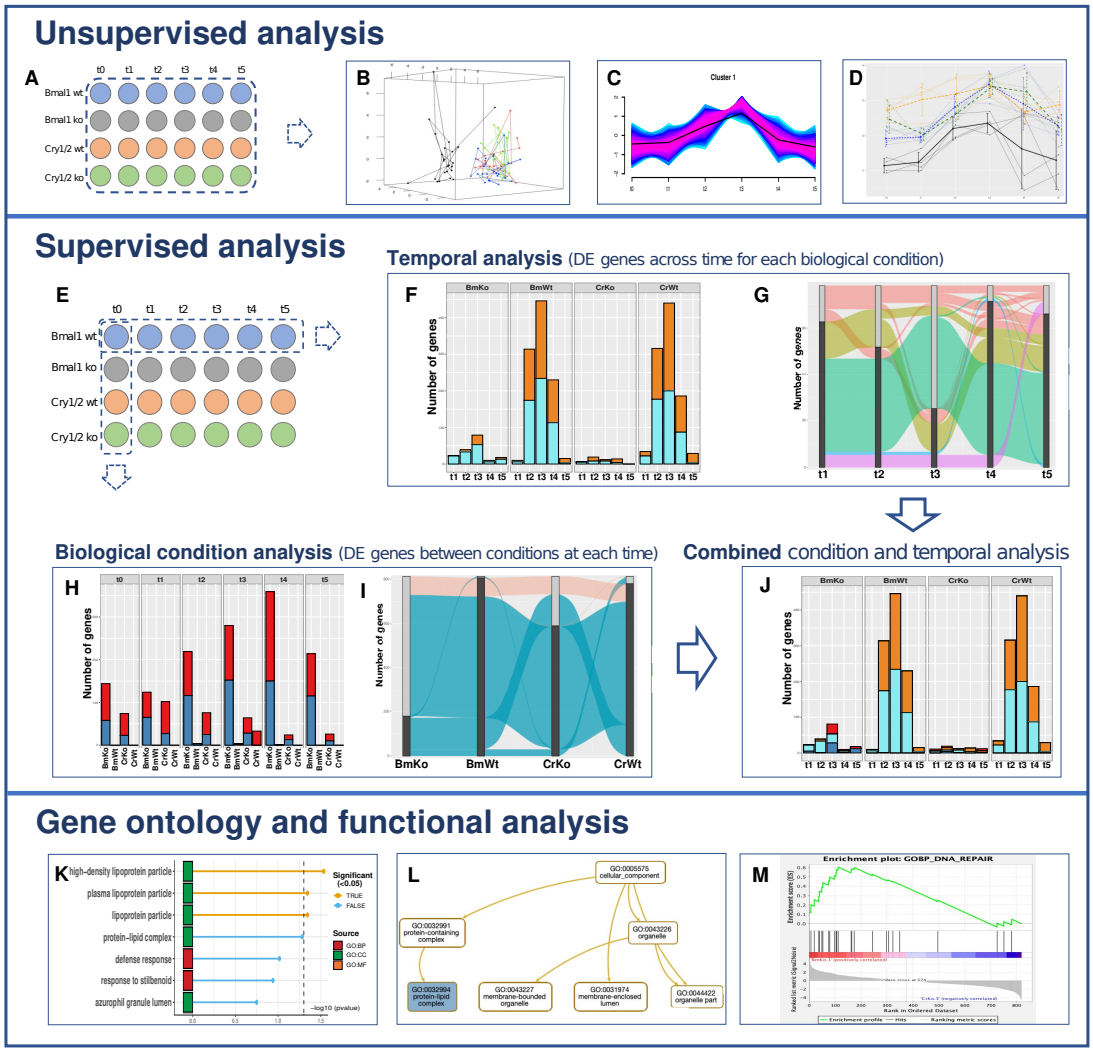


Figure 1: Outputs from the package MultiRNAflow with a dataset containing several biological conditions and several time points (experimental design shown in (A)). Exploratory analysis includes 3D PCA (B), temporal clustering of expression (C) and detailed temporal gene expression (D). Supervised statistical analysis (experimental map shown in (E)) includes DE genes between each time and the reference time for each condition (F and G); specific DE genes for each condition at each time (H) or at least at one time point (I); signature DE genes of each condition and each time (J). GO enrichment analysis is realized with the R package gprofiler2 (K) or by generating input files for several GO software programs, such as Webgestalt (L) or GSEA (M).

1.5.1 Normalization

The function **DATAnormalization()** of our package allows to realize the three methods of normalization proposed in DESeq2 and RPKM, which must be performed on raw counts for analysis (like DE analysis):

- Relative log expression (rle) [29]. Each column of the raw counts is scaled by a specific scalar called size factors, estimated using the "median ratio method".
- Regularized logarithm (rlog) [5]. This method of normalization transforms the count data to the log2 scale in a way that minimizes differences between samples for rows (so genes) with small counts. This transformation removes the dependence of the variance on the mean, particularly the high variance of the logarithm of count data when the mean is low. This method of normalization is realized by the R function `rlog()` of the package DESeq2.
- Variance Stabilizing Transformation (vst) [29]. This method of normalization is similar to the rlog normalization. The vst normalization is faster but the rlog normalization is more robust in the case when the size factors vary widely. This method of normalization is realized by the R function `vst()` of the package DESeq2.

As mentioned in the DESeq2 manual, the rle transformation is used to realize DE analysis and the rlog and vst transformations are advised for unsupervised analysis (factorial analysis, clustering) or other machine learning methods.

In addition, the function **DATAnormalization()** allows to plot the distribution of normalized read counts for each sample using boxplots.

1.5.2 Exploratory data analysis (unsupervised analysis)

1.5.2.1 Factorial analysis

Factorial analysis is realized by the two functions **PCAanalysis()** and **HCPCanalysis()** which implement two methods: Principal Component Analysis (PCA) and Hierarchical Clustering on Principle Components (HCPC). The two methods allow to visualize the temporal evolution of the transcription within each group of individuals and similarities or differences in transcriptional behaviors between groups. Of note, the PCA visualization is optimized thanks to the dynamic 3D PCA (see Figure 1.B) allowing several viewing angles.

- **Case 1**. In the PCA graphs, samples are colored according to biological condition.
- **Case 2**. In the PCA graphs, consecutive time points for a same sample are linked to help visualization of temporal patterns.
- **Cases 3 and 4**. The PCA graphs combine the two previous displaying.

1.5.2.2 Visualization of groups of genes with similar temporal behavior

The R function **MFUZZanalysis()** allows, in **case 2**, **case 3** and **case 4**, to find the most common temporal behavior among all genes and all individuals in a given biological condition. This is done using the R package Mfuzz [18, 19] based on soft clustering. When there are several replicates per time, the Mfuzz package realizes soft clustering from the mean expression per time for each gene (see Figure 1.C). When there are several biological conditions, the algorithm realizes the Mfuzz analysis for each biological condition.

As with most clustering method, we need to find out the optimal number of clusters. Although a method is already implemented in the Mfuzz package, this method seems to fail when the number of genes is too big. Our function **MFUZZclusternumber()** finds the optimal number of cluster using kmeans() from the R package stats [30] or HCPC() from the R package FactoMineR [16]. Among the outputs, **MFUZZclusternumber()** returns

- a graph indicating the selected number of clusters for each biological condition
- the results of the soft clustering for each biological condition (see Figure 1.C)

1.5.2.3 Visualization of the data

The R function **DATAplotExpressionGenes()** allows to plot gene expression profiles of a selection of genes according to time and/or biological conditions (see Figure 1.D). The user can either use raw counts data or normalized data.

- **Case 1**. The output is a graph where are plotted: a box plot, a violin plot, and error bars (standard deviation) for each biological condition.
- **Case 2**. The output is a graph where are plotted: the evolution of the expression of each replicate across time (red lines) and the evolution of the mean and the standard deviation of the expression across time (black lines).
- **Cases 3 and 4**. The output is a graph where are plotted: the evolution of the mean and the standard deviation of the expression across time for each biological condition.

1.5.3 Statistical analysis of the transcriptional response of different biological conditions of individuals over time

1.5.3.1 Differentially expressed (DE) genes with DESeq2

The function **DEanalysisGlobal()** search for DE genes.

Case 1. Our algorithm searches for differentially expressed (DE) genes between all pairs of biological conditions. This allows in particular to determine which genes are specific to each biological condition. A gene is called specific to a biological condition A, if the gene is DE between A and any other biological conditions, but not DE between any pairs of other biological conditions.
Among the outputs, **DEanalysisGlobal()** returns

- a Venn barplot which gives the number of genes for each possible intersection. We consider that a set of pairs of biological conditions forms an intersection if there is at least one gene which is DE for each of these pairs of biological conditions, but not for the others.
- a barplot which gives the number of specific (and over- and under-expressed, also often called up- and down-regulated) genes per biological condition.

Case 2. Our algorithm looks for differentially expressed genes between each time t_i ($1 \leq i \leq n$) and the reference time t_0.
Among the outputs, **DEanalysisGlobal()** returns

- an alluvial graph of differentially expressed (DE) genes.
- a Venn barplot which gives the number of genes per temporal pattern. By temporal pattern, we mean the set of times t_i such that the gene is DE between t_i and the reference time t_0.

Case 3 and **Case 4**. Our algorithm realizes a mix of the two previous cases. First, for each biological condition, the algorithm realizes **Case 2**. Then for each time, the algorithm realizes **Case 1**. We then can find specific genes. A gene is called specific to a biological condition A at a time t_i, if the gene is DE between A and any other biological conditions at time t_i, but not DE between any pairs of other biological conditions at time t_i. The algorithm also finds signature genes: a gene is called signature of a biological condition A at a given time t_i if the gene is specific for A at time t_i and DE between t_i versus t_0 for A.

Among the outputs, **DEanalysisGlobal()** returns

- An alluvial graph of differentially expressed (DE) genes, for each biological condition (see Figure 1.G).
- A barplot which gives the number of DE genes per time, for each biological condition (see Figure 1.F).
- A barplot which gives the number of specific genes for each biological condition, one per time (see Figure 1.H).
- An alluvial graph of genes which are specific at least at one time, for each biological condition (see Figure 1.I).
- A graph which gives for each biological condition, the number of signature genes and non signature genes per time t_i versus the reference time t_0 (see Figure 1.J).

1.5.3.2 Heatmaps, ratio intensity (MA) plots and volcano plots

Clustering of samples versus genes allows visualization of correlations between gene expressions according to biological conditions or times. Clustering of samples versus samples allows visualization of correlations between individuals and groups. Given the high number of genes in a dataset, the heatmaps are realized after the supervised analysis in order to reduce the number of genes.

1.5.3.3 Gene ontology and gene enrichment

Gene Ontology (GO) enrichment analysis search for a functional profile of DE genes and better understand the underlying biological processes. We recommend the most used online tools and software: GSEA [23] (see Figure 1.M), DAVID [21], WebGestalt [22] (see Figure 1.L), g:Profiler [24], Panther [25], ShinyGO [26], Enrichr [27] and GOrilla [28]. Each of these softwares and online tools requires specific input files in order to realize their analysis. The R function **GSEApreprocessing()** automatically creates all required files.

Alternatively, the function **GSEAquickAnalysis()** provides a GSEA analysis with the R package gprofiler2 [20]. Among the ouputs, **GSEAquickAnalysis()** returns

- a lollipop graph (Figure 2) showing the most important Gene Ontologies ranked by their $-\log_{10}(\text{pvalue})$. The y-axis indicates the `MaxNumberGO` most significant gene ontologies and pathways associated to the selected DE genes. The gene ontologies and pathways are sorted into descending order. The x-axis indicates the $-log10(pvalues)$. The higher is a lollipop the more significant is a gene ontology or pathway. A lollipop is yellow if the pvalues is smaller than 0.05 (significant) and blue otherwise.
- A Manhattan plot (Figure 3) ranking all genes ontologies according to the functional database (G0::BP, G0::CC, G0::MF and KEGG)

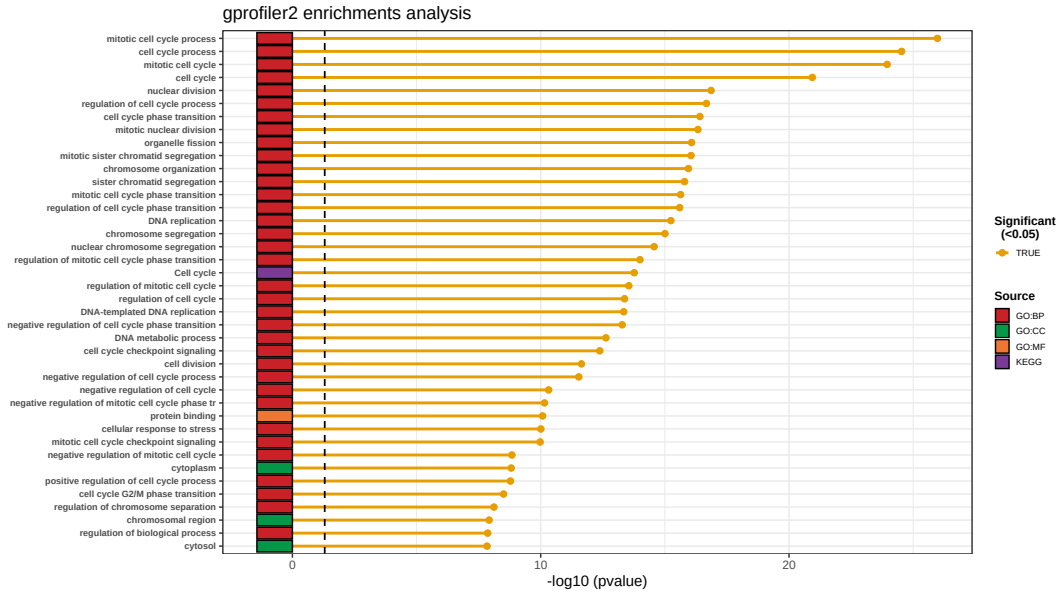


Figure 2: Lollipop chart giving the most significant gene ontologies

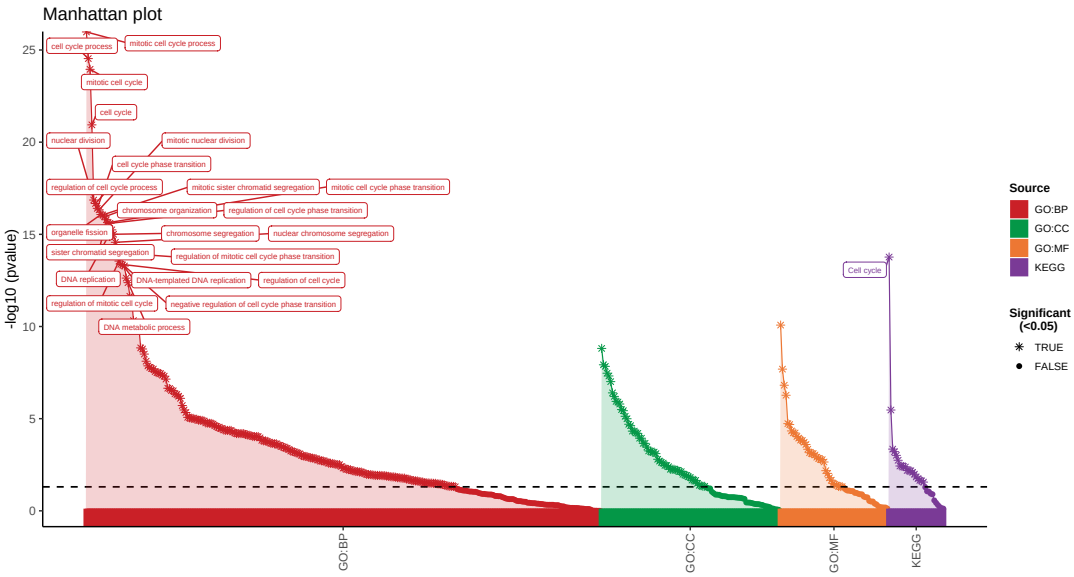


Figure 3: Mahattan plot indicating all genes ontologies

1.6 Dataset used as examples in the package

In order to explain the use of each function in our package, we use four datasets.

1.6.1 Example of MultiRNAflow in case 1, several biological conditions: Mouse dataset 1

The **Mouse dataset 1** [31] is accessible on the Gene Expression Omnibus (GEO) database with the accession number GSE169116.

This dataset contains the transcription profile of 12 mice belonging to 4 biological conditions:

- 3 mice with wild type Notch1 and wild type Tcf1
- 3 mice with wild type Notch1 and Tcf1 knocked-down
- 3 mice with Notch1 induced and wild type Tcf1
- 3 mice with Notch1 induced and Tcf1 knocked-down.

The dataset contains temporal expression data of 39017 genes [31].

To illustrate the use of our package in **case 1**, we selected 500 genes giving a representative sample of each DE profile across biological conditions, in particular genes that are specific to each biological condition.
This sub dataset is saved in the file **RawCounts_Antoszewski2022_MOUSEsub500**.

1.6.2 Example of MultiRNAflow in case 2, several time points: Fission dataset

The **Fission dataset** [32] is accessible on the Gene Expression Omnibus (GEO) database with the accession number GSE56761. The dataset can also be obtained with the R package "fission" [32].

This dataset contains the temporal transcription profile of 18 wild type fission yeasts (wt) and 18 fission yeasts where atf1 is knocked-out (mut), hence 36 samples. The dataset contains temporal expression data of 7039 genes. Data were collected 0, 15, 30, 60, 120 and 180 minutes after an osmotic stress. The gene atf1 codes for a transcription factor which alters sensitivity to oxidative stress.

To illustrate the use of our package in **case 2**, we focus on the biological condition wt and select 500 genes giving a representative sample of each temporal DE profile in this biological condition. This sub dataset is saved in the file **RawCounts_Leong2014_FISSIONsub500wt**.

1.6.3 Example of MultiRNAflow in case 3, several time points and two biological conditions: Leukemia dataset

The **Leukemia dataset** [14] is accessible on the Gene Expression Omnibus (GEO) database with the accession number GSE130385.

This dataset contains the temporal transcription profile of 3 Proliferating (P) and 3 Non Proliferating (NP) primary chronic lymphocytic leukemia (CLL) B-cells samples. Data were collected at 0, 1h, 1h30, 3h30, 6h30, 12h, 24h, 48h and 96h after cell stimulation (so $(3+3)\times 9 = 54$ samples in total). The latest time point corresponds to the emergence of the proliferation clusters. The dataset contains temporal expression data of 25369 genes.

To illustrate the use of our package in **case 3** with two biological conditions, we selected 500 genes giving a representative sample of each DE profile across time and biological conditions, in particular genes that are signature genes of each biological condition. This sub dataset is saved in the file **RawCounts_Schleiss2021_CLLsub500**.

1.6.4 Example of MultiRNAflow in case 4, several time points and more than two biological conditions: Mouse dataset 2

The **Mouse dataset 2** [15] is accessible on the Gene Expression Omnibus (GEO) database with the accession number GSE135898.

This dataset contains the temporal transcription profile of 16 mice with Bmal1 and Cry1/2 knocked-down under an ad libitum (AL) or night restricted feeding (RF) regimen. Data were collected at 0h, 4h, 8h, 12h, 16h and 20h. Therefore, there are six time points and eight biological conditions. As there are only two mice per biological condition, we decided not to take into account the effect of the regimen. This leads to 4 biological conditions with 4 mice in each. The dataset contains temporal expression data of 40327 genes.

To illustrate the use of our package in **case 3** with more than two biological conditions, we take 500 genes, over the global 40327 genes in the original dataset. This sub dataset is saved in the file **RawCounts_Weger2021_MOUSEsub500**.

2 Preamble

2.1 R version and R packages to install

Before installing the necessary packages, you must install (or update) the R software in a version superior or equal to 4.2.1 "Funny-Looking Kid" (released on 2022/06/23) from CRAN (Comprehensive R Archive Network).

Then, in order to use the MultiRNAflow package, the following R packages must be installed:

- From CRAN: reshape2 (>= 1.4.4), ggplot2 (>= 3.4.0), ggalluvial (>= 0.12.3), ggrepel (>= 0.9.2), FactoMineR (>= 2.6), factoextra (>= 1.0.7), plot3D (>= 1.4), plot3Drgl (>= 1.0.3), ggplotify (>= 0.1.2), UpSetR (>= 1.4.0), gprofiler2 (>= 0.2.1).
- From CRAN and usually already included by default in R: graphics (>= 4.2.2), grDevices (>= 4.2.2), grid (>= 4.2.2), utils (>= 4.2.2), stats (>= 4.2.2).
- From Bioconductor: SummarizedExperiment (>= 1.28.0), DESeq2 (>= 1.38.1), ComplexHeatmap (>= 2.14.0), Mfuzz (>= 2.58.0).

Before installing a package, for instance the package FactoMineR, the user must check if the package is already installed with the command `library(FactoMineR)`. If the package has not been previously installed, the user must use the command `install.packages("FactoMineR")` (packages from CRAN). For beginners in programming, we recommend to follow the steps below for importing CRAN and Bioconductor packages.

For the packages which must be download from CRAN,

```
Cran.pck <- c("reshape2", "ggplot2", "ggrepel", "ggalluvial",
              "FactoMineR", "factoextra",
              "plot3D", "plot3Drgl", "ggplotify", "UpSetR", "gprofiler2")
```

the user can copy and paste the following lines of code for each package in order to download the missing packages.

```
Select.package.CRAN <- "FactoMineR"
if (!require(package=Select.package.CRAN,
             quietly=TRUE, character.only=TRUE, warn.conflicts=FALSE)) {
    install.packages(pkgs=Select.package.CRAN, dependencies=TRUE)
}# if(!require(package=Cran.pck[i], quietly=TRUE, character.only=TRUE))
```

If the package is already installed (for instance here "FactoMineR"), the previous lines of code will return nothing.

For the packages which must be download from Bioconductor,

```
Bioconductor.pck <- c("SummarizedExperiment", "S4Vectors", "DESeq2",
                      "Mfuzz", "ComplexHeatmap")
```

the user must first copy and paste the following lines of code in order to install "BiocManager"

```
if (!require(package="BiocManager",
             quietly=TRUE, character.only=TRUE, warn.conflicts=FALSE)) {
    install.packages("BiocManager")
}# if(!require(package="BiocManager", quietly=TRUE, character.only=TRUE))
```

then copy and paste the following lines of code in order to install the version 3.16 of bioconductor (it works with R version 4.2.0)

```
BiocManager::install(version="3.18")
```

and then copy and paste the following lines of code for each package in order to download the missing packages.

```
Select.package.Bioc <- "DESeq2"
if(!require(package=Select.package.Bioc,
            quietly=TRUE, character.only=TRUE, warn.conflicts=FALSE)){
  BiocManager::install(pkgs=Select.package.Bioc)
}## if(!require(package=Select.package.Bioc, quietly=TRUE, character.only=TRUE))
```

If the package is already installed (for instance here "DESeq2"), the previous lines of code will return nothing.

Once all packages have been installed, the user may load our package.

```
library(MultiRNAflow)
```

2.2 Main functions

Our package contains 38 functions among which 11 are main functions. The user should only use

- **DATAprepSE()** to store all information about the dataset in a standardized way (SummarizedExperiment class object)
- these main functions for exploratory data analysis (unsupervised analysis)
  - **DATAnormalization()**. This function allows to normalize raw counts data and the results will be used by the functions **PCAanalysis()**, **HCPCanalysis()**, **MFUZZanalysis()** and **DATAplotExpressionGenes()**.
  - **PCAanalysis()**. The function realizes the PCA analysis.
  - **HCPCanalysis()**. The function realizes the clustering analysis with the R package HCPC.
  - **MFUZZanalysis()**. The function realizes the temporal clustering analysis with the R package Mfuzz
  - **DATAplotExpressionGenes()**. This function allows to plot the profile expression of all chosen genes.

- these main functions for supervised analysis
  - **DEanalysisGlobal()**. This function realizes the differential expression analysis.
  - **DEplotVolcanoMA()**. The function plots and save all Volcano and MA plots from the output of **DEanalysisGlobal()**.
  - **DEplotHeatmaps()**. The function plots a correlation heatmap and a heatmap of the normalized data for a selection of DE genes.
  - **GSEAQuickAnalysis()**. The function realizes the GSEA analysis with the R package gprofiler2.
  - **GSEApreprocessing()**. The function saves files to be used by 8 GSEA software and online tools.

2.3 Load of the dataset

If the user wants to use our package with one of the dataset included in MultiRNAflow, he must first write in the R console either

```
data("RawCounts_Antoszewski2022_MOUSEsub500")
```

in order to load the Example of MultiRNAflow in case 1, several biological conditions: Mouse dataset 1, either

```
data("RawCounts_Leong2014_FISSIONsub500wt")
```

in order to load the Example of MultiRNAflow in case 2, several time points: Fission dataset, either

```
data("RawCounts_Schleiss2021_CLLsub500")
```

in order to load the Example of MultiRNAflow in case 3, several time points and two biological conditions: Leukemia dataset, or

```
data("RawCounts_Weger2021_MOUSEsub500")
```

in order to load the Example of MultiRNAflow in case 4, several time points and more than two biological conditions: Mouse dataset 2.

2.4 Structure of the dataset

The dataset must be a data.frame containing raw counts data. If it is not the case, the function **DATAprepSE()** will stop and print an error. Each line should correspond to a gene, each column to a sample, except a particular column that may contain strings of characters describing the names of the genes. The first line of the data.frame should contain the names of the columns (strings of characters) that must have the following structure.

```
data("RawCounts_Leong2014_FISSIONsub500wt")
colnames(RawCounts_Leong2014_FISSIONsub500wt)
```

```
##  [1] "Gene"     "wt_t0_r1" "wt_t0_r2" "wt_t0_r3" "wt_t1_r1" "wt_t1_r2"
##  [7] "wt_t1_r3" "wt_t2_r1" "wt_t2_r2" "wt_t2_r3" "wt_t3_r1" "wt_t3_r2"
## [13] "wt_t3_r3" "wt_t4_r1" "wt_t4_r2" "wt_t4_r3" "wt_t5_r1" "wt_t5_r2"
## [19] "wt_t5_r3"
```

In this example, "Gene" indicates the column which contains the names of the different genes. The other column names contain all kind of information about the sample, including the biological condition, the time of measurement and the name of the individual (e.g patient, replicate, mouse, yeasts culture...). Other kinds of information can be stored in the column names (such as patient information), but they will not be used by the package. The various information in the column names must be separated by underscores. The order of these information is arbitrary but must be the same for all columns. For instance, the sample "wt_t0_r1" corresponds to the first replicate (r1) of the wild type yeast (wt) at time t_0 (reference time).

The information located to the left of the first underscore will be considered to be in position 1, the information located between the first underscore and the second one will be considered to be in position 2, and so on. In the previous example, the biological condition is in position 1, the time is in position 2 and the replicate is in position 3.

In most of the functions of our package, the order of the previous information in the column names will be indicated with the inputs `Group.position`, `Time.position` and `Individual.position`. Similarly the input `Column.gene` will indicate the number of the column containing gene names. For example, in the previous dataset, the function **DATAprepSE()** must be called with the following arguments:

```
resSEexample <- DATAprepSE(RawCounts=RawCounts_Leong2014_FISSIONsub500wt,
                           Column.gene=1,
                           Group.position=NULL,
                           Time.position=2,
                           Individual.position=3)
```

Here, the argument `Column.gene=1` means that the first column of the dataset contain genes name, `Time.position=2` means that the time of measurements is between the first and the second underscores in the columns names, `Individual.position=3` means that the name of the individual is between the second and the third underscores in the columns names and `Group.position=NULL` means that there is only one biological condition in the dataset (corresponding to **case 2**). Similarly, `Time.position=NULL` would mean that there is only one time of measurement for each individual (corresponding to **case 2**) and `Column.gene=NULL` would mean that there is no column containing gene names.

2.5 Structure of the output

The output of the main functions (see Section 2.2) is a *SummarizedExperiment* class object. The following lines of this section and Figure 4 come from the vignette of the bioconductor package SummarizedExperiment.

The *SummarizedExperiment* class is used to store rectangular matrices of experimental results, which are commonly produced by sequencing experiments such as RNA-Seq. Each *SummarizedExperiment* object stores

- the experiment data `SummarizedExperiment` (RNAseq raw counts data, normalized data ...) which can be retrieved with the R function `SummarizedExperiment::assays()`
- information and features of samples (phenotypes for instance) which can be retrieved with the R function `SummarizedExperiment::colData()`
- information and features of genes (length of genes, gene ontology, DE genes ...) which can be retrieved with the R function `SummarizedExperiment::rowData()`
- Meta-data (any other kind of information). `S4Vectors::metadata()` is just a simple list, so it is appropriate for any experiment wide metadata the user wishes to save, such as storing model formulas, description of the experimental methods, publication references ...

We illustrate the main functions of the package and how to recover the different outputs of the package from the *SummarizedExperiment* object int the following sections.

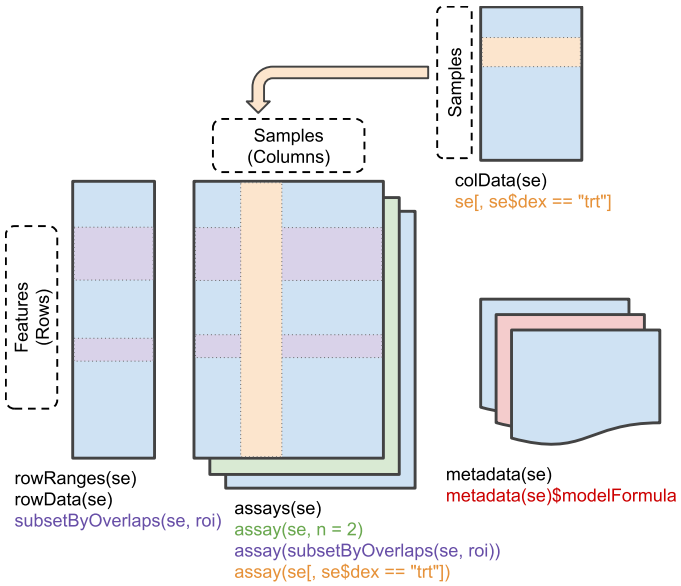


Figure 4: Summarized Experiment structure

3 Detailed analysis of a dataset with several times point and two biological conditions (case 3)

In this section we use the Chronic lymphocytic leukemia (CLL) subdataset **RawCounts_Schleiss2021_CLLsub500** (see subsection Example of MultiRNAflow in case 3, several time points and two biological conditions: Leukemia dataset) in order to explain the use of our package in **case 3** when there are only two biological conditions.

3.1 Preprocessing step with DATAprepSE()

The preprocessing step is realized by our R function **DATAprepSE()** to store all information about the dataset in a standardized way (*SummarizedExperiment* class object). The user must realize this step in order to realize exploratory data analysis (unsupervised analysis, section 3.2) or statistical analysis of the transcriptional response (supervised analysis, section 3.3).

The following lines of code realize the preprocessing step.

```
SEresleuk500 <- DATAprepSE(RawCounts=RawCounts_Schleiss2021_CLLsub500,
                           Column.gene=1,
                           Group.position=2,
                           Time.position=4,
                           Individual.position=3,
                           VARfilter=0,
                           SUMfilter=0,
                           RNAlength=NULL)
```

The inputs `VARfilter` and `SUMfilter` allow to filter the dataset by keeping only rows (i.e. genes) such as the sum or the variances of counts is greater than the selected threshold. The user can also filter genes by keeping only those which have a known transcript length with the input `RNAlength`.

The function returns a *SummarizedExperiment* class object containing

- general information about the dataset
- information to be used for exploratory data analysis
- a *DESeqDataSet* class object (`DESeq2obj`) to be used for statistical analysis of the transcriptional response.

```
names(S4Vectors::metadata(SEresleuk500))
```

```
## [1] "RAWcolnames"      "colGene"          "colDataINFO"
## [4] "RNAfiltering"     "DESeq2obj"        "Results"
## [7] "SEidentification"
```

```
str(S4Vectors::metadata(SEresleuk500)$Results)
```

```
## List of 2
##  $ UnsupervisedAnalysis:List of 5
##   ..$ Normalization  : NULL
##   ..$ PCA            : NULL
##   ..$ HCPC           : NULL
```

```
##   ..$ Mfuzz          : NULL
##   ..$ GenesExpression: NULL
##  $ SupervisedAnalysis  :List of 5
##   ..$ Normalization : NULL
##   ..$ DEanalysis    : NULL
##   ..$ VolcanoMAplots: NULL
##   ..$ Heatmaps      : NULL
##   ..$ Rgprofiler2   : NULL
```

All results of the different analysis presented below will be stored in `Results` of `S4Vectors::metadata()`. We will show in each section how to retrieve information and plots.

Write `?DATAprepSE` in your console for more information about the function.

3.2 Exploratory data analysis (unsupervised analysis)

3.2.1 Normalization with DATAnormalization()

The following lines of code realize the normalization step from the results of the function **DATAprepSE()** (subsection 3.1)

```
SEresNORMleuk500 <- DATAnormalization(SEres=SEresleuk500,
                                      Normalization="vst",
                                      Blind.rlog.vst=FALSE,
                                      Plot.Boxplot=FALSE,
                                      Colored.By.Factors=TRUE,
                                      Color.Group=NULL,
                                      path.result=NULL)
```

If `Plot.Boxplot=TRUE` a boxplot showing the distribution of the normalized expression (`Normalization="vst"` means that the vst method is used) of genes for each sample is returned. If the user gives information about transcript length with the input `RNAlength` of the function **DATAprepSE** (section 3.1), the user can set `Normalization="rpkm"` to normalize the dataset with the RPKM formula.

If the user wants to see the results of the normalization, he must first executing the following lines of code.

```
## Save 'Results' of the metadata in an object
resleuk500 <- S4Vectors::metadata(SEresNORMleuk500)$Results

## Save the results of Normalization in an object
resNORMleuk500 <- resleuk500[[1]][[1]]
###
names(S4Vectors::metadata(SEresNORMleuk500))

## [1] "RAWcolnames"      "colGene"          "colDataINFO"
## [4] "RNAfiltering"     "DESeq2obj"        "Results"
## [7] "SEidentification"

str(resleuk500, max.level=2)

## List of 2
##  $ UnsupervisedAnalysis:List of 5
```

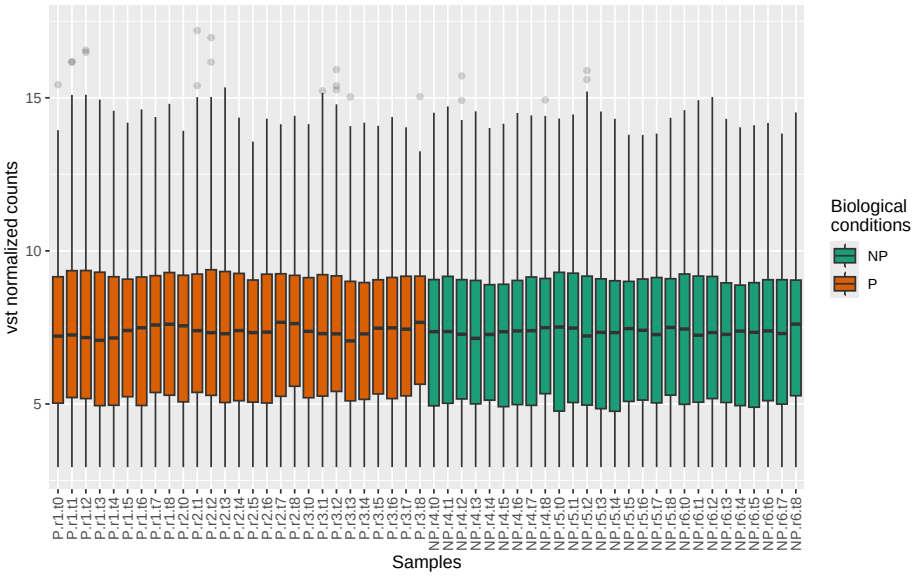
```
##   ..$ Normalization  :List of 2
##   ..$ PCA            : NULL
##   ..$ HCPC           : NULL
##   ..$ Mfuzz          : NULL
##   ..$ GenesExpression: NULL
##  $ SupervisedAnalysis  :List of 5
##   ..$ Normalization : NULL
##   ..$ DEanalysis    : NULL
##   ..$ VolcanoMAplots: NULL
##   ..$ Heatmaps      : NULL
##   ..$ Rgprofiler2   : NULL
```

The user can see that `UnsupervisedAnalysis$Normalization` is no longer `NULL` and now contains two elements : the method of normalization used (`resNORMleuk500$normMethod`) and the boxplot (`resNORMleuk500$normBoxplot`). Boxplots showing the results of normalization can be plotted as follows.

```
resNORMleuk500$normMethod
```

```
## [1] "vst"
```

```
print(resNORMleuk500$normBoxplot)
```



The user can now have access to the normalized data.

```
normData <- SummarizedExperiment::assays(SEresNORMleuk500)
names(SummarizedExperiment::assays(SEresNORMleuk500))
```

```
## [1] "counts" "vst"
```

"counts" represents the raw counts data and "vst" the normalized data. The user can look at the normalized data by executing the line `normData$vst` or `normData[[2]]`.

If `Colored.By.Factors=TRUE`, the color of the boxplots would be different for different biological conditions. By default (if `Color.Group=NULL`), a color will be automatically applied for each biological condition. You can change the colors by creating the following data.frame

```
colorLeuk <- data.frame(Name=c("NP", "P"),
                        Col=c("black", "red"))
```

and setting `Color.Group=colorLeuk`.

The x-labels give biological information, time information and individual information separated by dots. If the user wants to see the 6th first rows of the normalized data, he can write in his console
`head(SEresNORMleuk500$NormalizedData, n=6)`.

The user can save the graph in a folder thanks to the input path.result. If `path.result=NULL` the results will still be plotted but not saved in a folder.

Write `?DATAnormalization` in your console for more information about the function.

Interpretation of the results: When data are not normalized, boxplots of unnormalized log expression data show large differences in distribution between different samples.
The figure shows that the normalized gene expression distribution of the different samples is similar. Normalization has therefore been successful, and we can now begin exploratory analysis of the dataset.

3.2.2 Factorial analysis: PCA with PCAanalysis() and clustering with HCPCanalysis()

3.2.2.1 PCA (case 3)

When samples belong to different biological conditions and different time points, the following lines of code return from the results of the function **DATAnormalization()** (Section 3.2.1):

- The results of the R function `PCA()` from the package FactoMineR.
- one 2D PCA graph, one 3D PCA graph and the same 3D PCA graph in a rgl window (only if `motion3D=FALSE`) where samples are colored with different colors for different biological conditions. Furthermore, lines are drawn between each pair of consecutive points for each individual (if `Mean.Accross.Time=FALSE`, otherwise lines will be drawn only between mean values of all individuals for each time point and biological conditions).
- one 2D PCA graph, one 3D PCA graph and the same 3D PCA graph in a rgl window (only if `motion3D=FALSE`) for each biological condition, where samples are colored with different colors for different time points. Furthermore, lines are drawn between each pair of consecutive points for each sample (if `Mean.Accross.Time=FALSE`, otherwise lines will be drawn only between mean values of all individuals for each time point and biological conditions).
- the same graphs described above but without lines.

```
SEresPCALeuk500 <- PCAanalysis(SEresNorm=SEresNORMleuk500,
                               gene.deletion=NULL,
                               sample.deletion=NULL,
                               Plot.PCA=FALSE,
```

```
                                    Mean.Accross.Time=FALSE,
                                    Color.Group=NULL,
                                    Cex.label=0.9, Cex.point=0.8, epsilon=0.2,
                                    Phi=25, Theta=140,
                                    motion3D=FALSE,
                                    path.result=NULL, Name.folder.pca=NULL)
```

The graphs are

- stored in an SE object (see below how to visualize the elements)
- displayed if `Plot.PCA=TRUE`
- saved in a folder if the user selects a folder path in `path.result`. If `path.result=NULL` the results will not be saved in a folder.

If the user wants to see the results of the PCA analysis, he must first execute the following lines of code.

```
## Save 'Results' of the metadata in an object
resleuk500 <- S4Vectors::metadata(SEresPCALeuk500)$Results

## Save the results of normalization in an object
resPCALeuk500 <- resleuk500[[1]][[2]]
###
names(S4Vectors::metadata(SEresPCALeuk500))
```

```
## [1] "RAWcolnames"      "colGene"          "colDataINFO"
## [4] "RNAfiltering"     "DESeq2obj"        "Results"
## [7] "SEidentification"
```

```
str(resleuk500, max.level=2, give.attr=FALSE)
```

```
## List of 2
##  $ UnsupervisedAnalysis:List of 5
##   ..$ Normalization  :List of 2
##   ..$ PCA            :List of 9
##   ..$ HCPC           : NULL
##   ..$ Mfuzz          : NULL
##   ..$ GenesExpression: NULL
##  $ SupervisedAnalysis  :List of 5
##   ..$ Normalization : NULL
##   ..$ DEanalysis    : NULL
##   ..$ VolcanoMAplots: NULL
##   ..$ Heatmaps      : NULL
##   ..$ Rgprofiler2   : NULL
```

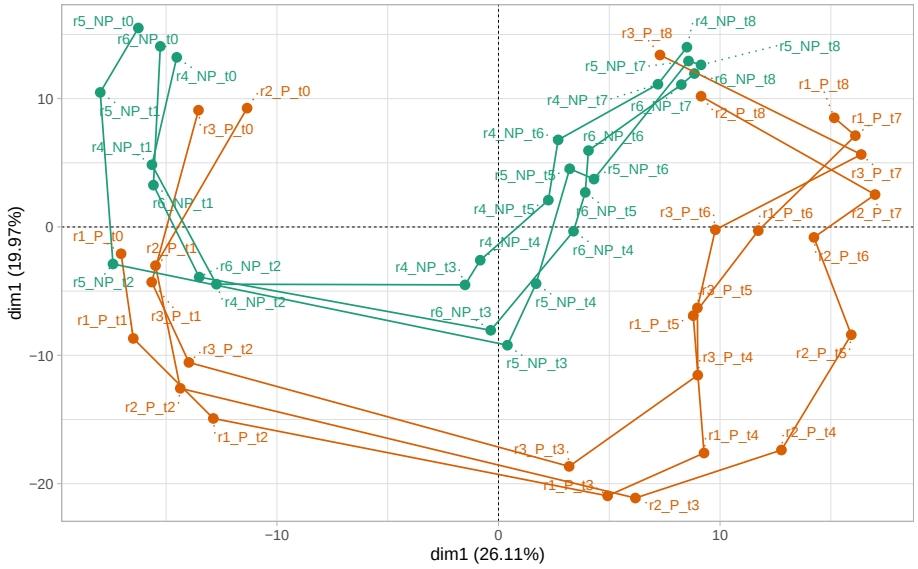
```
names(resPCALeuk500)
```

```
## [1] "nb.quali.var"               "List.Factors"
## [3] "PCAresults"                 "PCA_2D"
## [5] "PCA_3D"                     "PCA_2DtemporalLinks"
## [7] "PCA_3DtemporalLinks"        "PCA_BiologicalCondition_NP"
## [9] "PCA_BiologicalCondition_P"
```

The user can see that `UnsupervisedAnalysis$PCA` is no longer `NULL` and now contains 9 elements :

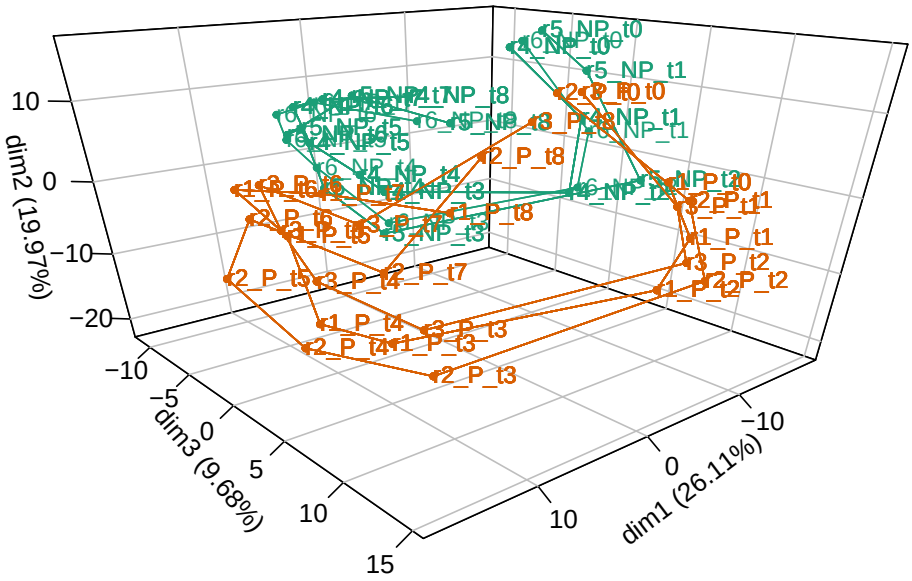
- The results of PCA (`PCAresults`) which can be retrieved by executing the following line of code in your console `resPCALeuk500$PCAresults`
- `resPCALeuk500$PCA_2D` and `resPCALeuk500$PCA_2DtemporalLinks` contain respectively 2D PCA with all biological conditions without and with temporal links. The following lines of codes plot the 2D PCA with temporal links

```
print(resPCALeuk500$PCA_2DtemporalLinks)
```



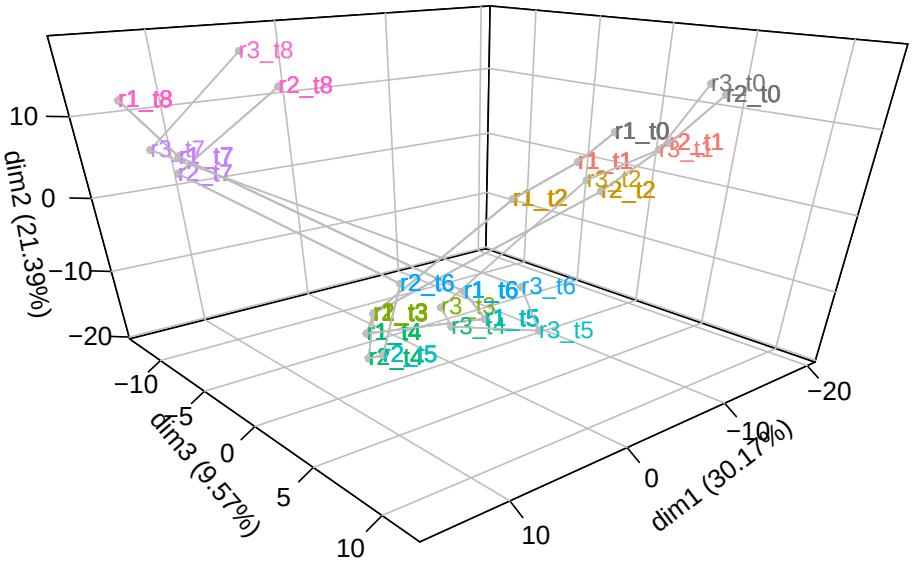
- `resPCALeuk500$PCA_3D` and `resPCALeuk500$PCA_3DtemporalLinks` contain respectively 3D PCA with all biological conditions without and with temporal links. The following lines of codes plot the 3D PCA with temporal links

```
print(resPCALeuk500$PCA_3DtemporalLinks)
```



- `resPCALeuk500$PCA_BiologicalCondition_NP` and `resPCALeuk500$PCA_BiologicalCondition_P` contain the different 2D and 3D PCA plots for each biological condition. The following lines of codes plot the 3D PCA with temporal links returns the 3D PCA with temporal links for the biological condition P.

```
print(resPCALeuk500$PCA_BiologicalCondition_P$PCA_3DtemporalLinks)
```



By default (if `Color.Group=NULL`), a color will be automatically assigned to each biological condition. The user can change the colors by creating the following data.frame

```
colorLeuk <- data.frame(Name=c("NP","P"),
                        Col=c("black","red"))
```

and setting `Color.Group=colorLeuk`. The user cannot change the color associated to each time point.

If the user wants to delete, for instance, the genes 'ABCA7' and 'ADAM28' (respectively the second and sixth gene) and/or delete the samples 'CLL_P_r1_t1' and 'CLL_P_r2_t2', he can set

- `gene.deletion=c("ABCA7","ADAM28")` and/or `sample.deletion=c("CLL_P_r1_t1", "CLL_P_r2_t2")`
- `gene.deletion=c(2, 6)` and/or `sample.deletion=c(3, 13)`.
  The integers in `gene.deletion` and `sample.deletion` represent respectively the row numbers and the column numbers of `RawCounts` where the selected genes and samples are located.

Write `?PCAanalysis` in your console for more information about the function.

Interpretation of the results: The three graphs clearly show that PCA has made it possible to discern the temporal evolution of transcription within each biological condition. The first two graphs also clearly distinguishes the samples according to the biological conditions.

3.2.2.2 HCPC (case 3)

The lines of code below return from the results of the function **DATAnormalization()** (see Section 3.2.1):

- The results of the R function `HCPC()` from the package FactoMineR.
- A dendrogram
- A graph indicating by color for each sample, its cluster, the biological condition and the time point associated to the sample.
- One 2D PCA graph, one 3D PCA graph and the same 3D PCA graph in a rgl window (only if `motion3D=FALSE`). These PCA graphs are identical to the outputs of `PCAanalysis()` with all samples but samples are colored with different colors for different clusters.

The input `SEresNorm` can be either

- `SEresNorm=SEresNORMleuk500` and the results of `HCPCanalysis()` will be added in the SE object which contains the results of `DATAnormalization()`
- or `SEresNorm=SEresPCALeuk500` and the results of `HCPCanalysis()` will be added in the SE object which contains the results of `DATAnormalization()` and `PCAanalysis()`.

```
SEresHCPCLeuk500 <- HCPCanalysis(SEresNorm=SEresPCALeuk500,
                                 gene.deletion=NULL,
                                 sample.deletion=NULL,
                                 Plot.HCPC=FALSE,
                                 Phi=25,Theta=140,
                                 Cex.point=0.7,
                                 epsilon=0.2,
                                 Cex.label=0.9,
                                 motion3D=FALSE,
                                 path.result=NULL,
                                 Name.folder.hcpc=NULL)
```

The graphs are

- stored in an SE object (see below how to visualize the elements)

- displayed if `Plot.HCPC=TRUE`
- saved in a folder if the user selects a folder path in `path.result`. If `path.result=NULL` the results will not be saved in a folder.

If the user wants to see the results of the HCPC analysis, he must first execute the following lines of code.

```
## Save 'Results' of the metadata in an object
resleuk500 <- S4Vectors::metadata(SEresHCPCLeuk500)$Results

## Save the results of HCPC in an object
resHCPCLeuk500 <- resleuk500[[1]][[3]]
###
names(S4Vectors::metadata(SEresHCPCLeuk500))
```

```
## [1] "RAWcolnames"      "colGene"          "colDataINFO"
## [4] "RNAfiltering"     "DESeq2obj"        "Results"
## [7] "SEidentification"
```

```
str(resleuk500, max.level=2, give.attr=FALSE)
```

```
## List of 2
##  $ UnsupervisedAnalysis:List of 5
##   ..$ Normalization  :List of 2
##   ..$ PCA            :List of 4
##   ..$ HCPC           :List of 6
##   ..$ Mfuzz          : NULL
##   ..$ GenesExpression: NULL
##  $ SupervisedAnalysis  :List of 5
##   ..$ Normalization : NULL
##   ..$ DEanalysis    : NULL
##   ..$ VolcanoMAplots: NULL
##   ..$ Heatmaps      : NULL
##   ..$ Rgprofiler2   : NULL
```

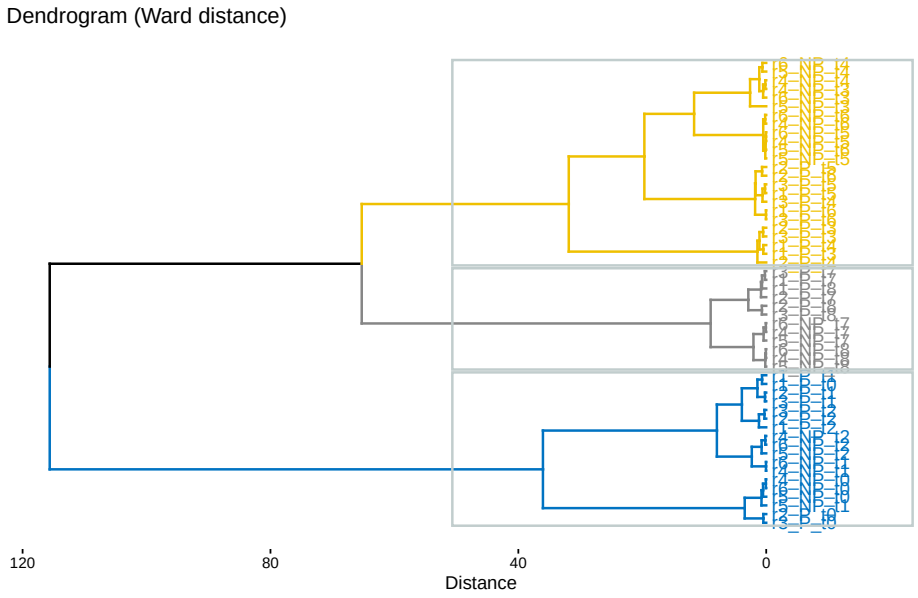
```
names(resHCPCLeuk500)
```

```
## [1] "resHCPC"                   "Samples.FactorCluster"
## [3] "Dendrogram"                "Cluster_SampleDistribution"
## [5] "PCA2DclustersHCPC"         "PCA3DclustersHCPC"
```

The user can see that `UnsupervisedAnalysis$HCPC` is no longer `NULL` and now contains 6 elements :

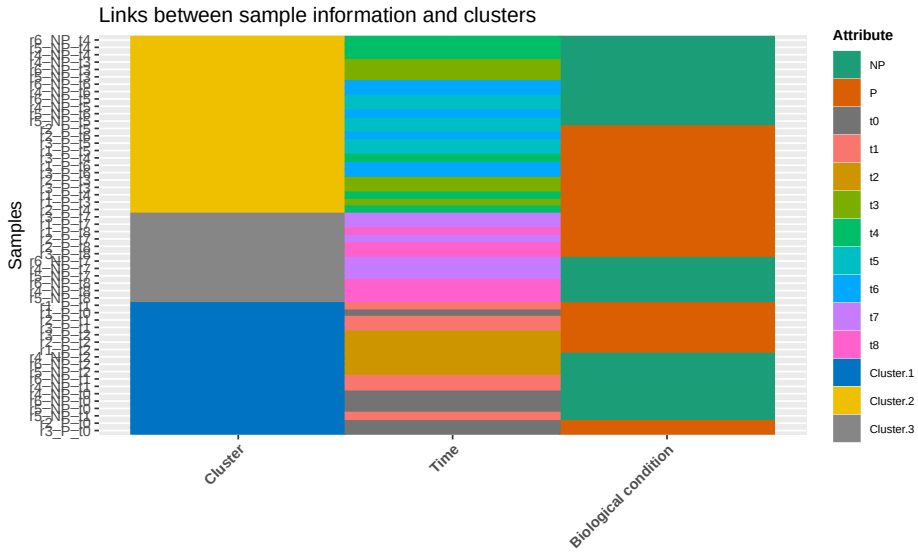
- The results of HCPC (`resHCPC`) can be retrieved by executing the following line of code in your console `resHCPCLeuk500$resHCPC`
- `resHCPCLeuk500$resHCPCLeuk500` contains the dendrogram. The following lines of codes plot the dendrogram.

```
print(resHCPCLeuk500$Dendrogram)
```



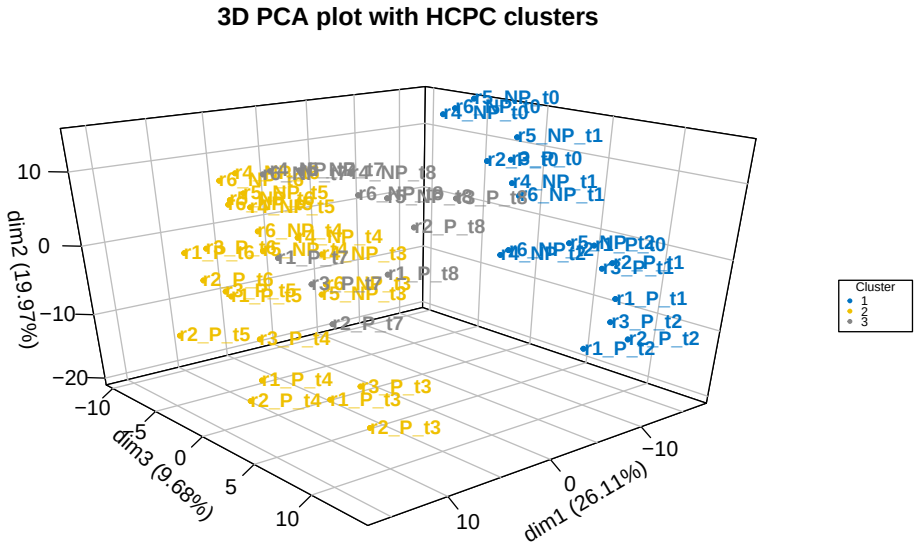
- `resHCPCLeuk500$Cluster_SampleDistribution` contains the graph indicating for each sample, its cluster, the biological condition and the time point associated to the sample, using a color code. The following lines of codes plot the 3D PCA with temporal links

```
print(resHCPCLeuk500$Cluster_SampleDistribution)
```



- `resPCALeuk500$PCA2DclustersHCPC` and `resPCALeuk500$PCA3DclustersHCPC` contain the 2D and 3D PCA plots where samples are colored with different colors for different clusters. The following lines of codes plot the 3D PCA.

```
print(resHCPCLeuk500$PCA3DclustersHCPC)
```



Write `?HCPCanalysis` in your console for more information about the function.

Interpretation of the results: This HCPC analysis shows that

- cluster 1 contains all samples taken at measurement times t_0, t_1 and t_2 of both biological conditions.
- cluster 2 contains all samples taken at measurement times t_3, t_4, t_5 and t_6 of both biological conditions.
- cluster 3 contains all samples taken at measurement times t_7 and t_8 of both biological conditions.

This indicates that the expression of genes at these three groups of times are very different.

3.2.3 Temporal clustering analysis with MFUZZanalysis()

The following function realizes the temporal clustering analysis. It takes as input, a number of clusters (`DataNumberCluster`) that can be chosen automatically if `DataNumberCluster=NULL` and the results of the function **DATAnormalization()** (see Section 3.2.1). The lines of code below return for each biological condition

- the summary of the results of the R function `mfuzz()` from the package Mfuzz.
- the scaled height plot, computed with the `HCPC()` function, and shows the number of clusters chosen automatically (if `DataNumberCluster=NULL`). If `Method="hcpc"`, the function plots the scaled within-cluster inertia, but if `Method="kmeans"`, the function plots the scaled within-cluster inertia. As the number of genes can be very high, we recommend to select `Method="hcpc"` which is by default.
- the output graphs from the R package `Mfuzz` showing the most common temporal behavior among all genes for each biological condition. The plots below correspond to the biological condition 'P'.

The input `SEresNorm` can be either

- `SEresNorm=SEresNORMleuk500` and the results of `MFUZZanalysis()` will be added in the SE object which contains the results of `DATAnormalization()`

- either `SEresNorm=SEresPCALeuk500` and the results of `MFUZZanalysis()` will be added in the SE object which contains the results of `DATAnormalization()` and `PCAanalysis()`
- or `SEresNorm=SEresHCPCLeuk500` and the results of `MFUZZanalysis()` will be added in the SE object which contains the results of `DATAnormalization()`, `PCAanalysis()` and `HCPCanalysis()`.

```
SEresMfuzzLeuk500 <- MFUZZanalysis(SEresNorm=SEresHCPCLeuk500,
                                   DataNumberCluster=NULL,
                                   Method="hcpc",
                                   Membership=0.7,
                                   Min.std=0.1,
                                   Plot.Mfuzz=FALSE,
                                   path.result=NULL,
                                   Name.folder.mfuzz=NULL)
```

```
## 0 genes excluded.
## 6 genes excluded.
## 0 genes excluded.
## 9 genes excluded.
```

The excluded genes are those which standard deviation are under a certain threshold (the R function `mfuzz()` from the package Mfuzz).

The graphs are

- stored in an SE object (see below how to visualize the elements)
- displayed if `Plot.Mfuzz=TRUE`
- saved in a folder if the user selects a folder path in `path.result`. If `path.result=NULL` the results will not be saved in a folder.

If the user wants to see the results of the Mfuzz analysis, he must first execute the following lines of code.

```
## Save 'Results' of the metadata in an object
resleuk500 <- S4Vectors::metadata(SEresMfuzzLeuk500)$Results

## Save the results of Mfuzz in an object
resMfuzzLeuk500 <- resleuk500[[1]][[4]]
###
names(S4Vectors::metadata(SEresMfuzzLeuk500))
```

```
## [1] "RAWcolnames"      "colGene"          "colDataINFO"
## [4] "RNAfiltering"     "DESeq2obj"        "Results"
## [7] "SEidentification"
```

```
str(resleuk500, max.level=2, give.attr=FALSE)
```

```
## List of 2
##  $ UnsupervisedAnalysis:List of 5
##   ..$ Normalization  :List of 2
##   ..$ PCA            :List of 4
##   ..$ HCPC           :List of 6
##   ..$ Mfuzz          :List of 5
##   ..$ GenesExpression: NULL
```

```
##  $ SupervisedAnalysis  :List of 5
##   ..$ Normalization : NULL
##   ..$ DEanalysis    : NULL
##   ..$ VolcanoMAplots: NULL
##   ..$ Heatmaps      : NULL
##   ..$ Rgprofiler2   : NULL
```

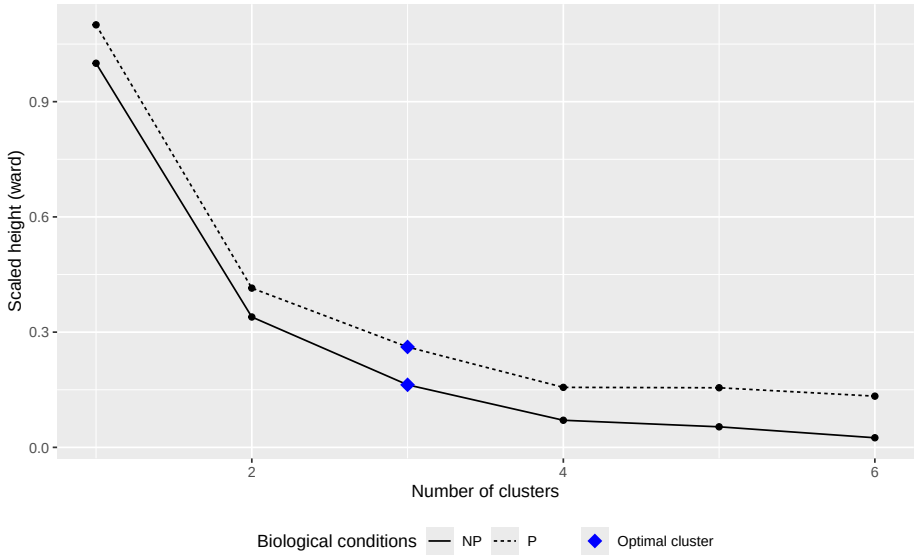
```
names(resMfuzzLeuk500)
```

```
## [1] "Data.Mfuzz"           "Result.Mfuzz"         "ClustersNumbers"
## [4] "Mfuzz.Plots.Group_NP" "Mfuzz.Plots.Group_P"
```

The user can see that `UnsupervisedAnalysis$HCPC` is no longer `NULL` and now contains 6 elements :

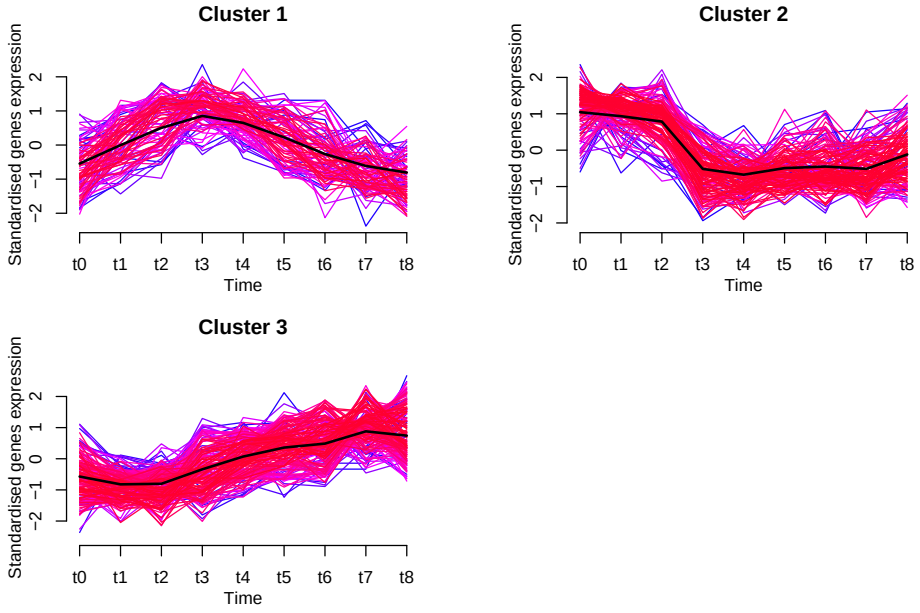
- The user can execute the command `resMfuzzLeuk500$DataClustSel` to see the number of cluster associated to each biological condition.
- The user can execute the command `head(resMfuzzLeuk500$Data.Mfuzz)` in order to see the data used for the Mfuzz analysis, and `head(resMfuzzLeuk500$Result.Mfuzz)` in order to see for each gene, the temporal cluster associated to each biological condition.
- `resMfuzzLeuk500$ClustersNumbers` contains the graph indicating the selected cluster for each biological condition. The following lines of codes plot this graph.

```
print(resMfuzzLeuk500$ClustersNumbers)
```



- `resMfuzzLeuk500$Mfuzz.Plots.Group_NP` and `resMfuzzLeuk500$Mfuzz.Plots.Group_P` contains the different temporal clusters for respectively the biological condition P and NP. The following lines of codes plot the temporal clusters for the biological condition P.

```
print(resMfuzzLeuk500$Mfuzz.Plots.Group_P)
```



Other temporal information are shown in the alluvial graph of the subsection DE analysis with DEanalysisGlobal() that can be compared with the previous graphs.

Write `?MFUZZanalysis` in your console for more information about the function.

Interpretation of the results: We observe that the biological condition P admits at least 3 important clusters. Clusters 2 and 3 are interesting because they show either a significant peak at time t_3 or an abrupt change in behavior. This suggests that cell activation causes a significant change in gene expression at time t_3.

3.2.4 Genes expression profile with DATAplotExpressionGenes()

The lines of code below allow to plot, from the results of the function **DATAnormalization()** (see Section 3.2.1), for each biological condition: the evolution of the 25th gene expression of the three replicates across time and the evolution of the mean and the standard deviation of the 25th gene expression across time. The color of the different lines are different for different biological conditions.

The input `SEresNorm` can be either

- `SEresNorm=SEresNORMleuk500` and the results of `DATAplotExpressionGenes()` will be added in the SE object which contains the results of `DATAnormalization()`
- either `SEresNorm=SEresPCALeuk500` and the results of `DATAplotExpressionGenes()` will be added in the SE object which contains the results of `DATAnormalization()` and `PCA analysis()`
- either `SEresNorm=SEresHCPCLeuk500` and the results of `DATAplotExpressionGenes()` will be added in the SE object which contains the results of `DATAnormalization()`, `PCAanalysis()` and `HCPCanalysis()`.
- or `SEresNorm=SEresMfuzzLeuk500` and the results of `DATAplotExpressionGenes()` will be added in the SE object which contains the results of `DATAnormalization()`, `PCAanalysis()`, `HCPCanalysis()` and `MFUZZanalysis()`.

```
SEresEVOleuk500 <- DATAplotExpressionGenes(SEresNorm=SEresMfuzzLeuk500,
                                           Vector.row.gene=c(25, 30),
                                           Color.Group=NULL,
                                           Plot.Expression=FALSE,
                                           path.result=NULL,
                                           Name.folder.profile=NULL)
```

The graphs are

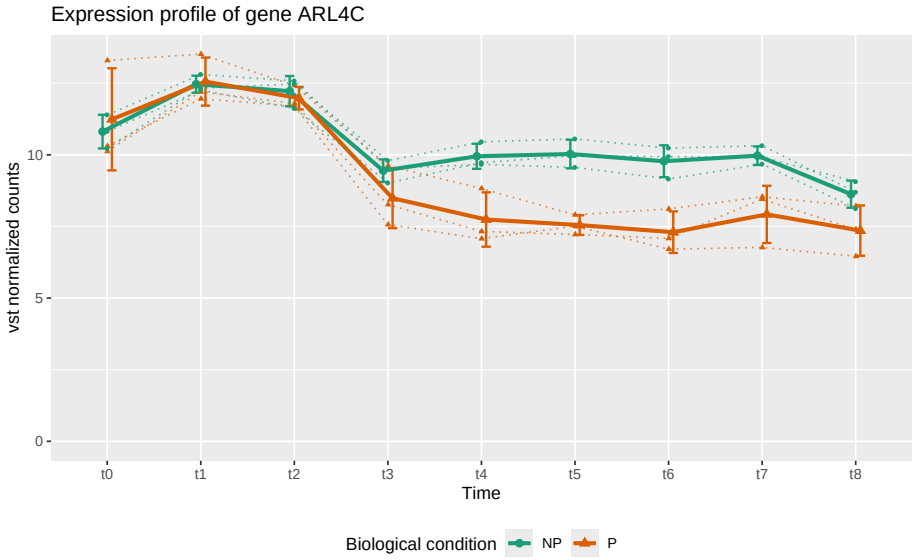
- stored in an SE object (see below how to visualize the elements)
- displayed if `Plot.Expression=TRUE`
- saved in a folder if the user selects a folder path in `path.result`. If `path.result=NULL` the results will not be saved in a folder.

If the user wants to see the results of the DATAplotExpressionGenes(), he must first execute the following lines of code.

```
## Save 'Results' of the metadata in an object
resleuk500 <- S4Vectors::metadata(SEresEVOleuk500)$Results

## Save the results of DE analysis in an object
resEVOleuk500 <- resleuk500[[1]][[5]]
###
names(S4Vectors::metadata(SEresEVOleuk500))

## [1] "RAWcolnames"      "colGene"          "colDataINFO"
## [4] "RNAfiltering"     "DESeq2obj"        "Results"
## [7] "SEidentification"

str(resleuk500, max.level=2, give.attr=FALSE)

## List of 2
##  $ UnsupervisedAnalysis:List of 5
##   ..$ Normalization  :List of 2
##   ..$ PCA            :List of 4
##   ..$ HCPC           :List of 6
##   ..$ Mfuzz          :List of 5
##   ..$ GenesExpression:List of 2
##  $ SupervisedAnalysis  :List of 5
##   ..$ Normalization : NULL
##   ..$ DEanalysis    : NULL
##   ..$ VolcanoMAplots: NULL
##   ..$ Heatmaps      : NULL
##   ..$ Rgprofiler2   : NULL

names(resEVOleuk500)

## [1] "ARL4C_profile"  "ATP5G1_profile"
```

- `resEVOleuk500$ARL4C_profile` or `resEVOleuk500[[1]]` contains the graph or expression profile of gene ARL4C_profile, and `resEVOleuk500$ARL4C_profile` or `resEVOleuk500[[2]]` contains the graph or expression profile of gene ATP5G1_profile

```
print(resEVOleuk500$ARL4C_profile)
```



By default (if `Color.Group=NULL`), a color will be automatically assigned to each biological condition. The user can change the colors by creating the following data.frame

```
colorLeuk <- data.frame(Name=c("NP", "P"), Col=c("black", "red"))
```

and setting `Color.Group=colorLeuk`. If the user wants to select several genes, for instance the 97th, the 192th, the 194th and the 494th, he needs to set `Vector.row.gene=c(97,192,194,494)`. Write `?DATAplotExpressionGenes` in your console for more information about the function.

Interpretation of the results: The expression profile of the ARL4C gene is interesting, since the behavior of this gene is similar in both biological conditions over the first 4 time points (up to t_3). From t_4 to t_8, expression of the normalized gene is much lower in the P biological condition. Stimulation of the P cell membrane therefore cascades to modify the expression of several genes, resulting in inhibition of ARL4C gene expression.

3.3 Statistical analysis of the transcriptional response (supervised analysis)

3.3.1 DE analysis with DEanalysisGlobal()

The lines of code below

- returns a data.frame. See subsection Data.frame summarizing all the DE analysis (case 3)
- plots the following graphs
  - *Results from the temporal statistical analysis* (**case 2** for each biological condition). See subsection Graphs from the results of the temporal statistical analysis
  - *Results from the statistical analysis by biological condition* (**case 1** for each fixed time). See subsection Graphs from the results of the biological condition analysis.

- *Results from the combination of temporal and biological statistical analysis*. See subsection Graphs from the results of the combination of temporal and biological statistical analysis

The input `SEres` can be either

- `SEres=SEresleuk500`, and the results of `DEanalysisGlobal()` will be stored in the initial SE object
- either `SEres=SEresNORMleuk500` and the results of `DEanalysisGlobal()` will be added in the SE object which contains the results of `DATAnormalization()`
- either `SEres=SEresPCALeuk500` and the results of `DEanalysisGlobal()` will be added in the SE object which contains the results of `DATAnormalization()` and `PCAanalysis()`
- either `SEres=SEresHCPCLeuk500` and the results of `DEanalysisGlobal()` will be added in the SE object which contains the results of `DATAnormalization()`, `PCAanalysis()` and `HCPCanalysis()`.
- either `SEres=SEresMfuzzLeuk500` and the results of `DEanalysisGlobal()` will be added in the SE object which contains the results of `DATAnormalization()`, `PCAanalysis()`, `HCPCanalysis()` and `MFUZZanalysis()`.
- or `SEres=SEresEVOleuk500` and the results of `DEanalysisGlobal()` will be added in the SE object which contains the results of `DATAnormalization()`, `PCAanalysis()`, `HCPCanalysis()`, `MFUZZanalysis()` and `DATAplotExpressionGenes()`.

```
SEresDELeuk500 <- DEanalysisGlobal(SEres=SEresEVOleuk500,
                                   pval.min=0.05,
                                   pval.vect.t=NULL,
                                   log.FC.min=1,
                                   LRT.supp.info=FALSE,
                                   Plot.DE.graph=FALSE,
                                   path.result=NULL,
                                   Name.folder.DE=NULL)
```

```
## [1] "Preprocessing"
## [1] "Differential expression step with DESeq2::DESeq()"
## [1] "Case 3 analysis : Biological conditions and Times."
## [1] "DE time analysis for each biological condition."
## [1] "DE group analysis for each time measurement."
## [1] "Combined time and group results."
```

```
## data("Results_DEanalysis_sub500")
## SEresDELeuk500 <- Results_DEanalysis_sub500$DE_Schleiss2021_CLLsub500
```

Due to time consuming of the DE analysis, we stored in the object `Results_DEanalysis_sub500` (uncommented lines) a list of three objects

- `Results_DEanalysis_sub500$DE_Schleiss2021_CLLsub500`, stored the results of `DEanalysisGlobal()` with `RawCounts_Schleiss2021_CLLsub500`.
- `Results_DEanalysis_sub500$DE_Antoszewski2022_MOUSEsub500`, stored the results of `DEanalysisGlobal()` with `RawCounts_Antoszewski2022_MOUSEsub500`.
- `Results_DEanalysis_sub500$DE_Leong2014_FISSIONsub500wt`, stored the results of `DEanalysisGlobal()` with `RawCounts_Leong2014_FISSIONsub500wt`.

The graphs are

- stored in an SE object (see below how to visualize the elements)
- displayed if `Plot.DE.graph=TRUE` (see the following subsections 3.3.1.2, 3.3.1.3 and 3.3.1.4)
- saved in a folder if the user selects a folder path in `path.result`. If `path.result=NULL` the results will not be saved in a folder.

Write `?DEanalysisGlobal` in your console for more information about the function.

3.3.1.1 Data.frame summarizing all the DE analysis (case 3)

The output data.frame can be extracted with the following line of code,

```
DEsummaryLeuk <- SummarizedExperiment::rowData(SEresDELeuk500)
```

As we use abbreviated column names, we propose a glossary in order to help the user to understand meaning of each column. The glossary of the column names can be extracted with the following lines of code,

```
resDELeuk500 <- S4Vectors::metadata(SEresDELeuk500)$Results[[2]][[2]]
resGlossaryLeuk <- resDELeuk500$Glossary
```

and then write `DEsummaryLeuk` and `resGlossaryLeuk` in the R console.

The data.frame `DEsummaryLeuk` contains

- gene names (column 1)
- *Results from the temporal statistical analysis* (**case 2** for each biological condition)
  - pvalues, log2 fold change and DE genes between each time t_i versus the reference time t_0, for each biological condition ($3\times(T-1)\times N_{bc} = 3\times 8\times 2 = 48$ columns).
  - $N_{bc} = 2$ binary columns (1 and 0), one per biological condition (with N_{bc} the number of biological conditions). A '1' in one of these two columns means that the gene is DE at least between one time t_i versus the reference time t_0, for the biological condition associated to the column (see Section 3.3.1.2).
  - $N_{bc} = 2$ columns where each element is succession of 0 and 1, one per biological condition. The positions of '1,' in one of these two columns, indicate the set of times t_i such that the gene is DE between t_i and the reference time t_0, for the biological condition associated to the column. So each element of the column is what we called previously, a temporal pattern.
- *Results from the statistical analysis by biological condition* (**case 1** for each fixed time)
  - pvalues, log2 fold change and DE genes between each pairs of biological conditions, for each fixed time. ($3 \times \frac{N_{bc}\times(N_{bc}-1)}{2} \times T = 3 \times 1 \times 9 = 27$ columns).
  - $T = 9$ binary columns (1 and 0), one per time. A '1' in one of these columns, means that the gene is DE between at least one pair of biological conditions, for the fixed time associated to the column.
  - $N_{bc} \times T = 2 \times 9 = 18$ binary columns, which give the specific genes for each biological condition at each time t_i. A '1' in one of these columns means that the gene is specific to the biological condition at the time associated to the column.

'0' otherwise. A gene is called specific to a given biological condition BC1 at a time t_i, if the gene is DE between BC1 and any other biological conditions at time t_i, but not DE between any pair of other biological conditions at time t_i.

- $N_{bc} \times T = 2 \times 9 = 18$ columns filled with -1, 0 or 1. A '1' in one of these columns means that the gene is up-regulated (or over-expressed) for the biological condition at the time associated to the column. A gene is called up-regulated for a given biological condition BC1 at time t_i if the gene is specific to the biological condition BC1 at time t_i and expressions in BC1 at time t_i are higher than in the other biological conditions at time t_i. A '-1' in one of these columns means that the gene is down-regulated (or under-expressed) for the biological condition at the time associated to the column. A gene is called down-regulated for a given biological condition at a time t_i BC1 if the gene is specific to the biological condition BC1 at time t_i and expressions in BC1 at time t_i are lower than in the other biological conditions at time t_i. A '0' in one of these columns means that the gene is not specific to the biological condition at the time associated to the column.
- $N_{bc} = 2$ binary columns (1 and 0). A '1' in one of these columns means the gene is specific at least at one time t_i, for the biological condition associated to the column. '0' otherwise.

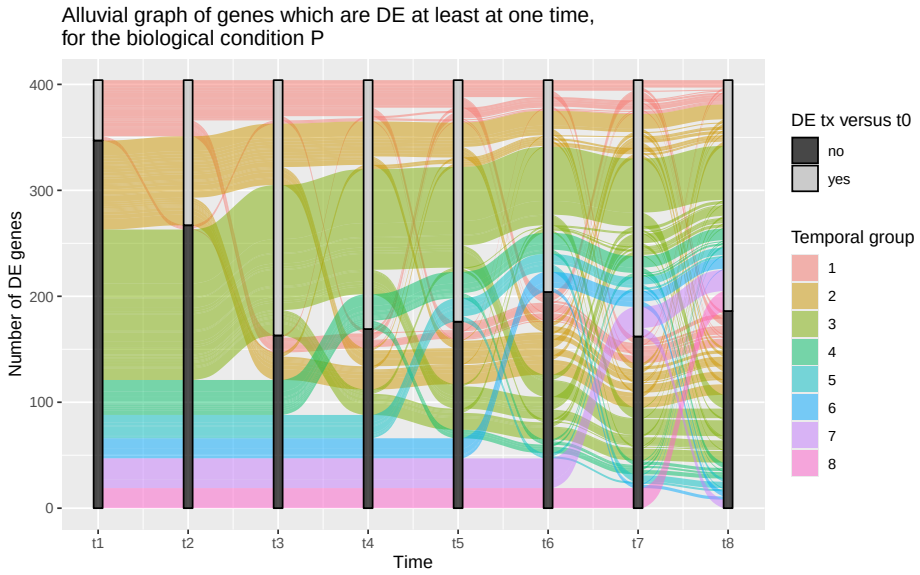
- *Results from the combination of temporal and biological statistical analysis*
  - $N_{bc} \times T = 2 \times 9 = 18$ binary columns, which give the signature genes for each biological condition at each time t_i. A '1' in one of these columns means that the gene is a signature gene to the biological condition at the time associated to the column. '0' otherwise. A gene is called signature of a biological condition BC1 at a given time t_i, if the gene is specific to the biological condition BC1 at time t_i and DE between t_i versus the reference time t_0 for the biological condition BC1.
  - $N_{bc} = 2$ binary columns (1 and 0). A '1' in one of these columns means the gene is signature at least at one time t_i, for the biological condition associated to the column. '0' otherwise.

3.3.1.2 Graphs from the results of the temporal statistical analysis

From the temporal statistical analysis, the user can plot the following graphs.

$N_{bc} = 2$ *alluvial graphs*, one per biological condition (with N_{bc} the number of biological conditions). The code below prints the alluvial graph for the biological condition P.

```
print(resDELeuk500$DEplots_TimePerGroup$Alluvial.graph.Group_P)
```

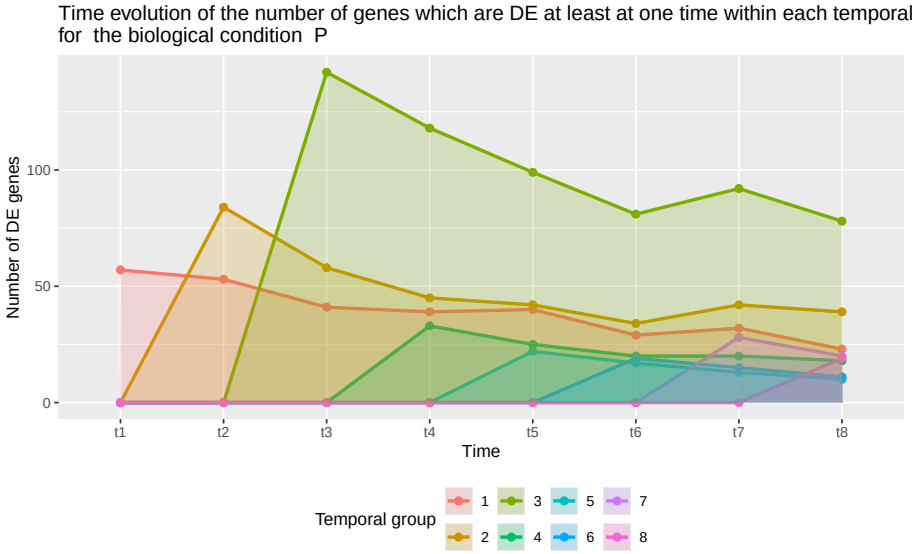


Description The x-axis of the graph is labeled with all times except t_0. For each vertical barplot, there are two strata: 1 and 0 whose sizes indicate respectively the number of DE genes and of non DE genes, between the time corresponding to the barplot and the reference time t_0. The alluvial graph is composed of curves, each corresponding to a single gene, which are gathered in alluvia. An alluvium is composed of all genes having the same curve: for example, an alluvium going from the stratum 0 at time t_1 to the stratum 1 at time t_2 corresponds to the set of genes which are not DE at t_1 and are DE at time t_2. Each alluvium connects pairs of consecutive barplots and its thickness gives the number of genes in the alluvium. The color of each alluvium indicates the temporal group, defined as the set of genes which are all first DE at the same time with respect to the reference time t_0.

Interpretation of the results The alluvial graph can be used to determine the number of DE genes at least at one time for a given biological condition, since this is the size of each barplot. Using these graphs alone (one for each biological condition), we can compare the number of DE genes at least one time for all biological conditions. This graph also provides information on all temporal patterns and also the number of genes present in each of these patterns. According to the graph, the two most important DE temporal patterns for biological condition P are: "00111111" and "00000011". This allows us to deduce that times t_3 and t_7 are very important. This can also be visualized by the size of each alluvium at time t_1 which gives the number of genes in each time group.

$N_{bc} = 2$ *graphs* showing the number of DE genes as a function of time for each temporal group, one per biological condition. By temporal group, we mean the sets of genes which are first DE at the same time. The code below prints the graph for the biological condition P.

```
print(resDELeuk500$DEplots_TimePerGroup$NumberDEgenes.acrossTime.perTemporalGroup.Group_P)
```

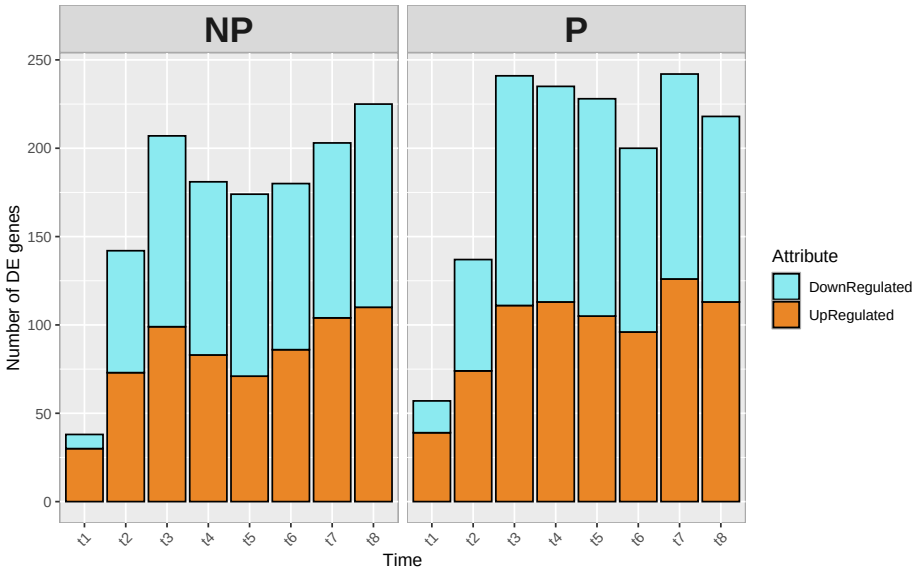


Description The graphs plot the time evolution of the number of DE genes within each temporal group, one per biological condition. The x-axis labels indicate all times except t_0.

Interpretation of the results This graph confirms the importance of time groups t_3 and t_7. Genes belonging to time groups t_1 and t_2 seem to correspond to genes upstream of the activation cascade after stimulation of the cell membrane, and have a direct impact on hub genes which activate numerous biological pathways. This would explain the very large number of genes belonging to the time group t_3. Time t_7 corresponds to genes downstream of the activation cascade, and is probably largely made up of genes with a direct or indirect role in cell proliferation. This can be checked using our GO tools (see Section 3.4).

One barplot showing the number of DE genes up-regulated and down-regulated per time, for each biological condition. The graph shows the number of DE genes per time, for each biological condition. The x-axis labels indicate all times except t_0. The code below prints the barplot.

```
print(resDELeuk500$DEplots_TimePerGroup$NumberDEgenes_UpDownRegulated_perTimeperGroup)
```

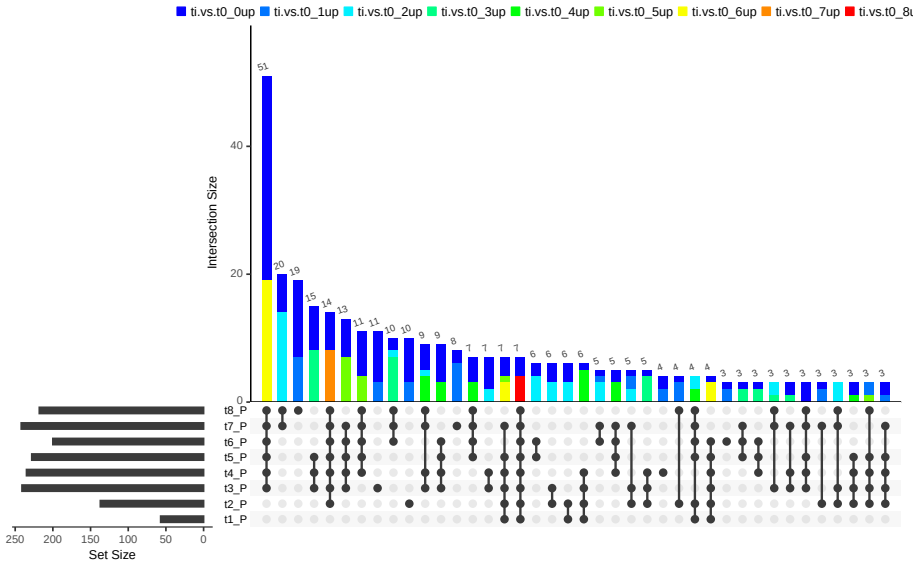


Description For each DE gene, we compute the sign of the log2 fold change between time t_i and time t_0. If the sign is positive (resp. negative), the gene is categorized as up-regulated (resp. down-regulated). In the graph, the up-regulated (resp. down-regulated) genes are indicated in orange (resp. in light blue).

Interpretation of the results This graph shows that the number of DE genes is greater in biological condition P at all times, especially at times t_1, t_3 and t_7. This means that in many biological pathways, there are far more genes activated or inhibited in the biological condition P, and these are the genes that interest biologists.

$2 \times N_{bc} = 4$ *upset graphs*, realized with the R package UpSetR [33], showing the number of DE genes belonging to each DE temporal pattern, for each biological condition. By temporal pattern, we mean the set of times t_i such that the gene is DE between t_i and the reference time t_0.

```
print(resDELeuk500$DEplots_TimePerGroup$VennBarplot.withNumberUpRegulated.Group_P)
```



Description The graphs plot the number of genes in each DE temporal pattern in a Venn barplot, one per biological condition. By DE temporal pattern, we mean a subset of times in $t_1, \ldots, t_n$. We say that a gene belongs to a DE temporal pattern if the gene is DE versus t_0 only at the times in this DE temporal patterns. For each gene in a given DE temporal pattern, we compute the number of DE times where it is up-regulated and we use a color code in the Venn barplot to indicate the number of genes in a temporal pattern that are up-regulated a given number of times (dark blue if it is always down-regulated, lighter blue if it is up-regulated only once, *etc*). For example,

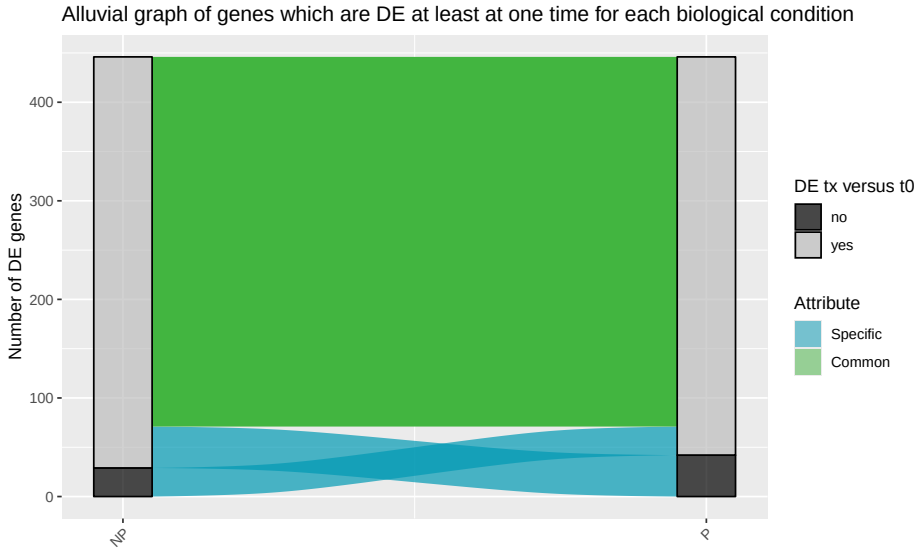
- the size of the dark blue barplot of the first barplot ("ti.vs.t0_0up") gives the number of genes which are DE between t_3 versus t_0, t_4 versus t_0, ..., t_7 versus t_0 and t_8 versus t_0 and always down-regulated at each of this six times compared to t_0
- the size of the light blue barplot of the first ninth ("ti.vs.t0_2up") gives the number of genes which are DE between t_6 versus t_0, t_7 versus t_0 and t_8 versus t_0 and up-regulated at only two times among t_6, t_7 and t_8 compared to t_0.

The same graph is also given without colors with the R command `print(resDELeuk500$DEplots_TimePerGroup$VennBarp`

Interpretation of the results This graph identifies the most important temporal patterns for each biological condition and shows whether the genes in these temporal patterns are over- or under-expressed. In our case, the graph confirms that the two most important temporal patterns for biological condition P are: "00111111" and "00000011". Moreover, whatever the temporal pattern, almost all genes are either always over-expressed compared to t_0 or always under-expressed compared to t_0.

One alluvial graph, for DE genes which are DE at least at one time for each biological condition

```
print(resDELeuk500$DEplots_TimePerGroup$AlluvialGraph_DE.1tmin_perGroup)
```



Description The alluvial graph is quite simple here because there are only 2 biological conditions. It shows common and no common genes between which are DE at least at one time for the two biological conditions.

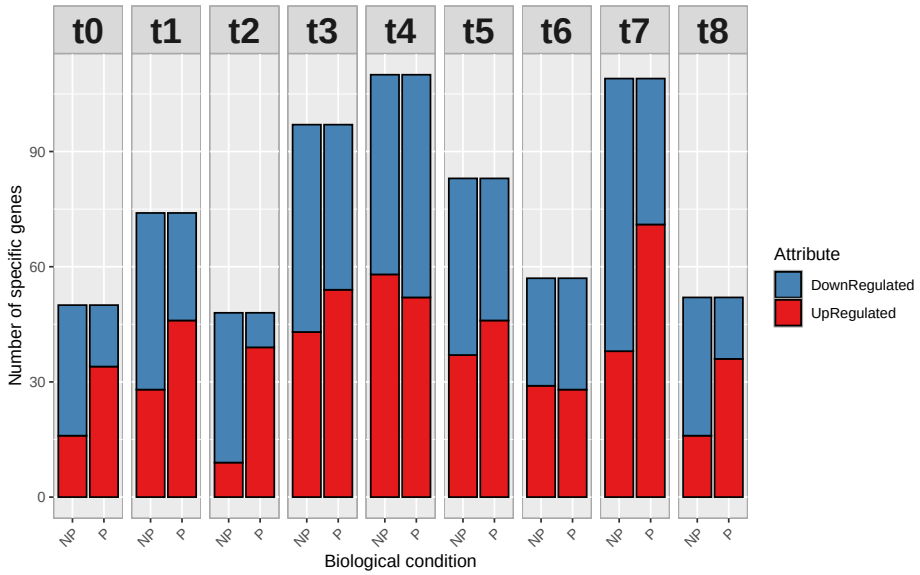
Interpretation of the results This graph shows that majority of DE genes are common between both biological condition.

3.3.1.3 Graphs from the results of the biological condition analysis

From the statistical analysis by biological condition, the function plots the following graphs.

One barplot showing the number of specific genes per biological condition, for each time.

```
print(resDELeuk500$DEplots_GroupPerTime$NumberSpecificGenes_UpDownRegulated_perBiologicalCondition)
```



Description A gene is called specific to a given biological condition BC1 at a time t_i, if the gene is DE between BC1 and any other biological conditions at time t_i, but not DE between any pair of other biological conditions at time t_i. If the levels expression of a specific genes of the biological condition BC1 ii higher (resp. lower) in BC1 than the other biological conditions then the gene is categorized as specific up-regulated (resp. down-regulated). In the graph, the up-regulated (resp. down-regulated) genes are indicated in red (resp. in blue). As there are only two biological conditions here, the number of DE genes is equal to the number of specific genes and the number of specific up-regulated (resp. down-regulated) genes in condition P at any time t_i is equal to the number of specific down-regulated (resp. up-regulated) genes in condition NP at the same time t_i.

Interpretation of the results We can see that the times when the number of specific DE genes is highest are t_3, t_4 and t_7. This means that it is at these times that real differences appear between P and NP cells. What's also interesting is the high proportion of over-expressed genes in the P biological condition at most measurement times, implying that cell proliferation is more impacted by "stimulatory" biological pathways than by "inhibitory" ones.

alluvial graph showing the number of DE genes which are specific at least at one time for each group, plotted only if there are more than two biological conditions (which is not the case here).

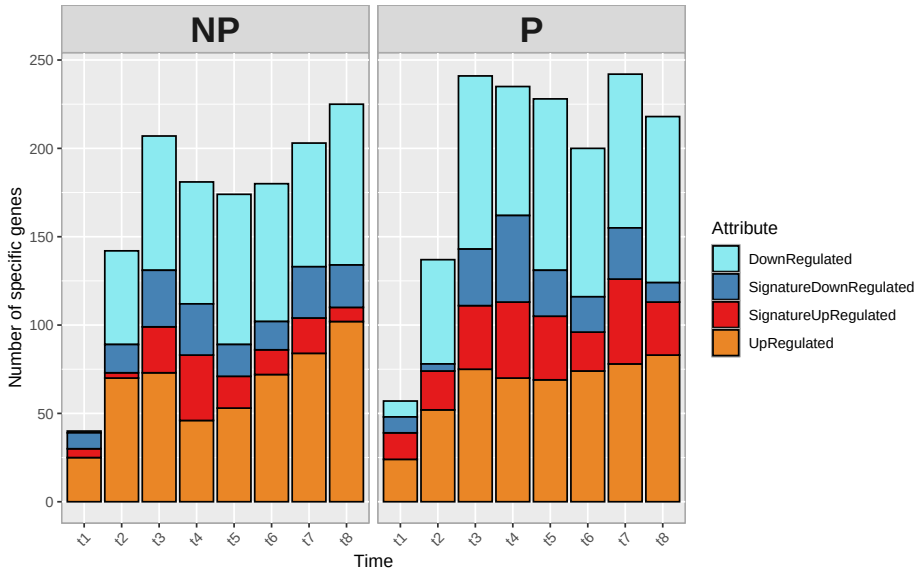
$\binom{N_{bc}}{2} = \binom{4}{2} = 6$ *upset graph* showing the number of genes corresponding to each possible intersection in a Venn barplot at a given time, plotted only if there are more than two biological conditions (which is not the case here). We recall that a set of pairs of biological conditions forms an intersection at a given time t_i when there is at least one gene which is DE for each of these pairs of biological conditions at time t_i, but not for the others at time t_i

3.3.1.4 Graphs from the results of the combination of temporal and biological statistical analysis

From the combination of temporal and biological statistical analysis, the function plots the following graphs.

One barplot showing the number of signature genes and DE genes (but not signature) per time, for each biological condition.

```
print(resDELeuk500$DEplots_TimeAndGroup$Number_DEgenes_SignatureGenes_UpDownRegulated_perTimeperGroup)
```



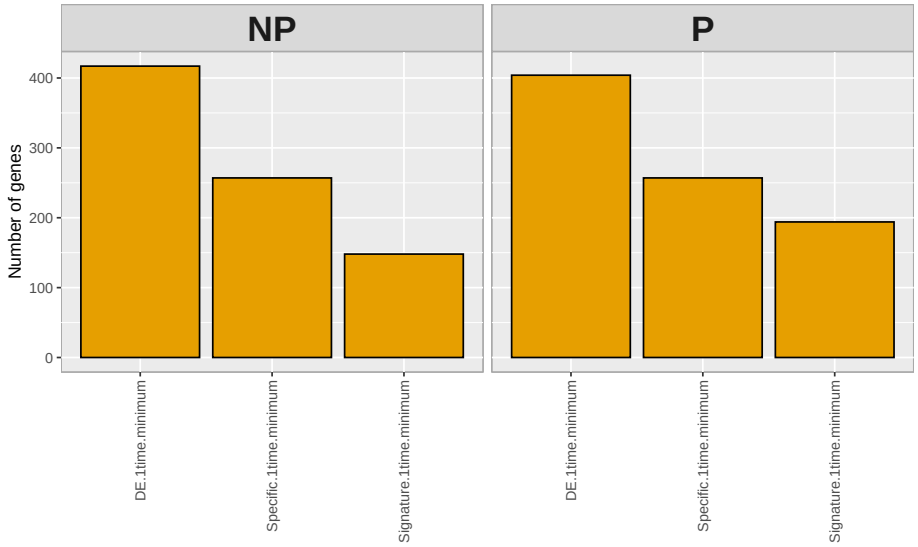
Description A gene is called signature of a biological condition BC1 at a given time t_i, if the gene is specific to the biological condition BC1 at time t_i and DE between t_i versus the reference time t_0 for the biological condition BC1. For each DE gene, we compute the sign of the log2 fold change between time t_i and time t_0.

- If the sign is positive (resp. negative) for the biological condition P (resp. NP) and the gene is specific to P (resp. NP) then the gene is categorized as signature up-regulated (resp. down-regulated) to the biological condition P (resp. NP). In the graph, the up-regulated (resp. down-regulated) genes are indicated in red (resp. in blue).
- If the sign is positive (resp. negative) for the biological condition P (resp. NP) and the gene is not specific to P (resp. NP) then the gene is categorized as up-regulated (resp. down-regulated) to the biological condition P (resp. NP). In the graph, the up-regulated (resp. down-regulated) genes are indicated in orange (resp. in light blue).

Interpretation of the results The times when the number of DE genes (here it is the same as specific) is highest are t_3, t_4 and t_7. This means that it is at these times that real differences appear between P and NP cells. What's also interesting is the high proportion of over-expressed genes in the P biological condition at most measurement times, implying that cell proliferation is more impacted by "stimulatory" biological pathways than by "inhibitory" ones.

One barplot showing the number of genes which are DE at least at one time, specific at least at one time and signature at least at one time, for each biological condition.

```
print(resDELeuk500$DEplots_TimeAndGroup$Number_DEgenes1TimeMinimum_Specific1TimeMinimum_Signature1TimeMinimum
```



Interpretation of the results The barplot summarizes well the combination of temporal and biological analysis because the figure gives for each biological condition the number of genes which are DE at least at one time, the number of specific genes at least at one time and the number of signature genes at least at one time.

One alluvial graph for DE genes which are signature at least at one time for each biological condition, only if there are more than two biological conditions (which is not the case here).

3.3.2 Volcano plots, ratio intensity (MA) plots and Heatmaps with DEplotVolcanoMA() and DEplotHeatmaps()

3.3.2.1 Volcano and MA plots (case 3)

The following lines of code allow to plot

- $\binom{N_{bc}}{2} \times T + (T-1) \times N_{bc} = \frac{N_{bc}(N_{bc}-1)}{2} \times T + (T-1) \times N_{bc} = 25$ volcano plots (with $N_{bc} = 2$ the number of biological conditions and $T = 9$ the number of time points).
- $\binom{N_{bc}}{2} \times T + (T-1) \times N_{bc} = 25$ MA plots.

allowing to separate non DE genes, DE genes below a threshold of log2 fold change and DE genes above a threshold of log2 fold change.

```
SEresVolcanoMAleuk <- DEplotVolcanoMA(SEresDE=SEresDELeuk500,
                                      NbGene.plotted=2,
                                      SizeLabel=3,
                                      Display.plots=FALSE,
                                      Save.plots=FALSE)
```

If the user wants to save the graphs, the input `Save.plots` must be

- either `Save.plots=TRUE`, and the graph will be saved in the same location than the input `path.result` of the function `DEanalysisGlobal()`.
- either a strings of characters giving the path to a folder where the graphs will be saved. The user then chooses the path of the folder where results can be saved.

If the user wants to display the graph, he must set `Display.plots=TRUE`.

If the user wants to see the results of the **DEplotVolcanoMA**, he must first execute the following lines of code.

```
## Save 'Results' of the metadata in an object
resleuk500 <- S4Vectors::metadata(SEresVolcanoMAleuk)$Results

## Save the results of DEplotVolcanoMA in an object
resVolMAleuk500 <- resleuk500[[2]][[3]]
###
names(S4Vectors::metadata(SEresVolcanoMAleuk))
```

```
## [1] "RAWcolnames"      "colGene"          "colDataINFO"
## [4] "RNAfiltering"     "DESeq2obj"        "Results"
## [7] "SEidentification"
```

```
str(resleuk500, max.level=2, give.attr=FALSE)
```

```
## List of 2
##  $ UnsupervisedAnalysis:List of 5
##   ..$ Normalization  :List of 2
##   ..$ PCA            :List of 4
##   ..$ HCPC           :List of 6
##   ..$ Mfuzz          :List of 5
##   ..$ GenesExpression:List of 2
```

```
##  $ SupervisedAnalysis  :List of 5
##   ..$ Normalization :List of 2
##   ..$ DEanalysis    :List of 4
##   ..$ VolcanoMAplots:List of 3
##   ..$ Heatmaps      : NULL
##   ..$ Rgprofiler2   : NULL
```

```
names(resVolMAleuk500)
```

```
## [1] "highest2DEgenes" "Volcano"         "MAplots"
```

`resVolMAleuk500$Volcano` and `resVolMAleuk500$MAplots` contain the volcano and MA plots

- between each time t_i versus the reference time t_0 for each biological condition,
- between each pair of biological condition for each time point,

as can be seen with the following line of code

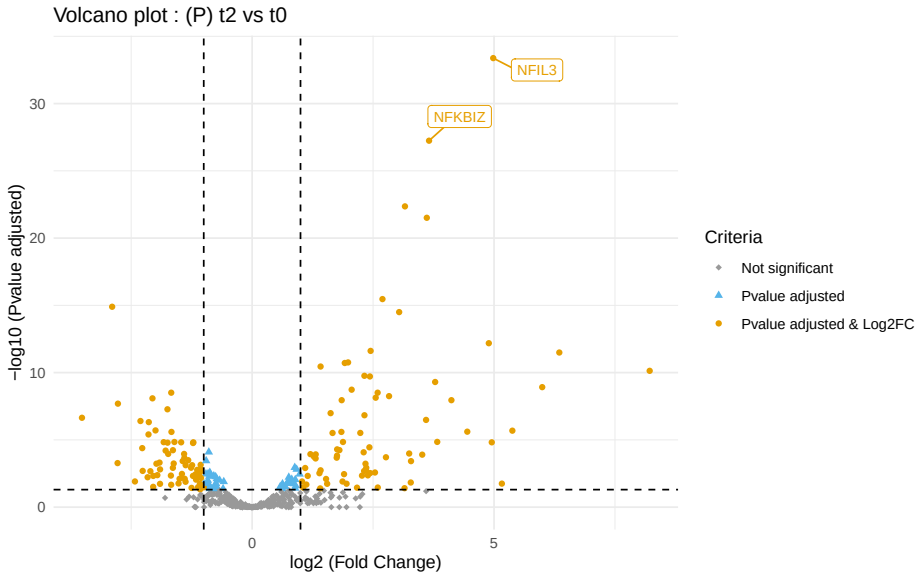
```
str(resVolMAleuk500$Volcano, max.level=1, give.attr=FALSE)
```

```
## List of 11
##  $ NP:List of 8
##  $ P :List of 8
##  $ t0:List of 1
##  $ t1:List of 1
##  $ t2:List of 1
##  $ t3:List of 1
##  $ t4:List of 1
##  $ t5:List of 1
##  $ t6:List of 1
##  $ t7:List of 1
##  $ t8:List of 1
```

and `resVolMAleuk500$highest2DEgenes` contains the `NbGene.plotted` most important genes for each volcano and MA plot. If we call PV_g and FC_g respectively the p-value and log fold change of a gene g for a given volcano plot, then the most important genes for each volcano and MA plot are those which maximize $PV_g^2 + FC_g^2$.

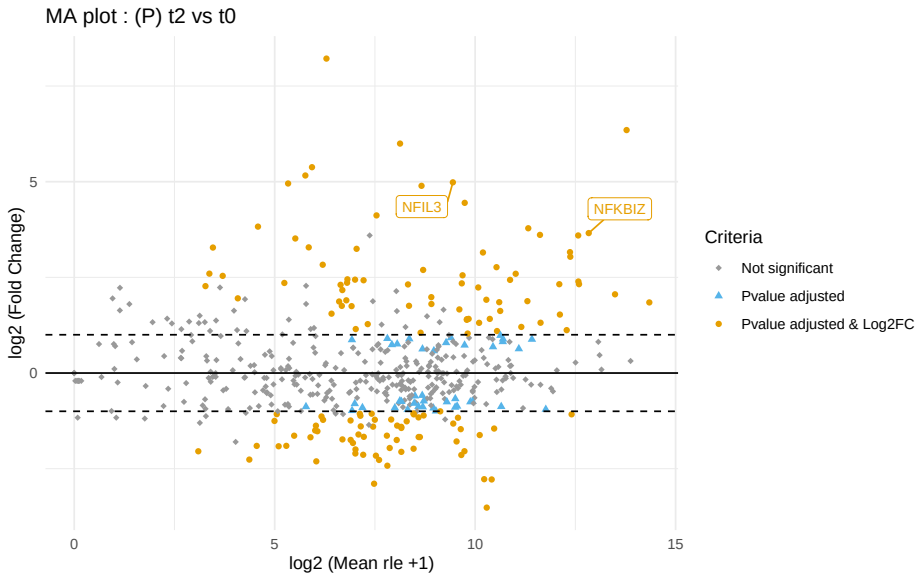
The following lines plot the volcano plot for the biological condition P between t_2 and t_0

```
print(resVolMAleuk500$Volcano$P$P_t2_vs_t0)
```



and the following lines plot the MA plot for the biological condition P between t_2 and t_0.

```
print(resVolMAleuk500$MA$P$P_t2_vs_t0)
```



Interpretation of the results of volcano plot The most over-expressed (resp. under-expressed) genes are to the right (resp. left) of the volcano plot, and the most significant (resp. least significant) genes are at the top (resp. bottom) of the volcano plot. The volcano plot thus enables rapid visual identification of the "best" DE genes, which should have the lowest possible pvalue and the highest possible absolute log2 fold change.

Interpretation of the results of MA plot Highly expressed genes are to the right of the graph, and genes at the top (resp. bottom) of the graph are the most over-expressed (resp. under-expressed). The "interesting" genes are therefore at the top right and bottom right.

Write `?DEplotVolcanoMA` in your console for more information about the function.

3.3.2.2 Heatmaps (case 3)

The following lines of code allow to plot a correlation heatmap between samples (correlation heatmap) and a heatmap across samples and genes called Zscore heatmap, for a subset of genes that can be selected by the user. The second heatmap is build from the normalized count data after being both centered and scaled (Zscore).

The input `SEresDE` can be either

- `SEresDE=SEresDELeuk500`, and the results of `DEplotHeatmaps()` will be added in the SE object which contains the results of `DEanalysisGlobal()`
- or `SEresDE=SEresVolcanoMAleuk` and the results of `DEplotHeatmaps()` will be added in the SE object which contains the results of `DEanalysisGlobal()` and `DEplotVolcanoMA()`.

```
SEresHeatmapLeuk <- DEplotHeatmaps(SEresDE=SEresVolcanoMAleuk,
                                   ColumnsCriteria=c(18, 19),
                                   Set.Operation="union",
                                   NbGene.analysis=20,
                                   SizeLabelRows=5,
                                   SizeLabelCols=5,
                                   Display.plots=FALSE,
                                   Save.plots=FALSE)
```

For the Zscore heatmap, The subset of genes is selected as followss

1. the user selects one or more binary column of the data.frame `DEsummaryLeuk` (see Section 3.3.1.1) with the input `ColumnsCriteria` which contains the column numbers of `DEsummaryLeuk` to be selected.
2. Three cases are possible:
   - If `Set.Operation="union"` then the rows extracted from the different datasets (raw counts, normalized data and `DEsummaryLeuk`) included in the SE object `SEresDELeuk500` are those such that the sum of the selected columns of `DEsummaryLeuk` given in `ColumnsCriteria` is >0. This means that the selected genes are those having at least one '1' in one of the selected columns.
   - If `Set.Operation="intersect"` then the rows extracted from the different datasets (raw counts, normalized data and `DEsummaryLeuk`) included in the SE object `SEresDELeuk500` are those such that the product of the selected columns of `DEsummaryLeuk` given in `ColumnsCriteria` is >0. This means that the selected genes are those having '1' in all of the selected columns.
   - If `Set.Operation="setdiff"` then the rows extracted from the different datasets (raw counts, normalized data and `DEsummaryLeuk`) included in the SE object `SEresDELeuk500` are those such that only one element of the selected columns of `DEsummaryLeuk` given in `ColumnsCriteria` is >0. This means that the selected genes are those having '1' in only one of the selected columns.
3. Finally, the selected subset of genes will be the `NbGene.analysis` genes extracted in step 2 above, which have the highest sum of absolute log2 fold change.

If the user wants to save the graphs, the input `Save.plots` must be

- either `Save.plots=TRUE`, and the graph will be saved in the same location than the input `path.result` of the function `DEanalysisGlobal()`.
- or a strings of characters giving the path to a folder where the graphs will be saved. The user then chooses the path of the folder where results can be saved.

If the user wants to display the graph, he must set `Display.plots=TRUE`.

If the user wants to see the results of the **DEplotHeatmaps**, he must first execute the following lines of code.

```
## Save 'Results' of the metadata in an object
resleuk500 <- S4Vectors::metadata(SEresHeatmapLeuk)$Results

## Save the results of DEplotHeatmaps in an object
resHeatmapLeuk <- resleuk500[[2]][[4]]
###
names(S4Vectors::metadata(SEresHeatmapLeuk))
```

```
## [1] "RAWcolnames"      "colGene"          "colDataINFO"
## [4] "RNAfiltering"     "DESeq2obj"        "Results"
## [7] "SEidentification"
```

```
str(resleuk500, max.level=2, give.attr=FALSE)
```

```
## List of 2
##  $ UnsupervisedAnalysis:List of 5
##   ..$ Normalization  :List of 2
##   ..$ PCA            :List of 4
##   ..$ HCPC           :List of 6
##   ..$ Mfuzz          :List of 5
##   ..$ GenesExpression:List of 2
##  $ SupervisedAnalysis  :List of 5
##   ..$ Normalization :List of 2
##   ..$ DEanalysis    :List of 4
##   ..$ VolcanoMAplots:List of 3
##   ..$ Heatmaps      :List of 4
##   ..$ Rgprofiler2   : NULL
```

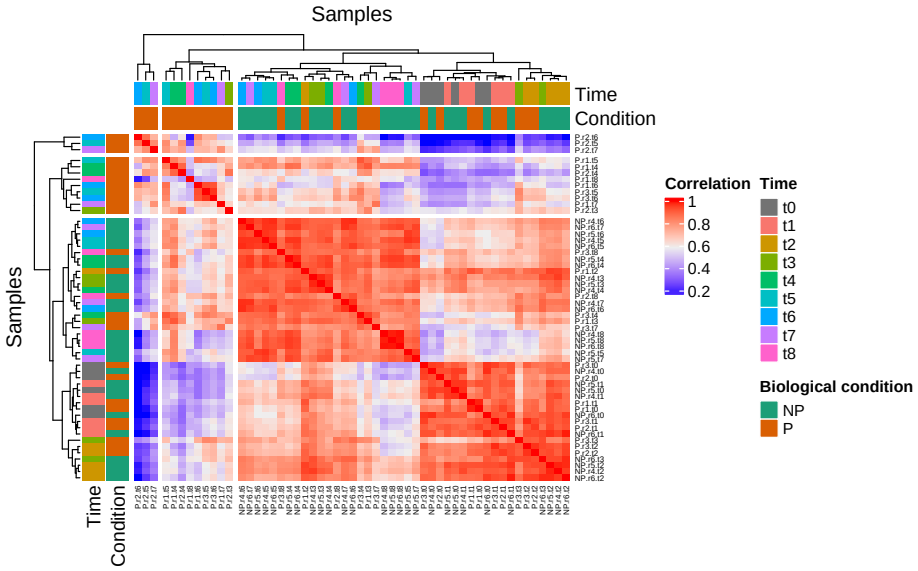
```
names(resHeatmapLeuk)
```

```
## [1] "Zscores"             "CorrelationMatrix"   "Heatmap_Zscore"
## [4] "Heatmap_Correlation"
```

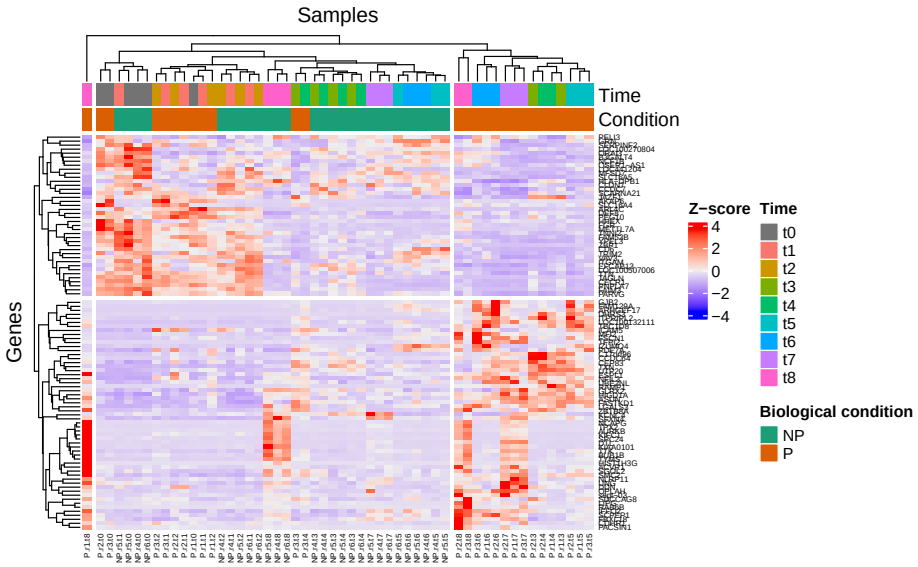
- `resHeatmapLeuk$CorrelationMatrix` contains the correlation matrix between samples
- `resHeatmapLeuk$Zscores` contains the matrix of scaled rle normalized data of the `NbGene.analysis` selected genes.
- `resHeatmapLeuk$Heatmap_Correlation` contains the heatmap associated to the matrix `resHeatmapLeuk$CorrelationMatrix` (*correlation heatmap*).
- `resHeatmapLeuk$Heatmap_Zscore` contains the heatmap associated to the matrix `resHeatmapLeuk$CorrelationMatrix` (*Zscore heatmap*).

The following lines of code plot the *correlation heatmap* and the *Zscore heatmap*

```
print(resHeatmapLeuk$Heatmap_Correlation)
```




```
print(resHeatmapLeuk$Heatmap_Zscore)
```



Write `?DEplotHeatmaps` in your console for more information about the function.

3.4 Gene Ontology (GO) analysis with GSEAQuickAnalysis() and GSEApreprocessing()

3.4.1 Gene ontology with the R package gprofiler2

The lines of code below realize an enrichment analysis with the R package gprofiler2 for a selection of genes. Beware, an internet connection is needed. The function returns

- a data.frame (output `metadata(SEresgprofiler2Leuk)$Rgprofiler2$GSEAresults`) giving information about all detected gene ontologies for the list of associated genes.
- a lollipop graph (see section Gene ontology and gene enrichment). The y-axis indicates the `MaxNumberGO` most significant gene ontologies and pathways associated to the selected genes. The gene ontologies and patways are sorted into descending order. The x-axis indicates the $-log10(pvalues)$. The higher is a lollipop the more significant is a gene ontology or pathway. A lollipop is yellow if the pvalues is smaller than 0.05 (significant) and blue otherwise.
- A Manhattan plot (see section Gene ontology and gene enrichment) indicating all genes ontologies ordered according to the functional database (G0::BP, G0::CC, G0::MF and KEGG)

The input `SEresDE` can be either

- `SEresDE=SEresDELeuk500`, and the results of `GSEAQuickAnalysis()` will be added in the SE object which contains the results of `DEanalysisGlobal()`
- either `SEresDE=SEresVolcanoMAleuk` and the results of `GSEAQuickAnalysis()` will be added in the SE object which contains the results of `DEanalysisGlobal()` and `DEplotVolcanoMA()`.
- or `SEresDE=SEresHeatmapLeuk` and the results of `GSEAQuickAnalysis()` will be added in the SE object which contains the results of `DEanalysisGlobal()`, `DEplotVolcanoMA()` and `DEplotHeatmaps()`.

```
SEresgprofiler2Leuk <- GSEAQuickAnalysis(Internet.Connection=FALSE,
                                         SEresDE=SEresHeatmapLeuk,
                                         ColumnsCriteria=c(18),
                                         ColumnsLog2ordered=NULL,
                                         Set.Operation="union",
                                         Organism="hsapiens",
                                         MaxNumberGO=20,
                                         Background=FALSE,
                                         Display.plots=FALSE,
                                         Save.plots=FALSE)
##
## head(SEresgprofiler2Leuk$GSEAresults)
```

As **GSEAQuickAnalysis()** requires an internet connection, we needed to add the input `Internet.Connection` in order to be sure to pass the tests realized on our package by Bioconductor. The input `Internet.Connection` is set by default to `FALSE` and as long as `Internet.Connection=FALSE`, no enrichment analysis will be done. Once the user is sure to have an internet connection, the user may set `Internet.Connection=TRUE` in order to realize the enrichment analysis.

The subset of genes is selected as follows

1. the user selects one or more binary column of the data.frame `DEsummaryLeuk` (see Section 3.3.1.1) with the input `ColumnsCriteria` which contains the column numbers of `DEsummaryLeuk` to be selected.
2. Three cases are possible:

- If `Set.Operation="union"` then the rows extracted from the different datasets (raw counts, normalized data and `DEsummaryLeuk`) included in the SE object `SEres DELeuk500` are those such that the sum of the selected columns of `DEsummaryLeuk` given in `ColumnsCriteria` is >0. This means that the selected genes are those having at least one '1' in one of the selected columns.
- If `Set.Operation="intersect"` then the rows extracted from the different datasets (raw counts, normalized data and `DEsummaryLeuk`) included in the SE object `SEres DELeuk500` are those such that the product of the selected columns of `DEsumma ryLeuk` given in `ColumnsCriteria` is >0. This means that the selected genes are those having '1' in all of the selected columns.
- If `Set.Operation="setdiff"` then the rows extracted from the different datasets (raw counts, normalized data and `DEsummaryLeuk`) included in the SE object `SEres DELeuk500` are those such that only one element of the selected columns of `DEsum maryLeuk` given in `ColumnsCriteria` is >0. This means that the selected genes are those having '1' in only one of the selected columns.

3. Finally, the selected subset of genes will be the `NbGene.analysis` genes extracted in step 2 above, which have the highest sum of absolute log2 fold change.

If `ColumnsLog2ordered` is a vector of integers, the enrichment analysis will take into account the genes order as the first genes will be considered to have the highest biological importance and the last genes the lowest. It corresponds to the columns number of `DEsummaryLeuk` (the output of `DEanalysisGlobal()`) which must contains log_2 fold change values. The rows of `DEsummaryLeuk` (corresponding to genes) will be decreasingly ordered according to the sum of absolute log_2 fold change (the selected columns must contain log_2 fold change values) before the enrichment analysis. If `ColumnsLog2ordered=NULL`, then the enrichment analysis will not take into account the genes order.

If the user wants to save the graphs, the input `Save.plots` must be

- either `Save.plots=TRUE`, and the graph will be saved in the same location than the input `path.result` of the function `DEanalysisGlobal()`.
- or a strings of characters giving the path to a folder where the graphs will be saved. The user then chooses the path of the folder where results can be saved.

`SEresgprofiler2Leuk` contains the results of enrichment analysis where selected genes are those DE at time t_7 for the biological condition P.

To retrieve the results, the user must first execute
`resgprofiler2Leuk <- S4Vectors::metadata(SEresgprofiler2Leuk)$Results[[2]][[5]]`

- `resgprofiler2Leuk$GSEAresults` contains a matrix which gives the results of the GSEA analysis: GO terms, GO names, p-value, GOparents, genes associated to each GO ...
- `resgprofiler2Leuk$selectedGenes` contains the list of selected genes
- `resgprofiler2Leuk$lollipopChart` contains the lollipop graph of the GSEA analysis (see Figure 2)
- `resgprofiler2Leuk$manhattanPlot` contains the Manhattan graph of the GSEA analysis (see Figure 3).

As expected in Section 3.3.1.2, DE genes at time t_7 for the biological condition P are linked to the cellular proliferation.

Write `?GSEAQuickAnalysis` in your console for more information about the function.

3.4.2 Preprocessing for GSEA, DAVID, WebGestalt, gProfiler, Panther, ShinyGO, Enrichr and GOrilla

The following lines of code will generate all files, for a selection of genes, in order to use the following software and online tools : GSEA, DAVID, WebGestalt, gProfiler, Panther, ShinyGO, Enrichr and GOrilla.

```
SEresPreprocessingLeuk <- GSEApreprocessing(SEresDE=SEresDELeuk500,
                                            ColumnsCriteria=c(18, 19),
                                            Set.Operation="union",
                                            Rnk.files=FALSE,
                                            Save.files=FALSE)
```

The subset of genes is selected as follows

1. the user selects one or more binary column of the data.frame `DEsummaryLeuk` (see Section 3.3.1.1) with the input `ColumnsCriteria` which contains the column numbers of `DEsummaryLeuk` to be selected.
2. Three cases are possible:
   - If `Set.Operation="union"` then the rows extracted from the different datasets (raw counts, normalized data and `DEsummaryLeuk`) included in the SE object `SEres DELeuk500` are those such that the sum of the selected columns of `DEsummaryLeuk` given in `ColumnsCriteria` is >0. This means that the selected genes are those having at least one '1' in one of the selected columns.
   - If `Set.Operation="intersect"` then the rows extracted from the different datasets (raw counts, normalized data and `DEsummaryLeuk`) included in the SE object `SEres DELeuk500` are those such that the product of the selected columns of `DEsumma ryLeuk` given in `ColumnsCriteria` is >0. This means that the selected genes are those having '1' in all of the selected columns.
   - If `Set.Operation="setdiff"` then the rows extracted from the different datasets (raw counts, normalized data and `DEsummaryLeuk`) included in the SE object `SEres DELeuk500` are those such that only one element of the selected columns of `DEsum maryLeuk` given in `ColumnsCriteria` is >0. This means that the selected genes are those having '1' in only one of the selected columns.
3. Finally, the selected subset of genes will be the `NbGene.analysis` genes extracted in step 2 above, which have the highest sum of absolute log2 fold change.

If the user wants to save the files, the input `Save.plots` must be

- either `Save.plots=TRUE`, and the graph will be saved in the same location than the input `path.result` of the function `DEanalysisGlobal()`.
- either a strings of characters giving the path to a folder where the graphs will be saved. The user then chooses the path of the folder where results can be saved.

Write `?GSEApreprocessing` in your console for more information about the function.

4 Quick description of the analysis of a dataset with several biological conditions (case 1)

In this section we use the mouse subdataset **RawCounts_Antoszewski2022_MOUSEsub500** (see the subsection 1.6.1) in order to explain the use of our package in **Case 1** (several biological conditions and a single time).
Most of the outputs in **Case 1** are of a similar form as those shown in **Case 3** (Section 3), except for some outputs whose list is precisely given below

- Subsection 4.2.3 page 61
- Subsection 4.3.1.2 page 63.

Furthermore, as there is only one time point, no Mfuzz analysis is realized.

4.1 Preprocessing step with DATAprepSE()

The preprocessing step is mandatory and is realized by our R function **DATAprepSE()** to store all information about the dataset in a standardized way (SummarizedExperiment class object, see Section 2.5). It can be done using the following lines of code.

```
SEresPrepMus1 <- DATAprepSE(RawCounts=RawCounts_Antoszewski2022_MOUSEsub500,
                            Column.gene=1,
                            Group.position=1,
                            Time.position=NULL,
                            Individual.position=2)
```

The function returns

- A *SummarizedExperiment* class object containing all information of the dataset to be used for exploratory data analysis
- A *DESeqDataSet* class object to be used for statistical analysis of the transcriptional response.

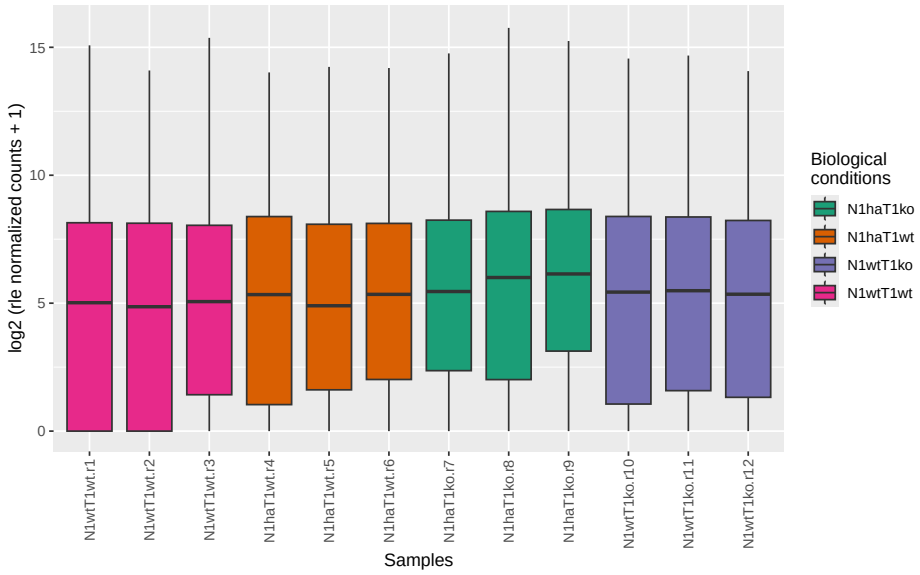
Write `?DATAprepSE` in your console for more information about the function.

4.2 Exploratory data analysis (unsupervised analysis)

4.2.1 Normalization with DATAnormalization()

The following lines of code realize the normalization step from the results of the function **DATAprepSE()** (subsection 4.1)

```
SEresNormMus1 <- DATAnormalization(SEres=SEresPrepMus1,
                                   Normalization="rle",
                                   Blind.rlog.vst=FALSE,
                                   Plot.Boxplot=TRUE,
                                   Colored.By.Factors=TRUE,
                                   Color.Group=NULL,
                                   path.result=NULL)
```



If `Plot.Boxplot=TRUE` a boxplot showing the distribution of the normalized expression (`Normalization="rle"` means that the rle method is used) of genes for each sample is returned.

If `Colored.By.Factors=TRUE`, the color of the boxplots will be different for different biological conditions. By default (if `Color.Group=NULL`), a color will be automatically assigned for each biological condition. Colors can be changed by creating the following data.frame

```
colMus1 <- data.frame(Name=c("N1wtT1wt", "N1haT1wt", "N1haT1ko", "N1wtT1ko"),
                      Col=c("black", "red", "green", "blue"))
```

and setting `Color.Group=colMus1`.

The x-labels give biological information and individual information separated by dots. If the user wants to see the 6th first rows of the normalized data, he can write in his console `head(res.Norm.Mus500$NormalizedData, n=6)`.

The user can save the graph in a folder thanks to the input path.result. If `path.result=NULL` the results will still be plotted but not saved in a folder.

Write `?DATAnormalization` in your console for more information about the function.

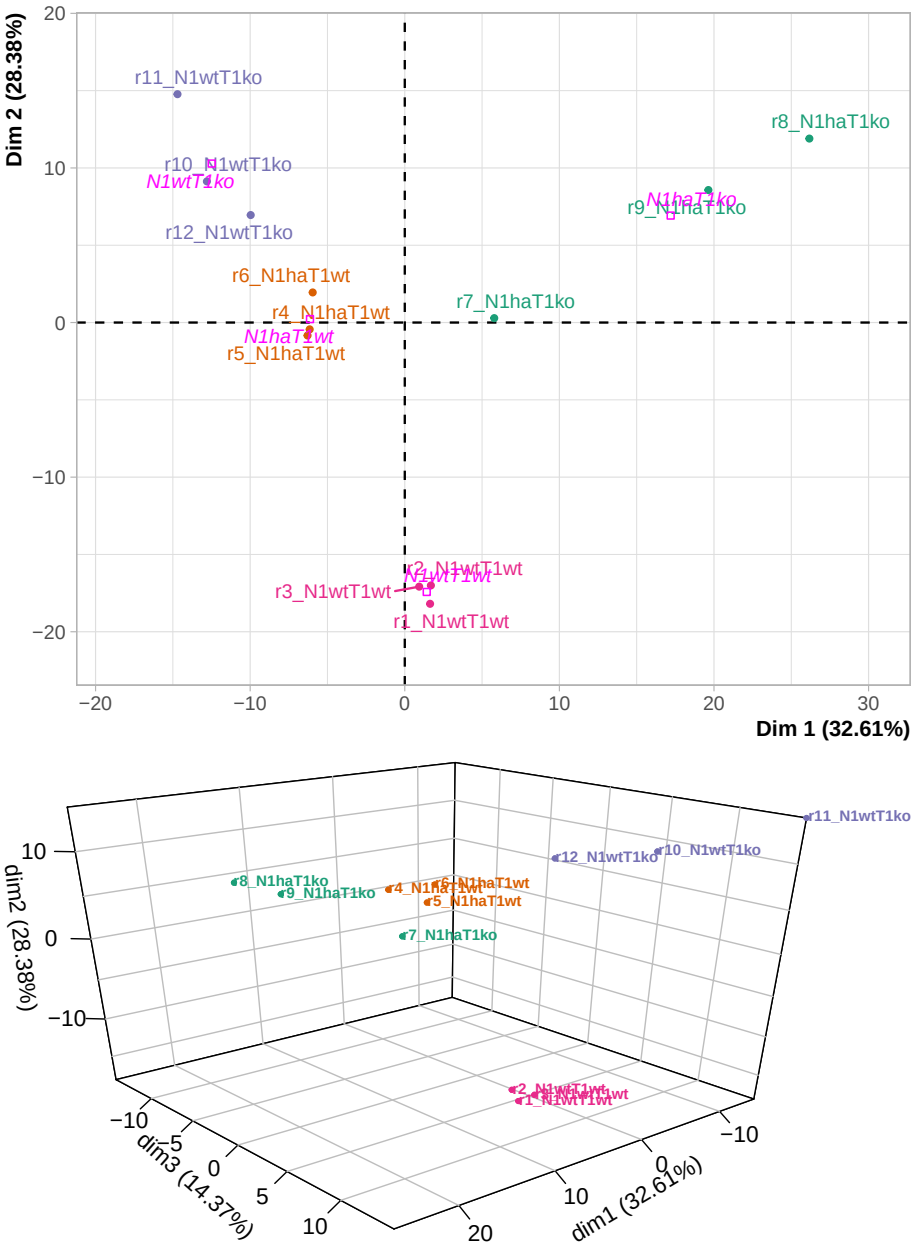
4.2.2 Factorial analysis: PCA with PCAanalysis() and clustering with HCPCanalysis()

4.2.2.1 PCA (case 1)

When samples belong only to different biological conditions, the lines of code below return from the results of the function **DATAnormalization()**

- the results of the R function `PCA()` from the package FactoMineR.
- one 2D PCA graph, one 3D PCA graph and the same 3D PCA graph in a rgl window (only if `motion3D=FALSE`) where samples are colored with different colors for different biological conditions.

```
SEresPCAMus1 <- PCAanalysis(SEresNorm=SEresNormMus1,
                            sample.deletion=NULL,
                            gene.deletion=NULL,
                            Plot.PCA=TRUE,
                            Mean.Accross.Time=FALSE,
                            Color.Group=NULL,
                            Cex.label=0.8,
                            Cex.point=0.7, epsilon=0.2,
                            Phi=25,Theta=140,
                            motion3D=FALSE,
                            path.result=NULL)
```



The graphs are

- stored in an SE object
- displayed if `Plot.PCA=TRUE`
- saved in a folder if the user selects a folder path in `path.result`. If `path.result=NULL` the results will not be saved in a folder.

By default (if `Color.Group=NULL`), a color will be automatically assigned to each biological condition. The user can change the colors by creating the following data.frame

```
colMus1 <- data.frame(Name=c("N1wtT1wt", "N1haT1wt", "N1haT1ko", "N1wtT1ko"),
                      Col=c("black", "red", "green", "blue"))
colMus1
```

```
##       Name   Col
## 1 N1wtT1wt black
## 2 N1haT1wt   red
## 3 N1haT1ko green
## 4 N1wtT1ko  blue
```

and setting `Color.Group=colMus1`.

If the user wants to delete, for instance, the genes 'ENSMUSG00000064842' and 'ENSMUSG00000051951' (respectively the second and forth gene) and/or delete the samples 'N1wtT1wt_r2' and 'N1haT1wt_r5', he can set

- `gene.deletion=c("ENSMUSG00000064842","ENSMUSG00000051951")` and/or `sample.deletion=c("N1wtT1wt_r2","N1haT1wt_r5")`
- `gene.deletion=c(2,4)` and/or `sample.deletion=c(3,6)`.
  The integers in `gene.deletion` and `sample.deletion` represent respectively the row numbers and the column numbers of `RawCounts` where the selected genes and samples are located.

Write `?PCAanalysis` in your console for more information about the function.

4.2.2.2 HCPC (case 1)

The user can realize the clustering with HCPC using the function **HCPCanalysis()** as below. The following lines of code return from the results of the function **DATAnormalization()**

- The results of the R function `HCPC()` from the package FactoMineR.
- A dendrogram
- A graph indicating for each sample, its cluster and the biological condition associated to the sample, using a color code
- One 2D PCA graph, one 3D PCA graph and the same 3D PCA graph in a rgl window (only if `motion3D=FALSE`). These PCA graphs are identical to the outputs of **PCA-analysis()** but samples are colored with different colors for different clusters.

```
SEresHCPCMus1 <- HCPCanalysis(SEresNorm=SEresNormMus1,
                              gene.deletion=NULL,
                              sample.deletion=NULL,
                              Plot.HCPC=FALSE,
                              Cex.label=0.8, Cex.point=0.7, epsilon=0.2,
                              Phi=25, Theta=140,
                              motion3D=FALSE,
                              path.result=NULL)
```

The graphs are

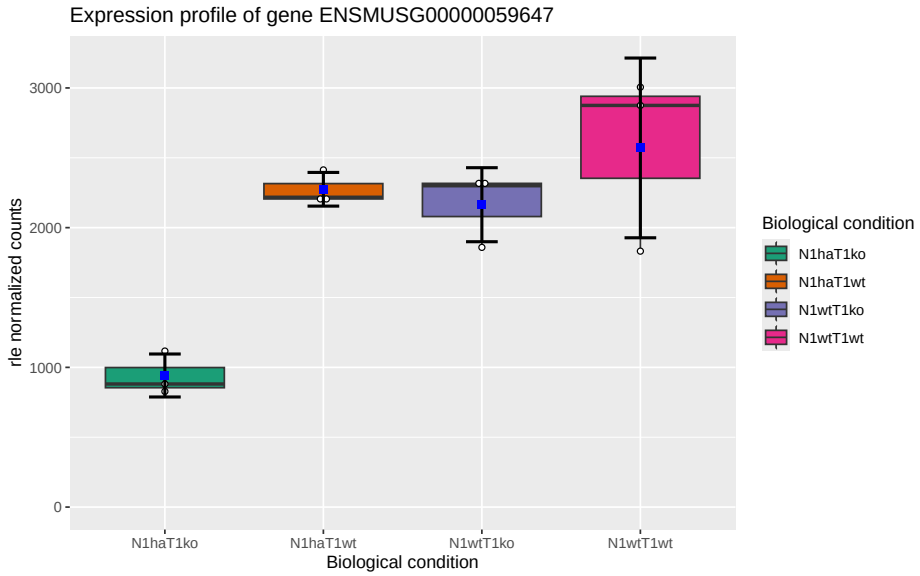
- stored in an SE object
- displayed if `Plot.HCPC=TRUE`
- saved in a folder if the user selects a folder path in `path.result`. If `path.result=NULL` the results will not be saved in a folder.

Write `?HCPCanalysis` in your console for more information about the function.

4.2.3 Genes expression profile with DATAplotExpressionGenes()

The lines of code below allow to plot, from the results of the function **DATAnormalization()**, the expression profile of the 10th gene showing for each biological condition: a box plot and error bars (standard deviation). Each box plot, violin plot and error bars is associated to the distribution of the expression of the 10th genes in all samples belonging to the corresponding biological condition.

```
resEVOmus1500 <- DATAplotExpressionGenes(SEresNorm=SEresNormMus1,
                                         Vector.row.gene=c(10),
                                         Color.Group=NULL,
                                         Plot.Expression=TRUE,
                                         path.result=NULL)
```



If the user wants to select several genes , for instance the 28th, the 38th, the 39th and the 50th, he needs to set `Vector.row.gene=c(28,38:39,50)`.

The graphs are

- stored in an SE object
- displayed if `Plot.Expression=TRUE`
- saved in a folder if the user selects a folder path in `path.result`. If `path.result=NULL` the results will not be saved in a folder.

By default (if `Color.Group=NULL`), a color will be automatically assigned to each biological condition. The user can change the colors by creating the following data.frame

```
colMus1 <- data.frame(Name=c("N1wtT1wt", "N1haT1wt", "N1haT1ko", "N1wtT1ko"),
                      Col=c("black", "red", "green", "blue"))
```

and setting `Color.Group=colMus1`.

Write `?DATAplotExpressionGenes` in your console for more information about the function.

4.3 Statistical analysis of the transcriptional response (supervised analysis)

4.3.1 DE analysis with DEanalysisGlobal()

The function below

- returns a data.frame. See subection Data.frame summarising all the DE analysis (case 1).
- plots the following graphs
  - an upset graph, realized with the R package UpSetR [33], which corresponds to a Venn diagram barplot. If there are only two biological conditions, the graph will not be plotted. See subsection Description of graphs
  - a barplot showing the number of specific genes per biological condition. See subection Description of graphs.

Due to time consuming of the DE analysis, we stored in the object `Results_DEanalysis_sub500` (uncommented lines) a list of three objects

- `Results_DEanalysis_sub500$DE_Schleiss2021_CLLsub500`, stored the results of `DEanalysisGlobal()` with `RawCounts_Schleiss2021_CLLsub500`.
- `Results_DEanalysis_sub500$DE_Antoszewski2022_MOUSEsub500`, stored the results of `DEanalysisGlobal()` with `RawCounts_Antoszewski2022_MOUSEsub500`.
- `Results_DEanalysis_sub500$DE_Leong2014_FISSIONsub500wt`, stored the results of `DEanalysisGlobal()` with `RawCounts_Leong2014_FISSIONsub500wt`.

```
SEresDEMus1 <- DEanalysisGlobal(SEres=SEresPrepMus1,
                                pval.min=0.05,
                                pval.vect.t=NULL,
                                log.FC.min=1,
                                LRT.supp.info=FALSE,
                                Plot.DE.graph=FALSE,
                                path.result=NULL,
                                Name.folder.DE=NULL)

## [1] "Preprocessing"
## [1] "Differential expression step with DESeq2::DESeq()"
## [1] "Case 1 analysis : Biological conditions only"

## data("Results_DEanalysis_sub500")
## SEresDEMus1 <- Results_DEanalysis_sub500$DE_Antoszewski2022_MOUSEsub500
```

The graphs are

- stored in an SE object
- displayed if `Plot.DE.graph=TRUE`
- saved in a folder if the user selects a folder path in `path.result`. If `path.result=NULL` the results will not be saved in a folder.

Write `?DEanalysisGlobal` in your console for more information about the function.

4.3.1.1 Data.frame summarising all the DE analysis (case 1)

The output data.frame can be extracted with the following line of code,

```
DEsummaryMus1 <- SummarizedExperiment::rowData(SEresDEMus1)
```

As we use abbreviated column names, we propose a glossary in order to help the user to understand meaning of each column. The glossary of the column names can be extracted with the following lines of code,

```
resDEMus1 <- S4Vectors::metadata(SEresDEMus1)$Results[[2]][[2]]
resGlossaryMus1 <- resDEMus1$Glossary
```

and then write `DEsummaryMus1` and `resGlossaryMus1` in the R console.

The data.frame contains

- gene names (column 1)
- pvalues, log2 fold change and DE genes between each pairs of biological conditions (for a total of $3 \times \frac{N_{bc} \times (N_{bc}-1)}{2} = 3 \times 6 = 18$ columns).
- a binary column (1 and 0) where 1 means that the gene is DE between at least one pair of biological conditions.
- $N_{bc} = 4$ binary columns (with N_{bc} the number of biological conditions), which give the specific genes for each biological condition. A '1' in one of these columns means that the gene is specific to the biological condition associated to the column. 0 otherwise. A gene is called specific to a given biological condition BC1, if the gene is DE between BC1 and any other biological conditions, but not DE between any pair of other biological conditions.
- $N_{bc} = 4$ columns filled with -1, 0 and 1, one per biological condition. A '1' in one of these columns means that the gene is up-regulated (or over-expressed) for the biological condition associated to the column. A gene is called up-regulated for a given biological condition BC1 if the gene is specific to the biological condition BC1 and expressions in BC1 are higher than in the other biological conditions. A '-1' in one of these columns means the gene is down-regulated (or under-expressed) for the biological condition associated to the column. A gene is called down-regulated for a given biological condition BC1 if the gene is specific to the biological condition BC1 and expressions in BC1 are lower than in the other biological conditions. A '0' in one of these columns means that the gene is not specific to the biological condition associated to the column.

4.3.1.2 Description of graphs

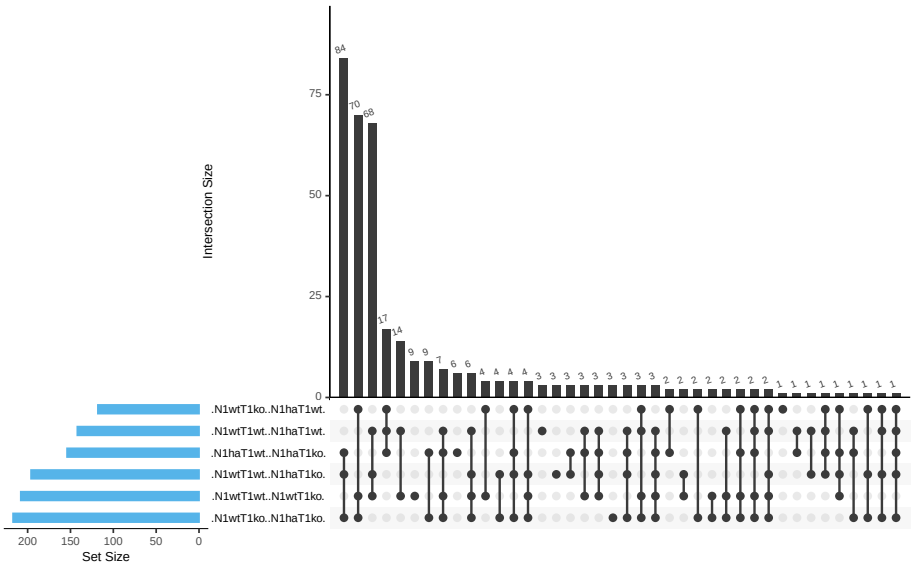
The user can plot three graphs.

One upset graph. The graph plots the number of genes corresponding to each possible intersection in a Venn barplot. We say that

- a set of pairs of biological conditions forms an intersection when there is at least one gene which is DE for each of these pairs of biological conditions, but not for the others.
- a gene corresponds to a given intersection if this genes is DE for each pair of biological conditions in the intersection, but not for the others.

The following line of code plots the Venn barplot

```
print(resDEMus1$VennBarplot)
```



The second column of Venn barplot the gives the number of genes corresponding to the intersection

$$\Big\{\{\text{N1wtT1wt, N1haT1wt}\},\{\text{N1wtT1wt, N1haT1ko}\},\{\text{N1wtT1wt, N1wtT1ko}\}\Big\}\ .$$

In other words, this is the number of genes DE between

- N1wtT1wt versus N1haT1wt
- N1wtT1wt versus N1haT1ko
- N1wtT1wt versus N1wtT1ko

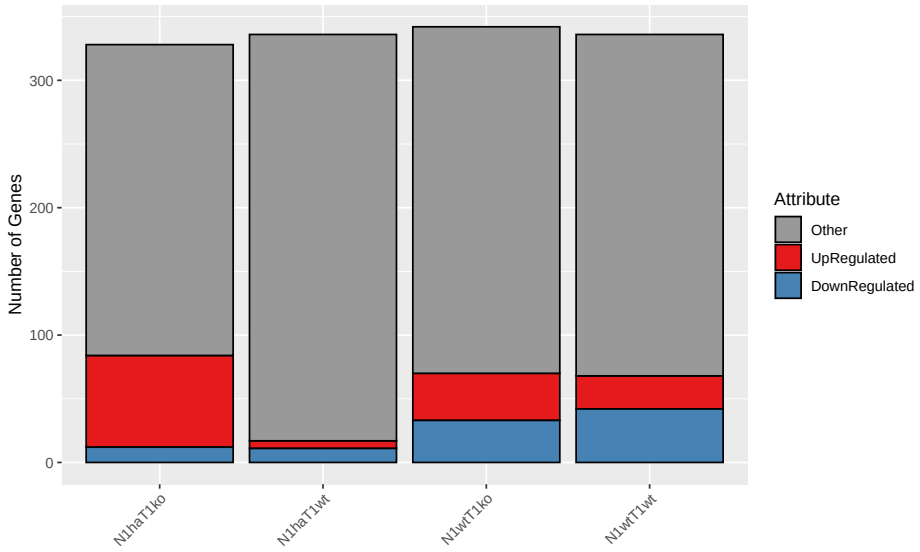
and not DE between any other pairs of biological conditions. Note that this columns actually gives the number of specific genes to the biological condition N1wtT1wt.

One barplot. The graph plots for each biological condition BC1

- The number of up- and down-regulated genes which are specific to the biological condition BC1.
- The number of genes categorized as "Other." A gene belongs to the "Other" category if it is DE between BC1 and at least one other biological condition and not specific.

The following line of code plots the barplot

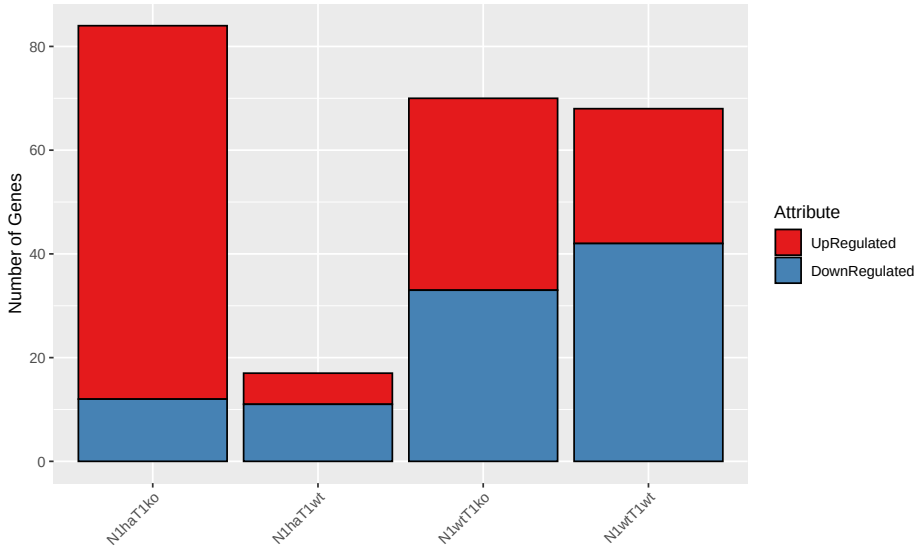
```
print(resDEMus1$NumberDEgenes_SpecificAndNoSpecific)
```



One barplot. The second barplot is the same graph without the "Other" category will be plotted too. If there are only two biological condition, only the second barplot will be plotted.

The following line of code plots the second barplot

```
print(resDEMus1$NumberDEgenes_SpecificGenes)
```



4.3.2 Volcano plots, ratio intensity (MA) plots and Heatmaps with DEplotVolcanoMA() and DEplotHeatmaps()

4.3.2.1 Volcano and MA plots (case 1)

The following lines of code allow to plot

- $\binom{N_{bc}}{2} = \frac{N_{bc}(N_{bc}-1)}{2} = 6$ volcano plots (with $N_{bc} = 4$ the number of biological conditions).
- $\frac{N_{bc}(N_{bc}-1)}{2} = 6$ MA plots.

allowing to separate non DE genes, DE genes below a threshold of log2 fold change and DE genes above a threshold of log2 fold change (see Section 3.3.2.1 for more details).

```
SEresVolcanoMAMus1 <- DEplotVolcanoMA(SEresDE=SEresDEMus1,
                                      NbGene.plotted=2,
                                      SizeLabel=3,
                                      Display.plots=FALSE,
                                      Save.plots=FALSE)
```

If the user wants to save the graphs, the input `Save.plots` must be

- either `Save.plots=TRUE`, and the graph will be saved in the same location than the input `path.result` of the function `DEanalysisGlobal()`.
- or a string of characters giving the path to a folder where the graphs will be saved. The user then chooses the path of the folder where results can be saved.

If the user wants to display the graph, he must set `Display.plots=TRUE`.

Write `?DEplotVolcanoMA` in your console for more information about the function.

4.3.2.2 Heatmaps (case 1)

The following lines of code allow to plot a correlation heatmap between samples (correlation heatmap) and a heatmap across samples and genes called Zscore heatmap, for a subset of genes that can be selected by the user. The second heatmap is build from the normalized count data after being both centered and scaled (Zscore).

```
SEresVolcanoMAMus1 <- DEplotHeatmaps(SEresDE=SEresDEMus1,
                                     ColumnsCriteria=2,#c(2,4),
                                     Set.Operation="union",
                                     NbGene.analysis=20,
                                     SizeLabelRows=5,
                                     SizeLabelCols=5,
                                     Display.plots=FALSE,
                                     Save.plots=FALSE)
```

For the Zscore heatmap, The subset of genes is selected as followss

1. the user selects one or more binary column of the data.frame `DEsummaryMus1` (see Section 4.3.1.1) with the input `ColumnsCriteria` which contains the column numbers of `DEsummaryMus1` to be selected.

2. Three cases are possible:
    - If `Set.Operation="union"` then the rows extracted from the different datasets (raw counts and normalized data and `DEsummaryMus1`) included in the SE object `SEresDEMus1` are those such that the sum of the selected columns of `DEsummaryMus1` given in `ColumnsCriteria` is >0. This means that the selected genes are those having at least one '1' in one of the selected columns.
    - If `Set.Operation="intersect"` then the rows extracted from the different datasets (raw counts and normalized data and `DEsummaryMus1`) included in the SE object `SEresDEMus1` are those such that the product of the selected columns of `DEsummaryMus1` given in `ColumnsCriteria` is >0. This means that the selected genes are those having '1' in all of the selected columns.
    - If `Set.Operation="setdiff"` then the rows extracted from the different datasets (raw counts and normalized data and `DEsummaryMus1`) included in the SE object `SEresDEMus1` are those such that only one element of the selected columns of `DEsummaryMus1` given in `ColumnsCriteria` is >0. This means that the selected genes are those having '1' in only one of the selected columns.
3. Finally, the selected subset of genes will be the `NbGene.analysis` genes extracted in step 2 above, which have the highest sum of absolute log2 fold change.

If the user wants to save the graphs, the input `Save.plots` must be

- either `Save.plots=TRUE`, and the graph will be saved in the same location than the input `path.result` of the function `DEanalysisGlobal()`.
- or a strings of characters giving the path to a folder where the graphs will be saved. The user then chooses the path of the folder where results can be saved.

If the user wants to display the graph, he must set `Display.plots=TRUE`.

Write `?DEplotHeatmaps` in your console for more information about the function.

4.4 Gene Ontology (GO) analysis with GSEAQuickAnalysis() and GSEApreprocessing()

4.4.1 Gene ontology with the R package gprofiler2

The lines of code below realize an enrichment analysis with the R package gprofiler2 for a selection of genes. Beware, an internet connection is needed. The function returns

- a data.frame (output `metadata(SEresgprofiler2Mus1)$Rgprofiler2$GSEAresults`) giving information about all detected gene ontologies for the list of associated genes.
- a lollipop graph (see section Gene ontology and gene enrichment). The y-axis indicates the `MaxNumberGO` most significant gene ontologies and pathways associated to the selected genes. The gene ontologies and patways are sorted into descending order. The x-axis indicates the $-log10(pvalues)$. The higher is a lollipop the more significant is a gene ontology or pathway. A lollipop is yellow if the pvalues is smaller than 0.05 (significant) and blue otherwise.
- A Manhattan plot (see section Gene ontology and gene enrichment) indicating all genes ontologies ordered according to the functional database (G0::BP, G0::CC, G0::MF and KEGG)

```
SEresgprofiler2Mus1 <- GSEAQuickAnalysis(Internet.Connection=FALSE,
                                         SEresDE=SEresDEMus1,
                                         ColumnsCriteria=2,
                                         ColumnsLog2ordered=NULL,
                                         Set.Operation="union",
                                         Organism="mmusculus",
                                         MaxNumberGO=10,
                                         Background=FALSE,
                                         Display.plots=FALSE,
                                         Save.plots=FALSE)
##
## head(4Vectors::metadata(SEresgprofiler2Mus1)$Rgprofiler2$GSEAresults)
```

As **GSEAQuickAnalysis()** requires an internet connection, we needed to add the input `Internet.Connection` in order to be sure to pass the tests realized on our package by Bioconductor. The input `Internet.Connection` is set by default to `FALSE` and as long as `Internet.Connection=FALSE`, no enrichment analysis will be done. Once the user is sure to have an internet connection, the user may set `Internet.Connection=TRUE` in order to realize the enrichment analysis.

The subset of genes is selected as follows

1. the user selects one or more binary column of the data.frame `DEsummaryMus1` (see Section 4.3.1.1) with the input `ColumnsCriteria` which contains the column numbers of `DEsummaryMus1` to be selected.
2. Three cases are possible:
   - If `Set.Operation="union"` then the rows extracted from the different datasets (raw counts and normalized data and `DEsummaryMus1`) included in the SE object `SEresDEMus1` are those such that the sum of the selected columns of `DEsummaryMus1` given in `ColumnsCriteria` is >0. This means that the selected genes are those having at least one '1' in one of the selected columns.

- If `Set.Operation="intersect"` then the rows extracted from the different datasets (raw counts and normalized data and `DEsummaryMus1`) included in the SE object `SEresDEMus1` are those such that the product of the selected columns of `DEsummaryMus1` given in `ColumnsCriteria` is >0. This means that the selected genes are those having '1' in all of the selected columns.
- If `Set.Operation="setdiff"` then the rows extracted from the different datasets (raw counts and normalized data and `DEsummaryMus1`) included in the SE object `SEresDEMus1` are those such that only one element of the selected columns of `DEsummaryMus1` given in `ColumnsCriteria` is >0. This means that the selected genes are those having '1' in only one of the selected columns.

3. Finally, the selected subset of genes will be the `NbGene.analysis` genes extracted in step 2 above, which have the highest sum of absolute log2 fold change.

If `ColumnsLog2ordered` is a vector of integers, the enrichment analysis will take into account the genes order as the first genes will be considered to have the highest biological importance and the last genes the lowest. It corresponds to the columns number of `DEsummaryMus1`, the output of `DEanalysisGlobal()`, which must contains log_2 fold change values. The rows of `DEsummaryMus1` (corresponding to genes) will be decreasingly ordered according to the sum of absolute log_2 fold change (the selected columns must contain log_2 fold change values) before the enrichment analysis. If `ColumnsLog2ordered=NULL`, then the enrichment analysis will not take into account the genes order.

If the user wants to save the graphs, the input `Save.plots` must be

- either `Save.plots=TRUE`, and the graph will be saved in the same location than the input `path.result` of the function `DEanalysisGlobal()`.
- or a strings of characters giving the path to a folder where the graphs will be saved. The user then chooses the path of the folder where results can be saved.

Write `?GSEAQuickAnalysis` in your console for more information about the function.

4.4.2 Preprocessing for GSEA, DAVID, WebGestalt, gProfiler, Panther, ShinyGO, Enrichr and GOrilla

The following lines of code will generate all files, for a selection of genes, in order to use the following software and online tools : GSEA, DAVID, WebGestalt, gProfiler, Panther, ShinyGO, Enrichr and GOrilla.

```
SEresPreprocessingMus1 <- GSEApreprocessing(SEresDE=SEresDEMus1,
                                            ColumnsCriteria=2,
                                            Set.Operation="union",
                                            Rnk.files=FALSE,
                                            Save.files=FALSE)
```

The subset of genes is selected as follows

1. the user selects one or more binary column with the input `ColumnsCriteria` which corresponds to the column number of `rowData(SEresDEMus1)`
2. Then the subset of genes will be
   - If `Set.Operation="union"` then the rows extracted from the different datasets included in `SEresDEMus1` are those such that the sum of the selected columns of `SummarizedExperiment::rowData(SEresDEMus1)` by `ColumnsCriteria` is >0.
   - If `Set.Operation="intersect"` then the rows extracted from the different datasets included in `SEresDEMus1` are those such that the product of the selected columns of `SummarizedExperiment::rowData(SEresDEMus1)` by `ColumnsCriteria` is >0.
   - If `Set.Operation="setdiff"` then the rows extracted from the different datasets included in `SEresDEMus1` are those such that only one element of the selected columns of `SummarizedExperiment::rowData(SEresDEMus1)` by `ColumnsCriteria` is >0.

If the user wants to save the files, the input `Save.plots` must be

- either `Save.plots=TRUE`, and the graph will be saved in the same location than the input `path.result` of the function `DEanalysisGlobal()`.
- or a strings of characters giving the path to a folder where the graphs will be saved. The user then chooses the path of the folder where results can be saved.

Write `?GSEApreprocessing` in your console for more information about the function.

5 Quick description of the analysis of a dataset with several time points (case 2)

In this section we use the fission yeast subdataset **RawCounts_Leong2014_FISSIONsub500wt** (see the subsection Example of MultiRNAflow in case 2, several time points: Fission dataset) in order to explain the use of our package in **Case 2** (several times point and a single biological condiion).
Most of the outputs in **Case 2** are of a similar form as those shown in **Case 3** (Section 3), except for the output described in Subsection 5.2.4 page 75 which can be considered as slightly different.

5.1 Preprocessing step with DATAprepSE()

The preprocessing step is mandatory and is realized by our R function **DATAprepSE()** to store all information about the dataset in a standardized way (SummarizedExperiment class object, see Section 2.5). It can be done using the following lines of code.

```
SEresPrepFission <- DATAprepSE(RawCounts=RawCounts_Leong2014_FISSIONsub500wt,
                               Column.gene=1,
                               Group.position=NULL,
                               Time.position=2,
                               Individual.position=3)
```

The function returns

- A *SummarizedExperiment* class object containing all information of the dataset to be used for exploratory data analysis
- A *DESeqDataSet* class object to be used for statistical analysis of the transcriptional response.

Write `?DATAprepSE` in your console for more information about the function.

5.2 Exploratory data analysis (unsupervised analysis)

5.2.1 Normalization with DATAnormalization()

The following lines of code realize the normalization step from the results of the function **DATAprepSE()** (subsection 5.1).

```
SEresNormYeast <- DATAnormalization(SEres=SEresPrepFission,
                                    Normalization="rlog",
                                    Blind.rlog.vst=FALSE,
                                    Plot.Boxplot=FALSE,
                                    Colored.By.Factors=TRUE,
                                    Color.Group=NULL,
                                    Plot.genes=FALSE,
                                    path.result=NULL)
```

If `Plot.Boxplot=TRUE` a boxplot showing the distribution of the normalized expression (`Normalization="rlog"` means that the rlog method is used) of genes for each sample is returned.

If `Colored.By.Factors=TRUE`, the color of the boxplots would be different for different time points. In case 2, the option `Color.Group` is not used. The x-labels indicate time information and individual information separated by a dot.

If the user wants to see the 10 first rows of the normalized data, he can write in his console `head(SEresNormYeast$NormalizedData, n=10)`.

The user chooses the path of the folder where the graph can be saved. If `path.result=NULL`, results are plotted but not saved.

Write `?DATAnormalization` in your console for more information about the function.

5.2.2 Factorial analysis: PCA with PCAanalysis() and clustering with HCPCanalysis()

5.2.2.1 PCA (case 2)

When samples belong only to different times of measure, the lines of code below return from the results of the function **DATAnormalization()**

- the results of the function `PCA()`
- one 2D PCA graph, one 3D PCA graph and the same 3D PCA graph in a rgl window (only if `motion3D=FALSE`) where samples are colored with different colors for different time points. Furthermore, lines are drawn in gray between each pair of consecutive points for each individual (if `Mean.Accross.Time=FALSE`, otherwise lines will be drawn only between mean values of all individuals for each time point).
- the same graphs as above but without lines (not shown).

```
SEresPCAyeast <- PCAanalysis(SEresNorm=SEresNormYeast,
                             gene.deletion=NULL,
                             sample.deletion=NULL,
                             Plot.PCA=FALSE,
                             Mean.Accross.Time=FALSE,
                             Cex.label=0.8, Cex.point=0.7, epsilon=0.3,
                             Phi=25,Theta=140,
                             motion3D=FALSE,
                             path.result=NULL)
```

The graphs are

- stored in an SE object
- displayed if `Plot.PCA=TRUE`
- saved in a folder if the user selects a folder path in `path.result`. If `path.result=NULL` the results will not be saved in a folder.

The user cannot select a color for each time. A specific palette is automatically used.

These figures show that the temporal behavior is similar between individuals.

If the user wants for instance, to perform the PCA analysis without the genes 'SPAC212.11' and 'SPNCRNA.70' (first and third gene) and/or without the samples 'wt_t0_r2' and 'wt_t1_r1', he can set

- either `gene.deletion=c("SPAC212.11","SPNCRNA.70")` and/or `sample.deletion=c("wt_t0_r2","wt_t1_r1")`,
- or `gene.deletion=c(1,3)` and/or `sample.deletion=c(3,5)`.
  The integers in `gene.deletion` and `sample.deletion` represent respectively the row numbers (resp. the column numbers) of `RawCounts` corresponding to genes (resp. samples) that need to be removed from `RawCounts`.

Write `?PCAanalysis` in your console for more information about the function.

5.2.2.2 HCPC (case 2)

The user can realize the clustering with HCPC using the function **HCPCanalysis()** as below. The following lines of code return from the results of the function **DATAnormalization()**

- The results of the R function `HCPC()` from the package FactoMineR.
- A dendrogram
- A graph indicating for each sample, its cluster and the biological condition associated to the sample, using a color code
- One 2D PCA graph, one 3D PCA graph and the same 3D PCA graph in a rgl window (only if `motion3D=FALSE`). These PCA graphs are identical to the outputs of **PCAanalysis()** but samples are colored with different colors for different clusters.

```
SEresHCPCyeast <- HCPCanalysis(SEresNorm=SEresNormYeast,
                               gene.deletion=NULL,
                               sample.deletion=NULL,
                               Plot.HCPC=FALSE,
                               Cex.label=0.9,
                               Cex.point=0.7,
                               epsilon=0.2,
                               Phi=25,Theta=140,
                               motion3D=FALSE,
                               path.result=NULL)
```

The graphs are

- stored in an SE object
- displayed if `Plot.HCPC=TRUE`
- saved in a folder if the user selects a folder path in `path.result`. If `path.result=NULL` the results will not be saved in a folder.

Write `?HCPCanalysis` in your console for more information about the function.

5.2.3 Temporal clustering analysis with MFUZZanalysis()

The following function realizes the temporal clustering analysis. It takes as input, a number of clusters (`DataNumberCluster`) that can be chosen automatically if `DataNumberCluster=NULL` and the results of the function **DATAnormalization()** (see Section 5.2.1). The lines of code below return for each biological condition

- the summary of the results of the R function `mfuzz()` from the package Mfuzz.

- the scaled height plot, computed with the `HCPC()` function, and shows the number of clusters chosen automatically (if `DataNumberCluster=NULL`). If `Method="hcpc"`, the function plots the scaled within-cluster inertia, but if `Method="kmeans"`, the function plots the scaled within-cluster inertia. As the number of genes can be very high, we recommend to select `Method="hcpc"` which is by default.
- the output graphs from the R package `Mfuzz` showing the most common temporal behavior among all genes for each biological condition. The plots below correspond to the biological condition 'P'.

```
SEresMfuzzYeast <- MFUZZanalysis(SEresNorm=SEresNormYeast,
                                 DataNumberCluster=NULL,
                                 Method="hcpc",
                                 Membership=0.5,
                                 Min.std=0.1,
                                 Plot.Mfuzz=FALSE,
                                 path.result=NULL)

## 0 genes excluded.
## 31 genes excluded.
```

The graphs are

- stored in an SE object
- displayed if `Plot.Mfuzz=TRUE`
- saved in a folder if the user selects a folder path in `path.result`. If `path.result=NULL` the results will not be saved in a folder.

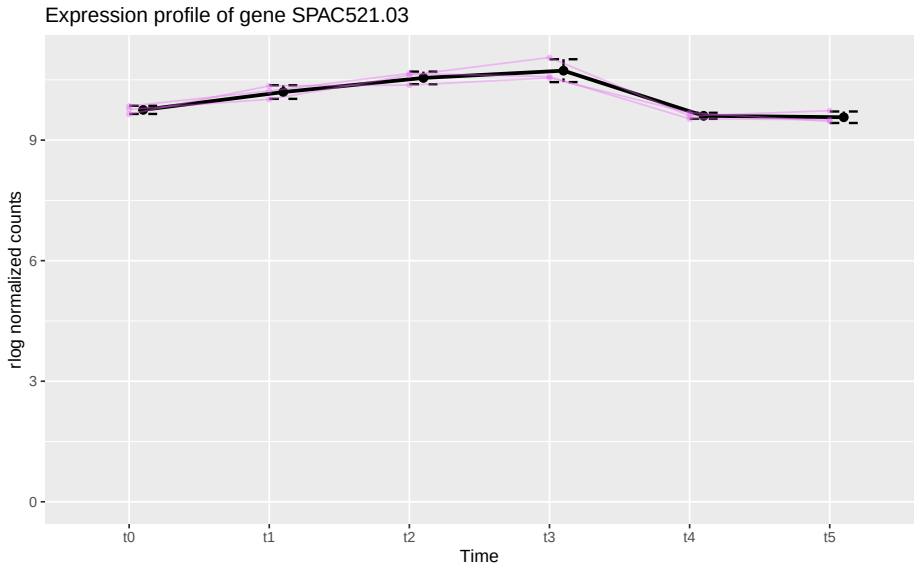
Write `?MFUZZanalysis` in your console for more information about the function.

5.2.4 Genes expression profile with DATAplotExpressionGenes()

The lines of code below plot, from the results of the function **DATAnormalization()**,

- the evolution of the 17th gene expression of all three replicates across time (purple lines)
- the evolution of the mean and the standard deviation of the 17th gene expression across time (black lines).

```
SEresEVOyeast <- DATAplotExpressionGenes(SEresNorm=SEresNormYeast,
                                         Vector.row.gene=c(17),
                                         Plot.Expression=TRUE,
                                         path.result=NULL)
```



The graphs are

- stored in an SE object
- displayed if `Plot.Expression=TRUE`
- saved in a folder if the user selects a folder path in `path.result`. If `path.result=NULL` the results will not be saved in a folder.

If the user wants to select several genes, for instance the 1st, the 2nd, the 17th and the 19th, he needs to set `Vector.row.gene=c(1,2,17,19)`.

Write `?DATAplotExpressionGenes` in your console for more information about the function.

5.3 Statistical analysis of the transcriptional response for the four dataset (supervised analysis)

5.3.1 DE analysis with DEanalysisGlobal()

The function below

- returns a data.frame. See subsection Data.frame summarising all the DE analysis (case 2).
- plots the following graphs
  - an alluvial graph. See subsection Description of graphs
  - a graph showing the number of DE genes as a function of time for each temporal group. By temporal group, we mean the sets of genes which are first DE at the same time. See subsection Description of graphs
  - a barplot showing the number of DE genes (up- or down-regulated) for each time. See subsection Description of graphs.
  - two upset graphs, realized with the R package UpSetR [33], showing the number of DE genes belonging to each DE temporal pattern. By temporal pattern, we mean the set of times t_i such that the gene is DE between t_i and the reference time t_0. See subsection Description of graphs.

The commented lines take too much time, uncomment them in order to use the function `DEanalysisGlobal()`. Due to time consuming of the DE analysis, we stored in the object `Results_DEanalysis_sub500` (uncommented lines) a list of three objects

- `Results_DEanalysis_sub500$DE_Schleiss2021_CLLsub500`, stored the results of `DEanalysisGlobal()` with `RawCounts_Schleiss2021_CLLsub500`.
- `Results_DEanalysis_sub500$DE_Antoszewski2022_MOUSEsub500`, stored the results of `DEanalysisGlobal()` with `RawCounts_Antoszewski2022_MOUSEsub500`.
- `Results_DEanalysis_sub500$DE_Leong2014_FISSIONsub500wt`, stored the results of `DEanalysisGlobal()` with `RawCounts_Leong2014_FISSIONsub500wt`.

```
# DEyeastWt <- DEanalysisGlobal(SEres=SEresPrepFission, log.FC.min=1,
#                               pval.min=0.05, pval.vect.t=NULL,
#                               LRT.supp.info=FALSE, Plot.DE.graph =FALSE,
#                               path.result=NULL, Name.folder.DE=NULL)
data("Results_DEanalysis_sub500")
DEyeastWt <- Results_DEanalysis_sub500$DE_Leong2014_FISSIONsub500wt
```

The graphs are

- stored in an SE object
- displayed if `Plot.DE.graph=TRUE`
- saved in a folder if the user selects a folder path in `path.result`. If `path.result=NULL` the results will not be saved in a folder.

Write `?DEanalysisGlobal` in your console for more information about the function.

5.3.1.1 Data.frame summarising all the DE analysis (case 2)

The output data.frame can be extracted with the following line of code,

```
DEsummaryFission <- SummarizedExperiment::rowData(DEyeastWt)
```

As we use abbreviated column names, we propose a glossary in order to help the user to understand meaning of each column. The glossary of the column names can be extracted with the following lines of code,

```
resDEyeast <- S4Vectors::metadata(DEyeastWt)$Results[[2]][[2]]
resGlossaryFission <- resDEyeast$Glossary
```

and then write `DEsummaryFission` and `resGlossaryFission` in the R console.

The data.frame contains

- gene names
- pvalues, log2 fold change and DE genes between each time t_i versus the reference time t_0 (for a total of $T-1=5$ columns).
- a binary column (1 and 0) where 1 means that the gene is DE at least at between one time t_i versus the reference time t_0.
- a column where each element is succession of 0 and 1. The positions of '1' indicate the set of times t_i such that the gene is DE between t_i and the reference time t_0. So each element of the column is what we called previously, a temporal pattern.

5.3.1.2 Description of graphs

The function returns the following plots

1. An alluvial graph. The x-axis of the alluvial graph is labeled with all times except t_0. For each vertical barplot, there are two strata: 1 and 0 whose sizes indicate respectively the number of DE genes and of non DE genes, between the time corresponding to the barplot and the reference time t_0. The alluvial graph is composed of curves, each corresponding to a single gene, which are gathered in alluvia. An alluvium is composed of all genes having the same curve: for example, an alluvium going from the stratum 0 at time t_1 to the stratum 1 at time t_2 corresponds to the set of genes which are not DE at t_1 and are DE at time t_2. Each alluvium connects pairs of consecutive barplots and its thickness gives the number of genes in the alluvium. The color of each alluvium indicates the temporal group, defined as the set of genes which are all first DE at the same time with respect to the reference time t_0.
2. A graph giving the time evolution of the number of DE genes within each temporal group. The x-axis labels indicate all times except t_0.
3. A barplot showing the number of DE genes per time. The x-axis labels indicate all times except t_0. For each DE gene, we compute the sign of the log2 fold change between time ti and time t_0. If the sign is positive (resp. negative), the gene is categorized as up-regulated (resp. down-regulated). In the graph, the up-regulated (resp. down-regulated) genes are indicated in red (resp. in blue).

4. An upset graph, realized with the R package UpSetR [33], plotting the number of genes in each DE temporal pattern in a Venn barplot. By DE temporal pattern, we mean a subset of times in $t_1, \ldots, t_n$. We say that a gene belongs to a DE temporal pattern if the gene is DE versus t_0 only at the times in this DE temporal patterns. For each gene in a given DE temporal pattern, we compute the number of DE times where it is up-regulated and we use a color code in the Venn barplot to indicate the number of genes in a temporal pattern that are up-regulated a given number of times.
5. The same upset graph is also plotted without colors.

5.3.2 Volcano plots, ratio intensity (MA) plots and Heatmaps with DEplotVolcanoMA() and DEplotHeatmaps()

5.3.2.1 Volcano and MA plots (case 2)

The following lines of code allow to plot

- $T-1=6-1=5$ volcano plots (with $T=6$ the number time points)
- $T-1=5$ MA plots

allowing to separate non DE genes, DE genes below a threshold of log2 fold change and DE genes above a threshold of log2 fold change (see Section 3.3.2.1 for more details).

```
SEresVolcanoMAFission <- DEplotVolcanoMA(SEresDE=DEyeastWt,
                                         NbGene.plotted=2,
                                         SizeLabel=3,
                                         Display.plots=FALSE,
                                         Save.plots=TRUE)
```

If the user wants to save the graphs, the input `Save.plots` must be

- either `Save.plots=TRUE`, and the graph will be saved in the same location than the input `path.result` of the function `DEanalysisGlobal()`.
- either a strings of characters giving the path to a folder where the graphs will be saved. The user then chooses the path of the folder where results can be saved.

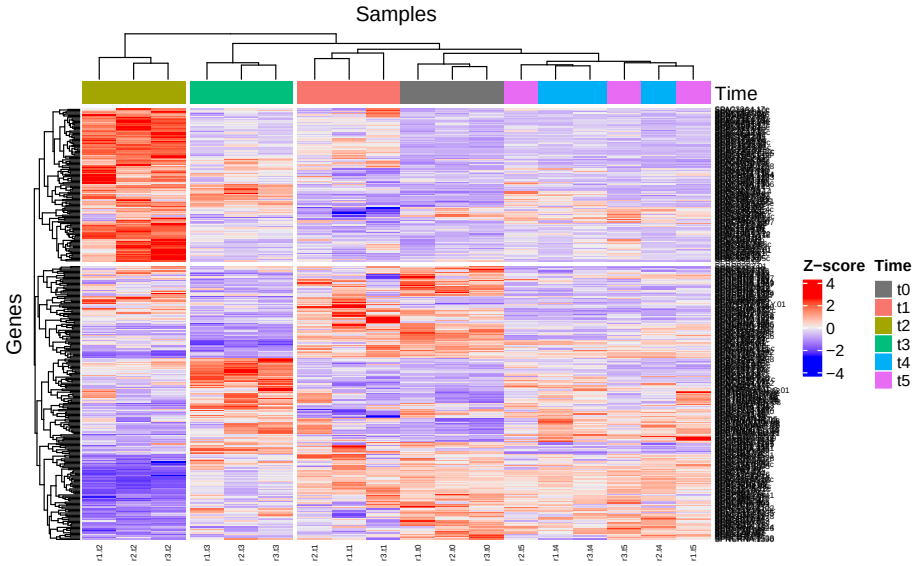
If the user wants to display the graph, he must set `Display.plots=TRUE`.

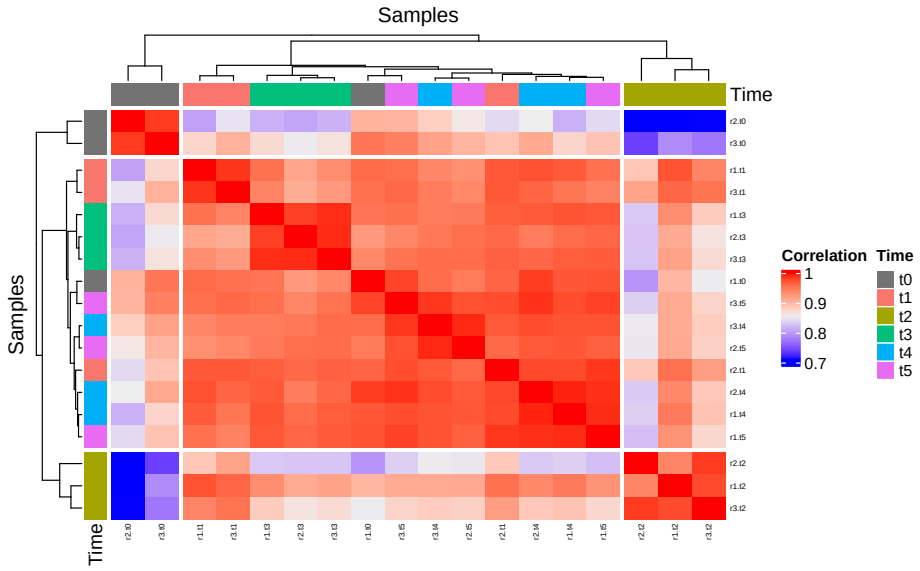
Write `?DEplotVolcanoMA` in your console for more information about the function.

5.3.2.2 Heatmaps (case 2)

The following lines of code allow to plot a correlation heatmap between samples (correlation heatmap) and a heatmap across samples and genes called Zscore heatmap, for a subset of genes that can be selected by the user. The second heatmap is build from the normalized count data after being both centered and scaled (Zscore).

```
SEresHeatmapFission <- DEplotHeatmaps(SEresDE=DEyeastWt,
                                      ColumnsCriteria=2,
                                      Set.Operation="union",
                                      NbGene.analysis=20,
                                      Color.Group=NULL,
                                      SizeLabelRows=5,
                                      SizeLabelCols=5,
                                      Display.plots=TRUE,
                                      Save.plots=FALSE)
```





For the Zscore heatmap, The subset of genes is selected as followss

1. the user selects one or more binary column of the data.frame `DEsummaryFission` (see Section 5.3.1.1) with the input `ColumnsCriteria` which contains the column numbers of `DEsummaryFission` to be selected.
2. Three cases are possible:
   - If `Set.Operation="union"` then the rows extracted from the different datasets (raw counts and normalized data and `DEsummaryFission`) included in the SE object `DEyeastWt` are those such that the sum of the selected columns of `DEsummaryFission` given in `ColumnsCriteria` is >0. This means that the selected genes are those having at least one '1' in one of the selected columns.
   - If `Set.Operation="intersect"` then the rows extracted from the different datasets (raw counts and normalized data and `DEsummaryFission`) included in the SE object `DEyeastWt` are those such that the product of the selected columns of `DEsummaryFission` given in `ColumnsCriteria` is >0. This means that the selected genes are those having '1' in all of the selected columns.
   - If `Set.Operation="setdiff"` then the rows extracted from the different datasets (raw counts and normalized data and `DEsummaryFission`) included in the SE object `DEyeastWt` are those such that only one element of the selected columns of `DEsummaryFission` given in `ColumnsCriteria` is >0. This means that the selected genes are those having '1' in only one of the selected columns.
3. Finally, the selected subset of genes will be the `NbGene.analysis` genes extracted in step 2 above, which have the highest sum of absolute log2 fold change.

If the user wants to save the graphs, the input `Save.plots` must be

- either `Save.plots=TRUE`, and the graph will be saved in the same location than the input `path.result` of the function `DEanalysisGlobal()`.
- either a strings of characters giving the path to a folder where the graphs will be saved. The user then chooses the path of the folder where results can be saved.

If the user wants to display the graph, he must set `Display.plots=TRUE`. Write `?DEplotHeatmaps` in your console for more information about the function.

5.4 Gene Ontology (GO) analysis with GSEAQuickAnalysis() and GSEApreprocessing()

5.4.1 Gene ontology with the R package gprofiler2

The lines of code below realize an enrichment analysis with the R package gprofiler2 for a selection of genes. Beware, an internet connection is needed. The function returns

- a data.frame (output `metadata(SEresGprofiler2Fission)$Rgprofiler2$GSEAresults`) giving information about all detected gene ontologies for the list of associated genes.
- a lollipop graph (see section Gene ontology and gene enrichment). The y-axis indicates the `MaxNumberGO` most significant gene ontologies and pathways associated to the selected genes. The gene ontologies and patways are sorted into descending order of $-log10(pvalues)$. The x-axis indicates the $-log10(pvalues)$. The higher is a lollipop the more significant is a gene ontology or pathway. A lollipop is yellow if the pvalues is smaller than 0.05 (significant) and blue otherwise.
- A Manhattan plot (see section Gene ontology and gene enrichment) indicating all genes ontologies ordered according to the functional database (G0::BP, G0::CC, G0::MF and KEGG)

```
SEresGprofiler2Fission <- GSEAQuickAnalysis(Internet.Connection=FALSE,
                                            SEresDE=DEyeastWt,
                                            ColumnsCriteria=2,
                                            ColumnsLog2ordered=NULL,
                                            Set.Operation="union",
                                            Organism="spombe",
                                            MaxNumberGO=20,
                                            Background=FALSE,
                                            Display.plots=FALSE,
                                            Save.plots=FALSE)
##
## head(SEresGprofiler2Fission$GSEAresults)
```

As **GSEAQuickAnalysis()** requires an internet connection, we needed to add the input `Internet.Connection` in order to be sure to pass the tests realized on our package by Bioconductor. The input `Internet.Connection` is set by default to `FALSE` and as long as `Internet.Connection=FALSE`, no enrichment analysis will be done. Once the user is sure to have an internet connection, the user may set `Internet.Connection=TRUE` in order to realize the enrichment analysis.

The subset of genes is selected as follows

1. the user selects one or more binary column of the data.frame `DEsummaryFission` (see Section 5.3.1.1) with the input `ColumnsCriteria` which contains the column numbers of `DEsummaryFission` to be selected.
2. Three cases are possible:

- If `Set.Operation="union"` then the rows extracted from the different datasets (raw counts and normalized data and `DEsummaryFission`) included in the SE object `DEyeastWt` are those such that the sum of the selected columns of `DEsummaryFission` given in `ColumnsCriteria` is >0. This means that the selected genes are those having at least one '1' in one of the selected columns.
- If `Set.Operation="intersect"` then the rows extracted from the different datasets (raw counts and normalized data and `DEsummaryFission`) included in the SE object `DEyeastWt` are those such that the product of the selected columns of `DEsummaryFission` given in `ColumnsCriteria` is >0. This means that the selected genes are those having '1' in all of the selected columns.
- If `Set.Operation="setdiff"` then the rows extracted from the different datasets (raw counts and normalized data and `DEsummaryFission`) included in the SE object `DEyeastWt` are those such that only one element of the selected columns of `DEsummaryFission` given in `ColumnsCriteria` is >0. This means that the selected genes are those having '1' in only one of the selected columns.

3. Finally, the selected subset of genes will be the `NbGene.analysis` genes extracted in step 2 above, which have the highest sum of absolute log2 fold change.

If `ColumnsLog2ordered` is a vector of integers, the enrichment analysis will take into account the genes order as the first genes will be considered to have the highest biological importance and the last genes the lowest. It corresponds to the columns number of `DEsummaryFission`, the output of `DEanalysisGlobal()`, which must contains log_2 fold change values. The rows of `DEsummaryFission` (corresponding to genes) will be decreasingly ordered according to the sum of absolute log_2 fold change (the selected columns must contain log_2 fold change values) before the enrichment analysis. If `ColumnsLog2ordered=NULL`, then the enrichment analysis will not take into account the genes order.

If the user wants to save the graphs, the input `Save.plots` must be

- either `Save.plots=TRUE`, and the graph will be saved in the same location than the input `path.result` of the function `DEanalysisGlobal()`.
- or a strings of characters giving the path to a folder where the graphs will be saved. The user then chooses the path of the folder where results can be saved.

Write `?GSEAQuickAnalysis` in your console for more information about the function.

5.4.2 Preprocessing for GSEA, DAVID, WebGestalt, gProfiler, Panther, ShinyGO, Enrichr and GOrilla

The following lines of code will generate all files, for a selection of genes, in order to use the following software and online tools : GSEA, DAVID, WebGestalt, gProfiler, Panther, ShinyGO, Enrichr and GOrilla.

```
SEresPreprocessingYeast <- GSEApreprocessing(SEresDE=DEyeastWt,
                                             ColumnsCriteria=2,
                                             Set.Operation="union",
                                             Rnk.files=FALSE,
                                             Save.files=FALSE)
```

The subset of genes is selected as follows

1. the user selects one or more binary column of the data.frame `DEsummaryFission` (see Section 5.3.1.1) with the input `ColumnsCriteria` which contains the column numbers of `DEsummaryFission` to be selected.
2. Three cases are possible:
   - If `Set.Operation="union"` then the rows extracted from the different datasets (raw counts and normalized data and `DEsummaryFission`) included in the SE object `DEyeastWt` are those such that the sum of the selected columns of `DEsummaryFission` given in `ColumnsCriteria` is >0. This means that the selected genes are those having at least one '1' in one of the selected columns.
   - If `Set.Operation="intersect"` then the rows extracted from the different datasets (raw counts and normalized data and `DEsummaryFission`) included in the SE object `DEyeastWt` are those such that the product of the selected columns of `DEsummaryFission` given in `ColumnsCriteria` is >0. This means that the selected genes are those having '1' in all of the selected columns.
   - If `Set.Operation="setdiff"` then the rows extracted from the different datasets (raw counts and normalized data and `DEsummaryFission`) included in the SE object `DEyeastWt` are those such that only one element of the selected columns of `DEsummaryFission` given in `ColumnsCriteria` is >0. This means that the selected genes are those having '1' in only one of the selected columns.
3. Finally, the selected subset of genes will be the `NbGene.analysis` genes extracted in step 2 above, which have the highest sum of absolute log2 fold change.

If the user wants to save the files, the input `Save.plots` must be

- either `Save.plots=TRUE`, and the graph will be saved in the same location than the input `path.result` of the function `DEanalysisGlobal()`.
- or a strings of characters giving the path to a folder where the graphs will be saved. The user then chooses the path of the folder where results can be saved.

Write `?GSEApreprocessing` in your console for more information about the function.

6 Quick description of the analysis of a dataset with several time points and more than two biological conditions (case 4)

In this section we use the mouse subdataset **RawCounts_Weger2021_MOUSEsub500** (see Subsection Example of MultiRNAflow in case 4, several time points and more than two biological conditions: Mouse dataset 2) in order to explain the use of our package in **Case 4** (several times point and more than two biological conditions).
Most of the outputs in **Case 4** are of a similar form as those shown in **Case 3** (Section 3), except for some outputs whose list is precisely given below

- Subsection 6.3.1.2 page 91
- Subsection 6.3.1.3 page 93.

6.1 Preprocessing step with DATAprepSE()

The preprocessing step is mandatory and is realized by our R function **DATAprepSE()** to store all information about the dataset in a standardized way (SummarizedExperiment class object, see Section 2.5). It can be done using the following lines of code.

```
SEresPrepMus2 <- DATAprepSE(RawCounts=RawCounts_Weger2021_MOUSEsub500,
                            Column.gene=1,
                            Group.position=1,
                            Time.position=2,
                            Individual.position=3)
```

The function returns

- *SummarizedExperiment* class object containing all information of the dataset to be used for exploratory data analysis
- *DESeqDataSet* class object to be used for statistical analysis of the transcriptional response.

Write `?DATAprepSE` in your console for more information about the function.

6.2 Exploratory data analysis (unsupervised analysis)

6.2.1 Normalization with DATAnormalization()

The following lines of code realize the normalization step from the results of the function **DATAprepSE()** (subsection 6.1).

```
SEresNormMus2 <- DATAnormalization(SEres=SEresPrepMus2,
                                   Normalization="vst",
                                   Blind.rlog.vst=FALSE,
                                   Plot.Boxplot=FALSE,
                                   Colored.By.Factors=TRUE,
                                   Color.Group=NULL,
                                   path.result=NULL)
```

If `Plot.Boxplot=TRUE` a boxplot showing the distribution of the normalized expression (`Normalization="vst"` means that the vst method is used) of genes for each sample is returned.

If `Colored.By.Factors=TRUE`, the color of the boxplots would be different for different biological conditions. By default (if `Color.Group=NULL`), a color will be automatically applied for each biological condition. You can change the colors by creating the following data.frame

```
colMus2 <- data.frame(Name=c("BmKo", "BmWt" ,"CrKo", "CrWt"),
                      Col=c("red", "blue", "orange", "darkgreen"))
```

and setting `Color.Group=colMus2`.

The x-labels give biological information, time information and individual information separated by dots. If the user wants to see the 6th first rows of the normalized data, he can write in his console `head(SEresNormMus2$NormalizedData, n=6)`

The user can save the graph in a folder thanks to the input path.result. If `path.result=NULL` the results will still be plotted but not saved in a folder.

Write `?DATAnormalization` in your console for more information about the function.

6.2.2 Factorial analysis: PCA with PCAanalysis() and clustering with HCPCanalysis()

6.2.2.1 PCA (case 4)

When samples belong to different biological conditions and different time points, the previous lines of code return from the results of the function **DATAnormalization()**:

- The results of the R function `PCA()` from the package FactoMineR.
- one 2D PCA graph, one 3D PCA graph and the same 3D PCA graph in a rgl window (only if `motion3D=FALSE`) where samples are colored with different colors for different biological conditions. Furthermore, lines are drawn between each pair of consecutive points for each individual (if `Mean.Accross.Time=FALSE`, otherwise lines will be drawn only between mean values of all individuals for each time point and biological conditions).
- one 2D PCA graph, one 3D PCA graph and the same 3D PCA graph in a rgl window (only if `motion3D=FALSE`) for each biological condition, where samples are colored with different colors for different time points. Furthermore, lines are drawn between each pair of consecutive points for each sample (if `Mean.Accross.Time=FALSE`, otherwise lines will be drawn only between mean values of all individuals for each time point and biological conditions).
- the same graphs described above but without lines.

```
SEresPCAmus2 <- PCAanalysis(SEresNorm=SEresNormMus2,
                            gene.deletion=NULL,
                            sample.deletion=NULL,
                            Plot.PCA=FALSE, motion3D=FALSE,
                            Mean.Accross.Time=FALSE,
                            Color.Group=NULL,
                            Cex.label=0.6, Cex.point=0.7, epsilon=0.2,
                            Phi=25, Theta=140,
                            path.result=NULL,
                            Name.folder.pca=NULL)
```

The graphs are

- stored in an SE object
- displayed if `Plot.PCA=TRUE`
- similar to those described in subsection PCA (case 3).
- saved in a folder if the user selects a folder path in `path.result`. If `path.result=NULL` the results will not be saved in a folder.

By default (if `Color.Group=NULL`), a color will be automatically applied for each biological condition. You can change the colors by creating the following data.frame

```
colMus2 <- data.frame(Name=c("BmKo", "BmWt", "CrKo", "CrWt"),
                      Col=c("red", "blue", "orange", "darkgreen"))
```

and setting `Color.Group=colMus2`. The user cannot change the color associated to each time point.

If you want to delete, for instance, the genes 'ENSMUSG00000025921' and 'ENSMUSG00000026113' (respectively the second and sixth gene) and/or delete the samples 'BmKo_t2_r1' and 'BmKo_t5_r2', set

- `gene.deletion=c("ENSMUSG00000025921", "ENSMUSG00000026113")` and/or `sample.deletion=c("BmKo_t2_r1", "BmKo_t5_r2")`
- `gene.deletion=c(2,6)` and/or `sample.deletion=c(3,13)`.
  The integers in `gene.deletion` and `sample.deletion` represent respectively the row numbers and the column numbers of `RawCounts` where the selected genes and samples are located.

Write `?PCAanalysis` in your console for more information about the function.

6.2.2.2 HCPC (case 4)

The user can realize the clustering with HCPC using the function **HCPCanalysis()** as below. The following lines of code return from the results of the function **DATAnormalization()**

- The results of the R function `HCPC()` from the package FactoMineR.
- A dendrogram
- A graph indicating for each sample, its cluster and the biological condition associated to the sample, using a color code
- One 2D PCA graph, one 3D PCA graph and the same 3D PCA graph in a rgl window (only if `motion3D=FALSE`). These PCA graphs are identical to the outputs of **PCAanalysis()** but samples are colored with different colors for different clusters.

```
SEresHCPCmus2 <- HCPCanalysis(SEresNorm=SEresNormMus2,
                              gene.deletion=NULL,
                              sample.deletion=NULL,
                              Plot.HCPC=FALSE,
                              Phi=25,Theta=140,
                              Cex.point=0.6,
                              epsilon=0.2,
                              Cex.label=0.6,
                              motion3D=FALSE,
                              path.result=NULL,
                              Name.folder.hcpc=NULL)
```

The graphs are

- stored in an SE object
- displayed if `Plot.HCPC=TRUE`
- similar to those described in subsection HCPC (case 3).
- saved in a folder if the user selects a folder path in `path.result`. If `path.result=NULL` the results will not be saved in a folder.

Write `?HCPCanalysis` in your console for more information about the function.

6.2.3 Temporal clustering analysis with MFUZZanalysis()

The following function realizes the temporal clustering analysis. It takes as input, a number of clusters (`DataNumberCluster`) that can be chosen automatically if `DataNumberCluster=NULL` and the results of the function **DATAnormalization()** (see Section 6.2.1). The lines of code below return for each biological condition

- the summary of the results of the R function `mfuzz()` from the package Mfuzz.
- the scaled height plot, computed with the `HCPC()` function, and shows the number of clusters chosen automatically (if `DataNumberCluster=NULL`). If `Method="hcpc"`, the function plots the scaled within-cluster inertia, but if `Method="kmeans"`, the function plots the scaled within-cluster inertia. As the number of genes can be very high, we recommend to select `Method="hcpc"` which is by default.
- the output graphs from the R package `Mfuzz` showing the most common temporal behavior among all genes for each biological condition. The plots below correspond to the biological condition 'P'.

```
SEresMfuzzLeuk500 <- MFUZZanalysis(SEresNorm=SEresNormMus2,
                                   DataNumberCluster=NULL,
                                   Method="hcpc",
                                   Membership=0.5,
                                   Min.std=0.1,
                                   Plot.Mfuzz=FALSE,
                                   path.result=NULL, Name.folder.mfuzz=NULL)

## 0 genes excluded.
## 31 genes excluded.
## 0 genes excluded.
## 25 genes excluded.
## 0 genes excluded.
## 38 genes excluded.
## 0 genes excluded.
## 22 genes excluded.
```

The graphs are

- stored in an SE object
- displayed if `Plot.Mfuzz=TRUE`
- similar to those described in subsection Temporal clustering analysis with MFUZZanalysis().
- saved in a folder if the user selects a folder path in `path.result`. If `path.result=NULL` the results will not be saved in a folder.

Other temporal information are shown in the alluvial graph of the subsection **??** that can be compared with the previous graphs.

Write `?MFUZZanalysis` in your console for more information about the function.

6.2.4 Plot expression of data with DATAplotExpressionGenes()

In this section we use the mouse subdataset **RawCounts_Weger2021_MOUSEsub500** (see Subsection Example of MultiRNAflow in case 4, several time points and more than two biological conditions: Mouse dataset 2) in order to explain **DATAplotExpressionGenes()** in **case 3** when there are more than two biological conditions.

The previous lines of code allow to plot, from the results of the function **DATAnormalization()**, for each biological condition: the evolution of the 17th gene expression of the three replicates across time and the evolution of the mean and the standard deviation of the 17th gene expression across time. The color of the different lines are different for different biological conditions.

```
SEresEVOmus2 <- DATAplotExpressionGenes(SEresNorm=SEresNormMus2,
                                        Vector.row.gene=c(17),
                                        Color.Group=NULL,
                                        Plot.Expression=FALSE,
                                        path.result=NULL)
```

The graphs are

- stored in an SE object
- displayed if `Plot.Expression=TRUE`
- saved in a folder if the user selects a folder path in `path.result`. If `path.result=NULL` the results will not be saved in a folder.

By default (if `Color.Group=NULL`), a color will be automatically assigned to each biological condition. The user can change the colors by creating the following data.frame

```
colMus2 <- data.frame(Name=c("BmKo", "BmWt", "CrKo", "CrWt"),
                      Col=c("red", "blue", "orange", "darkgreen"))
```

and setting `Color.Group=colMus2`.

If the user wants to select several genes, for instance the 97th, the 192th, the 194th and the 494th, he needs to set `Vector.row.gene=c(97,192,194,494)`.

Write `?DATAplotExpressionGenes` in your console for more information about the function.

6.3 Statistical analysis of the transcriptional response for the four dataset (supervised analysis)

6.3.1 DE analysis with DEanalysisGlobal()

To keep the execution time of the algorithm fast, we will take only three biological conditions and three times.

```
Sub3bc3T <- RawCounts_Weger2021_MOUSEsub500[, seq_len(73)]
SelectTime <- grep("_t0_", colnames(Sub3bc3T))
SelectTime <- c(SelectTime, grep("_t1_", colnames(Sub3bc3T)))
SelectTime <- c(SelectTime, grep("_t2_", colnames(Sub3bc3T)))
Sub3bc3T <- Sub3bc3T[, c(1, SelectTime)]

SEresPrepMus23b3t <- DATAprepSE(RawCounts=Sub3bc3T,
                                Column.gene=1,
                                Group.position=1,
                                Time.position=2,
                                Individual.position=3)
```

The lines of code above

- return a data.frame. See subsection Data.frame summarizing all the DE analysis (case 3).
- plot the following graphs (similar to those described in section DE analysis with DEanalysisGlobal())
  - *Results from the temporal statistical analysis* (**case 2** for each biological condition). See subsection Graphs from the results of the temporal statistical analysis
  - *Results from the statistical analysis by biological condition* (**case 1** for each fixed time). See subsection Graphs from the results of the biological condition analysis.
  - *Results from the combination of temporal and biological statistical analysis*. See subsection Graphs from the results of the combination of temporal and biological statistical analysis

```
SEresDE3tMus2 <- DEanalysisGlobal(SEres=SEresPrepMus23b3t,
                                  pval.min=0.05,
                                  pval.vect.t=NULL,
                                  log.FC.min=1,
                                  LRT.supp.info=FALSE,
                                  Plot.DE.graph=FALSE,
                                  path.result=NULL, Name.folder.DE=NULL)

## [1] "Preprocessing"
## [1] "Differential expression step with DESeq2::DESeq()"
## [1] "Case 3 analysis : Biological conditions and Times."
## [1] "DE time analysis for each biological condition."
## [1] "DE group analysis for each time measurement."
## [1] "Combined time and group results."
```

The output data.frame can be extracted with the following line of code,

```
DEsummaryMus2 <- SummarizedExperiment::rowData(SEresDE3tMus2)
```

As we use abbreviated column names, we propose a glossary in order to help the user to understand meaning of each column. The glossary of the column names can be extracted with the following lines of code,

```
resDEmus2 <- S4Vectors::metadata(SEresDE3tMus2)$Results[[2]][[2]]
resGlossaryMus2 <- resDEmus2$Glossary
```

and then write `DEsummaryMus2` and `resGlossaryMus2` in the R console.

The following subsection will show graphs not shown in DE analysis with DEanalysisGlobal().

6.3.1.1 Graphs from the results of the temporal statistical analysis

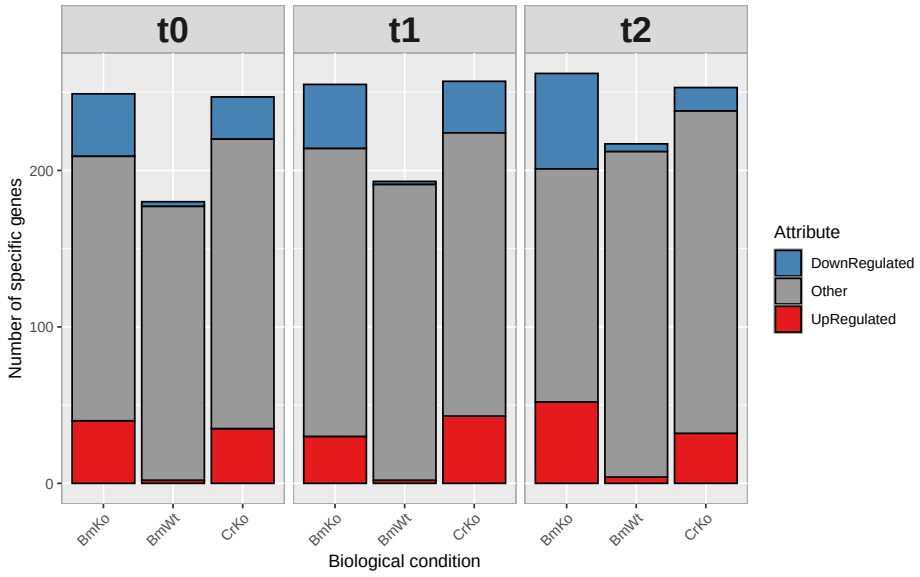
As we are in the similar case described in section Graphs from the results of the temporal statistical analysis, the results will not be described here.

6.3.1.2 Graphs from the results of the biological condition analysis

As we are in case 4, the results are more complex.

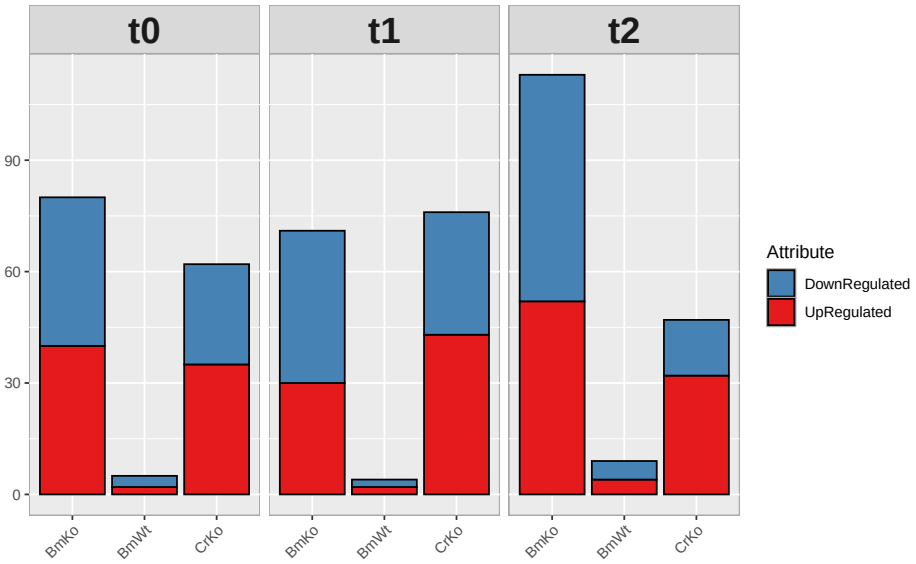
One barplot showing the number of specific genes per biological condition and no specific genes (category 'Other'), for each time.

```
resDEmus2$DEplots_GroupPerTime$NumberDEgenes_SpecificAndNoSpecific_perBiologicalCondition
```



One barplot showing the number of specific genes per biological condition, for each time. This is the same barplot than in section Graphs from the results of the biological condition analysis

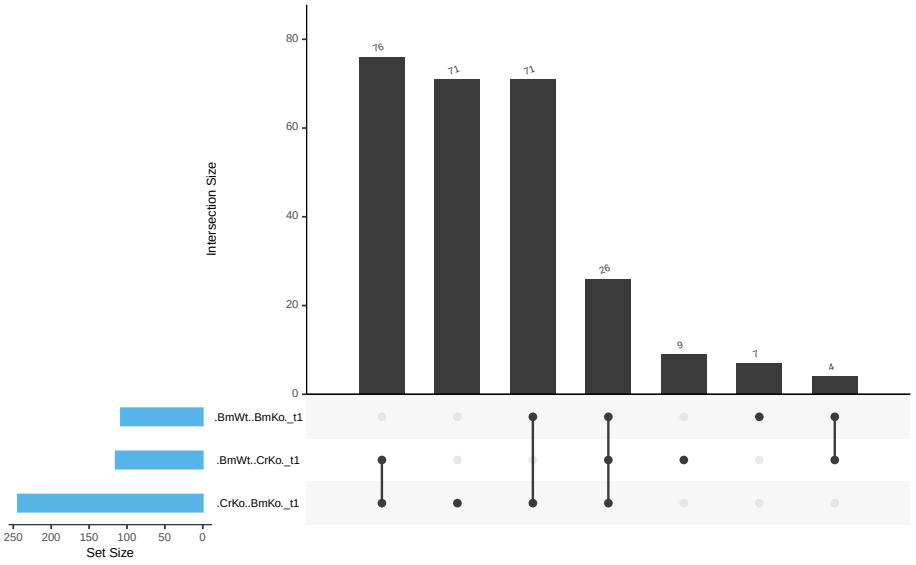
```
resDEmus2$DEplots_GroupPerTime$NumberSpecificGenes_UpDownRegulated_perBiologicalCondition
```



See Section Graphs from the results of the biological condition analysis for more information about specific genes.

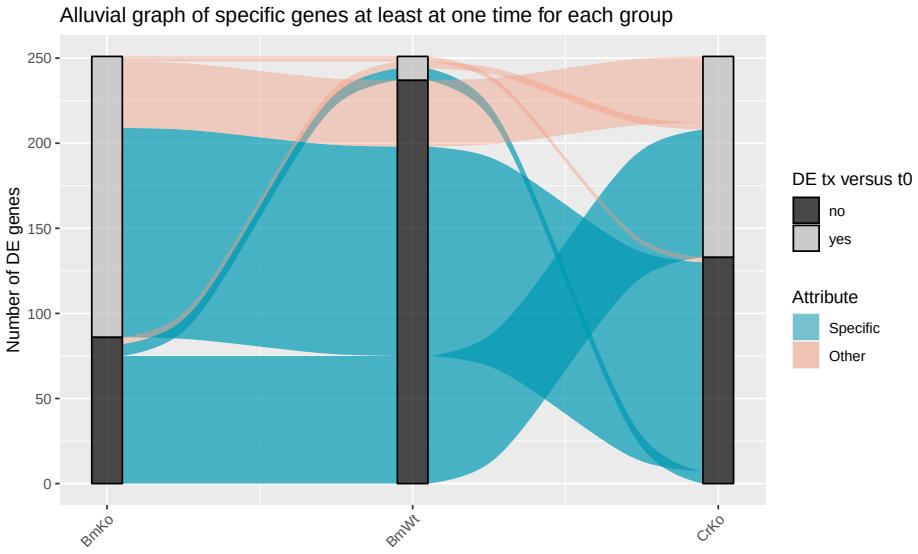
$\binom{N_{bc}}{2} = \binom{3}{2} = 3$ *upset graph* showing the number of genes corresponding to each possible intersection in a Venn barplot at a given time, only if there are more than two biological conditions (which the case here). We recall that a set of pairs of biological conditions forms an intersection at a given time t_i when there is at least one gene which is DE for each of these pairs of biological conditions at time t_i, but not for the others at time t_i. The following line of code plots the Venn barplot for the time t_1.

```
resDEmus2$DEplots_GroupPerTime$VennBarplot_BiologicalConditions_atTime.t1
```



alluvial graph showing the number of DE genes which are specific at least at one time for each group, only if there are more than two biological conditions (which is the case here).

```
resDEmus2$DEplots_GroupPerTime$AlluvialGraph_SpecificGenes1tmin_perBiologicalCondition
```

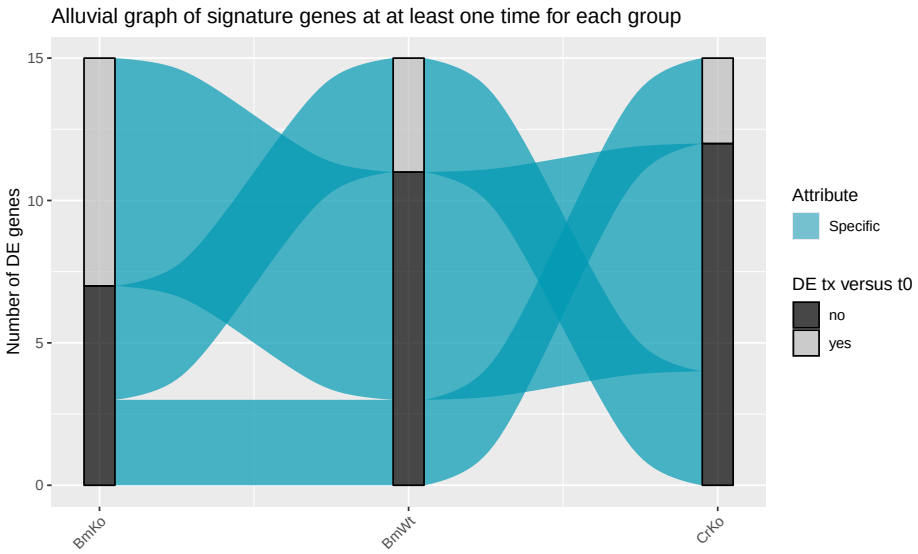


6.3.1.3 Graphs from the results of the combination of temporal and biological statistical analysis

From the combination of temporal and biological statistical analysis, the function plots the following graphs. The only difference with the section Graphs from the results of the combination of temporal and biological statistical analysis is the following graph.

One alluvial graph for DE genes which are signature at least at one time for each biological condition, only if there are more than two biological conditions (which is not the case here).

```
print(resDEmus2$DEplots_TimeAndGroup$Alluvial_SignatureGenes_1TimeMinimum_perGroup)
```



6.3.2 Volcano plots, ratio intensity (MA) plots and Heatmaps with DEplotVolcanoMA() and DEplotHeatmaps()

6.3.2.1 Volcano and MA plots (case 4)

The following lines of code allow to plot

- $\binom{N_{bc}}{2} \times T + (T-1) \times N_{bc} = \frac{N_{bc}(N_{bc}-1)}{2} \times T + (T-1) \times N_{bc} = 56$ volcano plots (with $N_{bc} = 4$ the number of biological conditions and $T = 6$ the number of time points).
- $\binom{N_{bc}}{2} \times T + (T-1) \times N_{bc} = 56$ MA plots.

allowing to separate non DE genes, DE genes below a threshold of log2 fold change and DE genes above a threshold of log2 fold change (see Section 3.3.2.1 for more details).

```
SEresVolcanoMAmus2 <- DEplotVolcanoMA(SEresDE=SEresDE3tMus2,
                                      NbGene.plotted=2,
                                      SizeLabel=3,
                                      Display.plots=FALSE,
                                      Save.plots=FALSE)
```

For more details, see Section 3.3.2.

If the user wants to save the graphs, the input `Save.plots` must be

- either `Save.plots=TRUE`, and the graph will be saved in the same location than the input `path.result` of the function `DEanalysisGlobal()`.
- or a strings of characters giving the path to a folder where the graphs will be saved. The user then chooses the path of the folder where results can be saved.

If the user wants to display the graph, he must set `Display.plots=TRUE`.

Write `?DEplotVolcanoMA` in your console for more information about the function.

6.3.2.2 Heatmaps (case 4)

The following lines of code allow to plot a correlation heatmap between samples (correlation heatmap) and a heatmap across samples and genes called Zscore heatmap, for a subset of genes that can be selected by the user. The second heatmap is build from the normalized count data after being both centered and scaled (Zscore).

```
SEresHeatmapMus2 <- DEplotHeatmaps(SEresDE=SEresDE3tMus2,
                                   ColumnsCriteria=2:5,
                                   Set.Operation="union",
                                   NbGene.analysis=20,
                                   SizeLabelRows=5,
                                   SizeLabelCols=5,
                                   Display.plots=FALSE,
                                   Save.plots=FALSE)
```

Both graphs are similar to those described in subsection Heatmaps (case 2) and Heatmaps (case 1).

For the Zscore heatmap, The subset of genes is selected as followss

1. the user selects one or more binary column of the data.frame `DEsummaryMus2` (see Section 6.3.1) with the input `ColumnsCriteria` which contains the column numbers of `DEsummaryMus2` to be selected.
2. Three cases are possible:
   - If `Set.Operation="union"` then the rows extracted from the different datasets (raw counts and normalized data and `DEsummaryMus2`) included in the SE object `SEresDE3tMus2` are those such that the sum of the selected columns of `DEsummaryMus2` given in `ColumnsCriteria` is >0. This means that the selected genes are those having at least one '1' in one of the selected columns.
   - If `Set.Operation="intersect"` then the rows extracted from the different datasets (raw counts and normalized data and `DEsummaryMus2`) included in the SE object `SEresDE3tMus2` are those such that the product of the selected columns of `DEsummaryMus2` given in `ColumnsCriteria` is >0. This means that the selected genes are those having '1' in all of the selected columns.
   - If `Set.Operation="setdiff"` then the rows extracted from the different datasets (raw counts and normalized data and `DEsummaryMus2`) included in the SE object `SEresDE3tMus2` are those such that only one element of the selected columns of `DEsummaryMus2` given in `ColumnsCriteria` is >0. This means that the selected genes are those having '1' in only one of the selected columns.
3. Finally, the selected subset of genes will be the `NbGene.analysis` genes extracted in step 2 above, which have the highest sum of absolute log2 fold change.

If the user wants to save the graphs, the input `Save.plots` must be

- either `Save.plots=TRUE`, and the graph will be saved in the same location than the input `path.result` of the function `DEanalysisGlobal()`.
- either a strings of characters giving the path to a folder where the graphs will be saved. The user then chooses the path of the folder where results can be saved.

If the user wants to display the graph, he must set `Display.plots=TRUE`.

Write `?DEplotHeatmaps` in your console for more information about the function.

6.4 Gene Ontology (GO) analysis with GSEAQuickAnalysis() and GSEApreprocessing()

6.4.1 Gene ontology with the R package gprofiler2

The lines of code below realize an enrichment analysis with the R package gprofiler2 for a selection of genes. Beware, an internet connection is needed. The function returns

- a data.frame (output `metadata(SEresGprofiler2Mus2)$Rgprofiler2$GSEAresults`) giving information about all detected gene ontologies for the list of associated genes.
- a lollipop graph (see section Gene ontology and gene enrichment). The y-axis indicates the `MaxNumberGO` most significant gene ontologies and pathways associated to the selected genes. The gene ontologies and patways are sorted into descending order. The x-axis indicates the $-log10(pvalues)$. The higher is a lollipop the more significant is a gene ontology or pathway. A lollipop is yellow if the pvalues is smaller than 0.05 (significant) and blue otherwise.
- A Manhattan plot (see section Gene ontology and gene enrichment) indicating all genes ontologies ordered according to the functional database (G0::BP, G0::CC, G0::MF and KEGG)

```
SEresGprofiler2Mus2 <- GSEAQuickAnalysis(Internet.Connection=FALSE,
                                         SEresDE=SEresDE3tMus2,
                                         ColumnsCriteria=2:5,
                                         ColumnsLog2ordered=NULL,
                                         Set.Operation="union",
                                         Organism="mmusculus",
                                         MaxNumberGO=20,
                                         Background=FALSE,
                                         Display.plots=FALSE,
                                         Save.plots=FALSE)
##
## head(SEresGprofiler2Mus2$GSEAresults)
```

As **GSEAQuickAnalysis()** requires an internet connection, we needed to add the input `Internet.Connection` in order to be sure to pass the tests realized on our package by Bioconductor. The input `Internet.Connection` is set by default to `FALSE` and as long as `Internet.Connection=FALSE`, no enrichment analysis will be done. Once the user is sure to have an internet connection, the user may set `Internet.Connection=TRUE` in order to realize the enrichment analysis.

The subset of genes is selected as follows

1. the user selects one or more binary column of the data.frame `DEsummaryMus2` (see Section 6.3.1) with the input `ColumnsCriteria` which contains the column numbers of `DEsummaryMus2` to be selected.
2. Three cases are possible:
   - If `Set.Operation="union"` then the rows extracted from the different datasets (raw counts and normalized data and `DEsummaryMus2`) included in the SE object `SEresDE3tMus2` are those such that the sum of the selected columns of `DEsummaryMus2` given in `ColumnsCriteria` is >0. This means that the selected genes are those having at least one '1' in one of the selected columns.
   - If `Set.Operation="intersect"` then the rows extracted from the different datasets (raw counts and normalized data and `DEsummaryMus2`) included in the SE object `SEresDE3tMus2` are those such that the product of the selected columns of `DEsummaryMus2` given in `ColumnsCriteria` is >0. This means that the selected genes are those having '1' in all of the selected columns.
   - If `Set.Operation="setdiff"` then the rows extracted from the different datasets (raw counts and normalized data and `DEsummaryMus2`) included in the SE object `SEresDE3tMus2` are those such that only one element of the selected columns of `DEsummaryMus2` given in `ColumnsCriteria` is >0. This means that the selected genes are those having '1' in only one of the selected columns.
3. Finally, the selected subset of genes will be the `NbGene.analysis` genes extracted in step 2 above, which have the highest sum of absolute log2 fold change.

If `ColumnsLog2ordered` is a vector of integers, the enrichment analysis will take into account the genes order as the first genes will be considered to have the highest biological importance and the last genes the lowest. It corresponds to the columns number of `DEsummaryMus2`, the output of `DEanalysisGlobal()`, which must contains log_2 fold change values. The rows of `DEsummaryMus2` (corresponding to genes) will be decreasingly ordered according to the sum of absolute log_2 fold change (the selected columns must contain log_2 fold change values) before the enrichment analysis. If `ColumnsLog2ordered=NULL`, then the enrichment analysis will not take into account the genes order.

If the user wants to save the graphs, the input `Save.plots` must be

- either `Save.plots=TRUE`, and the graph will be saved in the same location than the input `path.result` of the function `DEanalysisGlobal()`.
- or a strings of characters giving the path to a folder where the graphs will be saved. The user then chooses the path of the folder where results can be saved.

Write `?GSEAQuickAnalysis` in your console for more information about the function.

6.4.2 Preprocessing for GSEA, DAVID, WebGestalt, gProfiler, Panther, ShinyGO, Enrichr and GOrilla

The following lines of code will generate all files, for a selection of genes, in order to use the following software and online tools : GSEA, DAVID, WebGestalt, gProfiler, Panther, ShinyGO, Enrichr and GOrilla.

```
SEresPreprocessingMus2 <- GSEApreprocessing(SEresDE=SEresDE3tMus2,
                                            ColumnsCriteria=2:5,
                                            Set.Operation="union",
                                            Rnk.files=FALSE,
                                            Save.files=TRUE)
```

The subset of genes is selected as follows

1. the user selects one or more binary column of the data.frame `DEsummaryMus2` (see Section 6.3.1) with the input `ColumnsCriteria` which contains the column numbers of `DEsummaryMus2` to be selected.
2. Three cases are possible:
   - If `Set.Operation="union"` then the rows extracted from the different datasets (raw counts and normalized data and `DEsummaryMus2`) included in the SE object `SEresDE3tMus2` are those such that the sum of the selected columns of `DEsummaryMus2` given in `ColumnsCriteria` is >0. This means that the selected genes are those having at least one '1' in one of the selected columns.
   - If `Set.Operation="intersect"` then the rows extracted from the different datasets (raw counts and normalized data and `DEsummaryMus2`) included in the SE object `SEresDE3tMus2` are those such that the product of the selected columns of `DEsummaryMus2` given in `ColumnsCriteria` is >0. This means that the selected genes are those having '1' in all of the selected columns.
   - If `Set.Operation="setdiff"` then the rows extracted from the different datasets (raw counts and normalized data and `DEsummaryMus2`) included in the SE object `SEresDE3tMus2` are those such that only one element of the selected columns of `DEsummaryMus2` given in `ColumnsCriteria` is >0. This means that the selected genes are those having '1' in only one of the selected columns.
3. Finally, the selected subset of genes will be the `NbGene.analysis` genes extracted in step 2 above, which have the highest sum of absolute log2 fold change.

If the user wants to save the files, the input `Save.plots` must be

- either `Save.plots=TRUE`, and the graph will be saved in the same location than the input `path.result` of the function `DEanalysisGlobal()`.
- or a strings of characters giving the path to a folder where the graphs will be saved. The user then chooses the path of the folder where results can be saved.

Write `?GSEApreprocessing` in your console for more information about the function.

7 Session info

Here is the output of `sessionInfo()` on the system on which this document was compiled.

- R Under development (unstable) (2025-10-20 r88955), `x86_64-pc-linux-gnu`
- Locale: `LC_CTYPE=en_US.UTF-8`, `LC_NUMERIC=C`, `LC_TIME=en_GB`, `LC_COLLATE=C`, `LC_MONETARY=en_US.UTF-8`, `LC_MESSAGES=en_US.UTF-8`, `LC_PAPER=en_US.UTF-8`, `LC_NAME=C`, `LC_ADDRESS=C`, `LC_TELEPHONE=C`, `LC_MEASUREMENT=en_US.UTF-8`, `LC_IDENTIFICATION=C`
- Time zone: `America/New_York`
- TZcode source: `system (glibc)`
- Running under: `Ubuntu 24.04.3 LTS`
- Matrix products: default
- BLAS: `/home/biocbuild/bbs-3.23-bioc/R/lib/libRblas.so`
- LAPACK: `/usr/lib/x86_64-linux-gnu/lapack/liblapack.so.3.12.0`
- Base packages: base, datasets, grDevices, graphics, methods, stats, tcltk, utils
- Other packages: Biobase 2.71.0, BiocGenerics 0.57.0, BiocStyle 2.39.0, DynDoc 1.89.0, Mfuzz 2.71.0, MultiRNAflow 1.9.0, e1071 1.7-16, generics 0.1.4, knitr 1.50, widgetTools 1.89.0
- Loaded via a namespace (and not attached): BiocManager 1.30.26, BiocParallel 1.45.0, Cairo 1.7-0, ComplexHeatmap 2.27.0, DESeq2 1.51.0, DT 0.34.0, DelayedArray 0.37.0, FactoMineR 2.12, Formula 1.2-5, GenomicRanges 1.63.0, GetoptLong 1.0.5, GlobalOptions 0.1.2, IRanges 2.45.0, MASS 7.3-65, Matrix 1.7-4, MatrixGenerics 1.23.0, R6 2.6.1, RColorBrewer 1.1-3, Rcpp 1.1.0, S4Arrays 1.11.0, S4Vectors 0.49.0, S7 0.2.0, Seqinfo 1.1.0, SparseArray 1.11.1, SummarizedExperiment 1.41.0, TH.data 1.1-4, UpSetR 1.4.0, XVector 0.51.0, abind 1.4-8, backports 1.5.0, base64enc 0.1-3, bookdown 0.45, broom 1.0.10, bslib 0.9.0, cachem 1.1.0, car 3.1-3, carData 3.0-5, circlize 0.4.16, class 7.3-23, cli 3.6.5, clue 0.3-66, cluster 2.1.8.1, coda 0.19-4.1, codetools 0.2-20, colorspace 2.1-2, compiler 4.6.0, crayon 1.5.3, data.table 1.17.8, dendextend 1.19.1, dichromat 2.0-0.1, digest 0.6.37, doParallel 1.0.17, dplyr 1.1.4, emmeans 2.0.0, estimability 1.5.1, evaluate 1.0.5, factoextra 1.0.7, farver 2.1.2, fastmap 1.2.0, flashClust 1.01-2, foreach 1.5.2, fs 1.6.6, ggalluvial 0.12.5, ggplot2 4.0.0, ggplotify 0.1.3, ggpubr 0.6.2, ggrepel 0.9.6, ggsignif 0.6.4, glue 1.8.0, gprofiler2 0.2.3, grid 4.6.0, gridExtra 2.3, gridGraphics 0.5-1, gtable 0.3.6, highr 0.11, htmltools 0.5.8.1, htmlwidgets 1.6.4, httr 1.4.7, iterators 1.0.14, jquerylib 0.1.4, jsonlite 2.0.0, labeling 0.4.3, lattice 0.22-7, lazyeval 0.2.2, leaps 3.2, lifecycle 1.0.4, locfit 1.5-9.12, magick 2.9.0, magrittr 2.0.4, matrixStats 1.5.0, misc3d 0.9-1, multcomp 1.4-29, multcompView 0.1-10, mvtnorm 1.3-3, parallel 4.6.0, pillar 1.11.1, pkgconfig 2.0.3, plot3D 1.4.2, plot3Drgl 1.0.5, plotly 4.11.0, plyr 1.8.9, png 0.1-8, proxy 0.4-27, purrr 1.1.0, rappdirs 0.3.3, reshape2 1.4.4, rgl 1.3.24, rjson 0.2.23, rlang 1.1.6, rmarkdown 2.30, rstatix 0.7.3, sandwich 3.1-1, sass 0.4.10, scales 1.4.0, scatterplot3d 0.3-44, shape 1.4.6.1, splines 4.6.0, stats4 4.6.0, stringi 1.8.7, stringr 1.5.2, survival 3.8-3, tibble 3.3.0, tidyr 1.3.1, tidyselect 1.2.1, tinytex 0.57, tkWidgets 1.89.0, tools 4.6.0, vctrs 0.6.5, viridis 0.6.5, viridisLite 0.4.2, withr 3.0.2, xfun 0.54, xtable 1.8-4, yaml 2.3.10, yulab.utils 0.2.1, zoo 1.8-14

References

[1] Nir Yosef, Alex K. Shalek, Jellert T. Gaublomme, Hulin Jin, Youjin Lee, Amit Awasthi, Chuan Wu, Katarzyna Karwacz, Sheng Xiao, Marsela Jorgolli, David Gennert, Rahul Satija, Arvind Shakya, Diana Y. Lu, John J. Trombetta, Meenu R. Pillai, Peter J. Ratcliffe, Mathew L. Coleman, Mark Bix, Dean Tantin, Hongkun Park, Vijay K. Kuchroo, and Aviv Regev. Dynamic regulatory network controlling TH17 cell differentiation. 496(7446):461–468, 2013. doi:10.1038/nature11981.

[2] Ziv Bar-Joseph, Anthony Gitter, and Itamar Simon. Studying and modelling dynamic biological processes using time-series gene expression data. 13(8):552–564, 2012. doi:10.1038/nrg3244.

[3] Shanrong Zhao, Zhan Ye, and Robert Stanton. Misuse of RPKM or TPM normalization when comparing across samples and sequencing protocols. 26(8):903–909, 2020. doi:10.1261/rna.074922.120.

[4] Gunter P. Wagner, Koryu Kin, and Vincent J. Lynch. Measurement of mRNA abundance using RNA-seq data: RPKM measure is inconsistent among samples. 131(4):281–285, 2012. doi:10.1007/s12064-012-0162-3.

[5] Michael I Love, Wolfgang Huber, and Simon Anders. Moderated estimation of fold change and dispersion for RNA-seq data with DESeq2. 15(12):550, 2014. doi:10.1186/s13059-014-0550-8.

[6] Mark D. Robinson, Davis J. McCarthy, and Gordon K. Smyth. edgeR: a bioconductor package for differential expression analysis of digital gene expression data. 26(1):139–140, 2010. doi:10.1093/bioinformatics/btp616.

[7] Ali Mortazavi, Brian A Williams, Kenneth McCue, Lorian Schaeffer, and Barbara Wold. Mapping and quantifying mammalian transcriptomes by RNA-seq. 5(7):621–628, 2008. doi:10.1038/nmeth.1226.

[8] Bo Li, Victor Ruotti, Ron M. Stewart, James A. Thomson, and Colin N. Dewey. RNA-seq gene expression estimation with read mapping uncertainty. 26(4):493–500, 2010. doi:10.1093/bioinformatics/btp692.

[9] Federico Marini, Jan Linke, and Harald Binder. ideal: an r/bioconductor package for interactive differential expression analysis. 21(1):565, 2020. doi:10.1186/s12859-020-03819-5.

[10] Kuan-Hao Chao, Yi-Wen Hsiao, Yi-Fang Lee, Chien-Yueh Lee, Liang-Chuan Lai, Mong-Hsun Tsai, Tzu-Pin Lu, and Eric Y. Chuang. RNASeqR: An r package for automated two-group RNA-seq analysis workflow. 18(5):2023–2031, 2021. doi:10.1109/TCBB.2019.2956708.

[11] Xi Wang and Murray J. Cairns. SeqGSEA: a Bioconductor package for gene set enrichment analysis of RNA-Seq data integrating differential expression and splicing. *Bioinformatics (Oxford, England)*, 30(12):1777–1779, June 2014. doi:10.1093/bioinformatics/btu090.

[12] Marie Lataretu and Martin Hölzer. RNAflow: An effective and simple RNA-seq differential gene expression pipeline using nextflow. 11(12):1487, 2020. doi:10.3390/genes11121487.

[13] Rodolphe Loubaton, Nicolas Champagnat, Pierre Vallois, and Laurent Vallat. MultiRNAflow: integrated analysis of temporal RNA-seq data with multiple biological conditions. *Bioinformatics*, 40(5):btae315, May 2024. doi:10.1093/bioinformatics/btae315.

[14] Cedric Schleiss, Raphael Carapito, Luc-Matthieu Fornecker, Leslie Muller, Nicodeme Paul, Ouria Tahar, Angelique Pichot, Manuela Tavian, Alina Nicolae, Laurent Miguet, Laurent Mauvieux, Raoul Herbrecht, Sarah Cianferani, Jean-Noel Freund, Christine Carapito, Myriam Maumy-Bertrand, Seiamak Bahram, Frederic Bertrand, and Laurent Vallat. Temporal multiomic modeling reveals a b-cell receptor proliferative program in chronic lymphocytic leukemia. 35(5):1463–1474, 2021. doi:10.1038/s41375-021-01221-5.

[15] Benjamin D. Weger, Cedric Gobet, Fabrice P. A. David, Florian Atger, Eva Martin, Nicholas E. Phillips, Aline Charpagne, Meltem Weger, Felix Naef, and Frederic Gachon. Systematic analysis of differential rhythmic liver gene expression mediated by the circadian clock and feeding rhythms. 118(3):e2015803118, 2021. doi:10.1073/pnas.2015803118.

[16] Sebastien Le, Julie Josse, and François Husson. **FactoMineR** : An *R* package for multivariate analysis. 25(1), 2008. doi:10.18637/jss.v025.i01.

[17] Zuguang Gu, Roland Eils, and Matthias Schlesner. Complex heatmaps reveal patterns and correlations in multidimensional genomic data. 32(18):2847–2849, 2016. doi:10.1093/bioinformatics/btw313.

[18] Matthias E. Futschik and Bronwyn Carlisle. Noise-robust soft clustering of gene expression time-course data. 03(4):965–988, 2005. doi:10.1142/S0219720005001375.

[19] Lokesh Kumar and Matthias E. Futschik. Mfuzz: A software package for soft clustering of microarray data. 2(1):5–7, 2007. doi:10.6026/97320630002005.

[20] Liis Kolberg, Uku Raudvere, Ivan Kuzmin, Jaak Vilo, and Hedi Peterson. gprofiler2 – an r package for gene list functional enrichment analysis and namespace conversion toolset g:profiler. 9:709, 2020. doi:10.12688/f1000research.24956.2.

[21] Brad T Sherman, Ming Hao, Ju Qiu, Xiaoli Jiao, Michael W Baseler, H Clifford Lane, Tomozumi Imamichi, and Weizhong Chang. DAVID: a web server for functional enrichment analysis and functional annotation of gene lists (2021 update). 50:W216–W221, 2022. doi:10.1093/nar/gkac194.

[22] Yuxing Liao, Jing Wang, Eric J Jaehnig, Zhiao Shi, and Bing Zhang. WebGestalt 2019: gene set analysis toolkit with revamped UIs and APIs. 47:W199–W205, 2019. doi:10.1093/nar/gkz401.

[23] Aravind Subramanian, Pablo Tamayo, Vamsi K. Mootha, Sayan Mukherjee, Benjamin L. Ebert, Michael A. Gillette, Amanda Paulovich, Scott L. Pomeroy, Todd R. Golub, Eric S. Lander, and Jill P. Mesirov. Gene set enrichment analysis: A knowledge-based approach for interpreting genome-wide expression profiles. 102(43):15545–15550, 2005. doi:10.1073/pnas.0506580102.

[24] Uku Raudvere, Liis Kolberg, Ivan Kuzmin, Tambet Arak, Priit Adler, Hedi Peterson, and Jaak Vilo. g:profiler: a web server for functional enrichment analysis and conversions of gene lists (2019 update). 47:W191–W198, 2019. doi:10.1093/nar/gkz369.

[25] Paul D. Thomas, Dustin Ebert, Anushya Muruganujan, Tremayne Mushayahama, Laurent-Philippe Albou, and Huaiyu Mi. Panther: Making genome-scale phylogenetics accessible to all. 31(1):8–22, 2022. doi:10.1002/pro.4218.

[26] Steven Xijin Ge, Dongmin Jung, and Runan Yao. ShinyGO: a graphical gene-set enrichment tool for animals and plants. 36(8):2628–2629, 2020. doi:10.1093/bioinformatics/btz931.

[27] Maxim V. Kuleshov, Matthew R. Jones, Andrew D. Rouillard, Nicolas F. Fernandez, Qiaonan Duan, Zichen Wang, Simon Koplev, Sherry L. Jenkins, Kathleen M. Jagodnik, Alexander Lachmann, Michael G. McDermott, Caroline D. Monteiro, Gregory W. Gundersen, and Avi Ma'ayan. Enrichr: a comprehensive gene set enrichment analysis web server 2016 update. 44:W90–W97, 2016. doi:10.1093/nar/gkw377.

[28] Eran Eden, Roy Navon, Israel Steinfeld, Doron Lipson, and Zohar Yakhini. GOrilla: a tool for discovery and visualization of enriched GO terms in ranked gene lists. 10(1):48, 2009. doi:10.1186/1471-2105-10-48.

[29] Simon Anders and Wolfgang Huber. Differential expression analysis for sequence count data. 11(10):R106, 2010. doi:10.1186/gb-2010-11-10-r106.

[30] R Core Team. R: a language and environment for statistical computing, 2021. URL: https://www.R-project.org/.

[31] Mateusz Antoszewski, Nadine Fournier, Gustavo A. Ruiz Buendia, Joao Lourenco, Yuanlong Liu, Tara Sugrue, Christelle Dubey, Marianne Nkosi, Colin E. J. Pritchard, Ivo J. Huijbers, Gabriela C. Segat, Sandra Alonso-Moreno, Elisabeth Serracanta, Laura Belver, Adolfo A. Ferrando, Giovanni Ciriello, Andrew P. Weng, Ute Koch, and Freddy Radtke. Tcf1 is essential for initiation of oncogenic notch1-driven chromatin topology in t-ALL. 139(16):2483–2498, 2022. doi:10.1182/blood.2021012077.

[32] Hui Sun Leong, Keren Dawson, Chris Wirth, Yaoyong Li, Yvonne Connolly, Duncan L. Smith, Caroline R. M. Wilkinson, and Crispin J. Miller. A global non-coding RNA system modulates fission yeast protein levels in response to stress. 5(1):3947, 2014. doi:10.1038/ncomms4947.

[33] Jake R Conway, Alexander Lex, and Nils Gehlenborg. UpSetR: an r package for the visualization of intersecting sets and their properties. 33(18):2938–2940, 2017. doi:10.1093/bioinformatics/btx364.