Package 'profileplyr'

April 6, 2026

Type Package

Title Visualization and annotation of read signal over genomic ranges with profileplyr

Version 1.26.1

Date 2025-07-22

Author Tom Carroll and Doug Barrows

Maintainer
Tom Carroll <tc.infomatics@gmail.com>, Doug Barrows <doug.barrows@gmail.com>

Depends R (>= 3.6), BiocGenerics, SummarizedExperiment

Description
Quick and straightforward visualization of read signal over genomic intervals is key for generating hypotheses from sequencing data sets (e.g. ChIP-seq, ATAC-seq, bisulfite/methyl-seq). Many tools both inside
and outside of R and Bioconductor are available to explore these types of data, and they typically start
with a bigWig or BAM file and end with some representation of the signal (e.g. heatmap). profileplyr leverages
many Bioconductor tools to allow for both flexibility and additional functionality in workflows that end with
visualization of the read signal.

License GPL (>= 3)

RoxygenNote 7.3.2

biocViews ChIPSeq, DataImport, Sequencing, ChipOnChip, Coverage

Imports GenomicRanges, stats, soGGi, methods, utils, S4Vectors, R.utils, dplyr, magrittr, tidyr, IRanges, rjson, ChIPseeker,GenomicFeatures,TxDb.Hsapiens.UCSC.hg19.knownGene,TxDb.Hsapiens.UCSC.hg38.knownGene,TxD
TxDb.Mmusculus.UCSC.mm9.knownGene,org.Hs.eg.db,org.Mm.eg.db,rGREAT, pheatmap, ggplot2, EnrichedHeatmap, ComplexHeatmap, grid, circlize, BiocParallel, rtracklayer, Seqinfo, grDevices, rlang, tiff, Rsamtools

Suggests GenomeInfoDb, BiocStyle, testthat, knitr, rmarkdown, png, Cairo

VignetteBuilder knitr

Encoding UTF-8

git_url https://git.bioconductor.org/packages/profileplyr

git_branch RELEASE_3_22

git_last_commit 009cf8c

git_last_commit_date 2025-11-19

Repository Bioconductor 3.22

Date/Publication 2026-04-05

Contents

`annotateRanges` *Annotate profileplyr ranges to genes using ChIPseeker*

Description

The ranges from the deepTools matrix will be subset based on whether they overlap with specified annotated regions (using ChIPseeker) throughout the genome

Usage

```
annotateRanges(
  object = "profileplyr",
  annotation_subset = "character",
  TxDb,
  annoDb = "character",
  tssRegion = "numeric",
  changeGroupToAnnotation = "logical",
```

```
  heatmap_grouping = "character",
  ...
)

## S4 method for signature 'profileplyr'
annotateRanges(
  object = "profileplyr",
  annotation_subset = NULL,
  TxDb = NULL,
  annoDb = NULL,
  tssRegion = c(-3000, 3000),
  changeGroupToAnnotation = FALSE,
  heatmap_grouping = "group",
  ...
)
```

Arguments

object	A profileplyr object
annotation_subset	If specific annotations (from ChIPseeker package) are desired, specify them here in a character vector. Can be one or any combination of "Promoter", "Exon", "Intron", "Downstream", "Distal Intergenic", "3p UTR", or "5p UTR". This argument is optional and all annotation types will be included if argument is left out.
TxDb	This must be either a TxDb object, a character string that is a path to a GTF file, or character string indicating genome if one of the following - "hg19", "hg38", "mm9", "mm10".
annoDb	The annotation package to be used. If the 'TxDb' argument is set to "hg19", "hg38", "mm9", or "mm10" this will automatically be set and this can be left as NULL.
tssRegion	This needs to be a vector of two numbers that will define promoter regions. The first number must be negative, while the second number must be positive. Default values are c(-3000, 3000)
changeGroupToAnnotation	If the grouping should be changed to the annotations (typically when the ranges will be exported for visualization based on this annotation), this should be TRUE. The default if FALSE, which will keep the grouping that existed before annotating the object. This is typical if the output will be used for finding overlaps with gene lists in the 'groupBy' function.
heatmap_grouping	Only relevant if 'keepAnnotationAsGroup' is set to TRUE. This argument needs to be either "group", or "annotation". This will determine how the ranges are grouped in the resulting object. Default is heatmap_grouping = "Group". If there are no groups in the deepTools matrix that was used in the function, this argument is unnecessary
...	pass to annotatePeak

Details

tbd

Value

A profileplyr object

Methods (by class)

- `annotateRanges(profileplyr)`: Annotate profileplyr ranges to genes using ChIPseeker

Examples

```
library(SummarizedExperiment)
example <- system.file("extdata", "example_deepTools_MAT", package = "profileplyr")
object <- import_deepToolsMat(example)
object <- object[1:2, , ]

# annotate ranges with genes using ChIPseeker
# (NOTE: can choose subset of annotations with 'annotation_subset' argument)

annotateRanges(object, TxDb = "mm10")
```

---

annotateRanges_great *Annotate profileplyr ranges to genes using rGREAT*

Description

The ranges from the deepTools matrix will be subset based on whether they overlap with specified annotated regions related to a user defined gene list.

Usage

```
annotateRanges_great(object = "profileplyr", species = "character", ...)

## S4 method for signature 'profileplyr'
annotateRanges_great(object = "profileplyr", species = "character", ...)
```

Arguments

`object`	A profileplyr object
`species`	GREAT accepts "hg19", "mm10", "mm9", "danRer7" (zebrafish)
`...`	pass to `submitGreatJob`

Details

tbd

Value

A profileplyr object

Methods (by class)

- `annotateRanges_great(profileplyr)`: Annotate profileplyr ranges to genes using rGREAT

Examples

```
library(SummarizedExperiment)
example <- system.file("extdata", "example_deepTools_MAT", package = "profileplyr")
object <- import_deepToolsMat(example)
object <- object[1:5, , ]

# annotate ranges with genes using GREAT with following command:
annotateRanges_great(object, species = "mm10")
```

`as_profileplyr` *Import ChIPprofile object to profileplyr*

Description

Function to convert soGGi ChIPprofile objects to profileplyr object .

Usage

```
as_profileplyr(chipProfile, names = NULL)
```

Arguments

`chipProfile`	A ChIPprofile object as created by soGGi regionPlot() function.
`names`	Column to select row IDs/names from ChIPprofile mcols.

Value

A profileplyr object

Examples

```
library(soGGi)
data("ik_Profiles")
proplyr <- as_profileplyr(ik_Profiles,names="ID")
export_deepToolsMat(proplyr,con=file.path(tempdir(),"ik_Profiles.MAT"))
```

`BamBigwig_to_chipProfile`
BamBigwig_to_chipProfile

Description

Generate a soGGi ChIPprofile object with multiple BAM/bigWig files or multiple BED files as the input

Usage

```
BamBigwig_to_chipProfile(
  signalFiles,
  testRanges,
  format,
  style = "percentOfRegion",
  nOfWindows = 100,
  bin_size = 20,
  distanceAround = NULL,
  distanceUp = 1000,
  distanceDown = 1000,
  ...,
  quant_params = NULL
)
```

Arguments

signalFiles	paths to either BAM files or bigwig files. More than one path can be in this character vector, but all paths in one function call must point to be either all BAM files or all bigWig files, not a combination of the two.
testRanges	A character vector with paths to BED files.
format	character string of "bam", "bigwig", "RleList" or "PWM"
style	a character string, "percentOfRegion" (default) for normalized length divided into bins set by the 'nOfWindows' argument, "point" for per base pair plot where the number of base pairs per bin is set by the 'bin_size' argument, and "region" for combined plot
nOfWindows	The number of windows/bins the normalised ranges will be divided into if 'style' is set to 'percentOfRegion'. Default is 100.
bin_size	If 'style' is set to 'point' then this will determine the size of each bin over which signal is quantified. The default is 20 base pairs.
distanceAround	This controls the distance around the region that is included. If 'style' is 'percentOfRegion', then the default is 100, meaning that a distance equal to 100 percent of that particular region on either side of the region will be included in the heatmap. If 'style' is 'point',then this is the number of basepairs from the center of each range, in either direction, that the heatmap will show. If style is 'point' and 'distanceAround' is NULL, then distanceUp and distanceDown will be used.
distanceUp	If 'style' is set to 'point' then this will determine the distance (in base pairs) upstream from the center of each peak signal will be quantified. If the 'distanceAround' argument is set (i.e. not NULL), that will be used for the quantification range and 'distanceUp will be ignored.
distanceDown	If 'style' is set to 'point' then this will determine the distance (in base pairs) downstream from the center of each peak signal will be quantified. If the 'distanceAround' argument is set (i.e. not NULL), that will be used for the quantification range and 'distanceDown' will be ignored.
...	pass to regionPlot() within the soGGi package
quant_params	An optional BiocParallelParam instance determining the parallel back-end to be used during evaluation. When this argument is set to NULL (default) SerialParam() will be used. For parallelization, MulticoreParam() can be used.

Value

A profileplyr object

Examples

```
signalFiles <- c(system.file("extdata",
                             "Sorted_Hindbrain_day_12_1_filtered.bam",
                              package = "profileplyr"))
require(Rsamtools)
for (i in seq_along(signalFiles)){
 indexBam(signalFiles[i])
}
testRanges <- system.file("extdata",
                          "newranges_small.bed",
                          package = "profileplyr")
BamBigwig_to_chipProfile(signalFiles,
                         testRanges,
                         format = "bam",
                         paired=FALSE,
                         style="percentOfRegion",
                         )
```

---

clusterRanges *Cluster Ranges*

Description

Cluster the ranges in a deepTools object based on signal within each range

Usage

```
clusterRanges(
  object = "profileplyr",
  fun = "function",
  scaleRows = "logical",
  kmeans_k = "integer",
  clustering_callback = "function",
  clustering_distance_rows = "ANY",
  cluster_method = "function",
  cutree_rows = "integer",
  silent = "logical",
  show_rownames = "logical",
  cluster_sample_subset = "ANY"
)

## S4 method for signature 'profileplyr'
clusterRanges(
  object = "profileplyr",
  fun = rowMeans,
  scaleRows = TRUE,
  kmeans_k = NULL,
  clustering_callback = function(x, ...) {
```

```
    return(x)
 },
  clustering_distance_rows = "euclidean",
  cluster_method = "complete",
  cutree_rows = NULL,
  silent = TRUE,
  show_rownames = FALSE,
  cluster_sample_subset = NULL
)
```

Arguments

object	A profileplyr object
fun	The function used to summarize the ranges (e.g. rowMeans or rowMax). This is ignored when only one sample is used for clustering; in this case the lone heatmap is clustered based the signal across the bins.
scaleRows	If TRUE, the rows of the matrix containing the signal in each bin that is used as the input for clustering will be scaled (as specified by pheatmap)
kmeans_k	The number of kmeans groups used for clustering
clustering_callback	Clustering callback function to be passed to pheatmap
clustering_distance_rows	distance measure used in clustering rows. Possible values are "correlation" for Pearson correlation and all the distances supported by dist, such as "euclidean", etc. If the value is none of the above it is assumed that a distance matrix is provided.
cluster_method	clustering method used. Accepts the same values as hclust
cutree_rows	The number of clusters for hierarchical clustering
silent	Whether or not a heatmap (from pheatmap) is shown with the output. This will not change what is returned with the function as it will always be a profileplyr object. If silent = FALSE, the heatmap will be shown which may be helpful in quick evaluation of varying numbers of clusters before proceeding with downstream analysis. The default is silent = TRUE, meaning no heatmap will be shown.
show_rownames	for any heatmaps printed while running this function, set to TRUE if rownames should be displayed. Default is FALSE.
cluster_sample_subset	Either a character or numeric vector indicating the subset of heatmaps to be used for clustering. If a character vector, all elements of the vector must match names of the samples of the profileplyr object (found with 'rownames(sampleData(object))'). For an numeric vector, the profileplyr object will be subset based on the samples that correspond to these numbers (i.e. the numeric index of that sample within 'rownames(sampleData(object))'). When only sample is chosen, the lone heatmap selected will be clustered by signal across the bins of that sample. When more than one sample are selected, the 'fun' argument will be used to summarize the ranges and cluster across these selected samples.

Details

tbd

Value

A profileplyr object

Methods (by class)

- `clusterRanges(profileplyr)`: Cluster Ranges

Examples

```
example <- system.file("extdata", "example_deepTools_MAT", package = "profileplyr")
object <- import_deepToolsMat(example)

# k-means clustering
clusterRanges(object, fun = rowMeans, kmeans_k = 3)

# hierarchical clustering, print heatmap, yet still return profileplyr object
clusterRanges(object, fun = rowMeans, cutree_rows = 3, silent = FALSE)
```

---

convertToEnrichedHeatmapMat

export a profileplyr object to a list of matrices that can be used as an input for EnrichedHeatmap

---

Description

export a profileplyr object to a list of matrices that can be used as an input for EnrichedHeatmap

Usage

```
convertToEnrichedHeatmapMat(object = "profileplyr", sample_names = "character")

## S4 method for signature 'profileplyr'
convertToEnrichedHeatmapMat(object = "profileplyr", sample_names = NULL)
```

Arguments

`object`	A profileplyr object
`sample_names`	A character vector that will set the names of the heatmap components that are generated from the profileplyr assays() matrices. This argument is optional, by default the names will be the name of the samples in the profileplyr object rownames(sampleData(object)).

Details

Takes a profileplyr object and converts all of the matrices in the assays() section of the object to matrices that can be used as an input for EnrichedHeatmap

Value

A list of normalized matrices that can be used for generating visualizations with EnrichedHeatmap

Methods (by class)

- `convertToEnrichedHeatmapMat(profileplyr)`: export a profileplyr object to a list of matrices that can be used as an input for EnrichedHeatmap

Examples

```
example <- system.file("extdata", "example_deepTools_MAT", package = "profileplyr")
object <- import_deepToolsMat(example)

library(EnrichedHeatmap)
EH_mat <- convertToEnrichedHeatmapMat(object)
EnrichedHeatmap(EH_mat[[1]], name = names(EH_mat[1]), column_title = names(EH_mat[1]))
```

---

export_deepToolsMat *Export and import profileplyr from/to deeptools*

Description

Export and Import files

Usage

```
export_deepToolsMat(
  object = "profileplyr",
  con = "character",
  decreasing = "logical",
  overwrite = "logical"
)

## S4 method for signature 'profileplyr'
export_deepToolsMat(
  object = "profileplyr",
  con = "character",
  decreasing = FALSE,
  overwrite = FALSE
)

import_deepToolsMat(con)
```

Arguments

`object`	A profileplyr object
`con`	Connection to write/read deeptools data to/from.
`decreasing`	If object@params$mcolToOrderBy has been set and not NULL, then the ranges will be ordered by the column indicated in this slot of the metadata. By default, the order will be increasing for the factor or numeric value. For decreasing order, choose decreasing = TRUE.
`overwrite`	Logical specifying whether to overwrite output if it exists.

Details

A profileplyr object

Value

The path to deepTools matrix file

A profileplyr object

Methods (by class)

- `export_deepToolsMat(profileplyr)`: Export and import profileplyr from/to deeptools

Examples

```
example <- system.file("extdata", "example_deepTools_MAT", package = "profileplyr")
object <- import_deepToolsMat(example)
export_deepToolsMat(object,file.path(tempdir(),"ATAC_Example.MAT"))
```

---

generateEnrichedHeatmap

generateEnrichedHeatmap

Description

export a profileplyr object directly to an object of the EnrichedHeatmap class

Usage

```
generateEnrichedHeatmap(
  object,
  include_group_annotation = TRUE,
  extra_annotation_columns = NULL,
  sample_names = NULL,
  return_ht_list = FALSE,
  ylim = "common_max",
  top_anno_height = unit(2, "cm"),
  samples_to_sortby = NULL,
  decreasing = FALSE,
  all_color_scales_equal = TRUE,
  matrices_color = NULL,
  color_by_sample_group = NULL,
  matrices_pos_line = FALSE,
  matrices_pos_line_gp = gpar(lty = 2),
  top_anno_axis_font = gpar(fontsize = 8),
  matrices_column_title_gp = gpar(fontsize = 8, fontface = "bold"),
  matrices_axis_name = NULL,
  matrices_axis_name_gp = gpar(fontsize = 8),
  group_anno_color = NULL,
  group_anno_width = 3,
  group_anno_row_title_gp = gpar(fontsize = 10),
  group_anno_column_names_gp = gpar(fontsize = 10),
```

```
 extra_anno_color = vector(mode = "list", length = length(extra_annotation_columns)),
  extra_anno_top_annotation = TRUE,
  extra_anno_width = (rep(6, length(extra_annotation_columns))),
  only_extra_annotation_columns = FALSE,
  gap = 2,
  genes_to_label = NULL,
  gene_label_font_size = 6,
  show_heatmap_legend = NULL,
  legend_params = list(),
  use_raster = length(object) > 2000,
  raster_device = "CairoPNG",
  raster_quality = 2,
  raster_device_param = list()
)
```

Arguments

object	A profileplyr object
include_group_annotation	If TRUE (default value) then the Heatmap will be grouped based on the range metadata column specified by 'rowGroupsInUse'
extra_annotation_columns	A character vector of names that match column names of mcols(object). Extra annotation columns will be added to the heatmap based on the values of these indicated range metadata columns.
sample_names	A character vector that will set the names of the heatmap components that are generated from the profileplyr assays() matrices. This argument is optional, by default the names will be the name of the samples in the profileplyr object rownames(sampleData(object)).
return_ht_list	Whether the returned object is the heatmap list and not the actual figure. This will be a list of the various components (heatmaps and annotation columns) that can be added to with additional columns in a customized manner.
ylim	A numeric vector of two numbers that specifies the minimum and maximum of the yaxis of all the heatmaps generated for the matrices. The default is to use the max of the heatmap with the highest signal (ylim = 'common_max'). If ylim = NULL, different ranges will be inferred for each heatmap. If ylim is a single numeric vector, then that range will be used for all heatmaps. Different ranges can be set for each heatmap by making ylim a list that is the same length as the number of heatmaps/matrices, with each element of the list corresponding to each heatmap. Lastly, ylim can be a character string matching a column name in sampleData(object), and this will make the heatmaps with the same grouping have the same ylims as determined by the common max within groups.
top_anno_height	The height (as a unit object) of the top annotation of all heatmaps representing the matrices
samples_to_sortby	Only relevant if object@params$mcolToOrderBy is NULL (i.e it hasn't been changed), meaning that the rows are sorted by the mean signal of all heatmaps. This argument allows sorting by the mean of a subset of samples, and should be either a character or numeric vector. If numeric, then the samples/matrices that have that index in the profileplyr object will be used to order the rows of

the heatmap. If a character vector, then the elements must match the name of a sample in the object (rownames(sampleData(object))), and these samples will be used to order the heatmap.

`decreasing` If object@params$mcolToOrderBy has been changed and is not NULL, then the ranges will be ordered by the column indicated in this slot of the metadata. By default, the order will be increasing for the factor or numeric value. For decreasing order, choose decreasing = TRUE.

`all_color_scales_equal`
If TRUE (default value) then the same color scale will be used for each separate heatmap. If FALSE, color scales will be inferred for each heatmap as indicated by the legends.

`matrices_color` Either a single character vector, a numeric vector, a function call to colorRamp2 from the circlize package, or a list. For anything but a list, all the heatmaps generated for the matrices of the profileplyr object will be the same and will be colored as specified here. The character and numeric vector inputs must be either two or three elements in length (denoting color progressions - three elements will give a middle color break), and each element must be a character string or number that points to a color. By default, numeric vectors use the colors in palette(), however this can be expanded with other R color lists(e.g. colors()). If this argument is a list then it's length must equal the number of matrices/samples that exist in the input profileplyr object. The components of the list can be either a numeric vector, character vector, or color function (they do not have to all be the same type of specification). Each element in the list will be the color mapping to the corresponding element in the profileplyr object.

`color_by_sample_group`
A character vector that is identical to a column name in sampleData(object), and if set, the heatmaps will be colored based on that column (should be a factor, if not it will be converted to one)

`matrices_pos_line`
A logical for whether to draw a vertical line(s) at the position of the target (for both a single point or a window). Default is true.

`matrices_pos_line_gp`
Graphics parameters for the vertical position lines. Should be set with the gpar() function from the grid() package.

`top_anno_axis_font`
The fontsize of the y-axis labels for the top annotation of all heatmaps representing the matrices

`matrices_column_title_gp`
Graphics parameters for the titles on top of each range/matrix. Should be set with the gpar() function from the grid() package.

`matrices_axis_name`
Names for axis which is below the heatmap. For profileplyr object made from BamBigwig_to_chipProfile/as_profileplyr functions, the names will be of length three, with the middle point being the midpoint of each range. If the profileplyr object was made from a deeptools matrix with import_deepToolsMat(), the names will be length three if matrix was generated with 'computeMatrix reference-point', or length of four if matrix was generated with 'computeMatrix scale-regions' corresponding to upstream, start of targets, end of targets and downstream (or length of two if no upstream/downstream included).

`matrices_axis_name_gp`
Graphics parameters for the text on the x-axis of each matrix heatmap. Should be set with the gpar() function from the grid() package.

group_anno_color
: This will specify colors for the grouping column if the 'include_group_annotation' argument is set to TRUE. Since the group column of the range metadata should always be a discrete value, this should be either a numeric vector or character vector with color names. By default, numeric vectors use the colors in palette(), however this can be expanded with other R color lists(e.g. colors()). The length of this vector must equal the number of groups.

group_anno_width
: A numeric value that is used to will set the width of the column bar (in mm using the unit() function from the grid package) for the grouping annotation column.

group_anno_row_title_gp
: Graphics parameters for the labels of the groups on the side of the heatmap. Should be set with the gpar() function from the grid() package.

group_anno_column_names_gp
: Graphics parameters for the label of the grouping annotation column. Should be set with the gpar() function from the grid() package.

extra_anno_color
: This will specify colors for the annotation columns added by the 'extra_annotation_columns' argument. This must be a list that is of equal length to the 'extra_annotation_columns' argument. Each element of this list will be used to specify the color scheme for the corresponding element of the 'extra_annotation_columns' vector. If an element is NULL, the default colors will be used for the column annotation. For a column with discrete variables this will typically be a vector of numbers or a vector of color names. By default, numeric vectors use the colors in palette(), however this can be expanded with other R color lists(e.g. colors()). For columns with continuous variables, this can also be a a vector of numbers or a vector of color names to signify the color progression, or it can be color mapping function using colorRamp2() from the circlize package.

extra_anno_top_annotation
: This is a logical vector that determines whether annotation plots are shown on top of the heatmaps for the extra annotations. This must either be a length of 1, in which case all of the heatmaps will abide by this value. Otherwise this must be a vector of equal length to the 'extra_annotation_columns' argument and the elements of this vector will correspond to the equivalent elements in 'extra annotation_columns'

extra_anno_width
: This will set the width of the individual extra annotation columns on the right side of the figure. This must be a numeric vector with each element setting the width for the corresponding element in the 'extra_annotation_columns' argument.

only_extra_annotation_columns
: If set to TRUE, only the heatmaps representing the extra annotation columns will be shown, and the range based heatmaps from the assay matrices will be excluded.

gap
: The size of the gap between heatmaps and annotations. Only relevant if return_ht_list = FALSE

genes_to_label
: A character vector of gene symbols that should match character strings in the 'SYMBOL' column that results from either 'annotateRanges' or 'annotateRanges_great'. Genes that are both in this vector and in the 'SYMBOL' column will be labeled on the heatmap.

gene_label_font_size
: The size of the text for the labels for genes specified in 'genes_to_label' argument.

show_heatmap_legend
: A logical vector with each position corresponding to each matrix heatmap (not including the 'extra_annotation_columns') that determines whether a legend is produced for that heatmap. By default a single legend is made if all heatmaps use the same color scale, or separate legends are made for each matrix heatmap if the scales are different.

legend_params
: A list that contains parameters for the legend. See color_mapping_legend-ColorMapping-method for all available parameters.

use_raster
: Whether render the heatmap body as a raster image. It helps to reduce file size when the matrix is huge.

raster_device
: Graphic device which is used to generate the raster image. Options are "png", "jpeg", "tiff", "CairoPNG", "CairoJPEG", "CairoTIFF"

raster_quality
: A value set to larger than 1 will improve the quality of the raster image.

raster_device_param
: A list of further parameters for the selected graphic device. For raster image support, please check https://jokergoo.github.io/ComplexHeatmap-reference/book/a-single-heatmap.html#heatmap-as-raster-image .

Details

Takes a profileplyr object and generates a heatmap that can be annotated by group or by range metadata columns of the profileplyr object

Value

By default a customized version of a heatmap from EnrichedHeatmap, if return_ht_list = TRUE then a heatmap list is returned that can be modified and then entered as an input for the EnrichedHeatmap function

Examples

```
example <- system.file("extdata", "example_deepTools_MAT", package = "profileplyr")
object <- import_deepToolsMat(example)

generateEnrichedHeatmap(object, include_group_annotation = FALSE)
```

---

generateProfilePlot *Import ChIPprofile object to profileplyr*

Description

Function to convert soGGi ChIPprofile objects to profileplyr object .

Usage

```
generateProfilePlot(
  object,
  sampleNames = rownames(sampleData(object)),
  colorGroup = params(object)$rowGroupsInUse,
  colorlist = NULL,
  facet_nrow = 1,
  facet_ncol = NULL,
  facet_scales = "fixed"
)
```

Arguments

object	A profileplyr object
sampleNames	The names used to label the samples in the profileplyr object. By default, the names stored in rownames(sampleData(object)) are used.
colorGroup	The name of the column in mcols(object) that will be used for color grouping in the plot. By default the column name in params(object)$rowGroupsInUse is used. If this column is not a factor variable, then it will be converted into one.
colorlist	A vector containing the colors to be used. The positions in the vector will be matched with the levels of the factor variable chosen in the 'colorGroup' argument.
facet_nrow	The number of rows when making the plot panels. This argument is passed to 'nrow' of the ggplot2 function facet_wrap.
facet_ncol	The number of columns when making the plot panels. This argument is passed to 'ncol' of the ggplot2 function facet_wrap.
facet_scales	Whether the scales of all plot panels should be fixed ("fixed", default), free ("free"), or free in one dimension ("free_x" or "free_y"). This argument is passed to 'scales' of the ggplot2 function facet_wrap.

Value

A profileplyr object

Examples

```
example <- system.file("extdata", "example_deepTools_MAT", package = "profileplyr")
object <- import_deepToolsMat(example)

generateProfilePlot(object)
```

---

gene_list_character *Character vector of the top differentially expressed genes from hindbrain versus liver as measured by RNA-seq*

Description

This dataset contains a character vector of the top differenetially expressed genes in the hindbrain versus liver as measured by RNA-seq (both genes that go up and those that go down). Data was downloaded from ENCODE.

Usage

```
data(gene_list_character)
```

Details

- gene_list_character

Value

A character vector of the top differentially expressed genes in the hindbrain versus liver as measured by RNA-seq/

---

gene_list_dataframe *Dataframe of top differentially expressed genes from hindbrain versus liver as measured by RNA-seq*

Description

This dataset contains a dataframe of the top differentially expressed genes in the hindbrain versus liver as measured by RNA-seq (both genes that go up and those that go down). The gene names are the rownames, and the first column is the 'stat' column from DESeq2. Data was downloaded from ENCODE.

Usage

```
data(gene_list_dataframe)
```

Details

- gene_list_dataframe

Value

A dataframe of top differentially expressed genes from hindbrain versus liver as measured by RNA-seq/

---

groupBy *group the rows and ranges of the profileplyr object*

Description

group the rows and ranges of the profileplyr object

Usage

```
groupBy(
  object = "profileplyr",
  group = "ANY",
  GRanges_names = "character",
  levels = "ANY",
  include_nonoverlapping = "logical",
  separateDuplicated = "logical",
  inherit_groups = "logical"
)

## S4 method for signature 'profileplyr'
groupBy(
  object = "profileplyr",
  group = "ANY",
  GRanges_names = NULL,
  levels = NULL,
  include_nonoverlapping = FALSE,
  separateDuplicated = TRUE,
  inherit_groups = FALSE
)
```

Arguments

object	A profileplyr object
group	How the ranges will be grouped. If this is a character string, then it must match a column name of the range metadata, and this column will be used for grouping of any exported deepTools matrix. If this is a GRanges, or GRangesList, then the ranges will be subset based on overlap with these GRanges. If this is a list, each element should contain ether 1) a character vector of genes, and ranges will be subset based on overlap with these genes, as determined by the annotations made by annotateRanges() or annotateRanges_great() functions, or 2) a data frame with the gene symbols as the rownames. Any additional columns of this dataframe will be added to the range metadata.
GRanges_names	The names of the GRanges that were used for the "GRanges" argument. This will be used to label these groups in the construction of the resulting profileplyr object.
levels	This will set the levels of the grouping column set by 'rowGroupsInUse' (if the grouping column is not a factor, it will be converted to one). If levels are not provided, they will remain unchanged if the grouping column was already a factor, or will use default leveling (e.g. alphabetical) if grouping column is not already a factor variable.
include_nonoverlapping	A logical argument, if FALSE (default) the regions from the original deepTools matrix that do not overlap with the user defined regions will be left out of the returned profileplyr object.
separateDuplicated	A logical argument, if TRUE (default) then regions that overlap multiple inputs to 'GRanges' argument will be separated and made into their own group. All possible combinations of region overlaps will be tested, so it is not recommended to have more than 3 groups if this option is TRUE. If FALSE, then

regions that overlap each individual 'GRanges' input will be in the output, and if one region overlaps multiple 'GRanges' inputs, then it will be duplicated in the output and will show up in the section for each group.

`inherit_groups` A logical whether that groups the exist in the profileplyr object in the 'object' argument should be included in the default grouping scheme for the output object of this function. The default is TRUE. If false, only the GRanges or gene list overlap annotation will be used for heatmap grouping.

Details

Takes a SE object and groups rows

Value

A profileplyr object

Methods (by class)

- `groupBy(profileplyr)`: group the rows and ranges of the profileplyr object

Examples

```
# group by gene list or list of data frames with genes as rownames
## not shown here but see vignette for grouping by gene lists

# group by GRanges

example <- system.file("extdata", "example_deepTools_MAT", package = "profileplyr")
object <- import_deepToolsMat(example)
data("K27ac_GRlist_hind_liver_top5000") # load pre-made GRanges
K27ac_groupByGR <- groupBy(object, group = K27ac_GRlist_hind_liver_top5000)

# switch rowGroupsInUse

switchGroup <- groupBy(K27ac_groupByGR, group = "GR_overlap_names")
params(switchGroup)$rowGroupsInUse
```

`inherit_group_function` *Redundant code for inheriting grouping wrapped into subsetbyRangeOverlap() or subsetbyGeneListOverlap() functions*

Description

Redundant code for inheriting grouping wrapped into subsetbyRangeOverlap() or subsetbyGeneListOverlap() functions

Usage

```
inherit_group_function(object, rowGroupsInUse_input, type, separateDuplicated)
```

Arguments

object	A profileplyr object
rowGroupsInUse_input	the inherited rowGroupsInUse
type	Either "GR" for subsetbyRangeOverlap() function or "GL" for subsetbyGeneListOverlap() function
separateDuplicated	A logical argument, if TRUE then regions that overlap multiple inputs to 'GRanges' argument will be separated and made into their own group. All possible combinations of region overlaps will be tested, so it is not recommended to have more than 3 groups if this option is TRUE. If FALSE, then regions that overlap each individual 'GRanges' input will be in the output, and if one region overlaps multiple 'GRanges' inputs, then it will be duplicated in the output and will show up in the section for each group.

Details

tbd

Value

A profileplyr object

K27ac_GRlist_hind_liver_top5000

GRangesList of the top 5000 H3K27ac peaks from hindbrain and liver downloaded from ENCODE

Description

This dataset contains a GRangesList of the H3K27ac peaks in either the hindbrain or the liver with the highest signal. Data was downloaded from ENCODE.

Usage

```
data(K27ac_GRlist_hind_liver_top5000)
```

Details

- K27ac_GRlist_hind_liver_top5000

Value

A GRangesList of the top 5000 H3K27ac peaks from hindbrain and liver downloaded from ENCODE/

orderBy *choose the column by which to order the ranges by within each group*

Description

choose the column by which to order the ranges by within each group

Usage

```
orderBy(object = "profileplyr", column = "ANY")

## S4 method for signature 'profileplyr'
orderBy(object = "profileplyr", column = "ANY")
```

Arguments

object	A profileplyr object
column	Which column of mcols(proplyrObject) should be used for ordering the ranges. If NULL removes any previous setting for row ordering.

Details

Takes a profileplyr object and orders the rows based on a user defined metadata column of rowRanges

Value

A profileplyr object

Methods (by class)

- orderBy(profileplyr): choose the column by which to order the ranges by within each group

Examples

```
example <- system.file("extdata", "example_deepTools_MAT", package = "profileplyr")
object <- import_deepToolsMat(example)

library(SummarizedExperiment)
cluster <- clusterRanges(object, fun = rowMeans, cutree_rows = 3)
cluster_order <- orderBy(cluster, column = "hierarchical_order")
params(cluster_order)$mcolToOrderBy
```

params *Retrieve and set parameters in profileplyr object*

Description

Retrieve and set parameters in profileplyr object

Usage

```
params(object)
```

Arguments

object	A profileplyr object

Value

A list containing parameters for profileplyr object.

Examples

```
example <- system.file("extdata", "example_deepTools_MAT", package = "profileplyr")
object <- import_deepToolsMat(example)
params(object)
```

profileplyr-class *Join, subset and manipulate ChIPprofile objects*

Description

Join, subset and manipulate ChIPprofile objects

Usage

```
## S4 method for signature 'profileplyr'
c(x, ...)

## S4 method for signature 'profileplyr,ANY,ANY,ANY'
x[i, j, k, ..., drop = FALSE]
```

Arguments

x	profileplyr object
...	Additional arguments.
i	An integer or character scalar indicating ranges of profileplyr object to return
j	An integer or character scalar indicating columns of profileplyr object to return or a An integer or character scalar indicating which profileplyr object samples to return
k	An integer or character scalar indicating samples of profileplyr object to return.
drop	A logical whether to drop empty samples

Value

A profileplyr object

`sampleData` *Retrieve and set sample data in profileplyr object*

Description

Retrieve and set sample data in profileplyr object

Usage

```
sampleData(object = "profileplyr")

## S4 method for signature 'profileplyr'
sampleData(object = "profileplyr")

sampleData(object) <- value

## S4 replacement method for signature 'profileplyr,DataFrame'
sampleData(object) <- value
```

Arguments

`object`	A profileplyr object
`value`	DataFrame of sample information

Value

A DataFrame containing sample data

A DataFrame containing sample data to replace current sample data

Examples

```
example <- system.file("extdata", "example_deepTools_MAT", package = "profileplyr")
object <- import_deepToolsMat(example)
sampleData(object)
sampleData(object)$scale <- c(1,10,1)
```

`subsetbyGeneListOverlap` *Subset ranges based on overlap with lists of Gene sets*

Description

The ranges from the deepTools matrix will be subset based on whether they overlap with user defined gene sets

Usage

```
subsetbyGeneListOverlap(
  object,
  group,
  include_nonoverlapping = FALSE,
  separateDuplicated = TRUE,
  inherit_groups = FALSE
)
```

Arguments

`object`	A profileplyr object
`group`	How the ranges will be grouped. If this is a character string, then it must match a column name of the range metadata, and this column will be used for grouping of any exported deepTools matrix. If this is a GRanges, or GRangesList, then the ranges will be subset based on overlap with these GRanges. If this is a list, each element should contain ether 1) a character vector of genes, and ranges will be subset based on overlap with these genes, as determined by the annotations made by annotateRanges() or annotateRanges_great() functions, or 2) a data frame with the gene symbols as the rownames. Any additional columns of this dataframe will be added to the range metadata.
`include_nonoverlapping`	A logical argument, if FALSE the regions from the original deepTools matrix that do not overlap with the user defined regions will be left out of the returned profileplyr object.
`separateDuplicated`	A logical argument, if TRUE (default) then regions that overlap multiple inputs to 'GRanges' argument will be separated and made into their own group. All possible combinations of region overlaps will be tested, so it is not recommended to have more than 3 groups if this option is TRUE. If FALSE, then regions that overlap each individual 'GRanges' input will be in the output, and if one region overlaps multiple 'GRanges' inputs, then it will be duplicated in the output and will show up in the section for each group.
`inherit_groups`	A logical whether that groups the exist in the profileplyr object in the 'object' argument should be included in the default grouping scheme for the output object of this function. The default is TRUE. If false, only the gene list overlap annotation will be used for heatmap grouping.

Details

tbd

Value

A profileplyr object

Examples

```
# see the groupby function within profileplyr for examples
```

`subsetbyRangeOverlap` *Subset ranges based on overlap with a GRanges object*

Description

The ranges from the deepTools matrix will be subset based on whether they overlap with user defined ranges

Usage

```
subsetbyRangeOverlap(
  object,
  group,
  GRanges_names = NULL,
  include_nonoverlapping = FALSE,
  separateDuplicated = TRUE,
  inherit_groups = FALSE
)
```

Arguments

`object`	A profileplyr object
`group`	How the ranges will be grouped. If this is a character string, then it must match a column name of the range metadata, and this column will be used for grouping of any exported deepTools matrix. If this is a GRanges, or GRangesList, then the ranges will be subset based on overlap with these GRanges. If this is a list, each element should contain ether 1) a character vector of genes, and ranges will be subset based on overlap with these genes, as determined by the annotations made by annotateRanges() or annotateRanges_great() functions, or 2) a data frame with the gene symbols as the rownames. Any additional columns of this dataframe will be added to the range metadata.
`GRanges_names`	The names of the GRanges that were used for the "GRanges" argument. This will be used to label these groups in the construction of the resulting profileplyr object.
`include_nonoverlapping`	A logical argument, if FALSE the regions from the original deepTools matrix that do not overlap with the user defined regions will be left out of the returned profileplyr object.
`separateDuplicated`	A logical argument, if TRUE then regions that overlap multiple inputs to 'GRanges' argument will be separated and made into their own group. All possible combinations of region overlaps will be tested, so it is not recommended to have

more than 3 groups if this option is TRUE. If FALSE, then regions that overlap each individual 'GRanges' input will be in the output, and if one region overlaps multiple 'GRanges' inputs, then it will be duplicated in the output and will show up in the section for each group.

`inherit_groups` A logical whether that groups the exist in the profileplyr object in the 'object' argument should be included in the default grouping scheme for the output object of this function. The default is TRUE. If false, only the GRanges overlap annotation will be used for heatmap grouping.

Details

tbd

Value

A profileplyr object

Examples

```
# see the groupby function within profileplyr for examples
```

`subset_GR_GL_common_top` *Redundant code wrapped into subsetbyRangeOverlap() or subsetbyGeneListOverlap() functions*

Description

Redundant code wrapped into subsetbyRangeOverlap() or subsetbyGeneListOverlap() functions

Usage

```
subset_GR_GL_common_top(object, overlap, input_names, type, separateDuplicated)
```

Arguments

`object` A profileplyr object

`overlap` hits object from subsetByOverlap function

`input_names` names of either the gene list of the granges that go into function

`type` Either "GR" for subsetbyRangeOverlap() function or "GL" for subsetbyGeneListOverlap() function

`separateDuplicated`
A logical argument, if TRUE (default) then regions that overlap multiple inputs to 'GRanges' argument will be separated and made into their own group. All possible combinations of region overlaps will be tested, so it is not recommended to have more than 3 groups if this option is TRUE. If FALSE, then regions that overlap each individual 'GRanges' input will be in the output, and if one region overlaps multiple 'GRanges' inputs, then it will be duplicated in the output and will show up in the section for each group.

Details

tbd

Value

A list of profileplyr objects

`summarize` *summarize the rows of a deepTools matrix*

Description

summarize the rows of a deepTools matrix

Usage

```
summarize(
  object = "profileplyr",
  fun = "function",
  output = "character",
  keep_all_mcols = "logical",
  sampleData_columns_for_longPlot = "character"
)

## S4 method for signature 'profileplyr'
summarize(
  object = "profileplyr",
  fun = "function",
  output = "character",
  keep_all_mcols = FALSE,
  sampleData_columns_for_longPlot = NULL
)
```

Arguments

`object`	A profileplyr object
`fun`	the function used to summarize the ranges (e.g. rowMeans or rowMax)
`output`	Must be either "matrix", "long", or "object".
`keep_all_mcols`	if output is 'long' and this is set to TRUE, then all metadata columns in the rowRanges will be included in the output. If FALSE (default value), then only the column indicated in the 'rowGroupsInUse' slot of the metadata will be included in the output dataframe.
`sampleData_columns_for_longPlot`	If output is set to 'long', then this argument can be used to add information stored in sampleData(object) to the summarized data frame. This needs to be a character vector with elements matching coumn names in sampleData(object).

Details

Takes a SE object and outputs a summarized experiment object with a matrix containing ranges as rows and each sample having one column with summary statistic

Value

If output="matrix" returns a matrix, if output="long" returns a data.frame in long format, if output="object" returns a SummarizedExperiment object with the summarized matrix.

Methods (by class)

- `summarize(profileplyr)`: summarize the rows of a deepTools matrix

Examples

```
example <- system.file("extdata", "example_deepTools_MAT", package = "profileplyr")
object <- import_deepToolsMat(example)

# output matrix (can be used to make a heatmap)

object_sumMat <- summarize(object, fun = rowMeans, output = "matrix")

# output long dataframe for ggplot

object_long <- summarize(object, fun = rowMeans, output = "long")
object_long[1:3, ]

library(ggplot2)
ggplot(object_long, aes(x = Sample, y = log(Signal))) + geom_boxplot()

# output profileplyr object containing summarized matrix

summarize(object, fun = rowMeans, output = "object")
```

Index