Package 'easybio'

February 12, 2025

Title Comprehensive Single-Cell Annotation and Transcriptomic Analysis Toolkit

Version 1.1.1

Description Provides a comprehensive toolkit for single-cell annotation with the 'CellMarker2.0' database (see Xia Li, Peng Wang, Yunpeng Zhang (2023) <doi:10.1093/nar/gkac947>). Streamlines biological label assignment in single-cell RNA-seq data and facilitates transcriptomic analysis, including preparation of TCGA<https://portal.gdc.cancer.gov/> and GEO<https://www.ncbi.nlm.nih.gov/geo/> datasets, differential expression analysis and visualization of enrichment analysis results. Additional utility functions support various bioinformatics workflows. See Wei Cui (2024) <doi:10.1101/2024.09.14.609619> for more details.

URL https://github.com/person-c/easybio

BugReports https://github.com/person-c/easybio/issues

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Imports data.table, ggplot2, httr2, R6, xml2

Depends R (>= 4.1.0)

LazyData true

Suggests knitr, patchwork, rmarkdown, ggrepel, Seurat, limma, GEOquery, fgsea, edgeR, testthat (>= 3.0.0)

biocViews limma, GEOquery, edgeR, fgsea

Language en-US

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Wei Cui [aut, cre, cph] (<https://orcid.org/0009-0004-8315-5899>)

Maintainer Wei Cui <m2c.w@outlook.com>

Repository CRAN

Date/Publication 2025-02-12 04:50:02 UTC

Contents

`Artist` *Visualization Artist for Custom Plots*

Description

The Artist class offers a suite of methods designed to create a variety of plots using ggplot2 for data exploration. Any methods prefixed with plot_ or test_ will log the command history along with their results, allowing you to review all outcomes later via the get_all_results() method. Notably, methods starting with plot_ will check if the result of the preceding command is of the htest class. If so, it will incorporate the previous command and its p-value as the title and subtitle, respectively. This class encompasses methods for crafting dumbbell plots, bubble plots, divergence bar charts, lollipop plots, contour plots, scatter plots with ellipses, donut plots, and pie charts. Each method is tailored to map data to specific visual aesthetics and to apply additional customizations as needed.

Value

The R6 class Artist.

Public fields

data Stores the dataset used for plotting.

command recode the command.

result record the plot.

Methods

Public methods:

- Artist$new()
- Artist$get_all_result()
- Artist$test_wilcox()
- Artist$test_t()
- Artist$plot_scatter()
- Artist$plot_box()
- Artist$dumbbbell()
- Artist$bubble()
- Artist$barchart_divergence()
- Artist$lollipop()
- Artist$contour()
- Artist$scatter_ellipses()
- Artist$donut()
- Artist$pie()
- Artist$clone()

Method new(): Initializes the Artist class with an optional dataset.

Usage:

```
Artist$new(data = NULL)
```

Arguments:

data A data frame containing the dataset to be used for plotting. Default is NULL.

Returns: An instance of the Artist class.

Method `get_all_result()`**:** Get all history result

Usage:

```
Artist$get_all_result()
```

Returns: a data.table object

Method `test_wilcox()`**:** Conduct wilcox.test

Usage:

```
Artist$test_wilcox(formula, data = self$data, ...)
```

Arguments:

`formula` `wilcox.test()` formula arguments

`data` A data frame containing the data to be plotted. Default is `self$data`.

`...` Additional aesthetic mappings passed to `wilcox.test()`.

Returns: A ggplot2 scatter plot.

Method `test_t()`**:** Conduct wilcox.test

Usage:

```
Artist$test_t(formula, data = self$data, ...)
```

Arguments:

`formula` `t.test()` formula arguments

`data` A data frame containing the data to be plotted. Default is `self$data`.

`...` Additional aesthetic mappings passed to `t.test()`.

Returns: A ggplot2 scatter plot.

Method `plot_scatter()`**:** Creates a scatter plot.

Usage:

```
Artist$plot_scatter(
  data = self$data,
  fun = function(x) x,
  x,
  y,
  ...,
  add = private$is_htest()
)
```

Arguments:

`data` A data frame containing the data to be plotted. Default is `self$data`.

`fun` function to process the `self$data`.

`x` The column name for the x-axis.

`y` The column name for the y-axis.

`...` Additional aesthetic mappings passed to `aes()`.

`add` whether to add the test result.

Returns: A ggplot2 scatter plot.

Method `plot_box()`**:** Creates a box plot.

Usage:

```
Artist$plot_box(
  data = self$data,
  fun = function(x) x,
  x,
  ...,
  add = private$is_htest()
)
```

Arguments:

`data` A data frame or tibble containing the data to be plotted. Default is `self$data`.

`fun` function to process the `self$data`.

`x` The column name for the x-axis.

`...` Additional aesthetic mappings passed to `aes()`.

`add` whether to add the test result.

Returns: A ggplot2 box plot.

Method `dumbbbell()`**:** Create a dumbbell plot

This method generates a dumbbell plot using the provided data, mapping the specified columns to the x-axis, y-axis, and color aesthetic.

Usage:

```
Artist$dumbbbell(data = self$data, x, y, col, ...)
```

Arguments:

`data` A data frame containing the data to be plotted.

`x` The column in `data` to map to the x-axis.

`y` The column in `data` to map to the y-axis.

`col` The column in `data` to map to the color aesthetic.

`...` Additional aesthetic mappings or other arguments passed to `ggplot`.

Returns: A ggplot object representing the dumbbell plot.

Method `bubble()`**:** Create a bubble plot

This method generates a bubble plot where points are mapped to the x and y axes, with their size and color representing additional variables.

Usage:

```
Artist$bubble(data = self$data, x, y, size, col, ...)
```

Arguments:

`data` A data frame containing the data to be plotted.

`x` The column in `data` to map to the x-axis.

`y` The column in `data` to map to the y-axis.

`size` The column in `data` to map to the size of the points.

`col` The column in `data` to map to the color of the points.

`...` Additional aesthetic mappings or other arguments passed to `ggplot`.

Returns: A ggplot object representing the bubble plot.

Method `barchart_divergence()`**:** Create a divergence bar chart

This method generates a divergence bar chart where bars are colored based on their positive or negative value.

Usage:

```
Artist$barchart_divergence(data = self$data, group, y, fill, ...)
```

Arguments:

`data` A data frame containing the data to be plotted.

`group` The column in `data` representing the grouping variable.

`y` The column in `data` to map to the y-axis.

`fill` The column in `data` to map to the fill color of the bars.

`...` Additional aesthetic mappings or other arguments passed to `ggplot`.

Returns: A ggplot object representing the divergence bar chart.

Method `lollipop()`**:** Create a lollipop plot

This method generates a lollipop plot, where points are connected to a baseline by vertical segments, with customizable colors and labels.

Usage:

```
Artist$lollipop(data = self$data, x, y, ...)
```

Arguments:

`data` A data frame containing the data to be plotted.

`x` The column in `data` to map to the x-axis.

`y` The column in `data` to map to the y-axis.

`...` Additional aesthetic mappings or other arguments passed to `ggplot`.

Returns: A ggplot object representing the lollipop plot.

Method `contour()`**:** Create a contour plot

This method generates a contour plot that includes filled and outlined density contours, with data points overlaid.

Usage:

```
Artist$contour(data = self$data, x, y, ...)
```

Arguments:

`data` A data frame containing the data to be plotted.

`x` The column in `data` to map to the x-axis.

`y` The column in `data` to map to the y-axis.

`...` Additional aesthetic mappings or other arguments passed to `ggplot`.

Returns: A ggplot object representing the contour plot.

Method `scatter_ellipses()`**:** Create a scatter plot with ellipses

This method generates a scatter plot where data points are colored by group, with ellipses representing the confidence intervals for each group.

Usage:

```
Artist$scatter_ellipses(data = self$data, x, y, col, ...)
```

Arguments:

`data` A data frame containing the data to be plotted.

`x` The column in `data` to map to the x-axis.

`y` The column in `data` to map to the y-axis.

`col` The column in `data` to map to the color aesthetic.

`...` Additional aesthetic mappings or other arguments passed to `ggplot`.

Returns: A ggplot object representing the scatter plot with ellipses.

Method `donut()`**:** Create a donut plot

This method generates a donut plot, which is a variation of a pie chart with a hole in the center. The sections of the donut represent the proportion of categories in the data.

Usage:

```
Artist$donut(data = self$data, x, y, fill, ...)
```

Arguments:

`data` A data frame containing the data to be plotted.

`x` The column in `data` to map to the x-axis.

`y` The column in `data` to map to the y-axis.

`fill` The column in `data` to map to the fill color of the sections.

`...` Additional aesthetic mappings or other arguments passed to `ggplot`.

Returns: A ggplot object representing the donut plot.

Method `pie()`**:** Create a pie chart

This method generates a pie chart where sections represent the proportion of categories in the data.

Usage:

```
Artist$pie(data = self$data, y, fill, ...)
```

Arguments:

`data` A data frame containing the data to be plotted.

`y` The column in `data` to map to the y-axis.

`fill` The column in `data` to map to the fill color of the sections.

`...` Additional aesthetic mappings or other arguments passed to `ggplot`.

Returns: A ggplot object representing the pie chart.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
Artist$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
library(data.table)
air <- subset(airquality, Month %in% c(5, 6))
setDT(air)
cying <- Artist$new(data = air)
cying$plot_scatter(x = Wind, y = Temp)
cying$test_wilcox(
  formula = Ozone ~ Month,
)
cying$plot_scatter(x = Wind, y = Temp)
cying$plot_scatter(f = \(x) x[, z := Wind * Temp], x = Wind, y = z)
```

`available_tissue_class` *Retrieve Available Tissue Classes for a Given Species*

Description

This function extracts and returns a unique list of available tissue classes from the CellMarker2.0 database for a specified species.

Usage

```
available_tissue_class(spc)
```

Arguments

spc	A character string specifying the species (e.g., "Human" or "Mouse").

Value

A character vector of unique tissue classes available for the given species. If no tissue classes are found, an empty vector is returned.

Examples

```
available_tissue_class("Human")
available_tissue_class("Mouse")
```

available_tissue_type *Retrieve Available Tissue Types for a Given Species*

Description

This function extracts and returns a unique list of available tissue types from the CellMarker2.0 database for a specified species.

Usage

```
available_tissue_type(spc)
```

Arguments

spc	A character string specifying the species (e.g., "Human" or "Mouse").

Value

A character vector of unique tissue types available for the given species. If no tissue types are found, an empty vector is returned.

Examples

```
available_tissue_type("Human")
available_tissue_type("Mouse")
```

check_marker *Verify Markers for Specific Clusters Using matchCellMarker*

Description

This function checks the markers for specified clusters returned by the `matchCellMarker2` function. It allows users to filter by species, cluster, and to specify whether to consider cis or trans interactions.

Usage

```
check_marker(
  marker,
  n,
  spc,
  tissueClass = available_tissue_class(spc),
  tissueType = available_tissue_type(spc),
  cl = c(),
  topcellN = 2,
  cis = FALSE
)
```

Arguments

marker	A data frame of markers obtained from Seurat::FindAllMarkers.
n	An integer specifying the top number of genes to match from the input markers.
spc	A character string specifying the species, which can be either 'Human' or 'Mouse'.
tissueClass	A character specifying the tissue classes, default available_tissue_class(spc).
tissueType	A character specifying the tissue types, default available_tissue_type(spc).
cl	An integer or vector of integers specifying the clusters to check.
topcellN	An integer specifying the number of top cells to check for each cluster.
cis	A logical value indicating whether to check marker directly from the top symbol of matchCellMarker2 or re-search marker for top cell in cellMarker2.

Value

A named list where each name corresponds to a cell type and each element is a vector of marker names.

Examples

```
# Example usage:
# Check the top 50 markers for clusters 1, 4, and 7 in the Human species.
## Not run:
library(easybio)
data(pbmc.markers)
check_marker(pbmc.markers, n = 50, spc = "Human", cl = c(1, 4, 7))

## End(Not run)
```

CHOL_DEGs *Example DEGs data from Limma-Voom workflow for TCGA-CHOL project*

Description

The data were obtained by the limma-voom workflow

dgeList *Construct a DGEList Object*

Description

This function creates a DGEList object from a count matrix, sample information, and feature information. It is designed to facilitate the analysis of differential gene expression using the edgeR package.

Usage

```
dgeList(count, sample.info, feature.info)
```

Arguments

count	A numeric matrix where rows represent features (e.g., genes) and columns represent samples. Row names should correspond to feature identifiers, and column names should correspond to sample identifiers.
sample.info	A data frame containing information about the samples. The number of rows should match the number of columns in the count matrix.
feature.info	A data frame containing information about the features. The number of rows should match the number of rows in the count matrix.

Value

A DGEList object as defined by the edgeR package, which includes the count data, sample information, and feature information.

dprocess_dgeList *Filter Low-Expressed Genes and Normalize DGEList Data*

Description

This function filters out low-expressed genes from a DGEList object and normalizes the count data. It also provides diagnostic plots for raw and filtered data.

Usage

```
dprocess_dgeList(x, group.column, min.count = 10)
```

Arguments

x	A DGEList object containing raw count data.
group.column	The name of the column in x$samples that contains the grouping information for the samples.
min.count	The minimum number of counts required for a gene to be considered expressed. Genes with counts below this threshold in any group will be filtered out. Defaults to 10.

Value

The function returns a DGEList object with low-expressed genes filtered out and normalization factors calculated.

---

finsert *Insert Specific Values into a Character Vector at Defined Positions*

Description

This function constructs a character vector of a specified length, inserting given values at positions determined by numeric indices. It is designed for single cell annotation tasks, where specific annotations need to be placed at certain positions in a vector.

Usage

```
finsert(
  x = expression(c(0, 1, 3) == "Neutrophil", c(2, 4, 8) == "Macrophage"),
  len = integer(),
  setname = TRUE,
  na = "Unknown"
)
```

Arguments

x	An expression defining the value to insert and the positions at which to insert them. The expression should be a list of logical comparisons, where the left side is a numeric vector of positions and the right side is the corresponding character value to insert.
len	The desired length of the output character vector. If the specified positions exceed this length, the vector will be padded with the na value.
setname	A logical value indicating whether to set names for the elements of the vector. If TRUE, names are set as character representations of the positions from 0 to the length of the vector minus one.
na	The default value to use for positions not specified in x. This value is also used to pad the vector if its length exceeds the positions specified in x.

Value

A named character vector with the specified values inserted at given positions and padded with the na value if necessary.

Examples

```
# Example usage:
# Insert "Neutrophil" at positions 0, 1, 3 and "Macrophage" at positions 2, 4, 8
# in a vector of length 10, with "Unknown" as the default value.
library(easybio)
annotated_vector <- finsert(
  x = expression(
    c(0, 1, 3) == "Neutrophil",
    c(2, 4, 8) == "Macrophage"
  ),
  len = 10,
  na = "Unknown"
)
print(annotated_vector)
```

---

get_attr *Retrieve Attributes from an R Object*

Description

This function extracts a specified attribute from an R object.

Usage

```
get_attr(x, attr_name)
```

Arguments

x	An R object that has attributes.
attr_name	The name of the attribute to retrieve.

Value

The value of the attribute with the given name.

get_marker *Retrieve Markers for Specific Cells from cellMarker2*

Description

This function extracts a list of markers for one or more cell types from the cellMarker2 dataset. It allows filtering by species, cell type, the number of markers to retrieve, and a minimum count threshold for marker occurrences.

Usage

```
get_marker(
  spc,
  cell = character(),
  tissueClass = available_tissue_class(spc),
  tissueType = available_tissue_type(spc),
  number = 5,
  min.count = 1
)
```

Arguments

spc	A character string specifying the species, which can be either 'Human' or 'Mouse'.
cell	A character vector of cell types for which to retrieve markers.
tissueClass	A character specifying the tissue classes, default available_tissue_class(spc).
tissueType	A character specifying the tissue types, default available_tissue_type(spc).
number	An integer specifying the number of top markers to return for each cell type.
min.count	An integer representing the minimum number of times a marker must have been reported to be included in the results.

Value

A named list where each name corresponds to a cell type and each element is a vector of marker names.

Examples

```
# Example usage:
# Retrieve the top 5 markers for 'Macrophage' and 'Monocyte' cell types in humans,
# with a minimum count of 1.
library(easybio)
markers <- get_marker(spc = "Human", cell = c("Macrophage", "Monocyte"))
print(markers)
```

groupStat *Perform Summary Analysis by Group Using Regular Expressions*

Description

This function applies a specified function to each group defined by a regular expression pattern applied to the names of a data object. It is useful for summarizing data when groups are defined by a pattern in the names rather than a specific column or index.

Usage

```
groupStat(f, x, xname = colnames(x), patterns)
```

Arguments

f	A function that takes a single argument and returns a summary of the data.
x	A data frame or matrix containing the data to be summarized.
xname	A character vector containing the names of the variables in x.
patterns	A list of regular expressions that define the groups.

Value

A list containing the summary statistics for each group.

Examples

```
library(easybio)
groupStat(f = \(x) x + 1, x = mtcars, patterns = list("mp", "t"))
```

groupStatI *Perform Summary Analysis by Group Using an column Index*

Description

This function applies a specified function to each group defined by an column index, and returns a summary of the results. It is useful for summarizing data by group when the groups are defined by an column index.

Usage

```
groupStatI(f, x, idx)
```

Arguments

f	A function that takes a single argument and returns a summary of the data.
x	A data frame or matrix containing the data to be summarized.
idx	A list of indices or group names that define the column groups.

Value

A list containing the summary statistics for each group.

Examples

```
library(easybio)
groupStatI(f = \(x) x + 1, x = mtcars, idx = list(c(1, 10), 2))
```

---

limmaFit *Fit a Linear Model for RNA-seq data using limma*

Description

This function fits a linear model to processed DGEList data using the limma package. It defines contrasts between groups and performs differential expression analysis.

Usage

```
limmaFit(x, group.column)
```

Arguments

x	A processed DGEList object containing normalized count data.
group.column	The name of the column in x$samples that contains the grouping information for the samples.

Value

An eBayes object containing the fitted linear model and results of the differential expression analysis.

list2dt *Convert a List with Vector Values to a Long Data.table*

Description

This function converts a named list with vector values in each element to a long data.table. The list is first flattened into a single vector, and then the data.table is created with two columns: one for the name of the original list element and another for the value.

Usage

```
list2dt(x)
```

Arguments

x	A named list where each element contains a vector of values.

Value

A long data.table with two columns: 'name' and 'value'.

Examples

```
library(easybio)
list2dt(list(a = c(1, 1), b = c(2, 2)))
```

list2graph *Convert a Named List into a Graph Based on Overlap*

Description

This function creates a graph from a named list, where the edges are determined by the overlap between the elements of the list. Each node in the graph represents an element of the list, and the weight of the edge between two nodes is the number of overlapping elements between the two corresponding lists.

Usage

```
list2graph(nodes)
```

Arguments

nodes	A named list where each element is a vector.

Value

A data.table representing the graph, with columns for the node names (node_1 and node_2) and the weight of the edge (interWeight).

matchCellMarker2 *Match Markers with cellMarker2 Dataset*

Description

This function matches markers from the FindAllMarkers output with the cellMarker2 dataset, filtering by species and selecting the top genes based on their average log2 fold change and adjusted p-values.

Usage

```
matchCellMarker2(
  marker,
  n,
  spc,
  tissueClass = available_tissue_class(spc),
  tissueType = available_tissue_type(spc)
)
```

Arguments

marker	A data frame of markers obtained from the FindAllMarkers function, expected to contain columns such as avg_log2FC, p_val_adj, and gene.
n	An integer specifying the top number of genes to match from the input markers.
spc	A character string specifying the species, which can be either 'Human' or 'Mouse'.
tissueClass	A character specifying the tissue classes, default available_tissue_class(spc).
tissueType	A character specifying the tissue types, default available_tissue_type(spc).

Value

A data frame containing matched markers from the cellMarker2 dataset, with additional columns indicating the number of matches and ordered symbols.

Examples

```
# Example usage:
# Match the top 50 differential genes from the pbmc.markers dataset with the Human
# species in the cellMarker2 dataset.
## Not run:
library(easybio)
data(pbmc.markers)
matchCellMarker2(pbmc.markers, n = 50, spc = "Human")[]

## End(Not run)
```

pbmc.markers *Example marker data from Seurat::FindAllMarkers()*

Description

The data were obtained by the seurat PBMC workflow. exact script for this data is available as system.file("example-single-cell.R", package="easybio")

plotEnrichment2 *Plot Enrichment for a Specific Pathway in fgsea*

Description

This function creates a plot of enrichment scores for a specified pathway. It provides a visual representation of the enrichment score (ES) along with the ranks and ticks indicating the GSEA walk length.

Usage

```
plotEnrichment2(pathways, pwayname, stats, gseaParam = 1, ticksSize = 0.2)
```

Arguments

pathways	A list of pathways.
pwayname	The name of the pathway for which to plot enrichment.
stats	A rank vector obtained from the 'fgsea' package.
gseaParam	The GSEA walk length parameter. Default is 1.
ticksSize	The size of the tick marks. Default is 0.2.

Value

A ggplot object representing the enrichment plot.

plotGSEA *Visualization of GSEA Result from* `fgsea::fgsea()`

Description

The `plotGSEA` function visualizes the results of a GSEA (Gene Set Enrichment Analysis) using data from the `fgsea` package. It generates a composite plot that includes an enrichment plot and a ranked metric plot.

Usage

```
plotGSEA(fgseaRes, pathways, pwayname, stats, save = FALSE)
```

Arguments

`fgseaRes`	A data table containing the GSEA results from the `fgsea` package.
`pathways`	A list of all pathways used in the GSEA analysis.
`pwayname`	The name of the pathway to visualize.
`stats`	A numeric vector representing the ranked statistics.
`save`	A logical value indicating whether to save the plot as a PDF file. Default is `FALSE`.

Value

ggplot2 object.

plotMarkerDistribution *Plot Distribution of a Marker Across Tissues and Cell Types*

Description

This function creates a dot plot displaying the distribution of a specified marker across different tissues and cell types, based on data from the CellMarker2.0 database.

Usage

```
plotMarkerDistribution(mkr = character())
```

Arguments

`mkr`	character, the name of the marker to be plotted.

Value

A ggplot2 object representing the distribution of the marker.

Examples

```
## Not run:
plotMarkerDistribution("CD14")

## End(Not run)
```

`plotORA` *Visualization of ORA Test Results*

Description

The `plotORA` function visualizes the results of an ORA (Over-Representation Analysis) test. It generates a plot with customizable aesthetics for x, y, point size, and fill, with an option to flip the axes.

Usage

```
plotORA(data, x, y, size, fill, flip = FALSE)
```

Arguments

`data`	A data frame containing the ORA results to be visualized.
`x`	The column in `data` to map to the x-axis.
`y`	The column in `data` to map to the y-axis.
`size`	The column in `data` to map to the size of the points.
`fill`	The column in `data` to map to the fill color of the bars or points. Use a constant value for a single category.
`flip`	A logical value indicating whether to flip the axes of the plot. Default is `FALSE`.

Value

ggplot2 object.

plotPossibleCell *Plot Possible Cell Distribution Based on matchCellMarker2() Results*

Description

This function creates a dot plot to visualize the distribution of possible cell types based on the results from the `matchCellMarker2()` function, utilizing data from the CellMarker2.0 database.

Usage

```
plotPossibleCell(marker, min.uniqueN = 2)
```

Arguments

`marker`	data.table, the result from the `matchCellMarker2()` function.
`min.uniqueN`	integer, the minimum number of unique marker genes that must be matched for a cell type to be included in the plot. Default is 2.

Value

A ggplot2 object representing the distribution of possible cell types.

plotRank *Visualization of GSEA Rank Statistics*

Description

The `plotRank` function visualizes the ranked statistics of a GSEA (Gene Set Enrichment Analysis) analysis. The function creates a plot where the x-axis represents the rank of each gene, and the y-axis shows the corresponding ranked list metric.

Usage

```
plotRank(stats)
```

Arguments

`stats`	A numeric vector containing the ranked statistics from a GSEA analysis.

Value

ggplot2 object

plotSeuratDot *Create Dot Plots for Markers from check_marker*

Description

This function generates dot plots for the markers obtained from the `check_marker` function for specified cluster groups within a Seurat object. The plots are saved to a temporary directory.

Usage

```
plotSeuratDot(srt, cls, ...)
```

Arguments

`srt`	A Seurat object containing the single-cell data.
`cls`	A list containing cluster groups to check. Each element of the list should correspond to a cluster or a group of clusters for which to generate dot plots.
`...`	Additional parameters to pass to the `check_marker` function.

Value

A list containing multiple DotPlot.

plotVolcano *Plot Volcano Plot for Differentially Expressed Genes*

Description

This function generates a volcano plot for differentially expressed genes (DEGs) using `ggplot2`. It allows for customization of the plot with different aesthetic parameters.

Usage

```
plotVolcano(data, data.text, x, y, color, label)
```

Arguments

`data`	A data frame containing the DEGs result.
`data.text`	A data frame containing labeled data for text annotation.
`x`	variable representing the aesthetic for the x-axis.
`y`	variable representing the aesthetic for the y-axis.
`color`	variable representing the column name for the color aesthetic.
`label`	variable representing the column name for the text label aesthetic.

Value

A ggplot object representing the volcano plot.

`prepare_geo` *Download and Process GEO Data*

Description

This function downloads gene expression data from the Gene Expression Omnibus (GEO) database. It retrieves either the expression matrix or the supplementary tabular data if the expression data is not available. The function also allows for the conversion of probe identifiers to gene symbols and can combine multiple probes into a single symbol.

Usage

```
prepare_geo(geo, dir = ".", combine = TRUE, method = "max")
```

Arguments

`geo`	A character string specifying the GEO Series ID (e.g., "GSE12345").
`dir`	A character string specifying the directory where files should be downloaded. Default is the current working directory (`"."`).
`combine`	A logical value indicating whether to combine multiple probes into a single gene symbol. Default is `TRUE`.
`method`	A character string specifying the method to use for combining probes into a single gene symbol. Options are `"max"` (take the maximum value) or `"mean"` (compute the average). Default is `"max"`.

Value

A list containing:

`data`	A data frame of the expression matrix.
`sample`	A data frame of the sample metadata.
`feature`	A data frame of the feature metadata, which includes gene symbols if combining probes.

prepare_tcga *Prepare TCGA Data for Analysis*

Description

This function prepares TCGA data for downstream analyses such as differential expression analysis with `limma` or survival analysis. It extracts and processes the necessary information from the TCGA data object, separating tumor and non-tumor samples.

Usage

```
prepare_tcga(data)
```

Arguments

`data`	A `SummarizedExperiment` object containing TCGA data, typically obtained from R package `TCGABiolinks`.

Value

A list.

setcolnames *Rename Column Names of a Data Frame or Matrix*

Description

This function renames the column names of a data frame or matrix to the specified names.

Usage

```
setcolnames(object, nm)
```

Arguments

`object`	A data frame or matrix whose column names will be renamed.
`nm`	A character vector containing the new names for the columns.

Value

A data frame or matrix with the new column names.

setrownames *Rename Row Names of a Data Frame or Matrix*

Description

This function renames the row names of a data frame or matrix to the specified names.

Usage

```
setrownames(object, nm)
```

Arguments

object	A data frame or matrix whose row names will be renamed.
nm	A character vector containing the new names for the rows.

Value

A data frame or matrix with the new row names.

setSavedir *Set a Directory for Saving Files*

Description

This function sets a directory path for saving files, creating the directory if it does not already exist. The directory path is created with the given arguments, which are passed directly to `file.path()`.

Usage

```
setSavedir(...)
```

Arguments

...	Arguments to be passed to `file.path()` to construct the directory path.

Value

The path to the newly created or existing directory.

split_matrix *Split a Matrix into Smaller Sub-matrices by Column or Row*

Description

This function splits a matrix into multiple smaller matrices by column or row. It is useful for processing large matrices in chunks, such as when performing analysis on a single computer with limited memory.

Usage

```
split_matrix(matrix, chunk_size, column = TRUE)
```

Arguments

matrix	A numeric or logical matrix to be split.
chunk_size	The number of columns or rows to include in each smaller matrix.
column	Divided by column(default is TRUE)

Value

A list of smaller matrices, each with chunk_size columns or rows.

Examples

```
library(easybio)
split_matrix(mtcars, chunk_size = 2)
split_matrix(mtcars, chunk_size = 5, column = FALSE)
```

theme_publication *Custom ggplot2 Theme for Academic Publications*

Description

theme_publication creates a custom ggplot2 theme designed for academic publications, ensuring clarity, readability, and a professional appearance. It is based on theme_classic() and includes additional refinements to axis lines, text, and other plot elements to meet the standards of high-quality academic figures.

Usage

```
theme_publication(base_size = 12, base_family = "sans")
```

Arguments

base_size	numeric, the base font size. Default is 12.
base_family	character, the base font family. Default is "sans".

Value

A ggplot2 theme object that can be applied to ggplot2 plots.

ggplot2 theme.

Examples

```
library(ggplot2)
p <- ggplot(mtcars, aes(mpg, wt)) +
  geom_point() +
  theme_publication()
print(p)
```

---

tuneParameters *Optimize Resolution and Gene Number Parameters for Cell Type Annotation*

Description

This function tunes the resolution parameter in Seurat::FindClusters() and the number of top differential genes (N) to obtain different cell type annotation results. The function generates UMAP plots for each parameter combination, allowing for a comparison of how different settings affect the clustering and annotation.

Usage

```
tuneParameters(srt, resolution = numeric(), N = integer(), spc)
```

Arguments

srt	Seurat object, the input data object to be analyzed.
resolution	numeric vector, a vector of resolution values to be tested in Seurat::FindClusters().
N	integer vector, a vector of values indicating the number of top differential genes to be used for matching in matchCellMarker2().
spc	character, the species parameter for the matchCellMarker2() function, specifying the organism.

Value

A list of ggplot2 objects, each representing a UMAP plot generated with a different combination of resolution and N parameters.

uniprot_id_map *Map UniProt IDs to Other Identifiers*

Description

This function maps UniProt IDs to other identifiers using UniProt's ID mapping service. It sends a request to the UniProt API to perform the mapping and retrieves the results in a tabular format.

Usage

```
uniprot_id_map(...)
```

Arguments

...	Parameters to be passed in the request body.

Value

A `data.table` containing the mapped identifiers.

Examples

```
## Not run:
uniprot_id_map(
  ids = "P21802,P12345",
  from = "UniProtKB_AC-ID",
  to = "UniRef90"
)

## End(Not run)
```

workIn *Perform Operations in a Specified Directory and Return to the Original Directory*

Description

This function allows you to perform operations in a specified directory and then return to the original directory. It is useful when you need to work with files or directories that are located in a specific location, but you want to return to the original working directory after the operation is complete.

Usage

```
workIn(dir, expr)
```

Arguments

`dir`	The directory path in which to operate. If the directory does not exist, it will be created recursively.
`expr`	An R expression to be evaluated within the specified directory.

Value

The result of evaluating the expression within the specified directory.

Index