Package 'tidybulk'

April 8, 2025

Type Package

Title Brings transcriptomics to the tidyverse

Version 1.19.1

Description This is a collection of utility functions that allow to perform exploration of and calculations to RNA sequencing data, in a modular, pipe-friendly and tidy fashion.

License GPL-3

Depends R (>= 4.4.0), ttservice (>= 0.3.6)

Imports tibble, readr, dplyr (>= 1.1.0), magrittr, tidyr, stringi, stringr, rlang, purrr, tidyselect, preprocessCore, stats, parallel, utils, lifecycle, scales, SummarizedExperiment, GenomicRanges, methods, S4Vectors, crayon, Matrix

Suggests BiocStyle, testthat, vctrs, AnnotationDbi, BiocManager, Rsubread, e1071, edgeR, limma, org.Hs.eg.db, org.Mm.eg.db, sva, GGally, knitr, qpdf, covr, Seurat, KernSmooth, Rtsne, ggplot2, widyr, clusterProfiler, msigdbr, DESeq2, broom, survival, boot, betareg, tidyHeatmap, pasilla, ggrepel, devtools, functional, survminer, tidySummarizedExperiment, markdown, uwot, matrixStats, igraph, EGSEA, IRanges, here, glmmSeq, pbapply, pbmcapply, lme4, glmmTMB, MASS, pkgconfig

VignetteBuilder knitr

RdMacros lifecycle

Biarch true

biocViews AssayDomain, Infrastructure, RNASeq, DifferentialExpression, GeneExpression, Normalization, Clustering, QualityControl, Sequencing, Transcription, Transcriptomics

Encoding UTF-8

LazyData true

RoxygenNote 7.3.2

LazyDataCompression xz

URL https://github.com/stemangiola/tidybulk

BugReports https://github.com/stemangiola/tidybulk/issues

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

git_branch devel

git_last_commit 587f005

git_last_commit_date 2025-03-17

Repository Bioconductor 3.21

Date/Publication 2025-04-07

Author Stefano Mangiola [aut, cre],
Maria Doyle [ctb]

Maintainer Stefano Mangiola <mangiolastefano@gmail.com>

Contents

`adjust_abundance` *Adjust transcript abundance for unwanted variation*

Description

adjust_abundance() takes as input A ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment)) and returns a consistent object (to the input) with an additional adjusted abundance column. This method uses scaled counts if present.

Usage

```
adjust_abundance(
  .data,
  .formula = NULL,
  .factor_unwanted = NULL,
  .factor_of_interest = NULL,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  method = "combat_seq",
  action = "add",
  ...,
  log_transform = NULL,
  transform = NULL,
  inverse_transform = NULL
)

## S4 method for signature 'spec_tbl_df'
adjust_abundance(
  .data,
  .formula = NULL,
  .factor_unwanted = NULL,
  .factor_of_interest = NULL,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  method = "combat_seq",
  action = "add",
  ...,
  log_transform = NULL,
  transform = NULL,
  inverse_transform = NULL
)

## S4 method for signature 'tbl_df'
adjust_abundance(
  .data,
  .formula = NULL,
  .factor_unwanted = NULL,
  .factor_of_interest = NULL,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  method = "combat_seq",
  action = "add",
  ...,
  log_transform = NULL,
  transform = NULL,
```

```
  inverse_transform = NULL
)

## S4 method for signature 'tidybulk'
adjust_abundance(
  .data,
  .formula = NULL,
  .factor_unwanted = NULL,
  .factor_of_interest = NULL,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  method = "combat_seq",
  action = "add",
  ...,
  log_transform = NULL,
  transform = NULL,
  inverse_transform = NULL
)

## S4 method for signature 'SummarizedExperiment'
adjust_abundance(
  .data,
  .formula = NULL,
  .factor_unwanted = NULL,
  .factor_of_interest = NULL,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  method = "combat_seq",
  action = "add",
  ...,
  log_transform = NULL,
  transform = NULL,
  inverse_transform = NULL
)

## S4 method for signature 'RangedSummarizedExperiment'
adjust_abundance(
  .data,
  .formula = NULL,
  .factor_unwanted = NULL,
  .factor_of_interest = NULL,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  method = "combat_seq",
  action = "add",
```

```
  ...,
  log_transform = NULL,
  transform = NULL,
  inverse_transform = NULL
)
```

Arguments

.data	A ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment))
.formula	DEPRECATED - A formula with no response variable, representing the desired linear model where the first covariate is the factor of interest and the second covariate is the unwanted variation (of the kind ~ factor_of_interest + batch)
.factor_unwanted	A tidy select, e.g. column names without double quotation. c(batch, country) These are the factor that we want to adjust for, including unwanted batcheffect, and unwanted biological effects.
.factor_of_interest	A tidy select, e.g. column names without double quotation. c(treatment) These are the factor that we want to preserve.
.sample	The name of the sample column
.transcript	The name of the transcript/gene column
.abundance	The name of the transcript/gene abundance column
method	A character string. Methods include combat_seq (default), combat and limma_remove_batch_effect.
action	A character string. Whether to join the new information to the input tbl (add), or just get the non-redundant tbl with the new information (get).
...	Further parameters passed to the function sva::ComBat
log_transform	DEPRECATED - A boolean, whether the value should be log-transformed (e.g., TRUE for RNA sequencing data)
transform	DEPRECATED - A function that will tranform the counts, by default it is log1p for RNA sequencing data, but for avoinding tranformation you can use identity
inverse_transform	DEPRECATED - A function that is the inverse of transform (e.g. expm1 is inverse of log1p). This is needed to tranform back the counts after analysis.

Details

‘r lifecycle::badge("maturing")‘

This function adjusts the abundance for (known) unwanted variation. At the moment just an unwanted covariate is allowed at a time using Combat (DOI: 10.1093/bioinformatics/bts034)

Underlying method: sva::ComBat(data, batch = my_batch, mod = design, prior.plots = FALSE, ...)

Value

A consistent object (to the input) with additional columns for the adjusted counts as '<COUNT COLUMN>_adjusted'

A consistent object (to the input) with additional columns for the adjusted counts as '<COUNT COLUMN>_adjusted'

A consistent object (to the input) with additional columns for the adjusted counts as '<COUNT COLUMN>_adjusted'

A consistent object (to the input) with additional columns for the adjusted counts as '<COUNT COLUMN>_adjusted'

A 'SummarizedExperiment' object

A 'SummarizedExperiment' object

Examples

```
cm = tidybulk::se_mini
cm$batch = 0
cm$batch[colnames(cm) %in% c("SRR1740035", "SRR1740043")] = 1

cm |>
identify_abundant() |>
adjust_abundance( .factor_unwanted = batch, .factor_of_interest =  condition, method="combat" )
```

`aggregate_duplicates` *Aggregates multiple counts from the same samples (e.g., from isoforms), concatenates other character columns, and averages other numeric columns*

Description

aggregate_duplicates() takes as input A 'tbl' (with at least three columns for sample, feature and transcript abundance) or 'SummarizedExperiment' (more convenient if abstracted to tibble with library(tidySummarizedExperiment)) and returns a consistent object (to the input) with aggregated transcripts that were duplicated.

Usage

```
aggregate_duplicates(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  aggregation_function = sum,
```

```
  keep_integer = TRUE
)

## S4 method for signature 'spec_tbl_df'
aggregate_duplicates(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  aggregation_function = sum,
  keep_integer = TRUE
)

## S4 method for signature 'tbl_df'
aggregate_duplicates(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  aggregation_function = sum,
  keep_integer = TRUE
)

## S4 method for signature 'tidybulk'
aggregate_duplicates(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  aggregation_function = sum,
  keep_integer = TRUE
)

## S4 method for signature 'SummarizedExperiment'
aggregate_duplicates(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  aggregation_function = sum,
  keep_integer = TRUE
)

## S4 method for signature 'RangedSummarizedExperiment'
aggregate_duplicates(
  .data,
  .sample = NULL,
  .transcript = NULL,
```

```
  .abundance = NULL,
  aggregation_function = sum,
  keep_integer = TRUE
)
```

Arguments

.data	A ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment))
.sample	The name of the sample column
.transcript	The name of the transcript/gene column
.abundance	The name of the transcript/gene abundance column
aggregation_function	A function for counts aggregation (e.g., sum, median, or mean)
keep_integer	A boolean. Whether to force the aggregated counts to integer

Details

‘r lifecycle::badge("maturing")‘

This function aggregates duplicated transcripts (e.g., isoforms, ensembl). For example, we often have to convert ensembl symbols to gene/transcript symbol, but in doing so we have to deal with duplicates. ‘aggregate_duplicates‘ takes a tibble and column names (as symbols; for ‘sample‘, ‘transcript‘ and ‘count‘) as arguments and returns a tibble with aggregate transcript with the same name. All the rest of the column are appended, and factors and boolean are appended as characters.

Underlying custom method: data |> filter(n_aggr > 1) |> group_by(!!.sample,!!.transcript) |> dplyr::mutate(!!.abundance := !!.abundance |> aggregation_function())

Value

A consistent object (to the input) with aggregated transcript abundance and annotation

A consistent object (to the input) with aggregated transcript abundance and annotation

A consistent object (to the input) with aggregated transcript abundance and annotation

A consistent object (to the input) with aggregated transcript abundance and annotation

A ‘SummarizedExperiment‘ object

A ‘SummarizedExperiment‘ object

Examples

```
# Create a aggregation column
se_mini = tidybulk::se_mini
SummarizedExperiment::rowData(se_mini )$gene_name = rownames(se_mini )

   aggregate_duplicates(
     se_mini,
   .transcript = gene_name
```

```
)
```

`arrange` *Arrange rows by column values*

Description

'arrange()' order the rows of a data frame rows by the values of selected columns.

Unlike other dplyr verbs, 'arrange()' largely ignores grouping; you need to explicit mention grouping variables (or use 'by_group = TRUE') in order to group by them, and functions of variables are evaluated once per data frame, not once per group.

Arguments

`.data`	A data frame, data frame extension (e.g. a tibble), or a lazy data frame (e.g. from dbplyr or dtplyr). See *Methods*, below, for more details.
`...`	<['tidy-eval'][dplyr_tidy_eval]> Variables, or functions or variables. Use [desc()] to sort a variable in descending order.
`.by_group`	If TRUE, will sort first by grouping variable. Applies to grouped data frames only.

Details

Locales The sort order for character vectors will depend on the collating sequence of the locale in use: see [locales()].

Missing values Unlike base sorting with 'sort()', 'NA' are: * always sorted to the end for local data, even when wrapped with 'desc()'. * treated differently for remote data, depending on the backend.

Value

An object of the same type as '.data'.

* All rows appear in the output, but (usually) in a different place. * Columns are not modified. * Groups are not modified. * Data frame attributes are preserved.

A tibble

Methods

This function is a **generic**, which means that packages can provide implementations (methods) for other classes. See the documentation of individual methods for extra arguments and differences in behaviour.

The following methods are currently available in loaded packages:

See Also

Other single table verbs: `filter()`, `mutate()`, `rename()`, `summarise()`

Examples

```
arrange(mtcars, cyl, disp)
```

`as_matrix` *Get matrix from tibble*

Description

Get matrix from tibble

Usage

```
as_matrix(tbl, rownames = NULL, do_check = TRUE)
```

Arguments

`tbl`	A tibble
`rownames`	The column name of the input tibble that will become the rownames of the output matrix
`do_check`	A boolean

Value

A matrix

Examples

```
tibble(.feature = "CD3G", count=1) |> as_matrix(rownames=.feature)
```

as_SummarizedExperiment *as_SummarizedExperiment*

Description

as_SummarizedExperiment() creates a ‘SummarizedExperiment‘ object from a ‘tbl‘ or ‘tidybulk‘ tbl formatted as | <SAMPLE> | <TRANSCRIPT> | <COUNT> | <...> |

Usage

```
as_SummarizedExperiment(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL
)

## S4 method for signature 'spec_tbl_df'
as_SummarizedExperiment(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL
)

## S4 method for signature 'tbl_df'
as_SummarizedExperiment(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL
)

## S4 method for signature 'tidybulk'
as_SummarizedExperiment(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL
)
```

Arguments

.data	A tibble
.sample	The name of the sample column

.transcript	The name of the transcript/gene column
.abundance	The name of the transcript/gene abundance column

Value

A 'SummarizedExperiment' object

A 'SummarizedExperiment' object

A 'SummarizedExperiment' object

A 'SummarizedExperiment' object

`bind_cols` *Left join datasets*

Description

Left join datasets

Arguments

x	tbls to join. (See dplyr)
y	tbls to join. (See dplyr)
by	A character vector of variables to join by. (See dplyr)
copy	If x and y are not from the same data source, and copy is TRUE, then y will be copied into the same src as x. (See dplyr)
suffix	If there are non-joined duplicate variables in x and y, these suffixes will be added to the output to disambiguate them. Should be a character vector of length 2. (See dplyr)
...	Data frames to combine (See dplyr)

Value

A tt object

Examples

```
annotation = tidybulk::se_mini |> tidybulk() |> as_tibble() |> distinct(.sample) |> mutate(source = "AU")
tidybulk::se_mini |> tidybulk() |> as_tibble() |> left_join(annotation)
```

bind_rows *Efficiently bind multiple data frames by row and column*

Description

This is an efficient implementation of the common pattern of ‘do.call(rbind, dfs)‘ or ‘do.call(cbind, dfs)‘ for binding many data frames into one.

Arguments

...	Data frames to combine. Each argument can either be a data frame, a list that could be a data frame, or a list of data frames. When row-binding, columns are matched by name, and any missing columns will be filled with NA. When column-binding, rows are matched by position, so all data frames must have the same number of rows. To match by value, not position, see mutate-joins.
.id	Data frame identifier. When ‘.id‘ is supplied, a new column of identifiers is created to link each row to its original data frame. The labels are taken from the named arguments to ‘bind_rows()‘. When a list of data frames is supplied, the labels are taken from the names of the list. If no names are found a numeric sequence is used instead.
add.cell.ids	from Seurat 3.0 A character vector of length(x = c(x, y)). Appends the corresponding values to the start of each objects’ cell names.

Details

The output of ‘bind_rows()‘ will contain a column if that column appears in any of the inputs.

Value

‘bind_rows()‘ and ‘bind_cols()‘ return the same type as the first input, either a data frame, ‘tbl_df‘, or ‘grouped_df‘.

Examples

```
data(se_mini)

se_mini_tidybulk = se_mini |> tidybulk()
bind_rows(    se_mini_tidybulk, se_mini_tidybulk  )

tt_bind = se_mini_tidybulk |> select(time, condition)
se_mini_tidybulk |> bind_cols(tt_bind)
```

breast_tcga_mini_SE *Needed for vignette breast_tcga_mini_SE*

Description

Needed for vignette breast_tcga_mini_SE

Usage

```
breast_tcga_mini_SE
```

Format

An object of class `SummarizedExperiment` with 500 rows and 251 columns.

check_if_counts_is_na *Check whether there are NA counts*

Description

Check whether there are NA counts

Usage

```
check_if_counts_is_na(.data, .abundance)
```

Arguments

`.data`	A tibble of read counts
`.abundance`	A character name of the read count column

Value

A tbl

check_if_duplicated_genes *Check whether there are duplicated genes/transcripts*

Description

Check whether there are duplicated genes/transcripts

Usage

```
check_if_duplicated_genes(
  .data,
  .sample = sample,
  .transcript = transcript,
  .abundance = `read count`
)
```

Arguments

.data	A tibble of read counts
.sample	A character name of the sample column
.transcript	A character name of the transcript/gene column
.abundance	A character name of the read count column

Value

A tbl

check_if_wrong_input *Check whether there are NA counts*

Description

Check whether there are NA counts

Usage

```
check_if_wrong_input(.data, list_input, expected_type)
```

Arguments

.data	A tibble of read counts
list_input	A list
expected_type	A character string

Value

A tbl

`cluster_elements` *Get clusters of elements (e.g., samples or transcripts)*

Description

cluster_elements() takes as input A ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment)) and identify clusters in the data.

Usage

```
cluster_elements(
  .data,
  .element = NULL,
  .feature = NULL,
  .abundance = NULL,
  method,
  of_samples = TRUE,
  transform = log1p,
  action = "add",
  ...,
  log_transform = NULL
)

## S4 method for signature 'spec_tbl_df'
cluster_elements(
  .data,
  .element = NULL,
  .feature = NULL,
  .abundance = NULL,
  method,
  of_samples = TRUE,
  transform = log1p,
  action = "add",
  ...,
  log_transform = NULL
)

## S4 method for signature 'tbl_df'
cluster_elements(
  .data,
  .element = NULL,
  .feature = NULL,
```

```
  .abundance = NULL,
  method,
  of_samples = TRUE,
  transform = log1p,
  action = "add",
  ...,
  log_transform = NULL
)

## S4 method for signature 'tidybulk'
cluster_elements(
  .data,
  .element = NULL,
  .feature = NULL,
  .abundance = NULL,
  method,
  of_samples = TRUE,
  transform = log1p,
  action = "add",
  ...,
  log_transform = NULL
)

## S4 method for signature 'SummarizedExperiment'
cluster_elements(
  .data,
  .element = NULL,
  .feature = NULL,
  .abundance = NULL,
  method,
  of_samples = TRUE,
  transform = log1p,
  action = "add",
  ...,
  log_transform = NULL
)

## S4 method for signature 'RangedSummarizedExperiment'
cluster_elements(
  .data,
  .element = NULL,
  .feature = NULL,
  .abundance = NULL,
  method,
  of_samples = TRUE,
  transform = log1p,
  action = "add",
  ...,
```

```
  log_transform = NULL
)
```

Arguments

.data	A ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment))
.element	The name of the element column (normally samples).
.feature	The name of the feature column (normally transcripts/genes)
.abundance	The name of the column including the numerical value the clustering is based on (normally transcript abundance)
method	A character string. The cluster algorithm to use, at the moment k-means is the only algorithm included.
of_samples	A boolean. In case the input is a tidybulk object, it indicates Whether the element column will be sample or transcript column
transform	A function that will tranform the counts, by default it is log1p for RNA sequencing data, but for avoinding tranformation you can use identity
action	A character string. Whether to join the new information to the input tbl (add), or just get the non-redundant tbl with the new information (get).
...	Further parameters passed to the function kmeans
log_transform	DEPRECATED - A boolean, whether the value should be log-transformed (e.g., TRUE for RNA sequencing data)

Details

‘r lifecycle::badge("maturing")‘

identifies clusters in the data, normally of samples. This function returns a tibble with additional columns for the cluster annotation. At the moment only k-means (DOI: 10.2307/2346830) and SNN clustering (DOI:10.1016/j.cell.2019.05.031) is supported, the plan is to introduce more clustering methods.

Underlying method for kmeans do.call(kmeans(.data, iter.max = 1000, ...)

Underlying method for SNN .data Seurat::CreateSeuratObject() Seurat::ScaleData(display.progress = TRUE,num.cores = 4, do.par = TRUE) Seurat::FindVariableFeatures(selection.method = "vst") Seurat::RunPCA(npcs = 30) Seurat::FindNeighbors() Seurat::FindClusters(method = "igraph", ...)

Value

A tbl object with additional columns with cluster labels

A tbl object with additional columns with cluster labels

A tbl object with additional columns with cluster labels

A tbl object with additional columns with cluster labels

A ‘SummarizedExperiment‘ object

A ‘SummarizedExperiment‘ object

Examples

```
cluster_elements(tidybulk::se_mini, centers = 2, method="kmeans")
```

`counts_ensembl` *Counts with ensembl annotation*

Description

Counts with ensembl annotation

Usage

```
counts_ensembl
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 119 rows and 6 columns.

`deconvolve_cellularity`
Get cell type proportions from samples

Description

deconvolve_cellularity() takes as input A ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment)) and returns a consistent object (to the input) with the estimated cell type abundance for each sample

Usage

```
deconvolve_cellularity(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  reference = NULL,
  method = "cibersort",
  prefix = "",
  action = "add",
  ...
)
```

```
## S4 method for signature 'spec_tbl_df'
deconvolve_cellularity(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  reference = NULL,
  method = "cibersort",
  prefix = "",
  action = "add",
  ...
)

## S4 method for signature 'tbl_df'
deconvolve_cellularity(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  reference = NULL,
  method = "cibersort",
  prefix = "",
  action = "add",
  ...
)

## S4 method for signature 'tidybulk'
deconvolve_cellularity(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  reference = NULL,
  method = "cibersort",
  prefix = "",
  action = "add",
  ...
)

## S4 method for signature 'SummarizedExperiment'
deconvolve_cellularity(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  reference = NULL,
  method = "cibersort",
  prefix = "",
```

```
  action = "add",
  ...
)

## S4 method for signature 'RangedSummarizedExperiment'
deconvolve_cellularity(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  reference = NULL,
  method = "cibersort",
  prefix = "",
  action = "add",
  ...
)
```

Arguments

.data	A ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment))
.sample	The name of the sample column
.transcript	The name of the transcript/gene column
.abundance	The name of the transcript/gene abundance column
reference	A data frame. The methods cibersort and llsr can accept a custom rectangular dataframe with genes as rows names, cell types as column names and gene-transcript abundance as values. For exampler tidybulk::X_cibersort. The transcript/cell_type data frame of integer transcript abundance. If NULL, the default reference for each algorithm will be used. For llsr will be LM22.
method	A character string. The method to be used. At the moment Cibersort (default, can accept custom reference), epic (can accept custom reference) and llsr (linear least squares regression, can accept custom reference), mcp_counter, quantiseq, xcell are available.
prefix	A character string. The prefix you would like to add to the result columns. It is useful if you want to reshape data.
action	A character string. Whether to join the new information to the input tbl (add), or just get the non-redundant tbl with the new information (get).
...	Further parameters passed to the function Cibersort

Details

‘r lifecycle::badge("maturing")‘

This function infers the cell type composition of our samples (with the algorithm Cibersort; Newman et al., 10.1038/nmeth.3337).

Underlying method: CIBERSORT(Y = data, X = reference, ...)

Value

A consistent object (to the input) including additional columns for each cell type estimated

A consistent object (to the input) including additional columns for each cell type estimated

A consistent object (to the input) including additional columns for each cell type estimated

A consistent object (to the input) including additional columns for each cell type estimated

A 'SummarizedExperiment' object

A 'SummarizedExperiment' object

Examples

```
# Subsetting for time efficiency
tidybulk::se_mini |> deconvolve_cellularity(cores = 1)
```

---

describe_transcript *Get DESCRIPTION from gene SYMBOL for Human and Mouse*

Description

Get DESCRIPTION from gene SYMBOL for Human and Mouse

describe_transcript

describe_transcript

describe_transcript

describe_transcript

describe_transcript

describe_transcript

Usage

```
describe_transcript(.data, .transcript = NULL)

## S4 method for signature 'spec_tbl_df'
describe_transcript(.data, .transcript = NULL)

## S4 method for signature 'tbl_df'
describe_transcript(.data, .transcript = NULL)

## S4 method for signature 'tidybulk'
describe_transcript(.data, .transcript = NULL)

.describe_transcript_SE(.data, .transcript = NULL)
```

```
## S4 method for signature 'SummarizedExperiment'
describe_transcript(.data, .transcript = NULL)

## S4 method for signature 'RangedSummarizedExperiment'
describe_transcript(.data, .transcript = NULL)
```

Arguments

.data	A tt or tbl object.
.transcript	A character. The name of the gene symbol column.

Value

A tbl

A consistent object (to the input) including additional columns for transcript symbol

A consistent object (to the input) including additional columns for transcript symbol

A consistent object (to the input) including additional columns for transcript symbol

A 'SummarizedExperiment' object

A consistent object (to the input) including additional columns for transcript symbol

A consistent object (to the input) including additional columns for transcript symbol

Examples

```
describe_transcript(tidybulk::se_mini)
```

---

distinct *distinct*

Description

distinct

Arguments

.data	A tbl. (See dplyr)
...	Data frames to combine (See dplyr)
.keep_all	If TRUE, keep all variables in .data. If a combination of ... is not distinct, this keeps the first row of values. (See dplyr)

Value

A tt object

Examples

```
tidybulk::se_mini |> tidybulk() |> distinct()
```

`ensembl_symbol_mapping` *Data set*

Description

Data set

Usage

```
ensembl_symbol_mapping
```

Format

An object of class `spec_tbl_df` (inherits from `tbl_df`, `tbl`, `data.frame`) with 291249 rows and 3 columns.

`ensembl_to_symbol` *Add transcript symbol column from ensembl id for human and mouse data*

Description

ensembl_to_symbol() takes as input a ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment)) and returns a consistent object (to the input) with the additional transcript symbol column

Usage

```
ensembl_to_symbol(.data, .ensembl, action = "add")

## S4 method for signature 'spec_tbl_df'
ensembl_to_symbol(.data, .ensembl, action = "add")

## S4 method for signature 'tbl_df'
ensembl_to_symbol(.data, .ensembl, action = "add")

## S4 method for signature 'tidybulk'
ensembl_to_symbol(.data, .ensembl, action = "add")
```

Arguments

.data	a ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment))
.ensembl	A character string. The column that is represents ensembl gene id
action	A character string. Whether to join the new information to the input tbl (add), or just get the non-redundant tbl with the new information (get).

Details

[Questioning]

This is useful since different resources use ensembl IDs while others use gene symbol IDs. At the moment this work for human (genes and transcripts) and mouse (genes) data.

Value

A consistent object (to the input) including additional columns for transcript symbol

A consistent object (to the input) including additional columns for transcript symbol

A consistent object (to the input) including additional columns for transcript symbol

A consistent object (to the input) including additional columns for transcript symbol

Examples

```
# This function was designed for data.frame
# Convert from SummarizedExperiment for this example. It is NOT reccomended.

tidybulk::se_mini |> tidybulk() |> as_tibble() |> ensembl_to_symbol(.feature)
```

---

fill_missing_abundance

Fill transcript abundance if missing from sample-transcript pairs

---

Description

fill_missing_abundance() takes as input A ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment)) and returns a consistent object (to the input) with new observations

Usage

```
fill_missing_abundance(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  fill_with
)

## S4 method for signature 'spec_tbl_df'
fill_missing_abundance(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  fill_with
)

## S4 method for signature 'tbl_df'
fill_missing_abundance(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  fill_with
)

## S4 method for signature 'tidybulk'
fill_missing_abundance(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  fill_with
)
```

Arguments

.data	A ‘tbl‘ formatted as	<SAMPLE>	<TRANSCRIPT>	<COUNT>	<...>	
.sample	The name of the sample column
.transcript	The name of the transcript column
.abundance	The name of the transcript abundance column
fill_with	A numerical abundance with which fill the missing data points

Details

[Questioning]

This function fills the abundance of missing sample-transcript pair using the median of the sample group defined by the formula

Value

A consistent object (to the input) non-sparse abundance

A consistent object (to the input) with filled abundance

A consistent object (to the input) with filled abundance

A consistent object (to the input) with filled abundance

Examples

```
print("Not run for build time.")

# tidybulk::se_mini |>  fill_missing_abundance( fill_with = 0)
```

`filter` *Subset rows using column values*

Description

‘filter()‘ retains the rows where the conditions you provide a ‘TRUE‘. Note that, unlike base subsetting with ‘[‘, rows where the condition evaluates to ‘NA‘ are dropped.

Arguments

`.data`	A tbl. (See dplyr)
`...`	<[‘tidy-eval‘][dplyr_tidy_eval]> Logical predicates defined in terms of the variables in ‘.data‘. Multiple conditions are combined with ‘&‘. Only rows where the condition evaluates to ‘TRUE‘ are kept.
`.preserve`	when ‘FALSE‘ (the default), the grouping structure is recalculated based on the resulting data, otherwise it is kept as is.

Details

dplyr is not yet smart enough to optimise filtering optimisation on grouped datasets that don’t need grouped calculations. For this reason, filtering is often considerably faster on [ungroup()]ed data.

Value

An object of the same type as ‘.data‘.

* Rows are a subset of the input, but appear in the same order. * Columns are not modified. * The number of groups may be reduced (if ‘.preserve‘ is not ‘TRUE‘). * Data frame attributes are preserved.

Useful filter functions

* [‘==‘], [‘>‘], [‘>=‘] etc * [‘&‘], [‘|‘], [‘¡], [xor()] * [is.na()] * [between()], [near()]

Grouped tibbles

Because filtering expressions are computed within groups, they may yield different results on grouped tibbles. This will be the case as soon as an aggregating, lagging, or ranking function is involved. Compare this ungrouped filtering:

The former keeps rows with ‘mass‘ greater than the global average whereas the latter keeps rows with ‘mass‘ greater than the gender

average.

Methods

This function is a **generic**, which means that packages can provide implementations (methods) for other classes. See the documentation of individual methods for extra arguments and differences in behaviour.

The following methods are currently available in loaded packages:

See Also

[filter_all()], [filter_if()] and [filter_at()].

Other single table verbs: `arrange()`, `mutate()`, `rename()`, `summarise()`

Examples

```
data(se)

se |> tidybulk() |> filter(dex=="untrt")

# Learn more in ?dplyr_tidy_eval
```

`flybaseIDs` *flybaseIDs*

Description

flybaseIDs

Usage

```
flybaseIDs
```

Format

An object of class `character` of length 14599.

`get_bibliography` *Produces the bibliography list of your workflow*

Description

get_bibliography() takes as input a ‘tidybulk‘

Usage

```
get_bibliography(.data)

## S4 method for signature 'tbl'
get_bibliography(.data)

## S4 method for signature 'tbl_df'
get_bibliography(.data)

## S4 method for signature 'spec_tbl_df'
get_bibliography(.data)

## S4 method for signature 'tidybulk'
get_bibliography(.data)

## S4 method for signature 'SummarizedExperiment'
get_bibliography(.data)

## S4 method for signature 'RangedSummarizedExperiment'
get_bibliography(.data)
```

Arguments

`.data`	A ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment))

Details

‘r lifecycle::badge("maturing")‘

This methods returns the bibliography list of your workflow from the internals of a tidybulk object (attr(., "internals"))

Value

NULL. It prints a list of bibliography references for the software used through the workflow.

A consistent object (to the input) with additional columns for the statistics from the hypothesis test (e.g., log fold change, p-value and false discovery rate).

A consistent object (to the input) with additional columns for the statistics from the hypothesis test (e.g., log fold change, p-value and false discovery rate).

Examples

```
get_bibliography(tidybulk::se_mini)
```

`group_by` *Group by one or more variables*

Description

Most data operations are done on groups defined by variables. ‘group_by()‘ takes an existing tbl and converts it into a grouped tbl where operations are performed "by group". ‘ungroup()‘ removes grouping.

Arguments

`.data`	A tbl. (See dplyr)
`...`	In ‘group_by()‘, variables or computations to group by. In ‘ungroup()‘, variables to remove from the grouping.
`.add`	When ‘FALSE‘, the default, ‘group_by()‘ will override existing groups. To add to the existing groups, use ‘.add = TRUE‘. This argument was previously called ‘add‘, but that prevented creating a new grouping variable called ‘add‘, and conflicts with our naming conventions.
`.drop`	When ‘.drop = TRUE‘, empty groups are dropped. See [group_by_drop_default()] for what the default value is for this argument.

Value

A [grouped data frame][grouped_df()], unless the combination of ‘...‘ and ‘add‘ yields a non empty set of grouping columns, a regular (ungrouped) data frame otherwise.

Methods

These function are **generic**s, which means that packages can provide implementations (methods) for other classes. See the documentation of individual methods for extra arguments and differences in behaviour.

Methods available in currently loaded packages:

Examples

```
by_cyl <- mtcars |> group_by(cyl)
```

identify_abundant *Identify abundant transcripts/genes*

Description

Identifies transcripts/genes that are consistently expressed above a threshold across samples. This function adds a logical column '.abundant' to indicate which features pass the filtering criteria.

Usage

```
identify_abundant(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  factor_of_interest = NULL,
  design = NULL,
  minimum_counts = 10,
  minimum_proportion = 0.7
)

## S4 method for signature 'spec_tbl_df'
identify_abundant(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  factor_of_interest = NULL,
  design = NULL,
  minimum_counts = 10,
  minimum_proportion = 0.7
)

## S4 method for signature 'tbl_df'
identify_abundant(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  factor_of_interest = NULL,
  design = NULL,
  minimum_counts = 10,
  minimum_proportion = 0.7
)

## S4 method for signature 'tidybulk'
identify_abundant(
```

```
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  factor_of_interest = NULL,
  design = NULL,
  minimum_counts = 10,
  minimum_proportion = 0.7
)

## S4 method for signature 'SummarizedExperiment'
identify_abundant(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  factor_of_interest = NULL,
  design = NULL,
  minimum_counts = 10,
  minimum_proportion = 0.7
)

## S4 method for signature 'RangedSummarizedExperiment'
identify_abundant(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  factor_of_interest = NULL,
  design = NULL,
  minimum_counts = 10,
  minimum_proportion = 0.7
)
```

Arguments

`.data`	A ‘tbl‘ or ‘SummarizedExperiment‘ object containing transcript/gene abundance data
`.sample`	The name of the sample column
`.transcript`	The name of the transcript/gene column
`.abundance`	The name of the transcript/gene abundance column
`factor_of_interest`	The name of the column containing groups/conditions for filtering. Used by edgeR’s filterByExpr to define sample groups.
`design`	A design matrix for more complex experimental designs. If provided, this is passed to filterByExpr instead of factor_of_interest.
`minimum_counts`	A positive number specifying the minimum counts per million (CPM) threshold for a transcript to be considered abundant (default = 10)

minimum_proportion
: A number between 0 and 1 specifying the minimum proportion of samples that must exceed the minimum_counts threshold (default = 0.7)

Details

‘r lifecycle::badge("maturing")‘

This function uses edgeR’s filterByExpr() function to identify consistently expressed features. A feature is considered abundant if it has CPM > minimum_counts in at least minimum_proportion of samples in at least one experimental group (defined by factor_of_interest or design).

Value

Returns the input object with an additional logical column ‘.abundant‘ indicating which features passed the abundance threshold criteria.

A consistent object (to the input) with additional columns for the statistics from the hypothesis test (e.g., log fold change, p-value and false discovery rate).

A consistent object (to the input) with additional columns for the statistics from the hypothesis test (e.g., log fold change, p-value and false discovery rate).

A consistent object (to the input) with additional columns for the statistics from the hypothesis test (e.g., log fold change, p-value and false discovery rate).

A ‘SummarizedExperiment‘ object

A ‘SummarizedExperiment‘ object

References

McCarthy, D. J., Chen, Y., & Smyth, G. K. (2012). Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation. Nucleic Acids Research, 40(10), 4288-4297. DOI: 10.1093/bioinformatics/btp616

Examples

```
# Basic usage
se_mini |> identify_abundant()

# With custom thresholds
se_mini |> identify_abundant(
  minimum_counts = 5,
  minimum_proportion = 0.5
)

# Using a factor of interest
se_mini |> identify_abundant(factor_of_interest = condition)
```

impute_missing_abundance *impute transcript abundance if missing from sample-transcript pairs*

Description

impute_missing_abundance() takes as input A ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment)) and returns a consistent object (to the input) with additional sample-transcript pairs with imputed transcript abundance.

Usage

```
impute_missing_abundance(
  .data,
  .formula,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  suffix = "",
  force_scaling = FALSE
)

## S4 method for signature 'spec_tbl_df'
impute_missing_abundance(
  .data,
  .formula,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  suffix = "",
  force_scaling = FALSE
)

## S4 method for signature 'tbl_df'
impute_missing_abundance(
  .data,
  .formula,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  suffix = "",
  force_scaling = FALSE
)

## S4 method for signature 'tidybulk'
impute_missing_abundance(
```

```
  .data,
  .formula,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  suffix = "",
  force_scaling = FALSE
)

## S4 method for signature 'SummarizedExperiment'
impute_missing_abundance(
  .data,
  .formula,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  suffix = "",
  force_scaling = FALSE
)

## S4 method for signature 'RangedSummarizedExperiment'
impute_missing_abundance(
  .data,
  .formula,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  suffix = "",
  force_scaling = FALSE
)
```

Arguments

.data	A ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment))
.formula	A formula with no response variable, representing the desired linear model where the first covariate is the factor of interest and the second covariate is the unwanted variation (of the kind ~ factor_of_interest + batch)
.sample	The name of the sample column
.transcript	The name of the transcript/gene column
.abundance	The name of the transcript/gene abundance column
suffix	A character string. This is added to the imputed count column names. If empty the count column are overwritten
force_scaling	A boolean. In case a abundance-containing column is not scaled (columns with _scale suffix), setting force_scaling = TRUE will result in a scaling by library size, to compensating for a possible difference in sequencing depth.

Details

‘r lifecycle::badge("maturing")‘

This function imputes the abundance of missing sample-transcript pair using the median of the sample group defined by the formula

Value

A consistent object (to the input) non-sparse abundance

A consistent object (to the input) with imputed abundance

A consistent object (to the input) with imputed abundance

A consistent object (to the input) with imputed abundance

A ‘SummarizedExperiment‘ object

A ‘SummarizedExperiment‘ object

Examples

```
res =
 impute_missing_abundance(
  tidybulk::se_mini,
 ~ condition
)
```

`inner_join` *Inner join datasets*

Description

Inner join datasets

Right join datasets

Full join datasets

Arguments

`x`	tbls to join. (See dplyr)
`y`	tbls to join. (See dplyr)
`by`	A character vector of variables to join by. (See dplyr)
`copy`	If x and y are not from the same data source, and copy is TRUE, then y will be copied into the same src as x. (See dplyr)
`suffix`	If there are non-joined duplicate variables in x and y, these suffixes will be added to the output to disambiguate them. Should be a character vector of length 2. (See dplyr)
`...`	Data frames to combine (See dplyr)

Value

A tt object

A tt object

A tt object

Examples

```
annotation = tidybulk::se_mini |> tidybulk() |> as_tibble() |> distinct(.sample) |> mutate(source = "AU")
tidybulk::se_mini |> tidybulk() |> as_tibble() |> inner_join(annotation)


annotation = tidybulk::se_mini |> tidybulk() |> as_tibble() |> distinct(.sample) |> mutate(source = "AU")
tidybulk::se_mini |> tidybulk() |> as_tibble() |> right_join(annotation)


annotation = tidybulk::se_mini |> tidybulk() |> as_tibble() |> distinct(.sample) |> mutate(source = "AU")
tidybulk::se_mini |> tidybulk() |> as_tibble() |> full_join(annotation)
```

---

keep_abundant *Filter to keep only abundant transcripts/genes*

Description

Filters the data to keep only transcripts/genes that are consistently expressed above a threshold across samples. This is a filtering version of identify_abundant() that removes low-abundance features instead of just marking them.

Usage

```
keep_abundant(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  factor_of_interest = NULL,
  design = NULL,
  minimum_counts = 10,
  minimum_proportion = 0.7
)

## S4 method for signature 'spec_tbl_df'
keep_abundant(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
```

```
  factor_of_interest = NULL,
  design = NULL,
  minimum_counts = 10,
  minimum_proportion = 0.7
)

## S4 method for signature 'tbl_df'
keep_abundant(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  factor_of_interest = NULL,
  design = NULL,
  minimum_counts = 10,
  minimum_proportion = 0.7
)

## S4 method for signature 'tidybulk'
keep_abundant(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  factor_of_interest = NULL,
  design = NULL,
  minimum_counts = 10,
  minimum_proportion = 0.7
)

## S4 method for signature 'SummarizedExperiment'
keep_abundant(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  factor_of_interest = NULL,
  design = NULL,
  minimum_counts = 10,
  minimum_proportion = 0.7
)

## S4 method for signature 'RangedSummarizedExperiment'
keep_abundant(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
```

```
  factor_of_interest = NULL,
  design = NULL,
  minimum_counts = 10,
  minimum_proportion = 0.7
)
```

Arguments

.data	A 'tbl' or 'SummarizedExperiment' object containing transcript/gene abundance data
.sample	The name of the sample column
.transcript	The name of the transcript/gene column
.abundance	The name of the transcript/gene abundance column
factor_of_interest	The name of the column containing groups/conditions for filtering. Used by edgeR's filterByExpr to define sample groups.
design	A design matrix for more complex experimental designs. If provided, this is passed to filterByExpr instead of factor_of_interest.
minimum_counts	A positive number specifying the minimum counts per million (CPM) threshold for a transcript to be kept (default = 10)
minimum_proportion	A number between 0 and 1 specifying the minimum proportion of samples that must exceed the minimum_counts threshold (default = 0.7)

Details

[Questioning]

This function uses edgeR's filterByExpr() function to identify and keep consistently expressed features. A feature is kept if it has CPM > minimum_counts in at least minimum_proportion of samples in at least one experimental group (defined by factor_of_interest or design).

This function is similar to identify_abundant() but instead of adding an .abundant column, it filters out the low-abundance features directly.

Value

Returns a filtered version of the input object containing only the features that passed the abundance threshold criteria.

A consistent object (to the input) with additional columns for the statistics from the hypothesis test (e.g., log fold change, p-value and false discovery rate).

A consistent object (to the input) with additional columns for the statistics from the hypothesis test (e.g., log fold change, p-value and false discovery rate).

A consistent object (to the input) with additional columns for the statistics from the hypothesis test (e.g., log fold change, p-value and false discovery rate).

A 'SummarizedExperiment' object

A 'SummarizedExperiment' object

References

McCarthy, D. J., Chen, Y., & Smyth, G. K. (2012). Differential expression analysis of multifactor RNA-Seq experiments with respect to biological variation. Nucleic Acids Research, 40(10), 4288-4297. DOI: 10.1093/bioinformatics/btp616

Examples

```
# Basic usage
se_mini |> keep_abundant()

# With custom thresholds
se_mini |> keep_abundant(
  minimum_counts = 5,
  minimum_proportion = 0.5
)

# Using a factor of interest
se_mini |> keep_abundant(factor_of_interest = condition)
```

---

keep_variable *Keep variable transcripts*

Description

keep_variable() takes as input A ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment)) and returns a consistent object (to the input) with additional columns for the statistics from the hypothesis test.

Usage

```
keep_variable(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  top = 500,
  transform = log1p,
  log_transform = TRUE
)

## S4 method for signature 'spec_tbl_df'
keep_variable(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
```

```
  top = 500,
  transform = log1p,
  log_transform = NULL
)

## S4 method for signature 'tbl_df'
keep_variable(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  top = 500,
  transform = log1p,
  log_transform = NULL
)

## S4 method for signature 'tidybulk'
keep_variable(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  top = 500,
  transform = log1p,
  log_transform = NULL
)

## S4 method for signature 'SummarizedExperiment'
keep_variable(.data, top = 500, transform = log1p)

## S4 method for signature 'RangedSummarizedExperiment'
keep_variable(.data, top = 500, transform = log1p)
```

Arguments

.data	A ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment))
.sample	The name of the sample column
.transcript	The name of the transcript/gene column
.abundance	The name of the transcript/gene abundance column
top	Integer. Number of top transcript to consider
transform	A function that will tranform the counts, by default it is log1p for RNA sequencing data, but for avoinding tranformation you can use identity
log_transform	DEPRECATED - A boolean, whether the value should be log-transformed (e.g., TRUE for RNA sequencing data)

Details

‘r lifecycle::badge("maturing")‘

At the moment this function uses edgeR https://doi.org/10.1093/bioinformatics/btp616

Value

A consistent object (to the input) with additional columns for the statistics from the hypothesis test (e.g., log fold change, p-value and false discovery rate).

Underlying method: s <- rowMeans((x - rowMeans(x)) ^ 2) o <- order(s, decreasing = TRUE) x <- x[o[1L:top], , drop = FALSE] variable_trancripts = rownames(x)

A consistent object (to the input) with additional columns for the statistics from the hypothesis test (e.g., log fold change, p-value and false discovery rate).

A consistent object (to the input) with additional columns for the statistics from the hypothesis test (e.g., log fold change, p-value and false discovery rate).

A consistent object (to the input) with additional columns for the statistics from the hypothesis test (e.g., log fold change, p-value and false discovery rate).

A ‘SummarizedExperiment‘ object

A ‘SummarizedExperiment‘ object

Examples

```
keep_variable(tidybulk::se_mini, top = 500)
```

---

log10_reverse_trans *log10_reverse_trans*

Description

it perform log scaling and reverse the axis. Useful to plot negative log probabilities. To not be used directly but with ggplot (e.g. scale_y_continuous(trans = "log10_reverse"))

Usage

```
log10_reverse_trans()
```

Details

‘r lifecycle::badge("maturing")‘

Value

A scales object

Examples

```
library(ggplot2)
library(tibble)

tibble(pvalue = c(0.001, 0.05, 0.1), fold_change = 1:3) %>%
 ggplot(aes(fold_change , pvalue)) +
 geom_point() +
 scale_y_continuous(trans = "log10_reverse")
```

---

logit_trans *logit scale*

Description

it perform logit scaling with right axis formatting. To not be used directly but with ggplot (e.g. scale_y_continuous(trans = "log10_reverse"))

Usage

```
logit_trans()
```

Details

‘r lifecycle::badge("maturing")‘

Value

A scales object

Examples

```
library(ggplot2)
library(tibble)

tibble(pvalue = c(0.001, 0.05, 0.1), fold_change = 1:3) %>%
 ggplot(aes(fold_change , pvalue)) +
 geom_point() +
 scale_y_continuous(trans = "log10_reverse")
```

mutate *Create, modify, and delete columns*

Description

'mutate()' adds new variables and preserves existing ones; 'transmute()' adds new variables and drops existing ones. New variables overwrite existing variables of the same name. Variables can be removed by setting their value to 'NULL'.

Arguments

.data	A tbl. (See dplyr)
...	<['tidy-eval'][dplyr_tidy_eval]> Name-value pairs. The name gives the name of the column in the output. The value can be: * A vector of length 1, which will be recycled to the correct length. * A vector the same length as the current group (or the whole data frame if ungrouped). * 'NULL', to remove the column. * A data frame or tibble, to create multiple columns in the output.

Value

An object of the same type as '.data'.

For 'mutate()':

* Rows are not affected. * Existing columns will be preserved unless explicitly modified. * New columns will be added to the right of existing columns. * Columns given value 'NULL' will be removed * Groups will be recomputed if a grouping variable is mutated. * Data frame attributes are preserved.

For 'transmute()':

* Rows are not affected. * Apart from grouping variables, existing columns will be remove unless explicitly kept. * Column order matches order of expressions. * Groups will be recomputed if a grouping variable is mutated. * Data frame attributes are preserved.

Useful mutate functions

* ['+'], ['-'], [log()], etc., for their usual mathematical meanings

* [lead()], [lag()]

* [dense_rank()], [min_rank()], [percent_rank()], [row_number()], [cume_dist()], [ntile()]

* [cumsum()], [cummean()], [cummin()], [cummax()], [cumany()], [cumall()]

* [na_if()], [coalesce()]

* [if_else()], [recode()], [case_when()]

Grouped tibbles

Because mutating expressions are computed within groups, they may yield different results on grouped tibbles. This will be the case as soon as an aggregating, lagging, or ranking function is involved. Compare this ungrouped mutate:

With the grouped equivalent:

The former normalises ‘mass‘ by the global average whereas the latter normalises by the averages within gender levels.

Methods

These function are **generic**s, which means that packages can provide implementations (methods) for other classes. See the documentation of individual methods for extra arguments and differences in behaviour.

Methods available in currently loaded packages:

See Also

Other single table verbs: `arrange()`, `filter()`, `rename()`, `summarise()`

Examples

```
# Newly created variables are available immediately
mtcars |> as_tibble() |> mutate(
  cyl2 = cyl * 2,
  cyl4 = cyl2 * 2
)
```

---

pivot_sample *Extract sample-wise information*

Description

pivot_sample() takes as input a ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment)) and returns a ‘tbl‘ with only sample-related columns

Usage

```
pivot_sample(.data, .sample = NULL)

## S4 method for signature 'spec_tbl_df'
pivot_sample(.data, .sample = NULL)

## S4 method for signature 'tbl_df'
pivot_sample(.data, .sample = NULL)
```

```
## S4 method for signature 'tidybulk'
pivot_sample(.data, .sample = NULL)

## S4 method for signature 'SummarizedExperiment'
pivot_sample(.data, .sample = NULL)

## S4 method for signature 'RangedSummarizedExperiment'
pivot_sample(.data, .sample = NULL)
```

Arguments

.data	A ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment))
.sample	The name of the sample column

Details

‘r lifecycle::badge("maturing")‘

This functon extracts only sample-related information for downstream analysis (e.g., visualisation). It is disruptive in the sense that it cannot be passed anymore to tidybulk function.

Value

A ‘tbl‘ with transcript-related information

A consistent object (to the input)

A consistent object (to the input)

Examples

```
pivot_sample(tidybulk::se_mini )
```

---

pivot_transcript *Extract transcript-wise information*

Description

pivot_transcript() takes as input a ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment)) and returns a ‘tbl‘ with only transcript-related columns

Usage

```
pivot_transcript(.data, .transcript = NULL)

## S4 method for signature 'spec_tbl_df'
pivot_transcript(.data, .transcript = NULL)

## S4 method for signature 'tbl_df'
pivot_transcript(.data, .transcript = NULL)

## S4 method for signature 'tidybulk'
pivot_transcript(.data, .transcript = NULL)

## S4 method for signature 'SummarizedExperiment'
pivot_transcript(.data, .transcript = NULL)

## S4 method for signature 'RangedSummarizedExperiment'
pivot_transcript(.data, .transcript = NULL)
```

Arguments

.data	A ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment))
.transcript	The name of the transcript column

Details

‘r lifecycle::badge("maturing")‘

This functon extracts only transcript-related information for downstream analysis (e.g., visualisation). It is disruptive in the sense that it cannot be passed anymore to tidybulk function.

Value

A ‘tbl‘ with transcript-related information

A consistent object (to the input)

A consistent object (to the input)

Examples

```
pivot_transcript(tidybulk::se_mini  )
```

quantile_normalise_abundance *Normalise by quantiles the counts of transcripts/genes*

Description

quantile_normalise_abundance() takes as input A ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment)) and Scales transcript abundance compansating for sequencing depth (e.g., with TMM algorithm, Robinson and Oshlack doi.org/10.1186/gb-2010-11-3-r25).

Usage

```
quantile_normalise_abundance(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  method = "limma_normalize_quantiles",
  target_distribution = NULL,
  action = "add"
)

## S4 method for signature 'spec_tbl_df'
quantile_normalise_abundance(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  method = "limma_normalize_quantiles",
  target_distribution = NULL,
  action = "add"
)

## S4 method for signature 'tbl_df'
quantile_normalise_abundance(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  method = "limma_normalize_quantiles",
  target_distribution = NULL,
  action = "add"
)

## S4 method for signature 'tidybulk'
```

```
quantile_normalise_abundance(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  method = "limma_normalize_quantiles",
  target_distribution = NULL,
  action = "add"
)

## S4 method for signature 'SummarizedExperiment'
quantile_normalise_abundance(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  method = "limma_normalize_quantiles",
  target_distribution = NULL,
  action = NULL
)

## S4 method for signature 'RangedSummarizedExperiment'
quantile_normalise_abundance(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  method = "limma_normalize_quantiles",
  target_distribution = NULL,
  action = NULL
)
```

Arguments

.data	A ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment))
.sample	The name of the sample column
.transcript	The name of the transcript/gene column
.abundance	The name of the transcript/gene abundance column
method	A character string. Either "limma_normalize_quantiles" for limma::normalizeQuantiles or "preprocesscore_normalize_quantiles_use_target" for preprocessCore::normalize.quantiles.use.target for large-scale datasets.
target_distribution	A numeric vector. If NULL the target distribution will be calculated by preprocessCore. This argument only affects the "preprocesscore_normalize_quantiles_use_target" method.

action A character string between "add" (default) and "only". "add" joins the new information to the input tbl (default), "only" return a non-redundant tbl with the just new information.

Details

‘r lifecycle::badge("maturing")‘

Tranform the feature abundance across samples so to have the same quantile distribution (using preprocessCore).

Underlying method

If ‘limma_normalize_quantiles‘ is chosen

.data |>limma::normalizeQuantiles()

If ‘preprocesscore_normalize_quantiles_use_target‘ is chosen

.data |> preprocessCore::normalize.quantiles.use.target(target = preprocessCore::normalize.quantiles.determine.target(.data))

Value

A tbl object with additional columns with scaled data as ‘<NAME OF COUNT COLUMN>_scaled‘

A tbl object with additional columns with scaled data as ‘<NAME OF COUNT COLUMN>_scaled‘

A tbl object with additional columns with scaled data as ‘<NAME OF COUNT COLUMN>_scaled‘

A tbl object with additional columns with scaled data as ‘<NAME OF COUNT COLUMN>_scaled‘

A ‘SummarizedExperiment‘ object

A ‘SummarizedExperiment‘ object

Examples

```
tidybulk::se_mini |>
  quantile_normalise_abundance()
```

---

reduce_dimensions *Dimension reduction of the transcript abundance data*

Description

reduce_dimensions() takes as input A ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment)) and calculates the reduced dimensional space of the transcript abundance.

Usage

```
reduce_dimensions(
  .data,
  .element = NULL,
  .feature = NULL,
  .abundance = NULL,
  method,
  .dims = 2,
  top = 500,
  of_samples = TRUE,
  transform = log1p,
  scale = TRUE,
  action = "add",
  ...,
  log_transform = NULL
)

## S4 method for signature 'spec_tbl_df'
reduce_dimensions(
  .data,
  .element = NULL,
  .feature = NULL,
  .abundance = NULL,
  method,
  .dims = 2,
  top = 500,
  of_samples = TRUE,
  transform = log1p,
  scale = TRUE,
  action = "add",
  ...,
  log_transform = NULL
)

## S4 method for signature 'tbl_df'
reduce_dimensions(
  .data,
  .element = NULL,
  .feature = NULL,
  .abundance = NULL,
  method,
  .dims = 2,
  top = 500,
  of_samples = TRUE,
  transform = log1p,
  scale = TRUE,
  action = "add",
  ...,
```

```
  log_transform = NULL
)

## S4 method for signature 'tidybulk'
reduce_dimensions(
  .data,
  .element = NULL,
  .feature = NULL,
  .abundance = NULL,
  method,
  .dims = 2,
  top = 500,
  of_samples = TRUE,
  transform = log1p,
  scale = TRUE,
  action = "add",
  ...,
  log_transform = NULL
)

## S4 method for signature 'SummarizedExperiment'
reduce_dimensions(
  .data,
  .element = NULL,
  .feature = NULL,
  .abundance = NULL,
  method,
  .dims = 2,
  top = 500,
  of_samples = TRUE,
  transform = log1p,
  scale = TRUE,
  action = "add",
  ...,
  log_transform = NULL
)

## S4 method for signature 'RangedSummarizedExperiment'
reduce_dimensions(
  .data,
  .element = NULL,
  .feature = NULL,
  .abundance = NULL,
  method,
  .dims = 2,
  top = 500,
  of_samples = TRUE,
  transform = log1p,
```

```
  scale = TRUE,
  action = "add",
  ...,
  log_transform = NULL
)
```

Arguments

.data	A ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment))
.element	The name of the element column (normally samples).
.feature	The name of the feature column (normally transcripts/genes)
.abundance	The name of the column including the numerical value the clustering is based on (normally transcript abundance)
method	A character string. The dimension reduction algorithm to use (PCA, MDS, tSNE).
.dims	An integer. The number of dimensions your are interested in (e.g., 4 for returning the first four principal components).
top	An integer. How many top genes to select for dimensionality reduction
of_samples	A boolean. In case the input is a tidybulk object, it indicates Whether the element column will be sample or transcript column
transform	A function that will tranform the counts, by default it is log1p for RNA sequencing data, but for avoinding tranformation you can use identity
scale	A boolean for method="PCA", this will be passed to the ‘prcomp‘ function. It is not included in the ... argument because although the default for ‘prcomp‘ if FALSE, it is advisable to set it as TRUE.
action	A character string. Whether to join the new information to the input tbl (add), or just get the non-redundant tbl with the new information (get).
...	Further parameters passed to the function prcomp if you choose method="PCA" or Rtsne if you choose method="tSNE", or uwot::tumap if you choose method="umap"
log_transform	DEPRECATED - A boolean, whether the value should be log-transformed (e.g., TRUE for RNA sequencing data)

Details

‘r lifecycle::badge("maturing")‘

This function reduces the dimensions of the transcript abundances. It can use multi-dimensional scaling (MDS; DOI.org/10.1186/gb-2010-11-3-r25), principal component analysis (PCA), or tSNE (Jesse Krijthe et al. 2018)

Underlying method for PCA: prcomp(scale = scale, ...)

Underlying method for MDS: limma::plotMDS(ndim = .dims, plot = FALSE, top = top)

Underlying method for tSNE: Rtsne::Rtsne(data, ...)

Underlying method for UMAP:

df_source = .data |>

Filter NA symbol filter(!!.feature |> is.na() |> not()) |>

Prepare data frame distinct(!!.feature,!!.element,!!.abundance) |>

Filter most variable genes keep_variable_transcripts(top) |> reduce_dimensions(method="PCA", .dims = calculate_for_pca_dimensions, action="get") |> as_matrix(rownames = quo_name(.element)) |> uwot::tumap(...)

Value

A tbl object with additional columns for the reduced dimensions

A tbl object with additional columns for the reduced dimensions

A tbl object with additional columns for the reduced dimensions

A tbl object with additional columns for the reduced dimensions

A 'SummarizedExperiment' object

A 'SummarizedExperiment' object

Examples

```
counts.MDS =
 tidybulk::se_mini |>
 identify_abundant() |>
 reduce_dimensions( method="MDS", .dims = 3)


counts.PCA =
 tidybulk::se_mini |>
 identify_abundant() |>
 reduce_dimensions(method="PCA", .dims = 3)
```

`reexports` *Objects exported from other packages*

Description

These objects are imported from other packages. Follow the links below to see their documentation.

dplyr do, select

tibble as_tibble, tibble

remove_redundancy *Drop redundant elements (e.g., samples) for which feature (e.g., transcript/gene) abundances are correlated*

Description

remove_redundancy() takes as input A ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment)) for correlation method or | <DIMENSION 1> | <DIMENSION 2> | <...> | for reduced_dimensions method, and returns a consistent object (to the input) with dropped elements (e.g., samples).

Usage

```
remove_redundancy(
  .data,
  .element = NULL,
  .feature = NULL,
  .abundance = NULL,
  method,
  of_samples = TRUE,
  correlation_threshold = 0.9,
  top = Inf,
  transform = identity,
  Dim_a_column,
  Dim_b_column,
  log_transform = NULL
)

## S4 method for signature 'spec_tbl_df'
remove_redundancy(
  .data,
  .element = NULL,
  .feature = NULL,
  .abundance = NULL,
  method,
  of_samples = TRUE,
  correlation_threshold = 0.9,
  top = Inf,
  transform = identity,
  Dim_a_column = NULL,
  Dim_b_column = NULL,
  log_transform = NULL
)

## S4 method for signature 'tbl_df'
remove_redundancy(
```

```
  .data,
  .element = NULL,
  .feature = NULL,
  .abundance = NULL,
  method,
  of_samples = TRUE,
  correlation_threshold = 0.9,
  top = Inf,
  transform = identity,
  Dim_a_column = NULL,
  Dim_b_column = NULL,
  log_transform = NULL
)

## S4 method for signature 'tidybulk'
remove_redundancy(
  .data,
  .element = NULL,
  .feature = NULL,
  .abundance = NULL,
  method,
  of_samples = TRUE,
  correlation_threshold = 0.9,
  top = Inf,
  transform = identity,
  Dim_a_column = NULL,
  Dim_b_column = NULL,
  log_transform = NULL
)

## S4 method for signature 'SummarizedExperiment'
remove_redundancy(
  .data,
  .element = NULL,
  .feature = NULL,
  .abundance = NULL,
  method,
  of_samples = TRUE,
  correlation_threshold = 0.9,
  top = Inf,
  transform = identity,
  Dim_a_column = NULL,
  Dim_b_column = NULL,
  log_transform = NULL
)

## S4 method for signature 'RangedSummarizedExperiment'
remove_redundancy(
```

```
  .data,
  .element = NULL,
  .feature = NULL,
  .abundance = NULL,
  method,
  of_samples = TRUE,
  correlation_threshold = 0.9,
  top = Inf,
  transform = identity,
  Dim_a_column = NULL,
  Dim_b_column = NULL,
  log_transform = NULL
)
```

Arguments

.data	A ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment))
.element	The name of the element column (normally samples).
.feature	The name of the feature column (normally transcripts/genes)
.abundance	The name of the column including the numerical value the clustering is based on (normally transcript abundance)
method	A character string. The method to use, correlation and reduced_dimensions are available. The latter eliminates one of the most proximar pairs of samples in PCA reduced dimensions.
of_samples	A boolean. In case the input is a tidybulk object, it indicates Whether the element column will be sample or transcript column
correlation_threshold	A real number between 0 and 1. For correlation based calculation.
top	An integer. How many top genes to select for correlation based method
transform	A function that will tranform the counts, by default it is log1p for RNA sequencing data, but for avoinding tranformation you can use identity
Dim_a_column	A character string. For reduced_dimension based calculation. The column of one principal component
Dim_b_column	A character string. For reduced_dimension based calculation. The column of another principal component
log_transform	DEPRECATED - A boolean, whether the value should be log-transformed (e.g., TRUE for RNA sequencing data)

Details

‘r lifecycle::badge("maturing")‘

This function removes redundant elements from the original data set (e.g., samples or transcripts). For example, if we want to define cell-type specific signatures with low sample redundancy. This

function returns a tibble with dropped redundant elements (e.g., samples). Two redundancy estimation approaches are supported: (i) removal of highly correlated clusters of elements (keeping a representative) with method="correlation"; (ii) removal of most proximal element pairs in a reduced dimensional space.

Underlying method for correlation: widyr::pairwise_cor(sample, transcript,count, sort = TRUE, diag = FALSE, upper = FALSE)

Underlying custom method for reduced dimensions: select_closest_pairs = function(df) couples <- df |> head(n = 0)

while (df |> nrow() > 0) pair <- df |> arrange(dist) |> head(n = 1) couples <- couples |> bind_rows(pair) df <- df |> filter(¡sample 1‘ ¡sample 2‘)

couples

Value

A tbl object with with dropped redundant elements (e.g., samples).

A tbl object with with dropped redundant elements (e.g., samples).

A tbl object with with dropped redundant elements (e.g., samples).

A tbl object with with dropped redundant elements (e.g., samples).

A ‘SummarizedExperiment‘ object

A ‘SummarizedExperiment‘ object

Examples

```
 tidybulk::se_mini |>
 identify_abundant() |>
   remove_redundancy(
    .element = sample,
    .feature = transcript,
     .abundance =  count,
     method = "correlation"
     )

counts.MDS =
 tidybulk::se_mini |>
 identify_abundant() |>
  reduce_dimensions( method="MDS", .dims = 3)

remove_redundancy(
 counts.MDS,
 Dim_a_column = `Dim1`,
 Dim_b_column = `Dim2`,
 .element = sample,
  method = "reduced_dimensions"
)
```

rename *Rename columns*

Description

Rename individual variables using ‘new_name = old_name‘ syntax.

Arguments

.data	A tbl. (See dplyr)
...	<[‘tidy-select‘][dplyr_tidy_select]> Use ‘new_name = old_name‘ to rename selected variables.

Value

An object of the same type as ‘.data‘. * Rows are not affected. * Column names are changed; column order is preserved * Data frame attributes are preserved. * Groups are updated to reflect new names.

Scoped selection and renaming

Use the three scoped variants ([rename_all()], [rename_if()], [rename_at()]) to renaming a set of variables with a function.

Methods

This function is a **generic**, which means that packages can provide implementations (methods) for other classes. See the documentation of individual methods for extra arguments and differences in behaviour.

The following methods are currently available in loaded packages:

See Also

Other single table verbs: arrange(), filter(), mutate(), summarise()

Examples

```
iris <- as_tibble(iris) # so it prints a little nicer
rename(iris, petal_length = Petal.Length)
```

`resolve_complete_confounders_of_non_interest` *Resolve Complete Confounders of Non-Interest*

Description

This function identifies and resolves complete confounders among specified factors of non-interest within a 'SummarizedExperiment' object. Complete confounders occur when the levels of one factor are entirely predictable based on the levels of another factor. Such relationships can interfere with downstream analyses by introducing redundancy or collinearity.

Usage

```
resolve_complete_confounders_of_non_interest(se, ...)
```

Arguments

`se`	A 'SummarizedExperiment' object. This object contains assay data, row data (e.g., gene annotations), and column data (e.g., sample annotations).
`...`	Factors of non-interest (column names from 'colData(se)') to examine for complete confounders.

Details

The function systematically examines pairs of specified factors and determines whether they are completely confounded. If a pair of factors is found to be confounded, one of the factors is adjusted or removed to resolve the issue. The adjusted 'SummarizedExperiment' object is returned, preserving all assays and metadata except the resolved factors.

Complete confounders of non-interest can create dependencies between variables that may bias statistical models or violate their assumptions. This function systematically addresses this by: 1. Creating new columns with the suffix "___altered" for each specified factor to preserve original values 2. Identifying pairs of factors in the specified columns that are fully confounded 3. Resolving confounding by adjusting one of the factors in the "___altered" columns

The function creates new columns with the "___altered" suffix to store the modified values while preserving the original data. This allows users to compare the original and adjusted values if needed.

The resolution strategy depends on the analysis context and can be modified in the helper function 'resolve_complete_confounders_of_non_interest_pair_SE()'. By default, the function adjusts one of the confounded factors in the "___altered" columns.

Value

A 'SummarizedExperiment' object with resolved confounders. The object retains its structure, including assays and metadata, but the column data ('colData') is updated with new "___altered" columns containing the resolved factors.

See Also

SummarizedExperiment for creating and handling 'SummarizedExperiment' objects.

Examples

```
# Load necessary libraries
library(SummarizedExperiment)
library(dplyr)

# Sample annotations
sample_annotations <- data.frame(
  sample_id = paste0("Sample", seq(1, 9)),
  factor_of_interest = c(rep("treated", 4), rep("untreated", 5)),
  A = c("a1", "a2", "a1", "a2", "a1", "a2", "a1", "a2", "a3"),
  B = c("b1", "b1", "b2", "b1", "b1", "b1", "b2", "b1", "b3"),
  C = c("c1", "c1", "c1", "c1", "c1", "c1", "c1", "c1", "c3"),
  stringsAsFactors = FALSE
)

# Simulated assay data
assay_data <- matrix(rnorm(100 * 9), nrow = 100, ncol = 9)

# Row data (e.g., gene annotations)
row_data <- data.frame(gene_id = paste0("Gene", seq_len(100)))

# Create SummarizedExperiment object
se <- SummarizedExperiment(
  assays = list(counts = assay_data),
  rowData = row_data,
  colData = DataFrame(sample_annotations)
)

# Apply the function to resolve confounders
se_resolved <- resolve_complete_confounders_of_non_interest(se, A, B, C)

# View the updated column data
colData(se_resolved)
```

---

resolve_complete_confounders_of_non_interest,SummarizedExperiment-method
resolve_complete_confounders_of_non_interest

Description

resolve_complete_confounders_of_non_interest

resolve_complete_confounders_of_non_interest

Usage

```
## S4 method for signature 'SummarizedExperiment'
resolve_complete_confounders_of_non_interest(se, ...)

## S4 method for signature 'RangedSummarizedExperiment'
resolve_complete_confounders_of_non_interest(se, ...)
```

Arguments

se	A 'SummarizedExperiment' object. This object contains assay data, row data (e.g., gene annotations), and column data (e.g., sample annotations).
...	Factors of non-interest (column names from 'colData(se)') to examine for complete confounders.

Value

A consistent object (to the input) with additional columns for the statistics from the hypothesis test (e.g., log fold change, p-value and false discovery rate).

A consistent object (to the input) with additional columns for the statistics from the hypothesis test (e.g., log fold change, p-value and false discovery rate).

---

rotate_dimensions *Rotate two dimensions (e.g., principal components) of an arbitrary angle*

Description

rotate_dimensions() takes as input a 'tbl' formatted as | <DIMENSION 1> | <DIMENSION 2> | <...> | and calculates the rotated dimensional space of the transcript abundance.

Usage

```
rotate_dimensions(
  .data,
  dimension_1_column,
  dimension_2_column,
  rotation_degrees,
  .element = NULL,
  of_samples = TRUE,
  dimension_1_column_rotated = NULL,
  dimension_2_column_rotated = NULL,
  action = "add"
)

## S4 method for signature 'spec_tbl_df'
rotate_dimensions(
```

```
  .data,
  dimension_1_column,
  dimension_2_column,
  rotation_degrees,
  .element = NULL,
  of_samples = TRUE,
  dimension_1_column_rotated = NULL,
  dimension_2_column_rotated = NULL,
  action = "add"
)

## S4 method for signature 'tbl_df'
rotate_dimensions(
  .data,
  dimension_1_column,
  dimension_2_column,
  rotation_degrees,
  .element = NULL,
  of_samples = TRUE,
  dimension_1_column_rotated = NULL,
  dimension_2_column_rotated = NULL,
  action = "add"
)

## S4 method for signature 'tidybulk'
rotate_dimensions(
  .data,
  dimension_1_column,
  dimension_2_column,
  rotation_degrees,
  .element = NULL,
  of_samples = TRUE,
  dimension_1_column_rotated = NULL,
  dimension_2_column_rotated = NULL,
  action = "add"
)

## S4 method for signature 'SummarizedExperiment'
rotate_dimensions(
  .data,
  dimension_1_column,
  dimension_2_column,
  rotation_degrees,
  .element = NULL,
  of_samples = TRUE,
  dimension_1_column_rotated = NULL,
  dimension_2_column_rotated = NULL,
  action = "add"
```

```
)

## S4 method for signature 'RangedSummarizedExperiment'
rotate_dimensions(
  .data,
  dimension_1_column,
  dimension_2_column,
  rotation_degrees,
  .element = NULL,
  of_samples = TRUE,
  dimension_1_column_rotated = NULL,
  dimension_2_column_rotated = NULL,
  action = "add"
)
```

Arguments

.data	A ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment))
dimension_1_column	A character string. The column of the dimension 1
dimension_2_column	A character string. The column of the dimension 2
rotation_degrees	A real number between 0 and 360
.element	The name of the element column (normally samples).
of_samples	A boolean. In case the input is a tidybulk object, it indicates Whether the element column will be sample or transcript column
dimension_1_column_rotated	A character string. The column of the rotated dimension 1 (optional)
dimension_2_column_rotated	A character string. The column of the rotated dimension 2 (optional)
action	A character string. Whether to join the new information to the input tbl (add), or just get the non-redundant tbl with the new information (get).

Details

‘r lifecycle::badge("maturing")‘

This function to rotate two dimensions such as the reduced dimensions.

Underlying custom method: rotation = function(m, d) // r = the angle // m data matrix r = d * pi / 180 ((bind_rows(c(‘1‘ = cos(r), ‘2‘ = -sin(r)), c(‘1‘ = sin(r), ‘2‘ = cos(r))) |> as_matrix())

Value

A tbl object with additional columns for the reduced dimensions. additional columns for the rotated dimensions. The rotated dimensions will be added to the original data set as ‘<NAME OF DIMENSION> rotated <ANGLE>‘ by default, or as specified in the input arguments.

A tbl object with additional columns for the reduced dimensions. additional columns for the rotated dimensions. The rotated dimensions will be added to the original data set as '<NAME OF DIMENSION> rotated <ANGLE>' by default, or as specified in the input arguments.

A tbl object with additional columns for the reduced dimensions. additional columns for the rotated dimensions. The rotated dimensions will be added to the original data set as '<NAME OF DIMENSION> rotated <ANGLE>' by default, or as specified in the input arguments.

A tbl object with additional columns for the reduced dimensions. additional columns for the rotated dimensions. The rotated dimensions will be added to the original data set as '<NAME OF DIMENSION> rotated <ANGLE>' by default, or as specified in the input arguments.

A 'SummarizedExperiment' object

A 'SummarizedExperiment' object

Examples

```
counts.MDS =
 tidybulk::se_mini |>
 identify_abundant() |>
 reduce_dimensions( method="MDS", .dims = 3)

counts.MDS.rotated =  rotate_dimensions(counts.MDS, `Dim1`, `Dim2`, rotation_degrees = 45, .element = sample)
```

`rowwise` *Group input by rows*

Description

See [this repository](https://github.com/jennybc/row-oriented-workflows) for alternative ways to perform row-wise operations.

Arguments

`data`	Input data frame.
`...`	Variables to be preserved when calling summarise(). This is typically a set of variables whose combination uniquely identify each row. NB: unlike group_by() you can not create new variables here but instead you can select multiple variables with (e.g.) everything().

Details

'rowwise()' is used for the results of [do()] when you create list-variables. It is also useful to support arbitrary complex operations that need to be applied to each row.

Currently, rowwise grouping only works with data frames. Its main impact is to allow you to work with list-variables in [summarise()] and [mutate()] without having to use `[[1]]`. This makes 'summarise()' on a rowwise tbl effectively equivalent to [plyr::ldply()].

Value

A consistent object (to the input)

A ‘tbl‘

Examples

```
df <- expand.grid(x = 1:3, y = 3:1)
df_done <- df |> rowwise()
```

---

scale_abundance *Scale the counts of transcripts/genes*

Description

scale_abundance() takes as input A ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment)) and Scales transcript abundance compansating for sequencing depth (e.g., with TMM algorithm, Robinson and Oshlack doi.org/10.1186/gb-2010-11-3-r25).

Usage

```
scale_abundance(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  method = "TMM",
  reference_sample = NULL,
  .subset_for_scaling = NULL,
  action = "add",
  reference_selection_function = NULL
)

## S4 method for signature 'spec_tbl_df'
scale_abundance(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  method = "TMM",
  reference_sample = NULL,
  .subset_for_scaling = NULL,
  action = "add",
  reference_selection_function = NULL
)
```

```
## S4 method for signature 'tbl_df'
scale_abundance(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  method = "TMM",
  reference_sample = NULL,
  .subset_for_scaling = NULL,
  action = "add",
  reference_selection_function = NULL
)

## S4 method for signature 'tidybulk'
scale_abundance(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  method = "TMM",
  reference_sample = NULL,
  .subset_for_scaling = NULL,
  action = "add",
  reference_selection_function = NULL
)

## S4 method for signature 'SummarizedExperiment'
scale_abundance(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  method = "TMM",
  reference_sample = NULL,
  .subset_for_scaling = NULL,
  action = NULL,
  reference_selection_function = NULL
)

## S4 method for signature 'RangedSummarizedExperiment'
scale_abundance(
  .data,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  method = "TMM",
  reference_sample = NULL,
```

```
  .subset_for_scaling = NULL,
  action = NULL,
  reference_selection_function = NULL
)
```

Arguments

.data	A ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment))
.sample	The name of the sample column
.transcript	The name of the transcript/gene column
.abundance	The name of the transcript/gene abundance column
method	A character string. The scaling method passed to the back-end function (i.e., edgeR::calcNormFactors; "TMM","TMMwsp","RLE","upperquartile")
reference_sample	A character string. The name of the reference sample. If NULL the sample with highest total read count will be selected as reference.
.subset_for_scaling	A gene-wise quosure condition. This will be used to filter rows (features/genes) of the dataset. For example
action	A character string between "add" (default) and "only". "add" joins the new information to the input tbl (default), "only" return a non-redundant tbl with the just new information.
reference_selection_function	DEPRECATED. please use reference_sample.

Details

‘r lifecycle::badge("maturing")‘

Scales transcript abundance compensating for sequencing depth (e.g., with TMM algorithm, Robinson and Oshlack doi.org/10.1186/gb-2010-11-3-r25). Lowly transcribed transcripts/genes (defined with minimum_counts and minimum_proportion parameters) are filtered out from the scaling procedure. The scaling inference is then applied back to all unfiltered data.

Underlying method edgeR::calcNormFactors(.data, method = c("TMM","TMMwsp","RLE","upperquartile"))

Value

A tbl object with additional columns with scaled data as ‘<NAME OF COUNT COLUMN>_scaled‘

A tbl object with additional columns with scaled data as ‘<NAME OF COUNT COLUMN>_scaled‘

A tbl object with additional columns with scaled data as ‘<NAME OF COUNT COLUMN>_scaled‘

A tbl object with additional columns with scaled data as ‘<NAME OF COUNT COLUMN>_scaled‘

A ‘SummarizedExperiment‘ object

A ‘SummarizedExperiment‘ object

Examples

```
tidybulk::se_mini |>
  identify_abundant() |>
  scale_abundance()
```

`se` *SummarizedExperiment*

Description

SummarizedExperiment

Usage

```
se
```

Format

An object of class `RangedSummarizedExperiment` with 100 rows and 8 columns.

`se_mini` *SummarizedExperiment mini for vignette*

Description

SummarizedExperiment mini for vignette

Usage

```
se_mini
```

Format

An object of class `SummarizedExperiment` with 527 rows and 5 columns.

summarise *Summarise each group to fewer rows*

Description

'summarise()' creates a new data frame. It will have one (or more) rows for each combination of grouping variables; if there are no grouping variables, the output will have a single row summarising all observations in the input. It will contain one column for each grouping variable and one column for each of the summary statistics that you have specified.

'summarise()' and 'summarize()' are synonyms.

Arguments

.data	A tbl. (See dplyr)
...	<['tidy-eval'][dplyr_tidy_eval]> Name-value pairs of summary functions. The name will be the name of the variable in the result. The value can be: * A vector of length 1, e.g. 'min(x)', 'n()', or 'sum(is.na(y))'. * A vector of length 'n', e.g. 'quantile()'. * A data frame, to add multiple columns from a single expression.

Value

An object _usually_ of the same type as '.data'.

* The rows come from the underlying 'group_keys()'. * The columns are a combination of the grouping keys and the summary expressions that you provide. * If 'x' is grouped by more than one variable, the output will be another [grouped_df] with the right-most group removed. * If 'x' is grouped by one variable, or is not grouped, the output will be a [tibble]. * Data frame attributes are **not** preserved, because 'summarise()' fundamentally creates a new data frame.

Useful functions

* Center: [mean()], [median()] * Spread: [sd()], [IQR()], [mad()] * Range: [min()], [max()], [quantile()] * Position: [first()], [last()], [nth()], * Count: [n()], [n_distinct()] * Logical: [any()], [all()]

Backend variations

The data frame backend supports creating a variable and using it in the same summary. This means that previously created summary variables can be further transformed or combined within the summary, as in [mutate()]. However, it also means that summary variables with the same names as previous variables overwrite them, making those variables unavailable to later summary variables.

This behaviour may not be supported in other backends. To avoid unexpected results, consider using new names for your summary variables, especially when creating multiple summaries.

Methods

This function is a **generic**, which means that packages can provide implementations (methods) for other classes. See the documentation of individual methods for extra arguments and differences in behaviour.

The following methods are currently available in loaded packages:

See Also

Other single table verbs: `arrange()`, `filter()`, `mutate()`, `rename()`

Examples

```
# A summary applied to ungrouped tbl returns a single row

mtcars |>
  summarise(mean = mean(disp))
```

`symbol_to_entrez` *Get ENTREZ id from gene SYMBOL*

Description

Get ENTREZ id from gene SYMBOL

Usage

```
symbol_to_entrez(.data, .transcript = NULL, .sample = NULL)
```

Arguments

`.data`	A tt or tbl object.
`.transcript`	A character. The name of the gene symbol column.
`.sample`	The name of the sample column

Value

A tbl

Examples

```
# This function was designed for data.frame
# Convert from SummarizedExperiment for this example. It is NOT reccomended.

tidybulk::se_mini |> tidybulk() |> as_tibble() |> symbol_to_entrez(.transcript = .feature, .sample = .sample)
```

test_differential_abundance

Perform differential transcription testing using edgeR quasi-likelihood (QLT), edgeR likelihood-ratio (LR), limma-voom, limma-voom-with-quality-weights or DESeq2

Description

test_differential_abundance() takes as input A ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment)) and returns a consistent object (to the input) with additional columns for the statistics from the hypothesis test.

Usage

```
test_differential_abundance(
  .data,
  .formula,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  contrasts = NULL,
  method = "edgeR_quasi_likelihood",
  test_above_log2_fold_change = NULL,
  scaling_method = "TMM",
  omit_contrast_in_colnames = FALSE,
  prefix = "",
  action = "add",
  ...,
  significance_threshold = NULL,
  fill_missing_values = NULL,
  .contrasts = NULL
)

## S4 method for signature 'spec_tbl_df'
test_differential_abundance(
  .data,
  .formula,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  contrasts = NULL,
  method = "edgeR_quasi_likelihood",
  test_above_log2_fold_change = NULL,
  scaling_method = "TMM",
  omit_contrast_in_colnames = FALSE,
  prefix = "",
```

```
  action = "add",
  ...,
  significance_threshold = NULL,
  fill_missing_values = NULL,
  .contrasts = NULL
)

## S4 method for signature 'tbl_df'
test_differential_abundance(
  .data,
  .formula,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  contrasts = NULL,
  method = "edgeR_quasi_likelihood",
  test_above_log2_fold_change = NULL,
  scaling_method = "TMM",
  omit_contrast_in_colnames = FALSE,
  prefix = "",
  action = "add",
  ...,
  significance_threshold = NULL,
  fill_missing_values = NULL,
  .contrasts = NULL
)

## S4 method for signature 'tidybulk'
test_differential_abundance(
  .data,
  .formula,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  contrasts = NULL,
  method = "edgeR_quasi_likelihood",
  test_above_log2_fold_change = NULL,
  scaling_method = "TMM",
  omit_contrast_in_colnames = FALSE,
  prefix = "",
  action = "add",
  ...,
  significance_threshold = NULL,
  fill_missing_values = NULL,
  .contrasts = NULL
)

## S4 method for signature 'SummarizedExperiment'
```

```
test_differential_abundance(
  .data,
  .formula,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  contrasts = NULL,
  method = "edgeR_quasi_likelihood",
  test_above_log2_fold_change = NULL,
  scaling_method = "TMM",
  omit_contrast_in_colnames = FALSE,
  prefix = "",
  action = "add",
  ...,
  significance_threshold = NULL,
  fill_missing_values = NULL,
  .contrasts = NULL
)

## S4 method for signature 'RangedSummarizedExperiment'
test_differential_abundance(
  .data,
  .formula,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  contrasts = NULL,
  method = "edgeR_quasi_likelihood",
  test_above_log2_fold_change = NULL,
  scaling_method = "TMM",
  omit_contrast_in_colnames = FALSE,
  prefix = "",
  action = "add",
  ...,
  significance_threshold = NULL,
  fill_missing_values = NULL,
  .contrasts = NULL
)
```

Arguments

.data	A ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment))
.formula	A formula representing the desired linear model. If there is more than one factor, they should be in the order factor of interest + additional factors.
.sample	The name of the sample column
.transcript	The name of the transcript/gene column

`.abundance` The name of the transcript/gene abundance column

`contrasts` This parameter takes the format of the contrast parameter of the method of choice. For edgeR and limma-voom is a character vector. For DESeq2 is a list including a character vector of length three. The first covariate is the one the model is tested against (e.g., ~ factor_of_interest)

`method` A string character. Either "edgeR_quasi_likelihood" (i.e., QLF), "edgeR_likelihood_ratio" (i.e., LRT), "edger_robust_likelihood_ratio", "DESeq2", "limma_voom", "limma_voom_sample_weights "glmmseq_lme4", "glmmseq_glmmtmb"

`test_above_log2_fold_change`
A positive real value. This works for edgeR and limma_voom methods. It uses the 'treat' function, which tests that the difference in abundance is bigger than this threshold rather than zero https://pubmed.ncbi.nlm.nih.gov/19176553.

`scaling_method` A character string. The scaling method passed to the back-end functions: edgeR and limma-voom (i.e., edgeR::calcNormFactors; "TMM","TMMwsp","RLE","upperquartile"). Setting the parameter to \"none\" will skip the compensation for sequencing-depth for the method edgeR or limma-voom.

`omit_contrast_in_colnames`
If just one contrast is specified you can choose to omit the contrast label in the colnames.

`prefix` A character string. The prefix you would like to add to the result columns. It is useful if you want to compare several methods.

`action` A character string. Whether to join the new information to the input tbl (add), or just get the non-redundant tbl with the new information (get).

`...` Further arguments passed to some of the internal experimental functions. For example for glmmSeq, it is possible to pass .dispersion, and .scaling_factor column tidyeval to skip the caluclation of dispersion and scaling and use precalculated values. This is helpful is you want to calculate those quantities on many genes and do DE testing on fewer genes. .scaling_factor is the TMM value that can be obtained with tidybulk::scale_abundance.

`significance_threshold`
DEPRECATED - A real between 0 and 1 (usually 0.05).

`fill_missing_values`
DEPRECATED - A boolean. Whether to fill missing sample/transcript values with the median of the transcript. This is rarely needed.

`.contrasts` DEPRECATED - This parameter takes the format of the contrast parameter of the method of choice. For edgeR and limma-voom is a character vector. For DESeq2 is a list including a character vector of length three. The first covariate is the one the model is tested against (e.g., ~ factor_of_interest)

Details

'r lifecycle::badge("maturing")'

This function provides the option to use edgeR https://doi.org/10.1093/bioinformatics/btp616, limma-voom https://doi.org/10.1186/gb-2014-15-2-r29, limma_voom_sample_weights

https://doi.org/10.1093/nar/gkv412 or DESeq2 https://doi.org/10.1186/s13059-014-0550-8 to perform the testing. All methods use raw counts, irrespective of if scale_abundance or adjust_abundance have been calculated, therefore it is essential to add covariates such as batch effects (if applicable) in the formula.

Underlying method for edgeR framework:

.data |>

Filter keep_abundant(factor_of_interest = !!(as.symbol(parse_formula(.formula)[1])), minimum_counts = minimum_counts, minimum_proportion = minimum_proportion) |>

Format select(!!.transcript,!!.sample,!!.abundance) |> spread(!!.sample,!!.abundance) |> as_matrix(rownames = !!.transcript)

edgeR edgeR::DGEList(counts = .) |> edgeR::calcNormFactors(method = scaling_method) |> edgeR::estimateDisp(design) |>

Fit edgeR::glmQLFit(design) |> // or glmFit according to choice edgeR::glmQLFTest(coef = 2, contrast = my_contrasts) // or glmLRT according to choice

Underlying method for DESeq2 framework:

keep_abundant(factor_of_interest = !!as.symbol(parse_formula(.formula)[[1]]), minimum_counts = minimum_counts, minimum_proportion = minimum_proportion) |>

DESeq2 DESeq2::DESeqDataSet(design = .formula) |> DESeq2::DESeq() |> DESeq2::results()

Underlying method for glmmSeq framework:

counts = .data assay(my_assay)

Create design matrix for dispersion, removing random effects design = model.matrix(object = .formula |> lme4::nobars(), data = metadata)

dispersion = counts |> edgeR::estimateDisp(design = design)

glmmSeq(.formula, countdata = counts , metadata = metadata |> as.data.frame(), dispersion = dispersion, progress = TRUE, method = method |> str_remove("(?i)^glmmSeq_"),)

Value

A consistent object (to the input) with additional columns for the statistics from the test (e.g., log fold change, p-value and false discovery rate).

A consistent object (to the input) with additional columns for the statistics from the test (e.g., log fold change, p-value and false discovery rate).

A consistent object (to the input) with additional columns for the statistics from the hypothesis test (e.g., log fold change, p-value and false discovery rate).

A consistent object (to the input) with additional columns for the statistics from the hypothesis test (e.g., log fold change, p-value and false discovery rate).

A ‘SummarizedExperiment‘ object

A ‘SummarizedExperiment‘ object

Examples

```
 # edgeR

 tidybulk::se_mini |>
 identify_abundant() |>
 test_differential_abundance( ~ condition )

 # The function `test_differential_abundance` operates with contrasts too

 tidybulk::se_mini |>
 identify_abundant(factor_of_interest = condition) |>
 test_differential_abundance(
     ~ 0 + condition,
     contrasts = c( "conditionTRUE - conditionFALSE")
 )

 # DESeq2 - equivalent for limma-voom

my_se_mini = tidybulk::se_mini
my_se_mini$condition  = factor(my_se_mini$condition)

# demontrating with `fitType` that you can access any arguments to DESeq()
my_se_mini  |>
   identify_abundant(factor_of_interest = condition) |>
       test_differential_abundance( ~ condition, method="deseq2", fitType="local")

# testing above a log2 threshold, passes along value to lfcThreshold of results()
res <- my_se_mini  |>
   identify_abundant(factor_of_interest = condition) |>
        test_differential_abundance( ~ condition, method="deseq2",
            fitType="local",
            test_above_log2_fold_change=4 )

# Use random intercept and random effect models

 se_mini[1:50,] |>
  identify_abundant(factor_of_interest = condition) |>
  test_differential_abundance(
    ~ condition + (1 + condition | time),
    method = "glmmseq_lme4", cores = 1
  )

# confirm that lfcThreshold was used
## Not run:
    res |>
        mcols() |>
        DESeq2::DESeqResults() |>
        DESeq2::plotMA()

## End(Not run)

# The function `test_differential_abundance` operates with contrasts too
```

```
my_se_mini |>
identify_abundant() |>
test_differential_abundance(
    ~ 0 + condition,
    contrasts = list(c("condition", "TRUE", "FALSE")),
    method="deseq2",
        fitType="local"
)
```

`test_differential_cellularity` *Add differential tissue composition information to a tbl*

Description

test_differential_cellularity() takes as input A ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment)) and returns a consistent object (to the input) with additional columns for the statistics from the hypothesis test.

Usage

```
test_differential_cellularity(
  .data,
  .formula,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  method = "cibersort",
  reference = X_cibersort,
  significance_threshold = 0.05,
  ...
)

## S4 method for signature 'spec_tbl_df'
test_differential_cellularity(
  .data,
  .formula,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  method = "cibersort",
  reference = X_cibersort,
  significance_threshold = 0.05,
  ...
)
```

```
## S4 method for signature 'tbl_df'
test_differential_cellularity(
  .data,
  .formula,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  method = "cibersort",
  reference = X_cibersort,
  significance_threshold = 0.05,
  ...
)

## S4 method for signature 'tidybulk'
test_differential_cellularity(
  .data,
  .formula,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  method = "cibersort",
  reference = X_cibersort,
  significance_threshold = 0.05,
  ...
)

## S4 method for signature 'SummarizedExperiment'
test_differential_cellularity(
  .data,
  .formula,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  method = "cibersort",
  reference = X_cibersort,
  significance_threshold = 0.05,
  ...
)

## S4 method for signature 'RangedSummarizedExperiment'
test_differential_cellularity(
  .data,
  .formula,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  method = "cibersort",
```

```
  reference = X_cibersort,
  significance_threshold = 0.05,
  ...
)
```

Arguments

.data	A ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment))
.formula	A formula representing the desired linear model. The formula can be of two forms: multivariable (recommended) or univariable Respectively: \"factor_of_interest ~ .\" or \". ~ factor_of_interest\". The dot represents cell-type proportions, and it is mandatory. If censored regression is desired (coxph) the formula should be of the form \"survival::Surv\(y, dead\) ~ .\"
.sample	The name of the sample column
.transcript	The name of the transcript/gene column
.abundance	The name of the transcript/gene abundance column
method	A string character. Either \"cibersort\", \"epic\" or \"llsr\". The regression method will be chosen based on being multivariable: lm or cox-regression (both on logit-transformed proportions); or univariable: beta or cox-regression (on logit-transformed proportions). See .formula for multi- or univariable choice.
reference	A data frame. The transcript/cell_type data frame of integer transcript abundance
significance_threshold	A real between 0 and 1 (usually 0.05).
...	Further parameters passed to the method deconvolve_cellularity

Details

‘r lifecycle::badge("maturing")‘

This routine applies a deconvolution method (e.g., Cibersort; DOI: 10.1038/nmeth.3337) and passes the proportions inferred into a generalised linear model (DOI:dx.doi.org/10.1007/s11749-010-0189-z) or a cox regression model (ISBN: 978-1-4757-3294-8)

Underlying method for the generalised linear model: data |> deconvolve_cellularity(!!.sample, !!.transcript, !!.abundance, method=method, reference = reference, action="get", ...) [..] betareg::betareg(.my_formula, .)

Underlying method for the cox regression: data |> deconvolve_cellularity(!!.sample, !!.transcript, !!.abundance, method=method, reference = reference, action="get", ...) [..] mutate(.proportion_0_corrected = .proportion_0_corrected |> boot::logit()) survival::coxph(.my_formula, .)

Value

A consistent object (to the input) with additional columns for the statistics from the hypothesis test (e.g., log fold change, p-value and false discovery rate).

A ‘SummarizedExperiment‘ object

A ‘SummarizedExperiment‘ object

Examples

```
# Regular regression
test_differential_cellularity(
 tidybulk::se_mini ,
    . ~ condition,
    cores = 1
)

# Cox regression - multiple

tidybulk::se_mini |>

# Test
test_differential_cellularity(
    survival::Surv(days, dead) ~ .,
    cores = 1
)
```

`test_gene_enrichment` *analyse gene enrichment with EGSEA*

Description

test_gene_enrichment() takes as input a ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment)) and returns a ‘tbl‘ of gene set information

Usage

```
test_gene_enrichment(
  .data,
  .formula,
  .sample = NULL,
  .entrez,
  .abundance = NULL,
  contrasts = NULL,
  methods = c("camera", "roast", "safe", "gage", "padog", "globaltest", "ora"),
  gene_sets = c("h", "c1", "c2", "c3", "c4", "c5", "c6", "c7", "kegg_disease",
    "kegg_metabolism", "kegg_signaling"),
  species,
  cores = 10,
  method = NULL,
  .contrasts = NULL
)
```

```
## S4 method for signature 'spec_tbl_df'
test_gene_enrichment(
  .data,
  .formula,
  .sample = NULL,
  .entrez,
  .abundance = NULL,
  contrasts = NULL,
  methods = c("camera", "roast", "safe", "gage", "padog", "globaltest", "ora"),
  gene_sets = c("h", "c1", "c2", "c3", "c4", "c5", "c6", "c7", "kegg_disease",
    "kegg_metabolism", "kegg_signaling"),
  species,
  cores = 10,
  method = NULL,
  .contrasts = NULL
)

## S4 method for signature 'tbl_df'
test_gene_enrichment(
  .data,
  .formula,
  .sample = NULL,
  .entrez,
  .abundance = NULL,
  contrasts = NULL,
  methods = c("camera", "roast", "safe", "gage", "padog", "globaltest", "ora"),
  gene_sets = c("h", "c1", "c2", "c3", "c4", "c5", "c6", "c7", "kegg_disease",
    "kegg_metabolism", "kegg_signaling"),
  species,
  cores = 10,
  method = NULL,
  .contrasts = NULL
)

## S4 method for signature 'tidybulk'
test_gene_enrichment(
  .data,
  .formula,
  .sample = NULL,
  .entrez,
  .abundance = NULL,
  contrasts = NULL,
  methods = c("camera", "roast", "safe", "gage", "padog", "globaltest", "ora"),
  gene_sets = c("h", "c1", "c2", "c3", "c4", "c5", "c6", "c7", "kegg_disease",
    "kegg_metabolism", "kegg_signaling"),
  species,
  cores = 10,
  method = NULL,
```

```
  .contrasts = NULL
)

## S4 method for signature 'SummarizedExperiment'
test_gene_enrichment(
  .data,
  .formula,
  .sample = NULL,
  .entrez,
  .abundance = NULL,
  contrasts = NULL,
  methods = c("camera", "roast", "safe", "gage", "padog", "globaltest", "ora"),
  gene_sets = c("h", "c1", "c2", "c3", "c4", "c5", "c6", "c7", "kegg_disease",
    "kegg_metabolism", "kegg_signaling"),
  species,
  cores = 10,
  method = NULL,
  .contrasts = NULL
)

## S4 method for signature 'RangedSummarizedExperiment'
test_gene_enrichment(
  .data,
  .formula,
  .sample = NULL,
  .entrez,
  .abundance = NULL,
  contrasts = NULL,
  methods = c("camera", "roast", "safe", "gage", "padog", "globaltest", "ora"),
  gene_sets = c("h", "c1", "c2", "c3", "c4", "c5", "c6", "c7", "kegg_disease",
    "kegg_metabolism", "kegg_signaling"),
  species,
  cores = 10,
  method = NULL,
  .contrasts = NULL
)
```

Arguments

.data	A ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment))
.formula	A formula with no response variable, representing the desired linear model
.sample	The name of the sample column
.entrez	The ENTREZ ID of the transcripts/genes
.abundance	The name of the transcript/gene abundance column
contrasts	This parameter takes the format of the contrast parameter of the method of choice. For edgeR and limma-voom is a character vector. For DESeq2 is a

list including a character vector of length three. The first covariate is the one the model is tested against (e.g., ~ factor_of_interest)

`methods` A character vector. One or 3 or more methods to use in the testing (currently EGSEA errors if 2 are used). Type EGSEA::egsea.base() to see the supported GSE methods.

`gene_sets` A character vector or a list. It can take one or more of the following built-in collections as a character vector: c("h", "c1", "c2", "c3", "c4", "c5", "c6", "c7", "kegg_disease", "kegg_metabolism", "kegg_signaling"), to be used with EGSEA buildIdx. c1 is human specific. Alternatively, a list of user-supplied gene sets can be provided, to be used with EGSEA buildCustomIdx. In that case, each gene set is a character vector of Entrez IDs and the names of the list are the gene set names.

`species` A character. It can be human, mouse or rat.

`cores` An integer. The number of cores available

`method` DEPRECATED. Please use methods.

`.contrasts` DEPRECATED - This parameter takes the format of the contrast parameter of the method of choice. For edgeR and limma-voom is a character vector. For DESeq2 is a list including a character vector of length three. The first covariate is the one the model is tested against (e.g., ~ factor_of_interest)

Details

‘r lifecycle::badge("maturing")‘

This wrapper executes ensemble gene enrichment analyses of the dataset using EGSEA (DOI:0.12688/f1000research.12544.1

dge = data |> keep_abundant(factor_of_interest = !!as.symbol(parse_formula(.formula)[[1]]), !!.sample, !!.entrez, !!.abundance)

Make sure transcript names are adjacent [...] as_matrix(rownames = !!.entrez) edgeR::DGEList(counts = .)

idx = buildIdx(entrezIDs = rownames(dge), species = species, msigdb.gsets = msigdb.gsets, kegg.exclude = kegg.exclude)

dge |>

Calculate weights limma::voom(design, plot = FALSE) |>

Execute EGSEA egsea(contrasts = my_contrasts, baseGSEAs = methods, gs.annots = idx, sort.by = "med.rank", num.threads = cores, report = FALSE)

Value

A consistent object (to the input)

A consistent object (to the input)

A consistent object (to the input)

A consistent object (to the input)

A consistent object (to the input)

A consistent object (to the input)

Examples

```
## Not run:

library(SummarizedExperiment)
se = tidybulk::se_mini
rowData( se)$entrez = rownames(se )
df_entrez = aggregate_duplicates(se,.transcript = entrez )

library("EGSEA")

 test_gene_enrichment(
  df_entrez,
  ~ condition,
  .sample = sample,
  .entrez = entrez,
  .abundance = count,
         methods = c("roast" , "safe", "gage"  ,  "padog" , "globaltest", "ora" ),
      gene_sets = c("h", "c1", "c2", "c3", "c4", "c5", "c6", "c7", "kegg_disease", "kegg_metabolism", "kegg_signal
  species="human",
  cores = 2
 )


## End(Not run)
```

---

test_gene_overrepresentation

analyse gene over-representation with GSEA

Description

test_gene_overrepresentation() takes as input a ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment)) and returns a ‘tbl‘ with the GSEA statistics

Usage

```
test_gene_overrepresentation(
  .data,
  .entrez,
  .do_test,
  species,
  .sample = NULL,
  gene_sets = NULL,
  gene_set = NULL
)
```

```
## S4 method for signature 'spec_tbl_df'
test_gene_overrepresentation(
  .data,
  .entrez,
  .do_test,
  species,
  .sample = NULL,
  gene_sets = NULL,
  gene_set = NULL
)

## S4 method for signature 'tbl_df'
test_gene_overrepresentation(
  .data,
  .entrez,
  .do_test,
  species,
  .sample = NULL,
  gene_sets = NULL,
  gene_set = NULL
)

## S4 method for signature 'tidybulk'
test_gene_overrepresentation(
  .data,
  .entrez,
  .do_test,
  species,
  .sample = NULL,
  gene_sets = NULL,
  gene_set = NULL
)

## S4 method for signature 'SummarizedExperiment'
test_gene_overrepresentation(
  .data,
  .entrez,
  .do_test,
  species,
  .sample = NULL,
  gene_sets = NULL,
  gene_set = NULL
)

## S4 method for signature 'RangedSummarizedExperiment'
test_gene_overrepresentation(
  .data,
  .entrez,
```

```
  .do_test,
  species,
  .sample = NULL,
  gene_sets = NULL,
  gene_set = NULL
)
```

Arguments

.data	A ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment))
.entrez	The ENTREZ ID of the transcripts/genes
.do_test	A boolean column name symbol. It indicates the transcript to check
species	A character. For example, human or mouse. MSigDB uses the latin species names (e.g., \"Mus musculus\", \"Homo sapiens\")
.sample	The name of the sample column
gene_sets	A character vector. The subset of MSigDB datasets you want to test against (e.g. \"C2\"). If NULL all gene sets are used (suggested). This argument was added to avoid time overflow of the examples.
gene_set	DEPRECATED. Use gene_sets instead.

Details

‘r lifecycle::badge("maturing")‘

This wrapper execute gene enrichment analyses of the dataset using a list of transcripts and GSEA. This wrapper uses clusterProfiler (DOI: doi.org/10.1089/omi.2011.0118) on the back-end.

Get MSigDB data msigdb_data = msigdbr::msigdbr(species = species)

Filter for specific gene collections if provided if (!is.null(gene_collections)) msigdb_data = filter(msigdb_data, gs_collection

Process the data msigdb_data |> nest(data = -gs_collection) |> mutate(test = map(data, ~ clusterProfiler::enricher(my_entrez_rank, TERM2GENE=.x |> select(gs_name, ncbi_gene), pvalueCutoff = 1) |> as_tibble()))

Value

A consistent object (to the input)

A ‘spec_tbl_df‘ object

A ‘tbl_df‘ object

A ‘tidybulk‘ object

A ‘SummarizedExperiment‘ object

A ‘RangedSummarizedExperiment‘ object

Examples

```
print("Not run for build time.")

#se_mini = aggregate_duplicates(tidybulk::se_mini, .transcript = entrez)
#df_entrez = mutate(df_entrez, do_test = feature %in% c("TNFRSF4", "PLCH2", "PADI4", "PAX7"))

## Not run:
 test_gene_overrepresentation(
  df_entrez,
  .sample = sample,
  .entrez = entrez,
  .do_test = do_test,
  species="Homo sapiens",
   gene_sets =c("C2")
 )

## End(Not run)
```

`test_gene_rank` *analyse gene rank with GSEA*

Description

test_gene_rank() takes as input a ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment)) and returns a ‘tbl‘ with the GSEA statistics

Usage

```
test_gene_rank(
  .data,
  .entrez,
  .arrange_desc,
  species,
  .sample = NULL,
  gene_sets = NULL,
  gene_set = NULL
)

## S4 method for signature 'spec_tbl_df'
test_gene_rank(
  .data,
  .entrez,
  .arrange_desc,
  species,
  .sample = NULL,
  gene_sets = c("h", "c1", "c2", "c3", "c4", "c5", "c6", "c7"),
```

```
  gene_set = NULL
)

## S4 method for signature 'tbl_df'
test_gene_rank(
  .data,
  .entrez,
  .arrange_desc,
  species,
  .sample = NULL,
  gene_sets = c("h", "c1", "c2", "c3", "c4", "c5", "c6", "c7"),
  gene_set = NULL
)

## S4 method for signature 'tidybulk'
test_gene_rank(
  .data,
  .entrez,
  .arrange_desc,
  species,
  .sample = NULL,
  gene_sets = c("h", "c1", "c2", "c3", "c4", "c5", "c6", "c7"),
  gene_set = NULL
)

## S4 method for signature 'SummarizedExperiment'
test_gene_rank(
  .data,
  .entrez,
  .arrange_desc,
  species,
  .sample = NULL,
  gene_sets = NULL,
  gene_set = NULL
)

## S4 method for signature 'RangedSummarizedExperiment'
test_gene_rank(
  .data,
  .entrez,
  .arrange_desc,
  species,
  .sample = NULL,
  gene_sets = NULL,
  gene_set = NULL
)
```

Arguments

.data	A ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment))
.entrez	The ENTREZ ID of the transcripts/genes
.arrange_desc	A column name of the column to arrange in decreasing order
species	A character. For example, human or mouse. MSigDB uses the latin species names (e.g., \"Mus musculus\", \"Homo sapiens\")
.sample	The name of the sample column
gene_sets	A character vector or a list. It can take one or more of the following built-in collections as a character vector: c("h", "c1", "c2", "c3", "c4", "c5", "c6", "c7", "kegg_disease", "kegg_metabolism", "kegg_signaling"), to be used with EGSEA buildIdx. c1 is human specific. Alternatively, a list of user-supplied gene sets can be provided, to be used with EGSEA buildCustomIdx. In that case, each gene set is a character vector of Entrez IDs and the names of the list are the gene set names.
gene_set	DEPRECATED. Use gene_sets instead.

Details

[Maturing]

This wrapper execute gene enrichment analyses of the dataset using a list of transcripts and GSEA. This wrapper uses clusterProfiler (DOI: doi.org/10.1089/omi.2011.0118) on the back-end.

Undelying method: msigdbr::msigdbr(species = species)

Filter specific gene_sets if specified. This was introduced to speed up examples executionS when(!is.null(gene_sets) ~ filter(., gs_collection ~ (.)) |>

Execute calculation nest(data = -gs_collection) |> mutate(fit = map(data, ~ clusterProfiler::GSEA(my_entrez_rank, TERM2GENE=.x |> select(gs_name, ncbi_gene), pvalueCutoff = 1)

))

Value

A consistent object (to the input)

A ‘spec_tbl_df‘ object

A ‘tbl_df‘ object

A ‘tidybulk‘ object

A ‘SummarizedExperiment‘ object

A ‘RangedSummarizedExperiment‘ object

Examples

```
print("Not run for build time.")

## Not run:

df_entrez = tidybulk::se_mini
df_entrez = mutate(df_entrez, do_test = .feature %in% c("TNFRSF4", "PLCH2", "PADI4", "PAX7"))
df_entrez  = df_entrez |> test_differential_abundance(~ condition)


test_gene_rank(
 df_entrez,
  .sample = .sample,
 .entrez = entrez,
  species="Homo sapiens",
   gene_sets =c("C2"),
 .arrange_desc = logFC
 )

## End(Not run)
```

---

test_stratification_cellularity

Test of stratification of biological replicates based on tissue composition, one cell-type at the time, using Kaplan-meier curves.

Description

test_stratification_cellularity() takes as input A ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment)) and returns a consistent object (to the input) with additional columns for the statistics from the hypothesis test.

Usage

```
test_stratification_cellularity(
  .data,
  .formula,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  method = "cibersort",
  reference = X_cibersort,
  ...
)

## S4 method for signature 'spec_tbl_df'
```

```
test_stratification_cellularity(
  .data,
  .formula,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  method = "cibersort",
  reference = X_cibersort,
  ...
)

## S4 method for signature 'tbl_df'
test_stratification_cellularity(
  .data,
  .formula,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  method = "cibersort",
  reference = X_cibersort,
  ...
)

## S4 method for signature 'tidybulk'
test_stratification_cellularity(
  .data,
  .formula,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  method = "cibersort",
  reference = X_cibersort,
  ...
)

## S4 method for signature 'SummarizedExperiment'
test_stratification_cellularity(
  .data,
  .formula,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  method = "cibersort",
  reference = X_cibersort,
  ...
)

## S4 method for signature 'RangedSummarizedExperiment'
```

```
test_stratification_cellularity(
  .data,
  .formula,
  .sample = NULL,
  .transcript = NULL,
  .abundance = NULL,
  method = "cibersort",
  reference = X_cibersort,
  ...
)
```

Arguments

.data	A 'tbl' (with at least three columns for sample, feature and transcript abundance) or 'SummarizedExperiment' (more convenient if abstracted to tibble with library(tidySummarizedExperiment))
.formula	A formula representing the desired linear model. The formula can be of two forms: multivariable (recommended) or univariable Respectively: \"factor_of_interest ~ .\" or \". ~ factor_of_interest\". The dot represents cell-type proportions, and it is mandatory. If censored regression is desired (coxph) the formula should be of the form \"survival::Surv\(y, dead\) ~ .\"
.sample	The name of the sample column
.transcript	The name of the transcript/gene column
.abundance	The name of the transcript/gene abundance column
method	A string character. Either \"cibersort\", \"epic\" or \"llsr\". The regression method will be chosen based on being multivariable: lm or cox-regression (both on logit-transformed proportions); or univariable: beta or cox-regression (on logit-transformed proportions). See .formula for multi- or univariable choice.
reference	A data frame. The transcript/cell_type data frame of integer transcript abundance
...	Further parameters passed to the method deconvolve_cellularity

Details

'r lifecycle::badge("maturing")'

This routine applies a deconvolution method (e.g., Cibersort; DOI: 10.1038/nmeth.3337) and passes the proportions inferred into a generalised linear model (DOI:dx.doi.org/10.1007/s11749-010-0189-z) or a cox regression model (ISBN: 978-1-4757-3294-8)

Underlying method for the test: data |> deconvolve_cellularity(!!.sample, !!.transcript, !!.abundance, method=method, reference = reference, action="get", ...) [..] |> mutate(.high_cellularity = .proportion > median(.proportion)) |> survival::survdiff(data = data, .my_formula)

Value

A consistent object (to the input) with additional columns for the statistics from the hypothesis test (e.g., log fold change, p-value and false discovery rate).

A consistent object (to the input) with additional columns for the statistics from the hypothesis test (e.g., log fold change, p-value and false discovery rate).

A consistent object (to the input) with additional columns for the statistics from the hypothesis test (e.g., log fold change, p-value and false discovery rate).

Examples

```
tidybulk::se_mini |>
test_stratification_cellularity(
 survival::Surv(days, dead) ~ .,
 cores = 1
)
```

---

tidybulk *Creates an annotated 'tidybulk' tibble from a 'tbl' or 'SummarizedExperiment' object*

Description

tidybulk() creates an annotated 'tidybulk' tibble from a 'tbl' (with at least three columns for sample, feature and transcript abundance) or 'SummarizedExperiment' (more convenient if abstracted to tibble with library(tidySummarizedExperiment))

Usage

```
tidybulk(.data, .sample, .transcript, .abundance, .abundance_scaled = NULL)

## S4 method for signature 'spec_tbl_df'
tidybulk(.data, .sample, .transcript, .abundance, .abundance_scaled = NULL)

## S4 method for signature 'tbl_df'
tidybulk(.data, .sample, .transcript, .abundance, .abundance_scaled = NULL)

## S4 method for signature 'SummarizedExperiment'
tidybulk(.data, .sample, .transcript, .abundance, .abundance_scaled = NULL)

## S4 method for signature 'RangedSummarizedExperiment'
tidybulk(.data, .sample, .transcript, .abundance, .abundance_scaled = NULL)
```

Arguments

.data	A 'tbl' (with at least three columns for sample, feature and transcript abundance) or 'SummarizedExperiment' (more convenient if abstracted to tibble with library(tidySummarizedExperiment))

.sample	The name of the sample column
.transcript	The name of the transcript/gene column
.abundance	The name of the transcript/gene abundance column
.abundance_scaled	The name of the transcript/gene scaled abundance column

Details

‘r lifecycle::badge("maturing")‘

This function creates a tidybulk object and is useful if you want to avoid to specify .sample, .transcript and .abundance arguments all the times. The tidybulk object have an attribute called internals where these three arguments are stored as metadata. They can be extracted as attr(<object>, "internals").

Value

A ‘tidybulk‘ object

A ‘tidybulk‘ object

A ‘tidybulk‘ object

A ‘tidybulk‘ object

A ‘tidybulk‘ object

Examples

```
tidybulk(tidybulk::se_mini)
```

---

tidybulk_SAM_BAM *Creates a ‘tt‘ object from a list of file names of BAM/SAM*

Description

tidybulk_SAM_BAM() creates a ‘tt‘ object from A ‘tbl‘ (with at least three columns for sample, feature and transcript abundance) or ‘SummarizedExperiment‘ (more convenient if abstracted to tibble with library(tidySummarizedExperiment))

Usage

```
tidybulk_SAM_BAM(file_names, genome = "hg38", ...)

## S4 method for signature 'character,character'
tidybulk_SAM_BAM(file_names, genome = "hg38", ...)
```

Arguments

file_names	A character vector
genome	A character string specifying an in-built annotation used for read summarization. It has four possible values including "mm10", "mm9", "hg38" and "hg19"
...	Further parameters passed to the function Rsubread::featureCounts

Details

‘r lifecycle::badge("maturing")‘

This function is based on FeatureCounts package (DOI: 10.1093/bioinformatics/btt656). This function creates a tidybulk object and is useful if you want to avoid to specify .sample, .transcript and .abundance arguments all the times. The tidybulk object have an attribute called internals where these three arguments are stored as metadata. They can be extracted as attr(<object>, "internals").

Underlying core function Rsubread::featureCounts(annot.inbuilt = genome,nthreads = n_cores, ...)

Value

A ‘tidybulk‘ object

A ‘tidybulk‘ object

---

tximeta_summarizeToGene_object

Needed for tests tximeta_summarizeToGene_object, It is SummarizedExperiment from tximeta

---

Description

Needed for tests tximeta_summarizeToGene_object, It is SummarizedExperiment from tximeta

Usage

```
tximeta_summarizeToGene_object
```

Format

An object of class `RangedSummarizedExperiment` with 10 rows and 1 columns.

unnest *unnest*

Description

unnest

nest

Arguments

data	A tbl. (See tidyr)
cols	<[‘tidy-select‘][tidyr_tidy_select]> Columns to unnest. If you ‘unnest()‘ multiple columns, parallel entries must be of compatibble sizes, i.e. they’re either equal or length 1 (following the standard tidyverse recycling rules).
names_sep	If ‘NULL‘, the default, the names will be left as is. In ‘nest()‘, inner names will come from the former outer names; in ‘unnest()‘, the new outer names will come from the inner names. If a string, the inner and outer names will be used together. In ‘nest()‘, the names of the new outer columns will be formed by pasting together the outer and the inner column names, separated by ‘names_sep‘. In ‘unnest()‘, the new inner names will have the outer names (+ ‘names_sep‘) automatically stripped. This makes ‘names_sep‘ roughly symmetric between nesting and unnesting.
keep_empty	See tidyr::unnest
names_repair	See tidyr::unnest
ptype	See tidyr::unnest
.drop	See tidyr::unnest
.id	tidyr::unnest
.sep	tidyr::unnest
.preserve	See tidyr::unnest
.data	A tbl. (See tidyr)
...	Name-variable pairs of the form new_col = c(col1, col2, col3) (See tidyr)

Value

A tidySummarizedExperiment objector a tibble depending on input

A tt object

Examples

```
tidybulk::se_mini |> tidybulk() |> nest( data = -.feature) |> unnest(data)
```

```
tidybulk::se_mini %>% tidybulk() %>% nest( data = -.feature)
```

vignette_manuscript_signature_boxplot

Needed for vignette vignette_manuscript_signature_boxplot

Description

Needed for vignette vignette_manuscript_signature_boxplot

Usage

```
vignette_manuscript_signature_boxplot
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 899 rows and 12 columns.

vignette_manuscript_signature_tsne

Needed for vignette vignette_manuscript_signature_tsne

Description

Needed for vignette vignette_manuscript_signature_tsne

Usage

```
vignette_manuscript_signature_tsne
```

Format

An object of class `spec_tbl_df` (inherits from `tbl_df`, `tbl`, `data.frame`) with 283 rows and 10 columns.

vignette_manuscript_signature_tsne2 *Needed for vignette vignette_manuscript_signature_tsne2*

Description

Needed for vignette vignette_manuscript_signature_tsne2

Usage

```
vignette_manuscript_signature_tsne2
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 283 rows and 9 columns.

X_cibersort *Cibersort reference*

Description

Cibersort reference

Usage

```
X_cibersort
```

Format

An object of class `data.frame` with 547 rows and 22 columns.

%>% *Pipe operator*

Description

See `magrittr::%>%` for details.

Usage

```
lhs %>% rhs
```

Arguments

lhs	A value or the magrittr placeholder.
rhs	A function call using the magrittr semantics.

Value

The result of calling ‘rhs(lhs)‘.

Index