Package 'ISAnalytics'

April 8, 2021

Title Analyze gene therapy vector insertion sites data identified from genomics next generation sequencing reads for clonal tracking studies

Version 1.0.11

Date 2020-07-03

Description In gene therapy, stem cells are modified using viral vectors to deliver the therapeutic transgene and replace functional properties since the genetic modification is stable and inherited in all cell progeny. The retrieval and mapping of the sequences flanking the virus-host DNA junctions allows the identification of insertion sites (IS), essential for monitoring the evolution of genetically modified cells in vivo. A comprehensive toolkit for the analysis of IS is required to foster clonal trackign studies and supporting the assessment of safety and long term efficacy in vivo. This package is aimed at (1) supporting automation of IS workflow, (2) performing base and advance analysis for IS tracking (clonal abundance, clonal expansions and statistics for insertional mutagenesis, etc.), (3) providing basic biology insights of transduced stem cells in vivo.

License CC BY 4.0

URL https://calabrialab.github.io/ISAnalytics, https://github.com//calabrialab/isanalytics

BugReports https://github.com/calabrialab/ISAnalytics/issues

biocViews BiomedicalInformatics, Sequencing, SingleCell

Depends R (>= 4.0),
magrittr

Imports utils,
reactable,
htmltools,
dplyr,
readr,
tidyr,
purrr,
rlang,
forcats,
tibble,
BiocParallel,
stringr,
fs,
zip,
lubridate,
lifecycle,

ggplot2,
ggrepel,
stats,
upsetjs,
psych,
grDevices,
data.table,
readxl,
tools

Encoding UTF-8

LazyData false

Roxygen list(markdown = TRUE)

RoxygenNote 7.1.1

Suggests testthat,
covr,
knitr,
BiocStyle,
knitcitations,
sessioninfo,
rmarkdown,
roxygen2,
vegan,
withr

VignetteBuilder knitr

RdMacros lifecycle

Config/testthat/edition 3

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

git_branch RELEASE_3_12

git_last_commit a1014e7

git_last_commit_date 2021-04-08

Date/Publication 2021-04-08

R topics documented:

`aggregate_metadata` *Performs aggregation on metadata contained in the association file.*

Description

[Experimental] Groups metadata by grouping_keys and returns a summary of info for each group.
For more details on how to use this function: `vignette("Working with aggregate functions",package = "ISAnalytics")`

Usage

```
aggregate_metadata(
  association_file,
  grouping_keys = c("SubjectID", "CellMarker", "Tissue", "TimePoint"),
  import_stats = TRUE
)
```

Arguments

`association_file`	The imported association file (via `import_association_file`)
`grouping_keys`	A character vector of column names to form a group
`import_stats`	Should Vispa2 stats files be imported and included?

Value

A tibble

See Also

Other Aggregate functions: aggregate_values_by_key()

Examples

```
op <- options("ISAnalytics.widgets" = FALSE, "ISAnalytics.verbose" = FALSE)
path_AF <- system.file("extdata", "ex_association_file.tsv",
    package = "ISAnalytics"
)
root_correct <- system.file("extdata", "fs.zip", package = "ISAnalytics")
root_correct <- unzip_file_system(root_correct, "fs")
association_file <- import_association_file(path_AF, root_correct,
    dates_format = "dmy"
)
aggregated_meta <- aggregate_metadata(association_file, import_stats = FALSE)
options(op)
```

---

aggregate_values_by_key

Aggregates matrices values based on specified key.

Description

[Experimental] Performs aggregation on values contained in the integration matrices based on the key and the specified lambda. For more details on how to use this function: `vignette("Working with aggregate functions",package = "ISAnalytics")`

Usage

```
aggregate_values_by_key(
  x,
  association_file,
  value_cols = "Value",
  key = c("SubjectID", "CellMarker", "Tissue", "TimePoint"),
  lambda = list(sum = ~sum(.x, na.rm = TRUE)),
  group = c(mandatory_IS_vars(), annotation_IS_vars())
)
```

Arguments

x	A single integration matrix (tibble) or a list of imported integration matrices (tibble)
association_file	The imported association file
value_cols	A character vector containing the names of the columns to apply the given lambdas. Must be numeric or integer columns.
key	A string or a character vector with column names of the association file to take as key
lambda	A named list of functions or purrr-style lambdas. See details section.
group	Other variables to include in the grouping besides key, can be set to NULL

Details

Setting the lambda parameter:

The lambda parameter should always contain a named list of either functions or purrr-style lambdas. It is also possible to specify the namespace of the function in both ways, for example:

```
lambda = list(sum = sum, desc = psych::describe)
```

Using purrr-style lambdas allows to specify arguments for the functions, keeping in mind that the first parameter should always be `.x`:

```
lambda = list(sum = ~sum(.x, na.rm = TRUE))
```

It is also possible to use custom user-defined functions, keeping in mind that the symbol will be evaluated in the calling environment, for example if the function is called in the global environment and lambda contains "foo" as a function, "foo" will be evaluated in the global environment.

```
foo <- function(x) {
  sum(x)
}

lambda = list(sum = ~sum(.x, na.rm = TRUE), foo = foo)

# Or with lambda notation
lambda = list(sum = ~sum(.x, na.rm = TRUE), foo = ~foo(.x))
```

Constraints on aggregation functions:

Functions passed in the lambda parameters must respect a few constraints to properly work and it's the user responsibility to ensure this.

- Functions have to accept as input a numeric or integer vector
- Function should return a single value or a list/data frame: if a list or a data frame is returned as a result, all the columns will be added to the final data frame.

Value

A list of tibbles or a single tibble according to input

See Also

Other Aggregate functions: `aggregate_metadata()`

Examples

```
op <- options("ISAnalytics.widgets" = FALSE, "ISAnalytics.verbose" = FALSE)
path_AF <- system.file("extdata", "ex_association_file.tsv",
    package = "ISAnalytics"
)
root_correct <- system.file("extdata", "fs.zip", package = "ISAnalytics")
root_correct <- unzip_file_system(root_correct, "fs")
association_file <- import_association_file(path_AF, root_correct,
    dates_format = "dmy"
)
matrices <- import_parallel_Vispa2Matrices_auto(
    association_file = association_file, root = NULL,
    quantification_type = c("fragmentEstimate", "seqCount"),
    matrix_type = "annotated", workers = 2, matching_opt = "ANY"
)
```

```
agg <- aggregate_values_by_key(
    x = matrices,
    association_file = association_file,
    value_cols = c("fragmentEstimate", "seqCount")
)
options(op)
```

`annotation_IS_vars` *Names of the annotation variables for an integration matrix.*

Description

Contains the names of the columns that are present if the integration matrix is annotated.

Usage

```
annotation_IS_vars()
```

Value

A character vector

Examples

```
annotation_IS_vars()
```

`association_file_columns` *Names of the columns in the association file.*

Description

All the names of the columns present in the association file.

Usage

```
association_file_columns()
```

Value

A character vector

Examples

```
association_file_columns()
```

as_sparse_matrix *Converts tidy integration matrices in the original sparse matrix form.*

Description

[Maturing] This function is particularly useful when a sparce matrix structure is needed by a specific function (mainly from other packages).

Usage

```
as_sparse_matrix(
  x,
  fragmentEstimate = "fragmentEstimate",
  seqCount = "seqCount",
  barcodeCount = "barcodeCount",
  cellCount = "cellCount",
  ShsCount = "ShsCount"
)
```

Arguments

x	A single tidy integration matrix or a list of integration matrices. Supports also multi-quantification matrices obtained via comparison_matrix
fragmentEstimate	For multi-quantification matrix support: the name of the fragment estimate values column
seqCount	For multi-quantification matrix support: the name of the sequence count values column
barcodeCount	For multi-quantification matrix support: the name of the barcode count values column
cellCount	For multi-quantification matrix support: the name of the cell count values column
ShsCount	For multi-quantification matrix support: the name of the Shs Count values column

Value

Depending on input, 2 possible outputs:

- A single sparce matrix (tibble) if input is a single quantification matrix
- A list of sparce matrices divided by quantification if input is a single multi-quantification matrix or a list of matrices

See Also

Other Utility functions: generate_Vispa2_launch_AF(), generate_blank_association_file(), unzip_file_system()

Examples

```
path <- system.file("extdata", "ex_annotated_ISMatrix.tsv.xz",
    package = "ISAnalytics"
)
matrix <- import_single_Vispa2Matrix(path)
sparse <- as_sparse_matrix(matrix)
```

`CIS_grubbs` *Grubbs test for Common Insertion Sites (CIS).*

Description

[Experimental] Statistical approach for the validation of common insertion sites significance based on the comparison of the integration frequency at the CIS gene with respect to other genes contained in the surrounding genomic regions. For more details please refer to this paper: https://ashpublications.org/blood/article/117/20/5332/21206/Lentiviral-vector-common-integration-sites-

Usage

```
CIS_grubbs(
  x,
  genomic_annotation_file = system.file("extdata", "hg19.refGene.oracle.tsv.xz",
    package = "ISAnalytics"),
  grubbs_flanking_gene_bp = 1e+05,
  threshold_alpha = 0.05,
  add_standard_padjust = TRUE
)
```

Arguments

`x`	An integration matrix, must include the `mandatory_IS_vars()` columns and the `annotation_IS_vars()` columns
`genomic_annotation_file`	Database file for gene annotation, see details
`grubbs_flanking_gene_bp`	Number of base pairs flanking a gene
`threshold_alpha`	Significance threshold
`add_standard_padjust`	Compute the standard padjust?

Details

Genomic annotation file:

This file is a data base, or more simply a .tsv file to import, with genes annotation for the specific genome. The annotations for the human genome (hg19) is already included in this package. If for any reason the user is performing an analysis on another genome, this file needs to be changed respecting the USCS Genome Browser format, meaning the input file headers should be:

```
## name2, chrom, strand
##  min_txStart, max_txEnd, minmax_TxLen
##  average_TxLen, name, min_cdsStart
##  max_cdsEnd, minmax_CdsLen, average_CdsLen
```

Value

A data frame

See Also

Other Analysis functions: comparison_matrix(), compute_abundance(), cumulative_count_union(), sample_statistics(), separate_quant_matrices(), threshold_filter(), top_integrations()

Examples

```
op <- options(ISAnalytics.widgets = FALSE)

path_AF <- system.file("extdata", "ex_association_file.tsv",
    package = "ISAnalytics"
)
root_correct <- system.file("extdata", "fs.zip",
    package = "ISAnalytics"
)
root_correct <- unzip_file_system(root_correct, "fs")

matrices <- import_parallel_Vispa2Matrices_auto(
    association_file = path_AF, root = root_correct,
    quantification_type = c("seqCount", "fragmentEstimate"),
    matrix_type = "annotated", workers = 2, patterns = NULL,
    matching_opt = "ANY",
    dates_format = "dmy"
)

cis <- CIS_grubbs(matrices)

options(op)
```

CIS_volcano_plot *Trace volcano plot for computed CIS data.*

Description

[Experimental] Traces a volcano plot for IS frequency and CIS results.

Usage

```
CIS_volcano_plot(
  x,
 onco_db_file = system.file("extdata", "201806_uniprot-Proto-oncogene.tsv.xz", package
    = "ISAnalytics"),
  tumor_suppressors_db_file = system.file("extdata",
    "201806_uniprot-Tumor-suppressor.tsv.xz", package = "ISAnalytics"),
  species = "human",
  known_onco = known_clinical_oncogenes(),
  suspicious_genes = clinical_relevant_suspicious_genes(),
  significance_threshold = 0.05,
  annotation_threshold_ontots = 0.1,
  highlight_genes = NULL,
```

```
  title_prefix = NULL,
  return_df = FALSE
)
```

Arguments

x	Either a simple integration matrix or a data frame resulting from the call to CIS_grubbs with add_standard_padjust = TRUE
onco_db_file	Uniprot file for proto-oncogenes (see details)
tumor_suppressors_db_file	Uniprot file for tumor-suppressor genes
species	One between "human", "mouse" and "all"
known_onco	Data frame with known oncogenes. See details.
suspicious_genes	Data frame with clinical relevant suspicious genes. See details.
significance_threshold	The significance threshold
annotation_threshold_ontots	Value above which genes are annotated
highlight_genes	Either NULL or a character vector of genes to be highlighted in the plot even if they're not above the threshold
title_prefix	A string to be displayed in the title - usually the project name and other characterizing info
return_df	Return the data frame used to generate the plot? This can be useful if the user wants to manually modify the plot with ggplot2. If TRUE the function returns a list containing both the plot and the data frame.

Details

Input data frame:

Users can supply as x either a simple integration matrix or a data frame resulting from the call to CIS_grubbs with add_standard_padjust = TRUE. In the first case an internal call to the function CIS_grubbs is performed.

Oncogene and tumor suppressor genes files:

These files are included in the package for user convenience and are simply UniProt files with gene annotations for human and mouse. For more details on how this files were generated use the help ?filename function.

Known oncogenes:

The default values are contained in a data frame exported by this package, it can be accessed by doing:

```
head(known_clinical_oncogenes())
```

```
## # A tibble: 5 x 2
##   GeneName KnownClonalExpansion
##   <chr>    <lgl>
## 1 MECOM    TRUE
## 2 CCND2    TRUE
```

```
## 3 TAL1     TRUE
## 4 LMO2     TRUE
## 5 HMGA2    TRUE
```

If the user wants to change this parameter the input data frame must preserve the column structure. The same goes for the `suspicious_genes` parameter (DOIReference column is optional):

```
head(clinical_relevant_suspicious_genes())
```

```
## # A tibble: 6 x 3
##   GeneName ClinicalRelevance DOIReference
##   <chr>    <lgl>             <chr>
## 1 DNMT3A   TRUE              https://doi.org/10.1182/blood-2018-01-829937
## 2 TET2     TRUE              https://doi.org/10.1182/blood-2018-01-829937
## 3 ASXL1    TRUE              https://doi.org/10.1182/blood-2018-01-829937
## 4 JAK2     TRUE              https://doi.org/10.1182/blood-2018-01-829937
## 5 CBL      TRUE              https://doi.org/10.1182/blood-2018-01-829937
## 6 TP53     TRUE              https://doi.org/10.1182/blood-2018-01-829937
```

Value

A plot or a list containing a plot and a data frame

Examples

```
op <- options(ISAnalytics.widgets = FALSE)

path_AF <- system.file("extdata", "ex_association_file.tsv",
    package = "ISAnalytics"
)
root_correct <- system.file("extdata", "fs.zip",
    package = "ISAnalytics"
)
root_correct <- unzip_file_system(root_correct, "fs")

matrices <- import_parallel_Vispa2Matrices_auto(
    association_file = path_AF, root = root_correct,
    quantification_type = c("seqCount", "fragmentEstimate"),
    matrix_type = "annotated", workers = 2, patterns = NULL,
    matching_opt = "ANY",
    dates_format = "dmy"
)

cis <- CIS_grubbs(matrices)
plot <- CIS_volcano_plot(cis)
options(op)
```

---

clinical_relevant_suspicious_genes

Clinical relevant suspicious genes (for mouse and human).

Description

Clinical relevant suspicious genes (for mouse and human).

Usage

```
clinical_relevant_suspicious_genes()
```

Value

A data frame

See Also

Other Plotting function helpers: known_clinical_oncogenes()

Examples

```
clinical_relevant_suspicious_genes()
```

---

comparison_matrix *obtain a single integration matrix from individual quantification matrices.*

Description

[Maturing] Takes a list of integration matrices referring to different qunatification types and merges them in a single data frame that has multiple value columns, each renamed according to their quantification type of reference.

Usage

```
comparison_matrix(
  x,
  fragmentEstimate = "fragmentEstimate",
  seqCount = "seqCount",
  barcodeCount = "barcodeCount",
  cellCount = "cellCount",
  ShsCount = "ShsCount"
)
```

Arguments

x	A named list of integration matrices, ideally obtained via import_parallel_Vispa2Matrices_interactive or import_parallel_Vispa2Matrices_auto. Names must be quantification types.
fragmentEstimate	The name of the output column for fragment estimate values
seqCount	The name of the output column for sequence count values
barcodeCount	The name of the output column for barcode count values
cellCount	The name of the output column for cell count values
ShsCount	The name of the output column for Shs count values

Value

A tibble

See Also

quantification_types

Other Analysis functions: CIS_grubbs(), compute_abundance(), cumulative_count_union(), sample_statistics(), separate_quant_matrices(), threshold_filter(), top_integrations()

Examples

```
op <- options("ISAnalytics.widgets" = FALSE)
path <- system.file("extdata", "ex_association_file.tsv",
    package = "ISAnalytics"
)
root_pth <- system.file("extdata", "fs.zip", package = "ISAnalytics")
root <- unzip_file_system(root_pth, "fs")
matrices <- import_parallel_Vispa2Matrices_auto(
   association_file = path, root = root,
   quantification_type =  c("fragmentEstimate", "seqCount"),
   matrix_type =  "annotated", workers = 2, patterns =  NULL,
   matching_opt =  "ANY",
   dates_format = "dmy", multi_quant_matrix = FALSE
)
total_matrix <- comparison_matrix(matrices)
options(op)
```

---

compute_abundance *Computes the abundance for every integration event in the input data frame.*

Description

[Maturing] Abundance is obtained for every integration event by calculating the ratio between the single value and the total value for the given group.

Usage

```
compute_abundance(
  x,
  columns = "Value",
  percentage = TRUE,
  key = "CompleteAmplificationID",
  keep_totals = FALSE
)
```

Arguments

x	An integration matrix - aka a data frame that includes the mandatory_IS_vars() as columns. The matrix can either be aggregated (via aggregate_values_by_key()) or not.
columns	A character vector of column names to process, must be numeric or integer columns
percentage	Add abundance as percentage?
key	The key to group by when calculating totals

keep_totals — A value between TRUE, FALSE or df. If TRUE, the intermediate totals for each group will be kept in the output data frame as a dedicated column with a trailing "_tot". If FALSE, totals won't be included in the output data frame. If df, the totals are returned to the user as a separate data frame, together with the abundance data frame.

Details

Abundance will be computed upon the user selected columns in the columns parameter. For each column a corresponding relative abundance column (and optionally a percentage abundance column) will be produced.

Value

Either a single data frame with computed abundance values or a list of 2 data frames (abundance_df, quant_totals)

See Also

Other Analysis functions: CIS_grubbs(), comparison_matrix(), cumulative_count_union(), sample_statistics(), separate_quant_matrices(), threshold_filter(), top_integrations()

Examples

```
path <- system.file("extdata", "ex_annotated_ISMatrix.tsv.xz",
    package = "ISAnalytics"
)
matrix <- import_single_Vispa2Matrix(path)

# Simple integration matrix - grouping by CompleteAmplificationID
abundance1 <- compute_abundance(matrix)
abundance1

# Keeping totals as a separate data frame
abundance2 <- compute_abundance(matrix, keep_totals = "df")
abundance2
```

---

compute_near_integrations

Scans input matrix to find and merge near integration sites.

Description

[Experimental] This function scans the input integration matrix to detect eventual integration sites that are too "near" to each other and merges them into single integration sites adjusting their values if needed.

Usage

```
compute_near_integrations(
  x,
  threshold = 4,
  keep_criteria = "max_value",
  strand_specific = TRUE,
  max_value_column = "seqCount",
  map_as_widget = TRUE,
  map_as_file = TRUE,
  file_path = ".",
  export_widget_path = NULL
)
```

Arguments

x	A single integration matrix, either with a single "Value" column or multiple value columns corresponding to different quantification types (obtained via comparison_matrix)
threshold	A single integer that represents an absolute number of bases for which two integrations are considered distinct
keep_criteria	While scanning, which integration should be kept? The 2 possible choices for this parameter are: • "max_value": keep the integration site which has the highest value (and collapse other values on that integration). • "keep_first": keeps the first integration
strand_specific	Should strand be considered? If yes, for example these two integration sites c(chr = "1", strand = "+", integration_locus = 14568) and c(chr = "1", strand = "-", integration_locus = 14568) are considered different and not grouped together.
max_value_column	The column that has to be considered for searching the maximum value
map_as_widget	Produce recalibration map as an HTML widget?
map_as_file	Produce recalibration map as a .tsv file?
file_path	String representing the path were the file will be saved. By default the function produces a folder in the current working directory and generates file names with time stamps.
export_widget_path	A path on disk to save produced widgets or NULL if the user doesn't wish to save the html file

Details

The whole matrix is scanned with a sliding window mechanism: for each row in the integration matrix an interval is calculated based on the threshold value, then a "look ahead" operation is performed to detect subsequent rows which integration locuses fall in the interval. If CompleteAmplificationIDs of the near integrations are different only the locus value (and optionally GeneName and GeneStrand if the matrix is annotated) is modified, otherwise rows with the same id are aggregated and values are summed. If one of the map parameters is set to true the function will also produce a re-calibration map: this data frame contains the reference of pre-recalibration values for chr, strand and integration locus and the value to which that integration was changed to after.

Value

An integration matrix with same or less number of rows

Note

We do recommend to use this function in combination with comparison_matrix to automatically perform re-calibration on all quantification matrices.

Examples

```
path <- system.file("extdata", "ex_annotated_ISMatrix.tsv.xz",
    package = "ISAnalytics"
)
matrix <- import_single_Vispa2Matrix(path)
near <- compute_near_integrations(matrix,
    map_as_widget = FALSE,
    map_as_file = FALSE
)
```

---

cumulative_count_union

Integrations cumulative count in time by sample

Description

[Experimental] This function computes the cumulative number of integrations observed in each sample at different time points by assuming that if an integration is observed at time point "t" then it is also observed in time point "t+1".

Usage

```
cumulative_count_union(
  x,
  association_file = NULL,
  timepoint_column = "TimePoint",
  key = c("SubjectID", "CellMarker", "Tissue", "TimePoint"),
  include_tp_zero = FALSE,
  zero = "0000",
  aggregate = FALSE,
  ...
)
```

Arguments

x	A simple integration matrix or an aggregated matrix (see details)
association_file	NULL or the association file for x if aggregate is set to TRUE
timepoint_column	What is the name of the time point column?
key	The aggregation key - must always contain the timepoint_column

include_tp_zero
: Include timepoint 0?

zero
: How is 0 coded in the data frame?

aggregate
: Should x be aggregated?

...
: Additional parameters to pass to aggregate_values_by_key

Details

Input data frame:

The user can provide as input for the x parameter both a simple integration matrix AND setting the aggregate parameter to TRUE, or provide an already aggregated matrix via aggregate_values_by_key. If the user supplies a matrix to be aggregated the association_file parameter must not be NULL: aggregation will be done by an internal call to the aggregation function. If the user supplies an already aggregated matrix, the key parameter is the key used for aggregation - **NOTE: for this operation is mandatory that the time point column is included in the key.**

Assumptions on time point format:

By using the functions provided by this package, when imported, an association file will be correctly formatted for future usage. In the formatting process there is also a padding operation performed on time points: this means the functions expects the time point column to be of type character and to be correctly padded with 0s. If the chosen column for time point is detected as numeric the function will attempt the conversion to character and automatic padding. If you choose to import the association file not using the import_association_file function, be sure to check the format of the chosen column to avoid undesired results.

Value

A data frame

See Also

Other Analysis functions: CIS_grubbs(), comparison_matrix(), compute_abundance(), sample_statistics(), separate_quant_matrices(), threshold_filter(), top_integrations()

Examples

```
op <- options(ISAnalytics.widgets = FALSE)

path_AF <- system.file("extdata", "ex_association_file.tsv",
    package = "ISAnalytics"
)
root_correct <- system.file("extdata", "fs.zip",
    package = "ISAnalytics"
)
root_correct <- unzip_file_system(root_correct, "fs")

association_file <- import_association_file(path_AF, root_correct,
    dates_format = "dmy"
)
matrices <- import_parallel_Vispa2Matrices_auto(
    association_file = association_file, root = NULL,
    quantification_type = c("seqCount", "fragmentEstimate"),
    matrix_type = "annotated", workers = 2, patterns = NULL,
    matching_opt = "ANY", multi_quant_matrix = FALSE
```

```
)

#### EXTERNAL AGGREGATION
aggregated <- aggregate_values_by_key(matrices$seqCount, association_file)
cumulative_count <- cumulative_count_union(aggregated)

#### INTERNAL AGGREGATION
cumulative_count_2 <- cumulative_count_union(matrices$seqCount,
    association_file,
    aggregate = TRUE
)

options(op)
```

`date_columns_coll` *Possible choices for* `date_col` *parameter.*

Description

Possible choices for `date_col` parameter.

Usage

```
date_columns_coll()
```

Value

A character vector of column names

See Also

`remove_collisions`

Examples

```
dates <- date_columns_coll()
```

`date_formats` *Possible choices for the* `dates_format` *parameter in* `import_association_file`, `import_parallel_vispa2Matrices_interactive` *and* `import_parallel_vispa2Matrices_auto`.

Description

All options correspond to `lubridate` functions:

- ymd: year, month, date
- ydm: year, day, month
- mdy: month, day, year
- myd: month, year, day
- dmy: day, month, year
- dym: day, year, month
- yq: year quantile

Usage

```
date_formats()
```

Details

NOTE: use the same date format across the association file.

Value

A character vector

See Also

`import_association_file`, `import_parallel_Vispa2Matrices_auto`

Examples

```
date_formats()
```

`default_stats` *A set of pre-defined functions for* `sample_statistics`.

Description

A set of pre-defined functions for `sample_statistics`.

Usage

```
default_stats()
```

Value

A named list of functions/purrr-style lambdas

Examples

```
default_stats()
```

generate_blank_association_file

Creates a blank association file.

Description

This function is useful if you want a blank association file to start using both Vispa2 and this package or simply if you want a correct framework to fix a malformed association file you have already.

Usage

```
generate_blank_association_file(path)
```

Arguments

path	The path on disk where the file should be written

Value

returns NULL

See Also

Other Utility functions: as_sparse_matrix(), generate_Vispa2_launch_AF(), unzip_file_system()

Examples

```
temp <- tempfile()
generate_blank_association_file(temp)
```

generate_Vispa2_launch_AF

Creates a reduced association file for Vispa2 run, given project and pool

Description

The function selects the appropriate columns and prepares a file for the launch of Vispa2 pipeline for each project/pool pair specified.

Usage

```
generate_Vispa2_launch_AF(association_file, project, pool, path)
```

Arguments

association_file	The imported association file (via import_association_file)
project	A vector of characters containing project names
pool	A vector of characters containing pool names. **NOTE: the names should refer to the values contained in the PoolID column of the association file and NOT the concatenatePoolIDSeqRun column!**
path	A single string representing the path to the folder where files should be written. If the folder doesn't exist it will be created.

Details

Note: the function is vectorized, meaning you can specify more than one project and more than one pool as vectors of characters, but you must ensure that:

- Both project and pool vectors have the same length
- You correclty type names in corresponding positions, for example c("CLOEXP", "PROJECT1100", "PROJECT1100") - c("POOL6", "ABX-LR-PL5-POOL14-1", "ABX-LR-PL6-POOL15-1"). If you type a pool in the position of a corresponding project that doesn't match no file will be produced since that pool doesn't exist in the corresponding project.

Value

returns NULL

See Also

Other Utility functions: as_sparse_matrix(), generate_blank_association_file(), unzip_file_system()

Examples

```
op <- options("ISAnalytics.widgets" = FALSE)
temp <- tempdir()
path_af <- system.file("extdata", "ex_association_file.tsv",
    package = "ISAnalytics"
)
root_pth <- system.file("extdata", "fs.zip", package = "ISAnalytics")
root <- unzip_file_system(root_pth, "fs")
association_file <- import_association_file(path_af, root,
    dates_format = "dmy"
)
generate_Vispa2_launch_AF(association_file, "CLOEXP", "POOL6", temp)
options(op)
```

`import_association_file` *Import the association file from disk*

Description

[Maturing] Imports the association file and immediately performs a check on the file system starting from the root to assess the alignment between the two.

Usage

```
import_association_file(
  path,
  root = NULL,
  tp_padding = 4,
  dates_format = "ymd",
  separator = "\t",
  filter_for = NULL,
  export_widget_path = NULL
)
```

Arguments

`path`	The path on disk to the association file.
`root`	The path on disk of the root folder of Vispa2 output or NULL. See details.
`tp_padding`	Timepoint padding, indicates the number of digits of the "Timepoint" column once imported. Fills the content with 0s up to the length specified (ex: 1 becomes 0001 with a tp_padding of 4)
`dates_format`	A single string indicating how dates should be parsed. Must be a value in: `date_formats()`
`separator`	The column separator used in the file
`filter_for`	A named list where names represent column names that must be filtered. For example: list(ProjectID = c("PROJECT1", "PROJECT2)) will filter the association file so that it contains only those rows for which the value of the column "ProjectID" is one of the specified values. If multiple columns are present in the list all filtering conditions are applied as a logical AND.
`export_widget_path`	A path on disk to save produced widgets or NULL if the user doesn't wish to save the html file

Details

The import series of functions is designed to work in combination with the use of Vispa2 pipeline, please refer to this article for more details: VISPA2: A Scalable Pipeline for High-Throughput Identification and Annotation of Vector Integration Sites.
The pipeline automatically produces an hierarchical structure in the file system which follows this schema:

- /root_folder

- Optional intermediate folders
  * ProjectID
    |_bam
    |_bcmuxall
    |_bed
    |_iss
    |_quality
    |_report
    |_quantification
    *|___concatenatePoolIDSeqRun

For each ProjectID there may be several nested PoolIDs. The alignment function only looks for PoolIDs in the quantification folder, since it's the location of the matrices to import. For more details on how to properly use these functions, refer to the vignette - vignette("how_to_import_functions"). If 'NULL' the file system alignment step is skipped.

Value

A tibble with the contents of the association file plus a column containing the path in the file system for every project and pool if found.

See Also

date_formats

Other Import functions: import_parallel_Vispa2Matrices_auto(), import_parallel_Vispa2Matrices_interact
import_single_Vispa2Matrix()

Examples

```
op <- options("ISAnalytics.widgets" = FALSE)
path <- system.file("extdata", "ex_association_file.tsv",
    package = "ISAnalytics"
)
root_pth <- system.file("extdata", "fs.zip", package = "ISAnalytics")
root <- unzip_file_system(root_pth, "fs")
association_file <- import_association_file(path, root, dates_format = "dmy")
options(op)
```

---

import_parallel_Vispa2Matrices_auto

Import integration matrices based on the association file.

Description

[Maturing] These functions are designed to import the appropriate integration matrix files given the association file and the root folder of the file system where Vispa2 matrices are generated.

Usage

```
import_parallel_Vispa2Matrices_auto(
  association_file,
  quantification_type,
  matrix_type = "annotated",
  workers = 2,
  multi_quant_matrix = TRUE,
  export_report_path = NULL,
  patterns = NULL,
  matching_opt = matching_options(),
  ...
)
```

Arguments

association_file	A single string containing the path to the association file on disk, or a data frame resulting from a previous call to `import_association_file`
quantification_type	A vector of requested quantification_types. Must be one in `quantification_types()`
matrix_type	A single string representing the type of matrices to be imported. Can only be one in `"annotated"` or `"not_annotated"`
workers	A single integer representing the number of parallel workers to use for the import
multi_quant_matrix	If set to TRUE will produce a multi-quantification matrix (data frame) through `comparison_matrix` instead of a list.
export_report_path	A path on disk to save produced import report or NULL if the user doesn't wish to save the html file
patterns	A character vector of additional patterns to match on file names. Please note that patterns must be regular expressions. Can be NULL if no patterns needs to be matched.
matching_opt	A single value between `matching_options`
...	<dynamic-dots> Additional named arguments to pass to `import_association_file` and `comparison_matrix`

Details

Import family functions are designed to work in combination with Vispa2, for more details on this take a look here: VISPA2: A Scalable Pipeline for High-Throughput Identification and Annotation of Vector Integration Sites.
For more details on how to properly use these functions, refer to the vignette - vignette("how_to_import_functions")

Value

A named list of data frames containing data from all imported integration matrices, divided by quantification type or a multi-quantification matrix

Automatic version

The automatic version of import_parallel_Vispa2Matrices doesn't interact with the user directly, for this reason options in this modality are more limited compared to the interactive version. In automatic version you can't:

- Choose single projects or pools: to have a selection import the association file first and filter it according to your needs before calling the function (more details on this in the vignette)
- Choose duplicates: if, after filtering by the specified patterns, duplicates are found they are automatically ignored

Interactive version

The interactive version of import_parallel_Vispa2Matrices asks user for input and allows a more detailed choice of projects to import, pools to import and, if necessary, duplicate files. During the execution, a series of reports is shown in html format.

See Also

matching_options, https://stringr.tidyverse.org/articles/regular-expressions.html

Other Import functions: import_association_file(), import_parallel_Vispa2Matrices_interactive(), import_single_Vispa2Matrix()

Examples

```
op <- options("ISAnalytics.widgets" = FALSE)
path <- system.file("extdata", "ex_association_file.tsv",
    package = "ISAnalytics"
)
root_pth <- system.file("extdata", "fs.zip", package = "ISAnalytics")
root <- unzip_file_system(root_pth, "fs")
matrices <- import_parallel_Vispa2Matrices_auto(
    association_file = path,
    quantification_type = c("fragmentEstimate", "seqCount"),
    patterns = NULL, matching_opt = "ANY",
    root = root,
    dates_format = "dmy",
    workers = 2
)
options(op)
```

---

import_parallel_Vispa2Matrices_interactive

Import integration matrices based on the association file.

Description

[Maturing] These functions are designed to import the appropriate integration matrix files given the association file and the root folder of the file system where Vispa2 matrices are generated.

Usage

```
import_parallel_Vispa2Matrices_interactive(
  association_file,
  quantification_type,
  matrix_type = "annotated",
  workers = 2,
  multi_quant_matrix = TRUE,
  export_report_path = NULL,
  ...
)
```

Arguments

association_file	A single string containing the path to the association file on disk, or a data frame resulting from a previous call to import_association_file
quantification_type	A vector of requested quantification_types. Must be one in quantification_types()
matrix_type	A single string representing the type of matrices to be imported. Can only be one in "annotated" or "not_annotated"
workers	A single integer representing the number of parallel workers to use for the import
multi_quant_matrix	If set to TRUE will produce a multi-quantification matrix (data frame) through comparison_matrix instead of a list.
export_report_path	A path on disk to save produced import report or NULL if the user doesn't wish to save the html file
...	<dynamic-dots> Additional named arguments to pass to import_association_file and comparison_matrix

Details

Import family functions are designed to work in combination with Vispa2, for more details on this take a look here: VISPA2: A Scalable Pipeline for High-Throughput Identification and Annotation of Vector Integration Sites.
For more details on how to properly use these functions, refer to the vignette - vignette("how_to_import_functions")

Value

A named list of data frames containing data from all imported integration matrices, divided by quantification type or a multi-quantification matrix

Interactive version

The interactive version of import_parallel_Vispa2Matrices asks user for input and allows a more detailed choice of projects to import, pools to import and, if necessary, duplicate files. During the execution, a series of reports is shown in html format.

See Also

comparison_matrix, import_association_file

Other Import functions: import_association_file(), import_parallel_Vispa2Matrices_auto(), import_single_Vispa2Matrix()

Examples

```
## Not run:
# Can't run because it's interactive and requires user input
matrices <- import_parallel_Vispa2Matrices_interactive(
    association_file,
    quantification_type,
    matrix_type = "annotated",
    workers = 2,
    multi_quant_matrix = FALSE,
    export_report_path = NULL,
)

## End(Not run)
```

---

import_single_Vispa2Matrix

Import a single integration matrix from file

Description

[Stable] This function allows to read and import an integration matrix produced as the output of Vispa2 pipeline and converts it to a tidy format.

Usage

```
import_single_Vispa2Matrix(path, to_exclude = NULL, separator = "\t")
```

Arguments

path	The path to the file on disk
to_exclude	Either NULL or a character vector of column names that should be ignored when importing
separator	The column delimiter used

Details

The import series of functions is designed to work in combination with the use of Vispa2 pipeline, please refer to this article for more details: VISPA2: A Scalable Pipeline for High-Throughput Identification and Annotation of Vector Integration Sites. For more details on how to properly use these functions, refer to `vignette("How to use import functions",package = "ISAnalytics")`

Value

A data.table object in tidy format

See Also

Other Import functions: import_association_file(), import_parallel_Vispa2Matrices_auto(), import_parallel_Vispa2Matrices_interactive()

Examples

```
path_to_file <- system.file("extdata", "ex_annotated_ISMatrix.tsv.xz",
    package = "ISAnalytics"
)
isa_dataframe <- import_single_Vispa2Matrix(path_to_file)
```

ISAnalytics *ISAnalytics: Analyze gene therapy vector insertion sites data identified from genomics next generation sequencing reads for clonal tracking studies*

Description

[Maturing] In gene therapy, stem cells are modified using viral vectors to deliver the therapeutic transgene and replace functional properties since the genetic modification is stable and inherited in all cell progeny. The retrieval and mapping of the sequences flanking the virus-host DNA junctions allows the identification of insertion sites (IS), essential for monitoring the evolution of genetically modified cells in vivo. A comprehensive toolkit for the analysis of IS is required to foster clonal trackign studies and supporting the assessment of safety and long term efficacy in vivo. This package is aimed at (1) supporting automation of IS workflow, (2) performing base and advance analysis for IS tracking (clonal abundance, clonal expansions and statistics for insertional mutagenesis, etc.), (3) providing basic biology insights of transduced stem cells in vivo.

Useful resources

- VISPA2: A Scalable Pipeline for High-Throughput Identification and Annotation of Vector Integration Sites

ISAnalytics function families

- Import functions:
  - import_single_Vispa2Matrix
  - import_association_file
  - import_parallel_Vispa2Matrices_interactive
  - import_parallel_Vispa2Matrices_auto
- Aggregation functions:
  - aggregate_metadata
  - aggregate_values_by_key
- Collision removal functions:
  - remove_collisions
  - realign_after_collisions
- Recalibration functions:
  - compute_near_integrations

- Analysis functions:
  - compute_abundance
  - comparison_matrix
  - separate_quant_matrices
  - threshold_filter
  - top_integrations
  - sample_statistics
  - CIS_grubbs
  - cumulative_count_union
- Plotting functions:
  - CIS_volcano_plot
- Utility functions:
  - generate_blank_association_file
  - generate_Vispa2_launch_AF
  - unzip_file_system
  - as_sparse_matrix

Vignettes

- `vignette("How to use import functions",package = "ISAnalytics")`
- `vignette("Collision removal functionality",package = "ISAnalytics")`
- `vignette("Working with aggregate functions",package = "ISAnalytics")`

`known_clinical_oncogenes` *Known clinical oncogenes (for mouse and human).*

Description

Known clinical oncogenes (for mouse and human).

Usage

```
known_clinical_oncogenes()
```

Value

A data frame

See Also

Other Plotting function helpers: `clinical_relevant_suspicious_genes()`

Examples

```
known_clinical_oncogenes()
```

mandatory_IS_vars *Names of mandatory variables for an integration matrix.*

Description

Contains the names of the columns that need to be present in order for a tibble to be considered an integration matrix.

Usage

```
mandatory_IS_vars()
```

Value

A character vector

Examples

```
mandatory_IS_vars()
```

matching_options *Possible choices for the* matching_opt *parameter.*

Description

These are all the possible values for the matching_opt parameter in import_parallel_vispa2Matrices_auto.

Usage

```
matching_options()
```

Details

The values "ANY", "ALL" and "OPTIONAL", represent how the patterns should be matched, more specifically

- ANY = look only for files that match AT LEAST one of the patterns specified
- ALL = look only for files that match ALL of the patterns specified
- OPTIONAL = look preferentially for files that match, in order, all patterns or any pattern and if no match is found return what is found (keep in mind that duplicates are discarded in automatic mode)

Value

A vector of characters for matching_opt

See Also

import_parallel_Vispa2Matrices_auto

Other Import functions helpers: quantification_types()

Examples

```
opts <- matching_options()
```

`quantification_types` *Possible choices for the* `quantification_type` *parameter.*

Description

These are all the possible values for the `quantification_type` parameter in `import_parallel_vispa2Matrices_inte` and `import_parallel_vispa2Matrices_auto`.

Usage

```
quantification_types()
```

Details

The possible values are:

- fragmentEstimate
- seqCount
- barcodeCount
- cellCount
- ShsCount

Value

A vector of characters for quantification types

See Also

`import_parallel_Vispa2Matrices_interactive`, `import_parallel_Vispa2Matrices_auto`

Other Import functions helpers: `matching_options()`

Examples

```
quant_types <- quantification_types()
```

realign_after_collisions *Re-aligns matrices of other quantification types based on the processed sequence count matrix.*

Description

[Experimental] This function should be used to keep data consistent among the same analysis: if for some reason you removed the collisions by passing only the sequence count matrix to the `remove_collisions` function, you should call this function afterwards, providing a list of other quantification matrices. NOTE: if you provided a list of several quantification types to `remove_collisions` before, there is no need to call this function.

Usage

```
realign_after_collisions(sc_matrix, other_matrices)
```

Arguments

`sc_matrix`	The sequence count matrix already processed for collisions via `remove_collisions`
`other_matrices`	A named list of matrices to re-align. Names in the list must be quantification types (`quantification_types()`) except "seqCount".

Details

For more details on how to use collision removal functionality: `vignette("Collision removal functionality",package = "ISAnalytics")`

Value

A named list with re-aligned matrices

See Also

`remove_collisions`

Other Collision removal: `remove_collisions()`

Examples

```
op <- options("ISAnalytics.widgets" = FALSE)
path <- system.file("extdata", "ex_association_file.tsv",
    package = "ISAnalytics"
)
root_pth <- system.file("extdata", "fs.zip", package = "ISAnalytics")
root <- unzip_file_system(root_pth, "fs")
association_file <- import_association_file(path, root,
    dates_format = "dmy"
)
matrices <- import_parallel_Vispa2Matrices_auto(
    association_file = association_file, root = NULL,
    quantification_type = c("fragmentEstimate", "seqCount"),
    matrix_type = "annotated", workers = 2,
```

```
    patterns = NULL, matching_opt = "ANY",
    multi_quant_matrix = FALSE
)
sc_matrix <- remove_collisions(matrices$seqCount, association_file)
others <- matrices[!names(matrices) %in% "seqCount"]
aligned_matrices <- realign_after_collisions(sc_matrix, others)
options(op)
```

---

reduced_AF_columns *Names of the columns of the association file to consider for Vispa2 launch.*

Description

Selection of column names from the association file to be considered for Vispa2 launch. NOTE: the `TagID` column appears only once but needs to be repeated twice for generating the launch file. Use the appropriate function to generate the file automatically.

Usage

```
reduced_AF_columns()
```

Value

A character vector

Examples

```
reduced_AF_columns()
```

---

remove_collisions *Identifies and removes collisions based on the sequence count matrix.*

Description

[Experimental] A collision is an integration (aka a unique combination of chr, integration_locus and strand) which is observed in more than one independent sample (a unique pair of ProjectID and SubjectID). The function tries to decide to which subject an integration should be assigned and if no decision can be taken, the integration is completely removed from the data frame.

Usage

```
remove_collisions(
  x,
  association_file,
  date_col = "SequencingDate",
  reads_ratio = 10,
  seq_count_col = "seqCount",
  max_rows_reports = 50,
  save_widget_path = NULL
)
```

Arguments

x	A named list of matrices (names must be quantification types), a single integration matrix representing the sequence count matrix of interest or a multi-quantification matrix obtained via comparison_matrix
association_file	The association file imported via import_association_file
date_col	The date column that should be considered for the analysis. Must be one value in date_columns_coll()
reads_ratio	A single numeric value that represents the ratio that has to be considered when deciding between seqCount value.
seq_count_col	For support of multi-quantification matrix - the name of the sequence count values column
max_rows_reports	A numeric value, represents the maximum number of rows of the reports data frames that can be printed on console if the option ISAnalytics.verbose is active. If the data frames are too large they won't be printed on console - we recommend using widgets for detailed and more accessible info.
save_widget_path	Either NULL or a path where the html report file should be saved. If NULL the report is visualized via browser ONLY (not saved on disk).

Details

If you don't want the function to show details and messages do: options(ISAnalitics.verbose = FALSE). To restore to the original value: options(ISAnalitics.verbose = TRUE). For more details on how to use collision removal functionality: vignette("Collision removal functionality",package = "ISAnalytics")

Value

A list of tibbles with removed collisions

See Also

date_columns_coll

Other Collision removal: realign_after_collisions()

Examples

```
op <- options("ISAnalytics.widgets" = FALSE)
path <- system.file("extdata", "ex_association_file.tsv",
    package = "ISAnalytics"
)
root_pth <- system.file("extdata", "fs.zip", package = "ISAnalytics")
root <- unzip_file_system(root_pth, "fs")
association_file <- import_association_file(path, root,
    dates_format = "dmy"
)
matrices <- import_parallel_Vispa2Matrices_auto(
    association_file = association_file, root = NULL,
    quantification_type = c("fragmentEstimate", "seqCount"),
    matrix_type = "annotated", workers = 2,
```

```
    patterns = NULL, matching_opt = "ANY",
    multi_quant_matrix = FALSE
)
matrices <- remove_collisions(matrices, association_file)
options(op)
```

`sample_statistics` *Computes user specified functions on numerical columns and updates the metadata data frame accordingly.*

Description

[Experimental] The function operates on a data frame by grouping the content by the sample key and computing every function specified on every column in the `value_columns` parameter. After that the metadata data frame is updated by including the computed results as columns for the corresponding key. For this reason it's required that both `x` and `metadata` have the same sample key, and it's particularly important if the user is working with previously aggregated data. For example:

```
### Importing association file and matrices
path_AF <- system.file("extdata", "ex_association_file.tsv",
package = "ISAnalytics")
root_correct <- system.file("extdata", "fs.zip",
package = "ISAnalytics")
root_correct <- unzip_file_system(root_correct, "fs")

association_file <- import_association_file(path_AF, root_correct)
matrices <- import_parallel_Vispa2Matrices_auto(
association_file = association_file , root = NULL,
quantification_type = c("seqCount","fragmentEstimate"),
matrix_type = "annotated", workers = 2, patterns = NULL,
matching_opt = "ANY", dates_format = "dmy")

### Aggregating data (both by same key)
aggreggated_x <- aggregate_values_by_key(matrices$seqCount,
association_file)
aggregated_meta <- aggregate_metadata(association_file)

### Sample statistics
sample_stats <- sample_statistics(x = aggregated_x,
metadata = aggregated_meta,
sample_key = c("SubjectID", "CellMarker","Tissue", "TimePoint"))
```

Usage

```
sample_statistics(
  x,
  metadata,
  sample_key = "CompleteAmplificationID",
  value_columns = "Value",
  functions = default_stats()
)
```

Arguments

x	A data frame
metadata	The metadata data frame
sample_key	Character vector representing the key for identifying a sample
value_columns	THe name of the columns to be computed, must be numeric or integer
functions	A named list of function or purrr-style lambdas

Value

A list with modified x and metadata data frames

See Also

Other Analysis functions: CIS_grubbs(), comparison_matrix(), compute_abundance(), cumulative_count_union separate_quant_matrices(), threshold_filter(), top_integrations()

Examples

```
op <- options(ISAnalytics.widgets = FALSE)

path_AF <- system.file("extdata", "ex_association_file.tsv",
    package = "ISAnalytics"
)
root_correct <- system.file("extdata", "fs.zip",
    package = "ISAnalytics"
)
root_correct <- unzip_file_system(root_correct, "fs")

association_file <- import_association_file(path_AF, root_correct,
    dates_format = "dmy"
)
matrices <- import_parallel_Vispa2Matrices_auto(
    association_file = association_file, root = NULL,
    quantification_type = c("seqCount", "fragmentEstimate"),
    matrix_type = "annotated", workers = 2, patterns = NULL,
    matching_opt = "ANY", multi_quant_matrix = FALSE
)

stats <- sample_statistics(matrices$seqCount, association_file)
options(op)
```

---

separate_quant_matrices

Separate a multiple-quantification matrix into single quantification matrices.

Description

[Maturing] The function separates a single multi-quantification integration matrix, obtained via comparison_matrix, into single quantification matrices as a named list of tibbles.

Usage

```
separate_quant_matrices(
  x,
  fragmentEstimate = "fragmentEstimate",
  seqCount = "seqCount",
  barcodeCount = "barcodeCount",
  cellCount = "cellCount",
  ShsCount = "ShsCount"
)
```

Arguments

x	Single integration matrix with multiple quantification value columns, likely obtained via comparison_matrix.
fragmentEstimate	Name of the fragment estimate values column in input
seqCount	Name of the sequence count values column in input
barcodeCount	Name of the barcode count values column in input
cellCount	Name of the cell count values column in input
ShsCount	Name of the shs count values column in input

Value

A named list of tibbles, where names are quantification types

See Also

quantification_types

Other Analysis functions: CIS_grubbs(), comparison_matrix(), compute_abundance(), cumulative_count_union
sample_statistics(), threshold_filter(), top_integrations()

Examples

```
op <- options("ISAnalytics.widgets" = FALSE)
path <- system.file("extdata", "ex_association_file.tsv",
    package = "ISAnalytics"
)
root_pth <- system.file("extdata", "fs.zip", package = "ISAnalytics")
root <- unzip_file_system(root_pth, "fs")
association_file <- import_association_file(path = path, root = root,
dates_format = "dmy")
matrices <- import_parallel_Vispa2Matrices_auto(
    association_file = association_file,
    quantification_type = c("seqCount", "fragmentEstimate"),
    matrix_type = "annotated", workers = 2, patterns = NULL,
    matching_opt = "ANY"
)
separated_matrix <- separate_quant_matrices(matrices)
options(op)
```

`threshold_filter` *Filter data frames with custom predicates*

Description

[Experimental] Filter a single data frame or a list of data frames with custom predicates assembled from the function parameters.

Usage

```
threshold_filter(x, threshold, cols_to_compare = "Value", comparators = ">")
```

Arguments

`x`	A data frame or a list of data frames
`threshold`	A numeric/integer vector or a named list of numeric/integer vectors
`cols_to_compare`	A character vector or a named list of character vectors
`comparators`	A character vector or a named list of character vectors. Must be one of the allowed values between `c("<",">","==","!=",">=","<=")`

Details

A single data frame as input:

If the user chooses to operate on a single data frame, the other parameters should only be vectors: numeric vector for `threshold` and character vectors for both `cols_to_compare` and `comparators`. A filtering condition is obtained by combining element by element `cols_to_compare` + `comparators` + `threshold` (similarly to the `paste` function). For example:
threshold = c(20, 35, 50)
cols_to_compare = c("a", "b", "c")
comparators = "<"
given these vectors, the input data frame will be filtered by checking which values in column "a" are less than 20 **AND** which values in column "b" are less than 35 **AND** which values in column "c" are less than 50. Things the user should keep in mind are:

- The vectors of length 1 are going to be recycled if one or more parameters are longer (in the example, the `comparators` value)
- If vectors are not of length 1 they must have the same length
- Columns to compare, of course, need to be included in the input data frame and need to be numeric/integer
- The filtering will perform a logical "AND" on all the conditions, only rows that satisfy ALL the conditions are preserved

A list of data frames as input:

The input for the function may also be a list of data frames, either named or unnamed.

Unnamed list:
If the input is a simple unnamed list, the other parameters should be simple vectors (as for data frames). All the predicates will simply be applied to every data frame in the list: this is useful if it's desirable to filter for the same conditions different data frames that have the same structure but different data.

Named list:
It is also possible to filter different data frames with different sets of conditions. Besides having the possibility of defining the other parameters as simple vector, which has the same results as operating on an unnamed list, the user can define the parameters as named lists containing vectors. For example:

```
example_df <- tibble::tibble(a = c(20, 30, 40),
                             b = c(40, 50, 60),
                             c = c("a", "b", "c"),
                             d = c(3L, 4L, 5L))
example_list <- list(first = example_df,
                     second = example_df,
                     third = example_df)
print(example_list)
```

```
## $first
## # A tibble: 3 x 4
##       a     b c         d
##   <dbl> <dbl> <chr> <int>
## 1    20    40 a         3
## 2    30    50 b         4
## 3    40    60 c         5
##
## $second
## # A tibble: 3 x 4
##       a     b c         d
##   <dbl> <dbl> <chr> <int>
## 1    20    40 a         3
## 2    30    50 b         4
## 3    40    60 c         5
##
## $third
## # A tibble: 3 x 4
##       a     b c         d
##   <dbl> <dbl> <chr> <int>
## 1    20    40 a         3
## 2    30    50 b         4
## 3    40    60 c         5
```

```
filtered <- threshold_filter(example_list,
                             threshold = list(first = c(20, 60),
                                              third = c(25)),
                             cols_to_compare = list(first = c("a", "b"),
                                                    third = c("a")),
                             comparators = list(first = c(">", "<"),
                                                third = c(">=")))
print(filtered)
```

```
## $first
## # A tibble: 1 x 4
##       a     b c         d
##   <dbl> <dbl> <chr> <int>
## 1    30    50 b         4
##
## $second
## # A tibble: 3 x 4
```

```
##       a     b c         d
##   <dbl> <dbl> <chr> <int>
## 1    20    40 a         3
## 2    30    50 b         4
## 3    40    60 c         5
##
## $third
## # A tibble: 2 x 4
##       a     b c         d
##   <dbl> <dbl> <chr> <int>
## 1    30    50 b         4
## 2    40    60 c         5
```

The above signature will roughly be translated as:

- Filter the element "first" in the list by checking that values in column "a" are bigger than 20 AND values in column "b" are less than 60
- Don't apply any filter to the element "second" (returns the data frame as is)
- Filter the element "third" by checking that values in column "a" are equal or bigger than 25.

It is also possible to use some parameters as vectors and some as lists: vectors will be recycled for every element filtered.

```
filtered <- threshold_filter(example_list,
                             threshold = list(first = c(20, 60),
                                              third = c(25, 65)),
                             cols_to_compare = c("a", "b"),
                             comparators = list(first = c(">", "<"),
                                                third = c(">=", "<=")))
```

In this example, different threshold and comparators will be applied to the same columns in all data frames.

Things the user should keep in mind are:

- Names for the list parameters must be the same names in the input list
- Only elements explicited in list parameters as names will be filtered
- Lengths of both vectors and lists must be consistent

Value

A data frame or a list of data frames

See Also

Other Analysis functions: `CIS_grubbs()`, `comparison_matrix()`, `compute_abundance()`, `cumulative_count_union`
`sample_statistics()`, `separate_quant_matrices()`, `top_integrations()`

Examples

```
example_df <- tibble::tibble(
    a = c(20, 30, 40),
    b = c(40, 50, 60),
    c = c("a", "b", "c"),
    d = c(3L, 4L, 5L)
)
example_list <- list(
    first = example_df,
    second = example_df,
```

```
    third = example_df
)

filtered <- threshold_filter(example_list,
    threshold = list(
        first = c(20, 60),
        third = c(25)
    ),
    cols_to_compare = list(
        first = c("a", "b"),
        third = c("a")
    ),
    comparators = list(
        first = c(">", "<"),
        third = c(">=")
    )
)
```

`top_integrations` *Sorts and keeps the top n integration sites based on the values in a given column.*

Description

[Experimental] The input data frame will be sorted by the highest values in the columns specified and the top n rows will be returned as output. The user can choose to keep additional columns in the output by passing a vector of column names or passing 2 "shortcuts":

- `keep = "everything"` keeps all columns in the original data frame
- `keep = "nothing"` only keeps the mandatory columns (`mandatory_IS_vars()`) plus the columns in the `columns` parameter.

Usage

```
top_integrations(
  x,
  n = 50,
  columns = "fragmentEstimate_sum_RelAbundance",
  keep = "everything",
  key = NULL
)
```

Arguments

`x`	An integration matrix (data frame containing `mandatory_IS_vars()`)
`n`	How many integrations should be sliced (in total or for each group)? Must be numeric or integer and greater than 0
`columns`	Columns to use for the sorting. If more than a column is supplied primary ordering is done on the first column, secondary ordering on all other columns
`keep`	Names of the columns to keep besides `mandatory_IS_vars()` and `columns`
`key`	Either `NULL` or a character vector of column names to group by. If not `NULL` the input will be grouped and the top fraction will be extracted from each group.

Value

Either a data frame with at most n rows or a data frames with at most n*(number of groups) rows.

See Also

Other Analysis functions: `CIS_grubbs()`, `comparison_matrix()`, `compute_abundance()`, `cumulative_count_union()`, `sample_statistics()`, `separate_quant_matrices()`, `threshold_filter()`

Examples

```
smpl <- tibble::tibble(
    chr = c("1", "2", "3", "4", "5", "6"),
    integration_locus = c(14536, 14544, 14512, 14236, 14522, 14566),
    strand = c("+", "+", "-", "+", "-", "+"),
    CompleteAmplificationID = c("ID1", "ID2", "ID1", "ID1", "ID3", "ID2"),
    Value = c(3, 10, 40, 2, 15, 150),
    Value2 = c(456, 87, 87, 9, 64, 96),
    Value3 = c("a", "b", "c", "d", "e", "f")
)
top <- top_integrations(smpl,
    n = 3,
    columns = c("Value", "Value2"),
    keep = "nothing"
)
top_key <- top_integrations(smpl,
    n = 3,
    columns = "Value",
    keep = "Value2",
    key = "CompleteAmplificationID"
)
```

---

unzip_file_system *A utility function to unzip and use example file systems included in the package*

Description

This utility function is a simple shortcut to create a temporary directory, unzip and reference the examples file systems included in the package for testing purposes.

Usage

```
unzip_file_system(zipfile, name)
```

Arguments

`zipfile`	The zipped file to decompress
`name`	The name of the folder in the zipped archive ("fs" or "fserr")

Value

A path to reference

See Also

Other Utility functions: as_sparse_matrix(), generate_Vispa2_launch_AF(), generate_blank_association_fil

Examples

```
root_pth <- system.file("extdata", "fs.zip", package = "ISAnalytics")
root <- unzip_file_system(root_pth, "fs")
```

Index