Package ‘ISAnalytics’

October 18, 2022

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

Version 1.6.2

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 (>= 4.2), magrittr

Imports utils, dplyr, readr, tidyr, purrr, rlang, tibble, BiocParallel, stringr, fs, lubridate, lifecycle, ggplot2, ggrepel, stats, psych, data.table, readxl, tools, Rcapture, grDevices, forcats, glue, shiny, shinyWidgets, datamods, bslib

Encoding UTF-8

LazyData false

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.0

Suggests testthat, covr, knitr, BiocStyle, sessioninfo, rmarkdown, roxygen2, vegan, withr, extraDistr, ggalluvial, scales, gridExtra, R.utils, RefManageR, flexdashboard, DT, circlize, plotly, gtools, eulerr, openxlsx

VignetteBuilder knitr

RdMacros lifecycle

Config/testthat/edition 3

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

git_branch RELEASE_3_15

git_last_commit 27ee9bb

git_last_commit_date 2022-05-23

Date/Publication 2022-10-18

Author Andrea Calabria [aut, cre],
Giulio Spinozzi [aut],
Giulia Pais [aut]

Maintainer Andrea Calabria <calabria.andrea@hsr.it>

R topics documented:

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

Description

[Stable] Groups metadata by the specified grouping keys and returns a summary of info for each group. For more details on how to use this function: `vignette("workflow_start", package = "ISAnalytics")`

Usage

```
aggregate_metadata(
  association_file,
  grouping_keys = c("SubjectID", "CellMarker", "Tissue", "TimePoint"),
  aggregating_functions = default_meta_agg(),
  import_stats = lifecycle::deprecated()
)
```

Arguments

`association_file`
: The imported association file (via import_association_file)

`grouping_keys`
: A character vector of column names to form a grouping operation

`aggregating_functions`
: A data frame containing specifications of the functions to be applied to columns in the association file during aggregation. It defaults to default_meta_agg. The structure of this data frame should be maintained if the user wishes to change the defaults.

`import_stats`
: **[Deprecated]** The import of VISPA2 stats has been moved to its dedicated function, see import_Vispa2_stats.

Value

An aggregated data frame

See Also

Other Data cleaning and pre-processing: `aggregate_values_by_key()`, `compute_near_integrations()`, `default_meta_agg()`, `outlier_filter()`, `outliers_by_pool_fragments()`, `purity_filter()`, `realign_after_collisions()`, `remove_collisions()`, `threshold_filter()`

Examples

```
data("association_file", package = "ISAnalytics")
aggreg_meta <- aggregate_metadata(
    association_file = association_file
)
head(aggreg_meta)
```

aggregate_values_by_key *Aggregates matrices values based on specified key.*

Description

[Stable] 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("workflow_start", 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()),
  join_af_by = "CompleteAmplificationID"
)
```

Arguments

`x`	A single integration matrix or a list of imported integration matrices
`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
`join_af_by`	A character vector representing the joining key between the matrix and the metadata. Useful to re-aggregate already aggregated matrices.

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 data frames or a single data frame aggregated according to the specified arguments

See Also

Other Data cleaning and pre-processing: aggregate_metadata(), compute_near_integrations(), default_meta_agg(), outlier_filter(), outliers_by_pool_fragments(), purity_filter(), realign_after_collisions(), remove_collisions(), threshold_filter()

Examples

```
data("integration_matrices", package = "ISAnalytics")
data("association_file", package = "ISAnalytics")
aggreg <- aggregate_values_by_key(
    x = integration_matrices,
    association_file = association_file,
    value_cols = c("seqCount", "fragmentEstimate")
)
head(aggreg)
```

annotation_issues *Check for genomic annotation problems in IS matrices.*

Description

[Experimental] This helper function checks if each individual integration site, identified by the `mandatory_IS_vars()`, has been annotated with two or more distinct gene symbols.

Usage

```
annotation_issues(matrix)
```

Arguments

`matrix`	Either a single matrix or a list of matrices, ideally obtained via `import_parallel_Vispa2Matrices()` or `import_single_Vispa2Matrix()`

Value

Either `NULL` if no issues were detected or 1 or more data frames with genomic coordinates of the IS and the number of distinct genes associated

See Also

Other Import functions helpers: `date_formats()`, `default_af_transform()`, `default_iss_file_prefixes()`, `matching_options()`, `quantification_types()`

Examples

```
data("integration_matrices", package = "ISAnalytics")
annotation_issues(integration_matrices)
```

association_file *Example of association file.*

Description

This file is a simple example of association file. Use it as reference to properly fill out yours. To generate an empty association file to fill see the `generate_blank_association_file()` function.

Usage

```
data("association_file")
```

Format

An object of class `data.table` (inherits from `data.frame`) with 53 rows and 83 columns.

Details

The data was obtained manually by simulating real research data.

See Also

`generate_blank_association_file`

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

Description

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

Usage

```
as_sparse_matrix(
  x,
  single_value_col = "Value",
  fragmentEstimate = "fragmentEstimate",
  seqCount = "seqCount",
  barcodeCount = "barcodeCount",
  cellCount = "cellCount",
  ShsCount = "ShsCount",
  key = pcr_id_column()
)
```

Arguments

`x` A single tidy integration matrix or a list of integration matrices. Supports also multi-quantification matrices obtained via comparison_matrix

`single_value_col` Name of the column containing the values when providing a single-quantification 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 `key` The name of the sample identifier fields (for aggregated matrices can be a vector with more than 1 element)

Value

Depending on input, 2 possible outputs:

- A single sparse matrix (data frame) if input is a single quantification matrix
- A list of sparse matrices divided by quantification if input is a single multi-quantification matrix or a list of matrices

See Also

Other Utilities: `comparison_matrix()`, `generate_Vispa2_launch_AF()`, `generate_blank_association_file()`, `generate_default_folder_structure()`, `separate_quant_matrices()`, `transform_columns()`

Examples

```
data("integration_matrices", package = "ISAnalytics")
sparse <- as_sparse_matrix(integration_matrices)
```

available_outlier_tests

A character vector containing all the names of the currently supported outliers tests that can be called in the function outlier_filter.

Description

A character vector containing all the names of the currently supported outliers tests that can be called in the function outlier_filter.

Usage

```
available_outlier_tests()
```

Value

A character vector

Examples

```
available_outlier_tests()
```

available_tags *All available tags for dynamic vars look-up tables.*

Description

Contains all information associated with critical tags used in the dynamic vars system. To know more see `vignette("workflow_start", package="ISAnalytics")`.

Usage

```
available_tags()
```

Value

A data frame

Examples

```
available_tags()
```

blood_lineages_default *Default blood lineages info*

Description

A default table with info relative to different blood lineages associated with cell markers that can be supplied as a parameter to `HSC_population_size_estimate`

Usage

```
blood_lineages_default()
```

Value

A data frame

Examples

```
blood_lineages_default()
```

circos_genomic_density

Trace a circos plot of genomic densities.

Description

[Stable] For this functionality the suggested package circlize is required. Please note that this function is a simple wrapper of basic `circlize` functions, for an in-depth explanation on how the functions work and additional arguments please refer to the official documentation Circular Visualization in R

Usage

```
circos_genomic_density(
  data,
  gene_labels = NULL,
  label_col = NULL,
  cytoband_specie = "hg19",
  track_colors = "navyblue",
  grDevice = c("png", "pdf", "svg", "jpeg", "bmp", "tiff", "default"),
  file_path = getwd(),
  ...
)
```

Arguments

`data`	Either a single integration matrix or a list of integration matrices. If a list is provided, a separate density track for each data frame is plotted.
`gene_labels`	Either `NULL` or a data frame in bed format. See details.
`label_col`	Numeric index of the column of `gene_labels` that contains the actual labels. Relevant only if `gene_labels` is not set to `NULL`.
`cytoband_specie`	Specie for initializing the cytoband
`track_colors`	Colors to give to density tracks. If more than one integration matrix is provided as `data` should be of the same length. Values are recycled if length of `track_colors` is smaller than the length of the input data.
`grDevice`	The graphical device where the plot should be traced. `default`, if executing from RStudio is the viewer.
`file_path`	If a device other than `default` is chosen, the path on disk where the file should be saved. Defaults to `{current directory}/circos_plot.{device}`.
`...`	Additional named arguments to pass on to chosen device, `circlize::circos.par()`, `circlize::circos.genomicDensity()` and `circlize::circos.genomicLabels()`

Details

Providing genomic labels:

If genomic labels should be plotted alongside genomic density tracks, the user should provide them as a simple data frame in standard bed format, namely `chr`, `start`, `end` plus a column containing the labels. NOTE: if the user decides to plot on the default device (viewer in RStudio), he must ensure there is enough space for all elements to be plotted, otherwise an error message is thrown.

Value

`NULL`

See Also

Other Plotting functions: `CIS_volcano_plot()`, `HSC_population_plot()`, `fisher_scatterplot()`, `integration_alluvial_plot()`, `sharing_heatmap()`, `sharing_venn()`, `top_abund_tableGrob()`

Examples

```
data("integration_matrices", package = "ISAnalytics")
data("association_file", package = "ISAnalytics")
aggreg <- aggregate_values_by_key(
    x = integration_matrices,
    association_file = association_file,
    value_cols = c("seqCount", "fragmentEstimate")
)
by_subj <- aggreg %>%
    dplyr::group_by(.data$SubjectID) %>%
    dplyr::group_split()
circos_genomic_density(by_subj,
    track_colors = c("navyblue", "gold"),
    grDevice = "default", track.height = 0.1
)
```

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

Description

[Stable] 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-in`

Usage

```
CIS_grubbs(
  x,
  genomic_annotation_file = "hg19",
  grubbs_flanking_gene_bp = 1e+05,
  threshold_alpha = 0.05,
  by = NULL,
  return_missing_as_df = TRUE,
  results_as_list = 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

`by` Either `NULL` or a character vector of column names. If not NULL, the function will perform calculations for each group and return a list of data frames with the results. E.g. for `by = "SubjectID"`, CIS will be computed for each distinct SubjectID found in the table ("SubjectID" column must be included in the input data frame).

`return_missing_as_df`
Returns those genes present in the input df but not in the refgenes as a data frame?

`results_as_list`
Relevant only if `by` is not `NULL` - if `TRUE` return the group computations as a named list, otherwise return a single df with an additional column containing the group id

Details

Genomic annotation file:

A data frame containing genes annotation for the specific genome. From version `1.5.4` the argument `genomic_annotation_file` accepts only data frames or package provided defaults. The user is responsible for importing the appropriate tabular files if customization is needed. The annotations for the human genome (hg19) and murine genome (mm9) are already included in this package: to use one of them just set the argument `genomic_annotation_file` to either `"hg19"` or `"mm9"`. 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 include:

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

Value

A data frame

Required tags

The function will explicitly check for the presence of these tags:

- chromosome
- locus
- is_strand
- gene_symbol
- gene_strand

See Also

Other Analysis functions: HSC_population_size_estimate(), compute_abundance(), cumulative_is(), gene_frequency_fisher(), is_sharing(), iss_source(), sample_statistics(), top_integrations(), top_targeted_genes()

Examples

```
data("integration_matrices", package = "ISAnalytics")
cis <- CIS_grubbs(integration_matrices)
cis
```

CIS_volcano_plot *Trace volcano plot for computed CIS data.*

Description

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

Usage

```
CIS_volcano_plot(
  x,
  onco_db_file = "proto_oncogenes",
  tumor_suppressors_db_file = "tumor_suppressors",
  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). If different from default, should be supplied as a path to a file.

tumor_suppressors_db_file
: Uniprot file for tumor-suppressor genes. If different from default, should be supplied as a path to a file.

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 with colorful labels

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 or character vector to be displayed in the title - usually the project name and other characterizing info. If a vector is supplied, it is concatenated in a single string via `paste()`

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. 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 `?tumor_suppressors`, `?proto_oncogenes`

Known oncogenes:

The default values are included in this package and it can be accessed by doing:

```
known_clinical_oncogenes()
```

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):

```
clinical_relevant_suspicious_genes()
```

Value

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

Required tags

The function will explicitly check for the presence of these tags:

- gene_symbol

See Also

Other Plotting functions: `HSC_population_plot()`, `circos_genomic_density()`, `fisher_scatterplot()`, `integration_alluvial_plot()`, `sharing_heatmap()`, `sharing_venn()`, `top_abund_tableGrob()`

Examples

```
data("integration_matrices", package = "ISAnalytics")
cis_plot <- CIS_volcano_plot(integration_matrices,
    title_prefix = "PJ01"
)
cis_plot
```

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

[Stable] Takes a list of integration matrices referring to different quantification types and merges them into a single data frame with 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",
  value_col_name = "Value"
)
```

Arguments

`x`	A named list of integration matrices, ideally obtained via import_parallel_Vispa2Matrices. Names must be quantification types in `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_col_name`	Name of the column containing the corresponding values in the single matrices

Value

A single data frame

See Also

quantification_types

Other Utilities: `as_sparse_matrix()`, `generate_Vispa2_launch_AF()`, `generate_blank_association_file()`, `generate_default_folder_structure()`, `separate_quant_matrices()`, `transform_columns()`

Examples

```
sc <- tibble::tribble(
    ~chr, ~integration_locus, ~strand, ~CompleteAmplificationID, ~Value,
    "1", 45324, "+", "ID1", 543,
    "2", 52423, "-", "ID1", 42,
    "6", 54623, "-", "ID2", 67,
    "X", 12314, "+", "ID3", 8
)
fe <- tibble::tribble(
    ~chr, ~integration_locus, ~strand, ~CompleteAmplificationID, ~Value,
    "1", 45324, "+", "ID1", 56.76,
    "2", 52423, "-", "ID1", 78.32,
    "6", 54623, "-", "ID2", 123.45,
    "X", 12314, "+", "ID3", 5.34
)
comparison_matrix(list(
    fragmentEstimate = fe,
    seqCount = sc
))
```

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

Description

[Stable] 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 = c("fragmentEstimate_sum"),
  percentage = TRUE,
  key = c("SubjectID", "CellMarker", "Tissue", "TimePoint"),
  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)

Required tags

The function will explicitly check for the presence of these tags:

- All columns declared in `mandatory_IS_vars()`

See Also

Other Analysis functions: `CIS_grubbs()`, `HSC_population_size_estimate()`, `cumulative_is()`, `gene_frequency_fisher()`, `is_sharing()`, `iss_source()`, `sample_statistics()`, `top_integrations()`, `top_targeted_genes()`

Examples

```
data("integration_matrices", package = "ISAnalytics")
abund <- compute_abundance(
    x = integration_matrices,
    columns = "fragmentEstimate",
    key = "CompleteAmplificationID"
)
head(abund)
```

`compute_near_integrations` *Scans input matrix to find and merge near integration sites.*

Description

[Stable] 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,
  is_identity_tags = c("chromosome", "is_strand"),
  keep_criteria = c("max_value", "keep_first"),
  value_columns = c("seqCount", "fragmentEstimate"),
  max_value_column = "seqCount",
  sample_id_column = pcr_id_column(),
  additional_agg_lambda = list(.default = default_rec_agg_lambdas()),
  max_workers = 4,
  map_as_file = TRUE,
  file_path = default_report_path(),
  strand_specific = lifecycle::deprecated()
)
```

Arguments

`x` An integration matrix

`threshold` A single integer that represents an absolute number of bases for which two integrations are considered distinct. If the threshold is set to 3 it means, provided fields `chr` and `strand` are the same, integrations sites which have at least 3 bases in between them are considered distinct.

`is_identity_tags`
Character vector of tags that identify the integration event as distinct (except for `"locus"`). See details.

`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

`value_columns` Character vector, contains the names of the numeric experimental columns

`max_value_column`
The column that has to be considered for searching the maximum value

`sample_id_column`
The name of the column containing the sample identifier

`additional_agg_lambda`
A named list containing aggregating functions for additional columns. See details.

`max_workers` Maximum parallel workers allowed

`map_as_file` Produce recalibration map as a .tsv file?

`file_path` String representing the path were the file will be saved. Must be a folder. Relevant only if `map_as_file` is `TRUE`.

`strand_specific`
[Deprecated] Deprecated, use `is_identity_tags`

Details

The concept of "near":

An integration event is uniquely identified by all fields specified in the `mandatory_IS_vars()` look-up table. It can happen to find IS that are formally distinct (different combination of values in the fields), but that should not considered distinct in practice, since they represent the same integration event - this may be due to artefacts at the putative locus of the IS in the merging of multiple sequencing libraries.

We say that an integration event IS1 is near to another integration event IS2 if the absolute difference of their loci is strictly lower than the set `threshold`.

The IS identity:

There is also another aspect to be considered. Since the algorithm is based on a sliding window mechanism, on which groups of IS should we set and slide the window?

By default, we have 3 fields in the `mandatory_IS_vars()`: chr, integration_locus, strand, and we assume that all the fields contribute to the identity of the IS. This means that IS1 and IS2 can be compared only if they have the same chromosome and the same strand. However, if we would like to exclude the strand of the integration from our considerations then IS1 and IS2 can be selected from all the events that fall on the same chromosome. A practical example:

IS1 = `(chr = "1", strand = "+", integration_locus = 14568)`

IS2 = `(chr = "1", strand = "-", integration_locus = 14567)`

if `is_identity_tags = c("chromosome", "is_strand")` IS1 and IS2 are considered distinct because they differ in strand, therefore no correction will be applied to loci of either of the 2. If `is_identity_tags = c("chromosome")` then IS1 and IS2 are considered near, because the strand is irrelevant, hence one of the 2 IS will change locus.

Aggregating near IS:

IS that fall in the same interval are evaluated according to the criterion selected - if recalibration is necessary, rows with the same sample ID are aggregated in a single row with a quantification value that is the sum of all the merged rows.

If the input integration matrix contains annotation columns, that is additional columns that are not

- part of the mandatory IS vars (see `mandatory_IS_vars()`)
- part of the annotation IS vars (see `annotation_IS_vars()`)
- the sample identifier column
- the quantification column

it is possible to specify how they should be aggregated. Defaults are provided for each column type (character, integer, numeric...), but custom functions can be specified as a named list, where names are column names in `x` and values are functions to be applied. NOTE: functions must be purrr-style lambdas and they must perform some kind of aggregating operation, aka they must take a vector as input and return a single value. The type of the output should match the type of the target column. If you specify custom lambdas, provide defaults in the special element `.defaults`. Example:

```
list(
  numeric_col = ~ sum(.x),
  char_col = ~ paste0(.x, collapse = ", "),
  .defaults = default_rec_agg_lambdas()
)
```

Value

An integration matrix with same or less number of rows

Required tags

The function will explicitly check for the presence of these tags:

- chromosome
- locus
- is_strand
- gene_symbol

Note

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

See Also

Other Data cleaning and pre-processing: `aggregate_metadata()`, `aggregate_values_by_key()`, `default_meta_agg()`, `outlier_filter()`, `outliers_by_pool_fragments()`, `purity_filter()`, `realign_after_collisions()`, `remove_collisions()`, `threshold_filter()`

Examples

```
data("integration_matrices", package = "ISAnalytics")
rec <- compute_near_integrations(
    x = integration_matrices, map_as_file = FALSE
)
head(rec)
```

`cumulative_is` *Expands integration matrix with the cumulative IS union over time.*

Description

[Experimental] Given an input integration matrix that can be grouped over time, this function adds integrations in groups assuming that if an integration is observed at time point "t" then it is also observed in time point "t+1".

Usage

```
cumulative_is(
  x,
  key = c("SubjectID", "CellMarker", "Tissue", "TimePoint"),
  timepoint_col = "TimePoint",
  include_tp_zero = FALSE,
  counts = TRUE,
  keep_og_is = FALSE,
  expand = TRUE
)
```

Arguments

`x`	An integration matrix, ideally aggregated via `aggregate_values_by_key()`
`key`	The aggregation key used
`timepoint_col`	The name of the time point column
`include_tp_zero`	Should time point 0 be included?
`counts`	Add cumulative counts? Logical
`keep_og_is`	Keep original set of integrations as a separate column?
`expand`	If `FALSE`, for each group, the set of integration sites is returned in a separate column as a nested table, otherwise the resulting column is unnested.

Value

A data frame

Required tags

The function will explicitly check for the presence of these tags:

- All columns declared in `mandatory_IS_vars()`
- Checks if the matrix is annotated by assessing presence of `annotation_IS_vars()`

See Also

Other Analysis functions: `CIS_grubbs()`, `HSC_population_size_estimate()`, `compute_abundance()`, `gene_frequency_fisher()`, `is_sharing()`, `iss_source()`, `sample_statistics()`, `top_integrations()`, `top_targeted_genes()`

Examples

```
data("integration_matrices", package = "ISAnalytics")
data("association_file", package = "ISAnalytics")
aggreg <- aggregate_values_by_key(
    x = integration_matrices,
    association_file = association_file,
    value_cols = c("seqCount", "fragmentEstimate")
)
cumulated_is <- cumulative_is(aggreg)
cumulated_is
```

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, see more in the dedicated package documentation.

Usage

```
date_formats()
```

Value

A character vector

See Also

import_association_file, import_parallel_Vispa2Matrices_auto

Other Import functions helpers: annotation_issues(), default_af_transform(), default_iss_file_prefixes(), matching_options(), quantification_types()

Examples

```
date_formats()
```

default_af_transform *Default transformations to apply to association file columns.*

Description

A list of default transformations to apply to the association file columns after importing it via import_association_file()

Usage

```
default_af_transform(convert_tp)
```

Arguments

convert_tp	The value of the argument convert_tp in the call to import_association_file()

Value

A named list of lambdas

See Also

Other Import functions helpers: `annotation_issues()`, `date_formats()`, `default_iss_file_prefixes()`, `matching_options()`, `quantification_types()`

Examples

```
default_af_transform(TRUE)
```

---

default_iss_file_prefixes

Default regex prefixes for Vispa2 stats files.

Description

Note that each element is a regular expression.

Usage

```
default_iss_file_prefixes()
```

Value

A character vector of regexes

See Also

Other Import functions helpers: `annotation_issues()`, `date_formats()`, `default_af_transform()`, `matching_options()`, `quantification_types()`

Examples

```
default_iss_file_prefixes()
```

`default_meta_agg` *Default metadata aggregation function table*

Description

A default columns-function specifications for aggregate_metadata

Usage

```
default_meta_agg()
```

Details

This data frame contains four columns:

- `Column`: holds the name of the column in the association file that should be processed
- `Function`: contains either the name of a function (e.g. mean) or a purrr-style lambda (e.g. `~ mean(.x, na.rm = TRUE)`). This function will be applied to the corresponding column specified in `Column`
- `Args`: optional additional arguments to pass to the corresponding function. This is relevant ONLY if the corresponding `Function` is a simple function and not a purrr-style lambda.
- `Output_colname`: a `glue` specification that will be used to determine a unique output column name. See glue for more details.

Value

A data frame

See Also

Other Data cleaning and pre-processing: `aggregate_metadata()`, `aggregate_values_by_key()`, `compute_near_integrations()`, `outlier_filter()`, `outliers_by_pool_fragments()`, `purity_filter()`, `realign_after_collisions()`, `remove_collisions()`, `threshold_filter()`

Examples

```
default_meta_agg()
```

`default_rec_agg_lambdas` *Defaults for column aggregations in* `compute_near_integrations()`.

Description

Defaults for column aggregations in `compute_near_integrations()`.

Usage

```
default_rec_agg_lambdas()
```

Value

A named list of lambdas

Examples

```
default_rec_agg_lambdas()
```

`default_report_path` *Default folder for saving ISAnalytics reports. Supplied as default argument for several functions.*

Description

Default folder for saving ISAnalytics reports. Supplied as default argument for several functions.

Usage

```
default_report_path()
```

Value

A path

Examples

```
default_report_path()
```

`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()
```

`fisher_scatterplot` *Plot results of gene frequency Fisher's exact test.*

Description

[Stable] Plots results of Fisher's exact test on gene frequency obtained via `gene_frequency_fisher()` as a scatterplot.

Usage

```
fisher_scatterplot(
  fisher_df,
  p_value_col = "Fisher_p_value_fdr",
  annot_threshold = 0.05,
  annot_color = "red",
  gene_sym_col = "GeneName",
  do_not_highlight = NULL,
  keep_not_highlighted = TRUE
)
```

Arguments

`fisher_df` Test results obtained via `gene_frequency_fisher()`

`p_value_col` Name of the column containing the p-value to consider

`annot_threshold`
Annotate with a different color if a point is below the significance threshold. Single numerical value.

`annot_color` The color in which points below the threshold should be annotated

`gene_sym_col` The name of the column containing the gene symbol

`do_not_highlight`
Either `NULL`, a character vector, an expression or a purrr-style lambda. Tells the function to ignore the highlighting and labeling of these genes even if their p-value is below the threshold. See details.

`keep_not_highlighted`
If present, how should not highlighted genes be treated? If set to `TRUE` points are plotted and colored with the chosen color scale. If set to `FALSE` the points are removed entirely from the plot.

Details

Specifying genes to avoid highlighting:

In some cases, users might want to avoid highlighting certain genes even if their p-value is below the threshold. To do so, use the argument `do_not_highlight`: character vectors are appropriate for specific genes that are to be excluded, expressions or lambdas allow a finer control. For example we can supply:

```
expr <- rlang::expr(!stringr::str_starts(GeneName, "MIR") &
                      average_TxLen_1 >= 300)
```

with this expression, genes that have a p-value < threshold and start with "MIR" or have an average_TxLen_1 lower than 300 are excluded from the highlighted points. NOTE: keep in mind that expressions are evaluated inside a `dplyr::filter` context.

Similarly, lambdas are passed to the filtering function but only operate on the column containing the gene symbol.

```
lambda <- ~ stringr::str_starts(.x, "MIR")
```

Value

A plot

See Also

Other Plotting functions: `CIS_volcano_plot()`, `HSC_population_plot()`, `circos_genomic_density()`, `integration_alluvial_plot()`, `sharing_heatmap()`, `sharing_venn()`, `top_abund_tableGrob()`

Examples

```
data("integration_matrices", package = "ISAnalytics")
data("association_file", package = "ISAnalytics")
aggreg <- aggregate_values_by_key(
    x = integration_matrices,
    association_file = association_file,
    value_cols = c("seqCount", "fragmentEstimate")
)
cis <- CIS_grubbs(aggreg, by = "SubjectID")
fisher <- gene_frequency_fisher(cis$cis$PT001, cis$cis$PT002,
    min_is_per_gene = 2
)
fisher_scatterplot(fisher)
```

generate_blank_association_file

Create a blank association file.

Description

Produces a blank association file to start using both VISPA2 and ISAnalytics

Usage

```
generate_blank_association_file(path)
```

Arguments

path	The path on disk where the file should be written - must be a file

Value

NULL

See Also

Other Utilities: as_sparse_matrix(), comparison_matrix(), generate_Vispa2_launch_AF(), generate_default_folder_structure(), separate_quant_matrices(), transform_columns()

Examples

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

generate_default_folder_structure

Generate a default folder structure, following VISPA2 standards

Description

The function produces a folder structure in the file system at the provided path that respects VISPA2 standards, with package-included data.

Usage

```
generate_default_folder_structure(
  type = "correct",
  dir = tempdir(),
  af = "default",
  matrices = "default"
)
```

Arguments

type	One value between "correct", "incorrect" and "both". Tells the function wheter to produce a correct structure or introduce some errors (mainly for testing purposes).
dir	Path to the folder in which the structure will be produced
af	Either "default" for the association file provided as example in the package or a custom association file as a data frame
matrices	Either "default" for integration matrices provided as example in the package or a custom multi-quantification matrix

Value

A named list containing the path to the association file and the path to the top level folder(s) of the structure

Required tags

The function will explicitly check for the presence of these tags:

- project_id
- tag_seq
- vispa_concatenate

See Also

Other Utilities: as_sparse_matrix(), comparison_matrix(), generate_Vispa2_launch_AF(), generate_blank_association_file(), separate_quant_matrices(), transform_columns()

Examples

```
fs_path <- generate_default_folder_structure(type = "correct")
fs_path
```

generate_Vispa2_launch_AF

Creates a reduced association file for a 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
`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("PJ01", "PJ01") - c("POOL01", "POOL02"). 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

`NULL`

Required tags

The function will explicitly check for the presence of these tags:

- cell_marker
- fusion_id
- pcr_repl_id
- pool_id
- project_id
- subject
- tag_id
- tissue
- tp_days
- vector_id

The names of the pools in the `pool` argument is checked against the column corresponding to the `pool_id` tag.

See Also

Other Utilities: `as_sparse_matrix()`, `comparison_matrix()`, `generate_blank_association_file()`, `generate_default_folder_structure()`, `separate_quant_matrices()`, `transform_columns()`

Examples

```
temp <- tempdir()
data("association_file", package = "ISAnalytics")
generate_Vispa2_launch_AF(association_file, "PJ01", "POOL01", temp)
```

`gene_frequency_fisher` *Compute Fisher's exact test on gene frequencies.*

Description

[Experimental] Provided 2 data frames with calculations for CIS, via `CIS_grubbs()`, computes Fisher's exact test. Results can be plotted via `fisher_scatterplot()`.

Usage

```
gene_frequency_fisher(
  cis_x,
  cis_y,
  min_is_per_gene = 3,
  gene_set_method = c("intersection", "union"),
  onco_db_file = "proto_oncogenes",
  tumor_suppressors_db_file = "tumor_suppressors",
  species = "human",
  known_onco = known_clinical_oncogenes(),
  suspicious_genes = clinical_relevant_suspicious_genes(),
  significance_threshold = 0.05,
  remove_unbalanced_0 = TRUE
)
```

Arguments

`cis_x` A data frame obtained via `CIS_grubbs()`

`cis_y` A data frame obtained via `CIS_grubbs()`

`min_is_per_gene`
Used for pre-filtering purposes. Genes with a number of distinct integration less than this number will be filtered out prior calculations. Single numeric or integer.

`gene_set_method`
One between "intersection" and "union". When merging the 2 data frames, `intersection` will perform an inner join operation, while `union` will perform a full join operation.

`onco_db_file` Uniprot file for proto-oncogenes (see details). If different from default, should be supplied as a path to a file.

`tumor_suppressors_db_file`
Uniprot file for tumor-suppressor genes. If different from default, should be supplied as a path to a file.

`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`
Significance threshold for the Fisher's test p-value

`remove_unbalanced_0`
Remove from the final output those pairs in which there are no IS for one group or the other and the number of IS of the non-missing group are less than the mean number of IS for that group

Details

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 `?tumor_suppressors`, `?proto_oncogenes`

Known oncogenes:

The default values are included in this package and it can be accessed by doing:

```
known_clinical_oncogenes()
```

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):

```
clinical_relevant_suspicious_genes()
```

Value

A data frame

Required tags

The function will explicitly check for the presence of these tags: *

See Also

Other Analysis functions: CIS_grubbs(), HSC_population_size_estimate(), compute_abundance(), cumulative_is(), is_sharing(), iss_source(), sample_statistics(), top_integrations(), top_targeted_genes()

Examples

```
data("integration_matrices", package = "ISAnalytics")
data("association_file", package = "ISAnalytics")
aggreg <- aggregate_values_by_key(
    x = integration_matrices,
    association_file = association_file,
    value_cols = c("seqCount", "fragmentEstimate")
)
cis <- CIS_grubbs(aggreg, by = "SubjectID")
fisher <- gene_frequency_fisher(cis$cis$PT001, cis$cis$PT002,
    min_is_per_gene = 2
)
fisher
```

HSC_population_plot *Plot of the estimated HSC population size for each patient.*

Description

Plot of the estimated HSC population size for each patient.

Usage

```
HSC_population_plot(
  estimates,
  project_name,
  timepoints = "Consecutive",
  models = "Mth Chao (LB)"
)
```

Arguments

`estimates`	The estimates data frame, obtained via `HSC_population_size_estimate`
`project_name`	The project name, will be included in the plot title
`timepoints`	Which time points to plot? One between "All", "Stable" and "Consecutive"
`models`	Name of the models to plot (as they appear in the column of the estimates)

Value

A plot

See Also

Other Plotting functions: `CIS_volcano_plot()`, `circos_genomic_density()`, `fisher_scatterplot()`, `integration_alluvial_plot()`, `sharing_heatmap()`, `sharing_venn()`, `top_abund_tableGrob()`

Examples

```
data("integration_matrices", package = "ISAnalytics")
data("association_file", package = "ISAnalytics")
aggreg <- aggregate_values_by_key(
    x = integration_matrices,
    association_file = association_file,
    value_cols = c("seqCount", "fragmentEstimate")
)
aggreg_meta <- aggregate_metadata(
    association_file = association_file
)
estimate <- HSC_population_size_estimate(
    x = aggreg,
    metadata = aggreg_meta,
    stable_timepoints = c(90, 180, 360),
    cell_type = "Other"
)
p <- HSC_population_plot(estimate, "PJ01")
p
```

`HSC_population_size_estimate`

Hematopoietic stem cells population size estimate.

Description

[Stable] Hematopoietic stem cells population size estimate with capture-recapture models.

Usage

```
HSC_population_size_estimate(
  x,
  metadata,
  stable_timepoints = NULL,
  aggregation_key = c("SubjectID", "CellMarker", "Tissue", "TimePoint"),
  blood_lineages = blood_lineages_default(),
  timepoint_column = "TimePoint",
  seqCount_column = "seqCount_sum",
  fragmentEstimate_column = "fragmentEstimate_sum",
  seqCount_threshold = 3,
  fragmentEstimate_threshold = 3,
  nIS_threshold = 5,
  cell_type = "MYELOID",
  tissue_type = "PB"
)
```

Arguments

`x` An aggregated integration matrix. See details.

`metadata` An aggregated association file. See details.

`stable_timepoints` A numeric vector or NULL if there are no stable time points.

`aggregation_key` A character vector indicating the key used for aggregating x and metadata. Note that x and metadata should always be aggregated with the same key.

`blood_lineages` A data frame containing information on the blood lineages. Users can supply their own, provided the columns `CellMarker` and `CellType` are present.

`timepoint_column` What is the name of the time point column to use? Note that this column must be present in the key.

`seqCount_column` What is the name of the column in x containing the values of sequence count quantification?

`fragmentEstimate_column` What is the name of the column in x containing the values of fragment estimate quantification? If fragment estimate is not present in the matrix, param should be set to `NULL`.

`seqCount_threshold` A single numeric value. After re-aggregating `x`, rows with a value greater or equal will be kept, the others will be discarded.

`fragmentEstimate_threshold` A single numeric value. Threshold value for fragment estimate, see details.

`nIS_threshold` A single numeric value. If a group (row) in the metadata data frame has a count of distinct integration sites strictly greater than this number it will be kept, otherwise discarded.

cell_type	The cell types to include in the models. Note that the matching is case-insensitive.
tissue_type	The tissue types to include in the models. Note that the matching is case-insensitive.

Value

A data frame with the results of the estimates

Input formats

Both x and metadata should be supplied to the function in aggregated format (ideally through the use of aggregate_metadata and aggregate_values_by_key). Note that the aggregation_key, aka the vector of column names used for aggregation, must contain at least the columns associated with the tags subject, cell_marker, tissue and a time point column (the user can specify the name of the column in the argument timepoint_column).

On time points

If stable_timepoints is a vector with length > 1, the function will look for the first available stable time point and slice the data from that time point onward. If NULL is supplied instead, it means there are no stable time points available. Note that 0 time points are ALWAYS discarded. Also, to be included in the analysis, a group must have at least 2 distinct non-zero time points.

Setting a threshold for fragment estimate

If fragment estimate is present in the input matrix, the filtering logic changes slightly: rows in the original matrix are kept if the sequence count value is greater or equal than the seqCount_threshold AND the fragment estimate value is greater or equal to the fragmentEstimate_threshold IF PRESENT (non-zero value). This means that for rows that miss fragment estimate, the filtering logic will be applied only on sequence count. If the user wishes not to use the combined filtering with fragment estimate, simply set fragmentEstimate_threshold = 0.

Required tags

The function will explicitly check for the presence of these tags:

- subject
- tissue
- cell_marker

See Also

Other Analysis functions: CIS_grubbs(), compute_abundance(), cumulative_is(), gene_frequency_fisher(), is_sharing(), iss_source(), sample_statistics(), top_integrations(), top_targeted_genes()

Examples

```
data("integration_matrices", package = "ISAnalytics")
data("association_file", package = "ISAnalytics")
aggreg <- aggregate_values_by_key(
    x = integration_matrices,
    association_file = association_file,
    value_cols = c("seqCount", "fragmentEstimate")
)
aggreg_meta <- aggregate_metadata(association_file = association_file)
estimate <- HSC_population_size_estimate(
    x = aggreg,
    metadata = aggreg_meta,
    fragmentEstimate_column = NULL,
    stable_timepoints = c(90, 180, 360),
    cell_type = "Other"
)
```

---

import_association_file

Import the association file from disk

Description

[Stable] Imports the association file and optionally 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,
  dates_format = "ymd",
  separator = "\t",
  filter_for = NULL,
  import_iss = FALSE,
  convert_tp = TRUE,
  report_path = default_report_path(),
  transformations = default_af_transform(convert_tp),
  tp_padding = lifecycle::deprecated(),
  ...
)
```

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.
`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.

`import_iss` Import VISPA2 pool stats and merge them with the association file? Logical value

`convert_tp` Should be time points be converted into months and years? Logical value

`report_path` The path where the report file should be saved. Can be a folder or `NULL` if no report should be produced. Defaults to `{user_home}/ISAnalytics_reports`.

`transformations`
Either `NULL` or a named list of purrr-style lambdas where names are column names the function should be applied to.

`tp_padding` **[Deprecated]** Deprecated. Use `transformations` instead.

`...` Additional arguments to pass to `import_Vispa2_stats`

Details

Transformations:

Lambdas provided in input in the `transformations` argument, must be transformations, aka functions that take in input a vector and return a vector of the same length as the input.

If the transformation list contains column names that are not present in the data frame, they are simply ignored.

File system alignment:

If the `root` argument is set to `NULL` no file system alignment is performed. This allows to import the basic file but it won't be possible to perform automated matrix and stats import. For more details see the "How to use import functions" vignette: `vignette("workflow_start", package = "ISAnalytics")`

Time point conversion:

The time point conversion is based on the following logic, given `TPD` is the column containing the time point expressed in days and `TPM` and `TPY` are respectively the time points expressed as month and years

- If `TPD` is `NA` –> `NA` (for both months and years)
- `TPM` = 0, `TPY` = 0 if and only if `TPD` = 0 For conversion in months:
- `TPM` = ceiling(`TPD`/30) if `TPD` < 30 otherwise `TPM` = round(`TPD`/30) For conversion in years:
- `TPY` = ceiling(`TPD`/360)

Value

The data frame containing metadata

Required tags

The function will explicitly check for the presence of these tags:

- project_id
- pool_id
- tag_seq
- subject
- tissue
- tp_days
- cell_marker
- pcr_replicate
- vispa_concatenate
- pcr_repl_id
- proj_folder

The function will use all the available specifications contained in `association_file_columns(TRUE)` to read and parse the file. If the specifications contain columns with a type `"date"`, the function will parse the generic date with the format in the `dates_format` argument.

See Also

`transform_columns`

`date_formats`

Other Import functions: `import_Vispa2_stats()`, `import_parallel_Vispa2Matrices()`, `import_single_Vispa2Matri`

Examples

```
fs_path <- generate_default_folder_structure(type = "correct")
af <- import_association_file(fs_path$af,
    root = fs_path$root,
    report_path = NULL
)
head(af)
```

---

import_parallel_Vispa2Matrices

Import integration matrices from paths in the association file.

Description

[Stable] The function offers a convenient way of importing multiple integration matrices in an automated or semi-automated way. For more details see the "How to use import functions" vignette: `vignette("workflow_start", package = "ISAnalytics")`

Usage

```
import_parallel_Vispa2Matrices(
  association_file,
  quantification_type = c("seqCount", "fragmentEstimate"),
  matrix_type = c("annotated", "not_annotated"),
  workers = 2,
  multi_quant_matrix = TRUE,
  report_path = default_report_path(),
  patterns = NULL,
  matching_opt = matching_options(),
  mode = c("AUTO", "INTERACTIVE"),
  ...
)
```

Arguments

`association_file`
: Data frame imported via import_association_file (with file system alignment) or a string containing the path to the association file on disk.

`quantification_type`
: A vector of requested quantification_types. Possible choices are 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 through comparison_matrix instead of a list.

`report_path`
: The path where the report file should be saved. Can be a folder or `NULL` if no report should be produced. Defaults to `{user_home}/ISAnalytics_reports`.

`patterns`
: Relevant only if argument `mode` is set to `AUTO`. 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 need to be matched.

`matching_opt`
: Relevant only if argument `mode` is set to `AUTO`. A single value between matching_options

`mode`
: A single value between `AUTO` and `INTERACTIVE`. If `INTERACTIVE`, the function will ask for input from the user on console, otherwise the process is fully automated (with limitations, see vignette).

`...`
: <`dynamic-dots`> Additional named arguments to pass to `ìmport_association_file`, `comparison_matrix` and `import_single_Vispa2_matrix`

Value

Either a multi-quantification matrix or a list of integration matrices

Required tags

The function will explicitly check for the presence of these tags:

- project_id
- vispa_concatenate

See Also

Other Import functions: `import_Vispa2_stats()`, `import_association_file()`, `import_single_Vispa2Matrix()`

Examples

```
fs_path <- generate_default_folder_structure(type = "correct")
af <- import_association_file(fs_path$af,
    root = fs_path$root,
    report_path = NULL
)
matrices <- import_parallel_Vispa2Matrices(af,
    c("seqCount", "fragmentEstimate"),
    mode = "AUTO", report_path = NULL
)
head(matrices)
```

---

import_single_Vispa2Matrix

Import a single integration matrix from file

Description

[Stable] This function allows to read and import an integration matrix (ideally produced by VISPA2) and converts it to a tidy format.

Usage

```
import_single_Vispa2Matrix(
  path,
  separator = "\t",
  additional_cols = NULL,
  transformations = NULL,
  sample_names_to = pcr_id_column(),
  values_to = "Value",
  to_exclude = lifecycle::deprecated(),
  keep_excluded = lifecycle::deprecated()
)
```

Arguments

`path`	The path to the file on disk
`separator`	The column delimiter used, defaults to `\t`
`additional_cols`	Either `NULL`, a named character vector or a named list. See details.
`transformations`	Either `NULL` or a named list of purrr-style lambdas where names are column names the function should be applied to.
`sample_names_to`	Name of the output column holding the sample identifier. Defaults to `pcr_id_column()`
`values_to`	Name of the output column holding the quantification values. Defaults to `Value`.
`to_exclude`	**[Deprecated]** Deprecated. Use `additonal_cols` instead
`keep_excluded`	**[Deprecated]** Deprecated. Use `additonal_cols` instead

Details

Additional columns:

Additional columns are annotation columns present in the integration matrix to import that are not

- part of the mandatory IS vars (see `mandatory_IS_vars()`)
- part of the annotation IS vars (see `annotation_IS_vars()`)
- the sample identifier column
- the quantification column

When specified they tell the function how to treat those columns in the import phase, by providing a named character vector, where names correspond to the additional column names and values are a choice of the following:

- `"char"` for character (strings)
- `"int"` for integers
- `"logi"` for logical values (TRUE / FALSE)
- `"numeric"` for numeric values
- `"factor"` for factors
- `"date"` for generic date format - note that functions that need to read and parse files will try to guess the format and parsing may fail
- One of the accepted date/datetime formats by `lubridate`, you can use `ISAnalytics::date_formats()` to view the accepted formats
- `"_"` to drop the column

For more details see the "How to use import functions" vignette: `vignette("workflow_start", package = "ISAnalytics")`

Transformations:

Lambdas provided in input in the `transformations` argument, must be transformations, aka functions that take in input a vector and return a vector of the same length as the input.

If the transformation list contains column names that are not present in the data frame, they are simply ignored.

Value

A data.table object in tidy format

Required tags

The function will explicitly check for the presence of these tags:

- All columns declared in mandatory_IS_vars()

See Also

transform_columns

Other Import functions: import_Vispa2_stats(), import_association_file(), import_parallel_Vispa2Matrices()

Examples

```
fs_path <- generate_default_folder_structure(type = "correct")
matrix_path <- fs::path(
    fs_path$root, "PJ01", "quantification",
    "POOL01-1", "PJ01_POOL01-1_seqCount_matrix.no0.annotated.tsv.gz"
)
matrix <- import_single_Vispa2Matrix(matrix_path)
head(matrix)
```

---

import_Vispa2_stats *Import Vispa2 stats given the aligned association file.*

Description

[Stable] Imports all the Vispa2 stats files for each pool provided the association file has been aligned with the file system (see import_association_file).

Usage

```
import_Vispa2_stats(
  association_file,
  file_prefixes = default_iss_file_prefixes(),
  join_with_af = TRUE,
  pool_col = "concatenatePoolIDSeqRun",
  report_path = default_report_path()
)
```

Arguments

`association_file`
: The file system aligned association file (contains columns with absolute paths to the 'iss' folder)

`file_prefixes`
: A character vector with known file prefixes to match on file names. NOTE: the elements represent regular expressions. For defaults see default_iss_file_prefixes.

`join_with_af`
: Logical, if `TRUE` the imported stats files will be merged with the association file, if `FALSE` a single data frame holding only the stats will be returned.

`pool_col`
: A single string. What is the name of the pool column used in the Vispa2 run? This will be used as a key to perform a join operation with the stats files `POOL` column.

`report_path`
: The path where the report file should be saved. Can be a folder or `NULL` if no report should be produced. Defaults to `{user_home}/ISAnalytics_reports`.

Value

A data frame

Required tags

The function will explicitly check for the presence of these tags:

- project_id
- tag_seq
- vispa_concatenate
- pcr_repl_id

See Also

Other Import functions: `import_association_file()`, `import_parallel_Vispa2Matrices()`, `import_single_Vispa2Matrix()`

Examples

```
fs_path <- generate_default_folder_structure(type = "correct")
af <- import_association_file(fs_path$af,
    root = fs_path$root,
    import_iss = FALSE,
    report_path = NULL
)
stats_files <- import_Vispa2_stats(af,
    join_with_af = FALSE,
    report_path = NULL
)
head(stats_files)
```

inspect_tags *Retrieve description of a tag by name.*

Description

Given one or multiple tags, prints the associated description and functions where the tag is explicitly used.

Usage

```
inspect_tags(tags)
```

Arguments

`tags`	A character vector of tag names

Value

`NULL`

See Also

Other dynamic vars: `mandatory_IS_vars()`, `pcr_id_column()`, `reset_mandatory_IS_vars()`, `set_mandatory_IS_vars()`, `set_matrix_file_suffixes()`

Examples

```
inspect_tags(c("chromosome", "project_id", "x"))
```

integration_alluvial_plot *Alluvial plots for IS distribution in time.*

Description

[Stable] Alluvial plots allow the visualization of integration sites distribution in different points in time in the same group. This functionality requires the suggested package ggalluvial.

Usage

```
integration_alluvial_plot(
  x,
  group = c("SubjectID", "CellMarker", "Tissue"),
  plot_x = "TimePoint",
  plot_y = "fragmentEstimate_sum_PercAbundance",
  alluvia = mandatory_IS_vars(),
  alluvia_plot_y_threshold = 1,
  top_abundant_tbl = TRUE,
  empty_space_color = "grey90",
  ...
)
```

Arguments

`x`	A data frame. See details.
`group`	Character vector containing the column names that identify unique groups.
`plot_x`	Column name to plot on the x axis
`plot_y`	Column name to plot on the y axis
`alluvia`	Character vector of column names that uniquely identify alluvia
`alluvia_plot_y_threshold`	Numeric value. Everything below this threshold on y will be plotted in grey and aggregated. See details.
`top_abundant_tbl`	Logical. Produce the summary top abundant tables via top_abund_tableGrob?
`empty_space_color`	Color of the empty portion of the bars (IS below the threshold). Can be either a string of known colors, an hex code or `NA_character` to set the space transparent. All color specs accepted in ggplot2 are suitable here.
`...`	Additional arguments to pass on to top_abund_tableGrob

Details

Input data frame:

The input data frame must contain all the columns specified in the arguments `group`, `plot_x`, `plot_y` and `alluvia`. The standard input for this function is the data frame obtained via the compute_abundance function.

Plotting threshold on y:

The plotting threshold on the quantification on the y axis has the function to highlight only relevant information on the plot and reduce computation time. The default value is 1, that acts on the default column plotted on the y axis which contains a percentage value. This translates in natural language roughly as "highlight with colors only those integrations (alluvia) that at least in 1 point in time have an abundance value >= 1 %". The remaining integrations will be plotted as a unique layer in the column, colored as specified by the argument `empty_space_color`.

Customizing the plot:

The returned plots are ggplot2 objects and can therefore further modified as any other ggplot2 object. For example, if the user decides to change the fill scale it is sufficient to do

```
plot +
  ggplot2::scale_fill_viridis_d(...) + # or any other discrete fill scale
  ggplot2::theme(...) # change theme options
```

NOTE: if you requested the computation of the top ten abundant tables and you want the colors to match you should re-compute them

A note on strata ordering:

Strata in each column are ordered first by time of appearance and secondly in decreasing order of abundance (value of y). It means, for example, that if the plot has 2 or more columns, in the second column, on top, will appear first appear IS that appeared in the previous columns and then all other IS, ordered in decreasing order of abundance.

Value

For each group a list with the associated plot and optionally the summary tableGrob

See Also

Other Plotting functions: `CIS_volcano_plot()`, `HSC_population_plot()`, `circos_genomic_density()`, `fisher_scatterplot()`, `sharing_heatmap()`, `sharing_venn()`, `top_abund_tableGrob()`

Examples

```
data("integration_matrices", package = "ISAnalytics")
data("association_file", package = "ISAnalytics")
aggreg <- aggregate_values_by_key(
    x = integration_matrices,
    association_file = association_file,
    value_cols = c("seqCount", "fragmentEstimate")
)
abund <- compute_abundance(x = aggreg)
alluvial_plots <- integration_alluvial_plot(abund,
    alluvia_plot_y_threshold = 0.5
)
ex_plot <- alluvial_plots[[1]]$plot +
    ggplot2::labs(
        title = "IS distribution over time",
        subtitle = "Patient 1, MNC BM",
        y = "Abundance (%)",
        x = "Time point (days after GT)"
    )
print(ex_plot)
```

integration_matrices *Example of imported multi-quantification integration matrices.*

Description

The data was obtained manually by simulating real research data.

Usage

```
data("integration_matrices")
```

Format

Data frame with 1689 rows and 8 columns

chr The chromosome number (as character)

integration_locus Number of the base at which the viral insertion occurred

strand Strand of the integration

GeneName Symbol of the closest gene

GeneStrand Strand of the closest gene

CompleteAmplificationID Unique sample identifier

seqCount Value of the sequence count quantification

fragmentEstimate Value of the fragment estimate quantification

iss_source *Find the source of IS by evaluating sharing.*

Description

[Stable] The function computes the sharing between a reference group of interest for each time point and a selection of groups of interest. In this way it is possible to observe the percentage of shared integration sites between reference and each group and identify in which time point a certain IS was observed for the first time.

Usage

```
iss_source(
  reference,
  selection,
  ref_group_key = c("SubjectID", "CellMarker", "Tissue", "TimePoint"),
  selection_group_key = c("SubjectID", "CellMarker", "Tissue", "TimePoint"),
  timepoint_column = "TimePoint",
  by_subject = TRUE,
  subject_column = "SubjectID"
)
```

Arguments

- `reference` A data frame containing one or more groups of reference. Groups are identified by `ref_group_key`
- `selection` A data frame containing one or more groups of interest to compare. Groups are identified by `selection_group_key`
- `ref_group_key` Character vector of column names that identify a unique group in the `reference` data frame
- `selection_group_key` Character vector of column names that identify a unique group in the `selection` data frame
- `timepoint_column` Name of the column holding time point info?
- `by_subject` Should calculations be performed for each subject separately?
- `subject_column` Name of the column holding subjects information. Relevant only if `by_subject = TRUE`

Value

A list of data frames or a data frame

See Also

Other Analysis functions: `CIS_grubbs()`, `HSC_population_size_estimate()`, `compute_abundance()`, `cumulative_is()`, `gene_frequency_fisher()`, `is_sharing()`, `sample_statistics()`, `top_integrations()`, `top_targeted_genes()`

Examples

```
data("integration_matrices", package = "ISAnalytics")
data("association_file", package = "ISAnalytics")
aggreg <- aggregate_values_by_key(
    x = integration_matrices,
    association_file = association_file,
    value_cols = c("seqCount", "fragmentEstimate")
)
df1 <- aggreg %>%
    dplyr::filter(.data$Tissue == "BM")
df2 <- aggreg %>%
    dplyr::filter(.data$Tissue == "PB")
source <- iss_source(df1, df2)
source
ggplot2::ggplot(source$PT001, ggplot2::aes(
    x = as.factor(g2_TimePoint),
    y = sharing_perc, fill = g1
)) +
    ggplot2::geom_col() +
    ggplot2::labs(
        x = "Time point", y = "Shared IS % with MNC BM",
        title = "Source of is MNC BM vs MNC PB"
    )
```

is_sharing *Sharing of integration sites between given groups.*

Description

[Stable] Computes the amount of integration sites shared between the groups identified in the input data.

Usage

```
is_sharing(
  ...,
  group_key = c("SubjectID", "CellMarker", "Tissue", "TimePoint"),
  group_keys = NULL,
  n_comp = 2,
  is_count = TRUE,
  relative_is_sharing = TRUE,
  minimal = TRUE,
  include_self_comp = FALSE,
  keep_genomic_coord = FALSE,
  table_for_venn = FALSE
)
```

Arguments

`...` One or more integration matrices

`group_key` Character vector of column names which identify a single group. An associated group id will be derived by concatenating the values of these fields, separated by "_"

`group_keys` A list of keys for asymmetric grouping. If not NULL the argument `group_key` is ignored

`n_comp` Number of comparisons to compute. This argument is relevant only if provided a single data frame and a single key.

`is_count` Logical, if `TRUE` returns also the count of IS for each group and the count for the union set

`relative_is_sharing` Logical, if `TRUE` also returns the relative sharing.

`minimal` Compute only combinations instead of all possible permutations? If `TRUE` saves time and excludes redundant comparisons.

`include_self_comp` Include comparisons with the same group?

`keep_genomic_coord` If `TRUE` keeps the genomic coordinates of the shared integration sites in a dedicated column (as a nested table)

`table_for_venn` Add column with truth tables for venn plots?

Details

An integration site is always identified by the combination of fields in `mandatory_IS_vars()`, thus these columns must be present in the input(s).

The function accepts multiple inputs for different scenarios, please refer to the vignette `vignette("sharing_analyses", package = "ISAnalytics")` for a more in-depth explanation.

Output:

The function outputs a single data frame containing all requested comparisons and optionally individual group counts, genomic coordinates of the shared integration sites and truth tables for plotting venn diagrams.

Plotting sharing:

The sharing data obtained can be easily plotted in a heatmap via the function `sharing_heatmap` or via the function `sharing_venn`

Value

A data frame

Required tags

The function will explicitly check for the presence of these tags:

- All columns declared in `mandatory_IS_vars()`

See Also

Other Analysis functions: `CIS_grubbs()`, `HSC_population_size_estimate()`, `compute_abundance()`, `cumulative_is()`, `gene_frequency_fisher()`, `iss_source()`, `sample_statistics()`, `top_integrations()`, `top_targeted_genes()`

Examples

```
data("integration_matrices", package = "ISAnalytics")
data("association_file", package = "ISAnalytics")
aggreg <- aggregate_values_by_key(
    x = integration_matrices,
    association_file = association_file,
    value_cols = c("seqCount", "fragmentEstimate")
)
sharing <- is_sharing(aggreg)
sharing
```

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

Current dynamic vars specifications getters.

Description

Fetches the look-up tables for different categories of dynamic vars. For more details, refer to the dedicated vignette `vignette("workflow_start", package="ISAnalytics")`.

- `mandatory_IS_vars` returns the look-up table of variables that are used to uniquely identify integration events
- `annotation_IS_vars()` returns the look-up table of variables that contain genomic annotations
- `association_file_columns()` returns the look-up table of variables that contains information on how metadata is structured
- `iss_stats_specs()` returns the look-up table of variables that contains information on the format of pool statistics files produced automatically by VISPA2
- `matrix_file_suffixes()` returns the look-up table of variables that contains all default file names for each quantification type and it is used by automated import functions

Usage

```
mandatory_IS_vars(include_types = FALSE)

annotation_IS_vars(include_types = FALSE)

association_file_columns(include_types = FALSE)

iss_stats_specs(include_types = FALSE)

matrix_file_suffixes()
```

Arguments

`include_types`	If set to `TRUE` returns both the names and the types associated, otherwise returns only a character vector of names

Value

A character vector or a data frame

See Also

Other dynamic vars: `inspect_tags()`, `pcr_id_column()`, `reset_mandatory_IS_vars()`, `set_mandatory_IS_vars()`, `set_matrix_file_suffixes()`

Examples

```
# Names only
mandatory_IS_vars()

# Names and types
mandatory_IS_vars(TRUE)

# Names only
annotation_IS_vars()

# Names and types
annotation_IS_vars(TRUE)

# Names only
association_file_columns()

# Names and types
association_file_columns(TRUE)

# Names only
iss_stats_specs()

# Names and types
iss_stats_specs(TRUE)
```

```
# Names only
matrix_file_suffixes()
```

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: `annotation_issues()`, `date_formats()`, `default_af_transform()`, `default_iss_file_prefixes()`, `quantification_types()`

Examples

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

NGSdataExplorer *Launch the shiny application NGSdataExplorer.*

Description

Launch the shiny application NGSdataExplorer.

Usage

```
NGSdataExplorer()
```

Value

Nothing

Examples

```
## Not run:
NGSdataExplorer()

## End(Not run)
```

outliers_by_pool_fragments *Identify and flag outliers based on pool fragments.*

Description

[Stable] Identify and flag outliers based on expected number of raw reads per pool.

Usage

```
outliers_by_pool_fragments(
  metadata,
  key = "BARCODE_MUX",
  outlier_p_value_threshold = 0.01,
  normality_test = FALSE,
  normality_p_value_threshold = 0.05,
  transform_log2 = TRUE,
  per_pool_test = TRUE,
  pool_col = "PoolID",
  min_samples_per_pool = 5,
  flag_logic = "AND",
  keep_calc_cols = TRUE,
  report_path = default_report_path()
)
```

Arguments

`metadata` The metadata data frame

`key` A character vector of numeric column names

`outlier_p_value_threshold` The p value threshold for a read to be considered an outlier

`normality_test` Perform normality test? Normality is assessed for each column in the key using Shapiro-Wilk test and if the values do not follow a normal distribution, other calculations are skipped

`normality_p_value_threshold` Normality threshold

`transform_log2` Perform a log2 trasformation on values prior the actual calculations?

`per_pool_test` Perform the test for each pool?

`pool_col` A character vector of the names of the columns that uniquely identify a pool

`min_samples_per_pool` The minimum number of samples that a pool needs to contain in order to be processed - relevant only if `per_pool_test = TRUE`

`flag_logic` A character vector of logic operators to obtain a global flag formula - only relevant if the key is longer than one. All operators must be chosen between: AND, OR, XOR, NAND, NOR, XNOR

`keep_calc_cols` Keep the calculation columns in the output data frame?

`report_path` The path where the report file should be saved. Can be a folder, a file or NULL if no report should be produced. Defaults to `{user_home}/ISAnalytics_reports`.

Details

Modular structure:

The outlier filtering functions are structured in a modular fashion. There are 2 kind of functions:

- Outlier tests - Functions that perform some kind of calculation based on inputs and flags metadata
- Outlier filter - A function that takes one or more outlier tests, combines all the flags with a given logic and filters out rows that are flagged as outliers

This function is an outlier test, and calculates for each column in the key

- The zscore of the values
- The tstudent of the values
- The the associated p-value (tdist)

Optionally the test can be performed for each pool and a normality test can be run prior the actual calculations. Samples are flagged if this condition is respected:

- tdist < outlier_p_value_threshold & zscore < 0

If the key contains more than one column an additional flag logic can be specified for combining the results. Example: let's suppose the key contains the names of two columns, X and Y `key = c("X", "Y")` if we specify the the argument `flag_logic = "AND"` then the reads will be flagged based on this global condition: (tdist_X < outlier_p_value_threshold & zscore_X < 0) AND (tdist_Y < outlier_p_value_threshold & zscore_Y < 0)

The user can specify one or more logical operators that will be applied in sequence.

Value

A data frame of metadata with the column `to_remove`

See Also

Other Data cleaning and pre-processing: `aggregate_metadata()`, `aggregate_values_by_key()`, `compute_near_integrations()`, `default_meta_agg()`, `outlier_filter()`, `purity_filter()`, `realign_after_collisions()`, `remove_collisions()`, `threshold_filter()`

Examples

```
data("association_file", package = "ISAnalytics")
flagged <- outliers_by_pool_fragments(association_file,
    report_path = NULL
)
head(flagged)
```

`outlier_filter` *Filter out outliers in metadata, identified by the chosen outlier test.*

Description

[Experimental] Filter out outliers in metadata by using appropriate outlier tests.

Usage

```
outlier_filter(
  metadata,
  pcr_id_col = pcr_id_column(),
  outlier_test = c(outliers_by_pool_fragments),
  outlier_test_outputs = NULL,
  combination_logic = c("AND"),
  negate = FALSE,
  report_path = default_report_path(),
  ...
)
```

Arguments

`metadata` The metadata data frame

`pcr_id_col` The name of the pcr identifier column

`outlier_test` One or more outlier tests. Must be functions, either from `available_outlier_tests()` or custom functions that produce an appropriate output format (see details).

`outlier_test_outputs`
`NULL`, a data frame or a list of data frames. See details.

`combination_logic`	One or more logical operators ("AND", "OR", "XOR", "NAND", "NOR", "XNOR"). See datails.
`negate`	If `TRUE` will return only the metadata that was flagged to be removed. If `FALSE` will return only the metadata that wasn't flagged to be removed.
`report_path`	The path where the report file should be saved. Can be a folder or `NULL` if no report should be produced. Defaults to `{user_home}/ISAnalytics_reports`.
`...`	Additional named arguments passed to `outliers_test`

Details

Modular structure:

The outlier filtering functions are structured in a modular fashion. There are 2 kind of functions:

- Outlier tests - Functions that perform some kind of calculation based on inputs and flags metadata
- Outlier filter - A function that takes one or more outlier tests, combines all the flags with a given logic and filters out rows that are flagged as outliers

This function acts as the filter. It can either take one or more outlier tests as functions and call them through the argument `outlier_test`, or it can take directly outputs produced by individual tests in the argument `outlier_test_outputs` - if both are provided the second one has priority. The second method offers a bit more freedom, since single tests can be run independently and intermediate results saved and examined more in detail. If more than one test is to be performed, the argument `combination_logic` tells the function how to combine the flags: you can specify 1 logical operator or more than 1, provided it is compatible with the number of tests.

Writing custom outlier tests:

You have the freedom to provide your own functions as outlier tests. For this purpose, functions provided must respect this guidelines:

- Must take as input the whole metadata df
- Must return a df containing AT LEAST the `pcr_id_col` and a logical column `"to_remove"` that contains the flag
- The `pcr_id_col` must contain all the values originally present in the metadata df

Value

A data frame of metadata which has less or the same amount of rows

See Also

Other Data cleaning and pre-processing: `aggregate_metadata()`, `aggregate_values_by_key()`, `compute_near_integrations()`, `default_meta_agg()`, `outliers_by_pool_fragments()`, `purity_filter()`, `realign_after_collisions()`, `remove_collisions()`, `threshold_filter()`

Examples

```
data("association_file", package = "ISAnalytics")
filtered_af <- outlier_filter(association_file,
    key = "BARCODE_MUX",
    report_path = NULL
)
head(filtered_af)
```

---

pcr_id_column *Easily retrieve the name of the pcr id column.*

Description

The function is a shortcut to retrieve the currently set pcr id column name from the association file column tags look-up table. This column is needed every time a joining operation with metadata needs to be performed

Usage

```
pcr_id_column()
```

Value

The name of the column

See Also

Other dynamic vars: `inspect_tags`(), `mandatory_IS_vars`(), `reset_mandatory_IS_vars`(), `set_mandatory_IS_vars`(), `set_matrix_file_suffixes`()

Examples

```
pcr_id_column()
```

---

proto_oncogenes *Data frames for proto-oncogenes (human and mouse) amd tumor-suppressor genes from UniProt.*

Description

The file is simply a result of a research with the keywords "proto-oncogenes" and "tumor suppressor" for the target genomes on UniProt database.

Usage

```
data("proto_oncogenes")

data("tumor_suppressors")
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 569 rows and 13 columns.

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 523 rows and 13 columns.

Functions

- `tumor_suppressors`: Data frame for tumor suppressor genes

`purity_filter` *Filter integration sites based on purity.*

Description

[Stable] Filter that targets possible contamination between cell lines based on a numeric quantification (likely abundance or sequence count).

Usage

```
purity_filter(
  x,
  lineages = blood_lineages_default(),
  aggregation_key = c("SubjectID", "CellMarker", "Tissue", "TimePoint"),
  group_key = c("CellMarker", "Tissue"),
  selected_groups = NULL,
  join_on = "CellMarker",
  min_value = 3,
  impurity_threshold = 10,
  by_timepoint = TRUE,
  timepoint_column = "TimePoint",
  value_column = "seqCount_sum"
)
```

Arguments

`x`	An aggregated integration matrix, obtained via `aggregate_values_by_key()`
`lineages`	A data frame containing cell lineages information
`aggregation_key`	The key used for aggregating `x`
`group_key`	A character vector of column names for re-aggregation. Column names must be either in `x` or in `lineages`. See details.

`selected_groups`
Either NULL, a character vector or a data frame for group selection. See details.

`join_on` Common columns to perform a join operation on

`min_value` A minimum value to filter the input matrix. Integrations with a value strictly lower than `min_value` are excluded (dropped) from the output.

`impurity_threshold`
The ratio threshold for impurity in groups

`by_timepoint` Should filtering be applied on each time point? If `FALSE`, all time points are merged together

`timepoint_column`
Column in `x` containing the time point

`value_column` Column in `x` containing the numeric quantification of interest

Details

Setting input arguments:

The input matrix can be re-aggregated with the provided `group_key` argument. This key contains the names of the columns to group on (besides the columns holding genomic coordinates of the integration sites) and must be contained in at least one of `x` or `lineages` data frames. If the key is not found only in `x`, then a join operation with the `lineages` data frame is performed on the common column(s) `join_on`.

Group selection:

It is possible for the user to specify on which groups the logic of the filter should be applied to. For example: if we have `group_key = c("HematoLineage")` and we set `selected_groups = c("CD34", "Myeloid","Lymphoid")` it means that a single integration will be evaluated for the filter only for groups that have the values of "CD34", "Myeloid" and "Lymphoid" in the "HematoLineage" column. If the same integration is present in other groups it is kept as it is. `selected_groups` can be set to `NULL` if we want the logic to apply to every group present in the data frame, it can be set as a simple character vector as the example above if the group key has length 1 (and there is no need to filter on time point). If the group key is longer than 1 then the filter is applied only on the first element of the key.

If a more refined selection on groups is needed, a data frame can be provided instead:

```
group_key = c("CellMarker", "Tissue")
selected_groups = tibble::tribble(
~ CellMarker, ~ Tissue,
"CD34", "BM",
"CD14", "BM",
"CD14", "PB"
)
```

Columns in the data frame should be the same as group key (plus, eventually, the time point column). In this example only those groups identified by the rows in the provided data frame are processed.

Value

A data frame

See Also

Other Data cleaning and pre-processing: aggregate_metadata(), aggregate_values_by_key(), compute_near_integrations(), default_meta_agg(), outlier_filter(), outliers_by_pool_fragments(), realign_after_collisions(), remove_collisions(), threshold_filter()

Examples

```
data("integration_matrices", package = "ISAnalytics")
data("association_file", package = "ISAnalytics")
aggreg <- aggregate_values_by_key(
    x = integration_matrices,
    association_file = association_file,
    value_cols = c("seqCount", "fragmentEstimate")
)
filtered_by_purity <- purity_filter(
    x = aggreg,
    value_column = "seqCount_sum"
)
head(filtered_by_purity)
```

---

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_interact 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: annotation_issues(), date_formats(), default_af_transform(), default_iss_file_prefixes(), 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

[Stable] 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 remove_collisions(), 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,
  sample_column = pcr_id_column()
)
```

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".
sample_column	The name of the column containing the sample identifier

Details

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

Value

A named list with re-aligned matrices

See Also

remove_collisions

Other Data cleaning and pre-processing: aggregate_metadata(), aggregate_values_by_key(), compute_near_integrations(), default_meta_agg(), outlier_filter(), outliers_by_pool_fragments(), purity_filter(), remove_collisions(), threshold_filter()

Examples

```
data("integration_matrices", package = "ISAnalytics")
data("association_file", package = "ISAnalytics")
separated <- separate_quant_matrices(
    integration_matrices
)
no_coll <- remove_collisions(
    x = separated$seqCount,
    association_file = association_file,
    quant_cols = c(seqCount = "Value"),
    report_path = NULL
)
realigned <- realign_after_collisions(
    sc_matrix = no_coll,
    other_matrices = list(fragmentEstimate = separated$fragmentEstimate)
)
realigned
```

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()
```

`refGenes_hg19` *Gene annotation files for hg19, mm9 and mm10.*

Description

This file was obtained following this steps:

1. Download from http://hgdownload.soe.ucsc.edu/goldenPath/hg19/database/ the refGene.sql, knownGene.sql, knownToRefSeq.sql, kgXref.sql tables
2. Import everything it in mysql
3. Generate views for annotation:

```
SELECT kg.`chrom`, min(kg.cdsStart) as CDS_minStart,
max(kg.`cdsEnd`) as CDS_maxEnd, k2a.geneSymbol,
kg.`strand` as GeneStrand, min(kg.txStart) as TSS_minStart,
max(kg.txEnd) as TSS_maxStart,
kg.proteinID as ProteinID, k2a.protAcc as ProteinAcc, k2a.spDisplayID
FROM `knownGene` AS kg JOIN kgXref AS k2a
ON BINARY kg.name = k2a.kgID COLLATE latin1_bin
-- latin1_swedish_ci
-- WHERE k2a.spDisplayID IS NOT NULL and (k2a.`geneSymbol` LIKE 'Tcra%' or
k2a.`geneSymbol` LIKE 'TCRA%')
WHERE (k2a.spDisplayID IS NOT NULL or k2a.spDisplayID NOT LIKE '')
and k2a.`geneSymbol` LIKE 'Tcra%'
group by kg.`chrom`, k2a.geneSymbol
ORDER BY kg.chrom ASC , kg.txStart ASC
```

Usage

```
data("refGenes_hg19")

data("refGenes_mm9")
```

Format

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 27275 rows and 12 columns.

An object of class `tbl_df` (inherits from `tbl`, `data.frame`) with 24487 rows and 12 columns.

Functions

- `refGenes_mm9`: Data frame for murine mm9 genome

refGene_table_cols *Required columns for refGene file.*

Description

Required columns for refGene file.

Usage

```
refGene_table_cols()
```

Value

Character vector of column names

Examples

```
refGene_table_cols()
```

remove_collisions *Identifies and removes collisions.*

Description

[Stable] A collision is an integration (aka a unique combination of the provided `mandatory_IS_vars()`) which is observed in more than one independent sample. The function tries to decide to which independent sample should an integration event be assigned to, and if no decision can be taken, the integration is completely removed from the data frame. For more details refer to the vignette "Collision removal functionality": `vignette("workflow_start", package = "ISAnalytics")`

Usage

```
remove_collisions(
  x,
  association_file,
  independent_sample_id = c("ProjectID", "SubjectID"),
  date_col = "SequencingDate",
  reads_ratio = 10,
  quant_cols = c(seqCount = "seqCount", fragmentEstimate = "fragmentEstimate"),
  report_path = default_report_path(),
  max_workers = NULL
)
```

Arguments

`x` Either a multi-quantification matrix (recommended) or a named list of matrices (names must be quantification types)

`association_file` The association file imported via `import_association_file()`

`independent_sample_id` A character vector of column names that identify independent samples

`date_col` The date column that should be considered.

`reads_ratio` A single numeric value that represents the ratio that has to be considered when deciding between `seqCount` value.

`quant_cols` A named character vector where names are quantification types and values are the names of the corresponding columns. The quantification `seqCount` MUST be included in the vector.

`report_path` The path where the report file should be saved. Can be a folder or `NULL` if no report should be produced. Defaults to `{user_home}/ISAnalytics_reports`.

`max_workers` Maximum number of parallel workers to distribute the workload. If `NULL` (default) produces the maximum amount of workers allowed, a numeric value is requested otherwise. WARNING: a higher number of workers speeds up computation at the cost of memory consumption! Tune this parameter accordingly.

Value

Either a multi-quantification matrix or a list of data frames

Required tags

The function will explicitly check for the presence of these tags:

- project_id
- pool_id
- pcr_replicate

See Also

Other Data cleaning and pre-processing: `aggregate_metadata()`, `aggregate_values_by_key()`, `compute_near_integrations()`, `default_meta_agg()`, `outlier_filter()`, `outliers_by_pool_fragments()`, `purity_filter()`, `realign_after_collisions()`, `threshold_filter()`

Examples

```
data("integration_matrices", package = "ISAnalytics")
data("association_file", package = "ISAnalytics")
no_coll <- remove_collisions(
    x = integration_matrices,
    association_file = association_file,
    report_path = NULL
)
head(no_coll)
```

reset_mandatory_IS_vars *Resets dynamic vars to the default values.*

Description

Reverts all changes to dynamic vars to the default values. For more details, refer to the dedicated vignette `vignette("workflow_start", package="ISAnalytics")`.

- `reset_mandatory_IS_vars()` re-sets the look-up table for mandatory IS vars.
- `reset_annotation_IS_vars()` re-sets the look-up table for genomic annotation IS vars.
- `reset_af_columns_def()` re-sets the look-up table for association file columns vars
- `reset_iss_stats_specs()` re-sets the look-up table for VISPA2 pool statistics vars
- `reset_matrix_file_suffixes()` re-sets the matrix file suffixes look-up table
- `reset_dyn_vars_config()` re-sets all look-up tables

Usage

```
reset_mandatory_IS_vars()

reset_annotation_IS_vars()

reset_af_columns_def()

reset_iss_stats_specs()

reset_matrix_file_suffixes()

reset_dyn_vars_config()
```

Value

`NULL`

See Also

Other dynamic vars: `inspect_tags()`, `mandatory_IS_vars()`, `pcr_id_column()`, `set_mandatory_IS_vars()`, `set_matrix_file_suffixes()`

Examples

```
reset_mandatory_IS_vars()

reset_annotation_IS_vars()

reset_af_columns_def()

reset_iss_stats_specs()

reset_matrix_file_suffixes()

reset_dyn_vars_config()
```

---

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

Description

[Stable] 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:

```
data("integration_matrices", package = "ISAnalytics")
data("association_file", package = "ISAnalytics")
aggreg <- aggregate_values_by_key(
 x = integration_matrices,
 association_file = association_file,
 value_cols = c("seqCount", "fragmentEstimate")
)
aggreg_meta <- aggregate_metadata(association_file = association_file)

sample_stats <- sample_statistics(x = aggreg,
metadata = aggreg_meta,
value_columns = c("seqCount", "fragmentEstimate"),
sample_key = c("SubjectID", "CellMarker","Tissue", "TimePoint"))
```

Usage

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

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

add_integrations_count — Add the count of distinct integration sites for each group? Can be computed only if x contains the mandatory columns mandatory_IS_vars()

Value

A list with modified x and metadata data frames

Required tags

The function will explicitly check for the presence of these tags:

- All columns declared in mandatory_IS_vars()

These are checked only if add_integrations_count = TRUE.

See Also

Other Analysis functions: CIS_grubbs(), HSC_population_size_estimate(), compute_abundance(), cumulative_is(), gene_frequency_fisher(), is_sharing(), iss_source(), top_integrations(), top_targeted_genes()

Examples

```
data("integration_matrices", package = "ISAnalytics")
data("association_file", package = "ISAnalytics")
stats <- sample_statistics(
    x = integration_matrices,
    metadata = association_file,
    value_columns = c("seqCount", "fragmentEstimate")
)
stats
```

separate_quant_matrices

Separate a multiple-quantification matrix into single quantification matrices.

Description

[Stable] 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",
  key = c(mandatory_IS_vars(), annotation_IS_vars(), "CompleteAmplificationID")
)
```

Arguments

`x`	Single integration matrix with multiple quantification value columns, 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
`key`	Key columns to perform the joining operation

Value

A named list of data frames, where names are quantification types

See Also

quantification_types

Other Utilities: `as_sparse_matrix()`, `comparison_matrix()`, `generate_Vispa2_launch_AF()`, `generate_blank_association_file()`, `generate_default_folder_structure()`, `transform_columns()`

Examples

```
data("integration_matrices", package = "ISAnalytics")
separated <- separate_quant_matrices(
    integration_matrices
)
```

---

set_mandatory_IS_vars *Define custom dynamic vars.*

Description

This set of function allows users to specify custom look-up tables for dynamic variables. For more details, refer to the dedicated vignette `vignette("workflow_start", package="ISAnalytics")`.

- `set_mandatory_IS_vars()` sets the look-up table for mandatory IS vars.
- `set_annotation_IS_vars()` sets the look-up table for genomic annotation IS vars.
- `set_af_columns_def()` sets the look-up table for association file columns vars
- `set_iss_stats_specs()` sets the look-up table for VISPA2 pool statistics vars

Usage

```
set_mandatory_IS_vars(specs)

set_annotation_IS_vars(specs)

set_af_columns_def(specs)

set_iss_stats_specs(specs)
```

Arguments

`specs`	Either a named vector or a data frame with specific format. See details.

Details

The user can supply specifications in the form of a named vector or a data frame.

Named vector:

When using a named vector, names should be the names of the columns, values should be the type associated with each column in the form of a string. The vector gets automatically converted into a data frame with the right format (default values for the columns `transform` and `flag` are `NULL` and `required` respectively). Use of this method is however discouraged: data frame inputs are preferred since they offer more control.

Look-up table structure:

The look-up table for dynamic vars should always follow this structure:

names	types	transform	flag	tag
`<name of the column>`	`<type>`	`<a lambda or NULL>`	`<flag>`	`<tag>`

where

- `names` contains the name of the column as a character
- `types` contains the type of the column. Type should be expressed as a string and should be in one of the allowed types
- `char` for character (strings)
- `int` for integers
- `logi` for logical values (TRUE / FALSE)
- `numeric` for numeric values
- `factor` for factors
- `date` for generic date format - note that functions that need to read and parse files will try to guess the format and parsing may fail
- One of the accepted date/datetime formats by `lubridate`, you can use `ISAnalytics::date_formats()` to view the accepted formats
- `transform`: a purrr-style lambda that is applied immediately after importing. This is useful to operate simple transformations like removing unwanted characters or rounding to a certain precision. Please note that these lambdas need to be functions that accept a vector as input and only operate a transformation, aka they output a vector of the same length as the input. For more complicated applications that may require the value of other columns, appropriate functions should be manually applied post-import.
- `flag`: as of now, it should be set either to `required` or `optional` - some functions internally check for only required tags presence and if those are missing from inputs they fail, signaling failure to the user
- `tag`: a specific tag expressed as a string

Column types::

Type should be expressed as a string and should be in one of the allowed types

- `char` for character (strings)
- `int` for integers
- `logi` for logical values (TRUE / FALSE)
- `numeric` for numeric values
- `factor` for factors
- `date` for generic date format - note that functions that need to read and parse files will try to guess the format and parsing may fail
- One of the accepted date/datetime formats by `lubridate`, you can use `ISAnalytics::date_formats()` to view the accepted formats

Value

`NULL`

See Also

Other dynamic vars: inspect_tags(), mandatory_IS_vars(), pcr_id_column(), reset_mandatory_IS_vars(), set_matrix_file_suffixes()

Examples

```
tmp_mand_vars <- tibble::tribble(
    ~names, ~types, ~transform, ~flag, ~tag,
    "chrom", "char", ~ stringr::str_replace_all(.x, "chr", ""), "required",
    "chromosome",
    "position", "int", NULL, "required", "locus",
    "strand", "char", NULL, "required", "is_strand",
    "gap", "int", NULL, "required", NA_character_,
    "junction", "int", NULL, "required", NA_character_
)
set_mandatory_IS_vars(tmp_mand_vars)
print(mandatory_IS_vars(TRUE))
reset_mandatory_IS_vars()

tmp_annot_vars <- tibble::tribble(
    ~names, ~types, ~transform, ~flag, ~tag,
    "gene", "char", NULL, "required",
    "gene_symbol",
    "gene_strand", "char", NULL, "required", "gene_strand"
)
print(annotation_IS_vars(TRUE))
reset_annotation_IS_vars()

temp_af_cols <- tibble::tribble(
    ~names, ~types, ~transform, ~flag, ~tag,
    "project", "char", NULL, "required",
    "project_id",
    "pcr_id", "char", NULL, "required", "pcr_repl_id",
    "subject", "char", NULL, "required", "subject"
)
set_af_columns_def(temp_af_cols)
print(association_file_columns(TRUE))
reset_af_columns_def()

tmp_iss_vars <- tibble::tribble(
    ~names, ~types, ~transform, ~flag, ~tag,
    "pool", "char", NULL, "required",
    "vispa_concatenate",
    "tag", "char", NULL, "required", "tag_seq",
    "barcode", "int", NULL, "required", NA_character_
)
set_iss_stats_specs(tmp_iss_vars)
iss_stats_specs(TRUE)
reset_iss_stats_specs()
```

---

set_matrix_file_suffixes *Sets the look-up table for matrix file suffixes.*

Description

The function automatically produces and sets a look-up table of matrix file suffixes based on user input.

Usage

```
set_matrix_file_suffixes(
  quantification_suffix = list(seqCount = "seqCount", fragmentEstimate =
   "fragmentEstimate", barcodeCount = "barcodeCount", cellCount = "cellCount", ShsCount
    = "ShsCount"),
  annotation_suffix = list(annotated = ".no0.annotated", not_annotated = ""),
  file_ext = "tsv.gz",
 glue_file_spec = "{quantification_suffix}_matrix{annotation_suffix}.{file_ext}"
)
```

Arguments

`quantification_suffix`
: A named list - names must be quantification types in `quantification_types()`, and values must be single strings, containing the associated suffix. Please note that ALL quantification types must be specified or the function will produce an error.

`annotation_suffix`
: A named list - names must be `annotated` and `not_annotated`, values must be single strings, containing the associated suffix. Please note that both names must be present in the list or the function will produce an error.

`file_ext`
: The file extension (e.g. `tsv`, `tsv.gz`)

`glue_file_spec`
: A string specifying the pattern used to form the entire suffix, as per `glue::glue()` requirements. The string should contain the reference to `quantification_suffix`, `annotation_suffix` and `file_ext`.

Value

`NULL`

See Also

Other dynamic vars: `inspect_tags()`, `mandatory_IS_vars()`, `pcr_id_column()`, `reset_mandatory_IS_vars()`, `set_mandatory_IS_vars()`

Examples

```
set_matrix_file_suffixes(
    quantification_suffix = list(
        seqCount = "sc",
        fragmentEstimate = "fe",
        barcodeCount = "barcodeCount",
        cellCount = "cellCount",
        ShsCount = "ShsCount"
    ),
    annotation_suffix = list(annotated = "annot", not_annotated = "")
)
matrix_file_suffixes()
reset_matrix_file_suffixes()
```

sharing_heatmap *Plot IS sharing heatmaps.*

Description

[Stable] Displays the IS sharing calculated via is_sharing as heatmaps.

Usage

```
sharing_heatmap(
  sharing_df,
  show_on_x = "g1",
  show_on_y = "g2",
  absolute_sharing_col = "shared",
  title_annot = NULL,
  plot_relative_sharing = TRUE,
  rel_sharing_col = c("on_g1", "on_union"),
  show_perc_symbol_rel = TRUE,
  interactive = FALSE
)
```

Arguments

sharing_df The data frame containing the IS sharing data

show_on_x Name of the column to plot on the x axis

show_on_y Name of the column to plot on the y axis

absolute_sharing_col
  Name of the column that contains the absolute values of IS sharing

title_annot Additional text to display in the title

plot_relative_sharing
  Logical. Compute heatmaps also for relative sharing?

`rel_sharing_col`
: Names of the columns to consider as relative sharing. The function is going to plot one heatmap per column in this argument.

`show_perc_symbol_rel`
: Logical. Only relevant if `plot_relative_sharing` is set to TRUE, should the percentage symbol be displayed in relative heatmaps?

`interactive`
: Logical. Requires the package plotly is required for this functionality. Returns the heatmaps as interactive HTML widgets.

Value

A list of plots or widgets

See Also

is_sharing

Other Plotting functions: `CIS_volcano_plot()`, `HSC_population_plot()`, `circos_genomic_density()`, `fisher_scatterplot()`, `integration_alluvial_plot()`, `sharing_venn()`, `top_abund_tableGrob()`

Examples

```
data("integration_matrices", package = "ISAnalytics")
data("association_file", package = "ISAnalytics")
aggreg <- aggregate_values_by_key(
    x = integration_matrices,
    association_file = association_file,
    value_cols = c("seqCount", "fragmentEstimate")
)
sharing <- is_sharing(aggreg,
    minimal = FALSE,
    include_self_comp = TRUE
)
sharing_heatmaps <- sharing_heatmap(sharing_df = sharing)
sharing_heatmaps$absolute
sharing_heatmaps$on_g1
sharing_heatmaps$on_union
```

`sharing_venn` *Produce tables to plot sharing venn or euler diagrams.*

Description

[Stable] This function processes a sharing data frame obtained via `is_sharing()` with the option `table_for_venn = TRUE` to obtain a list of objects that can be plotted as venn or euler diagrams.

Usage

```
sharing_venn(sharing_df, row_range = NULL, euler = TRUE)
```

Arguments

sharing_df	The sharing data frame
row_range	Either NULL or a numeric vector of row indexes (e.g. c(1, 4, 5) will produce tables only for rows 1, 4 and 5)
euler	If TRUE will produce tables for euler diagrams, otherwise will produce tables for venn diagrams

Details

The functions requires the package eulerr. Each row of the input data frame is representable as a venn/euler diagram. The function allows to specify a range of row indexes to obtain a list of plottable objects all at once, leave it to NULL to process all rows.

To actually plot the data it is sufficient to call the function plot() and specify optional customization arguments. See eulerr docs for more detail on this.

Value

A list of data frames

See Also

Other Plotting functions: CIS_volcano_plot(), HSC_population_plot(), circos_genomic_density(), fisher_scatterplot(), integration_alluvial_plot(), sharing_heatmap(), top_abund_tableGrob()

Examples

```
data("integration_matrices", package = "ISAnalytics")
data("association_file", package = "ISAnalytics")
aggreg <- aggregate_values_by_key(
    x = integration_matrices,
    association_file = association_file,
    value_cols = c("seqCount", "fragmentEstimate")
)
sharing <- is_sharing(aggreg, n_comp = 3, table_for_venn = TRUE)
venn_tbls <- sharing_venn(sharing, row_range = 1:3, euler = FALSE)
venn_tbls
plot(venn_tbls[[1]])
```

---

threshold_filter *Filter data frames with custom predicates*

Description

[Stable] 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 × 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 × 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 × 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 × 4
##       a     b c         d
##   <dbl> <dbl> <chr> <int>
## 1    30    50 b         4
##
## $second
## # A tibble: 3 × 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 × 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 Data cleaning and pre-processing: `aggregate_metadata()`, `aggregate_values_by_key()`, `compute_near_integrations()`, `default_meta_agg()`, `outlier_filter()`, `outliers_by_pool_fragments()`, `purity_filter()`, `realign_after_collisions()`, `remove_collisions()`

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(">=")
    )
)
filtered
```

top_abund_tableGrob *Summary top abundant tableGrobs for plots.*

Description

Produce summary tableGrobs as R graphics. For this functionality the suggested package gridExtra is required. To visualize the resulting object:

```
gridExtra::grid.arrange(tableGrob)
```

Usage

```
top_abund_tableGrob(
  df,
  id_cols = mandatory_IS_vars(),
  quant_col = "fragmentEstimate_sum_PercAbundance",
  by = "TimePoint",
  alluvial_plot = NULL,
  top_n = 10,
  tbl_cols = "GeneName",
  include_id_cols = FALSE,
  digits = 2,
  perc_symbol = TRUE
)
```

Arguments

`df` A data frame

`id_cols` Character vector of id column names. To plot after alluvial, these columns must be the same as the `alluvia` argument of integration_alluvial_plot.

`quant_col` Column name holding the quantification value. To plot after alluvial, these columns must be the same as the `plot_y` argument of integration_alluvial_plot.

`by` The column name to subdivide tables for. The function will produce one table for each distinct value in `by`. To plot after alluvial, these columns must be the same as the `plot_x` argument of integration_alluvial_plot.

`alluvial_plot` Either NULL or an alluvial plot for color mapping between values of y.

`top_n` Integer. How many rows should the table contain at most?

`tbl_cols` Table columns to show in the final output besides `quant_col`.

`include_id_cols`
Logical. Include `id_cols` in the output?

`digits` Integer. Digits to show for the quantification column

`perc_symbol` Logical. Show percentage symbol in the quantification column?

Value

A tableGrob object

See Also

Other Plotting functions: `CIS_volcano_plot()`, `HSC_population_plot()`, `circos_genomic_density()`, `fisher_scatterplot()`, `integration_alluvial_plot()`, `sharing_heatmap()`, `sharing_venn()`

Examples

```
data("integration_matrices", package = "ISAnalytics")
data("association_file", package = "ISAnalytics")
aggreg <- aggregate_values_by_key(
    x = integration_matrices,
    association_file = association_file,
    value_cols = c("seqCount", "fragmentEstimate")
)
abund <- compute_abundance(x = aggreg)
grob <- top_abund_tableGrob(abund)
gridExtra::grid.arrange(grob)
```

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

Description

[Stable] 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 = 20,
  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.

Required tags

The function will explicitly check for the presence of these tags:

- All columns declared in `mandatory_IS_vars()`

See Also

Other Analysis functions: CIS_grubbs(), HSC_population_size_estimate(), compute_abundance(), cumulative_is(), gene_frequency_fisher(), is_sharing(), iss_source(), sample_statistics(), top_targeted_genes()

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"
)
```

top_targeted_genes *Top n targeted genes based on number of IS.*

Description

[Experimental] Produces a summary of the number of integration events per gene, orders the table in decreasing order and slices the first n rows - either on all the data frame or by group.

Usage

```
top_targeted_genes(
  x,
  n = 20,
  key = c("SubjectID", "CellMarker", "Tissue", "TimePoint"),
  consider_chr = TRUE,
  consider_gene_strand = TRUE,
  as_df = TRUE
)
```

Arguments

`x` An integration matrix - must be annotated

`n` Number of rows to slice

`key` If slice has to be performed for each group, the character vector of column names that identify the groups. If `NULL` considers the whole input data frame.

`consider_chr` Logical, should the chromosome be taken into account? See details.

`consider_gene_strand`
Logical, should the gene strand be taken into account? See details.

`as_df` If computation is performed by group, `TRUE` returns all groups merged in a single data frame with a column containing the group id. If `FALSE` returns a named list.

Details

Gene grouping:

When producing a summary of IS by gene, there are different options that can be chosen. The argument `consider_chr` accounts for the fact that some genes (same gene symbol) may span more than one chromosome: if set to `TRUE` counts of IS will be separated for those genes that span 2 or more chromosomes - in other words they will be in 2 different rows of the output table. On the contrary, if the argument is set to `FALSE`, counts will be produced in a single row.

NOTE: the function counts **DISTINCT** integration events, which logically corresponds to a union of sets. Be aware of the fact that counts per group and counts with different arguments might be different: if for example counts are performed by considering chromosome and there is one gene symbol with 2 different counts, the sum of those 2 will likely not be equal to the count obtained by performing the calculations without considering the chromosome.

The same reasoning can be applied for the argument `consider_gene_strand`, that takes into account the strand of the gene.

Value

A data frame or a list of data frames

Required tags

The function will explicitly check for the presence of these tags:

- chromosome
- locus
- gene_symbol
- gene_strand

Note that the tags "gene_strand" and "chromosome" are explicitly required only if `consider_chr = TRUE` and/or `consider_gene_strand = TRUE`.

See Also

Other Analysis functions: `CIS_grubbs()`, `HSC_population_size_estimate()`, `compute_abundance()`, `cumulative_is()`, `gene_frequency_fisher()`, `is_sharing()`, `iss_source()`, `sample_statistics()`, `top_integrations()`

Examples

```
data("integration_matrices", package = "ISAnalytics")
top_targ <- top_targeted_genes(
    integration_matrices,
    key = NULL
)
top_targ
```

transform_columns *Apply transformations to an arbitrary number of columns.*

Description

This function takes a named list of purr-style lambdas where names are the names of the columns in the data frame that must be transformed. NOTE: the columns are overridden, not appended.

Usage

```
transform_columns(df, transf_list)
```

Arguments

df	The data frame on which transformations should be operated
transf_list	A named list of purrr-style lambdas, where names are column names the function should be applied to.

Details

Lambdas provided in input must be transformations, aka functions that take in input a vector and return a vector of the same length as the input.

If the input transformation list contains column names that are not present in the input data frame, they are simply ignored.

Value

A data frame with transformed columns

See Also

Other Utilities: as_sparse_matrix(), comparison_matrix(), generate_Vispa2_launch_AF(), generate_blank_association_file(), generate_default_folder_structure(), separate_quant_matrices()

Examples

```
df <- tibble::tribble(
    ~A, ~B, ~C, ~D,
    1, 2, "a", "aa",
    3, 4, "b", "bb",
    5, 6, "c", "cc"
)
lambdas <- list(A = ~ .x + 1, B = ~ .x + 2, C = ~ stringr::str_to_upper(.x))
transform_columns(df, lambdas)
```

Index