

# Package ‘ISAnalytics’

April 10, 2023

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

**Version** 1.8.3

**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 tracking 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)

**Imports** utils, dplyr, readr, tidyr, purrr, rlang, tibble, stringr, fs, lubridate, lifecycle, ggplot2, ggrepel, stats, readxl, tools, grDevices, forcats, glue, shiny, shinyWidgets, datamods, bslib, DT

**Encoding** UTF-8

**LazyData** false

**Roxygen** list(markdown = TRUE)

**RoxygenNote** 7.2.3

**Suggests** testthat, covr, knitr, BiocStyle, sessioninfo, rmarkdown, roxygen2, vegan, withr, extraDistr, ggalluvial, scales, gridExtra, R.utils, RefManageR, flexdashboard, circlize, plotly, gtools, eulerr, openxlsx, jsonlite, pheatmap, BiocParallel, progressr, future, doFuture, foreach, psych, data.table, Rcapture

**VignetteBuilder** knitr

**RdMacros** lifecycle

**Config/testthat/edition** 3

**git\_url** <https://git.bioconductor.org/packages/ISAnalytics>

**git\_branch** RELEASE\_3\_16

**git\_last\_commit** 314ab5b

**git\_last\_commit\_date** 2023-04-03

**Date/Publication** 2023-04-10

**Author** Giulia Pais [aut, cre] (<<https://orcid.org/0009-0005-5621-4803>>),  
Andrea Calabria [aut],  
Giulio Spinozzi [aut]

**Maintainer** Giulia Pais <giuliapais1@gmail.com>

## R topics documented:

aggregate_metadata . . . . .	4
aggregate_values_by_key . . . . .	5
annotation_issues . . . . .	7
association_file . . . . .	7
as_sparse_matrix . . . . .	8
available_outlier_tests . . . . .	9
available_tags . . . . .	10
blood_lineages_default . . . . .	10
circos_genomic_density . . . . .	11
CIS_grubbs . . . . .	12
CIS_grubbs_overtime . . . . .	14
CIS_volcano_plot . . . . .	16
clinical_relevant_suspicious_genes . . . . .	18
comparison_matrix . . . . .	18
compute_abundance . . . . .	20
compute_near_integrations . . . . .	21
cumulative_is . . . . .	24
date_formats . . . . .	26
default_af_transform . . . . .	26
default_iss_file_prefixes . . . . .	27
default_meta_agg . . . . .	27
default_rec_agg_lambdas . . . . .	28
default_report_path . . . . .	29
default_stats . . . . .	29

enable_progress_bars . . . . .	30
export_ISA_settings . . . . .	30
fisher_scatterplot . . . . .	31
generate_blank_association_file . . . . .	33
generate_default_folder_structure . . . . .	33
generate_Vispa2_launch_AF . . . . .	34
gene_frequency_fisher . . . . .	36
HSC_population_plot . . . . .	38
HSC_population_size_estimate . . . . .	39
import_association_file . . . . .	42
import_ISA_settings . . . . .	45
import_parallel_Vispa2Matrices . . . . .	45
import_single_Vispa2Matrix . . . . .	47
import_Vispa2_stats . . . . .	49
inspect_tags . . . . .	50
integration_alluvial_plot . . . . .	51
integration_matrices . . . . .	53
iss_source . . . . .	54
is_sharing . . . . .	55
known_clinical_oncogenes . . . . .	57
mandatory_IS_vars . . . . .	58
matching_options . . . . .	59
NGSdataExplorer . . . . .	60
outliers_by_pool_fragments . . . . .	61
outlier_filter . . . . .	63
pcr_id_column . . . . .	64
proto_oncogenes . . . . .	65
purity_filter . . . . .	66
quantification_types . . . . .	68
realign_after_collisions . . . . .	69
reduced_AF_columns . . . . .	70
refGenes_hg19 . . . . .	71
refGene_table_cols . . . . .	72
remove_collisions . . . . .	72
reset_mandatory_IS_vars . . . . .	74
sample_statistics . . . . .	75
separate_quant_matrices . . . . .	77
set_mandatory_IS_vars . . . . .	78
set_matrix_file_suffixes . . . . .	81
sharing_heatmap . . . . .	82
sharing_venn . . . . .	83
top_abund_tableGrob . . . . .	84
top_cis_overtime_heatmap . . . . .	86
top_integrations . . . . .	90
top_targeted_genes . . . . .	91
transform_columns . . . . .	93

---

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

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

---

**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 <a href="#">comparison_matrix</a>
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\(\)](#), [enable\\_progressBars\(\)](#), [export\\_ISA\\_settings\(\)](#), [generate\\_Vispa2\\_launch\\_AF\(\)](#), [generate\\_blank\\_association\\_file\(\)](#), [generate\\_default\\_folder\\_structure\(\)](#), [import\\_ISA\\_settings\(\)](#), [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\(\)](#), [top\\_cis\\_overtime\\_heatmap\(\)](#)

**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
)
```

**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**

<code>x</code>	An integration matrix, must include the mandatory <code>mandatory_IS_vars()</code> columns and the <code>annotation_IS_vars()</code> columns
<code>genomic_annotation_file</code>	Database file for gene annotation, see details.
<code>grubbs_flanking_gene_bp</code>	Number of base pairs flanking a gene
<code>threshold_alpha</code>	Significance threshold
<code>by</code>	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 <code>by = "SubjectID"</code> , CIS will be computed for each distinct SubjectID found in the table ("SubjectID" column must be included in the input data frame).
<code>return_missing_as_df</code>	Returns those genes present in the input df but not in the <code>refgenes</code> as a data frame?
<code>results_as_list</code>	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 UCSC 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\_grubbs\_overtime     *Compute CIS and Grubbs test over different time points and groups.*

---

**Description**

**[Experimental]** Computes common insertion sites and Grubbs test for each separate group and separating different time points among the same group. The logic applied is the same as the function `CIS_grubbs()`.

**Usage**

```
CIS_grubbs_overtime(  
  x,  
  genomic_annotation_file = "hg19",  
  grubbs_flanking_gene_bp = 1e+05,  
  threshold_alpha = 0.05,  
  group = "SubjectID",  
  timepoint_col = "TimePoint",  
  as_df = TRUE,  
  return_missing_as_df = TRUE,  
  max_workers = NULL  
)
```

**Arguments**

<code>x</code>	An integration matrix, must include the mandatory <code>IS_vars()</code> columns and the <code>annotation_IS_vars()</code> columns
<code>genomic_annotation_file</code>	Database file for gene annotation, see details.
<code>grubbs_flanking_gene_bp</code>	Number of base pairs flanking a gene
<code>threshold_alpha</code>	Significance threshold
<code>group</code>	A character vector of column names that identifies a group. Each group must contain one or more time points.
<code>timepoint_col</code>	What is the name of the column containing time points?
<code>as_df</code>	Choose the result format: if TRUE the results are returned as a single data frame containing a column for the group id and a column for the time point, if FALSE results are returned in the form of nested lists (one table for each time point and for each group), if "group" results are returned as a list separated for each group but containing a single table with all time points.
<code>return_missing_as_df</code>	Returns those genes present in the input df but not in the refgenes as a data frame?
<code>max_workers</code>	Maximum number of parallel workers. If NULL the maximum number of workers is calculated automatically.

**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 UCSC 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 list with results and optionally missing genes info

**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_overtime <- CIS_grubbs_overtime(aggreg)
  cis_overtime

```

---

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 <a href="#">CIS_grubbs</a> with <code>add_standard_padjust = TRUE</code>
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\(\)](#), [top\\_cis\\_overtime\\_heatmap\(\)](#)

### 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\(\)](#), [enable\\_progressBars\(\)](#), [export\\_ISA\\_settings\(\)](#), [generate\\_Vispa2\\_launch\\_generate\\_blank\\_association\\_file\(\)](#), [generate\\_default\\_folder\\_structure\(\)](#), [import\\_ISA\\_settings\(\)](#), [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	<i>Computes the abundance for every integration event in the input data frame.</i>
-------------------	--

---

### 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: <ul style="list-style-type: none"> <li>• "max_value": keep the integration site which has the highest value (and collapse other values on that integration).</li> <li>• "keep_first": keeps the first integration</li> </ul>
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 where the file will be saved. Must be a folder. Relevant only if map_as_file is TRUE.
strand_specific	<b>[Deprecated]</b> 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 be 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 <code>aggregate_values_by_key()</code>
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	<i>Default transformations to apply to association file columns.</i>
----------------------	--

---

### 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	<i>Default folder for saving ISAnalytics reports. Supplied as default argument for several functions.</i>
---------------------	---

---

**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	<i>A set of pre-defined functions for sample_statistics.</i>
---------------	--

---

**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()
```

---

`enable_progressBars` *Enable global progress bars for ISAnalytics functions.*

---

### Description

This is a simple wrapper around functions from the package `progressr`. To customize the appearance of the progress bar, please refer to [progressr](#) documentation.

### Usage

```
enable_progressBars()
```

### Value

NULL

### See Also

Other Utilities: [as\\_sparse\\_matrix\(\)](#), [comparison\\_matrix\(\)](#), [export\\_ISA\\_settings\(\)](#), [generate\\_Vispa2\\_launch\\_AF\(\)](#), [generate\\_blank\\_association\\_file\(\)](#), [generate\\_default\\_folder\\_structure\(\)](#), [import\\_ISA\\_settings\(\)](#), [separate\\_quant\\_matrices\(\)](#), [transform\\_columns\(\)](#)

### Examples

```
enable_progressBars()
progressr::handlers(global = FALSE) # Deactivate
```

---

`export_ISA_settings` *Export a dynamic vars settings profile.*

---

### Description

This function allows exporting the currently set dynamic vars in json format so it can be quickly imported later. Dynamic variables need to be properly set via the setter functions before calling the function. For more details, refer to the dedicated vignette `vignette("workflow_start", package="ISAnalytics")`.

### Usage

```
export_ISA_settings(folder, setting_profile_name)
```

### Arguments

`folder` The path to the folder where the file should be saved. If the folder doesn't exist, it gets created automatically

`setting_profile_name` A name for the settings profile

**Value**

NULL

**See Also**

Other Utilities: [as\\_sparse\\_matrix\(\)](#), [comparison\\_matrix\(\)](#), [enable\\_progressBars\(\)](#), [generate\\_Vispa2\\_launch\\_AF\\_generate\\_blank\\_association\\_file\(\)](#), [generate\\_default\\_folder\\_structure\(\)](#), [import\\_ISA\\_settings\(\)](#), [separate\\_quant\\_matrices\(\)](#), [transform\\_columns\(\)](#)

**Examples**

```
tmp_folder <- tempdir()
export_ISA_settings(tmp_folder, "DEFAULT")
```

---

```
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**

<code>fisher_df</code>	Test results obtained via <code>gene_frequency_fisher()</code>
<code>p_value_col</code>	Name of the column containing the p-value to consider
<code>annot_threshold</code>	Annotate with a different color if a point is below the significance threshold. Single numerical value.
<code>annot_color</code>	The color in which points below the threshold should be annotated
<code>gene_sym_col</code>	The name of the column containing the gene symbol
<code>do_not_highlight</code>	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\(\)](#), [top\\_cis\\_overtime\\_heatmap\(\)](#)

## 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\(\)](#), [enable\\_progressBars\(\)](#), [export\\_ISA\\_settings\(\)](#), [generate\\_Vispa2\\_launch\\_AF\(\)](#), [generate\\_default\\_folder\\_structure\(\)](#), [import\\_ISA\\_settings\(\)](#), [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\(\)](#), [enable\\_progressBars\(\)](#), [export\\_ISA\\_settings\(\)](#), [generate\\_Vispa2\\_launch\\_AF\(\)](#), [generate\\_blank\\_association\\_file\(\)](#), [import\\_ISA\\_settings\(\)](#), [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 <code>import_association_file()</code> )
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 correctly 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\(\)](#), [enable\\_progressBars\(\)](#), [export\\_ISA\\_settings\(\)](#), [generate\\_blank\\_association\\_file\(\)](#), [generate\\_default\\_folder\\_structure\(\)](#), [import\\_ISA\\_settings\(\)](#), [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:

- `gene_symbol`

## 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 <a href="#">HSC_population_size_estimate</a>
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\(\)](#), [top\\_cis\\_overtime\\_heatmap\(\)](#)

**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 = aggregr,
  metadata = aggregr_meta,
  stable_timepoints = c(90, 180, 360),
  cell_type = "Other"
)
p <- HSC_population_plot(estimate$est, "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",
  max_workers = 4
)

```

**Arguments**

<code>x</code>	An aggregated integration matrix. See details.
<code>metadata</code>	An aggregated association file. See details.
<code>stable_timepoints</code>	A numeric vector or NULL if there are no stable time points. NOTE: the vector is NOT intended as a sequence min-max, every stable time point has to be specified individually
<code>aggregation_key</code>	A character vector indicating the key used for aggregating <code>x</code> and <code>metadata</code> . Note that <code>x</code> and <code>metadata</code> should always be aggregated with the same key.
<code>blood_lineages</code>	A data frame containing information on the blood lineages. Users can supply their own, provided the columns <code>CellMarker</code> and <code>CellType</code> are present.
<code>timepoint_column</code>	What is the name of the time point column to use? Note that this column must be present in the key.
<code>seqCount_column</code>	What is the name of the column in <code>x</code> containing the values of sequence count quantification?
<code>fragmentEstimate_column</code>	What is the name of the column in <code>x</code> containing the values of fragment estimate quantification? If fragment estimate is not present in the matrix, <code>param</code> should be set to NULL.
<code>seqCount_threshold</code>	A single numeric value. After re-aggregating <code>x</code> , rows with a value greater or equal will be kept, the others will be discarded.
<code>fragmentEstimate_threshold</code>	A single numeric value. Threshold value for fragment estimate, see details.
<code>nIS_threshold</code>	A single numeric value. If a group (row) in the <code>metadata</code> data frame has a count of distinct integration sites strictly greater than this number it will be kept, otherwise discarded.
<code>cell_type</code>	The cell types to include in the models. Note that the matching is case-insensitive.
<code>tissue_type</code>	The tissue types to include in the models. Note that the matching is case-insensitive.
<code>max_workers</code>	Maximum parallel workers allowed

**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`).



### Specifying more than one group

Groups for the estimates are computed as a pair of cell type and tissue. If the user wishes to compute estimates for more than one combination of cell type and tissue, it is possible to specify them as character vectors to the fields `cell_type` and `tissue_type` respectively, noting that:

- Vectors must have the same length or one of the 2 has to be of length 1
- It is a responsibility of the user to check whether the combination exists in the dataset provided.

Example:

```
estimate <- HSC_population_size_estimate(
  x = aggreg,
  metadata = aggreg_meta,
  cell_type = c("MYELOID", "T", "B"),
  tissue_type = "PB"
)

# Evaluated groups will be:
# - MYELOID PB
# - T PB
# - B PB
```

Note that estimates are computed individually for each group.

### 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. NOTE: the vector passed has to contain all individual time points, not just the minimum and maximum

### 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: <code>date_formats()</code>
separator	The column separator used in the file
filter_for	A named list where names represent column names that must be filtered. For example: <code>list(ProjectID = c("PROJECT1", "PROJECT2"))</code> 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 <code>{user_home}/ISAnalytics_reports</code> .
transformations	Either NULL or a named list of purrr-style lambdas where names are column names the function should be applied to.
tp_padding	<b>[Deprecated]</b> Deprecated. Use transformations instead.
...	Additional arguments to pass to <code>import_Vispa2_stats</code>

**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\\_Vispa2Matrices\(\)](#)

**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_ISA_settings`     *Import a dynamic vars settings profile.*

---

### Description

The function allows the import of an existing dynamic vars profile in json format. This is a quick and convenient way to set up the workflow, alternative to specifying lookup tables manually through the corresponding setter functions. For more details, refer to the dedicated vignette `vignette("workflow_start", package="ISAnalytics")`.

### Usage

```
import_ISA_settings(path)
```

### Arguments

`path`                    The path to the json file on disk

### Value

NULL

### See Also

Other Utilities: [as\\_sparse\\_matrix\(\)](#), [comparison\\_matrix\(\)](#), [enable\\_progressBars\(\)](#), [export\\_ISA\\_settings\(\)](#), [generate\\_Vispa2\\_launch\\_AF\(\)](#), [generate\\_blank\\_association\\_file\(\)](#), [generate\\_default\\_folder\\_structure\(\)](#), [separate\\_quant\\_matrices\(\)](#), [transform\\_columns\(\)](#)

### Examples

```
tmp_folder <- tempdir()
export_ISA_settings(tmp_folder, "DEFAULT")
import_ISA_settings(fs::path(tmp_folder, "DEFAULT_ISAsettings.json"))
reset_dyn_vars_config()
```

---

`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 = "AUTO",
  ...
)
```

**Arguments**

association_file	Data frame imported via <a href="#">import_association_file</a> (with file system alignment)
quantification_type	A vector of requested quantification_types. Possible choices are <a href="#">quantification_types</a>
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 <a href="#">comparison_matrix</a> 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	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	A single value between <a href="#">matching_options</a>
mode	Only AUTO is supported. As of ISAnalytics 1.8.3, the value INTERACTIVE is officially deprecated.
...	<dynamic-dots> Additional named arguments to pass to <a href="#">comparison_matrix</a> and <a href="#">import_single_Vispa2_matrix</a>

**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	<b>[Deprecated]</b> Deprecated. Use additional_cols instead
keep_excluded	<b>[Deprecated]</b> Deprecated. Use additional_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 frame 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**

<code>association_file</code>	The file system aligned association file (contains columns with absolute paths to the 'iss' folder)
<code>file_prefixes</code>	A character vector with known file prefixes to match on file names. NOTE: the elements represent regular expressions. For defaults see <a href="#">default_iss_file_prefixes</a> .
<code>join_with_af</code>	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.
<code>pool_col</code>	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.
<code>report_path</code>	The path where the report file should be saved. Can be a folder or NULL if no report should be produced. Defaults to <code>{user_home}/ISAnalytics_reports</code> .

**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  $\geq 1\%$ ". 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\(\)](#), [top\\_cis\\_overtime\\_heatmap\(\)](#)

**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 = aggre)
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 transformation 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	<i>Filter out outliers in metadata, identified by the chosen outlier test.</i>
----------------	--

---

## 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 per identifier column
outlier_test	One or more outlier tests. Must be functions, either from <code>available_outlier_tests()</code> 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 details.
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 <code>{user_home}/ISAnalytics_reports</code> .
...	Additional named arguments passed to <code>outliers_test</code>

## 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	<i>Data frames for proto-oncogenes (human and mouse) amd tumor-suppressor genes from UniProt.</i>
-----------------	---

---

**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_interac` 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	<i>Names of the columns of the association file to consider for Vispa2 launch.</i>
--------------------	--

---

**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 <code>import_association_file()</code>
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 <code>seqCount</code> value.
quant_cols	A named character vector where names are quantification types and values are the names of the corresponding columns. The quantification <code>seqCount</code> <b>MUST</b> be included in the vector.
report_path	The path where the report file should be saved. Can be a folder or <code>NULL</code> if no report should be produced. Defaults to <code>{user_home}/ISAnalytics_reports</code> .
max_workers	Maximum number of parallel workers to distribute the workload. If <code>NULL</code> (default) produces the maximum amount of workers allowed, a numeric value is requested otherwise. <b>WARNING:</b> 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	<i>Computes user specified functions on numerical columns and updates the metadata data frame accordingly.</i>
-------------------	--

---

**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

<code>x</code>	Single integration matrix with multiple quantification value columns, obtained via <a href="#">comparison_matrix</a> .
<code>fragmentEstimate</code>	Name of the fragment estimate values column in input
<code>seqCount</code>	Name of the sequence count values column in input
<code>barcodeCount</code>	Name of the barcode count values column in input
<code>cellCount</code>	Name of the cell count values column in input
<code>ShsCount</code>	Name of the shs count values column in input
<code>key</code>	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\(\)](#), [enable\\_progressBars\(\)](#), [export\\_ISA\\_settings\(\)](#), [generate\\_Vispa2\\_launch\\_AF\(\)](#), [generate\\_blank\\_association\\_file\(\)](#), [generate\\_default\\_folder\\_structure\(\)](#), [import\\_ISA\\_settings\(\)](#), [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	<i>Plot IS sharing heatmaps.</i>
-----------------	----------------------------------

---

**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 <code>plotly</code> 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\(\)](#), [top\\_cis\\_overtime\\_heatmap\(\)](#)

**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\(\)](#), [top\\_cis\\_overtime\\_heatmap\(\)](#)

**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]])
```

---

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,
  transform_by = NULL
)
```

**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 <a href="#">integration_alluvial_plot</a> .
quant_col	Column name holding the quantification value. To plot after alluvial, these columns must be the same as the plot_y argument of <a href="#">integration_alluvial_plot</a> .
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 <a href="#">integration_alluvial_plot</a> .
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?
transform_by	Either a function or a purrr-style lambda. This function is applied to the column by before separating columns. If NULL no function is applied. Useful to modify column order in final table.

**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\(\)](#), [top\\_cis\\_overtime\\_heatmap\(\)](#)

**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 = aggre)
grob <- top_abund_tableGrob(abund)
gridExtra::grid.arrange(grob)

# with transform
grob <- top_abund_tableGrob(abund, transform_by = ~ as.numeric(.x))

```

---

top\_cis\_overtime\_heatmap

*Heatmaps for the top N common insertion sites over time.*

---

**Description**

**[Experimental]** This function computes the visualization of the results of the function `CIS_grubbs_overtime()` in the form of heatmaps for the top N selected genes over time.

**Usage**

```

top_cis_overtime_heatmap(
  x,
  n_genes = 20,
  timepoint_col = "TimePoint",
  group_col = "group",
  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,
  plot_values = c("minus_log_p", "p"),
  p_value_correction = c("fdr", "bonferroni"),
  prune_tp_treshold = 20,
  gene_selection_param = c("trimmed", "n", "mean", "sd", "median", "mad", "min", "max"),
  fill_0_selection = TRUE,
  fill_NA_in_heatmap = FALSE,
  heatmap_color_palette = "default",
  title_generator = NULL,
  save_as_files = FALSE,
  files_format = c("pdf", "png", "tiff", "bmp", "jpg"),

```

```

    folder_path = NULL,
    ...
)

```

### Arguments

x	Output of the function CIS_grubbs_overtime(), either in single data frame form or nested lists
n_genes	Number of top genes to consider
timepoint_col	The name of the time point column in x
group_col	The name of the group column in x
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
plot_values	Which kind of values should be plotted? Can either be "p" for the p-value or "minus_log_p" for a scaled p-value of the Grubbs test
p_value_correction	One among "bonferroni" and "fdr"
prune_tp_treshold	Minimum number of genes to retain a time point. See details.
gene_selection_param	The descriptive statistic measure to decide which genes to plot, possible choices are "trimmed", "n", "mean", "sd", "median", "mad", "min", "max". See details.
fill_0_selection	Fill NA values with 0s before computing statistics for each gene? (TRUE/FALSE)
fill_NA_in_heatmap	Fill NA values with 0 when plotting the heatmap? (TRUE/FALSE)
heatmap_color_palette	Colors for values in the heatmaps, either "default" or a function producing a color palette, obtainable via grDevices::colorRampPalette.
title_generator	Either NULL or a function. See details.
save_as_files	Should heatmaps be saved to files on disk? (TRUE/FALSE)
files_format	The extension of the files produced, supported formats are "pdf", "png", "tiff", "bmp", "jpg". Relevant only if files_format = TRUE
folder_path	Path to the folder where files will be saved
...	Other params to pass to pheatmap::pheatmap

## 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()
```

### Top N gene selection:

Since the genes present in different time point slices are likely different, the decision process to select the final top N genes to represent in the heatmap follows this logic:

- Each time point slice is arranged either in ascending order (if we want to plot the p-value) or in descending order (if we want to plot the scaled p-value) and the top n genes are selected
- A series of statistics are computed over the union set of genes on ALL time points (min, max, mean, ...)
- A decision is taken by considering the ordered `gene_selection_param` (order depends once again if the values are scaled or not), and the first N genes are selected for plotting.

#### *Filling NA values prior calculations:*

It is possible to fill NA values (aka missing combinations of GENE/TP) with 0s prior computing the descriptive statistics on which gene selection is based. Please keep in mind that this has an impact on the final result, since for computing metrics such as the mean, NA values are usually removed, decreasing the overall number of values considered - this does not hold when NA values are substituted with 0s.

#### *The statistics:*

Statistics are computed for each gene over all time points of each group. More in detail, `n`: counts the number of instances (rows) in which the genes appears, aka it counts the time points in which the gene is present. NOTE: if `fill_0_selection` option is set to TRUE this value will be equal for all genes! All other statistics as per the argument `gene_selection_param` map to the corresponding R functions with the exception of `trimmed` which is a simple call to the mean function with the argument `trimmed = 0.1`.

### Aesthetics:

It is possible to customise the appearance of the plot through different parameters.

- `fill_NA_in_heatmap` tells the function whether missing combinations of GENE/TP should be plotted as NA or filled with a value (1 if p-value, 0 if scaled p-value)



- A title generator function can be provided to dynamically create a title for the plots: the function can accept two positional arguments for the group identifier and the number of selected genes respectively. If one or none of the arguments are of interest, they can be absorbed with ...
- heatmap\_color\_palette can be used to specify a function from which colors are sampled (refers to the colors of values only)
- To change the colors associated with annotations instead, use the argument annotation\_colors of pheatmap::pheatmap() - it must be set to a list with this format:

```
list(
  KnownGeneClass = c("OncoGene" = color_spec,
                    "Other" = color_spec,
                    "TumSuppressor" = color_spec),
  ClinicalRelevance = c("TRUE" = color_spec,
                       "FALSE" = color_spec),
  CriticalForInsMut = c("TRUE" = color_spec,
                       "FALSE" = color_spec)
)
```

### Value

Either a list of graphical objects or a list of paths where plots were saved

### See Also

Other Plotting functions: [CIS\\_volcano\\_plot\(\)](#), [HSC\\_population\\_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")
)
cis_overtime <- CIS_grubbs_overtime(aggreg)
hmaps <- top_cis_overtime_heatmap(cis_overtime$cis,
  fill_NA_in_heatmap = TRUE
)

# To re-plot:
# grid::grid.newpage()
# grid::grid.draw(hmaps$PT001$gtable)
```

---

top_integrations	<i>Sorts and keeps the top n integration sites based on the values in a given column.</i>
------------------	---

---

### 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	<i>Apply transformations to an arbitrary number of columns.</i>
-------------------	---

---

## 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\(\)](#), [enable\\_progressBars\(\)](#), [export\\_ISA\\_settings\(\)](#), [generate\\_Vispa2\\_launch\\_AF\(\)](#), [generate\\_blank\\_association\\_file\(\)](#), [generate\\_default\\_folder\\_structure\(\)](#), [import\\_ISA\\_settings\(\)](#), [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

- \* **Analysis functions helpers**
    - default\_stats, 29
  - \* **Analysis functions**
    - CIS\_grubbs, 12
    - compute\_abundance, 20
    - cumulative\_is, 24
    - gene\_frequency\_fisher, 36
    - HSC\_population\_size\_estimate, 39
    - is\_sharing, 55
    - iss\_source, 54
    - sample\_statistics, 75
    - top\_integrations, 90
    - top\_targeted\_genes, 91
  - \* **Data cleaning and pre-processing**
    - aggregate\_metadata, 4
    - aggregate\_values\_by\_key, 5
    - compute\_near\_integrations, 21
    - default\_meta\_agg, 27
    - outlier\_filter, 63
    - outliers\_by\_pool\_fragments, 61
    - purity\_filter, 66
    - realign\_after\_collisions, 69
    - remove\_collisions, 72
  - \* **Import functions helpers**
    - annotation\_issues, 7
    - date\_formats, 26
    - default\_af\_transform, 26
    - default\_iss\_file\_prefixes, 27
    - matching\_options, 59
    - quantification\_types, 68
  - \* **Import functions**
    - import\_association\_file, 42
    - import\_parallel\_Vispa2Matrices, 45
    - import\_single\_Vispa2Matrix, 47
    - import\_Vispa2\_stats, 49
  - \* **Outlier tests**
    - available\_outlier\_tests, 9
  - \* **Plotting function helpers**
    - clinical\_relevant\_suspicious\_genes, 18
    - known\_clinical\_oncogenes, 57
  - \* **Plotting functions**
    - circos\_genomic\_density, 11
    - CIS\_volcano\_plot, 16
    - fisher\_scatterplot, 31
    - HSC\_population\_plot, 38
    - integration\_alluvial\_plot, 51
    - sharing\_heatmap, 82
    - sharing\_venn, 83
    - top\_abund\_tableGrob, 84
    - top\_cis\_overtime\_heatmap, 86
  - \* **Utilities**
    - as\_sparse\_matrix, 8
    - comparison\_matrix, 18
    - enable\_progress\_bars, 30
    - export\_ISA\_settings, 30
    - generate\_blank\_association\_file, 33
    - generate\_default\_folder\_structure, 33
    - generate\_Vispa2\_launch\_AF, 34
    - import\_ISA\_settings, 45
    - separate\_quant\_matrices, 77
    - transform\_columns, 93
  - \* **datasets**
    - association\_file, 7
    - integration\_matrices, 53
    - proto\_oncogenes, 65
    - refGenes\_hg19, 71
  - \* **dynamic vars**
    - inspect\_tags, 50
    - mandatory\_IS\_vars, 58
    - pcr\_id\_column, 64
    - reset\_mandatory\_IS\_vars, 74
    - set\_mandatory\_IS\_vars, 78
    - set\_matrix\_file\_suffixes, 81
- aggregate\_metadata, 4, 6, 24, 27, 28, 40, 62, 64, 67, 69, 73

- aggregate\_values\_by\_key, [4](#), [5](#), [24](#), [28](#), [40](#), [62](#), [64](#), [67](#), [69](#), [73](#)
- annotation\_IS\_vars (mandatory\_IS\_vars), [58](#)
- annotation\_issues, [7](#), [26](#), [27](#), [60](#), [68](#)
- as\_sparse\_matrix, [8](#), [19](#), [30](#), [31](#), [33–35](#), [45](#), [77](#), [93](#)
- association\_file, [7](#)
- association\_file\_columns  
(mandatory\_IS\_vars), [58](#)
- available\_outlier\_tests, [9](#)
- available\_tags, [10](#)
- blood\_lineages\_default, [10](#)
- circos\_genomic\_density, [11](#), [17](#), [32](#), [38](#), [52](#), [83–85](#), [89](#)
- CIS\_grubbs, [12](#), [16](#), [17](#), [21](#), [25](#), [37](#), [42](#), [54](#), [57](#), [76](#), [91](#), [92](#)
- CIS\_grubbs\_overtime, [14](#)
- CIS\_volcano\_plot, [12](#), [16](#), [32](#), [38](#), [52](#), [83–85](#), [89](#)
- clinical\_relevant\_suspicious\_genes, [18](#), [57](#)
- comparison\_matrix, [8](#), [9](#), [18](#), [24](#), [30](#), [31](#), [33–35](#), [45](#), [46](#), [77](#), [93](#)
- compute\_abundance, [14](#), [20](#), [25](#), [37](#), [42](#), [52](#), [54](#), [57](#), [76](#), [91](#), [92](#)
- compute\_near\_integrations, [4](#), [6](#), [21](#), [28](#), [62](#), [64](#), [67](#), [69](#), [73](#)
- cumulative\_is, [14](#), [21](#), [24](#), [37](#), [42](#), [54](#), [57](#), [76](#), [91](#), [92](#)
- date\_formats, [7](#), [26](#), [27](#), [44](#), [60](#), [68](#)
- default\_af\_transform, [7](#), [26](#), [26](#), [27](#), [60](#), [68](#)
- default\_iss\_file\_prefixes, [7](#), [26](#), [27](#), [27](#), [49](#), [60](#), [68](#)
- default\_meta\_agg, [4](#), [6](#), [24](#), [27](#), [62](#), [64](#), [67](#), [69](#), [73](#)
- default\_rec\_agg\_lambdas, [28](#)
- default\_report\_path, [29](#)
- default\_stats, [29](#)
- enable\_progressBars, [9](#), [19](#), [30](#), [31](#), [33–35](#), [45](#), [77](#), [93](#)
- export\_ISA\_settings, [9](#), [19](#), [30](#), [30](#), [33–35](#), [45](#), [77](#), [93](#)
- fisher\_scatterplot, [12](#), [17](#), [31](#), [38](#), [52](#), [83–85](#), [89](#)
- gene\_frequency\_fisher, [14](#), [21](#), [25](#), [36](#), [42](#), [54](#), [57](#), [76](#), [91](#), [92](#)
- generate\_blank\_association\_file, [8](#), [9](#), [19](#), [30](#), [31](#), [33](#), [34](#), [35](#), [45](#), [77](#), [93](#)
- generate\_default\_folder\_structure, [9](#), [19](#), [30](#), [31](#), [33](#), [33](#), [35](#), [45](#), [77](#), [93](#)
- generate\_Vispa2\_launch\_AF, [9](#), [19](#), [30](#), [31](#), [33](#), [34](#), [34](#), [45](#), [77](#), [93](#)
- glue, [28](#)
- glue::glue(), [81](#)
- HSC\_population\_plot, [12](#), [17](#), [32](#), [38](#), [52](#), [83–85](#), [89](#)
- HSC\_population\_size\_estimate, [10](#), [14](#), [21](#), [25](#), [37](#), [38](#), [39](#), [54](#), [57](#), [76](#), [91](#), [92](#)
- import\_association\_file, [4](#), [26](#), [42](#), [46](#), [47](#), [49](#), [50](#)
- import\_ISA\_settings, [9](#), [19](#), [30](#), [31](#), [33–35](#), [45](#), [77](#), [93](#)
- import\_parallel\_Vispa2Matrices, [19](#), [44](#), [45](#), [49](#), [50](#)
- import\_parallel\_Vispa2Matrices\_auto, [26](#), [60](#), [68](#)
- import\_parallel\_Vispa2Matrices\_interactive, [68](#)
- import\_single\_Vispa2Matrix, [44](#), [47](#), [47](#), [50](#)
- import\_Vispa2\_stats, [4](#), [43](#), [44](#), [47](#), [49](#), [49](#)
- inspect\_tags, [50](#), [58](#), [65](#), [74](#), [80](#), [81](#)
- integration\_alluvial\_plot, [12](#), [17](#), [32](#), [38](#), [51](#), [83–85](#), [89](#)
- integration\_matrices, [53](#)
- is\_sharing, [14](#), [21](#), [25](#), [37](#), [42](#), [54](#), [55](#), [76](#), [82](#), [83](#), [91](#), [92](#)
- iss\_source, [14](#), [21](#), [25](#), [37](#), [42](#), [54](#), [57](#), [76](#), [91](#), [92](#)
- iss\_stats\_specs (mandatory\_IS\_vars), [58](#)
- known\_clinical\_oncogenes, [18](#), [57](#)
- mandatory\_IS\_vars, [51](#), [58](#), [65](#), [74](#), [80](#), [81](#)
- matching\_options, [7](#), [26](#), [27](#), [46](#), [59](#), [68](#)
- matrix\_file\_suffixes  
(mandatory\_IS\_vars), [58](#)
- NGSdataExplorer, [60](#)
- outlier\_filter, [4](#), [6](#), [9](#), [24](#), [28](#), [62](#), [63](#), [67](#), [69](#), [73](#)



- outliers\_by\_pool\_fragments, [4](#), [6](#), [24](#), [28](#),  
[61](#), [64](#), [67](#), [69](#), [73](#)
- pcr\_id\_column, [51](#), [58](#), [64](#), [74](#), [80](#), [81](#)
- proto\_oncogenes, [65](#)
- purity\_filter, [4](#), [6](#), [24](#), [28](#), [62](#), [64](#), [66](#), [69](#), [73](#)
- quantification\_types, [7](#), [19](#), [26](#), [27](#), [46](#), [60](#),  
[68](#), [77](#)
- realign\_after\_collisions, [4](#), [6](#), [24](#), [28](#), [62](#),  
[64](#), [67](#), [69](#), [73](#)
- reduced\_AF\_columns, [70](#)
- refGene\_table\_cols, [72](#)
- refGenes\_hg19, [71](#)
- refGenes\_mm9 (refGenes\_hg19), [71](#)
- remove\_collisions, [4](#), [6](#), [24](#), [28](#), [62](#), [64](#), [67](#),  
[69](#), [72](#)
- reset\_af\_columns\_def  
(reset\_mandatory\_IS\_vars), [74](#)
- reset\_annotation\_IS\_vars  
(reset\_mandatory\_IS\_vars), [74](#)
- reset\_dyn\_vars\_config  
(reset\_mandatory\_IS\_vars), [74](#)
- reset\_iss\_stats\_specs  
(reset\_mandatory\_IS\_vars), [74](#)
- reset\_mandatory\_IS\_vars, [51](#), [58](#), [65](#), [74](#),  
[80](#), [81](#)
- reset\_matrix\_file\_suffixes  
(reset\_mandatory\_IS\_vars), [74](#)
- sample\_statistics, [14](#), [21](#), [25](#), [37](#), [42](#), [54](#),  
[57](#), [75](#), [91](#), [92](#)
- separate\_quant\_matrices, [9](#), [19](#), [30](#), [31](#),  
[33–35](#), [45](#), [77](#), [93](#)
- set\_af\_columns\_def  
(set\_mandatory\_IS\_vars), [78](#)
- set\_annotation\_IS\_vars  
(set\_mandatory\_IS\_vars), [78](#)
- set\_iss\_stats\_specs  
(set\_mandatory\_IS\_vars), [78](#)
- set\_mandatory\_IS\_vars, [51](#), [58](#), [65](#), [74](#), [78](#),  
[81](#)
- set\_matrix\_file\_suffixes, [51](#), [58](#), [65](#), [74](#),  
[80](#), [81](#)
- sharing\_heatmap, [12](#), [17](#), [32](#), [38](#), [52](#), [56](#), [82](#),  
[84](#), [85](#), [89](#)
- sharing\_venn, [12](#), [17](#), [32](#), [38](#), [52](#), [56](#), [83](#), [83](#),  
[85](#), [89](#)
- threshold\_filter, [4](#), [6](#), [24](#), [28](#), [62](#), [64](#), [67](#),  
[69](#), [73](#)
- top\_abund\_tableGrob, [12](#), [17](#), [32](#), [38](#), [52](#), [83](#),  
[84](#), [84](#), [89](#)
- top\_cis\_overtime\_heatmap, [12](#), [17](#), [32](#), [38](#),  
[52](#), [83–85](#), [86](#)
- top\_integrations, [14](#), [21](#), [25](#), [37](#), [42](#), [54](#), [57](#),  
[76](#), [90](#), [92](#)
- top\_targeted\_genes, [14](#), [21](#), [25](#), [37](#), [42](#), [54](#),  
[57](#), [76](#), [91](#), [91](#)
- transform\_columns, [9](#), [19](#), [30](#), [31](#), [33–35](#), [44](#),  
[45](#), [49](#), [77](#), [93](#)
- tumor\_suppressors (proto\_oncogenes), [65](#)
- vars\_getters (mandatory\_IS\_vars), [58](#)
- vars\_res setters  
(reset\_mandatory\_IS\_vars), [74](#)
- vars\_setters (set\_mandatory\_IS\_vars), [78](#)