

Package: gDRutils (via r-universe)

October 9, 2026

Type Package

Title A package with helper functions for processing drug response data

Version 1.11.17

Date 2026-10-08

Description This package contains utility functions used throughout the gDR platform to fit data, manipulate data, and convert and validate data structures. This package also has the necessary default constants for gDR platform. Many of the functions are utilized by the gDRcore package.

License Artistic-2.0

LazyLoad yes

Depends R (>= 4.2)

Imports BumpyMatrix, checkmate, data.table, digest, drc, jsonlite, jsonvalidate, methods, MultiAssayExperiment, S4Vectors, stats, stringr, SummarizedExperiment, qs2, utils

Suggests BiocManager, BiocStyle, futile.logger, gDRstyle (>= 1.7.1), gDRtestData (>= 1.7.1), IRanges, knitr, lintr, mockery, purrr, rcmdcheck, rmarkdown, scales, testthat, tools, withr, yaml

URL <https://github.com/gdrplatform/gDRutils>,
<https://gdrplatform.github.io/gDRutils/>

BugReports <https://github.com/gdrplatform/gDRutils/issues>

biocViews Software, Infrastructure

VignetteBuilder knitr

ByteCompile TRUE

Roxygen list(markdown = TRUE)

SwitchrLibrary gDRutils

DeploySubPath gDRutils

Encoding UTF-8

Config/roxygen2/version 8.0.0

RoxygenNote 7.3.1

Config/pak/sysreqs cmake make libicu-dev libssl-dev libnode-dev zlib1g-dev

Repository <https://bioc.r-universe.dev>

Date/Publication 2026-10-09 06:04:39 UTC

RemoteUrl <https://github.com/bioc/gDRutils>

RemoteRef HEAD

RemoteSha 8ba4a03f7bddc1b0fa34ec42ddc504e2fee84f75

Contents

.set_invalid_fit_params	5
.standardize_conc	5
addClass	6
aggregate_assay	7
apply_bumpy_function	7
apply_fit	8
apply_fit_to_se	10
apply_fits	11
assert_choices	13
average_biological_replicates_dt	14
calc_sd	15
cap_assay_infinities	16
cap_xc50	17
convert_colData_to_json	18
convert_combo_data_to_dt	19
convert_combo_field_to_assay	20
convert_mae_assay_to_dt	20
convert_mae_to_json	22
convert_metadata_to_json	22
convert_rowData_to_json	23
convert_se_assay_to_custom_dt	24
convert_se_assay_to_dt	25
convert_se_to_json	27
define_matrix_grid_positions	28
demote_fields	29
df_to_bm_assay	30
extend_normalization_type_name	30
fit_curves	31
fit_drug_response_metrics	32
fit_drug_response_metrics_4p	34
flatten	35
gen_synthetic_data	36
get_additional_variables	37
get_assay_dt_duplicated_rows	37

get_assay_names	38
get_assay_req_uniq_cols	39
get_combo_assay_names	39
get_combo_base_assay_names	40
get_combo_excess_field_names	41
get_combo_score_assay_names	41
get_combo_score_field_names	42
get_default_identifiers	42
get_duplicated_rows	43
get_env_assay_names	43
get_env_var	44
get_expect_one_identifiers	45
get_experiment_groups	45
get_fit_profile	46
get_fit_profiles	47
get_gDR_session_info	47
get_identifiers_dt	48
get_idfs_synonyms	48
get_isobologram_columns	49
get_MAE_identifiers	49
get_non_empty_assays	50
get_required_identifiers	50
get_settings_from_json	51
get_supported_experiments	52
get_synthetic_data	52
get_testdata	53
get_testdata_codilution	53
get_testdata_combo	54
has_assay_dt_duplicated_rows	54
has_dt_duplicated_rows	55
has_single_codrug_data	55
has_valid_codrug_data	56
headers	57
identifiers	58
identify_unique_se_metadata_fields	59
is_any_exp_empty	60
is_combo_data	61
is_exp_empty	61
is_mae_empty	62
logisticFit	63
loop	65
MAEapply	66
map_conc_to_standardized_conc	67
mcolData	68
merge_assay	68
merge_MAE	69
merge_metadata	70
merge_SE	71

modifyData	72
mrowData	73
persist_fit_assay	74
predict_conc_from_efficacy	75
predict_efficacy_from_conc	76
predict_smooth_from_combo	77
prettify_flat_metrics	77
process_batch	78
promote_fields	79
refine_coldata	80
refine_rowdata	81
register_fit_profile	81
remove_codrug_data	82
remove_drug_batch	83
rename_bumpy	85
rename_DFrame	85
round_concentration	86
SE_metadata	86
set_constant_fit_params	88
set_unique_cl_names	89
set_unique_cl_names_dt	89
set_unique_drug_names	90
set_unique_drug_names_dt	91
set_unique_identifiers	92
set_unique_names_dt	93
shorten_normalization_type_name	93
split_big_table_for_xlsx	94
split_SE_components	95
standardize_mae	96
standardize_se	96
throw_msg_if_duplicates	97
update_env_idfs_from_mae	98
update_idfs_synonyms	98
validate_identifiers	99
validate_json	100
validate_MAE	100
validate_mae_with_schema	101
validate_SE	102
validate_se_assay_name	103

`.set_invalid_fit_params`*Set fit parameters for an invalid fit.*

Description

Set fit parameters for an invalid fit.

Usage

```
.set_invalid_fit_params(out, norm_values)
```

Arguments

<code>out</code>	Named list of fit parameters.
<code>norm_values</code>	Numeric vector used to estimate an <code>xc50</code> value.

Value

Modified named list of fit parameters.

Examples

```
.set_invalid_fit_params(list(), norm_values = rep(0.3, 6))
```

`.standardize_conc`*Standardize concentration values.*

Description

Standardize concentration values.

Usage

```
.standardize_conc(conc)
```

Arguments

<code>conc</code>	numeric vector of the concentrations
-------------------	--------------------------------------

Details

If no `conc` are passed, `NULL` is returned.

Value

vector of standardized concentrations

Examples

```
concs <- 10 ^ (seq(-1, 1, 0.9))  
.standardize_conc(concs)
```

addClass

add arbitrary S3 class to an object

Description

Modify and object's class attribute.

Usage

```
addClass(x, newClass)
```

Arguments

x	an object
newClass	character string; class to be added

Details

This is a simple convenience function that an item to the class attribute of an object so that it can be dispatched to a proper S3 method. This is purely for code clarity, so that individual methods do not clutter the definitions of higher order functions.

Value

The same object with an added S3 class.

Examples

```
addClass(data.table::data.table(), "someClass")
```

aggregate_assay	<i>Aggregate a BumpyMatrix assay by a given aggregation function.</i>
-----------------	---

Description

Aggregation can only be performed on nested variables.

Usage

```
aggregate_assay(asy, by, FUN)
```

Arguments

asy	A BumpyMatrix object.
by	Character vector of the nested fields to aggregate by.
FUN	A function to use to aggregate the data.

Value

A BumpyMatrix object aggregated by FUN.

Examples

```
mae <- get_synthetic_data("finalMAE_small.qs2")
se <- mae[[1]]
assay <- SummarizedExperiment::assay(se)
aggregate_assay(assay, FUN = mean, by = c("Barcode"))
```

apply_bumpy_function	<i>Apply a function to every element of a bumpy matrix.</i>
----------------------	---

Description

Apply a user-specified function to every element of a bumpy matrix.

Usage

```
apply_bumpy_function(
  se,
  FUN,
  req_assay_name,
  out_assay_name,
  parallelize = FALSE,
  ...
)
```

Arguments

se	A SummarizedExperiment object with bumpy matrices.
FUN	A function that will be applied to each element of the matrix in assay req_assay_name. Output of the function must return a data.table.
req_assay_name	String of the assay name in the se that the FUN will act on.
out_assay_name	String of the assay name that will contain the results of the applied function.
parallelize	Logical indicating whether or not to parallelize the computation.
...	Additional args to be passed to teh FUN.

Value

The original se object with a new assay, out_assay_name.

Examples

```
mae <- get_synthetic_data("finalMAE_small.qs2")
se <- mae[[1]]
FUN <- function(x) {
  data.table::data.table(Concentration = x$Concentration, CorrectedReadout = x$CorrectedReadout)
}
apply_bumpy_function(
  se,
  FUN = FUN,
  req_assay_name = "RawTreated",
  out_assay_name = "CorrectedReadout"
)
```

apply_fit

apply_fit

Description

Generic layer for applying a user-supplied fit function across all (row x column x slicing_col) triplets in a [SummarizedExperiment](#) and persisting the results into a named output assay.

Usage

```
apply_fit(
  se,
  fit_fn,
  data_type,
  slicing_cols = NULL,
  slicing_values = NULL,
  input_assay = NULL,
  output_assay,
```

```

summary_fn = NULL,
summary_assay = NULL,
merge = "merge",
on_error = "warn",
fit_source
)

```

Arguments

se	SummarizedExperiment
fit_fn	function(data.table) returning a named list or single-row data.frame. Receives data already filtered to one slicing_cols combination.
data_type	one of "single-agent", "combination", "time-course". Resolves profile defaults.
slicing_cols	character vector of column name(s) used to split each BumpyMatrix cell into sub-experiments. NULL uses the profile default.
slicing_values	character vector of values to iterate over in slicing_cols. NULL iterates all unique values found in each cell.
input_assay	name of the source BumpyMatrix assay. NULL uses the profile default.
output_assay	name of the assay to write fit results into. Required. Any assay name is accepted, including the native "Metrics" assay. With merge = "merge" (default), existing rows keyed by fit_source are replaced while all other rows are preserved — safe for co-existing alongside native gDR metrics. With merge = "replace", the entire assay is overwritten.
summary_fn	optional function(data.table) → named list called once per (row x column) cell on all rows returned by fit_fn for that cell. Requires summary_assay.
summary_assay	name of the assay to write summary results into. Required when summary_fn is provided.
merge	"merge" (idempotent upsert keyed by fit_source • slicing_cols) or "replace" (overwrite the whole assay).
on_error	"warn" (skip failed cells with a warning) or "stop" (propagate the error).
fit_source	character string recorded in the fit_source column of every output row. Forms part of the upsert key.

Details

The experiment type is declared via data_type, which resolves a built-in slicing profile (default slicing_cols, slicing_values, and input_assay). Any profile field can be overridden with the corresponding argument.

An optional summary_fn may be provided. It is called once per (row x column) cell and receives all rows that fit_fn produced for that cell, returning a single summary row written to summary_assay. This is the right place for aggregated metrics (e.g. mean synergy across normalization types) that span multiple slices.

Use the pipe to apply several custom fits to the same SE:

```
se |>
  apply_fit(bliss_fn, "combination",
            output_assay = "custom_bliss", ...) |>
  apply_fit(musyc_fn, "combination",
            output_assay = "musyc_params",
            summary_fn = musyc_summary_fn,
            summary_assay = "musyc_summary", ...)
```

Value

updated SummarizedExperiment with output_assay (and optionally summary_assay) added or updated.

Examples

```
mae <- get_synthetic_data("finalMAE_small.qs2")
se <- mae[["single-agent"]]
mean_fn <- function(dt) list(x_mean = mean(dt$x, na.rm = TRUE))
se_out <- apply_fit(
  se, mean_fn, "single-agent",
  output_assay = "custom_mean", fit_source = "demo"
)
```

apply_fit_to_se

apply_fit_to_se

Description

Apply a user-supplied fit function per (drug x cell line x normalization_type) triplet and persist results into the Metrics assay.

Usage

```
apply_fit_to_se(
  se,
  fit_fn,
  normalization_types = c("GR", "RV"),
  averaged_assay = "Averaged",
  metrics_assay = "Metrics",
  merge = "merge",
  on_error = "warn",
  fit_source = "custom"
)
```

Arguments

se `SummarizedExperiment` with an Averaged assay
fit_fn function taking a `data.table` and returning a named list
normalization_types character vector of types to iterate
averaged_assay name of the input assay
metrics_assay name of the output assay
merge "merge" (idempotent upsert) or "replace"
on_error "warn" (skip + warning) or "stop"
fit_source string recorded in the `fit_source` column

Details

This is a convenience wrapper around `apply_fit` for the single-agent use case with the standard Metrics assay.

Value

updated `SummarizedExperiment`

Examples

```

mae <- get_synthetic_data("finalMAE_small.qs2")
se <- mae[["single-agent"]]
simple_fn <- function(dt) list(x_mean = mean(dt$x, na.rm = TRUE))
se_out <- apply_fit_to_se(se, simple_fn, fit_source = "demo")
  
```

apply_fits

apply_fits

Description

Apply multiple fit functions in a **single pass** over each `BumpyMatrix` cell, writing each function's results into its own named output assay.

Usage

```

apply_fits(
  se,
  fit_fns,
  data_type,
  slicing_cols = NULL,
  slicing_values = NULL,
  input_assay = NULL,
  )
  
```

```

merge = "merge",
on_error = "warn",
fit_source
)

```

Arguments

se	SummarizedExperiment
fit_fns	named list of functions. Each name becomes an output assay name; each value is a function(data.table) → named list (or a named list of named lists for the shared pre-computation pattern).
data_type	one of "single-agent", "combination", "time-course".
slicing_cols	character vector; NULL uses profile default.
slicing_values	character vector; NULL uses profile default.
input_assay	string; NULL uses profile default.
merge	"merge" or "replace".
on_error	"warn" or "stop".
fit_source	character string stamped as fit_source in every output row.

Details

apply_fits() is the performance-efficient alternative to chaining multiple [apply_fit](#) calls when two or more fit functions operate on the **same input assay**. Instead of unsplitting the BumpyMatrix K times (once per function), it traverses each cell once and applies all functions in that single pass.

Use this when:

- You have two or more independent fit functions on the same data (e.g. Bliss + HSS on combination data).
- A single fit function produces results for multiple output assays (shared pre-computation pattern — see below).

Independent fit functions (most common): Names of fit_fns become the output assay names:

```

apply_fits(
  combo_se,
  fit_fns = list(
    custom_bliss = bliss_fit_fn,
    custom_hss   = hss_fit_fn
  ),
  data_type = "combination",
  fit_source = "synergy"
)

```

Shared pre-computation (advanced): A single function can return a **named list of named lists** to write multiple assays while computing expensive intermediates only once:

```

bliss_and_hss <- function(dt) {
  sa_curves <- fit_hill_curves(dt) # expensive – done once
  list(
    custom_bliss = list(bliss_score = compute_bliss(sa_curves, dt)),
    custom_hss   = list(hss_score   = compute_hss(sa_curves, dt))
  )
}
apply_fits(
  combo_se,
  fit_fns   = list(bliss_and_hss = bliss_and_hss),
  output_assay_map = c(bliss_and_hss = NA), # ignored; keys from return value
  ...
)

```

The multi-output pattern is detected automatically when a fit function returns a named list whose values are themselves named lists. Each top-level name maps to an assay; the inner named list provides the row columns.

Value

updated SummarizedExperiment with one new (or updated) assay per entry in `fit_fns`.

Examples

```

mae <- get_synthetic_data("finalMAE_small.qs2")
se <- mae[["single-agent"]]
fn_a <- function(dt) list(x_mean = mean(dt$x, na.rm = TRUE))
fn_b <- function(dt) list(x_sd = sd(dt$x, na.rm = TRUE))
se_out <- apply_fits(
  se,
  fit_fns = list(mean_metrics = fn_a, sd_metrics = fn_b),
  data_type = "single-agent",
  fit_source = "demo"
)

```

assert_choices

assert_choices

Description

assert_choices

Usage

```
assert_choices(x, choices, ...)
```

Arguments

x	charvec expected subset
choices	charvec reference set
...	Additional arguments to pass to <code>checkmate::test_choice</code>

Value

NULL

Examples

```
assert_choices("x", c("x","y"))
```

```
average_biological_replicates_dt
```

Average biological replicates on the data table side.

Description

Average biological replicates on the data table side.

Usage

```
average_biological_replicates_dt(
  dt,
  var,
  prettified = FALSE,
  fixed = TRUE,
  geometric_average_fields = get_header("metric_average_fields")$geometric_mean,
  fit_type_average_fields = get_header("metric_average_fields")$fit_type,
  blacklisted_fields = get_header("metric_average_fields")$blacklisted,
  add_sd = FALSE,
  copy_dt = TRUE
)
```

Arguments

dt	data.table with Metric data
var	String representing additional metadata of replicates
prettified	Flag indicating if the provided identifiers in the dt are prettified
fixed	Flag indicating whether to add a fix for -Inf in the geometric mean.
geometric_average_fields	Character vector of column names in dt to take the geometric average of.

fit_type_average_fields
 Character vector of column names in dt that should be treated as a column with fit type data
 blacklisted_fields
 Character vector of column names in dt that should be skipped in averaging
 add_sd
 Flag indicating whether to add standard deviation and count columns.
 copy_dt
 Flag indicating whether to copy the input data.table before modifying. Set to FALSE when the caller already provides a copy.

Value

data.table without replicates

Examples

```
dt <- data.table::data.table(a = c(seq_len(10), 1),
  b = c(rep("drugA", 10), rep("drugB", 1)))
average_biological_replicates_dt(dt, var = "a")
```

 calc_sd

Calculate Standard Deviation or Return Zero

Description

This function calculates the standard deviation of a numeric vector. If the vector has a length of 1 and it is numeric, it returns 0.

Usage

```
calc_sd(x)
```

Arguments

x A numeric vector.

Value

The standard deviation of the vector if its length is greater than 1 or it is not numeric, otherwise 0.

Examples

```
calc_sd(c(1, 2, 3, 4, 5)) # Should return the standard deviation
calc_sd(c(1)) # Should return 0
calc_sd(numeric(0)) # Should return NA
calc_sd(c("a", "b", "c")) # Should return NA
```



```

                                experiment_name = "single-agent")

# combination data
cdata <- get_synthetic_data("finalMAE_combo_matrix_small.qs2")
scaveraged_data <- convert_se_assay_to_dt(cdata[[get_supported_experiments("combo")]], "Averaged")
scmetrics_data <- convert_se_assay_to_dt(cdata[[get_supported_experiments("combo")]], "Metrics")
scmetrics_data_capped <- cap_assay_infinities(scaveraged_data,
                                             scmetrics_data,
                                             experiment_name = "combination")

```

cap_xc50	<i>Cap XC50 value.</i>
----------	------------------------

Description

Set IC50/GR50 value to Inf or -Inf based on upper and lower limits.

Usage

```
cap_xc50(xc50, max_conc, min_conc = NA, capping_fold = 5)
```

Arguments

xc50	Numeric value of the IC50/GR50 to cap.
max_conc	Numeric value of the highest concentration in a dose series used to calculate the xc50.
min_conc	Numeric value of the lowest concentration in a dose series used to calculate the xc50. If NA (default), using max_conc/1e5 instead.
capping_fold	Integer value of the fold number to use for capping. Defaults to 5.

Details

Note: xc50 and max_conc should share the same units. Ideally, the lower_cap should be based on the lowest tested concentration. However, since we don't record that, it is set 5 orders of magnitude below the highest dose.

Value

Capped IC50/GR50 value.

Examples

```
cap_xc50(xc50 = 1, max_conc = 2)
cap_xc50(xc50 = 2, max_conc = 5, min_conc = 1)
cap_xc50(xc50 = 26, max_conc = 5, capping_fold = 5)
```

`convert_colData_to_json`*Convert colData to JSON*

Description

Convert colData to JSON format for elasticsearch indexing.

Usage

```
convert_colData_to_json(  
  cdata,  
  identifiers,  
  req_cols = c("cellline", "cellline_name", "cellline_tissue", "cellline_ref_div_time")  
)
```

Arguments

cdata	data.table of colData.
identifiers	charvec with identifiers
req_cols	charvec required columns

Details

Standardizes the cdata to common schema fields and tidies formatting to be conducive to joining with other JSON responses.

Value

JSON string capturing the cdata.

Examples

```
cdata <- data.table::data.table(  
  mycellline = letters,  
  mycelllinename = letters,  
  mycelllinetissue = letters,  
  cellline_ref_div_time = "cellline_ref_div_time")  
identifiers <- list(cellline = "mycellline",  
  cellline_name = "mycelllinename",  
  cellline_ref_div_time = "cellline_ref_div_time",  
  cellline_tissue = "mycelllinetissue")  
convert_colData_to_json(cdata, identifiers)
```

`convert_combo_data_to_dt`*convert combo assays from SummarizedExperiments to the list of data.tables*

Description

convert combo assays from SummarizedExperiments to the list of data.tables

Usage

```
convert_combo_data_to_dt(  
  se,  
  c_assays = get_combo_assay_names(),  
  normalization_type = c("RV", "GR"),  
  prettify = TRUE  
)
```

Arguments

<code>se</code>	SummarizedExperiment object with dose-response data
<code>c_assays</code>	charvec of combo assays to be used
<code>normalization_type</code>	charvec of normalization_types expected in the data
<code>prettify</code>	boolean flag indicating whether or not to prettify the colnames of the returned data

Value

list of data.table(s) with combo data

Author(s)

Arkadiusz Gładki <arkadiusz.gladki@contractors.roche.com>

Examples

```
mae <- get_synthetic_data("finalMAE_combo_matrix_small.qs2")  
convert_combo_data_to_dt(mae[[1]])
```

convert_combo_field_to_assay
get combo assay names based on the field name

Description

get combo assay names based on the field name

Usage

```
convert_combo_field_to_assay(field)
```

Arguments

field String containing name of the field for which the assay name should be returned

Value

charvec

Examples

```
convert_combo_field_to_assay("hsa_score")
```

convert_mae_assay_to_dt
Convert a MultiAssayExperiment assay to a long data.table

Description

Convert an assay within a [SummarizedExperiment](#) object in a MultiAssayExperiment to a long data.table.

Usage

```
convert_mae_assay_to_dt(  
  mae,  
  assay_name,  
  experiment_name = NULL,  
  include_metadata = TRUE,  
  retain_nested_rownames = FALSE,  
  wide_structure = FALSE,  
  drop_masked = TRUE,  
  merge_additional_variables = FALSE  
)
```

Arguments

mae	A MultiAssayExperiment object holding experiments with raw and/or processed dose-response data in its assays.
assay_name	String of name of the assay to transform within an experiment of the mae.
experiment_name	String of name of the experiment in mae whose assay_name should be converted. Defaults to NULL to indicate to convert assay in all experiments into one data.table object.
include_metadata	Boolean indicating whether or not to include rowData() and colData() in the returned data.table. Defaults to TRUE.
retain_nested_rownames	Boolean indicating whether or not to retain the rownames nested within a BumpyMatrix assay. Defaults to FALSE. If the assay_name is not of the BumpyMatrix class, this argument's value is ignored. If TRUE, the resulting column in the data.table will be named as "<assay_name>_rownames".
wide_structure	Boolean indicating whether or not to transform data.table into wide format. wide_structure = TRUE requires retain_nested_rownames = TRUE however that will be validated in convert_se_assay_to_dt function
drop_masked	Boolean indicating whether to drop masked values; TRUE by default.
merge_additional_variables	Boolean indicating whether to merge additional variables identified by get_additional_variables into the DrugName column. Defaults to FALSE.

Details

NOTE: to extract information about 'Control' data, simply call the function with the name of the assay holding data on controls.

Value

data.table representation of the data in assay_name.

Author(s)

Bartosz Czech czech.bartosz@external.gene.com

See Also

flatten convert_se_assay_to_dt

Examples

```
mae <- get_synthetic_data("finalMAE_small.qs2")
convert_mae_assay_to_dt(mae, "Metrics")
```

convert_mae_to_json	Create JSON document.
---------------------	-----------------------

Description

Convert a MultiAssayExperiment object to a JSON document.

Usage

```
convert_mae_to_json(mae, with_experiments = TRUE)
```

Arguments

mae	SummarizedExperiment object.
with_experiments	logical convert experiment metadata as well?

Value

String representation of a JSON document.

Examples

```
mae <- get_synthetic_data("finalMAE_small.qs2")
convert_mae_to_json(mae)
convert_mae_to_json(mae, with_experiments = FALSE)
```

convert_metadata_to_json	Convert experiment metadata to JSON format for elasticsearch indexing.
--------------------------	--

Description

Convert experiment metadata to JSON format for elasticsearch indexing.

Usage

```
convert_metadata_to_json(se)
```

Arguments

se	SummarizedExperiment object.
----	------------------------------

Value

JSON string capturing experiment metadata.

Examples

```
md <- list(title = "my awesome experiment",
  description = "description of experiment",
  sources = list(list(name = "GeneData_Screener", id = "QCS-12345")))
se <- SummarizedExperiment::SummarizedExperiment(metadata = md)
convert_metadata_to_json(se)
```

`convert_rowData_to_json`*Convert rowData to JSON*

Description

Convert rowData to JSON format for elasticsearch indexing.

Usage

```
convert_rowData_to_json(
  rdata,
  identifiers,
  req_cols = c("drug", "drug_name", "drug_moa", "duration")
)
```

Arguments

<code>rdata</code>	data.table of rowData.
<code>identifiers</code>	charvec with identifiers
<code>req_cols</code>	charvec required columns

Details

Standardizes the `rdata` to common schema fields and tidies formatting to be conducive to joining with other JSON responses.

Value

JSON string capturing the `rdata`.

Examples

```
rdata <- data.table::data.table(
  mydrug = letters,
  mydrugname = letters,
  mydrugmoa = letters,
  Duration = 1)
identifiers <- list(drug = "mydrug", drug_name = "mydrugname", drug_moa = "mydrugmoa",
  duration = "Duration")
convert_rowData_to_json(rdata, identifiers)
```

```
convert_se_assay_to_custom_dt
```

Convert a SummarizedExperiment assay to a long data.table and conduct some post processing steps

Description

Convert an assay within a SummarizedExperiment object to a long data.table. Then conduct some post processing steps.

Usage

```
convert_se_assay_to_custom_dt(
  se,
  assay_name,
  output_table = NULL,
  cap_values = FALSE
)
```

Arguments

se	A SummarizedExperiment object holding raw and/or processed dose-response data in its assays.
assay_name	String of name of the assay to transform within the se.
output_table	String of type name of the output data.table.
cap_values	Logical indicating whether to apply capping (via capVals) for "Metrics" output. Default is FALSE.

Details

Current strategy is per-assay specific.

1. combo assays: conversion to data.table only (with wide_structure = FALSE)
2. 'Metrics' assay can be converted to three types of outputs:
 - Metrics_initial (conversion to data.table only, with wide_structure = FALSE)

- Metrics_raw: same as Metrics_initial followed by:
 - fix for 'EC50' and 'Metrics_rownames'
 - flatten
 - prettifying and dropping excess variables
- Metrics (same as Metrics_raw + cap_values if cap_values = TRUE)

1. 'Normalization' and 'Averaged' assay:

- conversion to data.table (with wide_structure = TRUE)
- prettifying and dropping excess variables

NOTE: to extract information about 'Control' data, simply call the function with the name of the assay holding data on controls. To extract the reference data in the same format as 'Averaged' use convert_se_ref_assay_to_dt.

Value

data.table representation of the data in assay_name with added information from colData.

See Also

convert_se_assay_to_dt

Examples

```
mae <- get_synthetic_data("finalMAE_small.qs2")
se <- mae[[1]]
convert_se_assay_to_custom_dt(se, "Metrics")
convert_se_assay_to_custom_dt(se, "Metrics", output_table = "Metrics_raw")
convert_se_assay_to_custom_dt(se, "Metrics", output_table = "Metrics_initial")
convert_se_assay_to_custom_dt(se, "Averaged")
convert_se_assay_to_custom_dt(se, "Metrics", cap_values = TRUE)
```

convert_se_assay_to_dt

Convert a SummarizedExperiment assay to a long data.table

Description

Convert an assay within a [SummarizedExperiment](#) object to a long data.table.

Usage

```
convert_se_assay_to_dt(
  se,
  assay_name,
  include_metadata = TRUE,
  retain_nested_rownames = FALSE,
  wide_structure = FALSE,
  unify_metadata = FALSE,
  drop_masked = TRUE,
  merge_additional_variables = FALSE
)
```

Arguments

<code>se</code>	A SummarizedExperiment object holding raw and/or processed dose-response data in its assays.
<code>assay_name</code>	String of name of the assay to transform within the <code>se</code> .
<code>include_metadata</code>	Boolean indicating whether or not to include <code>rowData(se)</code> and <code>colData(se)</code> in the returned <code>data.table</code> . Defaults to <code>TRUE</code> .
<code>retain_nested_rownames</code>	Boolean indicating whether or not to retain the rownames nested within a <code>BumpyMatrix</code> assay. Defaults to <code>FALSE</code> . If the <code>assay_name</code> is not of the <code>BumpyMatrix</code> class, this argument's value is ignored. If <code>TRUE</code> , the resulting column in the <code>data.table</code> will be named as " <code><assay_name>_rownames</code> ".
<code>wide_structure</code>	Boolean indicating whether or not to transform <code>data.table</code> into wide format. <code>wide_structure = TRUE</code> requires <code>retain_nested_rownames = TRUE</code> .
<code>unify_metadata</code>	Boolean indicating whether to unify <code>DrugName</code> and <code>CellLineName</code> in cases where <code>DrugNames</code> and <code>CellLineNames</code> are shared by more than one <code>Gnumber</code> and/or <code>clid</code> within the experiment.
<code>drop_masked</code>	Boolean indicating whether to drop masked values; <code>TRUE</code> by default.
<code>merge_additional_variables</code>	Boolean indicating whether to merge additional variables identified by <code>get_additional_variables</code> into the <code>DrugName</code> column. Defaults to <code>FALSE</code> .

Details

NOTE: to extract information about 'Control' data, simply call the function with the name of the assay holding data on controls. To extract the reference data in to same format as 'Averaged' use `convert_se_ref_assay_to_dt`.

Value

`data.table` representation of the data in `assay_name`.

See Also

`flatten`

Examples

```
mae <- get_synthetic_data("finalMAE_small.qs2")
se <- mae[[1]]
convert_se_assay_to_dt(se, "Metrics")
```

convert_se_to_json	<i>Convert a SummarizedExperiment object to a JSON document.</i>
--------------------	--

Description

Convert a SummarizedExperiment object to a JSON document.

Usage

```
convert_se_to_json(se)
```

Arguments

se SummarizedExperiment object.

Value

String representation of a JSON document.

Examples

```
md <- list(title = "my awesome experiment",
  description = "description of experiment",
  source = list(name = "GeneData_Screener", id = "QCS-12345"))
rdata <- data.table::data.table(
  mydrug = letters,
  mydrugname = letters,
  mydrugmoa = letters,
  Duration = 1)
cdata <- data.table::data.table(mycellline = letters, mycelllinename = letters,
  mycelllinetissue = letters, cellline_ref_div_time = letters)
identifiers <- list(cellline = "mycellline",
  cellline_name = "mycelllinename",
  cellline_tissue = "mycelllinetissue",
  cellline_ref_div_time = "cellline_ref_div_time",
  drug = "mydrug",
  drug_name = "mydrugname",
  drug_moa = "mydrugmoa",
  duration = "Duration")
se <- SummarizedExperiment::SummarizedExperiment(rowData = rdata,
  colData = cdata)
se <- set_SE_experiment_metadata(se, md)
se <- set_SE_identifiers(se, identifiers)
```

```
convert_se_to_json(se)
```

```
define_matrix_grid_positions
```

Define matrix grid positions

Description

Define matrix grid positions

Usage

```
define_matrix_grid_positions(conc1, conc2)
```

Arguments

conc1	drug_1 concentration
conc2	drug_2 concentration

Details

drug_1 is diluted along the rows as the y-axis and drug_2 is diluted along the columns and will be the x-axis.

Value

list with axis grid positions

Examples

```
cl_name <- "cellline_BC"
drug1_name <- "drug_001"
drug2_name <- "drug_026"

se <- get_synthetic_data("combo_matrix_small")[["combination"]]
dt_average <- convert_se_assay_to_dt(se, "Averaged")[normalization_type == "GR"]

ls_axes <- define_matrix_grid_positions(
  dt_average[["Concentration"]], dt_average[["Concentration_2"]])
```

demote_fields	<i>Demote a metadata field in the rowData or colData of a SummarizedExperiment object to a nested field of a BumpyMatrix assay.</i>
---------------	---

Description

Demote a metadata field in the rowData or colData of a SummarizedExperiment object to a nested field of a BumpyMatrix assay.

Usage

```
demote_fields(se, fields)
```

Arguments

se	A SummarizedExperiment object.
fields	Character vector of metadata fields to demote as nested columns.

Details

Revert this operation using promote_fields.

Value

A SummarizedExperiment object with new dimensions resulting from demoting given fields to nested columns.

See Also

promote_fields

Examples

```
mae <- get_synthetic_data("finalMAE_small.qs2")
se <- mae[[1]]
se <- promote_fields(se, "ReadoutValue", 2)
demote_fields(se, "ReadoutValue")
```

<code>df_to_bm_assay</code>	<i>df_to_bm_assay</i>
-----------------------------	-----------------------

Description

Convert data.table with dose-response data into a BumpyMatrix assay.

Usage

```
df_to_bm_assay(data, discard_keys = NULL, precomputed_metadata = NULL)
```

Arguments

- `data` data.table with drug-response data
- `discard_keys` a vector of keys that should be discarded
- `precomputed_metadata` optional named list from a prior [split_SE_components](#) call to avoid recomputation

Details

The 'assay' is simply a BumpyMatrix object with rownames being the treatment ids, colnames being the ids of the cell lines and values with dose-response data for given cell lines under given conditions.

Value

BumpyMatrix object

Examples

```
df_to_bm_assay(data.table::data.table(Gnumber = 2, clid = "A"))
```

<code>extend_normalization_type_name</code>	<i>extend abbreviated normalization type</i>
---	--

Description

extend abbreviated normalization type

Usage

```
extend_normalization_type_name(x)
```

Arguments

`x` string with normalization type

Value

string

Examples

```
extend_normalization_type_name("GR")
```

fit_curves

Fit curves

Description

Fit GR and RV curves from a data.table.

Usage

```
fit_curves(
  df_,
  series_identifiers,
  e_0 = 1,
  GR_0 = 1,
  n_point_cutoff = 4,
  range_conc = c(0.005, 5),
  force_fit = FALSE,
  pcutoff = 0.05,
  cap = 0.1,
  normalization_type = c("GR", "RV")
)
```

Arguments

`df_` data.table containing data to fit. See details.

`series_identifiers` character vector of the column names in data.table whose combination represents a unique series for which to fit curves.

`e_0` numeric value representing the x_0 value for the RV curve. Defaults to 1.

`GR_0` numeric value representing the x_0 value for the GR curve. Defaults to 1.

`n_point_cutoff` integer of how many points should be considered the minimum required to try to fit a curve. Defaults to 4.

`range_conc` numeric vector of length 2 indicating the lower and upper concentration ranges. Defaults to `c(5e-3, 5)`. See details.

force_fit	boolean indicating whether or not to force a constant fit. Defaults to FALSE.
pcutoff	numeric of pvalue significance threshold above or equal to which to use a constant fit. Defaults to 0.05.
cap	numeric value capping norm_values to stay below ($x_0 + \text{cap}$). Defaults to 0.1.
normalization_type	character vector of types of curves to fit. Defaults to c("GR", "RV").

Details

The `df_` expects the following columns:

- RelativeViability normalized relative viability values (if `normalization_type` includes "RV")
- GRvalue normalized GR values (if `normalization_type` includes "GR")

The `range_conc` is used to calculate the `x_AOC_range` statistic. The purpose of this statistic is to enable comparison across different experiments with slightly different concentration ranges.

Value

data.table of fit parameters as specified by the `normalization_type`.

Examples

```
df_ <- data.table::data.table(Concentration = c(0.001, 0.00316227766016838,
0.01, 0.0316227766016838),
x_std = c(0.1, 0.1, 0.1, 0.1), normalization_types = c("RV", "RV", "RV", "RV"),
x = c(0.9999964000144, 0.999964001439942, 0.999640143942423, 0.996414342629482))

fit_curves(df_, "Concentration", normalization_type = "RV")
```

```
fit_drug_response_metrics
      fit_drug_response_metrics
```

Description

Reference fit function replicating standard gDR Hill curve fitting (3-parameter model, x_0 fixed at 1). Produces output column-compatible with `fit_SE()` in `gDRcore` and `logisticFit`, including `p_value`, `rss`, `x_AOC_range`, `x_max`, and `x_sd_avg`. For use with `apply_fit_to_se` or `apply_fit` on single-agent or time-course data.

Usage

```
fit_drug_response_metrics(
  avg_dt,
  x_col = "x",
  capping_fold = 5,
  range_conc = c(0.005, 5),
  pcutoff = 0.05,
  n_point_cutoff = 4L,
  force_fit = FALSE,
  cap = 0.1
)
```

Arguments

avg_dt	data.table of averaged data for one (drug x cell line x normalization_type) triplet. Should contain columns Concentration, the response column named by x_col, normalization_type, and optionally x_std (used for x_sd_avg).
x_col	string; name of the response column in avg_dt. Defaults to "x" (the standard Averaged assay column name). Set to e.g. "NormalizedGrowthRate" when fitting time-course growth-rate data.
capping_fold	numeric capping fold passed to cap_xc50
range_conc	numeric vector of length 2 specifying the concentration range used to compute x_AOC_range. Default c(5e-3, 5), matching the fit_SE() default.
pcutoff	numeric p-value cutoff for the fit. Default 0.05.
n_point_cutoff	integer minimum number of concentration points required to attempt fitting. Default 4L.
force_fit	logical; if TRUE, force fitting even when the number of points is below n_point_cutoff. Default FALSE.
cap	numeric; upper cap on xc50 relative to the concentration range. Default 0.1.

Value

Named list of fit metrics fully compatible with the standard gDR Metrics assay schema: ec50, xc50, h, x_inf, x_0, r2, rss, p_value, x_mean, x_AOC, x_AOC_range, x_max, x_sd_avg, N_conc, maxlog10Concentration, fit_type, normalization_type.

See Also

[fit_drug_response_metrics_4p](#) for a 4-parameter variant that fits x_0 freely.

Examples

```
dt <- data.table::data.table(
  Concentration = c(0.001, 0.01, 0.1, 1, 10),
  x = c(0.95, 0.8, 0.5, 0.2, 0.1),
  normalization_type = "RV"
)
```

```

fit_drug_response_metrics(dt)

# Custom response column (e.g. time-course growth rates):
dt2 <- data.table::data.table(
  Concentration = c(0.001, 0.01, 0.1, 1, 10),
  NormalizedGrowthRate = c(0.98, 0.82, 0.55, 0.21, 0.09),
  normalization_type = "GR"
)
fit_drug_response_metrics(dt2, x_col = "NormalizedGrowthRate")

```

fit_drug_response_metrics_4p

fit_drug_response_metrics_4p

Description

Variant of [fit_drug_response_metrics](#) using a 4-parameter Hill model — x_0 is fitted freely rather than fixed at 1. Useful when the upper asymptote is expected to deviate from 1 (e.g. partial agonists, non-standard assay readouts).

Usage

```

fit_drug_response_metrics_4p(
  avg_dt,
  x_col = "x",
  capping_fold = 5,
  range_conc = c(0.005, 5),
  pcutoff = 0.05,
  n_point_cutoff = 4L,
  force_fit = FALSE,
  cap = 0.1
)

```

Arguments

avg_dt	data.table of averaged data for one (drug x cell line x normalization_type) triplet. Should contain columns Concentration, the response column named by x_col, normalization_type, and optionally x_std (used for x_sd_avg).
x_col	string; name of the response column in avg_dt. Defaults to "x" (the standard Averaged assay column name). Set to e.g. "NormalizedGrowthRate" when fitting time-course growth-rate data.
capping_fold	numeric capping fold passed to cap_xc50
range_conc	numeric vector of length 2 specifying the concentration range used to compute x_AOC_range. Default c(5e-3, 5), matching the fit_SE() default.
pcutoff	numeric p-value cutoff for the fit. Default 0.05.

- n_point_cutoff integer minimum number of concentration points required to attempt fitting. Default 4L.
- force_fit logical; if TRUE, force fitting even when the number of points is below n_point_cutoff. Default FALSE.
- cap numeric; upper cap on xc50 relative to the concentration range. Default 0.1.

Value

Named list of fit metrics. fit_type will be "DRC4pHillFitModel" on success.

See Also

[fit_drug_response_metrics](#) for the default 3-parameter variant that matches fit_SE() output.

Examples

```
dt <- data.table::data.table(
  Concentration = c(0.001, 0.01, 0.1, 1, 10),
  x = c(0.95, 0.8, 0.5, 0.2, 0.1),
  normalization_type = "RV"
)
fit_drug_response_metrics_4p(dt)
```

flatten	<i>Flatten a table</i>
---------	------------------------

Description

Flatten a stacked table into a wide format.

Usage

```
flatten(tbl, groups, wide_cols, sep = "_")
```

Arguments

- tbl table to flatten.
- groups character vector of column names representing unifying groups in expansion.
- wide_cols character vector of column names to flatten.
- sep string representing separator between wide_cols columns, used in column re-naming. Defaults to "_".

Details

flattened columns will be named with original column names prefixed by wide_cols columns, concatenated together and separated by sep.
A common use case for this function is when a flattened version of the "Metrics" assay is desired.

Value

table of flattened data as defined by wide_cols.

See Also

convert_se_assay_to_dt

Examples

```
n <- 4
m <- 5
grid <- expand.grid(normalization_type = c("GR", "RV"),
  source = c("GDS", "GDR"))
repgrid <- data.table::rbindlist(rep(list(grid), m))
repgrid$wide <- seq(m * n)
repgrid$id <- rep(LETTERS[1:m], each = n)

groups <- colnames(grid)
wide_cols <- c("wide")

flatten(repgrid, groups = groups, wide_cols = wide_cols)
```

gen_synthetic_data	<i>gen_synthetic_data</i>
--------------------	---------------------------

Description

Function for generating local synthetic data used for unit tests in modules

Usage

```
gen_synthetic_data(m = 1, n = 5)
```

Arguments

m	number of drugs
n	number of records

Value

list with drugs, cell_lines, raw_data and assay_data

Examples

```
gen_synthetic_data()
```

`get_additional_variables`*Identify and return additional variables in list of dt*

Description

Identify and return additional variables in list of dt

Usage

```
get_additional_variables(dt_list, unique = FALSE, prettified = FALSE)
```

Arguments

<code>dt_list</code>	list of data.table or data.table containing additional variables
<code>unique</code>	logical flag indicating if all variables should be returned or only those containing more than one unique value
<code>prettified</code>	Flag indicating if the provided identifiers in the dt are prettified

Value

vector of variable names with additional variables

Examples

```
dt <- data.table::data.table(  
  Gnumber = seq_len(10),  
  Concentration = runif(10),  
  Ligand = c(rep(0.5, 5), rep(0, 5))  
)  
get_additional_variables(dt)
```

`get_assay_dt_duplicated_rows`*Helper function to find duplicated rows in assay data*

Description

Helper function to find duplicated rows in assay data

Usage

```
get_assay_dt_duplicated_rows(dt, output = "index")
```

Arguments

dt	data.table
output	string with the output format to be returned

Value

integer vector or data.table with duplicated rows

Examples

```
sdata <- get_synthetic_data("finalMAE_small.qs2")
smetrics_data <- convert_se_assay_to_dt(sdata[[1]], "Metrics")
get_assay_dt_duplicated_rows(smetrics_data, output = "data")
get_assay_dt_duplicated_rows(smetrics_data)
```

get_assay_names	<i>get assay names of the given se/dataset fetch the data from the se if provided as metadata use predefined values from get_env_assay_names otherwise</i>
-----------------	--

Description

get assay names of the given se/dataset fetch the data from the se if provided as metadata use predefined values from get_env_assay_names otherwise

Usage

```
get_assay_names(se = NULL, ...)
```

Arguments

se	SummarizedExperiment or NULL
...	Additional arguments to pass to get_env_assay_names.

Value

charvec

Author(s)

Arkadiusz Gładki <arkadiusz.gladki@contractors.roche.com>

Examples

```
get_assay_names()
```

`get_assay_req_uniq_cols`*get columns in the assay data required to have unique data*

Description

get columns in the assay data required to have unique (non-duplicated) data

Usage

```
get_assay_req_uniq_cols(dt)
```

Arguments

`dt` data.table with assay data

Value

charvec with columns required to have unique data

Examples

```
sdata <- get_synthetic_data("finalMAE_small.qs2")
smetrics_data <- convert_se_assay_to_dt(sdata[[1]], "Metrics")
get_assay_req_uniq_cols(smetrics_data)
```

`get_combo_assay_names` *get names of combo assays*

Description

get names of combo assays

Usage

```
get_combo_assay_names(se = NULL, ...)
```

Arguments

`se` SummarizedExperiment or NULL
`...` Additional arguments to pass to `get_assay_names`.

Value

charvec of combo assay names.

Author(s)

Arkadiusz Gładki <arkadiusz.gladki@contractors.roche.com>

Examples

```
get_combo_assay_names()
```

```
get_combo_base_assay_names
```

```
get names of combo base assays
```

Description

get names of combo base assays

Usage

```
get_combo_base_assay_names(se = NULL, ...)
```

Arguments

se SummarizedExperiment or NULL

... Additional arguments to pass to `get_combo_assay_names`.

Value

charvec

Author(s)

Arkadiusz Gładki <arkadiusz.gladki@contractors.roche.com>

Examples

```
get_combo_base_assay_names()
```

```
get_combo_excess_field_names
```

get names of combo excess fields

Description

get names of combo excess fields

Usage

```
get_combo_excess_field_names()
```

Value

charvec

Examples

```
get_combo_excess_field_names()
```

```
get_combo_score_assay_names
```

get names of combo score assays

Description

get names of combo score assays

Usage

```
get_combo_score_assay_names(se = NULL, ...)
```

Arguments

se	SummarizedExperiment or NULL
...	Additional arguments to pass to get_combo_assay_names.

Value

charvec

Author(s)

Arkadiusz Gładki <arkadiusz.gladki@contractors.roche.com>

Examples

```
get_combo_score_assay_names()
```

```
get_combo_score_field_names
```

get names of combo score fields

Description

get names of combo score fields

Usage

```
get_combo_score_field_names()
```

Value

charvec

Examples

```
get_combo_score_assay_names()
```

```
get_default_identifiers
```

Get gDR default identifiers required for downstream analysis.

Description

Get gDR default identifiers required for downstream analysis.

Usage

```
get_default_identifiers()
```

Value

charvec

Examples

```
get_default_identifiers()
```

get_duplicated_rows	<i>Helper function to find duplicated rows</i>
---------------------	--

Description

Helper function to find duplicated rows

Usage

```
get_duplicated_rows(x, col_names = NULL, output = "index")
```

Arguments

x	DataFrame or data.table
col_names	character vector, columns in which duplication are searched for
output	string with the output format to be returned - one of "index" (index of duplicates) or "data" (subset of input data with duplicates)

Value

integer vector or data.table with duplicated rows

Examples

```
dt <- data.table::data.table(a = c(1, 2, 3), b = c(3, 2, 2))
get_duplicated_rows(dt, "b")
get_duplicated_rows(dt, "b", output = "data")
```

get_env_assay_names	<i>get default assay names for the specified filters, i.e. set of assay types, assay groups and assay data types</i>
---------------------	--

Description

get default assay names for the specified filters, i.e. set of assay types, assay groups and assay data types

Usage

```
get_env_assay_names(
  type = NULL,
  group = NULL,
  data_type = NULL,
  prettify = FALSE,
  simplify = TRUE
)
```

Arguments

type	charvec of assay types
group	charvec of assay groups
data_type	charvec assay of data types
prettify	logical flag, prettify the assay name?
simplify	logical flag, simplify the output? will return single string instead of named vector with single element useful when function is expected to return single element/assay only

Value

charvec

Author(s)

Arkadiusz Gładki <arkadiusz.gladki@contractors.roche.com>

Examples

```
get_env_assay_names()
```

get_env_var	<i>safe wrapper of Sys.getenv()</i>
-------------	-------------------------------------

Description

So far the helper is needed to handle env vars containing : for which the backslash is automatically added in some contexts and R could not get the original value for these env vars.

Usage

```
get_env_var(x, ...)
```

Arguments

x	string with the name of the environmental variable
...	additional params for Sys.getenv

Value

sanitized value of the env variable

Examples

```
get_env_var("HOME")
```

`get_expect_one_identifiers`*Get identifiers that expect only one value for each identifier.*

Description

Get identifiers that expect only one value for each identifier.

Usage

```
get_expect_one_identifiers()
```

Value

charvec

Examples

```
get_expect_one_identifiers()
```

`get_experiment_groups` *get_experiment_groups*

Description

get experiment groups

Usage

```
get_experiment_groups(type = NULL)
```

Arguments

type String indicating the name of an assay group. Defaults to all experiment groups.

Value

list with experiment groups or string (if type not NULL)

Author(s)

Arkadiusz Gladki arkadiusz.gladki@contractors.roche.com

Examples

```
get_experiment_groups()
```

get_fit_profile	<i>Get a single fit profile by name</i>
-----------------	---

Description

Get a single fit profile by name

Usage

```
get_fit_profile(name)
```

Arguments

name string; profile name (e.g. "single-agent").

Value

Named list with the following fields:

slicing_cols Column(s) used to split each BumpyMatrix cell into sub-experiments before calling fit_fn (e.g. "normalization_type").

slicing_values Values of slicing_cols to iterate; NULL means all unique values found in each cell.

input_assay Default source assay name.

nested_cols Informational: the column(s) that index rows *inside* each BumpyMatrix cell (i.e., what fit_fn receives as row structure). Values are the canonical default identifiers resolved by get_env_identifiers() at package load — they may differ if identifiers are customised at runtime.

nested_cols_note Human-readable note about how nested_cols are resolved.

description Human-readable description of the profile.

Stops if the profile does not exist.

Examples

```
get_fit_profile("single-agent")
```

get_fit_profiles	<i>Get all registered fit profiles</i>
------------------	--

Description

Returns the list of all experiment-type profiles used by `apply_fit` and `apply_fits` to resolve default slicing configuration.

Usage

```
get_fit_profiles()
```

Value

Named list of profile definitions. Each element contains `slicing_cols`, `slicing_values`, `input_assay`, and `description`.

Examples

```
get_fit_profiles()
```

get_gDR_session_info	<i>get gDR package and their version installed in the environment</i>
----------------------	---

Description

get gDR package and their version installed in the environment

Usage

```
get_gDR_session_info(pattern = "^gDR")
```

Arguments

pattern	string with the pattern to grep R packages from the list of installed packages
---------	--

Value

data.table with gDR packages and their versions

Examples

```
get_gDR_session_info()
```

get_identifiers_dt	<i>Get descriptions for identifiers</i>
--------------------	---

Description

Get descriptions for identifiers

Usage

```
get_identifiers_dt(k = NULL, get_description = FALSE, get_example = FALSE)
```

Arguments

k	identifier key, string
get_description	return descriptions only, boolean
get_example	return examples only, boolean

Value

named list

Examples

```
get_identifiers_dt()
```

get_idfs_synonyms	<i>Get gDR synonyms for the identifiers</i>
-------------------	---

Description

Get gDR synonyms for the identifiers

Usage

```
get_idfs_synonyms()
```

Value

charvec

Examples

```
get_idfs_synonyms()
```

`get_isobologram_columns`*Get isobologram column names*

Description

Get isobologram column names

Usage

```
get_isobologram_columns(k = NULL, prettify = TRUE)
```

Arguments

k	key
prettify	change to upper case and add underscore, iso_level -> Iso_Level

Value

character vector of isobologram column names for combination data

Examples

```
get_isobologram_columns()  
get_isobologram_columns("iso_level", prettify = TRUE)
```

`get_MAE_identifiers` *get_MAE_identifiers*

Description

get the identifiers of all SE's in the MAE

Usage

```
get_MAE_identifiers(mae)
```

Arguments

mae	MultiAssayExperiment
-----	----------------------

Value

named list with identifiers for each SE

Examples

```
mae <- get_synthetic_data("finalMAE_small.qs2")
get_MAE_identifiers(mae)
```

```
get_non_empty_assays    get_non_empty_assays
```

Description

get non empty assays

Usage

```
get_non_empty_assays(mae)
```

Arguments

mae MultiAssayExperiment object

Value

charvec with non-empty experiments

Author(s)

Arkadiusz Gladki arkadiusz.gladki@contractors.roche.com

Examples

```
mae <- get_synthetic_data("finalMAE_small.qs2")
get_non_empty_assays(mae)
```

```
get_required_identifiers
```

Get identifiers required for downstream analysis.

Description

Get identifiers required for downstream analysis.

Usage

```
get_required_identifiers()
```

Value

charvec

Examples

```
get_required_identifiers()
```

```
get_settings_from_json
```

Get settings from JSON file In most common scenario the settings are stored in JSON file to avoid hardcoding

Description

Get settings from JSON file In most common scenario the settings are stored in JSON file to avoid hardcoding

Usage

```
get_settings_from_json(  
  s = NULL,  
  json_path = system.file(package = "gDRutils", "settings.json")  
)
```

Arguments

s	charvec with setting entry/entries
json_path	string with the path to the JSON file

Value

value/values for entry/entries from JSON file

Examples

```
if (!inchar(system.file(package="gDRutils"))) {  
  get_settings_from_json()  
}
```

get_supported_experiments	
	<i>get_supported_experiments</i>

Description

get supported experiments

Usage

```
get_supported_experiments(type = NULL)
```

Arguments

type	String indicating the type of experiment
------	--

Value

charvec with supported experiment name(s)

Author(s)

Arkadiusz Gladki arkadiusz.gladki@contractors.roche.com

Examples

```
get_supported_experiments()
```

get_synthetic_data	<i>Get synthetic data from gDRtestData package</i>
--------------------	--

Description

Get synthetic data from gDRtestData package

Usage

```
get_synthetic_data(qs)
```

Arguments

qs	dataset name or qs2 filename (e.g. "small" or "finalMAE_small.qs2")
----	---

Value

loaded data

Examples

```
get_synthetic_data("finalMAE_small.qs2")
```

get_testdata

get_testdata

Description

Function to obtain data from gDRtestData and prepare for unit tests

Usage

```
get_testdata()
```

Value

list with drugs, cell_lines, raw_data and assay_data

Examples

```
get_testdata()
```

get_testdata_codilution

get_testdata_codilution

Description

Function to obtain data from gDRtestData and prepare for unit tests

Usage

```
get_testdata_codilution()
```

Value

list with drugs, cell_lines, raw_data and assay_data

Examples

```
get_testdata_codilution()
```

```
get_testdata_combo      get_testdata_combo
```

Description

Function to obtain data from gDRtestData and prepare for unit tests

Usage

```
get_testdata_combo()
```

Value

list with drugs, cell_lines, raw_data and assay_data

Examples

```
get_testdata_combo()
```

```
has_assay_dt_duplicated_rows
      check if assay data contains duplicated data
```

Description

An auxiliary function that checks for duplicates in the assay data

Usage

```
has_assay_dt_duplicated_rows(dt)
```

Arguments

dt data.table with assay data

Value

logical flag indicating if a dt contains duplicated rows or not

Examples

```
sdata <- get_synthetic_data("finalMAE_small.qs2")
smetrics_data <- convert_se_assay_to_dt(sdata[[1]], "Metrics")
has_assay_dt_duplicated_rows(smetrics_data)
```

`has_dt_duplicated_rows`*check if data.table contains duplicated data*

Description

An auxiliary function that checks for duplicates in the data.table (or its subset)

Usage

```
has_dt_duplicated_rows(dt, col_names = NULL)
```

Arguments

dt	data.table
col_names	charvec with columns to be used for subsetting

Value

logical flag indicating if a dt contains duplicated rows or not

Examples

```
dt <- data.table::data.table(a = c(1, 2, 3), b = c(3, 2, 2))
has_dt_duplicated_rows(dt, "b")
```

`has_single_codrug_data`*Has Single Codrug Data*

Description

Has Single Codrug Data

Usage

```
has_single_codrug_data(  
  cols,  
  prettify_identifiers = TRUE,  
  codrug_identifiers = c("drug_name2", "concentration2")  
)
```

Arguments

cols character vector with the columns of the input data
 prettify_identifiers logical flag specifying if identifiers are expected to be prettified
 codrug_identifiers character vector with identifiers for the codrug columns

Value

logical flag

Examples

```
has_single_codrug_data("Drug Name")
has_single_codrug_data(c("Drug Name", "Cell Lines"))
has_single_codrug_data(c("Drug Name 2", "Concentration 2"))
has_single_codrug_data(
  get_prettified_identifiers(
    c("concentration2", "drug_name2"),
    simplify = FALSE
  )
)
```

has_valid_codrug_data *Has Valid Codrug Data*

Description

Has Valid Codrug Data

Usage

```
has_valid_codrug_data(
  data,
  prettify_identifiers = TRUE,
  codrug_name_identifier = "drug_name2",
  codrug_conc_identifier = "concentration2"
)
```

Arguments

data data.table with input data
 prettify_identifiers logical flag specifying if identifiers are expected to be prettified
 codrug_name_identifier string with the identifiers for the codrug drug_name column
 codrug_conc_identifier string with the identifiers for the codrug concentration column

Value

logical flag

Examples

```
dt <-
  data.table::data.table(
    "Drug Name" = letters[seq_len(3)],
    "Concentration" = seq_len(3),
    "Drug Name 2" = letters[4:6],
    "Concentration 2" = 4:6
  )
has_valid_codrug_data(dt)

dt$`Concentration 2` <- NULL
has_valid_codrug_data(dt)
```

headers

Get or reset headers for one or all header field(s) respectively

Description

Get the expected header(s) for one field or reset all header fields

Usage

```
get_header(k = NULL)
```

Arguments

k string of field (data type) to return headers for

Details

If `get_header` is called with no values, the entire available header list is returned.

Value

For `get_header` a character vector of headers for field `k`.

Examples

```
get_header(k = NULL)
get_header("manifest")
```

identifiers

Get, set, or reset identifiers for one or all identifier field(s)

Description

Get, set, or reset the expected identifier(s) for one or all identifier field(s). Identifiers are used by the gDR processing functions to identify which columns in a `data.table` correspond to certain expected fields. Functions of the family `*et_identifier` will look for identifiers from the environment while functions of the family `*et_SE_identifiers` will look for identifiers in the metadata slot of a `SummarizedExperiment` object. See details for expected identifiers and their definitions.

Usage

```
get_env_identifiers(k = NULL, simplify = TRUE)

get_prettified_identifiers(k = NULL, simplify = TRUE)

set_env_identifier(k, v)

reset_env_identifiers()
```

Arguments

<code>k</code>	String corresponding to identifier name.
<code>simplify</code>	Boolean indicating whether output should be simplified.
<code>v</code>	Character vector corresponding to the value for given identifier <code>k</code> .

Details

Identifiers supported by the gDR suite include:

- "barcode": String of column name containing barcode metadata
- "cellline": String of column name containing unique, machine-readable cell line identifiers
- "cellline_name": String of column name containing human-friendly cell line names
- "cellline_tissue": String of column name containing metadata on cell line tissue type
- "cellline_ref_div_time": String of column name containing reference division time for cell lines
- "cellline_parental_identifier": String of column name containing unique, machine-readable parental cell line identifiers. Used in the case of derived or engineered cell lines.
- "drug": String of column name containing unique, machine-readable drug identifiers
- "drug_name": String of column name containing human-friendly drug names
- "drug_moa": String of column name containing metadata for drug mode of action
- "duration": String of column name containing metadata on duration that cells were treated (in hours)

- "template": String of column name containing template metadata
- "untreated_tag": Character vector of entries that identify control, untreated wells
- "well_position": Character vector of column names containing metadata on well positions on a plate

Value

For any setting or resetting functionality, a NULL invisibly. For `get_env_identifiers` a character vector of identifiers for field `k`. For functions called with no arguments, the entire available identifier list is returned.

list or charvec depends on unify param

list or charvec depends on unify param

NULL

NULL

Examples

```
get_env_identifiers("duration") # "Duration"
```

```
identify_unique_se_metadata_fields
```

Identify unique metadata fields from a list of SummarizedExperiments

Description

Identify unique metadata fields from a list of SummarizedExperiments

Usage

```
identify_unique_se_metadata_fields(SElist)
```

Arguments

SElist named list of SummarizedExperiments

Value

character vector of unique names of metadata

Examples

```
mae <- get_synthetic_data("finalMAE_small.qs2")
se <- mae[[1]]
SElist <- list(
  se,
  se
)
identify_unique_se_metadata_fields(SElist)
```

is_any_exp_empty	<i>is_any_exp_empty</i>
------------------	-------------------------

Description

check if any experiment is empty

Usage

```
is_any_exp_empty(mae)
```

Arguments

mae	MultiAssayExperiment object
-----	-----------------------------

Value

logical

Author(s)

Arkadiusz Gladki arkadiusz.gladki@contractors.roche.com

Examples

```
mae <- get_synthetic_data("finalMAE_small.qs2")
is_any_exp_empty(mae)
```

is_combo_data	Checks if se is combo dataset.
---------------	--------------------------------

Description

Checks if se is combo dataset.

Usage

is_combo_data(se)

Arguments

se SummarizedExperiment

Value

logical

Examples

```
se <- get_synthetic_data("finalMAE_combo_matrix.qs2")[[1]]
is_combo_data(se)
se <- get_synthetic_data("finalMAE_combo_matrix.qs2")[[2]]
is_combo_data(se)
se <- get_synthetic_data("finalMAE_small.qs2")[[1]]
is_combo_data(se)
```

is_exp_empty	is_exp_empty
--------------	--------------

Description

check if experiment (SE) is empty

Usage

is_exp_empty(exp)

Arguments

exp SummarizedExperiment object.

Value

logical

Author(s)

Arkadiusz Gladki arkadiusz.gladki@contractors.roche.com

Examples

```
mae <- get_synthetic_data("finalMAE_small.qs2")
se <- mae[[1]]
is_exp_empty(se)
```

is_mae_empty

is_mae_empty

Description

check if all mae experiments are empty

Usage

```
is_mae_empty(mae)
```

Arguments

mae MultiAssayExperiment object

Value

logical

Author(s)

Arkadiusz Gladki arkadiusz.gladki@contractors.roche.com

Examples

```
mae <- get_synthetic_data("finalMAE_small.qs2")
is_mae_empty(mae)
```

logisticFit

*Logistic fit***Description**

Fit a logistic curve to drug response data.

Usage

```
logisticFit(
  concs,
  norm_values,
  std_norm_values = NA,
  x_0 = 1,
  priors = NULL,
  lower = NULL,
  range_conc = c(0.005, 5),
  force_fit = FALSE,
  pcutoff = 0.05,
  cap = 0.1,
  n_point_cutoff = 4,
  capping_fold = 5
)
```

Arguments

concs	concentrations that have not been transformed into log space.
norm_values	normalized response values (Untreated = 1).
std_norm_values	std of values.
x_0	upper limit. Defaults to 1. For co-treatments, this value should be set to NA.
priors	numeric vector containing starting values for all. mean parameters in the model. Overrides any self starter function.
lower	numeric vector of lower limits for all parameters in a 4-param model.
range_conc	range of concentration for calculating AOC_range.
force_fit	boolean indicating whether or not to force a parameter-based fit.
pcutoff	numeric of pvalue significance threshold above or equal to which to use a constant fit.
cap	numeric value capping norm_values to stay below (x_0 + cap).
n_point_cutoff	integer indicating number of unique concentrations required to fit curve.
capping_fold	Integer value of the fold number to use for capping IC50/GR50. Default is 5.

Details

Implementation of the genedata approach for curve fit: <https://screener.genedata.com/documentation/display/DOC21/Business-Rules-for-Dose-Response-Curve-Fitting,-Model-Selection,-and-Fit-Validity.html> #nolint

The output parameter names correspond to the following definitions:

- x_mean** The mean of a given dose-response metric
- x_AOC_range** The range of the area over the curve
- x_AOC** The area over the GR curve or, respectively, under the relative cell count curve, averaged over the range of concentration values
- xc50** The concentration at which the effect reaches a value of 0.5 based on interpolation of the fitted curve
- x_max** The maximum effect of the drug
- ec50** The drug concentration at half-maximal effect
- x_inf** The asymptotic value of the sigmoidal fit to the dose-response data as concentration goes to infinity
- x_0** The asymptotic metric value corresponding to a concentration of 0 for the primary drug
- h** The hill coefficient of the fitted curve, which reflects how steep the dose-response curve is
- r2** The goodness of the fit
- x_sd_avg** The standard deviation of GR/IC
- fit_type** This will be given by one of the following:
 - "DRC4pHillFitModel" Successfully fit with a 4-parameter model
 - "DRC3pHillFitModelFixS0" Successfully fit with a 3-parameter model
 - "DRCConstantFitResult" Successfully fit with a constant fit
 - "DRCTooFewPointsToFit" Not enough points to run a fit
 - "DRCInvalidFitResult" Fit was attempted but failed
- maxlog10Concentration** The highest log10 concentration
- N_conc** Number of unique concentrations

Value

data.table with metrics and fit parameters.

Examples

```
logisticFit(
  c(0.001, 0.00316227766016838, 0.01, 0.0316227766016838),
  c(0.9999964000144, 0.999964001439942, 0.999640143942423, 0.996414342629482),
  rep(0.1, 4),
  priors = c(2, 0.4, 1, 0.00658113883008419)
)
```

loop	<i>Conditional lapply with optional batch processing.</i>
------	---

Description

Conditional lapply with optional batch processing.

Usage

```
loop(
  x,
  FUN,
  parallelize = as.logical(Sys.getenv("GDR_PARALLELIZE", "FALSE")),
  use_batch = as.logical(Sys.getenv("GDR_USE_BATCH", "FALSE")),
  temp_dir = Sys.getenv("GDR_TEMP_DIR", tempdir()),
  batch_size = as.numeric(Sys.getenv("GDR_BATCH_SIZE", 100)),
  ...
)
```

Arguments

x	Vector (atomic or list) or an expression object. Other objects (including classed objects) will be coerced by as.list
FUN	A user-defined function to apply to each element of x.
parallelize	Logical indicating whether or not to parallelize the computation. Defaults to <code>as.logical(Sys.getenv("GDR_PARALLELIZE", "FALSE"))</code> .
use_batch	Logical indicating whether to use batch processing to save intermediate results. Defaults to FALSE.
temp_dir	Character string specifying the directory where batch results are saved. Defaults to <code>tempdir()</code> .
batch_size	Integer specifying the number of elements to process in each batch during batch mode. Defaults to 100.
...	Optional arguments passed to lapply .

Details

The function operates in two modes:

1. Regular mode: Directly applies FUN to the elements using `lapply`.
2. Batch mode: Saves results in batches to disk, allowing computation to resume from the last saved step. Batch mode is activated by setting `use_batch` to TRUE.

Value

List containing output of FUN applied to every element in x. When batch processing is enabled, results are saved incrementally and merged at the end of processing.

Examples

```
# Regular processing
loop(list(1, 2, 3), function(x) x^2, parallelize = FALSE, use_batch = FALSE)

# Batch processing
loop(1:10, function(x) x^2, parallelize = TRUE, use_batch = TRUE)
```

MAEapply

*Lapply through all the experiments in MultiAssayExperiment object***Description**

Lapply through all the experiments in MultiAssayExperiment object

Usage

```
MAEapply(mae, FUN, unify = FALSE, ...)
```

Arguments

mae	MultiAssayExperiment object
FUN	function that should be applied on each experiment of MultiAssayExperiment object
unify	logical indicating if the output should be a unlisted object of unique values across all the experiments
...	Additional args to be passed to teh FUN.

Value

list or vector depends on unify param

Author(s)

Bartosz Czech czech.bartosz@external.gene.com

Examples

```
mae <- get_synthetic_data("finalMAE_small.qs2")
MAEapply(mae, SummarizedExperiment::assayNames)
```

`map_conc_to_standardized_conc`*Create a mapping of concentrations to standardized concentrations.*

Description

Create a mapping of concentrations to standardized concentrations.

Usage

```
map_conc_to_standardized_conc(conc1, conc2)
```

Arguments

<code>conc1</code>	numeric vector of the concentrations for drug 1.
<code>conc2</code>	numeric vector of the concentrations for drug 2.

Details

The concentrations are standardized in that they will contain regularly spaced dilutions and close values will be rounded.

Value

data.table of 2 columns named "concs" and "rconcs" containing the original concentrations and their closest matched standardized concentrations respectively. and their new standardized concentrations.

Examples

```
ratio <- 0.5
conc1 <- c(0, 10 ^ (seq(-3, 1, ratio)))

shorter_range <- conc1[-1]
noise <- runif(length(shorter_range), 1e-12, 1e-11)
conc2 <- shorter_range + noise

map_conc_to_standardized_conc(conc1, conc2)
```

`mcolData`*mcolData*

Description

get colData of all experiments

Usage

```
mcolData(mae)
```

Arguments

mae MultiAssayExperiment object

Value

data.table with all-experiments colData

Author(s)

Arkadiusz Gladki arkadiusz.gladki@contractors.roche.com

Examples

```
mae <- get_synthetic_data("finalMAE_small.qs2")
mcolData(mae)
```

`merge_assay`*Merge assay data*

Description

Merge assay data

Usage

```
merge_assay(  
  SElist,  
  assay_name,  
  additional_col_name = "data_source",  
  discard_keys = NULL  
)
```

Arguments

SElist	named list of Summarized Experiments
assay_name	name of the assay that should be extracted and merged
additional_col_name	string of column name that will be added to assay data for the distinction of possible duplicated metrics that can arise from multiple projects
discard_keys	character vector of string that will be discarded during creating BumpyMatrix object

Value

BumpyMatrix or list with data.table + BumpyMatrix

Examples

```
mae <- get_synthetic_data("finalMAE_combo_2dose_nonoise.qs2")

listSE <- list(
  combo1 = mae[[1]],
  sa = mae[[2]]
)
merge_assay(listSE, "Normalized")
```

merge_MAE	<i>Merge multiple MultiAssayExperiment objects</i>
-----------	--

Description

Merge multiple MultiAssayExperiment objects

Usage

```
merge_MAE(
  MAElist,
  additional_col_name = "data_source",
  discard_keys = c("normalization_type", "fit_source", "record_id", "isDay0", "swap_sa",
    "control_type", "iso_level", "conc_1", "conc_2"),
  title = NULL,
  description = NULL,
  source_name = NULL,
  source_id = NULL
)
```

Arguments

MAElist	Named list of MultiAssayExperiment objects.
additional_col_name	String with the name of the column that will be added to assay data for the distinction of possible duplicated metrics that can arise from multiple projects.
discard_keys	Character vector of strings that will be discarded during creating BumpyMatrix object.
title	String specifying the final DataSetDB title. If NULL, auto-generates.
description	String specifying the final DataSetDB description. If NULL, auto-generates.
source_name	String specifying the standard DSDB source name. If NULL, auto-detects or uses "merged_analysis".
source_id	String specifying the unique DSDB source ID. If NULL, uses "merged_dataset".

Value

Merged MultiAssayExperiment object.

Examples

```
mae1 <- get_synthetic_data("finalMAE_combo_2dose_nonoise.qs2")
mae2 <- get_synthetic_data("finalMAE_combo_2dose_nonoise.qs2")
merge_MAE(list(mae1 = mae1, mae2 = mae2), title = "Test", description = "Test MAE")
```

merge_metadata	<i>Merge metadata</i>
----------------	-----------------------

Description

Merge metadata

Usage

```
merge_metadata(SElist, metadata_fields)
```

Arguments

SElist	named list of SummarizedExperiments
metadata_fields	vector of metadata names that will be merged

Value

list of merged metadata

Examples

```

mae <- get_synthetic_data("finalMAE_small.qs2")
se <- mae[[1]]
listSE <- list(
  se,
  se
)
metadata_fields <- identify_unique_se_metadata_fields(listSE)
merge_metadata(listSE, metadata_fields)

```

merge_SE

*Merge multiple Summarized Experiments***Description**

Merge multiple Summarized Experiments

Usage

```

merge_SE(
  SElist,
  additional_col_name = "data_source",
  discard_keys = c("normalization_type", "fit_source", "record_id", "isDay0", "swap_sa",
    "control_type", "iso_level", "conc_1", "conc_2")
)

```

Arguments

SElist	named list of Summarized Experiments
additional_col_name	string with the name of the column that will be added to assay data for the distinction of possible duplicated metrics that can arise from multiple projects
discard_keys	character vector of string that will be discarded during creating BumpyMatrix object

Value

merged SummarizedExperiment object

Examples

```

se1 <- get_synthetic_data("finalMAE_small.qs2")[[1]]
merge_SE(list(se1 = se1, se2 = se1))

```

modifyData	<i>modify assay with additional data</i>
------------	--

Description

Reduce dimensionality of an assay by dropping extra data or combining variables.

Usage

```
modifyData(x, ...)

## S3 method for class 'drug_name2'
modifyData(x, option, keep, ...)

## S3 method for class 'data_source'
modifyData(x, option, keep, ...)

## Default S3 method:
modifyData(x, option, keep, ...)
```

Arguments

x	a data.table containing an assay
...	additional arguments passed to methods
option	character string specifying the action to be taken, see Details
keep	character string specifying the value of the active variable that will be kept

Details

If an assay extracted from a SummarizedExperiment contains additional information, i.e. factors beyond DrugName and CellLineName, that information will be treated in one of three ways, depending on the value of option:

- drop: Some information will be discarded and only one value of the additional variable (chosen by the user) will be kept.
- toDrug: The information is pasted together with the primary drug name. All observations are kept.
- toCellLine: As above, but pasting is done with cell line name.

Depending on the type of additional information, the exact details will differ. This is handled in the app by adding special classes to the data tables and dispatching to S3 methods.

Following modification, the additional columns are discarded.

Value

modified object

Methods (by class)

- `modifyData(drug_name2)`: includes the name and concentration of the second drug
- `modifyData(data_source)`: includes the data source
- `modifyData(default)`: includes the name of other additional variables

Examples

```
dt <- data.table::data.table(a = as.character(1:10), b = "data")
dt <- addClass(dt, "a")
modifyData(dt, "average", keep = "b")
```

*mrowData**mrowData*

Description

get rowData of all experiments

Usage

```
mrowData(mae)
```

Arguments

`mae` MultiAssayExperiment object

Value

data.table with all-experiments rowData

Author(s)

Arkadiusz Gladki arkadiusz.gladki@contractors.roche.com

Examples

```
mae <- get_synthetic_data("finalMAE_small.qs2")
mrowData(mae)
```

persist_fit_assay	<i>Persist a data.table into a BumpyMatrix assay</i>
-------------------	--

Description

Performs an idempotent upsert (merge mode) keyed by upsert_key_cols, or a full overwrite (replace mode).

Usage

```
persist_fit_assay(se, new_dt, merge, assay_name, row, col, upsert_key_cols)
```

Arguments

se	SummarizedExperiment
new_dt	data.table including "row" and "column" index fields
merge	"merge" or "replace"
assay_name	assay name to write into
row	row index column name in new_dt
col	column index column name in new_dt
upsert_key_cols	character vector of column(s) forming the upsert key

Value

updated SummarizedExperiment

Examples

```
mae <- get_synthetic_data("finalMAE_small.qs2")
se <- mae[["single-agent"]]
dt <- data.table::data.table(
  row = rownames(se)[1], column = colnames(se)[1],
  fit_source = "demo", x_mean = 0.5
)
se <- persist_fit_assay(se, dt, "merge", "custom_mean", "row", "column", "fit_source")
SummarizedExperiment::assayNames(se)
```

`predict_conc_from_efficacy`*Predict a concentration for a given efficacy with fit parameters.*

Description

Predict a concentration for a given efficacy with fit parameters.

Usage

```
predict_conc_from_efficacy(efficacy, x_inf, x_0, ec50, h)
```

Arguments

<code>efficacy</code>	Numeric vector representing efficacies to predict concentrations for.
<code>x_inf</code>	Numeric vector representing the asymptotic value of the sigmoidal fit to the dose-response data as concentration goes to infinity.
<code>x_0</code>	Numeric vector representing the asymptotic metric value corresponding to a concentration of 0 for the primary drug.
<code>ec50</code>	Numeric vector representing the drug concentration at half-maximal effect.
<code>h</code>	Numeric vector representing the hill coefficient of the fitted curve, which reflects how steep

Details

The inverse of this function is `predict_efficacy_from_conc`.

Value

Numeric vector representing predicted concentrations from given efficacies and fit parameters.

See Also

`predict_efficacy_from_conc` `.calculate_x50`

Examples

```
predict_conc_from_efficacy(efficacy = c(1, 1.5), x_inf = 0.1, x_0 = 1, ec50 = 0.5, h = 2)
```

`predict_efficacy_from_conc`*Predict efficacy values given fit parameters and a concentration.*

Description

Predict efficacy values given fit parameters and a concentration.

Usage

```
predict_efficacy_from_conc(c, x_inf, x_0, ec50, h)
```

Arguments

<code>c</code>	Numeric vector representing concentrations to predict efficacies for.
<code>x_inf</code>	Numeric vector representing the asymptotic value of the sigmoidal fit to the dose-response data as concentration goes to infinity.
<code>x_0</code>	Numeric vector representing the asymptotic metric value corresponding to a concentration of 0 for the primary drug.
<code>ec50</code>	Numeric vector representing the drug concentration at half-maximal effect.
<code>h</code>	Numeric vector representing the hill coefficient of the fitted curve, which reflects how steep the dose-response curve is.

Details

The inverse of this function is `predict_conc_from_efficacy`.

Value

Numeric vector representing predicted efficacies from given concentrations and fit parameters.

See Also

`predict_conc_from_efficacy`

Examples

```
predict_efficacy_from_conc(c = 1, x_inf = 0.1, x_0 = 1, ec50 = 0.5, h = 2)
```

predict_smooth_from_combo

Predict a smoothed response for a drug combination

Description

Predicts a 'smooth' value for a single pair of drug concentrations by snapping to the nearest available models from a metrics table and averaging their predictions. This is the combination equivalent of 'predict_efficiency_from_conc'.

Usage

```
predict_smooth_from_combo(conc_1, conc_2, metrics_merged)
```

Arguments

conc_1 A single numeric value for the desired concentration of the first drug.

conc_2 A single numeric value for the desired concentration of the second drug.

metrics_merged A data.table containing all pre-calculated curve fit parameters. Expects columns: 'dilution_drug', 'cotrt_value', 'ratio', 'ec50', 'h', 'x_inf', 'x_0'.

Value

A single numeric value for the predicted 'smooth' response.

Examples

```
mae <- get_synthetic_data("combo_matrix")
se <- mae[[gDRutils::get_supported_experiments("combo")]]
dt_metrics <- gDRutils::convert_se_assay_to_dt(se[1, 1], "Metrics")[normalization_type == "RV"]
predict_smooth_from_combo(conc_1 = 1.2, conc_2 = 9.8, metrics_merged = dt_metrics)
```

prettify_flat_metrics *Prettify metric names in flat 'Metrics' assay*

Description

Map existing column names of a flattened 'Metrics' assay to prettified names.

Usage

```
prettify_flat_metrics(
  x,
  human_readable = FALSE,
  normalization_type = c("GR", "RV")
)
```

Arguments

`x` character vector of names to prettify.

`human_readable` boolean indicating whether or not to return column names in human readable format. Defaults to FALSE.

`normalization_type` character vector with a specified normalization type. Defaults to `c("GR", "RV")`.

Details

A common use case for this function is to prettify column names from a flattened version of the "Metrics" assay. Mode `"human_readable" = TRUE` is often used for prettification in the context of front-end applications, whereas `"human_readable" = FALSE` is often used for prettification in the context of the command line.

Value

character vector of prettified names.

Examples

```
x <- c("CellLineName", "Tissue", "Primary Tissue", "GR_gDR_x_mean", "GR_gDR_xc50", "RV_GDS_x_mean")
prettify_flat_metrics(x, human_readable = FALSE)
```

process_batch	<i>Process and save a batch of results.</i>
---------------	---

Description

Process and save a batch of results.

Usage

```
process_batch(
  batch,
  start_index,
  fun_name,
  unique_id,
  total_iterations,
  temp_dir,
  FUN,
  ...
)
```

Arguments

batch	A subset of the vector or list x to be processed.
start_index	Integer indicating the starting index of the batch in the original vector x.
fun_name	Character string representing the name of the function FUN for use in file naming.
unique_id	String with unique identifier for the current task and user to ensure file uniqueness.
total_iterations	Integer indicating the total number of iterations in the original vector x.
temp_dir	Character string specifying the directory where batch results are saved.
FUN	A user-defined function to apply to each element of the batch.
...	Optional arguments passed to FUN.

Details

The function applies FUN to each element in batch, saves the results to a file named according to the format <fun_name>_<unique_id>_<start_index>_of_<total_iterations>_batch.qs2, and clears memory using gc() after saving.

Value

This function does not return a value. It saves the processed batch results to disk as a .qs2 file.

Examples

```
process_batch(list(1, 2, 3), 100, "my_function", "unique_task_id_user", 1000, tempdir(), function(x) x^2)
```

promote_fields	<i>Promote a nested field to be represented as a metadata field of the SummarizedExperiment as either the rowData or colData.</i>
----------------	---

Description

Promote a nested field to be represented as a metadata field of the SummarizedExperiment as either the rowData or colData.

Usage

```
promote_fields(se, fields, MARGIN = c(1, 2))
```

Arguments

se	SummarizedExperiment object.
fields	Character vector of nested fields in a BumpyMatrix object to promote to metadata fields of a se.
MARGIN	Numeric of values 1 or 2 indicating whether to promote fields to rows or columns respectively.

Details

Revert this operation using `demote_fields`.

Value

A `SummarizedExperiment` object with new dimensions resulting from promoting given fields.

See Also

`demote_fields`

Examples

```
mae <- get_synthetic_data("finalMAE_small.qs2")
se <- mae[[1]]
se <- promote_fields(se, "ReadoutValue", 2)
```

`refine_coldata`

refine colData

Description

current improvements done on the `colData` as a standardization step:

- set default value for optional `colData` fields

Usage

```
refine_coldata(cd, se, default_v = "Undefined")
```

Arguments

<code>cd</code>	<code>DataFrame</code> with <code>colData</code>
<code>se</code>	a <code>SummarizedExperiment</code> object with drug-response data generate by gDR pipeline
<code>default_v</code>	string with default value for optional columns in <code>colData</code>

Value

refined `colData`

Examples

```
mae <- get_synthetic_data("finalMAE_small.qs2")
refine_coldata(SummarizedExperiment::colData(mae[[1]]), mae[[1]])
```

refine_rowdata	<i>refine rowData</i>
----------------	-----------------------

Description

current improvements done on the rowData as a standardization step:

- set default value for optional rowData fields

Usage

```
refine_rowdata(rd, se, default_v = "Undefined")
```

Arguments

rd	DataFrame with rowData
se	a SummarizedExperiment object with drug-response data generate by gDR pipeline
default_v	string with default value for optional columns in rowData

Value

refined rowData

Examples

```
mae <- get_synthetic_data("finalMAE_small.qs2")
refine_rowdata(SummarizedExperiment::colData(mae[[1]]), mae[[1]])
```

register_fit_profile	<i>Register or update a fit profile</i>
----------------------	---

Description

Adds a new experiment-type profile or overwrites an existing one. The profile is available immediately in the current R session and is picked up by [apply_fit](#) and [apply_fits](#).

Usage

```
register_fit_profile(
  name,
  slicing_cols,
  slicing_values = NULL,
  input_assay,
  description = ""
)
```

Arguments

name	string; unique profile identifier (e.g. "biochemical").
slicing_cols	character vector of column names used to split each BumpyMatrix cell into sub-experiments.
slicing_values	character vector of values to iterate, or NULL to iterate all unique values found in each cell.
input_assay	string; name of the default source assay.
description	string; human-readable description (optional).

Value

Invisibly returns the registered profile list.

Examples

```
register_fit_profile(
  "biochemical",
  slicing_cols = "assay_type",
  slicing_values = c("Ki", "IC50"),
  input_assay = "Biochemical",
  description = "Biochemical activity assay (Ki / IC50)"
)
get_fit_profile("biochemical")
```

remove_codrug_data	<i>Remove Codrug Data</i>
--------------------	---------------------------

Description

Remove Codrug Data

Usage

```
remove_codrug_data(
  data,
  prettify_identifiers = TRUE,
  codrug_identifiers = c("drug_name2", "concentration2")
)
```

Arguments

data	data.table with input data
prettify_identifiers	logical flag specifying if identifiers are expected to be prettified
codrug_identifiers	character vector with identifiers for the codrug columns

Value

data.table without combination columns

Examples

```
dt <-
  data.table::data.table(
    "Drug Name" = letters[seq_len(3)],
    "Concentration" = seq_len(3),
    "Drug Name 2" = letters[4:6],
    "Concentration 2" = 4:6
  )
dt
remove_codrug_data(dt)
```

remove_drug_batch	<i>Remove batch substring from drug id</i>
-------------------	--

Description

Gnumber, i.e. "G12345678" is currently the default format of drug_id. It's also used as a drug name in some cases.

Usage

```
remove_drug_batch(
  drug_vec,
  drug_p = "^G[0-9]{8}",
  sep_p = "[^0-9|^_]",
  batch_p = ".+"
)
```

Arguments

drug_vec	atomic vector (e.g., character or integer) with drug id(s)
drug_p	string with regex pattern for drug id. Set to Gnumber format by default: "G[0-9]{8}".
sep_p	string with regex pattern for separator. Set to any character except for digit and space
batch_p	string with regex pattern for batch substring. By default set to any character(s): ".+"

Details

By default, Gnumber(s) followed by any character (except for underscore and any digit) and any batch substring are cleaned:

- G00060245.18 => G00060245
- G00060245.1-8 => G00060245
- G02948263.1-1.DMA => G02948263
- Gnumber followed by the codrug
 - G03252046.1-2;G00376771 => G03252046
- Gnumber followed by the two codrugs
 - G03256376.1-2;G00376771.1-19;G02557755 => G03256376
- Gnumber followed by the drug name
 - G00018838, Cisplatin => G00018838

By default, Gnumber(s) followed by the "_" or digit (regardless the batch substring) are not cleaned:

- Gnumber with suffix added to prevent duplicated ids
 - G00060245_(G00060245.1-8)
- too long Gnumber
 - G123456789.1-12

Value

charvec with Gnumber(s)

Examples

```
remove_drug_batch("G00060245.18")
remove_drug_batch("G00060245.1-8")
remove_drug_batch("G00060245.1-1.DMA")

remove_drug_batch("G03252046.1-2;G00376771")
remove_drug_batch("G00018838, Cisplatin")
remove_drug_batch("G03256376.1-2;G00376771.1-19;G02557755")
remove_drug_batch("G00060245_(G00060245.1-8)")
remove_drug_batch(c("G00060245.18", "G00060245.1-8", "G00060245.1-1.DMA"))

remove_drug_batch("DRUG_01.123", drug_p = "DRUG_[0-9]+")
remove_drug_batch("G00001234:22-1", sep_p = ":")
remove_drug_batch("G00001234.28", batch_p = "[0-9]+")
```

rename_bumpy	<i>Rename BumpyMatrix</i>
--------------	---------------------------

Description

Rename BumpyMatrix

Usage

```
rename_bumpy(bumpy, mapping_vector)
```

Arguments

bumpy	a BumpyMatrix object
mapping_vector	a named vector for mapping old and new values. The names of the character vector indicate the source names, and the corresponding values the destination names.

Value

a renamed BumpyMatrix object

Examples

```
mae <- get_synthetic_data("finalMAE_small.qs2")
se <- mae[[1]]
assay <- SummarizedExperiment::assay(se)
rename_bumpy(assay, c("Concentration" = "conc"))
```

rename_DFrame	<i>Rename DFrame</i>
---------------	----------------------

Description

Rename DFrame

Usage

```
rename_DFrame(df, mapping_vector)
```

Arguments

df	a DFrame object
mapping_vector	a named vector for mapping old and new values. The names of the character vector indicate the source names, and the corresponding values the destination names.

Value

a renamed DFrame object

Examples

```
mae <- get_synthetic_data("finalMAE_small.qs2")
rename_DFrame(SummarizedExperiment::rowData(mae[[1]]), c("Gnumber" = "Gnumber1"))
```

round_concentration	<i>Round concentration to ndigit significant digits</i>
---------------------	---

Description

Round concentration to ndigit significant digits

Usage

```
round_concentration(x, ndigit = 3)
```

Arguments

- x value to be rounded.
- ndigit number of significant digits (default = 4).

Value

rounded x

Examples

```
round_concentration(x = c(0.00175,0.00324,0.0091), ndigit = 1)
```

SE_metadata	<i>Get and set metadata for parameters on a SummarizedExperiment object.</i>
-------------	--

Description

Set metadata for the fitting parameters that define the Metrics assay in SummarizedExperiment object metadata.

Usage

```
set_SE_fit_parameters(se, value)

set_SE_processing_metadata(se, value)

set_SE_keys(se, value)

set_SE_experiment_metadata(se, value, append = TRUE)

set_SE_experiment_raw_data(se, value)

get_SE_fit_parameters(se)

get_SE_processing_metadata(se)

get_SE_experiment_raw_data(se)

get_SE_experiment_metadata(se)

get_SE_keys(se, key_type = NULL)

get_SE_identifiers(se, id_type = NULL, simplify = TRUE)

set_SE_identifiers(se, value)
```

Arguments

se	a SummarizedExperiment object for which to add fit parameter metadata.
value	named list of metadata for fit parameters.
append	Boolean indicating whether to append the new metadata value to the existing entry.
key_type	string of a specific key type (i.e. 'nested_keys', etc.).
id_type	string of a specific id type (i.e. 'duration', 'cellline_name', etc.).
simplify	Boolean indicating whether output should be simplified.

Details

For `*et_SE_processing_metadata`, get/set metadata for the processing info that defines the `date_processed` and `packages_versions` in `SummarizedExperiment` object metadata. For `*et_SE_fit_parameters`, get/set metadata for fit parameters used to construct the `Metrics` assay in a `SummarizedExperiment` object.

Value

se with added metadata.

Examples

```

mae <- get_synthetic_data("finalMAE_small.qs2")
se <- mae[[1]]
get_SE_fit_parameters(se)

mae <- get_synthetic_data("finalMAE_small.qs2")
se <- mae[[1]]
meta <- get_SE_processing_metadata(se)

mae <- get_synthetic_data("finalMAE_small.qs2")
se <- mae[[1]]
get_SE_experiment_raw_data(se)

mae <- get_synthetic_data("finalMAE_small.qs2")
se <- mae[[1]]
get_SE_experiment_metadata(se)

mae <- get_synthetic_data("finalMAE_small.qs2")
se <- mae[[1]]
get_SE_identifiers(se)

```

```
set_constant_fit_params
```

Set fit parameters for a constant fit.

Description

Replace values for flat fits: $ec_{50} = 0$, $h = 0.0001$ and $xc_{50} = +/- Inf$

Usage

```
set_constant_fit_params(out, mean_norm_value)
```

Arguments

out	Named list of fit parameters.
mean_norm_value	Numeric value that be used to set all parameters that can be calculated from the mean.

Value

Modified named list of fit parameters.

Examples

```

na <- list(x_0 = NA)
set_constant_fit_params(na, mean_norm_value = 0.6)

```

set_unique_cl_names	<i>Set Unique Parental Identifiers</i>
---------------------	--

Description

This function sets the CellLineName field in colData to be unique by appending the clid in parentheses for duplicates.

Usage

```
set_unique_cl_names(se)
```

Arguments

se A SummarizedExperiment object.

Value

A SummarizedExperiment object with unique CellLineName in colData.

Examples

```
se <- SummarizedExperiment::SummarizedExperiment(
  assays = list(counts = matrix(1:4, ncol = 2)),
  colData = S4Vectors::DataFrame(CellLineName = c("ID1", "ID1"), clid = c("C1", "C2"))
)
se <- set_unique_cl_names(se)
```

set_unique_cl_names_dt

Set unique primary cell line identifiers in the table

Description

This function sets the primary cell line field in data.frame-like object to be unique by appending the secondary cell line field in parentheses for duplicates.

Usage

```
set_unique_cl_names_dt(
  dt,
  primary_name = get_env_identifiers("cellline_name"),
  secondary_name = get_env_identifiers("cellline"),
  sep = " "
)
```

Arguments

dt data.table, data.frame or DFrame with the data

primary_name string with the name of the primary cell line field

secondary_name string with the name of the secondary cell line field

sep string with separator added before suffix

Value

fixed input table with unique primary cell line field in dt

Examples

```
col_data <- S4Vectors::DataFrame(CellLineName = c("ID1", "ID1"), clid = c("C1", "C2"))
col_data <- set_unique_cl_names_dt(col_data)
```

set_unique_drug_names *Set Unique Drug Names*

Description

This function sets the DrugName, DrugName_2, and DrugName_3 fields in rowData to be unique by appending the corresponding Gnumber, Gnumber_2, and Gnumber_3 in parentheses for duplicates.

Usage

```
set_unique_drug_names(se)
```

Arguments

se A SummarizedExperiment object.

Value

A SummarizedExperiment object with unique DrugName fields in rowData.

Examples

```
se <- SummarizedExperiment::SummarizedExperiment(
  assays = list(counts = matrix(1:9, ncol = 3)),
  rowData = S4Vectors::DataFrame(DrugName = c("DrugA", "DrugA", "DrugB"),
    Gnumber = c("G1", "G2", "G5"),
    DrugName_2 = c("DrugC", "DrugC", "DrugD"),
    Gnumber_2 = c("G3", "G4", "G5"))
)
se <- set_unique_drug_names(se)
```

set_unique_drug_names_dt

Set unique primary drug identifiers in the table

Description

This function sets the primary drug field(s) in data.frame-like object to be unique by appending the secondary drug field(s) in parentheses for duplicates. By default DrugName, DrugName_2, and DrugName_3 are primary drug fields, while Gnumber, Gnumber_2, and Gnumber_3 are their respective secondary drug fields.

Usage

```
set_unique_drug_names_dt(
  dt,
  primary_names = unlist(get_env_identifiers()[c("drug_name", "drug_name2",
    "drug_name3"))]),
  secondary_names = unlist(get_env_identifiers()[c("drug", "drug2", "drug3"))]),
  sep = " "
)
```

Arguments

dt	data.table, data.frame or DFrame with the data
primary_names	charvec with the names of the primary drug field(s)
secondary_names	charvec with the name of the secondary drug field(s)
sep	string with separator added before suffix

Value

fixed input table with unique primary drug field in dt

Examples

```
row_data <- S4Vectors::DataFrame(
  DrugName = c("DrugA", "DrugA", "DrugB"),
  Gnumber = c("G1", "G2", "G5"),
  DrugName_2 = c("DrugC", "DrugC", "DrugD"),
  Gnumber_2 = c("G3", "G4", "G5")
)
row_data <- set_unique_drug_names_dt(row_data)
```

set_unique_identifiers

Set Unique Identifiers in MultiAssayExperiment

Description

This function sets the CellLineName in colData and DrugName fields in rowData to be unique for each SummarizedExperiment in a MultiAssayExperiment.

Usage

```
set_unique_identifiers(mae)
```

Arguments

mae A MultiAssayExperiment object.

Value

A MultiAssayExperiment object with unique identifiers.

Examples

```
se1 <- SummarizedExperiment::SummarizedExperiment(
  assays = list(counts = matrix(1:4, ncol = 2)),
  colData = S4Vectors::DataFrame(parental_identifier = c("ID1", "ID1"), clid = c("C1", "C2")),
  rowData = S4Vectors::DataFrame(DrugName = c("DrugA", "DrugA"), Gnumber = c("G1", "G2"))
)
rownames(SummarizedExperiment::colData(se1)) <- c("c11", "c12")
rownames(SummarizedExperiment::rowData(se1)) <- c("g1", "g")
se2 <- SummarizedExperiment::SummarizedExperiment(
  assays = list(counts = matrix(5:8, ncol = 2)),
  colData = S4Vectors::DataFrame(parental_identifier = c("ID2", "ID2"), clid = c("C3", "C4")),
  rowData = S4Vectors::DataFrame(DrugName = c("DrugB", "DrugB"), Gnumber = c("G3", "G4"))
)
rownames(SummarizedExperiment::colData(se2)) <- c("c13", "c14")
rownames(SummarizedExperiment::rowData(se2)) <- c("g3", "g4")
mae <- MultiAssayExperiment::MultiAssayExperiment(experiments = list(se1 = se1, se2 = se2))
mae <- set_unique_identifiers(mae)
```

set_unique_names_dt	<i>Set unique primary identifiers in the data.frame-like objects</i>
---------------------	--

Description

This function sets the primary field in the data.frame-like objects to be unique by appending the secondary field in parentheses for duplicates.

Usage

```
set_unique_names_dt(dt, primary_name, secondary_name, sep = " ")
```

Arguments

dt	data.table, data.frame or DFrame with data
primary_name	string with the name of the primary field
secondary_name	string with the name of the secondary field
sep	string with separator added before suffix

Value

fixed input table with unique primary field in the table

Examples

```
col_data <- S4Vectors::DataFrame(CellLineName = c("ID1", "ID1"), clid = c("C1", "C2"))
col_data <- set_unique_names_dt(col_data, primary_name = "CellLineName", secondary_name = "clid")
```

shorten_normalization_type_name	<i>shorten normalization type</i>
---------------------------------	-----------------------------------

Description

shorten normalization type

Usage

```
shorten_normalization_type_name(x)
```

Arguments

x	string with normalization type
---	--------------------------------

Value

shortened string representing the normalization type

Examples

```
shorten_normalization_type_name("GRvalue")
```

split_big_table_for_xlsx
<i>Split big table</i>

Description

Helper function for saving big tables in an Excel file. Excel has a sheet size limit, if the table is too large it will not be possible to save such a file. This function allows you to split the table into smaller parts so that saving can be possible

Usage

```
split_big_table_for_xlsx(dt_list, max_row = 1e+06, max_col = 16000)
```

Arguments

dt_list	list of data.tables. Each data.table will be checked and split if meet the criteria
max_row	integer defining the maximum number of rows in one sheet, the rows will be divided into portions of this size. Default value, 1 000 000, is based on excel limit - 1 048 576 with extra safety margin
max_col	integer defining the maximum number of columns in one sheet, the columns will be divided into portions of this size. Default value, 16 000, is based on excel limit - 16 384 with extra safety margin

Value

list of data.tables

Examples

```
too_large_dt <- list(data.table::data.table(matrix(seq_len(300)), nrow = 10))
split_big_table_for_xlsx(too_large_dt, max_row = 250)
```

split_SE_components	<i>split_SE_components</i>
---------------------	----------------------------

Description

Divide the columns of an input `data.table` into treatment metadata, condition metadata, experiment metadata, and assay data for further analysis. This will most commonly be used to identify the different components of a [SummarizedExperiment](#) object.

Usage

```
split_SE_components(df_, nested_keys = NULL, combine_on = 1L)
```

Arguments

<code>df_</code>	<code>data.table</code> with drug-response data
<code>nested_keys</code>	character vector of keys to exclude from the row or column metadata, and to instead nest within an element of the matrix. See details.
<code>combine_on</code>	integer value of 1 or 2, indicating whether unrecognized columns should be combined on row or column respectively. Defaults to 1.

Details

Named list containing the following elements:

"treatment_md": treatment metadata

"condition_md": condition metadata

"data_fields": all `data.table` column names corresponding to fields nested within a `BumpyMatrix` cell

"experiment_md": metadata that is constant for all entries of the `data.table`

"identifiers_md": key identifier mappings

The `nested_keys` provides the user the opportunity to specify that they would not like to use that metadata field as a differentiator of the treatments, and instead, incorporate it into a nested `DataFrame` in the `BumpyMatrix` matrix object.

In the event that if any of the `nested_keys` are constant throughout the whole `data.table`, they will still be included in the `DataFrame` of the `BumpyMatrix` and not in the `experiment_metadata`.

Columns within the `df_` will be identified through the following logic: First, the known data fields and any specified `nested_keys` are extracted. Following that, known cell and drug metadata fields are detected, and any remaining columns that represent constant metadata fields across all rows are extracted. Next, any cell line metadata will be heuristically extracted. Finally, all remaining columns will be combined on either the rows or columns as specified by `combine_on`.

Value

named list containing different elements of a [SummarizedExperiment](#); see details.

Examples

```
split_SE_components(data.table::data.table(clid = "CL1", Gnumber = "DrugA"))
```

standardize_mae	<i>Standardize MAE by switching from custom identifiers into gDR-default</i>
-----------------	--

Description

Standardize MAE by switching from custom identifiers into gDR-default

Usage

```
standardize_mae(mae, use_default = TRUE)
```

Arguments

mae	a MultiAssayExperiment object with drug-response data generate by gDR pipeline
use_default	boolean indicating whether or not to use default identifiers for standardization

Value

mae a MultiAssayExperiment with default gDR identifiers

Examples

```
mae <- get_synthetic_data("finalMAE_small.qs2")
S4Vectors::metadata(mae[[1]])$identifiers$drug <- "druug"
standardize_mae(mae)
```

standardize_se	<i>Standardize SE by switching from custom identifiers into gDR-default</i>
----------------	---

Description

Standardize SE by switching from custom identifiers into gDR-default

Usage

```
standardize_se(se, use_default = TRUE)
```

Arguments

se	a SummarizedExperiment object with drug-response data generate by gDR pipeline
use_default	boolean indicating whether or not to use default identifiers for standardization

Value

se a SummarizedExperiment with default gDR identifiers

Examples

```
mae <- get_synthetic_data("finalMAE_small.qs2")
se <- mae[[1]]
S4Vectors::metadata(se)$identifiers$drug <- "drug"
standardize_se(se)
```

throw_msg_if_duplicates

throw message if assay data.table contains duplicated rows

Description

An auxiliary function that checks for duplicated rows in assay data.table, In case of duplicates it throws a message. The message function is by default stop() The message function can be customized with msg_f parameter

Usage

```
throw_msg_if_duplicates(
  dt,
  assay_name = "unknown",
  msg_f = stop,
  preview_max_numb = 4
)
```

Arguments

dt	data.table with assay data
assay_name	string with the name of the assay
msg_f	function to be used to throw the message
preview_max_numb	number of rows to preview if duplicates found

Examples

```
sdata <- get_synthetic_data("finalMAE_small.qs2")
smetrics_data <- convert_se_assay_to_dt(sdata[[1]], "Metrics")
throw_msg_if_duplicates(smetrics_data, assay_name = "Metrics", msg_f = futile.logger::flog.info)
```

`update_env_idfs_from_mae`*Update environment identifiers from MAE object identifiers*

Description

Update environment identifiers from MAE object identifiers

Usage

```
update_env_idfs_from_mae(mae_idfs)
```

Arguments

`mae_idfs` A list containing MAE identifiers

Value

NULL

Examples

```
update_env_idfs_from_mae(list(get_env_identifiers()))
```

`update_idfs_synonyms` *Update gDR synonyms for the identifier*

Description

Update gDR synonyms for the identifier

Usage

```
update_idfs_synonyms(data, dict = get_idfs_synonyms())
```

Arguments

`data` list of charvec with identifiers data
`dict` list with dictionary

Value

list

Examples

```
mdict <- list(duration = "time")
iv <- c("Time", "Duration", "time")
update_idfs_synonyms(iv, dict = mdict)
```

`validate_identifiers` *Check that specified identifier values exist in the data.*

Description

Check that specified identifier values exist in the data and error otherwise.

Usage

```
validate_identifiers(
  df,
  identifiers = NULL,
  req_ids = NULL,
  exp_one_ids = NULL
)
```

Arguments

<code>df</code>	data.table with colnames.
<code>identifiers</code>	Named list of identifiers where the names are standardized identifier names. If not passed, defaults to <code>get_env_identifiers()</code> .
<code>req_ids</code>	Character vector of standardized identifier names required to pass identifier validation.
<code>exp_one_ids</code>	Character vector of standardized identifiers names where only one identifier value is expected. If not passed, defaults to <code>get_expect_one_identifiers()</code> .

Details

Note that this does NOT set the identifiers anywhere (i.e. environment or SummarizedExperiment object). If identifiers do not validate, will throw error as side effect.

Value

Named list of identifiers modified to pass validation against the input data. Errors with explanatory message if validation cannot pass with the given identifiers and data.

Examples

```
validate_identifiers(
  S4Vectors::DataFrame("Barcode" = NA, "Duration" = NA, "Template" = NA, "clid" = NA),
  req_ids = "barcode"
)
```

validate_json	<i>Validate JSON against a schema.</i>
---------------	--

Description

Validate JSON describing an object against a schema.

Usage

```
validate_json(json, schema_path)
```

Arguments

json	String of JSON in memory.
schema_path	String of the schema to validate against.

Details

This is most often used to validate JSON before passing it in as a document to an ElasticSearch index.

Value

Boolean of whether or not JSON successfully validated.

Examples

```
json <- '{}'
```

validate_MAE	<i>Validate MultiAssayExperiment object</i>
--------------	---

Description

Function validates correctness of SE included in MAE by checking multiple cases:

- detection of duplicated rowData/colData,
- incompatibility of rownames/colnames,
- occurrence of necessary assays,
- detection of mismatch of CLIDs inside colData and colnames (different order),
- correctness of metadata names.

Usage

```
validate_MAE(mae)
```

Arguments

mae MultiAssayExperiment object produced by the gDR pipeline

Value

NULL invisibly if the MultiAssayExperiment is valid. Throws an error if the MultiAssayExperiment is not valid.

Author(s)

Bartosz Czech czech.bartosz@external.gene.com

Examples

```
mae <- get_synthetic_data("finalMAE_small.qs2")
validate_MAE(mae)
```

```
validate_mae_with_schema
```

Validate MAE against a schema.

Description

Validate MAE object against a schema. Currently only SEs are validated TODO: add mae.json schema and validate full MAE object

Usage

```
validate_mae_with_schema(
  mae,
  schema_package = Sys.getenv("SCHEMA_PACKAGE", "gDRutils"),
  schema_dir_path = Sys.getenv("SCHEMA_DIR_PATH", "schemas"),
  schema = c(se = "se.json", mae = "mae.json")
)
```

Arguments

mae MultiAssayExperiment object

schema_package string name of the package with JSON schema files

schema_dir_path path to the dir with JSON schema files

schema named charvec with filenames of schemas to validate against.

Value

Boolean of whether or not mae is valid

Examples

```
mae <- get_synthetic_data("finalMAE_small.qs2")
validate_mae_with_schema(mae)
```

 validate_SE

Validate SummarizedExperiment object

Description

Function validates correctness of SE by checking multiple cases:

- detection of duplicated rowData/colData,
- incompatibility of rownames/colnames,
- occurrence of necessary assays,
- detection of mismatch of CLIDs inside colData and colnames (different order),
- correctness of metadata names.

Usage

```
validate_SE(se, expect_single_agent = FALSE)
```

Arguments

se SummarizedExperiment object produced by the gDR pipeline

expect_single_agent a logical indicating if the function should check whether the SummarizedExperiment is single-agent data

Value

NULL invisibly if the SummarizedExperiment is valid. Throws an error if the SummarizedExperiment is not valid.

Examples

```
mae <- get_synthetic_data("finalMAE_small.qs2")
se <- mae[[1]]
validate_SE(se)
```

`validate_se_assay_name`

Check whether or not an assay exists in a SummarizedExperiment object.

Description

Check for the presence of an assay in a SummarizedExperiment object.

Usage

```
validate_se_assay_name(se, name)
```

Arguments

<code>se</code>	A SummarizedExperiment object.
<code>name</code>	String of name of the assay to validate.

Value

NULL invisibly if the assay name is valid. Throws an error if the assay is not valid.

Examples

```
mae <- get_synthetic_data("finalMAE_small.qs2")
se <- mae[[1]]
validate_se_assay_name(se, "RawTreated")
```

Index

- * **SE_operators**
 - aggregate_assay, [7](#)
 - demote_fields, [29](#)
 - get_MAE_identifiers, [49](#)
 - identify_unique_se_metadata_fields, [59](#)
 - merge_assay, [68](#)
 - merge_MAE, [69](#)
 - merge_metadata, [70](#)
 - merge_SE, [71](#)
 - promote_fields, [79](#)
 - SE_metadata, [86](#)
 - split_SE_components, [95](#)
- * **assay_names**
 - get_assay_names, [38](#)
 - get_combo_assay_names, [39](#)
 - get_combo_base_assay_names, [40](#)
 - get_combo_score_assay_names, [41](#)
 - get_env_assay_names, [43](#)
- * **combination_data**
 - convert_combo_data_to_dt, [19](#)
 - convert_combo_field_to_assay, [20](#)
 - define_matrix_grid_positions, [28](#)
 - get_additional_variables, [37](#)
 - get_combo_excess_field_names, [41](#)
 - get_combo_score_field_names, [42](#)
 - has_single_codrug_data, [55](#)
 - has_valid_codrug_data, [56](#)
 - is_combo_data, [61](#)
 - remove_codrug_data, [82](#)
 - round_concentration, [86](#)
- * **convert**
 - convert_mae_assay_to_dt, [20](#)
 - convert_se_assay_to_custom_dt, [24](#)
 - convert_se_assay_to_dt, [25](#)
 - df_to_bm_assay, [30](#)
 - flatten, [35](#)
- * **duplicates**
 - get_assay_dt_duplicated_rows, [37](#)
 - get_assay_req_uniq_cols, [39](#)
 - get_duplicated_rows, [43](#)
 - has_assay_dt_duplicated_rows, [54](#)
 - has_dt_duplicated_rows, [55](#)
 - throw_msg_if_duplicates, [97](#)
- * **experiment**
 - get_experiment_groups, [45](#)
 - get_supported_experiments, [52](#)
 - validate_MAE, [100](#)
 - validate_SE, [102](#)
 - validate_se_assay_name, [103](#)
- * **fit_curves**
 - .set_invalid_fit_params, [5](#)
 - cap_xc50, [17](#)
 - fit_curves, [31](#)
 - logisticFit, [63](#)
 - predict_conc_from_efficacy, [75](#)
 - predict_efficacy_from_conc, [76](#)
 - predict_smooth_from_combo, [77](#)
 - set_constant_fit_params, [88](#)
- * **identifiers**
 - get_default_identifiers, [42](#)
 - get_expect_one_identifiers, [45](#)
 - get_identifiers_dt, [48](#)
 - get_idfs_synonyms, [48](#)
 - get_required_identifiers, [50](#)
 - headers, [57](#)
 - identifiers, [58](#)
 - prettify_flat_metrics, [77](#)
 - update_env_idfs_from_mae, [98](#)
 - update_idfs_synonyms, [98](#)
 - validate_identifiers, [99](#)
- * **json_const**
 - get_isobologram_columns, [49](#)
 - get_settings_from_json, [51](#)
- * **json_convert**
 - convert_colData_to_json, [18](#)
 - convert_mae_to_json, [22](#)
 - convert_metadata_to_json, [22](#)

- convert_rowData_to_json, 23
- convert_se_to_json, 27
- validate_mae_with_schema, 101
- * **json_validate**
 - validate_json, 100
- * **metadata_management**
 - addClass, 6
 - modifyData, 72
- * **metrics**
 - apply_fit, 8
 - apply_fit_to_se, 10
 - apply_fits, 11
 - persist_fit_assay, 74
- * **package_utils**
 - .standardize_conc, 5
 - apply_bumpy_function, 7
 - assert_choices, 13
 - average_biological_replicates_dt, 14
 - calc_sd, 15
 - cap_assay_infinities, 16
 - extend_normalization_type_name, 30
 - get_env_var, 44
 - get_gDR_session_info, 47
 - get_non_empty_assays, 50
 - get_synthetic_data, 52
 - is_any_exp_empty, 60
 - is_exp_empty, 61
 - is_mae_empty, 62
 - loop, 65
 - MAEapply, 66
 - map_conc_to_standardized_conc, 67
 - mcolData, 68
 - mrowData, 73
 - process_batch, 78
 - remove_drug_batch, 83
 - shorten_normalization_type_name, 93
 - split_big_table_for_xlsx, 94
- * **standardize_MAE**
 - refine_coldata, 80
 - refine_rowdata, 81
 - rename_bumpy, 85
 - rename_DFrame, 85
 - set_unique_cl_names, 89
 - set_unique_cl_names_dt, 89
 - set_unique_drug_names, 90
 - set_unique_drug_names_dt, 91
 - set_unique_identifiers, 92
 - set_unique_names_dt, 93
 - standardize_mae, 96
 - standardize_se, 96
- * **test_helpers**
 - gen_synthetic_data, 36
 - get_testdata, 53
 - get_testdata_codilution, 53
 - get_testdata_combo, 54
 - .set_invalid_fit_params, 5
 - .standardize_conc, 5
- addClass, 6
- aggregate_assay, 7
- apply_bumpy_function, 7
- apply_fit, 8, 11, 12, 32, 47, 81
- apply_fit_to_se, 10, 32
- apply_fits, 11, 47, 81
- as.list, 65
- assert_choices, 13
- average_biological_replicates_dt, 14
- calc_sd, 15
- cap_assay_infinities, 16
- cap_xc50, 17, 33, 34
- convert_colData_to_json, 18
- convert_combo_data_to_dt, 19
- convert_combo_field_to_assay, 20
- convert_mae_assay_to_dt, 20
- convert_mae_to_json, 22
- convert_metadata_to_json, 22
- convert_rowData_to_json, 23
- convert_se_assay_to_custom_dt, 24
- convert_se_assay_to_dt, 25
- convert_se_to_json, 27
- define_matrix_grid_positions, 28
- demote_fields, 29
- df_to_bm_assay, 30
- extend_normalization_type_name, 30
- fit_curves, 31
- fit_drug_response_metrics, 32, 34, 35
- fit_drug_response_metrics_4p, 33, 34
- flatten, 35
- gen_synthetic_data, 36
- get_additional_variables, 37
- get_assay_dt_duplicated_rows, 37

get_assay_names, 38
get_assay_req_uniq_cols, 39
get_combo_assay_names, 39
get_combo_base_assay_names, 40
get_combo_excess_field_names, 41
get_combo_score_assay_names, 41
get_combo_score_field_names, 42
get_default_identifiers, 42
get_duplicated_rows, 43
get_env_assay_names, 43
get_env_identifiers(identifiers), 58
get_env_var, 44
get_expect_one_identifiers, 45
get_experiment_groups, 45
get_fit_profile, 46
get_fit_profiles, 47
get_gDR_session_info, 47
get_header(headers), 57
get_identifiers_dt, 48
get_ids_synonyms, 48
get_isobologram_columns, 49
get_MAE_identifiers, 49
get_non_empty_assays, 50
get_prettified_identifiers
(identifiers), 58
get_required_identifiers, 50
get_SE_experiment_metadata
(SE_metadata), 86
get_SE_experiment_raw_data
(SE_metadata), 86
get_SE_fit_parameters(SE_metadata), 86
get_SE_identifiers(SE_metadata), 86
get_SE_keys(SE_metadata), 86
get_SE_processing_metadata
(SE_metadata), 86
get_settings_from_json, 51
get_supported_experiments, 52
get_synthetic_data, 52
get_testdata, 53
get_testdata_codilution, 53
get_testdata_combo, 54

has_assay_dt_duplicated_rows, 54
has_dt_duplicated_rows, 55
has_single_codrug_data, 55
has_valid_codrug_data, 56
headers, 57

identifiers, 58

identify_unique_se_metadata_fields, 59
is_any_exp_empty, 60
is_combo_data, 61
is_exp_empty, 61
is_mae_empty, 62

lapply, 65
logisticFit, 32, 63
loop, 65

MAEapply, 66
map_conc_to_standardized_conc, 67
mcolData, 68
merge_assay, 68
merge_MAE, 69
merge_metadata, 70
merge_SE, 71
modifyData, 72
mrowData, 73
MultiAssayExperiment, 21

persist_fit_assay, 74
predict_conc_from_efficacy, 75
predict_efficacy_from_conc, 76
predict_smooth_from_combo, 77
prettify_flat_metrics, 77
process_batch, 78
promote_fields, 79

refine_coldata, 80
refine_rowdata, 81
register_fit_profile, 81
remove_codrug_data, 82
remove_drug_batch, 83
rename_bumpy, 85
rename_DFrame, 85
reset_env_identifiers(identifiers), 58
round_concentration, 86

SE_metadata, 86
set_constant_fit_params, 88
set_env_identifier(identifiers), 58
set_SE_experiment_metadata
(SE_metadata), 86
set_SE_experiment_raw_data
(SE_metadata), 86
set_SE_fit_parameters(SE_metadata), 86
set_SE_identifiers(SE_metadata), 86
set_SE_keys(SE_metadata), 86

set_SE_processing_metadata
 (SE_metadata), 86
set_unique_cl_names, 89
set_unique_cl_names_dt, 89
set_unique_drug_names, 90
set_unique_drug_names_dt, 91
set_unique_identifiers, 92
set_unique_names_dt, 93
shorten_normalization_type_name, 93
split_big_table_for_xlsx, 94
split_SE_components, 30, 95
standardize_mae, 96
standardize_se, 96
SummarizedExperiment, 8, 9, 11, 12, 20, 25,
 26, 61, 87, 95, 103

throw_msg_if_duplicates, 97

update_env_idfs_from_mae, 98
update_idfs_synonyms, 98

validate_identifiers, 99
validate_json, 100
validate_MAE, 100
validate_mae_with_schema, 101
validate_SE, 102
validate_se_assay_name, 103