

Package: gDRcore (via r-universe)

October 11, 2026

Type Package

Title Processing functions and interface to process and analyze drug dose-response data

Version 1.11.18

Date 2026-10-08

Description This package contains core functions to process and analyze drug response data. The package provides tools for normalizing, averaging, and calculation of gDR metrics data. All core functions are wrapped into the pipeline function allowing analyzing the data in a straightforward way.

License Artistic-2.0

Depends R (>= 4.2)

Imports BumpyMatrix, BiocParallel, checkmate, drc, futile.logger, gDRutils (>= 1.11.12), jsonlite, MultiAssayExperiment, purrr, stringr, S4Vectors, SummarizedExperiment, data.table

Suggests BiocStyle, gDRstyle (>= 1.7.1), gDRimport (>= 1.7.1), gDRtestData (>= 1.7.1), IRanges, knitr, pkgbuild, qs2, testthat, yaml

VignetteBuilder knitr

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

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

biocViews Software, ShinyApps

ByteCompile TRUE

DeploySubPath gDRcore

Encoding UTF-8

LazyLoad yes

NeedsCompilation yes

Roxygen list(markdown = TRUE)

SwitchrLibrary gDRcore

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 2026-10-08 14:24:55 UTC

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

RemoteRef HEAD

RemoteSha 04cc3e026e2eaa0b284b7bdabc2528bb7a6be6c7

Contents

annotate_dt_with_cell_line	3
annotate_dt_with_drug	4
annotate_mae_with_cell_line	4
annotate_mae_with_drug	5
annotate_se_with_cell_line	6
annotate_se_with_drug	7
apply_combo_excess	7
apply_combo_isobolograms	8
apply_combo_sa_fits	10
apply_combo_scores	11
average_SE	13
bliss_fit_fn	18
calculate_excess	19
calculate_GR_value	20
calculate_score	22
cleanup_metadata	23
compute_growth_rates	24
convert_mae_to_raw_data	26
convert_se_to_raw_data	26
data_model	27
data_model.character	27
data_model.data.table	28
fit_SE.combinations	28
fit_SE.timecourse	29
get_assays_per_pipeline_step	32
get_cell_line_annotation	32
get_cellline_annotation_from_dt	33
get_default_nested_identifiers	34
get_drug_annotation	34
get_drug_annotation_from_dt	35
get_period_timepoints	36
growth_rates_to_se	37
grr_matches	38

<i>annotate_dt_with_cell_line</i>	3
-----------------------------------	---

hss_fit_fn	39
identify_data_type	40
identify_keys	41
map_df	43
map_ids_to_fits	44
map_untreated	45
merge_data	45
prepare_input	46
prepare_input.data.table	47
prepare_input.MultiAssayExperiment	48
process_perturbations	49
replace_conc_with_standardized_conc	50
split_raw_data	51
test_synthetic_data	53

Index	54
--------------	-----------

annotate_dt_with_cell_line	
<i>annotate_dt_with_cell_line</i>	

Description

Annotate cell line data with the provided annotation table

Usage

```
annotate_dt_with_cell_line(data, cell_line_annotation, fill = "unknown")
```

Arguments

data	data.table with dose-response data
cell_line_annotation	data.table with cell line annotations
fill	string indicating how unknown cell lines should be filled in the DB

Value

data.table with annotated cell lines

Examples

```
data <- data.table::data.table(
  clid = c("CL1", "CL2", "CL3"),
  Gnumber = c("D1", "D2", "D3")
)
cell_line_annotation <- get_cell_line_annotation(data)
annotated_metadata <- annotate_dt_with_cell_line(data, cell_line_annotation)
```

```
annotate_dt_with_drug  annotate_dt_with_drug
```

Description

Annotate drug data with the provided annotation table

Usage

```
annotate_dt_with_drug(data, drug_annotation, fill = "unknown")
```

Arguments

<code>data</code>	data.table with dose-response data
<code>drug_annotation</code>	data.table with drug annotations
<code>fill</code>	string indicating how unknown drugs should be filled in the DB

Value

data.table with annotated drugs

Examples

```
data <- data.table::data.table(
  clid = c("CL1", "CL2", "CL3"),
  Gnumber = c("D1", "D2", "D3")
)
drug_annotation <- get_drug_annotation(data)
annotated_metadata <- annotate_dt_with_drug(data, drug_annotation)
```

```
annotate_mae_with_cell_line
      annotate_mae_with_cell_line
```

Description

Annotate MultiAssayExperiment object with cell line annotations

Usage

```
annotate_mae_with_cell_line(mae, cell_line_annotation, fill = "unknown")
```

Arguments

mae	MultiAssayExperiment object containing dose-response data
cell_line_annotation	data.table with cell line annotations
fill	string indicating how unknown cell lines should be filled in the DB

Value

MultiAssayExperiment object with annotated cell lines

Examples

```
mae <- MultiAssayExperiment::MultiAssayExperiment(
  experiments = list(exp1 = SummarizedExperiment::SummarizedExperiment(
    rowData = data.table::data.table(clid = c("CL1", "CL2", "CL3"))
  ))
)
cell_line_annotation <- get_cell_line_annotation(data.table::as.data.table(
  SummarizedExperiment::rowData(
    MultiAssayExperiment::experiments(mae)[[1]]))
)
annotated_mae <- annotate_mae_with_cell_line(mae, cell_line_annotation)
```

annotate_mae_with_drug

annotate_mae_with_drug

Description

Annotate MultiAssayExperiment object with drug annotations

Usage

```
annotate_mae_with_drug(mae, drug_annotation, fill = "unknown")
```

Arguments

mae	MultiAssayExperiment object containing dose-response data
drug_annotation	data.table with drug annotations
fill	string indicating how unknown drugs should be filled in the DB

Value

MultiAssayExperiment object with annotated drugs

Examples

```
mae <- MultiAssayExperiment::MultiAssayExperiment(
  experiments = list(exp1 = SummarizedExperiment::SummarizedExperiment(
    rowData = data.table::data.table(Gnumber = c("D1", "D2", "D3"))
  ))
)
drug_annotation <- get_drug_annotation(data.table::as.data.table(
  SummarizedExperiment::rowData(
    MultiAssayExperiment::experiments(mae)[[1]]))
)
annotated_mae <- annotate_mae_with_drug(mae, drug_annotation)
```

```
annotate_se_with_cell_line
      annotate_se_with_cell_line
```

Description

Annotate SummarizedExperiment object with cell line annotations

Usage

```
annotate_se_with_cell_line(se, cell_line_annotation, fill = "unknown")
```

Arguments

<code>se</code>	SummarizedExperiment object containing dose-response data
<code>cell_line_annotation</code>	data.table with cell line annotations
<code>fill</code>	string indicating how unknown cell lines should be filled in the DB

Value

SummarizedExperiment object with annotated cell lines

Examples

```
se <- SummarizedExperiment::SummarizedExperiment(
  rowData = data.table::data.table(clid = c("CL1", "CL2", "CL3"))
)
cell_line_annotation <- get_cell_line_annotation(data.table::as.data.table(SummarizedExperiment::rowData(se)))
annotated_se <- annotate_se_with_cell_line(se, cell_line_annotation)
```

annotate_se_with_drug *annotate_se_with_drug*

Description

Annotate SummarizedExperiment object with drug annotations

Usage

```
annotate_se_with_drug(se, drug_annotation, fill = "unknown")
```

Arguments

se	SummarizedExperiment object containing dose-response data
drug_annotation	data.table with drug annotations
fill	string indicating how unknown drugs should be filled in the DB

Value

SummarizedExperiment object with annotated drugs

Examples

```
se <- SummarizedExperiment::SummarizedExperiment(
  rowData = data.table::data.table(Gnumber = c("D1", "D2", "D3"))
)
drug_annotation <- get_drug_annotation(data.table::as.data.table(SummarizedExperiment::rowData(se)))
annotated_se <- annotate_se_with_drug(se, drug_annotation)
```

apply_combo_excess *apply_combo_excess*

Description

Compute per-point HSA and Bliss excess values for a combination SE, writing the result to the excess assay. This is **step 3** of the [fit_SE.combinations](#) pipeline.

Usage

```
apply_combo_excess(
  se,
  series_identifiers = NULL,
  normalization_types = c("GR", "RV"),
  averaged_assay = "Averaged",
  metrics_assay = "Metrics",
  excess_assay = "excess"
)
```

Arguments

`se` [SummarizedExperiment](#) with Averaged and Metrics assays.

`series_identifiers` character vector of length 2: concentration column names. Defaults to `get_default_nested_identifiers(...)`.

`normalization_types` character vector. Default `c("GR", "RV")`.

`averaged_assay` string. Default "Averaged".

`metrics_assay` string. Default "Metrics".

`excess_assay` string; name of the output assay. Default "excess".

Details

Requires the Averaged and Metrics assays to be present (the latter as produced by [apply_combo_sa_fits](#) or `fit_SE.combinations`).

Value

Updated SE with an `excess_assay` assay containing per-point smooth, `hsa_excess`, and `bliss_excess` columns.

See Also

[apply_combo_sa_fits](#), [apply_combo_isobolograms](#), [apply_combo_scores](#), `fit_SE.combinations`

Examples

```
mae <- gDRutils::get_synthetic_data("finalMAE_combo_matrix_small")
combo_se <- mae[[gDRutils::get_supported_experiments("combo")]]
combo_se_excess <- apply_combo_excess(combo_se[1, 1])
"excess" %in% SummarizedExperiment::assayNames(combo_se_excess)
```

`apply_combo_isobolograms`

apply_combo_isobolograms

Description

Compute Loewe combination index isobolograms for a combination SE, writing results to the `isobolograms` and `all_iso_points` assays. This is **step 4** of the [fit_SE.combinations](#) pipeline.

Usage

```
apply_combo_isobolograms(
  se,
  series_identifiers = NULL,
  normalization_types = c("GR", "RV"),
  averaged_assay = "Averaged",
  metrics_assay = "Metrics",
  isobolograms_assay = "isobolograms",
  iso_points_assay = "all_iso_points"
)
```

Arguments

se [SummarizedExperiment](#) with Averaged and Metrics assays.

series_identifiers character vector of length 2. Defaults to `get_default_nested_identifiers(se, ...)`.

normalization_types character vector. Default `c("GR", "RV")`.

averaged_assay string. Default "Averaged".

metrics_assay string. Default "Metrics".

isobolograms_assay string; name of the Loewe curves output assay. Default "isobolograms".

iso_points_assay string; name of the iso points output assay. Default "all_iso_points".

Details

Requires the Averaged and Metrics assays (as produced by [apply_combo_sa_fits](#) or `fit_SE.combinations`). Isobologram analysis is only performed when at least 9 combo concentration points are available — otherwise NA is written, matching `fit_SE.combinations` behaviour.

Value

Updated SE with `isobolograms_assay` and `iso_points_assay` assays.

See Also

[apply_combo_sa_fits](#), [apply_combo_excess](#), [apply_combo_scores](#), `fit_SE.combinations`

Examples

```
mae <- gDRutils::get_synthetic_data("finalMAE_combo_matrix_small")
combo_se <- mae[[gDRutils::get_supported_experiments("combo")]]
combo_se_iso <- apply_combo_isobolograms(combo_se[1, 1])
"isobolograms" %in% SummarizedExperiment::assayNames(combo_se_iso)
```

apply_combo_sa_fits	<i>apply_combo_sa_fits</i>
---------------------	----------------------------

Description

Fit single-agent dose-response curves for each co-treatment concentration in a combination SE and store the results in the Metrics assay. This is **step 1+2** of the [fit_SE.combinations](#) pipeline:

Usage

```
apply_combo_sa_fits(
  se,
  series_identifiers = NULL,
  normalization_types = c("GR", "RV"),
  averaged_assay = "Averaged",
  metrics_assay = "Metrics",
  fit_source = "gDR"
)
```

Arguments

se [SummarizedExperiment](#) with an Averaged assay (combination experiment).

series_identifiers character vector of length 2: column names for the two concentration axes (e.g. `c("Concentration", "Concentration_2")`). Defaults to `get_default_nested_identifiers(se, data_model("combination"))`.

normalization_types character vector of normalization types to process. Default `c("GR", "RV")`.

averaged_assay string; name of the input assay. Default "Averaged".

metrics_assay string; name of the output assay. Default "Metrics".

fit_source string recorded in the `fit_source` column. Default "gDR".

Details

1. Standardise concentrations and build the complete dose-response matrix.
2. Fit SA curves per co-treatment concentration via `fit_combo_cotreatments()` and `fit_combo_codilutions()`.
3. Compute **smooth** SA predictions at every combo point via `map_ids_to_fits()` and store them together with the SA fit parameters.

The resulting Metrics assay contains `dilution_drug`, `cotrt_value`, `ec50`, `h`, `x_inf`, `x_0`, and `normalization_type` — the same schema as produced by `fit_SE.combinations()`.

Value

Updated [SummarizedExperiment](#) with a `metrics_assay` assay containing SA fit parameters per co-treatment concentration.

See Also

[apply_combo_excess](#), [apply_combo_isobolograms](#), [apply_combo_scores](#), [fit_SE.combinations](#)

Examples

```
mae <- gDRutils::get_synthetic_data("finalMAE_combo_matrix_small")
combo_se <- mae[[gDRutils::get_supported_experiments("combo")]]
SummarizedExperiment::assays(combo_se) <-
  SummarizedExperiment::assays(combo_se)["Averaged"]
combo_se_fitted <- apply_combo_sa_fits(combo_se[1, 1])
"Metrics" %in% SummarizedExperiment::assayNames(combo_se_fitted)
```

apply_combo_scores	<i>apply_combo_scores</i>
--------------------	---------------------------

Description

Compute Bliss and HSA synergy scores for a combination SE, replicating the exact scoring logic of [fit_SE.combinations](#).

Usage

```
apply_combo_scores(
  se,
  scores_assay = "scores",
  averaged_assay = "Averaged",
  metrics_assay = "Metrics",
  excess_assay = NULL,
  normalization_types = c("GR", "RV"),
  fit_source = "gDR",
  score_FUN = calculate_score
)
```

Arguments

se	SummarizedExperiment with "Averaged" and "Metrics" assays (combination experiment).
scores_assay	string; name of the output assay to write scores into. Default "scores".
averaged_assay	string; name of the input assay. Default "Averaged".
metrics_assay	string; name of the assay containing SA fit parameters. Default "Metrics".
excess_assay	string or NULL; if the SE already contains a pre-computed excess assay (e.g. from apply_combo_excess), pass its name here to skip excess re-computation and score directly from it. Default NULL (excess is computed internally).
normalization_types	character vector of normalization types to process. Default c("GR", "RV").

fit_source	string recorded in the fit_source column. Default "gDR".
score_FUN	function to reduce per-point excess values to a scalar score. Default calculate_score (mean of top 10-percentile).

Details

Unlike the low-level [bliss_fit_fn](#) and [hss_fit_fn](#) (which work on raw Averaged data per triplet), this function uses the SA fit parameters from the Metrics assay to generate curve-smoothed single-agent responses via [predict_efficacy_from_conc](#) before computing excess. This produces results numerically identical to `fit_SE.combinations`.

The function requires a Metrics assay containing columns `dilution_drug`, `cotrt_value`, `ec50`, `h`, `x_inf`, `x_0`, and `normalization_type` — as produced by `fit_SE.combinations` or [apply_fit_to_se](#) with `fit_drug_response_metrics`.

Scoring steps (per drug-combo × cell-line × normalization_type):

1. Predict smooth SA responses at every combo concentration using [predict_efficacy_from_conc](#) on `drug_1` and `drug_2` parameter sets from Metrics.
2. Average `col_values` (drug-1-along-conc1) and `row_values` (drug-2-along-conc2) → smooth.
3. Compute Bliss-expected using [calculate_Bliss](#) and HSA-expected using [calculate_HSA](#) on the smooth SA edges.
4. Compute excess via [calculate_excess](#).
5. Score = mean of top 10-percentile excess via [calculate_score](#).

Value

Updated `SummarizedExperiment` with a `scores_assay` assay containing `bliss_score` and `hsa_score` per triplet.

See Also

[bliss_fit_fn](#), [hss_fit_fn](#) for the simplified raw-data variants.

Examples

```
mae <- gDRutils::get_synthetic_data("finalMAE_combo_matrix_small")
combo_se <- mae[[gDRutils::get_supported_experiments("combo")]]
combo_se_out <- apply_combo_scores(combo_se)
"scores" %in% SummarizedExperiment::assayNames(combo_se_out)
```

average_SE	<i>Run drug response processing pipeline</i>
------------	--

Description

Run different components of the gDR drug response processing pipeline. Either: create a SummarizedExperiment and normalize raw treated and control data (create_and_normalize_SE), average data (average_SE), or fit the processed data (fit_SE). See details for more in-depth explanations.

Usage

```
average_SE(
  se,
  data_type,
  series_identifiers = NULL,
  normalized_assay = "Normalized",
  averaged_assay = "Averaged"
)

create_SE(
  df_,
  data_type,
  readout = "ReadoutValue",
  nested_identifiers = NULL,
  nested_confounders = intersect(names(df_), gDRutils::get_env_identifiers("barcode")),
  override_untrt_controls = NULL
)

fit_SE(
  se,
  data_type = "single-agent",
  nested_identifiers = NULL,
  averaged_assay = "Averaged",
  metrics_assay = "Metrics",
  n_point_cutoff = 4,
  range_conc = c(0.005, 5),
  force_fit = FALSE,
  pcutoff = 0.05,
  cap = 0.1,
  curve_type = c("GR", "RV")
)

normalize_SE(
  se,
  data_type,
  nested_identifiers = NULL,
  nested_confounders = gDRutils::get_SE_identifiers(se, "barcode", simplify = TRUE),
```

```

    control_mean_fxn = function(x) {
      mean(x, trim = 0.25)
    },
    control_assay = "Controls",
    raw_treated_assay = "RawTreated",
    normalized_assay = "Normalized",
    ndigit_rounding = 4
  )

create_and_normalize_SE(
  df_,
  data_type,
  readout = "ReadoutValue",
  control_mean_fxn = function(x) {
    mean(x, trim = 0.25)
  },
  nested_identifiers = NULL,
  nested_confounders = intersect(names(df_), gDRutils::get_env_identifiers("barcode")),
  override_untrt_controls = NULL,
  ndigit_rounding = 4,
  control_assay = "Controls",
  raw_treated_assay = "RawTreated",
  normalized_assay = "Normalized"
)

runDrugResponseProcessingPipeline(
  x,
  readout = "ReadoutValue",
  control_mean_fxn = function(x) {
    mean(x, trim = 0.25)
  },
  nested_identifiers_l = NULL,
  nested_confounders = gDRutils::get_env_identifiers("barcode"),
  override_untrt_controls = NULL,
  ndigit_rounding = 4,
  n_point_cutoff = 4,
  control_assay = "Controls",
  raw_treated_assay = "RawTreated",
  normalized_assay = "Normalized",
  averaged_assay = "Averaged",
  metrics_assay = "Metrics",
  split_data = TRUE,
  data_dir = NULL,
  partial_run = FALSE,
  start_from = get_pipeline_steps()[1],
  selected_experiments = NULL
)

```

Arguments

se	SummarizedExperiment object.
data_type	single-agent vs combination
series_identifiers	character vector of identifiers in measured or metric which define a unique data point.
normalized_assay	string of the assay name containing the normalized data. Defaults to "Normalized".
averaged_assay	string of the name of the averaged assay in the SummarizedExperiment . Defaults to "Averaged".
df_	data.table of raw drug response data containing both treated and untreated values. If a column called "BackgroundValue" exists in df_, it will be removed from the readout column.
readout	string of the name containing the cell viability readout values.
nested_identifiers	character vector with the nested_identifiers for the given SE with a given data_type
nested_confounders	Character vector of the nested_confounders for a given assay. nested_keys is character vector of column names to include in the data.tables in the assays of the resulting SummarizedExperiment object. Defaults to the nested_identifiers and nested_confounders if passed through create_and_normalize_SE or runDrugResponseProcessingPipeline.
override_untrt_controls	named list containing defining factors in the treatments. Defaults to NULL.
metrics_assay	string of the name of the metrics assay to output in the returned SummarizedExperiment Defaults to "Metrics".
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	vector of concentrations range values.
force_fit	boolean indicating whether or not to force the fit.
pcutoff	numeric cutoff value.
cap	numeric value representing the value to cap the highest allowed relative viability at.
curve_type	vector of curve type values.
control_mean_fxn	function indicating how to average controls. Defaults to mean(x, trim = 0.25).
control_assay	string containing the name of the assay representing the controls in the se. Defaults to "Controls".
raw_treated_assay	string containing the name of the assay representing the raw treated data in the se. Defaults to "RawTreated".
ndigit_rounding	integer indicating number of digits to round to in calculations. Defaults to 4.

x	data.table of MAE with drug response data
nested_identifiers_l	list with the nested_identifiers(character vectors) for single-agent and (optionally) for combination data
split_data	boolean indicating whether data provided as the MultiAssayExperiment should be split again into appropriate data types
data_dir	string with the path to the directory with intermediate data of experiments (qs2 files). If set to NULL (default) intermediate data is not saved/read in.
partial_run	logical flag indicating if the pipeline should be run partially (from the step defined with start_from)
start_from	string indicating the pipeline step from which partial run should be launched
selected_experiments	character vector with experiments for which pipeline should be run. This option works only for the pipeline being run partially (i.e. with partial_run flag set to TRUE)

Details

runDrugResponseProcessingPipeline is made up of 3 separate steps:

- "create_and_normalize_SE"
- "average_SE"
- "fit_SE"

For create_and_normalize_SE, this creates a SummarizedExperiment object from a data.table, where the data.table contains treatments on rows, and conditions on columns. A [SummarizedExperiment](#) object containing two assays is created: treated readouts will live in an assay called "RawTreated", and reference readouts live in an assay called "Controls". Subsequently, the treated and control elements will be normalized to output two metrics:

For average_SE, take the normalized assay and average the nested DataFrames across unique nested_identifiers.

For fit_SE, take the averaged assay and fit curves to obtain metrics, one set of metrics for each normalization type set.

Pipeline can be run partially with partial_run flag set to TRUE. The start_from string defines the step from which the pipeline will be launched. However, partial run of the pipeline is possible only if the whole pipeline was launched at least once with defined data_dir and intermediate data was saved as qs2 files into data_dir.

Pipeline can be run for the selected experiments by changing the default value of selected_experiments param. This scenario only works when partial_run is enabled.

Value

MAE object

Examples

```

d <- rep(seq(0.1, 0.9, 0.1), each = 4)
v <- rep(seq(0.1, 0.4, 0.1), 9)
df <- S4Vectors::DataFrame(
  Concentration = d,
  normalization_type = rep(c("GR", "RV"), length(v) * 2),
  x = rep(v, 2)
)
normalized <- BumpyMatrix::splitAsBumpyMatrix(row = 1, column = 1, x = df)

keys <- list(Trt = "Concentration")
assays <- list("Normalized" = normalized)
se <- SummarizedExperiment::SummarizedExperiment(assays = assays)
se <- gDRutils::set_SE_keys(se, keys)
se <- gDRutils::set_SE_identifiers(se, gDRutils::get_env_identifiers())
se1 <- average_SE(
  se,
  data_type = "single-agent",
  normalized_assay = "Normalized",
  averaged_assay = "Averaged"
)

td <- gDRimport::get_test_data()
l_tbl <- gDRimport::load_data(
  manifest_file = gDRimport::manifest_path(td),
  df_template_files = gDRimport::template_path(td),
  results_file = gDRimport::result_path(td)
)
imported_data <- merge_data(
  l_tbl$manifest,
  l_tbl$treatments,
  l_tbl$data
)

se <- purrr::quietly(create_SE)(imported_data, data_type = "single-agent")

td <- gDRimport::get_test_data()
l_tbl <- gDRimport::load_data(
  manifest_file = gDRimport::manifest_path(td),
  df_template_files = gDRimport::template_path(td),
  results_file = gDRimport::result_path(td)
)
imported_data <- merge_data(
  l_tbl$manifest,
  l_tbl$treatments,
  l_tbl$data
)

inl <- prepare_input(imported_data)
se <- create_SE(

```

```

inl$df_list[["single-agent"]],
data_type = "single-agent",
nested_confounders = inl$nested_confounders)

normalize_SE(se, data_type = "single-agent")
p_dir <- file.path(tempdir(), "pcheck")
dir.create(p_dir)
td <- gDRimport::get_test_data()
l_tbl <- gDRimport::load_data(
  manifest_file = gDRimport::manifest_path(td),
  df_template_files = gDRimport::template_path(td),
  results_file = gDRimport::result_path(td)
)
imported_data <- merge_data(
  l_tbl$manifest,
  l_tbl$treatments,
  l_tbl$data
)
runDrugResponseProcessingPipeline(
  imported_data,
  data_dir = p_dir
)

```

bliss_fit_fn

bliss_fit_fn

Description

Reference fit function for Bliss independence synergy scoring on combination data. Computes Bliss-expected response from single-agent edges and derives excess and score from raw averaged data.

Usage

```
bliss_fit_fn(avg_dt)
```

Arguments

avg_dt	data.table for one (drug1 x drug2 x cell line x normalization_type) combination. Must contain columns Concentration, Concentration_2, x, and normalization_type.
--------	--

Details

Intended for use with [apply_fit](#) on combination SEs:

```

gDRutils::apply_fit(
  combo_se, bliss_fit_fn, "combination",
  output_assay = "custom_bliss", fit_source = "bliss"
)

```

Value

Named list with `bliss_score`, `bliss_excess_mean`, `n_combo_points`, and `normalization_type`.

Examples

```
mae <- gDRutils::get_synthetic_data("finalMAE_combo_matrix_small")
combo_se <- mae[[gDRutils::get_supported_experiments("combo")]]
combo_se_out <- gDRutils::apply_fit(
  combo_se, bliss_fit_fn, "combination",
  output_assay = "custom_bliss", fit_source = "bliss"
)
```

`calculate_excess`*Calculate the difference between values in two data.tables*

Description

Calculate the difference between values, likely representing the same metric, from two `data.tables`.

Usage

```
calculate_excess(
  metric,
  measured,
  series_identifiers,
  metric_col,
  measured_col
)
```

Arguments

<code>metric</code>	data.table often representing readouts derived by calculating some metric. Examples of this could include hsa or bliss calculations from single-agent data.
<code>measured</code>	data.table often representing measured data from an experiment.
<code>series_identifiers</code>	character vector of identifiers in measured or metric which define a unique data point.
<code>metric_col</code>	string of the column in metric to use in excess calculation.
<code>measured_col</code>	string of the column in measured to use in excess calculation.

Value

data.table of measured, now with an additional column named `excess` (positive values for synergy/benefit).

Examples

```
metric <- data.table::data.table(
  Concentration = c(1, 2, 3, 1, 2, 3),
  Concentration_2 = c(1, 1, 1, 2, 2, 2),
  GRvalue = c(100, 200, 300, 400, 500, 600)
)
measured <- data.table::data.table(
  Concentration = c(3, 1, 2, 2, 1, 3),
  Concentration_2 = c(1, 1, 1, 2, 2, 2),
  testvalue = c(200, 0, 100, 400, 300, 500)
)
series_identifiers <- c("Concentration", "Concentration_2")
metric_col <- "GRvalue"
measured_col <- "testvalue"
calculate_excess(
  metric,
  measured,
  series_identifiers,
  metric_col,
  measured_col
)
```

calculate_GR_value	<i>Calculate a GR value.</i>
--------------------	------------------------------

Description

Calculate a GR value for a given set of dose response values.

Usage

```
calculate_GR_value(
  rel_viability,
  corrected_readout,
  day0_readout,
  untrt_readout,
  ndigit_rounding,
  duration,
  ref_div_time,
  cap = 1.25
)

calculate_time_dep_GR_value(
  corrected_readout,
  day0_readout,
  untrt_readout,
  ndigit_rounding
```

```
)

calculate_endpt_GR_value(
  rel_viability,
  duration,
  ref_div_time,
  cap = 1.25,
  ndigit_rounding
)
```

Arguments

rel_viability	numeric vector representing the Relative Viability.
corrected_readout	numeric vector containing the corrected readout.
day0_readout	numeric vector containing the day 0 readout.
untrt_readout	numeric vector containing the untreated readout.
ndigit_rounding	integer specifying the number of digits to use for calculation rounding.
duration	numeric value specifying the length of time the cells were treated (in hours).
ref_div_time	numeric value specifying the reference division time for the cell line in the experiment.
cap	numeric value representing the value to cap the highest allowed relative viability at.

Details

Note that this function expects that all numeric vectors are of the same length. `calculate_GR_value` will try to greedily calculate a GR value. If no day 0 readouts are available, the `duration` and `ref_div_time` will be used to try to back-calculate a day 0 value in order to produce a GR value.

In the case of calculating the reference GR value from multiple reference readout values, the vectorized calculation is performed and then the resulting vector should be averaged outside of this function.

Note that it is expected that the `ref_div_time` and `duration` are reported in the same units.

Value

numeric vector containing GR values, one value for each element of the input vectors.

See Also

`normalize_SE2`

Examples

```
duration <- 144
rv <- seq(0.1, 1, 0.1)
corrected <- seq(41000, 50000, 1000)
day0 <- seq(91000, 95500, 500)
untrt <- rep(c(115000, 118000), 5)

calculate_GR_value(
  rel_viability = rv,
  corrected_readout = corrected,
  day0_readout = day0,
  untrt_readout = untrt,
  ndigit_rounding = 4,
  duration = duration,
  ref_div_time = duration / 2
)

readouts <- rep(10000, 5)
calculate_time_dep_GR_value(readouts, readouts * 1.32, readouts * 2, 2)

readouts <- rep(10000, 5)
calculate_endpt_GR_value(readouts, 72, 1, ndigit_rounding = 2)
```

calculate_score	<i>Calculate score for HSA and Bliss</i>
-----------------	--

Description

Calculate score for HSA and Bliss

Usage

```
calculate_score(excess)
```

Arguments

excess numeric vector with excess

Value

numeric vector with calculated score

Examples

```
metric <- data.table::data.table(
  Concentration = c(1, 2, 3, 1, 2, 3),
  Concentration_2 = c(1, 1, 1, 2, 2, 2),
  GRvalue = c(100, 200, 300, 400, 500, 600)
```

```
)
measured <- data.table::data.table(
  Concentration = c(3, 1, 2, 2, 1, 3),
  Concentration_2 = c(1, 1, 1, 2, 2, 2),
  testvalue = c(200, 0, 100, 400, 300, 500)
)
series_identifiers <- c("Concentration", "Concentration_2")
metric_col <- "GRvalue"
measured_col <- "testvalue"
x <- calculate_excess(
  metric,
  measured,
  series_identifiers,
  metric_col,
  measured_col
)
calculate_score(x$x)
```

cleanup_metadata	<i>cleanup_metadata</i>
------------------	-------------------------

Description

Cleanup a data.table with metadata

Usage

```
cleanup_metadata(
  df_metadata,
  cell_line_annotation = NULL,
  drug_annotation = NULL
)
```

Arguments

df_metadata	a data.table with metadata
cell_line_annotation	optional data.table with cell line annotations; if NULL (default), annotations are looked up from gDRinternal or gDRtestData
drug_annotation	optional data.table with drug annotations; if NULL (default), annotations are looked up from gDRinternal or gDRtestData

Details

Adds annotations and check whether user provided correct input data.

Value

a data.table with cleaned metadata

Examples

```
df <- data.table::data.table(
  clid = "CELL_LINE",
  Gnumber = "DRUG_1",
  Concentration = c(0, 1),
  Duration = 72
)
cleanup_df <- cleanup_metadata(df)
```

compute_growth_rates *Compute per-period growth rates for time-course screens*

Description

Stage 1 of the time-course pipeline: fits a growth rate per well within each time window, aggregates replicates, and normalises the rates to the control wells of a reference period. The returned table keeps every treatment, controls included, together with the raw rate and its standard deviation, so it can be reported on directly. Pass it through [growth_rates_to_se](#) to obtain the stage 2 input, or use [fit_SE.timecourse](#) to run both stages at once.

Usage

```
compute_growth_rates(
  se,
  periods,
  normalization_map,
  lfc_assay = NULL,
  rate_fn = NULL
)
```

Arguments

se	SummarizedExperiment with a log-fold-change assay.
periods	named list of numeric vectors of length 2; half-open time windows c(start_hour, end_hour), see fit_SE.timecourse .
normalization_map	named character vector mapping every period to its reference period or to "None", see fit_SE.timecourse .
lfc_assay	string or NULL; name of the log-fold-change assay. NULL uses the input_assay of the "time-course" fit profile (see get_fit_profile).

`rate_fn` function or NULL; growth-rate estimator. It receives a `data.table` holding the timepoints of one well within one window — including the duration column and the log-fold-change column — and must return a single numeric value: the growth rate in doublings per day. NULL uses the slope of `lm(LogFoldChange ~ Duration)`.

Details

Growth rates are computed per (Barcode, WellRow, WellColumn, CellLine, Drug, Concentration, partner drug/concentration slots) group, so single-agent, doublet and triplet arms are never pooled. Control wells are detected via `gDRutils::get_env_identifiers("untreated_tag")` and a single baseline per (CellLine, period) is used for normalization.

Value

A `data.table` with one row per (CellLine, Drug, Concentration, partner slots, period) — controls included — holding `GrowthRate` (doublings/day), `sd_GrowthRate` across replicate wells, `rate_0` (the control baseline used, NA for periods mapped to "None" and when no control rows are present), `NormalizedGrowthRate` and `normalization_type`. All of these columns are always present, whatever the normalization map or the availability of controls. `NormalizedGrowthRate` is NA where the control baseline is not meaningfully positive: such a denominator either inverts the sign of the ratio (when negative) or leaves it undefined (when zero, within fitting tolerance) rather than scaling it, so no meaningful normalized rate exists. A warning names the cell line and period.

See Also

[growth_rates_to_se](#), [fit_SE.timecourse](#), [apply_fit](#)

Examples

```
## Not run:
periods <- list(early = c(0, 48), late = c(48, 96))
norm_map <- c(early = "None", late = "early")
growth_dt <- compute_growth_rates(se_tc, periods, norm_map)

# Stage 2 with any custom fit function:
se_fit <- gDRutils::apply_fit(
  growth_rates_to_se(growth_dt),
  fit_fn = my_fit_fn,
  data_type = "time-course-metrics",
  output_assay = "Metrics",
  merge = "replace",
  fit_source = "custom"
)

## End(Not run)
```

`convert_mae_to_raw_data`*Transform mae into raw data*

Description

Transform mae into raw data

Usage

```
convert_mae_to_raw_data(mae)
```

Arguments

mae	MultiAssayExperiment object with SummarizedExperiments containing "RawTreated" and "Controls" assays
-----	--

Value

data.table with raw data

Examples

```
mae <- gDRutils::get_synthetic_data("finalMAE_small")
convert_mae_to_raw_data(mae)
```

`convert_se_to_raw_data`*Transform se into raw_data*

Description

Transform se into raw_data

Usage

```
convert_se_to_raw_data(se)
```

Arguments

se	SummarizedExperiment object with "RawTreated" and "Controls" assays
----	---

Value

data.table with raw data

Examples

```
mae <- gDRutils::get_synthetic_data("finalMAE_small")
se <- mae[[1]]
convert_se_to_raw_data(se)
```

data_model	<i>Detect model of data</i>
------------	-----------------------------

Description

Detect model of data

Usage

```
data_model(x)
```

Arguments

x data.table with raw data or SummarizedExperiment object with gDR assays

Value

string with the information of the raw data follows single-agent or combination data model

Examples

```
data_model("single-agent")
```

data_model.character	<i>Detect model of data from experiment name</i>
----------------------	--

Description

Detect model of data from experiment name

Usage

```
## S3 method for class 'character'
data_model(x)
```

Arguments

x character with experiment name

Value

string with the information of the raw data follows single-agent or combination data model

data_model.data.table *Detect model of data in data.table*

Description

Detect model of data in data.table

Usage

```
## S3 method for class 'data.table'  
data_model(x)
```

Arguments

x data.table of raw drug response data containing both treated and untreated values.

Value

string with the information of the raw data follows single-agent or combination data model

fit_SE.combinations *fit_SE for combination screens*

Description

Perform fittings for combination screens.

Usage

```
fit_SE.combinations(  
  se,  
  data_type = gDRutils::get_supported_experiments("combo"),  
  series_identifiers = NULL,  
  normalization_types = c("GR", "RV"),  
  averaged_assay = "Averaged",  
  metrics_assay = "Metrics",  
  score_FUN = calculate_score  
)
```

Arguments

se	SummarizedExperiment object with a BumpyMatrix assay containing averaged data.
data_type	single-agent vs combination
series_identifiers	character vector of the column names in the nested DFrame corresponding to nested identifiers.
normalization_types	character vector of normalization types used for calculating combo matrix.
averaged_assay	string of the name of the averaged assay to use as input. in the se.
metrics_assay	string of the name of the metrics assay to output in the returned SummarizedExperiment . whose combination represents a unique series for which to fit curves.
score_FUN	function used to calculate score for HSA and Bliss

Details

This function assumes that the combination is set up with both concentrations nested in the assay.

Value

A SummarizedExperiment object with an additional assay containing the combination metrics.

Examples

```
fmae_cms <- gDRutils::get_synthetic_data("finalMAE_combo_matrix_small")

se1 <- fmae_cms[[gDRutils::get_supported_experiments("combo")]]
SummarizedExperiment::assays(se1) <-
  SummarizedExperiment::assays(se1)["Averaged"]
fit_SE.combinations(se1[1, 1])
```

fit_SE.timecourse	<i>fit_SE for time-course (Incucyte) screens</i>
-------------------	--

Description

Two-stage pipeline for time-course dose-response data:

1. Compute growth rates per user-defined time window and normalise them to the control of the reference period ([compute_growth_rates](#)).
2. Fit a 3-parameter Hill (logistic) curve on the NormalizedGrowthRate $\sim \log_{10}(\text{Concentration})$ dose-response for each (Drug, CellLine, period) combination.

Usage

```
fit_SE.timecourse(
  se,
  periods,
  normalization_map,
  lfc_assay = NULL,
  metrics_assay = "Metrics",
  rate_fn = NULL,
  fit_fn = NULL,
  fit_source = NULL,
  n_point_cutoff = 4L,
  range_conc = c(0.005, 5),
  force_fit = FALSE,
  pcutoff = 0.05,
  cap = 0.1
)
```

Arguments

- | | |
|-------------------|--|
| se | SummarizedExperiment with a LogFoldChange assay (output of <code>normalize_SE</code> for <code>data_type = "time-course"</code>). |
| periods | named list of numeric vectors of length 2; each element defines a half-open time window <code>c(start_hour, end_hour)</code> used for growth-rate calculation: a timepoint is used when <code>start_hour <= Duration < end_hour</code> , so contiguous windows partition the time-course without dropping the shared boundary. At minimum <code>periods\$early</code> must be provided. Example: <code>list(early = c(0, 48), mid = c(48, 96), late = c(96, 144))</code> . |
| normalization_map | named character vector mapping <i>every</i> period name to the reference period for DMSO normalization, or "None" to keep the raw growth rate. Its names must match names(<code>periods</code>) exactly and its values must be "None" or an existing period name. Example: <code>c(early = "None", mid = "early", late = "early")</code> . |
| lfc_assay | string; name of the log-fold-change assay in <code>se</code> . Defaults to the <code>input_assay</code> of the "time-course" fit profile. |
| metrics_assay | string; name of the output metrics assay. Default "Metrics". |
| rate_fn | function or NULL; growth-rate estimator passed to compute_growth_rates . NULL uses the default <code>lm(LogFoldChange ~ Duration)</code> slope. |
| fit_fn | function or NULL; dose-response fit applied to each <code>normalization_type</code> slice of the <code>GrowthRates</code> assay. NULL uses fit_drug_response_metrics on <code>NormalizedGrowthRate</code> . When a custom function is supplied, <code>n_point_cutoff</code> , <code>range_conc</code> , <code>force_fit</code> , <code>pcutoff</code> and <code>cap</code> are not used. |
| fit_source | string or NULL; value recorded in the <code>fit_source</code> column. NULL records "gDR" for the default fit and "custom" for a user-supplied <code>fit_fn</code> . |
| n_point_cutoff | integer; minimum number of unique concentrations required to attempt a curve fit. Default 4L. |

range_conc	numeric vector of length 2; concentration range for x_AOC_range. Default <code>c(5e-3, 5)</code> .
force_fit	logical; if TRUE fit is kept even if not statistically significant. Default FALSE.
pcutoff	numeric; p-value threshold for the F-test. Default 0.05.
cap	numeric; upper-capping fold above $x_0 = 1$. Default 0.1.

Details

The LogFoldChange assay consumed by this function is produced by [normalize_SE](#) when `data_type = "time-course"`.

Both stages accept a replacement function, so custom fitting works for time-course data the same way it does for single-agent and combination screens. The two stages can also be run separately: [compute_growth_rates](#) returns the growth rate table (controls included) and [growth_rates_to_se](#) turns it into the input of [apply_fit](#) with `data_type = "time-course-metrics"`.

Value

A new SummarizedExperiment where rows = (Drug, CellLine) and columns = period names, holding the GrowthRates assay from stage 1 and the Metrics assay (or metrics_assay) from stage 2. With the default `fit_fn` the metrics are: `ec50`, `xc50`, `h`, `x_inf`, `x_0`, `r2`, `x_mean`, `x_AOC`, `x_AOC_range`, `x_max`, `fit_type`, `N_conc`, `maxlog10Concentration`, and `period`.

See Also

[compute_growth_rates](#) for stage 1 on its own, [fit_SE](#) for single-agent fitting, [fit_SE.combinations](#) for combination screens, [apply_fit](#) for the generic fitting engine.

Examples

```
## Not run:
# After normalize_SE(se_tc, data_type = "time-course"):
periods <- list(early = c(0, 48), mid = c(48, 96), late = c(96, 144))
norm_map <- c(early = "None", mid = "early", late = "early")
se_fit <- fit_SE.timecourse(se_tc, periods = periods,
                           normalization_map = norm_map)
gDRutils::convert_se_assay_to_dt(se_fit, "Metrics")

# Custom growth-rate estimator and custom dose-response fit:
se_custom <- fit_SE.timecourse(
  se_tc,
  periods = periods,
  normalization_map = norm_map,
  rate_fn = function(dt) 24 * stats::median(diff(dt$LogFoldChange) /
                                              diff(dt$Duration)),
  fit_fn = function(dt) data.table::data.table(x_mean = mean(dt$NormalizedGrowthRate))
)

## End(Not run)
```

```
get_assays_per_pipeline_step
```

get info about created/present assays in SE at the given pipeline step

Description

get info about created/present assays in SE at the given pipeline step

Usage

```
get_assays_per_pipeline_step(
  step,
  data_model,
  status = c("created", "present")
)
```

Arguments

step	string with pipeline step
data_model	single-agent vs combination
status	string return vector of assays created or present at the given step?

Value

assay

```
get_cell_line_annotation
```

get_cell_line_annotation

Description

Get cell line annotation data table

Usage

```
get_cell_line_annotation(
  data,
  fname = "cell_lines.csv",
  fill = "unknown",
  annotation_package = if ("gDRinternal" %in% .packages(all.available = TRUE)) {
    "gDRinternal"
  } else {
    "gDRtestData"
  }
)
```

Arguments

data	data.table with cell line identifiers to be matched
fname	string with file name containing the annotation
fill	string indicating how unknown cell lines should be filled in the DB
annotation_package	string indicating name of the package containing cell line annotation

Value

data.table with cell line annotations

Examples

```
data <- data.table::data.table(clid = c("CL1", "CL2", "CL3"))
cell_line_annotation <- get_cell_line_annotation(data)
```

```
get_cellline_annotation_from_dt
```

Retrieve the cell line annotation from the annotated dt input

Description

Retrieve the cell line annotation from the annotated dt input

Usage

```
get_cellline_annotation_from_dt(dt)
```

Arguments

dt	annotated data.table
----	----------------------

Value

data.table with cell line annotation

Examples

```
dt <- data.table::data.table(Gnumber = "A",
  clid = "CL123",
  CellLineName = "cl name",
  Tissue = "Bone",
  parental_identifler = "some cl",
  subtype = "cortical",
  ReferenceDivisionTime = 5)
get_cellline_annotation_from_dt(dt)
```

```
get_default_nested_identifiers
```

Get default nested identifiers

Description

Get default nested identifiers

Usage

```
get_default_nested_identifiers(x, data_model = NULL)

## S3 method for class 'data.table'
get_default_nested_identifiers(x, data_model = NULL)

## S3 method for class 'SummarizedExperiment'
get_default_nested_identifiers(x, data_model = NULL)
```

Arguments

x	data.table with raw data or SummarizedExperiment object with gDR assays
data_model	single-agent vs combination

Value

vector of nested identifiers

Examples

```
get_default_nested_identifiers(data.table::data.table())
```

```
get_drug_annotation    get_drug_annotation
```

Description

Get drug annotation data table

Usage

```
get_drug_annotation(
  data,
  fname = "drugs.csv",
  fill = "unknown",
  annotation_package = if ("gDRinternal" %in% .packages(all.available = TRUE)) {
    "gDRinternal"
  } else {
    "gDRtestData"
  }
)
```

Arguments

data	data.table with drug identifiers to be matched
fname	string with file name containing the annotation
fill	string indicating how unknown drugs should be filled in the DB
annotation_package	string indicating name of the package containing drug annotation

Value

data.table with drug annotations

Examples

```
data <- data.table::data.table(Gnumber = c("drug1", "drug2", "drug3"))
drug_annotation <- get_drug_annotation(data)
```

get_drug_annotation_from_dt

Retrieve the drug annotation from the annotated dt input

Description

Retrieve the drug annotation from the annotated dt input

Usage

```
get_drug_annotation_from_dt(dt)
```

Arguments

dt	annotated data.table
----	----------------------

Value

data.table with drug annotation

Examples

```
dt <- data.table::data.table(Gnumber = "A",
  DrugName = "drugA",
  drug_moa = "drug_moa_A")
get_drug_annotation_from_dt(dt)
```

get_period_timepoints *List the measurement timepoints entering each growth-rate window*

Description

Reports which durations actually go into the fit for every period, so that a report or a log can state the measurements it used rather than only the window bounds. The bounds alone are ambiguous: whether a timepoint sitting exactly on a boundary is included depends on the window convention, and a window may silently contain far fewer points than intended.

Usage

```
get_period_timepoints(se, periods, lfc_assay = NULL)
```

Arguments

se	SummarizedExperiment with lfc_assay.
periods	named list of two-element numeric windows, as passed to compute_growth_rates .
lfc_assay	string; assay to read durations from. NULL uses the default input assay of the "time-course" fit profile.

Details

Uses the same half-open [start, end) rule as [compute_growth_rates](#), so the answer always matches what was fitted.

Value

data.table with one row per period: period, window (the bounds as "start - end"), n (number of distinct timepoints) and timepoints (those timepoints, sorted, comma-separated). Periods matching no measurement yield n = 0.

timepoints is the union of the durations present anywhere in se, not a per-plate or per-well listing. On a uniform sampling grid the two coincide; if plates were read on different schedules, a duration contributed by only one barcode still appears here, and n is therefore an upper bound on what any single well contributed to the fit.

See Also

[compute_growth_rates](#), [fit_SE.timecourse](#)

Examples

```
## Not run:
get_period_timepoints(se_tc, list(early = c(44, 92), late = c(92, 140)))

## End(Not run)
```

growth_rates_to_se	<i>Convert a growth rate table into the stage 2 SummarizedExperiment</i>
--------------------	--

Description

Builds the GrowthRates SummarizedExperiment consumed by [apply_fit](#) with `data_type = "time-course-metrics"`: rows are (Drug, partner drug/concentration slots, CellLine) — so every combination arm is fitted as its own dose-response curve — and columns are the period names. Control and zero-concentration rows are dropped, since they carry no dose-response information.

Usage

```
growth_rates_to_se(growth_dt)
```

Arguments

`growth_dt` data.table; output of [compute_growth_rates](#).

Value

A SummarizedExperiment with a GrowthRates assay holding Concentration, the partner concentrations, NormalizedGrowthRate and normalization_type per cell.

See Also

[compute_growth_rates](#), [fit_SE.timecourse](#)

Examples

```
## Not run:
growth_dt <- compute_growth_rates(se_tc, periods, norm_map)
se_gr <- growth_rates_to_se(growth_dt)

## End(Not run)
```

grr_matches

*Value Matching***Description**

Returns a lookup table or list of the positions of ALL matches of its first argument in its second and vice versa. Similar to [match](#), though that function only returns the first match.

Usage

```
grr_matches(
  x,
  y,
  all.x = TRUE,
  all.y = TRUE,
  list = FALSE,
  indexes = TRUE,
  nomatch = NA
)
```

Arguments

<code>x</code>	vector. The values to be matched. Long vectors are not currently supported.
<code>y</code>	vector. The values to be matched. Long vectors are not currently supported.
<code>all.x</code>	logical; if TRUE, then each value in x will be included even if it has no matching values in y
<code>all.y</code>	logical; if TRUE, then each value in y will be included even if it has no matching values in x
<code>list</code>	logical. If TRUE, the result will be returned as a list of vectors, each vector being the matching values in y. If FALSE, result is returned as a data.table with repeated values for each match.
<code>indexes</code>	logical. Whether to return the indices of the matches or the actual values.
<code>nomatch</code>	the value to be returned in the case when no match is found. If not provided and <code>indexes=TRUE</code> , items with no match will be represented as NA. If set to NULL, items with no match will be set to an index value of <code>length+1</code> . If <code>indexes=FALSE</code> , they will default to NA.

Details

This behavior can be imitated by using joins to create lookup tables, but `matches` is simpler and faster: usually faster than the best joins in other packages and thousands of times faster than the built in [merge](#).

`all.x/all.y` correspond to the four types of database joins in the following way:

left `all.x=TRUE, all.y=FALSE`

right all.x=FALSE, all.y=TRUE
inner all.x=FALSE, all.y=FALSE
full all.x=TRUE, all.y=TRUE

Note that NA values will match other NA values.

Source of the function: <https://github.com/cran/grr/blob/master/R/grr.R>

Value

data.table

Examples

```
mat_elem <- data.table::data.table(
  DrugName = rep(c("untreated", "drugA", "drugB", "untreated"), 2),
  DrugName_2 = rep(c("untreated", "vehicle", "drugA", "drugB"), 2),
  clid = rep(c("C1", "C2"), each = 4)
)
untreated_tag <- gDRutils::get_env_identifiers("untreated_tag")
ref_idx <- which(
  mat_elem$DrugName %in% untreated_tag |
  mat_elem$DrugName_2 %in% untreated_tag
)
ref <- mat_elem[ref_idx, ]
treated <- mat_elem[-ref_idx, ]
valid <- c("DrugName", "DrugName_2")
trt <- lapply(valid, function(x) {
  colnames <- c("clid", x)
  treated[, colnames, with = FALSE]
})
trt <- do.call(paste,
  do.call(rbind, lapply(trt, function(x) setNames(x, names(trt[[1]]))))
)
ref <- lapply(valid, function(x) {
  colnames <- c("clid", x)
  ref[, colnames, with = FALSE]
})
ref <- do.call(paste,
  do.call(rbind, lapply(ref, function(x) setNames(x, names(ref[[1]]))))
)
grr_matches(trt, ref, list = FALSE, all.y = FALSE)
```

hss_fit_fn

hss_fit_fn

Description

Reference fit function for Highest Single Agent (HSA) synergy scoring on combination data. Uses raw averaged single-agent edges.

Usage

```
hss_fit_fn(avg_dt)
```

Arguments

`avg_dt` data.table for one (drug1 x drug2 x cell line x normalization_type) combination. Must contain columns Concentration, Concentration_2, x, and normalization_type.

Details

Intended for use with [apply_fit](#) on combination SEs:

```
gDRutils::apply_fit(
  combo_se, hss_fit_fn, "combination",
  output_assay = "custom_hss", fit_source = "hss"
)
```

Value

Named list with `hss_score`, `hss_excess_mean`, `n_combo_points`, and `normalization_type`.

Examples

```
mae <- gDRutils::get_synthetic_data("finalMAE_combo_matrix_small")
combo_se <- mae[[gDRutils::get_supported_experiments("combo")]]
combo_se_out <- gDRutils::apply_fit(
  combo_se, hss_fit_fn, "combination",
  output_assay = "custom_hss", fit_source = "hss"
)
```

identify_data_type	<i>Identify type of data</i>
--------------------	------------------------------

Description

Identify type of data

Usage

```
identify_data_type(dt, codilution_conc = 2, matrix_conc = 1)
```

Arguments

<code>dt</code>	data.table of raw drug response data containing both treated and untreated values
<code>codilution_conc</code>	integer of maximum number of concentration ratio of co-treatment to classify as codilution data type; defaults to 2
<code>matrix_conc</code>	integer of minimum number of concentration pairs of co-treatment to classify as co-treatment or matrix data type; defaults to 1

Value

data.table of raw drug response data with additional column type with the info of data type for a given row of data.table

Author(s)

Bartosz Czech czech.bartosz@external.gene.com

Examples

```
conc <- rep(seq(0, 0.3, 0.1), 2)
ctrl_dt <- S4Vectors::DataFrame(
  ReadoutValue = c(2, 2, 1, 1, 2, 1),
  Concentration = rep(0, 6),
  masked = FALSE,
  DrugName = rep(c("DRUG_10", "vehicle", "DRUG_8"), 2),
  CellLineName = "CELL1"
)

trt_dt <- S4Vectors::DataFrame(
  ReadoutValue = rep(seq(1, 4, 1), 2),
  Concentration = conc,
  masked = rep(FALSE, 8),
  DrugName = c("DRUG_10", "DRUG_8"),
  CellLineName = "CELL1"
)

input_dt <- data.table::as.data.table(rbind(ctrl_dt, trt_dt))
input_dt$Duration <- 72
input_dt$CorrectedReadout2 <- input_dt$ReadoutValue
identify_data_type(input_dt)
```

identify_keys

identify_keys

Description

Group columns from a data.table that correspond to different

Usage

```
identify_keys(
  df_,
  nested_keys = NULL,
  override_untrt_controls = NULL,
  identifiers = gDRutils::get_env_identifiers()
)
```

Arguments

df_ a data.table to identify keys for.

nested_keys character vector of keys to exclude from the returned list. The keys discarded should be identical to the keys in the third dimension of the SummarizedExperiment. Defaults to the "Barcode" and the masked identifier.

override_untrt_controls named list containing defining factors in the treatments. Defaults to NULL.

identifiers named list containing all identifiers to use during processing. By default, this value will be obtained by the environment.

Details

This is most likely to be used for provenance tracking and will be placed on the SummarizedExperiment metadata for downstream analyses to reference.

Value

named list of key types and their corresponding key values.

See Also

map_df, create_SE

Examples

```
n <- 64
md_df <- data.table::data.table(
  Gnumber = rep(c("vehicle", "untreated", paste0("G", seq(2)))), each = 16),
  DrugName = rep(c("vehicle", "untreated", paste0("GN", seq(2)))), each = 16),
  clid = paste0("C", rep_len(seq(4), n)),
  CellLineName = paste0("N", rep_len(seq(4), n)),
  replicates = rep_len(paste0("R", rep(seq(4), each = 4)), 64),
  drug_moa = "inhibitor",
  ReferenceDivisionTime = rep_len(c(120, 60), n),
  Tissue = "Lung",
  parental_identifier = "CL12345",
  Duration = 160
)
md_df <- unique(md_df)
ref <- md_df$Gnumber %in% c("vehicle", "untreated")
trt_df <- md_df[!ref, ]
```

```
identify_keys(trt_df)
```

map_df

Map treated conditions to their respective references.

Description

Map treated conditions to their respective Day0, untreated, or single-agent references using condition metadata.

Usage

```
map_df(
  trt_md,
  ref_md,
  override_untrt_controls = NULL,
  ref_cols,
  ref_type = c("Day0", "untrt_Endpoint")
)
```

Arguments

trt_md	data.table of treated metadata.
ref_md	data.table of untreated metadata.
override_untrt_controls	named list indicating what treatment metadata fields should be used as a control. Defaults to NULL.
ref_cols	character vector of the names of reference columns to include. Likely obtained from identify_keys().
ref_type	string of the reference type to map to. Should be one of c("Day0", "untrt_Endpoint").

Details

If `override_untrt_controls` is specified, the values in the named list will supersede the values in `trt_md` during the matching process. This is useful for mapping treatments to specific "standard" untreated controls.

Value

named list mapping treated metadata to untreated metadata.

See Also

identify_keys

Examples

```
# Standard Endpoint Mapping
trt_dt <- data.table::data.table(
  clid = c("C1", "C2"),
  Duration = 72,
  rn = c("T1", "T2")
)
ref_dt <- data.table::data.table(
  clid = c("C1", "C2"),
  Duration = 72,
  rn = c("R1", "R2")
)
map_df(trt_dt, ref_dt, ref_cols = "clid", ref_type = "untrt_Endpoint")
```

map_ids_to_fits

Get predicted values for a given fit and input.

Description

Map fittings to identifiers and compute the predicted values for corresponding fits.

Usage

```
map_ids_to_fits(pred, match_col, fittings, fitting_id_col)
```

Arguments

pred numeric vector for which you want predictions.

match_col vector to match on fittings to get the correct fit.

fittings data.table of fit metrics.

fitting_id_col string of the column name in fittings that should be used to match with **match_col**.

Value

Numeric vector of predicted values given pred inputs and fittings values.

Examples

```
pred <- c(1, 5, 5)
match_col <- c(1, 1, 2)
fitting_id_col <- "match_on_me"

fit1 <- data.table::data.table(h = 2.09, x_inf = 0.68, x_0 = 1, ec50 = 0.003)
fit2 <- data.table::data.table(h = 0.906, x_inf = 0.46, x_0 = 1, ec50 = 0.001)
fittings <- do.call(rbind, list(fit1, fit2))
fittings[[fitting_id_col]] <- c(1, 2)
```

```
map_ids_to_fits(pred, match_col, fittings, fitting_id_col)
```

map_untreated	<i>Identify untreated rows based on Drug treatment alone</i>
---------------	--

Description

Identify untreated rows based on Drug treatment alone

Usage

```
map_untreated(mat_elem)
```

Arguments

mat_elem data.table input containing drug name or drug identifier columns.

Value

logical vector

merge_data	<i>merge_data</i>
------------	-------------------

Description

Merge all the input data into a single data.table

Usage

```
merge_data(  
  manifest,  
  treatments,  
  data,  
  cell_line_annotation = NULL,  
  drug_annotation = NULL  
)
```

Arguments

manifest a data.table with a manifest info
treatments a data.table with a treatments info
data a data.table with a raw data info
cell_line_annotation
 optional data.table with cell line annotations; if NULL (default), annotations are looked up from gDRinternal or gDRtestData
drug_annotation
 optional data.table with drug annotations; if NULL (default), annotations are looked up from gDRinternal or gDRtestData

Value

a data.table with merged data and metadata.

Examples

```

td <- gDRimport::get_test_data()
l_tbl <- gDRimport::load_data(
  manifest_file = gDRimport::manifest_path(td),
  df_template_files = gDRimport::template_path(td),
  results_file = gDRimport::result_path(td)
)
merge_data(
  l_tbl$manifest,
  l_tbl$treatments,
  l_tbl$data
)
  
```

```
prepare_input
```

Prepare input data common for all experiments

Description

Current steps

- refining nested confounders
- refining nested identifiers
- splitting df_ into (per experiment) df_list

Usage

```
prepare_input(x, ...)
```

Arguments

`x` data.table with raw data or MAE object with dose-response data
`...` additional parameters

Value

list of input data

Examples

```
td <- gDRimport::get_test_data()
l_tbl <- gDRimport::load_data(
  manifest_file = gDRimport::manifest_path(td),
  df_template_files = gDRimport::template_path(td),
  results_file = gDRimport::result_path(td)
)
df_ <- merge_data(
  l_tbl$manifest,
  l_tbl$treatments,
  l_tbl$data
)
nested_confounders = intersect(
  names(df_),
  gDRutils::get_env_identifiers("barcode")
)
prepare_input(df_, nested_confounders, NULL)
```

```
prepare_input.data.table
```

Prepare input data common for all experiments

Description

Current steps

- refining nested confounders
- refining nested identifiers
- splitting df_ into (per experiment) df_list

Usage

```
## S3 method for class 'data.table'
prepare_input(
  x,
  nested_confounders = gDRutils::get_env_identifiers("barcode"),
  nested_identifiers_l = .get_default_nested_identifiers(),
  ...
)
```

Arguments

x data.table with raw data
nested_confounders Character vector of the nested_confounders for a given assay. nested_keys is character vector of column names to include in the data.tables in the assays of the resulting SummarizedExperiment object. Defaults to the nested_identifiers and nested_confounders if passed through
nested_identifiers_l list with the nested_identifiers(character vectors) for single-agent and (optionally) for combination data
... additional parameters

Value

list of input data

```
prepare_input.MultiAssayExperiment
```

Prepare input data common for all experiments

Description

Current steps

- refining nested confounders
- refining nested identifiers
- splitting df_ into (per experiment) df_list

Usage

```
## S3 method for class 'MultiAssayExperiment'
prepare_input(
  x,
  nested_confounders = gDRutils::get_SE_identifiers(x[[1]], "barcode"),
  nested_identifiers_l = .get_default_nested_identifiers(x[[1]]),
  raw_data_field = "experiment_raw_data",
  split_data = TRUE,
  ...
)
```

Arguments

<code>x</code>	MAE object with dose-response data
<code>nested_confounders</code>	Character vector of the nested_confounders for a given assay. <code>nested_keys</code> is character vector of column names to include in the data.tables in the assays of the resulting SummarizedExperiment object. Defaults to the nested_identifiers and nested_confounders if passed through
<code>nested_identifiers_l</code>	list with the nested_identifiers(character vectors) for single-agent and (optionally) for combination data
<code>raw_data_field</code>	metadata field with raw data
<code>split_data</code>	Boolean indicating need of splitting the data into experiment types
<code>...</code>	additional parameters

Value

list of input data

`process_perturbations` *Cleanup additional perturbations in the data.table*

Description

This function processes drug and concentration columns in a data.table. It checks if there is only one unique drug (excluding a specified untreated tag) and if there are exactly two doses (one of which is 0). If these conditions are met, it creates a new column named after the drug and fills it with the doses, then removes the original drug and concentration columns.

Usage

```
process_perturbations(
  dt,
  drugs_cotrt_ids,
  conc_cotrt_ids,
  untreated_tag = "vehicle"
)
```

Arguments

<code>dt</code>	A data.table containing the data.
<code>drugs_cotrt_ids</code>	A vector of column names related to drugs.
<code>conc_cotrt_ids</code>	A vector of column names related to concentrations.
<code>untreated_tag</code>	A string representing the untreated tag (default is "vehicle").

Value

A modified data.table with new columns for the drugs and removed original drug and concentration columns.

Examples

```
dt <- data.table::data.table(
  drug1 = c("vehicle", "drugA", "drugA"),
  conc1 = c(0, 10, 0),
  drug2 = c("vehicle", "drugB", "drugB"),
  conc2 = c(0, 20, 0)
)
drugs_cotrt_ids <- c("drug1", "drug2")
conc_cotrt_ids <- c("conc1", "conc2")
dt <- process_perturbations(dt, drugs_cotrt_ids, conc_cotrt_ids)
print(dt)
```

```
replace_conc_with_standardized_conc
      Standardize concentrations.
```

Description

Utilize a map to standardize concentrations.

Usage

```
replace_conc_with_standardized_conc(
  original_concs,
  conc_map,
  original_conc_col,
  standardized_conc_col
)
```

Arguments

original_concs numeric vector of concentrations to replace using conc_map.

conc_map data.table of two columns named original_conc_col and standardized_conc_col.

original_conc_col string of the name of the column in conc_map containing the original concentrations to replace.

standardized_conc_col string of the name of the column in conc_map containing the standardized concentrations to use for replacement.

Value

numeric vector of standardized concentrations.

See Also

map_conc_to_standardized_conc

Examples

```
conc_map <- data.table::data.table(
  orig = c(0.99, 0.6, 0.456, 0.4),
  std = c(1, 0.6, 0.46, 0.4)
)
original_concs <- c(0.456, 0.456, 0.4, 0.99)
exp <- c(0.46, 0.46, 0.4, 1)
obs <- replace_conc_with_standardized_conc(
  original_concs,
  conc_map,
  original_conc_col = "orig",
  standardized_conc_col = "std"
)
```

split_raw_data

Split raw data into list based on the data types

Description

Split raw data into list based on the data types

Usage

```
split_raw_data(dt, type_col = "type")
```

Arguments

dt	data.table of raw drug response data containing both treated and untreated values with column specified in type_col argument.
type_col	string with column names in dt with info about data type. Defaults to "type".

Value

list with split data based on its data type

Author(s)

Bartosz Czech czech.bartosz@external.gene.com

Examples

```

cell_lines <- gDRtestData::create_synthetic_cell_lines()
drugs <- gDRtestData::create_synthetic_drugs()
dt_layout <- drugs[4:6, as.list(cell_lines[7:8, ]), names(drugs)]
dt_layout <- gDRtestData::add_data_replicates(dt_layout)
dt_layout <- gDRtestData::add_concentration(
  dt_layout,
  concentrations = 10 ^ (seq(-3, .5, .5))
)

dt_2 <-
  drugs[c(21, 26), as.list(cell_lines[which(cell_lines$clid %in% dt_layout$clid)]), names(drugs)]
dt_2 <- gDRtestData::add_data_replicates(dt_2)
dt_2 <- gDRtestData::add_concentration(
  dt_2,
  concentrations = 10 ^ (seq(-3, .5, .5))
)
colnames(dt_2)[colnames(dt_2) %in% c(colnames(drugs), "Concentration")] <-
  paste0(
    colnames(dt_2)[colnames(dt_2) %in% c(colnames(drugs), "Concentration")],
    "_2"
  )
dt_layout_2 <- dt_layout[dt_2, on = intersect(names(dt_layout), names(dt_2)),
  allow.cartesian = TRUE]
dt_merged_data <- gDRtestData::generate_response_data(dt_layout_2, 0)
dt <- identify_data_type(dt_merged_data)
split_raw_data(dt)

conc <- rep(seq(0, 0.3, 0.1), 2)
ctrl_dt <- S4Vectors::DataFrame(
  ReadoutValue = c(2, 2, 1, 1, 2, 1),
  Concentration = rep(0, 6),
  masked = FALSE,
  DrugName = rep(c("DRUG_10", "vehicle", "DRUG_8"), 2),
  CellLineName = "CELL1"
)

trt_dt <- S4Vectors::DataFrame(
  ReadoutValue = rep(seq(1, 4, 1), 2),
  Concentration = conc,
  masked = rep(FALSE, 8),
  DrugName = c("DRUG_10", "DRUG_8"),
  CellLineName = "CELL1"
)
input_dt <- data.table::as.data.table(rbind(ctrl_dt, trt_dt))
input_dt$Duration <- 72
input_dt$CorrectedReadout2 <- input_dt$ReadoutValue
split_dt <- identify_data_type(input_dt)
split_raw_data(split_dt)

```

test_synthetic_data	<i>Testing synthetic data form gDRtestData package</i>
---------------------	--

Description

Testing synthetic data form gDRtestData package

Usage

```
test_synthetic_data(  
  original,  
  data,  
  dataName,  
  override_untrt_controls = NULL,  
  assays = c("Normalized", "Averaged", "Metrics"),  
  tolerance = 0.001  
)
```

Arguments

original	original MAE assay
data	datase MAE or data.table
dataName	dataset name
override_untrt_controls	named list containing defining factors in the treatments
assays	assays to test
tolerance	tolerance factor

Value

NULL

Examples

```
set.seed(2)  
cell_lines <- gDRtestData::create_synthetic_cell_lines()  
drugs <- gDRtestData::create_synthetic_drugs()  
data <- "finalMAE_small"  
original <- gDRutils::get_synthetic_data(data)  
test_synthetic_data(original, original, "test")
```

Index

- * **annotation**
 - annotate_dt_with_cell_line, [3](#)
 - annotate_dt_with_drug, [4](#)
 - annotate_mae_with_cell_line, [4](#)
 - annotate_mae_with_drug, [5](#)
 - annotate_se_with_cell_line, [6](#)
 - annotate_se_with_drug, [7](#)
 - get_cell_line_annotation, [32](#)
 - get_cellline_annotation_from_dt, [33](#)
 - get_drug_annotation, [34](#)
 - get_drug_annotation_from_dt, [35](#)
- * **calculate_GR**
 - calculate_GR_value, [20](#)
- * **combinations**
 - calculate_excess, [19](#)
 - calculate_score, [22](#)
- * **convert_to_raw_data**
 - convert_mae_to_raw_data, [26](#)
 - convert_se_to_raw_data, [26](#)
- * **data_type**
 - identify_data_type, [40](#)
 - process_perturbations, [49](#)
 - split_raw_data, [51](#)
- * **map_df**
 - map_df, [43](#)
 - map_ids_to_fits, [44](#)
 - map_untreated, [45](#)
- * **merge_data**
 - merge_data, [45](#)
- * **prepare_input**
 - prepare_input, [46](#)
 - prepare_input.data.table, [47](#)
 - prepare_input.MultiAssayExperiment, [48](#)
- * **runDrugResponseProcessingPipeline**
 - average_SE, [13](#)
 - compute_growth_rates, [24](#)
 - fit_SE.combinations, [28](#)
 - fit_SE.timecourse, [29](#)
 - get_period_timepoints, [36](#)
 - growth_rates_to_se, [37](#)
- * **test_utils**
 - test_synthetic_data, [53](#)
- * **utils**
 - cleanup_metadata, [23](#)
 - data_model, [27](#)
 - data_model.character, [27](#)
 - data_model.data.table, [28](#)
 - get_assays_per_pipeline_step, [32](#)
 - get_default_nested_identifiers, [34](#)
 - grr_matches, [38](#)
 - identify_keys, [41](#)
 - replace_conc_with_standardized_conc, [50](#)
- annotate_dt_with_cell_line, [3](#)
- annotate_dt_with_drug, [4](#)
- annotate_mae_with_cell_line, [4](#)
- annotate_mae_with_drug, [5](#)
- annotate_se_with_cell_line, [6](#)
- annotate_se_with_drug, [7](#)
- apply_combo_excess, [7](#), [9](#), [11](#)
- apply_combo_isobolograms, [8](#), [8](#), [11](#)
- apply_combo_sa_fits, [8](#), [9](#), [10](#)
- apply_combo_scores, [8](#), [9](#), [11](#), [11](#)
- apply_fit, [18](#), [25](#), [31](#), [37](#), [40](#)
- apply_fit_to_se, [12](#)
- average_SE, [13](#)
- bliss_fit_fn, [12](#), [18](#)
- calculate_Bliss, [12](#)
- calculate_endpt_GR_value (calculate_GR_value), [20](#)
- calculate_excess, [12](#), [19](#)
- calculate_GR_value, [20](#)
- calculate_HSA, [12](#)
- calculate_score, [12](#), [22](#)

calculate_time_dep_GR_value
 (calculate_GR_value), 20
cleanup_metadata, 23
compute_growth_rates, 24, 29–31, 36, 37
convert_mae_to_raw_data, 26
convert_se_to_raw_data, 26
create_and_normalize_SE (average_SE), 13
create_SE (average_SE), 13

data_model, 27
data_model.character, 27
data_model.data.table, 28

fit_drug_response_metrics, 30
fit_SE, 31
fit_SE (average_SE), 13
fit_SE.combinations, 7–11, 28, 31
fit_SE.timecourse, 24, 25, 29, 37

get_assays_per_pipeline_step, 32
get_cell_line_annotation, 32
get_cellline_annotation_from_dt, 33
get_default_nested_identifiers, 34
get_drug_annotation, 34
get_drug_annotation_from_dt, 35
get_fit_profile, 24
get_period_timepoints, 36
growth_rates_to_se, 24, 25, 31, 37
grr_matches, 38

hss_fit_fn, 12, 39

identify_data_type, 40
identify_keys, 41

map_df, 43
map_ids_to_fits, 44
map_untreated, 45
match, 38
merge, 38
merge_data, 45

normalize_SE, 31
normalize_SE (average_SE), 13

predict_efficacy_from_conc, 12
prepare_input, 46
prepare_input.data.table, 47
prepare_input.MultiAssayExperiment, 48
process_perturbations, 49

replace_conc_with_standardized_conc,
 50
runDrugResponseProcessingPipeline
 (average_SE), 13
runDrugResponseProcessingPipelineFxn
 (average_SE), 13

split_raw_data, 51
SummarizedExperiment, 8–11, 15, 16, 24, 29,
 30

test_synthetic_data, 53