Package: gDRcore (via r-universe)

October 29, 2026

Type Package

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

Version 1.10.0

Date 2026-02-18

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, futile.logger, gDRutils (>= 1.7.1), 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

RoxygenNote 7.3.3

Roxygen list(markdown = TRUE)

SwitchrLibrary gDRcore

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

Repository Bioconductor 3.23

Date/Publication 2026-04-28 13:01:09 UTC

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

RemoteRef RELEASE_3_23

RemoteSha a73889b1a0cd225f050009a2595c02b009a562c5

Contents

.map_references . . . 3
annotate_dt_with_cell_line . . . 3
annotate_dt_with_drug . . . 4
annotate_mae_with_cell_line . . . 5
annotate_mae_with_drug . . . 5
annotate_se_with_cell_line . . . 6
annotate_se_with_drug . . . 7
average_SE . . . 8
calculate_excess . . . 13
calculate_GR_value . . . 14
calculate_matrix_metric . . . 16
calculate_score . . . 18
cleanup_metadata . . . 19
convert_mae_to_raw_data . . . 19
convert_se_to_raw_data . . . 20
data_model . . . 21
data_model.character . . . 21
data_model.data.table . . . 22
fit_SE.combinations . . . 22
get_assays_per_pipeline_step . . . 23
get_cell_line_annotation . . . 24
get_cellline_annotation_from_dt . . . 25
get_default_nested_identifiers . . . 25
get_drug_annotation . . . 26
get_drug_annotation_from_dt . . . 27
grr_matches . . . 27
identify_data_type . . . 29
identify_keys . . . 30
map_df . . . 32
map_ids_to_fits . . . 33
map_untreated . . . 34
merge_data . . . 34
order_result_df . . . 35
prepare_input . . . 35

prepare_input.data.table 36
prepare_input.MultiAssayExperiment 37
process_perturbations 38
replace_conc_with_standardized_conc 39
split_raw_data 40
test_synthetic_data 42

Index **43**

`.map_references` *Map references*

Description

Map references

Usage

```
.map_references(
  mat_elem,
  rowData_colnames = c(gDRutils::get_env_identifiers("duration"), paste0(c("drug",
    "drug_name", "drug_moa"), "3"))
)
```

Arguments

`mat_elem`	data.table input containing experimental metadata and row identifiers.
`rowData_colnames`	character vector of variables (column names) used to identify and map reference treatments.

Value

list

`annotate_dt_with_cell_line` *annotate_dt_with_cell_line*

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

data	data.table with dose-response data
drug_annotation	data.table with drug annotations
fill	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

se	SummarizedExperiment 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

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)
```

average_SE *Run drug response processing pipeline*

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 concetrations 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 v ectors) 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 asssays 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 uniquenested_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
)
```

`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

`metric`	data.table often representing readouts derived by calculating some metric. Examples of this could include hsa or bliss calculations from single-agent data.
`measured`	data.table often representing measured data from an experiment.
`series_identifiers`	character vector of identifiers in `measured` or `metric` which define a unique data point.

metric_col	string of the column in metric to use in excess calculation.
measured_col	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 *Calculate a GR value.*

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_matrix_metric

Calculate a metric for combination data.

Description

Calculate a metric based off of single-agent values in combination screens.

Usage

```
calculate_HSA(sa1, series_id1, sa2, series_id2, metric)

calculate_Bliss(
  sa1,
  series_id1,
  sa2,
  series_id2,
  metric,
  measured_col = "smooth"
)
```

```
.calculate_matrix_metric(
  sa1,
  series_id1,
  sa2,
  series_id2,
  metric,
  FXN,
  measured_col = "x"
)
```

Arguments

sa1	data.table containing single agent data where entries in series_id2 are all 0. Columns of the data.table include identifiers and the metric of interest. Metric is stored in the 'x' column.
series_id1	String representing the column within sa1 that represents id1.
sa2	data.table containing single agent data where entries in series_id1 are all 0. Columns of the data.table include identifiers and the metric of interest.n Metric is stored in the 'x' column.
series_id2	String representing the column within sa2 that represents id2.
metric	String specifying the metric of interest. Usually either 'GRvalue' or 'Relative-Viability'.
measured_col	String specyfying the measured colname.
FXN	Function to apply to the single-agent fits to calculate a metric.

Details

calculate_HSA takes the minimum of the two single agents readouts. calculate_Bliss performs Bliss additivity calculation based on the single agent effects, defined as 1-x for the corresponding normalization. See https://www.sciencedirect.com/science/article/pii/S1359644619303460?via%3Dihub#tb0005 for more details.

Value

data.table containing a single row for every unique combination of the two series identifiers and the corresponding calculated metric for each row.

Examples

```
n <- 10
sa1 <- data.table::data.table(conc = seq(n), conc2 = rep(0, n), smooth = seq(n))
sa2 <- data.table::data.table(conc = rep(0, n), conc2 = seq(n), smooth = seq(n))
calculate_HSA(sa1, "conc", sa2, "conc2", "smooth")
n <- 10
sa1 <- data.table::data.table(conc = seq(n), conc2 = rep(0, n), smooth = seq(n))
sa2 <- data.table::data.table(conc = rep(0, n), conc2 = seq(n), smooth = seq(n))
calculate_Bliss(sa1, "conc", sa2, "conc2", "smooth")
```

`calculate_score` *Calculate score for HSA and Bliss*

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 *cleanup_metadata*

Description

Cleanup a data.table with metadata

Usage

```
cleanup_metadata(df_metadata)
```

Arguments

df_metadata	a data.table with metadata

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)
```

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 *Detect model of data*

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 *Detect model of data from experiment name*

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])
```

---

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_identifier = "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)
```

`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

x	vector. The values to be matched. Long vectors are not currently supported.
y	vector. The values to be matched. Long vectors are not currently supported.
all.x	logical; if TRUE, then each value in x will be included even if it has no matching values in y
all.y	logical; if TRUE, then each value in y will be included even if it has no matching values in x
list	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.
indexes	logical. Whether to return the indices of the matches or the actual values.
nomatch	the value to be returned in the case when no match is found. If not provided and indexes=TRUE, 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 length+1. If indexes=FALSE, 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)
```

`identify_data_type` *Identify type of data*

Description

Identify type of data

Usage

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

Arguments

`dt`	data.table of raw drug response data containing both treated and untreated values
`codilution_conc`	integer of maximum number of concentration ratio of co-treatment to classify as codilution data type; defaults to 2
`matrix_conc`	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 bartosz.czech@contractors.roche.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` *Identify untreated rows based on Drug treatment alone*

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` *merge_data*

Description

Merge all the input data into a single data.table

Usage

```
merge_data(manifest, treatments, data)
```

Arguments

`manifest`	a data.table with a manifest info
`treatments`	a data.table with a treaatments info
`data`	a data.table with a raw data info

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
)
```

---

order_result_df *Order_result_df*

Description

Order a data.table with results

Usage

```
order_result_df(df_)
```

Arguments

df_	a data.table with results

Value

a ordered data.table with results

---

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

x	MAE object with dose-response 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
raw_data_field	metadata field with raw data
split_data	Boolean indicating need of splitting the data into experiment types
...	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

dt	A data.table containing the data.
drugs_cotrt_ids	A vector of column names related to drugs.
conc_cotrt_ids	A vector of column names related to concentrations.
untreated_tag	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 bartosz.czech@contractors.roche.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` *Testing synthetic data form gDRtestData package*

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, 5
annotate_mae_with_drug, 5
annotate_se_with_cell_line, 6
annotate_se_with_drug, 7
get_cell_line_annotation, 24
get_cellline_annotation_from_dt, 25
get_drug_annotation, 26
get_drug_annotation_from_dt, 27
∗ **calculate_GR**
calculate_GR_value, 14
∗ **combinations**
calculate_excess, 13
calculate_matrix_metric, 16
calculate_score, 18
∗ **convert_to_raw_data**
convert_mae_to_raw_data, 19
convert_se_to_raw_data, 20
∗ **data_type**
identify_data_type, 29
process_perturbations, 38
split_raw_data, 40
∗ **map_df**
.map_references, 3
map_df, 32
map_ids_to_fits, 33
map_untreated, 34
∗ **merge_data**
merge_data, 34
∗ **prepare_input**
prepare_input, 35
prepare_input.data.table, 36
prepare_input.MultiAssayExperiment, 37
∗ **runDrugResponseProcessingPipeline**
average_SE, 8
fit_SE.combinations, 22
∗ **test_utils**
test_synthetic_data, 42
∗ **utils**
cleanup_metadata, 19
data_model, 21
data_model.character, 21
data_model.data.table, 22
get_assays_per_pipeline_step, 23
get_default_nested_identifiers, 25
grr_matches, 27
identify_keys, 30
order_result_df, 35
replace_conc_with_standardized_conc, 39
.calculate_matrix_metric *(calculate_matrix_metric)*, 16
.map_references, 3

annotate_dt_with_cell_line, 3
annotate_dt_with_drug, 4
annotate_mae_with_cell_line, 5
annotate_mae_with_drug, 5
annotate_se_with_cell_line, 6
annotate_se_with_drug, 7
average_SE, 8

calculate_Bliss *(calculate_matrix_metric)*, 16
calculate_endpt_GR_value *(calculate_GR_value)*, 14
calculate_excess, 13
calculate_GR_value, 14
calculate_HSA *(calculate_matrix_metric)*, 16
calculate_matrix_metric, 16
calculate_score, 18
calculate_time_dep_GR_value *(calculate_GR_value)*, 14
cleanup_metadata, 19

convert_mae_to_raw_data, 19
convert_se_to_raw_data, 20
create_and_normalize_SE *(*average_SE*)*, 8
create_SE *(*average_SE*)*, 8

data_model, 21
data_model.character, 21
data_model.data.table, 22

fit_SE *(*average_SE*)*, 8
fit_SE.combinations, 22

get_assays_per_pipeline_step, 23
get_cell_line_annotation, 24
get_cellline_annotation_from_dt, 25
get_default_nested_identifiers, 25
get_drug_annotation, 26
get_drug_annotation_from_dt, 27
grr_matches, 27

identify_data_type, 29
identify_keys, 30

map_df, 32
map_ids_to_fits, 33
map_untreated, 34
match, *27*
merge, *28*
merge_data, 34

normalize_SE *(*average_SE*)*, 8

order_result_df, 35

prepare_input, 35
prepare_input.data.table, 36
prepare_input.MultiAssayExperiment, 37
process_perturbations, 38

replace_conc_with_standardized_conc, 39
runDrugResponseProcessingPipeline *(*average_SE*)*, 8
runDrugResponseProcessingPipelineFxns *(*average_SE*)*, 8

split_raw_data, 40
SummarizedExperiment, *10, 11*, *23*

test_synthetic_data, 42