Package ‘gDRcore’

May 24, 2024

Type Package

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

Version 1.3.0

Date 2024-04-23

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
(>= 1.1.3), MultiAssayExperiment, purrr, stringr, S4Vectors, SummarizedExperiment, data.table

Suggests BiocStyle, gDRstyle (>= 1.1.2), gDRimport (>= 1.1.4), gDRTesData (>= 1.1.6), IRanges, knitr, pkgbuild, qs, 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 7.3.1

Roxygen list(markdown = TRUE)
SwitchrLibrary  gDRcore

**git_url** https://git.bioconductor.org/packages/gDRcore

**git_branch**  devel

**git_last_commit**  54a378c

**git_last_commit_date**  2024-04-30

**Repository**  Bioconductor 3.20

**Date/Publication**  2024-05-24

**Author**  Bartosz Czech [aut] (<https://orcid.org/0000-0002-9908-3007>),
Arkadiusz Gladki [cre, aut] (<https://orcid.org/0000-0002-7059-6378>),
Marc Hafner [aut] (<https://orcid.org/0000-0003-1337-7598>),
Pawel Piatkowski [aut],
Natalia Potocka [aut],
Dariusz Scigocki [aut],
Janina Smola [aut],
Sergiu Mocanu [aut],
Marcin Kamianowski [aut],
Allison Vuong [aut]

**Maintainer**  Arkadiusz Gladki <gladki.arkadiusz@gmail.com>

## Contents

- gDRcore-package .......................................................... 3
- .map_references ............................................................ 4
- .standardize_conc .......................................................... 5
- add_CellLine_annotation .................................................. 6
- add_Drug_annotation ....................................................... 7
- add_intermediate_data .................................................... 8
- average_SE ................................................................. 9
- calculate_excess .......................................................... 14
- calculate_GR_value ....................................................... 15
- calculate_matrix_metric ................................................ 17
- cleanup_metadata ......................................................... 19
- convert_mae_to_raw_data ............................................... 20
- convert_se_to_raw_data ................................................ 20
- data_model ................................................................. 21
- data_model.character .................................................... 21
- data_model.data.table ................................................... 22
- define_matrix_grid_positions ......................................... 22
- do_skip_step ............................................................. 23
- fit_SE.combinations ..................................................... 23
- generateCodilution ...................................................... 24
- generateCodilutionSmall ................................................ 25
- generateComboMatrix ..................................................... 25
- generateComboMatrixSmall .............................................. 25
- generateComboNoNoiseData .............................................. 26
gDRcore-package

generateComboNoNoiseData2 ........................................ 26
generateComboNoNoiseData3 ........................................ 26
generateLigandData ...................................................... 27
generateMediumData ..................................................... 27
generateNoiseRawData ................................................... 27
generateNoNoiseRawData ............................................... 28
generateTripleComboMatrix .......................................... 28
get_assays_per_pipeline_step ........................................ 29
get_default_nested_identifiers ...................................... 29
get_mae_from_intermediate_data .................................... 30
get_pipeline_steps ....................................................... 30
grr_matches ............................................................... 31
identify_data_type ...................................................... 32
identify_keys ............................................................. 34
is_preceding_step ....................................................... 35
map_conc_to_standardized_conc ..................................... 35
map_df ................................................................. 36
map_ids_to_fits ......................................................... 38
map_untreated .......................................................... 39
merge_data ............................................................. 39
order_result_df ........................................................ 40
prepare_input .......................................................... 40
prepare_input.data.table ............................................. 41
prepare_input.MultiAssayExperiment .............................. 42
read_intermediate_data ............................................... 43
remove_drug_batch ..................................................... 44
replace_conc_with_standardized_conc .............................. 44
round_concentration .................................................. 45
save_intermediate_data ................................................. 46
split_raw_data .......................................................... 46
test_synthetic_data ..................................................... 48

Index

---

gDRcore-package

---

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

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.

Value

package help page
Note
To learn more about functions start with help(package = "gDRcore")

Author(s)

Maintainer: Arkadiusz Gladki <gladki.arkadiusz@gmail.com> (ORCID)
Authors:
- Bartosz Czech <bartosz.czech@contractors.roche.com> (ORCID)
- Marc Hafner (ORCID)
- Pawel Piatkowski
- Natalia Potocka
- Dariusz Scigocki
- Janina Smola
- Sergiu Mocanu
- Marcin Kamianowski
- Allison Vuong

See Also
Useful links:
- https://github.com/gdrplatform/gDRcore
- https://gdrplatform.github.io/gDRcore/
- Report bugs at https://github.com/gdrplatform/gDRcore/issues

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 input data frame
rowData_colnames character vector of variables for the mapping of reference treatments
Details
Using the given rownames, map the treated and reference conditions.

Value
list

---

**.standardize_conc**

*Standardize concentration values.*

Description
Standardize concentration values.

Usage

```
.standardize_conc(conc)
```

Arguments

- `conc` numeric vector of the concentrations

Details
If no conc are passed, NULL is returned.

Value

vector of standardized concentrations

Examples

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

add_CellLine_annotation

Description

add cellline annotation to a data.table with metadata

Usage

add_CellLine_annotation(
  dt_metadata,
  DB_cellid_header = "cell_line_identifier",
  DB_cell_annotate = c("cell_line_name", "primary_tissue", "doubling_time",
    "parental_identifier", "subtype"),
  fname = "cell_lines.csv",
  fill = "unknown",
  annotationPackage = if ("gDRinternal" %in% .packages(all.available = TRUE)) {
    "gDRinternal"
  } else {
    "gDRtestData"
  },
  externalSource = Sys.getenv("GDR_CELLLINE_ANNOTATION")
)

Arguments

dt_metadata data.table with metadata
DB_cellid_header string with colnames with cell line identifier in the annotation file
DB_cell_annotate character vector with mandatory colnames used in the annotation file
fname string with file name with annotation
fill string indicating how unknown cell lines should be filled in the DB
annotationPackage string indication name of the package containing cellline annotation
externalSource string with path to external file with annotation data; by default it checks `GDR_CELLLINE_ANNOTATION` env var. This file should contain columns such as gnumber, drug_name and drug_moa

Details

The logic of adding cellline annotation for dt_metadata based on the annotation file stored in gDRtestData. Other fields are set as "unknown". This approach will be corrected once we will implement final solution for adding cell lines.
add_Drug_annotation

Value
data.table with metadata with annotated cell lines

Examples

add_CellLine_annotation(
data.table::data.table(
    clid = "123",
    CellLineName = "name of the cell line")
)

add_Drug_annotation

Description

add drug annotation to a data.table with metadata

Usage

add_Drug_annotation(
dt_metadata,
fname = "drugs.csv",
fill = "unknown",
annotationPackage = if ("gDRinternal" %in% .packages(all.available = TRUE)) {
    "gDRinternal"
} else {
    "gDRtestData"
},
externalSource = Sys.getenv("GDR_DRUG_ANNOTATION")
)

Arguments

dt_metadata  data.table with metadata
fname        string with file name with annotation
fill          string indicating how unknown cell lines should be filled in the DB
annotationPackage  string indication name of the package containing drug annotation
externalSource  string with path to external file with annotation data; by default it checks 'GDR_DRUG_ANNOTATION' env var. This file should contain columns such as gnumber, drug_name, and drug_moa
add_intermediate_data

Details

The logic of adding drug annotation for dt_metadata based on the annotation file stored in gDRtest-Data.

Value

data.table with metadata with annotated drugs

Examples

add_Drug_annotation(
  data.table::data.table(
    Gnumber = "drug_id",
    DrugName = "name of the drug"
  )
)

add_intermediate_data  add intermediate data (qs files) for given mae

Description

add intermediate data (qs files) for given mae

Usage

add_intermediate_data(mae, data_dir, steps = get_pipeline_steps())

Arguments

  mae     mae with dose-response data
  data_dir output directory
  steps    character vector with pipeline steps for which intermediate data should be saved

Value

  NULL
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,
  override_masked = FALSE,
  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),
countrol_mean_fxn = function(x) {
  mean(x, trim = 0.25)
},
countrol_assay = "Controls",
raw_treated_assay = "RawTreated",
normalized_assay = "Normalized",
n_digit_rounding = 4
)

create_and_normalize_SE(  
  df_,
  data_type,
  readout = "ReadoutValue",
countrol_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,
n_digit_rounding = 4,
countrol_assay = "Controls",
raw_treated_assay = "RawTreated",
normalized_assay = "Normalized"
)

runDrugResponseProcessingPipeline(  
  x,
  readout = "ReadoutValue",
countrol_mean_fxn = function(x) {
  mean(x, trim = 0.25)
},
nested_identifiers_l = NULL,
nested_confounders = gDRutils::get_env_identifiers("barcode"),
override_untrt_controls = NULL,
override_masked = FALSE,
n_digit_rounding = 4,
n_point_cutoff = 4,
countrol_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.
override_masked boolean indicating whether or not to override the masked wells in the averaging and include all wells. Defaults to FALSE.
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.
p_cutoff 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".
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 qs 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,
    masked = rep(c(TRUE, TRUE, TRUE, FALSE), 9),
    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", "masked_tag" = "masked")
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",
    override_masked = FALSE,
    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")
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.
**calculate_GR_value**

```
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, 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, 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
)
```

**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,
)```
calculate_GR_value
calculate_time_dep_GR_value

calculate_endpt_GR_value

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
dv <- seq(0.1, 1, 0.1)
corrected <- seq(41000, 50000, 1000)
day0 <- seq(91000, 95500, 500)
untre <- rep(c(115000, 118000), 5)

calculate_GR_value(
  rel_viability = dv,
  corrected_readout = corrected,
  day0_readout = day0,
  untrt_readout = untre,
  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,
calculate_matrix_metric

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. 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 specifying 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
cleanup_metadata

```r
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")
```

---

**Description**

Cleanup a data.table with metadata

**Usage**

```r
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**

```r
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
```r
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**

```r
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**

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

---

**data_model.character**

**Detect model of data from experiment name**

**Description**

Detect model of data from experiment name

**Usage**

```r
## 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**

```r
## 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

---

**define_matrix_grid_positions  Define matrix grid positions**

**Description**
Define matrix grid positions

**Usage**

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

**Arguments**

- `conc1` drug_1 concentration
- `conc2` drug_2 concentration

**Details**

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

**Value**
list with axis grid positions
do_skip_step

Description
check if the given step can be skipped if partial run is chosen

Usage
do_skip_step(current_step, start_from, steps = get_pipeline_steps())

Arguments
- current_step: string with the step to be evaluated
- start_from: string indicating the pipeline step from which partial run should be launched
- steps: charvect with all available steps

Value
logical

fit_SE.combinations

Description
Perform fittings for combination screens.

Usage
fit_SE.combinations(
  se,
  data_type = gDRutils::get_experiment_groups("combination"),
  series_identifiers = NULL,
  normalization_types = c("GR", "RV"),
  averaged_assay = "Averaged",
  metrics_assay = "Metrics"
)
generateCodilution

**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.

**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**

```r
fmae.cms <- gDRutils::get_synthetic_data("finalMAE_combo_matrix_small")

se1 <- fmae.cms[[gDRutils::get_experiment_groups("combination")]]
SummarizedExperiment::assays(se1) <-
  SummarizedExperiment::assays(se1)["Averaged"]
fit_SE.combinations(se1[1, 1])
```

---

generateCodilution
generateCodilution

description generateCodilution

**Usage**

`generateCodilution(cell_lines, drugs, save = TRUE)`

**Value**

data.table with raw input data or MAE with processed data
**generateCodilutionSmall**

Description

**generateCodilutionSmall**

Usage

```r
generateCodilutionSmall(cell_lines, drugs, save = TRUE)
```

Value

data.table with raw input data or MAE with processed data

**generateComboMatrix**

Description

**generateComboMatrix**

Usage

```r
generateComboMatrix(cell_lines, drugs, save = TRUE)
```

Value

data.table with raw input data or MAE with processed data

**generateComboMatrixSmall**

Description

**generateComboMatrixSmall**

Usage

```r
generateComboMatrixSmall(cell_lines, drugs, save = TRUE)
```

Value

data.table with raw input data or MAE with processed data
generateComboNoNoiseData

Description

generateComboNoNoiseData

Usage

generateComboNoNoiseData(cell_lines, drugs, save = TRUE)

Value

data.table with raw input data or MAE with processed data

generateComboNoNoiseData2

Description

generateComboNoNoiseData2

Usage

generateComboNoNoiseData2(cell_lines, drugs, save = TRUE)

Value

data.table with raw input data or MAE with processed data

generateComboNoNoiseData3

Description

generateComboNoNoiseData3

Usage

generateComboNoNoiseData3(cell_lines, drugs, save = TRUE)

Value

data.table with raw input data or MAE with processed data
generateLigandData

Description

generateLigandData

Usage

generateLigandData(cell_lines, drugs, save = TRUE)

Value

data.table with raw input data or MAE with processed data

generateMediumData

Description

generateMediumData

Usage

generateMediumData(cell_lines, drugs, save = TRUE)

Value

data.table with raw input data or MAE with processed data

generateNoiseRawData

Description

generateNoiseRawData

Usage

generateNoiseRawData(cell_lines, drugs, save = TRUE)

Value

data.table with raw input data or MAE with processed data
generateNoNoiseRawData

Description

generateNoNoiseRawData

Usage

generateNoNoiseRawData(cell_lines, drugs, save = TRUE)

Value

data.table with raw input data or MAE with processed data

generateTripleComboMatrix

Description

generateTripleComboMatrix

Usage

generateTripleComboMatrix(cell_lines, drugs, save = TRUE)

Value

data.table with raw input data or MAE with processed data
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_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)
get_pipeline_steps

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_mae_from_intermediate_data

get mae dataset from intermediate data

Description

get mae dataset from intermediate data

Usage

get_mae_from_intermediate_data(data_dir)

Arguments

data_dir  directory with intermediate data

Value

MAE object

get_pipeline_steps

get pipeline steps

Description

get pipeline steps

Usage

get_pipeline_steps()

Value

vector with steps
**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**

```r
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`
Identify type of data

**Description**

Identify type of data

```r
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.

**Value**

data.table

**Examples**

```r
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)
```
**Usage**

```r
df, codilution_conc = 2, matrix_conc = 1
```

**Arguments**

- `df`: 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**

```r
conc <- rep(seq(0, 0.3, 0.1), 2)
ctrl_df <- 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_df <- S4Vectors::DataFrame(
    ReadoutValue = rep(seq(1, 4, 1), 2),
    Concentration = conc,
    masked = rep(FALSE, 8),
    DrugName = c("DRUG_10", "DRUG_8"),
    CellLineName = "CELL1"
)

input_df <- data.table::as.data.table(rbind(ctrl_df, trt_df))
input_df$Duration <- 72
input_df$CorrectedReadout2 <- input_df$ReadoutValue
identify_data_type(input_df)
```
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)

### is_preceding_step

**check if the given step is preceding the step chosen in the partial run**

**Description**

check if the given step is preceding the step chosen in the partial run

**Usage**

```r
is_preceding_step(current_step, start_from, steps = get_pipeline_steps())
```

**Arguments**

- `current_step`: string with the step to be evaluated
- `start_from`: string indicating the pipeline step from which partial run should be launched
- `steps`: charvect with all available steps

**Value**

logical

### map_conc_to_standardized_conc

Create a mapping of concentrations to standardized concentrations.

**Description**

Create a mapping of concentrations to standardized concentrations.

**Usage**

```r
map_conc_to_standardized_conc(conc1, conc2)
```
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")
)
map_df

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", "ref_Endpoint").

Details

If `override_untrt_controls` is specified, TODO: FILL ME!

Value

named list mapping treated metadata to untreated metadata.

See Also

`identify_keys`

Examples

```r
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")
ref_df <- md_df[ref, ]
trt_df <- md_df[!ref, ]
Keys <- identify_keys(trt_df)
ref_type <- "untrt_Endpoint"
map_df(
  trt_df,
  ref_df,
  ref_cols = Keys[[ref_type]],
  ref_type = ref_type
)
```
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

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>pred</td>
<td>numeric vector for which you want predictions.</td>
</tr>
<tr>
<td>match_col</td>
<td>vector to match on fittings to get the correct fit.</td>
</tr>
<tr>
<td>fittings</td>
<td>data.table of fit metrics.</td>
</tr>
<tr>
<td>fitting_id_col</td>
<td>string of the column name in fittings that should be used to match with match_col.</td>
</tr>
</tbody>
</table>

Value

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

Examples

```r
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 input data frame

Details

Using the given rownames, map the untreated conditions

Value

list

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 treatments 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

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

Description

Prepare input data common for all experiments

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-reponse data
- `...`: additional parameters

**Value**

- list of input data

**Examples**

```r
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)
```

---

**Description**

Prepare input data common for all experiments

**Current steps**

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

**Usage**

```r
## 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 a 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

Description

Current steps

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

Usage

```r
## 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.

---

**read_intermediate_data**

- `read intermediate data for the given experiment and step to qs file`

**Description**

read intermediate data for the given experiment and step to qs file

**Usage**

`read_intermediate_data(path, step, experiment)`

**Arguments**

- `path`: string with the input directory of the qs file.
- `step`: string with the step name.
- `experiment`: string with the experiment name.

**Value**

- se
**replace_conc_with_standardized_conc**

*Standardize concentrations.*

**Description**

Utilize a map to standardize concentrations.

**Usage**

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

**remove_drug_batch**

*Remove batch from Gnumber*

**Description**

Remove batch from Gnumber

**Usage**

```r
remove_drug_batch(drug)
```

**Arguments**

- `drug` : drug name

**Value**

Gnumber without a batch

**Examples**

```r
remove_drug_batch("DRUG.123")
```
round_concentration

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)
obsv <- replace_conc_with_standardized_conc(
  original_concs,
  conc_map,
  original_conc_col = "orig",
  standardized_conc_col = "std"
)

round_concentration

Round concentration to ndigit significant digits

Description

Round concentration to ndigit significant digits

Usage

round_concentration(x, ndigit = 3)

Arguments

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

rounded x

Examples

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

save_intermediate_data

save intermediate data for the given experiment and step to qs file

Description

save intermediate data for the given experiment and step to qs file

Usage

save_intermediate_data(path, step, experiment, se)

Arguments

path string with the save directory for the qs file
step string with the step name
experiment string with the experiment name
se output se

Value

NULL

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(df, type_col = "type")
Arguments

- **df**: 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 `df` 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

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

df_2 <-
  drugs[c(21, 26), as.list(cell_lines[which(cell_lines$clid %in% df_layout$clid)]), names(drugs)]
df_2 <- gDRtestData::add_data_replicates(df_2)
df_2 <- gDRtestData::add_concentration(
  df_2,  
  concentrations = 10 ^ (seq(-3, .5, .5))
)
colnames(df_2)[colnames(df_2) %in% c(colnames(drugs), "Concentration")]
  <- paste0(
    colnames(df_2)[colnames(df_2) %in% c(colnames(drugs), "Concentration")],
    "_2"
  )
df_layout_2 <- df_layout[df_2, on = intersect(names(df_layout), names(df_2)),
   allow.cartesian = TRUE]
df_merged_data <- gDRtestData::generate_response_data(df_layout_2, 0)
df <- identify_data_type(df_merged_data)
split_raw_data(df)
conc <- rep(seq(0, 0.3, 0.1), 2)
ctrl_df <- 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"
)
```

test_synthetic_data

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      dataset 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

```r
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
  add_CellLine_annotation, 6
  add_Drug_annotation, 7
  remove_drug_batch, 44
* calculate_GR
  calculate_GR_value, 15
* combinations
  calculate_excess, 14
  calculate_matrix_metric, 17
  define_matrix_grid_positions, 22
* convert_to_raw_data
  convert_mae_to_raw_data, 20
  convert_se_to_raw_data, 20
* data_type
  identify_data_type, 32
  split_raw_data, 46
* internal
  add_intermediate_data, 8
  do_skip_step, 23
  gDRcore-package, 3
  generateCodilution, 24
  generateCodilutionSmall, 25
  generateComboMatrix, 25
  generateComboMatrixSmall, 25
  generateComboNoNoiseData, 26
  generateComboNoNoiseData2, 26
  generateComboNoNoiseData3, 26
  generateLigandData, 27
  generateMediumData, 27
  generateNoiseRawData, 27
  generateNoNoiseRawData, 28
  generateTripleComboMatrix, 28
  get_mae_from_intermediate_data, 30
  get_pipeline_steps, 30
  is_preceding_step, 35
  read_intermediate_data, 43
  save_intermediate_data, 46
* map_df
  .map_references, 4
  map_df, 36
  map_ids_to_fits, 38
  map_untrated, 39
* merge_data
  merge_data, 39
* prepare_input
  prepare_input, 40
  prepare_input.data.table, 41
  prepare_input.MultiAssayExperiment, 42
* runDrugResponseProcessingPipeline
  average_SE, 9
  fit_SE.combinations, 23
* test_utils
  test_synthetic_data, 48
* utils
  .standardize_conc, 5
  cleanup_metadata, 19
  data_model, 21
  data_model.character, 21
  data_model.data.table, 22
  get_assays_per_pipeline_step, 29
  get_default_nested_identifiers, 29
  grr_matches, 31
  identify_keys, 34
  map_conc_to_standardized_conc, 35
  order_result_df, 40
  replace_conc_with_standardized_conc, 44
  round_concentration, 45
  .calculate_matrix_metric
  (calculate_matrix_metric), 17
  .map_references, 4
  .standardize_conc, 5
  .calculate_matrix_metric
  (calculate_matrix_metric), 17
add_CellLine_annotation, 6
add_Drug_annotation, 7
add_intermediate_data, 8
average_SE, 9