Package: MetPhenoR (via r-universe)

November 11, 2026

Title A Transparent Analysis Pipeline for Mammalian Metabolic Phenotype Microarray Data

Version 1.3

Description Provides a transparent, end-to-end workflow for analyzing high-throughput phenotype microarray data in mammalian cell cultures. It begins with automated reading and noise correction, followed by comprehensive quality control via visual diagnostics, i.e. detecting missing wells, anomalous curve shapes and variance. Area under Curve is computed as a summary metric. Biological annotations (custom or in-built for PM-M01,PM-M02, PM-M11 - PM-M14) enable group-level comparisons.

License GPL (>= 3)

Encoding UTF-8

Depends R (>= 4.5.0)

Suggests knitr, rmarkdown, testthat (>= 3.0.0)

Imports readr, dplyr, magrittr, tidyselect, ggplot2, tidyr, stringr, purrr, tidyselect, scales, stats, patchwork, readr, ggforce, rlang

Config/testthat/edition 3

LazyData true

VignetteBuilder knitr

Config/roxygen2/version 8.1.0

URL https://github.com/KatjaDanielzik/MetPhenoR

BugReports https://github.com/KatjaDanielzik/MetPhenoR/issues

NeedsCompilation no

Author Katja Danielzik [aut, cre, cph] (ORCID: <https://orcid.org/0009-0007-5021-6212>), Willson Tjioe [aut] (ORCID: <https://orcid.org/0009-0003-3530-1727>), Julia Asche [ctb], Johann Matschke [ctb] (ORCID: <https://orcid.org/0000-0003-4878-8741>), Daniel Hoffmann [ctb] (ORCID: <https://orcid.org/0000-0003-2973-7869>)

Maintainer Katja Danielzik <katja.danielzik@uni-due.de>

Repository https://cran.r-universe.dev

Date/Publication 2026-11-11 22:50:06 UTC

RemoteUrl https://github.com/cran/MetPhenoR

RemoteRef HEAD

RemoteSha e5d2bba31eb16067c7fe09fc17c05443e97c66ad

Contents

annotate_chemical . . . 2
annotate_metadata . . . 4
classify_growth_curve_types . . . 5
compute_auc_ci . . . 6
compute_auc_ratio . . . 7
filter_data . . . 8
PlateInfo . . . 9
plot_auc . . . 10
plot_auc_ratio_by_group . . . 11
plot_auc_ratio_heatmap . . . 13
plot_auc_ratio_violin . . . 14
plot_controls_tween_all_plate . . . 15
plot_controls_tween_per_plate . . . 16
plot_growth_curve . . . 17
plot_latest_timepoint . . . 19
PM_M01_M02 . . . 20
PM_M11_M14 . . . 21
processed_data . . . 22
qc_data . . . 23
RawRead . . . 24
read_folder . . . 27

Index **30**

`annotate_chemical` *Annotate assay data with chemical identities*

Description

Integrates experimental assay data with annotation files to assign chemical identities to individual wells. Annotation tables are assumed to be preloaded in the R environment.

Usage

```
annotate_chemical(
  data,
  annotation,
  control_map = list(negative = c("Negative Control", "No Chemical"), positive =
    c("a-D-Glucose", "D-(+)-Glucose"))
)
```

Arguments

`data`	A list object containing a data frame element named `data`, typically generated by `read_folder()`. The data frame must include the columns `"Plate.Type"` and `"Well"`.
`annotation`	A data frame or a list of data frames containing annotation metadata. Each data frame must include the columns `"Plate.Type"`, `"Well"`, and `"Chemical"`.
`control_map`	Optional named list specifying control classifications. Each name defines a control category, and each element is a character vector of chemical names assigned to that category. Defaults include `negative` and `positive` controls.

Details

The function performs the following operations:

1. Accepts one or multiple annotation data frames.
2. Combines annotation tables into a unified data frame and removes duplicates.
3. Merges annotation metadata with the experimental dataset based on `"Plate.Type"` and `"Well"`.
4. Optionally assigns control categories based on chemical identity using `control_map`.

Column name matching is case-sensitive; ensure consistent naming across inputs.

Value

The input `data` object with an updated `data` element, containing merged annotation columns and, if applicable, an additional `control` column indicating control classification.

Examples

```
data("processed_data")
data("PM_M01_M02")
processed_data <- annotate_chemical(processed_data, annotation = PM_M01_M02)
head(processed_data$data)
```

annotate_metadata *Annotate assay data with metadata*

Description

Integrates experimental assay data with one or more annotation tables to assign chemical or treatment identities to wells.

Usage

```
annotate_metadata(data, annotation)
```

Arguments

data	A list containing a data frame element auc_ratio, typically produced by read_folder(). The data frame must include columns "Plate.Type", "Well", and "Chemical".
annotation	A data frame or a list of data frames containing annotation metadata. Each must include columns "Plate.Type", "Well", and "Chemical".

Details

The function performs the following steps:

1. Accepts one or multiple annotation data frames.
2. Combines annotation tables and removes duplicate rows.
3. Merges annotation metadata into the experimental dataset using "Chemical" as keys.

Value

The input data object with an updated auc_ratio element that includes annotation columns.

Examples

```
data("processed_data")
# subset for fast example
processed_data$data <- processed_data$data[1:13728,]
processed_data <- annotate_chemical(processed_data, annotation = PM_M01_M02)
processed_data <- classify_growth_curve_types(processed_data)
processed_data <- compute_auc_ci(processed_data, n_boot = 1)
processed_data <- compute_auc_ratio(processed_data)
processed_data <- annotate_metadata(processed_data, annotation = PM_M01_M02)
```

`classify_growth_curve_types` *Classify Growth Curve Dynamics*

Description

Classifies growth curve dynamics from optical density (OD) time-series data using smoothing splines and their first derivatives. Curve behavior is inferred based on derivative thresholds and quality control (QC) metrics, enabling categorization into biologically interpretable growth states.

Usage

```
classify_growth_curve_types(
  data,
  spar = 0.7,
  drop_threshold = -0.01,
  tol = 0.01,
  max_od_threshold = 0.1
)
```

Arguments

`data`	A structured list containing: **data** A data frame of OD measurements in long format. **meta** A list containing metadata, including a character vector `variables` defining grouping columns (e.g., strain, condition).
`spar`	Numeric. Smoothing parameter passed to `smooth.spline`. Controls the degree of smoothing (default: `0.7`).
`drop_threshold`	Numeric. Threshold for detecting negative growth phases based on the first derivative (default: `-0.01`).
`tol`	Numeric. Tolerance around zero for classifying flat growth (default: `0.01`).
`max_od_threshold`	Numeric. Maximum OD value below which a curve is classified as low, flat signal (default: `0.1`).

Details

The classification distinguishes between actively growing, declining, flat (low signal), slow-growing, and uncertain curves. Uncertainty is assigned when data quality is insufficient or exhibits excessive noise.

For each well, a smoothing spline is fitted to OD measurements over time. The first derivative of the fitted spline is used to quantify growth dynamics:

- Positive derivatives indicate growth phases.
- Negative derivatives indicate declines in OD.

- Near-zero derivatives indicate stationary or flat behavior.

Quality control flags are applied to identify unreliable classifications:

- Low data density (fewer than four time points).
- High noise, defined as a large standard deviation of derivatives relative to the OD range.

Curve classification is assigned according to the following hierarchy:

1. `"uncertain"`: QC criteria not satisfied.
2. `"OD_drop"`: Significant negative derivative detected.
3. `"flat_low"`: Flat curve with low maximum OD.
4. `"growing"`: Positive growth phase detected.
5. `"slow_growing"`: Residual category for weak or inconsistent growth.

Value

A list identical in structure to the input, augmented with a `classification` element containing:

derivative_data A data frame including spline-fitted values and first derivatives for each observation.

well_summary A summarized data frame with per-well QC metrics and assigned curve classifications.

Examples

```
data("processed_data")
processed_data <- classify_growth_curve_types(processed_data)
head(processed_data$classification)
```

`compute_auc_ci` *Estimate Area Under the Curve (AUC) with Bootstrap Confidence Intervals*

Description

Computes the Area Under the Curve (AUC) for time-series optical density (OD) measurements on a per-well basis using spline-based regression, followed by residual bootstrap to quantify uncertainty.

Usage

```
compute_auc_ci(
  data,
  target_curve_type = "growing",
  n_boot = 100,
  spline_df = 4,
  store = TRUE
)
```

Arguments

`data`	A structured list containing: **data** A data frame of raw OD time-series measurements. **meta** Metadata including grouping variables. **classification** Well-level annotations (must include `well_summary`).
`target_curve_type`	Character vector specifying which curve types to include (default: `"growing"`).
`n_boot`	Integer. Number of bootstrap replicates (default: 1000).
`spline_df`	Integer. Degrees of freedom for natural spline fitting (default: 4).
`store`	Logical. If `TRUE`, results are stored in `data$auc_results`.

Value

A tibble or modified data object (see details).

Examples

```
data(processed_data)
# subset for fast example
processed_data$data <- processed_data$data[1:13728,]
processed_data <- annotate_chemical(processed_data, annotation = PM_M01_M02)
processed_data <- annotate_chemical(processed_data, annotation = PM_M01_M02)
processed_data <- classify_growth_curve_types(processed_data)
processed_data <- compute_auc_ci(processed_data, n_boot = 1)
head(processed_data$auc_results)
```

`compute_auc_ratio` *Compute AUC Ratio Relative to Controls*

Description

This function calculates normalized AUC ratios relative to control wells and integrates the results into the structured `data` object, consistent with `compute_auc_ci()`.

Usage

```
compute_auc_ratio(
  data,
  control_labels = c("Negative Control", "No Chemical"),
  store = TRUE
)
```

Arguments

data	A structured list containing: **auc_results** Output from compute_auc_ci(). **meta** Metadata including grouping variables.
control_labels	Character vector defining control wells (default: c("Negative Control", "No Chemical")).
store	Logical; if TRUE, results are stored in data$auc_ratio_results (default: TRUE).

Value

If store = TRUE, returns updated data object. Otherwise returns a tibble with AUC ratio calculations.

Examples

```
data(processed_data)
# subset for fast example
processed_data$data <- processed_data$data[1:13728,]
processed_data <- annotate_chemical(processed_data, annotation = PM_M01_M02)
processed_data <- classify_growth_curve_types(processed_data)
processed_data <- compute_auc_ci(processed_data, n_boot = 1)
processed_data <- compute_auc_ratio(processed_data)
head(processed_data$auc_ratio)
```

---

filter_data *Exclude 96-well plate*

Description

Excludes one 96-well plate specified by plate_type and filters from data$data.

Usage

```
filter_data(data, plate_type, filters = NULL)
```

Arguments

data	A structured list containing: **data** A data frame of raw OD time-series measurements. **meta** Metadata including grouping variables. **classification** Well-level annotations (must include well_summary).
plate_type	Character vector of Plate.Type to exclude in combination with filters
filters	Named list of filtering conditions that should be excluded in plates with 'plate_type'. E.g. list(radiation.dose="0Gy",time.h="2h"), were list elements have to be the user provided 'variables' read_folder()

Value

Filtered data$data element.

Examples

```
data("processed_data")
# before filtering
qc_data(processed_data)
processed_data <- filter_data(processed_data,
  plate_type = "PM-M01",
  filters = list(
    cell.line = "A",
    treatment = "treatment", inhibitor = "inhibitor", time = "2h"
  )
)
# after filtering
qc_data(processed_data)
```

PlateInfo *An example '..._PlateInfo.csv" file*

Description

one PlateInfo.csv file

Usage

```
data("PlateInfo")
```

Format

A data frame with 1 observations on the following 13 variables.

`PlateId` a character vector

`'Project Name'` a character vector

`'Project Code'` a character vector

`'Plate Type'` a character vector

`Sample` a character vector

`'Loaded At'` a POSIXct

`'Incubation Hours'` a numeric vector

`'Incubation Temperature Celsius'` a numeric vector

`'Incubated Offline'` a logical vector

`'Cycle Time Minutes'` a numeric vector

`Position` a character vector

`Protocol` a logical vector

`Description` a logical vector

Source

data read in ## list all files including all subdirectories data_path <- "~/Documents/Metabolomics/odin_pipeline/data/ODIN_ for Katja/Drug plates/H460/" files <- list.files(path=data_path, include.dirs = TRUE, recursive = TRUE, full.names = TRUE) # detect metadata in PlateInfo files PlateInfo <- files[stringr::str_detect(string = files, pattern = "_PlateInfo.csv")] # read in Plate Info files PlateInfo <- lapply(PlateInfo,readr::read_csv) PlateInfo <- purrr::list_rbind(PlateInfo) # detect raw RawReads <- files[stringr::str_detect(string = files, pattern ="_RawReads.csv")] RawReads <- lapply(RawReads,readr::read_csv)

RawRead <- RawReads[[1]] use_data(RawRead)

PlateInfo <- PlateInfo[PlateInfo$PlateId==RawRead$PlateId[1],] PlateInfo$Sample <- gsub(" ","_",PlateInfo$Sample)

Examples

```
data(PlateInfo)
head(PlateInfo)
```

`plot_auc` *Plot AUC Profiles by Plate and Experimental Conditions*

Description

Generates faceted visualizations of area-under-the-curve (AUC) estimates for a selected plate across experimental conditions. The function highlights wells that deviate from reference limits derived from negative controls and overlays these limits for visual comparison.

Usage

```
plot_auc(data, selected_plate, facet_vars = NULL)
```

Arguments

`data`	A list containing processed results. Must include `auc_results` (data frame) and `meta$variables`.
`selected_plate`	Character scalar specifying the plate to analyse.
`facet_vars`	Character vector (length ≤ 2) specifying variables used for faceting. Defaults to the first two variables in `data$meta$variables`.

Details

Control limits are computed within each facet as the minimum and maximum bounds of the AUC confidence intervals from negative control wells.

The function performs the following steps:

- Filters data for the selected plate.
- Splits data by non-faceted experimental variables.

- Classifies wells as negative control, positive control, or test wells.
- Computes facet-specific control limits from negative controls.
- Identifies wells outside control limits based on AUC confidence bounds.
- Orders wells globally by mean AUC for consistent visualization.
- Generates faceted `ggplot2` visualizations with:
  - AUC estimates and confidence intervals,
  - shaded control regions and boundary lines,
  - color-coded response categories.

Response categories are encoded as:

- **Red**: below control limits
- **Grey**: within control limits
- **Green**: above control limits
- **Black**: control wells

Value

Invisibly returns a named list of `ggplot` objects, each corresponding to a unique combination of non-faceted grouping variables.

Examples

```
data(processed_data)
# subset for fast example
processed_data$data <- processed_data$data[1:13728,]
processed_data <- annotate_chemical(processed_data, annotation = PM_M01_M02)
processed_data <- classify_growth_curve_types(processed_data)
processed_data <- compute_auc_ci(processed_data, n_boot = 10)
plot_auc(processed_data, selected_plate = "PM-M01")
```

`plot_auc_ratio_by_group`

Plot AUC ratio heatmap by group

Description

Generates a heatmap of AUC ratios grouped by a specified column in annotation, allowing visualization of treatment effects across groups (e.g. pathways, drug targets).

Usage

```
plot_auc_ratio_by_group(
  data,
  group_var,
  x_var,
  group_filter = NULL,
  scale_type = c("natural", "log10")
)
```

Arguments

data	A list containing a data frame element auc_ratio. The data frame must include columns "Plate.Type", "Well", "Chemical", "AUC_ratio", and the specified pathway variable.
group_var	Character string specifying the column to define groups.
x_var	Character string indicating the column to be mapped to the x-axis. Defaults to first entry in data$meta$variables.
group_filter	Optional character vector specifying group values to include. If NULL, all groups are retained.
scale_type	Character string specifying the transformation applied to AUC ratios. Must be one of "natural" or "log10".

Details

The function performs the following steps:

1. Optionally filters the dataset for selected pathways.
2. Removes control wells without chemical treatment.
3. Constructs a unique identifier per well using plate, well, and chemical.
4. Optionally applies a log10 transformation to AUC ratios.
5. Generates a heatmap using ggplot2::geom_tile().
6. Facets the plot by pathway and additional experimental variables.

Value

A ggplot object representing the AUC ratio heatmap by pathway.

Examples

```
data(processed_data)
# subset for fast example
processed_data$data <- processed_data$data[1:13728,]
processed_data <- annotate_chemical(processed_data, annotation = PM_M01_M02)
processed_data <- classify_growth_curve_types(processed_data)
processed_data <- compute_auc_ci(processed_data, n_boot = 1)
processed_data <- compute_auc_ratio(processed_data)
processed_data <- annotate_metadata(processed_data, annotation = PM_M01_M02)
```

```
plot_auc_ratio_by_group(processed_data,
  group_var = "middle_hierarchy",
  x_var = "treatment",
  group_filter = "Nucleotide metabolism"
)
```

`plot_auc_ratio_heatmap` *Plot AUC ratio heatmap*

Description

Generates a heatmap visualization of AUC ratios across wells, chemicals, and experimental conditions using the `auc_ratio` element of the input data.

Usage

```
plot_auc_ratio_heatmap(
  data,
  plate_type,
  x_var = NULL,
  scale_type = c("natural", "log10")
)
```

Arguments

`data`	A list containing a data frame element `auc_ratio`. The data frame must include columns `"Plate.Type"`, `"Well"`, `"Chemical"`, and `"AUC_ratio"`.
`plate_type`	Character vector specifying one or more plate types to include.
`x_var`	Character string indicating the column to be mapped to the x-axis. Defaults to the first entry in data$meta$variables
`scale_type`	Character string specifying the transformation applied to AUC ratios. Must be one of `"natural"` or `"log10"`.

Details

The function performs the following steps:

1. Filters the dataset for the selected plate types.
2. Constructs a unique identifier per well using plate, well, and chemical.
3. Optionally applies a log10 transformation to AUC ratios.
4. Generates a heatmap using `ggplot2::geom_tile()`.
5. Facets the plot by plate type and additional experimental variables.

Value

A ggplot object representing the AUC ratio heatmap.

Examples

```
data(processed_data)
# subset for fast example
processed_data$data <- processed_data$data[1:13728,]
processed_data <- annotate_chemical(processed_data, annotation = PM_M01_M02)
processed_data <- classify_growth_curve_types(processed_data)
processed_data <- compute_auc_ci(processed_data, n_boot = 1)
processed_data <- compute_auc_ratio(processed_data)
processed_data <- annotate_metadata(processed_data, annotation = PM_M01_M02)
plot_auc_ratio_heatmap(processed_data, plate_type = "PM-M01", x_var = "treatment")
```

---

plot_auc_ratio_violin *Plot AUC ratio distribution as violin plots*

Description

Visualizes the distribution of AUC ratios across hierarchical groupings using violin plots combined with point distributions.

Usage

```
plot_auc_ratio_violin(
  data,
  group_var,
  group_filter = NULL,
  color_var,
  scale_type = c("natural", "log10")
)
```

Arguments

data	A list containing a data frame element auc_ratio. The data frame must include columns "AUC_ratio", the specified hierarchy variable, and the specified color variable.
group_var	Character string specifying the variable used for grouping along the y-axis.
group_filter	Optional character vector specifying values of group_var to include. If NULL, all values are retained.
color_var	Character string specifying the variable used for color and fill aesthetics. Defaults to first entry in data$meta$variables
scale_type	Character string specifying the x-axis scale. Must be one of "natural" or "log10".

Details

The function performs the following steps:

1. Optionally filters the dataset based on the hierarchy variable.
2. Removes invalid AUC ratio values (NA, NaN, Inf, and non-positive values).
3. Generates violin plots with overlaid point distributions using `ggforce::geom_sina()`.
4. Applies either a natural or log10 scale to the x-axis.
5. Facets the plot by the hierarchy variable and additional variables.

Value

A `ggplot` object representing the distribution of AUC ratios.

Examples

```
#' data("processed_data")
# subset for fast example
processed_data$data <- processed_data$data[1:13728,]
processed_data <- classify_growth_curve_types(processed_data)
processed_data <- annotate_chemical(processed_data, annotation = PM_M01_M02)
processed_data <- compute_auc_ci(processed_data, n_boot = 1)
processed_data <- compute_auc_ratio(processed_data)
processed_data <- annotate_metadata(processed_data, annotation = PM_M01_M02)
plot_auc_ratio_violin(processed_data,
  group_var = "middle_hierarchy",
  group_filter = "Nucleotide metabolism",
  color_var = "treatment"
)
```

---

plot_controls_tween_all_plate

Plot Controls vs Tween Across All Plates (with Control Reference Range)

Description

Similar to plot_controls_tween_per_plate but compares across all plates. Includes negative control reference ranges per facet.

Usage

```
plot_controls_tween_all_plate(data, variables = NULL)
```

Arguments

`data`	A list
`variables`	Character vector of column names used for faceting. If NULL, uses `data$meta$variables`.

Value

A ggplot object.

Examples

```
data(processed_data)
# subset for fast example
processed_data$data <- processed_data$data[1:13728,]
processed_data <- annotate_chemical(processed_data, annotation = PM_M01_M02)
processed_data <- classify_growth_curve_types(processed_data)
processed_data <- compute_auc_ci(processed_data, n_boot = 1)
plot_controls_tween_all_plate(processed_data)
```

---

plot_controls_tween_per_plate

AUC Plot for Controls and Tween Compounds within plate

Description

This function visualizes Area Under the Curve (AUC) estimates for control wells and Tween-treated conditions within a selected plate. The plot includes confidence intervals and highlights the reference range defined by negative controls.

Usage

```
plot_controls_tween_per_plate(
  data,
  selected_plate,
  variables = NULL,
  tween_labels = c("Tween 20", "Tween 40", "Tween 80")
)
```

Arguments

`data`	A structured list
`selected_plate`	A character string specifying the `Plate.Type` to visualize.
`variables`	A character vector defining experimental grouping variables. At least two variables are recommended for meaningful faceting. Defaults to data$meta$variables.
`tween_labels`	A character vector specifying the names of Tween compounds to include in the comparison (default: `c("Tween 20", "Tween 40", "Tween 80")`).

Details

Tween compounds are displayed alongside control groups to facilitate comparative assessment of growth responses relative to baseline and positive controls.

The function performs the following steps:

1. Filters the dataset for the selected plate and retains only control wells and specified Tween compounds.
2. Classifies observations into three groups: negative controls, positive controls, and Tween-treated wells.
3. Computes a reference range based on the minimum and maximum confidence interval bounds of negative controls within each experimental condition.
4. Visualizes AUC estimates with horizontal error bars representing confidence intervals.
5. Overlays vertical reference lines indicating:
   - Lower and upper bounds of the negative control range
   - Zero baseline (AUC = 0)
6. Applies faceting based on the specified experimental variables.

This representation enables direct comparison of Tween-induced responses against empirically derived control variability.

Value

A `ggplot2` object displaying AUC estimates, confidence intervals, and control reference ranges across experimental conditions.

Examples

```
data(processed_data)
# subset for fast example
processed_data$data <- processed_data$data[1:13728,]
processed_data <- annotate_chemical(processed_data, annotation = PM_M01_M02)
processed_data <- classify_growth_curve_types(processed_data)
processed_data <- compute_auc_ci(processed_data, n_boot = 1)
plot_controls_tween_per_plate(processed_data, selected_plate = "PM-M01")
```

---

plot_growth_curve *Visualize Growth Curves and Plate Layouts from Classified Data*

Description

Generates combined visualizations of growth curve classifications and corresponding raw optical density (OD) time-series data. For each experimental condition, a plate layout heatmap is displayed alongside faceted growth curves, enabling direct comparison between spatial well distribution and temporal growth dynamics.

Usage

```
plot_growth_curve(data, curve_types_to_plot = NULL, store_plots = TRUE)
```

Arguments

`data`	A structured list produced by upstream processing steps, containing: **data** A data frame of raw OD time-series measurements. **meta** A list containing metadata, including a character vector `variables` defining grouping variables (e.g., strain, condition). **classification** A list containing outputs from `classify_growth_curve_types`, including: **well_summary** Per-well classification results. **derivative_data** Spline and derivative data (not directly plotted).
`curve_types_to_plot`	Optional character vector specifying which curve types to include in the raw growth curve visualization. If `NULL` (default), all detected curve types except `"growing"` are included.
`store_plots`	Logical. If `TRUE` (default), generated plots are stored within `data$visualization$plate_plots` and the modified object is returned. If `FALSE`, only the list of plots is returned.

Details

For each unique combination of grouping variables and plate type, the function:

1. Constructs a plate layout heatmap (8×12 format) showing well-wise growth classifications.
2. Overlays raw OD time-series curves, grouped by well and faceted by curve type.
3. Combines both visualizations using **patchwork** for side-by-side comparison.

Missing wells in the plate layout are labeled as `"empty"`. If no curves match the selected `curve_types_to_plot`, an empty panel is displayed in place of the time-series plot.

Value

If `store_plots = TRUE`, returns the input `data` object augmented with:

visualization$plate_plots A named list of combined **ggplot2** objects, one per experimental condition.

If `store_plots = FALSE`, returns a named list of **ggplot2** objects.

See Also

`classify_growth_curve_types`

Examples

```
data("processed_data")
# subset for fast example
processed_data$data <- processed_data$data[1:13728,]
processed_data <- annotate_chemical(processed_data, annotation = PM_M01_M02)
processed_data <- classify_growth_curve_types(processed_data)
processed_data <- plot_growth_curve(processed_data)
print(processed_data[["growth_curve_plots"]])
```

---

plot_latest_timepoint *Plot Final Time Point per Experimental Condition*

Description

This function extracts the final (latest) time point for each well across experimental conditions and visualizes the corresponding measurements in a 96-well plate layout.

Usage

```
plot_latest_timepoint(data)
```

Arguments

data	A named list produced by `read_folder()`, containing: **data** A `data.frame` with experimental measurements **meta** A list containing metadata, including a character vector `variables` specifying grouping variables

Details

Each plot represents a single experimental condition and displays wells as spatially arranged points, colored according to their measured OD at the final time point.

The function performs the following steps:

1. Groups the data by experimental variables, Plate.Type, and Well.
2. Extracts the latest time point per well.
3. Converts well identifiers (e.g., "A01") into plate coordinates.
4. Generates a separate plate-layout plot for each experimental condition.

Color intensity reflects the magnitude of `diff_OD_590_740` at the final recorded time point.

Value

A named list of `ggplot2` objects, where each element corresponds to a unique experimental condition.

Examples

```
data("processed_data")
plot_latest_timepoint(processed_data)
```

PM_M01_M02 *Annotations for metabolic plates PM-M01 and PM-M02*

Description

Mapping metabolic plate wells to corresponding Chemical and KEGG annotations

Usage

```
data("PM_M01_M02")
```

Format

A data frame with 201 observations on the following 11 variables.

'Plate.Type' Odin Plate Type

Well Well ID on plate

Chemical Chemical plated in 'Well' in 'PlateType'

'Kegg Link' Link to KEGG database

Info e.g. 'Carbon Source'

KEGG KEGG ID of 'Chemical'

module_id KEGG module ID, can be multiple per 'KEGG'

module_name KEGG module name

upper_hierarchy upper hierarchy annotation of 'module_id'

middle_hierarchy middle hierarchy annotation of 'module_id

lower_hierarchy lower hierarchy annotation of 'lower_hierarchy

Details

source: PM-M[01] Annotation Exported (from Odin Biolog: PlateId + Well + Chemical), data/annotation/KEGG_backgroun
script: scripts/KEGGIDs_from_exported_Odin_annotation some KEGG IDs (comment) were wrong
and one could be added by hand includes pathway and functional module annotation from KEGG
(potentially multiple pathway annotations!)

Katja Danielzik hand curated the a-D-Glucose KEGG ID to C00267 (as also Julia Asche annotated
it) inside the script

Examples

```
data(PM_M01_M02)
head(data)
```

PM_M11_M14 *Annotations for anti-cancer drug plates PM-M11 to PM-M14*

Description

mapping of anti-cancer-drug plate wells to corresponding Chemical, KEGG ID, Inhibition Target, ...

Usage

```
data("PM_M11_M14")
```

Format

A data frame with 384 observations on the following 11 variables.

'Plate.Type' Odin Plate Type

Well Well ID on plate

Chemical Chemical plated in 'Well' in 'PlateType'

Info e.g. "antineoplastic, antiinflammatory"

KEGG KEGG ID of 'Chemical'

CAS CAS ID of 'Chemical'

'Inhibition Target' Inhibition Target of 'Chemical'

'Mechanism of Action' Mechanism of Action of 'Chemical'

'Affected Pathway' Affected Pathway of 'Chemical'

'Used to Treat' 'Chemical' is used to treat ...

'Clinical Status (March 2025)' Clinical Status of 'Chemical' in March 2025

Details

source: data/annotations PMM11.csv - PMM14.csv provided by Odin Biolog, data/annotations 20250312_SEA_ODIN_PMM11-14 Pathways.xlsx provided by Johann Matschke

Examples

```
data("PM_M11_M14")
head(PM_M11_M14)
```

processed_data *Example data set as result of read_folder.R with sample_column = "Sample" and variables = c("cell.line", "LKB1.status", "radiation.dose")*

Description

results of read_folder() function processed_data <- read_folder("your data path",variables = c("cell.line","radiation.dose","inh

Usage

```
data("processed_data")
```

Format

a list

- `data`: a data frame
  - `PlateId`: a character vector
  - 'Project Name': a character vector
  - 'Project Code': a character vector
  - 'Plate Type': a character vector
  - `Sample`: a character vector
  - `cell.line`: a character vector
  - `LKB1.status`: a character vector
  - `radiation.dose`: a character vector
  - 'Loaded At': a POSIXct
  - 'Incubation Hours': a numeric vector
  - 'Incubation Temperature Celsius': a numeric vector
  - 'Incubated Offline': a logical vector
  - 'Cycle Time Minutes': a numeric vector
  - `Position`: a character vector
  - `Protocol`: a logical vector
  - `Description`: a logical vector
  - `Id`: a character vector
  - 'Actual Temperature Celsius': a numeric vector
  - 'Target Temperature Celsius': a numeric vector
  - 'Read At': a POSIXct
  - `Hour`: a numeric vector
  - `Well`: a character vector
  - `diff_OD_590_740`: a numeric vector
- `meta`: A list
  - `data_path`: user set 'data_path'
  - `sample_column`: user set 'sample_column'
  - `variables`: user set 'variables'

Examples

```
data("processed_data")
head(processed_data)
```

`qc_data` *Quality Control for input data after noise correction*

Description

The function filters the first five wells,evaluates replicate consistency across experimental conditions, detects duplicated entries, and visualizes replicate counts using a heatmap.

Usage

```
qc_data(data)
```

Arguments

`data`	A named list produced by `read_folder()`, containing:
	data A `data.frame` with experimental measurements
	meta A list containing metadata, including a character vector `variables` a character vector specifying grouping variables

Details

The QC workflow comprises:

1. Subsetting the dataset to first five wells
2. Identifying duplicated measurements across grouping variables
3. Summarising replicate counts per condition
4. Visualizing replicate distribution as a heatmap

The function internally restricts analysis to a fixed set of wells (`A01-A05`). Replicate counts are computed across grouping variables defined by `data$meta$variables`, in addition to `"Plate.Type"`, `"Hour"`, and `"Well"`.

Duplicate entries are defined as observations sharing identical values across all grouping variables and occurring more than once.

The resulting heatmap displays the number of replicate measurements per condition, facilitating rapid identification of inconsistencies such as duplicated or missing replicates.

Value

A `ggplot2` object representing a heatmap of replicate counts across wells and experimental conditions.

See Also

read_folder

Examples

```
data("processed_data")
qc_data(processed_data)
```

RawRead *Example Raw Read from Odin Biolog*

Description

one RawReads.csv file

Usage

```
data("RawRead")
```

Format

A data frame with 288 observations on the following 102 variables.

`Id` a character vector

`PlateId` a character vector

`Wavelength` a numeric vector

'`Actual Temperature Celsius`' a numeric vector

'`Target Temperature Celsius`' a numeric vector

'`Read At`' a POSIXct

`A01` a numeric vector

`A02` a numeric vector

`A03` a numeric vector

`A04` a numeric vector

`A05` a numeric vector

`A06` a numeric vector

`A07` a numeric vector

`A08` a numeric vector

`A09` a numeric vector

`A10` a numeric vector

`A11` a numeric vector

`A12` a numeric vector

B01 a numeric vector
B02 a numeric vector
B03 a numeric vector
B04 a numeric vector
B05 a numeric vector
B06 a numeric vector
B07 a numeric vector
B08 a numeric vector
B09 a numeric vector
B10 a numeric vector
B11 a numeric vector
B12 a numeric vector
C01 a numeric vector
C02 a numeric vector
C03 a numeric vector
C04 a numeric vector
C05 a numeric vector
C06 a numeric vector
C07 a numeric vector
C08 a numeric vector
C09 a numeric vector
C10 a numeric vector
C11 a numeric vector
C12 a numeric vector
D01 a numeric vector
D02 a numeric vector
D03 a numeric vector
D04 a numeric vector
D05 a numeric vector
D06 a numeric vector
D07 a numeric vector
D08 a numeric vector
D09 a numeric vector
D10 a numeric vector
D11 a numeric vector
D12 a numeric vector
E01 a numeric vector

E02 a numeric vector
E03 a numeric vector
E04 a numeric vector
E05 a numeric vector
E06 a numeric vector
E07 a numeric vector
E08 a numeric vector
E09 a numeric vector
E10 a numeric vector
E11 a numeric vector
E12 a numeric vector
F01 a numeric vector
F02 a numeric vector
F03 a numeric vector
F04 a numeric vector
F05 a numeric vector
F06 a numeric vector
F07 a numeric vector
F08 a numeric vector
F09 a numeric vector
F10 a numeric vector
F11 a numeric vector
F12 a numeric vector
G01 a numeric vector
G02 a numeric vector
G03 a numeric vector
G04 a numeric vector
G05 a numeric vector
G06 a numeric vector
G07 a numeric vector
G08 a numeric vector
G09 a numeric vector
G10 a numeric vector
G11 a numeric vector
G12 a numeric vector
H01 a numeric vector
H02 a numeric vector

H03 a numeric vector

H04 a numeric vector

H05 a numeric vector

H06 a numeric vector

H07 a numeric vector

H08 a numeric vector

H09 a numeric vector

H10 a numeric vector

H11 a numeric vector

H12 a numeric vector

Source

data read in ## list all files including all subdirectories data_path <- "~/Documents/Metabolomics/odin_pipeline/data/ODIN_ for Katja/Drug plates/H460/" files <- list.files(path=data_path, include.dirs = TRUE, recursive = TRUE, full.names = TRUE) # detect metadata in PlateInfo files PlateInfo <- files[stringr::str_detect(string = files, pattern = "_PlateInfo.csv")] # read in Plate Info files PlateInfo <- lapply(PlateInfo,readr::read_csv) PlateInfo <- purrr::list_rbind(PlateInfo) # detect raw RawReads <- files[stringr::str_detect(string = files, pattern ="_RawReads.csv")] RawReads <- lapply(RawReads,readr::read_csv)

RawRead <- RawReads[[1]] use_data(RawRead)

Examples

```
data(RawRead)
head(RawRead)
```

---

read_folder *Process CSV Plate Data stored in folder into data frame in long format*

Description

Reads all 'PlateInfo' and 'RawRead' files from a directory (can have subdirectories), merges the 'PlateInfo' files and annotates the RawRead files with the 'Description' column of 'PlateInfo'. Proceeds by aligning the time points of OD values measured at 590 nm and 740 nm, calculates '590 nm' - '740 nm' for noise correction as advised by Biolog.

Usage

```
read_folder(data_path, sample_column, variables)
```

Arguments

data_path	Character. Path to directory containing CSV files. At least one file, which name ends on "PlateInfo.csv" containing the metadata with at least a column 'PlateId' and the 'sample_column' containing the description of the experiment. and at least one file containing the raw Odin reads, which name end on "RawReads.csv" and have at least one column 'PlateId' and columns 'A01' to 'H12' representing the wells of a 96-well plate.
sample_column	Character. Which column in '_PlateInfo.csv' contains the 'Description' (i.e. the Character vector that should be split according to 'variables')
variables	Character vector. Names of variables to extract from the 'Description' column (e.g., c("cell.line", "radiation.dose", "inhibitor")). Have to be separated by "_" and always in the same order for all "_PlateInfo.csv" files (e.g. in format cell.line, radiation.dose, time.point -> "A549_0Gy_0h")

Value

a list

result: A data frame in long format

- PlateId: Odin plate Id from 'PlateInfo', refers to a single 96-well plate
- Project Name: Odin project name from 'PlateInfo'
- Project Code: Odin project code from 'PlateInfo'
- Plate Type: Odin plate type from 'PlateInfo'
- Sample: Sample specification from 'PlateInfo'
- Loaded At: Loading time from 'PlateInfo'
- Incubation Hours: Incubation time from 'PlateInfo'
- Incubation Temperature Celsius: Incubation temperature from 'PlateInfo'
- Incubated Offline:
- Cycle Time Minutes: Time interval of measurements from 'PlateInfo'
- Position: Position of Plate in Odin from 'PlateInfo'
- Protocol: user specified protocol from 'PlateInfo'
- Description: user specified description of the experiment (or in 'Sample')
- ...: single variable columns from 'sample_column'
- Actual Temperature Celsius: measured temperature in Odin from 'RawRead'
- Target Temperature Celsius: Target temperature (should be equal to Incubation Temperature Celsius from 'PlateInfo')
- Read At: Time of Plate read from 'RawReads'
- Hour: difference between Read At and first entry in Read At per plate
- Well: Well Id of the 96-well plate
- diff_OD_590_740: OD_590nm-OD_740nm, noise correction for mammalian plates according to Biolog
- meta: A list
  - data_path: user set 'data_path'
  - sample_column: user set 'sample_column'
  - variables: user set 'variables'

Examples

```
# Raw Reads are in this format
data("RawRead")
head(RawRead)

# PlateInfo is in this format
data("PlateInfo")
head(PlateInfo)

# resulting data frame 'data' from read_folder
data("processed_data")
head(processed_data$data)
```

Index

∗ **datasets**
PlateInfo, 9
PM_M01_M02, 20
PM_M11_M14, 21
processed_data, 22
RawRead, 24

annotate_chemical, 2
annotate_metadata, 4

classify_growth_curve_types, 5, *18*
compute_auc_ci, 6
compute_auc_ratio, 7

filter_data, 8

PlateInfo, 9
plot_auc, 10
plot_auc_ratio_by_group, 11
plot_auc_ratio_heatmap, 13
plot_auc_ratio_violin, 14
plot_controls_tween_all_plate, 15
plot_controls_tween_per_plate, 16
plot_growth_curve, 17
plot_latest_timepoint, 19
PM_M01_M02, 20
PM_M11_M14, 21
processed_data, 22

qc_data, 23

RawRead, 24
read_folder, *24*, 27
read_folder(), *8*

smooth.spline, *5*