Package 'SerolyzeR'

May 6, 2025

Type Package

Title Reading, Quality Control and Preprocessing of MBA (Multiplex
Bead Assay) Data

Description Speeds up the process of loading raw data from MBA (Multiplex Bead Assay) examinations, performs quality control checks, and automatically normalises the data, preparing it for more advanced, downstream tasks. The main objective of the package is to create a simple environment for a user, who does not necessarily have experience with R language. The package is developed within the project 'PvSTATEM', which is an international project aiming for malaria elimination.

BugReports <https://github.com/mini-pw/SerolyzeR/issues>

Version 1.2.0

License BSD_3_clause + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Imports dplyr, ggplot2, nplr, R6, readxl, stringi, stringr, grid, png,
tools, ggrepel, lubridate, R.utils, svglite, fs, scales

Suggests knitr, qpdf, rmarkdown, testthat (>= 3.0.0)

Config/testfhat/edition 3

VignetteBuilder knitr

URL <https://github.com/mini-pw/SerolyzeR>,
<https://mini-pw.github.io/SerolyzeR/>

NeedsCompilation no

Author Tymoteusz Kwiecinski [aut, cre] (ORCID:
<https://orcid.org/0009-0006-7362-9821>),
Jakub Grzywaczewski [aut],
Mateusz Nizwantowski [aut],
Przemyslaw Biecek [ths] (ORCID:
<https://orcid.org/0000-0001-8423-1823>),
Nuno Sepulveda [ths] (ORCID: <https://orcid.org/0000-0002-8542-1706>)

Maintainer Tymoteusz Kwiecinski <tymoteuszkwiecinski@gmail.com>

Repository CRAN

Date/Publication 2025-05-06 08:20:02 UTC

Contents

create_standard_curve_model_analyte . . . . [2](#)
generate_levey_jennings_report . . . . [3](#)
generate_plate_report . . . . [5](#)
get_nmfi . . . . [6](#)
handle_high_dose_hook . . . . [7](#)
is_valid_data_type . . . . [9](#)
is_valid_sample_type . . . . [9](#)
Model . . . . [10](#)
Plate . . . . [13](#)
PlateBuilder . . . . [18](#)
plot_counts . . . . [21](#)
plot_layout . . . . [22](#)
plot_levey_jennings . . . . [23](#)
plot_mfi_for_analyte . . . . [24](#)
plot_standard_curve_analyte . . . . [25](#)
plot_standard_curve_analyte_with_model . . . . [26](#)
plot_standard_curve_stacked . . . . [28](#)
predict.Model . . . . [29](#)
process_dir . . . . [30](#)
process_file . . . . [33](#)
process_plate . . . . [34](#)
read_intelliflex_format . . . . [37](#)
read_layout_data . . . . [38](#)
read_luminex_data . . . . [38](#)
read_xponent_format . . . . [40](#)
translate_sample_names_to_sample_types . . . . [41](#)

---

create_standard_curve_model_analyte

Create a standard curve model for a certain analyte

Description

Create a standard curve model for a certain analyte

Usage

```
create_standard_curve_model_analyte(
  plate,
  analyte_name,
  data_type = "Median",
  source_mfi_range_from_all_analytes = FALSE,
  detect_high_dose_hook = TRUE,
  ...
)
```

Arguments

`plate`	(`Plate()`) Object of the Plate class
`analyte_name`	(`character(1)`) Name of the analyte for which we want to create the model
`data_type`	(`character(1)`) Data type of the value we want to use to fit the model - the same datatype as in the plate file. By default, it equals to `Median`
`source_mfi_range_from_all_analytes`	
	(`logical(1)`) If `TRUE`, the MFI range is calculated from all analytes; if `FALSE`, the MFI range is calculated only for the current analyte Defaults to `FALSE`
`detect_high_dose_hook`	
	(`logical(1)`) If `TRUE`, the high dose hook effect is detected and handled. For more information, please see the handle_high_dose_hook function documentation.
`...`	Additional arguments passed to the model
	Standard curve samples should not contain `na` values in mfi values nor in dilutions.

Value

(`Model()`) Standard Curve model

---

generate_levey_jennings_report

Generate a Levey-Jennings Report for Multiple Plates.

Description

This function generates a Levey-Jennings report for a list of plates. The report includes layout plot, levey jennings plot, for each analyte and selected dilutions. The report also includes stacked standard curves plot in both monochromatic and color versions for each analyte. The report is generated using the `levey_jennings_report_template.Rmd` template.

Usage

```
generate_levey_jennings_report(
  list_of_plates,
  report_title,
  dilutions = c("1/100", "1/400"),
  filename = NULL,
  output_dir = "reports",
  additional_notes = NULL
)
```

Arguments

list_of_plates A list of plate objects.

report_title   (character(1)) The title of the report.

dilutions   (character) A character vector specifying the dilutions to be included in the
                report. Default is c("1/100", "1/400").

filename   (character(1)) The name of the output HTML report file. If not provided
                or set to NULL, the filename will be based on the first plate name, formatted
                as {plate_name}_levey_jennings.html. If the filename does not contain the
                .html extension, it will be automatically added. Absolute file paths in filename
                will override output_dir. Existing files at the specified path will be overwrit-
                ten.

output_dir   (character(1)) The directory where the report will be saved. Defaults to 're-
                ports'. If NULL, the current working directory will be used. Necessary directories
                will be created if they do not exist.

additional_notes
                (character(1)) Additional notes to be included in the report. Markdown for-
                matting is supported. If not provided, the section will be omitted.

Value

A Levey-Jennings report in HTML format.

Examples

```
output_dir <- tempdir(check = TRUE)

dir_with_luminex_files <- system.file("extdata", "multiplate_lite",
  package = "SerolyzeR", mustWork = TRUE
)
list_of_plates <- process_dir(dir_with_luminex_files,
  return_plates = TRUE, format = "xPONENT", output_dir = output_dir
)
note <- "This is a Levey-Jennings report.\n**Author**: Jane Doe \n**Tester**: John Doe"

generate_levey_jennings_report(
  list_of_plates = list_of_plates,
  report_title = "QC Report",
  dilutions = c("1/100", "1/200"),
  output_dir = tempdir(),
  additional_notes = note
)
```

---

generate_plate_report   *Generate a report for a plate.*

Description

This function generates a report for a plate. The report contains all the necessary information about the plate, from the general plate parameters, such as examination date, to the breakdown of the analytes' plots. The report is generated using the `plate_report_template.Rmd` template.

Usage

```
generate_plate_report(
  plate,
  use_model = TRUE,
  filename = NULL,
  output_dir = "reports",
  counts_lower_threshold = 50,
  counts_higher_threshold = 70,
  additional_notes = NULL
)
```

Arguments

`plate`	A plate object.
`use_model`	(`logical(1)`) A logical value indicating whether the model should be used in the report.
`filename`	(`character(1)`) The name of the output HTML report file. If not provided or equals to `NULL`, the output filename will be based on the plate name, precisely: `{plate_name}_report.html`. By default the `plate_name` is the filename of the input file that contains the plate data. For more details please refer to Plate.
	If the passed filename does not contain `.html` extension, the default extension `.html` will be added. Filename can also be a path to a file, e.g. `path/to/file.html`. In this case, the `output_dir` and `filename` will be joined together. However, if the passed filepath is an absolute path and the `output_dir` parameter is also provided, the `output_dir` parameter will be ignored. If a file already exists under a specified filepath, the function will overwrite it.
`output_dir`	(`character(1)`) The directory where the output CSV file should be saved. Please note that any directory path provided will create all necessary directories (including parent directories) if they do not exist. If it equals to `NULL` the current working directory will be used. Default is 'reports'.
`counts_lower_threshold`	
	(`numeric(1)`) The lower threshold for the counts plots (works for each analyte). Default is 50.
`counts_higher_threshold`	
	(`numeric(1)`) The higher threshold for the counts plots (works for each analyte). Default is 70.

additional_notes

        (character(1)) Additional notes to be included in the report. Contents of this fields are left to the user's discretion. If not provided, the field will not be included in the report.

Value

A report.

Examples

```
plate_file <- system.file("extdata", "CovidOISExPONENT_CO_reduced.csv", package = "SerolyzeR")
# a plate file with reduced number of analytes to speed up the computation
layout_file <- system.file("extdata", "CovidOISExPONENT_CO_layout.xlsx", package = "SerolyzeR")
note <- "This is a test report.\n**Author**: Jane Doe \n**Tester**: John Doe"

plate <- read_luminex_data(plate_file, layout_file, verbose = FALSE)
example_dir <- tempdir(check = TRUE) # a temporary directory
generate_plate_report(plate,
  output_dir = example_dir,
  counts_lower_threshold = 40,
  counts_higher_threshold = 50,
  additional_notes = note
)
```

---

get_nmfi   *Calculate normalised MFI values for a plate*

Description

The function calculates the normalised MFI (nMFI) values for each of the analytes in the plate.

The nMFI values are calculated as the ratio of the test samples' MFI values to the standard curve samples with the target dilution.

When nMFI could be used? In general, it is preferred to use Relative Antibody Unit (RAU) values for any analysis. However, it is sometimes impossible to fit a model to the standard curve samples. This may happen if the MFI values of test samples are much higher than the MFI of standard curve samples. Then, the prediction would require significant data extrapolation, which could lead to unreliable results.

In such cases, the nMFI values could be used as a proxy for RAU values if we want, for instance, to account for plate-to-plate variation.

Usage

```
get_nmfi(
  plate,
  reference_dilution = 1/400,
  data_type = "Median",
  verbose = TRUE
)
```

Arguments

`plate`	(`Plate()`) a plate object for which to calculate the nMFI values
`reference_dilution`	
	(`numeric(1)` or `character(1)`) the dilution value of the standard curve sample to use as a reference for normalisation. The default is `1/400`. It should refer to a dilution of a standard curve sample in the given plate object. This parameter could be either a numeric value or a string. In case it is a character string, it should have format `1/d+`, where d+ is any positive integer.
`data_type`	(`character(1)`) type of data for the computation. Median is the default
`verbose`	(`logical(1)`) print additional information. The default is TRUE

Value

nmfi (`data.frame`) a data frame with normalised MFI values for each analyte in the plate and all test samples.

Examples

```
# read the plate
plate_file <- system.file("extdata", "CovidOISExPONTENT.csv", package = "SerolyzeR")
layout_file <- system.file("extdata", "CovidOISExPONTENT_layout.csv", package = "SerolyzeR")

plate <- read_luminex_data(plate_file, layout_file)

# artificially bump up the MFI values of the test samples (the Median data type is default one)
plate$data[["Median"]][plate$sample_types == "TEST", ] <-
  plate$data[["Median"]][plate$sample_types == "TEST", ] * 10

# calculate the nMFI values
nmfi <- get_nmfi(plate, reference_dilution = 1 / 400)

# we don't do any extrapolation and the values should be comparable across plates
head(nmfi)
# different params
nmfi <- get_nmfi(plate, reference_dilution = "1/50")
```

---

handle_high_dose_hook *Detect and handle the high dose hook effect*

Description

Typically, the MFI values associated with standard curve samples should decrease as we dilute the samples. However, sometimes in high dilutions, the MFI presents a non monotonic behavior. In that case, MFI values associated with dilutions above (or equal to) `high_dose_threshold` should be removed from the analysis.

For more information about this effect please refer to: Namburi, R. P. et. al. (2014) High-dose hook effect.

For the `nplr` model the recommended number of standard curve samples is at least 4. If the high dose hook effect is detected but the number of samples below the `high_dose_threshold` is lower than 4, additional warning is printed and the samples are not removed.

Warning: High dose hook effect affects which dilution and MFI values are used to fit the logistic model and by extension affects the maximum value to which the predicted RAU MFI values are extrapolated / truncated.

The function returns a logical vector that can be used to subset the MFI values.

Usage

```
handle_high_dose_hook(mfi, dilutions, high_dose_threshold = 1/200)
```

Arguments

`mfi`   (numeric())

`dilutions`   (numeric())

`high_dose_threshold`

           (numeric(1)) MFI values associated with dilutions above this threshold should be checked for the high dose hook effect

Value

sample selector (`logical()`)

Examples

```
plate_filepath <- system.file(
  "extdata", "CovidOISExPONTENT.csv",
  package = "SerolyzeR", mustWork = TRUE
) # get the filepath of the csv dataset
layout_filepath <- system.file(
  "extdata", "CovidOISExPONTENT_layout.xlsx",
  package = "SerolyzeR", mustWork = TRUE
)
plate <- read_luminex_data(plate_filepath, layout_filepath) # read the data

# here we plot the data with observed high dose hook effect
plot_standard_curve_analyte(plate, "RBD_omicron")

# here we create the model with the high dose hook effect handled
model <- create_standard_curve_model_analyte(plate, "RBD_omicron")
```

---

is_valid_data_type *Check validity of given data type*

Description

Check if the data type is valid. The data type is valid if it is one of the elements of the VALID_DATA_TYPES vector. The valid data types are:
c(Median, Net MFI, Count, Avg Net MFI, Mean, Peak).

Usage

```
is_valid_data_type(data_type)
```

Arguments

data_type   A string representing the data type.

Value

TRUE if the data type is valid, FALSE otherwise.

---

is_valid_sample_type *Check validity of given sample type*

Description

Check if the sample type is valid. The sample type is valid if it is one of the elements of the VALID_SAMPLE_TYPES vector. The valid sample types are:

c(ALL, BLANK, TEST, NEGATIVE CONTROL, STANDARD CURVE, POSITIVE CONTROL).

Usage

```
is_valid_sample_type(sample_type)
```

Arguments

sample_type   A string representing the sample type.

Value

TRUE if the sample type is valid, FALSE otherwise.

Model   *Logistic regression model for the standard curve*

Description

The Model class is a wrapper around the `nplr` model. It allows to predict the RAU (Relative Antibody Unit) values directly from the MFI values of a given sample.

The `nplr` model is fitted using the formula:

$$y = B + \frac{T - B}{(1 + 10^{b \cdot (x_{mid} - x)})^s},$$

where:

- y is the predicted value, MFI in our case,
- x is the independent variable, dilution in our case,
- B is the bottom plateau - the right horizontal asymptote,
- T is the top plateau - the left horizontal asymptote,
- b is the slope of the curve at the inflection point,
- x_{mid} is the x-coordinate at the inflection point,
- s is the asymmetric coefficient.

This equation is referred to as the Richards' equation. More information about the model can be found in the `nplr` package documentation.

After the model is fitted to the data, the RAU values can be predicted using the `predict` method. The RAU value is simply a predicted dilution value (using the standard curve) for a given MFI multiplied by 1,000 000 to have a more readable value. For more information about the differences between dilution, RAU and MFI values, please see the "Normalisation" section in the "Basic SerolyzeR functionalities" vignette.

Public fields

analyte (`character(1)`)
    Name of the analyte for which the model was fitted

dilutions (`numeric()`)
    Dilutions used to fit the model

mfi (`numeric()`)
    MFI values used to fit the model

mfi_min (`numeric(1)`)
    Minimum MFI used for scaling MFI values to the range [0, 1]

mfi_max (`numeric(1)`)
    Maximum MFI used for scaling MFI values to the range [0, 1]

model (`nplr`)
    Instance of the `nplr` model fitted to the data

```
log_dilution (logical())
```
    Indicator should the dilutions be transformed using the `log10` function

```
log_mfi (logical())
```
    Indicator should the MFI values be transformed using the `log10` function

```
scale_mfi (logical())
```
    Indicator should the MFI values be scaled to the range [0, 1]

Active bindings

```
top_asymptote (numeric(1))
```
    The top asymptote of the logistic curve

```
bottom_asymptote (numeric(1))
```
    The bottom asymptote of the logistic curve

Methods

Public methods:

- `Model$new()`
- `Model$predict()`
- `Model$get_plot_data()`
- `Model$print()`
- `Model$clone()`

Method new(): Create a new instance of Model R6 class

Usage:
```
Model$new(
  analyte,
  dilutions,
  mfi,
  npars = 5,
  verbose = TRUE,
  log_dilution = TRUE,
  log_mfi = TRUE,
  scale_mfi = TRUE,
  mfi_min = NULL,
  mfi_max = NULL
)
```

Arguments:

```
analyte (character(1))
```
    Name of the analyte for which the model was fitted.

```
dilutions (numeric())
```
    Dilutions used to fit the model

```
mfi  MFI (numeric())
```
    values used to fit the model

```
npars (numeric(1))
```
    Number of parameters to use in the model

```
verbose (logical())
```
    If TRUE prints messages, TRUE by default
```
log_dilution (logical())
```
    If TRUE the dilutions are transformed using the log10 function, TRUE by default
```
log_mfi (logical())
```
    If TRUE the MFI values are transformed using the log10 function, TRUE by default
```
scale_mfi (logical())
```
    If TRUE the MFI values are scaled to the range [0, 1], TRUE by default
```
mfi_min (numeric(1))
```
    Enables to set the minimum MFI value used for scaling MFI values to the range [0, 1]. Use
    values before any transformations (e.g., before the log10 transformation)
```
mfi_max (numeric(1))
```
    Enables to set the maximum MFI value used for scaling MFI values to the range [0, 1]. Use
    values before any transformations (e.g., before the log10 transformation)

Method `predict()`: Predict RAU values from the MFI values

Usage:
```
Model$predict(mfi, over_max_extrapolation = 0, eps = 1e-06)
```

Arguments:
```
mfi (numeric())
```
    MFI values for which we want to predict the RAU values
```
over_max_extrapolation (numeric(1))
```
    How much we can extrapolate the values above the maximum RAU value seen in standard curve samples RAU_{max}. Defaults to 0. If the value of the predicted RAU is above $RAU_{max} + \text{over\_max\_extrapolation}$, the value is censored to the value of that sum.
```
eps (numeric(1))
```
    A small value used to avoid numerical issues close to the asymptotes
    Warning: High dose hook effect affects which dilution and MFI values are used to fit the logistic model and by extension affects the over_max_extrapolation value. When a high dose hook effect is detected we remove the samples with dilutions above the high dose threshold (which by default is 1/200). Making the highest RAU value lower than the one calculated without handling of the high dose hook effect.

Returns: `(data.frame())`
Dataframe with the predicted RAU values for given MFI values The columns are named as follows:

  - `RAU` - the Relative Antibody Units (RAU) value
  - `MFI` - the predicted MFI value

Method `get_plot_data()`: Data that can be used to plot the standard curve.

Usage:
```
Model$get_plot_data()
```

Returns: `(data.frame())`
Prediction dataframe for scaled MFI (or logMFI) values in the range [0, 1]. Columns are named as in the `predict` method

Plate 13

Method `print()`: Function prints the basic information about the model such as the number of parameters or samples used

Usage:
`Model$print()`

Method `clone()`: The objects of this class are cloneable with this method.

Usage:
`Model$clone(deep = FALSE)`

Arguments:

deep Whether to make a deep clone.

Examples

```
plate_file <- system.file("extdata", "CovidOISExPONENT.csv", package = "SerolyzeR")
layout_file <- system.file("extdata", "CovidOISExPONENT_layout.csv", package = "SerolyzeR")
plate <- read_luminex_data(plate_file, layout_filepath = layout_file)
model <- create_standard_curve_model_analyte(plate, "S2", log_mfi = TRUE)
print(model)
```

Plate   *Plate Object*

Description

The `Plate` object represents a Luminex assay plate and stores data related to its samples, analytes, metadata, and batch information. This object is typically created by functions such as read_luminex_data(), process_file(), or process_dir().

It provides methods for accessing and manipulating data, including retrieving specific analyte measurements, filtering by sample type, and performing blank adjustments.

Public fields

plate_name (character(1))
     Name of the plate. Set to the name of the file from which the plate was read.

analyte_names (character())
     Names of the analytes that were examined on the plate.

sample_names (character())
     Names of the samples that were examined on the plate. The order of the samples in this vector is identical with order in the CSV source file.

batch_name (character(1))
     Name of the batch to which the plate belongs.

plate_datetime (POSIXct())
     A date and time when the plate was created by the machine

sample_locations (character())
    Locations of the samples on the plate. This vector is in the same order as the sample_names vector.

sample_types (character())
    Types of the samples that were examined on the plate. The possible values are c(ALL, BLANK, TEST, NEGATIVE CONTROL, STANDARD CURVE, POSITIVE CONTROL). This vector is in the same order as the sample_names vector. If the [Plate](#) object is read using [read_luminex_data](#), then the sample types are usually detected based on the sample names according to the rules described in [translate_sample_names_to_sample_types](#).

dilutions (character())
    A list containing names of the samples as keys and string representing dilutions as values. The dilutions are represented as strings. This vector is in the same order as the sample_names vector.

dilution_values (numeric())
    A list containing names of the samples as keys and numeric values representing dilutions as values. It is in the same order as the sample_names vector.

default_data_type (character(1))
    The default data type that will be returned by the get_data method. By default is set to Median.

data (list())
    A list containing dataframes with the data for each sample and analyte. The possible data types - the keys of the list are:
    c(Median, Net MFI, Count, Avg Net MFI, Mean, Peak).

    In each dataframe, the rows represent samples and the columns represent analytes.

batch_info (list())
    A list containing additional, technical information about the batch.

layout (character())
    A list containing information about the layout of the plate. The layout is read from the separate file and usually provides additional information about the dilutions, sample names, and the sample layout on the actual plate.

blank_adjusted (logical)
    A flag indicating whether the blank values have been adjusted.

Methods

Public methods:

- [Plate$new()](#)
- [Plate$print()](#)
- [Plate$summary()](#)
- [Plate$get_data()](#)
- [Plate$get_dilution()](#)
- [Plate$get_dilution_values()](#)
- [Plate$blank_adjustment()](#)
- [Plate$clone()](#)

Plate 15

Method new(): Method to initialize the Plate object

Usage:
```
Plate$new(
  plate_name,
  sample_names,
  analyte_names,
  batch_name = "",
  plate_datetime = NULL,
  sample_locations = NULL,
  sample_types = NULL,
  dilutions = NULL,
  dilution_values = NULL,
  default_data_type = NULL,
  data = NULL,
  batch_info = NULL,
  layout = NULL
)
```

Arguments:

plate_name (character(1))

Name of the plate. By default is set to an empty string, during the reading process it is set to the name of the file from which the plate was read.

sample_names (character())

Names of the samples that were examined on the plate. Sample names are by default ordered by location in the plate, using the row-major order. The first sample is the one in upper-left corner, then follows the ones in the first row, then the second row, and so on.

analyte_names (character())

Names of the analytes that were examined on the plate.

batch_name (character(1))

Name of the batch to which the plate belongs. By default is set to an empty string, during the reading process it is set to the batch field of the plate

plate_datetime (POSIXct())

Datetime object representing the date and time when the plate was created by the machine.

sample_locations (character())

Locations of the samples on the plate. Sample locations are ordered in the same way as samples in the input CSV file.

sample_types (character())

Types of the samples that were examined on the plate. The possible values are c(ALL, BLANK, TEST, NEGATIVE CONTROL, STANDARD CURVE, POSITIVE CONTROL). Sample types are ordered in the same way as the sample_names vector.

If the Plate object is initialised using the default methods (read_luminex_data or any of the processing methods: process_dir, process_file and process_plate) the sample types are detected based on the sample names according to the rules described in translate_sample_names_to_sample_types.

dilutions (character())

A list containing names of the samples as keys and string representing dilutions as values. The dilutions are represented as strings. The dilutions are ordered in the same way as the sample_names vector

dilution_values (numeric())

A list containing names of the samples as keys and numeric values representing dilutions as values. The dilution values are ordered in the same way as the sample_names vector

default_data_type (character(1))

The default data type that will be returned by the get_data method. By default is set to Median.

data (list())

A list containing dataframes with the data for each sample and analyte. The possible data types - the keys of the list are

c(Median, Net MFI, Count, Avg Net MFI, Mean, Peak). In each dataframe, the rows represent samples and the columns represent analytes. Rows of each dataframe are ordered in the same way as the sample_names vector.

batch_info (list())

A list containing additional, technical information about the batch.

layout (character())

A list containing information about the layout of the plate. The layout is read from the separate file and usually provides additional information about the dilutions, sample names, and the sample layout on the actual plate.

Method print(): Function prints the basic information about the plate such as the number of samples and analytes

Usage:

Plate$print(...)

Arguments:

... Additional parameters to be passed to the print function Print the summary of the plate

Method summary(): Function outputs basic information about the plate, such as examination date, batch name, and sample types.

Usage:

Plate$summary(..., include_names = FALSE)

Arguments:

... Additional parameters to be passed to the print function Get data for a specific analyte and sample type

include_names If include_names parameter is TRUE, a part from count of control samples, provides also their names. By default FALSE

Method get_data(): Function returns data for a specific analyte and sample.

Usage:

```
Plate$get_data(
  analyte,
  sample_type = "ALL",
  data_type = self$default_data_type
)
```

Arguments:

analyte An analyte name or its id of which data we want to extract. If set to 'ALL' returns data for all analytes.

Plate 17

sample_type is a type of the sample we want to extract data from. The possible values are
   c(ALL, BLANK, TEST, NEGATIVE CONTROL, STANDARD CURVE, POSITIVE CONTROL). Default
   value is ALL.

data_type The parameter specifying which data type should be returned. This parameter has
   to take one of values:
   c(Median, Net MFI, Count, Avg Net MFI, Mean, Peak). What's more, the data_type has
   to be present in the plate's data Default value is plate's default_data_type, which is usu-
   ally Median.

Returns: Dataframe containing information about a given sample type and analyte Get the
string representation of dilutions

Method get_dilution(): Function returns the dilution represented as strings for a specific
sample type.

Usage:

Plate$get_dilution(sample_type)

Arguments:

sample_type type of the samples that we want to obtain the dilution for. The possible values
   are
   c(ALL, BLANK, TEST, NEGATIVE CONTROL, STANDARD CURVE, POSITIVE CONTROL) Default
   value is ALL.

Returns: A list containing names of the samples as keys and string representing dilutions as
values. Get the numeric representation of dilutions

Method get_dilution_values(): Function returns the dilution values for a specific sample
type.

Usage:

Plate$get_dilution_values(sample_type)

Arguments:

sample_type type of the samples that we want to obtain the dilution values for. The possible
   values are
   c(ALL, BLANK, TEST, NEGATIVE CONTROL, STANDARD CURVE, POSITIVE CONTROL) Default
   value is ALL.

Returns: A list containing names of the samples as keys and numeric values representing
dilutions as values.
Adjust the MFI values by subtracting the background

Method blank_adjustment(): Function adjusts the values of samples (all samples excluding
the blanks) by clamping the values to the aggregated value of the BLANK samples for each analyte
separately.

The purpose of this operation is to unify the data by clamping values below the background noise.
how this method works was inspired by the paper https://doi.org/10.1038/s41598-020-57876-0
which covers the quality control in the MBA.

In short, this operation firstly calculates the aggregate of MFI in the BLANK samples (available
methods are: min, max, mean, median) and then replaces all values below this threshold with the
threshold value.

Method does not modifies the data of type `Count`.

This operation is recommended to be performed before any further analysis, but is optional. Skipping it before further analysis is allowed, but will result in a warning.

Usage:

`Plate$blank_adjustment(threshold = "max", in_place = TRUE)`

Arguments:

`threshold` The method used to calculate the background value for each analyte. Every value below this threshold will be clamped to the threshold value. By default `max`. Available methods are: `min`, `max`, `mean`, `median`.

`in_place` Whether the method should produce new plate with adjusted values or not, By default `TRUE` - operates on the current plate.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

`Plate$clone(deep = FALSE)`

Arguments:

`deep` Whether to make a deep clone.

PlateBuilder	*PlateBuilder*

Description

This class helps creating the Plate object. It is used to store the data and validate the final fields.

Active bindings

`layout_as_vector` Print the layout associated with the plate as a flattened vector of values.

Methods

Public methods:

- [PlateBuilder$new()](#)
- [PlateBuilder$set_sample_locations()](#)
- [PlateBuilder$set_dilutions()](#)
- [PlateBuilder$set_sample_types()](#)
- [PlateBuilder$set_sample_names()](#)
- [PlateBuilder$set_plate_datetime()](#)
- [PlateBuilder$set_data()](#)
- [PlateBuilder$set_default_data_type()](#)
- [PlateBuilder$set_batch_info()](#)
- [PlateBuilder$set_plate_name()](#)
- [PlateBuilder$set_layout()](#)

- [PlateBuilder$build()](#)
- [PlateBuilder$clone()](#)

Method new(): Initialize the PlateBuilder object

Usage:

```
PlateBuilder$new(sample_names, analyte_names, batch_name = "", verbose = TRUE)
```

Arguments:

sample_names • vector of sample names measured during an examination in the same order as in the data

analyte_names • vector of analytes names measured during an examination in the same order as in the data

batch_name • name of the batch during which the plate was examined obtained from the plate info. An optional parameter, by default set to "" - an empty string.

verbose • logical value indicating whether to print additional information. This parameter is stored as a private attribute of the object and reused in other methods

Method set_sample_locations(): Set the sample types used during the examination

Usage:

```
PlateBuilder$set_sample_locations(sample_locations)
```

Arguments:

sample_locations vector of sample locations pretty name ie. A1, B2

Method set_dilutions(): Extract and set the dilutions from layout, sample names or use a provided vector of values. The provided vector should be the same length as the number of samples and should contain dilution factors saved as strings

Usage:

```
PlateBuilder$set_dilutions(use_layout_dilutions = TRUE, values = NULL)
```

Arguments:

use_layout_dilutions logical value indicating whether to use names extracted from layout files to extract dilutions. If set to FALSE the function uses the sample names as a source for dilution

values a vector of dilutions to overwrite the extraction process
Set and extract sample types from the sample names. Optionally use the layout file to extract the sample types

Method set_sample_types():

Usage:

```
PlateBuilder$set_sample_types(use_layout_types = TRUE, values = NULL)
```

Arguments:

use_layout_types logical value indicating whether to use names extracted from layout files to extract sample types

values a vector of sample types to overwrite the extraction process

Method set_sample_names(): Set the sample names used during the examination. If the layout is provided, extract the sample names from the layout file. Otherwise, uses the original sample names from the Luminex file

Usage:

```
PlateBuilder$set_sample_names(use_layout_sample_names = TRUE)
```

Arguments:

use_layout_sample_names logical value indicating whether to use names extracted from lay-
    out files. If set to false, this function only checks if the sample names are provided in the
    plate

Method set_plate_datetime(): Set the plate datetime for the plate

Usage:

```
PlateBuilder$set_plate_datetime(plate_datetime)
```

Arguments:

plate_datetime a POSIXct datetime object representing the date and time of the examination

Method set_data(): Set the data used during the examination

Usage:

```
PlateBuilder$set_data(data)
```

Arguments:

data a named list of data frames containing information about the samples and analytes. The
    list is named by the type of the data e.g. Median, Net MFI, etc. The data frames contain
    information about the samples and analytes The rows are different measures, whereas the
    columns represent different analytes Example of how data$Median looks like:

Sample	Analyte1	Analyte2	Analyte3
Sample1	1.2	2.3	3.4
Sample2	4.5	5.6	6.7
...	...	...	...
Sample96	7.8	8.9	9.0

Method set_default_data_type(): Set the data type used for calculations

Usage:

```
PlateBuilder$set_default_data_type(data_type = "Median")
```

Arguments:

data_type a character value representing the type of data that is currently used for calculations.
    By default, it is set to Median

Method set_batch_info(): Set the batch info for the plate

Usage:

```
PlateBuilder$set_batch_info(batch_info)
```

Arguments:

batch_info a raw list containing metadata about the plate read from the Luminex file

Method set_plate_name(): Set the plate name for the plate. The plate name is extracted from
the filepath

Usage:

```
PlateBuilder$set_plate_name(file_path)
```

Arguments:

`file_path` a character value representing the path to the file

Method `set_layout()`: Set the layout matrix for the plate. This function performs basic validation

- verifies if the plate is a matrix of shape 8x12 with 96 wells

Usage:

```
PlateBuilder$set_layout(layout_matrix)
```

Arguments:

`layout_matrix` a matrix containing information about the sample names. dilutions, etc.

Method `build()`: Create a Plate object from the PlateBuilder object

Usage:

```
PlateBuilder$build(validate = TRUE, reorder = TRUE)
```

Arguments:

`validate` logical value indicating whether to validate the fields

`reorder` logical value indicating whether to reorder the data according to the locations on the plate. If `FALSE`, the samples will be ordered in the same manner as in the CSV plate file.

Method `clone()`: The objects of this class are cloneable with this method.

Usage:

```
PlateBuilder$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

---

plot_counts	*Plot counts in a 96-well plate*

Description

This function plots counts in a 96-well plate using a colour to represent the count ranges. There is a possibility of plotting exact counts in each well.

If the plot window is resized, it's best to re-run the function to adjust the scaling. Sometimes, when a legend is plotted, the whole layout may be shifted. It's best to stretch the window, and everything will be adjusted automatically.

Usage

```
plot_counts(
  plate,
  analyte_name,
  plot_counts = TRUE,
  plot_legend = FALSE,
  lower_threshold = 50,
  higher_threshold = 70
)
```

Arguments

`plate`	The plate object with the counts data
`analyte_name`	The name of the analyte
`plot_counts`	Logical indicating if the counts should be plotted
`plot_legend`	Logical indicating if the legend should be plotted
`lower_threshold`	
	The lower threshold for the counts, it separates green and yellow colours
`higher_threshold`	
	The higher threshold for the counts, it separates yellow and red colours

Value

A ggplot object

Examples

```
plate_filepath <- system.file("extdata", "CovidOISExPONTENT_CO.csv",
  package = "SerolyzeR", mustWork = TRUE
)
layout_filepath <- system.file("extdata", "CovidOISExPONTENT_CO_layout.xlsx",
  package = "SerolyzeR", mustWork = TRUE
)
plate <- read_luminex_data(plate_filepath, layout_filepath)
plot_counts(
  plate = plate, analyte_name = "OC43_NP_NA",
  plot_counts = TRUE, plot_legend = FALSE
)
```

`plot_layout`	*Plot layout of a 96-well plate*

Description

This function plots the layout of a 96-well plate using a colour to represent the sample types.

If the plot window is resized, it's best to re-run the function to adjust the scaling. Sometimes, the whole layout may be shifted when a legend is plotted. It's best to stretch the window, and everything will be adjusted automatically.

Usage

```
plot_layout(plate, plot_legend = TRUE)
```

Arguments

plate   The plate object with the layout information

plot_legend   Logical indicating if the legend should be plotted

Value

A ggplot object

Examples

```
plate_filepath <- system.file("extdata", "CovidOISExPONENT_CO.csv",
  package = "SerolyzeR", mustWork = TRUE
)
layout_filepath <- system.file("extdata", "CovidOISExPONENT_CO_layout.xlsx",
  package = "SerolyzeR", mustWork = TRUE
)
plate <- read_luminex_data(plate_filepath, layout_filepath)
plot_layout(plate = plate, plot_legend = TRUE)
```

---

plot_levey_jennings   *Plot Levey-Jennings chart*

Description

The function plots a Levey-Jennings chart for the given analyte in the list of plates. The Levey-Jennings chart is a graphical representation of the data that enables the detection of outliers and trends. It is a quality control tool that is widely used in the laboratories across the world.

Usage

```
plot_levey_jennings(
  list_of_plates,
  analyte_name,
  dilution = "1/400",
  sd_lines = c(1.96),
```

```
  data_type = "Median"
)
```

Arguments

`list_of_plates`	A list of plate objects for which to plot the Levey-Jennings chart
`analyte_name`	(`character(1)`) the analyte for which to plot the Levey-Jennings chart
`dilution`	(`character(1)`) the dilution for which to plot the Levey-Jennings chart. The default is "1/400"
`sd_lines`	(`numeric`) the vector of coefficients for the standard deviation lines to plot, for example, c(1.96, 2.58) will plot four horizontal lines: mean +/- 1.96*sd, mean +/- 2.58*sd default is c(1.96) which will plot two lines mean +/- 1.96*sd
`data_type`	(`character(1)`) the type of data used plot. The default is "Median"

Value

A ggplot object with the Levey-Jennings chart

Examples

```
# creating temporary directory for the example
output_dir <- tempdir(check = TRUE)

dir_with_luminex_files <- system.file("extdata", "multiplate_reallife_reduced",
  package = "SerolyzeR", mustWork = TRUE
)
list_of_plates <- process_dir(dir_with_luminex_files,
  return_plates = TRUE, format = "xPONENT", output_dir = output_dir
)
list_of_plates <- rep(list_of_plates, 10) # since we have only 3 plates i will repeat them 10 times

plot_levey_jennings(list_of_plates, "ME", dilution = "1/400", sd_lines = c(0.5, 1, 1.96, 2.58))
```

---

plot_mfi_for_analyte   *Plot MFI value distribution for a given analyte*

Description

Plot MFI value distribution for a given analyte

Usage

```
plot_mfi_for_analyte(
  plate,
  analyte_name,
  data_type = "Median",
```

```
    plot_type = "violin",
    scale_y = "log10",
    plot_outliers = FALSE
)
```

Arguments

`plate`	A plate object
`analyte_name`	The analyte to plot
`data_type`	The type of data to plot. Default is "Median"
`plot_type`	The type of plot to generate. Default is "violin". Available options are "boxplot" and "violin".
`scale_y`	What kind of transformation of the scale to apply. By default MFI is presented in a "log10" scale. Available options are described in the documentation of [scale_y_continuous](#) under `transform` parameter.
`plot_outliers`	When using "boxplot" type of a plot one can set this parameter to TRUE and display the names of samples for which MFI falls outside the 1.5 IQR interval

Value

A ggplot object

---

plot_standard_curve_analyte
                          Standard curves

Description

Plot standard curve samples of a plate of a given analyte.

Usage

```
plot_standard_curve_analyte(
  plate,
  analyte_name,
  data_type = "Median",
  decreasing_rau_order = TRUE,
  log_scale = c("all"),
  plot_line = TRUE,
  plot_blank_mean = TRUE,
  plot_rau_bounds = TRUE,
  plot_legend = TRUE,
  verbose = TRUE
)
```

Arguments

`plate`	A plate object
`analyte_name`	Name of the analyte of which standard curve we want to plot.
`data_type`	Data type of the value we want to plot - the same datatype as in the plate file. By default equals to `Net MFI`
`decreasing_rau_order`	
	If TRUE the RAU values are plotted in decreasing order, TRUE by default
`log_scale`	Which elements on the plot should be displayed in log scale. By default `"RAU"`. If NULL or `c()` no log scale is used, if `"all"` or `c("RAU", "MFI")` all elements are displayed in log scale.
`plot_line`	If TRUE a line is plotted, TRUE by default
`plot_blank_mean`	
	If TRUE the mean of the blank samples is plotted, TRUE by default
`plot_rau_bounds`	
	If TRUE the RAU values bounds are plotted, TRUE by default
`plot_legend`	If TRUE the legend is plotted, TRUE by default
`verbose`	If TRUE prints messages, TRUE by default

Value

ggplot object with the plot

Examples

```
path <- system.file("extdata", "CovidOISExPONENT.csv",
  package = "SerolyzeR", mustWork = TRUE
)
layout_path <- system.file("extdata", "CovidOISExPONENT_layout.xlsx",
  package = "SerolyzeR", mustWork = TRUE
)
plate <- read_luminex_data(path, layout_filepath = layout_path, verbose = FALSE)
plot_standard_curve_analyte(plate, "Spike_6P", plot_legend = FALSE, data_type = "Median")
```

`plot_standard_curve_analyte_with_model`
                    Plot standard curve of a certain analyte with fitted model

---

Description

Function plots the values of standard curve samples and the fitted model.

Usage

```
plot_standard_curve_analyte_with_model(
  plate,
  model,
  data_type = "Median",
  decreasing_rau_order = TRUE,
  log_scale = c("all"),
  plot_asymptote = TRUE,
  plot_test_predictions = TRUE,
  plot_blank_mean = TRUE,
  plot_rau_bounds = TRUE,
  plot_legend = TRUE,
  verbose = TRUE,
  ...
)
```

Arguments

`plate`	Plate object
`model`	fitted `Model` object, which predictions we want to plot
`data_type`	Data type of the value we want to plot - the same datatype as in the plate file. By default equals to `Median`
`decreasing_rau_order`	
	If TRUE the RAU values are plotted in decreasing order, `TRUE` by default.
`log_scale`	Which elements on the plot should be displayed in log scale. By default `"all"`. If NULL or `c()` no log scale is used, if `"all"` or `c("RAU", "MFI")` all elements are displayed in log scale.
`plot_asymptote`	If TRUE the asymptotes are plotted, `TRUE` by default
`plot_test_predictions`	
	If TRUE the predictions for the test samples are plotted, `TRUE` by default. The predictions are obtained through extrapolation of the model
`plot_blank_mean`	
	If TRUE the mean of the blank samples is plotted, `TRUE` by default
`plot_rau_bounds`	
	If TRUE the RAU bounds are plotted, `TRUE` by default
`plot_legend`	If TRUE the legend is plotted, `TRUE` by default
`verbose`	If TRUE prints messages, `TRUE` by default
`...`	Additional arguments passed to the `predict` function

Value

a ggplot object with the plot

Examples

```
path <- system.file("extdata", "CovidOISExPONENT.csv",
  package = "SerolyzeR", mustWork = TRUE
)
layout_path <- system.file("extdata", "CovidOISExPONENT_layout.xlsx",
  package = "SerolyzeR", mustWork = TRUE
)
plate <- read_luminex_data(path, layout_filepath = layout_path, verbose = FALSE)
model <- create_standard_curve_model_analyte(plate, analyte_name = "Spike_B16172")
plot_standard_curve_analyte_with_model(plate, model, decreasing_rau_order = FALSE)
```

---

plot_standard_curve_stacked

Standard curve stacked plot for levey-jennings report

Description

Function generates a plot of stacked on top of each other standard curves for a given analyte form a list of plates. The plot is created with the levey-jennings report in mind, but it can be run by itself.

Usage

```
plot_standard_curve_stacked(
  list_of_plates,
  analyte_name,
  data_type = "Median",
  decreasing_dilution_order = TRUE,
  monochromatic = TRUE,
  legend_type = NULL,
  log_scale = c("all"),
  verbose = TRUE
)
```

Arguments

list_of_plates	list of Plate objects
analyte_name	Name of the analyte of which standard curves we want to plot.
data_type	Data type of the value we want to plot - the same datatype as in the plate file. By default equals to Median
decreasing_dilution_order	
	If TRUE the dilution values are plotted in decreasing order, TRUE by default
monochromatic	If TRUE the color of standard curves changes from white (the oldest) to blue (the newest) it helps to observe drift in calibration of the device; otherwise, more varied colours are used, TRUE by default

legend_type   default value is NULL, then legend type is determined based on monochromatic value. If monochromatic is equal to TRUE then legend type is set to date, if it is FALSE then legend type is set to plate_name. User can override this behavior by setting explicitly legend_type to date or plate_name.

log_scale   Which elements on the plot should be displayed in log scale. By default "all". If NULL or c() no log scale is used, if "all" or c("dilutions", "MFI") all elements are displayed in log scale.

verbose   If TRUE prints messages, TRUE by default

Value

ggplot object with the plot

Examples

```
# creating temporary directory for the example
output_dir <- tempdir(check = TRUE)

dir_with_luminex_files <- system.file("extdata", "multiplate_reallife_reduced",
  package = "SerolyzeR", mustWork = TRUE
)
list_of_plates <- process_dir(dir_with_luminex_files,
  return_plates = TRUE, format = "xPONENT", output_dir = output_dir
)
plot_standard_curve_stacked(list_of_plates, "ME", data_type = "Median", monochromatic = FALSE)
```

---

predict.Model   *Predict the RAU values from the MFI values*

Description

More details can be found here: [Model](#)

Usage

```
## S3 method for class 'Model'
predict(object, mfi, ...)
```

Arguments

object   (Model()) Object of the Model class

mfi   (numeric()) MFI values for which we want to predict the RAU values Should be in the same scale as the MFI values used to fit the model

...   Additional arguments passed to the method

Value

(data.frame())

process_dir   *Process a Directory of Luminex Data Files*

Description

This function processes all Luminex plate files within a specified directory. Each plate file is processed using process_file(), and the resulting normalised data is saved. Optionally, quality control reports can be generated, and results from multiple plates can be merged into a single file.

Workflow:

1. Identify all Luminex plate files in the input_dir, applying recursive search if recurse = TRUE.
2. Detect the format of each file based on the format parameter or the filename.
3. Locate the corresponding layout file using the filename or use the common layout passed with the layout_filepath parameter.
4. Determine the appropriate output directory using get_output_dir().
5. Process each plate file using process_file().
6. If merge_outputs = TRUE, merge normalised data from multiple plates into a single CSV file.

Naming Conventions for Input Files:

- **If** format **is specified:**
  - Each plate file should be named as {plate_name}.csv.
  - The corresponding layout file should be named as {plate_name}_layout.csv or {plate_name}_layout.xlsx.
  - Alternatively, if layout_filepath is provided, it serves as a unified layout file for all plates.
- **If** format **equals** NULL **(automatic detection):**
  - Each plate file should be named as {plate_name}_{format}.csv, where {format} is either xPONENT or INTELLIFLEX.
  - The corresponding layout file should be named using the same convention as above, i.i. {plate_name}_{format}_layout.csv or {plate_name}_{format}_layout.xlsx.

Output File Structure:

- The output_dir parameter specifies where the processed files are saved.
- If output_dir is NULL, output files are saved in the same directory as the input files.
- By default, the output directory structure follows the input directory, unless flatten_output_dir = TRUE, which saves all outputs directly into output_dir.
- Output filenames follow the convention used in process_file().
  - For a plate named {plate_name}, the normalised output files are named as:
    * {plate_name}_RAU.csv for RAU normalisation.
    * {plate_name}_nMFI.csv for nMFI normalisation.
    * {plate_name}_MFI.csv for MFI normalisation.
  - If generate_reports = TRUE, a quality control report for every plate is saved as {plate_name}_report.pdf.

– If merge_outputs = TRUE, merged normalised files are named as:

* merged_RAU_{timestamp}.csv
* merged_nMFI_{timestamp}.csv
* merged_MFI_{timestamp}.csv

– If generate_multiplate_reports = TRUE, a multiplate quality control report is saved as multiplate_report_{timestamp}.pdf.

Usage

```
process_dir(
  input_dir,
  output_dir = NULL,
  recurse = FALSE,
  flatten_output_dir = FALSE,
  layout_filepath = NULL,
  format = NULL,
  normalisation_types = c("MFI", "RAU", "nMFI"),
  generate_reports = FALSE,
  generate_multiplate_reports = FALSE,
  merge_outputs = FALSE,
  column_collision_strategy = "intersection",
  return_plates = FALSE,
  dry_run = FALSE,
  verbose = TRUE,
  ...
)
```

Arguments

input_dir	(character(1)) Path to the directory containing plate files. Can contain subdirectories if recurse = TRUE.
output_dir	(character(1), optional) Path to the directory where output files will be saved. Defaults to NULL (same as input directory).
recurse	(logical(1), default = FALSE)
	• If TRUE, searches for plate files in subdirectories as well.
flatten_output_dir	(logical(1), default = FALSE)
	• If TRUE, saves output files directly in output_dir, ignoring the input directory structure.
layout_filepath	(character(1), optional) Path to a layout file. If NULL, the function attempts to detect it automatically.
format	(character(1), optional) Luminex data format. If NULL, it is automatically detected. Options: 'xPONENT', 'INTELLIFLEX'.
normalisation_types	(character(), default = c("MFI", "RAU", "nMFI"))

                   • The normalisation types to apply. Supported values: "MFI", "RAU", "nMFI".

generate_reports

               (logical(1), default = FALSE)

                   • If TRUE, generates single plate quality control reports for each processed plate file.

generate_multiplate_reports

               (logical(1), default = FALSE)

                   • If TRUE, generates a multiplate quality control report for all processed plates.

merge_outputs   (logical(1), default = FALSE)

                   • If TRUE, merges all normalised data into a single CSV file per normalisation type.

                   • The merged file is named merged_{normalisation_type}_{timestamp}.csv.

column_collision_strategy

               (character(1), default = 'intersection')

                   • Determines how to handle missing or extra columns when merging outputs.

                   • Options: 'union' (include all columns), 'intersection' (include only common columns).

return_plates   (logical(1), default = FALSE)

                   • If TRUE, returns a list of processed plates sorted by experiment date.

dry_run   (logical(1), default = FALSE)

                   • If TRUE, prints file details without processing them.

verbose   (logical(1), default = TRUE)

                   • If TRUE, prints detailed processing information.

...   Additional arguments passed to [process_file()](#).

Value

If return_plates = TRUE, returns a sorted list of [Plate](#) objects. Otherwise, returns NULL.

Examples

```
# Process all plate files in a directory
input_dir <- system.file("extdata", "multiplate_lite", package = "SerolyzeR", mustWork = TRUE)
output_dir <- tempdir(check = TRUE)
plates <- process_dir(input_dir, return_plates = TRUE, output_dir = output_dir)
```

---

process_file   *Process a File to Generate Normalised Data and Reports*

Description

This function reads a Luminex plate file by calling [read_luminex_data()](#) and then processes it by calling [process_plate()](#). It optionally generates also a quality control report using [generate_plate_report()](#). It reads the specified plate file, processes the plate object using all specified normalisation types (including raw MFI values), and saves the results. If generate_report = TRUE, a quality control report is also generated.

Workflow:

1. Read the plate file and layout file.
2. Process the plate data using the specified normalisation types (MFI, RAU, nMFI).
3. Save the processed data to CSV files in the specified output_dir. The files are named as {plate_name}_{normalisation_type}.csv.
4. Optionally, generate a quality control report. The report is saved as an HTML file in the output_dir, under the name {plate_name}_report.html.

Usage

```
process_file(
  plate_filepath,
  layout_filepath,
  output_dir = "normalised_data",
  format = "xPONENT",
  generate_report = FALSE,
  process_plate = TRUE,
  normalisation_types = c("MFI", "RAU", "nMFI"),
  blank_adjustment = FALSE,
  verbose = TRUE,
  ...
)
```

Arguments

plate_filepath (character(1)) Path to the Luminex plate file.

layout_filepath
                (character(1)) Path to the corresponding layout file.

output_dir   (character(1), default = 'normalised_data')

                • Directory where the output files will be saved.
                • If it does not exist, it will be created.

format   (character(1), default = 'xPONENT')

                • Format of the Luminex data.
                • Available options: 'xPONENT', 'INTELLIFLEX'.

generate_report

> (logical(1), default = FALSE)
>
>> • If TRUE, generates a quality control report using generate_plate_report().

process_plate   (logical(1), default = TRUE)

>> • If TRUE, processes the plate data using process_plate().
>> • If FALSE, only reads the plate file and returns the plate object without processing.

normalisation_types

> (character(), default = c("MFI", "RAU", "nMFI"))
>
>> • List of normalisation types to apply.
>> • Supported values: c("MFI", "RAU", "nMFI").

blank_adjustment

> (logical(1), default = FALSE)
>
>> • If TRUE, performs blank adjustment before processing.

verbose   (logical(1), default = TRUE)

>> • If TRUE, prints additional information during execution.

...   Additional arguments passed to read_luminex_data() and generate_plate_report().

Value

A Plate object containing the processed data.

Examples

```
# Example 1: Process a plate file with default settings (all normalisation types)
plate_file <- system.file("extdata", "CovidOISExPONENT_CO_reduced.csv", package = "SerolyzeR")
layout_file <- system.file("extdata", "CovidOISExPONENT_CO_layout.xlsx", package = "SerolyzeR")
example_dir <- tempdir(check = TRUE)
process_file(plate_file, layout_file, output_dir = example_dir)

# Example 2: Process the plate for only RAU normalisation
process_file(plate_file, layout_file, output_dir = example_dir, normalisation_types = c("RAU"))

# Example 3: Process the plate and generate a quality control report
process_file(plate_file, layout_file, output_dir = example_dir, generate_report = TRUE)
```

---

process_plate   *Process a Plate and Save Normalised Output to CSV*

Description

This function processes a plate and computes normalised values based on the specified `normalisation_type`. The function supports three types of normalisation:

- **RAU** (Relative Antibody Units) - Default normalisation type.
- **nMFI** (Normalised Median Fluorescence Intensity).
- **MFI** (Median Fluorescence Intensity, blank-adjusted raw MFI values).

Depending on the chosen normalisation type, the function:

- Adjusts for blanks (if `blank_adjustment = TRUE`).
- Computes the relevant normalised values for each analyte in the plate.
- Aggregates computed values into a single data frame.
- Optionally writes the results to a CSV file.

RAU Normalisation Workflow:

1. Blank adjustment (if enabled).
2. Fit standard curve models for each analyte.
3. Compute RAU values based on the fitted models.
4. Aggregate RAU values into a data frame.
5. Save the results to a CSV file.

nMFI Normalisation Workflow:

1. Blank adjustment (if enabled).
2. Compute nMFI values using the target dilution.
3. Aggregate nMFI values into a data frame.
4. Save the results to a CSV file.

MFI Normalisation Workflow:

1. Blank adjustment (if enabled).
2. Save the adjusted MFI values to a CSV file.

If the plate is already blank-adjusted, setting `blank_adjustment = TRUE` has no effect.

More details on the normalisation methods can be found in:

- RAU: create_standard_curve_model_analyte.
- nMFI: get_nmfi.

Usage

```
process_plate(
  plate,
  filename = NULL,
  output_dir = "normalised_data",
  write_output = TRUE,
  normalisation_type = "RAU",
  data_type = "Median",
  blank_adjustment = FALSE,
  verbose = TRUE,
  reference_dilution = 1/400,
  ...
)
```

Arguments

plate	(Plate) A plate object containing fluorescence intensity data.
filename	(character(1), optional) Output CSV filename. Defaults to {plate_name}_{normalisation_type}.c
	• If omitted (NULL), the filename is auto-generated.
	• If the filename lacks .csv, the extension is automatically added.
	• If output_dir is specified, it is combined with filename unless filename is an absolute path.
	• If a file already exists at the path, it will be overwritten.
output_dir	(character(1), default = 'normalised_data') Directory for saving the output file.
	• If the directory does not exist, it will be created.
	• If NULL, the current working directory is used.
write_output	(logical(1), default = TRUE) Whether to save the output to a CSV file.
normalisation_type	
	(character(1), default = 'RAU') The normalisation method to apply.
	• Allowed values: c(MFI, RAU, nMFI).
data_type	(character(1), default = 'Median') The data type to use for calculations.
blank_adjustment	
	(logical(1), default = FALSE) Whether to perform blank adjustment before computing values.
verbose	(logical(1), default = TRUE) Whether to print additional information during execution.
reference_dilution	
	(numeric(1) or character(1), default = 1/400)
	• The target dilution used as a reference for nMFI normalisation.
	• Ignored for RAU normalisation.
	• Can be numeric (e.g., 0.0025) or a string ('1/400').
...	Additional arguments passed to the model fitting function (create_standard_curve_model_analyte).

Value

A data frame containing the computed normalised values.

Examples

```
plate_file <- system.file("extdata", "CovidOISExPONTENT_CO_reduced.csv", package = "SerolyzeR")
layout_file <- system.file("extdata", "CovidOISExPONTENT_CO_layout.xlsx", package = "SerolyzeR")
plate <- read_luminex_data(plate_file, layout_file, verbose = FALSE)
example_dir <- tempdir(check = TRUE)

# Process plate with default settings (RAU normalisation)
process_plate(plate, output_dir = example_dir)

# Process plate without blank adjustment, custom filename
process_plate(plate,
  filename = "plate_no_blank_adjustment.csv",
  output_dir = example_dir, blank_adjustment = FALSE
)

# Process plate with nMFI normalisation
process_plate(plate,
  output_dir = example_dir, normalisation_type = "nMFI",
  reference_dilution = 1 / 400
)
```

---

read_intelliflex_format

Read the Intelliflex format data

Description

Read the Intelliflex format data

Usage

```
read_intelliflex_format(path, verbose = TRUE)
```

Arguments

path	Path to the INTELLIFLEX file
verbose	Print additional information. Default is TRUE

---

read_layout_data   *Read layout data from a file*

Description

Read layout data from a file

Usage

```
read_layout_data(layout_file_path, ...)
```

Arguments

layout_file_path

                 Path to the layout file

...   Additional arguments to pass to the underlying read function

Value

A matrix with the layout data. The row names are supposed to be letters A,B,C, etc. The column names are supposed to be numbers 1,2,3, etc.

---

read_luminex_data   *Read Luminex Data*

Description

Reads a Luminex plate file and returns a `Plate` object containing the extracted data. Optionally, a layout file can be provided to specify the arrangement of samples on the plate.

The function supports two Luminex data formats:

- **xPONENT**: Used by older Luminex machines.
- **INTELLIFLEX**: Used by newer Luminex devices.

  Workflow:
  1. Validate input parameters, ensuring the specified format is supported.
  2. Read the plate file using the appropriate parser:
     - xPONENT files are read using `read_xponent_format()`.
     - INTELLIFLEX files are read using `read_intelliflex_format()`.
  3. Post-process the extracted data:
     - Validate required data columns (`Median`, `Count`).
     - Extract sample locations and analyte names.
     - Parse the date and time of the experiment.

File Structure:

- **Plate File** (`plate_filepath`): A CSV file containing Luminex fluorescence intensity data.
- **Layout File** (`layout_filepath`) (optional): An Excel or CSV file containing the plate layout.
  - The layout file should contain a table with **8 rows and 12 columns**, where each cell corresponds to a well location.
  - The values in the table represent the sample names for each well.

Sample types detection:

The [read_luminex_data](#) method automatically detects the sample types based on the sample names, unless provided the sample_types parameter. The sample types are detected used the [translate_sample_names_to_sample_types](#) method. In the documentation of this method, which can be accessed with command ?translate_sample_names_to_sample_types, you can find the detailed description of the sample types detection.

Usage

```
read_luminex_data(
  plate_filepath,
  layout_filepath = NULL,
  format = "xPONENT",
  plate_file_separator = ",",
  plate_file_encoding = "UTF-8",
  use_layout_sample_names = TRUE,
  use_layout_types = TRUE,
  use_layout_dilutions = TRUE,
  default_data_type = "Median",
  sample_types = NULL,
  dilutions = NULL,
  verbose = TRUE
)
```

Arguments

plate_filepath (character(1)) Path to the Luminex plate file.

layout_filepath

        (character(1), optional) Path to the Luminex layout file.

format   (character(1), default = 'xPONENT')

- The format of the Luminex data file.
- Supported formats: 'xPONENT', 'INTELLIFLEX'.

plate_file_separator

        (character(1), default = ',')

- The delimiter used in the plate file (CSV format). Used only for the xPONENT format.

plate_file_encoding

        (character(1), default = 'UTF-8')

- The encoding used for reading the plate file. Used only for the xPONENT format.

use_layout_sample_names

                    (`logical(1)`, default = `TRUE`)

- Whether to use sample names from the layout file.

use_layout_types

                    (`logical(1)`, default = `TRUE`)

- Whether to use sample types from the layout file (requires a layout file).

use_layout_dilutions

                    (`logical(1)`, default = `TRUE`)

- Whether to use dilution values from the layout file (requires a layout file).

default_data_type

                    (`character(1)`, default = `'Median'`)

- The default data type used if none is explicitly provided.

sample_types   (`character()`, optional) A vector of sample types to override extracted values.

dilutions   (`numeric()`, optional) A vector of dilutions to override extracted values.

verbose   (`logical(1)`, default = `TRUE`)

- Whether to print additional information and warnings.

Value

A [`Plate`](#) object containing the parsed Luminex data.

Examples

```
# Read a Luminex plate file with an associated layout file
plate_file <- system.file("extdata", "CovidOISExPONENT.csv", package = "SerolyzeR")
layout_file <- system.file("extdata", "CovidOISExPONENT_layout.csv", package = "SerolyzeR")
plate <- read_luminex_data(plate_file, layout_file)

# Read a Luminex plate file without a layout file
plate_file <- system.file("extdata", "CovidOISExPONENT_CO.csv", package = "SerolyzeR")
plate <- read_luminex_data(plate_file, verbose = FALSE)
```

---

read_xponent_format   *Read the xPONENT format data*

Description

Read the xPONENT format data

Usage

```
read_xponent_format(
  path,
  exact_parse = FALSE,
  encoding = "utf-8",
  separator = ",",
  verbose = TRUE
)
```

Arguments

`path`	Path to the xPONENT file
`exact_parse`	Whether to parse the file exactly or not Exact parsing means that the batch, calibration and assay metadata will be parsed as well
`encoding`	Encoding of the file
`separator`	Separator for the CSV values
`verbose`	Whether to print the progress. Default is `TRUE`

`translate_sample_names_to_sample_types`
Translate sample names to sample types

Description

Function translates sample names to sample types based on the sample name from Luminex file and the sample name from the layout file, which may not be provided. The function uses regular expressions to match the sample names to the sample types.

It parses the names as follows:

If `sample_names` or `sample_names_from_layout` equals to `BLANK`, `BACKGROUND` or `B`, then SampleType equals to `BLANK`

If `sample_names` or `sample_names_from_layout` equals to `STANDARD CURVE`, `SC`, `S`, contains substring 1/\d+ and has prefix , `S_`, `S` , `S` or `CP3`, then SampleType equals to `STANDARD CURVE`. For instance, sample with a name `S_1/2` or `S 1/2` will be classified as `STANDARD CURVE`.

If `sample_names` or `sample_names_from_layout` equals to `NEGATIVE CONTROL`, `N`, or contains substring `NEG`, then SampleType equals to `NEGATIVE CONTROL`

If `sample_names` or `sample_names_from_layout` starts with `P` followed by whitespace, `POS` followed by whitespace, some sample name followed by substring 1/\d+ SampleType equals to `POSITIVE CONTROL`

Otherwise, the returned SampleType is `TEST`

Usage

```
translate_sample_names_to_sample_types(
  sample_names,
  sample_names_from_layout = NULL
)
```

Arguments

sample_names   (character())
                   Vector of sample names from Luminex file

sample_names_from_layout
                   (character())
                   Vector of sample names from Layout file values in this vector may be differ-
                   ent than sample_names and may contain additional information about the sam-
                   ple type like dilution. This vector when set has to have at least the length of
                   sample_names.

Value

A vector of valid sample_type strings of length equal to the length of sample_names

Examples

```
translate_sample_names_to_sample_types(c("B", "BLANK", "NEG", "TEST1"))
translate_sample_names_to_sample_types(c("S", "CP3"))
```

Index