Package 'PvSTATEM'

October 3, 2024

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 normalizes 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 of the same name - 'PvSTATEM', which is an international project aiming for malaria elimination.

BugReports https://github.com/mini-pw/PvSTATEM/issues

Version 0.0.4

License BSD_3_clause + file LICENSE

Encoding UTF-8

RoxygenNote 7.3.2

Imports dplyr, ggplot2, nplr, R6, readr, readxl, stringi, stringr, grid, png, tools, ggrepel,

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

Config/testfhat/edition 3

VignetteBuilder knitr

URL <https://github.com/mini-pw/PvSTATEM>

NeedsCompilation no

Author Tymoteusz Kwiecinski [aut, cre], Jakub Grzywaczewski [aut], Mateusz Nizwantowski [aut]

Maintainer Tymoteusz Kwiecinski <tymoteuszkwiecinski@gmail.com>

Repository CRAN

Date/Publication 2024-10-03 19:00:06 UTC

Contents

`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,
  ...
)
```

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

...
: Additional arguments passed to the model

Value

(Model()) Standard Curve model

---

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

This model uses the nplr package to fit the model. The 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.

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 the dilutions from the MFI values

Usage:

```
Model$predict(mfi)
```

Arguments:

`mfi (numeric())`
: MFI values for which we want to predict the dilutions.

Returns: `(data.frame())`
Dataframe with the predicted dilutions, MFI values, and the 97.5% confidence intervals The columns are named as follows:

- `dilution` - the dilution value
- `dilution.025` - the lower bound of the confidence interval
- `dilution.975` - the upper bound of the confidence interval
- `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

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", "CovidOISExPONTENT.csv", package = "PvSTATEM")
layout_file <- system.file("extdata", "CovidOISExPONTENT_layout.csv", package = "PvSTATEM")
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

A class to represent the luminex plate. It contains information about the samples and analytes that were examined on the plate as well as some additional metadata and batch info

Public fields

`plate_name (character(1))`
: Name of the plate.

`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.

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

`sample_locations (character())`
: Locations of the samples on the plate.

`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)`.

`dilutions (character())`
: A list containing names of the samples as keys and string representing dilutions as values. The dilutions are represented as strings.

`dilution_values (numeric())`
: A list containing names of the samples as keys and numeric values representing dilutions as values.

`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.

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

`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()`

Method `new()`**:** Method to initialize the Plate object

Usage:

```
Plate$new(
  plate_name,
  sample_names,
  analyte_names,
  batch_name = "",
  dilutions = NULL,
  dilution_values = NULL,
  sample_types = NULL,
  data = NULL,
  sample_locations = NULL,
  default_data_type = 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.

`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

`dilutions (character())`
: A list containing names of the samples as keys and string representing dilutions as values. The dilutions are represented as strings.

`dilution_values (numeric())`
: A list containing names of the samples as keys and numeric values representing dilutions as values.

`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)`.

`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.

`sample_locations (character())`
: Locations of the samples on the plate.

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

`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.

`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 usually `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`.

`inplace` 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$set_data()
- PlateBuilder$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 layout files. If set to false, this function only checks if the sample names are provided in the plate

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)
```

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)
```

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 is a function used to plot counts in a 96-well plate using a color to represent the count ranges. There is possibility to plot exact counts in each well.

If plot window is resized, it's best to re-run the function to adjust the scaling. Sometimes when legend is plotted, whole layout may be shifted, then 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 colors
higher_threshold	The higher threshold for the counts, it separates yellow and red colors

Value

A ggplot object

Examples

```
plate_filepath <- system.file("extdata", "CovidOISExPONTENT_CO.csv",
  package = "PvSTATEM", mustWork = TRUE
)
layout_filepath <- system.file("extdata", "CovidOISExPONTENT_CO_layout.xlsx",
  package = "PvSTATEM", 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 is a function used to plot the layout of a 96-well plate using a color to represent the sample types.

If plot window is resized, it's best to re-run the function to adjust the scaling. Sometimes when legend is plotted, whole layout may be shifted, then 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", "CovidOISExPONTENT_CO.csv",
  package = "PvSTATEM", mustWork = TRUE
)
layout_filepath <- system.file("extdata", "CovidOISExPONTENT_CO_layout.xlsx",
  package = "PvSTATEM", mustWork = TRUE
)
plate <- read_luminex_data(plate_filepath, layout_filepath)
plot_layout(plate = plate, plot_legend = TRUE)
```

---

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 = "boxplot"
)
```

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 "boxplot". Available options are "boxplot" and "violin".

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_dilution_order = TRUE,
  log_scale = c("all"),
  plot_line = TRUE,
  plot_blank_mean = TRUE,
  plot_dilution_bounds = 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_dilution_order`	If `TRUE` the dilutions are plotted in decreasing order, `TRUE` by default
`log_scale`	Which elements on the plot should be displayed in log scale. By default `"dilutions"`. If `NULL` or `c()` no log scale is used, if `"all"` or `c("dilutions", "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_dilution_bounds`	If `TRUE` the dilution bounds are plotted, `TRUE` by default
`verbose`	If `TRUE` prints messages, `TRUE` by default

Value

ggplot object with the plot

`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_dilution_order = TRUE,
  log_scale = c("all"),
  plot_asymptote = TRUE,
  plot_test_predictions = TRUE,
  plot_blank_mean = TRUE,
  plot_dilution_bounds = 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_dilution_order`	If `TRUE` the dilutions 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("dilutions", "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_dilution_bounds`	If `TRUE` the dilution bounds are plotted, `TRUE` by default
`verbose`	If `TRUE` prints messages, `TRUE` by default

Value

a ggplot object with the plot

`predict.Model` *Predict the dilutions 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 dilutions. 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_plate` *Process a plate and save computed dilutions to a CSV*

Description

The behavior can be summarized as follows:

1. Adjust blanks if not already done.
2. Fit a model to each analyte using standard curve samples.
3. Compute dilutions for each analyte using the corresponding model.
4. Aggregate computed dilutions into a single data frame.
5. Save the computed dilutions to a CSV file.

Usage

```
process_plate(
  plate,
  output_path = NULL,
  data_type = "Median",
  adjust_blanks = FALSE,
  verbose = TRUE,
  ...
)
```

Arguments

plate	(Plate()) a plate object
output_path	(character(1)) path to save the computed dilutions. If not provided the file will be saved in the working directory with the name dilutions_{plate_name}.csv. Where the {plate_name} is the name of the plate.
data_type	(character(1)) type of data to use for the computation. Median is the default
adjust_blanks	(logical(1)) adjust blanks before computing dilutions. Default is FALSE
verbose	(logical(1)) print additional information. Default is TRUE
...	Additional arguments to be passed to the fit model function (create_standard_curve_model_analyte)

Examples

```
plate_file <- system.file("extdata", "CovidOISExPONTENT.csv", package = "PvSTATEM")
layout_file <- system.file("extdata", "CovidOISExPONTENT_layout.csv", package = "PvSTATEM")

plate <- read_luminex_data(plate_file, layout_file)

tmp_dir <- tempdir(check = TRUE)
temporary_filepath <- file.path(tmp_dir, "output.csv")
process_plate(plate, output_path = temporary_filepath)
# create and save dataframe with computed dilutions
```

---

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 file containing Luminex data and returns a Plate object. If provided, can also read a layout file, which usually contains information about the sample names, sample types or its dilutions.

The function is capable of reading data in two different formats:

- xPONENT
- INTELLIFLEX which are produced by two different Luminex machines.

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	Path to the Luminex plate file
layout_filepath	Path to the Luminex layout file
format	The format of the Luminex data. Select from: xPONENT, INTELLIFLEX
plate_file_separator	The separator used in the plate file
plate_file_encoding	The encoding used in the plate file
use_layout_sample_names	Whether to use names from the layout file in extracting sample names.
use_layout_types	Whether to use names from the layout file in extracting sample types. Works only when layout file is provided
use_layout_dilutions	Whether to use dilutions from the layout file in extracting dilutions. Works only when layout file is provided
default_data_type	The default data type to use if none is specified
sample_types	a vector of sample types to use instead of the extracted ones
dilutions	a vector of dilutions to use instead of the extracted ones
verbose	Whether to print additional information and warnings. TRUE by default

Value

Plate file containing the Luminex data

Examples

```
plate_file <- system.file("extdata", "CovidOISExPONTENT.csv", package = "PvSTATEM")
layout_file <- system.file("extdata", "CovidOISExPONTENT_layout.csv", package = "PvSTATEM")
plate <- read_luminex_data(plate_file, layout_file)

plate_file <- system.file("extdata", "CovidOISExPONTENT_CO.csv", package = "PvSTATEM")
layout_file <- system.file("extdata", "CovidOISExPONTENT_CO_layout.xlsx", package = "PvSTATEM")
# To suppress warnings and additional information use verbose = FALSE
plate <- read_luminex_data(plate_file, layout_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

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 different than sample_names and may contain additional information about the sample 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