Package 'bioset'

November 3, 2017

Type Package

Title Convert a Matrix of Raw Values into Nice and Tidy Data

Version 0.2.0

Description Functions to help dealing with raw data from measurements, like reading and transforming raw values organised in matrices, calculating and converting concentrations and calculating precision of duplicates / triplicates / ... . It is compatible with and building on top of 'tidyverse'-packages.

URL https://github.com/randomchars42/bioset

BugReports https://github.com/randomchars42/bioset/issues

License MIT + file LICENSE

Depends R (>= 3.4.2)

Imports utils, stats, graphics, grDevices, tidyr (>= 0.7.1), dplyr (>= 0.7.4), rlang (>= 0.1.2), tibble (>= 1.3.4), magrittr (>= 1.5)

Suggests tidyverse (>= 1.1.1), ggplot2 (>= 2.2.1), testthat (>= 1.0.2), knitr (>= 1.17), rmarkdown (>= 1.6)

Encoding UTF-8

LazyData true

RoxygenNote 6.0.1

VignetteBuilder knitr

NeedsCompilation no

Author Eike Christian Kühn [aut, cre]

Maintainer Eike Christian Kühn <eike.kuehn@pixelwoelkchen.de>

Repository CRAN

Date/Publication 2017-11-03 12:39:28 UTC

R topics documented:

`bioset-package` *Convert a matrix of raw values into nice and tidy data.*

Description

bioset is intended to save you from commons tasks when dealing with raw data obtained e.g. from a measuring device.

Details

When importing values with bioset start with `set_read()`. To get an example for a workflow refer to the introductory vignette (`vignette("introduction", "bioset")`).

Author(s)

Maintainer: Eike Christian Kühn <eike.kuehn@pixelwoelkchen.de>

See Also

Useful links:

- https://github.com/randomchars42/bioset
- Report bugs at https://github.com/randomchars42/bioset/issues

`calc_factor_conc` *Get a factor to convert concentrations.*

Description

Calculate a factor to convert concentration "A" into concentration "B".

Usage

```
calc_factor_conc(from, to, molar_mass = 0, density_solute = 0,
  density_solution = 0)
```

Arguments

`from`	A string containing the units of concentration A.
`to`	A string containing the units of concentration B.
`molar_mass`	The molar mass of the solute (g / mol).
`density_solute`	The density of the solute (g / l).
`density_solution`	The density of the solution (g / l), not the solvent!

Details

The following concentrations can be converted:

mass / volume: ".g / .l", ".g / .m^3", "% w / v"

molar / volume: ".M", ".mol / .l", ".mol / .m^3"

volume / volume: ".l/.l", ".l / m^3", ".m^3/.m^3", ".m^3 / .l", "% v / v", "v / v"

mass / mass: ".g / .g", "w / w", "% w / w"

Where "." symbolises a metric prefix (see `calc_factor_prefix()`) :

For g, l, mol and M: d (deci), c (centi), m (milli), µ (micro), n (nano), p (pico) and f (femto).

For g you might use k (kilo) as well.

For m^3 (cubic metres) you may only use: d (deci), c (centi) and m (milli).

Note: % w / v is (incorrectly) taken as a short hand for 0.1 g / l.

Value

The factor to convert A into B.

See Also

Other conversion functions: `calc_factor_prefix`, `convert_conc`, `convert_prefix`

Examples

```
library("dplyr")

# generate test data
data <- tibble(
  sample = c("A", "B", "C"),
  conc = c(4.5, 2.3, 5.1),       # concentration in g  / l
)

fctr_ng_ml <- calc_factor_conc(from = "g/l", to = "ng/ml")
# give molar mass in g / mol
fctr_mol_l <- calc_factor_conc(from = "g/l", to = "M", molar_mass = 78.971)
# give densities in g / l
fctr_pc <- calc_factor_conc(from = "g/l", to = "%v/v", density_solute = 4810)

data %>%
  mutate(
    conc_ng_ml = conc * fctr_ng_ml,
    conc_mol_l = conc * fctr_mol_l,
    conc_pc = conc * fctr_pc
)

# throws an error
## Not run:
# will throw an error because molar_mass is missing
fctr_fail <- calc_factor_conc(from = "g/l", to = "mol/l")

## End(Not run)
```

`calc_factor_prefix` *Get a factor to convert metric prefixes.*

Description

Get a factor to convert metric prefixes into one another.

Usage

```
calc_factor_prefix(from, to)
```

Arguments

`from`	A string containing the prefixed unit A.
`to`	A string containing the prefixed unit B.

Details

Convert, e.g. "kg" to "µg". You can convert ".g", ".l", ".mol", ".m^3" (cubic metres), where "." symbolises a metric prefix:

For g, l and mol: d (deci), c (centi), m (milli), µ (micro), n (nano), p (pico) and f (femto).

For g you might use k (kilo) as well.

For m^3 (cubic metres) you may only use: d (deci), c (centi) and m (milli).

Value

A factor for multiplication with the value.

See Also

Other conversion functions: `calc_factor_conc`, `convert_conc`, `convert_prefix`

Examples

```
calc_factor_prefix(from = "ng", to = "kg")
calc_factor_prefix(from = "dm^3", to = "cm^3")
calc_factor_prefix(from = "fl", to = "pl")
calc_factor_prefix(from = "pmol", to = "nmol")
```

---

convert_conc *Convert a value of the given concentration into another concentration.*

Description

A convenience wrapper around `calc_factor_conc()`.

Usage

```
convert_conc(x, from, to, molar_mass = 0, density_solute = 0,
  density_solution = 0)
```

Arguments

`x`	The value to convert.
`from`	A string containing the units of concentration A.
`to`	A string containing the units of concentration B.
`molar_mass`	The molar mass of the solute (g / mol).
`density_solute`	The density of the solute (g / l).
`density_solution`	The density of the solution (g / l), not the solvent!

Details

The following concentrations can be converted:

mass / volume: ".g / .l", ".g / .m^3", "% w / v"

molar / volume: ".M", ".mol / .l", ".mol / .m^3"

volume / volume: ".l/.l", ".l / m^3", ".m^3/.m^3", ".m^3 / .l", "% v / v", "v / v"

mass / mass: ".g / .g", "w / w", "% w / w"

Where "." symbolises a metric prefix (see `calc_factor_prefix()`) :

For g, l, mol and M: d (deci), c (centi), m (milli), µ (micro), n (nano), p (pico) and f (femto).

For g you might use k (kilo) as well.

For m^3 (cubic metres) you may only use: d (deci), c (centi) and m (milli).

Note: % w / v is (incorrectly) taken as a short hand for 0.1 g / l.

Value

The converted value.

See Also

Other conversion functions: `calc_factor_conc`, `calc_factor_prefix`, `convert_prefix`

Examples

```
library("dplyr")

# generate test data
data <- tibble(
  sample = c("A", "B", "C"),
  conc = c(4.5, 2.3, 5.1),       # concentration in g  / l
)

data %>%
  mutate(
    conc_ng_ml = convert_conc(x = conc, from = "g/l", to = "ng/ml"),
    # give molar mass in g / mol
    conc_mol_l = convert_conc(
      x = conc, from = "g/l", to = "M", molar_mass = 78.971),
    # give densities in g / l
    conc_pc = convert_conc(
      x = conc, from = "g/l", to = "%v/v", density_solute = 4810)
)

# throws an error
## Not run:
# will throw an error because molar_mass is missing
fail <- convert_conc(x = 5, from = "g/l", to = "mol/l")

## End(Not run)
```

convert_prefix *Convert between metric prefixes.*

Description

A convenience wrapper around `calc_factor_prefix()`.

Usage

```
convert_prefix(x, from, to)
```

Arguments

x	The value to convert.
from	A string containing the prefixed unit A.
to	A string containing the prefixed unit B.

Details

Convert, e.g. "kg" to "µg". You can convert ".g", ".l", ".mol", ".m^3" (cubic metres), where "." symbolises a metric prefix:

For g, l and mol: d (deci), c (centi), m (milli), µ (micro), n (nano), p (pico) and f (femto).

For g you might use k (kilo) as well.

For m^3 (cubic metres) you may only use: d (deci), c (centi) and m (milli).

Value

The converted value.

See Also

Other conversion functions: `calc_factor_conc`, `calc_factor_prefix`, `convert_conc`

Examples

```
convert_prefix(x = 2, from = "ng", to = "kg")
convert_prefix(x = 2, from = "dm^3", to = "cm^3")
convert_prefix(x = 2, from = "fl", to = "pl")
convert_prefix(x = 2, from = "pmol", to = "nmol")
```

`models_linear` *Linear model functions.*

Description

Use these functions to calculate a linear model from data, plot the model and use it to calculate x-values from the model data and y-values (inverse function).

Those function are intended to be used in set_calc_concentrations / sets_read to be applied to the calibrators (`fit_linear`) and interpolate concentrations from the raw values (`interpolate_linear`). Use `plot_linear` to visually insepct goodness of fit.

- `fit_linear`: Calculate a linear model from x and y.
- `plot_linear`: Draw the plot for the model that can be calculated with `fit_linear`. Uses ggplot2::ggplot if available.
- `interpolate_linear`: Inverse `fit_linear` using `model` and calculate x values from y values.

Usage

```
fit_linear(x, y)

plot_linear(x, y)

interpolate_linear(y, model)
```

Arguments

`x`	The x coordinates of the points.
`y`	The y coordinates of the points.
`model`	The line model.

Value

- `fit_linear`: The line model.
- `plot_linear`: The plot.
- `interpolate_linear`: The calculated x values.

See Also

set_calc_concentrations, sets_read, models_lnln

Examples

```
# generate data
x <- c(1, 3, 4, 7)
y_known <- c(3.5, 6.5, 8, 12.5)  # x is known for these values
y_unknown <- c(5, 9.5, 11)       # we will calculate x for those

model <- fit_linear(x = x, y = y_known)
model

plot_linear(x = x, y = y_known)

interpolate_linear(y = y_unknown, model)

rm(x, y_known, y_unknown, model)
```

`models_lnln` *Model functions for data requiring ln-ln-transformation to fit a model.*

Description

Use these functions to transform x and y using the natural logarithm and calculate a linear model, plot the model and use it to calculate x-values from the model data and y-values (inverse function).

Those function are intended to be used in set_calc_concentrations / sets_read to be applied to the calibrators (`fit_lnln`) and interpolate concentrations from the raw values (`interpolate_lnln`). Use `plot_lnln` to visually insepct goodness of fit.

- `fit_lnln`: Apply ln to x and y and calculate a linear model from x and y.
- `plot_lnln`: Draw the plot for the model that can be calculated with `fit_lnln`. Uses ggplot2::ggplot if available.
- `interpolate_lnln`: Inverse `fit_lnln` using `model` and calculate x values from y values.

Usage

```
fit_lnln(x, y)

plot_lnln(x, y)

interpolate_lnln(y, model)
```

Arguments

`x`	The x coordinates of the points.
`y`	The y coordinates of the points.
`model`	The line model.

Value

- `fit_lnln`: The model.
- `plot_lnln`: The plot.
- `interpolate_lnln`: The calculated x values.

See Also

set_calc_concentrations, sets_read, models_linear

Examples

```
# generate data
x <- c(2.718282, 20.085537, 54.598150, 1096.633158)
# x is known for these values
y_known <- c(33.11545, 665.14163, 2980.95799, 268337.28652)
# we will calculate x for those:
y_unknown <- c(148.4132, 13359.7268, 59874.1417)

model <- fit_lnln(x = x, y = y_known)
model

plot_lnln(x = x, y = y_known)

interpolate_lnln(y = y_unknown, model)

rm(x, y_known, y_unknown, model)
```

---

sets_read *Read sets and calculate concentrations and variability.*

Description

Basicaly a wrapper around `set_read()`, `set_calc_concentrations()` and `set_calc_variability()`.

For a gentler introduction see examples and Vignette "Introduction".

May write the processed data into two files: `data_samples.csv`, `data_all.csv`.

Usage

```
sets_read(sets, cal_names, cal_values, exclude_cals = list(),
  additional_vars = c("name"), additional_sep = "_", sep = ",",
  dec = ".", path = ".", file_name = "set_#NUM#.csv",
  model_func = fit_linear, plot_func = plot_linear,
  interpolate_func = interpolate_linear, write_data = TRUE,
  use_written_data = FALSE)
```

Arguments

sets	The number of sets (e.g. 3`` attempts to readset_1.csv,set_2.csv,set_3.csv), seefile_name‘.
cal_names	A vector of strings containing the names of the samples used as calibrators.
cal_values	A numeric vector with the known concentrations of those samples (must be in the same order).
exclude_cals	A list of calibrators to exclude, e.g.: list(set1 = c("CAL1")).
additional_vars	Vector of strings containing the names for the additional columns.
additional_sep	String / RegExp that separates additional vars, e.g.: "ID_blue_cold" with additional_sep = "_" will be separated into three columns containing "ID", "blue" and "cold". If the separated data would exceed the columns in additional_vars the last column will contain a string with separator (e.g.: "blue_cold"). If data is missing NA is inserted.
sep	Separator used in the csv-file, either "," or ";" (see utils::read.csv()).
dec	The character used for decimal points (see utils::read.csv()). "AUTO" will result in "." if sep is "," and "," for ";".
path	The path to the file (no trailing "/" or "\" !).
file_name	Name of the file from which to read the data. May contain "#NUM#" as a placeholder if you have multiple files.
model_func	A function generating a model to fit the calibrators, e.g. fit_linear(), fit_lnln().
plot_func	Function used to display the fitted line.
interpolate_func	A function used to interpolate the concentrations of the other samples, based on the model, e.g. interpolate_linear(), interpolate_lnln().
write_data	Write the calculated data into data_all.csv and data_samples.csv?
use_written_data	Try to read data_all.csv and data_read.csv instead of raw data. Useful if you have to re-run the script, but the raw data does not change.

Value

A list:

- $all: here you will find all the data , including calibrators, duplicates, ... (saved in data_all.csv if write_data = TRUE)
- $samples: only one row per distinct sample here - no calibrators, no duplicates -> most often you will work with this data (saved in data_samples.csv if write_data = TRUE)
- $set1: a list
  - $plot: a plot showing you the function used to calculate the concentrations for this set. The points represent the calibrators.
  - $model: the model as returned by model_func
- ($set2 - $setN): the same information for every set you have

See Also

Other set functions: set_calc_concentrations, set_calc_variability, set_read

Examples

```
# files "set_1.csv" and "set_2.csv" containing raw values and the
# corresponding lables (consisting of ID and point in time like
# "ID_TIME")
read.csv(
  file = system.file("extdata", "set_1.csv", package = "bioset"),
  header = FALSE,
  colClasses = "character"
)
read.csv(
  file = system.file("extdata", "set_2.csv", package = "bioset"),
  header = FALSE,
  colClasses = "character"
)

# the known concentration of the calibrators contained in these plates
cals <- c(10, 20, 30, 40)      # ng / ml
names(cals) <- c("CAL1", "CAL2", "CAL3", "CAL4")

# read both files into a tibble
# columns "ID" and "time" separated by "_"
# and calculate concentrations using the calibrators
result <- sets_read(
  sets = 2,                       # expect 2 plates
  path = system.file("extdata", package = "bioset"),
  additional_vars = c("ID", "time"),  # expect the labels to contain ID and
                                      # point in time
  additional_sep = "_",               # separated by "_"
  cal_names = names(cals),       # that's what they're called in the files
  cal_values = cals,             # the concentration has to be known
  write_data = FALSE             # do not store the results in csv-files
)

# inspect results (all values contained in the two original files)
result$all
# (all values except CAL1-4)
result$samples
# inspect goodness of fit
# for plate 1
result$set_1$plot
result$set_1$model
# for plate 2
result$set_2$plot
result$set_2$model
```

`set_calc_concentrations` *Calculate concentrations for the set using contained calibrators.*

Description

If the data set is generated, for example by reading extinction rates or relative light units from a plate, these raw values can be converted to concentrations using data fields with known concentrations (calibrators).

Usage

```
set_calc_concentrations(data, cal_names, cal_values, col_names = name,
  col_values = value, col_target = conc, col_real = real,
  col_recov = recovery, model_func = fit_linear,
  interpolate_func = interpolate_linear)
```

Arguments

`data`	A tibble containing the data.
`cal_names`	A vector of strings containing the names of the samples used as calibrators.
`cal_values`	A numeric vector with the known concentrations of those samples (must be in the same order).
`col_names`	The name of the column where the `cal_names` can be found.
`col_values`	The name of the column holding the raw values.
`col_target`	The name of the column to created for the calculated concentration.
`col_real`	The name of the column to create for the known concentrations.
`col_recov`	The name of the column to create for the recovery of the calibrators.
`model_func`	A function generating a model to fit the calibrators, e.g. `fit_linear()`, `fit_lnln()`.
`interpolate_func`	A function used to interpolate the concentrations of the other samples, based on the model, e.g. `interpolate_linear()`, `interpolate_lnln()`.

Details

If the data set contains samples with known concentrations (calibrators) those can be used to interpolate the concentrations of the other samples.

Value

A tibble containing all original and additional columns.

See Also

Other set functions: `set_calc_variability`, `set_read`, `sets_read`

Examples

```
# generate data
library("tibble")

data <- tibble(
  name = c("CAL1", "CAL2", "CAL3", "A", "B", "C"),
  value = c(1, 5, 10, 2, 4, 6)
)

data

# the known concentration of the calibrators
cals <- c(1, 5, 10)
names(cals) <- c("CAL1", "CAL2", "CAL3")

set_calc_concentrations(
  data = data,
  cal_names = names(cals),
  cal_values = cals
)

# to set column names use notation like in dplyr / tidyverse
# set the name of the column holding the final concentration to "my_protein"
set_calc_concentrations(
  data = data,
  cal_names = names(cals),
  cal_values = cals,
  col_target = my_protein
)

## Not run:
# notice that col_target is given a string
# this will fail
set_calc_concentrations(
  data = data,
  cal_names = names(cals),
  cal_values =  cals,
  col_target = "my_protein"
)

## End(Not run)

# simulate data which has to be transformed to get a good fit
cals <- exp(cals)
data$value <- exp(data$value)

# use ln-transformation on values and known concentrations prior to
# fitting a model

data <- set_calc_concentrations(
  data = data,
  cal_names = names(cals),
```

```
  cal_values = cals,
  model_func = fit_lnln,
  interpolate_func = interpolate_lnln
)

data

# inspect goodnes of fit
plot_lnln(data$real, data$value)

rm(cals, data)
```

`set_calc_variability` *Calculate parameters of variability for a given set of values.*

Description

Calculate mean, standard deviation and coefficient of variation for groups of values.

Usage

```
set_calc_variability(data, ids, ...)
```

Arguments

`data`	A tibble containing the data.
`ids`	The column holding the names used to group the values.
`...`	The name(s) of the columns used to calculate the variability.

Details

Dealing with measured values, the measurement of sample "A" is often done in duplicates / triplicates / ... . This function groups all samples with the same name and calculates mean, standard deviation and coefficient of variation (= sd / mean).

Value

A tibble containing all original and additional columns (NAMEA_mean, NAMEA_n, NAMEA_sd, NAMEA_cv, (NAMEB_mean, ...)).

See Also

Other set functions: `set_calc_concentrations`, `set_read`, `sets_read`

Examples

```
# generate data
library("tibble")

data <- tibble(
  names = c("A", "B", "C", "A", "B", "C"),
  value = c(19, 59, 22, 18, 63, 28),
  conc = c(1.9, 5.9, 2.2, 1.8, 6.3, 2.8)
)

data

set_calc_variability(
  data = data,
  ids = names,
  value,
  conc
)

# to set column names use notation like in dplyr / tidyverse
## Not run:
# notice how strings are given as column names
set_calc_variability(
  data = data,
  ids = "names",
  "value",
  "conc"
)

## End(Not run)

rm(cals)
```

`set_read` *Read a data set from a data-sheet and turn it into a multi-column tibble.*

Description

Read a matrix of values from a csv sheet and sort them into a tibble. You can name the values and encode several additional properties into the name, which be split into several columns. Please refer to the vignette (`browseVignettes("roxygen2")`) and examples below for in-depth explanation and the whys and hows.

Usage

```
set_read(file_name = "set_#NUM#.csv", path = ".", num = 1, sep = ",",
  dec = ".", cols = 0, rows = 0, additional_vars = vector(),
  additional_sep = "[^[:alnum:]]+")
```

Arguments

file_name	Name of the file from which to read the data. May contain "#NUM#" as a placeholder if you have multiple files.
path	The path to the file (no trailing "/" or "\" !).
num	Number of the set to read, inserted for "#NUM#".
sep	Separator used in the csv-file, either "," or ";" (see utils::read.csv()).
dec	The character used for decimal points (see utils::read.csv()). "AUTO" will result in "." if sep is "," and "," for ";".
cols	Number of columns in the input matrix (0 means auto-detect).
rows	Number of rows containing values (not names / additional data) in the input matrix (0 means auto-detect).
additional_vars	Vector of strings containing the names for the additional columns.
additional_sep	String / RegExp that separates additional vars, e.g.: "ID_blue_cold" with additional_sep = "_" will be separated into three columns containing "ID", "blue" and "cold". If the separated data would exceed the columns in additional_vars the last column will contain a string with separator (e.g.: "blue_cold"). If data is missing NA is inserted.

Value

A tibble containing (at minimum) set, position, sample_id, name and value.

See Also

Other set functions: set_calc_concentrations, set_calc_variability, sets_read

Examples

```
# a file containing only values
read.csv(
  file = system.file("extdata", "values.csv", package = "bioset"),
  header = FALSE,
  colClasses = "character"
)

# read into a tibble
set_read(
  file_name = "values_names.csv",
  path = system.file("extdata", package = "bioset"),
)

# file containing names
read.csv(
  file = system.file("extdata", "values_names.csv", package = "bioset"),
  header = FALSE,
  colClasses = "character"
)
```

```
# read a file containing labels and store those in column "name"
set_read(
  file_name = "values_names.csv",
  path = system.file("extdata", package = "bioset"),
  additional_vars = c("name")
)

# file with names and properties
read.csv(
  file = system.file(
    "extdata", "values_names_properties.csv", package = "bioset"),
  header = FALSE,
  colClasses = "character"
)

# read a file containing labels and properties and store those in columns
# "name" and "time"
# splits names by every character that's not A-Z, a-z, 0-9
# to change that behaviour use additional_sep
set_read(
  file_name = "values_names_properties.csv",
  path = system.file("extdata", package = "bioset"),
  additional_vars = c("name", "time")
)

# read file "set_1.csv" containing labels
set_read(
  num = 1,
  path = system.file("extdata", package = "bioset"),
  additional_vars = c("name", "time")
)

# read file "set_2.csv" containing labels
set_read(
  num = 2,
  path = system.file("extdata", package = "bioset"),
  additional_vars = c("name", "time")
)

# read file "plate_2.csv" containing labels
set_read(
  num = 2,
  file_name = "plate_#NUM#.csv",
  path = system.file("extdata", package = "bioset"),
  additional_vars = c("name", "time")
)
```

Index