Package: epireview (via r-universe)

February 17, 2025

Title Tools to update and summarise the latest pathogen data from the Pathogen Epidemiology Review Group (PERG)

Version 1.4.3

Description Contains the latest open access pathogen data from the Pathogen Epidemiology Review Group (PERG). Tools are available to update pathogen databases with new peer-reviewed data as it becomes available, and to summarise the latest data using tables and figures.

License MIT + file LICENSE

Depends epitrix, ggplot2, ggforce, R (>= 4.1.0)

Imports dplyr, readr, scales, cli, vroom

Suggests knitr, lifecycle, tidyverse, testthat (>= 3.0.0), rmarkdown

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.2

VignetteBuilder knitr

Config/testthat/edition 3

URL https://mrc-ide.github.io/epireview/

LazyData true

BugReports https://github.com/mrc-ide/epireview/issues

Config/pak/sysreqs libfontconfig1-dev libfreetype6-dev libicu-dev libsodium-dev libx11-dev

Repository https://mrc-ide.r-universe.dev

RemoteUrl https://github.com/mrc-ide/epireview

RemoteRef main

RemoteSha 5c9671a605c7ade26dd8522687e63b545d534c82

Contents

`article_column_type` *Define the column types for the article data frame*

Description

This function defines the column types for the article data frame used in the epireview package. vroom is generally good at guessing the column types, but it is better to be explicit. Moreover, it reads a column of NAs as a logical vector, which is particularly undesirable for us. This function is intended to be used internally by `load_epidata_raw` where the files are being read.

Usage

```
article_column_type(pathogen)
```

Arguments

`pathogen`	name of pathogen. This argument is case-insensitive. Must be one of the priority pathogens You can get a list of the priority pathogens currently included in the package by calling the function `priority_pathogens`.

Value

A list of column types for the article data frame

See Also

parameter_column_type, outbreak_column_type, model_column_type

---

`assign_qa_score` *Assign quality assessment score to each article*

Description

Assign quality assessment score to each article

Usage

```
assign_qa_score(articles, ignore_errors = FALSE)
```

Arguments

articles	data.frame loaded from load_epidata function
ignore_errors	logical; if TRUE, the function will assign QA scores where possible (i.e. where all answers to quality assessment questions are not NA) and set the QA score to NAfor articles where all answers are NA. If FALSE, an error is thrown instead.

Details

We have used a bespoke 7 question quality assessment (QA) questionnaire to assess the quality of articles. The questions can be retrieved using the qa_questions function. The function assigns a QA score to each article as the number of questions answered 'yes' divided by the total number of questions answered (an answer might be NA if the question is not relevant to the article under consideration). Articles with all NA answers are excluded from the QA unless ignore_errors is set to TRUE.

Value

a named list consisting of two elements. The first element of the list is the article data.frame with an updated column containing three new columns: qs_denominator (total number of questions answered), qs_numerator (number of questions answered 'yes') and qa_score (QA score). The second element of the list (named errors) is a data.frame containing articles with all NA answers.

See Also

qa_questions

Examples

```
lassa <- load_epidata("lassa")
lassa_qa <- assign_qa_score(lassa$articles, ignore_errors = FALSE)
head(lassa_qa$articles[, c("qa_denominator", "qa_numerator", "qa_score")])
```

---

check_column_types *Checks that the column types of the input csv matches the column types expected by epireview.*

Description

This function creates a vroom object (the same type created by read_csv) and checks if there are any problems with the file. If there are it will provide the offending columns and the number problematic rows (per column). A csv with the details of the issue will be written to a tmp file and the location will be provided. This function will prevent data from being loaded until all column types are correct.

Usage

```
check_column_types(fname, col_types, raw_colnames)
```

Arguments

fname	The name of the csv file for which the column types are to be checked
col_types	The column types expected by epireview. These are specified in the column type functions (e.g., article_column_type, parameter_column_type) and are used to read in the data.
raw_colnames	The column names of the csv file

Details

The function is intended to be used internally by `load_epidata_raw` where the files are being read.

See Also

article_column_type parameter_column_type, outbreak_column_type, model_column_type

---

check_df_for_meta *Sanity checks before meta-analysis*

Description

Sanity checks before meta-analysis

Usage

```
check_df_for_meta(df, cols_needed)
```

Arguments

df	a parameter dataframe. This must have columns for each of the following: parameter_value, parameter_unit, plus two columns for the numerator and the denominator of the proportion of interest. This dataframe will typically be the `params` data.frame from the output of `load_epidata`.
cols_needed	a character vector specifying the names of the columns required for the meta-analysis.

Details

The function carries out a series of checks on the parameter dataframe to ensure that it is in the correct format for conducting a meta-analysis. It checks that the input (1) consists of a single parameter type; (2) has the columns required for the meta-analysis; (3) does not have any row where a value is present but the unit is missing, or vice versa. All such rows are removed; and (4) has the same unit across all values of the parameter. The function will throw an error if either there is more than one value of parameter_type, or if the columns needed for the meta-analysis are missing, or if the parameter_unit is not the same across all values of the parameter.

Value

a parameter dataframe with offending rows removed.

`check_ulim` *Check upper limit of parameter values*

Description

Check upper limit of parameter values

Usage

```
check_ulim(df, ulim, param)
```

Arguments

`df`	A data frame containing the parameter values.
`ulim`	The specified upper limit for the parameter values.
`param`	The name of the parameter.

Details

Each forest plot has a default upper limit for the parameter values, based on the expected range of the parameter. This function checks the upper limit of parameter values in a data frame and compares it with the default limit. If the maximum parameter value exceeds the specified upper limit, a warning message is displayed. The function also returns the suggested upper limit. It is used internally by the forest plot functions (only for warning the user). You can also use it directly to update the default upper limit in the forest plot functions.

Value

The suggested upper limit for the parameter values.

Examples

```
df <- data.frame(parameter_value = c(10, 20, 30))
check_ulim(df, 25, "parameter")
```

`color_palette` *Define a consistent color palette for use in figures. Palettes are currently defined for parameters and countries. Any other variable will return NULL*

Description

Define a consistent color palette for use in figures. Palettes are currently defined for parameters and countries. Any other variable will return NULL

Usage

```
color_palette(col_by = c("parameter_type", "population_country"), ...)
```

Arguments

col_by	a character vector specifying the parameter to color the palette by.
...	additional arguments to be passed to the underlying palette function. These are treated as names of the palette elements.

Value

a named list of colors that can be used in forest plots for manually setting colors

Examples

```
color_palette("parameter_type")
```

---

country_palette *country_palette Function*

Description

This function returns a color palette for countries.

Usage

```
country_palette(x = NULL)
```

Arguments

x	A vector of country names. If provided, the function will return a color palette for the specified countries.

Value

A color palette for the specified countries.

Examples

```
# Get color palette for all countries
country_palette()
#' # Get color palette for specific countries
country_palette(c("Liberia", "Guinea", "Sierra Leone"))

country_palette()
```

`custom_palette` *Create a custom color palette.*

Description

This utility function creates a named color vector from user-supplied vectors of labels and color values. The length of the label and color vectors must be the same. The resulting custom color palette can be used as the color palette in other plotting functions.

Usage

```
custom_palette(labels, colors)
```

Arguments

`labels`	A vector of labels to be used as names for the custom color palette.
`colors`	A vector of colors to be used for the custom color palette. This can be in the form of HEX codes, e.g., "#808080" or color names recognized by R, eg "deepskyblue"

Value

A custom palette in the form of a named color vector.

Examples

```
labels <- c("Liberia", "Guinea", "Sierra Leone")
colors <- c("#5A5156FF", "#E4E1E3FF", "#5050FFFF")

custom_pal <- custom_palette(labels, colors)
custom_pal
```

`delays_to_days` *This function converts delays in different units (hours, weeks, months) to days. It checks if all delays are in days and warns the user if not. It then converts hours to days by dividing by 24, weeks to days by multiplying by 7, and months to days by multiplying by 30.*

Description

This function converts delays in different units (hours, weeks, months) to days. It checks if all delays are in days and warns the user if not. It then converts hours to days by dividing by 24, weeks to days by multiplying by 7, and months to days by multiplying by 30.

Usage

```
delays_to_days(df)
```

Arguments

df	A data.frame containing delays and their units. This will typically be a subset of parameters from the data frame returned by load_epidata.

Value

Updated data.frame with delays converted to days.

Examples

```
df <- data.frame(
  parameter_value = c(24, 7, 1),
  parameter_unit = c("Hours", "Weeks", "Months")
)
delays_to_days(df)
# Output:
#   parameter_value parameter_unit
# 1               1           Days
# 2              49           Days
# 3              30           Days
```

---

epireview_read_file *Intended for internal use only; setting desirable defaults for reading in data files.*

Description

Intended for internal use only; setting desirable defaults for reading in data files.

Usage

```
epireview_read_file(fname, ...)
```

Arguments

fname	The name of the file to be read in.
...	Additional arguments to be passed to read_delim.

Details

This function is intended for internal use only. It is used to read in data files shipped with the package. The idea is to set desirable defaults in a single place. Mostly it makes reading files quiet by setting show_col_types to FALSE.

Value

A data frame read in from the file.

Author(s)

Sangeeta Bhatia

`filter_cols` *Filter columns of a data frame based on specified conditions.*

Description

This function filters the rows of a data frame based on specified conditions for selected columns.

Usage

```
filter_cols(x, cols, funs = c("in", "==", ">", "<"), vals)
```

Arguments

`x`	A data frame.
`cols`	A character vector specifying the columns to be filtered.
`funs`	A character vector specifying the filter functions for each column. Each function must be one of "in", "==", ">", "<" in quotes.
`vals`	A list of values to be used for filtering columns in `cols`.

Value

A data frame with rows filtered based on the specified conditions.

Examples

```
x <- load_epidata("marburg")
p <- x$params
filter_cols(p, "parameter_type", "==", "Attack rate")
filter_cols(
  p, "parameter_type", "in",
  list(parameter_type = c("Attack rate", "Seroprevalence - IFA"))
)
```

filter_df_for_metamean *Prepare parameter dataframe for meta analysis of means*

Description

Prepare parameter dataframe for meta analysis of means

Usage

```
filter_df_for_metamean(df)
```

Arguments

df	a parameter dataframe. This must have columns for each of the following: parameter_value, parameter_unit, population_sample_size, parameter_value_type, parameter_uncertainty_singe_type, parameter_uncertainty_type, parameter_uncertainty_lower_value, parameter_uncertainty_upper_value. This will typically be the params data.frame from the output of load_epidata.

Details

The function checks that the format of df is adequate for conducting a meta analysis of means. It filters the dataframe to only include rows that meet the required format. We can only conduct a meta analysis for a parameter if its estimates have been reported as (a) mean and standard deviation, (b) median and interquartile range, or (c) median and range. This function filters the parameter dataframe to only include rows that meet these criteria. It also checks that the parameter values are all in the same units; and that the sample size is reported for each parameter value.

Value

a parameter dataframe with relevant rows selected and additional columns added to facilitate the meta analysis of means. The additional columns are: xbar, median, q1, q3, min, max.

Examples

```
## preparing data for meta analyses of delay from symptom onset to
## hospitalisation for Lassa

df <- load_epidata("lassa")[["params"]]
o2h_df <- df[df$parameter_type %in% "Human delay - symptom onset>admission to care", ]
o2h_df_filtered <- filter_df_for_metamean(o2h_df)
## o2h_df_filtered could then be used directly in meta analyses as:
## mtan <- metamean(data = o2h_df_filtered, ...)
```

`filter_df_for_metaprop` *Prepare parameter dataframe for meta analysis of proportions*

Description

Prepare parameter dataframe for meta analysis of proportions

Usage

```
filter_df_for_metaprop(df, num_col, denom_col)
```

Arguments

`df`	a parameter dataframe. This must have columns for each of the following: parameter_value, parameter_unit, plus two columns for the numerator and the denominator of the proportion of interest. This dataframe will typically be the `params` data.frame from the output of `load_epidata`.
`num_col`	a string specifying the column name for the column containing the numerator of the proportion of interest.
`denom_col`	a string specifying the column name for the column containing the denominator of the proportion of interest.

Details

The function checks that the format of df is adequate for conducting a meta analysis of proportions. It filters the dataframe to only include rows that meet the required format by (1) removing rows where the denominator is missing, and (2) removing rows where both the numerator column or parameter value are missing. If the numerator column is missing and the parameter value is present, the numerator is imputed as the parameter value divided by 100 times the denominator.

Value

a parameter dataframe with relevant rows selected to enable meta analysis of proportions.

Examples

```
## preparing data for meta analyses of CFR for Lassa

df <- load_epidata("lassa")[["params"]]
cfr_df <- df[df$parameter_type %in% "Severity - case fatality rate (CFR)", ]
cfr_filtered <- filter_df_for_metaprop(cfr_df,
  num_col = "cfr_ifr_numerator", denom_col = "cfr_ifr_denominator"
)
## cfr_filtered could then be used directly in meta analyses as:
## mtan <- metaprop(data = cfr_filtered, ...)
```

`forest_plot` *Basic forest plot*

Description

Basic forest plot displays central estimate and uncertainty for a parameter from different studies. y-axis lists the study labels and the x-axis displays parameter.

Usage

```
forest_plot(
  df,
  facet_by = NA,
  shape_by = NA,
  col_by = NA,
  shp_palette = NULL,
  col_palette = NULL,
  unique_label = NA
)
```

Arguments

`df`	The data frame containing the data for the forest plot. data.frame with the following fields: article, label, mid, low, high The field 'y' is mapped to the y-axis with 'article_label' used as a display label. mid refers to the central estimate. low and high represent the lower and higher ends of the uncertainty interval
`facet_by`	(Optional) Variable to facet the plot by.
`shape_by`	(Optional) Variable to shape the points by.
`col_by`	(Optional) Variable to color the points by.
`shp_palette`	(Optional) Palette for shaping the points. Optional unless shape_by is not one of 'parameter_value_type'.
`col_palette`	Palette for coloring the points. Optional unless col_by is not one of 'parameter_type' or 'population_country'.
`unique_label`	(Optional) User can provide custom labels for forest plot y-axis. Must match length of dataframe.

Details

Generates a forest plot.

This function generates a forest plot using the provided data frame.

epireview provides a default palette for parameters and countries. If you wish to color by a different variable, you must provide a palette.

Value

A ggplot2 object representing the forest plot.

Examples

```
df <- data.frame(
  mid = c(1, 2, 3),
  article_label = c("A", "B", "C"),
  low = c(0.5, 1.5, 2.5),
  high = c(1.5, 2.5, 3.5),
  uncertainty_type = c("Range", "Range", "Range**")
)
forest_plot(df)
```

---

forest_plot_delay_int *Create forest plot for human delays*

Description

Create forest plot for human delays

Usage

```
forest_plot_delay_int(df, ulim, reorder_studies, ...)
```

Arguments

df	The data frame containing the necessary data for generating the forest plot.
ulim	The upper limit for the x-axis of the plot. Default is 10.
reorder_studies	Logical. If TRUE, the studies will be reordered using the reorder_studies function. Default is TRUE.
...	Additional arguments to be passed to the forest_plot function.

Details

There are a number of 'delays' that are relevant to the pathogens we study. Some of the more commonly, and hence likely to be extracted for most pathogens, are infectious period, incubation period, and serial interval. However, there are many others reported by relatively few studies or relevant to only a few pathogens. This function is intended to serve as a template for creating forest plots for these delays. It will first reparameterise any delays reported in terms of the gamma distribution to the mean and standard deviation (see reparam_gamma). Then, any parameters reported as inverse (e.g., per day instead of days) will be inverted (see invert_inverse_params). Finally, all delays will be converted to days (see delays_to_days) before reordering the studies (if requested) and plotting the forest plot. We also provide some utility functions for the commonly used delays that are simply wrapper for this function. If however you are interested in some other delay, you will need to use this function directly, ensuring that the data frame you provide has only the relevant delays.

Value

returns plot with a summary of the human delays

See Also

forest_plot_serial_interval

`forest_plot_doubling_time` *Forest plot for doubling time*

Description

This function calculates the doubling time and creates a forest plot to visualize the results.

Usage

```
forest_plot_doubling_time(df, ulim = 30, reorder_studies = TRUE, ...)
```

Arguments

`df`	The data frame containing the necessary data for generating the forest plot.
`ulim`	The upper limit for the x-axis of the plot. Default is 10.
`reorder_studies`	Logical. If TRUE, the studies will be reordered using the `reorder_studies` function. Default is TRUE.
`...`	Additional arguments to be passed to the `forest_plot` function.

Value

A forest plot object.

Examples

```
df <- load_epidata("ebola")[["params"]]
forest_plot_doubling_time(df, ulim = 20, reorder_studies = TRUE)
```

`forest_plot_incubation_period` *Create forest plot for incubation period*

Description

Create forest plot for incubation period

Usage

```
forest_plot_incubation_period(df, ulim = 30, reorder_studies = TRUE, ...)
```

Arguments

df	The data frame containing the necessary data for generating the forest plot.
ulim	The upper limit for the x-axis of the plot. Default is 10.
reorder_studies	Logical. If TRUE, the studies will be reordered using the reorder_studies function. Default is TRUE.
...	Additional arguments to be passed to the forest_plot function.

Details

This function is a wrapper for forest_plot_delay_int that is specifically for the incubation period.

---

forest_plot_infectious_period

Create forest plot for infectious period

Description

Create forest plot for infectious period

Usage

```
forest_plot_infectious_period(df, ulim = 30, reorder_studies = TRUE, ...)
```

Arguments

df	The data frame containing the necessary data for generating the forest plot.
ulim	The upper limit for the x-axis of the plot. Default is 10.
reorder_studies	Logical. If TRUE, the studies will be reordered using the reorder_studies function. Default is TRUE.
...	Additional arguments to be passed to the forest_plot function.

Details

This function is a wrapper for forest_plot_delay_int that is specifically for the infectious period.

`forest_plot_r0` *forest_plot_r0 function*

Description

This function generates a forest plot for the reproduction number (Basic R0) using the provided data frame.

Usage

```
forest_plot_r0(df, ulim = 10, reorder_studies = TRUE, ...)
```

Arguments

`df`	The data frame containing the necessary data for generating the forest plot.
`ulim`	The upper limit for the x-axis of the plot. Default is 10.
`reorder_studies`	Logical. If TRUE, the studies will be reordered using the `reorder_studies` function. Default is TRUE.
`...`	Arguments passed on to `forest_plot_rt`

Value

ggplot2 object.

Examples

```
df <- load_epidata("ebola")[["params"]]
forest_plot_r0(df, ulim = 2.5, reorder_studies = TRUE)
```

`forest_plot_rt` *Generate a forest plot for effective reproduction number (Rt)*

Description

This function generates a forest plot for the effective reproduction number (Rt) using the provided data frame.

Usage

```
forest_plot_rt(df, ulim = 10, reorder_studies = TRUE, ...)
```

Arguments

df	The data frame containing the necessary data for generating the forest plot.
ulim	The upper limit for the x-axis of the plot. Default is 10.
reorder_studies	Logical. If TRUE, the studies will be reordered using the reorder_studies function. Default is TRUE.
...	Additional arguments to be passed to the forest_plot function.

Value

A ggplot2 object representing the forest plot for effective reproduction number (Rt).

Examples

```
df <- load_epidata("ebola")[["params"]]
forest_plot_rt(df)
```

---

forest_plot_serial_interval

Create forest plot for serial interval

Description

Create forest plot for serial interval

Usage

```
forest_plot_serial_interval(df, ulim = 30, reorder_studies = TRUE, ...)
```

Arguments

df	The data frame containing the necessary data for generating the forest plot.
ulim	The upper limit for the x-axis of the plot. Default is 10.
reorder_studies	Logical. If TRUE, the studies will be reordered using the reorder_studies function. Default is TRUE.
...	Additional arguments to be passed to the forest_plot function.

Details

This function is a wrapper for forest_plot_delay_int that is specifically for the serial interval.

get_key_columns *Subset the epidemiological parameter columns by parameter type*

Description

Subset the epidemiological parameter columns by parameter type

Usage

```
get_key_columns(
  data,
 parameter_name = c("cfr", "delays", "sero", "risk_factors", "reproduction_number",
    "genomic", "attack_rate", "doubling_time", "growth_rate", "overdispersion",
    "relative_contribution"),
  all_columns = FALSE
)
```

Arguments

data	The parameter data.frame ($param) from load_epidata().
parameter_name	A character string with the parameter name. Options are: "cfr", "delay", "sero", "risk", "reproduction_number", and "genomic".
all_columns	The default is FALSE meaning that only the key columns specified for the specific parameter will be retrieved. If TRUE, then all columns in the data.frame will be retrieved.

Value

A data.frame with the key columns for the selected parameter.

Examples

```
lassa_data <- load_epidata("lassa")
lassa_params <- lassa_data$params
cfr_lassa <- get_parameter(
  data = lassa_params,
  parameter_name = "Severity - case fatality rate (CFR)"
)
get_key_columns(data = cfr_lassa, parameter_name = "cfr")
```

get_parameter *retrieve all parameters of specified type or class*

Description

retrieve all parameters of specified type or class

Usage

```
get_parameter(data, parameter_name)
```

Arguments

data	parameter dataframe output from load_epidata
parameter_name	name of the parameter type or parameter class to retrieve, ensuring the name matches that in data

Value

dataframe with all parameter estimates and columns

Examples

```
df <- load_epidata(pathogen = "ebola")
get_parameter(data = df$params, parameter_name = "Human delay - serial interval")
df <- load_epidata(pathogen = "marburg")
get_parameter(data = df$params, parameter_name = "Attack rate")
```

get_specific *Retrieve all incubation period parameters for a given pathogen*

Description

Retrieve all incubation period parameters for a given pathogen

Retrieve all serial interval estimates for a given pathogen

Retrieve all generation time estimates for a given pathogen

Retrieve all delay parameters for a given pathogen

Retrieve all CFR parameters for a given pathogen

Retrieve all risk factor parameters for a given pathogen

Retrieve all genomic parameters for a given pathogen

Retrieve all reproduction number parameters for a given pathogen

Retrieve all seroprevalence parameters for a given pathogen

Retrieve all doubling time parameters for a given pathogen

Retrieve all attack rate parameters for a given pathogen

Retrieve all growth rate parameters for a given pathogen

Retrieve all overdispersion parameters for a given pathogen

Retrieve all overdispersion parameters for a given pathogen

Usage

```
get_incubation_period(data, all_columns)

get_serial_interval(data, all_columns)

get_generation_time(data, all_columns)

get_delays(data, all_columns)

get_cfr(data, all_columns)

get_risk_factors(data, all_columns)

get_genomic(data, all_columns)

get_reproduction_number(data, all_columns)

get_seroprevalence(data, all_columns)

get_doubling_time(data, all_columns)

get_attack_rate(data, all_columns)

get_growth_rate(data, all_columns)

get_overdispersion(data, all_columns)

get_relative_contribution(data, all_columns)
```

Arguments

data	parameter dataframe output from load_epidata
all_columns	The default is FALSE meaning that only the key columns specified for the specific parameter will be retrieved. If TRUE, then all columns in the data.frame will be retrieved.

Value

dataframe with all parameter estimates of this type and key columns (see get_key_columns)

Examples

```
df <- load_epidata(pathogen = "lassa")
get_incubation_period(data = df$params, all_columns = FALSE)
```

`invert_inverse_params` *Inverts the values of selected parameters in a data frame. Sometimes parameters are reported in inverse form (e.g., a delay might be reported as per day instead of days). Here we carry out a very simple transformation to convert these to the correct form by inverting the parameter value and the uncertainty bounds. This may not be appropriate in all cases, and must be checked on a case-by-case basis. This function takes a data frame as input and inverts the values of selected parameters. The selected parameters are identified by the column 'inverse_param'. The function performs the following operations:*

- *Inverts the parameter values of the selected parameters.*
- *Swaps the upper and lower bounds of the selected parameters.*
- *Inverts the uncertainty values of the selected parameters.*
- *Updates the logical vector to indicate that the parameters are no longer inverted.*
- *Does not change the unit of the parameters, as it remains the same as the original parameter.*

Description

Inverts the values of selected parameters in a data frame. Sometimes parameters are reported in inverse form (e.g., a delay might be reported as per day instead of days). Here we carry out a very simple transformation to convert these to the correct form by inverting the parameter value and the uncertainty bounds. This may not be appropriate in all cases, and must be checked on a case-by-case basis. This function takes a data frame as input and inverts the values of selected parameters. The selected parameters are identified by the column 'inverse_param'. The function performs the following operations:

- Inverts the parameter values of the selected parameters.
- Swaps the upper and lower bounds of the selected parameters.
- Inverts the uncertainty values of the selected parameters.
- Updates the logical vector to indicate that the parameters are no longer inverted.
- Does not change the unit of the parameters, as it remains the same as the original parameter.

Usage

```
invert_inverse_params(df)
```

Arguments

`df`	A data frame containing the parameters to be inverted.

Value

The input data frame with the selected parameters inverted.

Examples

```
df <- data.frame(
  parameter_value = c(2, 3, 4),
  parameter_upper_bound = c(5, 6, 7),
  parameter_lower_bound = c(1, 2, 3),
  parameter_uncertainty_upper_value = c(0.1, 0.2, 0.3),
  parameter_uncertainty_lower_value = c(0.4, 0.5, 0.6),
  inverse_param = c(FALSE, TRUE, FALSE)
)
invert_inverse_params(df)
# Output:
```

---

load_epidata *Retrieve pathogen-specific data*

Description

Retrieve pathogen-specific data

Usage

```
load_epidata(pathogen, mark_multiple = TRUE)
```

Arguments

pathogen	name of pathogen. This argument is case-insensitive. Must be one of the priority pathogens You can get a list of the priority pathogens currently included in the package by calling the function priority_pathogens.
mark_multiple	logical. If TRUE, multiple studies from the same author in the same year will be marked with an numeric suffix to distinguish them. See mark_multiple_estimates for more details.

Details

The data extracted in the systematic review has been stored in four files - one each for articles, parameters, outbreaks, and transmission models. Data in these files can be linked using article identifier. This function will read in the pathogen-specific files and join them into a data.frame. This function also creates user-friendly short labels for the "parameter_type" column in params data.frame. See short_parameter_type for more details.

Value

a list of length 4. The first element is a data.frame called "articles" which contains all of the information about the articles extracted for this pathogen. The second element is a data.frame called "params" with articles information (authors, publication year, doi) combined with the parameters. The third element is a data.frame called "models" with all transmission models extracted for this pathogen including articles information as above. The fourth element is a data.frame called "outbreaks" which contains all of the outbreaks extracted for this pathogen, where available. If no data is available for a particular table, the corresponding element in the list will be NULL.

`load_epidata_raw` *Loads raw data for a particular pathogen*

Description

Loads raw data for a particular pathogen

Usage

```
load_epidata_raw(
  pathogen,
  table = c("article", "parameter", "outbreak", "model", "param_name")
)
```

Arguments

`pathogen`	name of pathogen. This argument is case-insensitive. Must be one of the priority pathogens You can get a list of the priority pathogens currently included in the package by calling the function `priority_pathogens`.
`table`	the table to be loaded. Must be one of "article", "parameter", "outbreak", "model" or "param_name"

Details

This function will return the raw data as a data.frame. The csv files of the models, outbreaks, and parameters for a pathogen do not contain information on the source but only an "article_id" that can be used to merge them with the articles. If you wish to retrieve linked information or multiple tables at the same time, use `load_epidata` instead.

Value

data.frame reading in the csv the specified pathogen table

See Also

`load_epidata()` for a more user-friendly interface

Examples

```
load_epidata_raw(pathogen = "marburg", table = "outbreak")
```

`make_unique_id` *Ensure that each article gets a unique id across all tables*

Description

Ensure that each article gets a unique id across all tables

Usage

```
make_unique_id(articles, df, df_name)
```

Arguments

`articles`	A dataframe with the articles table
`df`	A dataframe with the table that needs to be checked for duplicate covidence ids. Both articles and df will be loaded through `load_epidata`.
`df_name`	A the name of the df that will be loaded through `load_epidata`.

Details

In some instances, an article is associated with more than one id in the parameters, models, or outbreaks tables. This can lead to unexpected failures because we use the id to join the articles with other dataframes. This function will resolve the issue by first checking if a covidence id is mapped to more than one id. If it is, we replace one of the two ensuring that the same id is used across articles, models, outbreaks, and parameters. This function is not expected to be used directly by the user, but is called by `load_epidata`. Hence checks on arguments are not implemented.

Value

A dataframe with the same structure as df, but with unique ids for each covidence id.

Need for article ids

Why do we need article ids in the first place? Why not use covidence id? To ease the process of data extraction, we created a separate database for each extractor, with the goal of then merging the databases into a single database. Within each individual database, the different dataframes are linked by Access generated primary keys. These keys are unique to each database, but are not unique across databases. To merge the databases, we therefore generate a unique id for each article using `random_id`. Note that we cannot use covidence id for merging tables within each database because covidence id is only present in the articles table.

Why does an article end up with multiple ids?

Articles that have been extracted by two extractors will have multiple ids. In principle, this should not be a problem because as extractors resolve differences in data extracted, they generate a consensus entry by deleting one of the entries so that only one of the two ids should enter back into the database when the resolved entries are merged with the rest of the data. However, in practice, while resolving conflicts for multiple parameters or models, extractors may delete the row with Id1 in one case and the row with Id2 in another case. This can lead to the same article having multiple ids in parameters, models, or outbreaks. Because an article that has been double extracted will have been aasigned two ids, it is also possible that the retained id in articles is not the same as the retained id in parameters, models, or outbreaks. This can lead to rows in parameters, models, or outbreaks that are not linked to any article.

`marburg_article` *Data on the articles identified and included in the systematic review of articles related to Marburg virus disease.*

Description

This data set provides the details of all extracted articles included in the systematic review for Marburg virus disease (MVD).

Format

A csv file containing the following parameters:

- "article_id" = ID variable for the article.
- "pathogen" = pathogen name.
- "covidence_id" = article identifier used by the Imperial team.
- "first_author_first_name" = first author first name or initials.
- "article_title" = title of article.
- "doi" = the doi of the article.
- "journal" = journal that the article is published in.
- "year_publication" = year of article publication.
- "volume" = journal volume.
- "issue" = journal issue.
- "page_first" = first page number.
- "page_last" = last page number.
- "paper_copy_only" = if the article is not available online.
- "notes" = notes the extractor has made.
- "first_author_surname" = surname of the first author.
- "double_extracted" = either single extracted (0) or double extracted (1).

- "qa_m1" = quality assessment: "Is the methodological/statistical approach suitable? Q1. Clear and reproducible?" (either 0, 1, or NA)
- "qa_m2" = quality assessment: "Is the methodological/statistical approach suitable? Q2. Robust and appropriate for the aim?" (either 0, 1, or NA)
- "qa_a3" = quality assessment: "Are the assumptions appropriate? Q3. Clear and reproducible?" (either 0, 1, or NA)
- "qa_a4" = quality assessment: "Are the assumptions appropriate? Q4. Justified?" (either 0, 1, or NA)
- "qa_d5" = quality assessment: "Are the data appropriate for the selected methodological approach? Q5. Clearly described and reproducible?" (either 0, 1, or NA)
- "qa_d6" = quality assessment: "Are issues in data... Q6. Clearly discussed and acknowledged?" (either 0, 1, or NA)
- "qa_d7" = quality assessment: "Are issues in data... Q7. Accounted for in chosen methodological approach?" (either 0, 1, or NA)
- "score" = overall quality assessment score.

Source

Cuomo-Dannenburg G, McCain K, McCabe R, Unwin HJT, Doohan P, Nash RK, et al. Marburg Virus Disease outbreaks, mathematical models, and disease parameters: a Systematic Review. medRxiv; 2023. 2023.07.10.23292424. Available from: https://www.medrxiv.org/content/10.1101/2023.07.10.23292424v1

`marburg_dropdown_models` *Dropdown menu options for model extractions in the systematic review of articles related to Marburg virus disease (MVD).*

Description

This data set provides all the dropdown menu options extractors had when extracting data for mathematical models applied to MVD.

Format

A csv file containing the dropdown options for:

- Model type
- Stochastic or deterministic
- Transmission route
- Compartment type
- Assumptions
- Interventions

Source

Cuomo-Dannenburg G, McCain K, McCabe R, Unwin HJT, Doohan P, Nash RK, et al. Marburg Virus Disease outbreaks, mathematical models, and disease parameters: a Systematic Review. medRxiv; 2023. 2023.07.10.23292424. Available from: https://www.medrxiv.org/content/10.1101/2023.07.10.23292424v1

`marburg_dropdown_outbreaks`

Dropdown menu options for outbreak extractions in the systematic review of articles related to Marburg virus disease (MVD).

Description

This data set provides all the dropdown menu options extractors had when extracting outbreak data for MVD.

Format

A csv file containing the dropdown options for:

- Outbreak country
- Detection mode

Source

Cuomo-Dannenburg G, McCain K, McCabe R, Unwin HJT, Doohan P, Nash RK, et al. Marburg Virus Disease outbreaks, mathematical models, and disease parameters: a Systematic Review. medRxiv; 2023. 2023.07.10.23292424. Available from: https://www.medrxiv.org/content/10.1101/2023.07.10.23292424v1

`marburg_dropdown_parameters`

Dropdown menu options for parameter extractions in the systematic review of articles related to Marburg virus disease (MVD).

Description

This data set provides all the dropdown menu options extractors had when extracting parameter data for MVD.

Format

A csv file containing the dropdown options for:

- Population Country
- Parameter type
- Units
- Parameter uncertainty - single type
- Parameter uncertainty - paired type
- Distribution type
- Distribution parameter 1 - type
- Distribution parameter 2 - type
- Disaggregated by
- Sex
- Setting
- Group
- Timing
- Reproduction number method
- Time from
- Time to
- IFR_CFR_method
- Riskfactor outcome
- Riskfactor name
- Riskfactor occupation
- Riskfactor significant
- Riskfactor adjusted

Source

Cuomo-Dannenburg G, McCain K, McCabe R, Unwin HJT, Doohan P, Nash RK, et al. Marburg Virus Disease outbreaks, mathematical models, and disease parameters: a Systematic Review. medRxiv; 2023. 2023.07.10.23292424. Available from: https://www.medrxiv.org/content/10.1101/2023.07.10.23292424v1

`marburg_model` *Data on the models identified in the systematic review of articles related to Marburg virus disease.*

Description

This data set provides all extracted data for mathematical models applied to Marburg virus disease (MVD).

Format

A csv file containing the following parameters:

- model_data_id = ID variable for the model.
- article_id = ID variable for the article the model came from.
- model_type = whether the model was compartmental, branching process, agent/individual based, other, or unspecified.
- compartmental_type = if the model is compartmental, whether the model is SIS, SIR, SEIR, or "other".
- stoch_deter = whether the article specified whether the model was stochastic, deterministic, or both.
- theoretical_model = "TRUE" or "FALSE". "TRUE" if the model is not fit to data.
- interventions_type = interventions modelled e.g. vaccination, quarantine, vector control, treatment, contact tracing, hospitals, treatment centres, safe burials, behaviour changes, other, or unspecified.
- code_available = whether the code was made available in the article.
- transmission_route = which transmission route was modelled, e.g. airborne or close contact, human to human, vector to human, animal to human, sexual, or unspecified.
- assumptions = assumptions used in the model e.g. homogenous mixing, latent period is the same as incubation period, heterogeneity in transmission rates between groups or over time, age dependent susceptibility, or unspecified.
- covidence_id = article identifier used by the Imperial team.

Source

Cuomo-Dannenburg G, McCain K, McCabe R, Unwin HJT, Doohan P, Nash RK, et al. Marburg Virus Disease outbreaks, mathematical models, and disease parameters: a Systematic Review. medRxiv; 2023. 2023.07.10.23292424. Available from: https://www.medrxiv.org/content/10.1101/2023.07.10.23292424v1

`marburg_outbreak` *Data on the outbreaks identified in the systematic review of articles related to Marburg virus disease.*

Description

This data set provides all extracted data for outbreaks of Marburg virus disease (MVD).

Format

A csv file containing the following parameters:

- outbreak_id = ID variable for the outbreak.
- article_id = ID variable for the article the outbreak data came from.
- outbreak_start_day = Day that the outbreak started (numeric - DD).
- outbreak_start_month = Month that the outbreak started (three letter abbreviation e.g. "Aug").
- outbreak_start_year = Year that the outbreak started (numeric - YYYY).
- outbreak_end_day = Day that the outbreak ended (numeric - DD).
- outbreak_end_month = Month that the outbreak ended (three letter abbreviation e.g. "Aug").
- outbreak_date_year = Year that the outbreak ended (numeric - YYYY).
- outbreak_duration_months = Outbreak duration in months.
- outbreak_size = total outbreak size.
- asymptomatic_transmission = whether asymptomatic transmission occurred, either TRUE or FALSE.
- outbreak_country = country in which the outbreak occurred.
- outbreak_location = location that the outbreak occurred.
- cases_confirmed = total number of confirmed cases.
- cases_mode_detection = either diagnosed based on symptoms alone ("Symptoms"), confirmed via a laboratory test such as PCR ("Molecular (PCR etc)"), "Not specified", or NA.
- cases_suspected = total number of suspected cases.
- cases_asymptomatic = total number of asymptomatic cases.
- deaths = total number of deaths.
- cases_severe_hospitalised = total number of severe hospitalised cases.
- covidence_id = article identifier used by the Imperial team.

Source

Cuomo-Dannenburg G, McCain K, McCabe R, Unwin HJT, Doohan P, Nash RK, et al. Marburg Virus Disease outbreaks, mathematical models, and disease parameters: a Systematic Review. medRxiv; 2023. 2023.07.10.23292424. Available from: https://www.medrxiv.org/content/10.1101/2023.07.10.23292424v1

`marburg_parameter` *Data on the parameters identified in the systematic review of articles related to Marburg virus disease.*

Description

This data set provides all extracted parameters for Marburg virus disease (MVD).

Format

A csv file containing the following parameters:

- parameter_data_id = ID variable for the parameter.
- article_id = ID variable for the article the outbreak data came from.
- parameter_type = extracted parameter (see marburg_dropdown_parameters for the full list of parameters extracted).
- parameter_value = extracted parameter value as stated in the article.
- parameter_unit = units of the extracted parameter (see marburg_dropdown_parameters for the full list of parameter units).
- parameter_lower_bound = minimum value of the parameter across any dimension of disaggregation.
- parameter_upper_bound = maximum value of the parameter across any dimension of disaggregation.
- parameter_value_type = whether the parameter value is the mean, median, standard deviation.
- parameter_uncertainty_single_value = extracted uncertainty if only a single value provided.
- parameter_uncertainty_single_type = uncertainty type if only a single value is reported e.g. "Standard Error (SE)" (see marburg_dropdown_parameters for the full list of options).
- parameter_uncertainty_lower_value = lower value of paired uncertainty.
- parameter_uncertainty_upper_value = upper value of paired uncertainty.
- parameter_uncertainty_type = uncertainty type if a pair of values are provided, e.g. "CI90%", "CRI95%" (see marburg_dropdown_parameters for the full list of options).
- cfr_ifr_numerator = numerator of the cfr or ifr provided in the article.
- cfr_ifr_denominator = denominator of the cfr or ifr provided in the article.
- distribution_type = if uncertainty is given as a distribution (see marburg_dropdown_parameters for the full list of options).
- distribution_par1_value = value for distribution parameter 1.
- distribution_par1_type = distribution parameter 1 type, e.g. shape, scale (see marburg_dropdown_parameters for the full list of options).
- distribution_par1_uncertainty = whether the article reported uncertainty for the distribution parameter estimates (TRUE/FALSE).
- distribution_par2_value = value for distribution parameter 2.

- distribution_par2_type = distribution parameter 2 type, e.g. shape, scale (see marburg_dropdown_parameters for the full list of options).
- distribution_par2_uncertainty = whether the article reported uncertainty for the distribution parameter estimates (TRUE/FALSE).
- method_from_supplement = whether the parameter was taken from the supplementary material of the article. TRUE/FALSE.
- method_moment_value = time period, either "Pre outbreak", Start outbreak", "Mid outbreak", "Post Outbreak", or "Other", if specified in the article.
- cfr_ifr_method = whether the method used to calculate the cfr/ifr was "Naive", "Adjusted", "Unknown".
- method_r = method used to estimate the reproduction number (see marburg_dropdown_parameters for the full list of options).
- method_disaggregated_by = how the parameter is disaggregated e.g. by Age, Sex, Region.
- method_disaggregated = whether the parameter is disaggregated (TRUE/FALSE).
- method_disaggregated_only = whether only disaggregated data is available (TRUE/FALSE).
- riskfactor_outcome = the outcome(s) for which the risk factor was evaluated e.g. "Death" or "Infection" (see marburg_dropdown_parameters for the full list of options).
- riskfactor_name = the potential risk factor(s) evaluated e.g. "Age" (see marburg_dropdown_parameters for the full list of options).
- riskfactor_occupation = specific occupation(s) if occupation is a risk factor (see marburg_dropdown_parameters for the full list of options).
- riskfactor_significant = either "Significant", "Not significant", "Unspecified" or NA.
- riskfactor_adjusted = either "Adjusted", "Not adjusted", "Unspecified".
- population_sex = the sex of the study population, either "Female", "Male", "Both", "Unspecified".
- population_sample_type = how the study was conducted e.g. "Hospital based", "Population based" (see 'Setting' in marburg_dropdown_parameters for the full list of options).
- population_group = population group e.g. "Healthcare workers" (see marburg_dropdown_parameters for the full list of options).
- population_age_min = minimum age in sample (if reported).
- population_age_max = maximum age in the sample (if reported).
- population_sample_size = total number of participants/samples tested.
- population_country = country/countries that the study took place in.
- population_location = location that the study took place e.g. region, city.
- population_study_start_day = day study started (numeric - DD)
- population_study_start_month = month study started (three letter abbreviation e.g. "Feb")
- population_study_start_year = year study started (numeric - YYYY)
- population_study_end_day = day study ended (numeric - DD)
- population_study_end_month = month study ended (three letter abbreviation e.g. "Feb")
- population_study_end_year = year study ended (numeric - YYYY)

- genome_site = the portion of the pathogen's genome used to estimate any extracted parameters. I.e. the gene, gene segment, codon position, or a more generic description ('whole genome' or 'intergenic positions').
- genomic_sequence_available = whether the study sequenced new pathogen isolates and their accession numbers have been provided for retrieval from a public database. TRUE/FALSE.
- parameter_class = class of the extracted parameter. Either "Human delay", "Seroprevalence", "Severity", "Reproduction number", "Mutations", "Risk factors", or "Other transmission parameters".
- covidence_id = article identifier used by the Imperial team.

Source

Cuomo-Dannenburg G, McCain K, McCabe R, Unwin HJT, Doohan P, Nash RK, et al. Marburg Virus Disease outbreaks, mathematical models, and disease parameters: a Systematic Review. medRxiv; 2023. 2023.07.10.23292424. Available from: https://www.medrxiv.org/content/10.1101/2023.07.10.23292424v1

`mark_multiple_estimates` *Distinguish multiple estimates from the same study*

Description

Distinguish multiple estimates from the same study

Usage

```
mark_multiple_estimates(
  df,
  col = "parameter_type",
  label_type = c("letters", "numbers")
)
```

Arguments

`df`	The data frame containing the estimates.
`col`	The column name that identifies multiple enteries for a study. Duplicate values in this column for a study will be marked with a suffix. Although the user can choose any column here, the most logical choices are: for parameters - "parameter_type"; for models - "model_type"; for outbreaks - "outbreak_country".
`label_type`	Type of labels to add to distinguish multiple estimates. Must be one of "letters" or "numbers".

Details

If a study has more than one estimate/model/outbreak for the same parameter_type/model/outbreak, we add a suffix to the article_label to distinguish them otherwise they will be plotted on the same line in the forest plot. Say we have two estimates for the same parameter_type (p) from the same study (s), they will then be labeled as s 1 and s 2.

Value

The modified data frame with updated article_label

Examples

```
df <- data.frame(
  article_label = c("A", "A", "B", "B", "C"),
  parameter_type = c("X", "X", "Y", "Y", "Z")
)
mark_multiple_estimates(df, label_type = "numbers")
```

`model_column_type` *model_column_type*

Description

This function defines the column types for the models in the dataset. It returns a list of column types with their corresponding names.

Usage

```
model_column_type()
```

Value

A list of column types for the article data frame

See Also

parameter_column_type, outbreak_column_type, model_column_type

Examples

```
model_column_type()
```

outbreak_column_type *outbreak_column_type*

Description

This function defines the column types for the outbreaks in the dataset. It returns a list of column types with their corresponding names.

Usage

```
outbreak_column_type()
```

Value

A list of column types for the article data frame

See Also

parameter_column_type, outbreak_column_type, model_column_type

Examples

```
outbreak_column_type()
```

parameter_column_type *parameter_column_type*

Description

This function defines the column types for the parameters in the dataset. It returns a list of column types with their corresponding names.

Usage

```
parameter_column_type()
```

Value

A list of column types for the article data frame

See Also

parameter_column_type, outbreak_column_type, model_column_type

Examples

```
parameter_column_type()
```

parameter_palette *Define a consistent color palette for use in figures*

Description

Define a consistent color palette for use in figures

Usage

```
parameter_palette(x = NULL)
```

Arguments

x	a list of parameters. Optional. If missing, the entire palette is returned.

Value

a named list of colors that can be used in forest plots for manually setting colors with for example scale_color_manual

Author(s)

Sangeeta Bhatia

Examples

```
parameter_palette()
```

param_pm_uncertainty *Update parameter uncertainty columns in a data frame*

Description

This function updates the parameter uncertainty columns in a data frame when the uncertainty is given by a single value (standard deviation or standard error). It creates new columns called low' (parameter central value - uncertainty) and high' (parameter central value + uncertainty). These columns are used by forest_plot to plot the uncertainty intervals. If the uncertainty is given by a range, the midpoint of the range is used as the central value (mid) and the lower and upper bounds are used as the low and high values respectively.

Usage

```
param_pm_uncertainty(df)
```

Arguments

df	A data frame containing the parameter uncertainty columns. This will typically be the output of load_epidata.

Value

The updated data frame with parameter uncertainty columns

Examples

```
df <- data.frame(
  parameter_value = c(10, 20, 30),
  parameter_uncertainty_single_value = c(1, 2, 3),
  parameter_uncertainty_lower_value = c(5, 15, 25),
  parameter_uncertainty_upper_value = c(15, 25, 35),
  parameter_uncertainty_type = c(NA, NA, NA),
  parameter_uncertainty_singe_type = c("Standard Deviation", "Standard Error", NA)
)
updated_df <- param_pm_uncertainty(df)
updated_df
```

---

pretty_article_label *Make pretty labels for articles*

Description

This function generates pretty labels for articles. The labels are created by combining the surname of the first year and year of publication of an article. If the surname is missing, we will use the first name. If both are missing, a warning is issued and the Covidence ID is used instead.

Usage

```
pretty_article_label(articles, mark_multiple)
```

Arguments

articles	A data frame containing information about the articles. This will typically be the output of load_epidata_raw.
mark_multiple	logical. If TRUE, multiple studies from the same author in the same year will be marked with an numeric suffix to distinguish them. See mark_multiple_estimates for more details.

Value

A modified data frame with an additional column "article_label" containing the generated labels.

Examples

```
articles <- data.frame(
  first_author_surname = c("Smith", NA, "Johnson"),
  first_author_first_name = c(NA, "John", NA),
  year_publication = c(2010, NA, 2022),
  covidence_id = c("ABC123", "DEF456", "GHI789")
)
pretty_article_label(articles, mark_multiple = TRUE)
```

`priority_pathogens` *priority_pathogens*

Description

This data set gives the list of WHO priority pathogens for which the Pathogen Epidemiology Review Group (PERG) has carried out a systematic review. The data set gives the name of the pathogen as used in the package and associated information with the review.

Usage

```
priority_pathogens()
```

Details

Data on the priority pathogens included in the systematic review

Value

data.frame with the following fields

- pathogen: name of the pathogen as used in the package
- articles_screened: number of titles and abstracts screened for inclusion
- articles_extracted: number of articles from which data were extracted
- doi: doi of the accompanying systematic review

Examples

```
priority_pathogens()
```

qa_questions *Quality assessment questionnaire This function returns the list of 7 questions used to assess quality of articles.*

Description

Quality assessment questionnaire This function returns the list of 7 questions used to assess quality of articles.

Usage

```
qa_questions()
```

Value

a data.frame with the following columns: (a) qnames: these are the names of the corresponding columns in the articles data.frame; (b) qtext: the text of the question, and (c) notes: any additional notes.

reorder_studies *Reorder articles based on parameter value*

Description

This function takes a dataframe as input (this will typically be params data.frame from the output of load_epidata) and reorders it to provide a sensible order for plotting. For each country, studies are ordered by the parameter value or the midpoint of the parameter range if the parameter value is missing. It creates a new column 'y' which is an ordered factor with levels corresponding to the article_label. To order the studies in some other way, set reorder_studies to FALSE in the parameter specific forest plot functions (e.g. forest_plot_rt).

Usage

```
reorder_studies(df, reorder_by = "population_country")
```

Arguments

df	The input dataframe to be reordered
reorder_by	character. The name of the column to reorder the data by. Default is "population_country"

Value

The reordered dataframe

See Also

forest_plot_rt forest_plot_r0

Examples

```
ebola <- load_epidata("ebola")
params <- ebola$params
rt <- params[params$parameter_type == "Reproduction number (Effective, Re)", ]

reorder_studies(param_pm_uncertainty(rt))
```

---

reparam_gamma *Reparameterize Gamma Distribution*

Description

This function reparameterizes the gamma distribution in a given data frame. If a parameter has been expressed as a gamma distribution with shape and scale, we convert these to mean and standard deviation for plotting.

Usage

```
reparam_gamma(df)
```

Arguments

df	A data frame with updated columns for parameter value and uncertainty.

Value

data.frame modified data frame with reparameterized gamma distributions.

---

shape_palette *shape_palette function*

Description

This function generates a shape palette based on the specified shape_by parameter.

Usage

```
shape_palette(shape_by = c("parameter_value_type"), ...)
```

Arguments

shape_by	A character vector specifying the parameter to shape the palette by. Currently, only "value_type" is supported.
...	Additional arguments to be passed to the underlying palette function. These are treated as names of the palette elements.

Value

A shape palette based on the specified shape_by parameter.

Examples

```
shape_palette("parameter_value_type")
```

---

short_parameter_type *Short labels parameters for use in figures*

Description

Short labels parameters for use in figures

Usage

```
short_parameter_type(x, parameter_type_full, parameter_type_short)
```

Arguments

x	data.frame containing a column called "parameter_type", This will typically be the params data.frame from the output of load_epidata.
parameter_type_full	optional. User can specify the full name of a parameter type not already included in the function.
parameter_type_short	optional. Shorter value of parameter_type_full

Details

This function assigns short labels to otherwise very long parameter names. It is generally not intended to be called directly but is used by load_epidata when the data is loaded. The short parameter names are read from the file "param_name.csv" in the package. If you want to supply your own short names, you can do so by specifying the parameter_type_full and parameter_type_short arguments. Note however that if parameter_type_full does not contain all the parameter types in the data, the short label (parameter_type_short) will be NA for missing values. It is therefore easier and recommended that you update the column parameter_type_short once the data are loaded via load_epidata.

Value

data.frame with a new column called "parameter_type_short"

Author(s)

Sangeeta Bhatia

---

theme_epireview *Plotting theme for epireview A standard theme for figures in epireview.*

Description

Plotting theme for epireview A standard theme for figures in epireview.

Usage

```
theme_epireview(
  base_size = 11,
  base_family = "",
  base_line_size = base_size/22,
  base_rect_size = base_size/22
)
```

Arguments

base_size	base font size, given in pts.
base_family	base font family
base_line_size	base size for line elements
base_rect_size	base size for rect elements

---

update_article_info *Include key article information when DOI is missing*

Description

This function appends pretty_article_label to include the journal and we use this information to fill in any entry where the DOI is missing. (Note: DOIs were introduced in the late 1990s and so articles from before this time often do not have one.)

Usage

```
update_article_info(articles)
```

Arguments

articles	A data frame containing information about the articles. This will typically be the output of load_epidata_raw.

Value

A modified data frame with an updated column "doi" with NA values replaced with either (1) article_label appended to include journal where available or (2) just article_label when journal entry is NA.

Examples

```
articles <- data.frame(
  doi = c("10.123", NA, "10.234"),
  article_label = c("Smith 2020", "Smith 1979", "Smith 2023"),
  journal = c("Science", "Nature", "The Lancet")
)
update_article_info(articles)
```

---

value_type_palette *Define a consistent shape palette for use in forest plots We map shape aesthetic to value type i.e., mean, median etc. This function defines a shape palette that can be used in forest plots*

Description

Define a consistent shape palette for use in forest plots We map shape aesthetic to value type i.e., mean, median etc. This function defines a shape palette that can be used in forest plots

Usage

```
value_type_palette(x = NULL)
```

Arguments

x	a list of parameters

Value

a named list of shapes where names are value types (mean, median, std dev etc.)

Author(s)

Sangeeta Bhatia

Examples

```
value_type_palette()
```

Index