

Package: socialmixr (via r-universe)

September 22, 2026

Title Social Mixing Matrices for Infectious Disease Modelling

Version 0.7.0

Description Methods for sampling contact matrices from diary data for use in infectious disease modelling, as discussed in Mossong et al. (2008) <[doi:10.1371/journal.pmed.0050074](https://doi.org/10.1371/journal.pmed.0050074)>.

License MIT + file LICENSE

Depends R (>= 4.1.0)

Imports checkmate, countrycode, curl, data.table, grDevices, httr, jsonlite, lifecycle, lubridate, memoise, purrr, oai, wpp2017, xml2, cli, rlang, methods

Suggests contactsurveys, ggplot2, here, knitr, quarto, reshape2, rmarkdown, roxyglobals (>= 1.0.0), testthat, withr

VignetteBuilder knitr

Encoding UTF-8

LazyData true

NeedsCompilation no

RoxygenNote 7.3.3

URL <https://github.com/epiforecasts/socialmixr>,
<https://epiforecasts.io/socialmixr/>

BugReports <https://github.com/epiforecasts/socialmixr/issues>

Roxygen list(markdown = TRUE, roclets = c("`collate", ``namespace", ``rd", ``roxyglobals::global_roclet"))

Config/testthat/edition 3

Config/pak/sysreqs libicu-dev libxml2-dev libssl-dev

Repository <https://epiforecasts.r-universe.dev>

Date/Publication 2026-08-22 16:17:25 UTC

RemoteUrl <https://github.com/epiforecasts/socialmixr>

RemoteRef v0.7.0

RemoteSha eecd1b614ace27dae7ab3a048342398bfb4a86e3

Contents

[.contact_survey	2
age_groups_to_limits	3
align_ages	4
as_contact_survey	5
assign_age_groups	6
check	7
clean	8
compute_matrix	9
contact_age_distribution	10
contact_matrix	11
download_survey	14
flatten	15
get_citation	16
get_survey	17
is_doi	18
limits_to_age_groups	18
list_surveys	19
load_survey	20
matrix_plot	21
per_capita	22
polymod	23
rebin_ages	24
reduce_age_groups	25
split_matrix	25
survey	26
survey_countries	27
survey_country_population	28
symmetrise	29
weigh	30
wpp_age	32
wpp_countries	33
Index	35

[.contact_survey	<i>Subset a contact survey</i>
------------------	--------------------------------

Description

Filters a contact_survey object using an expression. The expression is evaluated against whichever table(s) contain the referenced columns (participants, contacts, or both). When participants are filtered, contacts are automatically pruned to matching part_ids.

Usage

```
## S3 method for class 'contact_survey'
x[i, ...]
```

Arguments

x	a contact_survey object
i	an expression to evaluate as a row filter (e.g. country == "United Kingdom")
...	ignored

Value

a filtered contact_survey object

Examples

```
data(polymod)
polymod[country == "United Kingdom"]
```

age_groups_to_limits *Convert age groups to lower age limits*

Description

Inverse of [limits_to_age_groups\(\)](#). Extracts lower age limits from age group labels.

Usage

```
age_groups_to_limits(x)
```

Arguments

x	age groups (a factor, as produced by limits_to_age_groups() or assign_age_groups())
---	--

Value

a numeric vector of lower age limits

Examples

```
age_groups_to_limits(
  limits_to_age_groups(c(0, 5, 10), notation = "brackets")
)
```

align_ages

*Align a population table to a contact matrix's grouping levels***Description**

Aligns a population table to the groupings of a contact matrix produced by `compute_matrix()`, returning the `survey_pop` data frame that `symmetrise()`, `split_matrix()` and `per_capita()` expect.

The age grouping is rebinned to the matrix's age groups (via `rebin_ages()`, summing) within each combination of the other groupings; the population must be at least as fine as the matrix's age groups, otherwise `rebin_ages()` errors. Categorical groupings are aggregated to the matrix's levels by exact name; a level not present in the matrix is an error.

Usage

```
align_ages(pop, x)
```

Arguments

<code>pop</code>	a data frame with a population column and one column per grouping: an age column of age-group labels for the age grouping, and a column named after each categorical grouping holding its levels.
<code>x</code>	a <code>contact_matrix</code> object as returned by <code>compute_matrix()</code>

Value

a data frame with one column per grouping (named after the grouping, holding the matrix's levels) plus a population column, ready to pass as `survey_pop`

Examples

```
data(polymod)
result <- polymod |>
  (\(s) s[country == "United Kingdom"])( ) |>
  assign_age_groups(age_limits = c(0, 5, 15)) |>
  compute_matrix()
uk_pop <- data.frame(
  age = limits_to_age_groups(0:80, notation = "brackets"),
  population = rep(1e5, 81)
)
result |> symmetrise(survey_pop = align_ages(uk_pop, result))
```

as_contact_survey *Check contact survey data*

Description

Checks that a survey fulfills all the requirements to work with the 'contact_matrix' function

Usage

```
as_contact_survey(
  x,
  id_column = "part_id",
  country_column = NULL,
  year_column = NULL,
  ...,
  id.column = deprecated(),
  country.column = deprecated(),
  year.column = deprecated()
)
```

Arguments

x	list containing • an element named 'participants', a data frame containing participant information • an element named 'contacts', a data frame containing contact information • (optionally) an element named 'reference', a list containing information needed to reference the survey, in particular it can contain a "title", "bibtype", "author", "doi", "publisher", "note", "year"
id_column	the column in both the participants and contacts data frames that links contacts to participants
country_column	the column in the participants data frame containing the country in which the participant was queried; if NULL (default), will use "country" column if present
year_column	the column in the participants data frame containing the year in which the participant was queried; if NULL (default), will use "year" column if present
...	additional arguments (currently ignored)
id.column, country.column, year.column	[Deprecated] Use the underscore versions (e.g., id_column) instead.

Value

invisibly returns a character vector of the relevant columns

Examples

```
data(polymod)
as_contact_survey(polymod)
```

assign_age_groups	<i>Assign age groups in survey data</i>
-------------------	---

Description

This function processes age data in a survey object. It imputes ages from ranges, handles missing values, and assigns age groups.

Usage

```
assign_age_groups(
  survey,
  age_limits = NULL,
  estimated_participant_age = c("mean", "sample", "missing"),
  estimated_contact_age = c("mean", "sample", "missing"),
  missing_participant_age = c("remove", "keep"),
  missing_contact_age = c("remove", "sample", "keep", "ignore")
)
```

Arguments

survey	a survey() object
age_limits	lower limits of the age groups over which to construct the matrix. Defaults to NULL. If NULL, age limits are inferred from participant and contact ages.
estimated_participant_age	if set to "mean" (default), people whose ages are given as a range (in columns named "..._est_min" and "..._est_max") but not exactly (in a column named "..._exact") will have their age set to the mid-point of the range; if set to "sample", the age will be sampled from the range; if set to "missing", age ranges will be treated as missing
estimated_contact_age	if set to "mean" (default), contacts whose ages are given as a range (in columns named "..._est_min" and "..._est_max") but not exactly (in a column named "..._exact") will have their age set to the mid-point of the range; if set to "sample", the age will be sampled from the range; if set to "missing", age ranges will be treated as missing
missing_participant_age	if set to "remove" (default), participants without age information are removed; if set to "keep", participants with missing age are kept and treated as a separate age group
missing_contact_age	if set to "remove" (default), participants that have contacts without age information are removed; if set to "keep", contacts with missing age are kept and treated as a separate age group; if set to "ignore", contacts with missing age are ignored in the contact analysis. The "sample" option is defunct (errors). For contacts that have only an age range (rather than a truly missing age),

estimated_contact_age controls how the range is resolved into a single age; it is not a substitute for missing_contact_age when the age is entirely missing.

Value

The survey object with processed age data.

Examples

```
polymod_grouped <- assign_age_groups(polymod)
polymod_grouped
polymod_custom <- assign_age_groups(polymod, age_limits = c(0, 5, 10, 15))
polymod_custom
```

check	<i>Check contact survey data</i>
-------	----------------------------------

Description

Checks that a survey fulfills all the requirements to work with the 'contact_matrix' function

Usage

```
## S3 method for class 'contact_survey'
check(
  x,
  id.column = "part_id",
  participant.age.column = "part_age",
  country.column = "country",
  year.column = "year",
  contact.age.column = "cnt_age",
  ...
)
```

Arguments

x	A survey() object
id.column	the column in both the participants and contacts data frames that links contacts to participants
participant.age.column	the column in the participants data frame containing participants' age; if this does not exist, at least columns "..._exact", "..._est_min" and "..._est_max" must exist (see the estimated.participant.age option in contact_matrix())
country.column	the column in the participants data frame containing the country in which the participant was queried
year.column	the column in the participants data frame containing the year in which the participant was queried

```

contact.age.column
    the column in the contacts data frame containing the age of contacts; if this
    does not exist, at least columns "..._exact", "..._est_min" and "..._est_max" must
    exist (see the estimated.contact.age option in contact\_matrix\(\))
...
    ignored

```

Value

invisibly returns a character vector of the relevant columns

Examples

```

data(polymod)
try(check(polymod))

```

clean	<i>Clean contact survey data</i>
-------	----------------------------------

Description

Cleans survey data to work with the 'contact_matrix' function

Usage

```

## S3 method for class 'contact_survey'
clean(
  x,
  participant_age_column = "part_age",
  ...,
  participant.age.column = deprecated()
)

```

Arguments

```

x                A survey\(\) object
participant_age_column
    the column in x$participants containing participants' age
...
    ignored
participant.age.column
    [Deprecated] Use participant_age_column instead.

```

Value

a cleaned survey in the correct format

Examples

```

data(polymod)
cleaned <- clean(polymod) # not really necessary, polymod is clean

```

compute_matrix

*Compute contact matrix from prepared survey data***Description**

Computes a contact matrix from a `contact_survey` that has been processed by `assign_age_groups()` and optionally `weigh()`. This is the final step in the pipeline workflow.

For post-processing, pipe the result into `symmetrise()`, `split_matrix()`, or `per_capita()`.

Usage

```
compute_matrix(survey, by = "age", counts = FALSE, weight_threshold = NULL)
```

Arguments

<code>survey</code>	a <code>survey()</code> object with the columns named in <code>by</code> present on both participants and contacts. Age groupings come from <code>assign_age_groups()</code> ; other groupings should already be present as <code>part_<name></code> / <code>cnt_<name></code> columns on the survey.
<code>by</code>	character vector or list of grouping specifications. Each entry is either the string "age" (uses <code>age.group</code> / <code>contact.age.group</code>), a stem string "<name>" (uses <code>part_<name></code> / <code>cnt_<name></code>), or an explicit <code>c(part = "X", cnt = "Y")</code> . Default "age" reproduces the single-grouping behaviour of previous releases.
<code>counts</code>	whether to return counts instead of means
<code>weight_threshold</code>	numeric; if provided, weights above this threshold are capped to the threshold value and then re-normalised (default NULL)

Value

a `contact_matrix` object with elements `matrix` (a rank-2K array) and `participants` (a long table with one row per grouping combination)

Multi-dimensional matrices

Passing more than one entry to `by` produces a matrix of rank 2K, where $K = \text{length}(\text{by})$. The first K dimensions index participants and the last K dimensions index contacts, in the order given to `by`. For example, `by = c("age", "gender")` returns an array with dimensions (age, gender, age, gender) — age and gender of the participant first, then of the contact. Dim names carry the levels of each grouping.

Examples

```
data(polymod)

# Single-grouping (age) – default
polymod |>
```

```
assign_age_groups(age_limits = c(0, 5, 15)) |>
compute_matrix()

# Two-grouping (age x gender)
polymod |>
  assign_age_groups(age_limits = c(0, 5, 15)) |>
  compute_matrix(by = c("age", "gender"))
```

contact_age_distribution

Extract the empirical age distribution of contacts from a survey

Description

Returns a data.frame of (age, proportion) pairs representing how contact ages are distributed in the survey. This can be passed to [assign_age_groups\(\)](#) as `estimated_contact_age` to impute ages from ranges using this distribution rather than uniform sampling.

Usage

```
contact_age_distribution(survey)
```

Arguments

survey a [survey\(\)](#) object

Value

a data.frame with columns age (integer) and proportion (numeric, summing to 1)

Examples

```
data(polymod)
dist <- contact_age_distribution(polymod)
head(dist)
plot(dist$age, dist$proportion, type = "h",
      xlab = "Age", ylab = "Proportion")
```

contact_matrix	<i>Generate a contact matrix from diary survey data</i>
----------------	---

Description

Samples a contact survey

Usage

```
contact_matrix(
  survey,
  countries = NULL,
  survey_pop = NULL,
  age_limits = NULL,
  filter = NULL,
  counts = FALSE,
  symmetric = FALSE,
  split = FALSE,
  sample_participants = FALSE,
  estimated_participant_age = c("mean", "sample", "missing"),
  estimated_contact_age = c("mean", "sample", "missing"),
  missing_participant_age = c("remove", "keep"),
  missing_contact_age = c("remove", "sample", "keep", "ignore"),
  weights = NULL,
  weigh_dayofweek = FALSE,
  weigh_age = FALSE,
  weight_threshold = NA,
  symmetric_norm_threshold = 2,
  sample_all_age_groups = FALSE,
  sample_participants_max_tries = 1000,
  return_part_weights = FALSE,
  return_demography = NA,
  per_capita = FALSE,
  ...,
  survey.pop = deprecated(),
  age.limits = deprecated(),
  sample.participants = deprecated(),
  estimated.participant.age = deprecated(),
  estimated.contact.age = deprecated(),
  missing.participant.age = deprecated(),
  missing.contact.age = deprecated(),
  weigh.dayofweek = deprecated(),
  weigh.age = deprecated(),
  weight.threshold = deprecated(),
  symmetric.norm.threshold = deprecated(),
  sample.all.age.groups = deprecated(),
  sample.participants.max.tries = deprecated(),
```

```

    return.part.weights = deprecated(),
    return.demography = deprecated(),
    per.capita = deprecated()
)

```

Arguments

survey	a survey() object.
countries	limit to one or more countries; if NULL (default), will use all countries in the survey; these can be given as country names or 2-letter (ISO Alpha-2) country codes.
survey_pop	survey population – a data frame with columns <code>lower.age.limit</code> and <code>population</code> . Passing NULL (the default) or a character vector of country names triggers the [Deprecated] implicit lookup via wpp_age() when <code>symmetric</code> , <code>split</code> , <code>per_capita</code> , <code>weigh_age</code> , or <code>return_demography</code> is TRUE; supply an explicit data frame (e.g. constructed from the <code>wpp2024</code> package or another source) instead. If the population is coarser than the requested age groups it is linearly interpolated to finer groups, but this is deprecated (it warns and will error in a future release); supply population at least as fine as <code>age_limits</code> .
age_limits	lower limits of the age groups over which to construct the matrix. If NULL (default), age limits are inferred from participant and contact ages.
filter	any filters to apply to the data, given as list of the form <code>(column=filter_value)</code> - only contacts that have 'filter_value' in 'column' will be considered. If multiple filters are given, they are all applied independently and in the sequence given. Default value is NULL; no filtering performed.
counts	whether to return counts (instead of means).
symmetric	whether to make matrix symmetric, such that $c_{ij}N_i = c_{ji}N_j$.
split	whether to split the contact matrix into the mean number of contacts, in each age group (split further into the product of the mean number of contacts across the whole population (<code>mean.contacts</code>), a normalisation constant (<code>normalisation</code>) and age-specific variation in contacts (<code>contacts</code>)), multiplied with an assortativity matrix (<code>assortativity</code>) and a population multiplier (<code>demography</code>). For more detail on this, see the "Getting Started" vignette.
sample_participants	whether to sample participants randomly (with replacement); done multiple times this can be used to assess uncertainty in the generated contact matrices. See the "Bootstrapping" section in the vignette for how to do this.
estimated_participant_age	if set to "mean" (default), people whose ages are given as a range (in columns named "..._est_min" and "..._est_max") but not exactly (in a column named "..._exact") will have their age set to the mid-point of the range; if set to "sample", the age will be sampled from the range; if set to "missing", age ranges will be treated as missing
estimated_contact_age	if set to "mean" (default), contacts whose ages are given as a range (in columns named "..._est_min" and "..._est_max") but not exactly (in a column named

	"..._exact") will have their age set to the mid-point of the range; if set to "sample", the age will be sampled from the range; if set to "missing", age ranges will be treated as missing.
missing_participant_age	if set to "remove" (default), participants without age information are removed; if set to "keep", participants with missing age are kept and will appear in the contact matrix in a row labelled "NA".
missing_contact_age	if set to "remove" (default), participants that have contacts without age information are removed; if set to "keep", contacts with missing age are kept and will appear in the contact matrix in a column labelled "NA"; if set to "ignore", contacts without age information are removed from the analysis (but the participants that made them are kept). The "sample" option is defunct (errors).
weights	column name(s) of the participant data of the <code>survey()</code> object with user-specified weights (default = empty vector).
weigh_dayofweek	whether to weigh social contacts data by the day of the week (weight $(5/7 / N_week / N)$ for weekdays and $(2/7 / N_weekend / N)$ for weekends).
weigh_age	whether to weigh social contacts data by the age of the participants (vs. the populations' age distribution).
weight_threshold	threshold value for the standardized weights before running an additional standardisation (default 'NA' = no cutoff).
symmetric_norm_threshold	threshold value for the normalization weights when <code>symmetric = TRUE</code> before showing a warning that that large differences in the size of the sub-populations are likely to result in artefacts when making the matrix symmetric (default 2).
sample_all_age_groups	what to do if sampling participants (with <code>sample_participants = TRUE</code>) fails to sample participants from one or more age groups; if FALSE (default), corresponding rows will be set to NA, if TRUE the sample will be discarded and a new one taken instead.
sample_participants_max_tries	maximum number of attempts when <code>sample_all_age_groups = TRUE</code> ; defaults to 1000.
return_part_weights	boolean to return the participant weights.
return_demography	boolean to explicitly return demography data that corresponds to the survey data (default 'NA' = if demography data is requested by other function parameters).
per_capita	whether to return a matrix with contact rates per capita (default is FALSE and not possible if 'counts=TRUE' or 'split=TRUE').
...	further arguments to pass to <code>get_survey()</code> and <code>check()</code> (especially column names).

```

survey.pop,          age.limits,          sample.participants,
estimated.participant.age,      estimated.contact.age,
missing.participant.age,      missing.contact.age,      weigh.dayofweek,
weigh.age,          weight.threshold,      symmetric.norm.threshold,
sample.all.age.groups,          sample.participants.max.tries,
return.part.weights, return.demography, per.capita

```

[Deprecated] Use the underscore-separated versions of these arguments instead.

Value

a contact matrix, and the underlying demography of the surveyed population

Author(s)

Sebastian Funk

Examples

```

data(polymod)
contact_matrix(
  survey = polymod,
  countries = "United Kingdom",
  age_limits = c(0, 1, 5, 15)
)

```

download_survey

Download a survey from its Zenodo repository

Description

[Defunct]

download_survey() is defunct. Use `contactsurveys::download_survey()` instead.

download_survey() downloads survey data from Zenodo.

Usage

```
download_survey(survey, dir = NULL, sleep = 1)
```

Arguments

survey	a URL (see <code>contactsurveys::list_surveys()</code>)
dir	a directory to save the files to; if not given, will save to a temporary directory
sleep	time to sleep between requests to avoid overloading the server (passed on to Sys.sleep)

Value

Always errors.

See Also

load_survey

Examples

```
# we recommend using the contactsurveys package for download_survey()
## Not run:
# if needed, discover surveys with:
contactsurveys::list_surveys()
peru_survey <- download_survey("https://doi.org/10.5281/zenodo.1095664")
# -->
peru_survey <- contactsurveys::download_survey(
  "https://doi.org/10.5281/zenodo.1095664"
)

## End(Not run)
```

flatten

*Flatten a multi-grouping contact matrix to its $T \times T$ form***Description**

Returns the contact matrix in the flattened representation of Manna et al. — a $T \times T$ matrix where each axis enumerates the Cartesian product of grouping levels. For a single-grouping matrix this is the matrix itself.

Row/column names join the grouping levels with a colon, e.g. " $[0, 5): F$ " for the age group $[0, 5)$ combined with gender level F. The participant axes vary fastest in the first grouping (column-major reshape).

Usage

flatten(x)

Arguments

x a contact_matrix object as returned by `compute_matrix()`

Value

a numeric $T \times T$ matrix

References

Manna A, Dall’Amico L, Tizzoni M, Karsai M, Perra N (2024). Generalized contact matrices allow integrating socioeconomic variables into epidemic models. *Science Advances* **10**(41), eadk4606. [doi:10.1126/sciadv.adk4606](https://doi.org/10.1126/sciadv.adk4606)

Examples

```
data(polymod)
polymod |>
  (\(s) s[country == "United Kingdom"]()) |>
  assign_age_groups(age_limits = c(0, 5, 15)) |>
  compute_matrix(by = c("age", "gender")) |>
  flatten()
```

get_citation	<i>Citation for a survey</i>
--------------	------------------------------

Description**[Defunct]**

get_citation() is defunct. Use `contactsurveys::get_citation()` instead.

Usage

```
get_citation(x)
```

Arguments

x a character vector of surveys to cite

Value

Always errors.

Examples

```
# we recommend using the contactsurveys package for get_citation()
## Not run:
data(polymod)
citation <- contactsurveys::get_citation(polymod)
print(citation)
print(citation, style = "bibtex")

## End(Not run)
```

get_survey	<i>Get a survey, either from its Zenodo repository, a set of files, or a survey variable</i>
------------	--

Description

[Defunct]

get_survey() is defunct. Use `contactsurveys::download_survey()` and then `load_survey()` instead.

Downloads survey data, or extracts them from files, and returns a clean data set. If a survey URL is accessed multiple times, the data will be cached (unless `clear_cache` is set to `TRUE`) to avoid repeated downloads.

If survey objects are used repeatedly the downloaded files can be saved and reloaded between sessions then survey objects can be saved/loaded using `base::saveRDS()` and `base::readRDS()`, or via the individual survey files that can be downloaded using `download_survey()` and subsequently loaded using `load_survey()`.

Usage

```
get_survey(survey, clear_cache = FALSE, ...)
```

Arguments

survey	a DOI or url to get the survey from, or a <code>survey()</code> object.
clear_cache	logical, whether to clear the cache before downloading the survey; by default, the cache is not cleared and so multiple calls of this function to access the same survey will not result in repeated downloads.
...	currently unused

Value

Always errors.

Examples

```
## Not run:
list_surveys()
peru_doi <- "https://doi.org/10.5281/zenodo.1095664"
peru_survey <- get_survey(peru_doi)
## --> We now recommend:
peru_survey <- contactsurveys::download_survey(peru_doi)
peru_data <- load_survey(peru_survey)

## End(Not run)
```

is_doi	<i>Checks if a character string is a DOI</i>
--------	--

Description

Checks if a character string is a DOI

Usage

```
is_doi(x)
```

Arguments

x	Character vector; the string or strings to check
---	--

Value

Logical; TRUE if x is a DOI, FALSE otherwise

Author(s)

Sebastian Funk

limits_to_age_groups	<i>Convert lower age limits to age groups.</i>
----------------------	--

Description

Mostly used for plot labelling

Usage

```
limits_to_age_groups(
  x,
  limits = sort(unique(x)),
  notation = c("dashes", "brackets")
)
```

Arguments

x	age limits to transform
limits	lower age limits; if not given, will use all limits in x
notation	whether to use bracket notation, e.g. [0,4) or dash notation, e.g. 0-4)

Value

Age groups as specified in notation

Examples

```
limits_to_age_groups(c(0, 5, 10))
```

list_surveys	<i>List all surveys available for download</i>
--------------	--

Description**[Defunct]**

list_surveys() is defunct. Use `contactsurveys::list_surveys()` instead.

Usage

```
list_surveys(clear_cache = FALSE)
```

Arguments

clear_cache	logical, whether to clear the cache before downloading the survey; by default, the cache is not cleared and so multiple calls of this function to access the same survey will not result in repeated downloads.
-------------	---

Value

Always errors.

Examples

```
# we recommend using the contactsurveys package now for listing surveys.
## Not run:
contactsurveys::list_surveys()

## End(Not run)
```

load_survey

*Load a survey from local files***Description**

Loads a survey from a local file system. Tables are expected as csv files, and a reference (if present) as JSON.

Usage

```
load_survey(files, participant_key = NULL, ...)
```

Arguments

files a vector of file names as returned by [download_survey\(\)](#)

participant_key character vector specifying columns that uniquely identify participant observations. For cross-sectional surveys this is typically just "part_id" (the default). For longitudinal surveys with multiple observations per participant, specify additional columns like `c("part_id", "wave")`. When NULL (the default), the function will auto-detect if additional columns are needed and inform you.

... options for [clean\(\)](#), which is called at the end of this

Value

a survey in the correct format. For longitudinal surveys with multiple observations per participant, the returned object includes an `observation_key` field containing the column names (excluding `part_id`) that distinguish observations for the same participant.

Examples

```
## Not run:
list_surveys()
peru_files <- download_survey("https://doi.org/10.5281/zenodo.1095664")
peru_survey <- load_survey(peru_files)

# For longitudinal surveys, specify the unique key explicitly:
france_files <- download_survey("https://doi.org/10.5281/zenodo.1157918")
france_survey <- load_survey(france_files,
  participant_key = c("part_id", "wave", "studyDay")
)

## End(Not run)
```

matrix_plot	<i>Draws an image plot of a contact matrix with a legend strip and the numeric values in the cells.</i>
-------------	---

Description

This function combines the R `image.plot` function with numeric contact rates in the matrix cells.

Usage

```
matrix_plot(
  mij,
  min.legend = 0,
  max.legend = NA,
  num.digits = 2,
  num.colors = 50,
  main,
  xlab,
  ylab,
  legend.width,
  legend.mar,
  legend.shrink,
  cex.lab,
  cex.axis,
  cex.text,
  color.palette = heat.colors
)
```

Arguments

<code>mij</code>	a contact matrix containing contact rates between participants of age <i>i</i> (rows) with contacts of age <i>j</i> (columns). This is the default matrix format of <code>contact_matrix()</code> .
<code>min.legend</code>	the color scale minimum (default = 0). Set to NA to use the minimum value of <code>mij</code> .
<code>max.legend</code>	the color scale maximum (default = NA). Set to NA to use the maximum value of <code>mij</code> .
<code>num.digits</code>	the number of digits when rounding the contact rates (default = 2). Use NA to disable this.
<code>num.colors</code>	the number of color breaks (default = 50)
<code>main</code>	the figure title
<code>xlab</code>	a title for the x axis (default: "Age group (year)")
<code>ylab</code>	a title for the y axis (default: "Contact age group (year)")
<code>legend.width</code>	width of the legend strip in characters. Default is 1.
<code>legend.mar</code>	width in characters of legend margin. Default is 5.1.

legend.shrink	amount to shrink the size of legend relative to the full height or width of the plot. Default is 0.9.
cex.lab	size of the x and y labels (default: 1.2)
cex.axis	size of the axis labels (default: 0.8)
cex.text	size of the numeric values in the matrix (default: 1)
color.palette	the color palette to use (default: <code>heat.colors()</code>). Other examples are <code>topo.colors()</code> , <code>terrain.colors()</code> and <code>hcl.colors()</code> . User-defined functions are also possible if they take the number of colors to be in the palette as function argument.

Details

This is a function using basic R graphics to visualise a social contact matrix.

Author(s)

Lander Willem

Examples

```
## Not run:
data(polymod)
mij <- contact_matrix(
  polymod,
  countries = "United Kingdom",
  age_limits = c(0, 18, 65)
)$matrix
matrix_plot(mij)

## End(Not run)
```

per_capita	<i>Convert a contact matrix to per-capita rates</i>
------------	---

Description

Divides each column of the contact matrix by the population of the contacted group, so that entry (a, b) becomes the mean number of contacts a member of group a makes with a single individual of group b. Multi-grouping matrices are handled the same way, with each combination of grouping levels treated as a group.

Usage

```
per_capita(x, survey_pop)
```

Arguments

x	a list as returned by <code>compute_matrix()</code> , with elements <code>matrix</code> and <code>participants</code>
survey_pop	a data frame; see <i>Population data</i> below

Value

x with `$matrix` replaced by the per-capita version

Population data

`survey_pop` is a data frame with one column per grouping, named after the grouping (e.g. age, gender) and holding that grouping's levels as they appear in the matrix, plus a population column with the size of each combination. One row per combination of levels is required, and levels are matched to the matrix exactly, without interpolation.

Use `align_ages()` to build this from a raw population table: it aggregates each grouping to the matrix's levels (interpolating the age grouping where needed) and labels the columns to match.

Examples

```
data(polymod)
result <- polymod |>
  (\(s) s[country == "United Kingdom"])( ) |>
  assign_age_groups(age_limits = c(0, 5, 15)) |>
  compute_matrix()
uk_pop <- data.frame(
  age = limits_to_age_groups(0:80, notation = "brackets"),
  population = rep(1e5, 81)
)
result |> per_capita(survey_pop = align_ages(uk_pop, result))
```

polymod

Social contact data from 8 European countries

Description

A dataset containing social mixing diary data from 8 European countries: Belgium, Germany, Finland, Great Britain, Italy, Luxembourg, The Netherlands and Poland. The Data are fully described in Mossong J, Hens N, Jit M, Beutels P, Auranen K, Mikolajczyk R, et al. (2008) Social Contacts and Mixing Patterns Relevant to the Spread of Infectious Diseases. PLoS Med 5(3): e74.

Usage

polymod

Format

A list of two data frames:

participants the study participant, with age, country, year and day of the week (starting with 1 = Monday)

contacts reported contacts of the study participants. The variable `phys_contact` has two levels (1 denotes physical contact while 2 denotes non-physical contact), `duration_multi` has five levels (1 is less than 5 minutes while 5 is more than 4 hours, increasing in the order found in Figure 1 in Mossong et al.), and `frequency_multi` has five levels (1 is daily, 2 is weekly, 3 is monthly, 4 is less often, and 5 is first time) All other variables are described on the Zenodo repository of the data, available at [doi:10.5281/zenodo.1043437](https://doi.org/10.5281/zenodo.1043437)

Source

[doi:10.1371/journal.pmed.0050074](https://doi.org/10.1371/journal.pmed.0050074)

rebin_ages

Rebin a population table to a set of age groups

Description

Rebins a population table to the coarser age groups defined by `age_limits`, summing populations into each group. Errors if `age_limits` are finer than the population data, since splitting a band would require assuming a within-band age distribution. Operates on an age column of age-group labels (e.g. from `limits_to_age_groups()` or `assign_age_groups()`) and returns the same form.

To align a population to a contact matrix's groupings for the post-processing functions, use `align_ages()` instead.

Usage

```
rebin_ages(pop, age_limits)
```

Arguments

<code>pop</code>	a data frame with an age column of age-group labels and a population column
<code>age_limits</code>	lower age limits of the (coarser) age groups to rebin to

Value

a data frame with an age column of age-group labels and a population column

Examples

```
it_pop <- data.frame(
  age = limits_to_age_groups(seq(0, 80, by = 5), notation = "brackets"),
  population = c(rep(2.5e6, 4), rep(3.5e6, 4), rep(5e6, 6), 5e6, 7e6, 4e6)
)
# rebin into 10-year age groups
rebin_ages(it_pop, age_limits = seq(0, 100, by = 10))
```

reduce_age_groups	<i>Reduce the number of age groups given a broader set of limits</i>
-------------------	--

Description

Operates on lower limits

Usage

```
reduce_age_groups(x, limits)
```

Arguments

x	vector of limits
limits	new limits

Value

vector with the new age groups

Examples

```
reduce_age_groups(seq_len(20), c(0, 5, 10))
```

split_matrix	<i>Decompose a contact matrix into mean contacts, normalisation and assortativity</i>
--------------	---

Description

Splits the contact matrix into the mean number of contacts across the whole population (`mean_contacts`), a normalisation constant (`normalisation`), age-specific contact rates (`contacts`), and an assortativity matrix (replacing `$matrix`). For details, see the "Getting Started" vignette.

Usage

```
split_matrix(x, survey_pop)
```

Arguments

x	a list as returned by <code>compute_matrix()</code> , with elements <code>matrix</code> and <code>participants</code>
survey_pop	a data frame; see <i>Population data</i> below

Details

`split_matrix()` supports single-grouping (rank-2) matrices only.

Value

x with \$matrix replaced by the assortativity matrix, plus additional elements \$mean.contacts, \$normalisation, and \$contacts

Population data

survey_pop is a data frame with one column per grouping, named after the grouping (e.g. age, gender) and holding that grouping's levels as they appear in the matrix, plus a population column with the size of each combination. One row per combination of levels is required, and levels are matched to the matrix exactly, without interpolation.

Use align_ages() to build this from a raw population table: it aggregates each grouping to the matrix's levels (interpolating the age grouping where needed) and labels the columns to match.

Examples

```
data(polymod)
result <- polymod |>
  (\(s) s[country == "United Kingdom"])( ) |>
  assign_age_groups(age_limits = c(0, 5, 15)) |>
  compute_matrix()
uk_pop <- data.frame(
  age = limits_to_age_groups(0:80, notation = "brackets"),
  population = rep(1e5, 81)
)
result |> split_matrix(survey_pop = align_ages(uk_pop, result))
```

survey	Contact survey (defunct)
--------	--------------------------

Description

[Defunct]
survey() is defunct. Use as_contact_survey() instead.

Usage

```
survey(participants, contacts, reference = NULL)
```

Arguments

- participants a data.frame containing information on participants
- contacts a data.frame containing information on contacts
- reference a list containing information needed to reference the survey, in particular it can contain a "title", "bibtype", "author", "doi", "publisher", "note", "year"

Value

Always errors.

Author(s)

Sebastian Funk

See Also

[as_contact_survey\(\)](#)

survey_countries	<i>List all countries contained in a survey</i>
------------------	---

Description

[Defunct]

Usage

```
survey_countries(survey, country.column = "country", ...)
```

Arguments

survey a DOI or url to get the survey from, or a [survey\(\)](#) object.
country.column column in the survey indicating the country
... further arguments for [get_survey\(\)](#)

Details

survey_countries() is defunct. Use `contactsurveys::download_survey()` and [load_survey\(\)](#) and then explore the country column yourself.

Value

Always errors.

Examples

```
## Not run:
data(polymod)
survey_countries(polymod)

## End(Not run)
## --> we now recommend
## Not run:
doi_peru <- "10.5281/zenodo.1095664" # nolint
# download the data with the contactsurveys package
```

```

peru_survey <- contactsurveys::download_survey(doi_peru)
# load the survey with socialmixr
peru_data <- socialmixr::load_survey(peru_survey)
# find the unique country - assuming your data has a "country" column:
unique(peru_data$participants$country)

## End(Not run)

```

survey_country_population

Get survey country population data

Description

[Deprecated]

This function is deprecated alongside [wpp_age\(\)](#), which it wraps. The underlying wpp2017 data is outdated. Construct a `data.frame` with columns `lower.age.limit` and `population` from a current source (e.g. the wpp2024 package from GitHub) and pass it to [contact_matrix\(\)](#) via the `survey_pop` argument instead.

Usage

```
survey_country_population(survey, countries = NULL)
```

Arguments

<code>survey</code>	A survey() object, with column "country" in "participants".
<code>countries</code>	Optional. A character vector of country names. If specified, this will be used instead of the potential "country" column in "participants".

Value

A data table with population data by age group for the survey countries, aggregated by lower age limit. The function will error if no country information is available from either the survey or countries argument.

Examples

```

if (requireNamespace("wpp2017", quietly = TRUE)) {
  survey_country_population(polymod, countries = "Belgium")
}

```

symmetrise

Symmetrise a contact matrix

Description

Makes a contact matrix symmetric so that $c_{ab}N_a = c_{ba}N_b$, where c_{ab} is the (a, b) entry and N_a is the population of group a. Each pair is replaced by half their sum, weighted by population size. Reciprocity requires that each grouping has the same levels on the participant and contact side; if not, the function aborts.

Usage

```
symmetrise(x, survey_pop, symmetric_norm_threshold = 2)
```

Arguments

x a list as returned by `compute_matrix()`, with elements `matrix` and `participants`

survey_pop a data frame; see *Population data* below

symmetric_norm_threshold threshold for the normalisation factor before issuing a warning (default 2)

Value

x with `$matrix` replaced by the symmetrised version

Population data

`survey_pop` is a data frame with one column per grouping, named after the grouping (e.g. age, gender) and holding that grouping's levels as they appear in the matrix, plus a population column with the size of each combination. One row per combination of levels is required, and levels are matched to the matrix exactly, without interpolation.

Use `align_ages()` to build this from a raw population table: it aggregates each grouping to the matrix's levels (interpolating the age grouping where needed) and labels the columns to match.

Examples

```
data(polymod)
result <- polymod |>
  (\(s) s[country == "United Kingdom"]()) |>
  assign_age_groups(age_limits = c(0, 5, 15)) |>
  compute_matrix()
uk_pop <- data.frame(
  age = limits_to_age_groups(0:80, notation = "brackets"),
  population = rep(1e5, 81)
)
result |> symmetrise(survey_pop = align_ages(uk_pop, result))
```

weigh	<i>Weigh survey participants</i>
-------	----------------------------------

Description

`weigh()` multiplies participant weights by values looked up from a target. The existing weight column is multiplied in place, so multiple calls compose; if no weight column is present, one is created with value 1.

`weigh_by_dayofweek()` and `weigh_by_age()` are thin convenience wrappers around the two most common recipes — the weekly weekday/weekend split and age post-stratification against a reference population. See the dedicated sections below for what they compute exactly.

Usage

```
weigh(survey, by, target = NULL, groups = NULL, ...)
```

```
weigh_by_dayofweek(survey)
```

```
weigh_by_age(survey, pop, ...)
```

Arguments

survey	a <code>survey()</code> object
by	column name in the participant data to join on
target	see <i>Target shapes accepted by weigh()</i> .
groups	a list of value sets mapping column values to groups (used with an unnamed numeric target vector); must be the same length as target.
...	ignored.
pop	a data frame with columns age (age-group labels) and population (used by <code>weigh_by_age()</code>).

Value

the survey object with updated participant weights

Target shapes accepted by `weigh()`

- `target = NULL` (the default) — multiply the numeric column by directly into weight. Useful when participants already carry a precomputed weight column.
- a two-column data frame whose key column is named `by` — pure discrete join: multiply the value column into weight where the key matches. Unmatched values get NA (with a warning).
- an unnamed numeric vector together with `groups` — each element of `target` is the *total* weight assigned across participants matching the corresponding entry in `groups`. The per-participant factor is `target[g] / n_in_group`.

- a named numeric vector — same as above but `names(target)` are matched against values of the `by` column.

A data frame `target` that does *not* have a column named `by` but does have `lower.age.limit` and `population` triggers a deprecation warning and falls back to the old hidden age post-stratification path; use `weigh_by_age()` instead.

`weigh_by_dayofweek()`

Rescales weights so that weekday participants together carry a total weight of 5 and weekend participants a total weight of 2 — the weekly 5/2 split that corrects for the typical over-representation of weekdays in diary surveys. Concretely, each weekday participant gets $5 / n_{\text{weekday}}$ and each weekend participant $2 / n_{\text{weekend}}$; participants with NA day-of-week get the neutral average $7 / N$. The `dayofweek` column is taken to use 0 = Sunday through 6 = Saturday (the POLYMOD convention).

Equivalent to: `weigh(survey, "dayofweek", target = c(5, 2), groups = list(1:5, c(0, 6)))`

`weigh_by_age()`

Convenience wrapper for age post-stratification. Participants are binned into the reference population's own age bands (whatever resolution `pop` is supplied at) and, for each band b , the weight becomes

$$w_b = \frac{P_b/P}{N_b/N},$$

where P_b is the target population in band b , P the total, and N_b , N the corresponding sample counts. No interpolation is performed, so the weighting resolution is that of the supplied reference population.

`survey` must already have been processed by `assign_age_groups()` so that a `part_age` column is available for the join.

Examples

```
data(polymod)
uk <- polymod[country == "United Kingdom"] |>
  assign_age_groups(age_limits = c(0, 5, 15))

# --- target = NULL ---
# Multiply an existing numeric column directly into the weight:
uk |> weigh("hh_size")

# --- data-frame target (discrete join) ---
# The key column of `target` must match `by`. Each participant
# has its weight multiplied by the matching value column.
age_target <- data.frame(
  age.group = c("[0,5)", "[5,15)", "[15,Inf)"),
  p = c(0.06, 0.12, 0.82)
)
uk |> weigh("age.group", target = age_target)
```

```

# Same idea, joining on `country` to pool participants across studies
# by a target population share:
country_target <- data.frame(
  country = c("United Kingdom", "Germany", "Italy"),
  p = c(0.3, 0.4, 0.3)
)
polymod |>
  assign_age_groups(age_limits = c(0, 5, 15)) |>
  weigh("country", target = country_target)

# --- unnamed vector + groups (total-weight semantics) ---
# Each `target[g]` is the *total* weight assigned to participants in
# `groups[[g]]`. Here weekdays together carry weight 5, weekend days
# together carry weight 2:
uk |> weigh("dayofweek", target = c(5, 2), groups = list(1:5, c(0, 6)))

# The same is available as the convenience:
uk |> weigh_by_dayofweek()

# --- named vector ---
# `names(target)` are matched against `by` values; each value is the
# total weight for participants with that key.
uk$participants[, agecat := ifelse(part_age < 18, "child", "adult")]
uk |> weigh("agecat", target = c(child = 0.25, adult = 0.75))

# --- age post-stratification ---
uk_pop <- data.frame(
  age = limits_to_age_groups(c(0, 5, 15, 65), notation = "brackets"),
  population = c(3500000, 6000000, 40000000, 10000000)
)
uk |> weigh_by_age(uk_pop)

```

wpp_age

*Get age-specific population data according to the World Population
Prospects 2017 edition*

Description

[Deprecated]

This function is deprecated in favour of passing population data directly to `contact_matrix()` via the `survey_pop` argument. Additionally, the underlying wpp2017 data is outdated. For more recent population data, use the wpp2024 package from GitHub.

Usage

```
wpp_age(countries, years)
```

Arguments

countries countries, will return all if not given
 years years, will return all if not given

Details

This uses data from the wpp2017 package but combines male and female, and converts age groups to lower age limits. If the requested year is not present in the historical data, WPP projections are used.

Value

data frame of age-specific population data

Examples

```
if (requireNamespace("wpp2017", quietly = TRUE)) {
  wpp_age("Italy", c(1990, 2000))
}

# For more recent data, use wpp2024 from GitHub:
# remotes::install_github("PPgp/wpp2024")
# library(wpp2024)
# data(popAge1dt)
# uk_pop <- popAge1dt[name == "United Kingdom" & year == 2020,
#                      .(lower.age.limit = age, population = pop * 1000)]
# contact_matrix(polymod, countries = "United Kingdom", survey_pop = uk_pop)
```

wpp_countries	<i>List all countries and regions for which socialmixr has population data</i>
---------------	--

Description**[Deprecated]**

This function is deprecated in favour of passing population data directly to `contact_matrix()` via the `survey_pop` argument, which removes the need for a country list. Additionally, the underlying wpp2017 data is outdated. For countries available in more recent WPP editions, use the wpp2024 package from GitHub.

Usage

```
wpp_countries()
```

Details

Uses the World Population Prospects data from the wpp2017 package.

Value

list of countries

Examples

```
if (requireNamespace("wpp2017", quietly = TRUE)) {  
  wpp_countries()  
}
```

Index

- * **datasets**
 - polymod, [23](#)
- [.contact_survey, [2](#)
- age_groups_to_limits, [3](#)
- align_ages, [4](#)
- align_ages(), [23](#), [24](#), [26](#), [29](#)
- as_contact_survey, [5](#)
- as_contact_survey(), [26](#), [27](#)
- assign_age_groups, [6](#)
- assign_age_groups(), [3](#), [9](#), [10](#), [24](#), [31](#)
- base::readRDS(), [17](#)
- base::saveRDS(), [17](#)
- check, [7](#)
- check(), [13](#)
- clean, [8](#)
- clean(), [20](#)
- compute_matrix, [9](#)
- compute_matrix(), [4](#), [15](#), [22](#), [25](#), [29](#)
- contact_age_distribution, [10](#)
- contact_matrix, [11](#)
- contact_matrix(), [7](#), [8](#), [21](#), [28](#), [32](#), [33](#)
- download_survey, [14](#)
- download_survey(), [17](#), [20](#)
- flatten, [15](#)
- get_citation, [16](#)
- get_survey, [17](#)
- get_survey(), [13](#), [27](#)
- hcl.colors(), [22](#)
- heat.colors(), [22](#)
- is_doi, [18](#)
- limits_to_age_groups, [18](#)
- limits_to_age_groups(), [3](#), [24](#)
- list_surveys, [19](#)
- load_survey, [20](#)
- load_survey(), [17](#), [27](#)
- matrix_plot, [21](#)
- per_capita, [22](#)
- per_capita(), [4](#), [9](#)
- polymod, [23](#)
- rebin_ages, [24](#)
- rebin_ages(), [4](#)
- reduce_age_groups, [25](#)
- split_matrix, [25](#)
- split_matrix(), [4](#), [9](#)
- survey, [26](#)
- survey(), [6–10](#), [12](#), [13](#), [17](#), [27](#), [28](#), [30](#)
- survey_countries, [27](#)
- survey_country_population, [28](#)
- symmetrise, [29](#)
- symmetrise(), [4](#), [9](#)
- Sys.sleep, [14](#)
- terrain.colors(), [22](#)
- topo.colors(), [22](#)
- weigh, [30](#)
- weigh(), [9](#), [30](#)
- weigh_by_age (weigh), [30](#)
- weigh_by_age(), [30](#), [31](#)
- weigh_by_dayofweek (weigh), [30](#)
- weigh_by_dayofweek(), [30](#)
- wpp_age, [32](#)
- wpp_age(), [12](#), [28](#)
- wpp_countries, [33](#)