Package 'diyar'

November 13, 2023

Type Package

Title Record Linkage and Epidemiological Case Definitions in 'R'

Date 2023-11-12

Version 0.5.1

URL https://olisansonwu.github.io/diyar/index.html

BugReports https://github.com/OlisaNsonwu/diyar/issues

Author Olisaeloka Nsonwu

Maintainer Olisaeloka Nsonwu <olisa.nsonwu@gmail.com>

Description An R package for iterative and batched record linkage,
and applying epidemiological case definitions.
'diyar' can be used for deterministic and probabilistic record linkage,
or multistage record linkage combining both approaches.
It features the implementation of nested match criteria, and mechanisms to
address missing data and conflicting matches during stepwise record linkage.
Case definitions are implemented by assigning records to groups based on
match criteria such as person or place, and overlapping time or duration of
events e.g. sample collection dates or periods of hospital stays.
Matching records are assigned a unique group ID. Index and duplicate records
are removed or further analyses as required.

License GPL-3

Encoding UTF-8

LazyData true

Imports methods, utils, ggplot2, rlang

RoxygenNote 7.2.3

Suggests knitr, rmarkdown, testthat, covr

VignetteBuilder knitr

Language en-GB

NeedsCompilation no

Depends R (>= 3.5.0)

Repository CRAN

Date/Publication 2023-11-12 23:13:18 UTC

R topics documented:

`attr_eval` *Sub-criteria attributes.*

Description

Recursive evaluation of a function (`func`) on each attribute (vector) in a `sub_criteria`.

Usage

```
attr_eval(x, func = length, simplify = TRUE)
```

Arguments

x	[sub_criteria]
func	[function]
simplify	If TRUE (default), coerce to a vector.

Value

vector; list

Examples

```
x <- sub_criteria(rep(1, 5), rep(5 * 10, 5))
attr_eval(x)
attr_eval(x, func = max)
attr_eval(x, func = max, simplify = FALSE)
attr_eval(sub_criteria(x, x), func = max, simplify = FALSE)
```

---

bys_funcs *Vectorised approach to group operations.*

Description

Vectorised approach to group operations.

Usage

```
bys_count(by)

bys_rank(..., by = NULL, from_last = FALSE)

bys_position(val, by = NULL, from_last = FALSE, ordered = TRUE)

bys_val(..., val, by = NULL, from_last = FALSE)

bys_nval(..., val, by = NULL, from_last = FALSE, n = 1, nmax = FALSE)

bys_min(val, by = NULL, na.rm = TRUE)

bys_max(val, by = NULL, na.rm = TRUE)

bys_sum(val, by = NULL, na.rm = TRUE)

bys_prod(val, by = NULL, na.rm = TRUE)

bys_cummin(val, by = NULL, na.rm = TRUE)

bys_cummax(val, by = NULL, na.rm = FALSE)
```

```
bys_cumsum(val, by = NULL, na.rm = TRUE)

bys_cumprod(val, by = NULL, na.rm = TRUE)

bys_lag(val, by = NULL, n = 1)

bys_lead(val, by = NULL, n = 1)
```

Arguments

by	[atomic]. Groups.
...	[atomic]. Sort levels
from_last	[logical] Sort order - TRUE (descending) or FALSE (ascending).
val	[atomic]. Value
ordered	If TRUE, values are sequential.
n	[integer] Position.
nmax	[logical] If TRUE, use length([by]) when n is greater than the number of records in a group.
na.rm	If TRUE, remove NA values

Value

[atomic]

Examples

```
x <- data.frame(
  group = c(2, 2, 1, 2, 1, 1, 1, 2, 1, 1),
  value = c(13, 14, 20, 9, 2, 1, 8, 18, 3, 17))

bys_count(x$group)
bys_position(x$value, by = x$group, from_last = TRUE)
bys_rank(by = x$group, val = x$value, from_last = TRUE)
bys_val(x$value, by = x$group, val = x$value, from_last = TRUE)
bys_nval(x$value, by = x$group, val = x$value, from_last = TRUE, n = 2)
bys_min(by = x$group, val = x$value)
bys_max(by = x$group, val = x$value)
bys_sum(by = x$group, val = x$value)
bys_prod(by = x$group, val = x$value)
bys_cummin(by = x$group, val = x$value)
bys_cummax(by = x$group, val = x$value)
bys_cumsum(by = x$group, val = x$value)
bys_cumprod(by = x$group, val = x$value)
bys_lag(by = x$group, val = x$value)
bys_lead(by = x$group, val = x$value)
```

combi *Vector combinations*

Description

Numeric codes for unique combination of vectors.

Usage

```
combi(...)
```

Arguments

`...`	[atomic]

Value

numeric

Examples

```
x <- c("A", "B", "A", "C", "B", "B")
y <- c("X", "X", "Z", "Z", "X", "Z")
combi(x, y)

# The code above is equivalent to but quicker than the one below.
z <- paste0(y, "-", x)
z <- match(z, z)
z
```

custom_sort *Nested sorting*

Description

Returns a sort order after sorting by a vector within another vector.

Usage

```
custom_sort(..., decreasing = FALSE, unique = FALSE)
```

Arguments

`...`	Sequence of `atomic` vectors. Passed to `order`.
`decreasing`	Sort order. Passed to `order`.
`unique`	If `FALSE` (default), ties get the same rank. If `TRUE`, ties are broken.

Value

`numeric` sort order.

Examples

```
a <- c(1, 1, 1, 2, 2)
b <- c(2, 3, 2, 1, 1)

custom_sort(a, b)
custom_sort(b, a)
custom_sort(b, a, unique = TRUE)
```

`delink` *Unlink group identifiers*

Description

Unlink records from an episode (`epid`), record group (`pid`) or pane (`pane`) object.

Usage

```
delink(x, lgk, ...)

## S3 method for class 'epid'
delink(x, lgk, ...)

## S3 method for class 'pane'
delink(x, lgk, ...)

## S3 method for class 'pid'
delink(x, lgk, ...)
```

Arguments

`x`	`[epid|pid|pane]`
`lgk`	`[logical]`. Subset of records to unlink.
`...`	Other arguments.

Value

epid; pid; pane

Examples

```
ep <- episodes(1:8)
unlinked_ep <- delink(ep, ep@sn %in% c(3, 8))
ep; unlinked_ep

pn <- partitions(1:8, length.out = 2, separate = TRUE)
unlinked_pn <- delink(pn, pn@.Data == 5)
pn; unlinked_pn

pd <- links(list(c(1, 1, 1, NA, NA),
                 c(NA, NA, 2, 2, 2)))
unlinked_pd <- delink(pd, pd@pid_cri == 1)
pd; unlinked_pd

# A warning is given if an index record is unlinked as this will lead to seemly impossible links.
ep2 <- episodes(1:8, 2, episode_type = "rolling")
unlinked_ep2 <- delink(ep2, ep2@sn %in% c(3, 5))
schema(ep2, custom_label = decode(ep2@case_nm), seed = 2)
schema(unlinked_ep2, custom_label = decode(unlinked_ep2@case_nm), seed = 2)
```

---

d_report *d_report*

Description

d_report

Usage

```
## S3 method for class 'd_report'
plot(
  x,
  ...,
  metric = c("cumulative_duration", "duration", "max_memory", "records_checked",
    "records_skipped", "records_assigned")
)

## S3 method for class 'd_report'
as.list(x, ...)

## S3 method for class 'd_report'
as.data.frame(x, ...)
```

Arguments

x	[d_report].
...	Arguments passed to other methods
metric	Report information

encode *Labelling in* diyar

Description

Encode and decode character and numeric values.

Usage

```
encode(x, ...)

decode(x, ...)

## Default S3 method:
encode(x, ...)

## S3 method for class 'd_label'
encode(x, ...)

## Default S3 method:
decode(x, ...)

## S3 method for class 'd_label'
decode(x, ...)

## S3 method for class 'd_label'
rep(x, ...)

## S3 method for class 'd_label'
x[i, ..., drop = TRUE]

## S3 method for class 'd_label'
x[[i, ..., drop = TRUE]]
```

Arguments

x	[d_label|atomic]
...	Other arguments.
i	i
drop	drop

Details

To minimise memory usage, most components of pid, epid and pane are integer objects with labels. encode() and decode() translates these codes and labels as required.

Value

d_label; atomic

Examples

```
cds <- encode(rep(LETTERS[1:5], 3))
cds

nms <- decode(cds)
nms
```

---

epid-class epid *object*

Description

S4 objects storing the result of episodes.

Usage

```
is.epid(x)

as.epid(x, ...)

## S3 method for class 'epid'
format(x, ...)

## S3 method for class 'epid'
unique(x, ...)

## S3 method for class 'epid'
summary(object, ...)

## S3 method for class 'epid_summary'
print(x, ...)

## S3 method for class 'epid'
as.data.frame(x, ..., decode = TRUE)

## S3 method for class 'epid'
as.list(x, ..., decode = TRUE)

## S4 method for signature 'epid'
show(object)

## S4 method for signature 'epid'
rep(x, ...)
```

```
## S4 method for signature 'epid'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'epid'
x[[i, j, ..., exact = TRUE]]

## S4 method for signature 'epid'
c(x, ...)
```

Arguments

x	x
...	...
object	object
decode	If TRUE, data is decoded
i	i
j	j
drop	drop
exact	exact

Slots

`sn` Unique record identifier.

`.Data` Unique episode identifier.

`wind_id` Unique reference ID for each match.

`wind_nm` Type of window i.e. "Case" or "Recurrence".

`case_nm` Record type in regards to case assignment.

`dist_wind_index` Unit difference between each record and its window's reference record.

`dist_epid_index` Unit difference between each record and its episode's reference record.

`epid_dataset` Data sources in each episode.

`epid_interval` The start and end dates of each episode. A number_line object.

`epid_length` The duration or length of (epid_interval).

`epid_total` The number of records in each episode.

`iteration` The iteration when a record was matched to it's group (.Data).

`options` Some options passed to the instance of episodes.

Examples

```
# A test for `epid` objects
ep <- episodes(date = 1)
is.epid(ep); is.epid(2)

ep <- episodes(date = 1)
is.epid(ep); is.epid(2)
```

episodes *Group dated events into episodes.*

Description

Dated events (records) within a certain duration of an index event are assigned to a unique group. Each group has unique ID and are described as "episodes". "episodes" can be "fixed" or "rolling" ("recurring"). Each episodes has a "Case" and/or "Recurrent" record while all other records within the group are either "Duplicates" of the "Case" or "Recurrent" event.

Usage

```
episodes(
  date,
  case_length = Inf,
  episode_type = "fixed",
  recurrence_length = case_length,
  episode_unit = "days",
  strata = NULL,
  sn = NULL,
  episodes_max = Inf,
  rolls_max = Inf,
  case_overlap_methods = 8,
  recurrence_overlap_methods = case_overlap_methods,
  skip_if_b4_lengths = FALSE,
  data_source = NULL,
  data_links = "ANY",
  custom_sort = NULL,
  skip_order = Inf,
  reference_event = "last_record",
  case_for_recurrence = FALSE,
  from_last = FALSE,
  group_stats = c("case_nm", "wind", "epid_interval"),
  display = "none",
  case_sub_criteria = NULL,
  recurrence_sub_criteria = case_sub_criteria,
  case_length_total = 1,
  recurrence_length_total = case_length_total,
  skip_unique_strata = TRUE,
  splits_by_strata = 1,
  batched = "semi"
)

links_wf_episodes(
  date,
  case_length = Inf,
  episode_type = "fixed",
```

```
  strata = NULL,
  sn = NULL,
  display = "none"
)

episodes_af_shift(
  date,
  case_length = Inf,
  sn = NULL,
  strata = NULL,
  group_stats = FALSE,
  episode_type = "fixed",
  data_source = NULL,
  episode_unit = "days",
  data_links = "ANY",
  display = "none"
)
```

Arguments

`date`	`[date|datetime|integer|number_line]`. Record date or period.
`case_length`	`[integer|number_line]`. Duration from an index event distinguishing one `"Case"` from another.
`episode_type`	`[character]`. Options are `"fixed"` (default) or `"rolling"`. See `Details`.
`recurrence_length`	`[integer|number_line]`. Duration from an index event distinguishing a `"Recurrent"` event from its `"Case"` or prior `"Recurrent"` event.
`episode_unit`	`[character]`. Unit of time for `case_length` and `recurrence_length`. Options are "seconds", "minutes", "hours", "days" (default), "weeks", "months" or "years". See `diyar::episode_unit`.
`strata`	`[atomic]`. Subsets of the dataset. Episodes are created separately by each `strata`.
`sn`	`[integer]`. Unique record ID.
`episodes_max`	`[integer]`. Maximum number of episodes permitted within each `strata`.
`rolls_max`	`[integer]`. Maximum number of times an index event can recur. Only used if `episode_type` is `"rolling"`.
`case_overlap_methods`	`[character|integer]`. Specific ways a period (record) most overlap with a `"Case"` event. See (`overlaps`).
`recurrence_overlap_methods`	`[character|integer]`. Specific ways a period (record) most overlap with a `"Recurrent"` event. See (`overlaps`).
`skip_if_b4_lengths`	`[logical]`. If `TRUE` (default), events before a lagged `case_length` or `recurrence_length` are skipped.

data_source [character]. Source ID for each record. If provided, a list of all sources in each episode is returned. See epid_dataset slot.

data_links [list|character]. data_source required in each epid. An episode without records from these data_sources will be unlinked. See Details.

custom_sort [atomic]. Preferential order for selecting index events. See custom_sort.

skip_order [integer]. End episode tracking in a strata when the an index event's custom_sort order is greater than the supplied skip_order.

reference_event
[character]. Specifies which of the records are used as index events. Options are "last_record" (default), "last_event", "first_record" or "first_event".

case_for_recurrence
[logical]. If TRUE, a case_length is applied to both "Case" and "Recurrent" events. If FALSE (default), a case_length is applied to only "Case" events.

from_last [logical]. Track episodes beginning from the earliest to the most recent record (FALSE) or vice versa (TRUE).

group_stats [character]. A selection of group metrics to return for each episode. Most are added to slots of the epid object. Options are NULL or any combination of "case_nm", "wind" and "epid_interval".

display [character]. Display progress update and/or generate a linkage report for the analysis. Options are; "none" (default), "progress", "stats", "none_with_report", "progress_with_report" or "stats_with_report".

case_sub_criteria
[sub_criteria]. Additional nested match criteria for events in a case_length.

recurrence_sub_criteria
[sub_criteria]. Additional nested match criteria for events in a recurrence_length.

case_length_total
[integer|number_line]. Minimum number of matched case_lengths required for an episode.

recurrence_length_total
[integer|number_line]. Minimum number of matched recurrence_lengths required for an episode.

skip_unique_strata
[logical]. If TRUE, a strata with a single event is skipped.

splits_by_strata
[integer]. Split analysis into n parts. This typically lowers max memory usage but increases run time.

batched [character]. Create and compare records in batches. Options are "yes", "no", and "semi". typically, the ("semi") option will have a higher max memory and shorter run-time while ("no") will have a lower max memory but longer run-time

Details

episodes() links dated records (events) that are within a set duration of each other in iterations. Every record is linked to a unique group (episode; epid object). These episodes represent occurrences of interest as specified by function's arguments and defined by a case definition.

Two main type of episodes are possible;

- "fixed" - An episode where all events are within a fixed duration of an index event.
- "rolling" - An episode where all events are within a recurring duration of an index event.

Every record in each episode is categorised as one of the following;

- "Case" - Index event of the episode (without a nested match criteria).
- "Case_CR" - Index event of the episode (with a nested match criteria).
- "Duplicate_C" - Duplicate of the index event.
- "Recurrent" - Recurrence of the index event (without a nested match criteria).
- "Recurrent_CR" - Recurrence of the index event (with a nested match criteria).
- "Duplicate_R" - Duplicate of the recurrent event.
- "Skipped" - Skipped records.

If data_links is supplied, every element of the list must be named "l" (links) or "g" (groups). Unnamed elements are assumed to be "l".

- If named "l", groups without records from every listed data_source will be unlinked.
- If named "g", groups without records from any listed data_source will be unlinked.

All records with a missing (NA) strata or date are skipped.

Wrapper functions or alternative implementations of episodes() for specific use cases or benefits:

- episodes_wf_splits() - Identical records are excluded from the main analysis.
- episodes_af_shift() - A mostly vectorised approach.
- links_wf_episodes() - The same functionality achieved with links.

See vignette("episodes") for further details.

Value

epid; list

See Also

episodes_wf_splits; custom_sort; sub_criteria; epid_length; epid_window; partitions; links; overlaps;

Examples

```
data(infections)
data(hospital_admissions)

# One 16-day (15-day difference) fixed episode per type of infection
episodes(date = infections$date,
         strata = infections$infection,
         case_length = 15,
         episodes_max = 1,
```

```
         episode_type = "fixed")

# Multiple 16-day episodes with an 11-day recurrence period
episodes(date = infections$date,
         strata = NULL,
         case_length = 15,
         episodes_max = Inf,
         episode_type = "rolling",
         recurrence_length = 10)

# Overlapping periods of hospital stays
dfr <- hospital_admissions[2:3]

dfr$admin_period <-
  number_line(dfr$admin_dt,dfr$discharge_dt)

dfr$ep <-
  episodes(date = dfr$admin_period,
           strata = NULL,
           case_length = index_window(dfr$admin_period),
           case_overlap_methods = "inbetween")

dfr
as.data.frame(dfr$ep)
```

`episodes_wf_splits` *Link events to chronological episodes.*

Description

`episodes_wf_splits` is a wrapper function of `episodes`. It's designed to be more efficient with larger datasets. Duplicate records which do not affect the case definition are excluded prior to episode tracking. The resulting episode identifiers are then recycled for the duplicate records.

Usage

```
episodes_wf_splits(..., duplicates_recovered = "ANY", reframe = FALSE)
```

Arguments

`...`	Arguments passed to `episodes`.
`duplicates_recovered`	`[character]`. Determines which duplicate records are recycled. Options are `"ANY"` (default), `"without_sub_criteria"`, `"with_sub_criteria"` or `"ALL"`. See `Details`.
`reframe`	`[logical]`. Determines if the duplicate records in a `sub_criteria` are reframed (`TRUE`) or excluded (`FALSE`).

Details

episodes_wf_splits() reduces or re-frames a dataset to the minimum datasets required to implement a case definition. This leads to the same outcome but with the benefit of a shorter processing time.

The duplicates_recovered argument determines which identifiers are recycled. Selecting the "with_sub_criteria" option will force only identifiers created resulting from a matched sub_criteria ("Case_CR" and "Recurrent_CR") are recycled. However, if "without_sub_criteria" is selected then only identifiers created that do not result from a matched sub_criteria ("Case" and "Recurrent") are recycled Excluded duplicates of "Duplicate_C" and "Duplicate_R" are always recycled.

The reframe argument will either reframe or subset a sub_criteria. Both will require slightly different functions for match_funcs or equal_funcs.

Value

epid; list

See Also

episodes; sub_criteria

Examples

```
# With 2,000 duplicate records of 20 events,
# `episodes_wf_splits()` will take less time than `episodes()`
dates <- seq(from = as.Date("2019-04-01"), to = as.Date("2019-04-20"), by = 1)
dates <- rep(dates, 2000)

system.time(
  ep1 <- episodes(dates, 1)
)
system.time(
  ep2 <- episodes_wf_splits(dates, 1)
)

# Both leads to the same outcome.
all(ep1 == ep2)
```

---

links *Multistage record linkage*

Description

Assign records to unique groups based on an ordered set of match criteria.

Usage

```
links(
  criteria,
  sub_criteria = NULL,
  sn = NULL,
  strata = NULL,
  data_source = NULL,
  data_links = "ANY",
  display = "none",
  group_stats = FALSE,
  expand = TRUE,
  shrink = FALSE,
  recursive = "none",
  check_duplicates = FALSE,
  tie_sort = NULL,
  batched = "yes",
  repeats_allowed = FALSE,
  permutations_allowed = FALSE,
  ignore_same_source = FALSE
)
```

Arguments

criteria	[list|atomic]. Ordered list of attributes to be compared. Each element of the list is a stage in the linkage process. See Details.
sub_criteria	[list|sub_criteria]. Nested match criteria. This must be paired to a stage of the linkage process (criteria). See sub_criteria
sn	[integer]. Unique record ID.
strata	[atomic]. Subsets of the dataset. Record-groups are created separately for each strata. See Details.
data_source	[character]. Source ID for each record. If provided, a list of all sources in each record-group is returned. See pid_dataset slot.
data_links	[list|character]. data_source required in each pid. A record-group without records from these data_sources will be unlinked. See Details.
display	[character]. Display progress update and/or generate a linkage report for the analysis. Options are; "none" (default), "progress", "stats", "none_with_report", "progress_with_report" or "stats_with_report".
group_stats	[character]. A selection of group specific information to be return for each record-group. Most are added to slots of the pid object. Options are NULL or any combination of "XX", "XX" and "XX".
expand	[logical]. If TRUE, a record-group gains new records if a match is found at the next stage of the linkage process. *Not interchangeable with* shrink.
shrink	[logical]. If TRUE, a record-group loses existing records if no match is found at the next stage of the linkage process. *Not interchangeable with* expand.
recursive	[logical]. If TRUE, within each iteration of the process, a match can spawn new matches. Ignored when batched is "no".

check_duplicates
: [logical]. If TRUE, within each iteration of the process, duplicates values of an attributes are not checked. The outcome of the logical test on the first instance of the value will be recycled for the duplicate values. Ignored when batched is "no".

tie_sort
: [atomic]. Preferential order for breaking match ties within an iteration of record linkage.

batched
: [character] Determines if record-pairs are created and compared in batches. Options are "yes", "no" or "semi".

repeats_allowed
: [logical] If TRUE, pairs made up of repeat records are not created and compared. Only used when batched is "no".

permutations_allowed
: [logical] If TRUE, permutations of record-pairs are created and compared. Only used when batched is "no".

ignore_same_source
: [logical] If TRUE, only records-pairs from a different data_source are created and compared.

Details

The priority of matches decreases with each subsequent stage of the linkage process. Therefore, the attributes in criteria should be in an order of decreasing relevance.

Records with missing data (NA) for each criteria are skipped at the respective stage, while records with missing data strata are skipped from every stage.

If a record is skipped from a stage, another attempt will be made to match the record at the next stage. If a record is still unmatched by the last stage, it is assigned a unique group ID.

A sub_criteria adds nested match criteria to each stage of the linkage process. If used, only records with a matching criteria and sub_criteria are linked.

In links, each sub_criteria must be linked to a criteria. This is done by adding each sub_criteria to a named element of a list - "cr" concatenated with the corresponding stage's number. For example, 3 sub_criteria linked to criteria 1, 5 and 13 will be;

$$list(cr1 = sub_criteria(...), cr5 = sub_criteria(...), cr13 = sub_criteria(...))$$

Any unlinked sub_criteria will be ignored.

Every element in data_links must be named "l" (links) or "g" (groups). Unnamed elements of data_links will be assumed to be "l".

- If named "l", groups without records from every listed data_source will be unlinked.
- If named "g", groups without records from any listed data_source will be unlinked.

See vignette("links") for more information.

Value

pid; list

See Also

links_af_probabilistic; episodes; predefined_tests; sub_criteria

Examples

```
data(patient_records)
dfr <- patient_records
# An exact match on surname followed by an exact match on forename
stages <- as.list(dfr[c("surname", "forename")])
p1 <- links(criteria = stages)

# An exact match on forename followed by an exact match on surname
p2 <- links(criteria = rev(stages))

# Nested matches
# Same sex OR birth year
m.cri.1 <- sub_criteria(
  format(dfr$dateofbirth, "%Y"), dfr$sex,
  operator = "or")

# Same middle name AND a 10 year age difference
age_diff <- function(x, y){
  diff <- abs(as.numeric(x) - as.numeric(y))
  wgt <-  diff %in% 0:10 & !is.na(diff)
  wgt
}
m.cri.2 <- sub_criteria(
  format(dfr$dateofbirth, "%Y"), dfr$middlename,
  operator = "and",
  match_funcs = c(age_diff, exact_match))

# Nested match criteria 'm.cri.1' OR 'm.cri.2'
n.cri <- sub_criteria(
  m.cri.1, m.cri.2,
  operator = "or")

# Record linkage with additional match criteria
p3 <- links(
  criteria = stages,
  sub_criteria = list(cr1 = m.cri.1,
                      cr2 = m.cri.2))

# Record linkage with additonal nested match criteria
p4 <- links(
  criteria = stages,
  sub_criteria = list(cr1 = n.cri,
                      cr2 = n.cri))

dfr$p1 <- p1; dfr$p2 <- p2
dfr$p3 <- p3; dfr$p4 <- p4

head(dfr)
```

link_wf *Record linkage*

Description

Deterministic and probabilistic record linkage Assign unique identifiers to records based on partial, nested or calculated probabilities.

Usage

```
links_af_probabilistic(
  attribute,
  blocking_attribute = NULL,
  cmp_func = diyar::exact_match,
  attr_threshold = 1,
  probabilistic = TRUE,
  m_probability = 0.95,
  u_probability = NULL,
  score_threshold = 1,
  repeats_allowed = FALSE,
  permutations_allowed = FALSE,
  data_source = NULL,
  ignore_same_source = TRUE,
  display = "none"
)

links_wf_probabilistic(
  attribute,
  blocking_attribute = NULL,
  cmp_func = diyar::exact_match,
  attr_threshold = 1,
  probabilistic = TRUE,
  m_probability = 0.95,
  u_probability = NULL,
  score_threshold = 1,
  id_1 = NULL,
  id_2 = NULL,
  return_weights = FALSE,
  ...
)

prob_score_range(attribute, m_probability = 0.95, u_probability = NULL)
```

Arguments

attribute	[atomic|list|data.frame|matrix|d_attribute]. Attributes to compare.
blocking_attribute	[atomic]. Passed to criteria in links.
cmp_func	[list|function]. String comparators for each attribute. See Details.
attr_threshold	[list|numeric|number_line]. Weight-thresholds for each cmp_func. See Details.
probabilistic	[logical]. If TRUE, scores are assigned base on Fellegi-Sunter model for probabilistic record linkage. See Details.
m_probability	[list|numeric]. The probability that a matching records are the same entity.
u_probability	[list|numeric]. The probability that a matching records are not the same entity.
score_threshold	[numeric|number_line]. Score-threshold for linked records. See Details.
repeats_allowed	[logical] Passed to repeats_allowed in links.
permutations_allowed	[logical] Passed to permutations_allowed in links.
data_source	[character]. Passed to data_source in links.
ignore_same_source	[logical] Passed to ignore_same_source in links.
display	[character]. Passed to display in links.
id_1	[list|numeric]. Record id or index of one half of a record-pair.
id_2	[list|numeric]. Record id or index of one half of a record-pair.
return_weights	If TRUE, returns the match-weights and score-thresholds for record pairs.
...	Arguments passed to links

Details

links_wf_probabilistic() - A wrapper function of links with a with a specific sub_criteria and to achieve to achieve probabilistic record linkage It excludes functionalities for the nested and multi-stage linkage. links_wf_probabilistic() requires a score_threshold in advance. To help with this, prob_score_range() can be used to return the range of scores attainable for a given set of attribute, m and u-probabilities. Additionally, id_1 and id_2 can be used to link specific records pairs, aiding the review of potential scores.

links_af_probabilistic() - A simpler version of links. It excludes functionalities for the batched, nested and multi-stage linkage. links_af_probabilistic() requires a score_threshold in advance, however, since it exports the match weights, the score_threshold can be changed after the analysis.

Value

pid; list

References

Fellegi, I. P., & Sunter, A. B. (1969). A Theory for Record Linkage. *Journal of the Statistical Association*, 64(328), 1183 - 1210. https://doi.org/10.1080/01621459.1969.10501049

Asher, J., Resnick, D., Brite, J., Brackbill, R., & Cone, J. (2020). An Introduction to Probabilistic Record Linkage with a Focus on Linkage Processing for WTC Registries. *International journal of environmental research and public health*, 17(18), 6937. https://doi.org/10.3390/ijerph17186937.
See `vignette("links")` for more information.

See Also

`links`

Examples

```
data(patient_records)
# Weighted (probabilistic) comparison of forename, middlename and surname
criteria_1 <- as.list(patient_records[c("forename", "middlename", "surname")])

# Possible scores when m-probability is 0.95
prob_scores <- prob_score_range(attribute = criteria_1,
                                m_probability = 0.95,
                                u_probability = NULL)
## Not run:
# Probabilistic record linkage with 'links_af_probabilistic()'
pids_1a <- links_af_probabilistic(attribute = criteria_1,
                        cmp_func = exact_match,
                        attr_threshold = 1,
                        probabilistic = TRUE,
                        m_probability = 0.95,
                        score_threshold = prob_scores$mid_scorce,
                        display = "stats")

# Equivalent with 'links_wf_probabilistic()'
pids_1b <- links_wf_probabilistic(attribute = criteria_1,
                                  cmp_func = exact_match,
                                  attr_threshold = 1,
                                  probabilistic = TRUE,
                                  m_probability = 0.95,
                                  score_threshold = prob_scores$mid_scorce,
                                  display = "progress",
                                  recursive = TRUE,
                                  check_duplicates = TRUE)

# Less thorough but faster equivalent with `links_wf_probabilistic()`
pids_1c <- links_wf_probabilistic(attribute = criteria_1,
                                  cmp_func = exact_match,
                                  attr_threshold = 1,
                                  probabilistic = TRUE,
                                  m_probability = 0.95,
                                  score_threshold = prob_scores$mid_scorce,
                                  display = "progress",
```

```
                                    recursive = FALSE,
                                    check_duplicates = FALSE)

# Each implementation can lead to different results
summary(pids_1a$pid)
summary(pids_1b$pid)
summary(pids_1c$pid)

## End(Not run)

# Weighted (non-probabilistic) comparison of forename, middlename and age difference
criteria_2 <- as.list(patient_records[c("forename", "middlename", "dateofbirth")])
age_diff <- function(x, y){
  diff <- abs(as.numeric(x) - as.numeric(y))
  wgt <-  diff %in% 0:(365 * 10) & !is.na(diff)
  wgt
}

pids_2a <- links_af_probabilistic(attribute = criteria_2,
                        blocking_attribute = patient_records$surname,
                        cmp_func = c(exact_match, exact_match, age_diff),
                        score_threshold = number_line(3, 5),
                        probabilistic = FALSE,
                        display = "stats")

# Larger weights can be assigned to particular attributes through `cmp_func`
# For example, a smaller age difference can contribute a higher score (e.g 0 to 3)
age_diff_2 <- function(x, y){
  diff <- as.numeric(abs(x - y))
  wgt <-  diff %in% 0:(365 * 10) & !is.na(diff)
  wgt[wgt] <- match(as.numeric(cut(diff[wgt], 3)), 3:1)
  wgt
}
pids_2b <- links_af_probabilistic(attribute = criteria_2,
                        blocking_attribute = patient_records$surname,
                        cmp_func = c(exact_match, exact_match, age_diff_2),
                        score_threshold = number_line(3, 5),
                        probabilistic = FALSE,
                        display = "stats")

head(pids_2a$pid_weights, 10)
head(pids_2b$pid_weights, 10)
```

`listr` *Grammatical lists.*

Description

A convenience function to format `atomic` vectors as a written list.

Usage

```
listr(x, sep = ", ", conj = " and ", lim = Inf)
```

Arguments

x	atomic vector.
sep	Separator.
conj	Final separator.
lim	Elements to include in the list. Other elements are abbreviated to " ...".

Value

character.

Examples

```
listr(1:5)
listr(1:5, sep = "; ")
listr(1:5, sep = "; ", conj = " and")
listr(1:5, sep = "; ", conj = " and", lim = 2)
```

---

make_ids *Convert an edge list to record identifiers.*

Description

Convert an edge list to record identifiers.

Usage

```
make_ids(x_pos, y_pos, id_length = max(x_pos, y_pos))
```

Arguments

x_pos	[integer]. Index of first half of a record-pair.
y_pos	[integer]. Index of second half of a record-pair.
id_length	Length of the record identifier.

Details

Record groups from non-recursive links have the lowest record ID (sn) in the set as their group ID.

Value

list

Examples

```
make_ids(x_pos = rep(7, 7), y_pos = 1:7)
make_ids(x_pos = c(1, 6), y_pos = 6:7)
make_ids(x_pos = 1:5, y_pos = c(1, 1, 2, 3, 4))
```

`make_pairs` *Combinations and permutations of record-sets.*

Description

Combinations and permutations of record-sets.

Usage

```
sets(n, r, permutations_allowed = TRUE, repeats_allowed = TRUE)

make_sets(
  x,
  r,
  strata = NULL,
  permutations_allowed = TRUE,
  repeats_allowed = TRUE
)

make_pairs(
  x,
  strata = NULL,
  repeats_allowed = TRUE,
  permutations_allowed = FALSE
)

make_pairs_wf_source(..., data_source = NULL)
```

Arguments

`n`	`[integer]`. Size of Vector.
`r`	`[integer]`. Number of elements in a set.
`permutations_allowed`	`[logical]`. If `TRUE`, permutations of the same set are included.
`repeats_allowed`	`[logical]`. If `TRUE`, repeat values are included in each set.
`x`	`[atomic]`. Vector.
`strata`	Subsets of `x`. Blocking attribute. Limits the creation of combinations or permutations to those from the same `strata`.
`...`	Arguments passed to `make_pairs`.
`data_source`	`[character]`. Data source identifier. Limits the creation of combinations or permutations to those from a different `data_source`

Details

`set()` - Create `r`-set combinations or permutations of `n` observations.

`make_set()` - Create `r`-set combinations or permutations of vector `x`.

`make_pairs()` - Create 2-set combinations or permutations of vector `x`.

`make_pairs_wf_source()` - Create 2-set combinations or permutations of vector `x` that are from different sources (`data_source`).

Value

A `list` of a vector's elements and corresponding indexes.

See Also

`eval_sub_criteria`

Examples

```
sets(4, 2)
sets(4, 2, repeats_allowed = FALSE, permutations_allowed = FALSE)
make_sets(month.abb[1:4], 2)
make_sets(month.abb[1:4], 3)

make_pairs(month.abb[1:4])
make_pairs(month.abb[1:4], strata = c(1, 1, 2, 2))
make_pairs_wf_source(month.abb[1:4], data_source = c(1, 1, 2, 2))
```

---

make_s4_ids *Create* `epid` *and* `pid` *objects with index of matching records*

Description

Create `epid` and `pid` objects with index of matching records

Usage

```
make_episodes(
  x_pos,
  y_pos,
  x_val,
  date,
  case_nm,
  wind_id,
  wind_nm,
  from_last,
  data_source,
```

```
  data_links,
  iteration,
  options,
  episode_unit
)

make_pids(
  x_pos,
  y_pos,
  x_val,
  link_id,
  pid_cri,
  data_source,
  data_links,
  iteration
)
```

Arguments

x_pos	[integer]. Index of one half of a record pair.
y_pos	[integer]. Index of one half of a record pair.
x_val	[integer]. Value of one half of a record pair.
date	[date|datetime|integer|number_line]. Record date or period.
case_nm	[integer|character] Record type in regards to case assignment (sub_criteria[Encoded]).
wind_id	[integer]. Unique reference ID for each match.
wind_nm	[list]. Type of window i.e. "Case" or "Recurrence".
from_last	[logical]. Chronological order of episode tracking i.e. ascending (TRUE) or descending (FALSE).
data_source	[character]. Source ID for each record.
data_links	[list|character]. data_source required in each record-group. A record-group without records from these data_sources will be unlinked.
iteration	The iteration when a record was matched to it's group (.Data).
options	[list]. Some options passed to the instance of episodes.
episode_unit	[character]. Time unit for case_length and recurrence_length. See episodes
link_id	[integer]. Unique reference ID for each match.
pid_cri	Match stage of the step-wise linkage.

merge_identifiers *Merge group identifiers*

Description

Consolidate two group identifiers.

Usage

```
merge_ids(...)

## Default S3 method:
merge_ids(id1, id2, tie_sort = NULL, expand = TRUE, shrink = FALSE, ...)

## S3 method for class 'pid'
merge_ids(id1, id2, tie_sort = NULL, expand = TRUE, shrink = FALSE, ...)

## S3 method for class 'epid'
merge_ids(id1, id2, tie_sort = NULL, expand = TRUE, shrink = FALSE, ...)

## S3 method for class 'pane'
merge_ids(id1, id2, tie_sort = NULL, expand = TRUE, shrink = FALSE, ...)
```

Arguments

...	Other arguments
id1	[integer|epid|pid|pane].
id2	[integer|epid|pid|pane].
tie_sort	[atomic]. Preferential order for breaking tied matches.
expand	[logical]. If TRUE, id1 gains new records if id2 indicates a match. *Not interchangeable with* shrink.
shrink	[logical]. If TRUE, id1 loses existing records id2 does not indicate a match. *Not interchangeable with* expand.

Details

Groups in id1 are expanded or shrunk by groups in id2.

A unique group with only one record is considered a non-matching record.

Note that the expand and shrink features are not interchangeable. The outcome when shrink is TRUE is not the same when expand is FALSE. See Examples.

See Also

links; links_af_probabilistic

Examples

```
id1 <- rep(1, 5)
id2 <- c(2, 2, 3, 3, 3)
merge_ids(id1, id2, shrink = TRUE)

id1 <- c(rep(1, 3), 6, 7)
id2 <- c(2,2,3,3,3)
merge_ids(id1, id2, shrink = TRUE)
merge_ids(id1, id2, expand = FALSE)

id1 <- rep(1, 5)
id2 <- c(1:3, 4, 4)
merge_ids(id1, id2, shrink = TRUE)
merge_ids(id1, id2, expand= FALSE)

data(missing_staff_id)
dfr <- missing_staff_id
id1 <- links(dfr[[5]])
id2 <- links(dfr[[6]])
merge_ids(id1, id2)
```

`number_line` `number_line`

Description

A range of `numeric` values.

Usage

```
number_line(l, r, id = NULL, gid = NULL)

as.number_line(x)

is.number_line(x)

left_point(x)

left_point(x) <- value

right_point(x)

right_point(x) <- value

start_point(x)

start_point(x) <- value
```

```
end_point(x)

end_point(x) <- value

number_line_width(x)

reverse_number_line(x, direction = "both")

shift_number_line(x, by = 1)

expand_number_line(x, by = 1, point = "both")

invert_number_line(x, point = "both")

number_line_sequence(
  x,
  by = NULL,
  length.out = 1,
  fill = TRUE,
  simplify = FALSE
)
```

Arguments

l	[numeric-based]. Left point of the number_line.
r	[numeric-based]. Right point of the number_line. Must be able to be coerced to a numeric object.
id	[integer]. Unique element identifier. Optional.
gid	[integer]. Unique group identifier. Optional.
x	[number_line]
value	[numeric based]
direction	[character]. Type of number_line reverse. Options are; "increasing", "decreasing" or "both" (default).
by	[integer]. Increment or decrement. Passed to seq() in number_line_sequence().
point	[character]. "start", "end", "left" or "right" point.
length.out	[integer]. Number of splits. For example, 1 for two parts or 2 for three parts. Passed to seq().
fill	[logical]. Retain (TRUE) or drop (FALSE) the remainder of an uneven split.
simplify	[logical]. If TRUE, returns a sequence of finite numbers.

Details

A number_line object represents a range of numbers. It is made up of a start and end point as the lower and upper ends of the range respectively. The location of the start point - left or right, determines whether it is an "increasing" or "decreasing" number_line. This is the direction of the number_line.

`reverse_number_line()` - reverse the direction of a `number_line`. A reversed `number_line` has its `left` and `right` points swapped. The `direction` argument specifies which type of `number_line` will be reversed. `number_line` with non-finite `start` or end points (i.e. `NA`, `NaN` and `Inf`) can't be reversed.

`shift_number_line()` - Shift a `number_line` towards the positive or negative end of the number line.

`expand_number_line()` - Increase or decrease the width of a `number_line`.

`invert_number_line()` - Change the `left` or `right` points from a negative to positive value or vice versa.

`number_line_sequence()` - Split a `number_line` into equal parts (`length.out`) or by a fixed recurring width (`by`).

Value

`number_line`

See Also

`overlaps`; `set_operations`; `episodes`; `links`

Examples

```
number_line(-100, 100)

# Also compatible with other numeric based object classes
number_line(as.POSIXct("2019-05-15 13:15:07", tz = "UTC"),
            as.POSIXct("2019-05-15 15:17:10", tz = "UTC"))

# Coerce compatible object classes to `number_line` objects
as.number_line(5.1); as.number_line(as.Date("2019-10-21"))

# A test for number_line objects
a <- number_line(as.Date("2019-04-25"), as.Date("2019-01-01"))
is.number_line(a)

# Structure of a number_line object
left_point(a); right_point(a); start_point(a); end_point(a)

# Reverse number_line objects
reverse_number_line(number_line(as.Date("2019-04-25"), as.Date("2019-01-01")))
reverse_number_line(number_line(200, -100), "increasing")
reverse_number_line(number_line(200, -100), "decreasing")

c <- number_line(5, 6)
# Shift number_line objects towards the positive end of the number line
shift_number_line(x = c(c, c), by = c(2, 3))
# Shift number_line objects towards the negative end of the number line
shift_number_line(x = c(c, c), by = c(-2, -3))

# Change the duration, width or length of a number_line object
```

```
d <- c(number_line(3, 6), number_line(6, 3))

expand_number_line(d, 2)
expand_number_line(d, -2)
expand_number_line(d, c(2,-1))
expand_number_line(d, 2, "start")
expand_number_line(d, 2, "end")

# Invert `number_line` objects
e <- c(number_line(3, 6), number_line(-3, -6), number_line(-3, 6))
e
invert_number_line(e)
invert_number_line(e, "start")
invert_number_line(e, "end")

# Split number line objects
x <- number_line(Sys.Date() - 5, Sys.Date())
x
number_line_sequence(x, by = 2)
number_line_sequence(x, by = 4)
number_line_sequence(x, by = 4, fill = FALSE)
number_line_sequence(x, length.out = 2)
```

`number_line-class` `number_line` *object*

Description

S4 objects representing a range of numeric values

Usage

```
## S4 method for signature 'number_line'
show(object)

## S4 method for signature 'number_line'
rep(x, ...)

## S4 method for signature 'number_line'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'number_line'
x[[i, j, ..., exact = TRUE]]

## S4 replacement method for signature 'number_line'
x[i, j, ...] <- value

## S4 replacement method for signature 'number_line'
x[[i, j, ...]] <- value
```

```
## S4 method for signature 'number_line'
x$name

## S4 replacement method for signature 'number_line'
x$name <- value

## S4 method for signature 'number_line'
c(x, ...)

## S3 method for class 'number_line'
unique(x, ...)

## S3 method for class 'number_line'
seq(x, precision = NULL, fill = FALSE, ...)

## S3 method for class 'number_line'
sort(x, decreasing = FALSE, ...)

## S3 method for class 'number_line'
format(x, ...)

## S3 method for class 'number_line'
as.list(x, ...)

## S3 method for class 'number_line'
as.data.frame(x, ...)
```

Arguments

object	object
x	x
...	...
i	i
j	j
drop	drop
exact	exact
value	value
name	slot name
precision	Round precision
fill	[logical]. Retain (TRUE) or drop (FALSE) the remainder of an uneven split.
decreasing	If TRUE, sort in descending order.

Slots

start First value in the range.

`id` Unique element id. Optional.

`gid` Unique group id. Optional.

`.Data` Length, duration or width of the range.

`overlaps` *Overlapping number line objects*

Description

Identify overlapping `number_line` objects

Usage

```
overlaps(x, y, methods = 8)

overlap(x, y)

none(x, y)

exact(x, y)

across(x, y)

x_across_y(x, y)

y_across_x(x, y)

chain(x, y)

x_chain_y(x, y)

y_chain_x(x, y)

aligns_start(x, y)

x_aligns_start_y(x, y)

y_aligns_start_x(x, y)

aligns_end(x, y)

x_aligns_end_y(x, y)

y_aligns_end_x(x, y)

inbetween(x, y)
```

```
x_inbetween_y(x, y)

y_inbetween_x(x, y)

overlap_method(x, y)

include_overlap_method(methods)

exclude_overlap_method(methods)

overlap_method_codes(methods)

overlap_method_names(methods)
```

Arguments

x	[number_line]
y	[number_line]
methods	[charater|integer]. Type of overlap. See as.data.frame(diyar::overlap_methods$options) for options.

Details

There are 6 mutually exclusive types of overlap;

- exact() - identical start_point and end_point points.
- inbetween() - Both start_point and end_point of one number_line object are within the start_point and end_point of another.
- across() - Only the start_point or end_point of one number_line object is in between the start_point and end_point of another.
- chain() - end_point of one number_line object is identical to the start_point of another.
- aligns_start() - identical start_point only.
- aligns_end() - identical end_point only.

Except exact(), each type of overlap has two variations;

- x_‘method‘_y() - number_line-x starts before number_line-y.
- y_‘method‘_x() - number_line-y starts before number_line-x.

There are two mutually inclusive types of overlap;

- overlap() - a convenient option to select "ANY" and "ALL" type of overlap.
- none() - a convenient option to select "NO" type of overlap.

Selecting multiple types of overlap;

- overlaps() - select specific type(s) of overlap.

- `overlap_method()` - return the type of overlap for a pair of `number_line` objects.
- `overlap_method_codes()` - return the corresponding overlap method code for a specific type(s) of overlap.
- `overlap_method_names()` - return the corresponding type(s) of overlap for a specific overlap code.
- `include_overlap_method()` - return a `character(1)` value for specified type(s) of overlap.
- `exclude_overlap_method()` - return a `character(1)` value for all type(s) of overlap except those specified.

Value

`logical`; `character`

See Also

`number_line`; `set_operations`

Examples

```
a <- number_line(-100, 100)
g <- number_line(100, 100)
overlaps(a, g)

# It's neither an "exact" or "chain"-overlap
overlaps(a, g, methods = "exact|chain")

# It's an "aligns_end"-overlap
overlap_method(a, g)
overlaps(a, g, methods = "exact|chain|x_aligns_end_y")

# Corresponding overlap code
overlap_method_codes("exact|chain|x_aligns_end_y")
include_overlap_method(c("exact", "chain", "x_aligns_end_y"))

# Corresponding overlap name
overlap_method_names(overlap_method_codes("exact|chain|x_aligns_end_y"))

# Every other type overlap
exclude_overlap_method(c("exact", "chain", "x_aligns_end_y"))
overlap_method_names(exclude_overlap_method(c("exact", "chain", "x_aligns_end_y")))

# All the above is based on tests for each specific type of overlap as seen below
none(a, g)
exact(a, g)
across(a, g)
x_across_y(a, g)
y_across_x(a, g)
chain(a, g)
x_chain_y(a, g)
y_chain_x(a, g)
inbetween(a, g)
```

```
x_inbetween_y(a, g)
y_inbetween_x(a, g)
aligns_start(a, g)
x_aligns_start_y(a, g)
y_aligns_start_x(a, g)
aligns_end(a, g)
x_aligns_end_y(a, g)
y_aligns_end_x(a, g)
```

---

pane-class pane *object*

Description

S4 objects storing the result of partitions.

Usage

```
is.pane(x)

as.pane(x)

## S3 method for class 'pane'
format(x, ...)

## S3 method for class 'pane'
unique(x, ...)

## S3 method for class 'pane'
summary(object, ...)

## S3 method for class 'pane_summary'
print(x, ...)

## S3 method for class 'pane'
as.data.frame(x, ..., decode = TRUE)

## S3 method for class 'pane'
as.list(x, ..., decode = TRUE)

## S4 method for signature 'pane'
show(object)

## S4 method for signature 'pane'
rep(x, ...)

## S4 method for signature 'pane'
```

```
x[i, j, ..., drop = TRUE]

## S4 method for signature 'pane'
x[[i, j, ..., exact = TRUE]]

## S4 method for signature 'pane'
c(x, ...)
```

Arguments

x	x
...	...
object	object
decode	If TRUE, data is decoded
i	i
j	j
drop	drop
exact	exact

Slots

`sn` Unique record identifier.

`.Data` Unique pane identifier.

`case_nm` Record type in regards to index assignment.

`window_list` A list of considered windows for each pane.

`dist_pane_index` The difference between each event and it's index event.

`pane_dataset` Data sources in each pane.

`pane_interval` The start and end dates of each pane. A `number_line` object.

`pane_length` The duration or length of (`pane_interval`).

`pane_total` The number of records in each pane.

`options` Some options passed to the instance of `partitions`.

`window_matched` A list of matched windows for each pane.

Examples

```
# A test for pane objects
pn <- partitions(date = 1, by = 1)
is.pane(pn); is.pane(2)
```

partitions *Distribute events into specified intervals.*

Description

Distribute events into groups defined by time or numerical intervals. Each set of linked records are assigned a unique identifier with relevant group-level data.

Usage

```
partitions(
  date,
  window = NULL,
  windows_total = 1,
  separate = FALSE,
  sn = NULL,
  strata = NULL,
  data_links = "ANY",
  custom_sort = NULL,
  group_stats = FALSE,
  data_source = NULL,
  by = NULL,
  length.out = NULL,
  fill = TRUE,
  display = "none",
  precision = 1
)
```

Arguments

date	[date|datetime|integer|number_line]. Event date or period.
window	[integer|number_line]. Numeric or time intervals.
windows_total	[integer|number_line]. Minimum number of matched windows required for a pane. See details
separate	[logical]. If TRUE, events matched to different windows are not linked.
sn	[integer]. Unique record identifier. Useful for creating familiar pane identifiers.
strata	[atomic]. Subsets of the dataset. Panes are created separately for each strata.
data_links	[list|character]. A set of data_sources required in each pane. A pane without records from these data_sources will be unlinked. See Details.
custom_sort	[atomic]. Preferred order for selecting "index" events.
group_stats	[logical]. If TRUE (default), the returned pane object will include group specific information like panes start and end dates.
data_source	[character]. Unique data source identifier. Adds the list of datasets in each pane to the pane. Useful when the data is from multiple sources.

by	[integer]. Width of splits.
length.out	[integer]. Number of splits.
fill	[logical]. Retain (TRUE) or drop (FALSE) the remainder of an uneven split.
display	[character]. Display a status update. Options are; "none" (default), "progress" or "stats".
precision	Round precision

Details

Each assigned group is referred to as a pane A pane consists of events within a specific time or numerical intervals (window).

Each window must cover a separate interval. Overlapping windows are merged before events are distributed into panes. Events that occur over two windows are assigned to the last one listed.

Alternatively, you can create windows by splitting a period into equal parts (length.out), or into a sequence of intervals with fixed widths (by).

By default, the earliest event is taken as the "Index" event of the pane. An alternative can be chosen with custom_sort. Note that this is simply a convenience option because it has no bearing on how groups are assigned.

partitions() will categorise records into 3 types;

- "Index" - Index event/record of the pane.
- "Duplicate_I" - Duplicate of the "Index" record.
- "Skipped" - Records that are not assigned to a pane.

Every element in data_links must be named "l" (links) or "g" (groups). Unnamed elements of data_links will be assumed to be "l".

- If named "l", only groups with records from every listed data_source will be retained.
- If named "g", only groups with records from any listed data_source will be retained.

NA *values in* strata *excludes records from the partitioning process.*

See vignette("episodes") for more information.

Value

pane

See Also

pane; number_line_sequence; episodes; links; overlaps; number_line; schema

Examples

```
events <- c(30, 2, 11, 10, 100)
windows <- number_line(c(1, 9, 25), c(3, 12, 35))

events
partitions(date = events, length.out = 3, separate = TRUE)
partitions(date = events, by = 10, separate = TRUE)
partitions(date = events, window = windows, separate = TRUE)
partitions(date = events, window = windows, separate = FALSE)
partitions(date = events, window = windows, separate = FALSE, windows_total = 4)
```

`pid-class` `pid` *objects*

Description

S4 objects storing the result of `links`.

Usage

```
is.pid(x)

as.pid(x, ...)

## S3 method for class 'pid'
format(x, ...)

## S3 method for class 'pid'
unique(x, ...)

## S3 method for class 'pid'
summary(object, ...)

## S3 method for class 'pid_summary'
print(x, ...)

## S3 method for class 'pid'
as.data.frame(x, ..., decode = TRUE)

## S3 method for class 'pid'
as.list(x, ..., decode = TRUE)

## S4 method for signature 'pid'
show(object)

## S4 method for signature 'pid'
```

```
rep(x, ...)

## S4 method for signature 'pid'
x[i, j, ..., drop = TRUE]

## S4 method for signature 'pid'
x[[i, j, ..., exact = TRUE]]

## S4 method for signature 'pid'
c(x, ...)
```

Arguments

x	x
...	...
object	object
decode	If TRUE, data is decoded
i	i
j	j
drop	drop
exact	exact

Slots

`sn` Unique record identifier.

`.Data` Unique group identifier.

`link_id` Unique reference ID for each match.

`pid_cri` Match stage of the step-wise linkage.

`pid_dataset` Data sources in each group.

`pid_total` The number of records in each group.

`iteration` The iteration when a record was matched to it's group (`.Data`).

Examples

```
# A test for pid objects
pd <- links(criteria = 1)
is.pid(pd); is.pid(2)
```

predefined_tests *Predefined logical tests in* diyar

Description

A collection of predefined logical tests used with sub_criteria objects

Usage

```
exact_match(x, y)

range_match(x, y, range = 10)

prob_link(
  x,
  y,
  cmp_func,
  attr_threshold,
  score_threshold,
  probabilistic,
  return_weights = FALSE
)

true(x, y)

false(x, y)
```

Arguments

x	Attribute(s) to be compared against.
y	Attribute(s) to be compared by.
range	Difference between y and x.
cmp_func	Logical tests such as string comparators. See links_wf_probabilistic.
attr_threshold	Matching set of weight thresholds for each result of cmp_func. See links_wf_probabilistic.
score_threshold	Score threshold determining matched or linked records. See links_wf_probabilistic.
probabilistic	If TRUE, matches determined through a score derived base on Fellegi-Sunter model for probabilistic linkage. See links_wf_probabilistic.
return_weights	If TRUE, returns the match-weights and score-thresholds for record pairs.

Details

exact_match() - test that x == y

range_match() - test that x $\leq$ y $\leq$ (x + range)

`prob_link()` - Test that a record-pair relate to the same entity based on Fellegi and Sunter (1969) model for deciding if two records belong to the same entity.

In summary, record-pairs are created and categorised as matches and non-matches (`attr_threshold`) with user-defined functions (`cmp_func`). If `probabilistic` is `TRUE`, two probabilities (`m` and `u`) are used to calculate weights for matches and non-matches. The `m`-probability is the probability that matched records are actually from the same entity i.e. a true match, while `u`-probability is the probability that matched records are not from the same entity i.e. a false match. Record-pairs whose total score are above a certain threshold (`score_threshold`) are assumed to belong to the same entity.

Agreement (match) and disagreement (non-match) scores are calculated as described by Asher et al. (2020).

For each record pair, an agreement for attribute i is calculated as;

$$\log_2(m_i/u_i)$$

For each record pair, a disagreement score for attribute i is calculated as;

$$\log_2((1-m_i)/(1-u_i))$$

where m_i and u_i are the `m` and `u`-probabilities for each value of attribute i.

Note that each probability is calculated as a combined probability for the record pair. For example, if the values of the record-pair have `u`-probabilities of `0.1` and `0.2` respectively, then the `u`-probability for the pair will be `0.02`.

Missing data (`NA`) are considered non-matches and assigned a `u`-probability of `0`.

Examples

```
`exact_match`
exact_match(x = 1, y = 1)
exact_match(x = 1, y = 2)

`range_match`
range_match(x = 10, y = 16, range = 6)
range_match(x = 16, y = 10, range = 6)
```

`reframe` *Modify* `sub_criteria` *objects*

Description

Modify the attributes of a `sub_criteria` object.

Usage

```
reframe(x, ...)

## S3 method for class 'sub_criteria'
reframe(x, func = identity, ...)
```

Arguments

x	[sub_criteria].
...	Arguments passed to methods.
func	[function]. Transformation function.

See Also

sub_criteria; eval_sub_criteria; attr_eval

Examples

```
s_cri <- sub_criteria(month.abb, month.name)
reframe(s_cri, func = function(x) x[12])
reframe(s_cri, func = function(x) x[12:1])
reframe(s_cri, func = function(x) attrs(x[1:6], x[7:12]))
```

---

schema *Schema diagram for group identifiers*

Description

Create schema diagrams for number_line, epid, pid and pane objects.

Usage

```
schema(x, ...)

## S3 method for class 'number_line'
schema(x, show_labels = c("date", "case_overlap_methods"), ...)

## S3 method for class 'epid'
schema(
  x,
  title = NULL,
  show_labels = c("length_arrow"),
  show_skipped = TRUE,
  show_non_finite = FALSE,
  theme = "dark",
  seed = NULL,
  custom_label = NULL,
```

```
  ...
)

## S3 method for class 'pane'
schema(
  x,
  title = NULL,
  show_labels = c("window_label"),
  theme = "dark",
  seed = NULL,
  custom_label = NULL,
  ...
)

## S3 method for class 'pid'
schema(
  x,
  title = NULL,
  show_labels = TRUE,
  theme = "dark",
  orientation = "by_pid",
  seed = NULL,
  custom_label = NULL,
  ...
)
```

Arguments

x	[number_line|epid|pid|pane]
...	Other arguments.
show_labels	[logical|character]. Show/hide certain parts of the schema. See Details.
title	[character]. Plot title.
show_skipped	[logical]. Show/hide "Skipped" records.
show_non_finite	[logical]. Show/hide records with non-finite date values.
theme	[character]. Options are "dark" or "light".
seed	[integer]. See set.seed. Used to get a consistent arrangement of items in the plot.
custom_label	[character]. Custom label for each record of the identifier.
orientation	[character]. Show each record of a pid object within its group id ("by_pid") or its pid_cri ("by_pid_cri")

Details

A visual aid to describe the data linkage (links), episode tracking (episodes) or partitioning process (partitions).

show_labels **options (multi-select)**

- schema.epid - **TRUE**, **FALSE**, "sn", "epid", "date", "case_nm", "wind_nm", "length", "length_arrow", "case_overlap_methods" or "recurrence_overlap_methods"
- schema.pane - **TRUE**, **FALSE**, "sn", "pane", "date", "case_nm" or "window_label"
- schema.pid - **TRUE**, **FALSE**, "sn" or "pid"

Value

ggplot objects

Examples

```
schema(number_line(c(1, 2), c(2, 1)))

schema(episodes(1:10, 2))

schema(partitions(1:10, by = 2, separate = TRUE))

schema(links(list(c(1, 1, NA, NA), c(NA, 1, 1, NA))))
```

---

set_operations *Set operations on number line objects*

Description

Perform set operations on a pair of [number_line]s.

Usage

```
union_number_lines(x, y)

intersect_number_lines(x, y)

subtract_number_lines(x, y)
```

Arguments

x	[number_line]
y	[number_line]

Details

union_number_lines() - Combined the range of x and that of y

intersect_number_line() - Subset of x that overlaps with y and vice versa

subtract_number_lines() - Subset of x that does not overlap with y and vice versa.

The direction of the returned [number_line] will be that of the widest one (x or y). If x and y have the same length, it'll be an "increasing" direction.

If x and y do not overlap, NA ("NA ?? NA") is returned.

Value

[number_line]; list

See Also

number_line; overlaps

Examples

```
nl_1 <- c(number_line(1, 5), number_line(1, 5), number_line(5, 9))
nl_2 <- c(number_line(1, 2), number_line(2, 3), number_line(0, 6))

# Union
nl_1; nl_2; union_number_lines(nl_1, nl_2)


nl_3 <- number_line(as.Date(c("01/01/2020", "03/01/2020","09/01/2020"), "%d/%m/%Y"),
                    as.Date(c("09/01/2020", "09/01/2020","25/12/2020"), "%d/%m/%Y"))

nl_4 <- number_line(as.Date(c("04/01/2020","01/01/2020","01/01/2020"), "%d/%m/%Y"),
                    as.Date(c("05/01/2020","05/01/2020","03/01/2020"), "%d/%m/%Y"))

# Intersect
nl_3; nl_4; intersect_number_lines(nl_3, nl_4)

# Subtract
nl_3; nl_4; subtract_number_lines(nl_3, nl_4)
```

---

staff_records *Datasets in* diyar *package*

Description

Datasets in diyar package

Usage

```
data(staff_records)

data(missing_staff_id)

data(infections)

data(infections_2)

data(infections_3)
```

```
data(infections_4)

data(hospital_admissions)

data(patient_list)

data(patient_list_2)

data(hourly_data)

data(Opes)

data(episode_unit)

data(overlap_methods)

data(patient_records)
```

Format

data.frame

data.frame

data.frame

data.frame

data.frame

data.frame

data.frame

data.frame

An object of class `data.frame` with 5 rows and 4 columns.

data.frame

data.frame

list

list

data.frame

Details

staff_records - Staff record with some missing data

missing_staff_id - Staff records with missing staff identifiers

infections, infections_2, infections_3 and infections_4 - Reports of bacterial infections

hospital_admissions - Hospital admissions and discharges

patient_list & patient_list_2 - Patient list with some missing data

Hourly data

Opes - List of individuals with the same name

Duration in seconds for each 'episode_unit'

Permutations of number_line overlap methods

Examples

```
data(staff_records)
data(missing_staff_id)
data(infections)
data(infections_2)
data(infections_3)
data(infections_4)
data(hospital_admissions)
data(patient_list)
data(patient_list_2)
data(hourly_data)
data(Opes)
data(episode_unit)
data(overlap_methods)
data(patient_records)
```

---

sub_criteria *Match criteria*

Description

Match criteria for record linkage with links and episodes

Usage

```
sub_criteria(
  ...,
  match_funcs = c(exact = diyar::exact_match),
  equal_funcs = c(exact = diyar::exact_match),
  operator = "or"
)

attrs(..., .obj = NULL)

eval_sub_criteria(x, ...)

## S3 method for class 'sub_criteria'
print(x, ...)

## S3 method for class 'sub_criteria'
format(x, show_levels = FALSE, ...)

## S3 method for class 'sub_criteria'
```

```
eval_sub_criteria(
  x,
  x_pos = seq_len(max(attr_eval(x))),
  y_pos = rep(1L, length(x_pos)),
  check_duplicates = TRUE,
  depth = 0,
  ...
)
```

Arguments

...	[atomic] Attributes passed to or eval_sub_criteria() or eval_sub_criteria() Arguments passed to methods for eval_sub_criteria()
match_funcs	[function]. User defined logical test for matches.
equal_funcs	[function]. User defined logical test for identical record sets (all attributes of the same record).
operator	[character]. Options are "and" or "or".
.obj	[data.frame|list]. Attributes.
x	[sub_criteria]. Attributes.
show_levels	[logical]. If TRUE, show recursive depth for each logic statement of the match criteria.
x_pos	[integer]. Index of one half of a record pair.
y_pos	[integer]. Index of one half of a record pair.
check_duplicates	[logical]. If FALSE, does not check duplicate values. The result of the initial check will be recycled.
depth	[integer]. First order of recursion.

Details

sub_criteria() - Create a match criteria as a sub_criteria object. A sub_criteria object contains attributes to be compared, logical tests for the comparisons (see predefined_tests for examples) and another set of logical tests to determine identical records.

attrs() - Create a d_attribute object - a collection of atomic objects that can be passed to sub_criteria() as a single attribute.

eval_sub_criteria() - Evaluates a sub_criteria object.

At each iteration of links or episodes, record-pairs are created from each attribute of a sub_criteria object. eval_sub_criteria() evaluates each record-pair using the match_funcs and equal_funcs functions of a sub_criteria object. See predefined_tests for examples of match_funcs and equal_funcs.

User-defined functions are also permitted as match_funcs and equal_funcs. Such functions must meet three requirements:

1. It must be able to compare the attributes.

2. It must have two arguments named `x` and `y`, where `y` is the value for one observation being compared against all other observations (`x`).
3. It must return a `logical` object i.e. `TRUE` or `FALSE`.

`attrs()` is useful when the match criteria requires an interaction between the multiple attributes. For example, attribute 1 + attribute 2 > attribute 3.

Every attribute, including those in `attrs()`, must have the same length or a length of 1.

Value

`sub_criteria`

See Also

`predefined_tests`; `links`; `episodes`; `eval_sub_criteria`

Examples

```
# Attributes
attr_1 <- c(30, 28, 40, 25, 25, 29, 27)
attr_2 <- c("M", "F", "U", "M", "F", "U", "M")

# A match criteria
## Example 1 - A maximum difference of 10 in attribute 1
s_cri1 <- sub_criteria(attr_1, match_funcs = range_match)
s_cri1

# Evaluate the match criteria
## Compare the first element of 'attr_1' against all other elements
eval_sub_criteria(s_cri1)
## Compare the second element of 'attr_1' against all other elements
x_pos_val <- seq_len(max(attr_eval(s_cri1)))
eval_sub_criteria(s_cri1,
                  x_pos = x_pos_val,
                  y_pos = rep(2, length(x_pos_val)))

## Example 2 - `s_cri1` AND an exact match on attribute 2
s_cri2 <- sub_criteria(
  s_cri1,
  sub_criteria(attr_2, match_funcs = exact_match),
  operator = "and")
s_cri2

## Example 3 - `s_cri1` OR an exact match on attribute 2
s_cri3 <- sub_criteria(
  s_cri1,
  sub_criteria(attr_2, match_funcs = exact_match),
  operator = "or")
s_cri3

# Evaluate the match criteria
eval_sub_criteria(s_cri2)
```

```
eval_sub_criteria(s_cri3)

# Alternatively, using `attr()`
AND_func <- function(x, y) range_match(x$a1, y$a1) & x$a2 == y$a2
OR_func <- function(x, y) range_match(x$a1, y$a1) | x$a2 == y$a2

## Create a match criteria
s_cri2b <- sub_criteria(attrs(.obj = list(a1 = attr_1, a2 = attr_2)),
                        match_funcs = AND_func)
s_cri3b <- sub_criteria(attrs(.obj = list(a1 = attr_1, a2 = attr_2)),
                        match_funcs = OR_func)

# Evaluate the match criteria
eval_sub_criteria(s_cri2b)
eval_sub_criteria(s_cri3b)
```

`windows` *Windows and lengths*

Description

Covert `windows` to and from `case_lengths` and `recurrence_lengths`.

Usage

```
epid_windows(date, lengths, episode_unit = "days")

epid_lengths(date, windows, episode_unit = "days")

index_window(date, from_last = FALSE)
```

Arguments

`date`	As used in `episodes`.
`lengths`	The duration (`lengths`) between a `date` and `window`.
`episode_unit`	Time unit of `lengths`. Options are "seconds", "minutes", "hours", "days", "weeks", "months" or "years". See `diyar::episode_unit`
`windows`	The range (`windows`) relative to a `date` for a given duration (`length`).
`from_last`	As used in `episodes`.

Details

`epid_windows` - returns the corresponding `window` for a given a `date`, and `case_length` or `recurrence_length`.

`epid_lengths` - returns the corresponding `case_length` or `recurrence_length` for a given `date` and `window`.

`index_window` - returns the corresponding `case_length` or `recurrence_length` for the `date` only.

`index_window(date = x)` is a convenience function for `epid_lengths(date = x, window = x)`.

Value

number_line.

Examples

```
# Which `window` will a given `length` cover?
date <- Sys.Date()
epid_windows(date, 10)
epid_windows(date, number_line(5, 10))
epid_windows(date, number_line(-5, 10))
epid_windows(date, -5)


# Which `length` is required to cover a given `window`?
date <- number_line(Sys.Date(), Sys.Date() + 20)
epid_lengths(date, Sys.Date() + 30)
epid_lengths(date, number_line(Sys.Date() + 25, Sys.Date() + 30))
epid_lengths(date, number_line(Sys.Date() - 10, Sys.Date() + 30))
epid_lengths(date, Sys.Date() - 10)

# Which `length` is required to cover the `date`?
index_window(20)
index_window(number_line(15, 20))
```

Index