Package 'diyar'

August 9, 2021

Type Package

Title Record Linkage and Epidemiological Case Definitions in R

Date 2021-08-07

Version 0.3.1

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 record linkage and implementing epidemiological case definitions in R.
Record linkage is implemented either through a multistage deterministic approach or a probabilistic approach.
Matching records are assigned to unique groups. There are mechanisms to address missing data and conflicting matches across linkage stages.
Track and assign events (e.g. sample collection) and periods (e.g. hospital admission) to unique groups based on a case definition.
The tracking process permits several options such as episode lengths and recurrence.
Duplicate events or records can then be identified for removal or further analyses.

License GPL-3

Encoding UTF-8

LazyData true

Imports methods, grDevices, graphics, utils, Rfast, ggplot2, rlang

RoxygenNote 7.1.1

Suggests knitr, rmarkdown, testthat, covr, stringdist

VignetteBuilder knitr

Language en-GB

NeedsCompilation no

Depends R (>= 3.5.0)

Repository CRAN

Date/Publication 2021-08-09 08:00:07 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)
```

---

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)
```

diyar_label *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, ...)

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

## 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
i	i
j	j
drop	drop
exact	exact

Slots

`sn` Unique record identifier.

`.Data` Unique `episode` identifier.

`wind_id` Unique window identifier.

`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 of the tracking process when a record was linked to its episode.

`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)
```

`episodes` *Link events to chronological episodes.*

Description

Link events with matching attributes and within specified durations of each other. Each set of linked records are assigned a unique identifier with relevant group-level information.

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 = FALSE,
  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
)

episodes_wf_splits(
  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 = FALSE,
  display = "none",
  case_sub_criteria = NULL,
  recurrence_sub_criteria = case_sub_criteria,
  case_length_total = 1,
  recurrence_length_total = case_length_total
)

fixed_episodes(
  date,
  case_length = Inf,
  episode_unit = "days",
  to_s4 = TRUE,
  case_overlap_methods = 8,
  deduplicate = FALSE,
  display = "none",
  bi_direction = FALSE,
  recurrence_length = case_length,
  recurrence_overlap_methods = case_overlap_methods,
  include_index_period = TRUE,
  ...,
  overlap_methods = 8,
  overlap_method = 8,
  x
)

rolling_episodes(
  date,
  case_length = Inf,
  recurrence_length = case_length,
  episode_unit = "days",
  to_s4 = TRUE,
  case_overlap_methods = 8,
  recurrence_overlap_methods = case_overlap_methods,
  deduplicate = FALSE,
  display = "none",
```

```
  bi_direction = FALSE,
  include_index_period = TRUE,
  ...,
  overlap_methods = 8,
  overlap_method = 8,
  x
)

episode_group(df, ..., episode_type = "fixed")
```

Arguments

date	[date|datetime|integer|number_line]. Event date or period.
case_length	[integer|number_line]. Duration from index event distinguishing one "case" from another. This is the case window.
episode_type	[character]. Options are "fixed" (default), "rolling" or "recursive". See Details.
recurrence_length	[integer|number_line]. Duration from the last "duplicate" event distinguishing a "recurrent" event from its index event. This is the recurrence window.
episode_unit	[character]. Time units 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 for each strata.
sn	[integer]. Unique record identifier. Useful for creating familiar epid identifiers.
episodes_max	[integer]. The maximum number of episodes permitted within each strata.
rolls_max	[integer]. Maximum number of times the index "case" can recur. Only used if episode_type is "rolling".
case_overlap_methods	[character|integer]. Methods of overlap considered when tracking duplicates of "case" events. See (overlaps)
recurrence_overlap_methods	[character|integer]. Methods of overlap considered when tracking duplicates of "recurrent" events. See (overlaps)
skip_if_b4_lengths	[logical]. If TRUE (default), when using lagged case_length or recurrence_length, events before the cut-off point or period are skipped.
data_source	[character]. Unique data source identifier. Adds the list of datasets in each episode to the epid. Useful when the dataset has data from multiple sources.
data_links	[list|character]. A set of data_sources 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 or reference events.

skip_order [integer]. "nth" level of custom_sort. Episodes with index events beyond this level of preference are skipped.

reference_event
[character]. Specifies which of the duplicates are used as reference events for subsequent windows. Options are "last_record" (default), "last_event", "first_record" or ""firs_event".

case_for_recurrence
[logical]. If TRUE, both "case" and "recurrent" events will have a case window. If FALSE (default), only case events will have a case window. Only used if episode_type is "rolling".

from_last [logical]. Chronological order of episode tracking i.e. ascending (TRUE) or descending (FALSE).

group_stats [logical]. If TRUE (default), episode-specific information like episode start and end dates are returned.

display [character]. The progress messages printed on screen. Options are; "none" (default), "progress" and "stats".

case_sub_criteria
[sub_criteria]. Matching conditions for "case" windows in addition to temporal links.

recurrence_sub_criteria
[sub_criteria]. Matching conditions for "recurrence" windows in addition to temporal links.

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

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

skip_unique_strata
[logical]. If TRUE (default), all strata with a single record are skipped skipped.

to_s4 [logical]. Deprecated. Output type - epid (TRUE) or data.frame (FALSE).

deduplicate [logical]. Deprecated. If TRUE, "duplicate" events are excluded from the epid.

bi_direction [logical]. Deprecated. If TRUE, "duplicate" events before and after the index event are tracked.

include_index_period
[logical]. Deprecated. If TRUE, events overlapping with the index event or period are linked even if they are outside the cut-off period.

... Arguments passed to episodes.

overlap_methods
[character]. Deprecated. Please use case_overlap_methods or recurrence_overlap_methods. Methods of overlap considered when tracking duplicate event. See (overlaps)

overlap_method [character]. Deprecated. Please use case_overlap_methods or recurrence_overlap_methods. Methods of overlap considered when tracking event. All event are checked by the same set of overlap_method.

x [date|datetime|integer|number_line]. Deprecated. Record date or period. Please use date.

df [data.frame]. Deprecated. One or more datasets appended together. See Details.

Details

All dated records within a specified duration of an index record are linked together as an episode. By default, this process occurs in ascending order, beginning with the earliest event. This can be changed to a descending (from_last) or custom order (custom_sort). Ties are always broken by the chronological order of events.

A "fixed" episode has a fixed maximum duration determined by case_length, while a "rolling" episode can continue to recur. A "rolling" episode will persist as long as is specified by rolls_max.

episodes() will categorise records into 5 type of events;

- "Case" - Index event of the episode.
- "Duplicate_C" - Duplicate of the index event.
- "Recurrent" - Recurrence of the index event.
- "Duplicate_R" - Duplicate of the recurrent event.
- "Skipped" - Records excluded from the episode tracking process.

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

NA *values in* strata *excludes records from the episode tracking process*

episode_group() has been retired. It now only exists to support previous code with minimal input from users. Moving forward, please use episodes().

rolling_episodes() and rolling_episodes() are convenience functions to support previous code with minimal input from users. Moving forward, please use episodes().

See vignette("episodes") for more information.

Value

epid

See Also

custom_sort; sub_criteria; epid_length; epid_window; partitions; links; overlaps; number_line; schema

Examples

```
data(infections)
data(hospital_admissions)

db_1 <- infections
db_1$patient_id <- c(rep("PID 1",8), rep("PID 2",3))

# Fixed episodes
# One 16-day (15-day difference) episode per patient
db_1$epids_p <- episodes(date = db_1$date,
                         strata = db_1$patient_id,
                         case_length = 15,
                         episodes_max = 1)
# Rolling episodes
# 16-day episodes with recurrence periods of 11 days
db_1$rd_b <- episodes(date = db_1$date,
                      case_length = 15,
                      recurrence_length = 10,
                      episode_type = "rolling")

# Interval grouping
hospital_admissions$admin_period <- number_line(hospital_admissions$admin_dt,
                                                hospital_admissions$discharge_dt)
admissions <- hospital_admissions[c("admin_period","epi_len")]

# Episodes of overlapping periods of admission
hospital_admissions$epids_i <- episodes(date = hospital_admissions$admin_period,
                                        case_length = 0,
                                        case_overlap_methods = "inbetween")
```

---

links *Multistage deterministic record linkage*

Description

Match records in successive stages with different matching conditions. Each set of linked records are assigned a unique identifier with relevant group-level information.

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 = FALSE,
  check_duplicates = FALSE,
  tie_sort = NULL
)

record_group(df, ..., to_s4 = TRUE)
```

Arguments

criteria	[list|atomic]. Attributes to compare. Each element of the list is a stage in the linkage process. See Details.
sub_criteria	[list|sub_criteria]. Additional matching criteria for each stage of the linkage process. See sub_criteria
sn	[integer]. Unique record identifier. Useful for creating familiar pid identifiers.
strata	[atomic]. Subsets of the dataset. Record-groups are created separately for each strata. See Details.
data_source	[character]. Data source identifier. Adds the list of data sources in each record-group to the pid. Useful when the dataset has data from multiple sources.
data_links	[list|character]. A set of data_sources required in each pid. A record-group without records from these data_sources will be unlinked. See Details.
display	[character]. Progress messages printed on screen. Options are; "none" (default), "progress" or "stats".
group_stats	[logical]. If TRUE (default), return group specific information like record counts for each pid.
expand	[logical]. If TRUE, allows a record-group to expand with each subsequent stages of the linkage process. *Not interchangeable with* shrink.
shrink	[logical]. If TRUE, forces a record-group to shrink with each subsequent 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. See vignette("links").
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. See vignette("links").
tie_sort	[atomic]. Preferential order for selecting breaking tied matches within a stage.
df	[data.frame]. Deprecated. One or more datasets appended together. See Details.
...	Arguments passed to links.
to_s4	[logical]. Deprecated. Output type - pid (TRUE) or data.frame (FALSE).

Details

Match priority decreases with each subsequent stage of linkage i.e. earlier stages (`criteria`) are considered superior. Therefore, it's important for each `criteria` to be listed in an order of decreasing relevance.

Records with missing `criteria` values (`NA`) are skipped at each stage of the linkage process, while records with missing `strata` values (`NA`) are skipped from the entire linkage process.

If a record is skipped, another attempt will be made to match the record at the next stage. If a record does not match any other record by the end of the linkage process (or it has a missing `strata`), it is assigned to a unique record-group.

A `sub_criteria` can be used to request additional matching conditions for each stage of the linkage process. When used, only records with matching `criteria` and `sub_criteria` are linked.

In `links`, each `sub_criteria` must be linked to a `criteria`. This is done by adding a `sub_criteria` to a named element of a `list`. Each element's name must correspond to a stage. See below for an example of 3 `sub_criteria` linked to `criteria` 1, 5 and 13.

For example;

$$list(\text{"}cr1\text{"} = sub_criteria, \text{"}cr5\text{"} = sub_criteria, \text{"}cr13\text{"} = sub_criteria).$$

`sub_criteria` can be nested to achieve nested conditions. A `sub_criteria` can be linked to different `criteria`. Any unlinked `sub_criteria` will be ignored.

By default, attributes in a `sub_criteria` are compared for an `exact_match`. However, user-defined logical tests (`function`) are also permitted. Such tests must meet 3 requirements:

1. It must be able to compare two `atomic` vectors.
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 either `TRUE` or `FALSE`.

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 remain linked.
- If named `"g"`, only groups with records from any listed `data_source` will remain linked.

`record_group()` has been retired and is no longer supported. It only exists to support previous code with minimal input from users. Moving forward, please use `links()`.

See `vignette("links")` for more information.

Value

`pid`

See Also

`episodes`; `partitions`; `predefined_tests`; `sub_criteria`; `schema`

Examples

```
# Exact match
attr_1 <- c(1, 1, 1, NA, NA, NA, NA, NA)
attr_2 <- c(NA, NA, 2, 2, 2, NA, NA, NA)
links(criteria = list(attr_1, attr_2))

# User-defined tests using `sub_criteria()`
# Matching `sex` and a 20-year age range
age <- c(30, 28, 40, 25, 25, 29, 27)
sex <- c("M", "M", "M", "F", "M", "M", "F")
f1 <- function(x, y) abs(y - x) %in% 0:20
links(criteria = sex,
      sub_criteria = list(cr1 = sub_criteria(age, match_funcs = f1)))

# Multistage matches
# Relevance of matches: `forename` > `surname`
data(staff_records); staff_records
links(criteria = list(staff_records$forename, staff_records$surname),
      data_source = staff_records$sex)

# Relevance of matches:
# `staff_id` > `age` (AND (`initials`, `hair_colour` OR `branch_office`))
data(missing_staff_id); missing_staff_id
links(criteria = list(missing_staff_id$staff_id, missing_staff_id$age),
      sub_criteria = list(cr2 = sub_criteria(missing_staff_id$initials,
                                             missing_staff_id$hair_colour,
                                             missing_staff_id$branch_office)),
      data_source = missing_staff_id$source_1)

# Group expansion
match_cri <- list(c(1,NA,NA,1,NA,NA),
                  c(1,1,1,2,2,2),
                  c(3,3,3,2,2,2))
links(criteria = match_cri, expand = TRUE)
links(criteria = match_cri, expand = FALSE)
links(criteria = match_cri, shrink = TRUE)
```

---

links_wf_probabilistic

Probabilistic record linkage

Description

A specific use case of `links` for probabilistic record linkage.

Usage

```
links_wf_probabilistic(
```

```
  attribute,
  blocking_attribute = NULL,
  cmp_func = diyar::exact_match,
  cmp_threshold = 0.95,
  probabilistic = TRUE,
  m_probability = 0.95,
  score_threshold = 1,
  id_1 = NULL,
  id_2 = NULL,
  ...
)

prob_score_range(attribute, m_probability = 0.95)
```

Arguments

attribute	[list]. Attributes to compare.
blocking_attribute	[atomic]. Subsets of the dataset.
cmp_func	[list|function]. String comparators for each attribute. See Details.
cmp_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 match from the string comparator is actually from the same entity.
score_threshold	[numeric|number_line]. Score-threshold for linked records. See Details.
id_1	[list|numeric]. One half of a specific pair of records to check for match weights and score-thresholds.
id_2	[list|numeric]. One half of a specific pair of records to check for match weights and score-thresholds.
...	Arguments passed to links

Details

links_wf_probabilistic is a wrapper function of links for probabilistic record linkage. Its implementation is 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 (cmp_func). Two probabilities (m and u) are then estimated for each record pair to score 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. m-probabilities must be supplied but u-probabilities are calculated for each value of an attribute. This is calculated as the frequency of each value in the dataset. 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.

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

By default, matches and non-matches for each attribute are determined as an exact_match with a binary outcome. String comparators can also be used with thresholds (cmp_threshold) for each similarity score. If probabilistic is FALSE, the sum of all similarity scores is used as the score_threshold instead of deriving one from the m and u-probabilities.

links_wf_probabilistic requires a score_threshold in advance of the linkage process. This differs from the typical approach where a score_threshold is selected after the linkage process, following a review of all calculated scores. To help with this, prob_score_range will return the range of scores attainable for a given set of attributes. Additionally, id_1 and id_2 can be used to link specific records pairs, aiding the review of potential scores.

A blocking_attribute can be used to reduce processing time by restricting comparisons to subsets of the dataset.

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 Also

links; episodes; partitions; predefined_tests; sub_criteria

Examples

```
# Using exact matches
dfr <- missing_staff_id[c("staff_id",  "initials",
                          "hair_colour", "branch_office")]
score_range <- prob_score_range(attribute = as.list(dfr))
prob_pids1 <- links_wf_probabilistic(attribute = as.list(dfr),
                                     score_threshold = score_range$minimum_score)
prob_pids1

# Using other logical tests e.g. string comparators
# For example, matching last word in `hair_colour` and `branch_office`
last_word_wf <- function(x) tolower(gsub("^.* ", "", x))
last_word_cmp <- function(x, y) last_word_wf(x) == last_word_wf(y)
prob_pids2 <- links_wf_probabilistic(attribute = as.list(dfr),
                                     cmp_func = c(diyar::exact_match,
                                                  diyar::exact_match,
                                                  last_word_cmp,
                                                  last_word_cmp),
                                     score_threshold = score_range$mid_scorce)
prob_pids2

# Results for specific record pairs
prob_pids3 <- links_wf_probabilistic(attribute = as.list(dfr),
                                     cmp_func = c(diyar::exact_match,
                                                  diyar::exact_match,
                                                  last_word_cmp,
                                                  last_word_cmp),
                                     score_threshold = score_range$mid_scorce,
                                     id_1 = c(1, 1, 1),
                                     id_2 = c(6, 7, 4))
prob_pids3
```

`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)
```

---

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. Must be able to be coerced to a numeric object.
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" objects to be reversed. 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]. Split into number_line or sequence of finite numbers

Details

A number_line represents a range of numbers on a number line. It is made up of a start and end point which are 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" range. This is the direction of the number_line.

reverse_number_line() - reverses 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

```
date <- function(x) as.Date(x, "%d/%m/%Y")
dttm <- function(x) as.POSIXct(x, "UTC", format = "%d/%m/%Y %H:%M:%S")

number_line(-100, 100)

# Also compatible with other numeric based object classes
number_line(dttm("15/05/2019 13:15:07"), dttm("15/05/2019 15:17:10"))

# Coerce applicable object classes to `number_line` objects
as.number_line(5.1); as.number_line(date("21/10/2019"))

# A test for number_line objects
a <- number_line(date("25/04/2019"), date("01/01/2019"))
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(date("25/04/2019"), date("01/01/2019")))
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,ANY,ANY,ANY'
x[i, j, ...] <- value

## S4 replacement method for signature 'number_line,ANY,ANY,ANY'
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, fill = TRUE, simplify = 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
fill	[logical]. Retain (TRUE) or drop (FALSE) the remainder of an uneven split
simplify	[logical]. Split into number_line or sequence of finite numbers
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)

exact(x, y)

reverse(x, y)

across(x, y)

chain(x, y)

aligns_start(x, y)

aligns_end(x, y)

inbetween(x, y)

overlap_method(x, y)

include_overlap_method(methods)

exclude_overlap_method(methods)

overlap_method_codes(methods)
```

Arguments

x	[number_line]
y	[number_line]
methods	[charater|integer]. Methods of overlap. Check different pairs of number_line objects by different methods. Options are "exact", "reverse", "inbetween", "across", "chain", "aligns_start" and "aligns_end". Combinations are also supported see diyar::overlap_methods$options.

Details

9 logical test;

`exact()` - Identical left and right points.

`reverse()` - Swapped left and right points.

`inbetween()` - start and end point of one `number_line` object is within the start and end point of another.

`across()` - start or end point of one `number_line` object is in between the start and end point of another.

`chain()` - endpoint of one `number_line` object is the same as the start point of another.

`aligns_start()` - identical start points only.

`aligns_end()` - identical end point only.

`overlap()` - any kind of overlap. A convenient `method` for "ANY" and "ALL" methods of overlap.

`overlaps()` - overlap by a specified combination of the methods.

Describe methods of overlap;

`overlap_method()` - Shows how a pair of `number_line` object has overlapped. Does not show `"overlap"` since `overlap()` is always `TRUE` when any other method is `TRUE`.

`include_overlap_method()` and `exclude_overlap_method()` - Conveniently create the required values for `methods`, and `case_overlap_methods` and `recurrence_overlap_methods` in `episodes`.

`overlap_method_codes()` - Numeric codes for the supported combination of overlap methods.

Value

`logical`; `character`

See Also

`number_line`; `set_operations`

Examples

```
a <- number_line(-100, 100)
b <- number_line(10, 11.2)
c <- number_line(100, 200)
d <- number_line(100, 120)
e <- number_line(50, 120)
g <- number_line(100, 100)
f <- number_line(120, 50)

overlaps(a, g)
overlaps(a, g, methods = "exact|chain")

overlap(a, b)
overlap(a, e)

exact(a, g)
exact(a, a)
```

```
reverse(e, e)
reverse(e, f)

across(a, b)
across(a, e)

chain(c, d)
chain(a, c)

aligns_start(c, d)
aligns_start(a, c)

aligns_end(d, e)
aligns_end(a, c)

inbetween(a, g)
inbetween(b, a)

overlap_method(a, c)
overlap_method(d, c)
overlap_method(a, g)
overlap_method(b, e)

include_overlap_method("across")
include_overlap_method(c("across", "chain"))

exclude_overlap_method("across")
exclude_overlap_method(c("across", "chain"))

overlap_method_codes("across")
overlap_method_codes("across|chain|exact")
```

---

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, ...)

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

## 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
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 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 = number_line(0, Inf),
  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"
)
```

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 dataset has data 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]. The progress messages printed on screen. Options are; "none" (default) or "stats".

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.

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 tracking 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, ...)
```

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

## 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
i	i
j	j
drop	drop
exact	exact

Slots

`sn` Unique record identifier.

`.Data` Unique group identifier.

`link_id` Unique record identifier for matching records.

`pid_cri` Matching criteria.

`pid_dataset` Data sources in each group.

`pid_total` The number of records in each group.

`iteration` The iteration of the linkage process when a record was linked to its group.

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)

range_match_legacy(x, y)

prob_link(
  x,
  y,
  cmp_threshold,
  m_probability,
  score_threshold,
  return_weights,
  probabilistic,
  cmp_func
)
```

Arguments

x	Value of an attribute(s) to be compare against.
y	Value of an attribute(s) to be compare by.
range	Difference between y and x.
cmp_threshold	Matching set of weight thresholds for each result of cmp_func. See links_wf_probabilistic.
m_probability	Matching set of m-probabilities. The probability that a match from cmp_func is a true match. See links_wf_probabilistic.
score_threshold	Score threshold determining matched or linked records. See links_wf_probabilistic.
return_weights	If TRUE, returns the match-weights and score-thresholds for record pairs. 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.
cmp_func	Logical tests such as string comparators. See links_wf_probabilistic.

Details

`exact_match()` - test that `x == y`

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

`range_match_legacy()` - test that `overlap(as.number_line(x@gid),y)` is `TRUE`.

`prob_link()` - Test that a record sets `x` and `y` are from the same entity based on calculated weights and probability scores.

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)

`range_match_legacy`
x_nl <- number_line(10, 16, gid = 10)
y_nl1 <- number_line(16, 10)
y_nl2 <- number_line(16, 10)

range_match_legacy(x = x_nl, y = y_nl1)
range_match_legacy(x = x_nl, y = y_nl2)
```

---

schema *Schema diagram for linked records in* diyar

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", "length_label", "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)
```

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

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)
```

---

sub_criteria *Sub-criteria*

Description

Matching criteria for each iteration of `links` and `episodes`.

Usage

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

eval_sub_criteria(x, ...)
```

```
## S3 method for class 'sub_criteria'
eval_sub_criteria(
  x,
  strata = seq_len(max(attr_eval(x))),
  index_record = c(TRUE, rep(FALSE, length(strata) - 1)),
  sn = seq_len(length(strata)),
  check_duplicates = TRUE,
  ...
)
```

Arguments

...	[atomic].. Attributes.
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".
x	[sub_criteria]
strata	[integer]. Subsets of the dataset
index_record	[logical]. Represents the y-value of the x-y record pair to be compared. See (predefined_tests).
sn	[integer] Unique index for each record.
check_duplicates	[logical]. If FALSE, does not check duplicate values. The result of the initial check will be recycled.

Details

sub_criteria() is the mechanism for providing matching criteria to an iteration of links or episodes. It creates a sub_criteria class object which contains the attributes to be compared, logical tests for the comparisons (see predefined_tests for examples) and another set of logical tests to determine identical records.

**Determining identical records reduces processing time.*

Value

sub_criteria

Logical

See Also

predefined_tests, links and episodes

Examples

```
# Sub-criteria
s_cri1 <- sub_criteria(c(30, 28, 40, 25, 25, 29, 27),
                       match_funcs = range_match)
s_cri2 <- sub_criteria(c(30, 28, 40, 25, 25, 29, 27),
                       match_funcs = exact_match)

# Nested sub-criteria
s_cri3 <- sub_criteria(s_cri1, s_cri2, operator = "or")
s_cri4 <- sub_criteria(s_cri1, s_cri3, operator = "and")

`eval_sub_criteria`
# 3 values
strata <- rep(1, 3)
index_record <- c(TRUE, FALSE, FALSE)
sn <- 1:3

# Test for a match in either attribute
sub_cri_1 <- sub_criteria(c(1, 1, 0), c(2, 1, 2))
eval_sub_criteria(sub_cri_1, strata, index_record, sn)

# Test for a match in both attributes
sub_cri_2 <- sub_criteria(c(1, 1, 0), c(2, 1, 2), operator = "and")
eval_sub_criteria(sub_cri_2, strata, index_record, sn)
```

`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