Package 'diyar'

September 17, 2020

Type Package

Title Multistage Record Linkage and Case Definition for Epidemiological Analysis

Date 2020-09-16

Version 0.2.0

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 Perform multistage deterministic linkages and apply case definitions to datasets.
Records are linked by different matching criteria in a specified order of certainty.
The linkage process handles missing data and conflicting matches using this same order.
Track events (e.g. sample collection) and period (e.g. hospital admission) to episodes.
This process permits several options such as episode lengths and recurrence.
Record linkage and episode tracking assign unique group identifiers to matched records.
Duplicate events or records can then be identified or sub-analyses performed within each group.

License GPL-3

Encoding UTF-8

LazyData true

Imports methods, grDevices, graphics, utils, Rfast

RoxygenNote 7.1.1

Suggests knitr, rmarkdown, testthat, covr

VignetteBuilder knitr

Language en-GB

NeedsCompilation no

Repository CRAN

Date/Publication 2020-09-17 09:30:12 UTC

R topics documented:

`combns` *Generate every permutation of n elements.*

Description

An extension of `combn` to generate permutations not ordinarily captured by `combn`. Each argument should be used as would be used in `combn`.

Usage

```
combns(x, m, FUN = NULL, simplify = TRUE, ...)
```

Arguments

`x`	Vector source for combination.
`m`	Number of elements required. Multiple counts can be supplied.
`FUN`	Function applied to each combination.
`simplify`	Logical indicating if the result should be simplified to an array or returned as a list.
`...`	further arguments passed to FUN. Optional.

Details

`combns` - Return every possible permutation of . An extension of `combn`.

Value

number_line.

Examples

```
f1 <- function(x) paste0(x, collapse = ",")
combn(x = 1:3, m = 3, FUN = f1, simplify = TRUE)
combns(x = 1:3, m = 3, FUN = f1, simplify = TRUE)
combns(x = 1:3, m = 1:3, FUN = f1, simplify = TRUE)
```

---

custom_sort *Nested sorting*

Description

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

Usage

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

Arguments

...	Sequence of atomic vectors. Passed to **link**order.
decreasing	Sort order. Passed to **link**order.

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)
```

epid-class epid *object*

Description

S4 objects to store the results of episodes

Usage

```
as.epid(x)

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

## S3 method for class 'epid'
unique(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

episodes *Track episodes for case definitions and record deduplication.*

Description

Link events into a chronological sequence of episodes.

Usage

```
episodes(
  date,
  case_length = Inf,
  episode_type = "fixed",
  recurrence_length = case_length,
  episode_unit = "days",
  episodes_max = Inf,
  rolls_max = Inf,
  overlap_methods_c = "overlap",
  overlap_methods_r = overlap_methods_c,
  sn = NULL,
  strata = NULL,
  skip_if_b4_lengths = FALSE,
  data_source = NULL,
  data_links = "ANY",
  custom_sort = NULL,
  skip_order = Inf,
  recurrence_from_last = TRUE,
  case_for_recurrence = FALSE,
  from_last = FALSE,
  group_stats = FALSE,
  display = "none"
)

fixed_episodes(
  date,
  case_length = Inf,
  episode_unit = "days",
  to_s4 = TRUE,
  overlap_methods_c = "overlap",
  deduplicate = FALSE,
  display = "progress",
  bi_direction = FALSE,
  recurrence_length = case_length,
  overlap_methods_r = overlap_methods_c,
  include_index_period = TRUE,
  ...,
  overlap_methods = "overlap",
```

```
  overlap_method = "overlap",
  x
)

rolling_episodes(
  date,
  case_length = Inf,
  recurrence_length = case_length,
  episode_unit = "days",
  to_s4 = TRUE,
  overlap_methods_c = "overlap",
  overlap_methods_r = overlap_methods_c,
  deduplicate = FALSE,
  display = "progress",
  bi_direction = FALSE,
  include_index_period = TRUE,
  ...,
  overlap_methods = "overlap",
  overlap_method = "overlap",
  x
)

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

Arguments

`date`	Event date (`date`, `datetime` or `numeric`) or period (`number_line`).
`case_length`	Cut-off point (`numeric`) or period (`number_line`), distinguishing one `"case"` from another. This is the case window.
`episode_type`	`"fixed"` or `"rolling"`.
`recurrence_length`	Cut-off point or period distinguishing a `"recurrent"` event from its index `"case"`. This is the recurrence window. By default, it's the same as `case_length`.
`episode_unit`	Time units for `case_length` and `recurrence_length`. Options are "seconds", "minutes", "hours", "days", "weeks", "months" or "years". See `diyar::episode_unit`.
`episodes_max`	The maximum number of episodes permitted within each `strata`.
`rolls_max`	Maximum number of times the index `"case"` can recur. Only used if `episode_type` is `"rolling"`.
`overlap_methods_c`	Methods of overlap considered when tracking duplicates of `"case"` events. See (`overlaps`)
`overlap_methods_r`	Methods of overlap considered when tracking duplicates of `"recurrent"` events. See (`overlaps`)
`sn`	Unique numerical record identifier. Useful for creating familiar episode identifiers.

`strata` Subsets. Episodes are tracked separately within each subset. `links` is useful for creating these.

`skip_if_b4_lengths` If `TRUE` (default), `events` before the cut-off points or periods are skipped.

`data_source` Unique data source identifier. Useful when the dataset has data from multiple sources.

`data_links` A set of `data_sources` required in each episode. A `strata` without records from these data sources will be skipped, and episodes without these will be unlinked. See `Details`.

`custom_sort` Preferential order for selecting index (`"case"`) events. Required for tracking episodes in a non-chronological sequence.

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

`recurrence_from_last` If `TRUE` (default), the reference event for a `recurrence window` will be the last event from the previous window. If `FALSE` (default), it will be the first event. Only used if `episode_type` is `"rolling"`.

`case_for_recurrence` 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` Chronological sequence of episode tracking. Ascending (`TRUE`) or descending `TRUE`.

`group_stats` If `TRUE` (default), episode-specific information like episode start and endpoints are returned. See `Value`.

`display` The messages printed on screen. Options are; `"none"` (default) or, `"progress"` and `"stats"` for a progress update or a more detailed breakdown of the tracking process.

`to_s4` Data type of returned object. `epid` (`TRUE`) or `data.frame` (`FALSE`).

`deduplicate` if `TRUE`, `"duplicate"` events are excluded from the output.

`bi_direction` If `TRUE`, `"duplicate"` events before and after the index event are tracked.

`include_index_period` If `TRUE`, overlaps with the index event or period are linked even if they are outside the cut-off period.

`...` Arguments passed to `episodes`

`overlap_methods` Deprecated. Please use `overlap_methods_c` or `overlap_methods_r`. Methods of overlap considered when tracking duplicate event. See (`overlaps`)

`overlap_method` Deprecated. Please use `overlap_methods_c` or `overlap_methods_r`. Methods of overlap considered when tracking event. All event are checked by the same set of `overlap_method`.

`x` Deprecated. Record date or period. Please use `date`

`df` `data.frame`. One or more datasets appended together. See `Details`.

Details

Episodes are tracked from index events in chronological sequence as determined by `from_last`. You can use `custom_sort` for a non-chronological sequence. However, ties will be broken by chronological orders.

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

`episodes()` will categorise records into 5 types of events;

- `"Case"` - Index case of the episode.
- `"Duplicate_C"` - Duplicate of the index case.
- `"Recurrent"` - Recurrent event of the index case.
- `"Duplicate_R"` - Duplicate of the recurrent event.
- `"Skipped"` - Those skipped from the episode tracking process.

`data_source` - including this populates the `epid_dataset` slot. See `Value`.

`data_links` should be a `list` of `atomic` vectors with every element named `"l"` (links) or `"g"` (groups).

- `"l"` - Episodes with records from every listed data source will be retained.
- `"g"` - Episodes with records from any listed data source will be retained.

`data_links` and `skip_order` are useful for skipping episodes that are not required to minimise processing time.

`episode_group()` as it existed before `v0.2.0` has been retired. Its now exists to support previous code with minimal disruption. Please use `episodes()` moving forward.

`rolling_episodes()` and `rolling_episodes()` are wrapper functions for tracking `"fixed"` and `"rolling"` episodes respectively. They exist for convenience, to support previous code and arguments with minimal disruption.

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

Value

`epid` objects or `data.frame` if `to_s4` is `FALSE`

- `sn` - unique record identifier as provided (or generated)
- `epid | .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_from_wind` - duration of each event from its window's reference event
- `dist_from_epid` - duration of each event from its episode's reference event
- `epid_dataset` - data sources in each episode

- `epid_interval` - episode start and end dates. A `number_line` object.
- `epid_length` - the difference between episode start and end dates (`difftime`). If possible, it's the same unit as `episode_unit` otherwise, a difference in days is returned
- `epid_total` - number of records in each episode
- `iteration` - iteration of the process when each event was tracked to its episode.

See Also

`epid_length`, `epid_window`, `links`, `overlaps` and `number_line`

Examples

```
library(diyar)
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,
                                       overlap_methods_c = "inbetween")
```

`links` *Multistage deterministic record linkage*

Description

Link records in ordered stages with flexible matching conditions.

Usage

```
links(
  criteria,
  sub_criteria = NULL,
  sn = NULL,
  strata = NULL,
  data_source = NULL,
  data_links = "ANY",
  display = "progress",
  group_stats = FALSE,
  expand = TRUE,
  shrink = FALSE
)

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

Arguments

criteria	list of attributes to compare at each stage. Comparisons are done as an exact match i.e. (==). See Details.
sub_criteria	list of additional attributes to compare at each stage. Comparisons are done as an exact match or with user-defined logical tests function. See sub_criteria
sn	Unique numerical record identifier. Useful for creating familiar episode identifiers.
strata	Subsets. Record groups are tracked separately within each subset.
data_source	Unique data source identifier. Useful when the dataset contains data from multiple sources.
data_links	A set of data_sources required in each record group. A strata without records from these data sources will be skipped, and record groups without these will be unlinked. See Details.
display	The messages printed on screen. Options are; "none" (default) or, "progress" and "stats" for a progress update or a more detailed breakdown of the linkage process.
group_stats	If TRUE (default), group-specific information like record counts. See Value.
expand	If TRUE, allows increases in the size of a record group at subsequent stages of the linkage process.
shrink	If TRUE, allows reductions in the size of a record group at subsequent stages of the linkage process.
df	data.frame. One or more datasets appended together. See Details.
...	Arguments passed to links
to_s4	Data type of returned object. pid (TRUE) or data.frame (FALSE).

Details

`links()` performs an ordered multistage deterministic linkage. The relevance or priority of each stage is determined by the order in which they have been listed.

`sub_criteria` specifies additional matching conditions for each stage (`criteria`) of the process. If `sub_criteria` is not `NULL`, only records with matching `criteria` and `sub_criteria` are linked. If a record has missing values for any `criteria`, that record is skipped at that stage, and another attempt is made at the next stage. If there are no matches for a record at every stage, that record is assigned a unique group ID.

By default, records are compared for an exact match. However, user-defined logical tests (function) are also permitted. The function must be able to compare two atomic vectors and return either TRUE or FALSE. The function must have two arguments - x for the attribute and y for what it'll be compared against.

A match at each stage is considered more relevant than a match at the next stage. Therefore, `criteria` should always be listed in order of decreasing relevance.

`data_source` - including this populates the `pid_dataset` slot. See `Value`.

`data_links` should be a `list` of `atomic` vectors with every element named `"l"` (links) or `"g"` (groups).

- `"l"` - Record groups with records from every listed data source will be retained.
- `"g"` - Record groups with records from any listed data source will be retained.

`data_links` is useful for skipping record groups that are not required.

`record_group()` as it existed before `v0.2.0` has been retired. Its now exists to support previous code and arguments with minimal disruption. Please use `links()` moving forward.

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

Value

`pid` objects or `data.frame` if `to_s4` is `FALSE`)

- `sn` - unique record identifier as provided (or generated)
- `pid | .Data` - unique group identifier
- `link_id` - unique record identifier of matching records
- `pid_cri` - matching criteria
- `pid_dataset` - data sources in each group
- `pid_total` - number of records in each group
- `iteration` - iteration of the process when each record was linked to its record group

See Also

`episodes`, `predefined_tests` and `sub_criteria`

Examples

```
library(diyar)
# Exact match
links(criteria = c("Obinna","James","Ojay","James","Obinna"))

# User-defined tests using `sub_criteria()`
# Matching `sex` and + 20-year age gaps
age <- c(30, 28, 40, 25, 25, 29, 27)
sex <- c("M", "M", "M", "F", "M", "M", "F")
f1 <- function(x, y) (y - x) %in% 0:20
links(criteria = sex,
      sub_criteria = list(s1 = sub_criteria(age, funcs = f1)))

# Multistage linkage
# 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(s2 = sub_criteria(missing_staff_id$initials,
                                            missing_staff_id$hair_colour,
                                            missing_staff_id$branch_office)),
      data_source = missing_staff_id$source_1)
```

`listr` *Written 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 objects*

Description

number_line - A range of numeric values on a number line.

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")

compress_number_line(
  x,
  methods = "overlap",
  collapse = FALSE,
  deduplicate = TRUE,
  method = "overlap"
)

number_line_sequence(x, by = 1, length.out = NULL)
```

Arguments

l	Left point of the number_line object. Must be able to be coerced to a numeric object
r	Right point of the number_line object. Must be able to be coerced to a numeric object
id	Unique numeric element identifier. Optional
gid	Unique numeric group identifier. Optional
x	number_line object
value	numeric based value
direction	Type of "number_line" objects to be reversed. Options are; "increasing", "decreasing" or "both" (default).
by	increment or decrement. Passed to seq() in number_line_sequence()
point	"start" or "end" point
methods	Methods of overlap. Check different pairs of number_line objects with the different methods
collapse	If TRUE, collapse the compressed results yet again.
deduplicate	if TRUE, retains only one number_line object per set of overlapping number_line.
method	Method of overlap. Check every pair of number_line objects with the same method. Deprecated. Please use methods instead.
length.out	desired length of the sequence. Passed to seq()

Details

A number_line object represents a range of real numbers on a number line.

Visually, it's presented as the left (l) and right (r) points of the range. This may differ from start and end points. The start point is the lowest value in the range, regardless of whether it's at the left or right point.

The location of the start point - left or right, indicates whether it's an "increasing" or "decreasing" range. This is the direction of the number_line.

`reverse_number_line()` - reverses the direction of a `number_line` object. A reversed `number_line` object has its `l` and `r` points swapped. The `direction` argument specifies which type of `number_line` objects will be reversed. `number_line` objects with non-finite `starts` or `ends` `points` i.e. (`NA`, `NaN` and `Inf`) can't be reversed.

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

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

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

`compress_number_line()` - `"compress"` or `"collapse"` overlapping `number_line` objects into a new `number_line` object that covers the `start` and `end` points of the originals. This results in duplicate `number_line` objects with the `start` and `end` points of the new expanded `number_line` object. See `overlaps` for further details on overlapping `number_line` objects. If a familiar (but unique) `id` is used when creating the `number_line` objects, `compress_number_line` can be an alternative for simple implementations of `links` or `episodes`.

`number_line_sequence()` - Convert a `number_line` object into a sequence of finite numbers. The direction of the sequence will correspond to that of the `number_line` object.

Value

`number_line` object

See Also

`overlaps`, `set_operations`, `episodes` and `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); number_line(10, 11.2)

# Other numeric based object classes are also compatible
number_line(dttm("15/05/2019 13:15:07"), dttm("15/05/2019 15:17:10"))

# However, a warning is given if 'l' and 'r' have different classes.
# Consider if this needs to be corrected.
number_line(2, date("05/01/2019"))

# Convert numeric based objects 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(0, -100)
b <- number_line(date("25/04/2019"), date("01/01/2019"))
is.number_line(a); is.number_line(b)

# 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")

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

# Change the 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")

# Collapse number line objects
x <- c(number_line(10,10), number_line(10,20), number_line(5,30),  number_line(30,40))
compress_number_line(x, deduplicate = FALSE)
compress_number_line(x)
compress_number_line(x, collapse=TRUE)
compress_number_line(x, collapse=TRUE, methods = "inbetween")

# Convert a number line object to its series of real numbers
number_line_sequence(number_line(1, 5))
number_line_sequence(number_line(5, 1), .5)
number_line_sequence(number_line(5:1, 1:5), 1:5)

nl <- number_line(dttm("01/04/2019 00:00:00"), dttm("04/04/2019 00:00:00"))

number_line_sequence(c(nl, nl), c(episode_unit[["days"]] * 1.5, episode_unit[["hours"]] * 12))
```

number_line-class number_line *object*

Description

S4 objects representing a series of finite numbers on a number line Used for range matching in record_group and interval grouping in fixed_episodes, rolling_episodes and episode_group

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'
sort(x, decreasing = FALSE, ...)

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

Arguments

object	object
x	x
...	...
i	i

j	j
drop	drop
exact	exact
value	value
name	slot name
decreasing	logical. Should the sort be increasing or decreasing

Slots

`start` Start of the number line

`id` Unique `numeric` ID. Providing this is optional.

`gid` Unique `numeric` Group ID. Providing this is optional.

`.Data` Length/with and direction of the `number_line` object.

`overlaps` *Overlapping number line objects*

Description

Identify overlapping `number_line` objects

Usage

```
overlaps(x, y, methods = "overlap", method = "overlap")

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)
```

Arguments

x	number_line object.
y	number_line object.
methods	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".
method	Deprecated. Please use methods instead. Method of overlap. Check every pair of number_line objects by the same method.

Details

8 logical test;

exact() - Identical left and right points.

reverse() - Swapped left and right points.

inbetween() - start and endpoints of one number_line object is within the start and endpoints of another.

across() - start or endpoints of one number_line object is within the start and endpoints 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 endpoints only.

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

overlaps() - overlap by any set of the 7 methods above.

Describe methods of overlap;

overlap_method() - Shows if and how a pair of number_line object have 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 overlap_methods in episodes.

Value

logical; character

See Also

number_line and 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"))
```

`pid-class` `pid` *objects*

Description

S4 objects to store the results of `record_group`

Usage

```
as.pid(x, ...)

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

## S3 method for class 'pid'
unique(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

`predefined_tests` *User defined tests*

Description

A collection of predefined logical tests for `sub_criteria`

Usage

```
exact_match(x, y)

range_match(x, y, range = 10)

range_match_legacy(x, y)
```

Arguments

x	Every value of an attribute to compare in links.
y	One value to which is compared against all other values of the same attribute (x)
range	Difference between y and x.

Examples

```
library(diyar)

# `exact_match` - test that `x` is equal to `y`
exact_match(x = 1, y = "1")
exact_match(x = 1, y = 1)

# `range_match` - test that `y` is between `x` and `x + range`
range_match(x = 10, y = 16, range = 6)
range_match(x = 16, y = 10, range = 6)

# `range_match_legacy` - test that `x` (10) is within `y` (16 - 10)
x_nl <- number_line(10, 16, gid = 10)
y_nl <- number_line(16, 10)

range_match_legacy(x = x_nl, y = y_nl)
```

---

set_operations *Set operations on number line objects*

Description

Perform set operations on a pair of number_line objects.

Usage

```
union_number_lines(x, y)

intersect_number_lines(x, y)

subtract_number_lines(x, y)
```

Arguments

x	number_line object
y	number_line object

Details

union_number_lines() - Return a number_line object with the combined range of x and y

intersect_number_line() - Returns a subset of x that overlaps with y and vice versa

subtract_number_lines() - Returns a subset of x that does not overlap with y and vice versa. Returns a list with two elements;

- n1 - subset before the overlapped range between x and y.
- n2 - subset after the overlapped range between x and y

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 and 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)
```

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

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'

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)
```

---

sub_criteria *Sub-criteria for* links

Description

Additional matching conditions for each stage in links.

Usage

```
sub_criteria(..., funcs = diyar::exact_match)
```

Arguments

...	Additional attributes to compare.
funcs	User defined logical test.

Details

sub_criteria() is the mechanism for providing a sub_criteria to an instance of links.

Each attribute (atomic vectors) is compared as an exact match or with a user-defined logical test. The test must be supplied as a function with at least two arguments named `x` and `y`. Where `y` is the value for one observation being compared against all other observations (`x`).

Each attribute must have the same length.

funcs must be a function or list of functions to use with each attribute.

Each funcs must evaluate to TRUE or FALSE.

Each sub_criteria must be linked to a criteria in links. You can link multiple sub_criteria to one criteria. Any unlinked sub_criteria will be ignored.

Value

function or list of functions

Examples

```
library(diyar)

sub_criteria(c(30, 28, 40, 25, 25, 29, 27), funcs = range_match)

# Link each `sub_critera` a `criteria`.
list(
c1 = sub_criteria(c(30, 28, 40, 25, 25, 29, 27), funcs = range_match),
c2 = sub_criteria(c(30, 28, 40, 25, 25, 29, 27), funcs = range_match))
```

---

to_s4 *Change the returned outputs of* diyar *functions*

Description

Convert the returned output of number_line, record_group, episode_group, fixed_episodes and rolling_episodes from a data.frame to number_line, pid or epid objects, and vice versa.

Usage

```
to_s4(df)

to_df(s4)
```

Arguments

df	data.frame
s4	pid or epid objects

Value

to_s4 - pid or epid objects

to_df - data.frame object

Examples

```
data(infections)
dates <- infections$date
output <- fixed_episodes(dates, case_length=30)
output

# from the a pid/epid object to a data.frame
df_output <- to_df(output)
df_output

# from a data.frame to pid/epid object
s4_output <- to_s4(df_output)
s4_output

all(s4_output == output)
```

---

windows *Window and lengths*

Description

Interpret windows, case_lengths and recurrence_lengths as used in episodes.

Usage

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

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

index_window(date, from_last = F)
```

Arguments

date	As used in episodes.
lengths	case_length or recurrence_length arguments as used in episodes.
episode_unit	Time unit of lengths. Options are "seconds", "minutes", "hours", "days", "weeks", "months" or "years". See diyar::episode_unit.
windows	A range or period relative to date for a given lengths.
from_last	As used in episodes.

Details

epid_windows - returns the corresponding period 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 period.

Value

number_line.

Examples

```
# `epid_windows`
epid_windows(Sys.Date(), 10)
epid_windows(Sys.Date(), number_line(5, 10))
epid_windows(Sys.Date(), number_line(-5, 10))
epid_windows(Sys.Date(), -5)

# `epid_lengths`
epid_lengths(number_line(01, 20), 30)
epid_lengths(number_line(01, 20), number_line(25, 30))
epid_lengths(number_line(01, 20), number_line(-10, 30))
epid_lengths(number_line(01, 20), -10)

index_window(20)
index_window(as.number_line(20))
index_window(number_line(15, 20))
```

Index