Package 'diyar'

June 13, 2020

Type Package

Title Multistage Record Linkage and Case Definition for Epidemiological Analysis

Date 2020-06-11

Version 0.1.0

URL https://cran.r-project.org/package=diyar

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

Author Olisaeloka Nsonwu

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

Description Perform multistage deterministic linkages, apply case definitions to datasets, and deduplicate records.
Records (rows) from datasets are linked by different matching criteria and sub-criteria (columns) in a specified order of certainty.
The linkage process handles missing data and conflicting matches based on this same order of certainty.
For episode grouping, rows of dated events (e.g. sample collection) or interval of events (e.g. hospital admission) are
grouped into chronological episodes beginning with a ``Case". The process permits several options such as
episode lengths and recurrence periods which are used to build custom preferences for case assignment (definition).
The record linkage and episode grouping processes assign unique group IDs to matching records or those grouped into episodes.
This then allows for record deduplication or sub-analysis within these groups.

License GPL-3

Encoding UTF-8

LazyData true

Imports methods, grDevices, graphics, utils, dplyr (>= 0.7.5)

RoxygenNote 6.1.1

Suggests stringdist, knitr, rmarkdown, testthat, covr

VignetteBuilder knitr

Language en-GB

NeedsCompilation no

Repository CRAN

Date/Publication 2020-06-13 15:00:02 UTC

R topics documented:

`epid-class` `epid` *object*

Description

S4 objects to store the results of `fixed_episodes`, `rolling_episodes` and `episode_group`

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

`episode_group` *Episode grouping for case definitions and record deduplication*

Description

Group events into chronological episodes

Usage

```
episode_group(df, sn = NULL, strata = NULL, date, case_length,
  episode_type = "fixed", episode_unit = "days", episodes_max = Inf,
  recurrence_length = NULL, rolls_max = Inf,
  skip_if_b4_lengths = TRUE, data_source = NULL, data_links = "ANY",
  custom_sort = NULL, skip_order = NULL, from_last = FALSE,
  overlap_method = c("exact", "across", "inbetween", "aligns_start",
  "aligns_end", "chain"), overlap_methods = NULL, bi_direction = FALSE,
  group_stats = FALSE, display = TRUE, deduplicate = FALSE,
  to_s4 = TRUE, recurrence_from_last = TRUE,
  case_for_recurrence = FALSE, include_index_period = TRUE)

fixed_episodes(date, sn = NULL, strata = NULL, case_length,
  episode_unit = "days", episodes_max = Inf,
  skip_if_b4_lengths = TRUE, data_source = NULL, data_links = "ANY",
  custom_sort = NULL, skip_order = NULL, from_last = FALSE,
  overlap_method = c("exact", "across", "inbetween", "aligns_start",
  "aligns_end", "chain", "overlap", "none"), overlap_methods = "overlap",
  bi_direction = FALSE, group_stats = FALSE, display = TRUE,
  deduplicate = FALSE, x, to_s4 = TRUE, include_index_period = TRUE)
```

```
rolling_episodes(date, sn = NULL, strata = NULL, case_length,
  recurrence_length = NULL, episode_unit = "days",
  episodes_max = Inf, rolls_max = Inf, skip_if_b4_lengths = TRUE,
  data_source = NULL, data_links = "ANY", custom_sort = NULL,
  skip_order = NULL, from_last = FALSE, overlap_method = c("exact",
  "across", "inbetween", "aligns_start", "aligns_end", "chain", "overlap",
  "none"), overlap_methods = "overlap", bi_direction = FALSE,
  group_stats = FALSE, display = TRUE, deduplicate = FALSE, x,
  to_s4 = TRUE, recurrence_from_last = TRUE,
  case_for_recurrence = FALSE, include_index_period = TRUE)
```

Arguments

df	data.frame. One or more datasets appended together.
sn	Unique numerical record identifier. Optional.
strata	Subsets of the dataset. Episode grouping will be done separately within each subset of the dataset. In episode_group, you can use multiple columns. record_group can create useful strata e.g. patient identifiers.
date	Event date (date, datetime or numeric) or period (number_line).
case_length	Duration after a "case" within which subsequent events are considered "duplicate" events. This period is referred to as the the case window. Can be a (number_line) range.
episode_type	"fixed" or "rolling".
episode_unit	Time units. Options are "seconds", "minutes", "hours", "days", "weeks", "months" or "years". See diyar::episode_unit.
episodes_max	Maximum number of episodes to have within each strata.
recurrence_length	Duration after the last or first event (see recurrence_from_last) of the previous window within which subsequent events are considered "recurrent" events. This period is referred to as the recurrence window. If recurrence_length is not supplied, it's assumed to be the same as case_length. Can be a (number_line) range.
rolls_max	Maximum number of times an episode can reoccur. Only used if episode_type is "rolling".
skip_if_b4_lengths	If TRUE (default), records events before the case_length or the recurrence_length range are skipped.
data_source	Unique dataset identifier. Useful when the dataset contains data from multiple sources. In episode_group, you can use multiple columns supplied as column names.
data_links	Breakup episodes that will not include records from these data_sources. data_links should be a list with every element named 'l' (links) or 'g' (groups). Useful in skipping episodes that are not required to minimise processing time. Ignored if data_source is NULL.

custom_sort
: Preferential order for "case" assignment. Useful in specifying that episode grouping begins at particular events regardless of chronological order. In episode_group, you can use multiple columns as sort levels.

skip_order
: Skip episodes whose case events have custom_sort values that are less than or equal to the "nth" level of custom_sort. Useful in skipping episodes that are not required and so minimises the overall processing time. Ignored if custom_sort is NULL.

from_last
: If TRUE, episode grouping will be backwards in time - starting at the most recent event and proceeding to the earliest. If FALSE, it'll be forward in time - starting at the earliest event and proceeding to the most recent one.

overlap_method
: Methods of overlap considered when grouping event periods. Each pair of periods are checked with the same set of overlap_method. Deprecated please use overlap_methods instead.

overlap_methods
: Methods of overlap considered when grouping event periods. Different pairs of periods can be checked with different sets of overlap_methods

bi_direction
: If FALSE (default), "duplicate" events will be those within the case_length before **or** after the "case" as determined by from_last. If TRUE, "duplicate" events will be those within the same period before **and** after the "case".

group_stats
: If TRUE, the output will include additional information with useful stats for each episode group.

display
: If TRUE (default), a progress message is printed on screen.

deduplicate
: if TRUE, "dupilcate" events are excluded from the output.

to_s4
: If TRUE (default), episodes are returned as an epid object.

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 recurrence events will have a case window. If FALSE (default), only case events will have a case window. Only used if episode_type is "rolling".

include_index_period
: If TRUE, overlaps with the index event or period are grouped together even if they are outside the cut-off range (case_length or recurrence_length).

x
: Record date or interval. Deprecated. Please use date

Details

Episode grouping begins at a reference event ("case") and proceeds forward or backward in time depending on from_last. If custom_sort is used, episode grouping can be forced to begin at certain events before proceeding forward or backwards in time. The maximum duration of a "fixed" episode is the case_length. This period is referred to as the case window. The maximum duration of a "rolling" episode is the case_length plus all periods of recurrence. The recurrence periods are referred to as recurrence windows. This is a specified duration (recurrence_length) after the

last or first (depending on recurrence_from_last) event in the previous window. Events within this period are considered "recurrent" events.

When a data_source identifier is provided, epid_dataset is included in the output. This lists the source of every event in each episode.

fixed_episodes() and rolling_episodes() are wrapper functions of episode_group(). They are convenient alternatives with the same functionalities.

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_total - number of records in each episode
- epid_dataset - data sources in each episode
- epid_interval - episode start and end dates. A number_line object.
- epid_length - 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

See Also

record_group, 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$epids_p <- fixed_episodes(date=db_1$date, strata = db_1$patient_id,
case_length = 15, episodes_max = 1, display = FALSE)

# Rolling episodes
# Case length of 16 days and recurrence periods of 11 days
db_1$rd_b <- rolling_episodes(db_1$date, case_length = 15,
recurrence_length = 10, display = FALSE)
```

```
# 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 overlaping periods of admission
hospital_admissions$epi_0 <- fixed_episodes(date=hospital_admissions$admin_period,
case_length = 0, group_stats = TRUE, to_s4=TRUE)

# Note - episode_group() takes column names not actual values
```

---

number_line *Number line objects*

Description

number_line - A range of numeric based 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, method = c("exact", "across", "chain",
  "aligns_start", "aligns_end", "inbetween", "overlap", "none"),
  collapse = FALSE, deduplicate = TRUE, methods = "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 finite numeric value
r	Right point of the number_line object. Must be able to be coerced to a finite numeric value
id	Unique numeric element ID. Optional
gid	Unique numeric group ID. 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
method	Method of overlap. Check every pair of number_line objects with the same method. Deprecated. Please use methods instead.
collapse	If TRUE, collapse the compressed results yet again.
deduplicate	if TRUE, retains only one number_line object per set of overlapping number_line.
methods	Methods of overlap. Check different pairs of number_line objects with the different methods
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 number 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 object.

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 determines which type of number_line

objects will be reversed. number_line objects with non-finite numeric starts or end 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 to the opposite end of the number line.

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 record_group or episode_group.

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

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")

# Change the width or length of a number_line object
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, method = c("exact", "across", "chain", "aligns_start",
  "aligns_end", "inbetween", "overlap", "none"), methods = "overlap")

overlap(x, y)

exact(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
method	Method of overlap. Check every pair of number_line objects with the same method. Deprecated. Please use methods instead.
methods	Methods of overlap. Check different pairs of number_line objects using different methods

Details

7 logical test;

exact() - Identical start and end points

inbetween() - start and end points of one number_line object is in between the start and end points of another.

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

chain() - Chained i.e. end point of one number_line object is the same as the start point of another.

aligns_start() - Identical start points only.

aligns_end() - Identical end points only.

overlap() - Any kind of overlap. All other methods are mutually exclusive. overlap() is just a convenient method for "ANY" and "ALL" methods of overlap.

overlaps() - Overlap by any or all 7 methods above.

Describe methods of overlap;

overlap_method() - Shows if and 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 overlap_methods in episode_group.

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)
```

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

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

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

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

`record_group` *Multistage deterministic record linkage*

Description

Group matching or partially matching records in multiple stages of relevance using different criteria.

Usage

```
record_group(df, sn = NULL, criteria, sub_criteria = NULL,
  strata = NULL, data_source = NULL, group_stats = FALSE,
  display = TRUE, to_s4 = TRUE)
```

Arguments

df	data.frame. One or more datasets appended together.
sn	Unique numerical record identifier. Optional.
criteria	Column names of attributes to match. Each criteria is a stage in the process and the order in which they are listed determines the relevance of matches.
sub_criteria	Matching sub-criteria. Additional matching conditions for each stage (criteria).
strata	Subsets of the dataset. Record grouping will be done separately with each subset of the dataset. You can use multiple columns supplied as column names.
data_source	Unique dataset identifier. Useful when df contains data from multiple sources.
group_stats	If TRUE, output will include additional columns with useful stats for each record group.
display	If TRUE (default), a progress message is printed on screen.
to_s4	If TRUE (default), record groups are returned as a pid object.

Details

Record grouping occurs in stages of matching criteria.

Records are matched in two ways: an exact match i.e. the equivalent of (==), or range matching. An example of range matching is matching a date give or take 5 days, or matching an age give or take 2 years. To do this, create the range as a number_line object and supply it to the criteria or sub_criteria argument. The actual value within each range must be assigned to the gid slot of the number_line object.

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

sub_criteria can be used to force additional matching conditions at each stage. If sub_criteria is not NULL, only records with matching criteria and sub_criteria values are grouped together. 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.

When a data_source identifier is provided, pid_dataset is included in the output. This lists the source of every record in each record group.

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

See Also

episode_group and number_line

Examples

```
library(diyar)
three_people <- data.frame(forename=c("Obinna","James","Ojay","James","Obinna"),
                           stringsAsFactors = FALSE)

three_people$pids_a <- record_group(three_people, criteria= forename, to_s4 = TRUE)
three_people

# To handle missing or unknown data, recode missing or unknown values to NA or "".
three_people$forename[c(1,4)] <- NA
three_people$pids_b <- record_group(three_people, criteria= forename, to_s4 =TRUE)
three_people

data(staff_records); staff_records

# Range matching
dob <- staff_records["sex"]
dob$age <- c(30,28,40,25,25,29,27)

# age range: age + 20 years
dob$range_a <- number_line(dob$age, dob$age+20, gid=dob$age)
dob$pids_a <- record_group(dob, criteria = sex, sub_criteria = list(s1a="range_a"), to_s4 = TRUE)
dob[c("sex","age","range_a","pids_a")]

# age range: age +- 20 years
dob$range_b <- number_line(dob$age-20, dob$age+20, gid=dob$age)
dob$pids_b <- record_group(dob, criteria = sex, sub_criteria = list(s1a="range_b"), to_s4 = TRUE)
dob[c("sex","age","range_b","pids_b")]

dob$pids_c <- record_group(dob, criteria = range_b, to_s4 = TRUE)
dob[c("age","range_b","pids_c")]


# Multistage record grouping
staff_records$pids_a <- record_group(staff_records, sn = r_id, criteria = c(forename, surname),
                                     data_source = sex, display = FALSE, to_s4 = TRUE)
staff_records

# Add `sex` to the second stage (`cri`) to be more certain
staff_records$cri_2 <- paste0(staff_records$surname,"-", staff_records$sex)
staff_records$pids_b <- record_group(staff_records, r_id, c(forename, cri_2),
                                     data_source = dataset, display = FALSE, to_s4 = TRUE)
staff_records

# Using sub-criteria
data(missing_staff_id); missing_staff_id

missing_staff_id$pids <- record_group(missing_staff_id, r_id, c(staff_id, age),
```

```
list(s2a=c("initials","hair_colour","branch_office")), data_source = source_1, to_s4 = TRUE)

missing_staff_id
```

`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()` - Combined range of `x` and `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. Returns a list with two elements;

- `n1` - subset before the overlapped range
- `n2` - subset before the overlapped range

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_1 <- 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_2 <- 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_1; nl_2; intersect_number_lines(nl_1, nl_2)

# Subtract
nl_1; nl_2; subtract_number_lines(nl_1, nl_2)
```

---

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

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)
```

---

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)
```

Index