Package ‘diyar’

December 8, 2019

Type Package
Title Multistage Record Linkage and Case Definition for Epidemiological Analysis
Date 2019-12-08
Version 0.0.3
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, utils, rlang, dplyr (>= 0.7.5), lubridate (>= 1.7.4), tidyr (>= 0.8.2)
RoxygenNote 6.1.1
Suggests phonics, ggplot2, janitor, knitr, rmarkdown, testthat, covr
VignetteBuilder: knitr
Language: en-GB
NeedsCompilation: no
Repository: CRAN
Date/Publication: 2019-12-08 22:20:02 UTC

**R topics documented:**

- epid-class ............................................. 2
- episode_group .................................. 3
- number_line ....................................... 7
- number_line-class ................................ 10
- overlap ............................................ 12
- pid-class .......................................... 13
- record_group ....................................... 14
- staff_records ....................................... 16
- to_s4 ............................................. 17

Index .............................................................. 19

<table>
<thead>
<tr>
<th>epid-class</th>
<th>epid object</th>
</tr>
</thead>
</table>

**Description**

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

**Usage**

```r
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]
```
episode_group

  ## 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 record deduplication and case assignment

Description

  Group records 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, data_source = NULL,  
custom_sort = NULL, from_last = FALSE, overlap_method = c("across", 
"inbetween", "aligns_start", "aligns_end", "chain"),  
bi_direction = FALSE, group_stats = FALSE, display = TRUE,  
deduplicate = FALSE, to_s4 = FALSE)

  fixed_episodes(date, sn = NULL, strata = NULL, case_length,  
episode_unit = "days", episodes_max = Inf, data_source = NULL,  
custom_sort = NULL, from_last = FALSE, overlap_method = c("across", 
"inbetween", "aligns_start", "aligns_end", "chain"),  
bi_direction = FALSE, group_stats = FALSE, display = TRUE,  
deduplicate = FALSE, x, to_s4 = FALSE)

  rolling_episodes(date, sn = NULL, strata = NULL, case_length,  
recurrence_length = NULL, episode_unit = "days",  
episodes_max = Inf, rolls_max = Inf, data_source = NULL,  
custom_sort = NULL, from_last = FALSE, overlap_method = c("across", 
"inbetween", "aligns_start", "aligns_end", "chain"),  
bi_direction = FALSE, group_stats = FALSE, display = TRUE,  
deduplicate = FALSE, x, to_s4 = FALSE)
Arguments

df  data.frame. One or more datasets appended together.

sn  Unique numerical record identifier. Optional.

strata Subsets of the dataset within which episode grouping will be done separately. episode_group supports the use of multiple columns supplied as column names. record_group can be used to create the strata.

date  Date (date, datetime or numeric) or period (number_line) of events.

case_length  Period after a "Case (C)" within which another record from the same strata is considered a "Duplicate (D)" record.

episode_type  "fixed" or "rolling".

episode_unit  Time units as supported by lubridate’s duration function.

episodes_max  Maximum number of times to group episodes within each strata.

recurrence_length  Period after the last record ("Case (C)", "Duplicate (D)" or "Recurrent (R)") of an episode within which another record from the same strata is considered a "Recurrent (R)" record. If recurrence_length is not supplied, case_length is used as the recurrence_length.

rolls_max  Maximum number of times an event can reoccur within an episode. Only used if episode_type is "rolling".

data_source  Unique dataset identifier. Useful when the dataset contains data from multiple sources. episode_group support the use of multiple columns supplied as column names.

custom_sort  If TRUE, "Case (C)" assignment will be done with preference to this sort order. Useful in specifying that episode grouping begins at particular records regardless of chronological order. episode_group supports the use of multiple columns supplied as column names.

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

overlap_method  A set of ways for grouped intervals to overlap. Options are; "across", "aligns_start", "aligns_end", "inbetween", "chain". See overlap functions.

bi_direction  If FALSE, "Duplicate (D)" records will be those within the case_length period, before or after the "Case (C)" as determined by from_last. If TRUE, "Duplicate (D)" records will be those within the same period before and after the "Case (C)".

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

display  If TRUE, status messages are printed on screen.

deduplicate  if TRUE, "Duplicate (D)" records are excluded from the output.

to_s4  if TRUE, changes the returned output to an epid object.

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

Details

Episode grouping begins at a reference record ("Case (C)") 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 records before proceeding forward or backwards in time. The maximum duration of a "fixed" episode is the case_length while, the maximum duration of a "rolling" episode is the case_length plus all recurrence periods. A recurrence period is a fixed period (recurrence_length) after the last record of an episode. Records within this period are taken as a "Recurrent (R)" record of the initial "Case"

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

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

Value
data.frame (epid objects if to_s4 is TRUE)

- sn - unique record identifier as provided
- epid | .Data - unique episode identifier
- case_nm - record type in regards to case assignment
- 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

epid objects will be the default output in the next release.

See Also

record_group, overlap and number_line

Examples

library(dplyr)
library(lubridate)

#1. Fixed episodes
data(infections); infections
db_1 <- infections
# 16-day (difference of 15 days) episodes beginning from the earliest record
db_1$d <- fixed_episodes(db_1$date, case_length = 15, to_s4 = TRUE, display = FALSE)
# 16-hour (difference of 15 hours) episodes beginning from the earliest record
db_1$dh <- fixed_episodes(db_1$date, case_length = 15, episode_unit = "hours", to_s4 = TRUE, display = FALSE)
db_1

#2. Rolling episodes
# Case length and recurrence periods of 16 days
db_1$rd_a <- rolling_episodes(db_1$date, case_length = 15, to_s4 = TRUE, display = FALSE)
# 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, to_s4 = TRUE, display = FALSE)
# Case length of 16 days and 2 recurrence periods of 11 days
db_1$rd_c <- rolling_episodes(db_1$date, case_length = 15, recurrence_length = 10, rolls_max = 2, to_s4 = TRUE, display = FALSE)

db_1

# 3. Stratified episode grouping
db_3 <- infections
db_3$patient_id <- c(rep("PID 1",8), rep("PID 2",3))
# One 16-day episode per patient
db_3$epids_p <- fixed_episodes(date=db_3$date, strata = db_3$patient_id, case_length = 15, episodes_max = 1, to_s4 = TRUE, display = FALSE)

db_3

# 4. Case assignment
db_4 <- infections

## 4.1 Chronological order
db_4$forward_time <- fixed_episodes(db_4$date, case_length = 1, episode_unit = "month", to_s4 = TRUE, display = FALSE)

## 4.2 User defined order
db_4b <- infections
db_4b$ord1 <- ifelse(db_4b$infection =="RTI",0,1)

# RTI > UTI, or RTI > BSI
db_4b$ord1 <- ifelse(db_4b$infection =="RTI",0,1)
# UTI > BSI > RTI

# RTI > UTI, or RTI > BSI

# UTI > BSI > RTI

db_4b$ord2 <- factor(db_4b$infection, levels = c("UTI","BSI","RTI"))

db_4b$epids_1 <- fixed_episodes(db_4b$date, case_length = 15, custom_sort = db_4b$ord1, to_s4 = TRUE, display = FALSE)

db_4b$epids_2 <- fixed_episodes(db_4b$date, case_length = 15, custom_sort = db_4b$ord2, to_s4 = TRUE, display = FALSE)

db_4b$epids_2b <- fixed_episodes(db_4b$date, case_length = 15, custom_sort = db_4b$ord2, bi_direction = TRUE, to_s4 = TRUE, display = FALSE)

db_4b

# 5. Interval grouping
data(hospital_admissions)

hospital_admissions$admin_period <- number_line(hospital_admissions$admin_dt, hospital_admissions$discharge_dt)
admissions <- hospital_admissions[c("admin_period","epi_len")]
admissions
# Episodes of overlapping periods of admission
admissions$epi_0 <- fixed_episodes(date=admissions$admin_period, case_length = 0, group_stats = TRUE, to_s4=TRUE)
admissions

# Overlapping periods of admission separated by 1 month
admissions$epi_1 <- fixed_episodes(date=admissions$admin_period, case_length = 1, episode_unit = "months", group_stats = TRUE, to_s4 = TRUE, display = FALSE)
admissions

# Episodes of chained admission periods, and those with aligned end periods
admissions$epi_0b <- fixed_episodes(date=admissions$admin_period, case_length = 0, overlap_method = c("chain","aligns_end"), group_stats = TRUE, to_s4 = TRUE, display = FALSE)
admissions["epi_0b"]

# Note - episode_group() takes column names not actual values
db_5 <- infections
db_5$recur <- 20
db_5$epids_f <- episode_group(db_5, date=date, episode_type = "fixed", case_length = epi_len, to_s4 = TRUE, display = FALSE)
db_5$epids_r <- episode_group(db_5, date=date, episode_type = "rolling", case_length = epi_len, recurrence_length = recur, to_s4 = TRUE, display = FALSE)
db_5

---

**number_line**

**Number line objects**

**Description**

A set of functions to create and manipulate `number_line` objects.

**Usage**

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

`as.number_line(x)`

`is.number_line(x)`

`left_point(x)`

`right_point(x)`

`start_point(x)`

`end_point(x)`
number_line_width(x)

reverse_number_line(x, direction = "both")

shift_number_line(x, by = 1)

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

compress_number_line(x, method = c("across", "chain", "aligns_start", "aligns_end", "inbetween"), collapse = FALSE, deduplicate = TRUE)

number_line_sequence(x, by = 1)

Arguments

l  Left point of the number_line object. Should be, or can be coerced to a numeric object
r  Right point of the number_line object. Should be, or can be coerced to a numeric object
id Unique numeric ID. Providing this is optional
gid Unique numeric Group ID. Providing this is optional
x  number_line object
direction Type of "number_line" objects whose direction are to be reversed. Options are; "increasing", "decreasing" or "both".
by increment or decrement
point "start" or "start" point
method Method of overlap
collapse If TRUE, collapse the compressed results based on method of overlaps
deduplicate if TRUE, retains only one number_line object among duplicates

Details

A number_line object represents a series of real numbers on a number line.
Visually, it's presented as the left (l) and right (r) points of the series. This may differ from start and end points. The start point is the lowest number in the series, regardless of whether it's at the left or right point.
The location of the start point - left or right, indicate if it's an "increasing" or "decreasing" series. This is referred to as 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 but maintains the same width or length. 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() - a convenience function to shift a number_line object towards the positive or negative end of the number line.
expand_number_line() - a convenience function to increase or decrease the width or length of a number_line object.

compress_number_line() - Collapses overlapping number_line objects into a new number_line objects that covers the start and end points of the originals. This results in duplicate number_line objects with start and end points of the new expanded number_line object. See overlap 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 a simple alternative to record_group or episode_group.

number_line_sequence() - a convenience function to convert a number_line object into a sequence of finite numbers. The sequence will also include the start and end points. The direction of the sequence will correspond to that of the number_line object.

Value

number_line object

Examples

library(lubridate)

number_line(-100, 100); number_line(10, 11.2)

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

# A warning is given if 'l' and 'r' have different classes. Consider if these need to be corrected
number_line(2, dmy("05/01/2019"))

# Convert numeric based objects to number_line objects
as.number_line(5.1); as.number_line(dmy("21/10/2019"))

# Test for number_line objects
a <- number_line(0, -100)
b <- number_line(dmy("25/04/2019"), dmy("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(dmy("25/04/2019"), dmy("01/01/2019")))
reverse_number_line(number_line(200, -100), "increasing")
reverse_number_line(number_line(200, -100), "decreasing")

# Shift number_line objects
number_line(5,6)
# Towards the positive end of the number line
shift_number_line(number_line(5,6), 2)
# Towards the negative end of the number line
shift_number_line(number_line(6,1), -2)
# Increase or reduce the width or length of a \code{number_line} object
\begin{verbatim}
c(number_line(3,6), number_line(6,3))
expand_number_line(c(number_line(3,6), number_line(6,3)), 2)
expand_number_line(c(number_line(3,6), number_line(6,3)), -1)
expand_number_line(c(number_line(3,6), number_line(6,3)), 2, "start")
expand_number_line(c(number_line(3,6), number_line(6,3)), -2, "end")
\end{verbatim}

# Collapse number line objects
\begin{verbatim}
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, method = "inbetween")
\end{verbatim}

# Convert a number line object to its series of real numbers
\begin{verbatim}
number_line_sequence(number_line(1, 5))
number_line_sequence(number_line(5, 1), .5)
number_line_sequence(number_line(dmy("01/04/2019"), dmy("10/04/2019")), 1)
\end{verbatim}

# The length of the resulting vector will depend on the object class
\begin{verbatim}
nl <- number_line(dmy_hms("01/04/2019 00:00:00"), dmy_hms("04/04/2019 00:00:00"))
head(number_line_sequence(nl, 1.5), 15)
d <- duration(1.5,"days")
number_line_sequence(nl, d)
\end{verbatim}

### number_line-class

**number_line** object

**Description**

S4 objects representing a series of finite numbers on a number line. Used for range matching in \code{record_group} and interval grouping in \code{fixed_episodes}, \code{rolling_episodes} and \code{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,ANY,ANY,ANY'
\[x[i, j, \ldots] \leftarrow value\]

## S4 replacement method for signature 'number_line,ANY,ANY,ANY'
\[x[[i, j, \ldots]] \leftarrow value\]

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

## S4 replacement method for signature 'number_line'
x$name \leftarrow 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**
- **x**
- **...**
- **i**
- **j**
- **drop**
- **exact**
- **value**
- **name**
- **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/width and direction of the number_line object.
Description

A set of functions to identify overlapping number_line objects.

Usage

\[
\text{overlap}(x, y, \text{method} = \text{c("across", "chain", "aligns\_start", "aligns\_end", "inbetween")})
\]

\[
\text{across}(x, y)
\]

\[
\text{chain}(x, y)
\]

\[
\text{aligns\_start}(x, y)
\]

\[
\text{aligns\_end}(x, y)
\]

\[
\text{inbetween}(x, y)
\]

\[
\text{overlap\_method}(x, y)
\]

Arguments

- **x** number_line object
- **y** number_line object
- **method** Method of overlap

Value

- logical object
- character object

Examples

\[
a <- \text{number\_line}(-100, 100)
b <- \text{number\_line}(10, 11.2)
c <- \text{number\_line}(100, 200)
d <- \text{number\_line}(100, 120)
e <- \text{number\_line}(50, 120)
g <- \text{number\_line}(100, 100)
\text{across}(a, b)
\text{across}(a, e)
\text{chain}(c, d)
\text{chain}(a, c)
\]
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)

---

**Description**

S4 objects to store the results of `record_group`

**Usage**

```r
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`
- `...`: `...`
record_group

object object
i i
j j
drop drop
exact exact

record_group

Multistage deterministic record linkage

Description

Group matching records from one or more datasets.

Usage

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

Arguments

df data.frame. One or more datasets appended together.

sn Unique numerical record identifier. Optional.

criteria Column names of attributes to match. Records with matching values in these columns are grouped together.

sub_criteria Matching sub-criteria. Additional matching conditions for each stage (criteria).

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, status messages are printed on screen.

to_s4 if TRUE, changes the returned output to a pid object.

Details

Record grouping occurs in stages of matching criteria.

Records are matched in two ways; an exact match - the equivalent of (==), or matching a range of numeric values. 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 a number_line object based on the range of values, and assign the actual value assigned to gid. Then use the number_line as a sub_criteria.

A match at each stage is considered more relevant than those at subsequent stages. 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, it’s skipped at that stage, and another attempt is made at the next stage. If all criteria values are missing, that record is assigned a unique group ID.

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

Value

data.frame (pid objects if to_s4 is TRUE)

- sn - unique record identifier as provided
- pid | .Data - unique group identifier
- pid_cri - matched criteria for each record in the group
- pid_dataset - data sources in each group
- pid_total - number of records in each group

pid objects will be the default output from the next release.

See Also

episode_group and number_line

Examples

library(dplyr)
library(tidyr)

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["sex","age","range_a","pids_a"]

# age range: age +- 20 years
staff_records

```r
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 <- unite(staff_records, cri_2, c(surname, sex), sep ="-"
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
```

---

**staff_records**  
*Dummy datasets for diyar package*

**Description**  
Dummy datasets for 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)

Format

tibble

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

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)

---
to_s4

Change the returned outputs of diyar functions from data.frames to S4 objects, and vice versa

Description

Convert the returned output of record_group, episode_group, fixed_episodes and rolling_episodes from the current default (data.frame) to pid or epid objects, and vice versa.
Usage

to_s4(df)
to_df(s4)

Arguments

df data.frame. Returned output of record_group, episode_group, fixed_episodes and rolling_episodes
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 a data.frame to pid/epid object
output_2 <- to_s4(output)
output_2

# from the a pid/epid object to a data.frame
output_3 <- to_df(output_2)
output_3
Index

*Topic datasets
  staff_records, 16
  [, epid-method (epid-class), 2
  [, number_line-method
    (number_line-class), 10
  [, pid-method (pid-class), 13
  [<-, number_line, ANY, ANY, ANY-method
    (number_line-class), 10
  [<-, number_line-method
    (number_line-class), 10
  [[, epid-method (epid-class), 2
  [[, number_line-method
    (number_line-class), 10
  [[, pid-method (pid-class), 13
  [[<-, number_line, ANY, ANY, ANY-method
    (number_line-class), 10
  [[<-, number_line-method
    (number_line-class), 10
  $, number_line-method
    (number_line-class), 10
  $<-, number_line-method
    (number_line-class), 10

across (overlap), 12
aligns_end (overlap), 12
aligns_start (overlap), 12
as.epid (epid-class), 2
as.number_line (number_line), 7
as.pid (pid-class), 13

c.epid-method (epid-class), 2
c,number_line-method
  (number_line-class), 10
c.pid-method (pid-class), 13
chain (overlap), 12
compress_number_line (number_line), 7

duration, 4
end_point (number_line), 7

epid, 4, 5, 17, 18
epid-class, 2
episode_group, 2, 3, 4, 9, 10, 15, 17, 18
expand_number_line (number_line), 7

fixed_episodes, 2, 10, 17, 18
fixed_episodes (episode_group), 3
format.epid (epid-class), 2
format.number_line (number_line-class), 10
format.pid (pid-class), 13

hospital_admissions (staff_records), 16
hourly_data (staff_records), 16

inbetween (overlap), 12
infections (staff_records), 16
infections_2 (staff_records), 16
infections_3 (staff_records), 16
infections_4 (staff_records), 16
is.number_line (number_line), 7

left_point (number_line), 7

missing_staff_id (staff_records), 16

number_line, 4, 5, 7, 14, 15
number_line-class, 10
number_line_sequence (number_line), 7
number_line_width (number_line), 7

Opes (staff_records), 16
overlap, 4, 5, 9, 12
overlap_method (overlap), 12

patient_list (staff_records), 16
patient_list_2 (staff_records), 16
pid, 14, 15, 17, 18
pid-class, 13

record_group, 4, 5, 9, 10, 13, 14, 17, 18
rep.epid-method (epid-class), 2
rep.number_line-method
  (number_line-class), 10
rep.pid-method (pid-class), 13
reverse_number_line (number_line), 7
right_point (number_line), 7
rolling_episodes, 2, 10, 17, 18
rolling_episodes (episode_group), 3
shift_number_line (number_line), 7
show.epid-method (epid-class), 2
show.number_line-method
  (number_line-class), 10
show.pid-method (pid-class), 13
sort.number_line (number_line-class), 10
staff_records, 16
start_point (number_line), 7
to_df (to_s4), 17
to_s4, 17
unique.epid (epid-class), 2
unique.number_line (number_line-class), 10
unique.pid (pid-class), 13