Package ‘edeaR’

April 27, 2023

Type Package

Title Exploratory and Descriptive Event-Based Data Analysis

Version 0.9.4

Description Exploratory and descriptive analysis of event based data. Provides methods for describing and selecting process data, and for preparing event log data for process mining. Builds on the S3-class for event logs implemented in the package ‘bupaR’.

License MIT + file LICENSE

Depends R(>= 3.5.0)

Imports bupaR (>= 0.5.1), dplyr, data.table, ggplot2, ggthemes, glue, tibble, shiny, miniUI, tidyR, shinyTime, lubridate, purrr, stringr, rlang (>= 1.0.0), cli (>= 3.2.0), zoo, hms, lifecycle

Encoding UTF-8

RoxygenNote 7.2.3


Suggests knitr, eventdataR, rmarkdown, covr, testthat (>= 3.0.0)

VignetteBuilder knitr

BugReports https://github.com/bupaverse/edeaR/issues/

Config/testthat/edition 3

NeedsCompilation no

Author Gert Janssenswillen [aut, cre], Gerard van Hulzen [ctb], Marijke Swennen [ctb], Ivan Esin [ctb], Hasselt University [cph]

Maintainer Gert Janssenswillen <gert.janssenswillen@uhasselt.be>

Repository CRAN

Date/Publication 2023-04-27 08:33:06 UTC
R topics documented:

- activity_frequency ........................................... 3
- activity_presence ........................................... 5
- add_fixed_holiday .......................................... 8
- add_floating_holiday ....................................... 8
- add_holiday_periods ........................................ 9
- augment ....................................................... 9
- calculate_queuing_length .................................. 10
- calculate_queuing_times .................................... 11
- change_day ................................................... 13
- create_work_schedule ...................................... 13
- end_activities ............................................... 14
- filter_activity ............................................... 16
- filter_activity_frequency .................................. 17
- filter_activity_instance .................................... 19
- filter_activity_presence ................................... 21
- filter_case ................................................... 23
- filter_case_condition ....................................... 24
- filter_endpoints ............................................ 26
- filter_endpoints_condition ................................ 27
- filter_flow_time ............................................. 29
- filter_idle_time .............................................. 31
- filter_infrequent_flows .................................... 33
- filter_lifecycle .............................................. 34
- filter_lifecycle_presence .................................. 35
- filter_precedence ........................................... 37
- filter_precedence_condition ................................ 40
- filter_precedence_resource ................................ 42
- filter_processing_time ...................................... 44
- filter_resource ............................................... 46
- filter_resource_frequency .................................. 47
- filter_throughput_time ..................................... 49
- filter_time_period .......................................... 51
- filter_trace ................................................... 53
- filter_trace_frequency ...................................... 54
- filter_trace_length ......................................... 56
- filter_trim ..................................................... 58
- filter_trim_lifecycle ....................................... 60
- idle_time ..................................................... 62
- number_of_repetitions ..................................... 64
- number_of_selfloops ...................................... 67
- number_of_traces ............................................ 70
- plot .......................................................... 71
- print.work_schedule ........................................ 73
- processing_time ............................................. 73
- redo_repetitions_referral_matrix ......................... 76
- redo_selfloops_referral_matrix ......................... 77
### activity_frequency

#### Description

Provides summary statistics about the frequency of activity types at the level of log, traces, cases, activity types.

#### Usage

```r
activity_frequency(
  log,
  level = c("log", "trace", "activity", "case"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)
```

```
# S3 method for class 'eventlog'
activity_frequency(
  log,
  level = c("log", "trace", "activity", "case"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)
```

```
# S3 method for class 'grouped_eventlog'
activity_frequency(
  log,
  level = c("log", "trace", "activity", "case"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
)
```

eventlog = deprecated()

## S3 method for class 'activitylog'
activity_frequency(
    log,
    level = c("log", "trace", "activity", "case"),
    append = deprecated(),
    append_column = NULL,
    sort = TRUE,
    eventlog = deprecated()
)

## S3 method for class 'grouped_activitylog'
activity_frequency(
    log,
    level = c("log", "trace", "activity", "case"),
    append = deprecated(),
    append_column = NULL,
    sort = TRUE,
    eventlog = deprecated()
)

Arguments

log

log: Object of class log or derivatives (grouped_log, eventlog, activitylog, etc.).

level

character (default "log"): Level of granularity for the analysis: "log" (default), "trace", "case", or "activity". For more information, see vignette("metrics", "edeaR") and 'Details' below.

append

logical (default FALSE) [Deprecated]: The arguments append and append_column have been deprecated in favour of augment. Indicating whether to append results to original log. Ignored when level is "log" or "trace".

append_column

[Deprecated] The arguments append and append_column have been deprecated in favour of augment. Which of the output columns to append to log, if append = TRUE. Default column depends on chosen level.

sort

logical (default TRUE): Sort output on count. Only for levels with frequency count output.

eventlog

[Deprecated]: please use log instead.

Details

Argument level has the following options:

- At log level, this metric shows the summary statistics of the frequency of activities throughout the complete log.
On case level, this metric shows the absolute and relative number of times the different activity types occur in each case. The absolute number shows the number of distinct activity types that occur in each of the cases. The relative number is calculated based on the total activity executions in the case.

On trace level, this metric presents the absolute and relative number of times a specific activity type occurs in each trace.

On activity level, this metric provides the absolute and relative frequency of a specific activity in the complete log.

Methods (by class)

- `activity_frequency(eventlog)`: Computes the activity frequency for an eventlog.
- `activity_frequency(grouped_eventlog)`: Computes the activity frequency for a grouped_eventlog.
- `activity_frequency(activitylog)`: Computes the activity frequency for an activitylog.
- `activity_frequency(grouped_activitylog)`: Computes the activity frequency for a grouped_activitylog.

References


See Also

Other metrics: activity_presence(), end_activities(), idle_time(), number_of_repetitions(), number_of_selfloops(), number_of_traces(), processing_time(), resource_frequency(), resource_involvement(), resource_specialisation(), start_activities(), throughput_time(), trace_coverage(), trace_length()

---

`activity_presence`  
**Metric: Activity Presence**

**Description**

Calculates for each activity type in what percentage of cases it is present.

**Usage**

```r
activity_presence(
  log,
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)
```

## S3 method for class 'eventlog'
activity_presence(
    log,
    append = deprecated(),
    append_column = NULL,
    sort = TRUE,
    eventlog = deprecated()
)

## S3 method for class 'grouped_eventlog'
activity_presence(
    log,
    append = deprecated(),
    append_column = NULL,
    sort = TRUE,
    eventlog = deprecated()
)

## S3 method for class 'activitylog'
activity_presence(
    log,
    append = deprecated(),
    append_column = NULL,
    sort = TRUE,
    eventlog = deprecated()
)

## S3 method for class 'grouped_activitylog'
activity_presence(
    log,
    append = deprecated(),
    append_column = NULL,
    sort = TRUE,
    eventlog = deprecated()
)

Arguments

log: Object of class log or derivatives (grouped_log, eventlog, activitylog, etc.).

append: logical (default FALSE) [Deprecated]: The arguments append and append_column have been deprecated in favour of augment. Indicating whether to append results to original log. Ignored when level is "log" or "trace".

append_column: [Deprecated] The arguments append and append_column have been deprecated in favour of augment. Which of the output columns to append to log, if append = TRUE. Default column depends on chosen level.
sort logical (default TRUE): Sort output on count. Only for levels with frequency count output.

eventlog [Deprecated]: please use log instead.

Details

An indication of variance can be the presence of the activities in the different cases. This metric shows for each activity the absolute number of cases in which each activity occurs together with its relative presence.

Methods (by class)

- activity_presence(eventlog): Compute activity presence for an eventlog.
- activity_presence(grouped_eventlog): Compute activity presence for a grouped_eventlog.
- activity_presence(activitylog): Compute activity presence for an activitylog.
- activity_presence(grouped_activitylog): Compute activity presence for a grouped_activitylog.

References


See Also

Other metrics: activity_frequency(), end_activities(), idle_time(), number_of_repetitions(), number_of_selfloops(), number_of_traces(), processing_time(), resource_frequency(), resource_involvement(), resource_specialisation(), start_activities(), throughput_time(), trace_coverage(), trace_length()

Examples

```r
## Not run:
data <- data.frame(case = rep("A",5),
activity_id = c("A","B","C","D","E"),
activity_instance_id = 1:5,
lifecycle_id = rep("complete",5),
timestamp = 1:5,
resource = rep("resource 1", 5))

log <- bupaR::eventlog(data,case_id = "case",
activity_id = "activity_id",
activity_instance_id = "activity_instance_id",
lifecycle_id = "lifecycle_id",
timestamp = "timestamp",
resource_id = "resource")

activity_presence(log)

## End(Not run)
```
add_fixed_holiday  Add fixed holiday to work schedule

Description
Add fixed holiday to work schedule

Usage
add_fixed_holiday(work_schedule, name, month, day)

Arguments
- work_schedule: Work schedule created with create_work_schedule
- name: Name of holiday
- month: Month in which fixed holiday takes place
- day: Day of fixed holiday

add_floating_holiday  Add floating holiday to work schedule

Description
Add floating holiday to work schedule

Usage
add_floating_holiday(work_schedule, name, dates)

Arguments
- work_schedule: Work schedule created with create_work_schedule
- name: Name of holiday
- dates: Dates of floating holiday. Make sure to list all dates relevant to your time frame
add_holiday_periods

Add holiday period to work schedule

Description
Add holiday period to work schedule

Usage
add_holiday_periods(work_schedule, from, to)

Arguments
work_schedule Work schedule created with create_work_schedule
from Start of holiday period (included)
to End of holiday period (included)

augment

Augment Log

Description
Augment log with results from metric computation.

Usage
augment(metric, log, columns, prefix = "")

## S3 method for class 'log_metric'
augment(metric, log, columns, prefix = "")

## S3 method for class 'case_metric'
augment(metric, log, columns, prefix = "")

## S3 method for class 'activity_metric'
augment(metric, log, columns, prefix = "")

## S3 method for class 'resource_metric'
augment(metric, log, columns, prefix = "")

## S3 method for class 'resource_activity_metric'
augment(metric, log, columns, prefix = "")

## S3 method for class 'trace_metric'
augment(metric, log, columns, prefix = ")

## S3 method for class 'log_activity_metric'
augment(metric, log, columns, prefix = ")

## S3 method for class 'log_resource_metric'
augment(metric, log, columns, prefix = ")

## S3 method for class 'log_resource_activity_metric'
augment(metric, log, columns, prefix = ")

## S3 method for class 'log_trace_metric'
augment(metric, log, columns, prefix = ")"
calculate_queuing_length

Arguments

- **metric**
  - Metric computed by edeaR

- **log**
  - `log`: Object of class `log` or derivatives (grouped_log, eventlog, activitylog, etc.) that was used to compute the metric.

- **columns**
  - `character` vector: Column names from the metric that you want to add to the log. If missing, defaults to all columns.

- **prefix**
  - `character`: Prefix to be added to the newly added metric columns in the log.

Value

Object of class `log` or derivatives (grouped_log, eventlog, activitylog, etc.). Same class as the log input.

Methods (by class)

- `augment(log_metric)`: Augment log metric
- `augment(case_metric)`: Augment case metric
- `augment(activity_metric)`: Augment activity metric
- `augment(resource_metric)`: Augment resource metric
- `augment(resource_activity_metric)`: Augment resource-activity metric
- `augment(trace_metric)`: Augment trace metric

Examples

```r
## Not run:
sepsis %>%
  throughput_time("case") %>%
  augment(sepsis)
## End(Not run)
```

---

calculate_queuing_length

*Calculate queuing length*

Description

[Experimental]

Usage

```r
calculate_queuing_length(
  queueing_times,
  level = c("log", "activity", "resource"),
  time_interval
)
```
Arguments

queueing_times Object of class queuing_times, returned by calculate_queuing_times.
level character (default "log"): Level of granularity for the analysis: "log", "activity", "resource". For more information, see 'Details' below.
time_interval The time interval after which the queue length should be calculated. For more information, see 'Details' below and the by argument of seq.Date.

Details

Argument level has the following options:

- At log level, this metric calculates the total number of activity instances that are queued at a given moment in time.
- At resource level, this metric calculates the total number activity instances that are queued for a given resource.
- On activity level, this metric calculates the total number of activity instances that are queue for a given activity type.

Argument time_interval has the following options (see also the by argument of seq.Date):

- A numeric as number of days.
- An object of class difftime.
- A character string, which could be one of "day", "week", "month", "quarter", or "year". The first day for which queue length is calculated, is the first timestamp found in the log.

See Also

calculate_queuing_times, seq.Date

description

[Experimental]

Usage

calculate_queuing_times(
  log,
  units = c("auto", "secs", "mins", "hours", "days", "weeks"),
  eventlog = deprecated()
)

## S3 method for class 'eventlog'
calculate_queuing_times(
  log,
  units = c("auto", "secs", "mins", "hours", "days", "weeks"),
  eventlog = deprecated()
)

### S3 method for class 'activitylog'

```r
calculate_queuing_times(
  log,
  units = c("auto", "secs", "mins", "hours", "days", "weeks"),
  eventlog = deprecated()
)
```

**Arguments**

- **log**: Object of class `log` or derivatives (`grouped_log`, `eventlog`, `activitylog`, etc.).
- **units**: `character` (default "auto"): The time unit in which the throughput times should be reported. Should be one of the following values: "auto" (default), "secs", "mins", "hours", "days", "weeks". See also the `units` argument of `difftime`.
- **eventlog** [Deprecated]: please use `log` instead.

**Value**

Returns a list of all the activity instances, with the time they started, and the time since they were queued. Notice that this does not take into account any process model notion! The time since they are queued is the completion time of the previous activity in the log.

**Methods (by class)**

- `calculate_queuing_times(eventlog)`: Calculate queueing times for `eventlog` and `grouped_eventlog`.
- `calculate_queuing_times(activitylog)`: Calculate queueing times for `activitylog` and `grouped_activitylog`.

**References**


**See Also**

`difftime`
change_day

Adjust days in work schedule

Description

Adjust days in work schedule

Usage

change_day(work_schedule, day, start_time, end_time)

Arguments

work_schedule  Work schedule created with create_work_schedule
day            A numeric vector containing the days to be changed. 1 = monday.
start_time     The new start time for selected days (hh:mm:ss)
end_time       The new end time for selected days (hh:mm:ss)

create_work_schedule

Create work schedule

Description

Create work schedule

Usage

create_work_schedule(start_time = "9:00:00", end_time = "17:00:00")

Arguments

start_time     Character indicating the usual start time for workdays (hh:mm:ss)
end_time       Character indicating the usual end time for workdays (hh:mm:ss)
Description

Analyse the end activities in the process.

Usage

```r
end_activities(
  log,
  level = c("log", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)
```

## S3 method for class 'eventlog'
```r
end_activities(
  log,
  level = c("log", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)
```

## S3 method for class 'grouped_eventlog'
```r
end_activities(
  log,
  level = c("log", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
 (487,1289),(532,1289)
  eventlog = deprecated()
)
```

## S3 method for class 'activitylog'
```r
end_activities(
  log,
  level = c("log", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)
```
## S3 method for class 'grouped_activitylog'
end_activities(
  log,
  level = c("log", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)

### Arguments

- **log**: Object of class log or derivatives (grouped_log, eventlog, activitylog, etc.).
- **level**: character (default "log"): Level of granularity for the analysis: "log" (default), "case", "activity", "resource", or "resource-activity". For more information, see vignette("metrics", "edeaR") and 'Details' below.
- **append**: logical (default FALSE) [Deprecated]: The arguments append and append_column have been deprecated in favour of augment. Indicating whether to append results to original log. Ignored when level is "log" or "trace".
- **append_column**: [Deprecated] The arguments append and append_column have been deprecated in favour of augment. Which of the output columns to append to log, if append = TRUE. Default column depends on chosen level.
- **sort**: logical (default TRUE): Sort output on count. Only for levels with frequency count output.
- **eventlog**: [Deprecated]: please use log instead.

### Details

Argument level has the following options:

- At log level, this metric shows the absolute and relative number of activities that are the last activity in one or more of the cases.
- On case level, this metric provides an overview of the end activity of each case.
- On activity level, this metric calculates for each activity the absolute and relative number of cases that end with this activity type. Similar to the start_activities metric, the relative number is calculated as a portion of the number of cases, being the number of "opportunities" that an activity could be the end activity. The cumulative sum is added to have an insight in the number of activities that is required to cover a certain part of the total.
- At resource level, an overview of which resources execute the last activity per case is provided.
- On resource-activity level, this metric shows for each occurring resource-activity combination the absolute and relative number of times this resource executes this activity as an end activity in a case.
Methods (by class)

- `end_activities(eventlog)`: Computes the end activities for an `eventlog`.
- `end_activities(grouped_eventlog)`: Computes the end activities for a `grouped_eventlog`.
- `end_activities(activitylog)`: Computes the end activities for an `activitylog`.
- `end_activities(grouped_activitylog)`: Computes the end activities for a `grouped_activitylog`.

References


See Also

- `start_activities`

Other metrics: `activity_frequency()`, `activity_presence()`, `idle_time()`, `number_of_repetitions()`, `number_of_selfloops()`, `number_of_traces()`, `processing_time()`, `resource_frequency()`, `resource_involvement()`, `resource_specialisation()`, `start_activities()`, `throughput_time()`, `trace_coverage()`, `trace_length()`

filter_activity  Filter Activity

Description

Filters the log based on activities

Usage

```
filter_activity(log, activities, reverse = FALSE, eventlog = deprecated())
```

## S3 method for class 'log'
```
filter_activity(log, activities, reverse = FALSE, eventlog = deprecated())
```

## S3 method for class 'grouped_log'
```
filter_activity(log, activities, reverse = FALSE, eventlog = deprecated())
```

Arguments

- `log`: Object of class `log` or derivatives (`grouped_log`, `eventlog`, `activitylog`, etc.).
- `activities`: `character` vector: Containing one or more activity identifiers.
- `reverse`: `logical` (default `FALSE`): Indicating whether the selection should be reversed.
- `eventlog` [Deprecated]: please use `log` instead.
filter_activity_frequency

Description

Filters the log based on frequency of activities.

Usage

```r
filter_activity_frequency(
  log,
  interval = NULL,  
  percentage = NULL,
  reverse = FALSE,
  eventlog = deprecated()
)
```

## S3 method for class 'log'

```r
filter_activity_frequency(
```
log,  
interval = NULL,  
percentage = NULL,  
reverse = FALSE,  
eventlog = deprecated()  
)

## S3 method for class 'grouped_log'
filter_activity_frequency(
  log,  
interval = NULL,  
percentage = NULL,  
reverse = FALSE,  
eventlog = deprecated()
)

Arguments

log log: Object of class log or derivatives (grouped_log, eventlog, activitylog, etc.).
percentage, interval  
The target coverage of activity instances. Provide either percentage or interval.
percentage (numeric): A percentile of p will return the most common activity
  types of the log, which account for at least p% of the activity instances.
interval (numeric vector of length 2): An activity frequency interval. Half
  open interval can be created using NA.
For more information, see 'Details' below.

reverse logical (default FALSE): Indicating whether the selection should be reversed.
eventlog [Deprecated]: please use log instead.

Details

Filtering the log based on activity frequency can be done in two ways: using an interval of allowed
frequencies, or specify a coverage percentage:

- percentage: When filtering using a percentage p%, the filter will return p% of the activity
  instances, starting from the activity labels with the highest frequency. The filter will retain
  additional activity labels as long as the number of activity instances does not exceed the per-
  centage threshold.
- interval: When filtering using an interval, activity labels will be retained when their absolute
  frequency fall in this interval. The interval is specified using a numeric vector of length 2. Half
  open intervals can be created by using NA, e.g., c(10, NA) will select activity labels which
  occur 10 times or more.

Value

When given an object of type log, it will return a filtered log. When given an object of type
grouped_log, the filter will be applied in a stratified way (i.e. each separately for each group). The
returned log will be grouped on the same variables as the original log.
**filter_activity_instance**

**Methods (by class)**

- `filter_activity_frequency(log)`: Filters activities for a *log*.
- `filter_activity_frequency(grouped_log)`: Filters activities for a *grouped_log*.

**References**


**See Also**

Other filters: `filter_activity_instance()`, `filter_activity_presence()`, `filter_activity()`, `filter_case_condition()`, `filter_case()`, `filter_endpoints_condition()`, `filter_endpoints()`, `filter_flow_time()`, `filter_idle_time()`, `filter_infrequent_flows()`, `filter_lifecycle_presence()`, `filter_lifecycle()`, `filter_precedence_condition()`, `filter_precedence_resource()`, `filter_precendence()`, `filter_processing_time()`, `filter_resource_frequency()`, `filter_resource()`, `filter_throughput_time()`, `filter_time_period()`, `filter_trace_frequency()`, `filter_trace_length()`, `filter_trace()`, `filter_trim_lifecycle()`

---

**filter_activity_instance**

*Filter Activity Instance*

**Description**

Filters the log based on activity instance identifier. This method has an `activity_instances` argument, to which a vector of identifiers can be given. The selection can be negated with the `reverse` argument.

**Usage**

```r
filter_activity_instance(
  log,
  activity_instances,
  reverse = FALSE,
  eventlog = deprecated()
)
```

```r
# S3 method for class 'eventlog'
filter_activity_instance(
  log,
  activity_instances,
  reverse = FALSE,
  eventlog = deprecated()
)
```
filter_activity_instance

## S3 method for class 'grouped_eventlog'
filter_activity_instance(
  log,
  activity_instances,
  reverse = FALSE,
  eventlog = deprecated()
)

Arguments

log          eventlog: Object of class eventlog or derivatives (grouped_eventlog).
activity_instances
  A vector of activity instance identifiers.
reverse       logical (default FALSE): Indicating whether the selection should be reversed.
eventlog      [Deprecated]: please use log instead.

Value

When given an object of type log, it will return a filtered log. When given an object of type grouped_log, the filter will be applied in a stratified way (i.e. each separately for each group). The returned log will be grouped on the same variables as the original log.

Methods (by class)

- filter_activity_instance(eventlog): Filters activities for an eventlog.
- filter_activity_instance(grouped_eventlog): Filters activities for a grouped_eventlog.

References


See Also

Other filters: filter_activity_frequency(), filter_activity_presence(), filter_activity(), filter_case_condition(), filter_case(), filter_endpoints_condition(), filter_endpoints(), filter_flow_time(), filter_idle_time(), filter_infrequent_flows(), filter.lifecycle Presence(), filter.lifecycle(), filter.precedence_condition(), filter.precedence_resource(), filter.precedence(), filter.processing.time(), filter.resource_frequency(), filter.resource(), filter_throughput.time(), filter.time.period(), filter.trace.frequency(), filter.trace.length(), filter.trace(), filter.trim.lifecycle(), filter.trim()
**filter_activity_presence**

*Filter Activity Presence*

**Description**

Filters cases based on the presence (or absence) of activities.

**Usage**

```r
filter_activity_presence(
  log,
  activities = NULL,
  method = c("all", "none", "one_of", "exact", "only"),
  reverse = FALSE,
  eventlog = deprecated()
)
```

```r
## S3 method for class 'log'
filter_activity_presence(
  log,
  activities = NULL,
  method = c("all", "none", "one_of", "exact", "only"),
  reverse = FALSE,
  eventlog = deprecated()
)
```

```r
## S3 method for class 'grouped_log'
filter_activity_presence(
  log,
  activities = NULL,
  method = c("all", "none", "one_of", "exact", "only"),
  reverse = FALSE,
  eventlog = deprecated()
)
```

**Arguments**

- **log**: Object of class `log` or derivatives (`grouped_log`, `eventlog`, `activitylog`, etc.).
- **activities**: character vector: Containing one or more activity identifiers.
- **method**: character (default "all"): Filter method: "all" (default), "none", "one_of", "exact", or "only". For more information, see Details below.
- **reverse**: logical (default FALSE): Indicating whether the selection should be reversed.
- **eventlog**: [Deprecated]: please use log instead.
Details

This function allows to filter cases that contain certain activities. It requires as input a vector containing one or more activity labels and it has a `method` argument with following options:

- "all" means that all the specified activity labels must be present for a case to be selected.
- "none" means that they are not allowed to be present.
- "one_of" means that at least one of them must be present.
- "exact" means that only exactly these activities can be present (although multiple times and in random orderings).
- "only" means that only (a set of) these activities are allowed to be present.

When only one activity label is supplied, note that methods "all" and "one_of" will be identical.

Value

When given an object of type `log`, it will return a filtered `log`. When given an object of type `grouped_log`, the filter will be applied in a stratified way (i.e. each separately for each group). The returned log will be grouped on the same variables as the original log.

Methods (by class)

- `filter_activity_presence(log)`: Filters activities for a `log`.
- `filter_activity_presence(grouped_log)`: Filters activities for a `grouped_log`.

References


See Also

Other filters: `filter_activity_frequency()`, `filter_activity_instance()`, `filter_activity()`, `filter_case_condition()`, `filter_case()`, `filter_endpoints_condition()`, `filter_endpoints()`, `filter_flow_time()`, `filter_idle_time()`, `filter_infrequent_flows()`, `filter_lifecycle_presence()`, `filter_lifecycle()`, `filter_precedence_condition()`, `filter_precedence_resource()`, `filter_precedence()`, `filter_processing_time()`, `filter_resource_frequency()`, `filter_resource()`, `filter_throughput_time()`, `filter_time_period()`, `filter_trace_frequency()`, `filter_trace_length()`, `filter_trace()`, `filter_trim_lifecycle()`, `filter_trim()`
Description

Filters the log based on case identifier. This method has a cases argument, to which a vector of identifiers can be given. The selection can be negated with the reverse argument.

Usage

```r
filter_case(log, cases, reverse = FALSE, eventlog = deprecated())
```

```r
## S3 method for class 'log'
filter_case(log, cases, reverse = FALSE, eventlog = deprecated())
```

```r
## S3 method for class 'grouped_log'
filter_case(log, cases, reverse = FALSE, eventlog = deprecated())
```

Arguments

- `log` Object of class `log` or derivatives (`grouped_log, eventlog, activitylog,` etc.).
- `cases` character vector: A vector of cases identifiers.
- `reverse` logical (default FALSE): Indicating whether the selection should be reversed.
- `eventlog` [Deprecated]; please use `log` instead.

Value

When given an object of type `log`, it will return a filtered `log`. When given an object of type `grouped_log`, the filter will be applied in a stratified way (i.e. each separately for each group). The returned log will be grouped on the same variables as the original log.

Methods (by class)

- `filter_case(log)`: Filters cases for a `log`.
- `filter_case(grouped_log)`: Filters cases for a `grouped_log`.

References

See Also

Other filters: `filter_activity_frequency()`, `filter_activity_instance()`, `filter_activity_presence()`, `filter_activity()`, `filter_case_condition()`, `filter_endpoints_condition()`, `filter_endpoints()`, `filter_flow_time()`, `filter_idle_time()`, `filter_infrequent_flows()`, `filter_lifecycle_presence()`, `filter_lifecycle()`, `filter_precedence_condition()`, `filter_precedence_resource()`, `filter_precedence()`, `filter_processing_time()`, `filter_resource_frequency()`, `filter_resource()`, `filter_throughput_time()`, `filter_time_period()`, `filter_trace_frequency()`, `filter_trace_length()`, `filter_trace()`, `filter_trim_lifecycle()`, `filter_trim()`

---

**filter_case_condition**  
*Filter Case Condition*

**Description**

Filters cases using a condition. Only keeps cases if the condition is valid for at least one event.

**Usage**

```r
filter_case_condition(
  log,
  ...,  
  condition = NULL,
  reverse = FALSE,
  eventlog = deprecated()
)
```

```r
## S3 method for class 'log'
filter_case_condition(
  log,
  ...,  
  condition = deprecated(),
  reverse = FALSE,
  eventlog = deprecated()
)
```

```r
## S3 method for class 'grouped_log'
filter_case_condition(
  log,
  ...,  
  condition = deprecated(),
  reverse = FALSE,
  eventlog = deprecated()
)
```
filter_case_condition

Arguments

log: Object of class log or derivatives (grouped_log, eventlog, activitylog, etc.).

... data-masking: Expressions that return a logical value, and are defined in terms of the variables in log. If multiple expressions are included, they are combined with the & operator. Only rows for which all conditions evaluate to TRUE are kept. For more information, see filter.

condition [Deprecated]: please use data-masking expressions instead.
reverse logical (default FALSE): Indicating whether the selection should be reversed.
eventlog [Deprecated]: please use log instead.

Value

When given an object of type log, it will return a filtered log. When given an object of type grouped_log, the filter will be applied in a stratified way (i.e. each separately for each group). The returned log will be grouped on the same variables as the original log.

Methods (by class)

- filter_case_condition(log): Filters cases for a log.
- filter_case_condition(grouped_log): Filters cases for a grouped_log.

References


See Also

filter

Other filters: filter_activity_frequency(), filter_activity_instance(), filter_activity_presence(), filter_activity(), filter_case(), filter_endpoints_condition(), filter_endpoints(), filter_flow_time(), filter_idle_time(), filter_infrequent_flows(), filter_lifecycle_presence(), filter_lifecycle(), filter_precedence_condition(), filter_precedence_resource(), filter_precedence(), filter_processing_time(), filter_resource_frequency(), filter_resource(), filter_throughput_time(), filter_time_period(), filter_trace_frequency(), filter_trace_length(), filter_trace(), filter_trim_lifecycle(), filter_trim()
filter_endpoints  Filter Start and End Activities

Description
Filters the log based on a provided set of start and end activities

The filter_endpoints method filters cases based on the first and last activity label. It can be used in two ways: by specifying vectors with allowed start activities and/or allowed end activities, or by specifying a percentile. In the latter case, the percentile value will be used as a cut off. For example, when set to 0.9, it will select the most common endpoint pairs which together cover at least 90% of the cases, and filter the log accordingly.

Usage

```r
filter_endpoints(
  log, 
  start_activities = NULL, 
  end_activities = NULL, 
  percentage = NULL, 
  reverse = FALSE, 
  eventlog = deprecated()
)
```

## S3 method for class 'log'

```r
filter_endpoints(
  log, 
  start_activities = NULL, 
  end_activities = NULL, 
  percentage = NULL, 
  reverse = FALSE, 
  eventlog = deprecated()
)
```

## S3 method for class 'grouped_log'

```r
filter_endpoints(
  log, 
  start_activities = NULL, 
  end_activities = NULL, 
  percentage = NULL, 
  reverse = FALSE, 
  eventlog = deprecated()
)
```

Arguments

- `log`: Object of class `log` or derivatives (grouped_log, eventlog, activitylog, etc.).
**Description**

Filters cases where the first and/or last activity adhere to the specified conditions.
Usage

```r
filter_endpoints_condition(
    log,
    start_condition = NULL,
    end_condition = NULL,
    reverse = FALSE,
    eventlog = deprecated()
)
```

```r
## S3 method for class 'eventlog'
filter_endpoints_condition(
    log,
    start_condition = NULL,
    end_condition = NULL,
    reverse = FALSE,
    eventlog = deprecated()
)
```

```r
## S3 method for class 'grouped_log'
filter_endpoints_condition(
    log,
    start_condition = NULL,
    end_condition = NULL,
    reverse = FALSE,
    eventlog = deprecated()
)
```

```r
## S3 method for class 'activitylog'
filter_endpoints_condition(
    log,
    start_condition = NULL,
    end_condition = NULL,
    reverse = FALSE,
    eventlog = deprecated()
)
```

```r
filter_endpoints_conditions(
    log,
    start_condition = NULL,
    end_condition = NULL,
    reverse = FALSE,
    eventlog = deprecated()
)
```

Arguments

- `log`: Object of class `log` or derivatives (`grouped_log, eventlog, activitylog`, etc.).
filter_flow_time

start_condition, end_condition
A logical condition.

reverse logical (default FALSE): Indicating whether the selection should be reversed.

eventlog [Deprecated]; please use log instead.

When given an object of type log, it will return a filtered log. When given an object of type grouped_log, the filter will be applied in a stratified way (i.e. each separately for each group). The returned log will be grouped on the same variables as the original log.

Methods (by class)

• filter_endpoints_condition(eventlog): Filters cases for an eventlog.
• filter_endpoints_condition(grouped_log): Filters cases for a grouped_log.
• filter_endpoints_condition(activitylog): Filters cases for an activitylog.

References


See Also

Other filters: filter_activity_frequency(), filter_activity_instance(), filter_activity_presence(), filter_activity(), filter_case_condition(), filter_case(), filter_endpoints(), filter_flow_time(), filter_idle_time(), filter_infrequent_flows(), filter_lifecycle_presence(), filter_lifecycle(), filter_precedence_condition(), filter_precedence_resource(), filter_precedence(), filter_processing_time(), filter_resource_frequency(), filter_resource(), filter_throughput_time(), filter_time_period(), filter_trace_frequency(), filter_trace_length(), filter_trace(), filter_trim_lifecycle(), filter_trim()
units = c("secs", "mins", "hours", "days", "weeks")

## S3 method for class 'log'
filter_flow_time(
  log,
  from,
  to,
  interval,
  reverse = FALSE,
  units = c("secs", "mins", "hours", "days", "weeks")
)

## S3 method for class 'grouped_log'
filter_flow_time(
  log,
  from,
  to,
  interval,
  reverse = FALSE,
  units = c("secs", "mins", "hours", "days", "weeks")
)

Arguments

log log: Object of class log or derivatives (grouped_log, eventlog, activitylog, etc.).
from, to character vector of length 1: The antecedent and consequent to filter on. Both are character vectors containing exactly one activity identifier.
interval numeric vector of length 2: A duration interval. Half open interval can be created using NA.
reverse logical (default FALSE): Indicating whether the selection should be reversed.
units character (default "secs"): The time unit in which the processing times should be reported. Should be one of the following values: "secs" (default), "mins", "hours", "days", "weeks". See also the units argument of difftime().

Value

When given an object of type log, it will return a filtered log. When given an object of type grouped_log, the filter will be applied in a stratified way (i.e. each separately for each group). The returned log will be grouped on the same variables as the original log.

Methods (by class)

- filter_flow_time(log): Filters on flow time for a bupaR::log.
- filter_flow_time(grouped_log): Filters on flow time for a bupaR::grouped_log.
**filter_idle_time**

**Description**

Filters cases based on their *idle_time*.

This filter can be used by using an interval or by using a percentage. The percentage will always start with the cases with the lowest idle time first and stop including cases when the specified percentile is reached. On the other hand, an absolute interval can be defined instead to filter cases which have an idle time in this interval. The time units in which this interval is defined can be supplied with the units argument.

**Usage**

```r
filter_idle_time(
  log,
  interval = NULL,
  percentage = NULL,
  reverse = FALSE,
  units = c("secs", "mins", "hours", "days", "weeks")
)
```

```r
## S3 method for class 'log'
filter_idle_time(
  log,
  interval = NULL,
  percentage = NULL,
  reverse = FALSE,
  units = c("secs", "mins", "hours", "days", "weeks")
)
```

**References**


**See Also**

processing_time(), difftime() Other filters: filter_activity_frequency(), filter_activity_instance(), filter_activity_presence(), filter_activity(), filter_case_condition(), filter_case(), filter_endpoints_condition(), filter_endpoints(), filter_idle_time(), filter_infrequent_flows(), filter_lifecycle_presence(), filter_lifecycle(), filter_precedence_condition(), filter_precedence_resource(), filter_precedence(), filter_processing_time(), filter_resource_frequency(), filter_resource(), filter_throughput_time(), filter_time_period(), filter_trace_frequency(), filter_trace_length(), filter_trace(), filter_trim_lifecycle(), filter_trim()
## S3 method for class 'grouped_log'
filter_idle_time(
  log,
  interval = NULL,
  percentage = NULL,
  reverse = FALSE,
  units = c("secs", "mins", "hours", "days", "weeks")
)

### Arguments

- **log**: Object of class `log` or derivatives (`grouped_log`, `eventlog`, `activitylog`, etc.).
- **interval** and **percentage**: Provide either `interval` or `percentage`. 
  - `interval` (numeric vector of length 2): A duration interval. Half open interval can be created using `NA`.
  - `percentage` (numeric): A percentage to be used for relative filtering.
- **reverse**: logical (default `FALSE`): Indicating whether the selection should be reversed.
- **units**: character (default "secs"): The time unit in which the processing times should be reported. Should be one of the following values: "secs" (default), "mins", "hours", "days", "weeks". See also the units argument of `difftime()`.

### Value

When given an object of type `log`, it will return a filtered `log`. When given an object of type `grouped_log`, the filter will be applied in a stratified way (i.e. each separately for each group). The returned log will be grouped on the same variables as the original log.

### Methods (by class)

- `filter_idle_time(log)`: Filters cases for a `log`.
- `filter_idle_time(grouped_log)`: Filters cases for a `grouped_log`.

### References


### See Also

`idle_time()`, `difftime()`

Other filters: `filter_activity_frequency()`, `filter_activity_instance()`, `filter_activity_presence()`, `filter_activity()`, `filter_case_condition()`, `filter_case()`, `filter_endpoints_condition()`, `filter_endpoints()`, `filter_flow_time()`, `filter_infrequent_flows()`, `filter.lifecycle_presence()`, `filter.lifecycle()`, `filter.precedence.condition()`, `filter.precedence.resource()`, `filter.precedence()`, `filter.processing.time()`, `filter.resource.frequency()`, `filter.resource()`, `filter_throughput.time()`, `filter.time.period()`, `filter.trace.frequency()`, `filter.trace.length()`, `filter.trace()`, `filter.trim.lifecycle()`,
filter_infrequent_flows

Filter Infrequent Flows

Description

[Experimental]
Filter cases based on infrequent flows.

Usage

filter_infrequent_flows(log, min_n = 2, eventlog = deprecated())

## S3 method for class quotesingle.Var

filter_infrequent_flows(log, min_n = 2, eventlog = deprecated())

## S3 method for class quotesingle.Var
grouped_eventlog

filter_infrequent_flows(log, min_n = 2, eventlog = deprecated())

## S3 method for class quotesingle.Var

activitylog

filter_infrequent_flows(log, min_n = 2, eventlog = deprecated())

## S3 method for class quotesingle.Var
grouped_activitylog

filter_infrequent_flows(log, min_n = 2, eventlog = deprecated())

Arguments

log log: Object of class log or derivatives (grouped_log, eventlog, activitylog, etc.).

min_n numeric (default 2): Cases containing a flow that occurs less than min_n times are discarded.

eventlog [Deprecated]: please use log instead.

Value

When given an object of type log, it will return a filtered log. When given an object of type grouped_log, the filter will be applied in a stratified way (i.e. each separately for each group). The returned log will be grouped on the same variables as the original log.

Methods (by class)

- filter_infrequent_flows(eventlog): Filters infrequent flows for an eventlog.
- filter_infrequent_flows(grouped_eventlog): Filters infrequent flows for a grouped_eventlog.
- filter_infrequent_flows(activitylog): Filters infrequent flows for an activitylog.
- filter_infrequent_flows(grouped_activitylog): Filters infrequent flows for a grouped_activitylog.
References

See Also
Other filters: `filter_activity_frequency()`, `filter_activity_instance()`, `filter_activity_presence()`, `filter_activity()`, `filter_case_condition()`, `filter_case()`, `filter_endpoints_condition()`, `filter_endpoints()`, `filter_flow_time()`, `filter_idle_time()`, `filter_lifecycle_presence()`, `filter_lifecycle()`, `filter_precedence_condition()`, `filter_precedence_resource()`, `filter_precedence()`, `filter_processing_time()`, `filter_resource_frequency()`, `filter_resource()`, `filter_throughput_time()`, `filter_time_period()`, `filter_trace_frequency()`, `filter_trace_length()`, `filter_trace()`, `filter_trim_lifecycle()`, `filter_trim()`

---

### filter_lifecycle

#### Filter Life Cycle

**Description**
Filters the log based on the life cycle identifier.

**Usage**

```r
filter_lifecycle(
  log,  
lifecycles,  
reverse = FALSE,  
lifecycle = deprecated(),  
eventlog = deprecated()
)
```

```r
## S3 method for class 'eventlog'
filter_lifecycle(
  log,  
lifecycles,  
reverse = FALSE,  
lifecycle = deprecated(),  
eventlog = deprecated()
)
```

```r
## S3 method for class 'grouped_eventlog'
filter_lifecycle(
  log,  
lifecycles,  
reverse = FALSE,  
lifecycle = deprecated(),  
eventlog = deprecated()
)
```
**Arguments**

- **log** `eventlog`: Object of class `eventlog` or derivatives (`grouped_eventlog`).
- **lifecycles** `character` vector: A vector of life cycle identifiers.
- **reverse** `logical` (default FALSE): Indicating whether the selection should be reversed.
- **lifecycle** [Deprecated]; please use `lifecycles` instead.
- **eventlog** [Deprecated]; please use `log` instead.

**Value**

When given an object of type `log`, it will return a filtered `log`. When given an object of type `grouped_log`, the filter will be applied in a stratified way (i.e. each separately for each group). The returned log will be grouped on the same variables as the original log.

**Methods (by class)**

- `filter_lifecycle(eventlog)`: Filters based on life cycle identifiers for an `eventlog`.
- `filter_lifecycle(grouped_eventlog)`: Filters based on life cycle identifiers a `grouped_eventlog`.

**References**


**See Also**

- `lifecycle_id`

Other filters: `filter_activity_frequency()`, `filter_activity_instance()`, `filter_activity_presence()`, `filter_activity()`, `filter_case_condition()`, `filter_case()`, `filter_endpoints_condition()`, `filter_endpoints()`, `filter_flow_time()`, `filter_idle_time()`, `filter_infrequent_flows()`, `filter_lifecycle_presence()`, `filter_precedence_condition()`, `filter_precedence_resource()`, `filter_precedence()`, `filter_processing_time()`, `filter_resource_frequency()`, `filter_resource()`, `filter_throughput_time()`, `filter_time_period()`, `filter_trace_frequency()`, `filter_trace_length()`, `filter_trace()`, `filter_trim_lifecycle()`, `filter_trim()`

**Description**

Filters activity instances based on the presence (or absence) of life cycles.
**Usage**

```r
filter_lifecycle_presence(
  log, 
  lifecycles, 
  method = c("all", "none", "one_of", "exact", "only"), 
  reverse = FALSE, 
  lifecycle = deprecated(), 
  eventlog = deprecated()
)
```

## S3 method for class 'eventlog'

```r
filter_lifecycle_presence(
  log, 
  lifecycles, 
  method = c("all", "none", "one_of", "exact", "only"), 
  reverse = FALSE, 
  lifecycle = deprecated(), 
  eventlog = deprecated()
)
```

## S3 method for class 'grouped_eventlog'

```r
filter_lifecycle_presence(
  log, 
  lifecycles, 
  method = c("all", "none", "one_of", "exact", "only"), 
  reverse = FALSE, 
  lifecycle = deprecated(), 
  eventlog = deprecated()
)
```

**Arguments**

- **log**: Object of class `eventlog` or derivatives (`grouped_eventlog`).
- **lifecycles**: character vector: A vector of life cycle identifiers.
- **method**: character (default "all"): Filter method: "all" (default), "none", "one_of", "exact", or "only". For more information, see **Details** below.
- **reverse**: logical (default FALSE): Indicating whether the selection should be reversed.
- **lifecycle**: [Deprecated]; please use lifecycles instead.
- **eventlog**: [Deprecated]; please use log instead.

**Details**

This function allows to filter activity instances that (do not) contain certain life cycle identifiers. It requires as input a vector containing one or more life cycle labels and it has a method argument with following options:

- "all" means that all the specified life cycle labels must be present for an activity instance to be selected.
• "none" means that they are not allowed to be present.
• "one_of" means that at least one of them must be present.
• "exact" means that only exactly these life cycle labels can be present (although multiple times and in random orderings).
• "only" means that only (a set of) these life cycle labels are allowed to be present.

Value
When given an object of type log, it will return a filtered log. When given an object of type grouped_log, the filter will be applied in a stratified way (i.e. each separately for each group). The returned log will be grouped on the same variables as the original log.

Methods (by class)
• filter_lifecycle_presence(eventlog): Filters activity instances on the presence of life cycle labels for an eventlog.
• filter_lifecycle_presence(grouped_eventlog): Filters activity instances on the presence of life cycle labels for a grouped_eventlog.

References

See Also
lifecycle_id
Other filters: filter_activity_frequency(), filter_activity_instance(), filter_activity_presence(), filter_activity(), filter_case_condition(), filter_case(), filter_endpoints_condition(), filter_endpoints(), filter_flow_time(), filter_idle_time(), filter_infrequent_flows(), filter_lifecycle(), filter_precedence_condition(), filter_precedence_resource(), filter_precedence(), filter_processing_time(), filter_resource_frequency(), filter_resource(), filter_throughput_time(), filter_time_period(), filter_trace_frequency(), filter_trace_length(), filter_trace(), filter_trim_lifecycle(), filter_trim()
Usage

```r
filter_precedence(
  log,  
  antecedents,  
  consequents,  
  precedence_type = c("directly_follows", "eventually_follows"),  
  filter_method = c("all", "one_of", "none"),  
  reverse = FALSE,  
  eventlog = deprecated()
)
```

```r
## S3 method for class 'log'
filter_precedence(
  log,  
  antecedents,  
  consequents,  
  precedence_type = c("directly_follows", "eventually_follows"),  
  filter_method = c("all", "one_of", "none"),  
  reverse = FALSE,  
  eventlog = deprecated()
)
```

```r
## S3 method for class 'grouped_log'
filter_precedence(
  log,  
  antecedents,  
  consequents,  
  precedence_type = c("directly_follows", "eventually_follows"),  
  filter_method = c("all", "one_of", "none"),  
  reverse = FALSE,  
  eventlog = deprecated()
)
```

Arguments

- `log`: Object of class `log` or derivatives (`grouped_log`, `eventlog`, `activitylog`, etc.).
- `antecedents`, `consequents`: `character` vector: The set of antecedent and consequent activities. Both are `character` vectors containing at least one activity identifier. All pairs of antecedents and consequents are turned into separate precedence rules.
- `precedence_type`: `character` (default "directly_follows"): When "directly_follows", the consequent activity should happen immediately after the antecedent activities. When "eventually_follows", other events are allowed to happen in between.
- `filter_method`: `character` (default "all"): When "all", only cases where all the relations are valid are preserved. When "one_of", all the cases where at least one of the conditions hold, are
In order to extract a subset of an event log which conforms with a set of precedence rules, one can use the `filter_precedence` method. There are two types of precedence relations which can be tested: activities that should directly follow ("directly_follows") each other, or activities that should eventually follow ("eventually_follows") each other. The type can be set with the `precedence_type` argument.

Further, the filter requires a vector of one or more antecedents (containing activity labels), and one or more consequents.

Finally, a `filter_method` argument can be set. This argument is relevant when there is more than one antecedent or consequent. In such a case, you can specify that all possible precedence combinations must be present ("all"), at least one of them ("one_of"), or none ("none").

Value

When given an object of type `log`, it will return a filtered `log`. When given an object of type `grouped_log`, the filter will be applied in a stratified way (i.e. each separately for each group). The returned log will be grouped on the same variables as the original log.

Methods (by class)

- `filter_precedence(log)`: Filters cases for a `log`.
- `filter_precedence(grouped_log)`: Filters cases for a `grouped_log`.

References


See Also

Other filters: `filter_activity_frequency()`, `filter_activity_instance()`, `filter_activity_presence()`, `filter_activity()`, `filter_case_condition()`, `filter_case()`, `filter_endpoints_condition()`, `filter_endpoints()`, `filter_flow_time()`, `filter_idle_time()`, `filter_infrequent_flows()`, `filter_lifecycle_presence()`, `filter_lifecycle()`, `filter_precedence_condition()`, `filter_precedence_resource()`, `filter_processing_time()`, `filter_resource_frequency()`, `filter_resource()`, `filter_throughput_time()`, `filter_time_period()`, `filter_trace_frequency()`, `filter_trace_length()`, `filter_trace()`, `filter_trim_lifecycle()`, `filter_trim()`
Examples

```r
eventdataR::patients %>%
  filter_precedence(antecedents = "Triage and Assessment",
                    consequents = "Blood test",
                    precedence_type = "directly_follows") %>%
bupaR::traces()

eventdataR::patients %>%
  filter_precedence(antecedents = "Triage and Assessment",
                    consequents = c("Blood test", "X-Ray", "MRI SCAN"),
                    precedence_type = "eventually_follows",
                    filter_method = "one_of") %>%
bupaR::traces()
```

---

**filter_precedence_condition**

*Filter Precedence Relations*

---

**Description**

Filters cases based on the precedence relations between two sets of activities. For more information, see `filter_precedence`.

**Usage**

```r
filter_precedence_condition(
  log,
  antecedent_condition, consequent_condition,
  precedence_type = c("directly_follows", "eventually_follows"),
  reverse = FALSE,
  eventlog = deprecated()
)
```

```r
## S3 method for class 'log'
filter_precedence_condition(
  log,
  antecedent_condition, consequent_condition,
  precedence_type = c("directly_follows", "eventually_follows"),
  reverse = FALSE,
  eventlog = deprecated()
)
```

```r
## S3 method for class 'grouped_log'
filter_precedence_condition(
```
filter_precedence_condition

```r
log,
antecedent_condition,
consequent_condition,
precedence_type = c("directly_follows", "eventually_follows"),
reverse = FALSE,
eventlog = deprecated()
)

Arguments

log: Object of class log or derivatives (grouped_log, eventlog, activitylog, etc.).
antecedent_condition, consequent_condition
The antecedent and consequent conditions.
precedence_type
character (default "directly_follows"): When "directly_follows", the consequent activity should happen immediately after the antecedent activities. When "eventually_follows", other events are allowed to happen in between.
reverse
logical (default FALSE): Indicating whether the selection should be reversed.
eventlog
[Deprecated]: please use log instead.

Value

When given an object of type log, it will return a filtered log. When given an object of type grouped_log, the filter will be applied in a stratified way (i.e. each separately for each group). The returned log will be grouped on the same variables as the original log.

Methods (by class)

- filter_precedence_condition(log): Filters cases for a log.
- filter_precedence_condition(grouped_log): Filters cases for a grouped_log.

References


See Also

Other filters: filter_activity_frequency(), filter_activity_instance(), filter_activity_presence(), filter_activity(), filter_case_condition(), filter_case(), filter_endpoints_condition(), filter_endpoints(), filter_flow_time(), filter_idle_time(), filter_infrequent_flows(), filter.lifecycle_presence(), filter.lifecycle(), filter_precedence_resource(), filter_precedence(), filter_processing_time(), filter_resource_frequency(), filter_resource(), filter_throughput_time(), filter_time_period(), filter_trace_frequency(), filter_trace_length(), filter_trace(), filter_trim_lifecycle(), filter.trim()
**Description**

Filters cases based on the precedence relations between two sets of activities, where both antecedent and consequent have to be executed by the same resource.

**Usage**

```r
filter_precedence_resource(
  log,
  antecedents,
  consequents,
  precedence_type = c("directly_follows", "eventually_follows"),
  filter_method = c("all", "one_of", "none"),
  reverse = FALSE,
  eventlog = deprecated()
)
```

```r
## S3 method for class 'eventlog'
filter_precedence_resource(
  log,
  antecedents,
  consequents,
  precedence_type = c("directly_follows", "eventually_follows"),
  filter_method = c("all", "one_of", "none"),
  reverse = FALSE,
  eventlog = deprecated()
)
```

```r
## S3 method for class 'activitylog'
filter_precedence_resource(
  log,
  antecedents,
  consequents,
  precedence_type = c("directly_follows", "eventually_follows"),
  filter_method = c("all", "one_of", "none"),
  reverse = FALSE,
  eventlog = deprecated()
)
```

```r
## S3 method for class 'grouped_log'
filter_precedence_resource(
  log,
  antecedents,
  consequents,
  precedence_type = c("directly_follows", "eventually_follows"),
  filter_method = c("all", "one_of", "none"),
  reverse = FALSE,
  eventlog = deprecated()
)
```
filter_precedence_resource

```r
consquences,
preceodence_type = c("directly_follows", "eventually_follows"),
filter_method = c("all", "one_of", "none"),
reverse = FALSE,
eventlog = deprecated()
)
```

### Arguments

**log**  
Object of class log or derivatives (grouped_log, eventlog, activitylog, etc.).

**antecedents, consequents**  
character vector: The set of antecedent and consequent activities. Both are character vectors containing at least one activity identifier. All pairs of antecedents and consequents are turned into separate precedence rules.

**precedence_type**  
character (default "directly_follows"): When "directly_follows", the consequent activity should happen immediately after the antecedent activities. When "eventually_follows", other events are allowed to happen in between.

**filter_method**  
character (default "all"): When "all", only cases where all the relations are valid are preserved. When "one_of", all the cases where at least one of the conditions hold, are preserved. When "none", none of the relations are allowed.

**reverse**  
logical (default FALSE): Indicating whether the selection should be reversed.

**eventlog**  
[Deprecated]: please use log instead.

### Value

When given an object of type log, it will return a filtered log. When given an object of type grouped_log, the filter will be applied in a stratified way (i.e. each separately for each group). The returned log will be grouped on the same variables as the original log.

### Methods (by class)

- filter_precedence_resource(eventlog): Filters cases for an eventlog.
- filter_precedence_resource(activitylog): Filters cases for an activitylog.
- filter_precedence_resource(grouped_log): Filters cases for a grouped_log.

### References

filter_processing_time

Description

Filters cases based on their processing_time.

This filter can be used by using an interval or by using a percentage. The percentage will always start with the shortest cases first and stop including cases when the specified percentile is reached. On the other hand, an absolute interval can be defined instead to filter cases which have a processing time in this interval. The time units in which this interval is defined can be supplied with the units argument.

Usage

filter_processing_time(
  log,
  interval = NULL,
  percentage = NULL,
  reverse = FALSE,
  units = c("secs", "mins", "hours", "days", "weeks"),
  eventlog = deprecated()
)

## S3 method for class 'log'
filter_processing_time(
  log,
  interval = NULL,
  percentage = NULL,
  reverse = FALSE,
  units = c("secs", "mins", "hours", "days", "weeks"),
  eventlog = deprecated()
)

## S3 method for class 'grouped_log'
filter_processing_time(

See Also

filter_precedence()
Other filters: filter_activity_frequency(), filter_activity_instance(), filter_activity_presence(), filter_activity(), filter_case_condition(), filter_case(), filter_endpoints_condition(), filter_endpoints(), filter_flow_time(), filter_idle_time(), filter_infrequent_flows(), filter_lifecycle_presence(), filter_lifecycle(), filter_precedence_condition(), filter_precedence(), filter_processing_time(), filter_resource_frequency(), filter_resource(), filter_throughput_time(), filter_time_period(), filter_trace_frequency(), filter_trace_length(), filter_trace(), filter_trim_lifecycle(), filter_trim()
`filter_processing_time`  

```r
log,  
interval = NULL,  
percentage = NULL,  
reverse = FALSE,  
units = c("secs", "mins", "hours", "days", "weeks"),  
eventlog = deprecated()
```

**Arguments**

- `log`: Object of class `log` or derivatives (`grouped_log`, `eventlog`, `activitylog`, etc.).
- `interval`, `percentage`: Provide either `interval` or `percentage`.  
  - `interval` (`numeric` vector of length 2): A duration interval. Half open interval can be created using `NA`.  
  - `percentage` (`numeric`): A percentage to be used for relative filtering.
- `reverse`: `logical` (default `FALSE`): Indicating whether the selection should be reversed.
- `units`: `character` (default "secs"): The time unit in which the processing times should be reported. Should be one of the following values: "secs" (default), "mins", "hours", "days", "weeks". See also the units argument of `difftime()`.
- `eventlog`: [Deprecated]: please use `log` instead.

**Value**

When given an object of type `log`, it will return a filtered `log`. When given an object of type `grouped_log`, the filter will be applied in a stratified way (i.e. each separately for each group). The returned log will be grouped on the same variables as the original log.

**Methods (by class)**

- `filter_processing_time(log)`: Filters cases for a `log`.
- `filter_processing_time(grouped_log)`: Filters cases for a `grouped_log`.

**References**


**See Also**

`processing_time()`, `difftime()`

Other filters: `filter_activity_frequency()`, `filter_activity_instance()`, `filter_activity_presence()`, `filter_activity()`, `filter_case_condition()`, `filter_case()`, `filter_endpoints_condition()`, `filter_endpoints()`, `filter_flow_time()`, `filter_idle_time()`, `filter_infrequent_flows()`, `filter_lifecycle_presence()`, `filter_lifecycle()`, `filter_precedence_condition()`, `filter_precedence_resource()`, `filter_precedence()`, `filter_resource_frequency()`, `filter_resource()`, `filter_throughput_time()`, `filter_time_period()`, `filter_trace_frequency()`, `filter_trace_length()`, `filter_trace()`, `filter_trim_lifecycle()`, `filter_trim()`
filter_resource  

Filter Resource

Description
Filters the log based on resource identifiers

This method can be used to filter on resource identifiers. It has a `resources` argument, to which a vector of identifiers can be given. The selection can be negated with the `reverse` argument.

Usage

```r
filter_resource(log, resources, reverse = FALSE, eventlog = deprecated())
```

## S3 method for class 'log'

```r
filter_resource(log, resources, reverse = FALSE, eventlog = deprecated())
```

## S3 method for class 'grouped_log'

```r
filter_resource(log, resources, reverse = FALSE, eventlog = deprecated())
```

Arguments

- `log`  
  Object of class `log` or derivatives (`grouped_log`, `eventlog`, `activitylog`, etc.).

- `resources`  
  Character vector: A vector of resources identifiers.

- `reverse`  
  Logical (default `FALSE`): Indicating whether the selection should be reversed.

- `eventlog`  
  [Deprecated]: please use `log` instead.

Value

When given an object of type `log`, it will return a filtered `log`. When given an object of type `grouped_log`, the filter will be applied in a stratified way (i.e. each separately for each group). The returned log will be grouped on the same variables as the original log.

Methods (by class)

- `filter_resource(log)`: Filters resources for a `log`.
- `filter_resource(grouped_log)`: Filters resources for a `grouped_log`.

References

See Also

Other filters: `filter_activity_frequency()`, `filter_activity_instance()`, `filter_activity_presence()`, `filter_activity()`, `filter_case_condition()`, `filter_case()`, `filter_endpoints_condition()`, `filter_endpoints()`, `filter_flow_time()`, `filter_idle_time()`, `filter_infrequent_flows()`, `filter_lifecycle_presence()`, `filter_lifecycle()`, `filter_precedence_condition()`, `filter_precedence_resource()`, `filter_precedence()`, `filter_processing_time()`, `filter_resource_frequency()`, `filter_throughput_time()`, `filter_time_period()`, `filter_trace_frequency()`, `filter_trace_length()`, `filter_trace()`, `filter_trim_lifecycle()`, `filter_trim()`
Arguments

log \[ \text{log} \]: Object of class \text{log} or derivatives (\text{grouped_log, eventlog, activitylog}, etc.).

percentage, interval

The target coverage of activity instances. Provide either percentage or interval.
percentage \[ \text{numeric} \]: A percentile of \( p \) will return the most common resource types of the log, which account for at least \( p\% \) of the activity instances.

interval \[ \text{numeric vector of length 2} \]: A resource frequency interval. Half open interval can be created using \text{NA}.

For more information, see 'Details' below.

reverse \[ \text{logical} \] (default FALSE): Indicating whether the selection should be reversed.

eventlog \[ \text{[Deprecated]} \]: please use \text{log} instead.

Details
Filtering the log based on resource frequency can be done in two ways: using an interval of allowed frequencies, or specify a coverage percentage:

- percentage: When filtering using a percentage \( p\% \), the filter will return \( p\% \) of the activity instances, starting from the resource labels with the highest frequency. The filter will retain additional resource labels as long as the number of activity instances does not exceed the percentage threshold.

- interval: When filtering using an interval, resource labels will be retained when their absolute frequency fall in this interval. The interval is specified using a numeric vector of length 2. Half open intervals can be created by using \text{NA}, e.g., \( c(10, \text{NA}) \) will select resource labels which occur 10 times or more.

Value

When given an object of type \text{log}, it will return a filtered \text{log}. When given an object of type \text{grouped_log}, the filter will be applied in a stratified way (i.e. each separately for each group). The returned log will be grouped on the same variables as the original log.

Methods (by class)

- \text{filter_resource_frequency(log)}: Filters resources for a \text{log}.

- \text{filter_resource_frequency(grouped_log)}: Filters resources for a \text{grouped_log}.

References

See Also
Other filters: \text{filter_activity_frequency()}, \text{filter_activity_instance()}, \text{filter_activity_presence()}, \text{filter_activity()}, \text{filter_case_condition()}, \text{filter_case()}, \text{filter_endpoints_condition()}, \text{filter_endpoints()}, \text{filter_flow_time()}, \text{filter_idle_time()}, \text{filter_infrequent_flows()}.
**Description**

Filters cases based on their *throughput_time*.

This filter can be used by using an interval or by using a percentage. The percentage will always start with the shortest cases first and stop including cases when the specified percentile is reached. On the other hand, an absolute interval can be defined instead to filter cases which have a throughput time in this interval. The time units in which this interval is defined can be supplied with the *units* argument.

**Usage**

```r
filter_throughput_time(
  log,
  interval = NULL,
  percentage = NULL,
  reverse = FALSE,
  units = c("secs", "mins", "hours", "days", "weeks"),
  eventlog = deprecated()
)
```

## S3 method for class 'log'

```r
filter_throughput_time(
  log,
  interval = NULL,
  percentage = NULL,
  reverse = FALSE,
  units = c("secs", "mins", "hours", "days", "weeks"),
  eventlog = deprecated()
)
```

## S3 method for class 'grouped_log'

```r
filter_throughput_time(
  log,
  interval = NULL,
  percentage = NULL,
  reverse = FALSE,
  units = c("secs", "mins", "hours", "days", "weeks"),
  eventlog = deprecated()
)
```
Arguments

log: Object of class log or derivatives (grouped_log, eventlog, activitylog, etc.).

interval, percentage

Provide either interval or percentage.
interval (numeric vector of length 2): A duration interval. Half open interval can be created using NA.
percentage (numeric): A percentage to be used for relative filtering.

reverse: logical (default FALSE): Indicating whether the selection should be reversed.

units: character (default "secs"): The time unit in which the processing times should be reported. Should be one of the following values: "secs" (default), "mins", "hours", "days", "weeks". See also the units argument of difftime().

eventlog: [Deprecated]; please use log instead.

Value

When given an object of type log, it will return a filtered log. When given an object of type grouped_log, the filter will be applied in a stratified way (i.e. each separately for each group). The returned log will be grouped on the same variables as the original log.

Methods (by class)

- filter_throughput_time(log): Filters cases for a log.
- filter_throughput_time(grouped_log): Filters cases for a grouped_log.

References


See Also

throughput_time(), difftime()
filter_time_period  

Filter Time Period

Description

Function to filter the log using a time period.

Usage

```
filter_time_period(
  log, 
  interval = NULL, 
  filter_method = c("contained", "intersecting", "start", "complete", "trim"), 
  force_trim = FALSE, 
  reverse = FALSE, 
  eventlog = deprecated()
)
```

```
## S3 method for class 'eventlog'
filter_time_period(
  log, 
  interval = NULL, 
  filter_method = c("contained", "intersecting", "start", "complete", "trim"), 
  force_trim = FALSE, 
  reverse = FALSE, 
  eventlog = deprecated()
)
```

```
## S3 method for class 'grouped_eventlog'
filter_time_period(
  log, 
  interval = NULL, 
  filter_method = c("contained", "intersecting", "start", "complete", "trim"), 
  force_trim = FALSE, 
  reverse = FALSE, 
  eventlog = deprecated()
)
```

```
## S3 method for class 'activitylog'
filter_time_period(
  log, 
  interval = NULL, 
  filter_method = c("contained", "intersecting", "start", "complete", "trim"), 
  force_trim = FALSE, 
  reverse = FALSE, 
  eventlog = deprecated()
)
```
## S3 method for class 'grouped_activitylog'

```r
define_time_period(
    log,
    interval = NULL,
    filter_method = c("contained", "intersecting", "start", "complete", "trim"),
    force_trim = FALSE,
    reverse = FALSE,
    eventlog = deprecated()
)
```

### Arguments

- **log**
  - Object of class `log` or derivatives (`grouped_log`, `eventlog`, `activitylog`, etc.).

- **interval**
  - Date or POSIXct vector: A time interval (vector of length 2 of type Date or POSIXct). Half-open intervals can be created with NA.

- **filter_method**
  - character (default "contained"):: Filtering method: "contained" (default), "intersecting", "start", "complete", or "trim". For more information, see 'Details' below.

- **force_trim**
  - logical (default FALSE): If TRUE in combination with filter_method "trim", activity instances on the edges of the interval are cut at the exact edge of the interval.

- **reverse**
  - logical (default FALSE): Indicating whether the selection should be reversed.

- **eventlog**
  - [Deprecated]: please use log instead.

### Details

Event data can be filtered by supplying a time window to the method `filter_time_period`. There are 5 different values for `filter_method`:

- "contained": Keeps all the events related to cases contained in the time period.
- "intersecting": Keeps all the events related to cases in which at least one event started and/or ended in the time period.
- "start": Keeps all the events related to cases started in the time period.
- "complete": Keeps all the events related to cases complete in the time period.
- "trim": Keeps all the events which started and ended in the time frame.

### Value

When given an object of type `log`, it will return a filtered `log`. When given an object of type `grouped_log`, the filter will be applied in a stratified way (i.e. each separately for each group). The returned log will be grouped on the same variables as the original log.
Methods (by class)

- `filter_time_period(eventlog)`: Filters activity instances for an `eventlog`.
- `filter_time_period(grouped_eventlog)`: Filters activity instances for a `grouped_eventlog`.
- `filter_time_period(activitylog)`: Filters activity instances for an `activitylog`.
- `filter_time_period(grouped_activitylog)`: Filters activity instances for a `grouped_activitylog`.

References


See Also

Other filters: `filter_activity_frequency()`, `filter_activity_instance()`, `filter_activity_presence()`, `filter_activity()`, `filter_case_condition()`, `filter_case()`, `filter_endpoints_condition()`, `filter_endpoints()`, `filter_flow_time()`, `filter_idle_time()`, `filter_infrequent_flows()`, `filterLifecycle_presence()`, `filterLifecycle()`, `filterprecedence_condition()`, `filterprecedence_resource()`, `filterprecedence()`, `filterprocessing_time()`, `filterresource_frequency()`, `filterresource()`, `filterthroughput_time()`, `filter_trace_frequency()`, `filter_trace_length()`, `filter_trace()`, `filtertrimLifecycle()`, `filtertrim()`

---

**filter_trace**

*Filter Trace*

Description

Filters the log based on trace identifier.

This method can be used to filter on trace identifier, which can be obtained from `case_list`. It has a `trace_ids` argument, to which a vector of identifiers can be given. The selection can be negated with the `reverse` argument.

Usage

```r
filter_trace(log, trace_ids, reverse = FALSE, eventlog = deprecated())
```

## S3 method for class 'log'

```r
filter_trace(log, trace_ids, reverse = FALSE, eventlog = deprecated())
```

## S3 method for class 'grouped_log'

```r
filter_trace(log, trace_ids, reverse = FALSE, eventlog = deprecated())
```
**filter_trace_frequency**

**Arguments**

- **log**: Object of class `log` or derivatives (`grouped_log`, `eventlog`, `activitylog`, etc.).
- **trace_ids**: character vector: A vector of trace identifiers.
- **reverse**: logical (default `FALSE`): Indicating whether the selection should be reversed.
- **eventlog**: [Deprecated]; please use `log` instead.

**Value**

When given an object of type `log`, it will return a filtered `log`. When given an object of type `grouped_log`, the filter will be applied in a stratified way (i.e. each separately for each group). The returned log will be grouped on the same variables as the original log.

**Methods (by class)**

- `filter_trace(log)`: Filters cases for a `log`.
- `filter_trace(grouped_log)`: Filters cases for a `grouped_log`.

**References**


**See Also**

`case_list`

Other filters: `filter_activity_frequency()`, `filter_activity_instance()`, `filter_activity_presence()`, `filter_activity()`, `filter_case_condition()`, `filter_case()`, `filter_endpoints_condition()`, `filter_endpoints()`, `filter_flow_time()`, `filter_idle_time()`, `filter_infrequent_flows()`, `filter_lifecycle_presence()`, `filter_lifecycle()`, `filter_precedence_condition()`, `filter_precedence_resource()`, `filter_precedence()`, `filter_processing_time()`, `filter_resource_frequency()`, `filter_resource()`, `filter_throughput_time()`, `filter_time_period()`, `filter_trace_frequency()`, `filter_trace_length()`, `filter_trim_lifecycle()`, `filter_trim()`

---

**filter_trace_frequency**

*Filter Trace Frequency*

**Description**

Filters the log based on the frequency of traces, using an interval or a percentile cut off.
filter_trace_frequency

Usage

filter_trace_frequency(
  log,
  interval = NULL,
  percentage = NULL,
  reverse = FALSE,
  eventlog = deprecated()
)

## S3 method for class 'log'
filter_trace_frequency(
  log,
  interval = NULL,
  percentage = NULL,
  reverse = FALSE,
  eventlog = deprecated()
)

## S3 method for class 'grouped_log'
filter_trace_frequency(
  log,
  interval = NULL,
  percentage = NULL,
  reverse = FALSE,
  eventlog = deprecated()
)

Arguments

log: Object of class log or derivatives (grouped_log, eventlog, activitylog, etc.).

percentage, interval
The target coverage of activity instances. Provide either percentage or interval.
percentage (numeric): A percentile of p will select the most common traces
of the log, until at least p% of the cases is covered.
interval (numeric vector of length 2): A trace frequency interval. The filter
will select cases of which the trace has a frequency inside the interval. Half open
interval can be created using NA.
For more information, see 'Details' below.

reverse: logical (default FALSE): Indicating whether the selection should be reversed.

eventlog: [Deprecated]: please use log instead.

Details

Filtering the log based on trace frequency can be done in two ways: using an interval of allowed
frequencies, or specify a coverage percentage:
• **percentage**: When filtering using a percentage p%, the filter will return p% of the cases, starting from the traces with the highest frequency. The filter will retain additional traces as long as the number of activity instances does not exceed the percentage threshold.

• **interval**: When filtering using an interval, traces will be retained when their absolute frequency fall in this interval. The interval is specified using a numeric vector of length 2. Half open intervals can be created by using `NA`, e.g., `c(10, NA)` will select cases with a trace that occurs 10 times or more.

**Value**

When given an object of type `log`, it will return a filtered `log`. When given an object of type `grouped_log`, the filter will be applied in a stratified way (i.e. each separately for each group). The returned log will be grouped on the same variables as the original log.

**Methods (by class)**

- `filter_trace_frequency(log)`: Filters cases for a `log`.
- `filter_trace_frequency(grouped_log)`: Filters cases for a `grouped_log`.

**References**


**See Also**

Other filters: `filter_activity_frequency()`, `filter_activity_instance()`, `filter_activity_presence()`, `filter_activity()`, `filter_case_condition()`, `filter_case()`, `filter_endpoints_condition()`, `filter_endpoints()`, `filter_flow_time()`, `filter_idle_time()`, `filter_infrequent_flows()`, `filter_lifecycle_presence()`, `filter_lifecycle()`, `filter_precedence_condition()`, `filter_precedence_resource()`, `filter_precedence()`, `filter_processing_time()`, `filter_resource_frequency()`, `filter_resource()`, `filter_throughput_time()`, `filter_time_period()`, `filter_trace_length()`, `filter_trace()`, `filter_trim_lifecycle()`,
Usage

filter_trace_length(
  log,
  interval = NULL,
  percentage = NULL,
  reverse = FALSE,
  eventlog = deprecated()
)

## S3 method for class 'log'
filter_trace_length(
  log,
  interval = NULL,
  percentage = NULL,
  reverse = FALSE,
  eventlog = deprecated()
)

## S3 method for class 'grouped_log'
filter_trace_length(
  log,
  interval = NULL,
  percentage = NULL,
  reverse = FALSE,
  eventlog = deprecated()
)

Arguments

log: Object of class log or derivatives (grouped_log, eventlog, activitylog, etc.).

interval, percentage

Provide either interval or percentage.

interval (numeric vector of length 2): A trace length interval. Half open interval can be created using NA.

percentage (numeric): A percentage p to be used for relative filtering.

reverse: logical (default FALSE): Indicating whether the selection should be reversed.

eventlog: [Deprecated]: please use log instead.

Value

When given an object of type log, it will return a filtered log. When given an object of type grouped_log, the filter will be applied in a stratified way (i.e. each separately for each group). The returned log will be grouped on the same variables as the original log.

Methods (by class)

- filter_trace_length(log): Filters cases for a log.
- filter_trace_length(grouped_log): Filters cases for a grouped_log.

References


See Also

trace_length

Other filters: filter_activity_frequency(), filter_activity_instance(), filter_activity_presence(), filter_activity(), filter_case_condition(), filter_case(), filter_endpoints_condition(), filter_endpoints(), filter_flow_time(), filter_idle_time(), filter_infrequent_flows(), filter_lifecycle_presence(), filter_lifecycle(), filter_precedence_condition(), filter_precedence_resource(), filter_precedence(), filter_processing_time(), filter_resource_frequency(), filter_resource(), filter_throughput_time(), filter_time_period(), filter_trace_frequency(), filter_trace(), filter_trim_lifecycle(), filter_trim()

filter_trim

Description

Trim cases from the first event of a set of start activities to the last event of a set of end activities.

One can trim cases by removing one or more activity instances at the start and/or end of a case. Trimming is performed until all cases have a start and/or end point belonging to a set of allowed activity labels. This filter requires a set of allowed start activities and/or a set of allowed end activities. If one of them is not provided it will not trim the cases at this edge. The selection can be reversed, which means that only the trimmed events at the start and end of cases are retained. As such, this argument allows to cut intermediate parts out of traces.

Usage

filter_trim(
  log,
  start_activities = NULL,
  end_activities = NULL,
  reverse = FALSE,
  eventlog = deprecated()
)

## S3 method for class 'eventlog'
filter_trim(
  log,
  start_activities = NULL,
  end_activities = NULL,
  reverse = FALSE,
### S3 method for class 'grouped_eventlog'

```r
filter_trim(
    log, 
    start_activities = NULL, 
    end_activities = NULL, 
    reverse = FALSE, 
    eventlog = deprecated()
)
```

### S3 method for class 'activitylog'

```r
filter_trim(
    log, 
    start_activities = NULL, 
    end_activities = NULL, 
    reverse = FALSE, 
    eventlog = deprecated()
)
```

### S3 method for class 'grouped_activitylog'

```r
filter_trim(
    log, 
    start_activities = NULL, 
    end_activities = NULL, 
    reverse = FALSE, 
    eventlog = deprecated()
)
```

#### Arguments

- `log`: Object of class `log` or derivatives (`grouped_log`, `eventlog`, `activitylog`, etc.).
- `start_activities`, `end_activities`: character vector (default `NULL`): A vector of activity identifiers, or `NULL`.
- `reverse`: logical (default `FALSE`): Indicating whether the selection should be reversed.
- `eventlog`: [Deprecated]: please use `log` instead.

#### Value

When given an object of type `log`, it will return a filtered `log`. When given an object of type `grouped_log`, the filter will be applied in a stratified way (i.e. each separately for each group). The returned log will be grouped on the same variables as the original log.

#### Methods (by class)

- `filter_trim(eventlog)`: Filters activity instances for an `eventlog`.

• filter_trigm(grouped_eventlog): Filters activity instances for a grouped_eventlog.
• filter_trigm(activitylog): Filters activity instances for an activitylog.
• filter_trigm(grouped_activitylog): Filters activity instances for a grouped_activitylog.

References

See Also
Other filters: filter_activity_frequency(), filter_activity_instance(), filter_activity_presence(), filter_activity(), filter_case_condition(), filter_case(), filter_endpoints_condition(), filter_endpoints(), filter_flow_time(), filter_idle_time(), filter_infrequent_flows(), filter_lifecycle_presence(), filter_lifecycle(), filter_precedence_condition(), filter_precedence_resource(), filter_precedence(), filter_processing_time(), filter_resource_frequency(), filter_resource(), filter_throughput_time(), filter_time_period(), filter_trace_frequency(), filter_trace_length(), filter_trace(), filter_trim_lifecycle()

---

filter_trim_lifecycle  Filter Trim Life Cycle

Description
Trim activity instances from the first event of a set of start life cycle labels to the last event of a set of end life cycle labels.

One can trim activity instances by removing one or more events at the start and/or end of the activity instances. Trimming is performed until all activity instances have a start and/or end point belonging to a set of allowed life cycle labels. This filter requires a set of allowed start life cycle labels and/or a set of allowed life cycle labels. If one of them is not provided it will not trim the activity instances at this edge. The selection can be reversed, which means that only the trimmed events at the start and end of activity instances are retained. As such, this argument allows to cut intermediate parts out of activity instances.

Usage
filter_trim_lifecycle(
    log,  # log
    start_lifecycles = NULL,  # start_lifecycles
    end_lifecycles = NULL,  # end_lifecycles
    reverse = FALSE,  # reverse
    start_lifecycle = deprecated(),  # start_lifecycle
    end_lifecycle = deprecated(),  # end_lifecycle
    eventlog = deprecated()  # eventlog
)
filter_trim_lifecycle

## S3 method for class 'eventlog'
filter_trim_lifecycle(
  log,
  start_lifecycles = NULL,
  end_lifecycles = NULL,
  reverse = FALSE,
  start_lifecycle = deprecated(),
  end_lifecycle = deprecated(),
  eventlog = deprecated()
)

## S3 method for class 'grouped_eventlog'
filter_trim_lifecycle(
  log,
  start_lifecycles = NULL,
  end_lifecycles = NULL,
  reverse = FALSE,
  start_lifecycle = deprecated(),
  end_lifecycle = deprecated(),
  eventlog = deprecated()
)

### Arguments

- **log** `eventlog`: Object of class `eventlog` or derivatives (`grouped_eventlog`).
- **start_lifecycles, end_lifecycles** `character` vector (default `NULL`): A vector of life cycle identifiers, or `NULL`.
- **reverse** `logical` (default `FALSE`): Indicating whether the selection should be reversed.
- **start_lifecycle** [Deprecated]: please use `start_lifecycles` instead.
- **end_lifecycle** [Deprecated]: please use `end_lifecycles` instead.
- **eventlog** [Deprecated]: please use `log` instead.

### Value

When given an object of type `log`, it will return a filtered `log`. When given an object of type `grouped_log`, the filter will be applied in a stratified way (i.e. each separately for each group). The returned log will be grouped on the same variables as the original log.

### Methods (by class)

- `filter_trim_lifecycle(eventlog)`: Filters activity instances for an `eventlog`.
- `filter_trim_lifecycle(grouped_eventlog)`: Filters activity instances for a `grouped_eventlog`.

### References

See Also

lifecycle_id

Other filters: filter_activity_frequency(), filter_activity_instance(), filter_activity_presence(), filter_activity(), filter_case_condition(), filter_case(), filter_endpoints_condition(), filter_endpoints(), filter_flow_time(), filter_idle_time(), filter_infrequent_flows(), filter.lifecycle_presence(), filter.lifecycle(), filter.precedence.condition(), filter.precedence.resource(), filter.precedence(), filter.processing_time(), filter.resource.frequency(), filter.resource(), filter_throughput_time(), filter.time.period(), filter.trace_frequency(), filter.trace_length(), filter.trace(), filter.trim()

idle_time

Description

Calculates the amount of time that no activity occurs.

Usage

idle_time(
  log,
  level = c("log", "trace", "case", "resource"),
  append = deprecated(),
  append_column = NULL,
  units = c("auto", "secs", "mins", "hours", "days", "weeks"),
  sort = TRUE,
  eventlog = deprecated()
)

## S3 method for class 'eventlog'
idle_time(
  log,
  level = c("log", "trace", "case", "resource"),
  append = deprecated(),
  append_column = NULL,
  units = c("auto", "secs", "mins", "hours", "days", "weeks"),
  sort = TRUE,
  eventlog = deprecated()
)

## S3 method for class 'grouped_eventlog'
idle_time(
  log,
  level = c("log", "case", "trace", "resource"),
  append = deprecated(),
  append_column = NULL,
Arguments

log: Object of class log or derivatives (grouped_log, eventlog, activitylog, etc.).

level: character (default "log"): Level of granularity for the analysis: "log" (default), "trace", "case", or "resource". For more information, see vignette("metrics", "edeaR") and Details below.

append: logical (default FALSE) [Deprecated]: The arguments append and append_column have been deprecated in favour of augment. Indicating whether to append results to original log. Ignored when level is "log" or "trace".

append_column [Deprecated] The arguments append and append_column have been deprecated in favour of augment. Which of the output columns to append to log, if append = TRUE. Default column depends on chosen level.

units: character (default "auto"): The time unit in which the throughput times should be reported. Should be one of the following values: "auto" (default), "secs", "mins", "hours", "days", "weeks". See also the units argument of difftime().

sort: logical (default TRUE): Sort by decreasing idle time. Only relevant for "trace" and "resource" level.

eventlog [Deprecated]: please use log instead.
Details

Argument level has the following options:

- At "log" level, the idle time metric provides an overview of summary statistics of the idle time per case, aggregated over the complete log.
- On "trace" level, the idle time metric provides an overview of the summary statistics of the idle time for each trace in the log.
- On "case" level, the idle time metric provides an overview of the total idle time per case.
- On "resource" level, this metric can be used to get an insight in the amount of time each resource "wastes" during the process.

Methods (by class)

- `idle_time(eventlog)`: Computes the idle time for an `eventlog`.
- `idle_time(grouped_eventlog)`: Computes the idle time for a `grouped_eventlog`.
- `idle_time(activitylog)`: Computes the idle time for an `activitylog`.
- `idle_time(grouped_activitylog)`: Computes the idle time for a `grouped_activitylog`.

References


See Also

`throughput_time()`, `processing_time()`, `difftime()`

Other metrics: `activity_frequency()`, `activity_presence()`, `end_activities()`, `number_of_repetitions()`, `number_of_selfloops()`, `number_of_traces()`, `processing_time()`, `resource_frequency()`, `resource_involvement()`, `resource_specialisation()`, `start_activities()`, `throughput_time()`, `trace_coverage()`, `trace_length()`

---

**number_of_repetitions**  
*Number of Repetitions*

Description

Provides information statistics on the number of repetitions.

A repetition is an execution of an activity within a case while that activity has already been executed before, but one or more other activities are executed in between.
Usage

number_of_repetitions(
  log,
  type = c("all", "repeat", "redo"),
  level = c("log", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)

## S3 method for class 'eventlog'
number_of_repetitions(
  log,
  type = c("all", "repeat", "redo"),
  level = c("log", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)

## S3 method for class 'grouped_eventlog'
number_of_repetitions(
  log,
  type = c("all", "repeat", "redo"),
  level = c("log", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)

## S3 method for class 'activitylog'
number_of_repetitions(
  log,
  type = c("all", "repeat", "redo"),
  level = c("log", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)

## S3 method for class 'grouped_activitylog'
number_of_repetitions(
  log,
  type = c("all", "repeat", "redo"),
Arguments

log: Object of class log or derivatives (grouped_log, eventlog, activitylog, etc.).

type: character (default "all"): The type of repetitions: "all" (default), "repeat", or "redo". For more information, see 'Details' below.

level: character (default "log"): Level of granularity for the analysis: "log" (default), "case", "activity", "resource", or "resource-activity". For more information, see vignette("metrics", "edeaR") and 'Details' below.

append: logical (default FALSE) [Deprecated]: The arguments append and append_column have been deprecated in favour of augment. Indicating whether to append results to original log. Ignored when level is "log" or "trace".

append_column: [Deprecated] The arguments append and append_column have been deprecated in favour of augment. Which of the output columns to append to log, if append = TRUE. Default column depends on chosen level.

sort: logical (default TRUE): Sort output on count. Only for levels with frequency count output.

eventlog: [Deprecated]: please use log instead.

Details

Argument level has the following options:

• At "log" level, this metric shows the summary statistics of the number of repetitions within a case, which can provide insights in the amount of waste in a log. Each combination of two or more occurrences of the same activity, executed not immediately following each other, by the same resource is counted as one repeat repetition of this activity.

• On "case" level, this metric provides the absolute and relative number of repetitions in each case.

• On "activity" level, this metric shows which activities occur the most in a repetition. The absolute and relative number of both repeat and redo repetitions is provided by this metric, giving an overview per activity.

• On "resource" level, it can be interesting to have an overview of which resources need more than one time to execute an activity in a case or which resources need to have an activity redone later on in the case by another resource. This metric provides the absolute and relative number of times each resource appears in a repetition.
• On "resource-activity" level, this metric provides specific information about which activities and which resources are involved in the repetitions. For this metric the absolute and relative number of repeat and redo repetitions is provided. Again, two different relative numbers are provided, one relative to the total number of executions of the activity in the complete log, and one relative to the total number of executions performed by the resource throughout the complete log.

Similar to the self-loop metric, a distinction should be made between "repeat" and "redo" repetitions, as can be set by the type argument:

• "repeat" repetitions are activity executions of the same activity type that are executed not immediately following each other, but by the same resource.
• "redo" repetitions are activity executions of the same activity type that are executed not immediately following each other and by a different resource than the first activity occurrence of this activity type.

Methods (by class)

• number_of_repetitions(eventlog): Computes the number of repetitions for an eventlog.
• number_of_repetitions(grouped_eventlog): Computes the number of repetitions for a grouped_eventlog.
• number_of_repetitions(activitylog): Computes the number of repetitions for an activitylog.
• number_of_repetitions(grouped_activitylog): Computes the number of repetitions for a grouped_activitylog.

References


See Also

number_of_selfloops

Other metrics: activity_frequency(), activity_presence(), end_activities(), idle_time(), number_of_selfloops(), number_of_traces(), processing_time(), resource_frequency(), resource_involvement(), resource_specialisation(), start_activities(), throughput_time(), trace_coverage(), trace_length()
Usage

number_of_selfloops(
  log,
  type = c("all", "repeat", "redo"),
  level = c("log", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)

## S3 method for class 'eventlog'
number_of_selfloops(
  log,
  type = c("all", "repeat", "redo"),
  level = c("log", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)

## S3 method for class 'grouped_eventlog'
number_of_selfloops(
  log,
  type = c("all", "repeat", "redo"),
  level = c("log", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)

## S3 method for class 'activitylog'
number_of_selfloops(
  log,
  type = c("all", "repeat", "redo"),
  level = c("log", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)

## S3 method for class 'grouped_activitylog'
number_of_selfloops(
  log,
  type = c("all", "repeat", "redo"),
  level = c("log", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
level = c("log", "case", "activity", "resource", "resource-activity"),
append = deprecated(),
append_column = NULL,
sort = TRUE,
eventlog = deprecated()
)

Arguments

log log: Object of class log or derivatives (grouped_log, eventlog, activitylog, etc.).
type character (default "all"): The type of repetitions: "all" (default), "repeat", or "redo". For more information, see 'Details' below.
level character (default "log"): Level of granularity for the analysis: "log" (default), "case", "activity", "resource", or "resource-activity". For more information, see vignette("metrics", "edeaR") and 'Details' below.
append logical (default FALSE) [Deprecated]: The arguments append and append_column have been deprecated in favour of augment. Indicating whether to append results to original log. Ignored when level is "log" or "trace".
append_column [Deprecated] The arguments append and append_column have been deprecated in favour of augment. Which of the output columns to append to log, if append = TRUE. Default column depends on chosen level.
sort logical (default TRUE): Sort output on count. Only for levels with frequency count output.
eventlog [Deprecated]: please use log instead.

Details

Two types of self-loops are defined, which can be chosen using the type argument:

- "repeat" self-loops are activity executions of the same activity type that are executed immediately following each other by the same resource.
- "redo" self-loops are activity executions of the same activity type that are executed immediately following each other by a different resource.

Argument level has the following options:

- At "log" level, the summary statistics of the number of self-loops within a trace can give a first insight in the amount of waste in a log. As stated earlier, each combination of two occurrences of the same activity executed by the same resource will be counted as one repeat self-loop of this activity.
- On "case" level, an overview is provided of the absolute and relative number of repeat and redo self-loops in each case. To calculate the relative number, each (repeat or redo) self-loop is counted as 1 occurrence, and the other activity instances are also counted as 1.
- On "activity" level, the absolute and relative number of self-loops per activity can be an indication for which activities are causing the most waste in the process.
• On "resource" level, this metric can give insights into which resources needs to repeat their work most often within a case, or for which resource the work they did should be redone by another resource within the same case. This metric shows the absolute and relative number of both repeat and redo self-loops for each resource in the log.

• On "resource-activity" level, this metric can be used to get an insight in which activities are the most crucial for which resources. This metric shows the absolute and relative number of both repeat and redo self-loops for each of the resource-activity combinations that occur in the log. Two different relative numbers are provided here, one from the resource perspective and one from the activity perspective. At the resource perspective, the denominator is the total number of executions by the resource under consideration. At the activity perspective, the denominator is the total number of occurrences of the activity under consideration.

Methods (by class)

- `number_of_selfloops(eventlog)`: Computes the number of self-loops for an `eventlog`.
- `number_of_selfloops(grouped_eventlog)`: Computes the number of self-loops for a `grouped_eventlog`.
- `number_of_selfloops(activitylog)`: Computes the number of self-loops for an `activitylog`.
- `number_of_selfloops(grouped_activitylog)`: Computes the number of self-loops for a `grouped_activitylog`.

References


See Also

- `number_of_repetitions`

Other metrics: `activity_frequency()`, `activity_presence()`, `end_activities()`, `idle_time()`, `number_of_repetitions()`, `number_of_traces()`, `processing_time()`, `resource_frequency()`, `resource_involvement()`, `resource_specialisation()`, `start_activities()`, `throughput_time()`, `trace_coverage()`, `trace_length()`

**number_of_traces**

<table>
<thead>
<tr>
<th>number_of_traces</th>
<th>Number of Traces</th>
</tr>
</thead>
<tbody>
<tr>
<td>Computes how many traces there are.</td>
<td></td>
</tr>
</tbody>
</table>

This metric provides two values, the absolute and relative number of traces that occur in the log. The relative number shows expected number of traces needed to cover 100 cases.
Usage

number_of_traces(log, eventlog = deprecated())

## S3 method for class 'log'
number_of_traces(log, eventlog = deprecated())

## S3 method for class 'grouped_log'
number_of_traces(log, eventlog = deprecated())

Arguments

log log: Object of class log or derivatives (grouped_log, eventlog, activitylog, etc.).

eventlog [Deprecated]; please use log instead.

Methods (by class)

- number_of_traces(log): Number of traces in a log.
- number_of_traces(grouped_log): Number of traces in a grouped_log.

References


See Also

traces

Other metrics: activity_frequency(), activity_presence(), end_activities(), idle_time(), number_of_repetitions(), number_of_selfloops(), processing_time(), resource_frequency(), resource_involvement(), resource_specialisation(), start_activities(), throughput_time(), trace_coverage(), trace_length()
Usage

```r
## S3 method for class 'activity_frequency'
plot(x, ...)

## S3 method for class 'activity_presence'
plot(x, ...)

## S3 method for class 'end_activities'
plot(x, ...)

## S3 method for class 'idle_time'
plot(x, ...)

## S3 method for class 'processing_time'
plot(x, ...)

## S3 method for class 'referral_matrix'
plot(x, ...)

## S3 method for class 'resource_frequency'
plot(x, ...)

## S3 method for class 'resource_involvement'
plot(x, ...)

## S3 method for class 'resource_specialisation'
plot(x, ...)

## S3 method for class 'start_activities'
plot(x, ...)

## S3 method for class 'throughput_time'
plot(x, ...)

## S3 method for class 'trace_coverage'
plot(x, ...)

## S3 method for class 'trace_length'
plot(x, ...)

## S3 method for class 'number_of_selfloops'
plot(x, ...)

## S3 method for class 'number_of_repetitions'
plot(x, ...)
```
Arguments

x Data to plot
...

Additional variables

Value

A ggplot object, which can be customized further, if deemed necessary.

Description

Print work schedule

Usage

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

Arguments

x Work schedule to print
...

Additional arguments (ignored)

Description

Provides summary statistics about the processing time of the process.

In contrast to the `throughput_time()` of the cases in a log, the metrics concerning the active time or the actual processing time provide summary statistics on the processing time of events on the level of the complete log, the specific cases, traces, the activities, and the resource-activity combinations.
usage

processing_time(
  log,
  level = c("log", "trace", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  units = c("auto", "secs", "mins", "hours", "days", "weeks"),
  sort = TRUE,
  work_schedule = NULL,
  eventlog = deprecated()
)

## S3 method for class 'eventlog'
processing_time(
  log,
  level = c("log", "trace", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  units = c("auto", "secs", "mins", "hours", "days", "weeks"),
  sort = TRUE,
  work_schedule = NULL,
  eventlog = deprecated()
)

## S3 method for class 'grouped_eventlog'
processing_time(
  log,
  level = c("log", "trace", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  units = c("auto", "secs", "mins", "hours", "days", "weeks"),
  sort = TRUE,
  work_schedule = NULL,
  eventlog = deprecated()
)

## S3 method for class 'activitylog'
processing_time(
  log,
  level = c("log", "trace", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  units = c("auto", "secs", "mins", "hours", "days", "weeks"),
  sort = TRUE,
  work_schedule = NULL,
  eventlog = deprecated()
## S3 method for class 'grouped_activitylog'

```r
processing_time(
  log,
  level = c("log", "trace", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  units = c("auto", "secs", "mins", "hours", "days", "weeks"),
  sort = TRUE,
  work_schedule = NULL,
  eventlog = deprecated()
)
```

### Arguments

- **log**: Object of class `log` or derivatives (`grouped_log`, `eventlog`, `activitylog`, etc.).
- **level**: `character` (default "log"): Level of granularity for the analysis: "log" (default), "trace", "case", "activity", "resource", or "resource-activity". For more information, see vignette("metrics", "edeaR") and Details below.
- **append**: `logical` (default FALSE) [Deprecated]: The arguments `append` and `append_column` have been deprecated in favour of `augment`. Indicating whether to append results to original log. Ignored when level is "log" or "trace".
- **append_column**: [Deprecated] The arguments `append` and `append_column` have been deprecated in favour of `augment`. Which of the output columns to append to log, if `append = TRUE`. Default column depends on chosen level.
- **units**: `character` (default "auto"): The time unit in which the processing times should be reported. Should be one of the following values: "auto" (default), "secs", "mins", "hours", "days", "weeks". See also the `units` argument of `difftime()`.
- **sort**: `logical` (default TRUE): Sort on decreasing processing time. For "case" level only.
- **work_schedule**: A schedule of working hours. If provided, only working hours are counted as processing time.
- **eventlog**: [Deprecated]; please use `log` instead.

### Details

Argument `level` has the following options:

- At "log" level, this metric calculates the summary statistics of the actual processing time per case, summarised over the complete event log.
- On "trace" level, the summary statistics of processing time can be calculated for each possible sequence of activities that appears in the event log.
- On "case" level, a list of cases with their processing time are provided.
- On "activity" level, an overview of the average processing time -or the service time- of each activity can be calculated.
• At "resource" level, this metric calculates the processing time per resource.
• On "resource-activity" level, the efficiency of resources by looking at the combination of each resource with each activity can be investigated.

Methods (by class)
• processing_time(eventlog): Computes processing time for an eventlog.
• processing_time(grouped_eventlog): Computes processing time for a grouped_eventlog.
• processing_time(activitylog): Computes processing time for an activitylog.
• processing_time(grouped_activitylog): Computes processing time for a grouped_activitylog.

References

See Also
idle_time(), throughput_time(), difftime()
Other metrics: activity_frequency(), activity_presence(), end_activities(), idle_time(), number_of_repetitions(), number_of_selfloops(), number_of_traces(), resource_frequency(), resource_involvement(), resource_specialisation(), start_activities(), throughput_time(), trace_coverage(), trace_length()

redo_repetitions_referral_matrix
Referral matrix repetitions

Description
Provides a list of initiators and completers of redo repetitions

Usage
redo_repetitions_referral_matrix(log, eventlog = deprecated())

## S3 method for class 'eventlog'
redo_repetitions_referral_matrix(log, eventlog = deprecated())

## S3 method for class 'activitylog'
redo_repetitions_referral_matrix(log, eventlog = deprecated())

Arguments
log log: Object of class log or derivatives (grouped_log, eventlog, activitylog, etc.).
eventlog [Deprecated]: please use log instead.
**redo_selfloops_referral_matrix**

**Methods (by class)**

- redo_repetitions_referral_matrix(eventlog): Compute matrix for eventlog
- redo_repetitions_referral_matrix(activitylog): Compute matrix for activitylog

**References**


**See Also**

number_of_repetitions

---

**redo_selfloops_referral_matrix**

*Referral matrix selfloops*

**Description**

Provides a list of initiators and completers of redo selfloops

**Usage**

```r
redo_selfloops_referral_matrix(log, eventlog = deprecated())
```

```r
# S3 method for class 'eventlog'
redo_selfloops_referral_matrix(log, eventlog = deprecated())
```

```r
# S3 method for class 'activitylog'
redo_selfloops_referral_matrix(log, eventlog = deprecated())
```

**Arguments**

- `log`: Object of class log or derivatives (grouped_log, eventlog, activitylog, etc.).
- `eventlog` [Deprecated]: please use log instead.

**Methods (by class)**

- redo_selfloops_referral_matrix(eventlog): Compute matrix for eventlog
- redo_selfloops_referral_matrix(activitylog): Compute matrix for activitylog

**References**

See Also

number_of_selfloops

---

resource_frequency  Resource Frequency

Description

Analyses the frequency of resources at different levels of analysis.

Usage

```r
resource_frequency(
  log,
  level = c("log", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)

## S3 method for class 'eventlog'
resource_frequency(
  log,
  level = c("log", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)

## S3 method for class 'grouped_eventlog'
resource_frequency(
  log,
  level = c("log", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)

## S3 method for class 'activitylog'
resource_frequency(
  log,
  level = c("log", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
```
resource_frequency

```r
append_column = NULL,
    sort = TRUE,
    eventlog = deprecated()
)
```

```r
## S3 method for class 'grouped_activitylog'
resource_frequency(
    log,
    level = c("log", "case", "activity", "resource", "resource-activity"),
    append = deprecated(),
    append_column = NULL,
    sort = TRUE,
    eventlog = deprecated()
)
```

**Arguments**

- **log**
  - Object of class `log` or derivatives (`grouped_log`, `eventlog`, `activitylog`, etc.).

- **level**
  - character (default "log"): Level of granularity for the analysis: "log" (default), "case", "activity", "resource", or "resource-activity". For more information, see vignette("metrics", "edeaR") and 'Details' below.

- **append**
  - logical (default FALSE) [Deprecated]: The arguments `append` and `append_column` have been deprecated in favour of `augment`. Indicating whether to append results to original log. Ignored when level is "log" or "trace".

- **append_column**
  - [Deprecated] The arguments `append` and `append_column` have been deprecated in favour of `augment`. Which of the output columns to append to log, if `append = TRUE`. Default column depends on chosen level.

- **sort**
  - logical (default TRUE): Sort output on count. Only for levels with frequency count output.

- **eventlog**
  - [Deprecated]: please use log instead.

**Details**

Argument `level` has the following options:

- At "log" level, summary statistics show the number of times a resource executes an activity in the complete log.

- On "case" level, summary statistics of the frequency of resources can be used to get a better view on the variance between the different cases, to get an insight into the number of different resources working on each case together with the number of activities a resource executes per case.

- On "activity" level, the resource frequency states how many different resources are executing a specific activity in the complete log.
• On "resource" level, this metric simply shows the absolute and relative frequency of occurrences of each resource in the complete log.

• On "resource-activity" level, the absolute and relative number of times each resource-activity combination occurs in the complete log can be calculated. Two different relative numbers are provided here, one from the resource perspective and one from the activity perspective. At the resource perspective, the denominator is the total number of executions by the resource under consideration. At the activity perspective, the denominator is the total number of occurrences of the activity under consideration.

Methods (by class)

• resource_frequency(eventlog): Computes the resource frequency for an eventlog.
• resource_frequency(grouped_eventlog): Computes the resource frequency for a grouped_eventlog.
• resource_frequency(activitylog): Computes the resource frequency for an activitylog.
• resource_frequency(grouped_activitylog): Computes the resource frequency for a grouped_activitylog.

References


See Also

resource_involvement

Other metrics: activity_frequency(), activity_presence(), end_activities(), idle_time(), number_of_repetitions(), number_of_selfloops(), number_of_traces(), processing_time(), resource_involvement(), resource_specialisation(), start_activities(), throughput_time(), trace_coverage(), trace_length()
resource_involvement

Usage

resource_involvement(
  log,
  level = c("case", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)

## S3 method for class 'log'
resource_involvement(
  log,
  level = c("case", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)

## S3 method for class 'grouped_log'
resource_involvement(
  log,
  level = c("case", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)

Arguments

log `log`: Object of class `log` or derivatives (`grouped_log`, `eventlog`, `activitylog`, etc.).

decl world level `character`: Level of granularity for the analysis: "case" (default), "resource", or "resource-activity". For more information, see vignette("metrics", "edeaR") and 'Details' below.

decl world append `logical` (default FALSE)[Deprecated]: The arguments `append` and `append_column` have been deprecated in favour of `augment`. Indicating whether to append results to original log. Ignored when level is "log" or "trace".

append_column [Deprecated] The arguments `append` and `append_column` have been deprecated in favour of `augment`. Which of the output columns to append to log, if `append = TRUE`. Default column depends on chosen level.

decl world sort `logical` (default TRUE): Sort output on count. Only for levels with frequency count output.
Resource Specialisation

Description

Analyses whether resources specialise in specific activities.

This metric can give an overview of which resources are performing certain activities more than others, and which resources are responsible for containing all knowledge or capabilities on one topic.
resource_specialisation

Usage

resource_specialisation(
  log,
  level = c("log", "activity", "resource"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)

resource_specialization(
  log,
  level = c("log", "activity", "resource"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)

## S3 method for class 'log'
resource_specialisation(
  log,
  level = c("log", "activity", "resource"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)

## S3 method for class 'grouped_log'
resource_specialisation(
  log,
  level = c("log", "activity", "resource"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)

Arguments

log log: Object of class log or derivatives (grouped_log, eventlog, activitylog, etc.).

level character (default "log"): Level of granularity for the analysis: "log" (de-
dfault), "activity", or "resource". For more information, see vignette("metrics", "edear") and 'Details' below.

append logical (default FALSE) [Deprecated]: The arguments append and append_column
have been deprecated in favour of augment.
resource_specialisation

Indicating whether to append results to original log. Ignored when level is "log" or "trace".

append_column
[Deprecated] The arguments append and append_column have been deprecated in favour of augment.
Which of the output columns to append to log, if append = TRUE. Default column depends on chosen level.

sort logical (default TRUE): Sort output on count. Only for levels with frequency count output.

eventlog
[Deprecated]: please use log instead.

Details

Argument level has the following options:

- At "log" level, this metric provides summary statistics on the number of distinct activities executed per resource.
- On "activity" level, this metric provides an overview of the absolute and relative number of different resources executing this activity within the complete log. This will give insights into which activities resources are specialised in.
- On "resource" level, this metric shows the absolute and relative number of distinct activities that each resource executes.

Methods (by class)

- resource_specialisation(log): Computes the resource specialisation for a log.
- resource_specialisation(grouped_log): Computes the resource specialisation for a grouped_log.

References


See Also

Other metrics: activity_frequency(), activity_presence(), end_activities(), idle_time(), number_of_repetitions(), number_of_selfloops(), number_of_traces(), processing_time(), resource_frequency(), resource_involvement(), start_activities(), throughput_time(), trace_coverage(), trace_length()
Description

Provides summary statistics on the sizes of repetitions.

Usage

```r
size_of_repetitions(
  log,
  type = c("all", "repeat", "redo"),
  level = c("log", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  eventlog = deprecated()
)
```

```r
## S3 method for class 'eventlog'
size_of_repetitions(
  log,
  type = c("all", "repeat", "redo"),
  level = c("log", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  eventlog = deprecated()
)
```

```r
## S3 method for class 'grouped_eventlog'
size_of_repetitions(
  log,
  type = c("repeat", "redo"),
  level = c("log", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  eventlog = deprecated()
)
```

```r
## S3 method for class 'activitylog'
size_of_repetitions(
  log,
  type = c("repeat", "redo"),
  level = c("log", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  eventlog = deprecated()
)
```
## S3 method for class 'grouped_activitylog'

```r
size_of_repetitions(
  log,
  type = c("repeat", "redo"),
  level = c("log", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  eventlog = deprecated()
)
```

### Arguments

- **log**: Object of class `log` or derivatives (`grouped_log`, `eventlog`, `activitylog`, etc.).
- **type**: character (default "all"): The type of repetitions: "all" (default), "repeat", or "redo". For more information, see 'Details' below.
- **level**: character (default "log"): Level of granularity for the analysis: "log" (default), "case", "activity", "resource", or "resource-activity". For more information, see vignette("metrics", "edeaR") and 'Details' below.
- **append**: logical (default FALSE) [Deprecated]: The arguments `append` and `append_column` have been deprecated in favour of `augment`. Indicating whether to append results to original log. Ignored when level is "log" or "trace".
- **append_column** [Deprecated]: The arguments `append` and `append_column` have been deprecated in favour of `augment`. Which of the output columns to append to log, if `append = TRUE`. Default column depends on chosen level.
- **eventlog** [Deprecated]: please use `log` instead.

### Methods (by class)

- `size_of_repetitions(eventlog)`: Size of repetitions for eventlog
- `size_of_repetitions(grouped_eventlog)`: Size of repetitions for grouped event log
- `size_of_repetitions(activitylog)`: Size of repetitions for activitylog
- `size_of_repetitions(grouped_activitylog)`: Size of repetitions for grouped activitylog

### References


### See Also

`number_of_repetitions`
size_of_selfloops  Metric: Size of selfloops

Description

Provides summary statistics on the sizes of selfloops

Usage

size_of_selfloops(
  log,
  type = c("all", "repeat", "redo"),
  level = c("log", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  eventlog = deprecated()
)

## S3 method for class 'eventlog'
size_of_selfloops(
  log,
  type = c("all", "repeat", "redo"),
  level = c("log", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  eventlog = deprecated()
)

## S3 method for class 'grouped_eventlog'
size_of_selfloops(
  log,
  type = c("repeat", "redo"),
  level = c("log", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  eventlog = deprecated()
)

## S3 method for class 'activitylog'
size_of_selfloops(
  log,
  type = c("all", "repeat", "redo"),
  level = c("log", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  eventlog = deprecated()
)
## S3 method for class 'grouped_activitylog'

`size_of_selfloops`

```r
size_of_selfloops(
  log,
  type = c("all", "repeat", "redo"),
  level = c("log", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  eventlog = deprecated()
)
```

### Arguments

- **log**: Object of class `log` or derivatives (`grouped_log`, `eventlog`, `activitylog`, etc.).
- **type**: `character` (default "all"): The type of repetitions: "all" (default), "repeat", or "redo". For more information, see 'Details' below.
- **level**: `character` (default "log"): Level of granularity for the analysis: "log" (default), "case", "activity", "resource", or "resource-activity". For more information, see vignette("metrics", "edeaR") and 'Details' below.
- **append**: `logical` (default FALSE) [Deprecated]: The arguments `append` and `append_column` have been deprecated in favour of `augment`. Indicating whether to append results to original log. Ignored when level is "log" or "trace".
- **append_column**: [Deprecated] The arguments `append` and `append_column` have been deprecated in favour of `augment`. Which of the output columns to append to `log`, if `append = TRUE`. Default column depends on chosen level.
- **eventlog**: [Deprecated]: please use `log` instead.

### Methods (by class)

- `size_of_selfloops(eventlog)`: Size of selfloops for `eventlog`
- `size_of_selfloops(grouped_eventlog)`: Size of selfloops for grouped `eventlog`
- `size_of_selfloops(activitylog)`: Size of selfloops for `activitylog`
- `size_of_selfloops(grouped_activitylog)`: Size of selfloops for grouped `activitylog`

### References


### See Also

`number_of_selfloops`
Description

Analyse the start activities in the process.

Usage

```r
start_activities(
  log,
  level = c("log", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)
```

```r
## S3 method for class 'eventlog'
start_activities(
  log,
  level = c("log", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)
```

```r
## S3 method for class 'grouped_eventlog'
start_activities(
  log,
  level = c("log", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)
```

```r
## S3 method for class 'activitylog'
start_activities(
  log,
  level = c("log", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)
```
## S3 method for class 'grouped_activitylog'

```r
start_activities(
  log,
  level = c("log", "case", "activity", "resource", "resource-activity"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)
```

### Arguments

- **log**: Object of class `log` or derivatives (`grouped_log`, `eventlog`, `activitylog`, etc.).
- **level**: character (default "log"): Level of granularity for the analysis: "log" (default), "case", "activity", "resource", or "resource-activity". For more information, see vignette("metrics", "edeaR") and 'Details' below.
- **append**: logical (default FALSE) [Deprecated]: The arguments append and append_column have been deprecated in favour of `augment`. Indicating whether to append results to original log. Ignored when level is "log" or "trace".
- **append_column**: [Deprecated] The arguments append and append_column have been deprecated in favour of `augment`. Which of the output columns to append to log, if append = TRUE. Default column depends on chosen level.
- **sort**: logical (default TRUE): Sort output on count. Only for levels with frequency count output.
- **eventlog**: [Deprecated]: please use log instead.

### Details

Argument level has the following options:

- On "log" level, this metric shows the absolute and relative number of activities that are the first activity in one or more of the cases.
- On "case" level, this metric provides an overview of the start activity of each case.
- On "activity" level, this metric calculates for each activity the absolute and relative number of cases that start with this activity type. Similar to the `end_activities` metric, the relative number is calculated as a portion of the number of cases, being the number of "opportunities" that an activity could be the start activity. The cumulative sum is added to have an insight in the number of activities that is required to cover a certain part of the total.
- On "resource" level, an overview of which resources execute the first activity per case are provided.
- On "resource-activity" level, this metric shows for each occurring resource-activity combination the absolute and relative number of times this resource executes this activity as an start activity in a case.
Methods (by class)

• `start_activities(eventlog)`: Computes the start activities for an `eventlog`.
• `start_activities(grouped_eventlog)`: Computes the start activities for a `grouped_eventlog`.
• `start_activities(activitylog)`: Computes the start activities for an `activitylog`.
• `start_activities(grouped_activitylog)`: Computes the start activities for a `grouped_activitylog`.

References


See Also

`end_activities`

Other metrics: `activity_frequency()`, `activity_presence()`, `end_activities()`, `idle_time()`, `number_of_repetitions()`, `number_of_selfloops()`, `number_of_traces()`, `processing_time()`, `resource_frequency()`, `resource_involvement()`, `resource_specialisation()`, `throughput_time()`, `trace_coverage()`, `trace_length()`

---

**throughput_time**  
*Throughput Time of Cases*

**Description**

Provides summary statistics concerning the throughput times of cases.

**Usage**

```r
throughput_time(
  log,
  level = c("log", "trace", "case"),
  append = deprecated(),
  append_column = NULL,
  units = c("auto", "secs", "mins", "hours", "days", "weeks"),
  sort = TRUE,
  work_schedule = NULL,
  eventlog = deprecated()
)
```

```r
## S3 method for class 'eventlog'
throughput_time(
  log,
  level = c("log", "trace", "case"),
  append = deprecated(),
  append_column = NULL,
  units = c("auto", "secs", "mins", "hours", "days", "weeks"),
)
```

## S3 method for class 'grouped_eventlog'

```r
throughput_time(
  log,
  level = c("log", "trace", "case"),
  append = deprecated(),
  append_column = NULL,
  units = c("auto", "secs", "mins", "hours", "days", "weeks"),
  sort = TRUE,
  work_schedule = NULL,
  eventlog = deprecated()
)
```

## S3 method for class 'activitylog'

```r
throughput_time(
  log,
  level = c("log", "trace", "case"),
  append = deprecated(),
  append_column = NULL,
  units = c("auto", "secs", "mins", "hours", "days", "weeks"),
  sort = TRUE,
  work_schedule = NULL,
  eventlog = deprecated()
)
```

## S3 method for class 'grouped_activitylog'

```r
throughput_time(
  log,
  level = c("log", "trace", "case"),
  append = deprecated(),
  append_column = NULL,
  units = c("auto", "secs", "mins", "hours", "days", "weeks"),
  sort = TRUE,
  work_schedule = NULL,
  eventlog = deprecated()
)
```

### Arguments

- **log**: Object of class `log` or derivatives (`grouped_log, eventlog, activitylog, etc.`).
- **level**: character (default "log"): Level of granularity for the analysis: "log" (default), "trace", or "case". For more information, see `vignette("metrics", "edeaR")` and Details below.
throughput_time

append logical (default FALSE) [Deprecated]: The arguments append and append_column have been deprecated in favour of augment. Indicating whether to append results to original log. Ignored when level is "log" or "trace".

append_column [Deprecated]: The arguments append and append_column have been deprecated in favour of augment. Which of the output columns to append to log, if append = TRUE. Default column depends on chosen level.

units character (default "auto"): The time unit in which the throughput times should be reported. Should be one of the following values: "auto" (default), "secs", "mins", "hours", "days", "weeks". See also the units argument of difftime.

sort logical (default TRUE): Sort output on count. Only for levels with frequency count output.

work_schedule A schedule of working hours. If provided, only working hours are counted as processing time.

eventlog [Deprecated]: please use log instead.

Details

Argument level has the following options:

- At "log" level, the summary statistics describing the throughput time of cases in an aggregated fashion.
- On "trace" level, the throughput time of the different process variants or traces in the log are calculated.
- On "case" level, the throughput time is defined as the total duration of the case, or the difference between the timestamp of the end event and the timestamp of the start event of the case. Possible idle_time() is also included in this calculation.

For other levels (e.g. "activity", "resource", or "resource-activity"), the throughput time is equal to the processing_time() and are, therefore, not supported by this method.

Methods (by class)

- throughput_time(eventlog): Computes throughput time for an eventlog.
- throughput_time(grouped_eventlog): Computes throughput time for a grouped_eventlog.
- throughput_time(activitylog): Computes throughput time for an activitylog.
- throughput_time(grouped_activitylog): Computes throughput time for a grouped_activitylog.

References

See Also

idle_time(), processing_time(), difftime()

Other metrics: activity_frequency(), activity_presence(), end_activities(), idle_time(), number_of_repetitions(), number_of_selfloops(), number_of_traces(), processing_time(), resource_frequency(), resource_involvement(), resource_specialisation(), start_activities(), trace_coverage(), trace_length()

---

trace_coverage Trace Coverage

Description

Analyses the structuredness of a log by use of trace frequencies.

Usage

trace_coverage(
  log,
  level = c("log", "trace", "case"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)

## S3 method for class 'log'
trace_coverage(
  log,
  level = c("log", "trace", "case"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)

## S3 method for class 'grouped_log'
trace_coverage(
  log,
  level = c("log", "trace", "case"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)
**Arguments**

- **log**: Object of class `log` or derivatives (`grouped_log`, `eventlog`, `activitylog`, etc.).
- **level**: `character` (default "log"): Level of granularity for the analysis: "log" (default), "trace", or "case". For more information, see vignette("metrics", "edeaR") and Details below.
- **append**: `logical` (default FALSE) [Deprecated]: The arguments append and append_column have been deprecated in favour of augment. Indicating whether to append results to original log. Ignored when level is "log" or "trace".
- **append_column** [Deprecated] The arguments append and append_column have been deprecated in favour of augment. Which of the output columns to append to log, if append = TRUE. Default column depends on chosen level.
- **sort**: `logical` (default TRUE): Sort output on count. Only for levels with frequency count output.
- **eventlog** [Deprecated]: please use log instead.

**Details**

Argument level has the following options:

- At "log" level, summary statistics of the coverage of traces are returned.
- On "trace" level, the absolute and relative frequency of each trace are returned.
- On "case" level, the coverage of the corresponding trace is returned for each case.

**Methods (by class)**

- `trace_coverage(log)`: Calculates trace coverage metric for a `log`.
- `trace_coverage(grouped_log)`: Calculates trace coverage metric for a `grouped_log`.

**References**


**See Also**

Other metrics: `activity_frequency()`, `activity_presence()`, `end_activities()`, `idle_time()`, `number_of_repetitions()`, `number_of_selfloops()`, `number_of_traces()`, `processing_time()`, `resource_frequency()`, `resource_involvement()`, `resource_specialisation()`, `start_activities()`, `throughput_time()`, `trace_length()`
Description

Analysis of trace lengths

This metric provides an overview of the number of activities that occur in each trace.

An important remark is that this metric takes into account each instance of an activity, but not the individual lifecycle events.

Usage

```r
trace_length(
  log,
  level = c("log", "trace", "case"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)
```

## S3 method for class 'eventlog'
trace_length(
  log,
  level = c("log", "trace", "case"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)

## S3 method for class 'grouped_eventlog'
trace_length(
  log,
  level = c("log", "trace", "case"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)

## S3 method for class 'activitylog'
trace_length(
  log,
  level = c("log", "trace", "case"),
  append = deprecated(),
)
trace_length

```
append_column = NULL,
sort = TRUE,
eventlog = deprecated()
)

## S3 method for class 'grouped_activitylog'
trace_length(
  log,
  level = c("log", "trace", "case"),
  append = deprecated(),
  append_column = NULL,
  sort = TRUE,
  eventlog = deprecated()
)
```

**Arguments**

- **log**: Object of class `log` or derivatives (`grouped_log`, `eventlog`, `activitylog`, etc.).
- **level**: character (default "log"): Level of granularity for the analysis: "log" (default), "trace", or "case". For more information, see vignette("metrics", "edeaR") and Details below.
- **append**: logical (default FALSE) [Deprecated]: The arguments `append` and `append_column` have been deprecated in favour of `augment`. Indicating whether to append results to original log. Ignored when level is "log" or "trace".
- **append_column** [Deprecated] The arguments `append` and `append_column` have been deprecated in favour of `augment`. Which of the output columns to append to log, if `append = TRUE`. Default column depends on chosen level.
- **sort**: logical (default TRUE): Sort output on count. Only for levels with frequency count output.
- **eventlog** [Deprecated]: please use `log` instead.

**Details**

Argument `level` has the following options:

- At "log" level, the summary statistics describing the trace length of cases in an aggregated fashion.
- On "trace" level, the trace length of the different process variants or traces in the log are calculated.
- On "case" level, the trace lengths for each case are computed.

**Methods (by class)**

- `trace_length(eventlog)`: Computes trace length for an `eventlog`. 

• `trace_length(grouped_eventlog)`: Computes trace length for a `grouped_eventlog`.
• `trace_length(activitylog)`: Computes trace length for an `activitylog`.
• `trace_length(grouped_activitylog)`: Computes trace length for a `grouped_activitylog`.

References

See Also
Other metrics: `activity_frequency()`, `activity_presence()`, `end_activities()`, `idle_time()`, `number_of_repetitions()`, `number_of_selfloops()`, `number_of_traces()`, `processing_time()`, `resource_frequency()`, `resource_involvement()`, `resource_specialisation()`, `start_activities()`, `throughput_time()`, `trace_coverage()`
Index

* filters_case
  filter_activity_presence, 21
  filter_case, 23
  filter_case_condition, 24
  filter_endpoints, 26
  filter_endpoints_condition, 27
  filter_flow_time, 29
  filter_idle_time, 31
  filter_infrequent_flows, 33
  filter_lifecycle, 34
  filter_lifecycle_presence, 35
  filter_precedence, 37
  filter_precedence_condition, 40
  filter_precedence_resource, 42
  filter_processing_time, 44
  filter_resource, 46
  filter_resource_frequency, 47
  filter_throughput_time, 49
  filter_time_period, 51
  filter_trace, 53
  filter_trace_frequency, 54
  filter_trace_length, 56
  filter_trim, 58
  filter_trim_lifecycle, 60
* filters_event
  filter_activity, 16
  filter_activity_frequency, 17
  filter_activity_instance, 19
  filter_lifecycle, 34
  filter_lifecycle_presence, 35
  filter_resource, 46
  filter_resource_frequency, 47
  filter_time_period, 51
  filter_trim, 58
  filter_trim_lifecycle, 60
* filters
  filter_activity, 16
  filter_activity_frequency, 17
  filter_activity_instance, 19
  filter_case, 23
  filter_case_condition, 24
  filter_endpoints, 26
  filter_endpoints_condition, 27
  filter_flow_time, 29
  filter_idle_time, 31
  filter_infrequent_flows, 33
  filter_lifecycle, 34
  filter_lifecycle_presence, 35
  filter_precedence, 37
  filter_precedence_condition, 40
  filter_precedence_resource, 42
  filter_processing_time, 44
  filter_resource, 46
  filter_resource_frequency, 47
  filter_throughput_time, 49
  filter_time_period, 51
  filter_trace, 53
  filter_trace_frequency, 54
  filter_trace_length, 56
  filter_trim, 58
  filter_trim_lifecycle, 60
* metrics_organizational
  resource_frequency, 78
  resource_involvement, 80
  resource_specialisation, 82
* metrics_other
  augment, 9
* metrics_repetition
  number_of_repetitions, 64
  number_of_selfloops, 67
  redo_repetitions_referral_matrix, 76
  redo_selfloops_referral_matrix, 77
  size_of_repetitions, 85
  size_of_selfloops, 87
* metrics_structuredness
  activity_frequency, 3
  activity_presence, 5
  end_activities, 14
  number_of_traces, 70
  start_activities, 89
  trace_coverage, 94
  trace_length, 96
* metrics_time
filter_endpoints_condition, 17, 19, 20, 22, 24, 25, 27, 29, 31, 32, 34, 35, 37, 39, 41, 44, 45, 47, 48, 50, 53, 54, 56, 58, 60, 62

filter_endpoints_conditions
(filter_endpoints_condition), 27

filter_flow_time, 17, 19, 20, 22, 24, 25, 27, 29, 31, 32, 34, 35, 37, 39, 41, 44, 45, 47, 48, 50, 53, 54, 56, 58, 60, 62

filter_idle_time, 17, 19, 20, 22, 24, 25, 27, 29, 31, 32, 34, 35, 37, 39, 41, 44, 45, 47, 48, 50, 53, 54, 56, 58, 60, 62

filter_infrequent_flows, 17, 19, 20, 22, 24, 25, 27, 29, 31, 32, 33, 35, 37, 39, 41, 44, 45, 47, 48, 50, 53, 54, 56, 58, 60, 62

filter_lifecycle, 17, 19, 20, 22, 24, 25, 27, 29, 31, 32, 34, 35, 37, 39, 41, 44, 45, 47, 49, 50, 53, 54, 56, 58, 60, 62

filter_lifecycle_presence, 17, 19, 20, 22, 24, 25, 27, 29, 31, 32, 34, 35, 37, 39, 41, 44, 45, 47, 49, 50, 53, 54, 56, 58, 60, 62

filter_precedence, 17, 19, 20, 22, 24, 25, 27, 29, 31, 32, 34, 35, 37, 39, 40, 41, 44, 45, 47, 49, 50, 53, 54, 56, 58, 60, 62

filter_precedence(), 44

filter_precedence_condition, 17, 19, 20, 22, 24, 25, 27, 29, 31, 32, 34, 35, 37, 39, 40, 41, 44, 45, 47, 49, 50, 53, 54, 56, 58, 60, 62

filter_precedence_resource, 17, 19, 20, 22, 24, 25, 27, 29, 31, 32, 34, 35, 37, 39, 41, 42, 45, 47, 49, 50, 53, 54, 56, 58, 60, 62

filter_processing_time, 17, 19, 20, 22, 24, 25, 27, 29, 31, 32, 34, 35, 37, 39, 41, 44, 45, 47, 49, 50, 53, 54, 56, 58, 60, 62

filter_resource, 17, 19, 20, 22, 24, 25, 27, 29, 31, 32, 34, 35, 37, 39, 41, 44, 45, 47, 49, 50, 53, 54, 56, 58, 60, 62

filter_resource_frequency, 17, 19, 20, 22, 24, 25, 27, 29, 31, 32, 34, 35, 37, 39, 41, 44, 45, 47, 49, 50, 53, 54, 56, 58, 60, 62

filter_throughput_time, 17, 19, 20, 22, 24, 25, 27, 29, 31, 32, 34, 35, 37, 39, 41, 44, 45, 47, 49, 50, 53, 54, 56, 58, 60, 62

filter_time_period, 17, 19, 20, 22, 24, 25, 29, 31, 32, 34, 35, 37, 39, 41, 44, 45, 47, 49, 50, 51, 54, 56, 58, 60, 62

filter_trace, 17, 19, 20, 22, 24, 25, 27, 29, 31, 32, 34, 35, 37, 39, 41, 44, 45, 47, 49, 50, 53, 56, 58, 60, 62

filter_trace_frequency, 17, 19, 20, 22, 24, 25, 27, 29, 31, 32, 34, 35, 37, 39, 41, 44, 45, 47, 49, 50, 53, 54, 56, 58, 60, 62

filter_trace_length, 17, 19, 20, 22, 24, 25, 27, 29, 31, 32, 34, 35, 37, 39, 41, 44, 45, 47, 49, 50, 53, 54, 56, 58, 60, 62

filter_trim_lifecycle, 17, 19, 20, 22, 24, 25, 27, 29, 31, 32, 34, 35, 37, 39, 41, 44, 45, 47, 49, 50, 53, 54, 56, 58, 60, 62

 grouped_activitylog, 5, 7, 12, 16, 33, 35, 37, 53, 55, 60, 64, 67, 70, 76, 80, 91, 93, 98

grouped_eventlog, 5, 7, 12, 16, 20, 33, 35–37, 53, 60, 61, 64, 67, 70, 76, 80, 91, 93, 98


idle_time, 5, 7, 16, 31, 62, 67, 70, 71, 76, 80, 82, 84, 91, 94, 95, 98

 idle_time(), 32, 76, 93, 94

lifecycle_id, 35, 37, 62


 logical, 4, 6, 7, 15, 16, 18, 20, 21, 23, 25, 27, 29, 30, 32, 35, 36, 39, 41, 43, 45, 46, 48, 50, 52, 54, 55, 57, 59, 61, 63, 66, 69, 75, 79, 81, 83, 84, 86, 88, 90, 93, 95, 97
INDEX

NA, 18, 30, 32, 45, 48, 50, 52, 55–57
NULL, 27, 59, 61
number_of_repetitions, 5, 7, 16, 64, 64, 70, 71, 76, 77, 80, 82, 84, 86, 91, 94, 95, 98
number_of_selfloops, 5, 7, 16, 64, 67, 67, 71, 76, 78, 80, 82, 84, 89, 91, 94, 95, 98
number_of_traces, 5, 7, 16, 64, 67, 70, 70, 76, 80, 82, 84, 91, 94, 95, 98
numeric, 11, 18, 27, 30, 32, 33, 45, 48, 50, 55, 57
plot, 71
POSIXct, 52
print.work_schedule, 73
processing_time, 5, 7, 16, 44, 64, 67, 70, 71, 73, 80, 82, 84, 91, 94, 95, 98
processing_time(), 31, 45, 64, 93, 94
redo_repetitions_referral_matrix, 76
redo_selfloops_referral_matrix, 77
resource_frequency, 5, 7, 16, 64, 67, 70, 71, 76, 78, 80, 82, 84, 91, 94, 95, 98
resource_involvement, 5, 7, 16, 64, 67, 70, 71, 76, 80, 82, 84, 91, 94, 95, 98
resource_specialisation, 5, 7, 16, 64, 67, 70, 71, 76, 80, 82, 82, 91, 94, 95, 98
resource_specialization
(resource_specialisation), 82
self-loop, 67
seq.Date, 11
size_of_repetitions, 85
size_of_selfloops, 87
start_activities, 5, 7, 15, 16, 64, 67, 70, 71, 76, 80, 82, 84, 89, 94, 95, 98
throughput_time, 5, 7, 16, 49, 64, 67, 70, 71, 76, 80, 82, 84, 91, 94, 95, 98
throughput_time(), 50, 64, 73, 76
trace_coverage, 5, 7, 16, 64, 67, 70, 71, 76, 80, 82, 84, 91, 94, 94, 98
trace_length, 5, 7, 16, 56, 58, 64, 67, 70, 71, 76, 80, 82, 84, 91, 94, 95, 96
traces, 71