Package ‘behavr’

August 9, 2017

Title Cannonical Data Structure for Behavioural Data

Date 2017-07-25

Version 0.3.0.9003

Description
Implements an S3 class based on data.table to store and process efficiently ethomics (high-throughput behavioural) data.

Imports data.table,
hms,
methods

Suggests testthat

License GPL-3

Encoding UTF-8

LazyData true

URL https://github.com/rethomics/behavr

BugReports https://github.com/rethomics/behavr/issues

RoxygenNote 6.0.1

Roxygen list(markdown = TRUE)

R topics documented:

behavr *todo*

Description

todo

Usage

```
behavr(x, metadata)

is.behavr(x)
```

Arguments

`x`	data.table containing all measurments
`metadata`	data.table containing the metadata

Details

Both `x` and `metadata` should have a **column as a key** with **the same name** (typically named `id`).

Examples

```
set.seed(1)
met <- data.table::data.table(id = 1:5,
                              condition=letters[1:5],
                              sex=c("M","M","M","F", "F"),
                              key="id")
data <- met[  ,
              list(t=1L:100L,
                  x=rnorm(100),
                  y=rnorm(100),
                  eating=runif(100) > .5 ),
              by="id"]

d <- behavr(data,met)
print(d)
summary(d)
```

`bind_behavr_list` *Put together a list of behavr tables*

Description

Bind rows of both data and metadata of all behavr tables in a list. It checks keys, number and names of columns are the same across all data. In addition, it forbigs to bind metadata if it generates duplicates (same id in two different metadata)

Usage

```
bind_behavr_list(l)
```

Arguments

`l`	list of behavr

Value

a single behavr object

Examples

```
met <- data.table::data.table(id = 1:5,
                                condition=letters[1:5],
                                sex=c("M","M","M","F", "F"),
                                key="id")
data <- met[,list(t=1L:100L,
                  x=rnorm(100),
                  y=rnorm(100),
                  eating=runif(100) > .5 ),
                  by="id"]
d1 <- behavr(data,met)

met[,id:= id+5]
data[,id:= id+5]
data.table::setkeyv(met, "id")
data.table::setkeyv(data, "id")
d2 <- behavr(data,met)

d_all <- bind_behavr_list(list(d1, d2))
print(d_all)
```

bin_apply — *Bin a variable (typically time) and compute an aggregate for each bin*

Description

This function is typically used to summarise (i.e. computing an aggreate of) a variable (y) for bins of a another variable x (typically time).

Usage

```
bin_apply(data, y, x = t, x_bin_length = mins(30), wrap_x_by = NULL,
  FUN = mean, string_xy = FALSE, ...)

bin_apply_all(data, ...)
```

Arguments

`data`	data.table or behavr table (see details)
`y`	variable to be aggregated
`x`	variable to be binned
`x_bin_length`	length of the bins (same using as 'x“)
`wrap_x_by`	numeric value defining wrapping period. `NULL` means no wrapping
`FUN`	function used to aggregate (e.g. mean, median, sum and so on)
`string_xy`	logical whether the names of the variables are quoted
`...`	additional arguments to be passed to `FUN`

Details

`bin_apply` expects data from a single individal. `bin_apply_all` works on multiple individuals identifies by a unique key. `wrapping` is typically used to compute averages accros several periods. For instance,`wrap_x_by = days(1)`, means bins will aggreate values accross sevral days. The resulting x can be interpreted as "time relative to the onset of the day" (i.e. ZT).

Examples

```
query <- data.frame(experiment_id="toy_experiment",
                    region_id=1:5)
dt <- toy_activity_data(query, duration=days(4))

# average by 30min time bins, default
dt_binned <- bin_apply_all(dt,moving)
# equivalent to
dt_binned <- dt[, bin_apply(.SD, moving),by="id"]

# More advanced usage
dt <- toy_dam_data(query, duration=days(4))
```

```
# nsum activity per 60minutes
dt_binned <- bin_apply_all(dt,
                           activity,
                           x=t,
                           x_bin_length = mins(60),
                           FUN=sum)


#' # average activity. time in ZT
dt_binned <- bin_apply_all(dt,
                           activity,
                           x=t,
                           wrap_x_by=days(1)
                           )
```

`meta` *Retreive metadata*

Description

This function returns the meta data from a behavr object

Usage

```
meta(x)

setmeta(x, new)
```

Arguments

`x`	a behavr object
`new`	a new metadata table

Value

a data.table representing the metadata in `x`

See Also

behavr to generate a `behavr` object, xmv to map metavariables to data

Examples

```
set.seed(1)
met <- data.table::data.table(id = 1:5,
                              condition=letters[1:5],
                              sex=c("M","M","M","F", "F"),
                              key="id")
```

```
data <- met[  ,
              list(t=1L:100L,
                  x=rnorm(100),
                  y=rnorm(100),
                  eating=runif(100) > .5 ),
              by="id"]

d <- behavr(data,met)
## show metadata
meta(d)
# same as:
d[meta=TRUE]
## set metadata
m <- d[meta=TRUE]
# only id > 2 is kept
setmeta(d, m[id < 3])
meta(d)
```

print.behavr — *Print and summarise a behavr table*

Description

Print and summarise a behavr table

Usage

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

## S3 method for class 'behavr'
summary(object, ...)
```

Arguments

`x, object`	behavr table
`...`	arguments passed on to further method

See Also

behavr, print.default, summary.default

rejoin *Join data and metadata*

Description

This function joins the data of a behavr object to its metadata

Usage

```
rejoin(x)
```

Arguments

`x`	a behavr object

Value

a data.table

See Also

behavr to generate a `behavr` object

Examples

```
set.seed(1)
met <- data.table::data.table(id = 1:5,
                              condition=letters[1:5],
                              sex=c("M","M","M","F", "F"),
                              key="id")
data <- met[  ,
              list(t=1L:100L,
                  x=rnorm(100),
                  y=rnorm(100),
                  eating=runif(100) > .5 ),
              by="id"]

d <- behavr(data,met)
summary_d <- d[, .(test=mean(x)), by=id]
rejoin(summary_d)
```

time_conversion *Time conversion utilities*

Description

Trivial functions to convert time to seconds – as `rethomics` uses second as a conventionnal unit of time.

Usage

```
days(x)

hours(x)

mins(x)
```

Arguments

`x`	Numerical vector to be converted in second

Details

Given an dummy function that takes time in second like: `myFunction(t)`, it is always preferqable to call `myFunction(days(1.5))` rather than `myFunction(60*60*24*1.5)`.

Value

Number of seconds corresponding to `x` (1d = 86400s, 1h = 3600s and 1min = 60s)

toy_activity_data *Generate toy activity and sleep data mimiking Drosophila behaviour in tubes*

Description

This function generates random data that emulates some of the features of fruit fly activity and sleep. This is designed **exclusively to provide material for examples and tests** since it generates "realistic" datasets of arbitrary length.

Usage

```
toy_activity_data(query = NULL, seed = 1, rate_range = 1/c(60, 10),
  duration = days(5), sampling_period = 10, ...)

toy_ethoscope_data(...)

toy_dam_data(...)
```

Arguments

`query`	query (i.e. a dataframe where every row defines an animal). Typically queries have, at least, the columns `experiment_id` and `region_id`. The default value (`NULL`), will generate data for a single animal.
`seed`	Random seed used.
`rate_range`	a parameter defining the boundaries of rate at which animals wake up. It will be uniformely distributed between animals, but fixed for each animal.
`duration`	Length (in second) of the data to generate.
`sampling_period`	sampling period (in second) of the resulting data.
`...`	Additional arguments to be passed to `simulateAnimalActivity`

Value

A behavr table with the query columns as meta variables. In addition to `id` and `t` columns different method will output different variables:

- `toy_activity_data` will have `asleep` and `moving` (1/10s)
- `toy_dam_data` will have `activity` (1/60s)
- `toy_ethoscope_data` `xy_dist_log10x1000` `has_interacted` `x` (2/1s)

Examples

```
# just one animal, no query needed
dt <- toy_ethoscope_data(duration=days(3))

# advanced, using a query
query<- data.frame(experiment_id="toy_experiment",
                   region_id=1:10,
                   condition=c("A","B"))


# Data that could come from loadEthoscopeData:
dt <- toy_ethoscope_data(query,duration=days(1))
print(dt)

# Some DAM-like data
dt <- toy_dam_data(query,seed=2,duration=days(3))
print(dt)

# some data that would come from `sleepAnnotation` or `sleepDAMAnnotation`
dt <- toy_activity_data(query,3)
print(dt)
```

xmv — *Extract a metavariable and map it against the data*

Description

This function eXpands a MetaVariable from a parent behavr object. That is it matches this variable (from metadata) to the data *by id*.

Usage

```
xmv(var)
```

Arguments

`var`	the variable to be extracted

Details

This function *can only be called within between the* `[]` *of a parent* behavr object. It is intended to facilitate operations between data and metadata. For instance, when one wants to modify a column of the data according a metavariable.

Value

a vector of the same type as `var`, but of the same length as the number of row in the parent data. As each row of data is matched against metadata for this specific variable.

Examples

```
library(data.table)
set.seed(1)
data <- data.table(
                   id = rep(c("A","B"), times=c(10,26)),
                   t = c(1:10,5:30),
                   x = rnorm(36), key="id"
                   )

metadata = data.table(id=c("A","B"), treatment=c("w","z"), lifespan=c(19,32), ref_x=c(1,0),key="id")
dt <- behavr(data,metadata)
summary(dt)

#### Subsetting using metadata
dt[xmv(treatment) == "w"]
dt[xmv(treatment) == "w"]
dt[xmv(lifespan) < 30]

#### Allocating new columns using metavariable
# Just joining lifespan (not necessary)
dt[, lif := xmv(lifespan)]
```

```
print(dt)
# Anonymously (more useful)
dt[, x2 := x-xmv(ref_x)]
print(dt)
```

Index