Package 'shinyHugePlot'

August 22, 2022

Type Package

Title Efficient Plotting of Large-Sized Data

Version 0.0.1

Maintainer Junta Tagusari <j.tagusari@eng.hokudai.ac.jp>

Language en-US

Depends R (>= 4.2.0), plotly (>= 4.10.0), shiny (>= 1.7.1)

Imports R6 (>= 2.5.1), dplyr (>= 1.0.9), tibble (>= 3.1.7), tidyr (>= 1.2.0), data.table (>= 1.14.2), stringr (>= 1.4.0), nanotime (>= 0.3.6), assertthat (>= 0.2.1), bit64 (>= 4.0.5), purrr (>= 0.3.4)

Suggests testthat

Description A tool to plot data with large sample size using 'shiny' and 'plotly'.
Relatively small samples are chosen from the data using an appropriate algorithm according to a user-defined x range.
Jonas Van Der Donckt, Jeroen Van Der Donckt, Emiel Deprost (2022) <https://github.com/predict-idlab/plotly-resampler>.

License MIT + file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 7.2.1

NeedsCompilation no

Author Junta Tagusari [aut, cre],
Jonas Van Der Donckt [cph],
Jeroen Van Der Donckt [cph],
Emiel Deprost [cph]

Repository CRAN

Date/Publication 2022-08-22 10:00:02 UTC

R topics documented:

`abstract_aggregator` *R6 super class for the aggregation*

Description

An abstract class for the aggregation, which defines the structure of the class and is not available on a stand-alone basis.

Format

An `R6::R6Class` object

Public fields

`interleave_gaps` Whether `NA` values should be added when there are gaps / irregularly sampled data

`accepted_datatype` Vector of supported data classes

`nan_position` Character that indicates where `NA`s are placed when gaps are detected

Methods

Public methods:

- abstract_aggregator$new()
- abstract_aggregator$aggregate()
- abstract_aggregator$clone()

Method `new()`**:** Constructor of abstract_aggregator

Usage:

```
abstract_aggregator$new(
  interleave_gaps = FALSE,
  nan_position = "end",
  accepted_datatype = NULL
)
```

Arguments:

`interleave_gaps` Boolean, optional. Whether `NA` values should be added when there are gaps / irregularly sampled data. A quantile-based approach is employed. By default, `FALSE`.

`nan_position` Character, optional. Indicates where `NA`s are placed when gaps are detected. If `"end"`, the first point after a gap will be replaced. If `"begin"`, the last point before a gap will be replaced. If `"both"`, both the encompassing gap data points are replaced. This parameter is only effective when `interleave_gaps == TRUE`.

`accepted_datatype` Character vector, optional. This parameter indicates the supported data classes. If all data classes are accepted, set it to `NULL`.

Method `aggregate()`**:** Aggregates the given input and returns samples.

Usage:

```
abstract_aggregator$aggregate(x, y, n_out)
```

Arguments:

`x, y` Indexes and values that has to be aggregated.

`n_out` Integer. The number of samples that the aggregated data contains.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
abstract_aggregator$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

abstract_downsampler *R6 class of abstract down-sampler*

Description

An abstract class for the down-sampler, which defines the structure of the class and is not available on a stand-alone basis.

Format

An R6::R6Class object

Public fields

figure plotly object.

Methods

Public methods:

- abstract_downsampler$new()
- abstract_downsampler$add_trace()
- abstract_downsampler$get_figure_data()
- abstract_downsampler$clone()

Method new(): Constructing an abstract down-sampler.

Usage:

```
abstract_downsampler$new(
  figure = plotly::plot_ly(),
  is_downsample = TRUE,
  n_out = 1000L,
  aggregator = eLTTB_aggregator$new(),
 legend_options = list(downsample_prefix = "<b style=\"color:sandybrown\">[R]</b> ",
    downsample_suffix = "", is_aggsize_shown = TRUE, agg_prefix =
    "<i style=\"color:#fc9944\"> ~", agg_suffix = "</i>"),
  tz = Sys.timezone()
)
```

Arguments:

figure Plotly structure that will be down-sampled.

is_downsample Boolean. Whether down-sampling is done. By default TRUE.

n_out Integer or numeric. The number of samples shown after down-sampling. By default 1000.

aggregator An instance of an R6 class for aggregation. Select one out of LTTB_aggregator, min_max_ovlp_aggregator, min_max_aggregator, eLTTB_aggregator, nth_pnt_aggregator, custom_stat_aggregator, mean_aggregator, median_aggregator, min_aggregator, max_aggregator, or custom_func_aggregator. By default eLTTB_aggregator.

`legend_options` Named list, optional. Names of the elements are `prefix_downsample`, `suffix_downsample`, `is_aggsize_shown`, `agg_prefix`, and `agg_suffix`. The `prefix_downsample` and `suffix_downsample` will be added to the legend name when the traces are down-sampled. By default, prefix is a bold orange `[R]` and suffix is none. The `is_aggsize_shown` is boolean. Whether the mean aggregation bin size will be added to the legend name. By default `TRUE`. The `agg_prefix` and `agg_suffix` are employed to show the mean aggregation size

`tz` Character, optional. Time zone used to display time-series data. By default `Sys.timezone()`.

Method `add_trace()`**:** Adds a trace to the figure (`self$figure`) and returns nothing.

Usage:

```
abstract_downsampler$add_trace(
  ...,
  n_out = NULL,
  aggregator = NULL,
  hf_x = NULL,
  hf_y = NULL,
  hf_text = NULL,
  hf_hovertext = NULL,
  tz = Sys.timezone()
)
```

Arguments:

`...` Arguments passed along to the plotly trace. (e.g., `id`, `x`, `y`, `data`, `type`, `mode`).

`n_out` Integer, optional. The max number of samples that are shown in the figure. By default, `private$n_out_default`.

`aggregator` An instance of an R6 class for aggregation. The aggregator used for downsampling. By default, `private$aggregator_default`.

`hf_x`, `hf_y`, `hf_text`, `hf_hovertext` Optional. The original high frequency data for `x`, `y`, `text` and `hovertext`.

`tz` Character, optional. The timezone used in the plotly. By default, `Sys.timezone()`.

Method `get_figure_data()`**:** Returns the list of the data used for the plotly figure.

Usage:

```
abstract_downsampler$get_figure_data()
```

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
abstract_downsampler$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

apply_downsampler *Wrapper for applying down-sampling*

Description

This function is a wrapper that employs an aggregator.

Usage

```
apply_downsampler(obj, ...)

## S3 method for class 'numeric'
apply_downsampler(obj, n_out = 1000L, aggregator = eLTTB_aggregator, ...)

## S3 method for class 'data.frame'
apply_downsampler(obj, n_out = 1000L, aggregator = eLTTB_aggregator, ...)
```

Arguments

obj	Numeric vector or data.frame. If a numeric vector is given, a specific down-sampling method will be employed and a down-sampled vector will be returned. If a data.frame is given, a specific down-sampling method will be employed using x and y columns and a down-sampled data.frame will be returned.
...	Not used.
n_out	Integer, optional. Number of samples get by the down-sampling. By default, 1000.
aggregator	R6 class for the aggregation, optional. Select one out of LTTB_aggregator, min_max_ovlp_aggregator, min_max_aggregator, eLTTB_aggregator, nth_pnt_aggregator, custom_stat_aggregator, mean_aggregator, median_aggregator, min_aggregator, max_aggregator, or custom_func_aggregator. By default eLTTB_aggregator.

Examples

```
data(noise_fluct)

y_agg <- apply_downsampler(noise_fluct$level)

d_agg <- noise_fluct %>%
  dplyr::select(x = sec, y = level) %>%
  apply_downsampler()
```

custom_func_aggregator *R6 Class for Aggregation using a user-defined function.*

Description

Arbitrary function can be applied using this aggregation class.

Format

An `R6::R6Class` object

Super class

`shinyHugePlot::abstract_aggregator` -> `custom_func_aggregator`

Public fields

`aggregation_func` User-defined function to aggregate data, of which arguments are `x`, `y` and `n_out`.

Methods

Public methods:

- `custom_func_aggregator$new()`
- `custom_func_aggregator$clone()`

Method `new()`**:** Constructor of the Aggregator.

Usage:

```
custom_func_aggregator$new(
  aggregation_func,
  interleave_gaps = FALSE,
  nan_position = "end",
  accepted_datatype = NULL
)
```

Arguments:

`aggregation_func` User-defined function to aggregate data, of which arguments are `x`, `y` and `n_out`.

`interleave_gaps, nan_position, accepted_datatype` Arguments pass to the constructor of the `abstract_aggregator` class.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
custom_func_aggregator$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
custom_agg_func <- function(x, y, n_out) {
  bin_width <- floor(length(x)/n_out)
  x_idx <- seq(floor(bin_width / 2), bin_width * n_out, bin_width)
  y_mat <- y[1:(bin_width * n_out)] %>%
    matrix(nrow = bin_width)
  y_agg <- apply(y_mat, 2, quantile, probs = 0.25)
  return(list(x = x[x_idx], y = y_agg))
}
data(noise_fluct)
agg <- custom_func_aggregator$new(custom_agg_func)
d_agg <- agg$aggregate(noise_fluct$sec, noise_fluct$level, 1000)
plot(d_agg$x, d_agg$y, type = "l")
```

---

custom_stat_aggregator

R6 Class for aggregation which returns the custom statistical values

---

Description

This aggregator divides the data into no-overlapping intervals and calculate specific statistical values such as the mean.

Format

An `R6::R6Class` object

Super class

`shinyHugePlot::abstract_aggregator` -> `custom_stat_aggregator`

Methods

Public methods:

- `custom_stat_aggregator$new()`
- `custom_stat_aggregator$clone()`

Method `new()`**:** Constructor of the Aggregator.

Usage:

```
custom_stat_aggregator$new(
  y_func = function(x) mean(x, na.rm = TRUE),
  interleave_gaps = FALSE,
  nan_position = "end"
)
```

Arguments:

`y_func` Function. Statistical values are calculated using this function. By default, `mean`.

`interleave_gaps`, `nan_position` Arguments pass to the constructor of the `abstract_aggregator` class.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
custom_stat_aggregator$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
data(noise_fluct)
agg <- custom_stat_aggregator$new(y_func = function(x) mean(x, na.rm = TRUE))
d_agg <- agg$aggregate(noise_fluct$sec, noise_fluct$level, 1000)
plot(d_agg$x, d_agg$y, type = "l")
```

---

eLTTB_aggregator *R6 Class for Efficient LTTB aggregation*

Description

Efficient version off LTTB by first reducing really large data with the `min_max_ovlp_aggregator` and then further aggregating the reduced result with `LTTB_aggregator`.

Format

An `R6::R6Class` object

Super class

`shinyHugePlot::abstract_aggregator` -> `eLTTB_aggregator`

Public fields

`LTTB` An R6 LTTB_aggregator instance

`minmax` An R6 `min_max_ovlp_aggregator` instance

Methods

Public methods:

- `eLTTB_aggregator$new()`
- `eLTTB_aggregator$clone()`

Method `new()`**:** Constructor of the aggregator.

Usage:

```
eLTTB_aggregator$new(interleave_gaps = FALSE, nan_position = "end")
```

Arguments:

`interleave_gaps, nan_position` Arguments pass to the constructor of the `abstract_aggregator` class.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
eLTTB_aggregator$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
data(noise_fluct)
agg <- eLTTB_aggregator$new()
d_agg <- agg$aggregate(noise_fluct$sec, noise_fluct$level, 1000)
plot(d_agg$x, d_agg$y, type = "l")
```

`LTTB` *Downsample with the Largest Triangle Three Buckets (LTTB) aggregation method*

Description

Downsample with the Largest Triangle Three Buckets (LTTB) aggregation method

Usage

```
LTTB(x, y, n_out)
```

Arguments

`x, y`	numeric vectors.
`n_out`	length of the output. This must be larger than 2 and lesser than the number of the rows of ‘data‘

Value

named list of ‘x‘ and ‘y‘

LTTB_aggregator *Aggregation using Largest Triangle Three Buckets (LTTB) method.*

Description

The LTTB method aggregates the huge samples using the areas of the triangles formed by the samples. Numerical distances are employed in this class, which requires the ratio between x and y values. When the x is datetime, nanosecond is a unit. When the x is factor or character, it will be encoded into numeric codes.

Format

An `R6::R6Class` object

Super class

`shinyHugePlot::abstract_aggregator` -> `LTTB_aggregator`

Methods

Public methods:

- `LTTB_aggregator$new()`
- `LTTB_aggregator$clone()`

Method `new()`**:** Constructor of the Aggregator.

Usage:

```
LTTB_aggregator$new(
  interleave_gaps = FALSE,
  nan_position = "end",
  nt_y_ratio = 1e+09,
  x_y_ratio = 1
)
```

Arguments:

`interleave_gaps`, `nan_position` Arguments pass to the constructor of the `abstract_aggregator` class.

`x_y_ratio`, `nt_y_ratio` Numeric. These parameters set the unit length of the numeric `x` and `nanotime` x. For example, setting `x_y_ratio` to 2 is equivalent to assuming 2 is the unit length of `x` (and 1 is always the unit length of `y`). The unit length is employed to calculate the area of the triangles.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
LTTB_aggregator$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
data(noise_fluct)
agg <- LTTB_aggregator$new()
d_agg <- agg$aggregate(noise_fluct$sec, noise_fluct$level, 1000)
plot(d_agg$x, d_agg$y, type = "l")
```

`max_aggregator` *R6 Class for aggregation which returns the max values*

Description

The maximum value for each interval is calculated.

Format

An `R6::R6Class` object

Super classes

`shinyHugePlot::abstract_aggregator` -> `shinyHugePlot::custom_stat_aggregator` -> `mean_aggregator`

Methods

Public methods:

- `max_aggregator$new()`
- `max_aggregator$clone()`

Method `new()`**:** Constructor of the Aggregator.

Usage:

```
max_aggregator$new(interleave_gaps = FALSE, nan_position = "end")
```

Arguments:

`interleave_gaps, nan_position` Arguments pass to the constructor of the `abstract_aggregator` class.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
max_aggregator$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

mean_aggregator *R6 Class for aggregation which returns the mean values*

Description

The mean value for each interval is calculated.

Format

An R6::R6Class object

Super classes

shinyHugePlot::abstract_aggregator -> shinyHugePlot::custom_stat_aggregator -> mean_aggregator

Methods

Public methods:

- mean_aggregator$new()
- mean_aggregator$clone()

Method new(): Constructor of the Aggregator.

Usage:

```
mean_aggregator$new(interleave_gaps = FALSE, nan_position = "end")
```

Arguments:

interleave_gaps, nan_position Arguments pass to the constructor of the abstract_aggregator class.

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
mean_aggregator$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

median_aggregator *R6 Class for aggregation which returns the median values*

Description

The median value for each interval is calculated.

Format

An `R6::R6Class` object

Super classes

`shinyHugePlot::abstract_aggregator` -> `shinyHugePlot::custom_stat_aggregator` -> `mean_aggregator`

Methods

Public methods:

- `median_aggregator$new()`
- `median_aggregator$clone()`

Method `new()`**:** Constructor of the Aggregator.

Usage:

```
median_aggregator$new(interleave_gaps = FALSE, nan_position = "end")
```

Arguments:

`interleave_gaps, nan_position` Arguments pass to the constructor of the `abstract_aggregator` class.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
median_aggregator$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`min_aggregator` *R6 Class for aggregation which returns the minimum values*

Description

The minimum value for each interval is calculated.

Format

An `R6::R6Class` object

Super classes

`shinyHugePlot::abstract_aggregator` -> `shinyHugePlot::custom_stat_aggregator` -> `mean_aggregator`

Methods

Public methods:

- `min_aggregator$new()`
- `min_aggregator$clone()`

Method `new()`**:** Constructor of the Aggregator.

Usage:

```
min_aggregator$new(interleave_gaps = FALSE, nan_position = "end")
```

Arguments:

`interleave_gaps, nan_position` Arguments pass to the constructor of the `abstract_aggregator` class.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
min_aggregator$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

`min_max_aggregator` *R6 Class for Min-Max Aggregation with fully overlapping windows*

Description

Divide the data into fully overlapping intervals and find the maximum and minimum values of each.

Format

An `R6::R6Class` object

Super class

`shinyHugePlot::abstract_aggregator` -> `min_max_aggregator`

Methods

Public methods:

- `min_max_aggregator$new()`
- `min_max_aggregator$clone()`

Method `new()`**:** Constructor of the Aggregator.

Usage:

```
min_max_aggregator$new(interleave_gaps = FALSE, nan_position = "end")
```

Arguments:

`interleave_gaps, nan_position` Arguments pass to the constructor of the `abstract_aggregator` class.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
min_max_aggregator$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
data(noise_fluct)
agg <- min_max_aggregator$new()
d_agg <- agg$aggregate(noise_fluct$sec, noise_fluct$level, 1000)
plot(d_agg$x, d_agg$y, type = "l")
```

min_max_ovlp_aggregator *R6 Class for Min-Max Aggregation that has 50% overlapping windows.*

Description

Divide the data into 50% overlapping intervals and find the maximum and minimum values of each.

Format

An `R6::R6Class` object

Super class

`shinyHugePlot::abstract_aggregator` -> `min_max_ovlp_aggregator`

Methods

Public methods:

- `min_max_ovlp_aggregator$new()`
- `min_max_ovlp_aggregator$clone()`

Method `new()`**:** Constructor of the Aggregator.

Usage:

```
min_max_ovlp_aggregator$new(interleave_gaps = FALSE, nan_position = "end")
```

Arguments:

`interleave_gaps, nan_position` Arguments pass to the constructor of the `abstract_aggregator` class.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
min_max_ovlp_aggregator$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
data(noise_fluct)
agg <- min_max_ovlp_aggregator$new()
d_agg <- agg$aggregate(noise_fluct$sec, noise_fluct$level, 1000)
plot(d_agg$x, d_agg$y, type = "l")
```

`nanosecond_to_label` *Generate an appropriate time-based label using nanotime value*

Description

Nano-second time is converted to an appropriate label. e.g., 1e9 nano seconds will be converted to `"1.0s"`

Usage

```
nanosecond_to_label(ns)
```

Arguments

`ns`	Numeric value(s) representing nano second.

Value

Character.

`nanotime_to_plotlytime` *Conversion of nanotime to plotly-style time*

Description

Plotly does not accept 64-bit nanotime format, so the conversion is necessary to import the time to plotly.

Usage

```
nanotime_to_plotlytime(time, tz)
```

Arguments

`time`	nanotime class time.
`tz`	time zone (e.g. "Asia/Tokyo")

Value

local time (strings) with the format of

noise_fluct *Time-series fluctuations in sound level*

Description

Results of the measurement of the sound level, where peaks due to road traffic are observed.

Author(s)

Junta Tagusari <j.tagusari@eng.hokudai.ac.jp>

nth_pnt_aggregator *R6 Class for Naive (but fast) aggregation which returns every Nth point.*

Description

Aggregation by extracting every Nth data.

Format

An R6::R6Class object

Super class

shinyHugePlot::abstract_aggregator -> nth_pnt_aggregator

Methods

Public methods:

- nth_pnt_aggregator$new()
- nth_pnt_aggregator$clone()

Method new(): Constructor of the Aggregator.

Usage:

```
nth_pnt_aggregator$new(interleave_gaps = FALSE, nan_position = "end")
```

Arguments:

interleave_gaps, nan_position Arguments pass to the constructor of the abstract_aggregator class.

Method clone(): The objects of this class are cloneable with this method.

Usage:

```
nth_pnt_aggregator$clone(deep = FALSE)
```

Arguments:

deep Whether to make a deep clone.

Examples

```
data(noise_fluct)
agg <- nth_pnt_aggregator$new()
d_agg <- agg$aggregate(noise_fluct$sec, noise_fluct$level, 1000)
plot(d_agg$x, d_agg$y, type = "l")
```

`numeric_to_label` *Generate an appropriate label*

Description

Numeric value is converted to an appropriate label. e.g., 1e9 will be converted to `"1.0G"`

Usage

```
numeric_to_label(x)
```

Arguments

`x`	Numeric value(s).

Value

Character.

`plotlytime_to_nanotime` *Conversion of plotly-style time to nanotime*

Description

Datetime obtained from plotly are converted to 64-bit nanotime format.

Usage

```
plotlytime_to_nanotime(time, tz)
```

Arguments

`time`	local time (strings) with the format of '%Y-%m-%d %H:%M:%S.s'
`tz`	time zone (e.g. "Asia/Tokyo")

Value

nanotime

range_stat_aggregator *R6 Class for aggregation which returns the 3 types of the statistical values*

Description

This aggregator divides the data into no-overlapping intervals and calculate specific statistical values such as the mean.

Format

An `R6::R6Class` object

Super class

`shinyHugePlot::abstract_aggregator` -> `custom_stat_aggregator`

Methods

Public methods:

- `range_stat_aggregator$new()`
- `range_stat_aggregator$clone()`

Method `new()`**:** Constructor of the Aggregator.

Usage:

```
range_stat_aggregator$new(
  ylwr = function(x) min(x, na.rm = TRUE),
  y = function(x) mean(x, na.rm = TRUE),
  yupr = function(x) max(x, na.rm = TRUE),
  interleave_gaps = FALSE,
  nan_position = "end"
)
```

Arguments:

`yupr`, `y`, `ylwr` Functions. Statistical values are calculated using this function. By default, `max`, `mean`, `min`, respectively. Note that the functions need to deal with NA values.

`interleave_gaps`, `nan_position` Arguments pass to the constructor of the `abstract_aggregator` class.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
range_stat_aggregator$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
data(noise_fluct)
agg <- range_stat_aggregator$new(ylwr = min, y = mean, yupr = max)
d_agg <- agg$aggregate(noise_fluct$sec, noise_fluct$level, 1000)
plot(d_agg$x, d_agg$ylwr, type = "l")
plot(d_agg$x, d_agg$y, type = "l")
plot(d_agg$x, d_agg$yupr, type = "l")
```

`shinyHugePlot` *shinyHugePlot*

Description

Efficiently Plot Huge Data using Plotly in R

`shiny_downsampler` *R6 Class implementing down-sampling in shiny app*

Description

This class includes high-frequency original data, plotly figure, and shiny. The plotly figure will be made by initializing the instance and using `add_trace` method. (note that the method is different from `plotly::add_trace`). The easiest way to run shiny app is using `show_shiny` method. Or, you can register the figures using `register_figures` method then you can run the app using `run_server` method.

Format

An `R6::R6Class` object

Super class

`shinyHugePlot::abstract_downsampler` -> `shiny_downsampler`

Public fields

`shiny_session` ShinySession R6 instance.

Methods

Public methods:

- shiny_downsampler$new()
- shiny_downsampler$update_figure_data()
- shiny_downsampler$show_shiny()
- shiny_downsampler$clone()

Method `new()`**:** Create a new downsampler

Usage:

```
shiny_downsampler$new(
  figure = plotly::plot_ly(),
  is_downsample = TRUE,
  n_out = 1000L,
  aggregator = eLTTB_aggregator$new(),
 legend_options = list(downsample_prefix = "<b style=\"color:sandybrown\">[R]</b> ",
    downsample_suffix = "", is_aggsize_shown = TRUE, agg_prefix =
    " <i style=\"color:#fc9944\">~", agg_suffix = "</i>"),
  tz = Sys.timezone()
)
```

Arguments:

`figure`, `is_downsample`, `n_out`, `aggregator`, `legend_options`, `tz` Arguments pass to the constructor of the `abstract_downsampler` class.

Method `update_figure_data()`**:** update the trace data according to the relayout order.

Usage:

```
shiny_downsampler$update_figure_data(relayout_order = list())
```

Arguments:

`relayout_order` Named list. The list is generated by converging the dictionary obtained from `plotlyjs_relayout`.

Method `show_shiny()`**:** Easily output the shiny app.

Usage:

```
shiny_downsampler$show_shiny(shiny_options = list())
```

Arguments:

`shiny_options` Named list. Arguments passed to `shinyApp` as the options.

Method `clone()`**:** The objects of this class are cloneable with this method.

Usage:

```
shiny_downsampler$clone(deep = FALSE)
```

Arguments:

`deep` Whether to make a deep clone.

Examples

```
library(plotly)
data(noise_fluct)
p <- plot_ly(noise_fluct) %>%
  add_trace(x = ~sec, y = ~level, type = "scatter", mode = "lines")
d_app <- shiny_downsampler$new(p)
d_app$show_shiny()
```

---

shiny_hugeplot *Wrapper for plot huge data in shiny and plotly*

Description

This function is an easy wrapper to plot the huge data. It employs an R6 shiny_downsampler instance to obtain samples from data using a specified aggregation method. The figure will be updated interactively according to the x-range that user can define manually in the shiny app.

Usage

```
shiny_hugeplot(obj, ...)

## Default S3 method:
shiny_hugeplot(
  obj = NULL,
  y = NULL,
  n_out = 1000L,
  aggregator = eLTTB_aggregator,
  run_shiny = TRUE,
  downsampler_options = list(),
  plotly_options = list(type = "scatter", mode = "lines+markers"),
  plotly_layout_options = list(),
  shiny_options = list(),
  ...
)

## S3 method for class 'nanotime'
shiny_hugeplot(
  obj = NULL,
  y = NULL,
  tz = Sys.timezone(),
  n_out = 1000L,
  aggregator = eLTTB_aggregator,
  run_shiny = TRUE,
  downsampler_options = list(),
  plotly_options = list(type = "scatter", mode = "lines+markers"),
```

```
  plotly_layout_options = list(xaxis = list(type = "date")),
  shiny_options = list(),
  ...
)

## S3 method for class 'matrix'
shiny_hugeplot(
  obj = NULL,
  n_out = 1000L,
  aggregator = eLTTB_aggregator,
  run_shiny = TRUE,
  downsampler_options = list(),
  plotly_options = list(type = "scatter", mode = "lines+markers"),
  plotly_layout_options = list(),
  shiny_options = list(),
  ...
)

## S3 method for class 'data.frame'
shiny_hugeplot(
  obj = NULL,
  tz = Sys.timezone(),
  n_out = 1000L,
  aggregator = eLTTB_aggregator,
  run_shiny = TRUE,
  downsampler_options = list(),
  plotly_options = list(type = "scatter", mode = "lines+markers"),
  plotly_layout_options = list(),
  shiny_options = list(),
  ...
)

## S3 method for class 'plotly'
shiny_hugeplot(
  obj,
  n_out = 1000L,
  aggregator = eLTTB_aggregator,
  run_shiny = TRUE,
  downsampler_options = list(),
  shiny_options = list(),
  ...
)
```

Arguments

obj	Numeric vector, nanotime vector, numeric matrix, data.frame, or plotly object. If a numeric vector is given, it is used as the y values of the figure. the x values are calculated by `seq_along(obj)`. If you use y argument together, this argument is interpreted as the x values. If a nanotime vector is given, it is used as

	the x values of the figure. You must also give y argument, used as the y values. Regarding nanotime, see the nanotime package. If a numeric matrix is given, the first and second column values are used as the x and y values. The matrix must have more than 2 columns. If a data.frame is given, d$x and d$y are used as the x and y values. If the class of the d$x is nanotime, the time-scale x axis is applied. The data.frame must have columns named x and y. If a plotly object is given, it will be displayed as the figure.
...	Not used.
y	Numeric vector, optional. The y values of the figure.
n_out	Integer, optional. Number of samples get by the down-sampling. By default, 1000.
aggregator	R6 class for the aggregation, optional. Select one out of LTTB_aggregator, min_max_ovlp_aggregator, min_max_aggregator, eLTTB_aggregator, nth_pnt_aggregator, custom_stat_aggregator, mean_aggregator, median_aggregator, min_aggregator, max_aggregator, or custom_func_aggregator. By default eLTTB_aggregator.
run_shiny	Boolean, optional. whether a generated shiny app will be run immediately. By default, TRUE.
downsampler_options	Named list, optional. Arguments passed to shiny_downsampler$new or the constructor of the specific aggregator. To set aggregator and n_shown_samples, use aggregator and n_out arguments.
plotly_options	Named list, optional. Arguments passed to plotly::plot_ly.
plotly_layout_options	Named list, optional. Arguments passed to plotly::layout.
shiny_options	Named list, optional. Arguments passed to shinyApp function.
tz	Timezone, optional. It is used to convert the nanotime to the time displayed in the figure. By default, Sys.timezone().

Examples

```
data(noise_fluct)

shiny_hugeplot(noise_fluct$level)
shiny_hugeplot(
  noise_fluct$t, noise_fluct$level,
  plotly_layout_options = list(xaxis = list(type = "date"))
)
```

---

updatePlotlyH *Wrapper for updating plotly with huge data*

Description

using this function, updater of the plotly with huge data is easily installed.

Usage

```
updatePlotlyH(session, outputId, relayout_order, sd_obj)
```

Arguments

session	The session object passed to function given to shinyServer.
outputId	Character. The outputId of the plotly that will be down-sampled
relayout_order	Named list. The list is generated by converging the dictionary obtained from plotlyjs_relayout.
sd_obj	The shiny_downsampler instance that is used for generating the figure and contains the full data.

Examples

```
data(noise_fluct)
fig <- plot_ly(x = d$x, y = d$y, type = "scatter", mode = "lines")

shd <- shiny_downsampler$new(figure = fig)

ui <- fluidPage(
  plotlyOutput(outputId = "hp", width = "800px", height = "600px")
)

server <- function(input, output, session) {

  output$hp <- renderPlotly(shd$figure)

  observeEvent(plotly::event_data("plotly_relayout"),{
    updatePlotlyH(session, "hp", plotly::event_data("plotly_relayout"), shd)
  })

}

shinyApp(ui = ui, server = server)
```

Index