

Package ‘parallelPlot’

July 23, 2025

Title `htmlwidget` for a Parallel Coordinates Plot

Version 0.4.0

Description Create a parallel coordinates plot, using `htmlwidgets` package and `d3.js`.

URL <https://gitlab.com/drti/parallelplot>

BugReports <https://gitlab.com/drti/parallelplot/-/issues>

Depends R (>= 3.5.0)

License MIT + file LICENSE

Encoding UTF-8

RoxygenNote 7.2.3

Imports htmlwidgets

Suggests testthat, shiny, knitr, rmarkdown

VignetteBuilder knitr

NeedsCompilation no

Author David Chazalviel [aut, cre],
Benoit Lehman [aut]

Maintainer David Chazalviel <david.chazalviel@club-internet.fr>

Repository CRAN

Date/Publication 2024-06-17 13:00:34 UTC

Contents

changeRow	2
getPlotConfig	3
getValue	4
highlightRow	6
parallelPlot	7
parallelPlot-shiny	10
setArrangeMethod	11
setCategoricalColorScale	12
setCategoriesRep	13

setContinuousColorScale	15
setCutoffs	16
setHistoVisibility	17
setInvertedAxes	18
setKeptColumns	19
setRefColumnDim	20
Index	22

changeRow	<i>Row edition</i>
-----------	--------------------

Description

Asks to change a row.

Usage

changeRow(id, rowIndex, newValues)

Arguments

- id output variable to read from (id which references the requested plot)
- rowIndex index of the changed row.
- newValues list of new values to attribute to the row (list associating a value to a column identifier).

Value

No return value, called from shiny applications for side effects.

Examples

```
if(interactive() && require(shiny)) {
  library(shiny)
  library(parallelPlot)

  ui <- fluidPage(
    sliderInput(
      "rowValueSlider",
      "Value for 'Sepal.Length' of first row:",
      min = 4, max = 8, step = 0.1,
      value = iris[["Sepal.Length"]][1]
    ),
    p("Slider controls the new value to assign to 'Sepal.Length' of the first row"),
    parallelPlotOutput("parPlot")
  )

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

```

    output$parPlot <- renderParallelPlot({
      parallelPlot(iris)
    })
    observeEvent(input$rowValueSlider, {
      newValues <- iris[1,]
      newValues[["Sepal.Length"]] <- input$rowValueSlider
      parallelPlot::changeRow("parPlot", 1, newValues)
    })
  }

  shinyApp(ui, server)
}

```

getPlotConfig	<i>Retrieve plot configuration</i>
---------------	------------------------------------

Description

Result will be sent through a reactive input (see example below).

Usage

```
getPlotConfig(id, configInputId)
```

Arguments

`id` Output variable to read from (id which references the requested plot).
`configInputId` Reactive input to write to.

Value

No return value, called from shiny applications for side effects.

Examples

```

## Not run:
if(interactive() && require(shiny)) {
  library(shiny)
  library(shinyjs)
  library(parallelPlot)

  ui <- fluidPage(
    useShinyjs(),
    p("Use button to save widget as an html file, reproducing its configuration"),
    actionButton("downloadButton", "Download Widget"),
    downloadButton("associatedDownloadButton", "Download Widget",
      style = "visibility: hidden;"
    ),
  ),

```

```

    parallelPlotOutput("parPlot")
  )

server <- function(input, output, session) {
  output$parPlot <- renderParallelPlot({
    parallelPlot(iris)
  })
  observeEvent(input$downloadButton, {
    parallelPlot::getPlotConfig("parPlot", "ConfigForDownload")
  })
  observeEvent(input$ConfigForDownload, {
    ppForDownload <- parallelPlot(
      data = iris,
      categorical = input$ConfigForDownload$categorical,
      categoriesRep = input$ConfigForDownload$categoriesRep,
      arrangeMethod = input$ConfigForDownload$arrangeMethod,
      inputColumns = input$ConfigForDownload$inputColumns,
      keptColumns = input$ConfigForDownload$keptColumns,
      histoVisibility = input$ConfigForDownload$histoVisibility,
      invertedAxes = input$ConfigForDownload$invertedAxes,
      cutoffs = input$ConfigForDownload$cutoffs,
      refRowIndex = input$ConfigForDownload$refRowIndex,
      refColumnDim = input$ConfigForDownload$refColumnDim,
      rotateTitle = input$ConfigForDownload$rotateTitle,
      columnLabels = input$ConfigForDownload$columnLabels,
      continuousCS = input$ConfigForDownload$continuousCS,
      categoricalCS = input$ConfigForDownload$categoricalCS,
      controlWidgets = NULL,
      cssRules = input$ConfigForDownload$cssRules,
      sliderPosition = input$ConfigForDownload$sliderPosition
    )
    shinyjs::runjs("document.getElementById('associatedDownloadButton').click();")
  })
  output$associatedDownloadButton <- downloadHandler(
    filename = function() {
      paste("parallelPlot-", Sys.Date(), ".html", sep = "")
    },
    content = function(tmpContentFile) {
      htmlwidgets::saveWidget(ppForDownload, tmpContentFile)
    }
  )
}

shinyApp(ui, server)
}

## End(Not run)

```

Description

Asks to retrieve the value of an attribute.

Usage

```
getValue(id, attrType, valueInputId)
```

Arguments

id	output variable to read from (id which references the requested plot)
attrType	which value is requested.
valueInputId	reactive input to write to.

Details

Available attributes are Cutoffs, SelectedTraces and ReferenceColumn. Result will be sent through a reactive input.

Value

No return value, called from shiny applications for side effects.

Examples

```
if(interactive() && require(shiny)) {
  library(shiny)
  library(parallelPlot)

  ui <- fluidPage(
    actionButton("getSelectedTracesAction", "Retrieve Selected Lines"),
    p("The button displays the list of uncutted rows (use brush to reduce it)"),
    parallelPlotOutput("parPlot")
  )

  server <- function(input, output, session) {
    output$parPlot <- renderParallelPlot({
      parallelPlot(iris)
    })
    observeEvent(input$getSelectedTracesAction, {
      attributeType <- "SelectedTraces"
      parallelPlot::getValue("parPlot", attributeType, "MySelectedTraces")
    })
    observeEvent(input$MySelectedTraces, {
      showModal(modalDialog(
        title = "Selected Lines",
        toString(input$MySelectedTraces)
      ))
    })
  }

  shinyApp(ui, server)
```

```
}
```

highlightRow

Row highlight

Description

Asks to change the highlighted row.

Usage

```
highlightRow(id, rowIndex)
```

Arguments

id	output variable to read from (id which references the requested plot)
rowIndex	index of the row to highlight; NULL means no row is to highlight.

Value

No return value, called from shiny applications for side effects.

Examples

```
if(interactive() && require(shiny)) {
  library(shiny)
  library(parallelPlot)

  ui <- fluidPage(
    actionButton("highlightRowAction", "Highlight Last Row"),
    actionButton("clearHlRowAction", "Remove Highlighting"),
    p("These buttons sets/unsets a selected line"),
    parallelPlotOutput("parPlot")
  )

  server <- function(input, output, session) {
    output$parPlot <- renderParallelPlot({
      parallelPlot(iris)
    })
    observeEvent(input$highlightRowAction, {
      lastRowIndex <- nrow(iris)
      parallelPlot::highlightRow("parPlot", lastRowIndex)
    })

    observeEvent(input$clearHlRowAction, {
      parallelPlot::highlightRow("parPlot", NULL)
    })
  }
}
```

```

    shinyApp(ui, server)
  }

```

parallelPlot

htmlwidget for d3.js parallel coordinate plot

Description

htmlwidget for d3.js parallel coordinate plot

Usage

```

parallelPlot(
  data,
  categorical = NULL,
  categoriesRep = "EquallySpacedLines",
  arrangeMethod = "fromRight",
  inputColumns = NULL,
  keptColumns = NULL,
  histoVisibility = NULL,
  invertedAxes = NULL,
  cutoffs = NULL,
  refRowIndex = NULL,
  refColumnDim = NULL,
  rotateTitle = FALSE,
  columnLabels = NULL,
  continuousCS = "Viridis",
  categoricalCS = "Category10",
  eventInputId = NULL,
  editionMode = "EditionOff",
  controlWidgets = FALSE,
  cssRules = NULL,
  sliderPosition = NULL,
  width = NULL,
  height = NULL,
  elementId = NULL
)

```

Arguments

<code>data</code>	<code>data.frame</code> with data to use in the chart.
<code>categorical</code>	List of list (one for each data column) containing the name of available categories, or <code>NULL</code> if column corresponds to continuous data; <code>NULL</code> is allowed, meaning all columns are continuous. A named list can also be provided to only indicate which columns are categorical, associating a column name to available categories.

categoriesRep	Within a category column, the height assigned to each category can either be: • equal for each category (EquallySizedBoxes); • or calculated to reflect the proportion of lines passing through each category (EquallySpacedLines).
arrangeMethod	Within a category box: • the position of lines can be calculated to minimize crossings on the left of the box (fromLeft); • the position of lines can be calculated to minimize crossings on the right (fromRight, default behavior); • lines can be split in two points to minimize crossings on the left and on the right (fromBoth). To turn this ordering off (for example for performance reasons), arrangeMethod can also be set to fromNone.
inputColumns	List of boolean (one for each data column), TRUE for an input column, FALSE for an output column; NULL is allowed, meaning all columns are inputs. A list of column names can also be provided to only indicate which columns are inputs.
keptColumns	List of boolean (one for each data column), FALSE if column has to be ignored; NULL is allowed, meaning all columns are available. A list of column names can also be provided to only indicate which columns are to be kept.
histoVisibility	List of boolean (one for each data column), TRUE if an histogram must be displayed; NULL is allowed, meaning no histogram must be displayed. A list of column names can also be provided to only indicate which columns must have an histogram displayed.
invertedAxes	List of boolean (one for each data column), TRUE if orientation of axis must be inverted; NULL is allowed, meaning no axis must be inverted. A list of column names can also be provided to only indicate which columns must have an inverted axis.
cutoffs	List of list (one for each data column) of list (one for each cutoff) containing two values (min and max values defining the cutoff) or NULL if there is no cutoff to apply; NULL is allowed, meaning all columns are without cutoff. A named list can also be provided to only indicate which columns have cutoffs, associating a column name to its cutoffs.
refRowIndex	Index of the sample row which has to appear horizontal; NULL is allowed, meaning there is no row to use as reference.
refColumnDim	Name of the reference column (used to determine the color to attribute to a row); NULL is allowed, meaning there is no coloring to apply.
rotateTitle	TRUE if column title must be rotated.
columnLabels	List of string (one for each data column) to display in place of column name found in data, or NULL if there is no alternative name; NULL is allowed, meaning all columns are without alternative name; can be used to insert line breaks.
continuousCS	Name of the color Scale to use for continuous data; supported names: Viridis, Inferno, Magma, Plasma, Warm, Cool, Rainbow, CubehelixDefault, Blues, Greens, Greys, Oranges, Purples, Reds, BuGn, BuPu, GnBu, OrRd, PuBuGn, PuBu, PuRd, RdBu, RdPu, YlGnBu, YlGn, YlOrBr, YlOrRd; default value is Viridis.

categoricalCS	Name of the color Scale to use for categorical data; supported names: Category10, Accent, Dark2, Paired, Set1; default value is Category10.
eventInputId	When plot event occurred, reactive input to write to; NULL is allowed, meaning no event is sent. An event is a list with two named elements 'type' and 'value'. • If type is equal to cutoffChange: – value\$adjusting is TRUE when pointer is moving, changing a cutoff; – value\$updatedDim is the name of last cut column; – value\$selectedTraces gives the indexes of uncut rows; – value\$cutoffs gives the new values for the cutoffs. • If type is equal to axeOrientationChange: – value\$invertedAxes has the same form than invertedAxes argument. • If type is equal to refColumnDimChange: – value\$refColumnDim is the new column to use as reference (see refColumnDim argument). • If type is equal to rowClicked: – value\$rowIndex is the index of the clicked row. • If type is equal to pointChange: – value\$dim defines the column of the edited point; – value\$rowIndex defines the row of the edited point; – value\$newValue gives the new value for the edited point.
editionMode	Supported edition modes: EditionOff, EditionOnDrag, EditionOnDragEnd; default value is EditionOff.
controlWidgets	Tells if some widgets must be available to control plot; NULL is allowed, meaning that !HTMLWidgets.shinyMode is to use; default value is FALSE.
cssRules	CSS rules to add. Must be a named list of the form list(selector = declarations), where selector is a valid CSS selector and declarations is a string or vector of declarations.
sliderPosition	Set initial position of slider, specifying which columns interval is visible. Default value is NULL which is equivalent to: <pre>list(dimCount = 8, startingDimIndex = 1)</pre>
width	Integer in pixels defining the width of the widget.
height	Integer in pixels defining the height of the widget.
elementId	Unique CSS selector id for the widget.

Value

An object of class `htmlwidget` that will intelligently print itself into HTML in a variety of contexts including the R console, within R Markdown documents, and within Shiny output bindings.

Examples

```
if(interactive()) {
  library(parallelPlot)

  categorical <-
    list(cyl = c(4, 6, 8), vs = c(0, 1), am = c(0, 1), gear = 3:5, carb = 1:8)
  parallelPlot(mtcars, categorical = categorical, refColumnDim = "cyl")
  # `cyl` and four last columns have a box representation for categories

  histoVisibility <- rep(TRUE, ncol(iris))
  parallelPlot(iris, histoVisibility = histoVisibility)
  # An histogram is displayed for each column

  histoVisibility <- names(iris) # Same as `rep(TRUE, ncol(iris))`
  cutoffs <- list(Sepal.Length = list(c(6, 7)), Species = c("virginica", "setosa"))
  parallelPlot(iris, histoVisibility = histoVisibility, cutoffs = cutoffs)
  # Cut lines are shaded;
  # an histogram for each column is displayed considering only kept lines

  parallelPlot(iris, refRowIndex = 1)
  # Axes are shifted vertically in such a way that first trace
  # of the dataset looks horizontal

  columnLabels <- gsub("\\.", "<br>", colnames(iris))
  parallelPlot(iris, refColumnDim = "Species", columnLabels = columnLabels)
  # Given names are displayed in place of dataset column names;
  # <br> is used to insert line breaks

  parallelPlot(iris, cssRules = list(
    "svg" = "background: #C2C2C2",
    ".tick text" = c("fill: red", "font-size: 1.8em")
  ))
  # Background of plot is grey and text of axes ticks is red and greater
}
```

parallelPlot-shiny *Shiny bindings for parallelPlot*

Description

Output and render functions for using parallelPlot within Shiny applications and interactive Rmd documents.

Usage

```
parallelPlotOutput(outputId, width = "100%", height = "600px")

renderParallelPlot(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

outputId	output variable to read from
width, height	Must be a valid CSS unit (like '100%', '400px', 'auto') or a number, which will be coerced to a string and have 'px' appended.
expr	An expression that generates a parallelPlot
env	The environment in which to evaluate expr.
quoted	Is expr a quoted expression (with quote())? This is useful if you want to save an expression in a variable.

Value

An output or render function that enables the use of the widget within Shiny applications.

setArrangeMethod	<i>Lines position</i>
------------------	-----------------------

Description

Within a category box:

- the position of lines can be calculated to minimize crossings on the left of the box (fromLeft);
- the position of lines can be calculated to minimize crossings on the right (fromRight, default behavior);
- lines can be split in two points to minimize crossings on the left and on the right (fromBoth). To turn this ordering off (for example for performance reasons), arrangeMethod can also be set to fromNone.

Usage

```
setArrangeMethod(id, arrangeMethod)
```

Arguments

id	Output variable to read from (id which references the requested plot).
arrangeMethod	One of the available arrange methods (fromLeft, fromRight, fromBoth, fromNone).

Value

No return value, called from shiny applications for side effects.

Examples

```

if(interactive() && require(shiny)) {
  library(shiny)
  library(parallelPlot)

  ui <- fluidPage(
    selectInput(
      "arrangeMethodSelect",
      "Arrange Method:",
      choices = list(
        "fromLeft" = "fromLeft", "fromRight" = "fromRight",
        "fromBoth" = "fromBoth", "fromNone" = "fromNone"
      ),
      selected = "fromRight"
    ),
    p("Selector controls the method used to arrange lines position in category boxes"),
    parallelPlotOutput("parPlot")
  )

  server <- function(input, output, session) {
    output$parPlot <- renderParallelPlot({
      categorical <-
        list(cyl = c(4, 6, 8), vs = c(0, 1), am = c(0, 1), gear = 3:5, carb = 1:8)
      parallelPlot(mtcars, categorical = categorical, refColumnDim = "cyl")
    })
    observeEvent(input$arrangeMethodSelect, {
      parallelPlot::setArrangeMethod("parPlot", input$arrangeMethodSelect)
    })
  }

  shinyApp(ui, server)
}

```

setCategoricalColorScale

Lines colors

Description

Tells which color scale to use when reference column is of type categorical.

Usage

```
setCategoricalColorScale(id, categoricalCsId)
```

Arguments

id output variable to read from (id which references the requested plot)

categoricalCsId one of the available color scale ids

Details

If a column is defined as the reference (for example by clicking on its header), a color scale is associated to this column. Available color scale ids are: Category10, Accent, Dark2, Paired, Set1.

Value

No return value, called from shiny applications for side effects.

Examples

```
if(interactive() && require(shiny)) {
  library(shiny)
  library(parallelPlot)

  ui <- fluidPage(
    selectInput("categoricalCsSelect", "Categorical Color Scale:",
      choices = list(
        "Category10" = "Category10", "Accent" = "Accent", "Dark2" = "Dark2",
        "Paired" = "Paired", "Set1" = "Set1"
      ),
      selected = "Category10"
    ),
    p("Selector controls used colors when reference column is of type categorical"),
    parallelPlotOutput("parPlot")
  )

  server <- function(input, output, session) {
    output$parPlot <- renderParallelPlot({
      parallelPlot(data = iris, refColumnDim = "Species")
    })
    observeEvent(input$categoricalCsSelect, {
      parallelPlot::setCategoricalColorScale("parPlot", input$categoricalCsSelect)
    })
  }

  shinyApp(ui, server)
}
```

setCategoriesRep

Categories Representation

Description

Within a category column, the height assigned to each category can either be:

- equal for each category (EquallySizedBoxes);
- or calculated to reflect the proportion of lines passing through each category (EquallySpacedLines).

Usage

```
setCategoriesRep(id, categoriesRep)
```

Arguments

id Output variable to read from (id which references the requested plot).

categoriesRep One of the available category representations (EquallySpacedLines, EquallySizedBoxes).

Value

No return value, called from shiny applications for side effects.

Examples

```
if(interactive() && require(shiny)) {
  library(shiny)
  library(parallelPlot)

  ui <- fluidPage(
    selectInput(
      "categoriesRepSelect",
      "Categories Representation:",
      choices = list(
        "EquallySpacedLines" = "EquallySpacedLines",
        "EquallySizedBoxes" = "EquallySizedBoxes"
      ),
      selected = "EquallySpacedLines"
    ),
    p("The selector controls the height assigned to each category"),
    parallelPlotOutput("parPlot")
  )

  server <- function(input, output, session) {
    output$parPlot <- renderParallelPlot({
      categorical <-
        list(cyl = c(4, 6, 8), vs = c(0, 1), am = c(0, 1), gear = 3:5, carb = 1:8)
      parallelPlot(mtcars, categorical = categorical, refColumnDim = "cyl")
    })
    observeEvent(input$categoriesRepSelect, {
      parallelPlot::setCategoriesRep("parPlot", input$categoriesRepSelect)
    })
  }

  shinyApp(ui, server)
}
```

setContinuousColorScale

Lines colors

Description

Tells which color scale to use when reference column is of type continuous.

Usage

```
setContinuousColorScale(id, continuousCsId)
```

Arguments

id	Output variable to read from (id which references the requested plot).
continuousCsId	One of the available color scale ids (Viridis, Inferno, Magma, Plasma, Warm, Cool, Rainbow, CubeHelixDefault, Blues, Greens, Greys, Oranges, Purples, Reds, BuGn, BuPu, GnBu, OrRd, PuBuGn, PuBu, PuRd, RdBu, RdPu, YlGnBu, YlGn, YlOrBr, YlOrRd).

Details

If a column is defined as the reference (for example by clicking on its header), a color scale is associated to this column. Available color scale ids are: Blues, RdBu, YlGnBu, YlOrRd, Reds.

Value

No return value, called from shiny applications for side effects.

Examples

```
if(interactive() && require(shiny)) {
  library(shiny)
  library(parallelPlot)

  ui <- fluidPage(
    selectInput(
      "continuousCsSelect",
      "Continuous Color Scale:",
      choices = list(
        "Viridis" = "Viridis", "Inferno" = "Inferno", "Magma" = "Magma",
        "Plasma" = "Plasma", "Warm" = "Warm", "Cool" = "Cool", "Rainbow" = "Rainbow",
        "CubeHelixDefault" = "CubeHelixDefault", "Blues" = "Blues",
        "Greens" = "Greens", "Greys" = "Greys", "Oranges" = "Oranges",
        "Purples" = "Purples", "Reds" = "Reds", "BuGn" = "BuGn", "BuPu" = "BuPu",
        "GnBu" = "GnBu", "OrRd" = "OrRd", "PuBuGn" = "PuBuGn", "PuBu" = "PuBu",
        "PuRd" = "PuRd", "RdBu" = "RdBu", "RdPu" = "RdPu", "YlGnBu" = "YlGnBu",
        "YlGn" = "YlGn", "YlOrBr" = "YlOrBr", "YlOrRd" = "YlOrRd"
      )
    ),
```

```

      selected = "Viridis"
    ),
    p("Selector controls used colors when reference column is of type continuous"),
    parallelPlotOutput("parPlot")
  )

  server <- function(input, output, session) {
    output$parPlot <- renderParallelPlot({
      parallelPlot(iris, refColumnDim = "Sepal.Length")
    })
    observeEvent(input$continuousCsSelect, {
      parallelPlot::setContinuousColorScale("parPlot", input$continuousCsSelect)
    })
  }

  shinyApp(ui, server)
}

```

setCutoffs

Cutoffs values

Description

Tells which cutoffs to use for each column.

Usage

```
setCutoffs(id, cutoffs)
```

Arguments

id	output variable to read from (id which references the requested plot)
cutoffs	Vector of list (one for each data column) of vector (one for each cutoff) containing two values for continuous input (min and max value defining the cutoff), or one value for categorical input (name of the category to keep), or NULL if there is no cutoff to apply; NULL is allowed, meaning all columns are without cutoff. A named list can also be provided to only indicate which columns must be assigned to a new cutoff.

Details

It's possible to filter some lines by defining cutoffs to apply to columns.

Value

No return value, called from shiny applications for side effects.

Examples

```

if(interactive() && require(shiny)) {
  library(shiny)
  library(parallelPlot)

  ui <- fluidPage(
    sliderInput("brushSlider", "Brush for 'Sepal.Length' column:",
      min = 4, max = 8, step = 0.1, value = c(4, 8)),
    p("The slider controls the rows which are kept by cutoff (others are shaded)",
      parallelPlotOutput("parPlot")
    )

    server <- function(input, output, session) {
      output$parPlot <- renderParallelPlot({
        parallelPlot(iris)
      })
      observeEvent(input$brushSlider, {
        cutoffs <- list()
        cutoffs["Sepal.Length"] <- list(list(input$brushSlider))
        parallelPlot::setCutoffs("parPlot", cutoffs)
      })
    }

    shinyApp(ui, server)
  }

```

setHistoVisibility	<i>Histograms visibility</i>
--------------------	------------------------------

Description

Tells which columns have to be displayed with histograms.

Usage

```
setHistoVisibility(id, histoVisibility)
```

Arguments

<code>id</code>	output variable to read from (id which references the requested plot)
<code>histoVisibility</code>	Vector of boolean (one for each data column), TRUE if an histogram must be displayed; NULL is allowed, meaning no histogram must be displayed. A named list can also be provided to only indicate which columns must be assigned to a new display.

Value

No return value, called from shiny applications for side effects.

Examples

```

if(interactive() && require(shiny)) {
  library(shiny)
  library(parallelPlot)

  ui <- fluidPage(
    checkboxInput("histCB", "Histogram Visibility", FALSE),
    p("The check box controls the visibility of histograms"),
    parallelPlotOutput("parPlot")
  )

  server <- function(input, output, session) {
    output$parPlot <- renderParallelPlot({
      parallelPlot(iris)
    })
    observeEvent(input$histCB, {
      histoVisibility <- rep(input$histCB, ncol(iris))
      parallelPlot::setHistoVisibility("parPlot", histoVisibility)
    })
  }

  shinyApp(ui, server)
}

```

setInvertedAxes	<i>Axis orientation</i>
-----------------	-------------------------

Description

Tells which axes have to be displayed with an inverted orientation.

Usage

```
setInvertedAxes(id, invertedAxes)
```

Arguments

id	output variable to read from (id which references the requested plot)
invertedAxes	Vector of boolean (one for each data column), TRUE if axis orientation must be inverted; NULL is allowed, meaning no axis must be inverted. A named list can also be provided to only indicate which axes must be assigned to a new orientation.

Value

No return value, called from shiny applications for side effects.

Examples

```

if(interactive() && require(shiny)) {
  library(shiny)
  library(parallelPlot)

  ui <- fluidPage(
    checkboxInput("orientationCB", "Axis orientation", FALSE),
    p("The check box controls the orientation of axes"),
    parallelPlotOutput("parPlot")
  )

  server <- function(input, output, session) {
    output$parPlot <- renderParallelPlot({
      parallelPlot(iris)
    })
    observeEvent(input$orientationCB, {
      invertedAxes <- rep(input$orientationCB, ncol(iris))
      parallelPlot::setInvertedAxes("parPlot", invertedAxes)
    })
  }

  shinyApp(ui, server)
}

```

setKeptColumns

Column visibility

Description

Tells which columns have to be visible.

Usage

```
setKeptColumns(id, keptColumns)
```

Arguments

id	output variable to read from (id which references the requested plot)
keptColumns	Vector of boolean (one for each data column), FALSE if column has to be hidden. A named list can also be provided to only indicate which columns must be assigned to a new visibility.

Value

No return value, called from shiny applications for side effects.

Examples

```
if(interactive() && require(shiny)) {
  library(shiny)
  library(parallelPlot)

  ui <- fluidPage(
    checkboxInput("hideColumnsCB", "Hide last columns", FALSE),
    p("The check box controls the visibility of the two last columns"),
    parallelPlotOutput("parPlot")
  )

  server <- function(input, output, session) {
    output$parPlot <- renderParallelPlot({
      parallelPlot(mtcars)
    })
    observeEvent(input$hideColumnsCB, {
      keptColumns <- vapply(
        1:ncol(mtcars),
        function(i) {
          return(iffelse(input$hideColumnsCB, ncol(mtcars) - i >= 2, TRUE))
        },
        logical(1)
      )
      parallelPlot::setKeptColumns("parPlot", keptColumns)
    })
  }

  shinyApp(ui, server)
}
```

setRefColumnDim

Line coloring

Description

Tells which column is used to determine the color to attribute to each row.

Usage

```
setRefColumnDim(id, dim)
```

Arguments

id	output variable to read from (id which references the requested plot)
dim	Name of the reference column (used to determine the color to attribute to a row); NULL is allowed, meaning there is no coloring to apply.

Value

No return value, called from shiny applications for side effects.

Examples

```
if(interactive() && require(shiny)) {
  library(shiny)
  library(parallelPlot)

  ui <- fluidPage(
    selectInput(
      "refColumnDimSelect",
      "Reference column:",
      choices = list(
        "None" = 1, "First" = 2, "Second" = 3
      ),
      selected = "None"
    ),
    p("Selector controls the column used to determine the color to attribute to rows"),
    parallelPlotOutput("parPlot")
  )

  server <- function(input, output, session) {
    output$parPlot <- renderParallelPlot({
      categorical <-
        list(cyl = c(4, 6, 8), vs = c(0, 1), am = c(0, 1), gear = 3:5, carb = 1:8)
      parallelPlot(mtcars, categorical = categorical)
    })
    observeEvent(input$refColumnDimSelect, {
      choice <- as.numeric(input$refColumnDimSelect)
      refColumnDim <- list(NULL, colnames(mtcars)[1], colnames(mtcars)[2])[[choice]]
      parallelPlot::setRefColumnDim("parPlot", refColumnDim)
    })
  }

  shinyApp(ui, server)
}
```

Index

`changeRow`, [2](#)

`getPlotConfig`, [3](#)
`getValue`, [4](#)

`highlightRow`, [6](#)

`parallelPlot`, [7](#)
`parallelPlot-shiny`, [10](#)
`parallelPlotOutput`
 `(parallelPlot-shiny)`, [10](#)

`renderParallelPlot`
 `(parallelPlot-shiny)`, [10](#)

`setArrangeMethod`, [11](#)
`setCategoricalColorScale`, [12](#)
`setCategoriesRep`, [13](#)
`setContinuousColorScale`, [15](#)
`setCutoffs`, [16](#)
`setHistoVisibility`, [17](#)
`setInvertedAxes`, [18](#)
`setKeptColumns`, [19](#)
`setRefColumnDim`, [20](#)