Package 'scatterPlotMatrix'

June 17, 2024

Title `htmlwidget` for a Scatter Plot Matrix

Version 0.3.0

Description Create a scatter plot matrix, using `htmlwidgets` package and `d3.js`.

URL https://ifpen-gitlab.appcollaboratif.fr/detocs/scatterplotmatrix

BugReports
https://ifpen-gitlab.appcollaboratif.fr/detocs/scatterplotmatrix/-/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]

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

Repository CRAN

Date/Publication 2024-06-17 13:00:07 UTC

Contents

`changeMouseMode` *Set mouse interaction type*

Description

Three types of mouse interactions are available (`tooltip`, `filter` or `zoom`).

Usage

```
changeMouseMode(id, interactionType)
```

Arguments

`id`	Output variable to read from (id which references the requested plot).
`interactionType`	Type of mouse interaction.

Value

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

Examples

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

  ui <- fluidPage(
    selectInput(
      "mouseMode",
      "Mouse Interactions:",
      c("Tooltip" = "tooltip", "Filter" = "filter", "Zoom" = "zoom")
    ),
    p("Selector controls type of mouse interactions with the scatterPlotMatrix"),
    scatterPlotMatrixOutput("spMatrix")
  )

  server <- function(input, output, session) {
    output$spMatrix <- renderScatterPlotMatrix({
      scatterPlotMatrix(iris)
    })
    observe({
```

```
    scatterPlotMatrix::changeMouseMode("spMatrix", input$mouseMode)
  })
}

shinyApp(ui, server)
}
```

`getPlotConfig` *Retrieve plot configuration*

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:
library(shiny)
library(shinyjs)
library(scatterPlotMatrix)

ui <- fluidPage(
  useShinyjs(),
  p("Button saves the widget as an html file, reproducing its configuration"),
  actionButton("downloadButton", "Download Widget"),
  downloadButton("associatedDownloadButton", "Download Widget",
    style = "visibility: hidden;"
  ),
  scatterPlotMatrixOutput("spMatrix", height = 960)
)

server <- function(input, output, session) {
  output$spMatrix <- renderScatterPlotMatrix({
    scatterPlotMatrix(iris)
  })
```

```
  observeEvent(input$downloadButton, {
    scatterPlotMatrix::getPlotConfig("spMatrix", "ConfigForDownload")
  })
  observeEvent(input$ConfigForDownload, {
    spmForDownload <<- scatterPlotMatrix(
      data = iris,
      categorical = input$ConfigForDownload$categorical,
      inputColumns = input$ConfigForDownload$inputColumns,
      cutoffs = input$ConfigForDownload$cutoffs,
      keptColumns = input$ConfigForDownload$keptColumns,
      zAxisDim = input$ConfigForDownload$zAxisDim,
      distribType = as.numeric(input$ConfigForDownload$distribType),
      regressionType = as.numeric(input$ConfigForDownload$regressionType),
      corrPlotType = input$ConfigForDownload$corrPlotType,
      corrPlotCS = input$ConfigForDownload$corrPlotCS,
      rotateTitle = input$ConfigForDownload$rotateTitle,
      columnLabels = input$ConfigForDownload$columnLabels,
      continuousCS = input$ConfigForDownload$continuousCS,
      categoricalCS = input$ConfigForDownload$categoricalCS,
      mouseMode = input$ConfigForDownload$mouseMode,
      controlWidgets = NULL,
      cssRules = input$ConfigForDownload$cssRules,
      plotProperties = input$ConfigForDownload$plotProperties,
      slidersPosition = input$ConfigForDownload$slidersPosition
    )
    shinyjs::runjs("document.getElementById('associatedDownloadButton').click();")
  })
  output$associatedDownloadButton <- downloadHandler(
    filename = function() {
      paste("scatterPlotMatrix-", Sys.Date(), ".html", sep = "")
    },
    content = function(tmpContentFile) {
      htmlwidgets::saveWidget(spmForDownload, tmpContentFile)
    }
  )
}

shinyApp(ui, server)

## End(Not run)
```

---

highlightPoint *Row highlight*

Description

Asks to change the highlighted row.

Usage

```
highlightPoint(id, pointIndex)
```

Arguments

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

Value

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

Examples

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

  ui <- fluidPage(
    actionButton("highlightPointAction", "Highlight Last Point"),
    actionButton("clearHlPointAction", "Remove Highlighting"),
    p("These buttons sets/unsets a selected line"),
    scatterPlotMatrixOutput("spMatrix")
  )

  server <- function(input, output, session) {
    output$spMatrix <- renderScatterPlotMatrix({
      scatterPlotMatrix(iris)
    })
    observeEvent(input$highlightPointAction, {
      lastRowIndex <- nrow(iris)
      scatterPlotMatrix::highlightPoint("spMatrix", lastRowIndex)
    })

    observeEvent(input$clearHlPointAction, {
      scatterPlotMatrix::highlightPoint("spMatrix", NULL)
    })
  }

  shinyApp(ui, server)
}
```

---

scatterPlotMatrix htmlwidget *for* d3.js *scatter plot matrix*

Description

htmlwidget for d3.js scatter plot matrix

Usage

```
scatterPlotMatrix(
  data,
  categorical = NULL,
  inputColumns = NULL,
  cutoffs = NULL,
  keptColumns = NULL,
  zAxisDim = NULL,
  distribType = 2,
  regressionType = 0,
  corrPlotType = "Circles",
  corrPlotCS = NULL,
  rotateTitle = FALSE,
  columnLabels = NULL,
  continuousCS = "Viridis",
  categoricalCS = "Category10",
  mouseMode = "tooltip",
  eventInputId = NULL,
  controlWidgets = FALSE,
  cssRules = NULL,
  plotProperties = NULL,
  slidersPosition = NULL,
  width = NULL,
  height = NULL,
  elementId = NULL
)
```

Arguments

data	data.frame with data to use in the chart.
categorical	List of list (one for each data column) containing the name of available categories, or NULL if column corresponds to continuous data; NULL is allowed, meaning all columns are continuous. A named list can also be provided to only indicate which columns are categorical, associating available categories to a column name.
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.
cutoffs	List of SpCutoff; a SpCutoff is a list defining a xDim, yDim and a list of xyCutoff; a xyCutoff is a pair of cutoff (one for x axis, one for y axis); a cutoff is a list containing two values (min and max values) or NULL if there is no cutoff to apply for this axis; NULL is allowed, meaning there is no cutoff to apply.
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.

`zAxisDim` Name of the column represented by z axis (used to determine the color to attribute to a point); `NULL` is allowed, meaning there is no coloring to apply.

`distribType` Binary code indicating the type of distribution plot (bit 1: density plot, bit 2: histogram).

`regressionType` Binary code indicating the type of regression plot (bit 1: linear, bit 2: loess).

`corrPlotType` String indicating the type of correlation plots to use. Supported values:

- `Circles` to use a circle tree map;
- `Text` to display values of correlation as colored text labels (color scale domain is `[-1; 1]`);
- `AbsText` to display values of correlation as colored text labels (color scale domain is `[0; 1]`, absolute value of correlations is used);
- `Empty` to not display values of correlation; default value is `Circles`.

`corrPlotCS` Name of the color Scale to use for correlation plot when plot type is `Text` or `AbsText`; 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 `NULL`, which corresponds to `RdBu` if `corrPlotType` is `Text`, or `Blues` if `corrPlotType` is `AbsText`.

`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`.

`mouseMode` Type of mouse interaction. Three types are available: `tooltip`, `filter` or `zoom`.

`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 `zAxisChange`:
  - `value` is the new column to use as reference (see `zAxisDim` argument).
- If `type` is equal to `cutoffChange`:
  - `value$adjusting` is `TRUE` when pointer is moving, changing a cutoff;
  - `value$cutoffs` gives the new values for the cutoffs.
- If `type` is equal to `pointClicked`:
  - `value$pointIndex` is the index of the clicked point.

`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.

`plotProperties` Adjust some properties which can not be set through CSS (mainly size, color and opacity of points). Default value is `NULL` which is equivalent to:

```
list(
  noCatColor = "#43665e",
  watermarkColor = "#ddd",
  point = list(
    alpha = 0.5,
    radius = 2
  ),
  regression = list(
    strokeWidth = 4
  )
)
```

`slidersPosition` Set initial position of sliders, specifying which columns intervals are visible. Default value is `NULL` which is equivalent to:

```
list(
  dimCount = 8,
  xStartingDimIndex = 1,
  yStartingDimIndex = 1
)
```

`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.

Examples

```
if(interactive()) {
  library(scatterPlotMatrix)

  scatterPlotMatrix(iris, zAxisDim = "Species")
  # Each point has a color depending of its 'Species' value

  categorical <-
    list(cyl = c(4, 6, 8), vs = c(0, 1), am = c(0, 1), gear = 3:5, carb = 1:8)
  scatterPlotMatrix(mtcars, categorical = categorical, zAxisDim = "cyl")
  # 'cyl' and four last columns have a box representation for categories
  # (use top slider to see the last three columns)

  scatterPlotMatrix(iris, zAxisDim = "Species", distribType = 1)
  # Distribution plots are of type 'density plot' (instead of histogram)

  scatterPlotMatrix(iris, zAxisDim = "Species", regressionType = 1)
  # Add linear regression plots

  columnLabels <- gsub("\\.", "<br>", colnames(iris))
  scatterPlotMatrix(iris, zAxisDim = "Species", columnLabels = columnLabels)
```

```
  # Given names are displayed in place of dataset column names;
  # <br> is used to insert line breaks

  scatterPlotMatrix(iris, cssRules = list(
      ".jitterZone" = "fill: pink",
      ".tick text" = c("fill: red", "font-size: 1.8em")
  ))
  # Background of plot is pink and text of axes ticks is red and greater

  scatterPlotMatrix(iris, plotProperties = list(
    noCatColor = "DarkCyan",
    point = list(
      alpha = 0.3,
      radius = 4
    )
  ))
  # Points of plots are different:
  # two times greater, with opacity reduced from 0.5 to 0.3, and a `DarkCyan` color

}
```

`scatterPlotMatrix-shiny` *Shiny bindings for scatterPlotMatrix*

Description

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

Usage

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

renderScatterPlotMatrix(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 scatterPlotMatrix
`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.

setCategoricalColorScale *Categorical color scale*

Description

Tells which color scale to use when the Z axis is set to a categorical column.

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 (Category10, Accent, Dark2, Paired, Set1).

Value

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

Examples

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

  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"),
    scatterPlotMatrixOutput("spMatrix")
  )

  server <- function(input, output, session) {
    output$spMatrix <- renderScatterPlotMatrix({
      scatterPlotMatrix(iris, zAxisDim = "Species")
    })
    observeEvent(input$categoricalCsSelect, {
     scatterPlotMatrix::setCategoricalColorScale("spMatrix", input$categoricalCsSelect)
    })
```

```
  }

  shinyApp(ui, server)
}
```

`setContinuousColorScale` *Continuous color scale*

Description

Tells which color scale to use when the Z axis is set to a continuous column.

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`).

Value

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

Examples

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

  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"),
    scatterPlotMatrixOutput("spMatrix")
  )

  server <- function(input, output, session) {
    output$spMatrix <- renderScatterPlotMatrix({
      scatterPlotMatrix(iris, zAxisDim = "Sepal.Length")
    })
    observeEvent(input$continuousCsSelect, {
      scatterPlotMatrix::setContinuousColorScale("spMatrix", input$continuousCsSelect)
    })
  }

  shinyApp(ui, server)
}
```

`setCorrPlotCS` *Color scale for correlation plots*

Description

Tells which color scale to use for correlation plots (only used when plot type is `Text` or `AbsText`).

Usage

```
setCorrPlotCS(id, corrPlotCsId)
```

Arguments

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

Value

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

Examples

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

```
  ui <- fluidPage(
    selectInput(
      "corrPlotCsSelect",
      "Correlation Plot 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 = "Plasma"
    ),
    p("The selector controls the color scale to use for correlation plot
       when plot type is 'Text' or 'AbsText'"),
    scatterPlotMatrixOutput("spMatrix")
  )

  server <- function(input, output, session) {
    output$spMatrix <- renderScatterPlotMatrix({
      scatterPlotMatrix(iris, corrPlotType = "Text")
    })
    observeEvent(input$corrPlotCsSelect, {
      scatterPlotMatrix::setCorrPlotCS("spMatrix", input$corrPlotCsSelect)
    })
  }

  shinyApp(ui, server)
}
```

`setCorrPlotType` *Correlation plot type*

Description

Tells which type of correlation plot to use.

Usage

```
setCorrPlotType(id, corrPlotType)
```

Arguments

`id`	Output variable to read from (id which references the requested plot).
`corrPlotType`	One of the available correlation plot types (`Empty`, `Circles`, `Text`, `AbsText`).

Value

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

Examples

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

  ui <- fluidPage(
    selectInput(
      "corrPlotTypeSelect",
      "Correlation Plot Type:",
      choices = list(
        "Empty" = "Empty",
        "Circles" = "Circles",
        "Text" = "Text",
        "AbsText" = "AbsText"
      ),
      selected = "Circles"
    ),
    p("The selector controls the type of correlation to use"),
    scatterPlotMatrixOutput("spMatrix")
  )

  server <- function(input, output, session) {
    output$spMatrix <- renderScatterPlotMatrix({
      scatterPlotMatrix(iris, zAxisDim = "Sepal.Length", continuousCS = "Plasma")
    })
    observeEvent(input$corrPlotTypeSelect, {
      scatterPlotMatrix::setCorrPlotType("spMatrix", input$corrPlotTypeSelect)
    })
  }

  shinyApp(ui, server)
}
```

`setCutoffs` *Cutoffs values*

Description

Tells which cutoffs to use for each pair of columns. It's possible to filter some points by defining cutoffs to apply to columns.

Usage

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

Arguments

id	output variable to read from (id which references the requested plot)
cutoffs	List of SpCutoff; a SpCutoff is a list defining a xDim, yDim and a list of xyCutoff; a xyCutoff is a pair of cutoff (one for x axis, one for y axis); a cutoff is a list containing two values (min and max values) or NULL if there is no cutoff to apply for this axis; NULL is allowed, meaning there is no cutoff to apply.

Value

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

Examples

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

  ui <- fluidPage(
    checkboxInput("setosaCB", "Setosa", TRUE),
    checkboxInput("versicolorCB", "Versicolor", TRUE),
    checkboxInput("viginicaCB", "Viginica", TRUE),
    scatterPlotMatrixOutput("spMatrix")
  )

  server <- function(input, output, session) {
    output$spMatrix <- renderScatterPlotMatrix({
      scatterPlotMatrix(
        data = iris,
        zAxisDim = "Species"
      )
    })

    observe({
      speciesCBs = c(input$setosaCB, input$versicolorCB, input$viginicaCB)
      toKeepIndexes <- Filter(function(i) speciesCBs[i], 1:length(speciesCBs))
      xyCutoffs <- sapply(toKeepIndexes, function(i) {
        list(list(NULL, c(i - 1.1, i - 0.9)))
      })
      scatterPlotMatrix::setCutoffs("spMatrix", list(
        list(xDim="Sepal.Length", yDim="Species", xyCutoffs = xyCutoffs)
      ))
    })
  }
  shinyApp(ui, server)
}
```

setDistribType *Distribution plots*

Description

Tells which type of representation to use for distribution plots.

Usage

```
setDistribType(id, distribType)
```

Arguments

id	Output variable to read from (id which references the requested plot).
distribType	Binary code indicating the type of distribution plot (bit 1: histogram, bit 2: density plot).

Value

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

Examples

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

  ui <- fluidPage(
    selectInput(
      "distribType",
      "Distribution Representation:",
      choices = list("Histogram" = 2, "Density Plot" = 1),
      selected = 2
    ),
    p("The selector controls type of representation to use for distribution plots"),
    scatterPlotMatrixOutput("spMatrix")
  )

  server <- function(input, output, session) {
    output$spMatrix <- renderScatterPlotMatrix({
      scatterPlotMatrix(iris)
    })
    observeEvent(input$distribType, {
      scatterPlotMatrix::setDistribType("spMatrix", input$distribType)
    })
  }

  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(scatterPlotMatrix)

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

  server <- function(input, output, session) {
    output$spMatrix <- renderScatterPlotMatrix({
      scatterPlotMatrix(iris)
    })
    observeEvent(input$hideColumnsCB, {
      keptColumns <- vapply(
        1:ncol(iris),
        function(i) {
          return(ifelse(input$hideColumnsCB, ncol(iris) - i >= 2, TRUE))
        },
        logical(1)
      )
      scatterPlotMatrix::setKeptColumns("spMatrix", keptColumns)
    })
  }
```

```
  shinyApp(ui, server)
}
```

`setRegressionType` *Regression plots*

Description

Tells which type of regression to use for regression plots.

Usage

```
setRegressionType(id, regressionType)
```

Arguments

`id`	Output variable to read from (id which references the requested plot).
`regressionType`	Binary code indicating the type of regression plot (bit 1: linear, bit 2: loess).

Value

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

Examples

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

  ui <- fluidPage(
    checkboxInput("linearRegressionCB", "Linear Regression", FALSE),
    checkboxInput("loessCB", "Local Polynomial Regression", FALSE),
    p("The chech boxes controls type of regression to use for regression plots"),
    scatterPlotMatrixOutput("spMatrix")
  )

  server <- function(input, output, session) {
    output$spMatrix <- renderScatterPlotMatrix({
      scatterPlotMatrix(iris)
    })
    observe({
      linearFlag <- ifelse(input$linearRegressionCB, 1, 0)
      loessFlag <- ifelse(input$loessCB, 2, 0)
      scatterPlotMatrix::setRegressionType("spMatrix", linearFlag + loessFlag)
    })
  }
```

```
  shinyApp(ui, server)
}
```

---

setZAxis *Z axis*

Description

Tells which column to use as reference to determine color of each points.

Usage

```
setZAxis(id, dim)
```

Arguments

id	Output variable to read from (id which references the requested plot).
dim	name of the column to use as reference.

Value

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

Examples

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

  ui <- fluidPage(
    fluidRow(
      column(
        2,
        selectInput("zAxisSelect", "Z Axis:", colnames(iris))
      ),
      column(
        2,
        checkboxInput("zAxisUsedCB", "Use Z Axis", FALSE)
      )
    ),
    scatterPlotMatrixOutput("spMatrix")
  )

  server <- function(input, output, session) {
    output$spMatrix <- renderScatterPlotMatrix({
      scatterPlotMatrix(iris)
    })

    observe({
```

```
      scatterPlotMatrix::setZAxis(
        "spMatrix",
        if (input$zAxisUsedCB) input$zAxisSelect else NULL
      )
    })
  }

  shinyApp(ui, server)
}
```

Index