Package: dataui (via r-universe)

September 12, 2024

Type Package

Title `Data-UI` Interactive Visualizations

Version 0.0.1

Date 2020-06-26

Maintainer Kent Russell <kent.russell@timelyportfolio.com>

URL https://github.com/timelyportfolio/dataui

BugReports https://github.com/timelyportfolio/dataui/issues

Description Create sparklines and histograms with 'data-ui'.

License MIT + file LICENSE

LazyData TRUE

Imports glue, htmltools, htmlwidgets, jsonlite, reactR

Suggests knitr, markdown, reactable, rmarkdown

Enhances DT, shiny

RoxygenNote 7.1.1

VignetteBuilder knitr

Repository https://timelyportfolio.r-universe.dev

RemoteUrl https://github.com/timelyportfolio/dataui

RemoteRef master

RemoteSha 39583c661edddb44214a269aed5393ba91b2789d

Contents

`dataui-shiny` *Shiny Bindings for 'dataui'*

Description

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

Usage

```
dataui0utput(outputId, width = "100%", height = "400px")

renderDataui(expr, env = parent.frame(), quoted = FALSE)

dataui_html(id, style, class, ...)
```

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 dataui
`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.

dui_add_deps *Add 'data-ui' Dependencies to Tag or 'htmlwidget'*

Description

Add 'data-ui' Dependencies to Tag or 'htmlwidget'

Usage

```
dui_add_deps(tag_htmlwidget = NULL)
```

Arguments

tag_htmlwidget	shiny.tag or htmlwidget to which you would like to add data-ui dependencies

Value

shiny.tag or htmlwidget

See Also

dui_chr

Examples

```
library(htmltools)

# not that you would want to do this but illustrates the usage of helper functions
browsable(
  dui_add_deps(
    tags$div(
      dui_chr(
        dui_sparkline(
          data = runif(100),
          components = list(
            dui_sparklineseries(strokeDasharray = "3,3"),
            dui_sparkpointseries(points = list("min","max"))
          )
        )
      )
    )
  )
)


library(dplyr)
library(htmltools)
library(dataui)
```

```
data("diamonds", package="ggplot2")

hist_all <- hist(diamonds$price, breaks = 20, plot = FALSE)

dat <- diamonds %>%
  group_by(cut) %>%
  summarize(
    n = n(),
    mean_price = mean(price),
    hist = list(unclass(hist(price, breaks = hist_all$breaks, plot = FALSE)))
  )

# calculate max so all sparklines use same scale
max_density <- max(unlist(lapply(dat$hist, function(x) {x$density})))

# convert hist column to character version of data-ui htmlwidget
dat <- dat %>%
  mutate(hist = lapply(hist, function(h){
    dui_chr(
      dui_sparkline(
        data = h$density,
        max = max_density,
        height = 80,
        margin = list(top = 0, bottom = 0, left = 20, right = 0),
        components = list(
          dui_sparkbarseries(),
          dui_tooltip(
            list(
              dui_sparkhorizontalrefline(
                stroke = "#ccc",
                strokeDasharray = "4,4"
              ),
              dui_sparkpointseries(
            renderLabel = htmlwidgets::JS("(d) => d ? (d*100000).toFixed(2) + '%' : null"),
                labelPosition = "right"
              )
            )
          )
        )
      )
    )
  }))

# datatable
library(DT)
DT::datatable(
  dat,
  escape = FALSE,
  options = list(
    fnDrawCallback = htmlwidgets::JS('function(){ HTMLWidgets.staticRender()}')
  )
) %>%
```

```
  formatCurrency("mean_price", digits = 0) %>%
  formatRound("n", digits = 0) %>%
  dui_add_deps()
```

`dui_add_reactable_dep` *Add 'dataui' Dependency to 'reactable'*

Description

Add 'dataui' Dependency to 'reactable'

Usage

```
dui_add_reactable_dep(rt)
```

Arguments

`rt`	`reactable` htmlwidget

Value

`reactable` htmlwidget with 'dataui' dependency attached

Examples

```
if(requireNamespace("reactable")) {
   dui_add_reactable_dep(reactable::reactable(mtcars))
}
```

`dui_barseries` *Bars for Histograms*

Description

Bars for Histograms

Usage

```
dui_barseries(rawData = NULL, ...)
```

Arguments

`rawData`	`vector`, `list`, `data.frame` of data
`...`	see data-ui series

Value

`reactR component` which is a special form of `htmltools::tag`

`dui_chr` *Character Version of 'data-ui'*

Description

Create a character version of interactive 'data-ui' for use with other 'htmlwidgets', such as tables or tags.

Usage

```
dui_chr(dui)
```

Arguments

`dui`	`data-ui` htmlwidget to convert to character

See Also

`dui_add_deps`

Examples

```
library(htmltools)

# not that you would want to do this but illustrates the usage of helper functions
browsable(
  dui_add_deps(
    tags$div(
      dui_chr(
        dui_sparkline(
          data = runif(100),
          components = list(
            dui_sparklineseries(strokeDasharray = "3,3"),
            dui_sparkpointseries(points = list("min","max"))
          )
        )
      )
    )
  )
)


library(dplyr)
library(htmltools)
library(dataui)
```

```
data("diamonds", package="ggplot2")

hist_all <- hist(diamonds$price, breaks = 20, plot = FALSE)

dat <- diamonds %>%
  group_by(cut) %>%
  summarize(
    n = n(),
    mean_price = mean(price),
    hist = list(unclass(hist(price, breaks = hist_all$breaks, plot = FALSE)))
  )

# calculate max so all sparklines use same scale
max_density <- max(unlist(lapply(dat$hist, function(x) {x$density})))

# convert hist column to character version of data-ui htmlwidget
dat <- dat %>%
  mutate(hist = lapply(hist, function(h){
    dui_chr(
      dui_sparkline(
        data = h$density,
        max = max_density,
        height = 80,
        margin = list(top = 0, bottom = 0, left = 20, right = 0),
        components = list(
          dui_sparkbarseries(),
          dui_tooltip(
            list(
              dui_sparkhorizontalrefline(
                stroke = "#ccc",
                strokeDasharray = "4,4"
              ),
              dui_sparkpointseries(
            renderLabel = htmlwidgets::JS("(d) => d ? (d*100000).toFixed(2) + '%' : null"),
                labelPosition = "right"
              )
            )
          )
        )
      )
    )
  }))

# datatable
library(DT)
DT::datatable(
  dat,
  escape = FALSE,
  options = list(
    fnDrawCallback = htmlwidgets::JS('function(){ HTMLWidgets.staticRender()}')
  )
) %>%
  formatCurrency("mean_price", digits = 0) %>%
```

```
  formatRound("n", digits = 0) %>%
  dui_add_deps()
```

`dui_densityseries` *Density Line Series for Histograms*

Description

Density Line Series for Histograms

Usage

```
dui_densityseries(rawData = NULL, ...)
```

Arguments

`rawData`	`vector`, `list`, `data.frame` of data
`...`	see data-ui series

Value

`reactR component` which is a special form of `htmltools::tag`

`dui_for_reactable` *Convert 'dataui' to 'reactable' Custom Render Function*

Description

Convert 'dataui' to 'reactable' Custom Render Function

Usage

```
dui_for_reactable(dui, jsarg = "cellInfo")
```

Arguments

`dui`	`dataui` htmlwidget to convert

Value

`htmlwidgets::JS` of class `JS_EVAL`

Examples

```
if(requireNamespace("reactable")) {
  # create data frame with a list column
  df <- data.frame(x=1)
  df$x[1] <- list(1:10)
  dui_add_reactable_dep(
    reactable::reactable(
      df,
      columns = list(
        x = reactable::colDef(
          cell = dui_for_reactable(dui_sparkline(
            data = htmlwidgets::JS("cellInfo.value"),
            components=list( dui_sparklineseries() )
          )
        ))
      )
    )
  )
}
```

`dui_histogram` *'data-ui' Histogram*

Description

Create interactive histogram visualizations with 'data-ui'. The histogram will perform the calculations in 'JavaScript' if the data is raw. If you would like more control over the calculation, then you can pass pre-binned values with help from `hist_to_binned_data`. `dui_histogram` works well as a full-featured visualization or can also be used as a 'sparkline' in smaller contexts.

Usage

```
dui_histogram(
  rawData = NULL,
  binCount = NULL,
  binType = NULL,
  binValues = NULL,
  cumulative = NULL,
  horizontal = NULL,
  limits = NULL,
  margin = NULL,
  normalized = NULL,
  renderTooltip = NULL,
  valueAccessor = NULL,
  onMouseMove = NULL,
  onMouseLeave = NULL,
  tooltipData = NULL,
  ariaLabel = NULL,
```

```
  components = NULL,
  width = 600,
  height = 400,
  elementId = NULL
)
```

Arguments

binCount	numeric specifying the approximate number of bins to calculate.
binType	character one of 'numeric'(default) or 'categorical'.
binValues	numeric vector of the bin or break values to override the automatic calculations.
cumulative	logical to specify whether or not the histogram will display cumulative sums of the counts.
horizontal	logical with TRUE meaning the chart will be in horizontal layout.
limits	numeric vector of length two to give a range for which values will be ignored if they are outside of the range.
margin	list of the form list(top =, right =, bottom =, left=) that will specify the margins for the 'sparkline' chart.
normalized	logical specifying whether or not to the values will be calculated as a percent of total.
renderTooltip	htmlwidget::JS function that will provide the 'React' element to render when a user moves their mouse over the visualization. The function should follow the signature ({ event, data, datum, color }) =>. If the function returns a falsy value then nothing will be rendered.
valueAccessor	htmlwidgets::JS function to let the chart know where to look for the y value in the data. An example would look like (d) => d.yval where yval is the property containing the value.
onMouseMove, onMouseLeave	htmlwidgets::JS function to run on mouse events.
tooltipData	currently not supported.
ariaLabel	character accessibility label for the chart
components	list of children (series or reference lines) to include in the histogram. Multiple components should be wrapped in list such as components = list(dui_densityseries, dui_barseries()).
width, height	numeric valid 'css' size unit. For height, this should always be numeric, but width might be something like '100%' or '50vw'.
elementId	character valid 'css' identifier for the htmlwidget container.

Value

react htmlwidget

dui_sparkbandline *Band Lines for 'Sparklines'*

Description

Band Lines for 'Sparklines'

Usage

```
dui_sparkbandline(...)
```

Arguments

...	see data-ui reference lines and bands

Value

`reactR component` which is a special form of `htmltools::tag`

dui_sparkbarseries *Bar Series for 'Sparklines'*

Description

Bar Series for 'Sparklines'

Usage

```
dui_sparkbarseries(...)
```

Arguments

...	see data-ui series

Value

`reactR component` which is a special form of `htmltools::tag`

dui_sparkhorizontalrefline *Horizontal Reference Line for 'Sparklines'*

Description

Horizontal Reference Line for 'Sparklines'

Usage

```
dui_sparkhorizontalrefline(...)
```

Arguments

...	see data-ui reference lines and bands

Value

`reactR component` which is a special form of `htmltools::tag`

dui_sparklabel *Labels for 'Sparklines'*

Description

Labels for 'Sparklines'

Usage

```
dui_sparklabel(...)
```

Arguments

...	see data-ui label

Value

`reactR component` which is a special form of `htmltools::tag`

dui_sparkline *'data-ui Sparklines'*

Description

Create interactive 'sparklines' visualizations for use as a standalone component or in combination with 'reactable' or other table libraries. There are a wide variety of components for use with `dui_sparkline`, including lines, bars, points, reference lines, and labels. These 'sparklines' automatically include responsive and tooltip behaviors.

Usage

```
dui_sparkline(
  data = NULL,
  className = NULL,
  margin = NULL,
  max = NULL,
  min = NULL,
  onMouseMove = NULL,
  onMouseLeave = NULL,
  renderTooltip = NULL,
  preserveAspectRatio = NULL,
  valueAccessor = NULL,
  viewBox = NULL,
  ariaLabel = NULL,
  components = list(),
  responsive = TRUE,
  width = "100%",
  height = 100,
  elementId = NULL
)
```

Arguments

`data`	`vector`, `list`, `data.frame` of data that will be passed down to the component series.
`className`	`character` 'css' class name to be added to the 'sparkline'.
`margin`	`list` of the form `list(top =, right =, bottom =, left=)` that will specify the margins for the 'sparkline' chart.
`max, min`	`numeric` value that will be the maximum/minimum for the 'y' axis/scale. This is useful if you have multiple 'sparklines' that you want to all use the same scale.
`onMouseMove, onMouseLeave`	`htmlwidgets::JS` function to run on mouse events. These are automatically assigned with 'sparkline' `htmlwidgets` so it is unlikely that you will want to override the default `NULL`.

renderTooltip	htmlwidget::JS function that will provide the 'React' element to render when a user moves their mouse over the visualization. The function should follow the signature ({ event, data, datum, color }) =>. If the function returns a falsy value then nothing will be rendered.
preserveAspectRatio	character to specify this attribute on the svg chart container. See preserveAspectRatio.
valueAccessor	htmlwidgets::JS function to let the chart know where to look for the y value in the data. An example would look like (d) => d.yval where yval is the property containing the value.
viewBox	character to specify this attribute on the svg. See viewBox.
ariaLabel	character label for accessibility for screen readers
components	list of children (series or reference lines) to include in the sparkline. Multiple components should be wrapped in list such as components = list(dui_sparklineseries, dui_sparkbarseries()). dui_tooltip component should also contain a similar list in effect forming a tree.
responsive	logical with default as TRUE. This argument is not passed on to the React component as a prop. Rather it is used in the R constructor to choose our SparklineResponsive component (TRUE) or SparklineWithTooltip (FALSE) component. Since height can be unstable and finicky, we only make width responsive.
width, height	numeric valid 'css' size unit. For height, this should always be numeric, but width might be something like '100%' or '50vw'.
elementId	character valid 'css' identifier for the htmlwidget container.

Value

react htmlwidget

`dui_sparklineargradient` *Linear Gradient for 'Sparklines'*

Description

Linear Gradient for 'Sparklines'

Usage

```
dui_sparklineargradient(...)
```

Arguments

...	see data-ui patterns and gradients

Value

reactR component which is a special form of htmltools::tag

---

dui_sparklineseries *Line Series for 'Sparklines'*

Description

Line Series for 'Sparklines'

Usage

```
dui_sparklineseries(...)
```

Arguments

...	see data-ui series

Value

reactR component which is a special form of htmltools::tag

---

dui_sparkpatternlines *Pattern Fill for 'Sparklines'*

Description

Pattern Fill for 'Sparklines'

Usage

```
dui_sparkpatternlines(...)
```

Arguments

...	see data-ui patterns and gradients

Value

reactR component which is a special form of htmltools::tag

dui_sparkpointseries *Point Series for 'Sparklines'*

Description

Point Series for 'Sparklines'

Usage

```
dui_sparkpointseries(...)
```

Arguments

...	see data-ui series

Value

reactR component which is a special form of htmltools::tag

dui_sparkrefline *Horizontal Reference Line for 'Sparklines'*

Description

Horizontal Reference Line for 'Sparklines'

Usage

```
dui_sparkrefline(...)
```

Arguments

...	see data-ui reference lines and bands

Value

reactR component which is a special form of htmltools::tag

dui_sparkverticalrefline *Vertical Reference Line for 'Sparklines'*

Description

Vertical Reference Line for 'Sparklines'

Usage

```
dui_sparkverticalrefline(...)
```

Arguments

...	see data-ui reference lines and bands

Value

`reactR component` which is a special form of `htmltools::tag`

dui_tooltip *Tooltip Container for 'Sparklines'*

Description

Tooltip Container for 'Sparklines'

Usage

```
dui_tooltip(components)
```

Arguments

components	`list` of children (series or reference lines) to include in the sparkline. Multiple components should be wrapped in `list` such as `components = list(dui_sparklineseries, dui_sparkverticalrefline())`.

Value

`reactR component` which is a special form of `htmltools::tag`

dui_xaxis *'X' Axis for Histograms*

Description

'X' Axis for Histograms

Usage

```
dui_xaxis(...)
```

Arguments

...	see data-ui axis

Value

`reactR component` which is a special form of `htmltools::tag`

dui_yaxis *'Y' Axis for Histograms*

Description

'Y' Axis for Histograms

Usage

```
dui_yaxis(...)
```

Arguments

...	see data-ui axis

Value

`reactR component` which is a special form of `htmltools::tag`

hist_to_binned_data *Convert Histogram to Binned Format*

Description

This is a convenience function to convert a histogram object from `hist` into the binned value format expected by 'data-ui' `dui_barseries` and `dui_densityseries` argument `binnedData`.

Usage

```
hist_to_binned_data(h = NULL, density = TRUE)
```

Arguments

`h`	histogram object or list from `hist`
`density`	`logical` to indicate the use of `density` or `count`

Value

`list`

Examples

```
library(dataui)

h <- graphics::hist(stats::rnorm(100), plot = FALSE)
dui_histogram(
  components = list(
    dui_barseries(binnedData = hist_to_binned_data(h))
  )
)
```

html_dependency_dataui *'JavaScript' Dependencies for Standalone Usage*

Description

'dataui' also ships a standalone version of the 'JavaScript' allowing for usage outside of a traditional `htmlwidget` context. The variable `dataui` is added to `window` object. Usage of `html_dependency_dataui` should likely be accompanied with `html_dependency_react` along with `html_dependency_reacttools` for `hydrate`.

Usage

```
html_dependency_dataui()
```

Value

htmltools::htmlDependency

Examples

```
library(htmltools)
library(dataui)

browsable(
  tagList(
    reactR::html_dependency_react(),
    reactR::html_dependency_reacttools(),
    html_dependency_dataui(),
    tags$div(id = "chart"),
    tags$div(id = "chart-hydrate"),
    tags$div(id = "chart-hydrate-fromwidget"),
    tags$script(HTML(
sprintf(
"
const sparklineProps = {
  ariaLabel: 'This is a Sparkline of...',
  width: 500,
  height: 100,
  margin: { top: 24, right: 80, bottom: 24, left: 8 },
};

const data = {data: [1,2,3,4,5,6]}


const renderTooltip = function (_ref) {
  var datum = _ref.datum;
  return React.createElement(
      'div',
      null,
      datum.x && React.createElement(
        'div',
        null,
        datum.x
      ),
      React.createElement(
        'div',
        null,
        datum.y ? datum.y.toFixed(2) : \"--\"
      )
  );
}

const spk = React.createElement(
  dataui.SparklineWithTooltip,
  Object.assign(
    data,
    sparklineProps,
```

```
    {renderTooltip: renderTooltip}
  ),
  React.createElement(dataui.SparklineLineSeries)
)


ReactDOM.render(spk, document.getElementById('chart'))

const spk_hydrate = window.reactR.hydrate(
  dataui,
  {
    name: 'SparklineWithTooltip',
    attribs: {...sparklineProps, ...data, ...renderTooltip},
    children: [
      {name: 'SparklineLineSeries', attribs: {stroke: 'purple'}, children: []}
    ]
  }
)

ReactDOM.render(spk_hydrate, document.getElementById('chart-hydrate'))

const spk_hydrate_from_widget = window.reactR.hydrate(
  dataui,
  %s
)

ReactDOM.render(spk_hydrate_from_widget, document.getElementById('chart-hydrate-fromwidget'))
",
  jsonlite::toJSON(
   dui_sparkline(data = 1:6, components = list(dui_sparklineseries(stroke="gray")))$x$tag,
    auto_unbox = TRUE,
    force = TRUE
  )
)

    ))
  )
)
```

Index