Package 'esquisse'

November 14, 2018

Type Package

Title Explore and Visualize Your Data Interactively

Version 0.1.7

Description
A 'shiny' gadget to create 'ggplot2' charts interactively with drag-and-drop to map your variables. You can quickly visualize your data accordingly to their type, export to 'PNG' or 'PowerPoint', and retrieve the code to reproduce the chart.

URL `https://github.com/dreamRs/esquisse`

BugReports `https://github.com/dreamRs/esquisse/issues`

License GPL-3 | file LICENSE

Encoding UTF-8

LazyData true

RoxygenNote 6.1.0

Imports miniUI, rstudioapi, shiny (>= 1.0.0), htmltools, jsonlite, shinyWidgets (>= 0.4.1), ggplot2 (>= 3.0.0), ggthemes, RColorBrewer, viridisLite, scales, stringi

Suggests officer, rvg, rio, DT, testthat

NeedsCompilation no

Author Fanny Meyer [aut],
Victor Perrier [aut, cre],
Ian Carroll [ctb] (Facets support)

Maintainer Victor Perrier `<victor.perrier@dreamrs.fr>`

Repository CRAN

Date/Publication 2018-11-14 09:30:03 UTC

R topics documented:

`chooseData-module` *Module for choosing data.frame*

Description

Module for choosing data.frame from user environment and select variable to use.

Usage

```
chooseDataUI(id, label = "Choose data", icon = "database")

chooseDataServer(input, output, session, dataModule = c("GlobalEnv",
  "ImportFile"), data = NULL, name = NULL, selectVars = TRUE,
  coerceVars = FALSE, launchOnStart = TRUE, size = "m")
```

Arguments

`id`	Module's id.
`label`	Button's label.
`icon`	Button's icon.
`input`	Standard `shiny` input.
`output`	Standard `shiny` output.
`session`	Standard `shiny` session.
`dataModule`	Data module to use, choose between `"GlobalEnv"` or `"ImportFile"`.
`data`	A `data.frame` to use by default.
`name`	Character, object's name to use for `data`.
`selectVars`	Display module to select variables, `TRUE` by default.
`coerceVars`	Display module to coerce variables between different class, `TRUE` by default.
`launchOnStart`	Opens modal window when the application starts.
`size`	Size for the modal window.

Value

a `reactiveValues` containing the data selected under slot `data` and the name of the selected `data.frame` under slot `name`.

Examples

```
if (interactive()) {


library(shiny)
library(esquisse)

ui <- fluidPage(
  tags$h2("Choose data module"),
  fluidRow(
    column(
      width = 4,
      tags$h4("Default"),
      chooseDataUI(id = "choose1"),
      verbatimTextOutput(outputId = "res1")
    ),
    column(
      width = 4,
      tags$h4("No var selection"),
      chooseDataUI(id = "choose2"),
      verbatimTextOutput(outputId = "res2")
    ),
    column(
      width = 4,
      tags$h4("Default data on start"),
      chooseDataUI(id = "choose3"),
      verbatimTextOutput(outputId = "res3")
    )
  )
)

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

  res_dat1 <- callModule(
    chooseDataServer, id = "choose1",
    launchOnStart = FALSE
  )
  output$res1 <- renderPrint({
    str(reactiveValuesToList(res_dat1))
  })

  res_dat2 <- callModule(
    chooseDataServer, id = "choose2", selectVars = FALSE,
    launchOnStart = FALSE
  )
  output$res2 <- renderPrint({
    str(reactiveValuesToList(res_dat2))
  })

  res_dat3 <- callModule(
    chooseDataServer, id = "choose3", data = iris,
    launchOnStart = FALSE
  )
  output$res3 <- renderPrint({
    str(reactiveValuesToList(res_dat3))
  })

}

shinyApp(ui, server)


}
```

coerce-module — *Coerce data.frame's columns module*

Description

Coerce data.frame's columns module

Usage

```
coerceUI(id)

coerceServer(input, output, session, data, reactiveValuesSlot = "data")
```

Arguments

`id`	Module's id
`input`	standard `shiny` input.
`output`	standard `shiny` output.
`session`	standard `shiny` session.
`data`	A `data.frame` or a `reactive` function returning a `data.frame` or a `reactivevalues` with a slot containing a `data.frame` (use `reactiveValuesSlot` to identify that slot)
`reactiveValuesSlot`	If `data` is a `reactivevalues`, specify the name of the slot containing data.

Value

a `reactiveValues` with two slots: `data` original `data.frame` with modified columns, and `names` column's names with call to coerce method.

Examples

```
if (interactive()) {
  library(esquisse)
  library(shiny)

  foo <- data.frame(
    num_as_char = as.character(1:10),
    char = sample(letters[1:3], 10, TRUE),
    fact = factor(sample(LETTERS[1:3], 10, TRUE)),
    date_as_char =  as.character(
      Sys.Date() + sample(seq(-10, 10), 10, TRUE)
    ),
    date_as_num = as.numeric(
      Sys.Date() + sample(seq(-10, 10), 10, TRUE)
    ),
    datetime = Sys.time() + sample(seq(-10, 10) * 1e4, 10, TRUE),
    stringsAsFactors = FALSE
  )

  ui <- fluidPage(
    tags$h2("Coerce module"),
    fluidRow(
      column(
        width = 4,
        coerceUI(id = "exemple", data = foo)
      ),
      column(
        width = 8,
        verbatimTextOutput(outputId = "print_result"),
        verbatimTextOutput(outputId = "print_names")
      )
    )
  )

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

    result <- callModule(module = coerceServer, id = "exemple", data = foo)

    output$print_result <- renderPrint({
      str(result$data)
    })
    output$print_names <- renderPrint({
      result$names
    })
  }

  shinyApp(ui, server)
}
```

`dragulaInput` *Drag And Drop Input Widget*

Description

Drag And Drop Input Widget

Usage

```
dragulaInput(inputId, sourceLabel, targetsLabels, targetsIds = NULL,
  choices = NULL, choiceNames = NULL, choiceValues = NULL,
  status = "primary", replace = FALSE, badge = TRUE, width = NULL,
  height = "200px")
```

Arguments

`inputId`	The `input` slot that will be used to access the value.
`sourceLabel`	Label display in the source box
`targetsLabels`	Labels for each target element.
`targetsIds`	Ids for retrieving values server-side, if `NULL`, the default, `targetsLabels` are used after removing all not-alphanumeric characters.
`choices`	List of values to select from (if elements of the list are named then that name rather than the value is displayed to the user). If this argument is provided, then `choiceNames` and `choiceValues` must not be provided, and vice-versa. The values should be strings; other types (such as logicals and numbers) will be coerced to strings.
`choiceNames, choiceValues`	List of names and values, respectively, that are displayed to the user in the app and correspond to the each choice (for this reason, choiceNames and choiceValues must have the same length). If either of these arguments is provided, then the other must be provided and choices must not be provided. The advantage of using both of these over a named list for choices is that choiceNames allows any type of UI object to be passed through (tag objects, icons, HTML code, ...), instead of just simple text.
`status`	If choices are displayed into a Bootstrap label, you can use Bootstrap status to color them, or `NULL`.
`replace`	When a choice is dragged in a target container already containing a choice, does the later be replaced by the new one ?
`badge`	Displays choices inside a Bootstrap badge. Use `FALSE` if you want to pass custom appearance with `choiceNames`.
`width`	Width of the input.
`height`	Height of each boxes, the total input height is this parameter X 2.

Value

a UI definition

Note

The output server-side is a list with two slots: `source` and `targets`.

See Also

`updateDragulaInput` to update choices server-side.

Examples

```
if (interactive()) {

library("shiny")
library("esquisse")

ui <- fluidPage(
  tags$h2("Demo dragulaInput"),
  tags$br(),
  dragulaInput(
    inputId = "dad",
    sourceLabel = "Source",
    targetsLabels = c("Target 1", "Target 2"),
    choices = names(iris),
    width = "400px"
  ),
  verbatimTextOutput(outputId = "result")
)


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

  output$result <- renderPrint(str(input$dad))

}

shinyApp(ui = ui, server = server)

}
```

`esquisser` *An add-in to easily create plots with ggplot2*

Description

An add-in to easily create plots with ggplot2

Usage

```
esquisser(data = NULL, coerceVars = getOption(x =
  "esquisse.coerceVars", default = FALSE), viewer = getOption(x =
  "esquisse.viewer", default = "dialog"))
```

Arguments

`data`	a data.frame, you can pass a data.frame explicitly to the function, otherwise you'll have to choose one in global environment.
`coerceVars`	If `TRUE` allow to coerce variables to different type when selecting data.
`viewer`	Where to display the gadget: `"dialog"`, `"pane"` or `"browser"` (see `viewer`).

Value

code to reproduce chart.

Examples

```
if (interactive()) {
# Launch with:
esquisser(iris)
# If in RStudio it will be launched by default in dialog window
# If not, it will be launched in browser

# change diplay mode with:
options("esquisse.display.mode" = "viewer")
# or
options("esquisse.display.mode" = "browser")
}
```

`esquisserServer` *Esquisse Shiny module*

Description

Launch `esquisse` in a classic Shiny app.

Usage

```
esquisserServer(input, output, session, data = NULL,
  dataModule = c("GlobalEnv", "ImportFile"), sizeDataModule = "m")

esquisserUI(id, header = TRUE, choose_data = TRUE)
```

Arguments

`input`	standard `shiny` input
`output`	standard `shiny` output
`session`	standard `shiny` session
`data`	a `reactiveValues` with at least a slot `data` containing a `data.frame` to use in the module. And a slot `name` corresponding to the name of the `data.frame`.
`dataModule`	Data module to use, choose between `"GlobalEnv"` or `"ImportFile"`.
`sizeDataModule`	Size for the modal window for selecting data.
`id`	Module's id.
`header`	Logical. Display or not `esquisse` header.
`choose_data`	Logical. Display or not the button to choose data.

Note

For the module to display correctly, it is necessary to place it in a container with a fixed height.

Examples

```
if (interactive()) {


### Part of a Shiny app ###

library(shiny)
library(esquisse)

ui <- fluidPage(
  tags$h1("Use esquisse as a Shiny module"),
  radioButtons(
    inputId = "data",
    label = "Data to use:",
    choices = c("iris", "mtcars"),
    inline = TRUE
  ),
  tags$div(
    style = "height: 700px;", # needs to be in fixed height container
    esquisserUI(
      id = "esquisse",
      header = FALSE, # dont display gadget title
      choose_data = FALSE # dont display button to change data
    )
  )
)

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

  data_r <- reactiveValues(data = iris, name = "iris")
  observeEvent(input$data, {
    if (input$data == "iris") {
      data_r$data <- iris
      data_r$name <- "iris"
    } else {
      data_r$data <- mtcars
      data_r$name <- "mtcars"
    }
  })

  callModule(module = esquisserServer, id = "esquisse", data = data_r)

}

shinyApp(ui, server)



### Whole Shiny app ###

library(shiny)
library(esquisse)


# Load some datasets in app environment
my_data <- data.frame(
  var1 = rnorm(100),
  var2 = sample(letters[1:5], 100, TRUE)
)



ui <- fluidPage(
  tags$div( # needs to be in fixed height container
    style = "position: fixed; top: 0; bottom: 0; right: 0; left: 0;",
    esquisserUI(id = "esquisse")
  )
)

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

  callModule(module = esquisserServer, id = "esquisse")

}

shinyApp(ui, server)

}
```

`filterData-module` *Modules for creating filters from a data.frame*

Description

Modules for creating filters from a data.frame

Usage

```
filterDataUI(id)

filterDataServer(input, output, session, data, vars = NULL,
  width = "100%")
```

Arguments

`id`	Module's id
`input`	standard `shiny` input.
`output`	standard `shiny` output.
`session`	standard `shiny` session.
`data`	a `data.frame` or a `reactive` function returning a `data.frame`.
`vars`	variables for which to create filters, by default all variables in `data`.
`width`	the width of the input, e.g. `400px`, or `100%`.

Value

a `reactiveValues` containing the data filtered under slot `data`, the R code to reproduce the filtering under slot `code` and a logical vector for indexing data under slot `index`.

Note

Column's names can be modified to be valid R names. discrete columns with more than 50 unique values will be discarded.

Examples

```
if (interactive()) {
library(shiny)
library(shinyWidgets)
library(esquisse)

ui <- fluidPage(

  tags$h1("Module Filter Data"),

  fluidRow(
    column(
      width = 4,
      radioButtons(
        inputId = "dataset", label = "Data:",
        choices = c("iris", "mtcars", "Titanic")
      ),
      filterDataUI("ex")
    ),
    column(
      width = 8,
      progressBar(
        id = "pbar", value = 100,
        total = 100, display_pct = TRUE
      ),
      DT::dataTableOutput(outputId = "tab"),
      verbatimTextOutput(outputId = "code")
    )
  )

)

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

  data <- reactive({
    if (input$dataset == "iris") {
      return(iris)
    } else if (input$dataset == "mtcars") {
      return(mtcars)
    } else {
      return(as.data.frame(Titanic))
    }
  })

  res <- callModule(module = filterDataServer,
                    id = "ex", data = data)

  observeEvent(res$data, {
    updateProgressBar(
      session = session, id = "pbar",
      value = nrow(res$data), total = nrow(data())
    )
  })

  output$tab <- DT::renderDataTable(res$data)

  output$code <- renderPrint(res$code)

}

shinyApp(ui, server)
}
```

ggplot_to_ppt *Utility to export ggplot objects to PowerPoint*

Description

You can use the RStudio addin to interactively select ggplot objects, or directly pass their names to the function.

Usage

```
ggplot_to_ppt(gg = NULL)
```

Arguments

gg	character. Name(s) of ggplot object(s), if NULL, launch the Shiny gadget.

Value

Path to the temporary PowerPoint file.

Examples

```
# Shiny gadget
if (interactive()) {

ggplot_to_ppt()



# Or with an object's name
library(ggplot2)
p <- ggplot(iris) +
  geom_point(aes(Sepal.Length, Sepal.Width))

ggplot_to_ppt("p")

}
```

updateDragulaInput *Update Dragula Input*

Description

Update Dragula Input

Usage

```
updateDragulaInput(session, inputId, choices = NULL,
  choiceNames = NULL, choiceValues = NULL, badge = TRUE,
  status = "primary")
```

Arguments

`session`	The `session` object passed to function given to `shinyServer`.
`inputId`	The id of the input object.
`choices`	List of values to select from (if elements of the list are named then that name rather than the value is displayed to the user). If this argument is provided, then `choiceNames` and `choiceValues` must not be provided, and vice-versa. The values should be strings; other types (such as logicals and numbers) will be coerced to strings.
`choiceNames, choiceValues`	List of names and values, respectively, that are displayed to the user in the app and correspond to the each choice (for this reason, choiceNames and choiceValues must have the same length). If either of these arguments is provided, then the other must be provided and choices must not be provided. The advantage of using both of these over a named list for choices is that choiceNames allows any type of UI object to be passed through (tag objects, icons, HTML code, ...), instead of just simple text.
`badge`	Displays choices inside a Bootstrap badge.
`status`	If choices are displayed into a Bootstrap badge, you can use Bootstrap status to color them, or `NULL`.

Examples

```
if (interactive()) {

library("shiny")
library("esquisse")

ui <- fluidPage(
  tags$h2("Update dragulaInput"),
  radioButtons(
    inputId = "update",
    label = "Dataset",
    choices = c("iris", "mtcars")
  ),
  tags$br(),
  dragulaInput(
    inputId = "myDad",
    sourceLabel = "Variables",
    targetsLabels = c("X", "Y", "fill", "color", "size"),
    choices = names(iris),
    replace = TRUE, width = "400px", status = "success"
  ),
  verbatimTextOutput(outputId = "result")
)

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

  output$result <- renderPrint(str(input$myDad))

  observeEvent(input$update, {
    if (input$update == "iris") {
      updateDragulaInput(
        session = session,
        inputId = "myDad",
        choices = names(iris),
        status = "success"
      )
    } else {
      updateDragulaInput(
        session = session,
        inputId = "myDad",
        choices = names(mtcars)
      )
    }
  }, ignoreInit = TRUE)

}

shinyApp(ui, server)

}
```

Index