Package 'esquisse'

September 1, 2022

Type Package

Title Explore and Visualize Your Data Interactively

Version 1.1.2

Description A 'shiny' gadget to create 'ggplot2' figures interactively with drag-and-drop to map your variables to different aesthetics.
You can quickly visualize your data accordingly to their type, export in various formats, and retrieve the code to reproduce the plot.

URL https://dreamrs.github.io/esquisse/, https://github.com/dreamRs/esquisse

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

License GPL-3 | file LICENSE

Encoding UTF-8

RoxygenNote 7.2.0

Imports datamods (>= 1.2.0), ggplot2 (>= 3.0.0), grDevices, htmltools (>= 0.5.0), jsonlite, phosphoricons, rlang (>= 0.3.1), rstudioapi, scales, shiny (>= 1.1.0), shinyWidgets (>= 0.6.0)

Suggests officer, rvg, rio, testthat (>= 2.1.0), knitr, rmarkdown, ggthemes, hrbrthemes

VignetteBuilder knitr

NeedsCompilation no

Author Fanny Meyer [aut], Victor Perrier [aut, cre], Ian Carroll [ctb] (Faceting support), Xiangnan Dang [ctb] (Facets rows and cols, X/Y limits), Nicolas Bevacqua [cph] (author of dragula JavaScript library), Daybrush (Younkue Choi) [cph] (author of moveable JavaScript library), Zeno Rocha [cph] (author of clipboard JavaScript library)

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

Repository CRAN

Date/Publication 2022-09-01 13:50:02 UTC

R topics documented:

`build_aes` *Build aesthetics to use in a plot*

Description

Build aesthetics to use in a plot

Usage

```
build_aes(data, ..., .list = NULL, geom = NULL)
```

Arguments

`data`	Data to use in the plot.
`...`	Named list of aesthetics.
`.list`	Alternative to `...` to use a preexisting named list.
`geom`	Geom to use, according to the geom aesthetics may vary.

Value

An expression

Examples

```
# Classic
build_aes(iris, x = "Sepal.Width")
build_aes(iris, x = "Sepal.Width", y = "Sepal.Width")

# Explicit geom : no change
build_aes(iris, x = "Species", geom = "bar")

# Little trick if data is count data
df <- data.frame(
  LET = c("A", "B"),
  VAL = c(4, 7)
)
build_aes(df, x = "LET", y = "VAL", geom = "bar")

# e.g. :
library(ggplot2)
ggplot(df) +
  build_aes(df, x = "LET", y = "VAL", geom = "bar") +
  geom_bar()
```

---

dragulaInput *Drag And Drop Input Widget*

Description

Drag And Drop Input Widget

Usage

```
dragulaInput(
  inputId,
  label = NULL,
  sourceLabel,
  targetsLabels,
  targetsIds = NULL,
  choices = NULL,
  choiceNames = NULL,
  choiceValues = NULL,
  selected = NULL,
  status = "primary",
  replace = FALSE,
  copySource = TRUE,
  badge = TRUE,
```

```
  ncolSource = "auto",
  ncolGrid = NULL,
  dragulaOpts = list(),
  boxStyle = NULL,
  width = NULL,
  height = "100px"
)
```

Arguments

inputId	The input slot that will be used to access the value.
label	Display label for the control, or NULL for no label.
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.
selected	Default selected values. Must be a list with targetsIds as names.
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 ?
copySource	When replace = TRUE, does elements in source must be copied or moved ?
badge	Displays choices inside a Bootstrap badge. Use FALSE if you want to pass custom appearance with choiceNames.
ncolSource	Number of columns occupied by the source, default is "auto", meaning full row.
ncolGrid	Number of columns used to place source and targets boxes, see examples.
dragulaOpts	Options passed to dragula JavaScript library.
boxStyle	CSS style string to customize source and target container.
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

```
library(shiny)
library(esquisse)

ui <- fluidPage(
  tags$h2("Demo dragulaInput"),
  tags$br(),
  fluidRow(
    column(
      width = 6,

      dragulaInput(
        inputId = "dad1",
        label = "Default:",
        sourceLabel = "Source",
        targetsLabels = c("Target 1", "Target 2"),
        choices = month.abb,
        width = "100%"
      ),
      verbatimTextOutput(outputId = "result1"),

      tags$br(),

      dragulaInput(
        inputId = "dad3",
        label = "On same row:",
        sourceLabel = "Source",
        targetsLabels = c("Target 1", "Target 2"),
        choices = month.abb,
        width = "100%",
        ncolSource = 1,
        ncolGrid = 3
      ),
      verbatimTextOutput(outputId = "result3")
    ),

    column(
      width = 6,
      dragulaInput(
        inputId = "dad2",
```

```
      label = "Two rows:",
      sourceLabel = "Source",
      targetsLabels = c("x", "y", "color", "fill", "size", "facet"),
      choices = names(mtcars),
      width = "100%",
      ncolGrid = 3
    ),
    verbatimTextOutput(outputId = "result2"),

    tags$br(),

    dragulaInput(
      inputId = "dad4",
      label = "Two rows not filled:",
      sourceLabel = "Source",
      targetsLabels = c("x", "y", "color", "fill", "size"),
      choices = names(mtcars),
      width = "100%",
      ncolGrid = 3
    ),
    verbatimTextOutput(outputId = "result4")
  )
 )
)


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

  output$result1 <- renderPrint(str(input$dad1))

  output$result2 <- renderPrint(str(input$dad2))

  output$result3 <- renderPrint(str(input$dad3))

  output$result4 <- renderPrint(str(input$dad4))

}

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

`dropInput` *Dropdown Input*

Description

A dropdown menu for selecting a value.

Usage

```
dropInput(
  inputId,
  choicesNames,
  choicesValues,
  selected = NULL,
  dropUp = FALSE,
  dropWidth = NULL,
  dropMaxHeight = NULL,
  dropPreScrollable = FALSE,
  btnClass = "btn-link",
  width = NULL
)
```

Arguments

inputId	The input slot that will be used to access the value.
choicesNames	A tagList of HTML tags to show in the dropdown menu.
choicesValues	Vector corresponding to choicesNames for retrieving values server-side.
selected	The initial selected value, must be an element of choicesValues, default to the first item of choicesValues.
dropUp	Open the menu above the button rather than below.
dropWidth	Width of the dropdown menu.
dropMaxHeight	Maximal height for the menu.
dropPreScrollable	Force scroll bar to appear in the menu.
btnClass	Class for buttons in dropdown menu, default is "btn-link", you can use for example "btn-default" to display regular buttons.
width	The width of the input.

See Also

updateDropInput

Examples

```
if (interactive()) {

  library(shiny)
  library(esquisse)

  ui <- fluidPage(
    tags$h2("Drop Input"),
    dropInput(
      inputId = "mydrop",
      choicesNames = tagList(
        list(icon("home"), style = "width: 100px;"),
```

```
        list(icon("flash"), style = "width: 100px;"),
        list(icon("cogs"), style = "width: 100px;"),
        list(icon("fire"), style = "width: 100px;"),
        list(icon("users"), style = "width: 100px;"),
        list(icon("info"), style = "width: 100px;")
      ),
      choicesValues = c("home", "flash", "cogs",
                        "fire", "users", "info"),
      dropWidth = "220px"
    ),
    verbatimTextOutput(outputId = "res")
  )

  server <- function(input, output, session) {
    output$res <- renderPrint({
      input$mydrop
    })
  }

  shinyApp(ui, server)

}
```

`esquisse` *Explore and Visualize Your Data Interactively*

Description

A 'shiny' gadget to create 'ggplot2' figures interactively with drag-and-drop to map your variables to different aesthetics. You can quickly visualize your data accordingly to their type, export in various formats, and retrieve the code to reproduce the plot.

Author(s)

Fanny Meyer & Victor Perrier (@dreamRs_fr)

Examples

```
## Not run:

esquisser()

# launch esquisse with specific data:
esquisser(mtcars)


## End(Not run)
```

esquisse-deprecated *Deprecated functions*

Description

Deprecated functions

Note

The following functions are deprecated and will be removed in next release:

- `esquisserUI` / `esquisserServer`: replaced by `esquisse_ui` / `esquisse_server`
- `filterDF_UI` / `filterDF`: moved to package datamods
- `chooseDataUI` / `chooseDataServer`: moved to package datamods
- `coerceUI` / `coerceServer`: moved to package datamods

esquisse-exports *esquisse exported operators and S3 methods*

Description

esquisse exported operators and S3 methods

esquisse-module *Esquisse module*

Description

Use esquisse as a module in a Shiny application.

Usage

```
esquisse_ui(
  id,
  header = TRUE,
  container = esquisseContainer(),
  controls = c("labs", "parameters", "appearance", "filters", "code"),
  insert_code = FALSE
)

esquisse_server(
  id,
  data_rv = NULL,
```

```
  default_aes = c("fill", "color", "size", "group", "facet"),
  import_from = c("env", "file", "copypaste", "googlesheets")
)

esquisseContainer(width = "100%", height = "700px", fixed = FALSE)
```

Arguments

id	Module ID.
header	Logical. Display or not esquisse header.
container	Container in which display the addin, default is to use esquisseContainer, see examples. Use NULL for no container (behavior in versions <= 0.2.1). Must be a function.
controls	Controls menu to be displayed. Use NULL to hide all menus.
insert_code	Logical, Display or not a button to insert the ggplot code in the current user script (work only in RStudio).
data_rv	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.
default_aes	Default aesthetics to be used, can be a character vector or reactive function returning one.
import_from	From where to import data, argument passed to datamods::import_ui.
width, height	The width and height of the container, e.g. "400px", or "100%"; see validateCssUnit.
fixed	Use a fixed container, e.g. to use use esquisse full page. If TRUE, width and height are ignored. Default to FALSE. It's possible to use a vector of CSS unit of length 4 to specify the margins (top, right, bottom, left).

Value

A reactiveValues with 3 slots :

- **code_plot** : code to generate plot.
- **code_filters** : a list of length two with code to reproduce filters.
- **data** : data.frame used in plot (with filters applied).

Examples

```
### 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
  ),
  checkboxGroupInput(
    inputId = "aes",
    label = "Aesthetics to use:",
    choices = c(
      "fill", "color", "size", "shape",
      "weight", "group", "facet", "facet_row", "facet_col"
    ),
    selected = c("fill", "color", "size", "facet"),
    inline = TRUE
  ),
  esquisse_ui(
    id = "esquisse",
    header = FALSE, # dont display gadget title
    container = esquisseContainer(height = "700px")
  )
)

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

  data_rv <- reactiveValues(data = iris, name = "iris")

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

  esquisse_server(
    id = "esquisse",
    data_rv = data_rv,
    default_aes = reactive(input$aes)
  )

}

if (interactive())
  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(
  esquisse_ui(
    id = "esquisse",
    container = esquisseContainer(fixed = TRUE)
  )
)

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

  esquisse_server(id = "esquisse")

}

if (interactive())
  shinyApp(ui, server)



## You can also use a vector of margins for the fixed argument,
# useful if you have a navbar for example

library(shiny)
library(esquisse)
library(datamods)

ui <- navbarPage(
  title = "My navbar app",
  tabPanel(
    title = "esquisse",
    esquisse_ui(
      id = "esquisse",
      header = FALSE,
      container = esquisseContainer(
        fixed = c(55, 0, 0, 0)
      )
    )
  )
)

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

  # lauch import data modal
  import_modal(
    id = "import-data",
    from = c("env", "file", "copypaste"),
    title = "Import data"
```

```
  )
  data_imported_r <- datamods::import_server("import-data")

  data_rv <- reactiveValues(data = data.frame())
  observeEvent(data_imported_r$data(), {
    data_rv$data <- data_imported_r$data()
    data_rv$name <- data_imported_r$name()
  })

  esquisse_server(id = "esquisse", data_rv = data_rv)

}

if (interactive())
  shinyApp(ui, server)
```

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

Description

Select data to be used and map variables to aesthetics to produce a chart, customize common elements and get code to reproduce the chart.

Usage

```
esquisser(
  data = NULL,
  controls = c("labs", "parameters", "appearance", "filters", "code"),
  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.
`controls`	Controls menu to be displayed. Use `NULL` to hide all menus.
`viewer`	Where to display the gadget: `"dialog"`, `"pane"` or `"browser"` (see `viewer`).

Value

`NULL`. You can view code used to produce the chart, copy it or insert it in current script.

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

# Launch esquisse in browser :
esquisser(iris, viewer = "browser")

# You can set this option in .Rprofile :
options("esquisse.viewer" = "viewer")
# or
options("esquisse.viewer" = "browser")

# esquisse use shiny::runApp
# see ?shiny::runApp to see options
# available, example to use custom port:

options("shiny.port" = 8080)
esquisser(iris, viewer = "browser")

}
```

---

esquisserServer *Esquisse Shiny module*

Description

DEPRECATED, see esquisse-module.

Usage

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

esquisserUI(
  id,
  header = TRUE,
  container = esquisseContainer(),
  choose_data = TRUE,
  insert_code = FALSE,
```

```
  disable_filters = FALSE
)
```

Arguments

input, output, session	Standards shiny server arguments.
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.
container	Container in which display the addin, default is to use esquisseContainer, see examples. Use NULL for no container (behavior in versions <= 0.2.1). Must be a function.
choose_data	Logical. Display or not the button to choose data.
insert_code	Logical, Display or not a button to insert the ggplot code in the current user script (work only in RStudio).
disable_filters	Logical. Disable the menu allowing to filter data used.

Value

A reactiveValues with 3 slots :

- **code_plot** : code to generate plot.
- **code_filters** : a list of length two with code to reproduce filters.
- **data** : data.frame used in plot (with filters applied).

Note

For the module to display correctly, it is necessary to place it in a container with a fixed height. Since version >= 0.2.2, the container is added by default.

---

ggcall *Generate code to create a* ggplot2

Description

Generate code to create a ggplot2

Usage

```
ggcall(
  data = NULL,
  mapping = NULL,
  geom = NULL,
  geom_args = list(),
  scales = NULL,
  scales_args = list(),
  coord = NULL,
  labs = list(),
  theme = NULL,
  theme_args = list(),
  facet = NULL,
  facet_row = NULL,
  facet_col = NULL,
  facet_args = list(),
  xlim = NULL,
  ylim = NULL
)
```

Arguments

`data`	Character. Name of the `data.frame`.
`mapping`	List. Named list of aesthetics.
`geom`	Character. Name of the geom to use (with or without "geom_").
`geom_args`	List. Arguments to use in the geom.
`scales`	Character vector. Scale(s) to use (with or without "scale_").
`scales_args`	List. Arguments to use in scale(s), if `scales` is length > 1, must be a named list with `scales` names.
`coord`	Character. Coordinates to use (with or without "coord_").
`labs`	List. Named list of labels to use for title, subtitle, x & y axis, legends.
`theme`	Character. Name of the theme to use (with or without "theme_").
`theme_args`	Named list. Arguments for `theme`.
`facet`	Character vector. Names of variables to use in `facet_wrap`.
`facet_row`	Character vector. Names of row variables to use in `facet_grid`.
`facet_col`	Character vector. Names of col variables to use in `facet_grid`.
`facet_args`	Named list. Arguments for `facet_wrap`.
`xlim`	A vector of length 2 representing limits on x-axis.
`ylim`	A vector of length 2 representing limits on y-axis.

Value

a `call` that can be evaluated with `eval`.

Examples

```
# Default:
ggcall()

# With data and aes
ggcall("mtcars", list(x = "mpg", y = "wt"))

# Evaluate the call
library(ggplot2)
eval(ggcall("mtcars", list(x = "mpg", y = "wt")))


# With a geom:
ggcall(
  data = "mtcars",
  mapping = list(x = "mpg", y = "wt"),
  geom = "point"
)

# With options
ggcall(
  data = "mtcars",
  mapping = list(x = "hp", y = "cyl", fill = "color"),
  geom = "bar",
  coord = "flip",
  labs = list(title = "My title"),
  theme = "minimal",
  facet = c("gear", "carb"),
  theme_args = list(legend.position = "bottom")
)

# Theme
ggcall(
  "mtcars", list(x = "mpg", y = "wt"),
  theme = "theme_minimal",
  theme_args = list(
    panel.ontop = TRUE,
    legend.title = rlang::expr(element_text(face = "bold"))
  )
)

# Theme from other package than ggplot2
ggcall(
  "mtcars", list(x = "mpg", y = "wt"),
  theme = "ggthemes::theme_economist"
)


# One scale
ggcall(
  data = "mtcars",
  mapping = list(x = "mpg", y = "wt", color = "qsec"),
```

```
  geom = "point",
  scales = "color_distiller",
  scales_args = list(palette = "Blues")
)

# Two scales
ggcall(
  data = "mtcars",
  mapping = list(x = "mpg", y = "wt", color = "qsec", size = "qsec"),
  geom = "point",
  scales = c("color_distiller", "size_continuous"),
  scales_args = list(
    color_distiller = list(palette = "Greens"),
    size_continuous = list(range = c(1, 20))
  )
)
```

`ggplot-output` *Render* `ggplot` *module*

Description

Display a plot on the client and allow to download it.

Usage

```
ggplot_output(
  id,
  width = "100%",
  height = "400px",
  downloads = downloads_labels(),
  ...
)

downloads_labels(
  label = ph("download-simple"),
  png = tagList(ph("image"), "PNG"),
  pdf = tagList(ph("file-pdf"), "PDF"),
  svg = tagList(ph("browsers"), "SVG"),
  jpeg = tagList(ph("image"), "JPEG"),
  pptx = tagList(ph("projector-screen"), "PPTX"),
  more = tagList(ph("gear"), i18n("More options"))
)

render_ggplot(
  id,
  expr,
  ...,
```

```
  env = parent.frame(),
  quoted = FALSE,
  filename = "export-ggplot"
)
```

Arguments

id	Module ID.
width	Width of the plot.
height	Height of the plot.
downloads	Labels for export options, use downloads_labels.
...	Parameters passed to shiny::plotOutput() (ggplot_output) or shiny::renderPlot() (render_ggplot).
label	Main label for export button
png, pdf, svg, jpeg, pptx	Labels to display in export menu, use NULL to disable specific format.
more	Label for "more" button, allowing to launch export modal.
expr	An expression that generates a ggplot object.
env	The environment in which to evaluate expression.
quoted	Is expr a quoted expression (with quote())? This is useful if you want to save an expression in a variable.
filename	A string of the filename to export WITHOUT extension, it will be added according to type of export.

Value

Server-side, a reactiveValues with the plot.

Examples

```
library(shiny)
library(ggplot2)
library(esquisse)


ui <- fluidPage(
  tags$h2("ggplot output"),
  selectInput("var", "Variable:", names(economics)[-1]),
  ggplot_output("MYID", width = "600px")
)

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

  render_ggplot("MYID", {
    ggplot(economics) +
      geom_line(aes(date, !!sym(input$var))) +
```

```
      theme_minimal() +
      labs(
        title = "A cool chart made with ggplot2",
        subtitle = "that you can export in various format"
      )
  })
}

if (interactive())
  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")

}
```

input-colors *Picker input to select color(s) or palette*

Description

Select menu to view and choose a color or a palette of colors.

Usage

```
colorPicker(
  inputId,
  label,
  choices,
  selected = NULL,
  textColor = "#000",
  plainColor = FALSE,
  multiple = FALSE,
  pickerOpts = list(),
  width = NULL
)

updateColorPicker(
  session = getDefaultReactiveDomain(),
  inputId,
  choices,
  textColor = "#000",
  plainColor = FALSE,
  multiple = FALSE
)

palettePicker(
  inputId,
  label,
  choices,
  selected = NULL,
  textColor = "#000",
  plainColor = FALSE,
  pickerOpts = list(),
  width = NULL
)

updatePalettePicker(
  session = getDefaultReactiveDomain(),
  inputId,
  choices,
  selected = NULL,
  textColor = "#000",
```

```
  plainColor = FALSE
)
```

Arguments

inputId	The input slot that will be used to access the value.
label	Display label for the control, or NULL for no label.
choices	List of values to select from. Values must be valid Hex colors. If elements of the list are named then that name rather than the value is displayed to the user.
selected	The initially selected value (or multiple values if multiple = TRUE). If not specified then defaults to the first value for single-select lists and no values for multiple select lists.
textColor	Color of the text displayed above colors, can be a vector of the same length as choices.
plainColor	Color the full space of the choice menu.
multiple	Is selection of multiple items allowed?
pickerOpts	Options for pickerInput.
width	The width of the input : 'auto', 'fit', '100px', '75%'.
session	Shiny session.

Value

A select control that can be added to a UI definition.

Examples

```
# colorPicker ------------------------------------------------------------

library(shiny)
library(esquisse)
library(scales)


ui <- fluidPage(
  tags$h2("colorPicker examples"),
  fluidRow(
    column(
      width = 3,
      colorPicker(
        inputId = "col1",
        label = "With a vector of colors:",
        choices = brewer_pal(palette = "Dark2")(8)
      ),
      verbatimTextOutput("res1"),
      colorPicker(
        inputId = "col5",
        label = "Update colors:",
```

```
      choices = brewer_pal(palette = "Blues", direction = -1)(8),
      textColor = "#FFF"
    ),
    verbatimTextOutput("res5"),
    radioButtons(
      "update", "Colors", c("Blues", "Greens", "Reds"),
      inline = TRUE
    )
  ),
  column(
    width = 3,
    colorPicker(
      inputId = "col2",
      label = "Change text color:",
      choices = brewer_pal(palette = "Blues")(8),
      textColor = c("black", "black", "black", "white",
                    "white", "white", "white", "white")
    ),
    verbatimTextOutput("res2")
  ),
  column(
    width = 3,
    colorPicker(
      inputId = "col3",
      label = "With a list of vector of colors:",
      choices = list(
        "Blues" = brewer_pal(palette = "Blues")(8),
        "Reds" = brewer_pal(palette = "Reds")(8),
        "Greens" = brewer_pal(palette = "Greens")(8)
      )
    ),
    verbatimTextOutput("res3")
  ),
  column(
    width = 3,
    colorPicker(
      inputId = "col4",
      label = "Plain color & multiple choices:",
      choices = brewer_pal(palette = "Paired")(8),
      plainColor = TRUE,
      multiple = TRUE,
      pickerOpts = list(`selected-text-format`= "count > 3")
    ),
    verbatimTextOutput("res4")
  )
 )
)

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

  output$res1 <- renderPrint(input$col1)
  output$res2 <- renderPrint(input$col2)
  output$res3 <- renderPrint(input$col3)
```

```
  output$res4 <- renderPrint(input$col4)
  output$res5 <- renderPrint(input$col5)

  observeEvent(input$update, {
    updateColorPicker(
      inputId = "col5",
      choices = brewer_pal(palette = input$update, direction = -1)(8),
      textColor = "#FFF"
    )
  })

}

if (interactive()) {
  shinyApp(ui, server)
}

# palettePicker ------------------------------------------------------------

library(shiny)
library(esquisse)
library(scales)

ui <- fluidPage(
  tags$h2("pickerColor examples"),

  fluidRow(
    column(
      width = 4,
      palettePicker(
        inputId = "pal1",
        label = "Select a palette:",
        choices = list(
          "Blues" = brewer_pal(palette = "Blues")(8),
          "Reds" = brewer_pal(palette = "Reds")(8)
        )
      ),
      verbatimTextOutput("res1"),
      palettePicker(
        inputId = "pal4",
        label = "Update palette:",
        choices = list(
          "Blues" = brewer_pal(palette = "Blues")(8),
          "Reds" = brewer_pal(palette = "Reds")(8)
        )
      ),
      verbatimTextOutput("res4"),
      radioButtons(
        "update", "Palettes:", c("default", "viridis", "brewer"),
        inline = TRUE
      )
    ),
    column(
```

```
      width = 4,
      palettePicker(
        inputId = "pal2",
        label = "With a list of palette:",
        choices = list(
          "Viridis" = list(
            "viridis" = viridis_pal(option = "viridis")(10),
            "magma" = viridis_pal(option = "magma")(10),
            "inferno" = viridis_pal(option = "inferno")(10),
            "plasma" = viridis_pal(option = "plasma")(10),
            "cividis" = viridis_pal(option = "cividis")(10)
          ),
          "Brewer" = list(
            "Blues" = brewer_pal(palette = "Blues")(8),
            "Reds" = brewer_pal(palette = "Reds")(8),
            "Paired" = brewer_pal(palette = "Paired")(8),
            "Set1" = brewer_pal(palette = "Set1")(8)
          )
        ),
        textColor = c(
          rep("white", 5), rep("black", 4)
        )
      ),
      verbatimTextOutput("res2")
    ),
    column(
      width = 4,
      palettePicker(
        inputId = "pal3",
        label = "With plain colors:",
        choices = list(
          "BrBG" = brewer_pal(palette = "BrBG")(8),
          "PiYG" = brewer_pal(palette = "PiYG")(8),
          "PRGn" = brewer_pal(palette = "PRGn")(8),
          "PuOr" = brewer_pal(palette = "PuOr")(8),
          "RdBu" = brewer_pal(palette = "RdBu")(8),
          "RdGy" = brewer_pal(palette = "RdGy")(8),
          "RdYlBu" = brewer_pal(palette = "RdYlBu")(8),
          "RdYlGn" = brewer_pal(palette = "RdYlGn")(8),
          "Spectral" = brewer_pal(palette = "Spectral")(8)
        ),
        plainColor = TRUE,
        textColor = "white"
      ),
      verbatimTextOutput("res3")
    )
  )
)

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

  output$res1 <- renderPrint(input$pal1)
  output$res2 <- renderPrint(input$pal2)
```

```
  output$res3 <- renderPrint(input$pal3)
  output$res4 <- renderPrint(input$pal4)

  observeEvent(input$update, {
    if (input$update == "default") {
      updatePalettePicker(
        inputId = "pal4",
        choices = list(
          "Blues" = brewer_pal(palette = "Blues")(8),
          "Reds" = brewer_pal(palette = "Reds")(8)
        )
      )
    } else if (input$update == "viridis") {
      updatePalettePicker(
        inputId = "pal4",
        choices = list(
          "viridis" = viridis_pal(option = "viridis")(10),
          "magma" = viridis_pal(option = "magma")(10),
          "inferno" = viridis_pal(option = "inferno")(10),
          "plasma" = viridis_pal(option = "plasma")(10),
          "cividis" = viridis_pal(option = "cividis")(10)
        ),
        textColor = "#FFF"
      )
    } else if (input$update == "brewer") {
      updatePalettePicker(
        inputId = "pal4",
        choices = list(
          "Blues" = brewer_pal(palette = "Blues")(8),
          "Reds" = brewer_pal(palette = "Reds")(8),
          "Paired" = brewer_pal(palette = "Paired")(8),
          "Set1" = brewer_pal(palette = "Set1")(8)
        )
      )
    }
  })
}

if (interactive()) {
  shinyApp(ui, server)
}
```

`match_geom_args` *Match list of arguments to arguments of geometry*

Description

Match list of arguments to arguments of geometry

Usage

```
match_geom_args(
  geom,
  args,
  add_aes = TRUE,
  mapping = list(),
  envir = "ggplot2"
)
```

Arguments

geom	Character. name of the geometry.
args	Named list, parameters to be matched to the geometry arguments.
add_aes	Add aesthetics parameters (like size, fill, ...).
mapping	Mapping used in plot, to avoid setting fixed aesthetics parameters.
envir	Package environment to search in.

Value

a list

Examples

```
# List of parameters
params <- list(
  bins = 30,
  scale = "width",
  adjust = 2,
  position = "stack",
  size = 1.6,
  fill = "#112246"
)

# Search arguments according to geom
match_geom_args(geom = "histogram", args = params)
match_geom_args(geom = "violin", args = params)
match_geom_args(geom = "bar", args = params, add_aes = FALSE)
match_geom_args(geom = "point", args = params)
match_geom_args(geom = "point", args = params, add_aes = FALSE)
```

---

module-chooseData *Module for choosing data.frame*

Description

DEPRECATED, please see package datamods for similar features.

Usage

```
chooseDataUI(id, label = "Data", icon = "database", width = "100%", ...)

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

Arguments

id	Module's id.
label	Label for button, passed to actionButton.
icon	Icon to appears on the button, passed to actionButton.
width	Width of button, passed to actionButton.
...	Other arguments passed to actionButton
input, output, session	standards shiny server arguments.
dataModule	Data module to use, choose between "GlobalEnv" (select ad data.frame from Global environment) or "ImportFile" (import an external file supported by import).
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.
selectedTypes	Type of variables selected by default in select variables module. Possible types are "discrete", "time", "continuous" and "id", by default "id" is discarded.
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.

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

Description

DEPRECATED, please see package datamods for similar features.

Usage

```
coerceUI(id)

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

Arguments

id	Module id. See callModule.
input, output, session	standards shiny server arguments.[2]
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.

module-filterDF *Shiny module to interactively filter a* data.frame

Description

DEPRECATED, please see package datamods for similar features.

Usage

```
filterDF_UI(id, show_nrow = TRUE)

filterDF(
  input,
  output,
  session,
  data_table = reactive(),
```

```
  data_vars = shiny::reactive(NULL),
  data_name = reactive("data"),
  label_nrow = "Number of rows:",
  drop_ids = TRUE,
  picker = FALSE
)
```

Arguments

id	Module id. See callModule.
show_nrow	Show number of filtered rows and total.
input, output, session	standards shiny server arguments.
data_table	reactive function returning a data.frame to filter.
data_vars	reactive function returning a character vector of variable to use for filters.
data_name	reactive function returning a character string representing data_table name.
label_nrow	Text to display before the number of rows of filtered data / source data.
drop_ids	Drop columns containing more than 90% of unique values, or than 50 distinct values.
picker	Use shinyWidgets::pickerInput instead of shiny::selectizeInput (default).

Value

A list with 2 elements :

- **data_filtered** : reactive function returning data filtered.
- **code** : reactiveValues with 2 slots : expr (raw expression to filter data) and dplyr (code with dplyr pipeline).

---

potential_geoms *Potential geometries according to the data*

Description

Potential geometries according to the data

Usage

```
potential_geoms(data, mapping, auto = FALSE)
```

Arguments

data	A data.frame
mapping	List of aesthetic mappings to use with data.
auto	Return only one geometry.

Value

A character vector

Examples

```
library(ggplot2)

# One continuous variable
potential_geoms(
  data = iris,
  mapping = aes(x = Sepal.Length)
)

# Automatic pick a geom
potential_geoms(
  data = iris,
  mapping = aes(x = Sepal.Length),
  auto = TRUE
)

# One discrete variable
potential_geoms(
  data = iris,
  mapping = aes(x = Species)
)

# Two continuous variables
potential_geoms(
  data = iris,
  mapping = aes(x = Sepal.Length, y = Sepal.Width)
)
```

---

run_module *Run module example*

Description

DEPRECATED, please see package datamods for similar features.

Usage

```
run_module(module = c("filterDF", "chooseData", "chooseData2", "coerce"))
```

Arguments

module	Module for which to see a demo.

safe_ggplot *Safely render a* ggplot *in Shiny application*

Description

Safely render a ggplot in Shiny application

Usage

```
safe_ggplot(expr, data = NULL, session = shiny::getDefaultReactiveDomain())
```

Arguments

expr	Code to produce a ggplot object.
data	Argument passed to eval_tidy to evaluate expression.
session	Session object to send notification to.

Value

Output of ggplot_build.

Examples

```
if (interactive()) {
  library(shiny)
  library(ggplot2)

  ui <- fluidPage(
    fluidRow(
      column(
        width = 3,
        selectInput(
          inputId = "var",
          label = "Var:",
          choices = c("Sepal.Width", "Do.Not.Exist")
        )
      ),
      column(
        width = 9,
        plotOutput(outputId = "plot")
      )
    )
  )

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

    output$plot <- renderPlot({
      p <- ggplot(iris) +
        geom_point(aes_string("Sepal.Length", input$var))
```

```
      safe_ggplot(p)
    })

  }

  shinyApp(ui, server)
}
```

---

save-ggplot-module *Save* ggplot *module*

Description

Save a ggplot object in various format and resize it before saving.

Usage

```
save_ggplot_ui(
  id,
  output_format = c("png", "pdf", "svg", "jpeg", "bmp", "eps", "tiff")
)

save_ggplot_modal(
  id,
  title = NULL,
  output_format = c("png", "pdf", "svg", "jpeg", "bmp", "eps", "tiff")
)

save_ggplot_server(id, plot_rv)
```

Arguments

id	Module ID.
output_format	Output formats offered to the user.
title	Modal's title.
plot_rv	A reactiveValues with a slot plot containing a ggplot object.

Value

No value. Use in UI & server of shiny application.

Examples

```
library(shiny)
library(ggplot2)
library(esquisse)
```

```
ui <- fluidPage(
  tags$h2("Save a ggplot"),
  selectInput("var", "Variable:", names(economics)[-1]),
  plotOutput("plot", width = "600px"),
  actionButton("save", "Save this plot")
)

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

  rv <- reactiveValues(plot = NULL)

  output$plot <- renderPlot({
    rv$plot <- ggplot(economics) +
      geom_line(aes(date, !!sym(input$var))) +
      theme_minimal()
    rv$plot
  })

  observeEvent(input$save, {
    save_ggplot_modal("ID", "Save plot")
  })
  save_ggplot_server("ID", rv)
}

if (interactive())
  shinyApp(ui, server)
```

`updateDragulaInput` *Update Dragula Input*

Description

Update `dragulaInput()` widget server-side.

Usage

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

Arguments

session	The session object passed to function given to shinyServer.
inputId	The input slot that will be used to access the value.
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.
selected	Default selected values. Must be a list with targetsIds as names.
selectedNames, selectedValues	Update selected items with custom names and values.
badge	Displays choices inside a Bootstrap badge. Use FALSE if you want to pass custom appearance with choiceNames.
status	If choices are displayed into a Bootstrap label, 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)

}
```

`updateDropInput` *Change the value of a drop input on the client*

Description

Change the value of a drop input on the client

Usage

```
updateDropInput(session, inputId, selected = NULL, disabled = NULL)
```

Arguments

`session`	The `session` object passed to function given to `shinyServer`.
`inputId`	The id of the input object.
`selected`	The initially selected value.
`disabled`	Choices (`choicesValues`) to disable.

See Also

dropInput

Examples

```
if (interactive()) {

  library(shiny)
  library(esquisse)

  myChoices <- tagList(
    list(icon("home"), style = "width: 100px;"),
    list(icon("flash"), style = "width: 100px;"),
    list(icon("cogs"), style = "width: 100px;"),
    list(icon("fire"), style = "width: 100px;"),
    list(icon("users"), style = "width: 100px;"),
    list(icon("info"), style = "width: 100px;")
  )


  ui <- fluidPage(
    tags$h2("Update Drop Input"),
    fluidRow(
      column(
        width = 6,
        dropInput(
          inputId = "mydrop",
          choicesNames = myChoices,
          choicesValues = c("home", "flash", "cogs", "fire", "users", "info"),
          dropWidth = "220px"
        ),
        verbatimTextOutput(outputId = "res")
      ),
      column(
        width = 6,
        actionButton("home", "Select home"),
        actionButton("flash", "Select flash"),
        actionButton("cogs", "Select cogs"),
        actionButton("fire", "Select fire"),
        actionButton("users", "Select users"),
        actionButton("info", "Select info"),
        checkboxGroupInput(
          inputId = "disabled",
          label = "Choices to disable",
          choices = c("home", "flash", "cogs", "fire", "users", "info")
        ),
        actionButton("disable", "Disable")
      )
    )
  )

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

```
    output$res <- renderPrint({
      input$mydrop
    })

    observeEvent(input$home, {
      updateDropInput(session, "mydrop", "home")
    })
    observeEvent(input$flash, {
      updateDropInput(session, "mydrop", "flash")
    })
    observeEvent(input$cogs, {
      updateDropInput(session, "mydrop", "cogs")
    })
    observeEvent(input$fire, {
      updateDropInput(session, "mydrop", "fire")
    })
    observeEvent(input$users, {
      updateDropInput(session, "mydrop", "users")
    })
    observeEvent(input$info, {
      updateDropInput(session, "mydrop", "info")
    })

    observeEvent(input$disable, {
      if (!is.null(input$disabled)) {
        updateDropInput(session, "mydrop", disabled = input$disabled)
      } else {
        updateDropInput(session, "mydrop", disabled = character(0))
      }
    })
  }

  shinyApp(ui, server)

}
```

`which_pal_scale` *Automatically select appropriate color scale*

Description

Automatically select appropriate color scale

Usage

```
which_pal_scale(
  mapping,
  palette = "ggplot2",
  data = NULL,
```

```
  fill_type = c("continuous", "discrete"),
  color_type = c("continuous", "discrete"),
  reverse = FALSE
)
```

Arguments

mapping	Aesthetics used in ggplot.
palette	Color palette.
data	An optional data.frame to choose the right type for variables.
fill_type, color_type	Scale to use according to the variable used in fill/color aesthetic : "discrete" or "continuous". Ignored if data is provided: it will be guessed from data.
reverse	Reverse colors order or not.

Value

a list

Examples

```
library(ggplot2)

# Automatic guess according to data
which_pal_scale(
  mapping = aes(fill = Sepal.Length),
  palette = "ggplot2",
  data = iris
)
which_pal_scale(
  mapping = aes(fill = Species),
  palette = "ggplot2",
  data = iris
)


# Explicitly specify type
which_pal_scale(
  mapping = aes(color = variable),
  palette = "Blues",
  color_type = "discrete"
)


# Both scales
which_pal_scale(
  mapping = aes(color = var1, fill = var2),
  palette = "Blues",
  color_type = "discrete",
  fill_type = "continuous"
)
```

Index