

# Package ‘esquisse’

January 27, 2020

**Type** Package

**Title** Explore and Visualize Your Data Interactively

**Version** 0.3.0

**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** 7.0.2

**Imports** miniUI, rstudioapi, shiny (>= 1.0.0), htmltools, jsonlite, shinyWidgets (>= 0.4.1), ggplot2 (>= 3.0.0), scales, stringi, rlang (>= 0.3.1), grDevices

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

**VignetteBuilder** knitr

**NeedsCompilation** no

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

**Maintainer** Victor Perrier <[victor.perrier@dreamrs.fr](mailto:victor.perrier@dreamrs.fr)>

**Repository** CRAN

**Date/Publication** 2020-01-27 17:40:03 UTC

**R topics documented:**

build_aes	2
dragulaInput	3
dropInput	5
esquisser	7
esquisserServer	8
ggcall	12
ggplot_to_ppt	14
input-colors	15
match_geom_args	19
module-chooseData	20
module-coerce	22
module-filterDF	24
potential_geoms	27
run_module	28
safe_ggplot	28
updateDragulaInput	29
updateDropInput	31
which_pal_scale	33
<b>Index</b>	<b>36</b>

---

build_aes	<i>Build aesthetics to use in a plot</i>
-----------	--

---

**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,
  sourceLabel,
  targetsLabels,
  targetsIds = NULL,
  choices = NULL,
  choiceNames = NULL,
  choiceValues = NULL,
  status = "primary",
  replace = FALSE,
  badge = TRUE,
  dragulaOpts = list(),
  boxStyle = NULL,
  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.
dragulaOpts	Options passed to dragula JS 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**

```
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)  
  
}
```

---

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)
}
```

---

esquisser

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

---

## Description

An add-in to easily create plots with ggplot2

## Usage

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

## Arguments

<code>data</code>	a data.frame, you can pass a data.frame explicitly to the function, otherwise you'll have to choose one in global environment.
<code>coerce_vars</code>	If TRUE allow to coerce variables to different type when selecting data.
<code>disable_filters</code>	Logical. Disable the menu allowing to filter data used.
<code>viewer</code>	Where to display the gadget: "dialog", "pane" or "browser" (see <a href="#">viewer</a> ).

## 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

  # 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

Launch esquisse in a classic Shiny app.

## 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
  )

  esquisseContainer(width = "100%", height = "700px", fixed = 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.

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).

### 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.

**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  
    ),  
    esquisserUI(  
      id = "esquisse",  
      header = FALSE, # dont display gadget title  
      choose_data = FALSE, # dont display button to change data,  
      container = esquisseContainer(height = "700px")  
    )  
  )  
  
  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(
  esquisserUI(
    id = "esquisse",
    container = esquisseContainer(fixed = TRUE)
  )
)

server <- function(input, output, session) {
  callModule(module = esquisserServer, id = "esquisse")
}

shinyApp(ui, server)

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

ui <- navbarPage(
  title = "My navbar app",
  tabPanel(
    title = "esquisse",
    esquisserUI(
      id = "esquisse",
      header = FALSE,
      container = esquisseContainer(
        fixed = c(50, 0, 0, 0)
      )
    )
  )
)

server <- function(input, output, session) {
  callModule(module = esquisserServer, id = "esquisse")
}

shinyApp(ui, server)
}
```

---

`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_args = list()  
)
```

**Arguments**

<code>data</code>	Character. Name of the data frame.
<code>mapping</code>	List. Named list of aesthetics.
<code>geom</code>	Character. Name of the geom to use (with or without "geom_").
<code>geom_args</code>	List. Arguments to use in the geom.
<code>scales</code>	Character vector. Scale(s) to use (with or without "scale_").
<code>scales_args</code>	List. Arguments to use in scale(s), if <code>scales</code> is length > 1, must be a named list with scales names.
<code>coord</code>	Character. Coordinates to use (with or without "coord_").
<code>labs</code>	List. Named list of labels to use for title, subtitle, x & y axis, legends.
<code>theme</code>	Character. Name of the theme to use (with or without "theme_").
<code>theme_args</code>	Named list. Arguments for <a href="#">theme</a> .
<code>facet</code>	Character vector. Names of variables to use in <a href="#">facet_wrap</a> .
<code>facet_args</code>	Named list. Arguments for <a href="#">facet_wrap</a> .

**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\_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  
)
```

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

### 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 <code>multiple = TRUE</code> ). 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 <code>pickerInput</code> .
width	The width of the input : 'auto', 'fit', '100px', '75%'.

### Value

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

### Examples

```
# colorPicker -----
if (interactive()) {
  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")
      ),
      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",
        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)

}

shinyApp(ui, server)

}

# palettePicker -----

if (interactive()) {

  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")
),
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)

}

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 match to geom's 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

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

## Usage

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

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	Button's label.

icon	Button's icon.
...	Arguments passed to <a href="#">actionButton</a>
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 <a href="#">import</a> ).
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.

### 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$h4("No variable selection"),
        chooseDataUI(id = "choose2"),
        verbatimTextOutput(outputId = "res2")
      ),
      column(
        width = 4,
        tags$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)

}

```

---

 module-coerce

*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 id. See [callModule](#).  
**input, output, session**                    standards shiny server arguments.<sup>2</sup>  
**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 = "example")
      ),
      column(
        width = 8,
        verbatimTextOutput(outputId = "print_result"),
        verbatimTextOutput(outputId = "print_names")
      )
    )
  )
}

```

```

server <- function(input, output, session) {
  result <- callModule(module = coerceServer, id = "example", data = reactive({foo}))

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

shinyApp(ui, server)
}

```

---

 module-filterDF

*Shiny module to interactively filter a data.frame*


---

### Description

Module generate inputs to filter `data.frame` according column's type. Code to reproduce the filter is returned as an expression with filtered data.

### 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

<code>id</code>	Module id. See <a href="#">callModule</a> .
<code>show_nrow</code>	Show number of filtered rows and total.
<code>input, output, session</code>	standards shiny server arguments.
<code>data_table</code>	<a href="#">reactive</a> function returning a <code>data.frame</code> to filter.



data_vars	<code>reactive</code> function returning a character vector of variable to use for filters.
data_name	<code>reactive</code> 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 <code>shinyWidgets::pickerInput</code> instead of <code>shiny::selectizeInput</code> (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).

### Examples

```
if (interactive()) {

  library(shiny)
  library(shinyWidgets)
  library(ggplot2)
  library(esquisse)

  # Add some NAs to mpg
  mpg_na <- mpg
  mpg_na[] <- lapply(
    X = mpg_na,
    FUN = function(x) {
      x[sample.int(n = length(x), size = sample(15:30, 1))] <- NA
    }
  )

  ui <- fluidPage(
    tags$h2("Filter data.frame"),

    radioButtons(
      inputId = "dataset",
      label = "Data:",
      choices = c(
        "iris", "mtcars", "economics",
        "midwest", "mpg", "mpg_na", "msleep", "diamonds",
        "faithful", "txhousing"
      ),
      inline = TRUE
    ),

    fluidRow(
      column(
```

```

      width = 3,
      filterDF_UI("filtering")
    ),
    column(
      width = 9,
      progressBar(
        id = "pbar", value = 100,
        total = 100, display_pct = TRUE
      ),
      DT::dataTableOutput(outputId = "table"),
      tags$p("Code dplyr:"),
      verbatimTextOutput(outputId = "code_dplyr"),
      tags$p("Expression:"),
      verbatimTextOutput(outputId = "code"),
      tags$p("Filtered data:"),
      verbatimTextOutput(outputId = "res_str")
    )
  )
)
)

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

  data <- reactive({
    get(input$dataset)
  })

  res_filter <- callModule(
    module = filterDF,
    id = "filtering",
    data_table = data,
    data_name = reactive(input$dataset)
  )

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

  output$table <- DT::renderDT({
    res_filter$data_filtered()
  }, options = list(pageLength = 5))

  output$code_dplyr <- renderPrint({
    res_filter$code$dplyr
  })
  output$code <- renderPrint({
    res_filter$code$expr
  })

  output$res_str <- renderPrint({

```

```
      str(res_filter$data_filtered())
    })
  }
  shinyApp(ui, server)
}
```

---

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

Run module example

### Usage

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

### Arguments

module            Module for which to see a demo.

### Examples

```
if (interactive()) {  
  
  # Demo for filterDF module  
  run_module("filterDF")  
  
}
```

---

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 <code>eval_tidy</code> 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)
}
```

**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)

}

```

---

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)
}

```

**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")  
)
```

**Arguments**

mapping	Aesthetics used in ggplot.
palette	Color palette
data	An optional data.frame to choose the right type for variables.
fill_type	Scale to use according to the variable used in fill: "discrete" or "continuous".
color_type	Scale to use according to the variable used in color: "discrete" or "continuous".

**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

actionButton, [21](#)

build\_aes, [2](#)

callModule, [23](#), [24](#)

chooseDataServer (module-chooseData), [20](#)

chooseDataUI (module-chooseData), [20](#)

coerceServer (module-coerce), [22](#)

coerceUI (module-coerce), [22](#)

colorPicker (input-colors), [15](#)

dragulaInput, [3](#)

dropInput, [5](#), [32](#)

esquisseContainer (esquisserServer), [8](#)

esquisser, [7](#)

esquisserServer, [8](#)

esquisserUI (esquisserServer), [8](#)

eval\_tidy, [29](#)

facet\_wrap, [12](#)

filterDF (module-filterDF), [24](#)

filterDF\_UI (module-filterDF), [24](#)

ggcall, [12](#)

ggplot\_build, [29](#)

ggplot\_to\_ppt, [14](#)

import, [21](#)

input-colors, [15](#)

match\_geom\_args, [19](#)

module-chooseData, [20](#)

module-coerce, [22](#)

module-esquisse (esquisserServer), [8](#)

module-filterDF, [24](#)

palettePicker (input-colors), [15](#)

pickerInput, [16](#)

potential\_geoms, [27](#)

reactive, [24](#), [25](#)

reactiveValues, [21](#), [25](#)

run\_module, [28](#)

safe\_ggplot, [28](#)

shiny::selectizeInput, [25](#)

shinyWidgets::pickerInput, [25](#)

theme, [12](#)

updateDragulaInput, [4](#), [29](#)

updateDropInput, [6](#), [31](#)

validateCssUnit, [9](#)

viewer, [7](#)

which\_pal\_scale, [33](#)