

Package ‘datamods’

November 23, 2021

Title Modules to Import and Manipulate Data in 'Shiny'

Version 1.2.0

Description 'Shiny' modules to import data into an application or 'addin' from various sources, and to manipulate them after that.

License GPL-3

URL <https://github.com/dreamRs/datamods>

BugReports <https://github.com/dreamRs/datamods/issues>

Encoding UTF-8

RoxygenNote 7.1.2

Imports data.table, DT, htmltools, phosphoricons, readxl, rio, rlang, shiny (>= 1.5.0), shinyWidgets (>= 0.5.3), tibble, tools

Suggests jsonlite, knitr, MASS, rmarkdown, testthat, validate

VignetteBuilder knitr

NeedsCompilation no

Author Victor Perrier [aut, cre, cph],
Fanny Meyer [aut],
Zauad Shahreer Abeer [aut]

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

Repository CRAN

Date/Publication 2021-11-23 19:30:07 UTC

R topics documented:

filter-data	2
get_data_packages	5
i18n	6
import-copypaste	7
import-file	9
import-globalenv	11
import-goolesheets	13

import-modal	15
list_pkg_data	17
show_data	18
update-variables	19
validation_ui	21

Index	24
--------------	-----------

filter-data	<i>Shiny module to interactively filter a data.frame</i>
-------------	--

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

```
filter_data_ui(id, show_nrow = TRUE, max_height = NULL)
```

```
filter_data_server(
  id,
  data = reactive(NULL),
  vars = reactive(NULL),
  name = reactive("data"),
  drop_ids = TRUE,
  widget_char = c("select", "picker"),
  widget_num = c("slider", "range"),
  widget_date = c("slider", "range"),
  label_na = "NA"
)
```

Arguments

id	Module id. See callModule .
show_nrow	Show number of filtered rows and total.
max_height	Maximum height for filters panel, useful if you have many variables to filter and limited space.
data	reactive function returning a data.frame to filter.
vars	reactive function returning a character vector of variables for which to add a filter. If a named list, names are used as labels.
name	reactive function returning a character string representing data name, only used for code generated.
drop_ids	Drop columns containing more than 90% of unique values, or than 50 distinct values.

widget_char	Widget to use for character variables: <code>shinyWidgets::pickerInput</code> or <code>shiny::selectizeInput</code> (default).
widget_num	Widget to use for numeric variables: <code>shinyWidgets::numericRangeInput</code> or <code>shiny::sliderInput</code> (default).
widget_date	Widget to use for date/time variables: <code>shiny::dateRangeInput</code> or <code>shiny::sliderInput</code> (default).
label_na	Label for missing value widget.

Value

- UI: HTML tags that can be included in shiny's UI
- Server: a list with three slots:
 - **filtered**: a reactive function returning the data filtered.
 - **code**: a reactive function returning the dplyr pipeline to filter data.
 - **expr**: a reactive function returning an expression to filter data.

Examples

```
library(shiny)
library(shinyWidgets)
library(datamods)
library(MASS)

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

datetime <- data.frame(
  date = seq(Sys.Date(), by = "day", length.out = 300),
  datetime = seq(Sys.time(), by = "hour", length.out = 300),
  num = sample.int(1e5, 300)
)

one_column_numeric <- data.frame(
  var1 = rnorm(100)
)

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

  radioButtons(
    inputId = "dataset",
    label = "Data:",
    choices = c(
```

```

      "iris",
      "mtcars",
      "mtcars_na",
      "Cars93",
      "datetime",
      "one_column_numeric"
    ),
    inline = TRUE
  ),

  fluidRow(
    column(
      width = 3,
      filter_data_ui("filtering", max_height = "500px")
    ),
    column(
      width = 9,
      progressBar(
        id = "pbar", value = 100,
        total = 100, display_pct = TRUE
      ),
      DT::dataTableOutput(outputId = "table"),
      tags$b("Code dplyr:"),
      verbatimTextOutput(outputId = "code_dplyr"),
      tags$b("Expression:"),
      verbatimTextOutput(outputId = "code"),
      tags$b("Filtered data:"),
      verbatimTextOutput(outputId = "res_str")
    )
  )
)

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

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

  vars <- reactive({
    if (identical(input$dataset, "mtcars")) {
      setNames(as.list(names(mtcars)[1:5]), c(
        "Miles/(US) gallon",
        "Number of cylinders",
        "Displacement (cu.in.)",
        "Gross horsepower",
        "Rear axle ratio"
      ))
    } else {
      NULL
    }
  })

  res_filter <- filter_data_server(

```

```
    id = "filtering",
    data = data,
    name = reactive(input$dataset),
    vars = vars,
    widget_num = "slider",
    widget_date = "slider",
    label_na = "Missing"
  )

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

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

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

  output$res_str <- renderPrint({
    str(res_filter$filtered())
  })
}

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

get_data_packages *Get packages containing datasets*

Description

Get packages containing datasets

Usage

```
get_data_packages()
```

Value

a character vector of packages names

Examples

```
if (interactive()) {
  get_data_packages()
}
```

i18n

Internationalization

Description

Simple mechanism to translate labels in a Shiny application.

Usage

```
i18n(x, translations = i18n_translations())

i18n_translations(package = packageName(parent.frame(2)))

set_i18n(value, packages = c("datamods", "esquisse"))
```

Arguments

x	Label to translate.
translations	Either a list or a data.frame with translations.
package	Name of the package where the function is called, use NULL outside a package. It will retrieve option "i18n.<PACKAGE>" (or "i18n" if no package) to returns appropriate labels.
value	Value to set for translation. Can be: <ul style="list-style-type: none"> • single character to use a supported language ("fr", "mk", "sq", "pt" for esquisse and datamods packages). • a list with labels as names and translations as values. • a data.frame with 2 column: label & translation. • path to a CSV file with same structure as for data.frame above.
packages	Name of packages for which to set i18n, default to esquisse and datamods

Value

i18n() returns a character, i18n_translations() returns a list or a data.frame.

Examples

```
library(datamods)

# Use with an objet
my.translations <- list(
  "Hello" = "Bonjour"
)
i18n("Hello", my.translations)

# Use with options()
options("i18n" = list(
  "Hello" = "Bonjour"
))
i18n("Hello")

# With a package
options("datamods.i18n" = "fr")
i18n("Browse...", translations = i18n_translations("datamods"))
# If you call i18n() from within a function of your package
# you don't need second argument, e.g.:
# i18n("Browse...")
```

import-copypaste *Import data with copy & paste*

Description

Let the user copy data from Excel or text file then paste it into a text area to import it.

Usage

```
import_copypaste_ui(id, title = TRUE)

import_copypaste_server(
  id,
  btn_show_data = TRUE,
  trigger_return = c("button", "change"),
  return_class = c("data.frame", "data.table", "tbl_df"),
  reset = reactive(NULL)
)
```

Arguments

<code>id</code>	Module's ID.
<code>title</code>	Module's title, if TRUE use the default title, use NULL for no title or a shiny.tag for a custom one.
<code>btn_show_data</code>	Display or not a button to display data in a modal window if import is successful.

<code>trigger_return</code>	When to update selected data: "button" (when user click on button) or "change" (each time user select a dataset in the list).
<code>return_class</code>	Class of returned data: <code>data.frame</code> , <code>data.table</code> or <code>tbl_df</code> (tibble).
<code>reset</code>	A reactive function that when triggered resets the data.

Value

- UI: HTML tags that can be included in shiny's UI
- Server: a list with three slots:
 - **status**: a reactive function returning the status: NULL, error or success.
 - **name**: a reactive function returning the name of the imported data as character.
 - **data**: a reactive function returning the imported data.frame.

Examples

```
library(shiny)
library(datamods)

ui <- fluidPage(
  tags$h3("Import data with copy & paste"),
  fluidRow(
    column(
      width = 4,
      import_copypaste_ui("myid")
    ),
    column(
      width = 8,
      tags$b("Import status:"),
      verbatimTextOutput(outputId = "status"),
      tags$b("Name:"),
      verbatimTextOutput(outputId = "name"),
      tags$b("Data:"),
      verbatimTextOutput(outputId = "data")
    )
  )
)

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

  imported <- import_copypaste_server("myid")

  output$status <- renderPrint({
    imported$status()
  })
  output$name <- renderPrint({
    imported$name()
  })
  output$data <- renderPrint({
    imported$data()
  })
}
```



```

    })
  }

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

```

import-file

Import data from a file

Description

Let user upload a file and import data

Usage

```

import_file_ui(
  id,
  title = TRUE,
  preview_data = TRUE,
  file_extensions = c(".csv", ".txt", ".xls", ".xlsx", ".rds", ".fst", ".sas7bdat",
    ".sav")
)

import_file_server(
  id,
  btn_show_data = TRUE,
  trigger_return = c("button", "change"),
  return_class = c("data.frame", "data.table", "tbl_df"),
  reset = reactive(NULL),
  read_fns = list()
)

```

Arguments

id	Module's ID.
title	Module's title, if TRUE use the default title, use NULL for no title or a shiny.tag for a custom one.
preview_data	Show or not a preview of the data under the file input.
file_extensions	File extensions accepted by <code>shiny::fileInput()</code> , can also be MIME type.
btn_show_data	Display or not a button to display data in a modal window if import is successful.
trigger_return	When to update selected data: "button" (when user click on button) or "change" (each time user select a dataset in the list).
return_class	Class of returned data: <code>data.frame</code> , <code>data.table</code> or <code>tbl_df</code> (tibble).
reset	A reactive function that when triggered resets the data.

- `read_fns` Named list with custom function(s) to read data:
- the name must be the extension of the files to which the function will be applied
 - the value must be a function that can have 4 arguments, passed by user through the interface:
 - `file`: path to the file
 - `sheet`: for Excel files, sheet to read
 - `skip`: number of row to skip
 - `encoding`: file encoding

Value

- UI: HTML tags that can be included in shiny's UI
- Server: a list with three slots:
 - **status**: a reactive function returning the status: NULL, error or success.
 - **name**: a reactive function returning the name of the imported data as character.
 - **data**: a reactive function returning the imported data.frame.

Examples

```
library(shiny)
library(datamods)

ui <- fluidPage(
  tags$h3("Import data from a file"),
  fluidRow(
    column(
      width = 4,
      import_file_ui(
        id = "myid",
        file_extensions = c(".csv", ".txt", ".xls", ".xlsx", ".json")
      )
    ),
    column(
      width = 8,
      tags$b("Import status:"),
      verbatimTextOutput(outputId = "status"),
      tags$b("Name:"),
      verbatimTextOutput(outputId = "name"),
      tags$b("Data:"),
      verbatimTextOutput(outputId = "data")
    )
  )
)

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

```

imported <- import_file_server(
  id = "myid",
  # Custom functions to read data
  read_fns = list(
    xls = function(file, sheet, skip, encoding) {
      readxl::read_xls(path = file, sheet = sheet, skip = skip)
    },
    json = function(file) {
      jsonlite::read_json(file, simplifyVector = TRUE)
    }
  )
)

output$status <- renderPrint({
  imported$status()
})
output$name <- renderPrint({
  imported$name()
})
output$data <- renderPrint({
  imported$data()
})

}

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

```

import-globalenv

Import data from an Environment

Description

Let the user select a dataset from its own environment or from a package's environment.

Usage

```

import_globalenv_ui(
  id,
  globalenv = TRUE,
  packages = get_data_packages(),
  title = TRUE
)

import_globalenv_server(
  id,
  btn_show_data = TRUE,
  trigger_return = c("button", "change"),
  return_class = c("data.frame", "data.table", "tbl_df"),

```

```

  reset = reactive(NULL)
)

```

Arguments

<code>id</code>	Module's ID.
<code>globalenv</code>	Search for data in Global environment.
<code>packages</code>	Name of packages in which to search data.
<code>title</code>	Module's title, if TRUE use the default title, use NULL for no title or a shiny tag for a custom one.
<code>btn_show_data</code>	Display or not a button to display data in a modal window if import is successful.
<code>trigger_return</code>	When to update selected data: "button" (when user click on button) or "change" (each time user select a dataset in the list).
<code>return_class</code>	Class of returned data: <code>data.frame</code> , <code>data.table</code> or <code>tbl_df</code> (tibble).
<code>reset</code>	A reactive function that when triggered resets the data.

Value

- UI: HTML tags that can be included in shiny's UI
- Server: a list with three slots:
 - **status**: a reactive function returning the status: NULL, error or success.
 - **name**: a reactive function returning the name of the imported data as character.
 - **data**: a reactive function returning the imported data.frame.

Examples

```

if (interactive()) {
  library(shiny)
  library(datamods)

  # Create some data.frames

  my_df <- data.frame(
    variable1 = sample(letters, 20, TRUE),
    variable2 = sample(1:100, 20, TRUE)
  )

  results_analysis <- data.frame(
    id = sample(letters, 20, TRUE),
    measure = sample(1:100, 20, TRUE),
    response = sample(1:100, 20, TRUE)
  )

  # Application

  ui <- fluidPage(
    fluidRow(

```

```

      column(
        width = 4,
        import_globalenv_ui("myid")
      ),
      column(
        width = 8,
        tags$b("Import status:"),
        verbatimTextOutput(outputId = "status"),
        tags$b("Name:"),
        verbatimTextOutput(outputId = "name"),
        tags$b("Data:"),
        verbatimTextOutput(outputId = "data")
      )
    )
  )
}

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

  imported <- import_globalenv_server("myid")

  output$status <- renderPrint({
    imported$status()
  })
  output$name <- renderPrint({
    imported$name()
  })
  output$data <- renderPrint({
    imported$data()
  })

}

shinyApp(ui, server)
}

```

import-googlesheets *Import data from Google sheet*

Description

Let user paste link to a Google sheet then import the data.

Usage

```

import_googlesheets_ui(id, title = TRUE)

import_googlesheets_server(
  id,
  btn_show_data = TRUE,
  trigger_return = c("button", "change"),

```

```

  return_class = c("data.frame", "data.table", "tbl_df"),
  reset = reactive(NULL)
)

```

Arguments

<code>id</code>	Module's ID.
<code>title</code>	Module's title, if TRUE use the default title, use NULL for no title or a shiny.tag for a custom one.
<code>btn_show_data</code>	Display or not a button to display data in a modal window if import is successful.
<code>trigger_return</code>	When to update selected data: "button" (when user click on button) or "change" (each time user select a dataset in the list).
<code>return_class</code>	Class of returned data: data.frame, data.table or tbl_df (tibble).
<code>reset</code>	A reactive function that when triggered resets the data.

Value

- UI: HTML tags that can be included in shiny's UI
- Server: a list with three slots:
 - **status**: a reactive function returning the status: NULL, error or success.
 - **name**: a reactive function returning the name of the imported data as character.
 - **data**: a reactive function returning the imported data.frame.

Examples

```

library(shiny)
library(datamods)

ui <- fluidPage(
  tags$h3("Import data from Googlesheets"),
  fluidRow(
    column(
      width = 4,
      import_googlesheets_ui("myid")
    ),
    column(
      width = 8,
      tags$b("Import status:"),
      verbatimTextOutput(outputId = "status"),
      tags$b("Name:"),
      verbatimTextOutput(outputId = "name"),
      tags$b("Data:"),
      verbatimTextOutput(outputId = "data")
    )
  )
)

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

```

```

imported <- import_googlesheets_server("myid")

output$status <- renderPrint({
  imported$status()
})
output$name <- renderPrint({
  imported$name()
})
output$data <- renderPrint({
  imported$data()
})

}

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

```

import-modal

Get all import functions in a Modal

Description

Let the user choose a method of his choice to import data

Usage

```

import_ui(id, from = c("env", "file", "coppypaste", "googlesheets"))

import_server(
  id,
  validation_opts = NULL,
  allowed_status = c("OK", "Failed", "Error"),
  return_class = c("data.frame", "data.table", "tbl_df")
)

import_modal(id, from, title = "Import data", size = "l")

```

Arguments

id	Module's id
from	The import_ui & server to use, i.e. the method. There are 4 options to choose from. ("env", "file", "coppypaste", "googlesheets")
validation_opts	list of arguments passed to [validation_server()].
allowed_status	Vector of statuses allowed to confirm dataset imported, if you want that all validation rules are successful before importing data use allowed_status = "OK".

return_class Class of returned data: `data.frame`, `data.table` or `tbl_df` (tibble).
title Modal window title.
size Modal window size, default to "l" (large).

Value

- UI: HTML tags that can be included in shiny's UI
- Server: a list with two slots:
 - **data**: a reactive function returning the imported data frame.
 - **name**: a reactive function returning the name of the imported data as character (if applicable).

Examples

```

library(shiny)
library(datamods)

ui <- fluidPage(
  fluidRow(
    column(
      width = 4,
      checkboxGroupInput(
        inputId = "from",
        label = "From",
        choices = c("env", "file", "coppypaste", "googlesheets"),
        selected = c("file", "coppypaste")
      ),
      actionButton("launch_modal", "Launch modal window")
    ),
    column(
      width = 8,
      tags$b("Imported data:"),
      verbatimTextOutput(outputId = "name"),
      verbatimTextOutput(outputId = "data")
    )
  )
)

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

  observeEvent(input$launch_modal, {
    req(input$from)
    import_modal(
      id = "myid",
      from = input$from,
      title = "Import data to be used in application"
    )
  })

  imported <- import_server("myid", return_class = "tbl_df")

```



```
output$name <- renderPrint({
  req(imported$name())
  imported$name()
})

output$data <- renderPrint({
  req(imported$data())
  imported$data()
})
}

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

list_pkg_data

List dataset contained in a package

Description

List dataset contained in a package

Usage

```
list_pkg_data(pkg)
```

Arguments

pkg Name of the package, must be installed.

Value

a character vector or NULL.

Examples

```
list_pkg_data("ggplot2")
```

show_data	<i>Display a table in a window</i>
-----------	------------------------------------

Description

Display a table in a window

Usage

```
show_data(  
  data,  
  title = NULL,  
  options = NULL,  
  show_classes = TRUE,  
  type = c("popup", "modal"),  
  width = "80%"  
)
```

Arguments

data	a data object (either a matrix or a data.frame).
title	Title to be displayed in window.
options	Options passed to datatable 's options argument.
show_classes	Show variables classes under variables names in table header.
type	Display table in a pop-up or in modal window.
width	Width of the window, only used if type = "popup".

Value

No value.

Examples

```
library(shiny)  
library(datamods)  
  
ui <- fluidPage(  
  actionButton(  
    inputId = "show1",  
    label = "Show data in popup",  
    icon = icon("eye")  
  ),  
  actionButton(  
    inputId = "show2",  
    label = "Show data in modal",  
    icon = icon("eye")  
  )  
)
```

```
),
  actionButton(
    inputId = "show3",
    label = "Show data without classes",
    icon = icon("eye")
  )
)

server <- function(input, output, session) {
  observeEvent(input$show1, {
    show_data(mtcars, title = "My data")
  })
  observeEvent(input$show2, {
    show_data(mtcars, title = "My data", type = "modal")
  })
  observeEvent(input$show3, {
    show_data(mtcars, title = "My data", show_classes = FALSE)
  })
}

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

update-variables

Select, rename and convert variables

Description

Select, rename and convert variables

Usage

```
update_variables_ui(id, title = TRUE)
```

```
update_variables_server(id, data, height = NULL)
```

Arguments

id	Module's ID
title	Module's title, if TRUE use the default title, use NULL for no title or a shiny.tag for a custom one.
data	a data.frame or a reactive function returning a data.frame.
height	Height for the table.

Value

A reactive function returning the updated data.

Examples

```

library(shiny)
library(datamods)

testdata <- data.frame(
  date_as_char = as.character(Sys.Date() + 0:9),
  date_as_num = as.numeric(Sys.Date() + 0:9),
  datetime_as_char = as.character(Sys.time() + 0:9 * 3600*24),
  datetime_as_num = as.numeric(Sys.time() + 0:9 * 3600*24),
  num_as_char = as.character(1:10),
  char = month.name[1:10],
  char_na = c("A", "A", "B", NA, "B", "A", NA, "B", "A", "B"),
  stringsAsFactors = FALSE
)

ui <- fluidPage(
  tags$h3("Select, rename and convert variables"),
  fluidRow(
    column(
      width = 6,
      # radioButtons()
      update_variables_ui("vars")
    ),
    column(
      width = 6,
      tags$b("original data:"),
      verbatimTextOutput("original"),
      verbatimTextOutput("original_str"),
      tags$b("Modified data:"),
      verbatimTextOutput("modified"),
      verbatimTextOutput("modified_str")
    )
  )
)

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

  updated_data <- update_variables_server(
    id = "vars",
    data = reactive(testdata)
  )

  output$original <- renderPrint({
    testdata
  })
  output$original_str <- renderPrint({
    str(testdata)
  })

  output$modified <- renderPrint({
    updated_data()
  })
}

```

```

    })
    output$modified_str <- renderPrint({
      str(updated_data())
    })
  }

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

```

validation_ui

Validation module

Description

Check that a dataset respect some validation expectations.

Usage

```
validation_ui(id, display = c("dropdown", "inline"), max_height = NULL, ...)
```

```

validation_server(
  id,
  data,
  n_row = NULL,
  n_col = NULL,
  n_row_label = "Valid number of rows",
  n_col_label = "Valid number of columns",
  btn_label = "Dataset validation:",
  rules = NULL
)

```

Arguments

<code>id</code>	Module's ID.
<code>display</code>	Display validation results in a dropdown menu by clicking on a button or display results directly in interface.
<code>max_height</code>	Maximum height for validation results element, useful if you have many rules.
<code>...</code>	Arguments passed to <code>actionButton</code> or <code>uiOutput</code> depending on display mode, you cannot use <code>inputId/outputId</code> , <code>label</code> or <code>icon</code> (button only).
<code>data</code>	a reactive function returning a <code>data.frame</code> .
<code>n_row</code> , <code>n_col</code>	A one-sided formula to check number of rows and columns respectively, see below for examples.
<code>n_row_label</code> , <code>n_col_label</code>	Text to be displayed with the result of the check for number of rows/columns.
<code>btn_label</code>	Label for the dropdown button, will be followed by validation result.
<code>rules</code>	An object of class <code>validator</code> created with <code>validate::validator</code> .

Value

- UI: HTML tags that can be included in shiny's UI
- Server: a list with two slots:
 - **status**: a reactive function returning the best status available between "OK", "Failed" or "Error".
 - **details**: a reactive function returning a list with validation details.

Examples

```
library(datamods)
library(shiny)

if (requireNamespace("validate")) {
  library(validate)

  # Define some rules to be applied to data
  myrules <- validator(
    is.character(Manufacturer) | is.factor(Manufacturer),
    is.numeric(Price),
    Price > 12, # we should use 0 for testing positivity, but that's for the example
    !is.na(Luggage.room),
    in_range(Cylinders, min = 4, max = 8),
    Man.trans.avail %in% c("Yes", "No")
  )
  # Add some labels
  label(myrules) <- c(
    "Variable Manufacturer must be character",
    "Variable Price must be numeric",
    "Variable Price must be strictly positive",
    "Luggage.room must not contain any missing values",
    "Cylinders must be between 4 and 8",
    "Man.trans.avail must be 'Yes' or 'No'"
  )
  # you can also add a description()

  ui <- fluidPage(
    tags$h2("Validation"),
    fluidRow(
      column(
        width = 4,
        radioButtons(
          inputId = "dataset",
          label = "Choose dataset:",
          choices = c("mtcars", "MASS::Cars93")
        ),
        tags$p("Dropdown example:"),
        validation_ui("validation1"),

        tags$br(),

        tags$p("Inline example:"),
```

```
      validation_ui("validation2", display = "inline")
    ),
    column(
      width = 8,
      tags$b("Status:"),
      verbatimTextOutput("status"),
      tags$b("Details:"),
      verbatimTextOutput("details")
    )
  )
)
)

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

  dataset <- reactive({
    if (input$dataset == "mtcars") {
      mtcars
    } else {
      MASS::Cars93
    }
  })

  results <- validation_server(
    id = "validation1",
    data = dataset,
    n_row = ~ . > 20, # more than 20 rows
    n_col = ~ . >= 3, # at least 3 columns
    rules = myrules
  )

  validation_server(
    id = "validation2",
    data = dataset,
    n_row = ~ . > 20, # more than 20 rows
    n_col = ~ . >= 3, # at least 3 columns
    rules = myrules
  )

  output$status <- renderPrint(results$status())
  output$details <- renderPrint(results$details())

}

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

Index

callModule, 2

datatable, 18

filter-data, 2

filter_data_server (filter-data), 2

filter_data_ui (filter-data), 2

get_data_packages, 5

i18n, 6

i18n_translations (i18n), 6

import-copypaste, 7

import-file, 9

import-globalenv, 11

import-goolesheets, 13

import-modal, 15

import_copypaste_server
(import-copypaste), 7

import_copypaste_ui (import-copypaste),
7

import_file_server (import-file), 9

import_file_ui (import-file), 9

import_globalenv_server
(import-globalenv), 11

import_globalenv_ui (import-globalenv),
11

import_goolesheets_server
(import-goolesheets), 13

import_goolesheets_ui
(import-goolesheets), 13

import_modal (import-modal), 15

import_server (import-modal), 15

import_ui (import-modal), 15

list_pkg_data, 17

reactive, 2

set_i18n (i18n), 6

shiny::dateRangeInput, 3

shiny::fileInput(), 9

shiny::selectizeInput, 3

shiny::sliderInput, 3

shinyWidgets::numericRangeInput, 3

shinyWidgets::pickerInput, 3

show_data, 18

update-variables, 19

update_variables_server
(update-variables), 19

update_variables_ui (update-variables),
19

validation_server (validation_ui), 21

validation_ui, 21