Package: datasetviewer (via r-universe)

July 10, 2026

Title 'SAS Studio'-Style Interactive Dataset Viewer

Version 0.1.1

Description An interactive dataset viewer that renders a fast, scrollable grid with a column-selection panel, per-column property metadata, and a names-versus-labels header toggle, modelled on the 'SAS Studio' table viewer. Runs from one codebase in interactive 'Shiny' apps and in static HTML documents, with a free-text row filter, header sort, and CSV export, and handles large datasets without row sampling by querying them in the browser with 'DuckDB-WASM'.

License MIT + file LICENSE

URL https://github.com/vthanik/datasetviewer, https://vthanik.github.io/datasetviewer/

BugReports https://github.com/vthanik/datasetviewer/issues

Encoding UTF-8

Roxygen list(markdown = TRUE)

Imports cli, htmltools, htmlwidgets, jsonlite, nanoparquet, rlang

Suggests artoo, dplyr, hms, knitr, quarto, shiny, testthat (>= 3.0.0), tibble, withr

VignetteBuilder quarto

Config/testthat/edition 3

Config/roxygen2/version 8.0.0

Config/pak/sysreqs cmake make libuv1-dev

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

Date/Publication 2026-06-30 17:00:26 UTC

RemoteUrl https://github.com/vthanik/datasetviewer

RemoteRef HEAD

RemoteSha 9b0efb1a36ce0b4b6868b2984335e5aba021efb2

Contents

`dataset_viewer` *View a dataset in an interactive SAS Studio-style grid*

Description

Renders a fast, scrollable data grid with a column-selection panel and per-column property metadata. The same widget renders in interactive Shiny apps and in static HTML documents.

Usage

```
dataset_viewer(
  x,
  ...,
  view = c("names", "labels"),
  width = NULL,
  height = NULL,
  elementId = NULL
)
```

Arguments

`x`	*Dataset to view.* `<data.frame	character(1)>`. A data frame, or a file path read via `artoo::read_dataset()` (xpt, Dataset-JSON, NDJSON, 'Parquet', RDS). An artoo-conformed frame supplies labels, formats, and lengths to the property panel; a plain frame uses synthesized metadata.
`...`	Reserved for future arguments.
`view`	*Initial header mode.* `<character(1)>`. `"names"` (default, matching SAS Studio) shows column names as headers; `"labels"` shows labels, falling back to names when a label is absent.
`width, height`	*Widget sizing.* `<character(1)	numeric(1)	NULL>`. Passed through to `htmlwidgets::createWidget()`.
`elementId`	*Explicit DOM id.* `<character(1)	NULL>`. Usually left `NULL` so htmlwidgets assigns one.

Details

Query engine. The data is sent to the browser once as 'Parquet' and queried in place with DuckDB-WASM, so filter, sort, and paging run over the whole dataset with no row sampling. The engine (~35 MB) loads from a CDN by default but is fetched into the package at install time when reachable, so a Shiny app can serve it to browsers with no internet at runtime. Set `options(datasetviewer.use_local_engine`

= FALSE) to force the CDN (for small self-contained HTML). See vignette("datasetviewer") for offline and corporate deployment, including the DATASETVIEWER_DUCKDB_* install-time environment variables.

Value

An htmlwidget. Print it to render, or use it as a Shiny output.

See Also

Shiny bindings: datasetviewerOutput(), renderDatasetViewer().

Examples

```
# ---- Example 1: view a plain data frame ----
#
# Wrap any data frame to get the interactive grid. Printing the widget in
# an interactive session or a rendered document shows it; here we inspect
# the payload so the example stays headless and self-contained (printing a
# widget would launch a browser under R CMD check).
viewer <- dataset_viewer(mtcars)
viewer$x$n_rows

# ---- Example 2: CDISC labels as headers ----
#
# With the sibling artoo package, a CDISC-conformed frame supplies column
# labels, formats, and storage lengths to the property pane and the
# names-versus-labels header toggle. Start on labels with view = "labels".
if (requireNamespace("artoo", quietly = TRUE)) {
  labelled <- dataset_viewer(artoo::cdisc_adsl, view = "labels")
  labelled$x$columns[[1]]$label
}
```

---

datasetviewer-shiny *Shiny bindings for the dataset viewer*

Description

Output and render functions to embed dataset_viewer() in a Shiny app. The widget pushes its live view state back to Shiny as inputs, so the app can reuse the user's filter, sort, and column selection server-side.

Usage

```
datasetviewerOutput(outputId, width = "100%", height = "500px")

renderDatasetViewer(expr, env = parent.frame(), quoted = FALSE)
```

Arguments

outputId	*Output slot id.* <character(1)>. Matched by datasetviewerOutput() and renderDatasetViewer().
width, height	*CSS sizing.* <character(1)>. Any valid CSS size; the grid fills the element.
expr	*Render expression.* A call that returns a dataset_viewer() widget.
env, quoted	*Evaluation control.* Standard htmlwidgets render plumbing; leave at their defaults.

Details

Inputs published. For an output with id "viewer" the widget sets, on every change:

- input$viewer_columns *(<character>)*: selected column names.
- input$viewer_filter *(<character(1)>)*: the filter expression.
- input$viewer_sort *(<list>)*: sort keys (name, dir).
- input$viewer_view *(<character(1)>)*: "names" or "labels".

Value

datasetviewerOutput() returns a Shiny output UI element; renderDatasetViewer() returns a Shiny render function.

See Also

dataset_viewer() for the widget these bindings embed.

Examples

```
# ---- Example 1: read the viewer's filter and selection server-side ----
#
# The app shows a dataset and echoes the filter expression and selected
# columns the user builds in the grid. Runs only in an interactive session.
if (interactive()) {
  library(shiny)
  ui <- fluidPage(
    datasetviewerOutput("viewer", height = "500px"),
    verbatimTextOutput("state")
  )
  server <- function(input, output, session) {
    output$viewer <- renderDatasetViewer(dataset_viewer(mtcars))
    output$state <- renderText({
      paste0(
        "filter: ", input$viewer_filter, "\n",
        "columns: ", paste(input$viewer_columns, collapse = ", ")
      )
    })
  }
  shinyApp(ui, server)
}
```

Index