Package 'perspectiveR'

March 30, 2026

Type Package

Title Interactive Pivot Tables and Visualizations with 'Perspective'

Version 0.3.0

Description An 'htmlwidgets' binding for the 'FINOS Perspective' <https://perspective-dev.github.io/> library, a high-performance 'WebAssembly'-powered data visualization engine. Provides interactive pivot tables, cross-tabulations, and multiple chart types (bar, line, scatter, heatmap, and more) that run entirely in the browser.
Supports self-service analytics with drag-and-drop column selection, group-by/split-by pivoting, filtering, sorting, aggregation, and computed expressions. Works in 'RStudio' Viewer, 'R Markdown', 'Quarto', and 'Shiny' with streaming data updates via proxy interface.

License Apache License (>= 2)

URL https://github.com/EydlinIlya/perspectiveR

BugReports https://github.com/EydlinIlya/perspectiveR/issues

Encoding UTF-8

RoxygenNote 7.3.3

Imports htmlwidgets (>= 1.6.0), htmltools, jsonlite

Suggests arrow, shiny (>= 1.1.0), testthat (>= 3.0.0), knitr, rmarkdown

VignetteBuilder knitr

Config/testthat/edition 3

NeedsCompilation no

Author Eli Eydlin [aut, cre]

Maintainer Eli Eydlin <ilyabeydlin@gmail.com>

Repository CRAN

Date/Publication 2026-03-30 09:00:03 UTC

Contents

`perspective` *Create a Perspective Interactive Viewer*

Description

Creates an interactive pivot table and visualization widget powered by the FINOS Perspective library. The viewer provides a self-service UI where users can interactively change chart types, group/split/filter/sort data, create computed columns, and configure aggregations.

Usage

```
perspective(
  data,
  columns = NULL,
  group_by = NULL,
  split_by = NULL,
  sort = NULL,
  filter = NULL,
  filter_op = NULL,
  expressions = NULL,
  aggregates = NULL,
  plugin = NULL,
  plugin_config = NULL,
  theme = "Pro Light",
  settings = TRUE,
```

```
  title = NULL,
  editable = FALSE,
  index = NULL,
  limit = NULL,
  use_arrow = FALSE,
  width = NULL,
  height = NULL,
  elementId = NULL
)
```

Arguments

data	A data.frame or matrix to display.
columns	Character vector of column names to show. If NULL, all columns are shown.
group_by	Character vector of column names to group rows by (row pivots).
split_by	Character vector of column names to split columns by (column pivots).
sort	A list of two-element vectors, each containing a column name and a direction ("asc", "desc", "col asc", "col desc", "asc abs", "desc abs", "col asc abs", "col desc abs"). For example: list(c("mpg", "desc")).
filter	A list of three-element vectors, each containing a column name, an operator ("==", "!=", ">", "<", ">=", "<=", "begins with", "contains", "ends with", "in", "not in", "is null", "is not null"), and a value. For example: list(c("cyl", "==", "6")).
filter_op	Character string controlling how multiple filters are combined: "and" (default) or "or". If NULL, the Perspective default ("and") is used.
expressions	Character vector of Perspective expression strings for computed columns. For example: c('"Profit" / "Sales"').
aggregates	A named list mapping column names to aggregate functions. For example: list(mpg = "avg", hp = "sum").
plugin	Character string specifying the visualization plugin. Options include: "Datagrid", "Y Bar", "X Bar", "Y Line", "X/Y Line", "Y Area", "Y Scatter", "XY Scatter", "Treemap", "Sunburst", "Heatmap".
plugin_config	A list of plugin-specific configuration options.
theme	Character string specifying the CSS theme. Options: "Pro Light", "Pro Dark", "Monokai", "Solarized Light", "Solarized Dark", "Vaporwave", "Dracula", "Gruvbox", "Gruvbox Dark". Default is "Pro Light".
settings	Logical; whether to show the settings/configuration panel sidebar. Default TRUE. This is the interactive UI where users drag-and-drop columns, change chart types, add filters, etc.
title	Character string for the viewer title. If NULL, no title is shown.
editable	Logical; whether the data in the grid is user-editable. Default FALSE.
index	Character string naming a column to use as the table's primary key. When set, psp_update() performs upserts (matching rows are updated instead of appended) and psp_remove() can delete rows by key. Must be the name of a column present in data. Default NULL (no index).

limit	Single positive integer specifying the maximum number of rows the table will hold. When new rows are added beyond this limit, the oldest rows are removed (rolling window). Mutually exclusive with index. Default NULL (no limit).
use_arrow	Logical; if TRUE, serialize data using Arrow IPC format (base64-encoded) for better performance with large datasets. Requires the arrow package. Default FALSE.
width	Widget width (CSS string or numeric pixels).
height	Widget height (CSS string or numeric pixels).
elementId	Optional explicit element ID for the widget.

Details

When used in a Shiny app, the following reactive inputs are available (where outputId is the ID passed to perspectiveOutput):

input$<outputId>_config Fires when the user changes the viewer configuration (columns, pivots, filters, etc.).

input$<outputId>_click Fires when the user clicks a cell or data point.

input$<outputId>_select Fires when the user selects rows or data points.

input$<outputId>_update Fires on each table data change when subscribed via psp_on_update.

input$<outputId>_export Contains exported data after calling psp_export.

input$<outputId>_state Contains saved viewer state after calling psp_save.

input$<outputId>_schema Contains the table schema after calling psp_schema.

input$<outputId>_size Contains the table row count after calling psp_size.

input$<outputId>_columns Contains the table column names after calling psp_columns.

input$<outputId>_validate_expressions Contains expression validation results after calling psp_validate_expressions.

Value

An htmlwidgets object that can be printed, included in R Markdown, Quarto documents, or Shiny apps.

Examples

```
# Basic data grid
perspective(mtcars)

# Bar chart grouped by cylinder count
perspective(mtcars, group_by = "cyl", plugin = "Y Bar")

# Filtered and sorted view
perspective(iris,
  columns = c("Sepal.Length", "Sepal.Width", "Species"),
  filter = list(c("Species", "==", "setosa")),
  sort = list(c("Sepal.Length", "desc"))
)
```

`perspectiveOutput` *Shiny Output for Perspective Viewer*

Description

Creates a Perspective viewer output element for use in a Shiny UI.

Usage

```
perspectiveOutput(outputId, width = "100%", height = "400px")
```

Arguments

`outputId`	Output variable name.
`width`	CSS width (default `"100%"`).
`height`	CSS height (default `"400px"`).

Details

The following reactive inputs are available (where `outputId` is the ID you supply):

`input$<outputId>_config` Viewer configuration changes.

`input$<outputId>_click` Cell/data-point click events.

`input$<outputId>_select` Row/data-point selection events.

`input$<outputId>_update` Table data changes (requires `psp_on_update`).

`input$<outputId>_export` Exported data (after `psp_export`).

`input$<outputId>_state` Saved viewer state (after `psp_save`).

`input$<outputId>_schema` Table schema (after `psp_schema`).

`input$<outputId>_size` Table row count (after `psp_size`).

`input$<outputId>_columns` Table column names (after `psp_columns`).

`input$<outputId>_validate_expressions` Expression validation results (after `psp_validate_expressions`).

Value

A Shiny output element.

Examples

```
if (interactive()) {
library(shiny)
ui <- fluidPage(
  perspectiveOutput("viewer", height = "600px")
)
}
```

perspectiveProxy *Create a Perspective Proxy Object for Shiny*

Description

Creates a proxy object that can be used to update an existing Perspective viewer in a Shiny app without re-rendering the entire widget. Use this with `psp_update`, `psp_replace`, `psp_clear`, `psp_restore`, and `psp_reset` to modify the viewer.

Usage

```
perspectiveProxy(session, outputId)
```

Arguments

`session`	The Shiny session object (usually `session`).
`outputId`	The output ID of the Perspective widget to control.

Value

A proxy object of class `"perspective_proxy"`.

Examples

```
if (interactive()) {
server <- function(input, output, session) {
  output$viewer <- renderPerspective({
    perspective(mtcars)
  })

  observeEvent(input$add_data, {
    proxy <- perspectiveProxy(session, "viewer")
    psp_update(proxy, new_data)
  })
}
}
```

psp_clear *Clear All Data from a Perspective Viewer*

Description

Removes all rows from the Perspective table (schema is preserved).

Usage

```
psp_clear(proxy)
```

Arguments

proxy	A perspectiveProxy object.

Value

The proxy object (invisibly), for chaining.

Examples

```
if (interactive()) {
proxy <- perspectiveProxy(session, "viewer")
psp_clear(proxy)
}
```

---

psp_columns *Get Table Column Names*

Description

Requests the column names of the Perspective table. The result is delivered asynchronously to `input$<outputId>_columns`.

Usage

```
psp_columns(proxy)
```

Arguments

proxy	A perspectiveProxy object.

Value

The proxy object (invisibly), for chaining.

Examples

```
if (interactive()) {
proxy <- perspectiveProxy(session, "viewer")
psp_columns(proxy)
}
```

psp_export *Export Data from a Perspective Viewer*

Description

Requests data export from the current Perspective view. The result is delivered asynchronously to `input$<outputId>_export`.

Usage

```
psp_export(
  proxy,
  format = c("json", "csv", "columns", "arrow"),
  start_row = NULL,
  end_row = NULL,
  start_col = NULL,
  end_col = NULL
)
```

Arguments

`proxy`	A `perspectiveProxy` object.
`format`	Export format: `"json"` (default), `"csv"`, `"columns"`, or `"arrow"` (base64-encoded Arrow IPC).
`start_row`	Optional single numeric value specifying the first row (0-based) to include in the export.
`end_row`	Optional single numeric value specifying the row (0-based, exclusive) at which to stop.
`start_col`	Optional single numeric value specifying the first column (0-based) to include.
`end_col`	Optional single numeric value specifying the column (0-based, exclusive) at which to stop.

Value

The proxy object (invisibly), for chaining.

Examples

```
if (interactive()) {
proxy <- perspectiveProxy(session, "viewer")
psp_export(proxy, format = "csv")
}
```

psp_on_update *Subscribe to Table Update Events*

Description

Enables or disables a subscription to table data changes. When enabled, every `table.update()` triggers `input$<outputId>_update` with a list containing `timestamp`, `port_id`, and `source` (`"edit"` for user edits, `"api"` for programmatic updates).

Usage

```
psp_on_update(proxy, enable = TRUE)
```

Arguments

`proxy`	A `perspectiveProxy` object.
`enable`	Logical; `TRUE` to subscribe, `FALSE` to unsubscribe. Default `TRUE`.

Value

The proxy object (invisibly), for chaining.

Examples

```
if (interactive()) {
proxy <- perspectiveProxy(session, "viewer")
psp_on_update(proxy)
}
```

psp_remove *Remove Rows by Key from a Perspective Viewer*

Description

Removes rows matching the given primary-key values from an indexed Perspective table. The table must have been created with an `index` column (see `perspective`).

Usage

```
psp_remove(proxy, keys)
```

Arguments

`proxy`	A `perspectiveProxy` object.
`keys`	A vector of key values identifying the rows to remove.

Value

The proxy object (invisibly), for chaining.

Examples

```
if (interactive()) {
proxy <- perspectiveProxy(session, "viewer")
psp_remove(proxy, keys = c("row1", "row2"))
}
```

`psp_replace` *Replace All Data in a Perspective Viewer*

Description

Replaces the entire dataset in the Perspective table.

Usage

```
psp_replace(proxy, data, use_arrow = FALSE)
```

Arguments

`proxy`	A `perspectiveProxy` object.
`data`	A data.frame containing the replacement data.
`use_arrow`	Logical; use Arrow IPC serialization. Default `FALSE`.

Value

The proxy object (invisibly), for chaining.

Examples

```
if (interactive()) {
proxy <- perspectiveProxy(session, "viewer")
psp_replace(proxy, updated_data)
}
```

psp_reset *Reset Viewer to Default State*

Description

Resets the Perspective viewer to its default configuration.

Usage

```
psp_reset(proxy)
```

Arguments

proxy	A perspectiveProxy object.

Value

The proxy object (invisibly), for chaining.

Examples

```
if (interactive()) {
proxy <- perspectiveProxy(session, "viewer")
psp_reset(proxy)
}
```

psp_restore *Restore Viewer Configuration*

Description

Applies a configuration object to the Perspective viewer, changing columns, group_by, split_by, filters, sort, aggregates, plugin, etc.

Usage

```
psp_restore(proxy, config)
```

Arguments

proxy	A perspectiveProxy object.
config	A list of Perspective viewer configuration options.

Value

The proxy object (invisibly), for chaining.

Examples

```
if (interactive()) {
proxy <- perspectiveProxy(session, "viewer")
psp_restore(proxy, list(plugin = "Y Bar", group_by = list("cyl")))
}
```

`psp_save` *Save Viewer State*

Description

Requests the current viewer configuration (columns, pivots, filters, sort, plugin, etc.). The result is delivered asynchronously to `input$<outputId>_state`.

Usage

```
psp_save(proxy)
```

Arguments

`proxy`	A `perspectiveProxy` object.

Value

The proxy object (invisibly), for chaining.

Examples

```
if (interactive()) {
proxy <- perspectiveProxy(session, "viewer")
psp_save(proxy)
}
```

`psp_schema` *Get Table Schema*

Description

Requests the schema (column names and types) of the Perspective table. The result is delivered asynchronously to `input$<outputId>_schema`.

Usage

```
psp_schema(proxy)
```

Arguments

proxy	A perspectiveProxy object.

Value

The proxy object (invisibly), for chaining.

Examples

```
if (interactive()) {
proxy <- perspectiveProxy(session, "viewer")
psp_schema(proxy)
}
```

---

psp_size *Get Table Row Count*

Description

Requests the number of rows in the Perspective table. The result is delivered asynchronously to `input$<outputId>_size`.

Usage

```
psp_size(proxy)
```

Arguments

proxy	A perspectiveProxy object.

Value

The proxy object (invisibly), for chaining.

Examples

```
if (interactive()) {
proxy <- perspectiveProxy(session, "viewer")
psp_size(proxy)
}
```

psp_update *Update (Append) Data in a Perspective Viewer*

Description

Sends new rows to be appended to the existing Perspective table.

Usage

```
psp_update(proxy, data, use_arrow = FALSE)
```

Arguments

proxy	A perspectiveProxy object.
data	A data.frame of new rows to append.
use_arrow	Logical; use Arrow IPC serialization. Default FALSE.

Value

The proxy object (invisibly), for chaining.

Examples

```
if (interactive()) {
proxy <- perspectiveProxy(session, "viewer")
psp_update(proxy, new_data)
}
```

psp_validate_expressions *Validate Expressions*

Description

Validates Perspective expression strings against the table without applying them. The result is delivered asynchronously to input$<outputId>_validate_expressions.

Usage

```
psp_validate_expressions(proxy, expressions)
```

Arguments

proxy	A perspectiveProxy object.
expressions	A non-empty character vector of expression strings.

Value

The proxy object (invisibly), for chaining.

Examples

```
if (interactive()) {
proxy <- perspectiveProxy(session, "viewer")
psp_validate_expressions(proxy, expressions = c("\"col1\" + \"col2\""))
}
```

`renderPerspective` *Render a Perspective Viewer in Shiny*

Description

Server-side rendering function for the Perspective widget.

Usage

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

Arguments

`expr`	An expression that returns a `perspective` widget.
`env`	The environment in which to evaluate `expr`.
`quoted`	Logical; is `expr` a quoted expression?

Value

A Shiny render function.

Examples

```
if (interactive()) {
server <- function(input, output) {
  output$viewer <- renderPerspective({
    perspective(mtcars, group_by = "cyl", plugin = "Y Bar")
  })
}
}
```

`run_example` *Run a perspectiveR Example App*

Description

Launches a bundled Shiny example app demonstrating the perspectiveR widget.

Usage

```
run_example(example = NULL, ...)
```

Arguments

`example`	Name of the example to run. Use `NULL` (default) to list available examples.
`...`	Additional arguments passed to `runApp`.

Value

If `example` is `NULL`, a character vector of available example names (invisibly). Otherwise, no return value; called for the side effect of launching a Shiny app.

Examples

```
# List available examples
run_example()

# Launch the demo app
if (interactive()) {
run_example("shiny-basic")
}
```

Index