Package 'VizModules'

July 27, 2026

Title Flexible, Interactive 'shiny' Modules for Almost Any Plot

Version 0.3.0

Description Offers a core selection of interactivity-first 'shiny' modules for many plot types meant to serve as flexible building blocks for applications and as the base for more complex modules. These modules allow for the rapid and convenient construction of 'shiny' apps with very few lines of code and decouple plotting from the underlying data. These modules allow for full plot aesthetic customization by the end user through UI inputs. Utility functions for simple UI organization, automated UI tooltips, and additional plot enhancements are also provided. Includes a multi-panel figure builder app for arranging multiple modules together in a free-form layout.

License MIT + file LICENSE

Depends shiny, dittoViz, plotly, R (>= 4.5), shinyBS, plotthis (>= 0.13.0)

Imports roclang, colourpicker, dplyr, DT, readxl, shinyjs, scales, shinyjqui, ggplot2, htmltools, jsonlite, methods, shinyWidgets, htmlwidgets, zip

Encoding UTF-8

LazyData true

Suggests drc, withr, knitr, rmarkdown, testthat (>= 3.0.0)

VignetteBuilder knitr

Config/testthat/edition 3

URL https://j-andrews7.github.io/VizModules/, https://github.com/j-andrews7/VizModules

Config/roxygen2/version 8.0.0

RoxygenNote 8.0.0

NeedsCompilation no

Author Jared Andrews [aut, cre] (ORCID: <https://orcid.org/0000-0002-0780-6248>), Jacob Martin [aut] (ORCID: <https://orcid.org/0009-0007-6896-4796>)

Maintainer Jared Andrews <jared.andrews07@gmail.com>

Repository CRAN

Date/Publication 2026-07-27 17:10:02 UTC

Contents

`add_ablines` *Build diagonal (abline) line shapes for a plotly figure*

Description

Creates shape specifications for one or more diagonal lines defined by slope and intercept. Lines are drawn across the provided or computed axis range.

Usage

```
add_ablines(
  fig,
  slopes,
  intercepts,
  colors = "#000000",
  widths = 1,
  linetypes = "solid",
  opacities = 1
)
```

Arguments

`fig`	A plotly figure object (used to determine x-axis range and detect subplots).
`slopes`	Numeric vector. Slopes for the diagonal lines.
`intercepts`	Numeric vector. Y-intercepts for the diagonal lines. Must be same length as slopes.
`colors`	Character vector. Line colors (hex or named colors).
`widths`	Numeric vector. Line widths in pixels.
`linetypes`	Character vector. Line types: "solid", "dashed", "dotted", "dotdash", "longdash", "twodash".
`opacities`	Numeric vector. Line opacities (0 to 1).

Details

If style vector lengths don't match the number of lines, only the first value of each style vector is used for all lines. If slopes and intercepts have different lengths, the shorter one is recycled. When the figure contains subplots (e.g., from faceting), lines are replicated across all panels with correct axis references.

Value

A list of shape specifications for use with plotly::layout().

Author(s)

Jared Andrews

Examples

```
fig <- plotly::plot_ly(mtcars, x = ~wt, y = ~mpg, type = "scatter", mode = "markers")
add_ablines(fig, slopes = 1, intercepts = 0, colors = "red")
```

`add_hlines` *Build horizontal line shapes for a plotly figure*

Description

Creates shape specifications for one or more horizontal lines at specified y-intercepts. Supports independent styling for each line.

Usage

```
add_hlines(
  fig,
  intercepts,
  colors = "#000000",
  widths = 1,
  linetypes = "solid",
  opacities = 1
)
```

Arguments

`fig`	A plotly figure object. Used to detect subplot axes for faceted plots.
`intercepts`	Numeric vector. Y-axis intercepts for horizontal lines.
`colors`	Character vector. Line colors (hex or named colors).
`widths`	Numeric vector. Line widths in pixels.
`linetypes`	Character vector. Line types: "solid", "dashed", "dotted", "dotdash", "longdash", "twodash".
`opacities`	Numeric vector. Line opacities (0 to 1).

Details

If style vector lengths don't match the number of intercepts, only the first value of each style vector is used for all lines. When the figure contains subplots (e.g., from faceting), lines are replicated across all panels with correct axis references.

Value

A list of shape specifications for use with plotly::layout().

Author(s)

Jared Andrews

Examples

```
fig <- plotly::plot_ly(mtcars, x = ~wt, y = ~mpg, type = "scatter", mode = "markers")
add_hlines(fig, intercepts = c(20, 30), colors = c("red", "blue"))
```

---

add_plot_config *Create default Plotly configuration*

Description

Constructs a configuration list for Plotly plots, enabling interactive editing of titles and legends, export options, and additional drawing tools in the modebar.

Usage

```
add_plot_config(
  download.format = "png",
  filename = as.character(Sys.Date()),
  include.modebar.buttons = TRUE,
  facet.by = NULL
)
```

Arguments

download.format	Character. The image format for downloads (e.g., "png", "svg", "jpeg").
filename	Character. The filename for downloaded images (default: current date).
include.modebar.buttons	Logical. Whether to include drawing tool buttons in the modebar (default: TRUE).
facet.by	Logical. Whether the figure is facetted to determine if axes labels for each plot should be editable or not.

Details

The configuration enables interactive editing of the plot title, legend text and position, colorbar position and title, and annotation tails. It also adds drawing tools (lines, paths, circles, rectangles, and an eraser) to the modebar. Native cartesian axis-title text editing is disabled because axis titles are rendered as draggable, editable annotations (see axis_titles_as_annotations() and build_facet_annotations()).

Value

A named list suitable for use as the `config` argument in Plotly calls, containing edit options, image download settings, extra modebar buttons, and logo display preferences.

Author(s)

Jacob Martin

Examples

```
add_plot_config()
add_plot_config(download.format = "svg", include.modebar.buttons = FALSE)
```

---

add_reference_lines *Add reference lines to a plotly figure from Shiny inputs*

Description

Convenience wrapper that adds horizontal, vertical, and/or diagonal lines to a plotly figure based on parsed Shiny input values.

Usage

```
add_reference_lines(
  fig,
  hline.intercepts = NULL,
  hline.colors = NULL,
  hline.widths = NULL,
  hline.linetypes = NULL,
  hline.opacities = NULL,
  vline.intercepts = NULL,
  vline.colors = NULL,
  vline.widths = NULL,
  vline.linetypes = NULL,
  vline.opacities = NULL,
  abline.slopes = NULL,
  abline.intercepts = NULL,
  abline.colors = NULL,
  abline.widths = NULL,
  abline.linetypes = NULL,
  abline.opacities = NULL
)
```

Arguments

`fig`	A plotly figure object.
`hline.intercepts`	Character. Comma-separated y-intercepts for horizontal lines.
`hline.colors`	Character. Comma-separated colors for horizontal lines.
`hline.widths`	Character. Comma-separated widths for horizontal lines.
`hline.linetypes`	Character. Comma-separated linetypes for horizontal lines.
`hline.opacities`	Character. Comma-separated opacities for horizontal lines.
`vline.intercepts`	Character. Comma-separated x-intercepts for vertical lines.
`vline.colors`	Character. Comma-separated colors for vertical lines.
`vline.widths`	Character. Comma-separated widths for vertical lines.
`vline.linetypes`	Character. Comma-separated linetypes for vertical lines.
`vline.opacities`	Character. Comma-separated opacities for vertical lines.
`abline.slopes`	Character. Comma-separated slopes for diagonal lines.
`abline.intercepts`	Character. Comma-separated y-intercepts for diagonal lines.
`abline.colors`	Character. Comma-separated colors for diagonal lines.
`abline.widths`	Character. Comma-separated widths for diagonal lines.
`abline.linetypes`	Character. Comma-separated linetypes for diagonal lines.
`abline.opacities`	Character. Comma-separated opacities for diagonal lines.

Value

The modified plotly figure with all specified lines added.

Author(s)

Jared Andrews

Examples

```
fig <- plotly::plot_ly(mtcars, x = ~wt, y = ~mpg, type = "scatter", mode = "markers")
add_reference_lines(fig,
    hline.intercepts = "20, 30", hline.colors = "red, blue",
    vline.intercepts = "3", abline.slopes = "5", abline.intercepts = "0"
)
```

add_vlines *Build vertical line shapes for a plotly figure*

Description

Creates shape specifications for one or more vertical lines at specified x-intercepts. Supports independent styling for each line.

Usage

```
add_vlines(
  fig,
  intercepts,
  colors = "#000000",
  widths = 1,
  linetypes = "solid",
  opacities = 1
)
```

Arguments

fig	A plotly figure object. Used to detect subplot axes for faceted plots.
intercepts	Numeric vector. X-axis intercepts for vertical lines.
colors	Character vector. Line colors (hex or named colors).
widths	Numeric vector. Line widths in pixels.
linetypes	Character vector. Line types: "solid", "dashed", "dotted", "dotdash", "longdash", "twodash".
opacities	Numeric vector. Line opacities (0 to 1).

Details

If style vector lengths don't match the number of intercepts, only the first value of each style vector is used for all lines. When the figure contains subplots (e.g., from faceting), lines are replicated across all panels with correct axis references.

Value

A list of shape specifications for use with plotly::layout().

Author(s)

Jared Andrews

Examples

```
fig <- plotly::plot_ly(mtcars, x = ~wt, y = ~mpg, type = "scatter", mode = "markers")
add_vlines(fig, intercepts = c(3, 4), colors = c("red", "blue"))
```

`adjusted_axis_label` *Build an adjustment-aware axis label*

Description

Wraps a base column name with the names of any data adjustments that are applied to it before plotting, so that an axis title accurately describes the values displayed. The wrapping order mirrors how the adjustments are applied in dittoViz (the recognized `adjustment` is applied first, then the `adj.fxn`), producing labels such as `"log2(z-score(units))"`.

Usage

```
adjusted_axis_label(base, adjustment = NULL, adj.fxn = NULL)
```

Arguments

`base`	Character scalar. The base axis label (typically the column name).
`adjustment`	Character scalar. A recognized data adjustment such as `"z-score"` or `"relative.to.max"`. Optional.
`adj.fxn`	Character scalar. The name of a transformation function such as `"log2"` or `"sqrt"`. Optional.

Details

Empty strings, `NA`, and `NULL` adjustments are ignored, so when no adjustment is requested the base label is returned unchanged.

Value

A character scalar containing the (possibly wrapped) axis label.

Author(s)

Jared Andrews

Examples

```
adjusted_axis_label("units")
adjusted_axis_label("units", adjustment = "z-score")
adjusted_axis_label("units", adjustment = "z-score", adj.fxn = "log2")
```

`adjust_column_values` *Adjust numeric column values in a data frame using mathematical transformations*

Description

Applies a named mathematical transformation to a specified numeric column in a data frame, adding the transformed values as a new column (original column name + ".adj"). The transformation name must be one of the allowed functions listed in `safe_resolve_adj_fxn` (e.g., "log2", "log10", "sqrt", "abs", "as.factor"). The original data frame is returned unchanged if no transformation is specified or if the supplied name is invalid.

Usage

```
adjust_column_values(
  df,
  x.col = NULL,
  y.col = NULL,
  color.col = NULL,
  x.adj.fun = NULL,
  y.adj.fun = NULL,
  color.adj.fun = NULL
)
```

Arguments

`df`	A data frame containing the column to be transformed.
`x.col`	Character scalar. Name of the column for x-axis values (optional).
`y.col`	Character scalar. Name of the column for y-axis values (optional).
`color.col`	Character scalar. Name of the column for color values (optional).
`x.adj.fun`	Character scalar. Name of a transformation function to apply to x-axis values, as accepted by `safe_resolve_adj_fxn` (e.g., "log2", "log10", "sqrt"). If `NULL` or an empty string, x-axis values are left unchanged.
`y.adj.fun`	Character scalar. Name of a transformation function to apply to y-axis values, as accepted by `safe_resolve_adj_fxn`. If `NULL` or an empty string, y-axis values are left unchanged.
`color.adj.fun`	Character scalar. Name of a transformation function to apply to color values, as accepted by `safe_resolve_adj_fxn`. If `NULL` or an empty string, color values are left unchanged.

Value

A data frame identical to input `df` but with transformed columns added (e.g., `mpg.adj`) when valid transformations are specified.

Author(s)

Jacob Martin, Jared Andrews

Examples

```
data(mtcars)
mtcars_mod <- adjust_column_values(mtcars, x.col = "mpg", x.adj.fun = "log2")
head(mtcars_mod$mpg.adj)
```

---

apply_axis_title_to_annotations

Apply axis title font styling to shared facet axis annotations

---

Description

When ggplotly converts a faceted ggplot, shared axis titles become annotations rather than axis title properties. This function finds those shared title annotations and applies the user's axis title font settings to them.

Usage

```
apply_axis_title_to_annotations(fig, input, isolate_fn = isolate)
```

Arguments

`fig`	A plotly figure object.
`input`	Shiny input object containing axis title font fields.
`isolate_fn`	Function to isolate reactive values. Defaults to `shiny::isolate`.

Value

The modified plotly figure with updated annotation fonts.

Author(s)

Jacob Martin

Examples

```
## Not run:
p <- ggplot2::ggplot(mtcars, ggplot2::aes(wt, mpg)) +
    ggplot2::geom_point() +
    ggplot2::facet_wrap(~cyl)
fig <- plotly::ggplotly(p)
input <- list(
    axis.title.font.size = 14, axis.title.font.family = "Arial",
    axis.title.font.color = "black", facet.title.font.size = 12,
```

```
    facet.title.font.family = "Arial", facet.title.font.color = "black"
)
apply_axis_title_to_annotations(fig, input, isolate_fn = identity)

## End(Not run)
```

`apply_facet_subplot_spacing` *Apply custom subplot spacing to a faceted ggplotly figure*

Description

`ggplotly()` assigns panel layout via plotly domain coordinates (`fig$x$layout$xaxis*/yaxis*$domain`) rather than honouring the ggplot2 `panel.spacing` theme option. This helper rewrites those domains so that each facet panel has a uniform size and the gap between panels is exactly `spacing` (expressed as a fraction of the plot area, e.g. `0.04`).

Usage

```
apply_facet_subplot_spacing(fig, spacing = 0.04, ncol = NULL, nrow = NULL)
```

Arguments

`fig`	A plotly figure object (typically the result of `ggplotly()`).
`spacing`	Numeric fraction of the plot area to leave between panels (default `0.04`). May be a single value applied to both directions, or a length-2 numeric vector `c(horizontal, vertical)` to control the gap between columns and rows independently. Must satisfy `horizontal * (ncol - 1) < 1` and `vertical * (nrow - 1) < 1`; otherwise the figure is returned unchanged.
`ncol`	Optional integer. Number of facet columns. If `NULL` or `NA`, detected from the number of distinct x-axis domain starts.
`nrow`	Optional integer. Number of facet rows. If `NULL` or `NA`, detected from the number of distinct y-axis domain starts.

Details

In addition to the axis domains, any paper-anchored layout annotations (e.g. facet strip titles) and shapes (e.g. strip backgrounds, panel borders) are remapped through a piecewise-linear transform built from the old and new domain intervals, so strip labels and panel borders move with their panels. Annotations/shapes whose `xref`/`yref` is tied to an axis (for example `"x"`, `"y2"`, or `"x2 domain"`) are left alone because they follow the rewritten axis automatically.

The number of columns and rows can be supplied manually, or detected automatically from the distinct x / y domain starts already present in the ggplotly output.

Value

The modified plotly figure with rewritten axis domains and remapped paper-anchored annotations/shapes. Figures with a single panel (or no layout) are returned unchanged.

Author(s)

Jacob Martin

Examples

```
p <- ggplot2::ggplot(mtcars, ggplot2::aes(wt, mpg)) +
    ggplot2::geom_point() +
    ggplot2::facet_wrap(~cyl)
fig <- plotly::ggplotly(p)
apply_facet_subplot_spacing(fig, spacing = 0.05)
```

---

apply_legend_styling *Apply uniform legend font styling to a plotly figure*

Description

Sets the legend title and entry-label font sizes on a plotly figure so the "Legend" UI inputs behave consistently across plot types. Existing legend settings (orientation, position, font family/colour) are preserved because `plotly::layout()` merges the supplied attributes into the current layout. `NULL` or `NA` sizes are ignored, leaving the corresponding font size untouched.

Usage

```
apply_legend_styling(fig, title.size = NULL, text.size = NULL, position = NULL)
```

Arguments

`fig`	A plotly figure object.
`title.size`	Numeric font size for the legend (or colorbar) title, or `NULL` to leave unchanged.
`text.size`	Numeric font size for the legend entry labels (or colorbar tick labels), or `NULL` to leave unchanged.
`position`	vector of an integer, an xanchor string, and a orientation argument. e.g. c(1.02, "left", "v"). Controls horizontal positioning of the legend.

Details

Numeric colour mappings (for example `fill.by`/`color.by` on a continuous variable) are rendered as a *colorbar* rather than a categorical legend. The layout-level legend font does not affect a colorbar, so the colorbar title and tick fonts are updated directly on each trace (and on any shared `coloraxis`) using the same sizes. This keeps the "Legend" controls functional for both categorical and continuous legends.

Value

The plotly figure with the requested legend font sizes applied. Returns the figure unchanged when `fig` is `NULL` or no valid sizes are supplied.

Author(s)

Jared Andrews

Examples

```
fig <- plotly::plot_ly(iris,
    x = ~Sepal.Length, y = ~Sepal.Width,
    color = ~Species, type = "scatter", mode = "markers"
)
apply_legend_styling(fig, title.size = 16, text.size = 10)
```

---

apply_plotly_newshape *Apply Plotly newshape styling from uniform Plotly inputs*

Description

Applies user-drawn shape styling to a Plotly figure using inputs from `uniform_plotly_inputs_ui()`. Updates the newshape layout property to style shapes drawn with Plotly's drawing tools (rectangles, circles, lines, etc.) in the modebar.

Usage

```
apply_plotly_newshape(fig, input, isolate_fn = isolate)
```

Arguments

`fig`	A plotly figure object.
`input`	Shiny input object containing shape styling fields: `shape.fill`, `shape.line.color`, `shape.line.width`, `shape.linetype`, `shape.opacity`.
`isolate_fn`	Function to isolate reactive values. Defaults to `shiny::isolate`.

Value

The modified plotly figure with updated newshape layout settings.

Author(s)

Jared Andrews

Examples

```
fig <- plotly::plot_ly(mtcars, x = ~wt, y = ~mpg, type = "scatter", mode = "markers")
shape_input <- list(
    shape.fill = "#ff000030", shape.line.color = "#ff0000",
    shape.line.width = 2, shape.linetype = "solid", shape.opacity = 0.5
)
apply_plotly_newshape(fig, shape_input, isolate_fn = identity)
```

apply_render_margins *Apply standard render-time margin layout to a plotly figure*

Description

The `renderPlotly` block in every plot module server applies the same user-configurable margins. This helper extracts that block into one call.

Usage

```
apply_render_margins(fig, input)
```

Arguments

`fig`	A plotly figure object.
`input`	Shiny input object. Expected to contain `margin.t`, `margin.b`, `margin.l`, and `margin.r`.

Value

The plotly figure with margins applied.

Author(s)

Jacob Martin

Examples

```
fig <- plotly::plot_ly(mtcars, x = ~wt, y = ~mpg, type = "scatter", mode = "markers")
input <- list(margin.t = 40, margin.b = 40, margin.l = 40, margin.r = 40)
apply_render_margins(fig, input)
```

apply_stat_annotations *Apply statistical annotation shapes and annotations to a plotly figure*

Description

Appends the shapes and annotations from `create_stat_annotations()` to an existing plotly figure's layout. Adjusts the y-axis range to accommodate the annotation brackets.

Usage

```
apply_stat_annotations(fig, stat_result, y.min = NULL)
```

Arguments

`fig`	A plotly figure object.
`stat_result`	List with `annotations`, `shapes`, and `y.max` as returned by `create_stat_annotations()`.
`y.min`	Numeric or NULL; minimum y-axis value. If NULL, the existing y-axis range is preserved.

Value

The modified plotly figure.

Author(s)

Jared Andrews

Examples

```
stats_df <- compute_pairwise_stats(
    df = example_iris,
    x = "Species",
    y = "Sepal.Length",
    test = "wilcox.test"
)
fig <- plotly::plot_ly(
    data = example_iris, x = ~Species, y = ~Sepal.Length, type = "box"
)
stat_result <- create_stat_annotations(
    stats_df = stats_df,
    fig = fig,
    df = example_iris,
    x = "Species",
    y = "Sepal.Length",
    display = "symbol"
)
apply_stat_annotations(fig, stat_result)
```

`apply_subplot_axis_styling` *Apply axis styling to all subplot axes in a plotly figure*

Description

When using plotly subplots (e.g., via split.by in dittoViz), axis styling must be applied to all subplot axes (xaxis, xaxis2, xaxis3, etc.) individually. This helper function detects how many subplots exist and applies the provided axis styling to all of them.

Usage

```
apply_subplot_axis_styling(fig, xaxis_style, yaxis_style)
```

Arguments

fig	A plotly figure object.
xaxis_style	A named list of axis styling parameters for x-axes.
yaxis_style	A named list of axis styling parameters for y-axes.

Value

The modified plotly figure with axis styling applied to all subplots.

Author(s)

Jared Andrews

Examples

```
fig <- plotly::plot_ly(mtcars, x = ~wt, y = ~mpg, type = "scatter", mode = "markers")
xaxis_style <- list(showline = TRUE, linecolor = "black", linewidth = 1)
yaxis_style <- list(showline = TRUE, linecolor = "black", linewidth = 1)
apply_subplot_axis_styling(fig, xaxis_style, yaxis_style)
```

---

apply_title_layout *Apply plot title styling to a plotly figure*

Description

Applies title font settings from the Shiny input object to an existing plotly figure. The title is centered horizontally and positioned using the supplied `title_y` value in the plotly layout.

Usage

```
apply_title_layout(plot, input, isolate_fn, title_y = 0.95, title_x = 0.5)
```

Arguments

plot	A plotly figure object.
input	Shiny input object containing title font fields.
isolate_fn	Function to isolate reactive values.
title_y	Numeric y position for the plot title in the plotly layout. Defaults to `0.95`.
title_x	Numeric position for the title in the plotly layout.

Value

The modified plotly figure with updated title styling.

Author(s)

Jacob Martin

Examples

```
## Not run:
p <- ggplot2::ggplot(mtcars, ggplot2::aes(wt, mpg)) + ggplot2::geom_point()
input <- list(
    title.font.size = 16, title.font.family = "Arial", title.font.color = "black"
)
apply_title_layout(p, input, isolate_fn = identity)

## End(Not run)
```

`axis_titles_as_annotations` *Convert native cartesian axis titles to draggable annotations*

Description

Plotly's native axis titles can have their text edited interactively but cannot be dragged to a new position. Faceted figures already render their shared x/y axis titles as paper-anchored annotations (via `build_facet_annotations()`), which the plot configuration makes both editable and draggable. This helper brings the same behaviour to single-panel (non-faceted) figures by replacing the native x/y axis titles with equivalent paper-anchored annotations.

Usage

```
axis_titles_as_annotations(fig)
```

Arguments

`fig`	A plotly figure object.

Details

The figure is first built with `plotly::plotly_build()` so that titles assigned via `layout()` (which are otherwise held in `layoutAttrs` until build time) are consolidated into the layout. Any pre-existing annotations (for example statistical brackets or facet labels) are preserved, and the font already applied to each native axis title is carried over to the corresponding annotation.

Multi-panel figures (faceting or `split.by`, detected by the presence of secondary axes such as `xaxis2`/`yaxis2`) are returned unchanged, since their shared titles are already draggable annotations.

Value

The plotly figure with single-panel axis titles converted to paper-anchored, draggable annotations. Returns the figure unchanged when it is faceted/split or has no axis titles.

Author(s)

Jared Andrews

Examples

```
fig <- plotly::plot_ly(mtcars, x = ~wt, y = ~mpg, type = "scatter", mode = "markers")
fig <- plotly::layout(fig, xaxis = list(title = "Weight"), yaxis = list(title = "MPG"))
axis_titles_as_annotations(fig)
```

`build_facet_annotations` *Build facet subplot annotations*

Description

Creates a list of plotly annotation objects suitable for labelling faceted subplots arranged in a grid of `nrows` rows. When `nrows = 1` (the default) the behaviour matches the previous single-row layout. Optionally appends a shared X-axis title (bottom centre) and a shared, rotated Y-axis title (left centre).

Usage

```
build_facet_annotations(
  facet_levels,
  x.title = NULL,
  y.title = NULL,
  title.font.size = 14,
  nrows = 1,
  fig = NULL,
  title.offset = 0.02
)
```

Arguments

`facet_levels`	Character vector of facet level labels, one per subplot.
`x.title`	Optional character, shared X-axis title. Default: `NULL`.
`y.title`	Optional character, shared Y-axis title. Default: `NULL`.
`title.font.size`	Numeric, font size for all annotation text. Default: 14.
`nrows`	Integer, number of rows the faceted subplots are arranged in. Used to compute per-subplot annotation coordinates for multi-row grids when `fig` is not supplied. Default: 1.
`fig`	Optional plotly figure. When supplied, per-panel title coordinates are read directly from the figure's xaxis/yaxis domains so that titles stay aligned with panels after domain-rewriting helpers such as `apply_facet_subplot_spacing()`. If `NULL` (the default), coordinates are computed from `nrows` assuming evenly spaced panels filling the full paper area.
`title.offset`	Numeric fraction of the figure height to place each subplot title above the top of its panel. Default: `0.02`.

Value

A list of annotation lists suitable for `plotly::layout(annotations = ...)`.

Author(s)

Jared Andrews

Examples

```
build_facet_annotations(c("A", "B", "C"), x.title = "X", y.title = "Y")
```

`build_facet_panel_borders` *Build paper-anchored panel border shapes for a faceted plotly figure*

Description

Native plotly `subplot()` figures with shared (`shareX`/`shareY`) axes do not render axis border lines on the matched/inner panels, so faceted plots end up with a box around only the first panel. This helper reconstructs each panel's rectangle from the grid of x/y axis domains in the figure layout and returns paper-anchored shapes that draw a uniform border around every panel.

Usage

```
build_facet_panel_borders(
  fig,
  n_facets,
  showline = TRUE,
  mirror = TRUE,
  linecolor = "black",
  linewidth = 0.5,
  ncol = NULL,
  nrow = NULL
)
```

Arguments

`fig`	A plotly figure object whose `x$layout` contains the per-panel `xaxis*`/`yaxis*` domains (typically after `subplot()` and `apply_facet_subplot_spacing()`).
`n_facets`	Integer, number of facet panels.
`showline`	Logical, whether to draw border lines. Default `TRUE`.
`mirror`	Logical, whether to mirror the lines to form a full box. Default `TRUE`.
`linecolor`	Character, colour of the border lines. Default `"black"`.
`linewidth`	Numeric, width of the border lines in pixels. Default `0.5`.

ncol	Optional integer. Number of facet columns. If NULL or NA, detected from the distinct x-axis domain starts.
nrow	Optional integer. Number of facet rows. If NULL or NA, detected from the distinct y-axis domain starts.

Details

Panel rectangles are derived from the geometry of the subplot grid rather than from a per-panel axis index. With shared axes plotly only keeps one axis per column (x) and one per row (y), so an xaxis{i}/yaxis{i} lookup keyed on the facet index breaks down for grids with more than one row or column. Instead the distinct x-axis domain starts define the columns (left to right) and the distinct y-axis domain starts define the rows (top to bottom), and each facet panel is mapped to a (row, column) cell in row-major fill order — matching how subplot() lays panels out.

The borders honour the same axis styling semantics used for single-panel figures:

- showline and mirror both TRUE: full rectangle border around each panel.
- only showline TRUE: left and bottom edges only.
- showline FALSE: no borders (empty list).

Value

A list of plotly shape definitions (each paper-anchored). Returns an empty list when borders should not be drawn or panel domains cannot be resolved.

Author(s)

Jacob Martin

Examples

```
p <- ggplot2::ggplot(mtcars, ggplot2::aes(wt, mpg)) +
    ggplot2::geom_point() +
    ggplot2::facet_wrap(~cyl)
fig <- plotly::ggplotly(p)
build_facet_panel_borders(fig, n_facets = 3)
```

---

build_model_row_spec *Build a dynamic row_spec from registered backends*

Description

Constructs the merged row_spec for multiDynamicInput() by combining the standard model fields (model_type, formula, line_colour, line_width) with any extra fields declared by registered backends. Each backend field is tagged with data-backend so the client can show/hide it based on the selected model type.

Usage

```
build_model_row_spec()
```

Value

A named list suitable for the `row_spec` argument of `multiDynamicInput()`.

Author(s)

Jacob Martin

Examples

```
build_model_row_spec()
```

`clean_facet_dim` *Clean and validate facet dimension value for lineplot module*

Description

Internal helper function that validates and sanitizes a numeric value intended for use as a **facet dimension** (rows or columns) in a **ggplot2 faceting layout**. Ensures the value is a positive numeric greater than or equal to 1, returning `NULL` for invalid inputs to gracefully handle missing or malformed facet specifications.

Usage

```
clean_facet_dim(val)
```

Arguments

`val`	`numeric(1)` or `NULL` Proposed facet dimension value (number of rows or columns).

Details

This function is used within **VizModules** lineplot functions to process user-supplied facet dimensions before passing to `facet_grid()` or `facet_wrap()`. Invalid values trigger sensible defaults rather than breaking the plot layout. **Valid inputs** return unchanged. **Invalid inputs** (NULL, NA, non-numeric, < 1) return `NULL`.

Value

`numeric(1)` or `NULL` Validated facet dimension value, or `NULL` if invalid.

Author(s)

Jacob Martin

Examples

```
clean_facet_dim(3)
clean_facet_dim(NA)
clean_facet_dim(NULL)
```

`collect_source_data` *Collect plot and source data for download*

Description

Collects the plot object, its underlying data, statistical testing details (if applied), and optional UI input values into a single list for downstream download generation.

Usage

```
collect_source_data(
  plot_reactive,
  stats_reactive = NULL,
  inputs_reactive = NULL
)
```

Arguments

`plot_reactive`	A reactive expression returning a `plotly` plot object.
`stats_reactive`	Optional. A reactive expression (e.g. a `shiny::reactiveVal()`) returning a `data.frame` of statistical test results. When `NULL` or when the reactive returns `NULL`, no statistics data is included.
`inputs_reactive`	Optional. A reactive expression returning a named list of UI input values. When `NULL` or when it returns `NULL`, no UI input data is included.

Value

A named list with elements:

plot The `plotly` plot object.

plot_data A `data.frame` of the plot's underlying data.

stats A `data.frame` of statistical test results, or `NULL`.

inputs A `data.frame` of UI input names and values, or `NULL`.

Author(s)

Jacob Martin

Examples

```
## Not run:
# Example usage in a Shiny app
library(shiny)
library(plotly)
library(VizModules)

ui <- fluidPage(
    plotlyOutput("my_plot"),
    downloadButton("download_data", "Download Plot and Data")
)

server <- function(input, output) {
    plot_reactive <- reactive({
       plot_ly(mtcars, x = ~mpg, y = ~hp, type = "scatter", mode = "markers")
    })

    data_list <- collect_source_data(plot_reactive)
    output$my_plot <- renderPlotly(plot_reactive())
    output$download_data <- create_source_download_handler(reactive(data_list))
}

shinyApp(ui, server)

## End(Not run)
```

`compute_pairwise_stats` *Compute pairwise statistical tests between groups*

Description

Performs pairwise statistical tests between groups defined by a categorical variable. Supports Wilcoxon rank-sum, t-test, Kruskal-Wallis, and ANOVA. Handles nested grouping (comparing color.by groups within each x-level) and per-facet testing.

Usage

```
compute_pairwise_stats(
  df,
  x,
  y,
  pairs = NULL,
  test = "wilcox.test",
  p.adjust.method = "holm",
  paired = FALSE,
  group.by = NULL,
  facet.by = NULL,
```

```
  per.facet = TRUE,
  sig.threshold = 0.05,
  sig.levels = c(`****` = 1e-04, `***` = 0.001, `**` = 0.01)
)
```

Arguments

df	Data frame containing the data.
x	Character; column name of the categorical x-axis variable.
y	Character; column name of the numeric response variable.
pairs	List of length-2 character vectors specifying group pairs to test. If NULL (default), tests all unique pairwise combinations.
test	Character; statistical test to use. One of "wilcox.test", "t.test", "kruskal.test", or "anova".
p.adjust.method	Character; method for p-value adjustment via stats::p.adjust(). Default "holm".
paired	Logical; whether to perform paired tests (only for "wilcox.test" and "t.test"). Default FALSE.
group.by	Character or NULL; column for nested grouping. When set, comparisons are made between levels of group.by within each level of x.
facet.by	Character or NULL; column for faceting. When set and per.facet = TRUE, tests run independently per facet panel.
per.facet	Logical; if TRUE and facet.by is set, run tests independently per facet panel. Default TRUE.
sig.threshold	Numeric; significance threshold for * vs ns. P-values at or below this are labeled *; above are labeled ns. Default 0.05. See sig.levels for the multi-star thresholds.
sig.levels	Named numeric vector; upper p-value bounds for multi-star significance symbols. Names are the displayed symbols and values are the thresholds. Default c("****" = 0.0001, "***" = 0.001, "**" = 0.01). Any number of levels can be provided. Evaluated from smallest to largest threshold so the most significant symbol always wins.

Value

A data.frame with columns: group1, group2, p.value, p.adj, p.signif, test, facet_level, x_level (when group.by is set).

Author(s)

Jared Andrews, Jacob Martin

Examples

```
compute_pairwise_stats(
    df = example_iris,
    x = "Species",
    y = "Sepal.Length",
    test = "wilcox.test"
)

# Custom significance levels: only two-star tiers, lower threshold for *
compute_pairwise_stats(
    df = example_iris,
    x = "Species",
    y = "Sepal.Length",
    test = "wilcox.test",
    sig.threshold = 0.01,
    sig.levels = c("**" = 0.001, "***" = 0.0001)
)
```

---

createModuleApp *Create an Example Module App from Any Module Trio*

Description

Factory function that generates a standard Shiny application for any VizModules module. The resulting app features a **Data Import** section for uploading data files, a **Data Table** for viewing and editing the active dataset, and a **Plot** area for configuring and displaying an interactive plot.

Usage

```
createModuleApp(
  inputs_ui_fn,
  output_ui_fn,
  server_fn,
  data_list,
  defaults = NULL,
  hide.inputs = NULL,
  hide.tabs = NULL,
  show.table = TRUE,
  title = "VizModules App"
)
```

Arguments

`inputs_ui_fn`	A function with signature `function(id, data, ...)` that returns module input UI elements (e.g. `plotthis_BarPlotInputsUI()`).
`output_ui_fn`	A function with signature `function(id)` that returns the module's output UI (e.g. `plotthis_BarPlotOutputUI()`).

server_fn	A function with signature function(id, data, ...) that drives the module server logic (e.g. plotthis_BarPlotServer()).
data_list	A named list of data frames. At least one element is required.
defaults	A named list of ui ids and their default values that can change the ui default settings on startup.
hide.inputs	A character vector of input IDs to hide. These inputs are still initialized and their values passed to the plot, but are not shown in the UI. Passed through to server_fn when it accepts a hide.inputs argument.
hide.tabs	A character vector of tab names to hide. Inputs in these tabs are still initialized and their values passed to the plot, but are not shown in the UI. Passed through to server_fn when it accepts a hide.tabs argument.
show.table	Logical. When TRUE (default), a filterable DT table is shown below the plot and its row selection drives the data passed to the plot module. When FALSE, the table and filter controls are hidden and the full (unfiltered) dataset is passed directly to the plot module.
title	A character string used as the page title (default: "VizModules App").

Details

Uploaded files (Excel, CSV, TSV, or tab-delimited text) are added to the available datasets and can be selected for plotting. If an uploaded file shares a name with an existing dataset, the existing one is overwritten with a warning.

Every module-specific *App() convenience function (e.g. plotthis_BarPlotApp(), linePlotApp()) is a thin wrapper around createModuleApp(). You can also call it directly for quick prototyping or to create apps for custom wrapper modules.

Value

A shiny::shinyApp() object.

Author(s)

Jared Andrews

Examples

```
library(VizModules)

# Quick-launch a bar plot app with custom data:
app <- createModuleApp(
    inputs_ui_fn = plotthis_BarPlotInputsUI,
    output_ui_fn = plotthis_BarPlotOutputUI,
    server_fn    = plotthis_BarPlotServer,
    data_list    = list("iris" = iris),
    title        = "My Bar Plot",
    defaults     = NULL
)
if (interactive()) runApp(app)
```

```
# Works with any module trio, including custom wrapper modules:
app2 <- createModuleApp(
    inputs_ui_fn = dittoViz_scatterPlotInputsUI,
    output_ui_fn = dittoViz_scatterPlotOutputUI,
    server_fn    = dittoViz_scatterPlotServer,
    data_list    = list("iris" = iris),
    title        = "Scatter",
    defaults    = NULL
)
if (interactive()) runApp(app2)
```

`create_axis_styles` *Create Plotly axis style list*

Description

Constructs a style list for a Plotly axis using values from a Shiny `input` object, including title font, axis lines, tick appearance, and gridline settings.

Usage

```
create_axis_styles(
  input,
  axis_side = c("x", "y"),
  isolate_fn = isolate,
  ggplot.axis.styling = TRUE
)
```

Arguments

`input`	Shiny input object. Expected to contain axis-related fields such as `title.font.family`, `text.colour`, `axis.showline`, `axis.mirror`, `axis.linecolor`, `axis.linewidth`, `axis.tickfont.size`, `axis.tickfont.color`, `axis.tickfont.family`, `axis.tickangle.x`, `axis.tickangle.y`, `axis.ticks`, `axis.tickcolor`, `axis.ticklen`, `axis.tickwidth`, `show.grid.x`, `show.grid.y`, and `grid.color`.
`axis_side`	Character. Which axis to style, either `"x"` or `"y"`. Determines whether `axis.tickangle.x` or `axis.tickangle.y` is used for the tick angle, and which gridline inputs are applied.
`isolate_fn`	Function. A function used to isolate Shiny inputs, typically `shiny::isolate`. Defaults to `isolate`.
`ggplot.axis.styling`	Logical. Whether ggplot axis styling is applied. Defaults to `TRUE`.

Details

The function collects axis- and font-related settings from the provided `input` object and assembles them into a list suitable for use as an axis specification in Plotly layouts. The tick angle and gridline visibility are chosen based on the value of `axis_side`. If gridline inputs are not present in the input object, defaults to showing gridlines.

Value

A named list containing Plotly-compatible axis styling components, including title font, line properties, tick label formatting, and gridline visibility.

Author(s)

Jacob Martin

Examples

```
# Build a fake input list and use identity as the isolate function
input <- list(
    axis.title.font.size = 14, axis.title.font.family = "Arial",
    axis.title.font.color = "black", axis.tickfont.size = 10,
    axis.tickfont.color = "black", axis.tickfont.family = "Arial",
    axis.tickangle.x = 0, axis.tickangle.y = 0, axis.ticks = "outside",
    axis.tickcolor = "black", axis.ticklen = 5, axis.tickwidth = 1,
    show.grid.x = TRUE, show.grid.y = TRUE, grid.color = "grey90"
)
create_axis_styles(input, axis_side = "x", isolate_fn = identity)
```

`create_ggplot_axis_style` *Create ggplot axis styling theme arguments*

Description

Creates ggplot2 theme arguments for axis borders and lines based on user inputs. This function handles axis styling through ggplot2 themes rather than plotly overlays, which provides better control especially when faceting is used.

Usage

```
create_ggplot_axis_style(input, isolate_fn = isolate)
```

Arguments

`input`	Shiny input object containing axis styling parameters.
`isolate_fn`	Function to use for isolating reactive values (default: isolate).

Details

When faceting is enabled, panel borders are always shown for the full plot. When faceting is disabled:

- If both axis.showline and axis.mirror are TRUE: Full panel border
- If only axis.showline is TRUE: Axis lines on x and y axes only
- Otherwise: No borders

Value

A named list of ggplot2 theme arguments to be passed to theme_args parameter.

Author(s)

Jacob Martin

Examples

```
input <- list(
    axis.showline = TRUE, axis.mirror = TRUE,
    axis.linecolor = "black", axis.linewidth = 1
)
create_ggplot_axis_style(input, isolate_fn = identity)
```

---

create_source_download_handler

Create download handler for plot with source data

Description

Generates a Shiny `downloadHandler()` that bundles the interactive plot and its supporting data into a single `.zip` archive.

Usage

```
create_source_download_handler(data_list, filename_base = "source_data")
```

Arguments

`data_list`	A reactive returning either a single summary list produced by `collect_source_data()` (with elements `plot`, `plot_data`, `stats`, and `inputs`), or a named list of such summaries (one per plot). When a named list of summaries is supplied, each summary is written to its own set of files (prefixed with the list name) so several plots can be bundled into a single archive.
`filename_base`	`character(1)`. Base name for the downloaded `.zip` file without extension. The final filename takes the form `<filename_base>_<Sys.Date()>.zip`.

Value

A `downloadHandler` object suitable for assignment to a Shiny output.

Author(s)

Jacob Martin

Examples

```
## Not run:
# Example usage in a Shiny app
library(shiny)
library(plotly)
library(VizModules)
ui <- fluidPage(
    plotlyOutput("my_plot"),
    downloadButton("download_data", "Download Plot and Data")
)

server <- function(input, output) {
    plot_reactive <- reactive({
        plot_ly(mtcars, x = ~mpg, y = ~hp, type = "scatter", mode = "markers")
    })

    data_list <- collect_source_data(plot_reactive)
    output$my_plot <- renderPlotly(plot_reactive())
    output$download_data <- create_source_download_handler(reactive(data_list))
}

shinyApp(ui, server)

## End(Not run)
```

`create_stat_annotations` *Create plotly shapes and annotations for statistical test results*

Description

Converts results from `compute_pairwise_stats()` into plotly-compatible shapes (brackets) and annotations (text labels). Sorts comparisons so that small-gap brackets are closest to the data and large-gap brackets are higher.

Usage

```
create_stat_annotations(
  stats_df,
  fig,
  df,
```

```
  x,
  y,
  display = "p.adj",
  hide.ns = FALSE,
  sig.threshold = 0.05,
  line.color = "#000000",
  line.width = 1,
  bracket.style = "capped",
  group.by = NULL,
  facet.by = NULL,
  x.order = NULL,
  font.size = 12,
  step.increase = 0.06,
  text.bump = 0.04,
  bracket.inset = 0.025
)
```

Arguments

stats_df	Data frame from compute_pairwise_stats().
fig	A plotly figure object. Used to detect subplot axis pairs for faceted plots.
df	The original data frame.
x	Character; x-axis column name.
y	Character; y-axis column name.
display	Character; what to display: "p.adj", "p.value", or "symbol". Default "p.adj".
hide.ns	Logical; hide non-significant results. Default FALSE.
sig.threshold	Numeric; significance threshold for determining non-significant results. Default 0.05.
line.color	Character; color for bracket lines. Default "#000000".
line.width	Numeric; width of bracket lines. Default 1.
bracket.style	Character; "capped" for ggpubr-style brackets with vertical ticks, or "flat" for a single horizontal line. Default "capped".
group.by	Character or NULL; nested grouping column.
facet.by	Character or NULL; faceting column.
x.order	Character vector; order of x-axis categories. If NULL, derived from unique values of x column.
font.size	Numeric; size of annotation text. Default 12.
step.increase	Numeric; fraction of y-range for spacing between successive brackets. Default 0.06.
text.bump	Numeric; fraction of y-range for vertical distance of text above the bracket line. Default 0.04.
bracket.inset	Numeric; fixed amount to inset each bracket endpoint from the group center position. Creates visual separation between adjacent brackets at the same y-level. Default 0.025.

Value

A list with components:

annotations List of plotly annotation objects.

shapes List of plotly shape objects.

y.max Numeric; maximum y value needed to accommodate all annotations.

Author(s)

Jared Andrews, Jacob Martin

Examples

```
stats_df <- compute_pairwise_stats(
    df = example_iris,
    x = "Species",
    y = "Sepal.Length",
    test = "wilcox.test"
)

fig <- plotly::plot_ly(
    data = example_iris, x = ~Species, y = ~Sepal.Length, type = "box"
)

stat_result <- create_stat_annotations(
    stats_df = stats_df,
    fig = fig,
    df = example_iris,
    x = "Species",
    y = "Sepal.Length",
    display = "symbol"
)

names(stat_result)
```

---

dataFilterServer *Server logic for the dataFilter module*

Description

Renders an interactive DT table with column-level filters and returns a reactive containing only the currently visible (filtered) rows. This reactive can be passed directly to any plotting module as its `data` argument.

Usage

```
dataFilterServer(id, data, factor.char.cols = TRUE, page.length = 10)
```

Arguments

id	The ID for the Shiny module. Must match the id used in dataFilterUI().
data	A reactive containing the data frame to display and filter.
factor.char.cols	Logical. When TRUE, all character columns in data are converted to factors before the table is rendered. This causes DT to display select-box filters for those columns instead of free-text search boxes. Defaults to TRUE.
page.length	Integer. The default number of rows shown per page. Defaults to 10.

Value

A reactive expression that evaluates to the filtered subset of data based on the current DT selection/filter state. Pass this reactive to a plotting module's data argument to keep the plot in sync with the table filters.

Author(s)

Jacob Martin

See Also

dataFilterUI()

Examples

```
library(shiny)
library(VizModules)

ui <- fluidPage(
    dataFilterUI("filter"),
    verbatimTextOutput("rows")
)

server <- function(input, output, session) {
    data <- reactive(iris)
    filtered <- dataFilterServer("filter", data, factor.char.cols = TRUE)
    output$rows <- renderPrint(nrow(filtered()))
}

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

dataFilterUI *UI component for the dataFilter module*

Description

Renders an interactive DT table that allows users to filter rows of a data frame. Place this in the UI where you want the filterable table to appear, using an `id` that matches the one passed to `dataFilterServer()`.

Usage

```
dataFilterUI(id)
```

Arguments

`id`	The ID for the Shiny module.

Value

A Shiny `tagList` containing the DT table output.

Author(s)

Jacob Martin

See Also

`dataFilterServer()`

Examples

```
library(VizModules)
dataFilterUI("myFilter")
```

default_palettes *Color palette options for palettePicker*

Description

Returns a list of predefined color palettes grouped by category (Defaults, Viridis, Diverging, Qualitative, Sequential) for use with color picker UI components.

Usage

```
default_palettes()
```

Value

A named list with two elements: `choices` (a nested list of palette name to color vector mappings, grouped by category) and `textColor` (a character vector of text colors for each palette).

Author(s)

Jared Andrews

Examples

```
pals <- default_palettes()
names(pals$choices)
```

---

dittoViz_scatterPlotApp

Create an example Modular scatterPlot Shiny Application

Description

This function generates a Shiny application with modular `dittoViz::scatterPlot()` components. The app features a **Data Import** section for uploading data, a **Data Table** for filtering the active dataset, and a **Plot** area for configuring and displaying an interactive scatter plot.

Usage

```
dittoViz_scatterPlotApp(
  data_list = NULL,
  defaults = NULL,
  hide.inputs = NULL,
  hide.tabs = NULL
)
```

Arguments

`data_list`	An optional named list of data frames. If `NULL` (the default), `list("sales" = gallery_sales)` is used as example data.
`defaults`	A named list of input IDs and their default values to apply on startup.
`hide.inputs`	A character vector of input IDs to hide. Their values are still initialized and used, but the controls are not shown in the UI.
`hide.tabs`	A character vector of tab names to hide. Inputs in these tabs are still initialized and used, but the controls are not shown in the UI.

Details

When `data_list` is not provided (or `NULL`), the app launches with `gallery_sales` as an example dataset. Uploaded data files are added to the available datasets and can be selected for plotting. If an uploaded file shares a name with an existing dataset, the existing one is overwritten with a warning.

This is a convenience wrapper around `createModuleApp()`.

Value

A Shiny app object.

Author(s)

Jared Andrews

See Also

dittoViz::scatterPlot(), dittoViz_scatterPlotInputsUI(), dittoViz_scatterPlotOutputUI(), dittoViz_scatterPlotServer()

Examples

```
library(VizModules)
# Launch with default example data:
app <- dittoViz_scatterPlotApp()
if (interactive()) runApp(app)

# Launch with custom data:
app2 <- dittoViz_scatterPlotApp(list("sales" = example_sales))
if (interactive()) runApp(app2)
```

---

dittoViz_scatterPlotInputsUI

Input UI components for the scatterPlot module

Description

This should be placed in the UI where the inputs should be shown, with an `id` that matches the `id` used in the `dittoViz_scatterPlotServer()` and `dittoViz_scatterPlotOutputUI()` functions.

Usage

```
dittoViz_scatterPlotInputsUI(
  id,
  data,
  defaults = NULL,
  title = NULL,
  columns = 2
)
```

Arguments

`id`	The ID for the Shiny module.
`data`	The data frame used for plot generation.
`defaults`	A named list of default values for the inputs.
`title`	An optional title for the UI grid.
`columns`	Number of columns for the UI grid.

Details

The user inputs for this module are separated from the outputs to allow for more flexible UI design.

The inputs will automatically be organized into a grid layout via the `organize_inputs()` function, with `columns` controlling the number of columns in the grid.

Defaults can be set for each input by providing a named list of values to the `defaults` argument. Nearly all parameters for `dittoViz::scatterPlot()` can be set via these inputs, so see the help for that function for an exhaustive list.

Note that some of the parameters may have input types that differ from the actual function, e.g. `shape.panel` is a text input for comma-separated integers, while the function expects a vector of integers. The module will parse such inputs into the appropriate format for `dittoViz::scatterPlot()` automatically.

Value

A Shiny tagList containing the UI elements

Plot parameters not implemented or with altered functionality

The following `dittoViz::scatterPlot()` parameters are not available via UI inputs or have been superseded:

- `xlab` - X-axis label (auto-generated to reflect any applied X adjustment, e.g. `"log2(z-score(units))"`; plotly allows interactive editing)
- `ylab` - Y-axis label (auto-generated to reflect any applied Y adjustment, e.g. `"log2(z-score(units))"`; plotly allows interactive editing)
- `main` - Plot title (plotly allows interactive editing)
- `sub` - Plot subtitle (not supported in plotly)
- `theme` - ggplot2 theme (not applicable to plotly)
- `legend.title` - Legend title (managed by plotly interactively)
- `legend.color.size` - Legend color size (not supported in plotly)
- `legend.shape.size` - Legend shape size (not supported in plotly)
- `add.xline` - Use `vline.intercepts` instead for vertical lines with full styling options
- `add.yline` - Use `hline.intercepts` instead for horizontal lines with full styling options
- `xline.linetype` - Use `vline.linetypes` instead
- `xline.color` - Use `vline.colors` instead
- `yline.linetype` - Use `hline.linetypes` instead
- `yline.color` - Use `hline.colors` instead
- `do.letter` - Lettering subplots (not implemented for plotly)
- `do.label` - Labeling points interactively (not compatible with plotly hover)
- `labels.size`, `labels.highlight`, `labels.use.numbers`, `labels.numbers.spacer`, `labels.repel`, `labels.repel.adjust`, `labels.split.by` - Point-label styling (tied to `do.label`, not implemented)

- `rename.color.groups` - Rename color groups (not implemented)
- `rename.shape.groups` - Rename shape groups (not implemented)
- `add.trajectory.curves` - Add trajectory curves from coordinate matrices (not implemented; use `add.trajectory.by.groups` instead)
- `do.raster` - Rasterize the point layer (not implemented; use `webgl` for performance instead)
- `raster.dpi` - Rasterization DPI (not applicable without `do.raster`)
- `show.grid.lines` - Toggle grid lines (managed via the Axes tab gridline controls)
- `legend.color.breaks.labels` - Labels for color-scale breaks (not implemented)

The new Lines tab provides enhanced functionality including multiple lines per type, individual line widths, opacities, and diagonal/ablines with slope control.

Plot parameters and defaults

The following `dittoViz::scatterPlot()` parameters can be accessed via UI inputs and/or the `defaults` argument:

- `x.by` - X-axis variable (UI: "X Data", default: 2nd column)
- `y.by` - Y-axis variable (UI: "Y Data", default: 3rd column)
- `color.by` - Coloring variable (UI: "Color By", default: "")
- `shape.by` - Shape variable (UI: "Shape By", default: "")
- `split.by` - Faceting variable (UI: "Split By", default: "")
- `rows.use` - Row filter expression (UI: "Rows Filter", default: "")
- `x.adjustment` - X-axis adjustment (UI: "X Adjustment", default: "")
- `y.adjustment` - Y-axis adjustment (UI: "Y Adjustment", default: "")
- `color.adjustment` - Color adjustment (UI: "Color Adjustment", default: "")
- `x.adj.fxn` - X adjustment function (UI: "X Adjustment Function", default: "")
- `y.adj.fxn` - Y adjustment function (UI: "Y Adjustment Function", default: "")
- `color.adj.fxn` - Color adjustment function (UI: "Color Adjustment Function", default: "")
- `size` - Point size (UI: "Point Size", default: 1)
- `size.by` - Numeric column mapped to point size (UI: "Size By", default: ""); when set, a custom circle size legend is drawn since plotly cannot render a native size legend
- `opacity` - Point opacity (UI: "Point Opacity", default: 1)
- `show.others` - Show others (UI: "Show Others", default: TRUE)
- `split.show.all.others` - Show split others (UI: "Show Split Others", default: TRUE)
- `plot.order` - Plot order (UI: "Plot Order", default: "unordered")
- `shape.panel` - Shape panel values (UI: "Shape Panel", default: "16, 15, 17, 23, 25, 8")
- `min.color` - Minimum color (UI: "Min Color", default: "#F0E442")
- `max.color` - Maximum color (UI: "Max Color", default: "#0072B2")
- `contour.color` - Contour color (UI: "Contour Color", default: "black")
- `contour.linetype` - Contour linetype (UI: "Contour Linetype", default: "solid")

- color.panel - Custom color values (UI: color.panel.ui, derived from palette)
- split.nrow - Number of split rows (UI: "Rows", default: NA)
- split.ncol - Number of split columns (UI: "Columns", default: NA)
- multivar.split.dir - Multivar split direction (UI: "Multivar Split Dir", default: "col")
- split.adjust.scales - Facet scales (UI: "Facet Scales", default: "fixed")
- annotate.by - Annotate by column (UI: "Annotate By", default: "")
- highlight.points - Points to highlight (UI: "Points to Highlight", default: "")
- highlight.color - Highlight fill (UI: "Highlight Fill", default: "#00FFF7")
- highlight.size - Highlight size (UI: "Highlight Size", default: 7)
- highlight.border.color - Highlight border color (UI: "Highlight Border Color", default: "#000000")
- highlight.border.width - Highlight border width (UI: "Highlight Border Width", default: 1)
- highlight.auto.annotate - Auto-annotate highlights (UI: "Auto-annotate Highlights", default: TRUE)
- annotation.color - Annotation color (UI: "Annotation Color", default: "black")
- annotation.ax - Annotation X offset (UI: "Annotation X Offset", default: 20)
- annotation.ay - Annotation Y offset (UI: "Annotation Y Offset", default: -20)
- annotation.size - Annotation size (UI: "Annotation Size", default: 10)
- annotation.showarrow - Show arrow (UI: "Show Arrow", default: TRUE)
- annotation.arrowcolor - Arrow color (UI: "Arrow Color", default: "black")
- annotation.arrowhead - Arrowhead style (UI: "Arrowhead Style", default: 2)
- annotation.arrowwidth - Arrow linewidth (UI: "Arrow Linewidth", default: 1.5)
- legend.color.breaks - Legend tick breaks (UI: "Legend Tick Breaks", default: "")
- size.legend.x - Custom size-legend x position (UI: "Size Legend X Position", default: 1.02); nudges the manual size legend (drawn when size.by is set) along the x-axis.
- size.legend.y - Custom size-legend y position (UI: "Size Legend Y Position", default: 0.95); nudges the manual size legend (drawn when size.by is set) along the y-axis.
- min.value - Minimum value (UI: "Min Value", default: NA)
- max.value - Maximum value (UI: "Max Value", default: NA)
- trajectory.group.by - Trajectory group by (UI: "Trajectory Group By", default: "")
- add.trajectory.by.groups - Add trajectory by groups (UI: "Add Trajectory By Groups", default: "")
- trajectory.arrow.size - Trajectory arrow size (UI: "Trajectory Arrow Size", default: 0.15)
- do.ellipse - Enable ellipses (UI: "Enable Ellipses", default: FALSE)
- do.contour - Enable contour (UI: "Enable Contour", default: FALSE)
- hover.data - Hover data columns (UI: "Hover Data", default: "")
- hover.round.digits - Hover round digits (UI: "Hover Round Digits", default: 5)

Parameters controlling additional functionality

The following parameters implementing new functionality or controlling plotly-specific features are also available:

- `webgl` - Plot with webGL (UI: "Plot with webGL", default: TRUE)
- `shape.fill` - Shape fill color (UI: "Shape Fill", default: "rgba(0, 0, 0, 0)")
- `shape.line.color` - Shape line color (UI: "Shape Line Color", default: "black")
- `shape.line.width` - Shape line width (UI: "Shape Line Width", default: 4)
- `shape.linetype` - Shape linetype (UI: "Shape Linetype", default: "solid")
- `shape.opacity` - Shape opacity (UI: "Shape Opacity", default: 1)
- `title.font.size` - Plot title font size (UI: "Title Size", default: 26)
- `title.font.family` - Font family for title text (UI: "Title Font", default: "Arial")
- `title.font.color` - Color for plot title (UI: "Title Color", default: "#000000")
- `axis.title.font.size` - Axis title font size (UI: "Axis Title Size", default: 18)
- `axis.title.font.color` - Axis title font color (UI: "Axis Title Color", default: "#000000")
- `axis.title.font.family` - Axis title font family (UI: "Axis Title Font", default: "Arial")
- `axis.showline` - Show axis border lines (UI: "Show Axis Borders", default: TRUE)
- `axis.mirror` - Mirror axis lines on opposite side (UI: "Mirror Axis Borders", default: TRUE)
- `show.grid.x` - Show X-axis major gridlines (UI: "Show X Gridlines", default: TRUE)
- `show.grid.y` - Show Y-axis major gridlines (UI: "Show Y Gridlines", default: TRUE)
- `grid.color` - Gridline color (UI: "Gridline Color", default: "#CCCCCC")
- `axis.linecolor` - Color of axis lines (UI: "Axis Line Color", default: "black")
- `axis.linewidth` - Width of axis lines (UI: "Axis Line Width", default: 0.5)
- `axis.tickfont.size` - Size of tick labels (UI: "Tick Label Size", default: 12)
- `axis.tickfont.color` - Color of tick labels (UI: "Tick Label Color", default: "black")
- `axis.tickfont.family` - Font family for tick labels (UI: "Tick Label Font", default: "Arial")
- `axis.tickangle.x` - Rotation angle for X-axis tick labels (UI: "X Tick Label Angle", default: 0)
- `axis.tickangle.y` - Rotation angle for Y-axis tick labels (UI: "Y Tick Label Angle", default: 0)
- `axis.ticks` - Position of tick marks (UI: "Tick Position", default: "outside")
- `axis.tickcolor` - Color of tick marks (UI: "Tick Mark Color", default: "black")
- `axis.ticklen` - Length of tick marks (UI: "Tick Mark Length", default: 5)
- `axis.tickwidth` - Width of tick marks (UI: "Tick Mark Width", default: 1)
- `facet.title.font.size` - Facet subplot title font size (UI: "Facet Subplot Title Size", default: 18)
- `facet.title.font.color` - Facet subplot title font color (UI: "Facet Title Color", default: "#000000")

- `facet.title.font.family` - Facet subplot title font family (UI: "Facet Title Font", default: "Arial")
- `hline.intercepts` - Y-coordinates for horizontal reference lines (UI: "Y-intercepts", default: "")
- `hline.colors` - Colors for horizontal lines (UI: "Colors", default: "#000000")
- `hline.widths` - Widths for horizontal lines (UI: "Widths", default: "1")
- `hline.linetypes` - Line types for horizontal lines (UI: "Line Types", default: "dashed")
- `hline.opacities` - Opacities for horizontal lines (UI: "Opacities (0-1)", default: "1")
- `vline.intercepts` - X-coordinates for vertical reference lines (UI: "X-intercepts", default: "")
- `vline.colors` - Colors for vertical lines (UI: "Colors", default: "#000000")
- `vline.widths` - Widths for vertical lines (UI: "Widths", default: "1")
- `vline.linetypes` - Line types for vertical lines (UI: "Line Types", default: "dashed")
- `vline.opacities` - Opacities for vertical lines (UI: "Opacities (0-1)", default: "1")
- `abline.slopes` - Slopes for diagonal reference lines (UI: "Slopes", default: "")
- `best.fit` - Enable line of best fit (UI: "Line of best fit:", default: FALSE)
- `line.best.smoothness` - Smoothness of line of best fit (UI: "Smoothness of line of best fit:", default: 1)
- `line.best.colour` - Color of line of best fit (UI: "Line of best fit colour:", default: "#000000")
- `linear.model` - Enable linear model line (UI: "Linear model line", default: FALSE)

Author(s)

Jared Andrews

See Also

`dittoViz::scatterPlot()`, `organize_inputs()`, `dittoViz_scatterPlotOutputUI()`, `dittoViz_scatterPlotServer()`, `dittoViz_scatterPlotApp()`

Examples

```
library(VizModules)
dittoViz_scatterPlotInputsUI("scatterPlot", example_mtcars)
```

dittoViz_scatterPlotOutputUI

Output UI components for the scatterPlot module

Description

This should be placed in the UI where the plot should be shown.

Usage

```
dittoViz_scatterPlotOutputUI(id, resizable = TRUE)
```

Arguments

id	The ID for the Shiny module.
resizable	Logical; when TRUE (the default) the plot output is wrapped in shinyjqui::jqui_resizable() so it can be resized by dragging. Set to FALSE when embedding the output in a container that already provides resizing.

Value

A Shiny plotlyOutput for the scatterplot

Author(s)

Jared Andrews

dittoViz_scatterPlotServer

Server logic for scatterPlot module

Description

Server logic for scatterPlot module

Usage

```
dittoViz_scatterPlotServer(
  id,
  data,
  hide.inputs = NULL,
  hide.tabs = NULL,
  manual.colors = NULL,
  defaults = NULL
)
```

Arguments

id	The ID for the Shiny module.
data	A reactive containing the data frame to plot.
hide.inputs	A character vector of input IDs to hide. These will still be initialized and their values passed to the plot function, but the user will not be able to see/adjust them in the UI.
hide.tabs	A character vector of tab names to hide. Inputs in these tabs will still be initialized and their values passed to the plot function, but the user will not be able to see/adjust them in the UI.
manual.colors	A named character vector of colors or a reactive returning a named character vector of colors.
defaults	A named list of default values for the inputs. When the reset button is clicked, inputs are reset to these values rather than hardcoded fallbacks. Typically the same list passed to the corresponding UI function.

Value

The moduleServer function for the scatterPlot module.

Author(s)

Jared Andrews

See Also

dittoViz::scatterPlot(), dittoViz_scatterPlotInputsUI(), dittoViz_scatterPlotOutputUI(), dittoViz_scatterPlotApp()

---

dittoViz_yPlotApp *Create an example Modular yPlot Shiny Application*

Description

This function generates a Shiny application with modular dittoViz::yPlot() components. The app features a **Data Import** section for uploading data, a **Data Table** for filtering the active dataset, and a **Plot** area for configuring and displaying an interactive y plot.

Usage

```
dittoViz_yPlotApp(
  data_list = NULL,
  defaults = NULL,
  hide.inputs = NULL,
  hide.tabs = NULL
)
```

Arguments

`data_list`	An optional named list of data frames. If `NULL` (the default), `list("demographics" = gallery_demographics)` is used as example data.
`defaults`	A named list of input IDs and their default values to apply on startup.
`hide.inputs`	A character vector of input IDs to hide. Their values are still initialized and used, but the controls are not shown in the UI.
`hide.tabs`	A character vector of tab names to hide. Inputs in these tabs are still initialized and used, but the controls are not shown in the UI.

Details

When `data_list` is not provided (or `NULL`), the app launches with `gallery_demographics` as an example dataset. Uploaded data files are added to the available datasets and can be selected for plotting. If an uploaded file shares a name with an existing dataset, the existing one is overwritten with a warning.

This is a convenience wrapper around `createModuleApp()`.

Value

A Shiny app object.

Author(s)

Jared Andrews

See Also

`dittoViz::yPlot()`, `dittoViz_yPlotInputsUI()`, `dittoViz_yPlotOutputUI()`, `dittoViz_yPlotServer()`

Examples

```
library(VizModules)
# Launch with default example data:
app <- dittoViz_yPlotApp()
if (interactive()) runApp(app)

# Launch with custom data:
app2 <- dittoViz_yPlotApp(list("demographics" = example_demographics))
if (interactive()) runApp(app2)
```

dittoViz_yPlotInputsUI

Input UI components for the yPlot module

Description

This should be placed in the UI where the inputs should be shown, with an `id` that matches the `id` used in the `dittoViz_yPlotServer()` and `dittoViz_yPlotOutputUI()` functions.

Usage

```
dittoViz_yPlotInputsUI(id, data, defaults = NULL, title = NULL, columns = 2)
```

Arguments

`id`	The ID for the Shiny module.
`data`	The data frame used for plot generation.
`defaults`	A named list of default values for the inputs.
`title`	An optional title for the UI grid.
`columns`	Number of columns for the UI grid.

Details

The user inputs for this module are separated from the outputs to allow for more flexible UI design.

The inputs will automatically be organized into a grid layout via the `organize_inputs()` function, with `columns` controlling the number of columns in the grid.

Defaults can be set for each input by providing a named list of values to the `defaults` argument. Nearly all parameters for `dittoViz::yPlot()` can be set via these inputs, so see the help for that function for an exhaustive list.

Value

A Shiny tagList containing the UI elements

Plot parameters not implemented or with altered functionality

The following `dittoViz::yPlot()` parameters are not available via UI inputs:

- `xlab` - X-axis label (plotly allows interactive editing)
- `ylab` - Y-axis label (auto-generated to reflect any applied Y adjustment, e.g. `"log2(z-score(units))"`; plotly allows interactive editing)
- `main` - Plot title (plotly allows interactive editing)
- `sub` - Plot subtitle (not supported in plotly)
- `theme` - ggplot2 theme (not applicable to plotly)

- `legend.title` - Legend title (managed by plotly interactively)
- `add.line` - Use `hline.intercepts` instead for horizontal lines with full styling options
- `line.linetype` - Use `hline.linetypes` instead
- `line.color` - Use `hline.colors` instead
- `line.linewidth` - Use `hline.widths` instead
- `line.opacity` - Use `hline.opacities` instead
- `multivar.aes` - Aesthetic used for multiple `var` columns (not implemented; one var at a time)
- `multivar.split.dir` - Facet direction for multiple `var` columns (not implemented)
- `rows.use` - Row subset to plot (not implemented)
- `colors` - Integer index/order into `color.panel` (managed via the palette UI)
- `shape.panel` - Shapes used with `shape.by` (not implemented)
- `y.breaks` - Custom continuous-axis breaks (not implemented)
- `x.labels` - Override group labels (not implemented)
- `x.labels.rotate` - Rotate group labels (handled by the Axes tab tick-angle controls)
- `x.reorder` - Reorder x-axis groups (not implemented)
- `boxplot.width` - Boxplot width (controlled via `boxgap` and `boxgroupgap`)
- `boxplot.outlier.size` - Outlier point size (not implemented)
- `boxplot.position.dodge` - Boxplot dodge (controlled via `boxgap`)
- `hover.data` - Columns shown on hover (not implemented; a default set is used)
- `hover.round.digits` - Hover value rounding (not implemented)
- `vlnplot.quantiles` - Violin quantiles (doesn't translate to plotly)

Plot parameters and defaults

The following `dittoViz::yPlot()` parameters can be accessed via UI inputs and/or the `defaults` argument:

- `var` - Y-axis variable (UI: "Y data (var)", default: 2nd numeric variable)
- `group.by` - Grouping variable for x-axis (UI: "Group by", default: 2nd categorical variable)
- `color.by` - Coloring variable (UI: "Color by", default: "")
- `shape.by` - Shape variable (UI: "Shape by", default: "")
- `split.by` - Faceting variable (UI: "Split by (facet)", default: "")
- `plots` - Plot types to show (UI: "Plots to show", default: c("boxplot", "jitter"))
- `color.panel` - Custom color values (UI: palette picker, derived from palette)
- `min` - Y-axis minimum (UI: "Y Axis Min", auto-calculated)
- `max` - Y-axis maximum (UI: "Y Axis Max", auto-calculated)
- `var.adjustment` - Y-axis data adjustment (UI: "Y Adjustment", default: "")
- `var.adj.fxn` - Y-axis adjustment function (UI: "Y Adjustment Function", default: "")
- `split.nrow` - Number of facet rows (UI: "Rows", default: 4)

- split.ncol - Number of facet columns (UI: "Columns", default: 4)
- split.adjust - Facet scale behavior (UI: "Facet Scaling", default: "fixed")
- do.raster - Rasterize jitter points (UI: "Rasterize Jitter", default: FALSE)
- raster.dpi - DPI for rasterization (UI: "Raster DPI", default: 600)
- jitter.size - Jitter point size (UI: "Jitter Point Size", default: 1)
- jitter.width - Jitter width (UI: "Jitter Width", default: 0.2)
- jitter.color - Jitter border color (UI: "Jitter Border Color", default: "#000000")
- jitter.shape.legend.size - Shape legend size (UI: "Shape Legend Size", default: 5)
- jitter.shape.legend.show - Show shape legend (UI: "Show Shape Legend", default: TRUE)
- jitter.position.dodge - Jitter position dodge (calculated from boxgap)
- boxplot.show.outliers - Show boxplot outliers
- boxplot.color - Boxplot outline color (UI: "Boxplot Color", default: "#000000")
- boxplot.fill - Fill boxplot (UI: "Fill Boxplot", default: TRUE)
- boxplot.lineweight - Boxplot line weight (UI: "Boxplot Line Weight", default: 0.5)
- vlnplot.lineweight - Violin line weight (UI: "Violin Line Weight", default: 0.5)
- vlnplot.scaling - Violin scaling method (UI: "Violin Scaling", default: "area")
- vlnplot.width - Violin width (derived from boxgap; not directly settable)
- ridgeplot.lineweight - Ridge line weight (UI: "Ridge Line Weight", default: 0.5)
- ridgeplot.scale - Ridge overlap scale (UI: "Ridge Scale (overlap)", default: 1.25)
- ridgeplot.ymax.expansion - Ridge Y-max expansion (UI: "Ridge Y-max Expansion", default: NA)
- ridgeplot.shape - Ridge shape (UI: "Ridge Shape", default: "smooth")
- ridgeplot.bins - Ridge bins (UI: "Ridge Bins", default: 30)
- ridgeplot.binwidth - Ridge binwidth (UI: "Ridge Binwidth", default: NULL)
- legend.show - Show legend (always TRUE; not directly settable)

Parameters controlling additional functionality

The following parameters implementing new functionality or controlling plotly-specific features are also available:

- boxmode - Boxplot mode grouping (calculated: "group" or "overlay" based on color.by)
- boxgap - Boxplot position dodge (UI: "Boxplot Position Dodge", default: 0.3)
- boxgroupgap - Boxplot group dodge (UI: "Boxplot Group Dodge", default: 0.2)
- title.font.size - Plot title font size (UI: "Title Size", default: 26)
- title.font.family - Font family for title text (UI: "Title Font", default: "Arial")
- title.font.color - Color for plot title (UI: "Title Color", default: "#000000")
- axis.title.font.size - Axis title font size (UI: "Axis Title Size", default: 18)
- axis.title.font.color - Axis title font color (UI: "Axis Title Color", default: "#000000")

- `axis.title.font.family` - Axis title font family (UI: "Axis Title Font", default: "Arial")
- `axis.showline` - Show axis border lines (UI: "Show Axis Lines", default: TRUE)
- `axis.mirror` - Mirror axis lines on opposite side (UI: "Mirror Axis Lines", default: TRUE)
- `show.grid.x` - Show X-axis major gridlines (UI: "Show X Major Gridlines", default: TRUE)
- `show.grid.y` - Show Y-axis major gridlines (UI: "Show Y Major Gridlines", default: TRUE)
- `axis.linecolor` - Color of axis lines (UI: "Axis Line Color", default: "black")
- `axis.linewidth` - Width of axis lines (UI: "Axis Line Width", default: 0.5)
- `axis.tickfont.size` - Size of tick labels (UI: "Tick Label Size", default: 12)
- `axis.tickfont.color` - Color of tick labels (UI: "Tick Label Color", default: "black")
- `axis.tickfont.family` - Font family for tick labels (UI: "Tick Label Font", default: "Arial")
- `axis.tickangle.x` - Rotation angle for X-axis tick labels (UI: "X-axis Tick Label Angle", default: 0)
- `axis.tickangle.y` - Rotation angle for Y-axis tick labels (UI: "Y-axis Tick Label Angle", default: 0)
- `axis.ticks` - Position of tick marks (UI: "Tick Position", default: "outside")
- `axis.tickcolor` - Color of tick marks (UI: "Tick Mark Color", default: "black")
- `axis.ticklen` - Length of tick marks (UI: "Tick Mark Length", default: 5)
- `axis.tickwidth` - Width of tick marks (UI: "Tick Mark Width", default: 1)
- `hline.intercepts` - Y-coordinates for horizontal reference lines (UI: "Y-intercepts", default: "")
- `hline.colors` - Colors for horizontal lines (UI: "Colors", default: "#000000")
- `hline.widths` - Widths for horizontal lines (UI: "Widths", default: "1")
- `hline.linetypes` - Line types for horizontal lines (UI: "Line Types", default: "dashed")
- `hline.opacities` - Opacities for horizontal lines (UI: "Opacities (0-1)", default: "1")
- `vline.intercepts` - X-coordinates for vertical reference lines (UI: "X-intercepts", default: "")
- `vline.colors` - Colors for vertical lines (UI: "Colors", default: "#000000")
- `vline.widths` - Widths for vertical lines (UI: "Widths", default: "1")
- `vline.linetypes` - Line types for vertical lines (UI: "Line Types", default: "dashed")
- `vline.opacities` - Opacities for vertical lines (UI: "Opacities (0-1)", default: "1")
- `abline.slopes` - Slopes for diagonal reference lines (UI: "Slopes", default: "")
- `abline.intercepts` - Y-intercepts for diagonal lines (UI: "Y-intercepts", default: "")
- `abline.colors` - Colors for diagonal lines (UI: "Colors", default: "#000000")
- `abline.widths` - Widths for diagonal lines (UI: "Widths", default: "1")
- `abline.linetypes` - Line types for diagonal lines (UI: "Line Types", default: "dashed")
- `abline.opacities` - Opacities for diagonal lines (UI: "Opacities (0-1)", default: "1")

Author(s)

Jared Andrews, Jacob Martin

See Also

dittoViz::yPlot(), organize_inputs(), dittoViz_yPlotOutputUI(), dittoViz_yPlotServer(), dittoViz_yPlotApp()

Examples

```
library(VizModules)
data(mtcars)
dittoViz_yPlotInputsUI("yPlot", mtcars)
```

---

dittoViz_yPlotOutputUI

Output UI components for the yPlot module

Description

This should be placed in the UI where the plot should be shown.

Usage

```
dittoViz_yPlotOutputUI(id, resizable = TRUE)
```

Arguments

id	The ID for the Shiny module.
resizable	Logical; when TRUE (the default) the plot output is wrapped in shinyjqui::jqui_resizable() so it can be resized by dragging. Set to FALSE when embedding the output in a container that already provides resizing.

Value

A Shiny plotlyOutput for the yPlot

Author(s)

Jared Andrews

`dittoViz_yPlotServer` *Server logic for yPlot module*

Description

Server logic for yPlot module

Usage

```
dittoViz_yPlotServer(
  id,
  data,
  hide.inputs = NULL,
  hide.tabs = NULL,
  defaults = NULL
)
```

Arguments

`id`	The ID for the Shiny module.
`data`	A `reactive` containing the data frame to plot.
`hide.inputs`	A character vector of input IDs to hide. These will still be initialized and their values passed to the plot function, but the user will not be able to see/adjust them in the UI.
`hide.tabs`	A character vector of tab names to hide. Inputs in these tabs will still be initialized and their values passed to the plot function, but the user will not be able to see/adjust them in the UI.
`defaults`	A named list of default values for the inputs. When the reset button is clicked, inputs are reset to these values rather than hardcoded fallbacks. Typically the same list passed to the corresponding UI function.

Value

The `moduleServer` function for the yPlot module.

Author(s)

Jared Andrews, Jacob Martin

See Also

`dittoViz::yPlot()`, `dittoViz_yPlotInputsUI()`, `dittoViz_yPlotOutputUI()`, `dittoViz_yPlotApp()`

dumbbellPlot *Create an Interactive Dumbbell Plot with plotly*

Description

Generates a customizable interactive dumbbell plot using plotly. Supports single dot mode (1 x variable) or dumbbell mode (2 x variables), with flexible coloring by either X or Y variables, faceting, and transformations.

Usage

```
dumbbellPlot(
  data,
  x,
  y,
  colour.by = "X variables",
  palette.selection,
  show.legend = TRUE,
  facet.by = NULL,
  line.colour = "gray80",
  facet.scales = "fixed",
  subplot.margin = 0.05,
  axis.showline = TRUE,
  axis.mirror = TRUE,
  axis.linecolor = "black",
  axis.linewidth = 0.5,
  axis.tickfont.size = 12,
  axis.tickfont.color = "black",
  axis.tickfont.family = "Arial",
  axis.tickangle.x = 0,
  axis.tickangle.y = 0,
  axis.ticks = "outside",
  axis.tickcolor = "black",
  axis.ticklen = 5,
  axis.tickwidth = 1,
  title.text = "",
  title.font.size = 26,
  title.font.family = "Arial",
  title.font.color = "black",
  title.x.position = 0.47,
  y.title = NULL,
  x.title = NULL,
  flip.x = FALSE,
  flip.y = FALSE,
  x.adjustment = NULL,
  order.by = NULL
)
```

Arguments

`data`	A data.frame or tibble containing the data to plot.
`x`	Character vector of column name(s) for x-axis values. Maximum 2 values allowed. If 1 value: creates single dot plot. If 2 values: creates dumbbell plot with connecting segments.
`y`	Character, column name for the y-axis (categorical variable recommended).
`colour.by`	Character, how to color the markers. Options: "X variables" (different colors for each x variable) or "Y variables" (different colors for each y category). Default: "X variables".
`palette.selection`	Character vector of hex colors for marker colors.
`show.legend`	Logical, whether to display the legend. Default: TRUE.
`facet.by`	Optional character, column name to facet plots by. Creates subplots for each unique value. Default: NULL.
`line.colour`	Character, hex color for the connecting lines between dumbbell points. Default: "gray80".
`facet.scales`	Character, controls axis scaling across facets. Options: "fixed" (same for all), "free" (independent), "free_x" (independent x-axis), "free_y" (independent y-axis). Default: "fixed".
`subplot.margin`	Numeric, spacing between facet panels as a fraction of the plot area. May be a single value (applied to both directions) or a length-2 vector `c(horizontal, vertical)` to control the gap between columns and rows separately. Default: 0.06.
`axis.showline`	Logical, whether to show axis border lines. Default: TRUE.
`axis.mirror`	Logical, whether to mirror axis lines on opposite side of plot. Default: TRUE.
`axis.linecolor`	Character, hex color for axis lines. Default: "black".
`axis.linewidth`	Numeric, width of axis lines in pixels. Default: 0.5.
`axis.tickfont.size`	Numeric, font size for axis tick labels. Default: 12.
`axis.tickfont.color`	Character, hex color for axis tick labels. Default: "black".
`axis.tickfont.family`	Character, font family for axis tick labels. Default: "Arial".
`axis.tickangle.x`	Numeric, rotation angle for x-axis tick labels in degrees. Default: 0.
`axis.tickangle.y`	Numeric, rotation angle for y-axis tick labels in degrees. Default: 0.
`axis.ticks`	Character, position of tick marks. Options: "outside", "inside", "none". Default: "outside".
`axis.tickcolor`	Character, hex color for tick marks. Default: "black".
`axis.ticklen`	Numeric, length of tick marks in pixels. Default: 5.
`axis.tickwidth`	Numeric, width of tick marks in pixels. Default: 1.

`title.text` Character, main title text for the plot. Default: "".

`title.font.size` Numeric, font size for plot title. Default: 26.

`title.font.family` Character, font family for plot title. Default: "Arial".

`title.font.color` Character, hex color for plot title text. Default: "black".

`title.x.position` Numeric, horizontal position of the plot title in paper coordinates (0 = left, 1 = right). Default: 0.47.

`y.title` Optional character, label for y-axis. If NULL, auto-generated from column name. Default: NULL.

`x.title` Optional character, label for x-axis. If NULL, auto-generated from column name. Default: NULL.

`flip.x` Logical, whether to reverse the x-axis direction. Default: FALSE.

`flip.y` Logical, whether to reverse the y-axis direction. Default: FALSE.

`x.adjustment` Optional character or function, transformation to apply to x values. Options: "log2", "log", "log10", "neg_log10", "log1p", "as.factor", "abs", "sqrt", or custom function. Default: NULL.

`order.by` Optional character vector, column name(s) to order data by before plotting. Default: NULL.

Details

The dumbbell plot is designed for comparing two values across categories.

Modes:

- **Single dot mode** (1 x variable): Shows one marker per y category
- **Dumbbell mode** (2 x variables): Shows two markers connected by a line per y category

Coloring options:

- **By X variables**: Each x variable gets a different color (e.g., Male=blue, Female=pink)
- **By Y variables**: Each y category gets a different color (e.g., School A=red, School B=blue)

Value

A plotly object representing the interactive dumbbell plot.

Author(s)

Jacob Martin

Examples

```
data <- data.frame(
    School = c("MIT", "Stanford", "Harvard"),
    Women = c(152, 96, 112),
    Men = c(95, 151, 165)
)

fig <- dumbbellPlot(
    data = data,
    x = c("Women", "Men"),
    y = "School",
    colour.by = "X variables",
    palette.selection = c("green", "blue"),
    show.legend = TRUE,
    line.colour = "gray80"
)
```

`dumbbellPlotApp` *Create a Shiny App for Dumbbell Plots*

Description

This function generates a Shiny application for interactive dumbbell plots. The app features a **Data Import** section for uploading data, a **Data Table** for filtering the active dataset, and a **Plot** area for configuring and displaying an interactive dumbbell plot.

Usage

```
dumbbellPlotApp(
  data_list = NULL,
  defaults = NULL,
  hide.inputs = NULL,
  hide.tabs = NULL
)
```

Arguments

`data_list`	An optional named list of data frames. If `NULL` (the default), `list("school_earnings" = example_school_earnings)` is used as example data.
`defaults`	A named list of input IDs and their default values to apply on startup.
`hide.inputs`	A character vector of input IDs to hide. Their values are still initialized and used, but the controls are not shown in the UI.
`hide.tabs`	A character vector of tab names to hide. Inputs in these tabs are still initialized and used, but the controls are not shown in the UI.

Details

When `data_list` is not provided (or `NULL`), the app launches with `example_school_earnings` as an example dataset. Uploaded data files are added to the available datasets and can be selected for plotting. If an uploaded file shares a name with an existing dataset, the existing one is overwritten with a warning.

This is a convenience wrapper around `createModuleApp()`.

Value

A Shiny app object.

Author(s)

Jacob Martin, Jared Andrews

See Also

`dumbbellPlot()`, `dumbbellPlotInputsUI()`, `dumbbellPlotOutputUI()`, `dumbbellPlotServer()`

Examples

```
library(VizModules)
# Launch with default example data:
app <- dumbbellPlotApp()
if (interactive()) runApp(app)

# Launch with custom data:
data <- data.frame(
    School = c("MIT", "Stanford", "Harvard"),
    Women = c(94, 96, 112),
    Men = c(152, 151, 165),
    Group = c("A", "B", "A")
)
app2 <- dumbbellPlotApp(list("School Earnings" = data))
if (interactive()) runApp(app2)
```

---

dumbbellPlotInputsUI *Input UI components for the dumbbellPlot module*

Description

This should be placed in the UI where the inputs should be shown, with an `id` that matches the `id` used in the `dumbbellPlotServer()` and `dumbbellPlotOutputUI()` functions.

Usage

```
dumbbellPlotInputsUI(id, data, defaults = NULL, title = NULL, columns = 2)
```

Arguments

id	The ID for the Shiny module.
data	The data frame used for plot generation.
defaults	A named list of default values for the inputs.
title	An optional title for the UI grid.
columns	Number of columns for the UI grid.

Details

The user inputs for this module are separated from the outputs to allow for more flexible UI design.

The inputs will automatically be organized into a grid layout via the `organize_inputs()` function, with `columns` controlling the number of columns in the grid.

Defaults can be set for each input by providing a named list of values to the `defaults` argument.

Value

A Shiny tagList containing the UI elements

Plot parameters not implemented or with altered functionality

The following `dumbbellPlot()` parameters are not exposed as UI inputs:

- `order.by` - Order rows by a Y value; not currently wired to a UI input (use `defaults` to set)

Plot parameters and defaults

The following `dumbbellPlot()` parameters can be accessed via UI inputs:

- `x` - X values (UI: "X Values (max 2)", defaults key: `x.value`, multiple: TRUE, max 2 enforced)
- `y` - Y value (UI: "Y Value", defaults key: `y.value`, single selection)
- `x.adjustment` - X-axis transformation (UI: "X Adjustment", default: "")
- `colour.by` - Color by X or Y (UI: "Colour By", default: "X variables")
- `facet.by` - Faceting variable (UI: "Facet By", default: "")
- `facet.scales` - Facet scale behavior (UI: "Facet Scales", default: "fixed")
- `line.colour` - Color of connecting lines (UI: "Colour of Connectors", default: "gray30")
- `palette.selection` - Color palette (UI: palette picker)

Parameters controlling additional functionality

The following parameters controlling plotly-specific features and styling are also available:

- `flip.x` - Flip X-axis (UI: "Flip X Axis", default: FALSE)
- `flip.y` - Flip Y-axis (UI: "Flip Y Axis", default: FALSE)
- `title.font.size` - Plot title font size (UI: "Title Size", default: 26)

- `title.font.family` - Font family for title text (UI: "Title Font", default: "Arial")
- `title.font.color` - Color for plot title (UI: "Title Color", default: "#000000")
- `axis.title.font.size` - Axis title font size (UI: "Axis Title Size", default: 18)
- `axis.title.font.color` - Axis title font color (UI: "Axis Title Color", default: "#000000")
- `axis.title.font.family` - Axis title font family (UI: "Axis Title Font", default: "Arial")
- `axis.showline` - Show axis border lines (UI: "Show Axis Borders", default: TRUE)
- `axis.mirror` - Mirror axis lines on opposite side (UI: "Mirror Axis Borders", default: TRUE)
- `show.grid.x` - Show X-axis major gridlines (UI: "Show X Gridlines", default: TRUE)
- `show.grid.y` - Show Y-axis major gridlines (UI: "Show Y Gridlines", default: TRUE)
- `grid.color` - Gridline color (UI: "Gridline Color", default: "#CCCCCC")
- `axis.linecolor` - Color of axis lines (UI: "Axis Line Color", default: "black")
- `axis.linewidth` - Width of axis lines (UI: "Axis Line Width", default: 0.5)
- `axis.tickfont.size` - Size of tick labels (UI: "Tick Label Size", default: 12)
- `axis.tickfont.color` - Color of tick labels (UI: "Tick Label Color", default: "black")
- `axis.tickfont.family` - Font family for tick labels (UI: "Tick Label Font", default: "Arial")
- `axis.tickangle.x` - Rotation angle for X-axis tick labels (UI: "X Tick Label Angle", default: 0)
- `axis.tickangle.y` - Rotation angle for Y-axis tick labels (UI: "Y Tick Label Angle", default: 0)
- `axis.ticks` - Position of tick marks (UI: "Tick Position", default: "outside")
- `axis.tickcolor` - Color of tick marks (UI: "Tick Mark Color", default: "black")
- `axis.ticklen` - Length of tick marks (UI: "Tick Mark Length", default: 5)
- `axis.tickwidth` - Width of tick marks (UI: "Tick Mark Width", default: 1)
- `facet.title.font.size` - Facet subplot title font size (UI: "Facet Subplot Title Size", default: 18)
- `facet.title.font.color` - Facet subplot title font color (UI: "Facet Title Color", default: "#000000")
- `facet.title.font.family` - Facet subplot title font family (UI: "Facet Title Font", default: "Arial")
- `hline.intercepts` - Y-coordinates for horizontal reference lines (UI: "Y-intercepts", default: "")
- `hline.colors` - Colors for horizontal lines (UI: "Colors", default: "#000000")
- `hline.widths` - Widths for horizontal lines (UI: "Widths", default: "1")
- `hline.linetypes` - Line types for horizontal lines (UI: "Line Types", default: "dashed")
- `hline.opacities` - Opacities for horizontal lines (UI: "Opacities (0-1)", default: "1")
- `vline.intercepts` - X-coordinates for vertical reference lines (UI: "X-intercepts", default: "")
- `vline.colors` - Colors for vertical lines (UI: "Colors", default: "#000000")
- `vline.widths` - Widths for vertical lines (UI: "Widths", default: "1")

- `vline.linetypes` - Line types for vertical lines (UI: "Line Types", default: "dashed")
- `vline.opacities` - Opacities for vertical lines (UI: "Opacities (0-1)", default: "1")
- `abline.slopes` - Slopes for diagonal reference lines (UI: "Slopes", default: "")
- `abline.intercepts` - Y-intercepts for diagonal lines (UI: "Y-intercepts", default: "")
- `abline.colors` - Colors for diagonal lines (UI: "Colors", default: "#000000")
- `abline.widths` - Widths for diagonal lines (UI: "Widths", default: "1")
- `abline.linetypes` - Line types for diagonal lines (UI: "Line Types", default: "dashed")
- `abline.opacities` - Opacities for diagonal lines (UI: "Opacities (0-1)", default: "1")
- `margin.t` - Top margin in pixels (UI: "Margin Top", default: 70)
- `margin.b` - Bottom margin in pixels (UI: "Margin Bottom", default: 70)
- `margin.l` - Left margin in pixels (UI: "Margin Left", default: 70)
- `margin.r` - Right margin in pixels (UI: "Margin Right", default: 70)
- `subplot.margin.x` - Horizontal spacing between facet panel columns (UI: "Subplot Spacing (Horizontal)")
- `subplot.margin.y` - Vertical spacing between facet panel rows (UI: "Subplot Spacing (Vertical)")
- `shape.fill` - Fill color for drawn shapes (UI: "Shape Fill", default: "rgba(0, 0, 0, 0)")
- `shape.line.color` - Outline color for drawn shapes (UI: "Shape Line Color", default: "black")
- `shape.line.width` - Outline width for drawn shapes (UI: "Shape Line Width", default: 4)
- `shape.linetype` - Line dash style for drawn shapes (UI: "Shape Linetype", default: "solid")
- `shape.opacity` - Opacity for drawn shapes (UI: "Shape Opacity", default: 1)

Author(s)

Jacob Martin

See Also

`dumbbellPlot()`, `organize_inputs()`, `dumbbellPlotOutputUI()`, `dumbbellPlotServer()`, `dumbbellPlotApp()`

Examples

```
library(VizModules)
data <- data.frame(
    School = c("MIT", "Stanford", "Harvard"),
    Women = c(94, 96, 112),
    Men = c(152, 151, 165)
)
dumbbellPlotInputsUI("dumbbellPlot", data)
```

dumbbellPlotOutputUI *Output UI components for the dumbbellPlot module*

Description

This should be placed in the UI where the plot should be shown.

Usage

```
dumbbellPlotOutputUI(id, resizable = TRUE)
```

Arguments

id	The ID for the Shiny module.
resizable	Logical; when TRUE (the default) the plot output is wrapped in shinyjqui::jqui_resizable() so it can be resized by dragging. Set to FALSE when embedding the output in a container that already provides resizing.

Value

A Shiny plotlyOutput for the dumbbellPlot

Author(s)

Jacob Martin, Jared Andrews

dumbbellPlotServer *Server logic for dumbbellPlot module*

Description

Server logic for dumbbellPlot module

Usage

```
dumbbellPlotServer(
  id,
  data,
  hide.inputs = NULL,
  hide.tabs = NULL,
  defaults = NULL
)
```

Arguments

id	The ID for the Shiny module.
data	A reactive containing the data frame to plot.
hide.inputs	A character vector of input IDs to hide. These will still be initialized and their values passed to the plot function, but the user will not be able to see/adjust them in the UI.
hide.tabs	A character vector of tab names to hide. Inputs in these tabs will still be initialized and their values passed to the plot function, but the user will not be able to see/adjust them in the UI.
defaults	A named list of default values for the inputs. When the reset button is clicked, inputs are reset to these values rather than hardcoded fallbacks. Typically the same list passed to the corresponding UI function.

Value

The moduleServer function for the dumbbellPlot module.

Author(s)

Jacob Martin

See Also

dumbbellPlot(), dumbbellPlotInputsUI(), dumbbellPlotOutputUI(), dumbbellPlotApp()

---

empty_plot *Create an empty ggplot2 plot or plotly plot with input text*

Description

This function creates an empty ggplot2 or plotly plot and places a user-provided text string in the middle of the plot.

Usage

```
empty_plot(text = NULL, plotly = FALSE)
```

Arguments

text	Character scalar to show in plot area.
plotly	Boolean indicating whether to return a plotly object.

Value

Either a ggplot object or a plotly object if plotly = TRUE.

Author(s)

Jared Andrews

See Also

ggplot2::geom_text(), ggplot2::theme_void()

Examples

```
library(VizModules)
empty_plot("No data to display")
```

---

example_bar *Bar dataset for bar and split bar plot examples*

Description

A small dataset with five groups, two categorical variables, and three numeric variables. Used as the default data for plotthis_BarPlotApp() and plotthis_SplitBarPlotApp().

Usage

```
example_bar
```

Format

A data frame with 5 rows and 5 columns:

Group Group label (A through E)

Type Category type (Alpha, Beta, or Gamma)

Values Primary numeric values (positive)

Numbers Secondary numeric values (can be negative)

Score Tertiary numeric values (can be negative)

Author(s)

Jacob Martin

Source

Generated in data-raw/generate_example_data.R.

`example_demographics` *Example demographics dataset*

Description

A simulated employee survey dataset with 500 rows spanning six departments and four job levels. Designed to showcase violin, box, yPlot, density, and histogram plot modules with realistic numeric distributions.

Usage

```
example_demographics
```

Format

A data frame with 500 rows and 9 columns:

department Employee department (factor: Engineering, Marketing, Sales, HR, Finance, Operations)

job_level Job seniority level (factor: Junior, Mid, Senior, Lead)

gender Employee gender (factor: Male, Female)

age Employee age in years

salary Annual salary in USD

satisfaction Job satisfaction score (1–10)

performance Performance rating (1–10)

tenure_years Years with the company

weekly_hours Average weekly hours worked (35–65)

Author(s)

Jared Andrews

Source

Simulated in data-raw/generate_example_data.R.

example_iris *Example grouped iris dataset*

Description

The classic iris dataset with an added 'Group' column to facilitate multi-group plot examples.

Usage

```
example_iris
```

Format

A data frame with 150 rows and 6 columns:

Sepal.Length Sepal length in cm

Sepal.Width Sepal width in cm

Petal.Length Petal length in cm

Petal.Width Petal width in cm

Species Species of the iris (factor: setosa, versicolor, virginica)

Group Group assignment (factor: A, B, C, D)

Author(s)

Jared Andrews

Source

Generated from the classic iris dataset.

example_markers *Example single-cell marker gene dataset for dot plots*

Description

A simulated single-cell marker-gene expression dataset with 104 rows covering eight immune cell types and thirteen canonical marker genes. Each cell type strongly expresses its own marker genes (high average expression and percent expressed) and weakly expresses the rest, making it a realistic example for `plotthis_DotPlotApp()` where dot size encodes the percent of cells expressing a gene and dot fill encodes average expression.

Usage

```
example_markers
```

Format

A data frame with 104 rows and 4 columns:

cell_type Immune cell type (factor: CD4 T, CD8 T, B, NK, Monocyte, Dendritic, Plasma, Platelet)

gene Marker gene symbol (factor with 13 levels, e.g. CD3D, MS4A1, NKG7, LYZ, MZB1, PPBP)

avg_expression Average expression of the gene in the cell type

pct_expressed Percent of cells in the cell type expressing the gene

Author(s)

Jacob Martin

Source

Simulated in data-raw/generate_example_data.R.

---

example_mtcars *Example mtcars dataset with factors*

Description

The classic mtcars dataset with key numeric columns converted to factors for categorical plotting examples.

Usage

```
example_mtcars
```

Format

A data frame with 32 rows and 11 columns:

mpg Miles per gallon

cyl Number of cylinders (factor)

disp Displacement (cubic inches)

hp Gross horsepower

drat Rear axle ratio

wt Weight (1000 lbs)

qsec 1/4 mile time

vs Engine (0 = V-shaped, 1 = straight) (factor)

am Transmission (0 = automatic, 1 = manual) (factor)

gear Number of forward gears (factor)

carb Number of carburetors (factor)

Author(s)

Jared Andrews

Source

Generated from the classic mtcars dataset.

`example_population` *Example population dataset A simulated population dataset with 400 rows covering 50 years and 8 age groups. Designed for line, area, and stacked bar plot examples.*

Description

Example population dataset A simulated population dataset with 400 rows covering 50 years and 8 age groups. Designed for line, area, and stacked bar plot examples.

Usage

```
example_population
```

Format

A data frame with 400 rows and 4 columns:

year Year of the population record (factor: 1975–2024)

age_group Age group category (factor: 0-9, 10-17, 18-34, 35-44, 45-54, 55-64, 65-74, 75+)

count Population count for the given year and age group

record_id Unique identifier for each population record

Author(s)

Jared Andrews

Source

Generated in data-raw/generate_example_data.R.

`example_rnaseq` *Example RNA-seq dataset for the RNA-seq showcase app*

Description

A simulated pseudo-bulk RNA-seq dataset with 288 rows covering six immune cell types, eight canonical marker genes, two conditions (Healthy / Disease), and three biological replicates per condition. Marker genes are strongly expressed in their canonical cell type; Disease replicates include a simulated ~1.2 log2FC upregulation for marker genes, making biological comparisons visually informative.

Usage

```
example_rnaseq
```

Format

A data frame with 288 rows and 7 columns:

cell_type Immune cell type (factor: CD4 T, CD8 T, B Cell, NK Cell, Monocyte, pDC)

gene Gene symbol (factor: CD3D, CD8A, MS4A1, NKG7, LYZ, LILRA4, CD14, GNLY)

condition Experimental condition (factor: Healthy, Disease)

replicate Biological replicate (factor: Rep1, Rep2, Rep3)

log2_cpm Simulated log2 counts-per-million expression value

avg_expression Mean log2_cpm across replicates for this cell_type $\times$ gene $\times$ condition

neg_log10_pval Simulated $-\log_{10}(p)$ value for differential expression summaries

Details

The dataset is designed to simultaneously support three VizModules plot types:

- DotPlot — summarised `avg_expression` and `pct_expressed` columns per cell type $\times$ gene $\times$ condition combination.
- yPlot — per-replicate `log2_cpm` values grouped by `cell_type` and coloured by `condition`.
- DensityPlot — per-replicate `log2_cpm` values grouped by `condition` and faceted by `cell_type`.

Author(s)

Jacob Martin

Source

Simulated in data-raw/generate_example_data.R.

example_sales *Example sales dataset*

Description

A simulated product-sales dataset (720 rows total). Designed to showcase bar, box, violin, area, line, scatter, split-bar, density, and histogram plot modules.

Usage

```
example_sales
```

Format

A data frame with 720 rows and 7 columns:

region Region of the sale (factor: North, South, East, West, Central, International)

revenue Revenue for month

year The year

month The month

units Units sold

sale_id Unique sale identifier

product_line Product line (factor: Gadgets, Widgets, Doohickeys)

Author(s)

Jared Andrews

Source

Generated in data-raw/generate_example_data.R.

example_school_earnings *Example school earnings dataset for dumbbell plots*

Description

A small dataset of median annual earnings for men and women at six universities, suitable for dumbbell plot examples.

Usage

```
example_school_earnings
```

Format

A data frame with 6 rows and 4 columns:

School University name

Women Median earnings for women (thousands of USD)

Men Median earnings for men (thousands of USD)

Group University type (STEM-heavy or Liberal Arts)

Author(s)

Jacob Martin

Source

Generated in data-raw/generate_example_data.R.

---

example_skills *Example multi-player skills dataset for radar plots*

Description

A dataset of skill ratings across five categories for three players, suitable for radar/spider chart examples.

Usage

```
example_skills
```

Format

A data frame with 15 rows and 3 columns:

category Skill category (Speed, Strength, Defense, Stamina, Agility)

value Skill rating (1-10)

player Player identifier (Player A, B, or C)

Author(s)

Jacob Martin

Source

Simulated in data-raw/generate_example_data.R.

figureBuilderApp *Create a VizModules Figure Builder Application*

Description

Build the multi-panel **Figure Builder** Shiny application as a returnable object. The app lets users add any VizModules plot module to a free-form A4 canvas, drag and resize each plot, filter each plot's data independently, label panels automatically, and export the whole figure as a single editable SVG (or bundle every plot's source data, HTML plot, and statistics into one `.zip`).

Usage

```
figureBuilderApp(
  data_list = NULL,
  module_registry = NULL,
  title = "VizModules Figure Builder",
  return_components = FALSE
)
```

Arguments

`data_list`	An optional named list of data frames that seed the dataset registry. If `NULL` (the default), the bundled example datasets (plus a `sales_by_product` summary suited to the pie plot) are used. At least one element is required and every element must be a data frame.
`module_registry`	An optional named list describing the plot modules to offer. If `NULL` (the default), all bundled VizModules modules are offered. Each entry is itself a list with components: `label` (character, shown in the picker), `dataset` (character, the dataset name its `defaults` were written for), `inputs_ui`, `output_ui`, and `server_fn` (the module's three functions), and `defaults` (a named list of input defaults applied only when `dataset` is the chosen dataset).
`title`	A character string used as the page title and header (default: `"VizModules Figure Builder"`).
`return_components`	Logical. When `FALSE` (the default) a `shiny::shinyApp()` object is returned. When `TRUE` a named list with `ui` and `server` elements is returned instead, which is convenient for deployment scripts that need an explicit `shinyApp(ui, server)` call.

Details

Datasets are supplied via `data_list` and seed the "Add Plot" dialog; users can also upload additional datasets (CSV, TSV, TXT, or RDS) at runtime. The set of available plot modules is controlled by `module_registry`, so the app can be extended with custom wrapper modules without editing the package.

This is the recommended way to launch a standalone Figure Builder. Internally it is a thin wrapper around the `figureBuilderUI()` / `figureBuilderServer()` Shiny module, so the same builder can be embedded inside a larger app (and instantiated more than once) by calling those two functions directly. The bundled app at `system.file("apps/figure-builder", package = "VizModules")` is itself a thin wrapper around this function.

Value

Either a `shiny::shinyApp()` object, or (when `return_components = TRUE`) a list with elements `ui` and `server`.

Author(s)

Jared Andrews

See Also

`figureBuilderUI()`, `figureBuilderServer()`

Examples

```
library(VizModules)

# Launch with the bundled example datasets and all modules:
app <- figureBuilderApp()
if (interactive()) runApp(app)

# Launch with your own datasets:
app2 <- figureBuilderApp(data_list = list("iris" = iris, "mtcars" = mtcars))
if (interactive()) runApp(app2)

# Return the UI and server separately (e.g. for a deployment app.R):
parts <- figureBuilderApp(return_components = TRUE)
if (interactive()) shinyApp(parts$ui, parts$server)
```

---

figureBuilderServer *Server logic for the Figure Builder module*

Description

Powers the multi-panel **Figure Builder** module rendered by `figureBuilderUI()`. Users can add any VizModules plot module to a free-form A4 canvas, drag and resize each plot, filter each plot's data independently, label panels automatically, and export the whole figure as a single editable SVG (or bundle every plot's source data, HTML plot, and statistics into one `.zip`).

Usage

```
figureBuilderServer(id, data_list = NULL, module_registry = NULL)
```

Arguments

id	The ID for the Shiny module. Must match the id given to figureBuilderUI().
data_list	An optional named list of data frames that seed the dataset registry. If NULL (the default), the bundled example datasets (plus a sales_by_product summary suited to the pie plot) are used. At least one element is required and every element must be a data frame.
module_registry	An optional named list describing the plot modules to offer. If NULL (the default), all bundled VizModules modules are offered. Each entry is itself a list with components: label (character, shown in the picker), dataset (character, the dataset name its defaults were written for), inputs_ui, output_ui, and server_fn (the module's three functions), and defaults (a named list of input defaults applied only when dataset is the chosen dataset).

Details

Call this from your app's server with the same id you passed to figureBuilderUI(). Because it is a proper Shiny module, several Figure Builders can coexist on one page, each with its own namespace and canvas.

Value

Invisibly returns NULL; called for its side effects (wiring up the Figure Builder module's reactive logic).

Author(s)

Jared Andrews

See Also

figureBuilderUI(), figureBuilderApp()

Examples

```
library(VizModules)
if (interactive()) {
    ui <- fluidPage(figureBuilderUI("figure_builder"))
    server <- function(input, output, session) {
        figureBuilderServer("figure_builder")
    }
    shinyApp(ui, server)
}
```

figureBuilderUI *UI component for the Figure Builder module*

Description

Renders the full multi-panel **Figure Builder** interface (sidebar controls plus the free-form A4 canvas) as a namespaced Shiny module. Place this in the UI where you want the builder to appear, using an `id` that matches the one passed to `figureBuilderServer()`.

Usage

```
figureBuilderUI(id, title = "VizModules Figure Builder")
```

Arguments

`id`	The ID for the Shiny module. Must match the `id` given to `figureBuilderServer()`.
`title`	A character string used as the header shown above the builder (default: `"VizModules Figure Builder"`). Pass `NULL` to omit the header, which is useful when the host page supplies its own title.

Details

Unlike `figureBuilderApp()` (which returns a complete, standalone app), this function returns a `tagList` you can drop into any page, so the builder can be embedded alongside other content and instantiated more than once (each instance keeps its own namespace, canvas, and downloads).

The returned UI bundles the JavaScript and CSS the canvas needs, and calls `shinyjs::useShinyjs()`, so no extra setup is required in the host app.

Value

A Shiny `tagList` containing the Figure Builder UI.

Author(s)

Jared Andrews

See Also

`figureBuilderServer()`, `figureBuilderApp()`

Examples

```
library(VizModules)
figureBuilderUI("figure_builder")
```

`finalize_manual_edits` *Render persistent manual plot layout edits across re-renders*

Description

Render-step companion to `setup_manual_edits()`. Call this inside your module's `plotly::renderPlotly()` on the freshly rebuilt figure, immediately before returning it.

Usage

```
finalize_manual_edits(fig, plot_source, store, session)
```

Arguments

`fig`	A plotly figure object, typically the result of your plotting pipeline. If `NULL`, it is returned unchanged.
`plot_source`	Character scalar. The **same** plotly event source id passed to `setup_manual_edits()`; assigned to `fig$x$source`.
`store`	The list returned by `setup_manual_edits()`.
`session`	The module's `session` object, used to namespace the colorbar drag input.

Details

It performs four jobs:

1. tags the figure with the module's plotly event `source` so its `plotly_relayout` events are captured by `setup_manual_edits()`;
2. restores any manually repositioned legend, annotations, axis titles, and colorbar captured so far;
3. records the figure so future relayout events can be matched to stable annotation keys (surviving re-ordering on rebuild); and
4. attaches the JavaScript listener that forwards colorbar drags.

Restored edits are applied under `shiny::isolate()` so re-applying them never triggers an additional re-render.

Value

The finalized plotly figure, ready to be returned from `plotly::renderPlotly()`.

Author(s)

Jared Andrews

See Also

`setup_manual_edits()` for the setup-step companion.

Examples

```
## Not run:
# Inside renderPlotly(), after building `fig`:
fig <- finalize_manual_edits(fig, plot_source, edit_store, session)
fig

## End(Not run)
```

`generate_pair_strings` *Generate comparison pair strings from data columns*

Description

Creates formatted pair strings for populating the comparison selector UI. Handles both standard x-axis comparisons and nested group.by comparisons.

Usage

```
generate_pair_strings(df, x, group.by = NULL)
```

Arguments

`df`	Data frame containing the data.
`x`	Character; x-axis column name.
`group.by`	Character or NULL; nested grouping column.

Value

A character vector of pair strings in "group1 vs group2" format.

Author(s)

Jared Andrews

Examples

```
generate_pair_strings(example_iris, x = "Species")
```

get_default *Resolve a default value from a named list*

Description

Looks up `key` in `defaults`. If present and passes `validator` (when supplied), returns the stored value; otherwise returns `fallback`. Uses standard `if`/`else` instead of vectorized `ifelse()` to avoid silent truncation of multi-valued defaults.

Usage

```
get_default(defaults, key, fallback, validator = NULL)
```

Arguments

`defaults`	A named list of default values, or NULL.
`key`	Character string — the name to look up.
`fallback`	The value to return when `key` is absent or fails validation.
`validator`	An optional single-argument predicate function (e.g., `is.numeric`, `is.logical`). When supplied, the stored value is returned only if `validator(value)` is `TRUE`.

Value

The resolved default value or `fallback`.

Author(s)

Jared Andrews

Examples

```
get_default(list(color = "red"), "color", "black")
get_default(list(), "missing", 10)
get_default(list(n = "x"), "n", 5, is.numeric)
```

get_documentation *Extract parameter documentation from an R function help page*

Description

Parses the Rd documentation for a given function and extracts parameter descriptions for specified parameter names.

Usage

```
get_documentation(package_name, type = "param", selected = NULL, cap = FALSE)
```

Arguments

package_name	A string in the format "package::function" indicating which function's documentation to parse.
type	The type of documentation section to extract. Currently only "param" is supported.
selected	A list of parameter names to extract. Note that co-documented parameters (e.g., x.by and y.by) should be grouped together in a vector within the list or an error will be thrown by extract_roc_text.
cap	Logical; if TRUE, capitalize the first letter of each description.

Value

A named list where names are parameter names and values are their documentation strings. Returns empty strings for parameters not found in the documentation.

Author(s)

Jacob Martin, Jared Andrews

---

get_model_backend *Get a registered model backend*

Description

Retrieve a model backend from the registry by name.

Usage

```
get_model_backend(name)
```

Arguments

name	Character string. The backend name (e.g. "lm", "drm").

Value

The backend list (with fit, predict, validate_classes), or NULL if no backend with that name is registered.

Author(s)

Jacob Martin

Examples

```
get_model_backend("lm")
get_model_backend("nonexistent")
```

`is_pure_type` *Check if column inputs contain mixed data types*

Description

This function validates that a vector of column names from a data frame contains columns of only one data type category: either all numeric OR all categorical (factor/character). Returns `FALSE` for mixed numeric + categorical columns. Single columns always return `TRUE`. Used for Shiny plotting module input validation.

Usage

```
is_pure_type(inputs, d)
```

Arguments

`inputs`	Character vector of column names to validate.
`d`	Data frame containing the columns specified in `inputs`.

Value

Logical scalar: `TRUE` if all numeric OR all categorical (factor/character); `FALSE` if mixed numeric + categorical/factor detected.

Author(s)

Jacob Martin

See Also

`base::for()`

Examples

```
df <- data.frame(num1 = 1:3, num2 = 4:6, cat1 = letters[1:3], fac1 = factor(1:3))
is_pure_type(c("num1", "num2"), df) # TRUE (all numeric)
is_pure_type(c("cat1", "fac1"), df) # TRUE (all categorical)
is_pure_type(c("num1"), df) # TRUE (single)
is_pure_type(c("num1", "cat1"), df) # FALSE (mixed numeric + cat)
```

linePlot *Create an Interactive Line Plot with plotly*

Description

Generates a customizable interactive line plot using plotly, supporting grouping, faceting, axis adjustments, and color palettes.

Usage

```
linePlot(
  data,
  x,
  y,
  palette.selection,
  plot.mode = "lines",
  line.type = "solid",
  colour.group.by = NULL,
  show.legend = TRUE,
  facet.by = NULL,
  facet.scales = "fixed",
  facet.nrow = NULL,
  facet.ncol = NULL,
  subplot.margin = 0.05,
  axis.showline = TRUE,
  axis.mirror = TRUE,
  axis.linecolor = "black",
  axis.linewidth = 0.5,
  axis.tickfont.size = 12,
  axis.tickfont.color = "black",
  axis.tickfont.family = "Arial",
  axis.tickangle.x = 0,
  axis.tickangle.y = 0,
  axis.ticks = "outside",
  axis.tickcolor = "black",
  axis.ticklen = 5,
  axis.tickwidth = 1,
  show.grid.x = TRUE,
  show.grid.y = TRUE,
  title.text = "",
  title.font.size = 14,
  title.font.family = "Arial",
  title.font.color = "black",
  title.x.position = 0.47,
  y.title = NULL,
  x.title = NULL,
  flip.x = FALSE,
```

```
  flip.y = FALSE,
  x.adjustment = NULL,
  y.adjustment = NULL,
  color.adjustment = NULL,
  order.by = NULL,
  error.colour = NULL,
  error.width = NULL,
  error.bar = FALSE
)
```

Arguments

data	A data.frame or tibble containing the data to plot.
x	Character vector of column name(s) for the x-axis. Multiple columns create separate traces.
y	Character vector of column name(s) for the y-axis. Multiple columns create separate traces.
palette.selection	Character vector of hex colors for line colors. Used to assign colors to groups or traces.
plot.mode	Character, plotly mode for plot type. Options: "lines", "markers", "lines+markers". Default: "lines".
line.type	Character, line style. Options: "solid", "dot", "dash", "longdash", "dashdot", "longdashdot". Default: "solid".
colour.group.by	Character or formula, column name(s) to group lines by color. Can be a formula like ~ column_name. Ignored if multiple x or y columns are provided. Default: NULL.
show.legend	Logical, whether to display the legend. Default: TRUE.
facet.by	Optional character, column name to facet plots by. Creates subplots for each unique value. Default: NULL.
facet.scales	Character, controls axis scaling across facets. Options: "fixed" (same for all), "free" (independent), "free_x" (independent x-axis), "free_y" (independent y-axis). Default: "fixed".
facet.nrow	Optional integer, number of rows in the faceted subplot grid. If NULL (default), a single row is used unless facet.ncol is supplied, in which case the number of rows is derived from the number of facet levels.
facet.ncol	Optional integer, number of columns in the faceted subplot grid. If NULL (default), columns are derived from facet.nrow and the number of facet levels. Only one of facet.nrow / facet.ncol needs to be set; if both are provided, facet.nrow takes precedence.
subplot.margin	Numeric, spacing between facet panels as a fraction of the plot area. May be a single value (applied to both directions) or a length-2 vector c(horizontal, vertical) to control the gap between columns and rows separately. Default: 0.05.

`axis.showline` Logical, whether to show axis border lines. Default: TRUE.

`axis.mirror` Logical, whether to mirror axis lines on opposite side of plot. Default: TRUE.

`axis.linecolor` Character, hex color for axis lines. Default: "black".

`axis.linewidth` Numeric, width of axis lines in pixels. Default: 0.5.

`axis.tickfont.size`
Numeric, font size for axis tick labels. Default: 12.

`axis.tickfont.color`
Character, hex color for axis tick labels. Default: "black".

`axis.tickfont.family`
Character, font family for axis tick labels. Default: "Arial".

`axis.tickangle.x`
Numeric, rotation angle for x-axis tick labels in degrees. Default: 0.

`axis.tickangle.y`
Numeric, rotation angle for y-axis tick labels in degrees. Default: 0.

`axis.ticks` Character, position of tick marks. Options: "outside", "inside", "none". Default: "outside".

`axis.tickcolor` Character, hex color for tick marks. Default: "black".

`axis.ticklen` Numeric, length of tick marks in pixels. Default: 5.

`axis.tickwidth` Numeric, width of tick marks in pixels. Default: 1.

`show.grid.x` Logical, whether to show gridlines on the x-axis. Default: TRUE.

`show.grid.y` Logical, whether to show gridlines on the y-axis. Default: TRUE.

`title.text` Character, main title text for the plot. Default: "".

`title.font.size`
Numeric, font size for plot title. Default: 14.

`title.font.family`
Character, font family for plot title. Default: "Arial".

`title.font.color`
Character, hex color for plot title text. Default: "black".

`title.x.position`
Numeric, horizontal position of the plot title in paper coordinates (0 = left, 1 = right). Default: 0.47.

`y.title` Optional character, label for y-axis. If NULL, auto-generated from column name. When `x` is a single categorical column, the plotted y-values are per-group means, so the title is wrapped as `mean(<y.title>)` to accurately describe the summary displayed. Default: NULL.

`x.title` Optional character, label for x-axis. If NULL, auto-generated from column name. Default: NULL.

`flip.x` Logical, whether to reverse the x-axis direction. Default: FALSE.

`flip.y` Logical, whether to reverse the y-axis direction. Default: FALSE.

`x.adjustment` Optional character or function, transformation to apply to x values. Options: "log2", "log", "log10", "neg_log10", "log1p", "as.factor", "abs", "sqrt", or custom function. Default: NULL.

y.adjustment	Optional character or function, transformation to apply to y values. Options: "log2", "log", "log10", "neg_log10", "log1p", "as.factor", "abs", "sqrt", or custom function. Default: NULL.
color.adjustment	Optional character or function, transformation to apply to color grouping variable. Same options as x.adjustment and y.adjustment. Default: NULL.
order.by	Optional character vector, column name(s) to order data by before plotting. Default: NULL.
error.colour	hex colour input to set the colour of the error bars on a plot with a categorical X axis and only 1 Y axis variable
error.width	numeric input to set the width of the error bars on a plot with a categorical X axis and only 1 Y axis variable
error.bar	Boolean value to determine if error bars will be on or off on a plot with a categorical X axis and only 1 Y axis variable

Value

A plotly object representing the interactive line plot.

Author(s)

Jacob Martin, Jared Andrews

Examples

```
palette <- plotthis::palette_list[["Set2"]]
fig <- linePlot(
    data = mtcars,
    x = "cyl",
    y = "mpg",
    plot.mode = "lines",
    line.type = "solid",
    colour.group.by = "mpg",
    palette.selection = palette,
    show.legend = TRUE
)
```

---

linePlotApp *Create an example Modular linePlot Shiny Application*

Description

This function generates a Shiny application with modular `linePlot()` components. The app features a **Data Import** section for uploading data, a **Data Table** for filtering the active dataset, and a **Plot** area for configuring and displaying an interactive line plot.

Usage

```
linePlotApp(
  data_list = NULL,
  defaults = NULL,
  hide.inputs = NULL,
  hide.tabs = NULL
)
```

Arguments

data_list	An optional named list of data frames. If NULL (the default), list("sales" = gallery_sales) is used as example data.
defaults	A named list of input IDs and their default values to apply on startup.
hide.inputs	A character vector of input IDs to hide. Their values are still initialized and used, but the controls are not shown in the UI.
hide.tabs	A character vector of tab names to hide. Inputs in these tabs are still initialized and used, but the controls are not shown in the UI.

Details

When data_list is not provided (or NULL), the app launches with gallery_sales as an example dataset. Uploaded data files are added to the available datasets and can be selected for plotting. If an uploaded file shares a name with an existing dataset, the existing one is overwritten with a warning.

This is a convenience wrapper around createModuleApp().

Value

A Shiny app object.

Author(s)

Jacob Martin, Jared Andrews

See Also

linePlot(), linePlotInputsUI(), linePlotOutputUI(), linePlotServer()

Examples

```
library(VizModules)
# Launch with default example data (example_sales):
app <- linePlotApp()
if (interactive()) runApp(app)

# Launch with custom data:
app2 <- linePlotApp(list("sales" = example_sales))
if (interactive()) runApp(app2)
```

`linePlotInputsUI` *Input UI components for the linePlot module*

Description

This should be placed in the UI where the inputs should be shown, with an `id` that matches the `id` used in the `linePlotServer()` and `linePlotOutputUI()` functions.

Usage

```
linePlotInputsUI(id, data, defaults = NULL, title = NULL, columns = 2)
```

Arguments

`id`	The ID for the Shiny module.
`data`	The data frame used for plot generation.
`defaults`	A named list of default values for the inputs.
`title`	An optional title for the UI grid.
`columns`	Number of columns for the UI grid.

Details

The user inputs for this module are separated from the outputs to allow for more flexible UI design.

The inputs will automatically be organized into a grid layout via the `organize_inputs()` function, with `columns` controlling the number of columns in the grid.

Defaults can be set for each input by providing a named list of values to the `defaults` argument. Nearly all parameters for `linePlot()` can be set via these inputs, so see the help for that function for an exhaustive list.

Value

A Shiny tagList containing the UI elements

Plot parameters and defaults

The following `linePlot()` parameters can be accessed via UI inputs and/or the `defaults` argument:

- `x` - X-axis variable(s) (UI: "Select X values", defaults key: `x.value`, default: 1st column, multiple: TRUE)
- `y` - Y-axis variable(s) (UI: "Select Y values", defaults key: `y.value`, default: 2nd column, multiple: TRUE)
- `title.font.size` - Plot title font size (UI: "Title Size", default: 26)
- `title.font.family` - Font family for title text (UI: "Title Font", default: "Arial")
- `title.font.color` - Color for plot title (UI: "Title Color", default: "#000000")

- `axis.title.font.size` - Axis title font size (UI: "Axis Title Size", default: 18)
- `axis.title.font.color` - Axis title font color (UI: "Axis Title Color", default: "#000000")
- `axis.title.font.family` - Axis title font family (UI: "Axis Title Font", default: "Arial")
- `group.by` - Grouping variable (UI: "Group by", default: 1st categorical variable)
- `order.by` - Order by Y values (UI: "Order by Y", default: FALSE)
- `x.adjustment` - X-axis adjustment function (UI: "X Adjustment", default: "")
- `y.adjustment` - Y-axis adjustment function (UI: "Y Adjustment", default: "")
- `facet.by` - Faceting variable (UI: "Facet by", default: "")
- `facet.scales` - Facet scale behavior (UI: "Facet scales", default: "fixed")
- `facet.nrow` - Number of rows in the facet grid (UI: "Rows", default: NULL; blank = auto)
- `facet.ncol` - Number of columns in the facet grid (UI: "Columns", default: NULL; blank = auto)
- `plot.mode` - Plot type (UI: "Plot type", default: "lines")
- `line.type` - Line type (UI: "Line type", default: "solid")
- `error.bar` - Show error bars (UI: "Error Bars", default: TRUE; requires a categorical X and a single Y)
- `error.colour` - Error bar color (UI: "Error Bar Colour", default: "#000000")
- `error.width` - Error bar cap width (UI: "Error Bar Width", default: 1)
- `palette.selection` - Color palette (UI: palette picker, derived from palette)
- `axis.showline` - Show axis border lines (UI: "Show Axis Borders", default: TRUE)
- `axis.mirror` - Mirror axis lines on opposite side (UI: "Mirror Axis Borders", default: TRUE)
- `axis.linecolor` - Color of axis lines (UI: "Axis Line Color", default: "black")
- `axis.linewidth` - Width of axis lines (UI: "Axis Line Width", default: 0.5)
- `axis.tickfont.size` - Size of tick labels (UI: "Tick Label Size", default: 12)
- `axis.tickfont.color` - Color of tick labels (UI: "Tick Label Color", default: "black")
- `axis.tickfont.family` - Font family for tick labels (UI: "Tick Label Font", default: "Arial")
- `axis.tickangle.x` - Rotation angle for X-axis tick labels (UI: "X Tick Label Angle", default: 0)
- `axis.tickangle.y` - Rotation angle for Y-axis tick labels (UI: "Y Tick Label Angle", default: 0)
- `axis.ticks` - Position of tick marks (UI: "Tick Position", default: "outside")
- `axis.tickcolor` - Color of tick marks (UI: "Tick Mark Color", default: "black")
- `axis.ticklen` - Length of tick marks (UI: "Tick Mark Length", default: 5)
- `axis.tickwidth` - Width of tick marks (UI: "Tick Mark Width", default: 1)
- `facet.title.font.size` - Facet subplot title font size (UI: "Facet Subplot Title Size", default: 18)
- `facet.title.font.color` - Facet subplot title font color (UI: "Facet Title Color", default: "#000000")

- `facet.title.font.family` - Facet subplot title font family (UI: "Facet Title Font", default: "Arial")
- `show.grid.x` - Show X-axis gridlines (UI: "Show X Gridlines", default: TRUE)
- `show.grid.y` - Show Y-axis gridlines (UI: "Show Y Gridlines", default: TRUE)
- `grid.color` - Gridline color (UI: "Gridline Color", default: "#CCCCCC")
- `x.title` - X-axis title (auto-calculated from data)
- `y.title` - Y-axis title (auto-calculated from data)
- `flip.x` - Flip X-axis (UI: "Flip X", default: FALSE)
- `flip.y` - Flip Y-axis (UI: "Flip Y", default: FALSE)

Parameters controlling additional functionality

The following parameters implementing plotly-specific features are also available:

- `hline.intercepts` - Y-coordinates for horizontal reference lines (UI: "Y-intercepts", default: "")
- `hline.colors` - Colors for horizontal lines (UI: "Colors", default: "#000000")
- `hline.widths` - Widths for horizontal lines (UI: "Widths", default: "1")
- `hline.linetypes` - Line types for horizontal lines (UI: "Line Types", default: "dashed")
- `hline.opacities` - Opacities for horizontal lines (UI: "Opacities (0-1)", default: "1")
- `vline.intercepts` - X-coordinates for vertical reference lines (UI: "X-intercepts", default: "")
- `vline.colors` - Colors for vertical lines (UI: "Colors", default: "#000000")
- `vline.widths` - Widths for vertical lines (UI: "Widths", default: "1")
- `vline.linetypes` - Line types for vertical lines (UI: "Line Types", default: "dashed")
- `vline.opacities` - Opacities for vertical lines (UI: "Opacities (0-1)", default: "1")
- `abline.slopes` - Slopes for diagonal reference lines (UI: "Slopes", default: "")
- `abline.intercepts` - Y-intercepts for diagonal lines (UI: "Y-intercepts", default: "")
- `abline.colors` - Colors for diagonal lines (UI: "Colors", default: "#000000")
- `abline.widths` - Widths for diagonal lines (UI: "Widths", default: "1")
- `abline.linetypes` - Line types for diagonal lines (UI: "Line Types", default: "dashed")
- `abline.opacities` - Opacities for diagonal lines (UI: "Opacities (0-1)", default: "1")

Author(s)

Jacob Martin, Jared Andrews

See Also

`linePlot()`, `organize_inputs()`, `linePlotOutputUI()`, `linePlotServer()`, `linePlotApp()`

Examples

```
library(VizModules)
data(mtcars)
linePlotInputsUI("linePlot", mtcars)
```

linePlotOutputUI *Output UI components for the linePlot module*

Description

This should be placed in the UI where the plot should be shown.

Usage

```
linePlotOutputUI(id, resizable = TRUE)
```

Arguments

id	The ID for the Shiny module.
resizable	Logical; when TRUE (the default) the plot output is wrapped in shinyjqui::jqui_resizable() so it can be resized by dragging. Set to FALSE when embedding the output in a container that already provides resizing.

Value

A Shiny plotlyOutput for the linePlot

Author(s)

Jacob Martin

linePlotServer *Server logic for linePlot module*

Description

Server logic for linePlot module

Usage

```
linePlotServer(id, data, hide.inputs = NULL, hide.tabs = NULL, defaults = NULL)
```

Arguments

id	The ID for the Shiny module.
data	A reactive containing the data frame to plot.
hide.inputs	A character vector of input IDs to hide. These will still be initialized and their values passed to the plot function, but the user will not be able to see/adjust them in the UI.

hide.tabs	A character vector of tab names to hide. Inputs in these tabs will still be initialized and their values passed to the plot function, but the user will not be able to see/adjust them in the UI.
defaults	A named list of default values for the inputs. When the reset button is clicked, inputs are reset to these values rather than hardcoded fallbacks. Typically the same list passed to the corresponding UI function.

Value

The moduleServer function for the linePlot module.

Author(s)

Jacob Martin

See Also

linePlot(), linePlotInputsUI(), linePlotOutputUI(), linePlotApp()

---

linetype_to_dash *Convert linetype name to plotly dash style*

Description

Maps common linetype names to plotly dash specifications.

Usage

```
linetype_to_dash(linetype)
```

Arguments

linetype	Character. Linetype name: "solid", "dashed", "dotted", "dotdash", "longdash", "twodash".

Value

Character. Plotly-compatible dash specification.

Author(s)

Jared Andrews

Examples

```
linetype_to_dash("dashed")
linetype_to_dash("dotted")
```

list_model_backends *List registered model backends*

Description

Returns the names of all currently registered model backends. The built-in backends (`lm`, `glm`, `loess`, `nls`) are always present; any backends added via `register_model_backend()` are included as well.

Usage

```
list_model_backends()
```

Value

A sorted character vector of backend names.

Author(s)

Jacob Martin

Examples

```
list_model_backends()
```

module_tack_ui *Create standard tack UI for module inputs*

Description

Generates a consistent set of control buttons for VizModules that includes Auto Update toggle, Update and Reset buttons, and a full source download button (self-contained HTML of the plot, source data, and statistics).

Usage

```
module_tack_ui(ns, defaults = NULL)
```

Arguments

`ns`	Namespace function from the module (e.g., `ns <- NS(id)`).
`defaults`	Optional named list of default values. Reserved for future use.

Value

A Shiny tagList containing the standard control buttons and inputs.

Author(s)

Jared Andrews

Examples

```
library(VizModules)
library(shiny)
ns <- NS("myModule")
module_tack_ui(ns)
```

`multiColorPicker` *Compact multi-group color picker input*

Description

Build a compact Shiny input that assigns colors to a set of groups using a palette or manual hex pickers. The value returned to `input[[inputId]]` is a named character vector of hex colors keyed by group.

Usage

```
multiColorPicker(
  inputId,
  label = NULL,
  groups,
  palette_options = NULL,
  selected_palette = NULL,
  colors = NULL,
  width = NULL,
  show_text = TRUE,
  compact = FALSE,
  panel = TRUE
)
```

Arguments

`inputId`	Character. Shiny input id.
`label`	Optional label displayed above the control.
`groups`	Character or factor vector of group names.
`palette_options`	Named list of palettes (each a character vector of colors). Defaults to the palettes from `default_palettes()`.
`selected_palette`	Optional name of the palette to preselect.
`colors`	Optional named vector of starting colors. Values are matched to `groups` by name when provided.

width	Optional CSS width for the container.
show_text	Logical. If TRUE, show editable hex text inputs beside the color pickers.
compact	Logical. If TRUE, renders a tighter layout with reduced spacing, smaller controls, and narrower palette selector.
panel	Logical. If FALSE, removes the surrounding panel/well styling (border, padding, background).

Value

A UI element that produces a named character vector of colors.

Author(s)

Jared Andrews

Examples

```
if (interactive()) {
    library(shiny)
    groups <- c("setosa", "virginica", "versicolor")

    ui <- fluidPage(
        multiColorPicker(
            "species_cols",
            "Species colors",
            groups = groups,
            selected_palette = "dittoColors"
        ),
        verbatimTextOutput("chosen")
    )

    server <- function(input, output, session) {
        output$chosen <- renderPrint(input$species_cols)
    }

    shinyApp(ui, server)
}
```

---

multiDynamicInput *Dynamic multi-row input for repeating groups of inputs*

Description

A custom Shiny input that lets users dynamically add and remove **rows**, where each row is a group of heterogeneous inputs (e.g. a select, a text box, a colour picker, and a numeric input on one line). A + Add button appends a row; an X button on each row deletes it. Rows are added and removed entirely on the client by cloning a hidden template and letting Shiny bind the new inputs, mirroring the architecture of multiColorPicker().

Usage

```
multiDynamicInput(
  inputId,
  label = NULL,
  row_spec,
  elements = NULL,
  max_per_row = 4,
  add_label = "+ Add",
  width = NULL,
  panel = TRUE
)
```

Arguments

inputId	Character. Shiny input id.
label	Optional label displayed next to the add button.
row_spec	A named list describing one row. Each element is itself a list specifying a single field, with either: • type — a string alias: one of "select", "text", "numeric", "slider", "checkbox", "colour"/"color"; or • fn — an input constructor function (e.g. shiny::dateInput). plus an optional args list of arguments passed to the constructor (everything except inputId, which is generated per row). The element name becomes the key under which that field's value is returned.
elements	Optional initial rows to display on startup. A named list of rows (each a named list keyed by the row_spec names) in the same shape as the return value. For example: list(models1 = list(model_type = "lm", formula = "y ~ x", line_colour = "#FF0000", line_width = 2)).
max_per_row	Integer. Maximum number of fields on one visual line before wrapping. Default 4.
add_label	Character. Label for the add button. Default "+ Add".
width	Optional CSS width for the container.
panel	Logical. If FALSE, removes the surrounding panel/well styling.

Details

The value reported to input[[inputId]] is a **named list of rows** (model1, model2, ...), each a named list keyed by the row_spec names.

The component is generic over input type. Each field in row_spec is described by an input constructor and its arguments, so any Shiny input that takes inputId as its first argument can be used. Common types have a short type = "..." alias.

Value

A UI element that produces a named list of rows.

Author(s)

Jacob Martin

Examples

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

    ui <- fluidPage(
        multiDynamicInput(
            "models",
            label = "Models",
            row_spec = list(
                model_type  = list(type = "select",
                    args = list(choices = c("lm", "glm", "loess", "nls"))),
                formula     = list(type = "text",
                    args = list(placeholder = "y ~ poly(x, 2)")),
                line_colour = list(type = "colour", args = list(value = "#000000"))
            )
        ),
        verbatimTextOutput("chosen")
    )

    server <- function(input, output, session) {
        output$chosen <- renderPrint(input$models)
    }

    shinyApp(ui, server)
}
```

---

organize_inputs *Organize arbitrary Shiny inputs into a grid layout*

Description

Organize arbitrary Shiny inputs into a grid layout

Usage

```
organize_inputs(
  tag.list,
  id = NULL,
  title = NULL,
  tack = NULL,
  columns = NULL,
  rows = NULL
)
```

Arguments

tag.list	A tagList containing UI inputs or a named list containing multiple tagLists containing UI inputs.
id	An optional ID for the tabsetPanel if a named list is provided.
title	An optional title for the grid, should be a UI element, e.g. h3("Title").
tack	An optional UI input to tack onto the end of the grid.
columns	Number of columns.
rows	Number of rows.

Value

A Shiny tagList with inputs organized into a grid, optionally nested inside a tabsetPanel.

Author(s)

Jared Andrews

Examples

```
library(VizModules)
# Example 1: Basic usage with a simple grid
ui.inputs <- tagList(
    textInput("name", "Name"),
    numericInput("age", "Age", value = 30),
    selectInput("gender", "Gender", choices = c("Male", "Female", "Other"))
)
organize_inputs(ui.inputs, columns = 2, rows = 2)

# Example 2: Using a named list to create tabs
ui.inputs.tabs <- list(
    Personal = tagList(
        textInput("firstname", "First Name"),
        textInput("lastname", "Last Name")
    ),
    Settings = tagList(
        checkboxInput("newsletter", "Subscribe to newsletter", value = TRUE),
        sliderInput("volume", "Volume", min = 0, max = 100, value = 50)
    )
)
organize_inputs(ui.inputs.tabs, columns = 1)

# Example 3: Adding an additional UI element with 'tack'
additional.ui <- actionButton("submit", "Submit")
organize_inputs(ui.inputs, tack = additional.ui, columns = 3)

# Example 4: Handling a case with more inputs than grid cells
many.inputs <- tagList(replicate(10, textInput("input", "Input")))
organize_inputs(many.inputs, columns = 3) # Creates more than one row
```

`parallelCoordinatesPlot` *Create an Interactive Parallel Coordinates Plot with plotly*

Description

Generates a customizable interactive parallel coordinates plot using plotly, supporting dimension selection, color mapping, and font styling.

Usage

```
parallelCoordinatesPlot(
  data,
  dimensions,
  color.by = NULL,
  color.scale = "Viridis",
  palette.selection = NULL,
  line.opacity = 0.5,
  line.width = 1,
  show.colorbar = TRUE,
  label.font.size = 12,
  label.font.color = "black",
  label.font.family = "Arial",
  tick.font.size = 10,
  tick.font.color = "black",
  tick.font.family = "Arial",
  title.text = "",
  title.font.size = 16,
  title.font.family = "Arial",
  title.font.color = "black",
  bgcolor = "#FFFFFF"
)
```

Arguments

`data`	A data.frame or tibble containing the data to plot.
`dimensions`	Character vector of column names to use as dimensions (axes). Must contain at least two columns. Non-numeric columns are mapped to integers.
`color.by`	Optional character, column name to color lines by. Numeric columns use a continuous colorscale (`color.scale`); categorical columns use a discrete palette (`palette.selection`) and are displayed with category names on the colorbar. Default: NULL.
`color.scale`	Character, plotly colorscale name for line coloring when `color.by` is numeric. Options include "Viridis", "Cividis", "Inferno", "Magma", "Plasma", "Blues", "Greens", "Reds", "Oranges", "RdBu", "RdYlBu", "Spectral", "Jet", "Hot", "Cool", "Portland". Default: "Viridis".

`palette.selection`
: Character vector of hex colors used to color lines when `color.by` is categorical. May be unnamed (colors applied in order to the sorted unique levels) or named by level. If `NULL`, the plotly `color.scale` is used as a fallback. Default: NULL.

`line.opacity`
: Numeric, opacity of lines between 0 and 1. Default: 0.5.

`line.width`
: Numeric, width of lines in pixels. Default: 1.

`show.colorbar`
: Logical, whether to show the colorbar. Default: TRUE.

`label.font.size`
: Numeric, font size for dimension labels. Default: 12.

`label.font.color`
: Character, hex color for dimension labels. Default: "black".

`label.font.family`
: Character, font family for dimension labels. Default: "Arial".

`tick.font.size`
: Numeric, font size for axis tick labels. Default: 10.

`tick.font.color`
: Character, hex color for axis tick labels. Default: "black".

`tick.font.family`
: Character, font family for axis tick labels. Default: "Arial".

`title.text`
: Character, main title text for the plot. Default: "".

`title.font.size`
: Numeric, font size for plot title. Default: 16.

`title.font.family`
: Character, font family for plot title. Default: "Arial".

`title.font.color`
: Character, hex color for plot title text. Default: "black".

`bgcolor`
: Character, hex color for the plot background. Default: "#FFFFFF".

Value

A plotly object representing the interactive parallel coordinates plot.

Author(s)

Jacob Martin, Jared Andrews

Examples

```
fig <- parallelCoordinatesPlot(
    data = mtcars,
    dimensions = c("mpg", "cyl", "disp", "hp", "wt"),
    color.by = "mpg",
    color.scale = "Viridis",
    line.opacity = 0.6
)
```

`parallelCoordinatesPlotApp` *Create an example Modular parallelCoordinatesPlot Shiny Application*

Description

This function generates a Shiny application with modular `parallelCoordinatesPlot()` components. The app features a **Data Import** section for uploading data, a **Data Table** for filtering the active dataset, and a **Plot** area for configuring and displaying an interactive parallel coordinates plot.

Usage

```
parallelCoordinatesPlotApp(
  data_list = NULL,
  defaults = NULL,
  hide.inputs = NULL,
  hide.tabs = NULL
)
```

Arguments

`data_list`	An optional named list of data frames. If `NULL` (the default), `list("sales" = example_sales)` is used as example data. Each data frame should contain at least two numeric or categorical columns.
`defaults`	A named list of input IDs and their default values to apply on startup.
`hide.inputs`	A character vector of input IDs to hide. Their values are still initialized and used, but the controls are not shown in the UI.
`hide.tabs`	A character vector of tab names to hide. Inputs in these tabs are still initialized and used, but the controls are not shown in the UI.

Details

When `data_list` is not provided (or `NULL`), the app launches with `gallery_sales` as an example dataset. Uploaded data files are added to the available datasets and can be selected for plotting. If an uploaded file shares a name with an existing dataset, the existing one is overwritten with a warning.

This is a convenience wrapper around `createModuleApp()`.

Value

A Shiny app object.

Author(s)

Jacob Martin, Jared Andrews

See Also

parallelCoordinatesPlot(), parallelCoordinatesPlotInputsUI(), parallelCoordinatesPlotOutputUI(), parallelCoordinatesPlotServer()

Examples

```
library(VizModules)
# Launch with default example data:
app <- parallelCoordinatesPlotApp()
if (interactive()) runApp(app)

# Launch with custom data:
app2 <- parallelCoordinatesPlotApp(list("sales" = example_sales))
if (interactive()) runApp(app2)
```

---

parallelCoordinatesPlotInputsUI

Input UI components for the parallelCoordinatesPlot module

Description

This should be placed in the UI where the inputs should be shown, with an `id` that matches the `id` used in the `parallelCoordinatesPlotServer()` and `parallelCoordinatesPlotOutputUI()` functions.

Usage

```
parallelCoordinatesPlotInputsUI(
  id,
  data,
  defaults = NULL,
  title = NULL,
  columns = 2
)
```

Arguments

`id`	The ID for the Shiny module.
`data`	The data frame used for plot generation.
`defaults`	A named list of default values for the inputs.
`title`	An optional title for the UI grid.
`columns`	Number of columns for the UI grid.

Details

The user inputs for this module are separated from the outputs to allow for more flexible UI design.

The inputs will automatically be organized into a grid layout via the `organize_inputs()` function, with `columns` controlling the number of columns in the grid.

Defaults can be set for each input by providing a named list of values to the `defaults` argument. Nearly all parameters for `parallelCoordinatesPlot()` can be set via these inputs, so see the help for that function for an exhaustive list.

Value

A Shiny tagList containing the UI elements

Plot parameters not implemented or with altered functionality

The following `parallelCoordinatesPlot()` parameters are not exposed as UI inputs:

- `title.text` - Plot title text (plotly allows interactive editing; use `defaults` to set)

Plot parameters and defaults

The following `parallelCoordinatesPlot()` parameters can be accessed via UI inputs and/or the `defaults` argument:

- `title.font.size` - Plot title font size (UI: "Title Size", default: 26)
- `title.font.family` - Font family for title text (UI: "Title Font", default: "Arial")
- `title.font.color` - Color for plot title (UI: "Title Color", default: "#000000")
- `dimensions` - Columns to use as axes (UI: "Select dimensions", multiple: TRUE)
- `color.by` - Column to color lines by (UI: "Color by", default: "")
- `color.scale` - Colorscale for lines when `color.by` is numeric (UI: "Color Scale", default: "Viridis")
- `palette.selection` - Discrete color palette used when `color.by` is categorical (UI: palette picker)
- `line.opacity` - Line opacity (UI: "Line opacity", default: 0.5)
- `line.width` - Line width (UI: "Line width", default: 1)
- `show.colorbar` - Show colorbar (UI: "Show colorbar", default: TRUE)
- `label.font.size` - Dimension label font size (UI: "Label font size", default: 12)
- `label.font.color` - Dimension label font color (UI: "Label font color", default: "black")
- `label.font.family` - Dimension label font family (UI: "Label font", default: "Arial")
- `tick.font.size` - Tick label font size (UI: "Tick font size", default: 10)
- `tick.font.color` - Tick label font color (UI: "Tick font color", default: "black")
- `tick.font.family` - Tick label font family (UI: "Tick font", default: "Arial")
- `bgcolor` - Plot background color (UI: "Background color", default: "#FFFFFF")

Author(s)

Jacob Martin, Jared Andrews

See Also

parallelCoordinatesPlot(), organize_inputs(), parallelCoordinatesPlotOutputUI(), parallelCoordinatesPlo
parallelCoordinatesPlotApp()

Examples

```
library(VizModules)
parallelCoordinatesPlotInputsUI("parcoords", mtcars)
```

---

parallelCoordinatesPlotOutputUI

Output UI components for the parallelCoordinatesPlot module

Description

This should be placed in the UI where the plot should be shown.

Usage

```
parallelCoordinatesPlotOutputUI(id, resizable = TRUE)
```

Arguments

id	The ID for the Shiny module.
resizable	Logical; when TRUE (the default) the plot output is wrapped in shinyjqui::jqui_resizable() so it can be resized by dragging. Set to FALSE when embedding the output in a container that already provides resizing.

Value

A Shiny plotlyOutput for the parallelCoordinatesPlot

Author(s)

Jacob Martin, Jared Andrews

parallelCoordinatesPlotServer *Server logic for parallelCoordinatesPlot module*

Description

Server logic for parallelCoordinatesPlot module

Usage

```
parallelCoordinatesPlotServer(
  id,
  data,
  hide.inputs = NULL,
  hide.tabs = NULL,
  defaults = NULL
)
```

Arguments

id	The ID for the Shiny module.
data	A reactive containing the data frame to plot.
hide.inputs	A character vector of input IDs to hide. These will still be initialized and their values passed to the plot function, but the user will not be able to see/adjust them in the UI.
hide.tabs	A character vector of tab names to hide. Inputs in these tabs will still be initialized and their values passed to the plot function, but the user will not be able to see/adjust them in the UI.
defaults	A named list of default values for the inputs. When the reset button is clicked, inputs are reset to these values rather than hardcoded fallbacks. Typically the same list passed to the corresponding UI function.

Value

The moduleServer function for the parallelCoordinatesPlot module.

Author(s)

Jacob Martin, Jared Andrews

See Also

parallelCoordinatesPlot(), parallelCoordinatesPlotInputsUI(), parallelCoordinatesPlotOutputUI(), parallelCoordinatesPlotApp()

parse_numeric_list *Parse comma-separated numeric string to vector*

Description

Parses a text string containing comma-separated numeric values into a numeric vector. Used for parsing user input for line intercepts, slopes, etc.

Usage

```
parse_numeric_list(text)
```

Arguments

text	Character. A string containing comma-separated numeric values (e.g., "1, 5, 8").

Details

Whitespace around values is trimmed. Non-numeric values are converted to NA. If all values are NA or the input is empty, returns NULL.

Value

A numeric vector of parsed values, or NULL if input is empty/invalid.

Author(s)

Jared Andrews

Examples

```
parse_numeric_list("1, 5, 8")
parse_numeric_list("")
```

parse_pair_strings *Parse pair strings from UI into list of length-2 vectors*

Description

Converts the "group1 vs group2" strings from the comparison selector back into a list of length-2 character vectors for compute_pairwise_stats().

Usage

```
parse_pair_strings(pair_strings)
```

Arguments

pair_strings	Character vector of pair strings from UI input.

Value

A list of length-2 character vectors, or NULL if input is empty.

Author(s)

Jared Andrews

Examples

```
parse_pair_strings(c("setosa vs versicolor", "versicolor vs virginica"))
```

---

piePlot *Create a plotly pie chart*

Description

Create a plotly pie chart

Usage

```
piePlot(
  df,
  labels,
  values,
  colors = NULL,
  palette = NULL,
  hole = 0,
  textinfo = "label+percent",
  textposition = "auto",
  insidetextorientation = "auto",
  sort = TRUE,
  direction = "counterclockwise",
  rotation = 0,
  show.legend = TRUE,
  legend.orientation = "h",
  legend.x = 0.5,
  legend.y = -0.1,
  legend.font.family = "Arial",
  legend.font.size = 12,
  legend.font.color = "#000000",
  title.text = "",
  title.font.family = "Arial",
```

```
  title.font.size = 18,
  title.font.color = "#000000",
  title.x = 0.5,
  text.font.family = "Arial",
  text.font.size = 12,
  text.font.color = "#000000",
  slice.line.color = "#FFFFFF",
  slice.line.width = 0
)
```

Arguments

df	A data frame where each row already represents a summarized slice (e.g., counts per category) with label and value columns.
labels	Character, name of the column to use for the slice labels.
values	Character, name of the column to use for the aggregated values (slice sizes).
colors	Optional character vector of hex colors for the slices. If named, values are matched to the values in labels; otherwise colours are recycled in data order.
palette	Optional character vector of fallback colors used when colors is not supplied or missing values are present.
hole	Numeric value between 0 and 1 for the hole size (0 for pie chart, >0 for donut chart). Default: 0.
textinfo	Character string specifying the text info to show on slices. Any combination of "label", "text", "value", "percent" joined with a "+" (e.g., "label+percent") or "none" to hide text. Default: "label+percent".
textposition	Character, position of the text relative to the slice. Options: "auto", "inside", "outside", or "none". Default: "auto".
insidetextorientation	Character, orientation for inside text. Options: "auto", "horizontal", "radial", or "tangential". Default: "auto".
sort	Logical, whether to sort slices by their values in descending order. Default: TRUE.
direction	Character, direction of slice progression. Options: "counterclockwise" or "clockwise". Default: "counterclockwise".
rotation	Numeric, starting angle of the first slice in degrees (0-360). Default: 0.
show.legend	Logical, whether to display the legend. Default: TRUE.
legend.orientation	Character, legend orientation. Options: "h" (horizontal) or "v" (vertical). Default: "h".
legend.x	Numeric, horizontal legend position offset (0-1, where 0=left, 1=right). Default: 0.5.
legend.y	Numeric, vertical legend position offset (-1 to 1). Default: -0.1.
legend.font.family	Character, font family for the legend text. Default: "Arial".

`legend.font.size`
: Numeric, font size for the legend text. Default: 12.

`legend.font.color`
: Character, hex color for the legend text. Default: "#000000".

`title.text`
: Character, main plot title text. Default: "".

`title.font.family`
: Character, font family for the title text. Default: "Arial".

`title.font.size`
: Numeric, font size for the title text. Default: 18.

`title.font.color`
: Character, hex color for the title text. Default: "#000000".

`title.x`
: Numeric, horizontal position for the plot title (0-1, where 0=left, 0.5=center, 1=right). Default: 0.5.

`text.font.family`
: Character, font family for the slice labels. Default: "Arial".

`text.font.size`
: Numeric, font size for the slice labels. Default: 12.

`text.font.color`
: Character, hex color for the slice labels. Default: "#000000".

`slice.line.color`
: Character, hex color for slice borders. Default: "#FFFFFF" (white).

`slice.line.width`
: Numeric, width of slice borders in pixels. Set to 0 for no borders. Default: 0.

Value

A plotly object.

Author(s)

Jacob Martin, Jared Andrews

Examples

```
status_counts <- data.frame(
    status = c("Upregulated", "Downregulated", "Not significant"),
    n = c(12, 7, 3)
)

piePlot(
    df = status_counts,
    labels = "status",
    values = "n",
    palette = c("#1B9E77", "#D95F02", "#7570B3"),
    sort = FALSE,
    title.text = "Genes by status"
)
```

piePlotApp *Create an example Modular piePlot Shiny Application*

Description

This function generates a Shiny application with modular piePlot components. The app features a **Data Import** section for uploading data, a **Data Table** for filtering the active dataset, and a **Plot** area for configuring and displaying an interactive pie plot.

Usage

```
piePlotApp(
  data_list = NULL,
  defaults = NULL,
  hide.inputs = NULL,
  hide.tabs = NULL
)
```

Arguments

`data_list`	An optional named list of summary data frames (one row per slice). If `NULL` (the default), aggregated example data is used. Each data frame should already contain a label column and an aggregated numeric value column.
`defaults`	A named list of input IDs and their default values to apply on startup.
`hide.inputs`	A character vector of input IDs to hide. Their values are still initialized and used, but the controls are not shown in the UI.
`hide.tabs`	A character vector of tab names to hide. Inputs in these tabs are still initialized and used, but the controls are not shown in the UI.

Details

When `data_list` is not provided (or `NULL`), the app launches with an aggregated `example_sales` dataset (revenue by product line). Uploaded data files are added to the available datasets and can be selected for plotting. If an uploaded file shares a name with an existing dataset, the existing one is overwritten with a warning.

This is a convenience wrapper around `createModuleApp()`.

Value

A Shiny app object.

Author(s)

Jacob Martin, Jared Andrews

See Also

piePlot(), piePlotInputsUI(), piePlotOutputUI(), piePlotServer()

Examples

```
library(VizModules)
# Launch with default example data:
app <- piePlotApp()
if (interactive()) runApp(app)

# Launch with custom data:
sales_summary <- aggregate(revenue ~ product_line, example_sales, sum)
app2 <- piePlotApp(list("sales" = sales_summary))
if (interactive()) runApp(app2)
```

---

piePlotInputsUI *Input UI components for the piePlot module*

Description

This should be placed in the UI where the inputs should be shown, with an `id` that matches the `id` used in the `piePlotServer()` and `piePlotOutputUI()` functions.

Usage

```
piePlotInputsUI(id, data, defaults = NULL, title = NULL, columns = 2)
```

Arguments

`id`	The ID for the Shiny module.
`data`	The data frame used for plot generation. Supply a summary table with one row per slice.
`defaults`	A named list of default values for the inputs.
`title`	An optional title for the UI grid.
`columns`	Number of columns for the UI grid.

Details

The user inputs for this module are separated from the outputs to allow for more flexible UI design.

The inputs will automatically be organized into a grid layout via the `organize_inputs()` function, with `columns` controlling the number of columns in the grid.

Defaults can be set for each input by providing a named list of values to the `defaults` argument. Provide summarized data (one row per slice) with columns for labels and aggregated values. Nearly all parameters for `piePlot()` can be set via these inputs, so see the help for that function for an exhaustive list.

Value

A Shiny tagList containing the UI elements

Plot parameters not implemented or with altered functionality

The following `piePlot()` parameters are not exposed as UI inputs:

- `palette` - Color palette name; use `colors` via the color picker UI instead
- `legend.x` - Legend horizontal position offset (use `defaults` to set)
- `legend.y` - Legend vertical position offset (use `defaults` to set)
- `title.text` - Plot title text (plotly allows interactive editing; use `defaults` to set)

Plot parameters and defaults

The following `piePlot()` parameters can be accessed via UI inputs and/or the `defaults` argument:

- `title.font.size` - Plot title font size (UI: "Title Size", default: 26)
- `title.font.family` - Font family for title text (UI: "Title Font", default: "Arial")
- `title.font.color` - Color for plot title (UI: "Title Color", default: "#000000")
- `labels` - Label column (UI: "Label column (summary data)", default: 2nd categorical column)
- `values` - Aggregated value column (UI: "Aggregated value column", default: 2nd numeric column)
- `sort` - Sort slices by value (UI: "Sort slices by value", default: TRUE)
- `direction` - Slice direction (UI: "Slice direction", default: "counterclockwise")
- `rotation` - Start angle in degrees (UI: "Start angle (degrees)", default: 0)
- `hole` - Center hole size for donut chart (UI: "Center hole size", default: 0)
- `colors` - Slice colors (UI: color picker, derived from palette)
- `slice.line.color` - Slice border color (UI: "Slice border color", default: "#FFFFFF")
- `slice.line.width` - Slice border width (UI: "Slice border width", default: 0)
- `textinfo` - Text to show on slices (UI: "Text to show on slices", default: c("label", "value", "percent"))
- `textposition` - Text position (UI: "Text position", default: "auto")
- `insidetextorientation` - Inside text orientation (UI: "Inside text orientation", default: "auto")
- `text.font.size` - Slice text size (UI: "Slice text size", default: 12)
- `text.font.family` - Slice text font (UI: "Slice text font", default: "Arial")
- `text.font.color` - Slice text color (UI: "Slice text color", default: "#000000")
- `title.x` - Title horizontal position (UI: "Title horizontal position", default: 0.5)
- `show.legend` - Show legend (UI: "Show legend", default: TRUE)
- `legend.orientation` - Legend orientation (UI: "Legend orientation", default: "h")
- `legend.font.family` - Legend font (UI: "Legend font", default: "Arial")
- `legend.font.size` - Legend font size (UI: "Legend font size", default: 12)
- `legend.font.color` - Legend font color (UI: "Legend font color", default: "#000000")

Author(s)

Jacob Martin, Jared Andrews

See Also

piePlot(), organize_inputs(), piePlotOutputUI(), piePlotServer(), piePlotApp()

Examples

```
library(VizModules)
pie_df <- as.data.frame(table(iris$Species))
names(pie_df) <- c("Species", "Count")
piePlotInputsUI("piePlot", pie_df)
```

---

piePlotOutputUI *Output UI components for the piePlot module*

Description

This should be placed in the UI where the plot should be shown.

Usage

```
piePlotOutputUI(id, resizable = TRUE)
```

Arguments

id	The ID for the Shiny module.
resizable	Logical; when TRUE (the default) the plot output is wrapped in shinyjqui::jqui_resizable() so it can be resized by dragging. Set to FALSE when embedding the output in a container that already provides resizing.

Value

A Shiny plotlyOutput for the piePlot

Author(s)

Jacob Martin, Jared Andrews

`piePlotServer` *Server logic for piePlot module*

Description

Server logic for piePlot module

Usage

```
piePlotServer(id, data, hide.inputs = NULL, hide.tabs = NULL, defaults = NULL)
```

Arguments

`id`	The ID for the Shiny module.
`data`	A `reactive` containing the data frame to plot. Provide a summarized table with columns for labels and aggregated values.
`hide.inputs`	A character vector of input IDs to hide. These will still be initialized and their values passed to the plot function, but the user will not be able to see/adjust them in the UI.
`hide.tabs`	A character vector of tab names to hide. Inputs in these tabs will still be initialized and their values passed to the plot function, but the user will not be able to see/adjust them in the UI.
`defaults`	A named list of default values for the inputs. When the reset button is clicked, inputs are reset to these values rather than hardcoded fallbacks. Typically the same list passed to the corresponding UI function.

Value

The `moduleServer` function for the piePlot module.

Author(s)

Jacob Martin, Jared Andrews

See Also

`piePlot()`, `piePlotInputsUI()`, `piePlotOutputUI()`, `piePlotApp()`

`plotthis_AreaPlotApp` *Create an example Modular AreaPlot Shiny Application*

Description

This function generates a Shiny application with modular `plotthis::AreaPlot()` components. The app features a **Data Import** section for uploading data, a **Data Table** for filtering the active dataset, and a **Plot** area for configuring and displaying an interactive area plot.

Usage

```
plotthis_AreaPlotApp(
  data_list = NULL,
  defaults = NULL,
  hide.inputs = NULL,
  hide.tabs = NULL
)
```

Arguments

`data_list`	An optional named list of data frames. If `NULL` (the default), `list("sales" = example_sales)` is used as example data.
`defaults`	A named list of input IDs and their default values to apply on startup.
`hide.inputs`	A character vector of input IDs to hide. Their values are still initialized and used, but the controls are not shown in the UI.
`hide.tabs`	A character vector of tab names to hide. Inputs in these tabs are still initialized and used, but the controls are not shown in the UI.

Details

When `data_list` is not provided (or `NULL`), the app launches with `example_sales` as an example dataset. Uploaded data files are added to the available datasets and can be selected for plotting. If an uploaded file shares a name with an existing dataset, the existing one is overwritten with a warning.

This is a convenience wrapper around `createModuleApp()`.

Value

A Shiny app object.

Author(s)

Jacob Martin, Jared Andrews

Examples

```
library(VizModules)
# Launch with default example data:
app <- plotthis_AreaPlotApp()
if (interactive()) runApp(app)

# Launch with custom data:
app2 <- plotthis_AreaPlotApp(list("sales" = example_sales))
if (interactive()) runApp(app2)
```

---

plotthis_AreaPlotInputsUI

Input UI components for the AreaPlot module

Description

This should be placed in the UI where the inputs should be shown, with an `id` that matches the `id` used in the `plotthis_AreaPlotServer()` and `plotthis_AreaPlotOutputUI()` functions.

Usage

```
plotthis_AreaPlotInputsUI(id, data, defaults = NULL, title = NULL, columns = 2)
```

Arguments

`id`	The ID for the Shiny module.
`data`	The data frame used for plot generation.
`defaults`	A named list of default values for the inputs.
`title`	An optional title for the UI grid.
`columns`	Number of columns for the UI grid.

Details

The user inputs for this module are separated from the outputs to allow for more flexible UI design.

The inputs will automatically be organized into a grid layout via the `organize_inputs()` function, with `columns` controlling the number of columns in the grid.

Defaults can be set for each input by providing a named list of values to the `defaults` argument. Nearly all parameters for `plotthis::AreaPlot()` can be set via these inputs, so see the help for that function for an exhaustive list.

Value

A Shiny tagList containing the UI elements

Plot parameters not implemented or with altered functionality

The following `plotthis::AreaPlot()` parameters are not available via UI inputs:

- `xlab` - X-axis label (plotly allows interactive editing)
- `ylab` - Y-axis label (plotly allows interactive editing)
- `title` - Plot title (plotly allows interactive editing)
- `subtitle` - Plot subtitle (not supported in plotly)
- `aspect.ratio` - Aspect ratio control (handled by plotly layout)
- `legend.position` - Legend positioning (plotly allows interactive repositioning)
- `split_by` - Split variable (returns a patchwork object, not supported in plotly), use `facet_by` instead
- `design` - Only applies if `split_by` is used
- `split_by_sep` - Only applies if `split_by` is used
- `axes` - Only applies if `split_by` is used
- `axis_titles` - Only applies if `split_by` is used
- `guides` - Only applies if `split_by` is used
- `byrow` - Only applies if `split_by` is used
- `nrow` - Only applies if `split_by` is used
- `ncol` - Only applies if `split_by` is used
- `x_sep` - Separator for multiple x columns (not yet implemented)
- `group_by_sep` - Separator for multiple group_by columns (not yet implemented)
- `group_name` - Group/fill legend name (not yet implemented)
- `palette` - Managed internally via the palette selection UI
- `palreverse` - Reverse the color palette (not yet implemented)
- `x_text_angle` - X-axis text angle (handled by axis.tickangle.x)
- `keep_empty` - Keep empty factor levels (not yet implemented)
- `keep_na` - Keep NA values (not yet implemented)
- `seed` - Random seed (not applicable)
- `combine` - Combine multiple plots (not applicable for plotly)
- `legend.direction` - Managed position of legend however this can be handled via plotly

Plot parameters and defaults

The following `plotthis::AreaPlot()` parameters can be accessed via UI inputs and/or the `defaults` argument:

- `x` - X-axis variable (UI: "X values", default: 2nd categorical variable)
- `y` - Y-axis variable (UI: "Y values", default: 2nd numeric variable)
- `group_by` - Grouping variable for area fill (UI: "Group by", default: 3rd categorical variable or "")

- facet_by - Faceting variable (UI: "Facet by", default: "")
- facet_scales - Facet scale behavior (UI: "Facet scale", default: "fixed")
- facet_ncol - Number of facet columns (UI: "Columns", default: NULL)
- facet_nrow - Number of facet rows (UI: "Rows", default: NULL)
- facet_byrow - Facet ordering direction (UI: "Facet by row", default: TRUE)
- palcolor - Custom color values (UI: palette picker, derived from palette)
- alpha - Area fill transparency (UI: "Alpha", default: 1)
- scale_y - Scale y-axis by total (UI: "Scale y-axis by total", default: FALSE)

Parameters controlling additional functionality

The following parameters implementing new functionality or controlling plotly-specific features are also available:

- title.font.size - Plot title font size (UI: "Title Size", default: 26)
- title.font.family - Font family for title text (UI: "Title Font", default: "Arial")
- title.font.color - Color for plot title (UI: "Title Color", default: "#000000")
- axis.title.font.size - Axis title font size (UI: "Axis Title Size", default: 18)
- axis.title.font.color - Axis title font color (UI: "Axis Title Color", default: "#000000")
- axis.title.font.family - Axis title font family (UI: "Axis Title Font", default: "Arial")
- axis.showline - Show axis border lines (UI: "Show axis lines", default: TRUE)
- axis.mirror - Mirror axis lines on opposite side (UI: "Mirror axis lines", default: TRUE)
- show.grid.x - Show X-axis major gridlines (UI: "Show X major gridlines", default: TRUE)
- show.grid.y - Show Y-axis major gridlines (UI: "Show Y major gridlines", default: TRUE)
- axis.linecolor - Color of axis lines (UI: "Axis line color", default: "black")
- axis.linewidth - Width of axis lines (UI: "Axis line width", default: 0.5)
- axis.tickfont.size - Size of tick labels (UI: "Tick label size", default: 12)
- axis.tickfont.color - Color of tick labels (UI: "Tick label color", default: "black")
- axis.tickfont.family - Font family for tick labels (UI: "Tick label font", default: "Arial")
- axis.tickangle.x - Rotation angle for X-axis tick labels (UI: "X-axis tick label angle", default: 0)
- axis.tickangle.y - Rotation angle for Y-axis tick labels (UI: "Y-axis tick label angle", default: 0)
- axis.ticks - Position of tick marks (UI: "Tick position", default: "outside")
- axis.tickcolor - Color of tick marks (UI: "Tick mark color", default: "black")
- axis.ticklen - Length of tick marks (UI: "Tick mark length", default: 5)
- axis.tickwidth - Width of tick marks (UI: "Tick mark width", default: 1)
- hline.intercepts - Y-coordinates for horizontal reference lines (UI: "Y-intercepts", default: "")
- hline.colors - Colors for horizontal lines (UI: "Colors", default: "#000000")

- `hline.widths` - Widths for horizontal lines (UI: "Widths", default: "1")
- `hline.linetypes` - Line types for horizontal lines (UI: "Line types", default: "dashed")
- `hline.opacities` - Opacities for horizontal lines (UI: "Opacities (0-1)", default: "1")
- `vline.intercepts` - X-coordinates for vertical reference lines (UI: "X-intercepts", default: "")
- `vline.colors` - Colors for vertical lines (UI: "Colors", default: "#000000")
- `vline.widths` - Widths for vertical lines (UI: "Widths", default: "1")
- `vline.linetypes` - Line types for vertical lines (UI: "Line types", default: "dashed")
- `vline.opacities` - Opacities for vertical lines (UI: "Opacities (0-1)", default: "1")
- `abline.slopes` - Slopes for diagonal reference lines (UI: "Slopes", default: "")
- `abline.intercepts` - Y-intercepts for diagonal lines (UI: "Y-intercepts", default: "")
- `abline.colors` - Colors for diagonal lines (UI: "Colors", default: "#000000")
- `abline.widths` - Widths for diagonal lines (UI: "Widths", default: "1")
- `abline.linetypes` - Line types for diagonal lines (UI: "Line types", default: "dashed")
- `abline.opacities` - Opacities for diagonal lines (UI: "Opacities (0-1)", default: "1")

Author(s)

Jacob Martin, Jared Andrews

See Also

`plotthis::AreaPlot()`, `organize_inputs()`, `plotthis_AreaPlotOutputUI()`, `plotthis_AreaPlotServer()`, `plotthis_AreaPlotApp()`

Examples

```
library(VizModules)
# Needs at least 2 categorical variables for grouping and x-axis
mtcars$cyl <- as.factor(mtcars$cyl)
mtcars$gear <- as.factor(mtcars$gear)
plotthis_AreaPlotInputsUI("areaPlot", mtcars)
```

---

plotthis_AreaPlotOutputUI

Output UI components for the AreaPlot module

Description

This should be placed in the UI where the plot should be shown.

Usage

```
plotthis_AreaPlotOutputUI(id, resizable = TRUE)
```

Arguments

id	The ID for the Shiny module.
resizable	Logical; when TRUE (the default) the plot output is wrapped in shinyjqui::jqui_resizable() so it can be resized by dragging. Set to FALSE when embedding the output in a container that already provides resizing.

Value

A Shiny plotlyOutput for the AreaPlot

Author(s)

Jacob Martin

---

plotthis_AreaPlotServer

Server logic for AreaPlot module

Description

Server logic for AreaPlot module

Usage

```
plotthis_AreaPlotServer(
  id,
  data,
  hide.inputs = NULL,
  hide.tabs = NULL,
  defaults = NULL
)
```

Arguments

id	The ID for the Shiny module.
data	A reactive containing the data frame to plot.
hide.inputs	A character vector of input IDs to hide. These will still be initialized and their values passed to the plot function, but the user will not be able to see/adjust them in the UI.
hide.tabs	A character vector of tab names to hide. Inputs in these tabs will still be initialized and their values passed to the plot function, but the user will not be able to see/adjust them in the UI.
defaults	A named list of default values for the inputs. When the reset button is clicked, inputs are reset to these values rather than hardcoded fallbacks. Typically the same list passed to the corresponding UI function.

Value

The `moduleServer` function for the AreaPlot module.

Author(s)

Jacob Martin, Jared Andrews

`plotthis_BarPlotApp` *Create an example Modular BarPlot Shiny Application*

Description

This function generates a Shiny application with modular `plotthis::BarPlot()` components. The app features a **Data Import** section for uploading data, a **Data Table** for filtering the active dataset, and a **Plot** area for configuring and displaying an interactive bar plot.

Usage

```
plotthis_BarPlotApp(
  data_list = NULL,
  defaults = NULL,
  hide.inputs = NULL,
  hide.tabs = NULL
)
```

Arguments

`data_list`	An optional named list of data frames. If `NULL` (the default), `list("Bar" = example_bar)` is used as example data.
`defaults`	A named list of input IDs and their default values to apply on startup.
`hide.inputs`	A character vector of input IDs to hide. Their values are still initialized and used, but the controls are not shown in the UI.
`hide.tabs`	A character vector of tab names to hide. Inputs in these tabs are still initialized and used, but the controls are not shown in the UI.

Details

When `data_list` is not provided (or `NULL`), the app launches with `example_bar` as an example dataset. Uploaded data files are added to the available datasets and can be selected for plotting. If an uploaded file shares a name with an existing dataset, the existing one is overwritten with a warning.

This is a convenience wrapper around `createModuleApp()`.

Value

A Shiny app object.

Author(s)

Jacob Martin, Jared Andrews

Examples

```
library(VizModules)
# Launch with default example data:
app <- plotthis_BarPlotApp()
if (interactive()) runApp(app)

# Launch with custom data:
app2 <- plotthis_BarPlotApp(list("Bar" = example_bar))
if (interactive()) runApp(app2)
```

---

plotthis_BarPlotInputsUI

Input UI components for the BarPlot module

Description

This should be placed in the UI where the inputs should be shown, with an `id` that matches the `id` used in the `plotthis_BarPlotServer()` and `plotthis_BarPlotOutputUI()` functions.

Usage

```
plotthis_BarPlotInputsUI(id, data, defaults = NULL, title = NULL, columns = 2)
```

Arguments

`id`	The ID for the Shiny module.
`data`	The data frame used for plot generation.
`defaults`	A named list of default values for the inputs.
`title`	An optional title for the UI grid.
`columns`	Number of columns for the UI grid.

Details

The user inputs for this module are separated from the outputs to allow for more flexible UI design.

The inputs will automatically be organized into a grid layout via the `organize_inputs()` function, with `columns` controlling the number of columns in the grid.

Defaults can be set for each input by providing a named list of values to the `defaults` argument. Nearly all parameters for `plotthis::BarPlot()` can be set via these inputs, so see the help for that function for an exhaustive list.

Value

A Shiny tagList containing the UI elements

Plot parameters not implemented or with altered functionality

The following `plotthis::BarPlot()` parameters are not available via UI inputs:

- `xlab` - X-axis label (plotly allows interactive editing)
- `ylab` - Y-axis label (plotly allows interactive editing)
- `title` - Plot title (plotly allows interactive editing)
- `subtitle` - Plot subtitle (not supported in plotly)
- `aspect.ratio` - Aspect ratio control (handled by plotly layout)
- `legend.position` - Legend positioning (plotly allows interactive repositioning)
- `position` - Bar position (auto, stack, dodge, fill) (not yet implemented)
- `position_dodge_preserve` - Preserve bar width when dodging (not yet implemented)
- `x_sep` - Separator for multiple x columns (not yet implemented)
- `group_by_sep` - Separator for multiple group_by columns (not yet implemented)
- `split_by_sep` - Separator for multiple split_by columns (not yet implemented)
- `fill_name` - Name of the fill legend (not yet implemented)
- `line_name` - Name of line (not yet implemented)
- `label` - Bar labels on top (not yet implemented)
- `label_nudge` - Label nudge distance (not yet implemented)
- `label_fg` - Label foreground color (not yet implemented)
- `label_size` - Label size (not yet implemented)
- `label_bg` - Label background color (not yet implemented)
- `label_bg_r` - Label background radius (not yet implemented)
- `group_name` - Group legend name (not yet implemented)
- `facet_args` - Additional facet arguments (not yet implemented)
- `add_bg` - Add background stripes (not yet implemented)
- `bg_palette` - Background palette (not yet implemented)
- `bg_palcolor` - Background palette colors (not yet implemented)
- `bg_alpha` - Background alpha (not yet implemented)
- `add_line` - Add horizontal line (not yet implemented)
- `line_color` - Horizontal line color (not yet implemented)
- `line_width` - Horizontal line width (not yet implemented)
- `line_type` - Horizontal line type (not yet implemented)
- `add_trend` - Add trend line (not yet implemented)
- `trend_color` - Trend line color (not yet implemented)
- `trend_linewidth` - Trend line width (not yet implemented)
- `trend_ptsize` - Trend point size (not yet implemented)
- `theme` - ggplot2 theme (managed internally)
- `theme_args` - Theme arguments (not yet implemented)

- palette - Managed internally via the palette selection UI
- x_text_angle - X-axis text angle (handled by axis.tickangle.x)
- legend.direction - Legend orientation (plotly allows interactive adjustment)
- keep_empty - Keep empty factor levels (not yet implemented)
- keep_na - Keep NA values (not yet implemented)
- combine - Combine multiple plots (not applicable for plotly)
- nrow - Only applies if split_by is used with combine
- ncol - Only applies if split_by is used with combine
- byrow - Only applies if split_by is used with combine
- seed - Random seed (not applicable)
- axes - Only applies if split_by is used with combine
- axis_titles - Only applies if split_by is used with combine
- guides - Only applies if split_by is used with combine
- design - Only applies if split_by is used with combine

Plot parameters and defaults

The following plotthis::BarPlot() parameters can be accessed via UI inputs and/or the defaults argument:

- x - X-axis variable (UI: "X values", default: 2nd categorical variable)
- y - Y-axis variable (UI: "Y values", default: 2nd numeric variable)
- group_by - Grouping variable for bar fill (UI: "Group by", default: 2nd categorical variable)
- fill_by - Variable used to fill the bars (UI: "Fill by", default: "")
- flip - Flip/swap the x and y axes (UI: "Rotate (swap X/Y)", default: FALSE)
- split_by - Split variable for separate plots (UI: "Split by", default: "")
- facet_by - Faceting variable (UI: "Facet by", default: "")
- facet_scales - Facet scale behavior (UI: "Facet scale", default: "fixed")
- facet_ncol - Number of facet columns (UI: "Facet number of columns", default: NULL)
- facet_nrow - Number of facet rows (UI: "Facet number of rows", default: NULL)
- facet_byrow - Facet ordering direction (UI: "Facet by row", default: TRUE)
- palcolor - Custom color values (UI: palette picker, derived from palette)
- palreverse - Reverse the color palette (UI: "Reverse palette", default: FALSE)
- alpha - Bar fill transparency (UI: "Alpha", default: 1)
- width - Bar width (UI: "Width", default: NA)
- expand - Axis expansion values (UI: "Expand", default: "")
- y_min - Y-axis minimum value (UI: "Y-axis min", default: 0)
- y_max - Y-axis maximum value (UI: "Y-axis max", default: max of data)
- lower_quantile - Lower quantile for the continuous fill color scale (UI: "Lower Quantile", default: 0); only affects a numeric fill_by

- upper_quantile - Upper quantile for the continuous fill color scale (UI: "Upper Quantile", default: 1); only affects a numeric fill_by
- lower_cutoff - Explicit lower cutoff for the continuous fill color scale (UI: "Lower Cutoff", default: NA); overrides lower_quantile when set
- upper_cutoff - Explicit upper cutoff for the continuous fill color scale (UI: "Upper Cutoff", default: NA); overrides upper_quantile when set

Parameters controlling additional functionality

The following parameters implementing new functionality or controlling plotly-specific features are also available:

- title.font.size - Plot title font size (UI: "Title Size", default: 26)
- title.font.family - Font family for title text (UI: "Title Font", default: "Arial")
- title.font.color - Color for plot title (UI: "Title Color", default: "#000000")
- axis.title.font.size - Axis title font size (UI: "Axis Title Size", default: 18)
- axis.title.font.color - Axis title font color (UI: "Axis Title Color", default: "#000000")
- axis.title.font.family - Axis title font family (UI: "Axis Title Font", default: "Arial")
- axis.showline - Show axis border lines (UI: "Show axis lines", default: TRUE)
- axis.mirror - Mirror axis lines on opposite side (UI: "Mirror axis lines", default: TRUE)
- show.grid.x - Show X-axis major gridlines (UI: "Show X major gridlines", default: TRUE)
- show.grid.y - Show Y-axis major gridlines (UI: "Show Y major gridlines", default: TRUE)
- axis.linecolor - Color of axis lines (UI: "Axis line color", default: "black")
- axis.linewidth - Width of axis lines (UI: "Axis line width", default: 0.5)
- axis.tickfont.size - Size of tick labels (UI: "Tick label size", default: 12)
- axis.tickfont.color - Color of tick labels (UI: "Tick label color", default: "black")
- axis.tickfont.family - Font family for tick labels (UI: "Tick label font", default: "Arial")
- axis.tickangle.x - Rotation angle for X-axis tick labels (UI: "X-axis tick label angle", default: 0)
- axis.tickangle.y - Rotation angle for Y-axis tick labels (UI: "Y-axis tick label angle", default: 0)
- axis.ticks - Position of tick marks (UI: "Tick position", default: "outside")
- axis.tickcolor - Color of tick marks (UI: "Tick mark color", default: "black")
- axis.ticklen - Length of tick marks (UI: "Tick mark length", default: 5)
- axis.tickwidth - Width of tick marks (UI: "Tick mark width", default: 1)
- hline.intercepts - Y-coordinates for horizontal reference lines (UI: "Y-intercepts", default: "")
- hline.colors - Colors for horizontal lines (UI: "Colors", default: "#000000")
- hline.widths - Widths for horizontal lines (UI: "Widths", default: "1")
- hline.linetypes - Line types for horizontal lines (UI: "Line types", default: "dashed")
- hline.opacities - Opacities for horizontal lines (UI: "Opacities (0-1)", default: "1")

- `vline.intercepts` - X-coordinates for vertical reference lines (UI: "X-intercepts", default: "")
- `vline.colors` - Colors for vertical lines (UI: "Colors", default: "#000000")
- `vline.widths` - Widths for vertical lines (UI: "Widths", default: "1")
- `vline.linetypes` - Line types for vertical lines (UI: "Line types", default: "dashed")
- `vline.opacities` - Opacities for vertical lines (UI: "Opacities (0-1)", default: "1")
- `abline.slopes` - Slopes for diagonal reference lines (UI: "Slopes", default: "")
- `abline.intercepts` - Y-intercepts for diagonal lines (UI: "Y-intercepts", default: "")
- `abline.colors` - Colors for diagonal lines (UI: "Colors", default: "#000000")
- `abline.widths` - Widths for diagonal lines (UI: "Widths", default: "1")
- `abline.linetypes` - Line types for diagonal lines (UI: "Line types", default: "dashed")
- `abline.opacities` - Opacities for diagonal lines (UI: "Opacities (0-1)", default: "1")

Author(s)

Jacob Martin, Jared Andrews

See Also

`plotthis::BarPlot()`, `organize_inputs()`, `plotthis_BarPlotOutputUI()`, `plotthis_BarPlotServer()`, `plotthis_BarPlotApp()`

Examples

```
library(VizModules)
data(mtcars)
plotthis_BarPlotInputsUI("BarPlot", mtcars)
```

`plotthis_BarPlotOutputUI`

Output UI components for the BarPlot module

Description

This should be placed in the UI where the plot should be shown.

Usage

```
plotthis_BarPlotOutputUI(id, resizable = TRUE)
```

Arguments

`id`	The ID for the Shiny module.
`resizable`	Logical; when `TRUE` (the default) the plot output is wrapped in `shinyjqui::jqui_resizable()` so it can be resized by dragging. Set to `FALSE` when embedding the output in a container that already provides resizing.

Value

A Shiny plotlyOutput for the BarPlot

Author(s)

Jacob Martin, Jared Andrews

`plotthis_BarPlotServer` *Server logic for BarPlot module*

Description

Server logic for BarPlot module

Usage

```
plotthis_BarPlotServer(
  id,
  data,
  hide.inputs = NULL,
  hide.tabs = NULL,
  defaults = NULL
)
```

Arguments

`id`	The ID for the Shiny module.
`data`	A `reactive` containing the data frame to plot.
`hide.inputs`	A character vector of input IDs to hide. These will still be initialized and their values passed to the plot function, but the user will not be able to see/adjust them in the UI.
`hide.tabs`	A character vector of tab names to hide. Inputs in these tabs will still be initialized and their values passed to the plot function, but the user will not be able to see/adjust them in the UI.
`defaults`	A named list of default values for the inputs. When the reset button is clicked, inputs are reset to these values rather than hardcoded fallbacks. Typically the same list passed to the corresponding UI function.

Value

The `moduleServer` function for the BarPlot module.

Author(s)

Jacob Martin, Jared Andrews

`plotthis_BoxPlotApp` *Create an example Modular BoxPlot Shiny Application*

Description

This function generates a Shiny application with modular `plotthis::BoxPlot()` components. The app features a **Data Import** section for uploading data, a **Data Table** for filtering the active dataset, and a **Plot** area for configuring and displaying an interactive box plot.

Usage

```
plotthis_BoxPlotApp(
  data_list = NULL,
  defaults = NULL,
  hide.inputs = NULL,
  hide.tabs = NULL
)
```

Arguments

`data_list`	An optional named list of data frames. If `NULL` (the default), `list("demographics" = example_demographics)` is used as example data.
`defaults`	A named list of input IDs and their default values to apply on startup.
`hide.inputs`	A character vector of input IDs to hide. Their values are still initialized and used, but the controls are not shown in the UI.
`hide.tabs`	A character vector of tab names to hide. Inputs in these tabs are still initialized and used, but the controls are not shown in the UI.

Details

When `data_list` is not provided (or `NULL`), the app launches with `example_demographics` as an example dataset. Uploaded data files are added to the available datasets and can be selected for plotting. If an uploaded file shares a name with an existing dataset, the existing one is overwritten with a warning.

This is a convenience wrapper around `createModuleApp()`.

Value

A Shiny app object.

Author(s)

Jacob Martin, Jared Andrews

Examples

```
library(VizModules)
# Launch with default example data:
app <- plotthis_BoxPlotApp()
if (interactive()) runApp(app)

# Launch with custom data:
app2 <- plotthis_BoxPlotApp(list("demographics" = example_demographics))
if (interactive()) runApp(app2)
```

---

plotthis_BoxPlotInputsUI

Input UI components for the BoxPlot module

Description

This should be placed in the UI where the inputs should be shown, with an `id` that matches the `id` used in the `plotthis_BoxPlotServer()` and `plotthis_BoxPlotOutputUI()` functions.

Usage

```
plotthis_BoxPlotInputsUI(id, data, defaults = NULL, title = NULL, columns = 2)
```

Arguments

`id`	The ID for the Shiny module.
`data`	The data frame used for plot generation.
`defaults`	A named list of default values for the inputs.
`title`	An optional title for the UI grid.
`columns`	Number of columns for the UI grid.

Details

The user inputs for this module are separated from the outputs to allow for more flexible UI design.

The inputs will automatically be organized into a grid layout via the `organize_inputs()` function, with `columns` controlling the number of columns in the grid.

Defaults can be set for each input by providing a named list of values to the `defaults` argument. Nearly all parameters for `plotthis::BoxPlot()` can be set via these inputs, so see the help for that function for an exhaustive list.

Value

A Shiny tagList containing the UI elements

Plot parameters not implemented or with altered functionality

The following `plotthis::BoxPlot()` parameters are not available via UI inputs:

- `xlab` - X-axis label (plotly allows interactive editing)
- `ylab` - Y-axis label (plotly allows interactive editing)
- `title` - Plot title (plotly allows interactive editing)
- `subtitle` - Plot subtitle (not supported in plotly)
- `aspect.ratio` - Aspect ratio control (handled by plotly layout)
- `legend.position` - Legend positioning (plotly allows interactive repositioning)
- `x_sep` - Separator for x columns (not applicable in UI context)
- `in_form` - Data input format (not applicable - always long form)
- `split_by` - Split variable (returns a patchwork object, not supported in plotly), use `facet_by` instead
- `split_by_sep` - Only applies if `split_by` is used
- `symnum_args` - Significance symbol arguments (the Stats tab manages symbol display)
- `keep_empty` - Keep empty values (not implemented)
- `keep_na` - Keep NA values (not implemented)
- `group_by_sep` - Separator for group columns (not applicable in UI context)
- `group_name` - Group legend name (handled by plotly)
- `paired_by` - Pairing variable for paired tests (use the Stats tab "Paired Test" input instead)
- `x_text_angle` - X-axis text angle (handled by plotly axis settings)
- `step_increase` - Step increase for significance brackets (set via the Stats tab "Bracket Spacing")
- `base` - Base plot type box/violin/bar (fixed to "box" in this module)
- `fill_mode` - Fill mode for grouped data (handled automatically)
- `position_dodge_preserve` - Preserve dodge width (not implemented)
- `add_errorbar` - Add error bars (only for base = "bar"; not implemented)
- `errorbar_color` - Error bar color (not implemented)
- `errorbar_width` - Error bar cap width (not implemented)
- `errorbar_linewidth` - Error bar line width (not implemented)
- `theme` - ggplot2 theme (not applicable in plotly)
- `theme_args` - Theme arguments (not applicable in plotly)
- `palette` - Managed internally via the palette selection UI
- `palreverse` - Reverse the color palette (not implemented)
- `alpha` - Alpha transparency (not implemented in UI)
- `stack` - Stack boxplots (not implemented)
- `add_beeswarm` - Add beeswarm points (not implemented in UI)
- `beeswarm_method` - Beeswarm arrangement method (not implemented)

- beeswarm_cex - Beeswarm point size factor (not implemented)
- beeswarm_priority - Beeswarm priority order (not implemented)
- beeswarm_dodge - Beeswarm dodge width (not implemented)
- add_trend - Add trend line (not implemented in UI)
- trend_color - Trend line color (not implemented)
- trend_linewidth - Trend line width (not implemented)
- trend_ptsize - Trend point size (not implemented)
- add_stat - Add statistical annotation (not implemented)
- stat_name - Statistical test name (not implemented)
- stat_color - Statistical annotation color (not implemented)
- stat_size - Statistical annotation size (not implemented)
- stat_stroke - Statistical annotation stroke (not implemented)
- stat_shape - Statistical annotation shape (not implemented)
- add_bg - Add background shading (not implemented)
- bg_palette - Background palette (not implemented)
- bg_palcolor - Background color (not implemented)
- bg_alpha - Background transparency (not implemented)
- add_line - Add horizontal line (not implemented in UI - use Lines tab)
- line_color - Line color (not implemented)
- line_width - Line width (not implemented)
- line_type - Line type (not implemented)
- comparisons - plotthis pairwise comparisons are not passed; equivalent pairwise significance testing is provided via the module's Stats tab (see "Statistical annotation parameters")
- ref_group - Reference group for comparisons (use the Stats tab instead)
- pairwise_method - Pairwise test method (set via the Stats tab "Test" input instead)
- multiplegroup_comparisons - Multiple-group comparison flag (use the Stats tab instead)
- multiple_method - Multiple-group test method (set via the Stats tab "Test" input instead)
- sig_label - Significance label format (set via the Stats tab "Display" input instead)
- sig_labelsize - Significance label size (handled by the Stats tab)
- hide_ns - Hide non-significant comparisons (use the Stats tab "Hide Non-Significant" input)
- seed - Random seed (not applicable)
- combine - Only applies if split_by is used
- nrow - Only applies if split_by is used
- ncol - Only applies if split_by is used
- byrow - Only applies if split_by is used
- axes - Only applies if split_by is used
- axis_titles - Only applies if split_by is used
- guides - Only applies if split_by is used
- legend.direction - Managed position of legend however this can be handled via plotly

Plot parameters and defaults

The following `plotthis::BoxPlot()` parameters can be accessed via UI inputs and/or the `defaults` argument:

- `x` - X-axis variable (UI: "X data", default: 2nd categorical variable)
- `y` - Y-axis variable (UI: "Y data", default: 2nd numeric variable)
- `group_by` - Grouping variable (UI: "Group by", default: "")
- `flip` - Flip/swap the x and y axes (UI: "Rotate (swap X/Y)", default: FALSE)
- `sort_x` - Sort X-axis by statistic (UI: "Sort X by", default: "")
- `y_max` - Maximum Y-axis value (UI: "Max Value of Y Axis", default: calculated)
- `y_min` - Minimum Y-axis value (UI: "Min Value of Y Axis", default: calculated)
- `add_point` - Add jitter points (UI: "Add Jitter Points", default: FALSE)
- `pt_size` - Point size (UI: "Point Size", default: 1)
- `pt_alpha` - Point transparency (UI: "Point Alpha", default: 1)
- `jitter_width` - Jitter width (UI: "Jitter Width", default: 0.3)
- `pt_color` - Point outline color (UI: "Point Outline Colour", default: "#000000")
- `highlight` - Highlight condition (UI: "Highlight", default: "")
- `highlight_color` - Highlight color (UI: "Highlight Colour", default: "#000000")
- `highlight_size` - Highlight size (UI: "Highlight Size", default: 1)
- `highlight_alpha` - Highlight transparency (UI: "Highlight Alpha", default: 1)
- `facet_by` - Faceting variable (UI: "Facet by", default: "")
- `facet_scales` - Facet scale behavior (UI: "Facet Scale", default: "fixed")
- `facet_ncol` - Number of facet columns (UI: "Columns", default: NULL)
- `facet_nrow` - Number of facet rows (UI: "Rows", default: NULL)
- `facet_byrow` - Facet ordering direction (UI: "Facet by Row", default: TRUE)
- `palcolor` - Custom color values (UI: palette picker, derived from palette)

Statistical annotation parameters

The module provides plotly-based significance testing via the Stats tab (a reimplementation of the plotthis significance-testing features). The following inputs are available:

- `stats.enabled` - Enable statistical annotations (UI: "Enable Stats", default: FALSE)
- `stat.test` - Test to apply: Wilcoxon, t-test, Kruskal-Wallis, or ANOVA (UI: "Test")
- `stat.p.adjust` - P-value adjustment method (UI: "P-value Adjustment", default: "holm")
- `stat.display` - Value to display: adjusted p-value, p-value, or symbols (UI: "Display")
- `stat.sig.threshold` - Significance threshold for symbols/hiding (UI: "Significance Threshold")
- `stat.hide.ns` - Hide non-significant comparisons (UI: "Hide Non-Significant", default: FALSE)
- `stat.paired` - Use a paired test (UI: "Paired Test", default: FALSE)

- stat.pairs - Group comparisons to test (UI: "Comparisons", multiple selection)
- stat.line.color - Bracket line color (UI: "Line Color", default: "#000000")
- stat.line.width - Bracket line width (UI: "Line Width")
- stat.bracket.style - Bracket style, capped or flat (UI: "Bracket Style")
- stat.step.increase - Vertical spacing between stacked brackets (UI: "Bracket Spacing")
- stat.text.bump - Offset of the significance text above the bracket (UI: "Text Offset")
- stat.bracket.inset - Horizontal inset of the brackets (UI: "Bracket Inset")
- stat.per.facet - Compute statistics independently per facet panel (UI: "Per Facet Panel")

Parameters controlling additional functionality

The following parameters implementing new functionality or controlling plotly-specific features are also available:

- boxplot.width - Width of boxplot (UI: "Boxplot Width", default: 0.8)
- show.outliers - Show outlier points (UI: "Show Outliers", default: TRUE)
- axis.title.font.size - Axis title font size (UI: "Axis title size", default: 18)
- title.font.size - Plot title font size (UI: "Title Size", default: 26)
- title.font.family - Font family for title text (UI: "Title Font", default: "Arial")
- title.font.color - Color for plot title (UI: "Title Color", default: "#000000")
- axis.title.font.size - Axis title font size (UI: "Axis Title Size", default: 18)
- axis.title.font.color - Axis title font color (UI: "Axis Title Color", default: "#000000")
- axis.title.font.family - Axis title font family (UI: "Axis Title Font", default: "Arial")
- axis.showline - Show axis border lines (UI: "Show axis lines", default: TRUE)
- axis.mirror - Mirror axis lines on opposite side (UI: "Mirror axis lines", default: TRUE)
- show.grid.x - Show X-axis major gridlines (UI: "Show X major gridlines", default: TRUE)
- show.grid.y - Show Y-axis major gridlines (UI: "Show Y major gridlines", default: TRUE)
- axis.linecolor - Color of axis lines (UI: "Axis line color", default: "black")
- axis.linewidth - Width of axis lines (UI: "Axis line width", default: 0.5)
- axis.tickfont.size - Size of tick labels (UI: "Tick label size", default: 12)
- axis.tickfont.color - Color of tick labels (UI: "Tick label color", default: "black")
- axis.tickfont.family - Font family for tick labels (UI: "Tick label font", default: "Arial")
- axis.tickangle.x - Rotation angle for X-axis tick labels (UI: "X-axis tick label angle", default: 0)
- axis.tickangle.y - Rotation angle for Y-axis tick labels (UI: "Y-axis tick label angle", default: 0)
- axis.ticks - Position of tick marks (UI: "Tick position", default: "outside")
- axis.tickcolor - Color of tick marks (UI: "Tick mark color", default: "black")
- axis.ticklen - Length of tick marks (UI: "Tick mark length", default: 5)
- axis.tickwidth - Width of tick marks (UI: "Tick mark width", default: 1)

- `hline.intercepts` - Y-coordinates for horizontal reference lines (UI: "Y-intercepts", default: "")
- `hline.colors` - Colors for horizontal lines (UI: "Colors", default: "#000000")
- `hline.widths` - Widths for horizontal lines (UI: "Widths", default: "1")
- `hline.linetypes` - Line types for horizontal lines (UI: "Line types", default: "dashed")
- `hline.opacities` - Opacities for horizontal lines (UI: "Opacities (0-1)", default: "1")
- `vline.intercepts` - X-coordinates for vertical reference lines (UI: "X-intercepts", default: "")
- `vline.colors` - Colors for vertical lines (UI: "Colors", default: "#000000")
- `vline.widths` - Widths for vertical lines (UI: "Widths", default: "1")
- `vline.linetypes` - Line types for vertical lines (UI: "Line types", default: "dashed")
- `vline.opacities` - Opacities for vertical lines (UI: "Opacities (0-1)", default: "1")
- `abline.slopes` - Slopes for diagonal reference lines (UI: "Slopes", default: "")
- `abline.intercepts` - Y-intercepts for diagonal lines (UI: "Y-intercepts", default: "")
- `abline.colors` - Colors for diagonal lines (UI: "Colors", default: "#000000")
- `abline.widths` - Widths for diagonal lines (UI: "Widths", default: "1")
- `abline.linetypes` - Line types for diagonal lines (UI: "Line types", default: "dashed")
- `abline.opacities` - Opacities for diagonal lines (UI: "Opacities (0-1)", default: "1")

Author(s)

Jacob Martin, Jared Andrews

See Also

`plotthis::BoxPlot()`, `organize_inputs()`, `plotthis_BoxPlotOutputUI()`, `plotthis_BoxPlotServer()`, `plotthis_BoxPlotApp()`

Examples

```
library(VizModules)
data(mtcars)
plotthis_BoxPlotInputsUI("BoxPlot", mtcars)
```

`plotthis_BoxPlotOutputUI`

Output UI components for the BoxPlot module

Description

This should be placed in the UI where the plot should be shown.

Usage

```
plotthis_BoxPlotOutputUI(id, resizable = TRUE)
```

Arguments

id	The ID for the Shiny module.
resizable	Logical; when TRUE (the default) the plot output is wrapped in shinyjqui::jqui_resizable() so it can be resized by dragging. Set to FALSE when embedding the output in a container that already provides resizing.

Value

A Shiny plotlyOutput for the boxPlot

Author(s)

Jacob Martin

---

plotthis_BoxPlotServer

Server logic for BoxPlot module

Description

Server logic for BoxPlot module

Usage

```
plotthis_BoxPlotServer(
  id,
  data,
  hide.inputs = NULL,
  hide.tabs = NULL,
  defaults = NULL
)
```

Arguments

id	The ID for the Shiny module.
data	A reactive containing the data frame to plot.
hide.inputs	A character vector of input IDs to hide. These will still be initialized and their values passed to the plot function, but the user will not be able to see/adjust them in the UI.
hide.tabs	A character vector of tab names to hide. Inputs in these tabs will still be initialized and their values passed to the plot function, but the user will not be able to see/adjust them in the UI.

defaults	A named list of default values for the inputs. When the reset button is clicked, inputs are reset to these values rather than hardcoded fallbacks. Typically the same list passed to the corresponding UI function.

Value

The moduleServer function for the BoxPlot module.

Author(s)

Jacob Martin, Jared Andrews

---

plotthis_DensityPlotApp

Create an example Modular DensityPlot Shiny Application

Description

This function generates a Shiny application with modular density plot components. The app features a **Data Import** section for uploading data, a **Data Table** for filtering the active dataset, and a **Plot** area for configuring and displaying an interactive density plot.

Usage

```
plotthis_DensityPlotApp(
  data_list = NULL,
  defaults = NULL,
  hide.inputs = NULL,
  hide.tabs = NULL
)
```

Arguments

data_list	An optional named list of data frames. If NULL (the default), list("demographics" = example_demographics) is used as example data.
defaults	A named list of input IDs and their default values to apply on startup.
hide.inputs	A character vector of input IDs to hide. Their values are still initialized and used, but the controls are not shown in the UI.
hide.tabs	A character vector of tab names to hide. Inputs in these tabs are still initialized and used, but the controls are not shown in the UI.

Details

When data_list is not provided (or NULL), the app launches with example_demographics as an example dataset. Uploaded data files are added to the available datasets and can be selected for plotting. If an uploaded file shares a name with an existing dataset, the existing one is overwritten with a warning.

This is a convenience wrapper around createModuleApp().

Value

A Shiny app object.

Author(s)

Jacob Martin, Jared Andrews

Examples

```
library(VizModules)
# Launch with default example data:
app <- plotthis_DensityPlotApp()
if (interactive()) runApp(app)

# Launch with custom data:
app2 <- plotthis_DensityPlotApp(list("demographics" = example_demographics))
if (interactive()) runApp(app2)
```

---

plotthis_DensityPlotInputsUI

Input UI components for the DensityPlot module

Description

This should be placed in the UI where the inputs should be shown, with an `id` that matches the `id` used in the `plotthis_DensityPlotServer()` and `plotthis_DensityPlotOutputUI()` functions.

Usage

```
plotthis_DensityPlotInputsUI(
  id,
  data,
  defaults = NULL,
  title = NULL,
  columns = 2
)
```

Arguments

`id`	The ID for the Shiny module.
`data`	The data frame used for plot generation.
`defaults`	A named list of default values for the inputs.
`title`	An optional title for the UI grid.
`columns`	Number of columns for the UI grid.

Details

The user inputs for this module are separated from the outputs to allow for more flexible UI design.

The inputs will automatically be organized into a grid layout via the `organize_inputs()` function, with `columns` controlling the number of columns in the grid.

Defaults can be set for each input by providing a named list of values to the `defaults` argument. Nearly all parameters for `plotthis::DensityPlot()` can be set via these inputs, so see the help for that function for an exhaustive list.

Value

A Shiny tagList containing the UI elements

Plot parameters not implemented or with altered functionality

The following `plotthis::DensityPlot()` parameters are not available via UI inputs:

- `xlab` - X-axis label (plotly allows interactive editing)
- `ylab` - Y-axis label (plotly allows interactive editing)
- `title` - Plot title (plotly allows interactive editing)
- `subtitle` - Plot subtitle (not supported in plotly)
- `legend.position` - Legend positioning (plotly allows interactive repositioning)
- `group_by_sep` - Separator for group columns (not applicable in UI context)
- `group_name` - Group legend name (handled by plotly)
- `xtrans` - X-axis transformation (not implemented in UI)
- `ytrans` - Y-axis transformation (not implemented in UI)
- `split_by` - Split variable (returns a patchwork object, not supported in plotly), use `facet_by` instead
- `split_by_sep` - Only applies if `split_by` is used
- `theme` - ggplot2 theme (not applicable in plotly)
- `theme_args` - Theme arguments (not applicable in plotly)
- `palette` - Managed internally via the palette selection UI
- `expand` - Axis expansion (not implemented)
- `seed` - Random seed (not applicable)
- `combine` - Only applies if `split_by` is used
- `nrow` - Only applies if `split_by` is used
- `ncol` - Only applies if `split_by` is used
- `byrow` - Only applies if `split_by` is used
- `axes` - Only applies if `split_by` is used
- `axis_titles` - Only applies if `split_by` is used
- `guides` - Only applies if `split_by` is used
- `design` - Only applies if `split_by` is used

- `palreverse` - Reverse the color palette (not implemented)
- `aspect.ratio` - Aspect ratio control (handled by plotly layout)
- `keep_empty` - Keep empty factor levels (not implemented)
- `keep_na` - Keep NA values (not implemented)
- `legend.direction` - Managed position of legend however this can be handled via plotly

Plot parameters and defaults

The following `plotthis::DensityPlot()` parameters can be accessed via UI inputs and/or the `defaults` argument:

- `x` - X-axis variable (UI: "X Data", default: 2nd numeric variable)
- `group_by` - Grouping variable (UI: "Group By", default: "")
- `flip` - Flip/swap the x and y axes (UI: "Rotate (swap X/Y)", default: FALSE)
- `position` - Position adjustment (UI: "Position", default: "identity")
- `alpha` - Density fill transparency (UI: "Plot Alpha", default: 0.5)
- `add_bars` - Add rug plot (UI: "Add Rug Plot", default: FALSE)
- `bar_height` - Rug bar height (UI: "Rug Bar Height", default: 0.04)
- `bar_alpha` - Rug bar transparency (UI: "Rug Bar Alpha", default: 1)
- `bar_width` - Rug bar width (UI: "Rug Bar Width", default: 1)
- `facet_by` - Faceting variable (UI: "Facet By", default: "")
- `facet_scales` - Facet scale behavior (UI: "Facet Scale", default: "fixed")
- `facet_ncol` - Number of facet columns (UI: "Columns", default: NULL)
- `facet_nrow` - Number of facet rows (UI: "Rows", default: NULL)
- `facet_byrow` - Facet ordering direction (UI: "Facet by Row", default: TRUE)
- `palcolor` - Custom color values (UI: palette picker, derived from palette)

Parameters controlling additional functionality

The following parameters implementing new functionality or controlling plotly-specific features are also available:

- `title.font.size` - Plot title font size (UI: "Title Size", default: 26)
- `title.font.family` - Font family for title text (UI: "Title Font", default: "Arial")
- `title.font.color` - Color for plot title (UI: "Title Color", default: "#000000")
- `axis.title.font.size` - Axis title font size (UI: "Axis Title Size", default: 18)
- `axis.title.font.color` - Axis title font color (UI: "Axis Title Color", default: "#000000")
- `axis.title.font.family` - Axis title font family (UI: "Axis Title Font", default: "Arial")
- `axis.showline` - Show axis border lines (UI: "Show axis lines", default: TRUE)
- `axis.mirror` - Mirror axis lines on opposite side (UI: "Mirror axis lines", default: TRUE)
- `show.grid.x` - Show X-axis major gridlines (UI: "Show X major gridlines", default: TRUE)

- `show.grid.y` - Show Y-axis major gridlines (UI: "Show Y major gridlines", default: TRUE)
- `axis.linecolor` - Color of axis lines (UI: "Axis line color", default: "black")
- `axis.linewidth` - Width of axis lines (UI: "Axis line width", default: 0.5)
- `axis.tickfont.size` - Size of tick labels (UI: "Tick label size", default: 12)
- `axis.tickfont.color` - Color of tick labels (UI: "Tick label color", default: "black")
- `axis.tickfont.family` - Font family for tick labels (UI: "Tick label font", default: "Arial")
- `axis.tickangle.x` - Rotation angle for X-axis tick labels (UI: "X-axis tick label angle", default: 0)
- `axis.tickangle.y` - Rotation angle for Y-axis tick labels (UI: "Y-axis tick label angle", default: 0)
- `axis.ticks` - Position of tick marks (UI: "Tick position", default: "outside")
- `axis.tickcolor` - Color of tick marks (UI: "Tick mark color", default: "black")
- `axis.ticklen` - Length of tick marks (UI: "Tick mark length", default: 5)
- `axis.tickwidth` - Width of tick marks (UI: "Tick mark width", default: 1)
- `hline.intercepts` - Y-coordinates for horizontal reference lines (UI: "Y-intercepts", default: "")
- `hline.colors` - Colors for horizontal lines (UI: "Colors", default: "#000000")
- `hline.widths` - Widths for horizontal lines (UI: "Widths", default: "1")
- `hline.linetypes` - Line types for horizontal lines (UI: "Line types", default: "dashed")
- `hline.opacities` - Opacities for horizontal lines (UI: "Opacities (0-1)", default: "1")
- `vline.intercepts` - X-coordinates for vertical reference lines (UI: "X-intercepts", default: "")
- `vline.colors` - Colors for vertical lines (UI: "Colors", default: "#000000")
- `vline.widths` - Widths for vertical lines (UI: "Widths", default: "1")
- `vline.linetypes` - Line types for vertical lines (UI: "Line types", default: "dashed")
- `vline.opacities` - Opacities for vertical lines (UI: "Opacities (0-1)", default: "1")
- `abline.slopes` - Slopes for diagonal reference lines (UI: "Slopes", default: "")
- `abline.intercepts` - Y-intercepts for diagonal lines (UI: "Y-intercepts", default: "")
- `abline.colors` - Colors for diagonal lines (UI: "Colors", default: "#000000")
- `abline.widths` - Widths for diagonal lines (UI: "Widths", default: "1")
- `abline.linetypes` - Line types for diagonal lines (UI: "Line types", default: "dashed")
- `abline.opacities` - Opacities for diagonal lines (UI: "Opacities (0-1)", default: "1")

Author(s)

Jacob Martin, Jared Andrews

See Also

`plotthis::DensityPlot()`, `organize_inputs()`, `plotthis_DensityPlotOutputUI()`, `plotthis_DensityPlotServer()`
`plotthis_DensityPlotApp()`

Examples

```
library(VizModules)
data(mtcars)
plotthis_DensityPlotInputsUI("densityPlot", mtcars)
```

`plotthis_DensityPlotOutputUI` *Output UI components for the DensityPlot module*

Description

This should be placed in the UI where the plot should be shown.

Usage

```
plotthis_DensityPlotOutputUI(id, resizable = TRUE)
```

Arguments

`id`	The ID for the Shiny module.
`resizable`	Logical; when `TRUE` (the default) the plot output is wrapped in `shinyjqui::jqui_resizable()` so it can be resized by dragging. Set to `FALSE` when embedding the output in a container that already provides resizing.

Value

A Shiny plotlyOutput for the DensityPlot

Author(s)

Jacob Martin

`plotthis_DensityPlotServer` *Density Plot Server Module*

Description

Server-side logic for the density plot module. This function manages reactive data processing, dynamic UI generation for color palettes, and the rendering of interactive Plotly density plots.

Usage

```
plotthis_DensityPlotServer(
  id,
  data,
  hide.inputs = NULL,
  hide.tabs = NULL,
  defaults = NULL
)
```

Arguments

id	character unique ID for the shiny namespace.
data	reactive A reactive expression returning a data frame to be plotted.
hide.inputs	character vector of input IDs to hide in the UI. Default is NULL.
hide.tabs	character vector of tab names to hide within the module. Default is NULL.
defaults	A named list of default values for the inputs. When the reset button is clicked, inputs are reset to these values rather than hardcoded fallbacks. Typically the same list passed to the corresponding UI function.

Value

The moduleServer function for the DensityPlot module.

Author(s)

Jacob Martin, Jared Andrews

---

plotthis_DotPlotApp *Create an example Modular DotPlot Shiny Application*

Description

This function generates a Shiny application with modular plotthis::DotPlot() components. The app features a **Data Import** section for uploading data, a **Data Table** for filtering the active dataset, and a **Plot** area for configuring and displaying an interactive dot plot.

Usage

```
plotthis_DotPlotApp(
  data_list = NULL,
  defaults = NULL,
  hide.inputs = NULL,
  hide.tabs = NULL
)
```

Arguments

data_list	An optional named list of data frames. If NULL (the default), list("markers" = example_markers) is used as example data.
defaults	A named list of input IDs and their default values to apply on startup.
hide.inputs	A character vector of input IDs to hide. Their values are still initialized and used, but the controls are not shown in the UI.
hide.tabs	A character vector of tab names to hide. Inputs in these tabs are still initialized and used, but the controls are not shown in the UI.

Details

When data_list is not provided (or NULL), the app launches with example_markers as an example dataset. Uploaded data files are added to the available datasets and can be selected for plotting. If an uploaded file shares a name with an existing dataset, the existing one is overwritten with a warning.

This is a convenience wrapper around createModuleApp().

Value

A Shiny app object.

Author(s)

Jacob Martin, Jared Andrews

Examples

```
library(VizModules)
# Launch with default example data:
app <- plotthis_DotPlotApp()
if (interactive()) runApp(app)

# Launch with custom data:
app2 <- plotthis_DotPlotApp(list("markers" = example_markers))
if (interactive()) runApp(app2)
```

---

plotthis_DotPlotInputsUI

Input UI components for the DotPlot module

Description

This should be placed in the UI where the inputs should be shown, with an id that matches the id used in the plotthis_DotPlotServer() and plotthis_DotPlotOutputUI() functions.

Usage

```
plotthis_DotPlotInputsUI(id, data, defaults = NULL, title = NULL, columns = 2)
```

Arguments

id	The ID for the Shiny module.
data	The data frame used for plot generation.
defaults	A named list of default values for the inputs.
title	An optional title for the UI grid.
columns	Number of columns for the UI grid.

Details

The user inputs for this module are separated from the outputs to allow for more flexible UI design.

The inputs will automatically be organized into a grid layout via the `organize_inputs()` function, with `columns` controlling the number of columns in the grid.

Defaults can be set for each input by providing a named list of values to the `defaults` argument. Nearly all parameters for `plotthis::DotPlot()` can be set via these inputs, so see the help for that function for an exhaustive list.

Value

A Shiny tagList containing the UI elements

Plot parameters not implemented or with altered functionality

The following `plotthis::DotPlot()` parameters are not available via UI inputs:

- `xlab` - X-axis label (plotly allows interactive editing)
- `ylab` - Y-axis label (plotly allows interactive editing)
- `title` - Plot title (plotly allows interactive editing)
- `subtitle` - Plot subtitle (not supported in plotly)
- `aspect.ratio` - Aspect ratio control (handled by plotly layout)
- `legend.position` - Legend positioning (plotly allows interactive repositioning)
- `legend.direction` - Legend orientation (plotly allows interactive adjustment)
- `x_sep` - Separator for multiple x columns (not yet implemented)
- `y_sep` - Separator for multiple y columns (not yet implemented)
- `split_by` - Split variable for separate plots (doesn't work with plotly; `facet_by` available instead)
- `split_by_sep` - Separator for multiple `split_by` columns (`split_by` not used in module)
- `size_name` - Size legend name (plotly allows interactive editing)
- `fill_name` - Fill legend name (not yet implemented)
- `fill_cutoff_name` - Fill cutoff legend name (not yet implemented)
- `theme` - ggplot2 theme (managed internally)
- `theme_args` - Theme arguments (not yet implemented)
- `palcolor` - Managed internally via the palette selection UI

- `border_alpha` - Dot border transparency (not exposed; uses the plotthis default of 1)
- `add_bg` - Add background stripes/shading (not yet implemented)
- `bg_palette` - Background palette (not yet implemented)
- `bg_palcolor` - Background palette colors (not yet implemented)
- `bg_alpha` - Background alpha (not yet implemented)
- `bg_direction` - Background stripe direction (not yet implemented)
- `x_text_angle` - X-axis text angle (handled by axis.tickangle.x)
- `keep_empty` - Keep empty factor levels (not yet implemented)
- `keep_na` - Keep NA values (not yet implemented)
- `combine` - Combine multiple plots (not applicable as `split_by` is not implemented)
- `seed` - Random seed (not applicable)
- `nrow` - Only applies if `split_by` is used with combine (`split_by` not used in module)
- `ncol` - Only applies if `split_by` is used with combine (`split_by` not used in module)
- `byrow` - Only applies if `split_by` is used with combine (`split_by` not used in module)
- `axes` - Only applies if `split_by` is used with combine (`split_by` not used in module)
- `axis_titles` - Only applies if `split_by` is used with combine (`split_by` not used in module)
- `guides` - Only applies if `split_by` is used with combine (`split_by` not used in module)
- `design` - Only applies if `split_by` is used with combine (`split_by` not used in module)

Plot parameters and defaults

The following `plotthis::DotPlot()` and custom parameters can be accessed via UI inputs and/or the `defaults` argument:

- `x` - X-axis variable (UI: "X Values", default: 2nd categorical variable)
- `y` - Y-axis variable (UI: "Y Values", default: 3rd categorical variable)
- `size_by` - Numeric column mapped to dot size (UI: "Size By", default: "" = count)
- `size_min` - Minimum dot size (UI: "Min Dot Size", default: 1)
- `size_max` - Maximum dot size (UI: "Max Dot Size", default: 6)
- `fill_by` - Numeric column mapped to dot fill (UI: "Fill By", default: "")
- `fill_cutoff` - Cutoff applied to the fill column (UI: "Fill Cutoff", default: NA)
- `fill_cutoff_direction` - Direction of the fill cutoff (UI: "Fill Cutoff Direction", default: "<"); combined with `fill_cutoff` into a `plotthis` expression such as `"< 18"`.
- `flip` - Flip the x and y axes (UI: "Rotate (swap X/Y)", default: FALSE)
- `facet_by` - Faceting variable (UI: "Facet By", default: "")
- `facet_scales` - Facet scale behavior (UI: "Facet Scale", default: "fixed")
- `facet_ncol` - Number of facet columns (UI: "Columns", default: NULL)
- `facet_nrow` - Number of facet rows (UI: "Rows", default: NULL)
- `facet_byrow` - Facet ordering direction (UI: "Facet by Row", default: TRUE)

- `palette` - Continuous fill palette (UI: "Color Palette", default: "Spectral")
- `palreverse` - Reverse the color palette (UI: "Reverse palette", default: FALSE)
- `alpha` - Dot fill transparency (UI: "Alpha", default: 1)
- `border_color` - Dot border color; only constant colors are supported (UI: "Border Color", default: "black")
- `border_size` - Dot border stroke width (UI: "Border Size", default: 0.5)
- `lower_quantile` - Lower quantile for the continuous fill color scale (UI: "Lower Quantile", default: 0)
- `upper_quantile` - Upper quantile for the continuous fill color scale (UI: "Upper Quantile", default: 1)
- `lower_cutoff` - Explicit lower cutoff for the continuous fill color scale (UI: "Lower Cutoff", default: NA); overrides `lower_quantile` when set
- `upper_cutoff` - Explicit upper cutoff for the continuous fill color scale (UI: "Upper Cutoff", default: NA); overrides `upper_quantile` when set
- `size.legend.x` - Custom size-legend x position (UI: "Size Legend X Position", default: 1.04); nudges the manual size legend (drawn when `size.by` is set) along the x-axis.
- `size.legend.y` - Custom size-legend y position (UI: "Size Legend Y Position", default: 0.35); nudges the manual size legend (drawn when `size.by` is set) along the y-axis.

Author(s)

Jacob Martin, Jared Andrews

See Also

`plotthis::DotPlot()`, `organize_inputs()`, `plotthis_DotPlotOutputUI()`, `plotthis_DotPlotServer()`, `plotthis_DotPlotApp()`

Examples

```
library(VizModules)
data(mtcars)
plotthis_DotPlotInputsUI("DotPlot", mtcars)
```

---

plotthis_DotPlotOutputUI

Output UI components for the DotPlot module

Description

This should be placed in the UI where the plot should be shown.

Usage

```
plotthis_DotPlotOutputUI(id, resizable = TRUE)
```

Arguments

id	The ID for the Shiny module.
resizable	Logical; when TRUE (the default) the plot output is wrapped in shinyjqui::jqui_resizable() so it can be resized by dragging. Set to FALSE when embedding the output in a container that already provides resizing.

Value

A Shiny plotlyOutput for the DotPlot

Author(s)

Jacob Martin, Jared Andrews

---

plotthis_DotPlotServer

Server logic for DotPlot module

Description

Server logic for DotPlot module

Usage

```
plotthis_DotPlotServer(
  id,
  data,
  hide.inputs = NULL,
  hide.tabs = NULL,
  defaults = NULL
)
```

Arguments

id	The ID for the Shiny module.
data	A reactive containing the data frame to plot.
hide.inputs	A character vector of input IDs to hide. These will still be initialized and their values passed to the plot function, but the user will not be able to see/adjust them in the UI.
hide.tabs	A character vector of tab names to hide. Inputs in these tabs will still be initialized and their values passed to the plot function, but the user will not be able to see/adjust them in the UI.
defaults	A named list of default values for the inputs. When the reset button is clicked, inputs are reset to these values rather than hardcoded fallbacks. Typically the same list passed to the corresponding UI function.

Value

The `moduleServer` function for the DotPlot module.

Author(s)

Jacob Martin, Jared Andrews

`plotthis_HistogramApp` *Create an example Modular Histogram Shiny Application*

Description

This function generates a Shiny application with modular histogram components. The app features a **Data Import** section for uploading data, a **Data Table** for filtering the active dataset, and a **Plot** area for configuring and displaying an interactive histogram.

Usage

```
plotthis_HistogramApp(
  data_list = NULL,
  defaults = NULL,
  hide.inputs = NULL,
  hide.tabs = NULL
)
```

Arguments

`data_list`	An optional named list of data frames. If `NULL` (the default), `list("demographics" = example_demographics)` is used as example data.
`defaults`	A named list of input IDs and their default values to apply on startup.
`hide.inputs`	A character vector of input IDs to hide. Their values are still initialized and used, but the controls are not shown in the UI.
`hide.tabs`	A character vector of tab names to hide. Inputs in these tabs are still initialized and used, but the controls are not shown in the UI.

Details

When `data_list` is not provided (or `NULL`), the app launches with `example_demographics` as an example dataset. Uploaded data files are added to the available datasets and can be selected for plotting. If an uploaded file shares a name with an existing dataset, the existing one is overwritten with a warning.

This is a convenience wrapper around `createModuleApp()`.

Value

A Shiny app object.

Author(s)

Jacob Martin, Jared Andrews

Examples

```
library(VizModules)
# Launch with default example data:
app <- plotthis_HistogramApp()
if (interactive()) runApp(app)

# Launch with custom data:
app2 <- plotthis_HistogramApp(list("demographics" = example_demographics))
if (interactive()) runApp(app2)
```

---

plotthis_HistogramInputsUI

Input UI components for the Histogram module

Description

This should be placed in the UI where the inputs should be shown, with an `id` that matches the `id` used in the `plotthis_HistogramServer()` and `plotthis_HistogramOutputUI()` functions.

Usage

```
plotthis_HistogramInputsUI(
  id,
  data,
  defaults = NULL,
  title = NULL,
  columns = 2
)
```

Arguments

`id`	The ID for the Shiny module.
`data`	The data frame used for plot generation.
`defaults`	A named list of default values for the inputs.
`title`	An optional title for the UI grid.
`columns`	Number of columns for the UI grid.

Details

The user inputs for this module are separated from the outputs to allow for more flexible UI design.

The inputs will automatically be organized into a grid layout via the `organize_inputs()` function, with `columns` controlling the number of columns in the grid.

Defaults can be set for each input by providing a named list of values to the `defaults` argument. Nearly all parameters for `plotthis::Histogram()` can be set via these inputs, so see the help for that function for an exhaustive list.

Value

A Shiny tagList containing the UI elements

Plot parameters not implemented or with altered functionality

The following `plotthis::Histogram()` parameters are not available via UI inputs:

- `xlab` - X-axis label (plotly allows interactive editing)
- `ylab` - Y-axis label (plotly allows interactive editing)
- `title` - Plot title (plotly allows interactive editing)
- `subtitle` - Plot subtitle (not supported in plotly)
- `legend.position` - Legend positioning (plotly allows interactive repositioning)
- `group_by_sep` - Separator for group columns (not applicable in UI context)
- `group_name` - Group legend name (handled by plotly)
- `xtrans` - X-axis transformation (not implemented in UI)
- `ytrans` - Y-axis transformation (not implemented in UI)
- `split_by` - Split variable (returns a patchwork object, not supported in plotly), use `facet_by` instead
- `split_by_sep` - Only applies if `split_by` is used
- `theme` - ggplot2 theme (not applicable in plotly)
- `theme_args` - Theme arguments (not applicable in plotly)
- `palette` - Managed internally via the palette selection UI
- `expand` - Axis expansion (not implemented)
- `seed` - Random seed (not applicable)
- `combine` - Only applies if `split_by` is used
- `nrow` - Only applies if `split_by` is used
- `ncol` - Only applies if `split_by` is used
- `byrow` - Only applies if `split_by` is used
- `axes` - Only applies if `split_by` is used
- `axis_titles` - Only applies if `split_by` is used
- `guides` - Only applies if `split_by` is used
- `design` - Only applies if `split_by` is used

- palreverse - Reverse the color palette (not implemented)
- aspect.ratio - Aspect ratio control (handled by plotly layout)
- keep_empty - Keep empty factor levels (not implemented)
- keep_na - Keep NA values (not implemented)
- legend.direction - Managed position of legend however this can be handled via plotly

Plot parameters and defaults

The following plotthis::Histogram() parameters can be accessed via UI inputs and/or the defaults argument:

- x - X-axis variable (UI: "X Data", default: 2nd numeric variable)
- group_by - Grouping variable (UI: "Group By", default: "")
- flip - Flip/swap the x and y axes (UI: "Rotate (swap X/Y)", default: FALSE)
- bins - Number of bins (UI: "Number of Bins", default: NA)
- binwidth - Width of bins (UI: "Bin Width", default: NA)
- use_trend - Show only trend line (UI: "Trend Line Only", default: FALSE)
- add_trend - Add trend line to histogram (UI: "Add Trend to Histogram", default: FALSE)
- trend_skip_zero - Skip zero values in trend (UI: "Skip Zero Values", default: FALSE)
- trend_alpha - Trend line transparency (UI: "Trend Line Alpha", default: 1)
- trend_linewidth - Trend line width (UI: "Trend Line Width", default: 0.8)
- trend_pt_size - Trend point size (UI: "Trend Point Size", default: 1.5)
- position - Position adjustment (UI: "Position", default: "identity")
- alpha - Histogram fill transparency (UI: "Plot Alpha", default: 1)
- add_bars - Add rug plot (UI: "Add Rug Plot", default: FALSE)
- bar_height - Rug bar height (UI: "Rug Bar Height", default: 0.04)
- bar_alpha - Rug bar transparency (UI: "Rug Bar Alpha", default: 1)
- bar_width - Rug bar width (UI: "Rug Bar Width", default: 1)
- facet_by - Faceting variable (UI: "Facet By", default: "")
- facet_scales - Facet scale behavior (UI: "Facet Scale", default: "fixed")
- facet_ncol - Number of facet columns (UI: "Columns", default: NULL)
- facet_nrow - Number of facet rows (UI: "Rows", default: NULL)
- facet_byrow - Facet ordering direction (UI: "Facet by Row", default: TRUE)
- palcolor - Custom color values (UI: palette picker, derived from palette)

Parameters controlling additional functionality

The following parameters implementing new functionality or controlling plotly-specific features are also available:

- `title.font.size` - Plot title font size (UI: "Title Size", default: 26)
- `title.font.family` - Font family for title text (UI: "Title Font", default: "Arial")
- `title.font.color` - Color for plot title (UI: "Title Color", default: "#000000")
- `axis.title.font.size` - Axis title font size (UI: "Axis Title Size", default: 18)
- `axis.title.font.color` - Axis title font color (UI: "Axis Title Color", default: "#000000")
- `axis.title.font.family` - Axis title font family (UI: "Axis Title Font", default: "Arial")
- `axis.showline` - Show axis border lines (UI: "Show axis lines", default: TRUE)
- `axis.mirror` - Mirror axis lines on opposite side (UI: "Mirror axis lines", default: TRUE)
- `show.grid.x` - Show X-axis major gridlines (UI: "Show X major gridlines", default: TRUE)
- `show.grid.y` - Show Y-axis major gridlines (UI: "Show Y major gridlines", default: TRUE)
- `axis.linecolor` - Color of axis lines (UI: "Axis line color", default: "black")
- `axis.linewidth` - Width of axis lines (UI: "Axis line width", default: 0.5)
- `axis.tickfont.size` - Size of tick labels (UI: "Tick label size", default: 12)
- `axis.tickfont.color` - Color of tick labels (UI: "Tick label color", default: "black")
- `axis.tickfont.family` - Font family for tick labels (UI: "Tick label font", default: "Arial")
- `axis.tickangle.x` - Rotation angle for X-axis tick labels (UI: "X-axis tick label angle", default: 0)
- `axis.tickangle.y` - Rotation angle for Y-axis tick labels (UI: "Y-axis tick label angle", default: 0)
- `axis.ticks` - Position of tick marks (UI: "Tick position", default: "outside")
- `axis.tickcolor` - Color of tick marks (UI: "Tick mark color", default: "black")
- `axis.ticklen` - Length of tick marks (UI: "Tick mark length", default: 5)
- `axis.tickwidth` - Width of tick marks (UI: "Tick mark width", default: 1)
- `hline.intercepts` - Y-coordinates for horizontal reference lines (UI: "Y-intercepts", default: "")
- `hline.colors` - Colors for horizontal lines (UI: "Colors", default: "#000000")
- `hline.widths` - Widths for horizontal lines (UI: "Widths", default: "1")
- `hline.linetypes` - Line types for horizontal lines (UI: "Line types", default: "dashed")
- `hline.opacities` - Opacities for horizontal lines (UI: "Opacities (0-1)", default: "1")
- `vline.intercepts` - X-coordinates for vertical reference lines (UI: "X-intercepts", default: "")
- `vline.colors` - Colors for vertical lines (UI: "Colors", default: "#000000")
- `vline.widths` - Widths for vertical lines (UI: "Widths", default: "1")
- `vline.linetypes` - Line types for vertical lines (UI: "Line types", default: "dashed")
- `vline.opacities` - Opacities for vertical lines (UI: "Opacities (0-1)", default: "1")

- `abline.slopes` - Slopes for diagonal reference lines (UI: "Slopes", default: "")
- `abline.intercepts` - Y-intercepts for diagonal lines (UI: "Y-intercepts", default: "")
- `abline.colors` - Colors for diagonal lines (UI: "Colors", default: "#000000")
- `abline.widths` - Widths for diagonal lines (UI: "Widths", default: "1")
- `abline.linetypes` - Line types for diagonal lines (UI: "Line types", default: "dashed")
- `abline.opacities` - Opacities for diagonal lines (UI: "Opacities (0-1)", default: "1")

Author(s)

Jacob Martin, Jared Andrews

See Also

`plotthis::Histogram()`, `organize_inputs()`, `plotthis_HistogramOutputUI()`, `plotthis_HistogramServer()`, `plotthis_HistogramApp()`

Examples

```
library(VizModules)
data(mtcars)
plotthis_HistogramInputsUI("histogram", mtcars)
```

`plotthis_HistogramOutputUI` *Output UI components for the histogramPlot module*

Description

This should be placed in the UI where the plot should be shown.

Usage

```
plotthis_HistogramOutputUI(id, resizable = TRUE)
```

Arguments

`id`	The ID for the Shiny module.
`resizable`	Logical; when `TRUE` (the default) the plot output is wrapped in `shinyjqui::jqui_resizable()` so it can be resized by dragging. Set to `FALSE` when embedding the output in a container that already provides resizing.

Value

A Shiny plotlyOutput for the histogramPlot

Author(s)

Jacob Martin

plotthis_HistogramServer *Histogram Plot Server Module*

Description

Server-side logic for the histogram plot module. This function manages reactive data processing, dynamic UI generation for color palettes, and the rendering of interactive Plotly histograms.

Usage

```
plotthis_HistogramServer(
  id,
  data,
  hide.inputs = NULL,
  hide.tabs = NULL,
  defaults = NULL
)
```

Arguments

id	character unique ID for the shiny namespace.
data	reactive A reactive expression returning a data frame to be plotted.
hide.inputs	character vector of input IDs to hide in the UI. Default is NULL.
hide.tabs	character vector of tab names to hide within the module. Default is NULL.
defaults	A named list of default values for the inputs. When the reset button is clicked, inputs are reset to these values rather than hardcoded fallbacks. Typically the same list passed to the corresponding UI function.

Value

The moduleServer function for the Histogram module.

Author(s)

Jacob Martin, Jared Andrews

plotthis_SplitBarPlotApp *Create an example Modular SplitBarPlot Shiny Application*

Description

This function generates a Shiny application with modular `plotthis::SplitBarPlot()` components. The app features a **Data Import** section for uploading data, a **Data Table** for filtering the active dataset, and a **Plot** area for configuring and displaying an interactive split bar plot.

Usage

```
plotthis_SplitBarPlotApp(
  data_list = NULL,
  defaults = NULL,
  hide.inputs = NULL,
  hide.tabs = NULL
)
```

Arguments

`data_list`	An optional named list of data frames. If `NULL` (the default), `list("Bar" = example_bar)` is used as example data.
`defaults`	A named list of input IDs and their default values to apply on startup.
`hide.inputs`	A character vector of input IDs to hide. Their values are still initialized and used, but the controls are not shown in the UI.
`hide.tabs`	A character vector of tab names to hide. Inputs in these tabs are still initialized and used, but the controls are not shown in the UI.

Details

When `data_list` is not provided (or `NULL`), the app launches with `example_bar` as an example dataset. Uploaded data files are added to the available datasets and can be selected for plotting. If an uploaded file shares a name with an existing dataset, the existing one is overwritten with a warning.

This is a convenience wrapper around `createModuleApp()`.

Value

A Shiny app object.

Author(s)

Jacob Martin, Jared Andrews

Examples

```
library(VizModules)
# Launch with default example data:
app <- plotthis_SplitBarPlotApp()
if (interactive()) runApp(app)

# Launch with custom data:
app2 <- plotthis_SplitBarPlotApp(list("Bar" = example_bar))
if (interactive()) runApp(app2)
```

---

plotthis_SplitBarPlotInputsUI

Input UI components for the SplitBarPlot module

Description

This should be placed in the UI where the inputs should be shown, with an `id` that matches the `id` used in the `plotthis_SplitBarPlotServer()` and `plotthis_SplitBarPlotOutputUI()` functions.

Usage

```
plotthis_SplitBarPlotInputsUI(
  id,
  data,
  defaults = NULL,
  title = NULL,
  columns = 2
)
```

Arguments

`id`	The ID for the Shiny module.
`data`	The data frame used for plot generation.
`defaults`	A named list of default values for the inputs.
`title`	An optional title for the UI grid.
`columns`	Number of columns for the UI grid.

Details

The user inputs for this module are separated from the outputs to allow for more flexible UI design.

The inputs will automatically be organized into a grid layout via the `organize_inputs()` function, with `columns` controlling the number of columns in the grid.

Defaults can be set for each input by providing a named list of values to the `defaults` argument. Nearly all parameters for `plotthis::SplitBarPlot()` can be set via these inputs, so see the help for that function for an exhaustive list.

Value

A Shiny tagList containing the UI elements

Plot parameters not implemented or with altered functionality

The following `plotthis::SplitBarPlot()` parameters are not available via UI inputs:

- `xlab` - X-axis label (plotly allows interactive editing)
- `ylab` - Y-axis label (plotly allows interactive editing)
- `title` - Plot title (plotly allows interactive editing)
- `subtitle` - Plot subtitle (not supported in plotly)
- `aspect.ratio` - Aspect ratio control (handled by plotly layout)
- `legend.position` - Legend positioning (plotly allows interactive repositioning)
- `y_sep` - Separator for y columns (not applicable in UI context)
- `split_by_sep` - Separator for split columns (not applicable in UI context)
- `order_y` - Y-axis ordering rules (handled by default logic)
- `lineheight` - Text line height (not applicable in plotly)
- `max_charwidth` - Maximum character width (not applicable in plotly)
- `fill_by_sep` - Separator for fill columns (not applicable in UI context)
- `fill_name` - Fill legend name (handled by plotly)
- `direction_name` - Direction legend name (not implemented)
- `direction_pos_name` - Positive direction name (not implemented)
- `direction_neg_name` - Negative direction name (not implemented)
- `theme` - ggplot2 theme (not applicable in plotly)
- `theme_args` - Theme arguments (not applicable in plotly)
- `palette` - Managed internally via the palette selection UI
- `keep_empty` - Keep empty values (not implemented)
- `keep_na` - Keep NA values (not implemented)
- `combine` - Only applies if `split_by` is used
- `nrow` - Only applies if `split_by` is used
- `ncol` - Only applies if `split_by` is used
- `byrow` - Only applies if `split_by` is used
- `seed` - Random seed (not applicable)
- `axes` - Only applies if `split_by` is used
- `axis_titles` - Only applies if `split_by` is used
- `guides` - Only applies if `split_by` is used
- `design` - Only applies if `split_by` is used
- `legend.direction` - Managed position of legend however this can be handled via plotly

Plot parameters and defaults

The following plotthis::SplitBarPlot() parameters can be accessed via UI inputs and/or the defaults argument:

- x - X-axis variable (UI: "X values", defaults key: x.data, default: 2nd numeric variable)
- y - Y-axis grouping variable (UI: "Y values", defaults key: y.data, default: 2nd categorical variable)
- fill_by - Fill color variable (UI: "Fill by", default: 2nd variable)
- flip - Flip/swap the x and y axes (UI: "Rotate (swap X/Y)", default: FALSE)
- alpha_by - Variable for alpha transparency (UI: "Alpha by", default: "")
- alpha_reverse - Reverse alpha order (UI: "Alpha reverse", default: FALSE)
- alpha_name - Alpha legend name (UI: "Alpha name", default: "")
- bar_height - Height of bars (UI: "Bar height", default: 0.9)
- facet_by - Faceting variable (UI: "Facet by", default: "")
- facet_scales - Facet scale behavior (UI: "Facet scale", default: "free_y")
- facet_ncol - Number of facet columns (UI: "Facet number of columns", default: NULL)
- facet_nrow - Number of facet rows (UI: "Facet number of rows", default: NULL)
- facet_byrow - Facet ordering direction (UI: "Facet by row", default: TRUE)
- split_by - Split variable (UI: "Split by", default: "")
- x_min - Minimum X-axis value (UI: "X-axis min", default: calculated from data)
- x_max - Maximum X-axis value (UI: "X-axis max", default: calculated from data)
- palcolor - Custom color values (UI: palette picker, derived from palette)
- palreverse - Reverse the color palette (UI: "Reverse palette", default: FALSE)
- lower_quantile - Lower quantile for the continuous fill color scale (UI: "Lower Quantile", default: 0); only affects a numeric fill_by
- upper_quantile - Upper quantile for the continuous fill color scale (UI: "Upper Quantile", default: 1); only affects a numeric fill_by
- lower_cutoff - Explicit lower cutoff for the continuous fill color scale (UI: "Lower Cutoff", default: NA); overrides lower_quantile when set
- upper_cutoff - Explicit upper cutoff for the continuous fill color scale (UI: "Upper Cutoff", default: NA); overrides upper_quantile when set

Parameters controlling additional functionality

The following parameters implementing new functionality or controlling plotly-specific features are also available:

- label.on.y.axis - Show category labels on the Y axis instead of on the plot (UI: "Labels on Y axis", default: FALSE). When enabled, the text position slider is hidden and labels appear as Y-axis tick labels.
- text.position - Position of category labels along the X axis (UI: "Position of category labels", default: 0). Only visible when label.on.y.axis is FALSE.

- `title.font.size` - Plot title font size (UI: "Title Size", default: 26)
- `title.font.family` - Font family for title text (UI: "Title Font", default: "Arial")
- `title.font.color` - Color for plot title (UI: "Title Color", default: "#000000")
- `axis.title.font.size` - Axis title font size (UI: "Axis Title Size", default: 18)
- `axis.title.font.color` - Axis title font color (UI: "Axis Title Color", default: "#000000")
- `axis.title.font.family` - Axis title font family (UI: "Axis Title Font", default: "Arial")
- `axis.showline` - Show axis border lines (UI: "Show axis lines", default: TRUE)
- `axis.mirror` - Mirror axis lines on opposite side (UI: "Mirror axis lines", default: TRUE)
- `show.grid.x` - Show X-axis major gridlines (UI: "Show X major gridlines", default: TRUE)
- `show.grid.y` - Show Y-axis major gridlines (UI: "Show Y major gridlines", default: TRUE)
- `axis.linecolor` - Color of axis lines (UI: "Axis line color", default: "black")
- `axis.linewidth` - Width of axis lines (UI: "Axis line width", default: 0.5)
- `axis.tickfont.size` - Size of tick labels (UI: "Tick label size", default: 12)
- `axis.tickfont.color` - Color of tick labels (UI: "Tick label color", default: "black")
- `axis.tickfont.family` - Font family for tick labels (UI: "Tick label font", default: "Arial")
- `axis.tickangle.x` - Rotation angle for X-axis tick labels (UI: "X-axis tick label angle", default: 0)
- `axis.tickangle.y` - Rotation angle for Y-axis tick labels (UI: "Y-axis tick label angle", default: 0)
- `axis.ticks` - Position of tick marks (UI: "Tick position", default: "outside")
- `axis.tickcolor` - Color of tick marks (UI: "Tick mark color", default: "black")
- `axis.ticklen` - Length of tick marks (UI: "Tick mark length", default: 5)
- `axis.tickwidth` - Width of tick marks (UI: "Tick mark width", default: 1)
- `hline.intercepts` - Y-coordinates for horizontal reference lines (UI: "Y-intercepts", default: "")
- `hline.colors` - Colors for horizontal lines (UI: "Colors", default: "#000000")
- `hline.widths` - Widths for horizontal lines (UI: "Widths", default: "1")
- `hline.linetypes` - Line types for horizontal lines (UI: "Line types", default: "dashed")
- `hline.opacities` - Opacities for horizontal lines (UI: "Opacities (0-1)", default: "1")
- `vline.intercepts` - X-coordinates for vertical reference lines (UI: "X-intercepts", default: "")
- `vline.colors` - Colors for vertical lines (UI: "Colors", default: "#000000")
- `vline.widths` - Widths for vertical lines (UI: "Widths", default: "1")
- `vline.linetypes` - Line types for vertical lines (UI: "Line types", default: "dashed")
- `vline.opacities` - Opacities for vertical lines (UI: "Opacities (0-1)", default: "1")
- `abline.slopes` - Slopes for diagonal reference lines (UI: "Slopes", default: "")
- `abline.intercepts` - Y-intercepts for diagonal lines (UI: "Y-intercepts", default: "")
- `abline.colors` - Colors for diagonal lines (UI: "Colors", default: "#000000")
- `abline.widths` - Widths for diagonal lines (UI: "Widths", default: "1")
- `abline.linetypes` - Line types for diagonal lines (UI: "Line types", default: "dashed")
- `abline.opacities` - Opacities for diagonal lines (UI: "Opacities (0-1)", default: "1")

Author(s)

Jacob Martin

See Also

plotthis::SplitBarPlot(), organize_inputs(), plotthis_SplitBarPlotOutputUI(), plotthis_SplitBarPlotServ
plotthis_SplitBarPlotApp()

Examples

```
library(VizModules)
mtcars$cyl <- as.factor(mtcars$cyl)
plotthis_SplitBarPlotInputsUI("splitBarPlot", mtcars)
```

---

plotthis_SplitBarPlotOutputUI

Output UI components for the SplitBarPlot module

Description

This should be placed in the UI where the plot should be shown.

Usage

```
plotthis_SplitBarPlotOutputUI(id, resizable = TRUE)
```

Arguments

id	The ID for the Shiny module.
resizable	Logical; when TRUE (the default) the plot output is wrapped in shinyjqui::jqui_resizable() so it can be resized by dragging. Set to FALSE when embedding the output in a container that already provides resizing.

Value

A Shiny plotlyOutput for the SplitBarPlot

Author(s)

Jacob Martin

plotthis_SplitBarPlotServer *Server logic for SplitBarPlot module*

Description

Server logic for SplitBarPlot module

Usage

```
plotthis_SplitBarPlotServer(
  id,
  data,
  hide.inputs = NULL,
  hide.tabs = NULL,
  defaults = NULL
)
```

Arguments

id	The ID for the Shiny module.
data	A reactive containing the data frame to plot.
hide.inputs	A character vector of input IDs to hide. These will still be initialized and their values passed to the plot function, but the user will not be able to see/adjust them in the UI.
hide.tabs	A character vector of tab names to hide. Inputs in these tabs will still be initialized and their values passed to the plot function, but the user will not be able to see/adjust them in the UI.
defaults	A named list of default values for the inputs. When the reset button is clicked, inputs are reset to these values rather than hardcoded fallbacks. Typically the same list passed to the corresponding UI function.

Value

The moduleServer function for the SplitBarPlot module.

Author(s)

Jacob Martin, Jared Andrews

See Also

plotthis::SplitBarPlot(), organize_inputs(), plotthis_SplitBarPlotInputsUI(), plotthis_SplitBarPlotOutp
plotthis_SplitBarPlotApp()

plotthis_ViolinPlotApp *Create an example Modular ViolinPlot Shiny Application*

Description

This function generates a Shiny application with modular `plotthis::ViolinPlot()` components. The app features a **Data Import** section for uploading data, a **Data Table** for filtering the active dataset, and a **Plot** area for configuring and displaying an interactive violin plot.

Usage

```
plotthis_ViolinPlotApp(
  data_list = NULL,
  defaults = NULL,
  hide.inputs = NULL,
  hide.tabs = NULL
)
```

Arguments

`data_list`	An optional named list of data frames. If `NULL` (the default), `list("demographics" = example_demographics)` is used as example data.
`defaults`	A named list of input IDs and their default values to apply on startup.
`hide.inputs`	A character vector of input IDs to hide. Their values are still initialized and used, but the controls are not shown in the UI.
`hide.tabs`	A character vector of tab names to hide. Inputs in these tabs are still initialized and used, but the controls are not shown in the UI.

Details

When `data_list` is not provided (or `NULL`), the app launches with `example_demographics` as an example dataset. Uploaded data files are added to the available datasets and can be selected for plotting. If an uploaded file shares a name with an existing dataset, the existing one is overwritten with a warning.

This is a convenience wrapper around `createModuleApp()`.

Value

A Shiny app object.

Author(s)

Jacob Martin, Jared Andrews

Examples

```
library(VizModules)
# Launch with default example data:
app <- plotthis_ViolinPlotApp()
if (interactive()) runApp(app)

# Launch with custom data:
app2 <- plotthis_ViolinPlotApp(list("demographics" = example_demographics))
if (interactive()) runApp(app2)
```

---

plotthis_ViolinPlotInputsUI

Input UI components for the ViolinPlot module

Description

This should be placed in the UI where the inputs should be shown, with an `id` that matches the `id` used in the `plotthis_ViolinPlotServer()` and `plotthis_ViolinPlotOutputUI()` functions.

Usage

```
plotthis_ViolinPlotInputsUI(
  id,
  data,
  defaults = NULL,
  title = NULL,
  columns = 2
)
```

Arguments

`id`	The ID for the Shiny module.
`data`	The data frame used for plot generation.
`defaults`	A named list of default values for the inputs.
`title`	An optional title for the UI grid.
`columns`	Number of columns for the UI grid.

Details

The user inputs for this module are separated from the outputs to allow for more flexible UI design.

The inputs will automatically be organized into a grid layout via the `organize_inputs()` function, with `columns` controlling the number of columns in the grid.

Defaults can be set for each input by providing a named list of values to the `defaults` argument. Nearly all parameters for `plotthis::ViolinPlot()` can be set via these inputs, so see the help for that function for an exhaustive list.

Value

A Shiny tagList containing the UI elements

Plot parameters not implemented or with altered functionality

The following `plotthis::ViolinPlot()` parameters are not available via UI inputs:

- `xlab` - X-axis label (plotly allows interactive editing)
- `ylab` - Y-axis label (plotly allows interactive editing)
- `title` - Plot title (plotly allows interactive editing)
- `subtitle` - Plot subtitle (not supported in plotly)
- `aspect.ratio` - Aspect ratio control (handled by plotly layout)
- `legend.position` - Legend positioning (plotly allows interactive repositioning)
- `x_sep` - Separator for x columns (not applicable in UI context)
- `in_form` - Data input format (not applicable - always long form)
- `split_by` - Split variable (returns a patchwork object, not supported in plotly), use `facet_by` instead
- `split_by_sep` - Only applies if `split_by` is used
- `symnum_args` - Significance symbol arguments (the Stats tab manages symbol display)
- `keep_empty` - Keep empty values (not implemented)
- `keep_na` - Keep NA values (not implemented)
- `group_by_sep` - Separator for group columns (not applicable in UI context)
- `group_name` - Group legend name (handled by plotly)
- `paired_by` - Pairing variable for paired tests (use the Stats tab "Paired Test" input instead)
- `x_text_angle` - X-axis text angle (handled by plotly axis settings)
- `step_increase` - Step increase for significance brackets (set via the Stats tab "Bracket Spacing")
- `fill_mode` - Fill mode for grouped data (handled automatically)
- `position_dodge_preserve` - Preserve dodge width (not implemented)
- `theme` - ggplot2 theme (not applicable in plotly)
- `theme_args` - Theme arguments (not applicable in plotly)
- `palette` - Managed internally via the palette selection UI
- `palreverse` - Reverse the color palette (not implemented)
- `alpha` - Alpha transparency (not implemented in UI)
- `stack` - Stack violins (not implemented)
- `add_beeswarm` - Add beeswarm points (not implemented in UI)
- `beeswarm_method` - Beeswarm arrangement method (not implemented)
- `beeswarm_cex` - Beeswarm point size factor (not implemented)
- `beeswarm_priority` - Beeswarm priority order (not implemented)

- beeswarm_dodge - Beeswarm dodge width (not implemented)
- add_trend - Add trend line (not implemented in UI)
- trend_color - Trend line color (not implemented)
- trend_linewidth - Trend line width (not implemented)
- trend_ptsize - Trend point size (not implemented)
- add_stat - Add statistical annotation (not implemented)
- stat_name - Statistical test name (not implemented)
- stat_color - Statistical annotation color (not implemented)
- stat_size - Statistical annotation size (not implemented)
- stat_stroke - Statistical annotation stroke (not implemented)
- stat_shape - Statistical annotation shape (not implemented)
- add_bg - Add background shading (not implemented)
- bg_palette - Background palette (not implemented)
- bg_palcolor - Background color (not implemented)
- bg_alpha - Background transparency (not implemented)
- add_line - Add horizontal line (not implemented in UI - use Lines tab)
- line_color - Line color (not implemented)
- line_width - Line width (not implemented)
- line_type - Line type (not implemented)
- comparisons - plotthis pairwise comparisons are not passed; equivalent pairwise significance testing is provided via the module's Stats tab (see "Statistical annotation parameters")
- ref_group - Reference group for comparisons (use the Stats tab instead)
- pairwise_method - Pairwise test method (set via the Stats tab "Test" input instead)
- multiplegroup_comparisons - Multiple-group comparison flag (use the Stats tab instead)
- multiple_method - Multiple-group test method (set via the Stats tab "Test" input instead)
- sig_label - Significance label format (set via the Stats tab "Display" input instead)
- sig_labelsize - Significance label size (handled by the Stats tab)
- hide_ns - Hide non-significant comparisons (use the Stats tab "Hide Non-Significant" input)
- seed - Random seed (not applicable)
- combine - Only applies if split_by is used
- nrow - Only applies if split_by is used
- ncol - Only applies if split_by is used
- byrow - Only applies if split_by is used
- axes - Only applies if split_by is used
- axis_titles - Only applies if split_by is used
- guides - Only applies if split_by is used
- legend.direction - Managed position of legend however this can be handled via plotly

Plot parameters and defaults

The following `plotthis::ViolinPlot()` parameters can be accessed via UI inputs and/or the `defaults` argument:

- `x` - X-axis variable (UI: "X Data", default: 2nd categorical variable)
- `y` - Y-axis variable (UI: "Y Data", default: 2nd numeric variable)
- `group_by` - Grouping variable (UI: "Group By", default: "")
- `flip` - Flip/swap the x and y axes (UI: "Rotate (swap X/Y)", default: FALSE)
- `sort_x` - Sort X-axis by statistic (UI: "Sort X By", default: "none")
- `y_max` - Maximum Y-axis value (UI: "Y Max", default: calculated)
- `y_min` - Minimum Y-axis value (UI: "Y Min", default: calculated)
- `add_point` - Add jitter points (UI: "Add Jitter Points", default: FALSE)
- `pt_size` - Point size (UI: "Point Size", default: 1)
- `pt_alpha` - Point transparency (UI: "Point Alpha", default: 1)
- `jitter_width` - Jitter width (UI: "Jitter Width", default: 0.5)
- `jitter_height` - Jitter height (UI: "Jitter Height", default: 0)
- `pt_color` - Point outline color (UI: "Point Outline Colour", default: "#000000")
- `add_box` - Add box plot overlay (UI: "Add Box", default: FALSE)
- `box_color` - Box outline color (UI: "Box Colour", default: "#000000")
- `box_width` - Box width (UI: "Box Width", default: 0.1)
- `box_ptsize` - Box point size (UI: "Box Point Size", default: 2.5)
- `highlight` - Highlight condition (UI: "Highlight", default: "")
- `highlight_color` - Highlight color (UI: "Highlight Colour", default: "#000000")
- `highlight_size` - Highlight size (UI: "Highlight Size", default: 1)
- `highlight_alpha` - Highlight transparency (UI: "Highlight Alpha", default: 1)
- `facet_by` - Faceting variable (UI: "Facet By", default: "")
- `facet_scales` - Facet scale behavior (UI: "Facet Scale", default: "fixed")
- `facet_ncol` - Number of facet columns (UI: "Columns", default: NULL)
- `facet_nrow` - Number of facet rows (UI: "Rows", default: NULL)
- `facet_byrow` - Facet ordering direction (UI: "Facet By Row", default: TRUE)
- `palcolor` - Custom color values (UI: palette picker, derived from palette)

Statistical annotation parameters

The module provides plotly-based significance testing via the Stats tab. The following inputs are available:

- `stats.enabled` - Enable statistical annotations (UI: "Enable Stats", default: FALSE)
- `stat.test` - Test to apply: Wilcoxon, t-test, Kruskal-Wallis, or ANOVA (UI: "Test")
- `stat.p.adjust` - P-value adjustment method (UI: "P-value Adjustment", default: "holm")

- `stat.display` - Value to display: adjusted p-value, p-value, or symbols (UI: "Display")
- `stat.sig.threshold` - Significance threshold for symbols/hiding (UI: "Significance Threshold")
- `stat.hide.ns` - Hide non-significant comparisons (UI: "Hide Non-Significant", default: FALSE)
- `stat.paired` - Use a paired test (UI: "Paired Test", default: FALSE)
- `stat.pairs` - Group comparisons to test (UI: "Comparisons", multiple selection)
- `stat.line.color` - Bracket line color (UI: "Line Color", default: "#000000")
- `stat.line.width` - Bracket line width (UI: "Line Width")
- `stat.bracket.style` - Bracket style, capped or flat (UI: "Bracket Style")
- `stat.step.increase` - Vertical spacing between stacked brackets (UI: "Bracket Spacing")
- `stat.text.bump` - Offset of the significance text above the bracket (UI: "Text Offset")
- `stat.bracket.inset` - Horizontal inset of the brackets (UI: "Bracket Inset")
- `stat.per.facet` - Compute statistics independently per facet panel (UI: "Per Facet Panel")

Parameters controlling additional functionality

The following parameters implementing new functionality or controlling plotly-specific features are also available:

- `title.font.size` - Plot title font size (UI: "Title Size", default: 26)
- `title.font.family` - Font family for title text (UI: "Title Font", default: "Arial")
- `title.font.color` - Color for plot title (UI: "Title Color", default: "#000000")
- `axis.title.font.size` - Axis title font size (UI: "Axis Title Size", default: 18)
- `axis.title.font.color` - Axis title font color (UI: "Axis Title Color", default: "#000000")
- `axis.title.font.family` - Axis title font family (UI: "Axis Title Font", default: "Arial")
- `axis.showline` - Show axis border lines (UI: "Show axis lines", default: TRUE)
- `axis.mirror` - Mirror axis lines on opposite side (UI: "Mirror axis lines", default: TRUE)
- `show.grid.x` - Show X-axis major gridlines (UI: "Show X major gridlines", default: TRUE)
- `show.grid.y` - Show Y-axis major gridlines (UI: "Show Y major gridlines", default: TRUE)
- `axis.linecolor` - Color of axis lines (UI: "Axis line color", default: "black")
- `axis.linewidth` - Width of axis lines (UI: "Axis line width", default: 0.5)
- `axis.tickfont.size` - Size of tick labels (UI: "Tick label size", default: 12)
- `axis.tickfont.color` - Color of tick labels (UI: "Tick label color", default: "black")
- `axis.tickfont.family` - Font family for tick labels (UI: "Tick label font", default: "Arial")
- `axis.tickangle.x` - Rotation angle for X-axis tick labels (UI: "X-axis tick label angle", default: 0)
- `axis.tickangle.y` - Rotation angle for Y-axis tick labels (UI: "Y-axis tick label angle", default: 0)
- `axis.ticks` - Position of tick marks (UI: "Tick position", default: "outside")
- `axis.tickcolor` - Color of tick marks (UI: "Tick mark color", default: "black")

- `axis.ticklen` - Length of tick marks (UI: "Tick mark length", default: 5)
- `axis.tickwidth` - Width of tick marks (UI: "Tick mark width", default: 1)
- `hline.intercepts` - Y-coordinates for horizontal reference lines (UI: "Y-intercepts", default: "")
- `hline.colors` - Colors for horizontal lines (UI: "Colors", default: "#000000")
- `hline.widths` - Widths for horizontal lines (UI: "Widths", default: "1")
- `hline.linetypes` - Line types for horizontal lines (UI: "Line types", default: "dashed")
- `hline.opacities` - Opacities for horizontal lines (UI: "Opacities (0-1)", default: "1")
- `vline.intercepts` - X-coordinates for vertical reference lines (UI: "X-intercepts", default: "")
- `vline.colors` - Colors for vertical lines (UI: "Colors", default: "#000000")
- `vline.widths` - Widths for vertical lines (UI: "Widths", default: "1")
- `vline.linetypes` - Line types for vertical lines (UI: "Line types", default: "dashed")
- `vline.opacities` - Opacities for vertical lines (UI: "Opacities (0-1)", default: "1")
- `abline.slopes` - Slopes for diagonal reference lines (UI: "Slopes", default: "")
- `abline.intercepts` - Y-intercepts for diagonal lines (UI: "Y-intercepts", default: "")
- `abline.colors` - Colors for diagonal lines (UI: "Colors", default: "#000000")
- `abline.widths` - Widths for diagonal lines (UI: "Widths", default: "1")
- `abline.linetypes` - Line types for diagonal lines (UI: "Line types", default: "dashed")
- `abline.opacities` - Opacities for diagonal lines (UI: "Opacities (0-1)", default: "1")

Author(s)

Jacob Martin, Jared Andrews

See Also

`plotthis::ViolinPlot()`, `organize_inputs()`, `plotthis_ViolinPlotOutputUI()`, `plotthis_ViolinPlotServer()`, `plotthis_ViolinPlotApp()`

Examples

```
library(VizModules)
data(mtcars)
plotthis_ViolinPlotInputsUI("ViolinPlot", mtcars)
```

plotthis_ViolinPlotOutputUI

Output UI components for the ViolinPlot module

Description

This should be placed in the UI where the plot should be shown.

Usage

```
plotthis_ViolinPlotOutputUI(id, resizable = TRUE)
```

Arguments

id	The ID for the Shiny module.
resizable	Logical; when TRUE (the default) the plot output is wrapped in shinyjqui::jqui_resizable() so it can be resized by dragging. Set to FALSE when embedding the output in a container that already provides resizing.

Value

A Shiny plotlyOutput for the ViolinPlot

Author(s)

Jacob Martin

plotthis_ViolinPlotServer

Server logic for ViolinPlot module

Description

Server logic for ViolinPlot module

Usage

```
plotthis_ViolinPlotServer(
  id,
  data,
  hide.inputs = NULL,
  hide.tabs = NULL,
  defaults = NULL
)
```

Arguments

id	The ID for the Shiny module.
data	A reactive containing the data frame to plot.
hide.inputs	A character vector of input IDs to hide. These will still be initialized and their values passed to the plot function, but the user will not be able to see/adjust them in the UI.
hide.tabs	A character vector of tab names to hide. Inputs in these tabs will still be initialized and their values passed to the plot function, but the user will not be able to see/adjust them in the UI.
defaults	A named list of default values for the inputs. When the reset button is clicked, inputs are reset to these values rather than hardcoded fallbacks. Typically the same list passed to the corresponding UI function.

Value

The moduleServer function for the ViolinPlot module.

Author(s)

Jacob Martin, Jared Andrews

---

radarPlot *Create a plotly radar chart*

Description

Create a plotly radar chart

Usage

```
radarPlot(
  df,
  theta,
  r,
  group = NULL,
  colors = NULL,
  palette = NULL,
  fill = "toself",
  line.width = 2,
  line.dash = "solid",
  marker.size = 5,
  marker.symbol = "circle",
  opacity = 0.6,
  radial.visible = TRUE,
  radial.range = NULL,
  radial.showline = TRUE,
```

```
  radial.linecolor = "#444444",
  radial.gridcolor = "#EEEEEE",
  angular.direction = "clockwise",
  angular.rotation = 90,
  angular.gridcolor = "#EEEEEE",
  show.legend = TRUE,
  legend.orientation = "h",
  legend.x = 0.5,
  legend.y = -0.1,
  legend.font.family = "Arial",
  legend.font.size = 12,
  legend.font.color = "#000000",
  title.text = "",
  title.font.family = "Arial",
  title.font.size = 18,
  title.font.color = "#000000",
  title.x = 0.5,
  bgcolor = "#FFFFFF",
  polar.bgcolor = "#FFFFFF"
)
```

Arguments

df	A data frame containing the data to plot. For a single trace, provide columns for categories (theta) and values (r). For multiple traces, include a grouping column. The function automatically closes the radar polygon by adding the first point to the end.
theta	Character, name of the column to use for the angular categories (axes).
r	Character, name of the column to use for the radial values.
group	Optional character, name of the column to use for grouping multiple traces. If NULL, a single trace is plotted. Default: NULL.
colors	Optional character vector of hex colors for the traces. If named, values are matched to the group values; otherwise colours are recycled.
palette	Optional character vector of fallback colors used when `colors` is not supplied or missing values are present.
fill	Logical or character, whether to fill the area under each trace. Use "toself" to fill to the first point, or FALSE for no fill. Default: "toself".
line.width	Numeric, width of the trace lines in pixels. Default: 2.
line.dash	Character, line dash style. Options: "solid", "dot", "dash", "longdash", "dash-dot", "longdashdot". Default: "solid".
marker.size	Numeric, size of the markers on the trace. Default: 5.
marker.symbol	Character, marker symbol. Options: "circle", "square", "diamond", "cross", "x", "triangle-up", etc. Default: "circle".
opacity	Numeric, opacity of the traces (0-1). Default: 0.6.
radial.visible	Logical, whether to show the radial axis. Default: TRUE.

`radial.range` Optional numeric vector of length 2 specifying the range of the radial axis (e.g., c(0, 100)). If NULL, automatically determined. Default: NULL.

`radial.showline`
Logical, whether to show the radial axis line. Default: TRUE.

`radial.linecolor`
Character, hex color for the radial axis line. Default: "#444444".

`radial.gridcolor`
Character, hex color for the radial grid lines. Default: "#EEEEEE".

`angular.direction`
Character, direction of angular axis. Options: "clockwise" or "counterclockwise". Default: "clockwise".

`angular.rotation`
Numeric, rotation angle for the angular axis in degrees. Default: 90.

`angular.gridcolor`
Character, hex color for the angular grid lines. Default: "#EEEEEE".

`show.legend` Logical, whether to display the legend. Default: TRUE.

`legend.orientation`
Character, legend orientation. Options: "h" (horizontal) or "v" (vertical). Default: "h".

`legend.x` Numeric, horizontal legend position offset (0-1). Default: 0.5.

`legend.y` Numeric, vertical legend position offset (-1 to 1). Default: -0.1.

`legend.font.family`
Character, font family for the legend text. Default: "Arial".

`legend.font.size`
Numeric, font size for the legend text. Default: 12.

`legend.font.color`
Character, hex color for the legend text. Default: "#000000".

`title.text` Character, main plot title text. Default: "".

`title.font.family`
Character, font family for the title text. Default: "Arial".

`title.font.size`
Numeric, font size for the title text. Default: 18.

`title.font.color`
Character, hex color for the title text. Default: "#000000".

`title.x` Numeric, horizontal position for the plot title (0-1). Default: 0.5.

`bgcolor` Character, hex color for the plot background. Default: "#FFFFFF".

`polar.bgcolor` Character, hex color for the polar area background. Default: "#FFFFFF".

Value

A plotly object.

Author(s)

Jacob Martin

Examples

```
# Single trace radar chart
# Note: Polygon is automatically closed by the function
skills <- data.frame(
    category = c("Speed", "Strength", "Defense", "Stamina"),
    value = c(8, 6, 7, 9)
)

radarPlot(
    df = skills,
    theta = "category",
    r = "value",
    title.text = "Player Stats"
)

# Multiple trace radar chart
# Note: Polygon is automatically closed for each trace
team_stats <- data.frame(
    category = rep(c("Speed", "Strength", "Defense", "Stamina"), 2),
    value = c(8, 6, 7, 9, 5, 9, 8, 6),
    player = rep(c("Player A", "Player B"), each = 4)
)

radarPlot(
    df = team_stats,
    theta = "category",
    r = "value",
    group = "player",
    title.text = "Team Comparison"
)
```

---

radarPlotApp *Create an example Modular radarPlot Shiny Application*

Description

This function generates a Shiny application with modular radarPlot components. The app features a **Data Import** section for uploading data, a **Data Table** for filtering the active dataset, and a **Plot** area for configuring and displaying an interactive radar plot.

Usage

```
radarPlotApp(
  data_list = NULL,
  defaults = NULL,
  hide.inputs = NULL,
  hide.tabs = NULL
)
```

Arguments

data_list	An optional named list of data frames. If NULL (the default), list("skills" = example_skills) is used. Each data frame should contain columns for categories (theta) and values (r). For multiple traces, include a grouping column.
defaults	A named list of input IDs and their default values to apply on startup.
hide.inputs	A character vector of input IDs to hide. Their values are still initialized and used, but the controls are not shown in the UI.
hide.tabs	A character vector of tab names to hide. Inputs in these tabs are still initialized and used, but the controls are not shown in the UI.

Details

When data_list is not provided (or NULL), the app launches with example_skills as an example dataset. Uploaded data files are added to the available datasets and can be selected for plotting. If an uploaded file shares a name with an existing dataset, the existing one is overwritten with a warning.

This is a convenience wrapper around createModuleApp().

Value

A Shiny app object.

Author(s)

Jacob Martin, Jared Andrews

See Also

radarPlot(), radarPlotInputsUI(), radarPlotOutputUI(), radarPlotServer()

Examples

```
library(VizModules)
# Launch with default example data:
app <- radarPlotApp()
if (interactive()) runApp(app)

# Launch with custom data:
skills <- data.frame(
    entity = c(
        rep("Player A", 6),
        rep("Player B", 6),
        rep("Player C", 6),
        rep("Player D", 6)
    ),
   category = rep(c("Pace", "Shooting", "Passing", "Dribbling", "Defending", "Physical"), 4),
    value = c(
        99, 89, 80, 92, 36, 78,
        89, 97, 65, 72, 45, 95,
        76, 86, 94, 86, 64, 78,
```

```
        62, 60, 71, 63, 94, 91
    )
)
app2 <- radarPlotApp(list("skills" = skills))
if (interactive()) runApp(app2)
```

`radarPlotInputsUI` *Input UI components for the radarPlot module*

Description

This should be placed in the UI where the inputs should be shown, with an `id` that matches the `id` used in the `radarPlotServer()` and `radarPlotOutputUI()` functions.

Usage

```
radarPlotInputsUI(id, data, defaults = NULL, title = NULL, columns = 2)
```

Arguments

`id`	The ID for the Shiny module.
`data`	The data frame used for plot generation.
`defaults`	A named list of default values for the inputs.
`title`	An optional title for the UI grid.
`columns`	Number of columns for the UI grid.

Details

The user inputs for this module are separated from the outputs to allow for more flexible UI design.

The inputs will automatically be organized into a grid layout via the `organize_inputs()` function, with `columns` controlling the number of columns in the grid.

Defaults can be set for each input by providing a named list of values to the `defaults` argument. Provide data with columns for categories (theta) and values (r). For multiple traces, include a grouping column. Nearly all parameters for `radarPlot()` can be set via these inputs, so see the help for that function for an exhaustive list.

Value

A Shiny tagList containing the UI elements

Plot parameters not implemented or with altered functionality

The following `radarPlot()` parameters are not exposed as UI inputs:

- `palette` - Color palette name; use `colors` via the color picker UI instead
- `legend.x` - Legend horizontal position offset (use `defaults` to set)
- `legend.y` - Legend vertical position offset (use `defaults` to set)
- `title.text` - Plot title text (plotly allows interactive editing; use `defaults` to set)

Plot parameters and defaults

The following `radarPlot()` parameters can be accessed via UI inputs and/or the `defaults` argument:

- `theta` - Category column for angular axes (UI: "Category column (theta)", default: 1st categorical column)
- `r` - Values column for radial distance (UI: "Values column (r)", default: 1st numeric column)
- `group` - Optional grouping column for multiple traces (UI: "Group column", default: NULL)
- `fill` - Fill area under trace (UI: "Fill area", default: "toself")
- `line.width` - Line width (UI: "Line width", default: 2)
- `line.dash` - Line dash style (UI: "Line style", default: "solid")
- `marker.size` - Marker size (UI: "Marker size", default: 5)
- `marker.symbol` - Marker symbol (UI: "Marker symbol", default: "circle")
- `opacity` - Trace opacity (UI: "Opacity", default: 0.6)
- `colors` - Trace colors (UI: color picker, derived from palette)
- `radial.visible` - Show radial axis (UI: "Show radial axis", default: TRUE)
- `radial.range` - Radial axis range (UI: "Radial min" and "Radial max", default: auto)
- `radial.showline` - Show radial axis line (UI: "Show radial line", default: TRUE)
- `radial.linecolor` - Radial axis line color (UI: "Radial line color", default: "#444444")
- `radial.gridcolor` - Radial grid color (UI: "Radial grid color", default: "#EEEEEE")
- `angular.direction` - Angular axis direction (UI: "Angular direction", default: "clockwise")
- `angular.rotation` - Angular axis rotation (UI: "Angular rotation", default: 90)
- `angular.gridcolor` - Angular grid color (UI: "Angular grid color", default: "#EEEEEE")
- `title.x` - Title horizontal position (UI: "Title horizontal position", default: 0.5)
- `title.font.size` - Plot title font size (UI: "Title Size", default: 26)
- `title.font.family` - Font family for title text (UI: "Title Font", default: "Arial")
- `title.font.color` - Color for plot title (UI: "Title Color", default: "#000000")
- `show.legend` - Show legend (UI: "Show legend", default: TRUE)
- `legend.orientation` - Legend orientation (UI: "Legend orientation", default: "h")
- `legend.font.family` - Legend font (UI: "Legend font", default: "Arial")
- `legend.font.size` - Legend font size (UI: "Legend font size", default: 12)
- `legend.font.color` - Legend font color (UI: "Legend font color", default: "#000000")
- `bgcolor` - Plot background color (UI: "Plot background color", default: "#FFFFFF")
- `polar.bgcolor` - Polar area background color (UI: "Polar area background", default: "#FFFFFF")

Author(s)

Jacob Martin

See Also

radarPlot(), organize_inputs(), radarPlotOutputUI(), radarPlotServer(), radarPlotApp()

Examples

```
library(VizModules)
skills <- data.frame(
    category = c("Speed", "Strength", "Defense", "Stamina", "Speed"),
    value = c(8, 6, 7, 9, 8)
)
radarPlotInputsUI("radarPlot", skills)
```

---

radarPlotOutputUI *Output UI components for the radarPlot module*

Description

This should be placed in the UI where the plot should be shown.

Usage

```
radarPlotOutputUI(id, resizable = TRUE)
```

Arguments

id	The ID for the Shiny module.
resizable	Logical; when TRUE (the default) the plot output is wrapped in shinyjqui::jqui_resizable() so it can be resized by dragging. Set to FALSE when embedding the output in a container that already provides resizing.

Value

A Shiny plotlyOutput for the radarPlot

Author(s)

Jared Andrews

`radarPlotServer` *Server logic for radarPlot module*

Description

Server logic for radarPlot module

Usage

```
radarPlotServer(
  id,
  data,
  hide.inputs = NULL,
  hide.tabs = NULL,
  defaults = NULL
)
```

Arguments

`id`	The ID for the Shiny module.
`data`	A `reactive` containing the data frame to plot. Provide data with columns for categories (theta) and values (r). For multiple traces, include a grouping column.
`hide.inputs`	A character vector of input IDs to hide. These will still be initialized and their values passed to the plot function, but the user will not be able to see/adjust them in the UI.
`hide.tabs`	A character vector of tab names to hide. Inputs in these tabs will still be initialized and their values passed to the plot function, but the user will not be able to see/adjust them in the UI.
`defaults`	A named list of default values for the inputs. When the reset button is clicked, inputs are reset to these values rather than hardcoded fallbacks. Typically the same list passed to the corresponding UI function.

Value

The `moduleServer` function for the radarPlot module.

Author(s)

Jacob Martin

See Also

`radarPlot()`, `radarPlotInputsUI()`, `radarPlotOutputUI()`, `radarPlotApp()`

recycle_line_style *Recycle style vector to match line count*

Description

Extends or truncates a style vector to match the number of lines being drawn. If the input length doesn't match the target length, uses only the first value for all lines (as documented behavior).

Usage

```
recycle_line_style(values, n, default)
```

Arguments

values	Vector. Style values (colors, widths, etc.) to recycle.
n	Integer. Target length (number of lines).
default	The default value to use if values is NULL or empty.

Value

A vector of length n with recycled values.

Author(s)

Jared Andrews

Examples

```
recycle_line_style(c("red", "blue"), 2, "black")
recycle_line_style(NULL, 3, "black")
```

register_model_backend *Register a model backend*

Description

Add or replace a model backend in the registry. Once registered, the backend name appears in the model-type dropdown and the custom-model-lines pipeline dispatches through it automatically.

Usage

```
register_model_backend(name, backend)
```

Arguments

name	Character string. The name that will appear in the model-type dropdown (e.g. "drm", "gam").
backend	A named list with elements fit, predict, and validate_classes as described above.

Details

A backend is a named list with three required elements:

fit A function with signature function(formula, data, ...) that returns a fitted model object. Extra UI fields from the multiDynamicInput() row are forwarded as ....

predict A function with signature function(model, newdata) that returns a numeric vector of predicted y-values, one per row of newdata.

validate_classes Character vector of class names. After fitting, the pipeline checks inherits(model, validate_classes) and rejects the model if it fails.

Value

Invisibly returns NULL. Called for its side effect.

Author(s)

Jacob Martin

Examples

```
# Register a custom backend for dose-response curves (requires drc)
if (requireNamespace("drc", quietly = TRUE)) {
}
```

---

reset_axes_inputs *Reset uniform axes inputs to defaults*

Description

Resets all inputs created by uniform_axes_inputs_ui() to their default values. Call inside an observeEvent(input$reset, ...) block to avoid duplicating 20 updateXxxInput calls in every module server.

Usage

```
reset_axes_inputs(session, defaults = NULL)
```

Arguments

session	The Shiny session object (from moduleServer).
defaults	A named list of default values to reset to, or NULL to use hardcoded fallbacks. Typically the same list passed to the UI function.

Value

Called for side effects; returns invisible(NULL).

Author(s)

Jared Andrews

Examples

```
## Not run:
# Call inside a module server's observeEvent(input$reset, ...) block:
reset_axes_inputs(session, defaults)

## End(Not run)
```

---

reset_legend_inputs *Reset uniform Legend inputs to defaults*

Description

Resets the legend styling inputs (legend title and entry label font sizes) created by uniform_legend_inputs_ui() back to their default values.

Usage

```
reset_legend_inputs(session, defaults = NULL)
```

Arguments

session	The Shiny session object.
defaults	A named list of default values. When provided, inputs reset to these values rather than hardcoded fallbacks. Typically the same list passed to the UI function.

Value

Called for side effects; returns invisible(NULL).

Author(s)

Jared Andrews

Examples

```
## Not run:
# Call inside a module server's observeEvent(input$reset, ...) block:
reset_legend_inputs(session, defaults)

## End(Not run)
```

`reset_lines_inputs` *Reset uniform lines inputs to defaults*

Description

Resets all inputs created by `uniform_lines_inputs_ui()` to their default values. Call inside an `observeEvent(input$reset, ...)` block to avoid duplicating line-reset boilerplate in every module server.

Usage

```
reset_lines_inputs(session, include.fit.lines = FALSE, defaults = NULL)
```

Arguments

`session`	The Shiny session object (from `moduleServer`).
`include.fit.lines`	Logical; if TRUE, also resets the fit-line inputs (best.fit, line.best.smoothness, line.best.colour, linear.model). Default is FALSE.
`defaults`	A named list of default values to reset to, or NULL to use hardcoded fallbacks. Typically the same list passed to the UI function.

Value

Called for side effects; returns `invisible(NULL)`.

Author(s)

Jared Andrews

Examples

```
## Not run:
# Call inside a module server's observeEvent(input$reset, ...) block:
reset_lines_inputs(session, defaults = defaults)

## End(Not run)
```

reset_plotly_inputs *Reset uniform Plotly inputs to defaults*

Description

Resets all inputs created by `uniform_plotly_inputs_ui()` to their default values. Call inside an `observeEvent(input$reset, ...)` block to avoid duplicating Plotly-reset boilerplate in every module server.

Usage

```
reset_plotly_inputs(session, defaults = NULL)
```

Arguments

`session`	The Shiny session object (from `moduleServer`).
`defaults`	A named list of default values to reset to, or NULL to use hardcoded fallbacks. Typically the same list passed to the UI function.

Value

Called for side effects; returns `invisible(NULL)`.

Author(s)

Jared Andrews

Examples

```
## Not run:
# Call inside a module server's observeEvent(input$reset, ...) block:
reset_plotly_inputs(session, defaults)

## End(Not run)
```

resolve_facet_layout *Resolve number of rows for a faceted subplot grid*

Description

Given the number of facet levels and optional user-supplied `facet.nrow` / `facet.ncol` values, computes the `nrows` argument to pass to `plotly::subplot`.

Usage

```
resolve_facet_layout(n_facets, facet.nrow = NULL, facet.ncol = NULL)
```

Arguments

n_facets	Integer, number of facet panels.
facet.nrow	Optional integer, user-requested number of rows.
facet.ncol	Optional integer, user-requested number of columns.

Details

Resolution rules:

- Both NULL/NA: returns 1 (single row, preserves legacy behaviour).
- Only facet.nrow supplied: returns that value.
- Only facet.ncol supplied: returns ceiling(n_facets / facet.ncol).
- Both supplied: facet.nrow wins.

The result is clamped to the range [1, n_facets].

Value

A positive integer giving the number of rows for plotly::subplot.

Author(s)

Jared Andrews

Examples

```
resolve_facet_layout(6, facet.nrow = 2)
resolve_facet_layout(6, facet.ncol = 3)
```

---

resolve_facet_sharing *Resolve facet axis sharing from facet.scales*

Description

Converts a facet.scales string (one of "fixed", "free", "free_x", "free_y") into the shareX / shareY logical values expected by plotly::subplot.

Usage

```
resolve_facet_sharing(facet.scales = "fixed")
```

Arguments

facet.scales	Character, one of "fixed" (default), "free", "free_x", or "free_y".

Value

A named list with logical elements shareX and shareY.

Author(s)

Jared Andrews

Examples

```
resolve_facet_sharing("fixed")
resolve_facet_sharing("free_x")
```

---

resolve_palette *Resolve a color palette for plot groups*

Description

Maps groups to colors using selected colors or a default palette. Handles named color vectors by matching to group names, fills in missing colors with fallback values, and ensures the output vector is named and matches group length.

Usage

```
resolve_palette(groups, selected_colors = NULL, default_palette = NULL)
```

Arguments

`groups`	A character vector of group names to assign colors to.
`selected_colors`	A named or unnamed character vector of colors to use. If named, colors are matched to groups by name. If NULL or empty, uses `default_palette`.
`default_palette`	A character vector of fallback colors to use when `selected_colors` is NULL/empty or when groups have missing colors. Defaults to "#000000" (black) if not provided.

Value

A named character vector of colors with names corresponding to groups, or NULL if groups is empty.

Author(s)

Jared Andrews

Examples

```
groups <- c("A", "B", "C")
colors <- c(A = "#FF0000", B = "#00FF00", C = "#0000FF")
resolve_palette(groups, colors)
# Returns: c(A = "#FF0000", B = "#00FF00", C = "#0000FF")

# Using default palette
resolve_palette(groups, NULL, c("#1B9E77", "#D95F02", "#7570B3"))
# Returns: c(A = "#1B9E77", B = "#D95F02", C = "#7570B3")
```

---

safe_eval_filter *Safely evaluate a user-provided filter expression against a data frame*

Description

Parses the expression text, validates that it only contains allowed operations (comparisons, logical operators, column references, and literals), then evaluates it in a restricted environment containing only the data frame columns. Returns a logical vector suitable for row subsetting, or `NULL` if the input is empty or invalid.

Usage

```
safe_eval_filter(expr_text, data)
```

Arguments

`expr_text`	Character string containing the filter expression (e.g., `"Sepal.Length > 5 & Species == 'setosa'"`).
`data`	A `data.frame` whose columns are made available for the expression.

Details

Use this function any time a module evaluates a user-typed expression directly (e.g., a row-filter text input). Never call `eval(str2expression())` on raw user input — doing so allows arbitrary code execution on the server.

Value

A logical vector the same length as `nrow(data)`, or `NULL` if the input is empty, unparseable, or contains disallowed operations.

Author(s)

Jared Andrews

Examples

```
safe_eval_filter("Sepal.Length > 5", iris)
safe_eval_filter("Sepal.Length > 5 & Species == 'setosa'", iris)
safe_eval_filter("", iris) # NULL
safe_eval_filter("system('echo pwned')", iris) # NULL + warning
```

---

safe_resolve_adj_fxn *Safely resolve an adjustment function name to an actual function*

Description

Validates that the provided function name is in the allowed list before converting it to a function reference. Returns NULL for empty strings or unrecognized names.

Usage

```
safe_resolve_adj_fxn(fn_name)
```

Arguments

fn_name	Character string — name of the adjustment function. Currently allowed values: "log2", "log", "log10", "neg_log10", "log1p", "as.factor", "abs", "sqrt".

Details

Use this instead of `eval(str2expression())` **when resolving function names from user input** (e.g., a dropdown that selects a transformation like "log2" or "sqrt").

Value

The corresponding function, or NULL if fn_name is empty or not in the allowed list.

Author(s)

Jared Andrews

Examples

```
safe_resolve_adj_fxn("log2") # returns log2
safe_resolve_adj_fxn("") # NULL
safe_resolve_adj_fxn("system") # warning + NULL
```

`setup_auto_update_logic` *Set up auto-update/isolate logic for reactive contexts*

Description

A helper function that encapsulates the common pattern of handling auto-update functionality in module servers. When auto-update is disabled, it adds a dependency on the update button. Returns a wrapper function that either isolates reactive expressions or passes them through unchanged.

Usage

```
setup_auto_update_logic(input)
```

Arguments

`input`	The Shiny input object from the module server, should have both `auto.update` (boolean) and `update` (button) inputs.

Details

This function consolidates the following common pattern:

```
auto_update <- input$auto.update
if (!auto_update) {
    input$update
}
isolate_fn <- if (auto_update) identity else isolate
```

Usage in a reactive context:

```
output$plot <- renderPlotly({
    isolate_fn <- setup_auto_update_logic(input)
    # Now use isolate_fn to wrap input values
    x_val <- isolate_fn(input$x.value)
})
```

Value

A function that wraps reactive expressions. Returns `identity` if auto-update is enabled (expressions will be reactive), or `isolate` if auto-update is disabled (expressions will not trigger reactivity).

Author(s)

Jared Andrews

Examples

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

    ui <- fluidPage(
        selectInput("x_var", "X variable", choices = names(mtcars), selected = "wt"),
        selectInput("y_var", "Y variable", choices = names(mtcars), selected = "mpg"),
        checkboxInput("auto.update", "Auto-update", value = TRUE),
        actionButton("update", "Update"),
        plotlyOutput("myPlot")
    )

    server <- function(input, output, session) {
        output$myPlot <- renderPlotly({
            isolate_fn <- setup_auto_update_logic(input)
            x_val <- isolate_fn(input$x_var)
            y_val <- isolate_fn(input$y_var)
           plot_ly(mtcars, x = ~ .data[[x_val]], y = ~ .data[[y_val]], type = "scatter",
                mode = "markers")
        })
    }

    shinyApp(ui, server)
}
```

`setup_manual_edits` *Set up persistent manual plot layout edits across re-renders*

Description

VizModules plots are fully rebuilt on every input change, which would normally discard any layout tweaks a user made by hand: dragging the legend, moving or editing annotations, repositioning the (draggable) axis titles, or sliding a continuous-colour legend (colorbar). `setup_manual_edits()` together with its companion `finalize_manual_edits()` add this persistence to a plotting module with two short lines of wiring, so manually repositioned elements survive subsequent rebuilds.

Usage

```
setup_manual_edits(input, session, plot_source)
```

Arguments

`input`	The module's `input` object.
`session`	The module's `session` object.
`plot_source`	Character scalar. A unique plotly event source id for this module instance, typically `session$ns("<plot-type>")` (e.g. `session$ns("scatter")`). It scopes the captured `plotly_relayout` events to this plot and **must match** the `plot_source` later passed to `finalize_manual_edits()`.

Details

Call `setup_manual_edits()` **once**, near the top of your module's `shiny::moduleServer()` body, so the observers it registers belong to the module's reactive domain. It creates a reactive store and registers the observers that capture `plotly_relayout` events (legend, annotation, and axis-title drags) plus the JavaScript-forwarded colorbar drag. Then, inside your `plotly::renderPlotly()`, pass the freshly built figure through `finalize_manual_edits()` before returning it.

Value

A list (the "edit store") to hand to `finalize_manual_edits()`, with components:

`edits` A `shiny::reactiveValues()` holding the captured `legend`, `annotations`, and `colorbar` edits.

`rendered_fig` A `shiny::reactiveVal()` holding the most recently rendered figure, used to map relayout annotation indices to stable, rebuild-proof keys.

Module wiring (three steps)

```
myPlotServer <- function(id, data) {
    moduleServer(id, function(input, output, session) {
        # 1. Unique event source + edit store (once, near the top).
        plot_source <- session$ns("myplot")
        edit_store <- setup_manual_edits(input, session, plot_source)

        output$myPlot <- renderPlotly({
            # 2. Create your plotly plot
            fig <- build_my_plotly_figure(data(), input)
            # 3. Finalize on the rebuilt figure, then return it.
            finalize_manual_edits(fig, plot_source, edit_store, session)
        })
    })
}
```

Author(s)

Jared Andrews

See Also

`finalize_manual_edits()` for the render-step companion.

Examples

```
## Not run:
# Call once inside a module server's moduleServer() body:
plot_source <- session$ns("myplot")
edit_store <- setup_manual_edits(input, session, plot_source)

## End(Not run)
```

string_to_linetypes *Parse and validate linetype string to a vector*

Description

Parses a comma-separated string of linetypes and validates each element. Invalid linetypes are replaced with "solid" and a warning is issued.

Usage

```
string_to_linetypes(x)
```

Arguments

x	A string of linetypes delimited by commas, e.g. "solid, dashed, dotted". Valid linetypes are: "solid", "dashed", "dotted", "dotdash", "longdash", "twodash".

Value

A character vector of validated linetypes. If the input is "" or NULL, returns "solid".

Author(s)

Jared Andrews

Examples

```
string_to_linetypes("solid, dashed, dotted")
string_to_linetypes("")
```

uniform_axes_inputs_ui *Generate uniform Axes input UI*

Description

Creates a standardized tagList of axis-related inputs for use across plot modules.

Usage

```
uniform_axes_inputs_ui(
  ns,
  defaults = NULL,
  include.rotate = FALSE,
  include.flip = FALSE
)
```

Arguments

ns	A namespace function, typically created by NS(id).
defaults	A named list of default values for the inputs.
include.rotate	Logical; whether to include the "Rotate" input for swapping x and y axes (e.g., horizontal bar plots). Default is FALSE.
include.flip	Logical; whether to include the "Flip" input for flipping the axis. Default is FALSE.

Value

A tagList containing the axis input UI elements.

Author(s)

Jared Andrews Jacob Martin

Examples

```
ns <- shiny::NS("plot")
uniform_axes_inputs_ui(ns)
uniform_axes_inputs_ui(ns, include.rotate = TRUE, include.flip = TRUE)
```

---

uniform_legend_inputs_ui

Generate uniform Legend input UI

Description

Creates a standardized tagList of legend styling inputs used across plot modules. Currently exposes the legend title and entry label font sizes so they can be adjusted consistently regardless of plot type.

Usage

```
uniform_legend_inputs_ui(ns, defaults = NULL)
```

Arguments

ns	A namespace function, typically created by NS(id).
defaults	A named list of default values for the inputs.

Value

A tagList containing the legend input UI elements.

Author(s)

Jared Andrews

Examples

```
ns <- shiny::NS("plot1")
uniform_legend_inputs_ui(ns)
uniform_legend_inputs_ui(ns, defaults = list(legend.title.size = 16, legend.text.size = 12))
```

`uniform_lines_inputs_ui` *Generate uniform Lines input UI*

Description

Creates a standardized tagList of line-related inputs (horizontal, vertical, and diagonal lines) for use across plot modules.

Usage

```
uniform_lines_inputs_ui(ns, defaults = NULL, include.fit.lines = FALSE)
```

Arguments

`ns`	A namespace function, typically created by `NS(id)`.
`defaults`	A named list of default values for the inputs.
`include.fit.lines`	Logical; whether to include "line of best fit" and "linear model line" inputs. Only applicable for scatter plots. Default is FALSE.

Value

A `tagList` containing the line input UI elements.

Author(s)

Jared Andrews

Examples

```
ns <- shiny::NS("plot")
uniform_lines_inputs_ui(ns)
uniform_lines_inputs_ui(ns, include.fit.lines = TRUE)
```

uniform_plotly_inputs_ui

Generate uniform Plotly input UI

Description

Creates a standardized tagList of Plotly-specific inputs used across all plot modules. Includes interactive download controls, plot margin adjustments, and user-drawn shape styling for Plotly's drawing tools. (Subplot spacing controls live in each module's "Facet" tab via `.uniform_subplot_spacing_inputs_ui()`.)

Usage

```
uniform_plotly_inputs_ui(ns, defaults = NULL)
```

Arguments

`ns`	A namespace function, typically created by `NS(id)`.
`defaults`	A named list of default values for the inputs.

Value

A `tagList` containing the Plotly input UI elements.

Author(s)

Jared Andrews

Examples

```
ns <- shiny::NS("plot")
uniform_plotly_inputs_ui(ns)
```

updateMultiColorPicker

Update a multiColorPicker input on the client

Description

Change the color values assigned to groups in an existing multiColorPicker input from the server side. You can supply explicit colors, apply a palette by name, or reset the widget back to its initial state.

Usage

```
updateMultiColorPicker(
  session,
  inputId,
  colors = NULL,
  palette = NULL,
  reset = FALSE
)
```

Arguments

session	The Shiny session object, typically session.
inputId	Character. The input id of the multiColorPicker to update.
colors	Optional named character vector of hex colors keyed by group name. Only groups present in the vector will be updated; others remain unchanged. Ignored when palette or reset is provided.
palette	Optional character string giving the name of a palette (as supplied in the widget's palette_options). The palette's colors are applied in order to the widget's groups' color pickers and the palette selector is updated to match. Ignored when reset is TRUE.
reset	Logical. If TRUE, reset the widget to its initial state (colors and selected palette). Overrides colors and palette.

Value

Invisibly returns NULL. Called for its side effect.

Author(s)

Jared Andrews

Examples

```
if (interactive()) {
    library(shiny)
    groups <- c("setosa", "virginica", "versicolor")

    ui <- fluidPage(
        multiColorPicker(
            "species_cols",
            "Species colors",
            groups = groups,
            selected_palette = "dittoColors"
        ),
        actionButton("randomize", "Randomize colors"),
        actionButton("apply_pal", "Apply ggplot2 palette"),
        actionButton("reset_cols", "Reset to initial"),
        verbatimTextOutput("chosen")
    )
```

```
    server <- function(input, output, session) {
        output$chosen <- renderPrint(input$species_cols)

        observeEvent(input$randomize, {
            new_colors <- setNames(
                sprintf("#%06X", sample(0xFFFFFF, length(groups))),
                groups
            )
            updateMultiColorPicker(session, "species_cols", colors = new_colors)
        })

        observeEvent(input$apply_pal, {
            updateMultiColorPicker(session, "species_cols", palette = "ggplot2")
        })

        observeEvent(input$reset_cols, {
            updateMultiColorPicker(session, "species_cols", reset = TRUE)
        })
    }

    shinyApp(ui, server)
}
```

`updateMultiDynamicInput` *Update a multiDynamicInput input on the client*

Description

Replace all rows of an existing `multiDynamicInput()` from the server, or clear it entirely.

Usage

```
updateMultiDynamicInput(session, inputId, elements = NULL, clear = FALSE)
```

Arguments

`session`	The Shiny session object, typically `session`.
`inputId`	Character. The input id of the multiDynamicInput to update.
`elements`	Optional named list of rows (same shape as the return value) to set. Ignored when `clear = TRUE`.
`clear`	Logical. If `TRUE`, remove all rows. Overrides `elements`.

Value

Invisibly returns `NULL`. Called for its side effect.

Author(s)

Jacob Martin

`validate_expression` *Validate a user-provided expression string for safety*

Description

Parses the expression text and walks the AST to ensure it only contains allowed operations (comparisons, logical operators, column references, and literals). Returns the original string if valid, or `NULL` if the input is empty, unparseable, or contains disallowed operations. This is useful when the expression string must be passed through to a downstream function (e.g., `plotthis::BoxPlot(highlight = ...)`) rather than evaluated directly.

Usage

```
validate_expression(expr_text, col_names)
```

Arguments

`expr_text`	Character string containing the expression to validate (e.g., `"group == 'A' & value > 10"`).
`col_names`	Character vector of allowed column/symbol names (typically `names(data)`).

Details

Use this when a module passes a user-typed expression string to an external plotting function that will evaluate it internally. The string is validated but not executed by this function.

Value

The original `expr_text` string if safe, or `NULL`.

Author(s)

Jared Andrews

Examples

```
validate_expression("Sepal.Length > 5", names(iris))
validate_expression("system('echo pwned')", names(iris)) # NULL + warning
validate_expression("", names(iris)) # NULL
```

Index