

Package ‘VizModules’

September 30, 2026

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

Version 0.5.0

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 (>= 0.7.0), htmlwidgets, zip

Encoding UTF-8

LazyData true

Suggests drc, withr, ComplexHeatmap, InteractiveComplexHeatmap, circlize, grid, svglite, 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 8.1.0

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 2026-09-29 22:20:02 UTC

Contents

add_ablines	5
add_hlines	7
add_plot_config	8
add_reference_lines	9
add_vlines	11
adjusted_axis_label	12
adjust_column_values	13
apply_axis_title_to_annotations	14
apply_facet_subplot_spacing	15
apply_legend_inputs	16
apply_legend_styling	17
apply_plotly_newshape	19
apply_render_margins	20
apply_stat_annotations	20
apply_subplot_axis_styling	22
apply_title_layout	23
axis_titles_as_annotations	24
build_facet_annotations	25
build_facet_panel_borders	26
build_model_row_spec	27
clean_facet_dim	28
collect_source_data	29
ComplexHeatmap_HeatmapApp	30
ComplexHeatmap_HeatmapInfoOutputUI	32
ComplexHeatmap_HeatmapInputsUI	33
ComplexHeatmap_HeatmapMainOutputUI	38
ComplexHeatmap_HeatmapOutputUI	39
ComplexHeatmap_HeatmapServer	40
ComplexHeatmap_HeatmapStaticOutputUI	42
ComplexHeatmap_HeatmapSubOutputUI	43
compute_pairwise_stats	44
createModuleApp	46
create_axis_styles	48
create_ggplot_axis_style	50
create_source_download_handler	51
create_stat_annotations	53
dataFilterServer	55
dataFilterUI	57
default_palettes	58
dittoViz_freqPlotApp	59
dittoViz_freqPlotInputsUI	60
dittoViz_freqPlotOutputUI	66

dittoViz_freqPlotServer	66
dittoViz_scatterPlotApp	67
dittoViz_scatterPlotInputsUI	69
dittoViz_scatterPlotOutputUI	75
dittoViz_scatterPlotServer	75
dittoViz_yPlotApp	76
dittoViz_yPlotInputsUI	78
dittoViz_yPlotOutputUI	83
dittoViz_yPlotServer	84
draw_to_png	85
draw_to_svg	86
dumbbellPlot	87
dumbbellPlotApp	91
dumbbellPlotInputsUI	92
dumbbellPlotOutputUI	96
dumbbellPlotServer	97
empty_plot	98
example_bar	98
example_composition	99
example_demographics	100
example_heatmap_column_data	101
example_heatmap_matrix	102
example_iris	103
example_markers	103
example_mtcars	104
example_population	105
example_rnaseq	106
example_sales	107
example_school_earnings	108
example_skills	108
figureBuilderApp	109
figureBuilderServer	111
figureBuilderUI	113
finalize_manual_edits	114
generate_pair_strings	115
get_default	116
get_documentation	117
get_model_backend	117
heatmap_fit_width	118
hide_input	119
is_pure_type	120
linePlot	121
linePlotApp	125
linePlotInputsUI	127
linePlotOutputUI	130
linePlotServer	131
linetype_to_dash	132
list_model_backends	132

moduleGalleryApp	133
module_tack_ui	134
multiColorPicker	135
multiDynamicInput	137
organize_inputs	138
parallelCoordinatesPlot	140
parallelCoordinatesPlotApp	142
parallelCoordinatesPlotInputsUI	143
parallelCoordinatesPlotOutputUI	145
parallelCoordinatesPlotServer	146
parse_numeric_list	147
parse_pair_strings	148
piePlot	148
piePlotApp	151
piePlotInputsUI	153
piePlotOutputUI	155
piePlotServer	155
plotthis_AreaPlotApp	156
plotthis_AreaPlotInputsUI	157
plotthis_AreaPlotOutputUI	161
plotthis_AreaPlotServer	162
plotthis_BarPlotApp	163
plotthis_BarPlotInputsUI	164
plotthis_BarPlotOutputUI	169
plotthis_BarPlotServer	169
plotthis_BoxPlotApp	170
plotthis_BoxPlotInputsUI	171
plotthis_BoxPlotOutputUI	178
plotthis_BoxPlotServer	178
plotthis_DensityPlotApp	179
plotthis_DensityPlotInputsUI	180
plotthis_DensityPlotOutputUI	184
plotthis_DensityPlotServer	185
plotthis_DotPlotApp	186
plotthis_DotPlotInputsUI	187
plotthis_DotPlotOutputUI	190
plotthis_DotPlotServer	191
plotthis_HistogramApp	192
plotthis_HistogramInputsUI	193
plotthis_HistogramOutputUI	197
plotthis_HistogramServer	198
plotthis_SplitBarPlotApp	199
plotthis_SplitBarPlotInputsUI	200
plotthis_SplitBarPlotOutputUI	205
plotthis_SplitBarPlotServer	205
radarPlot	206
radarPlotApp	209
radarPlotInputsUI	211

radarPlotOutputUI	214
radarPlotServer	214
recycle_line_style	215
register_model_backend	216
reset_annotation_inputs	217
reset_axes_inputs	218
reset_axis_title_text	218
reset_legend_inputs	219
reset_lines_inputs	220
reset_plotly_inputs	221
resolve_column_targets	222
resolve_facet_layout	223
resolve_facet_sharing	224
resolve_palette	225
safe_eval_filter	226
safe_resolve_adj_fxn	227
setup_auto_update_logic	228
setup_axis_range	229
setup_group_colors	231
setup_manual_edits	233
setup_reactive_defaults	235
show_input	236
stat_bracket_y_max	237
string_to_linetypes	239
toggle_input_cell	240
uniform_annotation_inputs_ui	241
uniform_axes_inputs_ui	242
uniform_legend_inputs_ui	243
uniform_lines_inputs_ui	244
uniform_plotly_inputs_ui	244
updateMultiColorPicker	245
updateMultiDynamicInput	247
update_viz_select	248
use_vizmodules_skills	249
validate_expression	250
viz_select_input	251

Index**253**

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", "long-dash", "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	<i>Build horizontal line shapes for a plotly figure</i>
------------	---

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", "long-dash", "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	Whether the figure is faceted into panels carrying their own titles: the facet column name(s), or TRUE. NULL, FALSE, an empty vector, or empty strings mean not faceted.

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\(\)](#)).

When the figure is faceted, plot title editing is disabled too. An empty editable title still draws plotly's "Click to enter Plot title" placeholder, which sits on top of the facet panel titles and swallows clicks meant for them.

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	<i>Add reference lines to a plotly figure from Shiny inputs</i>
---------------------	---

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	<i>Build vertical line shapes for a plotly figure</i>
------------	---

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

<code>fig</code>	A plotly figure object. Used to detect subplot axes for faceted plots.
<code>intercepts</code>	Numeric vector. X-axis intercepts for vertical lines.
<code>colors</code>	Character vector. Line colors (hex or named colors).
<code>widths</code>	Numeric vector. Line widths in pixels.
<code>linetypes</code>	Character vector. Line types: "solid", "dashed", "dotted", "dotdash", "long-dash", "twodash".
<code>opacities</code>	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	<i>Build an adjustment-aware axis label</i>
---------------------	---

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 modules apply the adjustments (the `adj.fxn` first, then the recognized adjustment rescales the result), producing labels such as `"z-score(log2(units))"`.

Usage

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

Arguments

<code>base</code>	Character scalar. The base axis label (typically the column name).
<code>adjustment</code>	Character scalar. A recognized data adjustment such as <code>"z-score"</code> or <code>"relative.to.max"</code> . Optional.
<code>adj.fxn</code>	Character scalar. The name of a transformation function such as <code>"log2"</code> or <code>"sqrt"</code> . 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	<i>Adjust numeric column values in a data frame using mathematical transformations</i>
----------------------	--

Description

Transforms the named numeric columns of a data frame the way the plot modules do for their adjustment inputs, adding the transformed values as a new column (original column name + ".adj"). The function is applied first, then an adjustment ("z-score" or "relative.to.max") rescales the result, computed over its finite values. The function 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 function 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,
  x.adjustment = NULL,
  y.adjustment = NULL,
  color.adjustment = 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 <code>safe_resolve_adj_fxn</code> (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 <code>safe_resolve_adj_fxn</code> . 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 <code>safe_resolve_adj_fxn</code> . If NULL or an empty string, color values are left unchanged.

`x.adjustment, y.adjustment, color.adjustment`

Character scalar. "z-score" or "relative.to.max" to rescale that column after its function is applied. If NULL or an empty string, no rescaling is done.

Details

Use this to compute anything drawn over an adjusted plot (axis limits, bracket headroom, fit lines) from the values the plot actually shows, rather than the raw column. Note that dittoViz, given both an `*.adjustment` and an `*.adj.fxn`, applies them in the opposite order; the modules pass it the whole transform as its function instead.

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)

# log10 first, then z-scored: the values the scatter module draws for that pair
mtcars_z <- adjust_column_values(mtcars,
  x.col = "hp", x.adj.fun = "log10", x.adjustment = "z-score"
)
range(mtcars_z$hp.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

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

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

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

Description

Reads the inputs created by `uniform_legend_inputs_ui()` (`legend.show`, `legend.font.family`, `legend.font.color`, `legend.title.size` and `legend.text.size`) and applies them with `apply_legend_styling()`. An input that has not reported yet (`NULL`) leaves its property unchanged, so the legend stays visible until `legend.show` is `FALSE`.

Usage

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

Arguments

fig	A plotly figure object.
input	Shiny input object (or a list) containing the legend fields.
isolate_fn	Function to isolate reactive values. Defaults to shiny::isolate.

Value

The plotly figure with the legend inputs applied.

Author(s)

Jared Andrews

See Also

[uniform_legend_inputs_ui\(\)](#), [reset_legend_inputs\(\)](#), [apply_legend_styling\(\)](#)

Examples

```
fig <- plotly::plot_ly(iris,
  x = ~Sepal.Length, y = ~Sepal.Width,
  color = ~Species, type = "scatter", mode = "markers"
)
legend_input <- list(
  legend.show = TRUE, legend.font.family = "Courier New", legend.font.color = "#333333",
  legend.title.size = 16, legend.text.size = 12
)
apply_legend_inputs(fig, legend_input, isolate_fn = identity)
```

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

Description

Shows or hides the legend and sets the font family, color and sizes of its title and entry labels, so the "Legend" UI inputs behave consistently across plot types. Existing legend settings (orientation, position, and any font property not supplied) are preserved because `plotly::layout()` merges the supplied attributes into the current layout. NULL, NA or blank values are ignored, leaving the corresponding property untouched.

Usage

```

apply_legend_styling(
  fig,
  title.size = NULL,
  text.size = NULL,
  position = NULL,
  font.family = NULL,
  font.color = NULL,
  show = NULL
)

```

Arguments

<code>fig</code>	A plotly figure object.
<code>title.size</code>	Numeric font size for the legend (or colorbar) title, or NULL to leave unchanged.
<code>text.size</code>	Numeric font size for the legend entry labels (or colorbar tick labels), or NULL to leave unchanged.
<code>position</code>	Optional length-2 vector <code>c(x, xanchor)</code> placing the legend horizontally, e.g. <code>c(1.02, "left")</code> . NULL (the default) leaves the position unchanged.
<code>font.family</code>	Character font family for the legend title and entry labels (and colorbar title and ticks), or NULL to leave unchanged.
<code>font.color</code>	Character color for the legend title and entry labels (and colorbar title and ticks), or NULL to leave unchanged.
<code>show</code>	Logical. FALSE hides the legend and every colorbar; TRUE or NULL (the default) leave their visibility unchanged.

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 and visibility do 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 values, and hiding the legend turns each colorbar's `showscale` off. This keeps the "Legend" controls functional for both categorical and continuous legends.

Value

The plotly figure with the requested legend styling applied. Returns the figure unchanged when `fig` is NULL or nothing valid is 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, font.family = "Courier New")
apply_legend_styling(fig, show = FALSE)
```

`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

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

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

<code>fig</code>	A plotly figure object.
<code>input</code>	Shiny input object. Expected to contain <code>margin.t</code> , <code>margin.b</code> , <code>margin.l</code> , and <code>margin.r</code> .

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, raising the y-axis top when the brackets need more room than the requested range gives them.

Usage

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

Arguments

<code>fig</code>	A plotly figure object.
<code>stat_result</code>	List with annotations, shapes, and <code>y.max</code> as returned by <code>create_stat_annotations()</code> .
<code>y.min</code>	Numeric or NULL; minimum y-axis value. If NULL (or not finite), the existing y-axis bottom is kept, falling back to the data minimum.
<code>y.max</code>	Numeric or NULL; the maximum the caller asked for. The drawn top is the larger of this and the height the brackets need. If NULL, the figure's existing top is used for that comparison.

Details

The axis is only ever raised, never lowered: a top asked for through `y.max` (or already on the figure) is kept when it is above the brackets, so turning statistics on cannot silently undo a y-axis maximum the user chose. Reserve the room up front with `stat_bracket_y_max()` and this becomes a no-op.

When `stat_result` came from `create_stat_annotations(free.y = TRUE)`, each panel's y-axis is raised to fit its own brackets and keeps its own bottom, and `y.min/y.max` are ignored: a single limit shared by every panel would undo the free scale.

The range is written straight into the figure's layout. Anything later queued with `plotly::layout()` that sets the axis range will still override it when the figure is built.

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

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

Details

The styling is queued with `plotly::layout()`, which merges it into each axis when the figure is built, so the axes' other properties (range, domain, title text, ...) are kept. Only the styling is queued, never a copy of the axis as it stands, so a property written to the figure's layout after this call (such as a range raised by `apply_stat_annotations()`) is not reverted at build time.

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	<i>Apply plot title styling to a plotly figure</i>
--------------------	--

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

<code>plot</code>	A plotly figure object.
<code>input</code>	Shiny input object containing title font fields.
<code>isolate_fn</code>	Function to isolate reactive values.
<code>title_y</code>	Numeric y position for the plot title in the plotly layout. Defaults to 0.95.
<code>title_x</code>	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,
  axis.title.font = NULL,
  facet.title.font = NULL
)
```

Arguments

<code>facet_levels</code>	Character vector of facet level labels, one per subplot.
<code>x.title</code>	Optional character, shared X-axis title. Default: <code>NULL</code> .
<code>y.title</code>	Optional character, shared Y-axis title. Default: <code>NULL</code> .
<code>title.font.size</code>	Numeric, font size for all annotation text. Default: 14.
<code>nrows</code>	Integer, number of rows the faceted subplots are arranged in. Used to compute per-subplot annotation coordinates for multi-row grids when <code>fig</code> is not supplied. Default: 1.
<code>fig</code>	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 <code>NULL</code> (the default), coordinates are computed from <code>nrows</code> assuming evenly spaced panels filling the full paper area.
<code>title.offset</code>	Numeric fraction of the figure height to place each subplot title above the top of its panel. Default: 0.02.
<code>axis.title.font</code>	Optional named list of plotly font properties (size, color, family) for the shared X/Y axis title annotations. If <code>NULL</code> (the default), they use <code>title.font.size</code> like the facet titles.

facet.title.font

Optional named list of plotly font properties (size, color, family) for the per-panel facet titles. If NULL (the default), they use title.font.size.

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

<code>fig</code>	A plotly figure object whose <code>x\$layout</code> contains the per-panel <code>xaxis*/yaxis*</code> domains (typically after <code>subplot()</code> and <code>apply_facet_subplot_spacing()</code>).
<code>n_facets</code>	Integer, number of facet panels.
<code>showline</code>	Logical, whether to draw border lines. Default TRUE.
<code>mirror</code>	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	<i>Collect plot and source data for download</i>
---------------------	--

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 named list of UI input values (for example `reactiveValuesToList(input)`), or a reactive expression returning one. When `NULL` or when it returns `NULL`, no UI input data is included.

Details

`plot_data` is scoped down from the plot's full source `data.frame` (as returned by `plotly::plotly_data()`) in two ways:

- Columns are limited to those actually mapped in the plot (x, y, color/fill, shape, size, labels/values, facets, etc.), detected by inspecting the built plotly figure's trace attributes and, for columns that don't survive conversion to plotly (namely `split.by/facet.by`), the UI input values.
- Rows are limited to those with complete data across the detected columns, since rows with `NA` in a plotted aesthetic are dropped by the underlying plotting functions.

If no plotted columns can be detected, the full source `data.frame` is returned unchanged.

Value

A named list with elements:

plot The plotly plot object.

plot_data A data.frame of the plot's underlying data, limited to the columns and rows actually rendered (see Details).

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, Jared Andrews

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")
  })

  output$my_plot <- renderPlotly(plot_reactive())
  # collect_source_data() reads reactives, so it has to run inside one.
  inputs_reactive <- reactive(reactiveValuesToList(input))
  output$download_data <- create_source_download_handler(
    reactive(collect_source_data(plot_reactive, inputs_reactive = inputs_reactive))
  )
}

shinyApp(ui, server)

## End(Not run)
```

Description

This function generates a Shiny application with modular `ComplexHeatmap::Heatmap()` components rendered interactively via **InteractiveComplexHeatmap**. 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 the interactive heatmap.

Usage

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

Arguments

<code>data_list</code>	An optional named list of data frames. If <code>NULL</code> (the default), <code>example_heatmap_matrix</code> is used as example data — paired with <code>example_heatmap_column_data</code> unless <code>column_data</code> says otherwise. When <code>column_data</code> is supplied it is attached to the first entry, which becomes <code>list(matrix = , column_annotatons =)</code> .
<code>column_data</code>	An optional data frame of per-sample metadata, enabling column annotations, column splitting, and metadata-aware column filtering. Defaults to <code>example_heatmap_column_data</code> when <code>data_list</code> is also <code>NULL</code> ; pass <code>data_list</code> explicitly to opt out. Attached to the first <code>data_list</code> entry (see <code>ComplexHeatmap_HeatmapServer()</code> 's <code>data</code> parameter for the expected shape — a key column matching the matrix's column names, plus arbitrary annotation columns). When supplied, the app is a minimal single-dataset <code>shinyApp()</code> (no Data Import/Data Table sections) wiring <code>data = list(matrix = <first element of data_list, or example_heatmap_matrix>, column_annotatons = <example_heatmap_column_data>)</code> directly into the module.
<code>defaults</code>	A named list of input IDs and their default values to apply on startup. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> to have the input follow the parent app's state; see <code>setup_reactive_defaults()</code> .
<code>hide.inputs</code>	A character vector of input IDs to hide. Their values are still initialized and used, but the controls are not shown in the UI.
<code>hide.tabs</code>	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 neither `data_list` nor `column_data` is provided, the app launches on the bundled pair — `example_heatmap_matrix` (a simulated gene x sample expression matrix) together with `example_heatmap_column_data` (its per-sample metadata), with `column_key` seeded to "sample". The column-annotation, column-split, and column-filter features are all inert without a metadata table, so this way a bare `ComplexHeatmap_HeatmapApp()` demonstrates the whole module.

Either way the app has the usual **Data Import** section for uploading data and a **Data Table** for filtering the active dataset. Filtering applies to the matrix; any companion metadata table rides along untouched. 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.

Unlike the other modules, this one depends on the Bioconductor packages **ComplexHeatmap**, **InteractiveComplexHeatmap**, and **circlize**, which must be installed (e.g. via `BiocManager::install()`).

This is a convenience wrapper around `createModuleApp()`, which accepts a dataset entry that is a list of tables and filters only the primary one, so the two-table `list(matrix = , column_annotations = )` shape this module's column features need is carried through without a bespoke app (see `ComplexHeatmap_HeatmapServer` data parameter).

Value

A Shiny app object.

Author(s)

Jacob Martin, Jared Andrews

See Also

`ComplexHeatmap::Heatmap()`, `ComplexHeatmap_HeatmapInputsUI()`, `ComplexHeatmap_HeatmapOutputUI()`, `ComplexHeatmap_HeatmapServer()`

Examples

```
library(VizModules)
# Launch on the bundled matrix + its per-sample metadata, so the column
# annotation/split/filter features are all usable:
app <- ComplexHeatmap_HeatmapApp()
if (interactive()) shiny::runApp(app)

# Matrix only, without the per-sample metadata:
app2 <- ComplexHeatmap_HeatmapApp(data_list = list(matrix = example_heatmap_matrix))
if (interactive()) shiny::runApp(app2)
```

ComplexHeatmap_HeatmapInfoOutputUI

Click/brush info output UI component for the ComplexHeatmap module

Description

Renders *only* the output panel showing information about the clicked or brushed cell(s) (e.g. row/column names and value), via `InteractiveComplexHeatmap::HeatmapInfoOutput()`. See `ComplexHeatmap_HeatmapMainOutput` for how the separated output pieces fit together.

Usage

```
ComplexHeatmap_HeatmapInfoOutputUI(id, title = NULL, width = 400, ...)
```

Arguments

<code>id</code>	The ID for the Shiny module. Must match the <code>id</code> used for ComplexHeatmap_HeatmapServer() and any other output pieces for the same heatmap.
<code>title</code>	Optional panel title. <code>NULL</code> (the default) omits the title.
<code>width</code>	Panel width in pixels.
<code>...</code>	Additional arguments passed to InteractiveComplexHeatmap::HeatmapInfoOutput() .

Value

A Shiny UI object for the click/brush info panel.

Author(s)

Jacob Martin

See Also

[InteractiveComplexHeatmap::HeatmapInfoOutput\(\)](#), [ComplexHeatmap_HeatmapOutputUI\(\)](#),
[ComplexHeatmap_HeatmapMainOutputUI\(\)](#), [ComplexHeatmap_HeatmapSubOutputUI\(\)](#)

Examples

```
library(VizModules)
if (requireNamespace("InteractiveComplexHeatmap", quietly = TRUE)) {
  ComplexHeatmap_HeatmapInfoOutputUI("heatmap", title = "Details")
}
```

ComplexHeatmap_HeatmapInputsUI

Input UI components for the ComplexHeatmap 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 [ComplexHeatmap_HeatmapServer\(\)](#) and [ComplexHeatmap_HeatmapOutputUI\(\)](#) functions.

Usage

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

Arguments

<code>id</code>	The ID for the Shiny module.
<code>data</code>	The data frame used for plot generation, <i>or</i> a list with a matrix data frame (required) and a <code>column_annotations</code> data frame (optional) — see ComplexHeatmap_HeatmapServer() 's <code>data</code> parameter for details. Row-annotation choices come from <code>matrix</code> 's columns; column-annotation choices (and the "Column Annotations" tab controls) only appear when <code>column_annotations</code> is supplied.
<code>defaults</code>	A named list of default values for the inputs. An entry may also be a shiny::reactive() or shiny::reactiveVal() ; it is resolved with shiny::isolate() to seed the control, and the module then keeps it live (see setup_reactive_defaults()).
<code>title</code>	An optional title for the UI grid.
<code>columns</code>	Number of columns for the UI grid.

Details

Unlike the other plotly-based modules, this module wraps [ComplexHeatmap::Heatmap\(\)](#) and renders its interactive output via the **InteractiveComplexHeatmap** package. The incoming data frame is converted to a numeric matrix (see the Data / Matrix tab) before being passed to `Heatmap()`.

The inputs are organized into a grid via [organize_inputs\(\)](#), with `columns` controlling the number of columns in the grid. Defaults for each input can be supplied via the `defaults` argument (see [get_default\(\)](#)).

Value

A Shiny `tagList` containing the UI elements.

Plot parameters and defaults

The following [ComplexHeatmap::Heatmap\(\)](#) parameters can be accessed via UI inputs and/or the `defaults` argument:

- `matrix.cols` - Numeric columns forming the matrix (UI: "Matrix Columns", default: all numeric columns)
- `rowname.col` - Column used as row names (UI: "Row Name Column", default: "")
- `name` - Heatmap / legend title (UI: "Heatmap Name", default: "value")
- `na.col` - Color for NA cells (UI: "NA Color", default: "grey")

- `scale` - Z-score the matrix by row, column, or not at all (UI: "Scale", default: "None"). Applied before plotting only — row/column annotation values and the source-data download always use the unscaled matrix. Scaling happens *after* the Filter tab's row/column filters, so a Z-score describes only the rows and columns on screen.
- `reverse.palette` - Reverse the palette (UI: "Reverse Palette", default: FALSE)
- `low_color`, `mid_color`, `high_color` - Colors for the low/mid/high end of the value scale, i.e. what col in `ComplexHeatmap::Heatmap()` is built from via `circlize::colorRamp2()` (UI: "Low/Mid/High Color", default: blue/white/red)
- `min_value`, `mid_value`, `max_value` - Values mapped to `low_color`/`mid_color`/`high_color` (UI: "Min/Mid/Max Value", default: NA = derived from the matrix: `min`, `mean(range(mat))`, and `max` respectively)
- `show_heatmap_legend` - Show the heatmap legend (UI: "Show Legend", default: TRUE)
- `border` - Draw heatmap border (UI: "Border", default: FALSE)
- `cluster_rows` - Cluster rows (UI: "Cluster Rows", default: TRUE)
- `cluster_columns` - Cluster columns (UI: "Cluster Columns", default: TRUE)
- `clustering_distance_rows` - Row distance metric (UI: "Row Distance", default: "euclidean")
- `clustering_distance_columns` - Column distance metric (UI: "Column Distance", default: "euclidean")
- `clustering_method_rows` - Row linkage method (UI: "Row Method", default: "complete")
- `clustering_method_columns` - Column linkage method (UI: "Column Method", default: "complete")
- `show_row_dend` - Show row dendrogram (UI: "Show Row Dendrogram", default: TRUE)
- `show_column_dend` - Show column dendrogram (UI: "Show Column Dendrogram", default: TRUE)
- `row_split_by` - Row split method: "None", "K-means", "Hierarchical", or "Annotation" (UI: "Row Split Method", default: "None"). Only one split mechanism is ever active per axis, which avoids the error `ComplexHeatmap::Heatmap()` raises when both a k-means and a hierarchical split are requested at once.
- `row_split_n` - Number of row groups, used when `row_split_by` is "K-means" or "Hierarchical" (UI: "Row Groups", default: NA; clamped to the number of matrix rows)
- `row_split_cols` - Columns whose values group the rows, used when `row_split_by` is "Annotation" (UI: "Row Split Columns", default: none). Several columns give nested slices, one per observed combination. Values come from the same place row annotation tracks read, so a split and a track on one column always agree.
- `column_split_by` - Column split method: "None", "K-means", "Hierarchical", or "Annotation" (UI: "Column Split Method", default: "None")
- `column_split_n` - Number of column groups, used when `column_split_by` is "K-means" or "Hierarchical" (UI: "Column Groups", default: NA; clamped to the number of matrix columns)
- `column_split_cols` - Columns of the `column_annotations` table whose values group the heatmap columns, used when `column_split_by` is "Annotation" (UI: "Column Split Columns"; only shown when data supplies a `column_annotations` table; default: none)

- `row_gap` - Gap between row slices, mm (UI: "Row Gap (mm)", default: 1)
- `column_gap` - Gap between column slices, mm (UI: "Column Gap (mm)", default: 1)
- `row_title` - Row title (UI: "Row Title", default: "")
- `column_title` - Column title (UI: "Column Title", default: ""). A title, when set, replaces the group names a split would otherwise title its slices with; a %s in it is filled in with each slice's group name. The same goes for `row_title`.
- `show_row_slice_titles` - Title each row slice with its group name when the rows are split and `row_title` is blank (UI: "Show Row Slice Titles", default: TRUE). Applies to every split method: annotation values for "Annotation", cluster numbers for "K-means" and "Hierarchical". Untick it to remove the titles and the space they take.
- `show_column_slice_titles` - The same for columns (UI: "Show Column Slice Titles", default: TRUE)
- `show_row_names` - Show row names (UI: "Show Row Names", default: TRUE)
- `show_column_names` - Show column names (UI: "Show Column Names", default: TRUE)
- `row_names_side` - Row names side (UI: "Row Names Side", default: "right")
- `column_names_side` - Column names side (UI: "Column Names Side", default: "bottom")
- `column_names_rot` - Column name rotation (UI: "Column Name Rotation", default: 90)
- `row_names_fontsize` - Row name font size (UI: "Row Name Size", default: 12)
- `column_names_fontsize` - Column name font size (UI: "Column Name Size", default: 12)
- `title_fontsize` - Row/column title font size (UI: "Title Size", default: 13.2)
- `row_annotations` - Row annotation tracks, built as `ComplexHeatmap::rowAnnotation()` and passed as `left_annotation/right_annotation` per row (UI: "Annotations" tab, "Row Annotations" `multiDynamicInput()` — each row picks a matrix column, a side (Left or Right), and the side and font size of that track's own name label (Bottom or Top, since `ComplexHeatmap` places a row annotation's name above or below it), and whether that track contributes a legend ("Show Legend", default TRUE); default: none). Each row's color control appears just below the list once a column is picked: numeric columns get Low/Mid/High color pickers, everything else gets a `multiColorPicker()` with one color per level. Seed those colors from defaults with a named color vector keyed by the annotation's *column name* — the row names in a `row_annotations/column_annotations` default are not stable, since the client reports rows back as `row1, row2, ...`
- `column_key` - Column in `column_annotations` matched against the matrix's selected column names (UI: "Annotations" tab, "Column Key"; only shown when data supplies a `column_annotations` table)
- `column_annotations` - Column annotation tracks, built as `ComplexHeatmap::columnAnnotation()` and passed as `top_annotation/bottom_annotation` per row (UI: "Annotations" tab, "Column Annotations" `multiDynamicInput()` — each row picks a column, a side (Top or Bottom), the side and font size of that track's own name label (Right or Left, since `ComplexHeatmap` places a column annotation's name beside it), and whether that track contributes a legend ("Show Legend", default TRUE), with the same per-row color controls as row annotations; only shown when data supplies a `column_annotations` table; default: none)

Plot parameters implementing new functionality

The "Filter" tab's two inputs have no `ComplexHeatmap::Heatmap()` equivalent — they narrow the matrix before it is built, so a specific set of genes or samples can be plotted without wiring up the separate `dataFilter` module:

- `row_filter` - Expression keeping only the matching rows (UI: "Row Filter", default: ""). Evaluated against the matrix data frame, so every one of its columns is in scope — annotation columns, the row-name column, and the matrix columns themselves.
- `column_filter` - Expression keeping only the matching matrix columns (UI: "Column Filter", default: ""). Matrix columns are sample names rather than rows of a data frame, so the expression is evaluated against a frame built with one row per selected matrix column: a synthetic column field holding the column name, plus every field of `column_annotatons` joined via `column_key`. `column %in% c("Healthy_1", "Healthy_2")` therefore works with no metadata table at all, while `condition == "Disease"` works as soon as one is supplied. If `column_annotatons` already has a field named `column`, the real one wins and no synthetic is added.

Both are evaluated with `safe_eval_filter()`, which permits comparisons, `&/|/!`, `%in%`, `is.na()`, arithmetic, and the string helpers `grepl`, `startsWith`, `endsWith`, `substr`, `nchar`, `toupper`, `tolower`, and `trimws`. Anything else — a function call outside that list, or a symbol that is not a column — is rejected and reported in the UI rather than evaluated. An expression yielding NA for a row drops that row.

Filtering runs before everything else: `scale`, the annotation tracks, the split methods, and the source download all describe the filtered matrix.

Both inputs are debounced by 700ms, so the heatmap redraws once you pause rather than on every keystroke of a half-typed expression.

Plot parameters not implemented

The following `ComplexHeatmap::Heatmap()` parameters are not exposed because they require R code, objects, or annotations that do not map cleanly to UI inputs: `cell_fun`, `layer_fun`, `post_fun`, `rect_gp`, `border_gp`, custom col mapping functions for the annotation tracks (beyond the Low/Mid/High colors and `multiColorPicker`), `row_order` / `column_order`, `row_labels` / `column_labels`, `jitter`, and all rasterization parameters (`use_raster`, `raster_*`).

Author(s)

Jacob Martin, Jared Andrews

See Also

`ComplexHeatmap::Heatmap()`, `organize_inputs()`, `ComplexHeatmap_HeatmapOutputUI()`, `ComplexHeatmap_HeatmapApp()`

Examples

```
library(VizModules)
ComplexHeatmap_HeatmapInputsUI("heatmap", example_heatmap_matrix)
```

ComplexHeatmap_HeatmapMainOutputUI

Main heatmap output UI component for the ComplexHeatmap module

Description

Renders *only* the original (main) interactive heatmap panel, via `InteractiveComplexHeatmap::originalHeatmapOutput()`. Use this together with `ComplexHeatmap_HeatmapSubOutputUI()` and/or `ComplexHeatmap_HeatmapInfoOutputUI()` to place the three interactive components independently in a custom layout, instead of `ComplexHeatmap_HeatmapOutputUI()` single combined widget. All pieces used for one heatmap must share the same module id as the `ComplexHeatmap_HeatmapServer()` call — no server-side changes are needed to switch between the combined and separated forms.

Usage

```
ComplexHeatmap_HeatmapMainOutputUI(
  id,
  title = NULL,
  width = 450,
  height = 350,
  fit.width = TRUE,
  ...
)
```

Arguments

<code>id</code>	The ID for the Shiny module. Must match the <code>id</code> used for <code>ComplexHeatmap_HeatmapServer()</code> and any other output pieces for the same heatmap.
<code>title</code>	Optional panel title. <code>NULL</code> (the default) omits the title.
<code>width, height</code>	Panel dimensions in pixels.
<code>fit.width</code>	Logical; when <code>TRUE</code> (the default) the panel is scaled once on load to fit the width of its container, making <code>width</code> a starting proportion rather than an absolute size. See <code>ComplexHeatmap_HeatmapOutputUI()</code> .
<code>...</code>	Additional arguments passed to <code>InteractiveComplexHeatmap::originalHeatmapOutput()</code> , e.g. <code>action</code> , <code>response</code> , <code>brush_opt</code> .

Value

A Shiny UI object for the main interactive heatmap panel.

Author(s)

Jacob Martin

See Also

[InteractiveComplexHeatmap::originalHeatmapOutput\(\)](#), [ComplexHeatmap_HeatmapOutputUI\(\)](#),
[ComplexHeatmap_HeatmapSubOutputUI\(\)](#), [ComplexHeatmap_HeatmapInfoOutputUI\(\)](#)

Examples

```
library(VizModules)
if (requireNamespace("InteractiveComplexHeatmap", quietly = TRUE)) {
  ComplexHeatmap_HeatmapMainOutputUI("heatmap", title = "Heatmap")
}
```

ComplexHeatmap_HeatmapOutputUI

Output UI components for the ComplexHeatmap module

Description

This should be placed in the UI where the heatmap should be shown. Unlike the `plotly` modules, the interactive output is provided by [InteractiveComplexHeatmap::InteractiveComplexHeatmapOutput\(\)](#), which supplies its own resize and export controls.

Usage

```
ComplexHeatmap_HeatmapOutputUI(id, resizable = TRUE, fit.width = TRUE, ...)
```

Arguments

<code>id</code>	The ID for the Shiny module.
<code>resizable</code>	Logical; accepted for signature parity with the other module output functions but ignored, since the <code>InteractiveComplexHeatmap</code> widget manages its own sizing.
<code>fit.width</code>	Logical; when <code>TRUE</code> (the default) the widget's panels are scaled once on load so the whole thing fits the width of its container, instead of sitting at the fixed pixel widths baked in by InteractiveComplexHeatmap . <code>width1/width2</code> (and their defaults) then act as the relative widths that get scaled, rather than absolute sizes. Heights are left alone, and the widget's own resize handle and size tab still override the fitted width afterwards. Pass <code>FALSE</code> for fixed pixel widths.
<code>...</code>	Additional arguments passed to InteractiveComplexHeatmap::InteractiveComplexHeatmapOutput() , e.g. <code>layout</code> , <code>compact</code> , <code>width1/height1</code> , <code>title1/title2/title3</code> .

Details

This renders the original heatmap, the selected sub-heatmap, and the click/brush info panel together as one widget, arranged per layout (see [InteractiveComplexHeatmap::InteractiveComplexHeatmapOutput\(\)](#) for the available layout strings, e.g. `"(1-2)|3"`, `"1|(2-3)"`, `"1-2-3"`).

Pass `compact = TRUE` for a smaller footprint (see the "[Compact mode](#)" [article section](#)): the sub-heatmap panel is dropped entirely and the click/brush info floats near the cursor instead of occupying its own static area — equivalent to `response = c(action, "brush-output")`, `output_ui_float = TRUE`,

per `InteractiveComplexHeatmap::InteractiveComplexHeatmapOutput()`'s own docs. layout has nothing left to arrange in compact mode, since only one static panel remains. No server-side change is needed to turn compact mode on or off.

A floating info panel is moved onto <body> by **InteractiveComplexHeatmap** and parked off-screen when idle. It is parked to the *left* here rather than to the right as the package does, since a box parked past the right edge extends the host page's scrollable width by 10,000px.

To place the three components independently anywhere in a custom UI (separate tabs, cards, columns, etc.), use `ComplexHeatmap_HeatmapMainOutputUI()`, `ComplexHeatmap_HeatmapSubOutputUI()`, and `ComplexHeatmap_HeatmapInfoOutputUI()` instead of this function. Use one approach or the other, not both, for the same module id. Compact mode is specific to this combined widget: the separated pieces (`originalHeatmapOutput()`, `subHeatmapOutput()`, `HeatmapInfoOutput()`, which back the three functions above) don't accept a compact argument at all.

Value

A Shiny UI object for the interactive heatmap.

Author(s)

Jacob Martin, Jared Andrews

See Also

`InteractiveComplexHeatmap::InteractiveComplexHeatmapOutput()`, `ComplexHeatmap_HeatmapMainOutputUI()`, `ComplexHeatmap_HeatmapSubOutputUI()`, `ComplexHeatmap_HeatmapInfoOutputUI()`

Examples

```
library(VizModules)
if (requireNamespace("InteractiveComplexHeatmap", quietly = TRUE)) {
  # Default combined widget:
  ComplexHeatmap_HeatmapOutputUI("heatmap")
  # Same widget, main heatmap on its own row above sub-heatmap + info:
  ComplexHeatmap_HeatmapOutputUI("heatmap", layout = "1|(2-3)")
  # Compact: no sub-heatmap panel, click/brush info floats near the cursor
  ComplexHeatmap_HeatmapOutputUI("heatmap", compact = TRUE)
  # Fixed pixel widths, ignoring the container:
  ComplexHeatmap_HeatmapOutputUI("heatmap", fit.width = FALSE)
}
```

ComplexHeatmap_HeatmapServer

Server logic for the ComplexHeatmap module

Description

Server logic for the ComplexHeatmap module

Usage

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

Arguments

<code>id</code>	The ID for the Shiny module.
<code>data</code>	A reactive yielding either a data frame (the matrix's columns plus any row-annotation columns, all together — the original, single-table behavior) or <code>list(matrix = <data.frame>, column_annotations = <data.frame>)</code> to additionally enable column annotations, where <code>column_annotations</code> is a per-sample metadata table keyed by a column matching the matrix's selected column names (see the "Column Key" input). A <code>NULL</code> value (or a list missing <code>matrix</code>) is treated as "not ready yet" and the module waits for data.
<code>hide.inputs</code>	A character vector of input IDs to hide. These will still be initialized and their values used, but the user will not be able to see/adjust them in the UI.
<code>hide.tabs</code>	A character vector of tab names to hide. Inputs in these tabs will still be initialized and used, but not shown in the UI.
<code>defaults</code>	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 UI function. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> , in which case the input tracks it as the parent app's state changes; see <code>setup_reactive_defaults()</code> .

Details

The incoming data frame is converted to a numeric matrix using the selected matrix columns (and optional row-name column). Row/column annotation tracks configured on the "Annotations" tab are built as `ComplexHeatmap::rowAnnotation()/ComplexHeatmap::columnAnnotation()` (one per side) and passed as `left_annotation/right_annotation/ top_annotation/bottom_annotation`. The heatmap is built with `ComplexHeatmap::Heatmap()` and registered for interactivity with `InteractiveComplexHeatmap` which draws it onto its own device to capture the interactive widget.

Both **ComplexHeatmap** and **InteractiveComplexHeatmap** are Bioconductor packages and are only required at runtime for this module; they are guarded with `requireNamespace()`.

Value

The `moduleServer` function for the `ComplexHeatmap` module. The returned reactive yields the source-download bundle (matrix data + inputs).

Author(s)

Jacob Martin, Jared Andrews

See Also

[ComplexHeatmap::Heatmap\(\)](#), [ComplexHeatmap_HeatmapInputsUI\(\)](#), [ComplexHeatmap_HeatmapOutputUI\(\)](#), [ComplexHeatmap_HeatmapApp\(\)](#)

ComplexHeatmap_HeatmapStaticOutputUI

Static (non-interactive) heatmap output UI component for the ComplexHeatmap module

Description

Renders the heatmap as a plain [shiny::plotOutput\(\)](#) instead of an **InteractiveComplexHeatmap** widget. The same [ComplexHeatmap_HeatmapServer\(\)](#) call backs both, so switching between them needs no server-side change – use this function *or* the interactive output functions for a given module id, not both.

Usage

```
ComplexHeatmap_HeatmapStaticOutputUI(
  id,
  resizable = TRUE,
  width = "100%",
  height = "100%"
)
```

Arguments

<code>id</code>	The ID for the Shiny module. Must match the <code>id</code> used for ComplexHeatmap_HeatmapServer() .
<code>resizable</code>	Logical; whether to wrap the plot in a resizable container. Unlike ComplexHeatmap_HeatmapOutputUI() , this is honoured, since a <code>plotOutput</code> has no resize handle of its own.
<code>width, height</code>	Passed to shiny::plotOutput() . The defaults fill the containing element, so the heatmap follows its container's size.

Details

What is given up is the widget's interactivity: cell hover/click, the sub-heatmap zoom, and the brush info panel. What is gained is a panel with no chrome of its own. **InteractiveComplexHeatmap** draws a grey border around the heatmap panel, a control tab strip beneath it, and sizes itself in fixed pixels; none of that can be switched off through an argument, since the border is set by an id selector in that package's own stylesheet. A `plotOutput` has none of it and fills its container at whatever width and height say, which is what a figure panel wants – it is how the ComplexHeatmap module appears in the Figure Builder (see [figureBuilderServer\(\)](#)).

Unlike the interactive output, this needs only **ComplexHeatmap** itself, not **InteractiveComplexHeatmap**.

Value

A Shiny UI object for the static heatmap.

Author(s)

Jared Andrews

See Also

[ComplexHeatmap_HeatmapOutputUI\(\)](#) for the interactive widget, [ComplexHeatmap_HeatmapServer\(\)](#)

Examples

```
library(VizModules)
if (requireNamespace("ComplexHeatmap", quietly = TRUE)) {
  ComplexHeatmap_HeatmapStaticOutputUI("heatmap")
  # Fixed size, no resize handle:
  ComplexHeatmap_HeatmapStaticOutputUI("heatmap",
    resizable = FALSE, width = "600px", height = "400px"
  )
}
```

ComplexHeatmap_HeatmapSubOutputUI

Sub-heatmap output UI component for the ComplexHeatmap module

Description

Renders *only* the selected sub-heatmap panel (the zoomed-in view of a brushed/selected region), via [InteractiveComplexHeatmap::subHeatmapOutput\(\)](#). See [ComplexHeatmap_HeatmapMainOutputUI\(\)](#) for how the separated output pieces fit together.

Usage

```
ComplexHeatmap_HeatmapSubOutputUI(
  id,
  title = NULL,
  width = 400,
  height = 350,
  fit.width = TRUE,
  ...
)
```

Arguments

<code>id</code>	The ID for the Shiny module. Must match the <code>id</code> used for ComplexHeatmap_HeatmapServer() and any other output pieces for the same heatmap.
<code>title</code>	Optional panel title. NULL (the default) omits the title.
<code>width, height</code>	Panel dimensions in pixels.
<code>fit.width</code>	Logical; when TRUE (the default) the panel is scaled once on load to fit the width of its container, making width a starting proportion rather than an absolute size. See ComplexHeatmap_HeatmapOutputUI() .
<code>...</code>	Additional arguments passed to InteractiveComplexHeatmap::subHeatmapOutput() .

Value

A Shiny UI object for the sub-heatmap panel.

Author(s)

Jacob Martin

See Also

[InteractiveComplexHeatmap::subHeatmapOutput\(\)](#), [ComplexHeatmap_HeatmapOutputUI\(\)](#), [ComplexHeatmap_HeatmapInfoOutputUI\(\)](#)

Examples

```
library(VizModules)
if (requireNamespace("InteractiveComplexHeatmap", quietly = TRUE)) {
  ComplexHeatmap_HeatmapSubOutputUI("heatmap", title = "Selected region")
}
```

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

<code>df</code>	Data frame containing the data.
<code>x</code>	Character; column name of the categorical x-axis variable.
<code>y</code>	Character; column name of the numeric response variable.
<code>pairs</code>	List of length-2 character vectors specifying group pairs to test. If <code>NULL</code> (default), tests all unique pairwise combinations.
<code>test</code>	Character; statistical test to use. One of <code>"wilcox.test"</code> , <code>"t.test"</code> , <code>"kruskal.test"</code> , or <code>"anova"</code> .
<code>p.adjust.method</code>	Character; method for p-value adjustment via <code>stats::p.adjust()</code> . Default <code>"holm"</code> .
<code>paired</code>	Logical; whether to perform paired tests (only for <code>"wilcox.test"</code> and <code>"t.test"</code>). Default <code>FALSE</code> .
<code>group.by</code>	Character or <code>NULL</code> ; column for nested grouping. When set, comparisons are made between levels of <code>group.by</code> within each level of <code>x</code> .
<code>facet.by</code>	Character or <code>NULL</code> ; column for faceting. When set and <code>per.facet = TRUE</code> , tests run independently per facet panel.
<code>per.facet</code>	Logical; if <code>TRUE</code> and <code>facet.by</code> is set, run tests independently per facet panel. Default <code>TRUE</code> .
<code>sig.threshold</code>	Numeric; significance threshold for * vs ns. P-values at or below this are labeled *; above are labeled ns. Default 0.05. See <code>sig.levels</code> for the multi-star thresholds.
<code>sig.levels</code>	Named numeric vector; upper p-value bounds for multi-star significance symbols. Names are the displayed symbols and values are the thresholds. Default <code>c("****" = 0.0001, "***" = 0.001, "**" = 0.01)</code> . 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",
    primary.table = NULL,
    sidebar.width = 4
  )

```

Arguments

<code>inputs_ui_fn</code>	A function with signature <code>function(id, data, ...)</code> that returns module input UI elements (e.g. <code>plotthis_BarPlotInputsUI()</code>).
<code>output_ui_fn</code>	A function with signature <code>function(id)</code> that returns the module's output UI (e.g. <code>plotthis_BarPlotOutputUI()</code>).
<code>server_fn</code>	A function with signature <code>function(id, data, ...)</code> that drives the module server logic (e.g. <code>plotthis_BarPlotServer()</code>).
<code>data_list</code>	A named list of datasets. Each element is either a data frame, or a named list of data frames for a module that needs companion tables alongside the one being filtered — e.g. <code>list(matrix = , column_annotatations =)</code> for <code>ComplexHeatmap_Heatmap</code> . Only the primary table (see <code>primary.table</code>) is filtered and shown in the Data Table; the rest are passed through to the module untouched.
<code>defaults</code>	A named list of ui ids and their default values that can change the ui default settings on startup. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> to have the input follow app state; see <code>setup_reactive_defaults()</code> .
<code>hide.inputs</code>	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 <code>server_fn</code> when it accepts a <code>hide.inputs</code> argument.
<code>hide.tabs</code>	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 <code>server_fn</code> when it accepts a <code>hide.tabs</code> argument.
<code>show.table</code>	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.
<code>title</code>	A character string used as the page title (default: "VizModules App").
<code>primary.table</code>	For a <code>data_list</code> entry that is a list of tables, the name of the one to filter and show in the Data Table. Defaults to the first data frame in the entry. Ignored for entries that are a plain data frame.
<code>sidebar.width</code>	Bootstrap column width (1-11) for the controls sidebar; the plot area takes the rest. Raise the plot's share for a module whose output needs room, e.g. <code>sidebar.width = 3</code> for the heatmap.

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

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

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

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, images of it, and its supporting data into a single .zip archive.

Usage

```
create_source_download_handler(
  data_list,
  filename_base = "source_data",
  images = TRUE,
  output_id = "download.source",
  session = getDefaultReactiveDomain()
)
```

Arguments

<code>data_list</code>	A reactive returning either a single summary list produced by <code>collect_source_data()</code> (with elements <code>plot</code> , <code>plot_data</code> , <code>stats</code> , and <code>inputs</code>), 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.
<code>filename_base</code>	<code>character(1)</code> . Base name for the downloaded .zip file without extension. The final filename takes the form <code><filename_base>_<Sys.Date()>.zip</code> .
<code>images</code>	<code>logical(1)</code> . Whether to include .svg and .png images of each plot. Requires the button to carry the markup <code>module_tack_ui()</code> gives it; a hand-rolled <code>shiny::downloadButton()</code> without it simply gets no images.
<code>output_id</code>	<code>character(1)</code> . The id this handler is assigned to within its module, used to find the images the browser sends. Only needs changing if the handler is assigned to something other than <code>output\$download.source</code> .
<code>session</code>	The module session. Defaults to the calling module's, which is what every in-package call site wants.

Details

The archive holds, per plot: the interactive plot as self-contained HTML (`<name>_plot.html`), an `<name>_plot.svg` and `<name>_plot.png` of it, and CSVs of the plot data, the statistics, and the UI inputs.

The images are photographed in the browser, off the graph the user is looking at, so they carry every edit made after the figure was built – reference lines, statistical brackets, restyled axes and legends, dragged annotations. The capture happens between the button's click and the download

itself, which is why the button pauses briefly before the archive arrives. When it cannot be done – the capture fails, the round trip times out, the plot sits on a hidden tab – the archive still downloads, without the images.

A module whose output is not a plotly graph has nothing for the browser to photograph and draws itself instead, by putting a `vector_svg` and/or `raster_png` function of (width, height, res) on its summary list, or on the reactive passed as `data_list`. `draw_to_svg()` and `draw_to_png()` build one from any grid or base drawing; `ComplexHeatmap_HeatmapServer()` is the worked example.

A summary may also carry `svg_key`, naming the captured image that belongs to it. The Figure Builder uses this because the browser knows a panel only by its id, while the summary is named after the panel's display label.

One caveat worth passing on to users: a plot drawn with WebGL (the dittoViz scatter plot's WebGL toggle) can only be photographed as a raster, so its points arrive as an embedded image inside an otherwise vector SVG. Turning WebGL off gives a fully editable file.

Value

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

Author(s)

Jacob Martin, Jared Andrews

See Also

`collect_source_data()`, `module_tack_ui()`

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")
  })

  output$my_plot <- renderPlotly(plot_reactive())
  # collect_source_data() reads reactives, so it has to run inside one.
  inputs_reactive <- reactive(reactiveValuesToList(input))
  output$download_data <- create_source_download_handler(
    reactive(collect_source_data(plot_reactive, inputs_reactive = inputs_reactive))
  )
}
```

```
}  
  
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,  
  dodge.width = 1,  
  free.y = FALSE  
)
```

Arguments

<code>stats_df</code>	Data frame from <code>compute_pairwise_stats()</code> .
<code>fig</code>	A plotly figure object. Used to detect subplot axis pairs for faceted plots.

<code>df</code>	The data frame the plot was drawn from, holding the values as they are plotted: if the plot transformed a column (e.g. <code>dittoViz</code> 's <code>var.adjustment/var.adj.fxn</code>), pass the transformed values (see adjust_column_values()), or the brackets are placed in a different coordinate space from the data. Non-finite values are ignored.
<code>x</code>	Character; x-axis column name.
<code>y</code>	Character; y-axis column name.
<code>display</code>	Character; what to display: <code>"p.adj"</code> , <code>"p.value"</code> , or <code>"symbol"</code> . Default <code>"p.adj"</code> .
<code>hide.ns</code>	Logical; hide non-significant results. Default <code>FALSE</code> .
<code>sig.threshold</code>	Numeric; significance threshold for determining non-significant results. Default <code>0.05</code> .
<code>line.color</code>	Character; color for bracket lines. Default <code>"#000000"</code> .
<code>line.width</code>	Numeric; width of bracket lines. Default <code>1</code> .
<code>bracket.style</code>	Character; <code>"capped"</code> for ggpubr-style brackets with vertical ticks, or <code>"flat"</code> for a single horizontal line. Default <code>"capped"</code> .
<code>group.by</code>	Character or <code>NULL</code> ; nested grouping column.
<code>facet.by</code>	Character or <code>NULL</code> ; faceting column.
<code>x.order</code>	Character vector; order of x-axis categories. If <code>NULL</code> , derived from unique values of <code>x</code> column.
<code>font.size</code>	Numeric; size of annotation text. Default <code>12</code> .
<code>step.increase</code>	Numeric; fraction of y-range for spacing between successive brackets. Default <code>0.06</code> .
<code>text.bump</code>	Numeric; fraction of y-range for vertical distance of text above the bracket line. Default <code>0.04</code> .
<code>bracket.inset</code>	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 <code>0.025</code> .
<code>dodge.width</code>	Numeric; width the <code>group.by</code> levels at one <code>x</code> category are dodged across, matching the dodge the plot was built with. Brackets between two <code>group.by</code> levels are placed on the same slot centres the boxes sit on, so this has to be the plot's dodge or they will not line up. Default <code>1</code> .
<code>free.y</code>	Logical; whether each facet panel has its own y scale (e.g. <code>scales = "free_y"</code>). Each panel's brackets are then stacked above that panel's own data rather than above the tallest panel's, and comparisons pooled across facets are drawn on every panel at that panel's height. Ignored when the figure has no facet panels. Default <code>FALSE</code> .

Details

The values in `y` are expected to be the values drawn on the y-axis, so the plot must show them running up the y-axis. A plot whose values run along the x-axis (a rotated box plot or a ridge plot, say) has no room for vertical brackets; test it with [compute_pairwise_stats\(\)](#) but do not draw brackets.

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.

y.min Numeric; the smallest finite value of y.

y.range.by.axis Under free.y, a named list of c(min, max) ranges keyed by y-axis reference ("y", "y2", ...): the panel's data minimum and the top its brackets need. NULL otherwise.

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)
```

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,
  col.visibility = FALSE,
  hide.columns = NULL,
  filter.max.options = 50
)
```

Arguments

<code>id</code>	The ID for the Shiny module. Must match the <code>id</code> used in <code>dataFilterUI()</code> .
<code>data</code>	A reactive containing the data frame to display and filter. Values that are not data frames are coerced with <code>as.data.frame()</code> ; a <code>NULL</code> value is treated as "not ready yet" and the table waits for data.
<code>factor.char.cols</code>	Logical. When <code>TRUE</code> , all character columns in <code>data</code> 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. Note that DT serialises every level of a factor column into the page, so this is best avoided for columns with very many distinct values. Defaults to <code>TRUE</code> .
<code>page.length</code>	Integer. The default number of rows shown per page. Defaults to 10.
<code>col.visibility</code>	Logical. When <code>TRUE</code> , adds a DT "Columns" button (Buttons extension <code>colvis</code>) so users can show/hide individual columns. Defaults to <code>FALSE</code> .
<code>hide.columns</code>	Character vector of column names (or a numeric vector of column positions) to hide when the table is first drawn. Hidden columns get no filter box, which keeps the interface focused on the columns that matter. They are still present in the returned data, so plotting modules can use them. Set <code>col.visibility = TRUE</code> if users should be able to bring them back; otherwise they stay hidden. Names that do not occur in data are ignored with a warning. Defaults to <code>NULL</code> (show every column).
<code>filter.max.options</code>	Integer. The maximum number of options a factor column's filter dropdown renders at once. Typing in the box narrows the list, so a low cap keeps high-cardinality columns usable. Defaults to 50.

Value

A reactive expression that evaluates to the filtered subset of data based on the current DT selection/filter state. All columns are retained, including any hidden via `hide.columns`. 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\(\)](#), [resolve_column_targets\(\)](#)

Examples

```
library(shiny)
library(VizModules)

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

server <- function(input, output, session) {
  data <- reactive(iris)
  # Petal columns are hidden on load, but the "Columns" button lets users
  # bring them back, and they remain in the returned data either way.
  filtered <- dataFilterServer("filter", data,
    factor.char.cols = TRUE,
    hide.columns = c("Petal.Length", "Petal.Width"),
    col.visibility = 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	<i>Color palette options for palettePicker</i>
------------------	--

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_freqPlotApp *Create an example Modular freqPlot Shiny Application*

Description

This function generates a Shiny application with modular `dittoViz::freqPlot()` 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 frequency plot.

Usage

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

Arguments

<code>data_list</code>	An optional named list of data frames. If NULL (the default), the module's example dataset is used, along with its showcase defaults.
<code>defaults</code>	A named list of input IDs and their default values to apply on startup. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> to have the input follow the parent app's state; see <code>setup_reactive_defaults()</code> .
<code>hide.inputs</code>	A character vector of input IDs to hide. Their values are still initialized and used, but the controls are not shown in the UI.
<code>hide.tabs</code>	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 on `example_composition` (twelve donors nested inside two conditions, the shape `dittoViz::freqPlot()` needs to compare per-sample frequencies across groups) with the settings the module gallery (`moduleGalleryApp()`) opens this module on, so its main features are on show from the start; any defaults you pass are applied over those. 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::freqPlot(), dittoViz_freqPlotInputsUI(), dittoViz_freqPlotOutputUI(), dittoViz_freqPlotServer()`**Examples**

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

# The same example data, as raw cell counts and without the statistics:
app2 <- dittoViz_freqPlotApp(defaults = list(scale = "count", stats.enabled = FALSE))
if (interactive()) runApp(app2)
```

`dittoViz_freqPlotInputsUI`*Input UI components for the freqPlot 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_freqPlotServer()` and `dittoViz_freqPlotOutputUI()` functions.

Usage

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

Arguments

<code>id</code>	The ID for the Shiny module.
<code>data</code>	The data frame used for plot generation.
<code>defaults</code>	A named list of default values for the inputs. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> ; it is resolved with <code>shiny::isolate()</code> to seed the control, and the module then keeps it live (see <code>setup_reactive_defaults()</code>).
<code>title</code>	An optional title for the UI grid.
<code>columns</code>	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::freqPlot()` can be set via these inputs, so see the help for that function for an exhaustive list.

Unlike most modules here, this one does not plot columns of the incoming data. It tabulates how often each level of "Frequency Of" occurs within each sample and plots *those* per-sample frequencies, one facet per level. Axis limits, statistics, the point annotations and the source download therefore all describe the summarised frequency table, not the input rows.

Under a free y facet scale ("free", "free_y") the Y Axis Min/Max are not applied and each panel's significance brackets sit above that panel's own data. With a ridge plot among the plot types the values run along the x-axis, so no brackets are drawn; the test results are still included in the source download.

Because the frequencies are per-sample, "Sample" must nest inside "Group By": each sample has to carry exactly one group value (and one color value, when "Color By" is set), or `dittoViz::freqPlot()` errors. "Group By" and "Color By" are therefore chosen freely and "Sample" is narrowed to the columns that nest inside them, which makes an erroring combination unselectable. With no sample column each group collapses to a single point per facet, which is rarely what is wanted.

Value

A Shiny tagList containing the UI elements

Plot parameters not implemented or with altered functionality

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

- `xlab` - X-axis label (plotly allows interactive editing)
- `ylab` - Y-axis label (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)
- `split.by` - Not a parameter of `freqPlot()`; it always facets on the frequency variable's levels. Use `vars.use` ("Levels To Show") to choose which of those facets are drawn
- `rows.use` - Row subset to plot (use the app's data table filter instead)
- `data.out`, `data.only` - Return the summary rather than the plot (the module summarises internally; use the Source Download button for the table)
- `do.hover` - Always TRUE, so the frequency, count and sample are readable on hover
- `colors` - Integer index/order into `color.panel` (managed via the palette UI)
- `color.panel` - Managed internally via the palette selection UI

- `y.breaks` - Custom continuous-axis breaks (not implemented)
- `x.labels` - Override group labels (not implemented)
- `x.reorder` - Reorder x-axis groups (not implemented)
- `x.labels.rotate` - Rotate group labels (handled by the Axes tab tick-angle controls)
- `var.labels.rename` - Rename facet labels (not implemented)
- `var.labels.reorder` - Reorder facet labels (not implemented)
- `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
- `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`)
- `vlnplot.quantiles` - Violin quantiles (doesn't translate to plotly)
- `jitter.position.dodge` - Jitter position dodge (calculated from `boxgap`)

Plot parameters and defaults

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

- `var` - Variable whose per-sample frequency is tabulated, one facet per level (UI: "Frequency Of", default: 1st low-cardinality categorical variable)
- `sample.by` - Sample identifier the frequencies are computed within (UI: "Sample", default: "", i.e. one point per group)
- `group.by` - Grouping variable forming the x-axis (UI: "Group By", default: 1st categorical variable that is constant within each sample)
- `color.by` - Coloring variable (UI: "Color By", default: "", which follows `group.by`)
- `vars.use` - Which levels of `var` get a facet (UI: "Levels To Show", default: "", i.e. all)
- `scale` - Plot frequencies as proportions or raw counts (UI: "Scale", default: "percent")
- `max.normalize` - Scale each facet to its own maximum (UI: "Max Normalize", default: FALSE)
- `plots` - Plot types to show (UI: "Plots", default: `c("boxplot", "jitter")`)
- `min` - Y-axis minimum (UI: "Y Axis Min", auto-calculated from the frequency table)
- `max` - Y-axis maximum (UI: "Y Axis Max", auto-calculated from the frequency table)
- `split.nrow` - Number of facet rows (UI: "Rows", default: NA)
- `split.ncol` - Number of facet columns (UI: "Columns", default: NA)
- `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)
- `hover.round.digits` - Hover value rounding (UI: "Hover Round Digits", default: 5)
- `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")
- `boxplot.show.outliers` - Show boxplot outliers (UI: "Show Outliers", default: FALSE)
- `boxplot.color` - Boxplot outline color (UI: "Boxplot Color", default: "#000000")
- `boxplot.fill` - Fill boxplot (UI: "Fill Boxplot", default: TRUE)
- `boxplot.linewidth` - Boxplot line weight (UI: "Boxplot Line Weight", default: 0.5)
- `vlnplot.linewidth` - 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.linewidth` - 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: NA)
- `legend.show` - Show the legend (UI: "Show Legend", default: TRUE)

Parameters controlling additional functionality

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

- `boxmode` - Always "overlay": the boxes carry explicit x positions matching ggplot's dodge, so plotly.js is not asked to dodge them
- `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)
- `legend.font.family` - Font family of the legend title and labels (UI: "Legend Font", default: "Arial")
- `legend.font.color` - Font color of the legend title and labels (UI: "Legend Font Color", default: "#000000")
- `legend.title.size` - Legend title font size (UI: "Legend Title Size", default: 14)
- `legend.text.size` - Legend entry label font size (UI: "Legend Text Size", default: 12)
- `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")
- `palette.colours` - Named character vector mapping group levels to colors, e.g. `c(A = "#FF0000", B = "blue")` (UI: "Plot colors"). Seeds the picker; unnamed groups fall back to the default palette and user edits take precedence.

- `annotate.by` - Column whose values identify and label jitter points (UI: "Annotate By", default: ""). Restricted to the columns carried in the plot's hover text, which for this plot are the sample and color columns
- `highlight.points` - Values from the `annotate.by` column to highlight (UI: "Points to Highlight", default: ""). Values containing spaces (e.g. "CD4 T") must be separated by commas or new lines
- `highlight.color` - Fill color for highlighted points (UI: "Highlight Fill", default: "#00FFF7")
- `highlight.size` - Size of highlighted points (UI: "Highlight Size", default: 7)
- `highlight.border.color` - Border color for highlighted points (UI: "Highlight Border Color", default: "#000000")
- `highlight.border.width` - Border width for highlighted points (UI: "Highlight Border Width", default: 1)
- `highlight.auto.annotate` - Label highlighted points automatically (UI: "Auto-annotate Highlights", default: TRUE)
- `annotation.color` - Annotation text color (UI: "Annotation Color", default: "black")
- `annotation.ax` - Horizontal label offset in pixels (UI: "Annotation X Offset", default: 20)
- `annotation.ay` - Vertical label offset in pixels (UI: "Annotation Y Offset", default: -20)
- `annotation.size` - Annotation font size (UI: "Annotation Size", default: 10)
- `annotation.showarrow` - Draw an arrow from label to point (UI: "Show Arrow", default: TRUE)
- `annotation.arrowcolor` - Annotation arrow color (UI: "Arrow Color", default: "black")
- `annotation.arrowhead` - Annotation arrowhead style (UI: "Arrowhead Style", default: 2)
- `annotation.arrowwidth` - Annotation arrow line width (UI: "Arrow Linewidth", default: 1.5)
- `stats.enabled` and the other `stat.*` parameters - Pairwise testing of the per-sample frequencies between x-axis groups. Tests always run within each facet, since frequencies of different levels are not comparable quantities; the "Per Facet Panel" control is therefore hidden and forced on. `stat.pairs` takes a character vector of "A vs B" strings naming levels of `group.by` (or of `color.by`, when it differs), e.g. "Healthy vs Disease"; left unset, every pair is tested

Author(s)

Jared Andrews

See Also

`dittoViz::freqPlot()`, `organize_inputs()`, `dittoViz_freqPlotOutputUI()`, `dittoViz_freqPlotServer()`, `dittoViz_freqPlotApp()`

Examples

```
library(VizModules)
dittoViz_freqPlotInputsUI("freqPlot", example_composition)
```

```
dittoViz_freqPlotOutputUI
```

Output UI components for the freqPlot module

Description

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

Usage

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

Arguments

<code>id</code>	The ID for the Shiny module.
<code>resizable</code>	Logical; when TRUE (the default) the plot output is wrapped in <code>shinyjquery::jquery_resizable()</code> 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 `freqPlot`

Author(s)

Jared Andrews

See Also

`dittoViz::freqPlot()`, `dittoViz_freqPlotInputsUI()`, `dittoViz_freqPlotServer()`, `dittoViz_freqPlotApp()`

```
dittoViz_freqPlotServer
```

Server logic for freqPlot module

Description

Server logic for `freqPlot` module

Usage

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

Arguments

<code>id</code>	The ID for the Shiny module.
<code>data</code>	A reactive containing the data frame to plot. Values that are not data frames are coerced with <code>as.data.frame()</code> ; a NULL value is treated as "not ready yet" and the module waits for data.
<code>hide.inputs</code>	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.
<code>hide.tabs</code>	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.
<code>defaults</code>	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. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> , in which case the input tracks it as the parent app's state changes; see <code>setup_reactive_defaults()</code> .

Value

The `moduleServer` function for the `freqPlot` module.

Author(s)

Jared Andrews

See Also

`dittoViz::freqPlot()`, `dittoViz_freqPlotInputsUI()`, `dittoViz_freqPlotOutputUI()`, `dittoViz_freqPlotApp()`

`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

<code>data_list</code>	An optional named list of data frames. If NULL (the default), the module's example dataset is used, along with its showcase defaults.
<code>defaults</code>	A named list of input IDs and their default values to apply on startup. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> to have the input follow the parent app's state; see <code>setup_reactive_defaults()</code> .
<code>hide.inputs</code>	A character vector of input IDs to hide. Their values are still initialized and used, but the controls are not shown in the UI.
<code>hide.tabs</code>	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 on `example_sales` with the settings the module gallery (`moduleGalleryApp()`) opens this module on, so its main features are on show from the start; any defaults you pass are applied over those. 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

<code>id</code>	The ID for the Shiny module.
<code>data</code>	The data frame used for plot generation.
<code>defaults</code>	A named list of default values for the inputs. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> ; it is resolved with <code>shiny::isolate()</code> to seed the control, and the module then keeps it live (see <code>setup_reactive_defaults()</code>).
<code>title</code>	An optional title for the UI grid.
<code>columns</code>	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.

Source data note: When source data is downloaded with faceting applied and `split.show.all.others = TRUE`, values will be duplicated due to them being shown in every panel.

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. "z-score(log2(units))"; plotly allows interactive editing)
- `ylab` - Y-axis label (auto-generated to reflect any applied Y adjustment, e.g. "z-score(log2(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` - Named character vector mapping `color.by` levels to colors, e.g. `c(setosa = "#FF0000", virginica = "blue")` (UI: `color.panel.ui`). Seeds the picker; unnamed groups fall back to the default palette and user edits take precedence.
- `single.point.color` - Point color used when `color.by` is unset (UI: "Point color", default: "#000000")
- `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` - Values from the `annotate.by` column to highlight (UI: "Points to Highlight", default: ""). Values containing spaces (e.g. "CD4 T") must be separated by commas or new lines
- `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.show` - Show the legend, colorbar and size legend (UI: "Show Legend", default: TRUE)
- `legend.color.title` - Color legend title (UI: "Legend Title", default: "make")
- `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")
- `legend.font.family` - Font family of the legend title and labels (UI: "Legend Font", default: "Arial")
- `legend.font.color` - Font color of the legend title and labels (UI: "Legend Font Color", default: "#000000")
- `legend.title.size` - Legend title font size (UI: "Legend Title Size", default: 14)
- `legend.text.size` - Legend entry label font size (UI: "Legend Text Size", default: 12)
- `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)

Each "Adjustment Function" is applied first and the matching "Adjustment" then rescales the result (e.g. `log10`, then `z-score`), the reverse of the order `dittoViz::scatterPlot()` applies its `*.adj.fxn` and `*.adjustment` in.

Fit lines (linear, best fit, and custom models) are fit to the values as plotted, after any X/Y adjustment, so they are drawn over the points they describe. A custom formula such as `mpg ~ hp` therefore models the adjusted values; do not repeat the adjustment inside it. No fit lines are drawn while an adjustment (`as.factor`) makes an axis categorical.

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

<code>id</code>	The ID for the Shiny module.
<code>resizable</code>	Logical; when TRUE (the default) the plot output is wrapped in <code>shinyjquery::jquery_resizable()</code> 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,  
  defaults = NULL  
)
```

Arguments

<code>id</code>	The ID for the Shiny module.
<code>data</code>	A reactive containing the data frame to plot. Values that are not data frames are coerced with <code>as.data.frame()</code> ; a NULL value is treated as "not ready yet" and the module waits for data.
<code>hide.inputs</code>	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.
<code>hide.tabs</code>	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.
<code>defaults</code>	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. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> , in which case the input tracks it as the parent app's state changes; see <code>setup_reactive_defaults()</code> .

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

<code>data_list</code>	An optional named list of data frames. If NULL (the default), the module's example dataset is used, along with its showcase defaults.
<code>defaults</code>	A named list of input IDs and their default values to apply on startup. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> to have the input follow the parent app's state; see <code>setup_reactive_defaults()</code> .
<code>hide.inputs</code>	A character vector of input IDs to hide. Their values are still initialized and used, but the controls are not shown in the UI.
<code>hide.tabs</code>	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 on `example_demographics` with the settings the module gallery (`moduleGalleryApp()`) opens this module on, so its main features are on show from the start; any defaults you pass are applied over those. 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

<code>id</code>	The ID for the Shiny module.
<code>data</code>	The data frame used for plot generation.
<code>defaults</code>	A named list of default values for the inputs. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> ; it is resolved with <code>shiny::isolate()</code> to seed the control, and the module then keeps it live (see <code>setup_reactive_defaults()</code>).
<code>title</code>	An optional title for the UI grid.
<code>columns</code>	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.

The "Y Data" input accepts several columns at once. "Multivar Aesthetic" (on the Facet tab) then decides how they are shown: "split" gives each variable its own facet (alongside any `split.by` facets), while "group" and "color" put the variables on the x-axis or the fill legend respectively. With several variables the Stats tab is only available for the "split" aesthetic with no `split.by` set, and comparisons are then run separately within each variable's facet. The other layouts either replace the x-axis groups being compared or facet on two dimensions at once, neither of which the significance brackets can be placed against.

The "Y Adjustment Function" is applied first and the "Y Adjustment" then rescales the result (e.g. `log10`, then z-score), the reverse of the order `dittoViz::yPlot()` applies its `var.adj.fxn` and `var.adjustment` in.

Statistics are computed on the values as plotted, after any Y adjustment, and the Y Axis Min/Max and significance brackets are in those units too. With a ridge plot among the plot types the values run along the x-axis, so no brackets are drawn; the test results are still included in the source data download.

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. "z-score(log2(units))"; plotly allows interactive editing). With several Y variables only the adjustment is shown, as the variables are named by the facet strips or the legend instead
- `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
- `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`)
- `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(s) (UI: "Y Data", default: 2nd numeric variable). Several may be selected; `multivar.aes` then controls how they are displayed
- `multivar.aes` - Aesthetic used for multiple var columns (UI: "Multivar Aesthetic", default: "split")

- `multivar.split.dir` - Facet direction for multiple var columns (UI: "Multivar Split Dir", default: "col")
- `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, in the units plotted (after any `var.adjustment/var.adj.fxn`); set with the `y.min` key (UI: "Y Axis Min", auto-calculated). Not applied under a free y facet scale, where each panel spans its own data.
- `max` - Y-axis maximum, in the units plotted; set with the `y.max` key (UI: "Y Axis Max", auto-calculated). Not applied under a free y facet scale.
- `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"). A free y scale ("free", "free_y") ignores Y Axis Min/Max and stacks each panel's significance brackets above that panel's own data.
- `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.linewidth` - Boxplot line weight (UI: "Boxplot Line Weight", default: 0.5)
- `vlplot.linewidth` - Violin line weight (UI: "Violin Line Weight", default: 0.5)
- `vlplot.scaling` - Violin scaling method (UI: "Violin Scaling", default: "area")
- `vlplot.width` - Violin width (derived from boxgap; not directly settable)
- `ridgeplot.linewidth` - 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)
- `hover.data` - Columns shown on hover (UI: "Hover Data", default: ""; empty uses a sensible default set of columns)
- `hover.round.digits` - Hover value rounding (UI: "Hover Round Digits", default: 5)
- `legend.show` - Show the legend (UI: "Show Legend", default: TRUE)
- `stats.enabled` and the other `stat.*` parameters - Pairwise testing between the `group.by` groups, or between the `color.by` levels within each group when `color.by` is set (Stats tab). `stat.pairs` takes a character vector of "A vs B" strings naming those levels, e.g. "Office vs Remote"; left unset, every pair is tested

Parameters controlling additional functionality

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

- `boxmode` - Always "overlay": the boxes carry explicit x positions matching ggplot's dodge, so plotly.js is not asked to dodge them
- `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)
- `legend.font.family` - Font family of the legend title and labels (UI: "Legend Font", default: "Arial")
- `legend.font.color` - Font color of the legend title and labels (UI: "Legend Font Color", default: "#000000")
- `legend.title.size` - Legend title font size (UI: "Legend Title Size", default: 14)
- `legend.text.size` - Legend entry label font size (UI: "Legend Text Size", default: 12)
- `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")
- `palette.colours` - Named character vector mapping group levels to colors, e.g. `c(A = "#FF0000", B = "blue")` (UI: "Plot colors"). Seeds the picker; unnamed groups fall back to the default palette and user edits take precedence.
- `annotate.by` - Column whose values identify and label jitter points (UI: "Annotate By", default: "")
- `highlight.points` - Values from the `annotate.by` column to highlight (UI: "Points to Highlight", default: ""). Values containing spaces (e.g. "CD4 T") must be separated by commas or new lines
- `highlight.color` - Fill color for highlighted points (UI: "Highlight Fill", default: "#00FFF7")
- `highlight.size` - Size of highlighted points (UI: "Highlight Size", default: 7)
- `highlight.border.color` - Border color for highlighted points (UI: "Highlight Border Color", default: "#000000")

- `highlight.border.width` - Border width for highlighted points (UI: "Highlight Border Width", default: 1)
- `highlight.auto.annotate` - Label highlighted points automatically (UI: "Auto-annotate Highlights", default: TRUE)
- `annotation.color` - Annotation text color (UI: "Annotation Color", default: "black")
- `annotation.ax` - Horizontal label offset in pixels (UI: "Annotation X Offset", default: 20)
- `annotation.ay` - Vertical label offset in pixels (UI: "Annotation Y Offset", default: -20)
- `annotation.size` - Annotation font size (UI: "Annotation Size", default: 10)
- `annotation.showarrow` - Draw an arrow from label to point (UI: "Show Arrow", default: TRUE)
- `annotation.arrowcolor` - Annotation arrow color (UI: "Arrow Color", default: "black")
- `annotation.arrowhead` - Annotation arrowhead style (UI: "Arrowhead Style", default: 2)
- `annotation.arrowwidth` - Annotation arrow line width (UI: "Arrow Linewidth", default: 1.5)

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

<code>id</code>	The ID for the Shiny module.
<code>resizable</code>	Logical; when TRUE (the default) the plot output is wrapped in <code>shinyjquery::jquery_resizable()</code> 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

<code>id</code>	The ID for the Shiny module.
<code>data</code>	A reactive containing the data frame to plot. Values that are not data frames are coerced with <code>as.data.frame()</code> ; a NULL value is treated as "not ready yet" and the module waits for data.
<code>hide.inputs</code>	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.
<code>hide.tabs</code>	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.
<code>defaults</code>	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. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> , in which case the input tracks it as the parent app's state changes; see <code>setup_reactive_defaults()</code> .

Value

The moduleServer function for the yPlot module.

Author(s)

Jared Andrews, Jacob Martin

See Also

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

draw_to_png	<i>Render a grid or base drawing to PNG bytes</i>
-------------	---

Description

The raster counterpart to `draw_to_svg()`, for the same job: letting a module whose output is not a plotly graph supply its own artwork to an export. Where `draw_to_svg()` yields markup to splice, this yields the bytes of a finished file.

Usage

```
draw_to_png(draw_fn, width, height, res = 72, scale = 2, bg = "white")
```

Arguments

draw_fn	A function of no arguments that draws onto the active device.
width, height	Size in pixels.
res	Pixels per inch used to size the drawing. The default matches <code>shiny::renderPlot()</code> 's own res, so the result is laid out on a canvas the same physical size as the one the plot was rendered at on screen – which matters for anything sized in absolute points; see the note on res in <code>draw_to_svg()</code> .
scale	Pixel density multiplier. The drawing is laid out at the same physical size (res scales with the pixel count) but rasterised at scale times the resolution, so the result is a crisp image rather than a screenshot. Matches the scale plotly's own image export defaults to.
bg	Background color.

Value

A raw vector holding a PNG file, or NULL if the requested size is not usable or the R build has no PNG support. An error raised by draw_fn itself propagates to the caller; the device is closed either way.

Author(s)

Jared Andrews

See Also

[draw_to_svg\(\)](#) for the vector counterpart.

Examples

```
png_bytes <- draw_to_png(function() plot(1:10), width = 480, height = 360)
head(png_bytes, 4)
```

draw_to_svg

Render a grid or base drawing to a self-contained SVG fragment

Description

Draws onto an SVG device and returns the markup, ready to be placed inside a larger SVG document. This is what lets a module whose output is not a plotly graph still contribute vector art to the Figure Builder's figure export: see the `vector_svg` attribute documented in [figureBuilderServer\(\)](#).

Usage

```
draw_to_svg(draw_fn, width, height, res = 72, id_prefix = NULL, bg = "white")
```

Arguments

<code>draw_fn</code>	A function of no arguments that draws onto the active device.
<code>width, height</code>	Size in pixels.
<code>res</code>	Pixels per inch used to convert width/height to the inches the SVG devices take. The default matches <code>shiny::renderPlot()</code> 's own <code>res</code> , so the fragment is drawn on a canvas the same physical size as the one the panel was rendered at on screen. That is not cosmetic: a <code>ComplexHeatmap</code> legend, its row labels and its titles are all sized in absolute points, so a canvas even slightly smaller than the on-screen one leaves them the same size while the heatmap body – the one flexible element – absorbs the entire shortfall. Exporting a legend-heavy heatmap 25% small squeezed its cells down to nothing.
<code>id_prefix</code>	Optional character scalar used to namespace the fragment's ids, so several fragments can share one document. See <code>.svg_namespace_ids()</code> .
<code>bg</code>	Background color.

Details

`svglite` is used when it is installed, because it writes real `<text>` elements, so labels stay editable in a vector editor. The cairo device (`grDevices::svg()`) is the fallback: still vector, but it converts text to glyph paths, which cannot be edited or restyled afterwards.

Value

A character scalar holding an <svg> element, or NULL if the requested size is not usable or the R build can reach neither SVG device (no **svglite** installed and no working cairo – which on macOS needs XQuartz). An error raised by `draw_fn` itself (a panel dragged too small to leave any plotting room, say) propagates to the caller, which is better placed to decide whether to drop that panel or fail the whole export; the device is closed either way.

The result is a *fragment*: it carries no XML declaration and no `xmlns`, because it is meant to be spliced into a document that already declares them. That makes it unopenable as a file on its own – writing one out directly needs the namespace restated first.

Author(s)

Jared Andrews

See Also

[draw_to_png\(\)](#) for the raster counterpart.

Examples

```
svg <- draw_to_svg(function() plot(1:10), width = 480, height = 360)
substr(svg, 1, 24)
```

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",
  point.size = 12,
  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,
axis.title.font.size = 18,
axis.title.font.color = "black",
axis.title.font.family = "Arial",
show.grid.x = TRUE,
show.grid.y = TRUE,
grid.color = "#CCCCCC",
facet.title.font.size = 18,
facet.title.font.color = "black",
facet.title.font.family = "Arial",
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

<code>data</code>	A data.frame or tibble containing the data to plot.
<code>x</code>	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.
<code>y</code>	Character, column name for the y-axis (categorical variable recommended).
<code>colour.by</code>	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".
<code>palette.selection</code>	Character vector of hex colors for marker colors. A named vector is matched by name to the x variables (<code>colour.by = "X variables"</code>) or the y categories ("Y variables"); an unnamed one is assigned in order, to the x variables as given

	or to the y categories in their order of appearance in data. Either way a category keeps its colour in every facet.
<code>show.legend</code>	Logical, whether to display the legend. Default: TRUE.
<code>facet.by</code>	Optional character, column name to facet plots by. Creates subplots for each unique value. Default: NULL.
<code>line.colour</code>	Character, hex color for the connecting lines between dumbbell points. Default: "gray80".
<code>point.size</code>	Numeric, diameter of the markers in pixels. Default: 12.
<code>facet.scales</code>	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".
<code>subplot.margin</code>	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.
<code>axis.showline</code>	Logical, whether to show axis border lines. Default: TRUE.
<code>axis.mirror</code>	Logical, whether to mirror axis lines on opposite side of plot. Default: TRUE.
<code>axis.linecolor</code>	Character, hex color for axis lines. Default: "black".
<code>axis.linewidth</code>	Numeric, width of axis lines in pixels. Default: 0.5.
<code>axis.tickfont.size</code>	Numeric, font size for axis tick labels. Default: 12.
<code>axis.tickfont.color</code>	Character, hex color for axis tick labels. Default: "black".
<code>axis.tickfont.family</code>	Character, font family for axis tick labels. Default: "Arial".
<code>axis.tickangle.x</code>	Numeric, rotation angle for x-axis tick labels in degrees. Default: 0.
<code>axis.tickangle.y</code>	Numeric, rotation angle for y-axis tick labels in degrees. Default: 0.
<code>axis.ticks</code>	Character, position of tick marks. Options: "outside", "inside", "none". Default: "outside".
<code>axis.tickcolor</code>	Character, hex color for tick marks. Default: "black".
<code>axis.ticklen</code>	Numeric, length of tick marks in pixels. Default: 5.
<code>axis.tickwidth</code>	Numeric, width of tick marks in pixels. Default: 1.
<code>axis.title.font.size</code>	Numeric, font size for the x/y axis titles. Default: 18.
<code>axis.title.font.color</code>	Character, hex color for the x/y axis titles. Default: "black".
<code>axis.title.font.family</code>	Character, font family for the x/y axis titles. Default: "Arial".
<code>show.grid.x</code>	Logical, whether to show gridlines on the x-axis. Default: TRUE.
<code>show.grid.y</code>	Logical, whether to show gridlines on the y-axis. Default: TRUE.

<code>grid.color</code>	Character, hex color for gridlines. Default: "#CCCCCC".
<code>facet.title.font.size</code>	Numeric, font size for the facet panel titles. Default: 18.
<code>facet.title.font.color</code>	Character, hex color for the facet panel titles. Default: "black".
<code>facet.title.font.family</code>	Character, font family for the facet panel titles. Default: "Arial".
<code>title.text</code>	Character, main title text for the plot. Default: "".
<code>title.font.size</code>	Numeric, font size for plot title. Default: 26.
<code>title.font.family</code>	Character, font family for plot title. Default: "Arial".
<code>title.font.color</code>	Character, hex color for plot title text. Default: "black".
<code>title.x.position</code>	Numeric, horizontal position of the plot title in paper coordinates (0 = left, 1 = right). Default: 0.47.
<code>y.title</code>	Optional character, label for y-axis. If NULL, auto-generated from column name. Default: NULL.
<code>x.title</code>	Optional character, label for x-axis. If NULL, auto-generated from column name. Default: NULL.
<code>flip.x</code>	Logical, whether to reverse the x-axis direction. Default: FALSE.
<code>flip.y</code>	Logical, whether to reverse the y-axis direction. Default: FALSE.
<code>x.adjustment</code>	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.
<code>order.by</code>	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), the module's example dataset is used, along with its showcase defaults. |
| defaults | A named list of input IDs and their default values to apply on startup. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> to have the input follow the parent app's state; see <code>setup_reactive_defaults()</code> . |
| 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 on `example_school_earnings` with the settings the module gallery (`moduleGalleryApp()`) opens this module on, so its main features are on show from the start; any defaults you pass are applied over those. 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

<code>id</code>	The ID for the Shiny module.
<code>data</code>	The data frame used for plot generation.
<code>defaults</code>	A named list of default values for the inputs. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> ; it is resolved with <code>shiny::isolate()</code> to seed the control, and the module then keeps it live (see <code>setup_reactive_defaults()</code>).
<code>title</code>	An optional title for the UI grid.
<code>columns</code>	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")
- `point.size` - Marker diameter in pixels (UI: "Point Size", default: 12)
- `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")
- `legend.show` - Show the legend and any colorbar (UI: "Show Legend", default: TRUE)
- `legend.font.family` - Font family of the legend title and labels (UI: "Legend Font", default: "Arial")
- `legend.font.color` - Font color of the legend title and labels (UI: "Legend Font Color", default: "#000000")

- `legend.title.size` - Legend title font size (UI: "Legend Title Size", default: 14)
- `legend.text.size` - Legend entry label font size (UI: "Legend Text Size", default: 12)
- `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)
- `palette.colours` - Named character vector mapping group levels to colors, e.g. `c(A = "#FF0000", B = "blue")` (UI: "Plot colors"). Seeds the picker; unnamed groups fall back to the default palette and user edits take precedence.

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

<code>id</code>	The ID for the Shiny module.
<code>resizable</code>	Logical; when TRUE (the default) the plot output is wrapped in <code>shinyjquery::jquery_resizable()</code> 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	<i>Server logic for dumbbellPlot module</i>
--------------------	---

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. Values that are not data frames are coerced with <code>as.data.frame()</code> ; a NULL value is treated as "not ready yet" and the module waits for data.
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. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> , in which case the input tracks it as the parent app's state changes; see <code>setup_reactive_defaults()</code> .

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 one row for each of six groups crossed with three types, so bars stack by Type and each Type facet of a split bar plot has one bar per Group. Values is positive and shrinks from Alpha to Gamma; Numbers and Score are signed, leaning positive for Alpha and negative for Gamma. Used as the default data for [plotthis_BarPlotApp\(\)](#) and [plotthis_SplitBarPlotApp\(\)](#).

Usage

```
example_bar
```

Format

A data frame with 18 rows and 5 columns:

Group Group label (A through F)

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_composition	<i>Example single-cell-style composition data for the freqPlot module</i>
---------------------	---

Description

A per-cell record table from a simulated 12-donor immune profiling experiment, shaped for `dittoViz::freqPlot()`. Each donor (sample) contributes 150 cells and maps to exactly one condition and one batch, which is the nesting `freqPlot()` requires to compare per-sample cell-type frequencies across groups. batch is crossed with condition (three donors each), so it works as a `color.by` without confounding the comparison.

Usage

```
example_composition
```

Format

A data frame with 1800 rows and 7 columns:

cell_id Unique cell identifier (character)

sample Donor identifier, P01-P12 (factor); 150 cells each

condition Disease state, Healthy or Disease (factor); six donors each

batch Processing batch, B1 or B2 (factor); crossed with condition

cell_type Annotated cell type (factor), the variable whose per-sample frequency `freqPlot()` tabulates

n_genes Number of genes detected in the cell (integer)

percent_mito Percentage of mitochondrial reads (numeric)

Details

Composition differs between the two conditions: the Disease donors show an expanded monocyte compartment and depleted CD4 T cells relative to Healthy.

Author(s)

Jared Andrews

Source

Simulated. Per-donor compositions are Dirichlet draws around condition-specific means, with cell counts drawn multinomially. See `data-row/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. Salary rises steeply with job level and varies by department; age and tenure rise with level; remote workers report higher satisfaction than office workers in every department, and long hours lower it. Gender has no effect on anything. Two employees are planted against the trend for highlighting: E042, a very unhappy remote engineer, and E137, a very happy office-based salesperson. Designed to showcase the box, yPlot, density, and histogram plot modules, including their statistical comparisons.

Usage

`example_demographics`

Format

A data frame with 500 rows and 11 columns:

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

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

gender Employee gender (factor: Female, Male)

age Employee age in years (21-67)

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

work_mode Where the employee works (factor: Office, Remote)

employee_id Unique employee identifier (E001-E500)

Author(s)

Jared Andrews

Source

Simulated in data-raw/generate_example_data.R.

example_heatmap_column_data

Example sample-metadata table for the ComplexHeatmap module

Description

Per-sample metadata for the 12 samples in [example_heatmap_matrix](#), keyed by sample. Supplying this alongside the matrix (as `list(matrix = example_heatmap_matrix, column_annotations = example_heatmap_column_data)`) enables column annotations in the ComplexHeatmap module.

Usage

example_heatmap_column_data

Format

A data frame with 12 rows and 4 columns:

sample Sample identifier (factor), matching the sample column names in [example_heatmap_matrix](#)

condition Experimental condition (factor: Healthy, Disease)

batch Processing batch (factor: B1, B2), crossed with condition

library_size Simulated sequencing library size (numeric)

Author(s)

Jacob Martin

Source

Simulated in data-raw/generate_example_data.R.

```
example_heatmap_matrix
```

Example gene-expression-style matrix for the ComplexHeatmap module

Description

A tidy data frame shaped as observations (genes, rows) by samples (columns) — the layout `ComplexHeatmap::Heatmap()` expects. 30 genes drawn from three functional pathways (10 genes each), profiled across 12 samples (6 "Healthy", 6 "Disease"). Values are simulated, *unscaled* log2-CPM-like expression: Immune and Cell Cycle pathway genes are elevated in Disease samples, Metabolic pathway genes are flat, so the module's row/column scaling, clustering, splitting, and row-annotation controls all have real signal to demonstrate on. Pairs with `example_heatmap_column_data` to additionally demonstrate column annotations (see `ComplexHeatmap_HeatmapApp()`'s `column_data` argument).

Usage

```
example_heatmap_matrix
```

Format

A data frame with 30 rows and 15 columns:

gene Gene symbol (character), used as row identifier

pathway Functional pathway the gene belongs to (factor: Immune, Metabolic, Cell Cycle) — a categorical row-annotation column

mean_expression Mean log2-CPM-like expression across the 12 samples — a numeric row-annotation column

Healthy_1, Healthy_2, Healthy_3, Healthy_4, Healthy_5, Healthy_6, Disease_1, Disease_2, Disease_3, Disease_4, Disease_5, Disease_6
Simulated log2-CPM-like expression values forming the heatmap matrix

Author(s)

Jacob Martin

Source

Simulated in `data-raw/generate_example_data.R`.

example_iris	<i>Example grouped iris dataset</i>
--------------	-------------------------------------

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	<i>Example single-cell marker gene dataset for dot plots</i>
-----------------	--

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	<i>Example mtcars dataset with factors</i>
----------------	--

Description

The classic mtcars dataset with the cyl, vs, and gear 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)
- gear** Number of forward gears (factor)
- carb** Number of carburetors

Author(s)

Jared Andrews

Source

Generated from the classic mtcars dataset.

example_population	<i>Example population dataset</i>
--------------------	-----------------------------------

Description

A simulated population dataset with 400 rows covering 50 years and 8 age groups. The youngest groups shrink over the period while those over 45 grow, so the population visibly ages in a stacked area plot (particularly one scaled to each year’s total). Used as the default data for [plotthis_AreaPlotApp\(\)](#).

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 pseudo-bulk RNA-seq dataset***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 neg_log10_pval 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	<i>Example sales dataset</i>
---------------	------------------------------

Description

A simulated product-sales dataset: one sale for each month of 2015-2024 in each of six regions (720 rows), with each year’s sales split evenly across three product lines. Units sold follow a trend per product line (Gadgets growing, Widgets flat, Doohickeys declining), peak in November and December, and scale by region. Revenue is units times a per-product unit price, so revenue against units falls on one line per product line. Two sales are planted outliers: Sale_352, a promotion that shifted far more units than usual, and Sale_540, a clearance sale at well under half price. Designed to showcase the scatter, line, parallel coordinates and pie plot modules.

Usage

example_sales

Format

A data frame with 720 rows and 8 columns:

- region** Region of the sale (factor: North, South, East, West, Central, International)
- revenue** Revenue of the sale (thousands of USD)
- year** The year (factor: 2015-2024)
- month** The month (factor: Jan-Dec)
- units** Units sold (integer)
- sale_id** Unique sale identifier
- product_line** Product line (factor: Gadgets, Widgets, Doohickeys)
- profit** Profit on the sale after a fixed overhead (thousands of USD; negative for a few low-volume Doohickey sales)

Author(s)

Jared Andrews

Source

Generated in data-raw/generate_example_data.R.

example_school_earnings	<i>Example school earnings dataset for dumbbell plots</i>
-------------------------	---

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	<i>Example multi-player skills dataset for radar plots</i>
----------------	--

Description

A dataset of skill ratings across six categories for three players with distinct profiles (a runner, a defender, and a playmaker), suitable for radar/spider chart examples.

Usage

example_skills

Format

A data frame with 18 rows and 3 columns:

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

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_region summary suited to the pie plot) are used. At least one element is required. An element is either a data frame, or a named list of data frames for a module that needs companion tables (the ComplexHeatmap module's list(matrix = , column_annotatons =)); in the latter case only the primary table is filtered and shown in the panel's table pane.

<code>module_registry</code>	An optional named list describing the plot modules to offer. If <code>NULL</code> (the default), all bundled <code>VizModules</code> modules are offered, each opening on the same example figure as in <code>moduleGalleryApp()</code> . Each entry is itself a list with components: <code>label</code> (character, shown in the picker), <code>dataset</code> (character, the dataset name its defaults were written for), <code>inputs_ui</code> , <code>output_ui</code> , and <code>server_fn</code> (the module's three functions), and <code>defaults</code> (a named list of input defaults applied only when <code>dataset</code> is the chosen dataset). An entry may also carry <code>primary.table</code> , naming which table of a multi-table dataset gets filtered (the first by default); its presence is also what marks the module as able to take a multi-table dataset at all, so modules without it are handed the primary table alone and any dataset stays usable with any module. See <code>figureBuilderServer()</code> for the <code>vector_svg</code> hook that lets a non-plotly module take part in the SVG figure export.
<code>title</code>	A character string used as the page title and header (default: "VizModules Figure Builder").
<code>return_components</code>	Logical. When <code>FALSE</code> (the default) a <code>shiny::shinyApp()</code> object is returned. When <code>TRUE</code> a named list with <code>ui</code> and <code>server</code> elements is returned instead, which is convenient for deployment scripts that need an explicit <code>shinyApp(ui, server)</code> call.

Details

Datasets are supplied via `data_list` and seed the "Add Plot" dialog; users can also upload additional datasets (CSV, TSV, or tab-delimited TXT) 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. It is also the **Figure Builder** tab of `moduleGalleryApp()`.

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()`, `moduleGalleryApp()`

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

<code>id</code>	The ID for the Shiny module. Must match the <code>id</code> given to <code>figureBuilderUI()</code> .
<code>data_list</code>	An optional named list of data frames that seed the dataset registry. If <code>NULL</code> (the default), the bundled example datasets (plus a <code>sales_by_region</code> summary suited to the pie plot) are used. At least one element is required. An element is either a data frame, or a named list of data frames for a module that needs companion tables (the <code>ComplexHeatmap</code> module's <code>list(matrix = , column_annotatons =)</code>); in the latter case only the primary table is filtered and shown in the panel's table pane.
<code>module_registry</code>	An optional named list describing the plot modules to offer. If <code>NULL</code> (the default), all bundled VizModules modules are offered, each opening on the same example figure as in <code>moduleGalleryApp()</code> . Each entry is itself a list with components: <code>label</code> (character, shown in the picker), <code>dataset</code> (character, the dataset name its defaults were written for), <code>inputs_ui</code> , <code>output_ui</code> , and <code>server_fn</code> (the module's three functions), and <code>defaults</code> (a named list of input defaults applied only when <code>dataset</code> is the chosen dataset). <code>defaults</code> is passed to both <code>inputs_ui</code> and <code>server_fn</code> , so it can seed server-rendered controls such as the group color picker. An entry may also carry <code>primary.table</code> , naming which table of a multi-table dataset gets filtered (the first by default); its presence is also

what marks the module as able to take a multi-table dataset at all, so modules without it are handed the primary table alone and any dataset stays usable with any module.

A module whose output is not a plotly graph can still contribute to the SVG figure export by attaching a `vector_svg` attribute to the reactive its server returns: a function(`width`, `height`, `res`) yielding an `<svg>` element drawn at that pixel size, which is spliced into the figure in place of the `Plotly.toImage()` result. `draw_to_svg()` builds one from any grid or base drawing; `ComplexHeatmap_HeatmapServer()` is the worked example. A panel whose module attaches nothing simply contributes no artwork.

The same renderers feed the source archive, which additionally wants a `raster_png` counterpart returning PNG bytes (see `draw_to_png()`). Either can be given as an attribute on the reactive, as here, or as a field on the summary list the reactive returns – the latter being order-independent, since the summary is rebuilt on every download. See `create_source_download_handler()`.

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()`, `moduleGalleryApp()`

Examples

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

figureBuilderUI	<i>UI component for the Figure Builder module</i>
-----------------	---

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 module gallery, [moduleGalleryApp\(\)](#), embeds it this way as its **Figure Builder** tab.

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\(\)](#), [moduleGalleryApp\(\)](#)

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,
  regen_keys = character(0)
)
```

Arguments

<code>fig</code>	A plotly figure object, typically the result of your plotting pipeline. If NULL, it is returned unchanged.
<code>plot_source</code>	Character scalar. The same plotly event source id passed to <code>setup_manual_edits()</code> ; assigned to <code>fig\$x\$source</code> .
<code>store</code>	The list returned by <code>setup_manual_edits()</code> .
<code>session</code>	The module's session object, used to namespace the colorbar drag input.
<code>regen_keys</code>	Character vector of annotation keys (e.g. "axis:y") whose text is regenerated on every rebuild and must not be overwritten by a captured edit. Pass the axis side(s) carrying an active data adjustment so the freshly built label (e.g. "log2(units)") always wins, while the captured position still persists. Defaults to none (all captured props are restored, the pre-existing behaviour).

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 relay layout 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

<code>df</code>	Data frame containing the data.
<code>x</code>	Character; x-axis column name.
<code>group.by</code>	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

<code>defaults</code>	A named list of default values, or <code>NULL</code> . Individual entries may be a <code>reactive()</code> / <code>reactiveVal</code> .
<code>key</code>	Character string — the name to look up.
<code>fallback</code>	The value to return when <code>key</code> is absent or fails validation.
<code>validator</code>	An optional single-argument predicate function (e.g., <code>is.numeric</code> , <code>is.logical</code>). When supplied, the stored value is returned only if <code>validator(value)</code> is <code>TRUE</code> . Reactive entries are validated on their resolved value.

Details

Entries that are a `shiny::reactive()` or `shiny::reactiveVal()` are resolved with `shiny::isolate()` before validation, so this returns the reactive's *current* value. See `setup_reactive_defaults()` for how modules keep such entries live at render time.

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_default(list(color = shiny::reactiveVal("blue")), "color", "black")
```

get_documentation	<i>Extract parameter documentation from an R function help page</i>
-------------------	---

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 <code>extract_roc_text</code> .
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	<i>Get a registered model backend</i>
-------------------	---------------------------------------

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")
```

heatmap_fit_width	<i>Fit a hand-built InteractiveComplexHeatmap widget to its container's width</i>
-------------------	---

Description

The heatmap module's output functions already do this (see their fit.width argument). This is the same behaviour for an app that calls `InteractiveComplexHeatmap::InteractiveComplexHeatmapOutput()` itself rather than going through the module — a heatmap driven directly by `InteractiveComplexHeatmap::makeInteractiveComplexHeatmapOutput()` say.

Usage

```
heatmap_fit_width(
  ui,
  heatmap_id,
  panels = c("heatmap", "sub_heatmap"),
  output = TRUE
)
```

Arguments

ui	The UI returned by <code>InteractiveComplexHeatmap::InteractiveComplexHeatmapOutput()</code> (or the <code>originalHeatmapOutput()/subHeatmapOutput()</code> pair).
heatmap_id	The heatmap_id passed to that output function.
panels	Character vector of panels to scale. Defaults to both; pass just "heatmap" for a compact = TRUE widget, which has no sub-heatmap.
output	Logical; also scale the click/brush info panel. Defaults to TRUE, matching what <code>ComplexHeatmap_HeatmapOutputUI()</code> does for the same combined widget; pass FALSE when the app lays that panel out itself.

Details

`InteractiveComplexHeatmapOutput()` bakes its `width1/width2` into the page as fixed pixels, so the widget over- or under-fills whatever room the app actually gives it until the resize handle is dragged. Wrapping the output in this rescales the panels to their container once the page is laid out; the widget's own resize controls still win afterwards.

A widget on a tab that is not the active one has no width to measure on load. It is fitted when its container is first laid out instead, so a heatmap the user has not opened yet is still correct the moment they do.

Value

ui, with the fitting script and its dependency attached.

Author(s)

Jared Andrews

See Also

[ComplexHeatmap_HeatmapOutputUI\(\)](#), whose `fit.width` argument is the module-side equivalent.

Examples

```
if (interactive() && requireNamespace("InteractiveComplexHeatmap", quietly = TRUE)) {
  heatmap_fit_width(
    InteractiveComplexHeatmap::InteractiveComplexHeatmapOutput(
      heatmap_id = "my_ht", width1 = 1480, height1 = 500
    ),
    heatmap_id = "my_ht"
  )
}
```

hide_input

Hide the grid cells wrapping module inputs

Description

Toggles the visibility of the `.vizmodules-input-cell` that wraps a given input (as laid out by [organize_inputs\(\)](#)). Hiding the cell rather than just the input itself lets the surrounding inputs reflow so the panel stays compact instead of leaving an empty gap, as a plain `shinyjs::hide()` on the input would.

Usage

```
hide_input(session, ids)
```

Arguments

session	The module session object (provides session\$ns).
ids	Character vector of un-namespaced input IDs to toggle.

Value

Invisibly NULL, called for the side effect of running client-side JS.

Author(s)

Jared Andrews

See Also

[show_input\(\)](#), [toggle_input_cell\(\)](#), [organize_inputs\(\)](#)

Examples

```
## Not run:
# Call inside a module server:
hide_input(session, c("size", "opacity"))

## End(Not run)
```

is_pure_type	<i>Check if column inputs contain mixed data types</i>
--------------	--

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 (anything non-numeric: factor, character, logical, Date, ...). 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 (non-numeric); FALSE if mixed numeric + categorical/factor detected.

Author(s)

Jacob Martin

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,
grid.color = "#CCCCCC",
axis.title.font.size = 18,
axis.title.font.color = "black",
axis.title.font.family = "Arial",
facet.title.font.size = 18,
facet.title.font.color = "black",
facet.title.font.family = "Arial",
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,
error.type = c("sd", "sem", "ci95"),
error.ci.method = c("normal", "t")
)

```

Arguments

<code>data</code>	A data.frame or tibble containing the data to plot.
<code>x</code>	Character vector of column name(s) for the x-axis. Multiple columns create separate traces.
<code>y</code>	Character vector of column name(s) for the y-axis. Multiple columns create separate traces.
<code>palette.selection</code>	Character vector of hex colors for line colors. Used to assign colors to groups or traces.
<code>plot.mode</code>	Character, plotly mode for plot type. Options: "lines", "markers", "lines+markers". Default: "lines".
<code>line.type</code>	Character, line style. Options: "solid", "dot", "dash", "longdash", "dashdot", "longdashdot". Default: "solid".
<code>colour.group.by</code>	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.
grid.color	Character, hex color for gridlines. Default: "#CCCCCC".
axis.title.font.size	Numeric, font size for the x/y axis titles. Default: 18.

<code>axis.title.font.color</code>	Character, hex color for the x/y axis titles. Default: "black".
<code>axis.title.font.family</code>	Character, font family for the x/y axis titles. Default: "Arial".
<code>facet.title.font.size</code>	Numeric, font size for the facet panel titles. Default: 18.
<code>facet.title.font.color</code>	Character, hex color for the facet panel titles. Default: "black".
<code>facet.title.font.family</code>	Character, font family for the facet panel titles. Default: "Arial".
<code>title.text</code>	Character, main title text for the plot. Default: "".
<code>title.font.size</code>	Numeric, font size for plot title. Default: 14.
<code>title.font.family</code>	Character, font family for plot title. Default: "Arial".
<code>title.font.color</code>	Character, hex color for plot title text. Default: "black".
<code>title.x.position</code>	Numeric, horizontal position of the plot title in paper coordinates (0 = left, 1 = right). Default: 0.47.
<code>y.title</code>	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 <code>mean(<y.title>)</code> to accurately describe the summary displayed. Default: NULL.
<code>x.title</code>	Optional character, label for x-axis. If NULL, auto-generated from column name. Default: NULL.
<code>flip.x</code>	Logical, whether to reverse the x-axis direction. Default: FALSE.
<code>flip.y</code>	Logical, whether to reverse the y-axis direction. Default: FALSE.
<code>x.adjustment</code>	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.
<code>y.adjustment</code>	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.
<code>color.adjustment</code>	Optional character or function, transformation to apply to color grouping variable. Same options as <code>x.adjustment</code> and <code>y.adjustment</code> . Default: NULL.
<code>order.by</code>	Optional character vector, column name(s) to order data by before plotting. Default: NULL.
<code>error.colour</code>	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
<code>error.width</code>	numeric input to set the width of the error bars on a plot with a categorical X axis and only 1 Y axis variable

<code>error.bar</code>	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. Each bar spans the plotted group mean plus or minus the amount <code>error.type</code> selects, computed from that group's y-values, where a group is a single x category, split further by <code>colour.group.by</code> and <code>facet.by</code> when those are set. A group with fewer than two observations has no spread and is drawn without a bar.
<code>error.type</code>	What the error bars show, one of "sd" (one standard deviation, the default), "sem" (one standard error of the mean, $sd / \sqrt{n}$) or "ci95" (a 95% confidence interval for the mean). Missing values are ignored when counting n.
<code>error.ci.method</code>	How <code>error.type = "ci95"</code> is computed, one of "normal" (the default; the standard error times the 97.5th percentile of the normal distribution, 1.96) or "t" (the standard error times the 97.5th percentile of the t distribution with $n - 1$ degrees of freedom). The t interval is wider for small groups and converges on the normal one as n grows. Ignored for the other error types.

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
)
```

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

<code>data_list</code>	An optional named list of data frames. If <code>NULL</code> (the default), the module's example dataset is used, along with its showcase defaults.
<code>defaults</code>	A named list of input IDs and their default values to apply on startup. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> to have the input follow the parent app's state; see <code>setup_reactive_defaults()</code> .
<code>hide.inputs</code>	A character vector of input IDs to hide. Their values are still initialized and used, but the controls are not shown in the UI.
<code>hide.tabs</code>	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 on `example_sales` with the settings the module gallery (`moduleGalleryApp()`) opens this module on, so its main features are on show from the start; any defaults you pass are applied over those. 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	<i>Input UI components for the linePlot module</i>
------------------	--

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. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> ; it is resolved with <code>shiny::isolate()</code> to seed the control, and the module then keeps it live (see <code>setup_reactive_defaults()</code>).
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.type` - What the error bars show: "sd", "sem" or "ci95" (UI: "Error Bar Type", defaults key: `error.bar.type`, default: "sd")
- `error.ci.method` - How a "ci95" interval is computed: "normal" or "t" (UI: "Confidence Interval Method", defaults key: `error.bar.ci.method`, default: "normal"; only shown while the error bar type is "ci95")
- `error.colour` - Error bar color (UI: "Error Bar Colour", defaults key: `error.bar.colour`, default: "#000000")
- `error.width` - Error bar cap width (UI: "Error Bar Width", defaults key: `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:

- `legend.show` - Show the legend and any colorbar (UI: "Show Legend", default: TRUE)
- `legend.font.family` - Font family of the legend title and labels (UI: "Legend Font", default: "Arial")
- `legend.font.color` - Font color of the legend title and labels (UI: "Legend Font Color", default: "#000000")
- `legend.title.size` - Legend title font size (UI: "Legend Title Size", default: 14)
- `legend.text.size` - Legend entry label font size (UI: "Legend Text Size", default: 12)
- `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")
- `palette.colours` - Named character vector mapping group levels to colors, e.g. `c(A = "#FF0000", B = "blue")` (UI: "Plot colors"). Seeds the picker; unnamed groups fall back to the default palette and user edits take precedence.

Author(s)

Jacob Martin, Jared Andrews

See Also

[linePlot\(\)](#), [organize_inputs\(\)](#), [linePlotOutputUI\(\)](#), [linePlotServer\(\)](#), [linePlotApp\(\)](#)

Examples

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

linePlotOutputUI	<i>Output UI components for the linePlot module</i>
------------------	---

Description

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

Usage

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

Arguments

<code>id</code>	The ID for the Shiny module.
<code>resizable</code>	Logical; when TRUE (the default) the plot output is wrapped in <code>shinyjquery::jquery_resizable()</code> 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	<i>Server logic for linePlot module</i>
----------------	---

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. Values that are not data frames are coerced with as.data.frame() ; a NULL value is treated as "not ready yet" and the module waits for data.
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. An entry may also be a shiny::reactive() or shiny::reactiveVal() , in which case the input tracks it as the parent app's state changes; see setup_reactive_defaults() .

Value

The moduleServer function for the linePlot module.

Author(s)

Jacob Martin

See Also

[linePlot\(\)](#), [linePlotInputsUI\(\)](#), [linePlotOutputUI\(\)](#), [linePlotApp\(\)](#)

linetype_to_dash	<i>Convert linetype name to plotly dash style</i>
------------------	---

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	<i>List registered model backends</i>
---------------------	---------------------------------------

Description

Returns the names of all currently registered model backends. The built-in backends (lm, glm, loess) 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()
```

```
moduleGalleryApp
```

```
Launch the VizModules module gallery
```

Description

Builds the **VizModules Gallery**: one tab per plot module, each opening on a bundled example dataset with the module's main features already switched on (significance brackets, highlighted and labelled points, fit lines, reference lines, annotation tracks and splits, and so on), plus a **Figure Builder** tab for composing several modules into one multi-panel figure and exporting it as an editable SVG.

Usage

```
moduleGalleryApp(title = "VizModules Gallery", return_components = FALSE)
```

Arguments

title	A character string used as the navbar title and page title.
return_components	Logical. When FALSE (the default) a <code>shiny::shinyApp()</code> 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 <code>shinyApp(ui, server)</code> call.

Details

Every tab carries the module's full set of controls and a filterable data table beneath the plot, so the gallery doubles as a tour of what each module can do. The example settings each tab opens on are the same ones the module's own `*App()` function (e.g. `plotthis_BoxPlotApp()`) and the Figure Builder start from, so any of them can be reproduced by passing the same defaults to the module in your own app.

The **Heatmap** tab needs the Bioconductor packages **ComplexHeatmap**, **InteractiveComplexHeatmap** and **circlize**, and is left out when they are not installed.

The Figure Builder tab embeds `figureBuilderUI()` / `figureBuilderServer()` on the bundled example datasets. To run a standalone Figure Builder, or one on your own datasets or modules, use `figureBuilderApp()`.

Value

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

Author(s)

Jared Andrews

See Also[figureBuilderApp\(\)](#), [createModuleApp\(\)](#)**Examples**

```
library(VizModules)
app <- moduleGalleryApp()
if (interactive()) runApp(app)

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

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, SVG and PNG images of it, and the source data and statistics as CSVs).

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.

Details

The download button carries the markup the image capture needs: the class the script binds to, and the module's namespace prefix, which is how the script works out which plot on the page belongs to this button. See [create_source_download_handler\(\)](#).

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	<i>Compact multi-group color picker input</i>
------------------	---

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

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

Details

A group's color swatch is a native `<input type="color">`, whose dialog reports a new value for every drag or click inside it. Rather than send each one - rebuilding a dependent plot dozens of times for a single color choice - the input reports once the dialog has closed, detected as the first pointer, key, or scroll event the page sees again (the dialog holds both while it is open) or the browser window regaining focus. Typing in a hex field is coalesced until the user pauses instead. One-shot actions - clicking a palette swatch, "Apply", "Reset", selecting another group, or committing a hex code with Enter or by clicking away - report immediately.

The widget reflows to its container: the palette selector shrinks and the buttons wrap below it when narrow, and long group names wrap rather than running under their controls, so it can be dropped into a sidebar or an input grid cell without controls escaping the panel.

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)
}
```


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

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

<code>data</code>	A data.frame or tibble containing the data to plot.
<code>dimensions</code>	Character vector of column names to use as dimensions (axes). Must contain at least two columns. Non-numeric columns are mapped to integers.
<code>color.by</code>	Optional character, column name to color lines by. Numeric columns use a continuous colorscale (<code>color.scale</code>); categorical columns use a discrete palette (<code>palette.selection</code>) and are displayed with category names on the colorbar. Default: NULL.
<code>color.scale</code>	Character, plotly colorscale name for line coloring when <code>color.by</code> is numeric. Options include "Viridis", "Cividis", "Inferno", "Magma", "Plasma", "Blues", "Greens", "Reds", "Oranges", "RdBu", "RdYlBu", "Spectral", "Jet", "Hot", "Cool", "Portland". Default: "Viridis".
<code>palette.selection</code>	Character vector of hex colors used to color lines when <code>color.by</code> is categorical. May be unnamed (colors applied in order to the sorted unique levels) or named by level. If NULL, the plotly <code>color.scale</code> is used as a fallback. Default: NULL.
<code>line.opacity</code>	Numeric, opacity of lines between 0 and 1. Default: 0.5.
<code>line.width</code>	Numeric, width of lines in pixels. Default: 1.
<code>show.colorbar</code>	Logical, whether to show the colorbar. Default: TRUE.
<code>label.font.size</code>	Numeric, font size for dimension labels. Default: 12.
<code>label.font.color</code>	Character, hex color for dimension labels. Default: "black".
<code>label.font.family</code>	Character, font family for dimension labels. Default: "Arial".
<code>tick.font.size</code>	Numeric, font size for axis tick labels. Default: 10.
<code>tick.font.color</code>	Character, hex color for axis tick labels. Default: "black".
<code>tick.font.family</code>	Character, font family for axis tick labels. Default: "Arial".
<code>title.text</code>	Character, main title text for the plot. Default: "".
<code>title.font.size</code>	Numeric, font size for plot title. Default: 16.
<code>title.font.family</code>	Character, font family for plot title. Default: "Arial".
<code>title.font.color</code>	Character, hex color for plot title text. Default: "black".
<code>bgcolor</code>	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

<code>data_list</code>	An optional named list of data frames. If <code>NULL</code> (the default), the module's example dataset is used, along with its showcase defaults. Each data frame should contain at least two numeric or categorical columns.
<code>defaults</code>	A named list of input IDs and their default values to apply on startup. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> to have the input follow the parent app's state; see <code>setup_reactive_defaults()</code> .
<code>hide.inputs</code>	A character vector of input IDs to hide. Their values are still initialized and used, but the controls are not shown in the UI.
<code>hide.tabs</code>	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 on `example_sales` with the settings the module gallery (`moduleGalleryApp()`) opens this module on, so its main features are on show from the start; any defaults you pass are applied over those. 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

<code>id</code>	The ID for the Shiny module.
<code>data</code>	The data frame used for plot generation.
<code>defaults</code>	A named list of default values for the inputs. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> ; it is resolved with <code>shiny::isolate()</code> to seed the control, and the module then keeps it live (see <code>setup_reactive_defaults()</code>).
<code>title</code>	An optional title for the UI grid.
<code>columns</code>	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")
- `palette.colours` - Named character vector mapping group levels to colors, e.g. `c(A = "#FF0000", B = "blue")` (UI: "Plot colors"). Seeds the picker; unnamed groups fall back to the default palette and user edits take precedence.

Author(s)

Jacob Martin, Jared Andrews

See Also

[parallelCoordinatesPlot\(\)](#), [organize_inputs\(\)](#), [parallelCoordinatesPlotOutputUI\(\)](#), [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

<code>id</code>	The ID for the Shiny module.
<code>resizable</code>	Logical; when TRUE (the default) the plot output is wrapped in <code>shinyjquery::jquery_resizable()</code> 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

<code>id</code>	The ID for the Shiny module.
<code>data</code>	A reactive containing the data frame to plot. Values that are not data frames are coerced with <code>as.data.frame()</code> ; a NULL value is treated as "not ready yet" and the module waits for data.
<code>hide.inputs</code>	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.
<code>hide.tabs</code>	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. An entry may also be a `shiny::reactive()` or `shiny::reactiveVal()`, in which case the input tracks it as the parent app's state changes; see `setup_reactive_defaults()`.

Value

The `moduleServer` function for the `parallelCoordinatesPlot` module.

Author(s)

Jacob Martin, Jared Andrews

See Also

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

<code>parse_numeric_list</code>	<i>Parse comma-separated numeric string to vector</i>
---------------------------------	---

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	<i>Parse pair strings from UI into list of length-2 vectors</i>
--------------------	---

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	<i>Create a plotly pie chart</i>
---------	----------------------------------

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.

<code>textinfo</code>	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".
<code>textposition</code>	Character, position of the text relative to the slice. Options: "auto", "inside", "outside", or "none". Default: "auto".
<code>insidetextorientation</code>	Character, orientation for inside text. Options: "auto", "horizontal", "radial", or "tangential". Default: "auto".
<code>sort</code>	Logical, whether to sort slices by their values in descending order. Default: TRUE.
<code>direction</code>	Character, direction of slice progression. Options: "counterclockwise" or "clockwise". Default: "counterclockwise".
<code>rotation</code>	Numeric, starting angle of the first slice in degrees (0-360). Default: 0.
<code>show.legend</code>	Logical, whether to display the legend. Default: TRUE.
<code>legend.orientation</code>	Character, legend orientation. Options: "h" (horizontal) or "v" (vertical). Default: "h".
<code>legend.x</code>	Numeric, horizontal legend position offset (0-1, where 0=left, 1=right). Default: 0.5.
<code>legend.y</code>	Numeric, vertical legend position offset (-1 to 1). Default: -0.1.
<code>legend.font.family</code>	Character, font family for the legend text. Default: "Arial".
<code>legend.font.size</code>	Numeric, font size for the legend text. Default: 12.
<code>legend.font.color</code>	Character, hex color for the legend text. Default: "#000000".
<code>title.text</code>	Character, main plot title text. Default: "".
<code>title.font.family</code>	Character, font family for the title text. Default: "Arial".
<code>title.font.size</code>	Numeric, font size for the title text. Default: 18.
<code>title.font.color</code>	Character, hex color for the title text. Default: "#000000".
<code>title.x</code>	Numeric, horizontal position for the plot title (0-1, where 0=left, 0.5=center, 1=right). Default: 0.5.
<code>text.font.family</code>	Character, font family for the slice labels. Default: "Arial".
<code>text.font.size</code>	Numeric, font size for the slice labels. Default: 12.
<code>text.font.color</code>	Character, hex color for the slice labels. Default: "#000000".
<code>slice.line.color</code>	Character, hex color for slice borders. Default: "#FFFFFF" (white).
<code>slice.line.width</code>	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 data frames. If NULL (the default), the module's example dataset is used, along with its showcase defaults. Each data frame should already contain a label column and an aggregated numeric value column, one row per slice.
-----------	--

<code>defaults</code>	A named list of input IDs and their default values to apply on startup. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> to have the input follow the parent app's state; see <code>setup_reactive_defaults()</code> .
<code>hide.inputs</code>	A character vector of input IDs to hide. Their values are still initialized and used, but the controls are not shown in the UI.
<code>hide.tabs</code>	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 on `sales_by_region` (example_sales revenue summed by region) with the settings the module gallery (`moduleGalleryApp()`) opens this module on, so its main features are on show from the start; any `defaults` you pass are applied over those. 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	<i>Input UI components for the piePlot module</i>
-----------------	---

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

<code>id</code>	The ID for the Shiny module.
<code>data</code>	The data frame used for plot generation. Supply a summary table with one row per slice.
<code>defaults</code>	A named list of default values for the inputs. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> ; it is resolved with <code>shiny::isolate()</code> to seed the control, and the module then keeps it live (see <code>setup_reactive_defaults()</code>).
<code>title</code>	An optional title for the UI grid.
<code>columns</code>	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")
- `slice.colors` - Named character vector mapping label levels to colors, e.g. `c(A = "#FF0000", B = "blue")` (UI: "Slice colors"). Seeds the picker; unnamed slices fall back to the default palette and user edits take precedence.

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	<i>Output UI components for the piePlot module</i>
-----------------	--

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 <code>shinyjquery::jquery_resizable()</code> 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	<i>Server logic for piePlot module</i>
---------------	--

Description

Server logic for piePlot module

Usage

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

Arguments

<code>id</code>	The ID for the Shiny module.
<code>data</code>	A reactive containing the data frame to plot. Provide a summarized table with columns for labels and aggregated values. Values that are not data frames are coerced with <code>as.data.frame()</code> ; a NULL value is treated as "not ready yet" and the module waits for data.
<code>hide.inputs</code>	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.
<code>hide.tabs</code>	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.
<code>defaults</code>	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. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> , in which case the input tracks it as the parent app's state changes; see <code>setup_reactive_defaults()</code> .

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

<code>data_list</code>	An optional named list of data frames. If NULL (the default), the module's example dataset is used, along with its showcase defaults.
<code>defaults</code>	A named list of input IDs and their default values to apply on startup. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> to have the input follow the parent app's state; see <code>setup_reactive_defaults()</code> .
<code>hide.inputs</code>	A character vector of input IDs to hide. Their values are still initialized and used, but the controls are not shown in the UI.
<code>hide.tabs</code>	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 on `example_population` with the settings the module gallery (`moduleGalleryApp()`) opens this module on, so its main features are on show from the start; any defaults you pass are applied over those. 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. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> ; it is resolved with <code>shiny::isolate()</code> to seed the control, and the module then keeps it live (see <code>setup_reactive_defaults()</code>).
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: first categorical variable other than `x.data`, 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)
- `legend.show` - Show the legend and any colorbar (UI: "Show Legend", default: TRUE)
- `legend.font.family` - Font family of the legend title and labels (UI: "Legend Font", default: "Arial")
- `legend.font.color` - Font color of the legend title and labels (UI: "Legend Font Color", default: "#000000")
- `legend.title.size` - Legend title font size (UI: "Legend Title Size", default: 14)
- `legend.text.size` - Legend entry label font size (UI: "Legend Text Size", default: 12)
- `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")
- `palette.colours` - Named character vector mapping group levels to colors, e.g. `c(A = "#FF0000", B = "blue")` (UI: "Plot colors"). Seeds the picker; unnamed groups fall back to the default palette and user edits take precedence.

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

<code>id</code>	The ID for the Shiny module.
<code>resizable</code>	Logical; when TRUE (the default) the plot output is wrapped in <code>shinyjqui::jqui_resizable()</code> 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

<code>id</code>	The ID for the Shiny module.
<code>data</code>	A reactive containing the data frame to plot. Values that are not data frames are coerced with <code>as.data.frame()</code> ; a NULL value is treated as "not ready yet" and the module waits for data.
<code>hide.inputs</code>	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.
<code>hide.tabs</code>	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.
<code>defaults</code>	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. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> , in which case the input tracks it as the parent app's state changes; see <code>setup_reactive_defaults()</code> .

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

<code>data_list</code>	An optional named list of data frames. If NULL (the default), the module's example dataset is used, along with its showcase defaults.
<code>defaults</code>	A named list of input IDs and their default values to apply on startup. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> to have the input follow the parent app's state; see <code>setup_reactive_defaults()</code> .
<code>hide.inputs</code>	A character vector of input IDs to hide. Their values are still initialized and used, but the controls are not shown in the UI.
<code>hide.tabs</code>	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 on `example_bar` with the settings the module gallery (`moduleGalleryApp()`) opens this module on, so its main features are on show from the start; any defaults you pass are applied over those. 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

<code>id</code>	The ID for the Shiny module.
<code>data</code>	The data frame used for plot generation.
<code>defaults</code>	A named list of default values for the inputs. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> ; it is resolved with <code>shiny::isolate()</code> to seed the control, and the module then keeps it live (see <code>setup_reactive_defaults()</code>).
<code>title</code>	An optional title for the UI grid.
<code>columns</code>	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` - Split variable (returns a patchwork object, not supported in plotly), use `facet_by` instead
- `split_by_sep` - Only applies if `split_by` is used
- `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 - 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

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)
- 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)
- `legend.show` - Show the legend and any colorbar (UI: "Show Legend", default: TRUE)
- `legend.font.family` - Font family of the legend title and labels (UI: "Legend Font", default: "Arial")

- `legend.font.color` - Font color of the legend title and labels (UI: "Legend Font Color", default: "#000000")
- `legend.title.size` - Legend title font size (UI: "Legend Title Size", default: 14)
- `legend.text.size` - Legend entry label font size (UI: "Legend Text Size", default: 12)
- `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")
- `palette.colours` - Named character vector mapping group levels to colors, e.g. `c(A = "#FF0000", B = "blue")` (UI: "Plot colors"). Seeds the picker; unnamed groups fall back to the default palette and user edits take precedence.
- `palette.name` - Gradient palette used when `fill.by` is numeric (UI: "Color Palette", default: "viridis")

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

<code>id</code>	The ID for the Shiny module.
<code>resizable</code>	Logical; when TRUE (the default) the plot output is wrapped in <code>shinyjqui::jqui_resizable()</code> 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

<code>id</code>	The ID for the Shiny module.
<code>data</code>	A reactive containing the data frame to plot. Values that are not data frames are coerced with <code>as.data.frame()</code> ; a NULL value is treated as "not ready yet" and the module waits for data.
<code>hide.inputs</code>	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.
<code>hide.tabs</code>	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.
<code>defaults</code>	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. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> , in which case the input tracks it as the parent app's state changes; see <code>setup_reactive_defaults()</code> .

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

<code>data_list</code>	An optional named list of data frames. If NULL (the default), the module's example dataset is used, along with its showcase defaults.
<code>defaults</code>	A named list of input IDs and their default values to apply on startup. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> to have the input follow the parent app's state; see <code>setup_reactive_defaults()</code> .
<code>hide.inputs</code>	A character vector of input IDs to hide. Their values are still initialized and used, but the controls are not shown in the UI.
<code>hide.tabs</code>	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 on `example_demographics` with the settings the module gallery (`moduleGalleryApp()`) opens this module on, so its main features are on show from the start; any defaults you pass are applied over those. 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. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> ; it is resolved with <code>shiny::isolate()</code> to seed the control, and the module then keeps it live (see <code>setup_reactive_defaults()</code>).
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 reimplement 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: TRUE)
- `stat.paired` - Use a paired test (UI: "Paired Test", default: FALSE)
- `stat.pairs` - Group comparisons to test (UI: "Comparisons", multiple selection, default: all pairs). As a default, a character vector of "A vs B" strings naming levels of `x.data` (or of `group.by`, when set), e.g. `c("Entry vs Mid", "Mid vs Senior")`; either order matches
- `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")

Brackets are stacked above the data, so none are drawn while the plot is rotated (the values then run along the x-axis); the test results are still included in the source data download. Under a free y facet scale ("free", "free_y") each panel's brackets sit above that panel's own data, and the Y axis min/max are not applied.

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)
- `legend.show` - Show the legend and any colorbar (UI: "Show Legend", default: TRUE)
- `legend.font.family` - Font family of the legend title and labels (UI: "Legend Font", default: "Arial")
- `legend.font.color` - Font color of the legend title and labels (UI: "Legend Font Color", default: "#000000")
- `legend.title.size` - Legend title font size (UI: "Legend Title Size", default: 14)
- `legend.text.size` - Legend entry label font size (UI: "Legend Text Size", default: 12)
- `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")
- `palette.colours` - Named character vector mapping group levels to colors, e.g. `c(A = "#FF0000", B = "blue")` (UI: "Plot colors"). Seeds the picker; unnamed groups fall back to the default palette and user edits take precedence.

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 <code>shinyjqui::jqui_resizable()</code> 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. Values that are not data frames are coerced with <code>as.data.frame()</code> ; a NULL value is treated as "not ready yet" and the module waits for data.
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. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> , in which case the input tracks it as the parent app's state changes; see <code>setup_reactive_defaults()</code> .

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

<code>data_list</code>	An optional named list of data frames. If NULL (the default), the module's example dataset is used, along with its showcase defaults.
<code>defaults</code>	A named list of input IDs and their default values to apply on startup. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> to have the input follow the parent app's state; see <code>setup_reactive_defaults()</code> .
<code>hide.inputs</code>	A character vector of input IDs to hide. Their values are still initialized and used, but the controls are not shown in the UI.
<code>hide.tabs</code>	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 on `example_demographics` with the settings the module gallery (`moduleGalleryApp()`) opens this module on, so its main features are on show from the start; any defaults you pass are applied over those. 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. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> ; it is resolved with <code>shiny::isolate()</code> to seed the control, and the module then keeps it live (see <code>setup_reactive_defaults()</code>).
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)
- `addBars` - 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)
- `legend.show` - Show the legend and any colorbar (UI: "Show Legend", default: TRUE)
- `legend.font.family` - Font family of the legend title and labels (UI: "Legend Font", default: "Arial")
- `legend.font.color` - Font color of the legend title and labels (UI: "Legend Font Color", default: "#000000")
- `legend.title.size` - Legend title font size (UI: "Legend Title Size", default: 14)
- `legend.text.size` - Legend entry label font size (UI: "Legend Text Size", default: 12)
- `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")
- `palette.colours` - Named character vector mapping group levels to colors, e.g. `c(A = "#FF0000", B = "blue")` (UI: "Plot colors"). Seeds the picker; unnamed groups fall back to the default palette and user edits take precedence.
- `single.fill.color` - Fill color used when no grouping is set (UI: "Fill color")

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

<code>id</code>	The ID for the Shiny module.
<code>resizable</code>	Logical; when TRUE (the default) the plot output is wrapped in <code>shinyjquery::jquery_resizable()</code> 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

<code>id</code>	character unique ID for the shiny namespace.
<code>data</code>	reactive A reactive expression returning a data frame to be plotted. Values that are not data frames are coerced with <code>as.data.frame()</code> ; a NULL value is treated as "not ready yet" and the module waits for data.
<code>hide.inputs</code>	character vector of input IDs to hide in the UI. Default is NULL.
<code>hide.tabs</code>	character vector of tab names to hide within the module. Default is NULL.
<code>defaults</code>	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. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> , in which case the input tracks it as the parent app's state changes; see <code>setup_reactive_defaults()</code> .

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

<code>data_list</code>	An optional named list of data frames. If NULL (the default), the module's example dataset is used, along with its showcase defaults.
<code>defaults</code>	A named list of input IDs and their default values to apply on startup. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> to have the input follow the parent app's state; see <code>setup_reactive_defaults()</code> .
<code>hide.inputs</code>	A character vector of input IDs to hide. Their values are still initialized and used, but the controls are not shown in the UI.
<code>hide.tabs</code>	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 on `example_markers` with the settings the module gallery (`moduleGalleryApp()`) opens this module on, so its main features are on show from the start; any defaults you pass are applied over those. 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. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> ; it is resolved with <code>shiny::isolate()</code> to seed the control, and the module then keeps it live (see <code>setup_reactive_defaults()</code>).
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.name - 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.
- `legend.show` - Show the legend, colorbar and size legend (UI: "Show Legend", default: TRUE)
- `legend.font.family` - Font family of the legend title and labels (UI: "Legend Font", default: "Arial")
- `legend.font.color` - Font color of the legend title and labels (UI: "Legend Font Color", default: "#000000")
- `legend.title.size` - Legend title font size (UI: "Legend Title Size", default: 14)
- `legend.text.size` - Legend entry label font size (UI: "Legend Text Size", default: 12)

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

<code>id</code>	The ID for the Shiny module.
<code>resizable</code>	Logical; when TRUE (the default) the plot output is wrapped in <code>shinyjquery::jquery_resizable()</code> 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

<code>id</code>	The ID for the Shiny module.
<code>data</code>	A reactive containing the data frame to plot. Values that are not data frames are coerced with <code>as.data.frame()</code> ; a NULL value is treated as "not ready yet" and the module waits for data.
<code>hide.inputs</code>	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.
<code>hide.tabs</code>	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. An entry may also be a `shiny::reactive()` or `shiny::reactiveVal()`, in which case the input tracks it as the parent app's state changes; see `setup_reactive_defaults()`.

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

<code>data_list</code>	An optional named list of data frames. If <code>NULL</code> (the default), the module's example dataset is used, along with its showcase defaults.
<code>defaults</code>	A named list of input IDs and their default values to apply on startup. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> to have the input follow the parent app's state; see <code>setup_reactive_defaults()</code> .
<code>hide.inputs</code>	A character vector of input IDs to hide. Their values are still initialized and used, but the controls are not shown in the UI.
<code>hide.tabs</code>	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 on `example_demographics` with the settings the module gallery (`moduleGalleryApp()`) opens this module on, so its main features are on show from the start; any defaults you pass are applied over those. 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

<code>id</code>	The ID for the Shiny module.
<code>data</code>	The data frame used for plot generation.
<code>defaults</code>	A named list of default values for the inputs. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> ; it is resolved with <code>shiny::isolate()</code> to seed the control, and the module then keeps it live (see <code>setup_reactive_defaults()</code>).
<code>title</code>	An optional title for the UI grid.
<code>columns</code>	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)
- addBars - 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)
- `legend.show` - Show the legend and any colorbar (UI: "Show Legend", default: TRUE)
- `legend.font.family` - Font family of the legend title and labels (UI: "Legend Font", default: "Arial")
- `legend.font.color` - Font color of the legend title and labels (UI: "Legend Font Color", default: "#000000")
- `legend.title.size` - Legend title font size (UI: "Legend Title Size", default: 14)
- `legend.text.size` - Legend entry label font size (UI: "Legend Text Size", default: 12)
- `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")
- `palette.colours` - Named character vector mapping group levels to colors, e.g. `c(A = "#FF0000", B = "blue")` (UI: "Plot colors"). Seeds the picker; unnamed groups fall back to the default palette and user edits take precedence.
- `single.fill.color` - Fill color used when no grouping is set (UI: "Fill color")

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 <code>shinyjquery::jquery_resizable()</code> 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. Values that are not data frames are coerced with <code>as.data.frame()</code> ; a NULL value is treated as "not ready yet" and the module waits for data.
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. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> , in which case the input tracks it as the parent app's state changes; see <code>setup_reactive_defaults()</code> .

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

<code>data_list</code>	An optional named list of data frames. If NULL (the default), the module's example dataset is used, along with its showcase defaults.
<code>defaults</code>	A named list of input IDs and their default values to apply on startup. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> to have the input follow the parent app's state; see <code>setup_reactive_defaults()</code> .
<code>hide.inputs</code>	A character vector of input IDs to hide. Their values are still initialized and used, but the controls are not shown in the UI.
<code>hide.tabs</code>	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 on `example_bar` with the settings the module gallery (`moduleGalleryApp()`) opens this module on, so its main features are on show from the start; any defaults you pass are applied over those. 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

<code>id</code>	The ID for the Shiny module.
<code>data</code>	The data frame used for plot generation.
<code>defaults</code>	A named list of default values for the inputs. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> ; it is resolved with <code>shiny::isolate()</code> to seed the control, and the module then keeps it live (see <code>setup_reactive_defaults()</code>).
<code>title</code>	An optional title for the UI grid.
<code>columns</code>	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` - Split variable (returns a patchwork object, not supported in plotly), use `facet_by` instead
- `split_by_sep` - Only applies if `split_by` is used
- `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: "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)
- `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` - Offset of the category labels along the value axis (UI: "Category Label Position", default: 0). At 0 each label sits beside the base of its bar; positive values move it over the bar, negative values further out on the other side. 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)
- `legend.show` - Show the legend and any colorbar (UI: "Show Legend", default: TRUE)
- `legend.font.family` - Font family of the legend title and labels (UI: "Legend Font", default: "Arial")
- `legend.font.color` - Font color of the legend title and labels (UI: "Legend Font Color", default: "#000000")
- `legend.title.size` - Legend title font size (UI: "Legend Title Size", default: 14)

- `legend.text.size` - Legend entry label font size (UI: "Legend Text Size", default: 12)
- `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")
- `palette.colours` - Named character vector mapping group levels to colors, e.g. `c(A = "#FF0000", B = "blue")` (UI: "Plot colors"). Seeds the picker; unnamed groups fall back to the default palette and user edits take precedence.
- `gradient.palette` - Gradient palette used when `fill_by` is numeric (UI: "Color palette", default: "Spectral")

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

<code>id</code>	The ID for the Shiny module.
<code>resizable</code>	Logical; when TRUE (the default) the plot output is wrapped in <code>shinyjqui::jqui_resizable()</code> 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. Values that are not data frames are coerced with <code>as.data.frame()</code> ; a NULL value is treated as "not ready yet" and the module waits for data.
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. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> , in which case the input tracks it as the parent app's state changes; see <code>setup_reactive_defaults()</code> .

Value

The moduleServer function for the SplitBarPlot module.

Author(s)

Jacob Martin, Jared Andrews

See Also

`plotthis::SplitBarPlot()`, `organize_inputs()`, `plotthis_SplitBarPlotInputsUI()`, `plotthis_SplitBarPlotOutputsUI()`, `plotthis_SplitBarPlotApp()`

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

<code>df</code>	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.
<code>theta</code>	Character, name of the column to use for the angular categories (axes).
<code>r</code>	Character, name of the column to use for the radial values.
<code>group</code>	Optional character, name of the column to use for grouping multiple traces. If NULL, a single trace is plotted. Default: NULL.
<code>colors</code>	Optional character vector of hex colors for the traces. If named, values are matched to the group values; otherwise colours are recycled.
<code>palette</code>	Optional character vector of fallback colors used when <code>colors</code> is not supplied or missing values are present.
<code>fill</code>	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".
<code>line.width</code>	Numeric, width of the trace lines in pixels. Default: 2.

<code>line.dash</code>	Character, line dash style. Options: "solid", "dot", "dash", "longdash", "dash-dot", "longdashdot". Default: "solid".
<code>marker.size</code>	Numeric, size of the markers on the trace. Default: 5.
<code>marker.symbol</code>	Character, marker symbol. Options: "circle", "square", "diamond", "cross", "x", "triangle-up", etc. Default: "circle".
<code>opacity</code>	Numeric, opacity of the traces (0-1). Default: 0.6.
<code>radial.visible</code>	Logical, whether to show the radial axis. Default: TRUE.
<code>radial.range</code>	Optional numeric vector of length 2 specifying the range of the radial axis (e.g., <code>c(0, 100)</code>). If NULL, automatically determined. Default: NULL.
<code>radial.showline</code>	Logical, whether to show the radial axis line. Default: TRUE.
<code>radial.linecolor</code>	Character, hex color for the radial axis line. Default: "#444444".
<code>radial.gridcolor</code>	Character, hex color for the radial grid lines. Default: "#EEEEEE".
<code>angular.direction</code>	Character, direction of angular axis. Options: "clockwise" or "counterclockwise". Default: "clockwise".
<code>angular.rotation</code>	Numeric, rotation angle for the angular axis in degrees. Default: 90.
<code>angular.gridcolor</code>	Character, hex color for the angular grid lines. Default: "#EEEEEE".
<code>show.legend</code>	Logical, whether to display the legend. Default: TRUE.
<code>legend.orientation</code>	Character, legend orientation. Options: "h" (horizontal) or "v" (vertical). Default: "h".
<code>legend.x</code>	Numeric, horizontal legend position offset (0-1). Default: 0.5.
<code>legend.y</code>	Numeric, vertical legend position offset (-1 to 1). Default: -0.1.
<code>legend.font.family</code>	Character, font family for the legend text. Default: "Arial".
<code>legend.font.size</code>	Numeric, font size for the legend text. Default: 12.
<code>legend.font.color</code>	Character, hex color for the legend text. Default: "#000000".
<code>title.text</code>	Character, main plot title text. Default: "".
<code>title.font.family</code>	Character, font family for the title text. Default: "Arial".
<code>title.font.size</code>	Numeric, font size for the title text. Default: 18.
<code>title.font.color</code>	Character, hex color for the title text. Default: "#000000".
<code>title.x</code>	Numeric, horizontal position for the plot title (0-1). Default: 0.5.
<code>bgcolor</code>	Character, hex color for the plot background. Default: "#FFFFFF".
<code>polar.bgcolor</code>	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

<code>data_list</code>	An optional named list of data frames. If <code>NULL</code> (the default), the module's example dataset is used, along with its showcase defaults. Each data frame should contain columns for categories (theta) and values (r). For multiple traces, include a grouping column.
<code>defaults</code>	A named list of input IDs and their default values to apply on startup. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> to have the input follow the parent app's state; see <code>setup_reactive_defaults()</code> .
<code>hide.inputs</code>	A character vector of input IDs to hide. Their values are still initialized and used, but the controls are not shown in the UI.
<code>hide.tabs</code>	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 on `example_skills` with the settings the module gallery (`moduleGalleryApp()`) opens this module on, so its main features are on show from the start; any defaults you pass are applied over those. 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. An entry may also be a shiny::reactive() or shiny::reactiveVal() ; it is resolved with shiny::isolate() to seed the control, and the module then keeps it live (see setup_reactive_defaults()).
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")
- `trace.colors` - Named character vector mapping group levels to colors, e.g. `c(A = "#FF0000", B = "blue")` (UI: "Trace colors"). Seeds the picker; unnamed groups fall back to the default palette and user edits take precedence.
- `single.color` - Trace color used when no grouping is set (UI: "Trace color:", default: "#1F77B4")

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	<i>Output UI components for the radarPlot module</i>
-------------------	--

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 <code>shinyjqui::jqui_resizable()</code> 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	<i>Server logic for radarPlot module</i>
-----------------	--

Description

Server logic for radarPlot module

Usage

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

Arguments

<code>id</code>	The ID for the Shiny module.
<code>data</code>	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. Values that are not data frames are coerced with <code>as.data.frame()</code> ; a NULL value is treated as "not ready yet" and the module waits for data.
<code>hide.inputs</code>	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.
<code>hide.tabs</code>	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.
<code>defaults</code>	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. An entry may also be a <code>shiny::reactive()</code> or <code>shiny::reactiveVal()</code> , in which case the input tracks it as the parent app's state changes; see <code>setup_reactive_defaults()</code> .

Value

The `moduleServer` function for the `radarPlot` module.

Author(s)

Jacob Martin

See Also

`radarPlot()`, `radarPlotInputsUI()`, `radarPlotOutputUI()`, `radarPlotApp()`

<code>recycle_line_style</code>	<i>Recycle style vector to match line count</i>
---------------------------------	---

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

<code>values</code>	Vector. Style values (colors, widths, etc.) to recycle.
<code>n</code>	Integer. Target length (number of lines).
<code>default</code>	The default value to use if <code>values</code> 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_annotation_inputs

Reset uniform Annotation inputs to defaults

Description

Resets the point highlighting and annotation inputs created by [uniform_annotation_inputs_ui\(\)](#) back to their default values.

Usage

```
reset_annotation_inputs(session, defaults = NULL, choices = "")
```

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.
choices	Character vector of valid "Annotate By" column names, used to validate the supplied default.

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_annotation_inputs(session, defaults, choices = c("", names(data())))

## End(Not run)
```

reset_axes_inputs	<i>Reset uniform axes inputs to defaults</i>
-------------------	--

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

<code>session</code>	The Shiny session object (from <code>moduleServer</code>).
<code>defaults</code>	A named list of default values to reset to, or <code>NULL</code> 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_axis_title_text	<i>Drop persisted axis-title text edits</i>
-----------------------	---

Description

Removes any captured text edit for the given axis-title annotation keys from an edit store, so the axis title is regenerated from its (possibly adjusted) data-derived label on the next rebuild rather than restoring a stale manual edit. Any captured position for those titles is left intact, so a dragged title keeps its place. Intended to be called when the plotted variable for an axis changes (e.g. from an `observeEvent()` or inside the plot-building reactive), matching the convention that a manual title only makes sense for the variable it was written for.

Usage

```
reset_axis_title_text(store, keys = c("axis:x", "axis:y"))
```

Arguments

store	The list returned by <code>setup_manual_edits()</code> .
keys	Character vector of axis annotation keys to clear text for. Defaults to both axis titles, <code>c("axis:x", "axis:y")</code> . The #<occurrence> suffix added by <code>.annotation_edit_keys()</code> is ignored when matching.

Value

Invisibly, TRUE if any stored text was removed, otherwise FALSE.

Author(s)

Jared Andrews

See Also

`setup_manual_edits()`, `finalize_manual_edits()`.

Examples

```
## Not run:
# Regenerate the y-axis title whenever the plotted variable changes:
observeEvent(input$var, reset_axis_title_text(edit_store, "axis:y"),
  ignoreInit = TRUE)

## End(Not run)
```

reset_legend_inputs	<i>Reset uniform Legend inputs to defaults</i>
---------------------	--

Description

Resets the legend inputs (visibility, font family and color, and the 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	<i>Reset uniform lines inputs to defaults</i>
--------------------	---

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

<code>session</code>	The Shiny session object (from <code>moduleServer</code>).
<code>include.fit.lines</code>	Logical; if TRUE, also resets the fit-line inputs (<code>best.fit</code> , <code>line.best.smoothness</code> , <code>line.best.colour</code> , <code>linear.model</code>). Default is FALSE.
<code>defaults</code>	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	<i>Reset uniform Plotly inputs to defaults</i>
---------------------	--

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 <code>moduleServer</code>).
defaults	A named list of default values to reset to, or <code>NULL</code> 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_column_targets`*Translate column names or positions into DT column targets*

Description

Maps a set of columns onto the zero-based targets indices DataTables expects inside a `columnDefs` entry. This is what `dataFilterServer()` uses to honour its `hide.columns` argument, but it is useful for any hand-rolled `DT::datatable()` where columns are referred to by name rather than by position – hiding them, setting widths, disabling ordering, and so on.

Usage

```
resolve_column_targets(data, columns, rownames = FALSE)
```

Arguments

<code>data</code>	A data frame, or a character vector of the column names in the order they are passed to <code>DT::datatable()</code> .
<code>columns</code>	NULL, a character vector of column names, or a numeric vector of one-based column positions to resolve.
<code>rownames</code>	Logical. Whether the table is drawn with a row-names column (the <code>rownames</code> argument of <code>DT::datatable()</code>). When TRUE, row names occupy column 0 and every data column shifts one to the right, so the returned targets are shifted to match. Defaults to FALSE.

Details

Columns that do not exist are dropped with a warning rather than raising an error, so a table fed by a changing data frame keeps rendering when a column comes and goes.

Value

An integer vector of zero-based column indices, possibly empty.

Author(s)

Jared Andrews

See Also

[dataFilterServer\(\)](#)

Examples

```
# Hide two columns of a plain DT table by name.
targets <- resolve_column_targets(iris, c("Petal.Length", "Petal.Width"))
targets

if (interactive()) {
  DT::datatable(
    iris,
    rownames = FALSE,
    options = list(
      columnDefs = list(list(visible = FALSE, targets = targets))
    )
  )
}
```

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

<code>n_facets</code>	Integer, number of facet panels.
<code>facet.nrow</code>	Optional integer, user-requested number of rows.
<code>facet.ncol</code>	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	<i>Resolve a color palette for plot groups</i>
-----------------	--

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,  
  manual_colors = 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 manual_colors or default_palette.
default_palette	A character vector of fallback colors to use when no other source supplies a color for a group. Defaults to "#000000" (black) if not provided.
manual_colors	An optional named character vector of caller-supplied colors, typically taken from a module's defaults. Used for groups that selected_colors does not name.

Details

Colors are layered in order of increasing precedence: default_palette, then manual_colors, then selected_colors. A user's on-screen choice therefore always wins over a caller-supplied mapping, and a caller-supplied mapping wins over the module's stock palette.

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")

# Caller-supplied mapping fills groups the user has not picked
resolve_palette(groups, c(A = "#FF0000"), "CCCCC", c(B = "#00FF00", C = "#0000FF"))
# Returns: c(A = "#FF0000", B = "#00FF00", C = "#0000FF")
```

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	<i>Safely resolve an adjustment function name to an actual function</i>
----------------------	---

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, params = NULL)
```

Arguments

input	The Shiny input object from the module server, should have both <code>auto.update</code> (boolean) and <code>update</code> (button) inputs.
params	Optional reactive-defaults store from <code>setup_reactive_defaults()</code> , or <code>NULL</code> . When supplied, an <code>input\$<key></code> read whose key is backed by the store resolves from the store instead of the client input, so a parameter driven by the parent app updates in a single render.

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. With `params = NULL` this is `identity` if auto-update is enabled (expressions will be reactive), or `isolate` if auto-update is disabled (expressions will not trigger reactivity). With a store supplied it is a wrapper with the same isolation semantics that additionally redirects store-backed `input$<key>` reads.

Author(s)

Jared Andrews

See Also[setup_reactive_defaults\(\)](#)**Examples**

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

  ui <- fluidPage(
    viz_select_input("x_var", "X variable", choices = names(mtcars), selected = "wt"),
    viz_select_input("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_axis_range*Track the axis limits a plot should draw with*

Description

Modules derive an axis range on the server and push it into their own `y.min/y.max` controls with `updateNumericInput()`, which is a client round-trip: the plot renders once with the stale limits and again when the browser echoes the new ones. `setup_axis_range()` gives the plot a server-side value to read instead, held in a `shiny::reactiveVal()` that only invalidates on a real change, so the echo costs nothing while a limit the user types comes straight through.

Usage

```
setup_axis_range(
  input,
  session,
  min_key = "y.min",
  max_key = "y.max",
  headroom = NULL,
  params = NULL
)
```

Arguments

<code>input</code>	The Shiny input object from inside <code>moduleServer()</code> .
<code>session</code>	The Shiny session object from inside <code>moduleServer()</code> .
<code>min_key, max_key</code>	Character strings — the limit controls' input ids, without namespacing.
<code>headroom</code>	Optional function of no arguments returning the smallest acceptable maximum, or NULL for none. Returning NULL or a non-finite value leaves the requested maximum alone.
<code>params</code>	Optional reactive-defaults store from <code>setup_reactive_defaults()</code> , or NULL. A limit backed by the store follows it rather than the client input.

Details

Pass `headroom` when something is drawn above the data that the limits have to clear — significance brackets, for instance, via `stat_bracket_y_max()`. It is evaluated reactively, and the maximum is raised to meet it whenever the requested one falls short; the control is updated to match, so the number on screen is the limit actually in use rather than one the plot has quietly overridden. The maximum is only ever raised this way, so a larger limit the user chose is left alone.

Seed the store alongside any `update*Input()` call that sets the limits (the y-data observer, the Reset button) so the echo arrives as a no-op:

```
y_range_store(list(min = y_range$min, max = y_range$max))
updateNumericInput(session, "y.min", value = y_range$min)
updateNumericInput(session, "y.max", value = y_range$max)
```

Value

A `shiny::reactiveVal()` holding `list(min = , max = )`. Call it with no arguments to read, and with a value to seed.

Author(s)

Jared Andrews

See Also

`stat_bracket_y_max()`, `setup_group_colors()`

Examples

```

if (interactive()) {
  library(shiny)

  server <- function(input, output, session) {
    y_range_store <- setup_axis_range(
      input, session,
      headroom = function() {
        if (!isTRUE(input$stats.enabled)) {
          return(NULL)
        }
        stat_bracket_y_max(iris, x = "Species", y = "Sepal.Length")
      }
    )

    output$plot <- renderPlot({
      lims <- y_range_store()
      plot(iris$Sepal.Length, ylim = c(lims$min, lims$max))
    })
  }
}

```

setup_group_colors	<i>Track the group-to-color mapping a plot should draw with</i>
--------------------	---

Description

A `multiColorPicker()` is rebuilt by `renderUI()` whenever the group set changes, and the freshly built widget reports its value back on a client round-trip. A plot that depends on the picker's raw `input$<key>` therefore rebuilds when that value lands, even at startup where the reported value is exactly what the server had already seeded the picker with.

Usage

```

setup_group_colors(
  input,
  key,
  groups,
  default_palette = NULL,
  defaults = NULL,
  params = NULL
)

```

Arguments

input	The Shiny input object from inside <code>moduleServer()</code> .
key	Character string — the picker's input id, without namespacing, e.g. <code>"palette.colours"</code> .
groups	A <code>reactive()</code> yielding the character vector of group levels currently in play.

default_palette	A character vector of fallback colors.
defaults	A named list of default values, or NULL. A named color mapping stored under key seeds groups the user has not picked.
params	Optional reactive-defaults store from <code>setup_reactive_defaults()</code> , or NULL. When key is backed by the store, the mapping follows it rather than the client input, matching what <code>setup_auto_update_logic()</code> would have done for a direct <code>input\$<key></code> read.

Details

`setup_group_colors()` gives the module a server-side channel for the mapping instead. It resolves the palette itself (via `resolve_palette()`) as soon as the group set is known, and holds the result in a `shiny::reactiveVal()`, which only invalidates on a *changed* value. A rebuilt picker echoing the mapping already in use therefore costs nothing, while a color the user actually picks comes straight through.

Use it in three places:

- Create the store next to the module's group-levels reactive.
- Seed it inside the picker's `renderUI()` with the same `initial_colors` the widget is built from, so the mapping is right even when the render was deferred (a picker on a hidden tab is suspended).
- Read `isolate_fn(store())` in the plot reactive, in place of `isolate_fn(input$<key>)`.

Note that `freezeReactiveValue()` does not cover this case: inside a `renderUI()` it pauses only the readers that run after it in that flush, and at startup the plot output runs first.

Value

A `shiny::reactiveVal()` holding a named character vector of colors aligned to the current groups, or NULL before any groups exist. Call it with no arguments to read, and with a value to seed.

Author(s)

Jared Andrews

See Also

`resolve_palette()`, `multiColorPicker()`, `setup_auto_update_logic()`

Examples

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

  server <- function(input, output, session) {
    groups <- reactive(levels(as.factor(iris$Species)))
    palette_store <- setup_group_colors(
      input, "palette.colours", groups,
      default_palette = dittoViz::dittoColors()
    )
  }
}
```

```

    )

    output$palette.selection <- renderUI({
      initial_colors <- isolate(palette_store())
      multiColorPicker(
        session$ns("palette.colours"),
        groups = groups(), colors = initial_colors
      )
    })

    output$plot <- renderPlot(barplot(1:3, col = palette_store()))
  }
}

```

setup_manual_edits	<i>Set up persistent manual plot layout edits across re-renders</i>
--------------------	---

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 <code>session\$ns("<plot-type>")</code> (e.g. <code>session\$ns("scatter")</code>). It scopes the captured <code>plotly_relayout</code> events to this plot and must match the <code>plot_source</code> later passed to <code>finalize_manual_edits()</code> .

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 layout 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)
```

`setup_reactive_defaults`*Resolve reactive defaults entries into a server-side parameter store*

Description

Plot module parameters normally travel through client-side inputs: the value is seeded into a control by `*InputsUI()` and read back at render time as `input$<key>`. That makes it impossible for a parent app to drive a parameter from app state without a visible double render. `update*Input()` is an asynchronous client round-trip, so the plot renders once with the stale value and again when the new value arrives from the browser.

Usage

```
setup_reactive_defaults(defaults, input, session)
```

Arguments

<code>defaults</code>	A named list of default values, or NULL. Individual entries may be a <code>reactive()/reactiveVal</code> .
<code>input</code>	The Shiny input object from inside <code>moduleServer()</code> .
<code>session</code>	The Shiny session object from inside <code>moduleServer()</code> .

Details

`setup_reactive_defaults()` fixes that by giving the module a *server-side* channel. Any defaults entry that is a `shiny::reactive()` or `shiny::reactiveVal()` is mirrored into an internal store that updates in the same reactive flush as the parent's data, so the plot renders once. The store is the render's source of truth (see `setup_auto_update_logic()`); the on-screen control is kept in sync separately and remains user-editable.

Semantics:

- **Precedence.** An external change always wins, overwriting a value the user had typed into the control.
- **User edits.** Editing the control writes back to the store, so manual overrides work exactly as they do for static defaults.
- **Reset.** The module's Reset button restores the reactive's *current* value, not the value it held at startup, because `get_default()` resolves the reactive when the reset observer runs.
- **Recognised forms.** Only `reactive()` and `reactiveVal` are treated as reactive defaults (via `shiny::is.reactive()`). A plain function is kept as a literal default value.

Control sync is cosmetic and uses a generic `sendInputMessage()`, which covers the standard text, numeric, checkbox, select, colour and switch inputs, plus `multiColorPicker()` when the value is a named vector of colors. A composite widget that ignores that message will simply not re-display the new value; the plot is still correct, because the render reads the store.

Value

NULL when defaults holds no reactive entries, in which case modules behave exactly as they did before. Otherwise a list with two functions, `has(key)` and `get(key)`, for [setup_auto_update_logic\(\)](#) to read.

Author(s)

Jared Andrews

See Also

[setup_auto_update_logic\(\)](#), [get_default\(\)](#)

Examples

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

  ui <- fluidPage(
    viz_select_input("sample", "Sample", c("S1", "S2", "S3")),
    dittoViz_scatterPlotInputsUI("p", mtcars),
    dittoViz_scatterPlotOutputUI("p")
  )

  server <- function(input, output, session) {
    # The plot title follows the selected sample, but stays editable.
    dittoViz_scatterPlotServer(
      "p",
      data = reactive(mtcars),
      defaults = list(main = reactive(input$sample))
    )
  }

  shinyApp(ui, server)
}
```

show_input

Show the grid cells wrapping module inputs

Description

Toggles the visibility of the `.vizmodules-input-cell` that wraps a given input (as laid out by [organize_inputs\(\)](#)). Hiding the cell rather than just the input itself lets the surrounding inputs reflow so the panel stays compact instead of leaving an empty gap, as a plain `shinyjs::hide()` on the input would.

Usage

```
show_input(session, ids)
```

Arguments

session	The module session object (provides session\$ns).
ids	Character vector of un-namespaced input IDs to toggle.

Value

Invisibly NULL, called for the side effect of running client-side JS.

Author(s)

Jared Andrews

See Also

[hide_input\(\)](#), [toggle_input_cell\(\)](#), [organize_inputs\(\)](#)

Examples

```
## Not run:
# Call inside a module server:
show_input(session, c("size", "opacity"))

## End(Not run)
```

stat_bracket_y_max	<i>Y-axis top needed to draw statistical annotation brackets in full</i>
--------------------	--

Description

Works out how high the significance brackets from [create_stat_annotations\(\)](#) will reach, so an axis range can reserve room for them up front rather than having the plot drawn with the brackets clipped or pushed outside the panel.

Usage

```
stat_bracket_y_max(
  df,
  x,
  y,
  pairs = NULL,
  group.by = NULL,
  facet.by = NULL,
  per.facet = TRUE,
  step.increase = 0.06,
  text.bump = 0.04,
  bracket.inset = 0.025,
  hide.ns = FALSE,
```

```

sig.threshold = 0.05,
test = "wilcox.test",
p.adjust.method = "holm",
paired = FALSE,
dodge.width = 1
)

```

Arguments

<code>df</code>	Data frame the statistics are computed on, holding the values as they are plotted. For a module that reshapes its data for testing (e.g. a multi-variable Y selection), pass the reshaped frame, not the raw one; for one that transforms a column before plotting it (e.g. a dittoViz var .adjustment), pass the transformed values (see adjust_column_values()).
<code>x</code>	Character; x-axis column name.
<code>y</code>	Character; y-axis column name(s). Several may be given, in which case the data range spans all of them.
<code>pairs</code>	List of length-2 character vectors, or NULL for all pairwise combinations.
<code>group.by</code>	Character or NULL; nested grouping column.
<code>facet.by</code>	Character or NULL; faceting column.
<code>per.facet</code>	Logical; whether tests are run within each facet.
<code>step.increase</code>	Numeric; fraction of the y-range between successive bracket levels. Default 0.06.
<code>text.bump</code>	Numeric; fraction of the y-range between a bracket and its label. Default 0.04.
<code>bracket.inset</code>	Numeric; endpoint inset, which affects how tightly brackets pack onto a level. Default 0.025.
<code>hide.ns</code>	Logical; whether non-significant brackets are dropped before drawing. When TRUE the tests are run so only the surviving comparisons are counted. Default FALSE.
<code>sig.threshold</code>	Numeric; significance cutoff used with <code>hide.ns</code> .
<code>test, p.adjust.method, paired</code>	Passed to compute_pairwise_stats() , and used only when <code>hide.ns</code> is TRUE. Match them to the render's settings, or the wrong comparisons are counted.
<code>dodge.width</code>	Numeric; width the <code>group.by</code> levels at one x category are dodged across. Match the plot's, or the brackets will not line up with the boxes. Default 1.

Details

The brackets are stacked above the data: each packing level sits `step.increase` of the data range above the last, the label sits `text.bump` above its bracket, and a final `step.increase` of clearance is left at the top. Which comparisons land on which level is decided by `.assign_bracket_levels()`, shared with the drawing code, so the two agree exactly.

Which comparisons there are to place depends only on the grouping columns and the user's pair selection, so no test needs to be run — except under `hide.ns = TRUE`, where the non-significant brackets are dropped before packing and the tests have to be run to know which those are. That

case repeats the work `compute_pairwise_stats()` does at render time; it is skipped for several y columns at once, where the comparisons no longer map one-to-one onto a single test run, and the result is then an upper bound.

`apply_stat_annotations()` still has the last word on the drawn range, so a bracket is never clipped even where this over- or under-estimates.

Value

A single number giving the y-axis maximum the brackets need, or NULL when nothing would be drawn.

Author(s)

Jared Andrews

See Also

`create_stat_annotations()`, `apply_stat_annotations()`

Examples

```
# Three species means three comparisons, which stack onto two levels.
stat_bracket_y_max(example_iris, x = "Species", y = "Sepal.Length")

# Compare against the raw data maximum.
max(example_iris$Sepal.Length)
```

string_to_linetypes	<i>Parse and validate linetype string to a vector</i>
---------------------	---

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("")
```

toggle_input_cell	<i>Hide or show the grid cell wrapping a module input</i>
-------------------	---

Description

Toggles the visibility of the `.vizmodules-input-cell` that wraps a given input (as laid out by [organize_inputs\(\)](#)). Hiding the cell rather than just the input itself lets the surrounding inputs reflow so the panel stays compact instead of leaving an empty gap, as a plain `shinyjs::hide()` on the input would.

Usage

```
toggle_input_cell(session, ids, show)
```

Arguments

session	The module session object (provides <code>session\$ns</code>).
ids	Character vector of un-namespaced input IDs to toggle.
show	Logical; TRUE to show the cell, FALSE to hide it.

Value

Invisibly NULL, called for the side effect of running client-side JS.

Author(s)

Jared Andrews

See Also

[hide_input\(\)](#), [show_input\(\)](#), [organize_inputs\(\)](#)

Examples

```
## Not run:
# Call inside a module server, e.g. from an observeEvent():
toggle_input_cell(session, "size", show = input$show.size)

## End(Not run)
```

`uniform_annotation_inputs_ui`*Generate uniform Annotation input UI*

Description

Creates a standardized tagList of the point highlighting and annotation inputs shared by modules that draw individual data points. Points are identified by the values of the column chosen in "Annotate By", which are also used as the label text.

Usage

```
uniform_annotation_inputs_ui(  
  ns,  
  defaults = NULL,  
  choices = "",  
  annotate.note = NULL  
)
```

Arguments

<code>ns</code>	A namespace function, typically created by <code>NS(id)</code> .
<code>defaults</code>	A named list of default values for the inputs.
<code>choices</code>	Character vector of column names offered by "Annotate By".
<code>annotate.note</code>	Character, or NULL. Extra sentence appended to the "Annotate By" tooltip, for module-specific caveats.

Value

A tagList containing the annotation input UI elements.

Author(s)

Jared Andrews

Examples

```
ns <- shiny::NS("plot1")  
uniform_annotation_inputs_ui(ns, choices = c("", "Species", "Sepal.Length"))
```

`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

<code>ns</code>	A namespace function, typically created by <code>NS(id)</code> .
<code>defaults</code>	A named list of default values for the inputs.
<code>include.rotate</code>	Logical; whether to include the "Rotate" input for swapping x and y axes (e.g., horizontal bar plots). Default is FALSE.
<code>include.flip</code>	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 inputs used across plot modules, so the legend can be shown/hidden and styled consistently regardless of plot type:

Usage

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

Arguments

<code>ns</code>	A namespace function, typically created by <code>NS(id)</code> .
<code>defaults</code>	A named list of default values for the inputs.

Details

- `legend.show` - Show the legend, colorbars and any size legend (UI: "Show Legend", default: TRUE)
- `legend.font.family` - Font family of the legend title and labels (UI: "Legend Font", default: "Arial")
- `legend.font.color` - Font color of the legend title and labels (UI: "Legend Font Color", default: "#000000")
- `legend.title.size` - Font size of the legend title (UI: "Legend Title Size", default: 14)
- `legend.text.size` - Font size of the legend entry labels (UI: "Legend Text Size", default: 12)

Apply them to a figure with [apply_legend_inputs\(\)](#) and restore them with [reset_legend_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.show = FALSE, legend.font.family = "Courier New",
  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

<code>ns</code>	A namespace function, typically created by <code>NS(id)</code> .
<code>defaults</code>	A named list of default values for the inputs.
<code>include.fit.lines</code>	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, include.shapes = TRUE)
```

Arguments

ns	A namespace function, typically created by NS(id).
defaults	A named list of default values for the inputs.
include.shapes	Logical; whether to include the controls styling shapes drawn with plotly's drawing tools. Pass FALSE for plots without cartesian axes (pie, radar, parallel coordinates), where those tools do not work. Default is TRUE.

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

<code>session</code>	The Shiny session object, typically <code>session</code> .
<code>inputId</code>	Character. The input id of the <code>multiDynamicInput</code> to update.
<code>elements</code>	Optional named list of rows (same shape as the return value) to set. Ignored when <code>clear = TRUE</code> .
<code>clear</code>	Logical. If <code>TRUE</code> , remove all rows. Overrides <code>elements</code> .

Value

Invisibly returns `NULL`. Called for its side effect.

Author(s)

Jacob Martin

update_viz_select	<i>Update a select input created by viz_select_input</i>
-------------------	--

Description

Companion to `viz_select_input()`, wrapping `shinyWidgets::updateVirtualSelect()` so the empty-string "no selection" choice is relabelled consistently with the UI side.

Usage

```
update_viz_select(session, inputId, choices = NULL, selected = NULL, ...)
```

Arguments

<code>session</code>	The session object passed to the module server function.
<code>inputId</code>	The id of the input to update.
<code>choices</code>	New choices for the input, or NULL to leave them unchanged.
<code>selected</code>	New value(s) to select. When NULL and choices is given, the current value is kept if it is still valid, falling back to the first choice. When NULL and choices is not given, the selection is left unchanged.
<code>...</code>	Further arguments passed to <code>shinyWidgets::updateVirtualSelect()</code> .

Details

Unlike `shinyWidgets::updateVirtualSelect()`, supplying choices without a selected does not clear the widget: the current value is kept when it is still one of the new choices, and the first choice is selected otherwise. This mirrors `shiny::updateSelectInput()`, which never leaves a single select with no value.

Value

No return value, called for its side effect of updating the input.

Author(s)

Jared Andrews

See Also

`viz_select_input()`, `shinyWidgets::updateVirtualSelect()`

Examples

```
library(shiny)
library(VizModules)

server <- function(input, output, session) {
  observeEvent(input$reset, {
    update_viz_select(session, "gene", choices = c("", "GENE1", "GENE2"))
  })
}
```

`use_vizmodules_skills` *Install the bundled VizModules agent skills into a project*

Description

Copies the skills that ship with VizModules into a project's skills directory, where GitHub Copilot, OpenAI Codex, Claude Code, and other tools following the [Agent Skills](#) convention will discover them.

Usage

```
use_vizmodules_skills(
  path = ".",
  client = c("agents", "copilot", "claude"),
  overwrite = FALSE
)
```

Arguments

<code>path</code>	Directory of the project to install into. The skills are written under this directory, in the subdirectory determined by <code>client</code> (for example <code>.agents/skills</code>), which is created if needed.
<code>client</code>	Which client convention to install the skills for. "agents" (the default) writes to <code>.agents/skills</code> , discovered by OpenAI Codex and by GitHub Copilot's project skill locations. "copilot" writes to <code>.github/skills</code> , GitHub Copilot's repository-native location. "claude" writes to <code>.claude/skills</code> , discovered by Claude Code. All three are equivalent copies of the same skills; choose whichever your tooling expects, or call this function more than once with different <code>client</code> values to install into several locations at once.
<code>overwrite</code>	Logical; if <code>FALSE</code> (the default) a skill whose directory already exists is skipped rather than replaced.

Details

Three skills are provided:

vizmodules-app Wiring plot modules into a Shiny app: defaults, hide.inputs/hide.tabs, the Stats tab, `createModuleApp()`, the data filter table, the figure builder, and source-data export. Ships a generated inventory of every module's column-mapping keys, colour key, and tab names.

vizmodules-custom-module Building a wrapper module on top of a base module: the namespace contract, reactive defaults, avoiding double renders, manual-edit persistence, and model-line backends.

vizmodules-new-module Authoring a module inside this package: the file trio, the required roxygen sections, the uniform input helpers, and file templates to copy.

Value

Invisibly, a character vector of the skill directories written.

Author(s)

Jared Andrews

Examples

```
# Install into a temporary project rather than the current one:
use_vizmodules_skills(tempdir())

## Not run:
# Install into the current project, refreshing any that already exist:
use_vizmodules_skills(".", overwrite = TRUE)

# Install for Claude Code specifically:
use_vizmodules_skills(".", client = "claude")

## End(Not run)
```

validate_expression	<i>Validate a user-provided expression string for safety</i>
---------------------	--

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
```

viz_select_input	<i>Create a select input that scales to very large choice sets</i>
------------------	--

Description

A drop-in replacement for `shiny::selectInput()` built on `shinyWidgets::virtualSelectInput()`. The underlying virtual-select library renders only the visible slice of the choice list, so a column with tens of thousands of unique values stays responsive and readable instead of producing an unusable wall of options.

Usage

```
viz_select_input(
  inputId,
  label,
  choices,
  selected = NULL,
  multiple = FALSE,
  search = NULL,
  width = "100%",
  ...
)
```

Arguments

<code>inputId</code>	The input slot that will be used to access the value.
<code>label</code>	Display label for the control, or NULL for no label.
<code>choices</code>	A vector or named list of values to select from, in the same form accepted by shiny::selectInput() .
<code>selected</code>	The initially selected value(s). Defaults to the first choice for single selects and nothing for multiple selects.
<code>multiple</code>	Logical, whether multiple values can be selected.
<code>search</code>	Logical, whether to show a search box. Defaults to NULL, which enables the search box once there are more than 10 choices.
<code>width</code>	The width of the input, e.g. "100%" or "400px".
<code>...</code>	Further arguments passed to shinyWidgets::virtualSelectInput() , including any raw virtual-select property such as <code>optionsCount</code> or <code>zIndex</code> .

Details

Defaults are chosen so this behaves like [shiny::selectInput\(\)](#): when `selected` is not supplied and `multiple = FALSE`, the first choice is selected. An empty-string choice (the "no selection" convention used throughout this package) is relabelled "(none)" so it is visible in the dropdown, while its value remains "".

Value

A `shiny.tag` object to be included in a UI definition.

Author(s)

Jared Andrews

See Also

[update_viz_select\(\)](#), [shinyWidgets::virtualSelectInput\(\)](#)

Examples

```
library(VizModules)
viz_select_input("gene", "Gene", choices = c("", paste0("GENE", 1:1000)))
```

Index

* datasets

- example_bar, 98
 - example_composition, 99
 - example_demographics, 100
 - example_heatmap_column_data, 101
 - example_heatmap_matrix, 102
 - example_iris, 103
 - example_markers, 103
 - example_mtcars, 104
 - example_population, 105
 - example_rnaseq, 106
 - example_sales, 107
 - example_school_earnings, 108
 - example_skills, 108
-
- add_ablines, 5
 - add_hlines, 7
 - add_plot_config, 8
 - add_reference_lines, 9
 - add_vlines, 11
 - adjust_column_values, 13
 - adjust_column_values(), 54, 238
 - adjusted_axis_label, 12
 - apply_axis_title_to_annotations, 14
 - apply_facet_subplot_spacing, 15
 - apply_facet_subplot_spacing(), 25, 26
 - apply_legend_inputs, 16
 - apply_legend_inputs(), 243
 - apply_legend_styling, 17
 - apply_legend_styling(), 16, 17
 - apply_plotly_newshape, 19
 - apply_render_margins, 20
 - apply_stat_annotations, 20
 - apply_stat_annotations(), 22, 239
 - apply_subplot_axis_styling, 22
 - apply_title_layout, 23
 - as.data.frame(), 56, 67, 76, 84, 97, 131, 146, 156, 162, 170, 179, 185, 191, 198, 206, 215
 - axis_titles_as_annotations, 24
 - axis_titles_as_annotations(), 8
 - build_facet_annotations, 25
 - build_facet_panel_borders, 26
 - build_model_row_spec, 27
 - circlize::colorRamp2(), 35
 - clean_facet_dim, 28
 - collect_source_data, 29
 - collect_source_data(), 51, 52
 - ComplexHeatmap::columnAnnotation(), 36, 41
 - ComplexHeatmap::Heatmap(), 31, 32, 34, 35, 37, 41, 42, 102
 - ComplexHeatmap::rowAnnotation(), 36, 41
 - ComplexHeatmap_HeatmapApp, 30
 - ComplexHeatmap_HeatmapApp(), 37, 42, 102
 - ComplexHeatmap_HeatmapInfoOutputUI, 32
 - ComplexHeatmap_HeatmapInfoOutputUI(), 38–40, 44
 - ComplexHeatmap_HeatmapInputsUI, 33
 - ComplexHeatmap_HeatmapInputsUI(), 32, 42
 - ComplexHeatmap_HeatmapMainOutputUI, 38
 - ComplexHeatmap_HeatmapMainOutputUI(), 32, 33, 40, 43, 44
 - ComplexHeatmap_HeatmapOutputUI, 39
 - ComplexHeatmap_HeatmapOutputUI(), 32, 33, 37–39, 42–44, 118, 119
 - ComplexHeatmap_HeatmapServer, 40
 - ComplexHeatmap_HeatmapServer(), 31–34, 37, 38, 42–44, 52, 112
 - ComplexHeatmap_HeatmapStaticOutputUI, 42
 - ComplexHeatmap_HeatmapSubOutputUI, 43
 - ComplexHeatmap_HeatmapSubOutputUI(), 33, 38–40
 - compute_pairwise_stats, 44
 - compute_pairwise_stats(), 53, 54, 148, 238, 239

- create_axis_styles, 48
- create_ggplot_axis_style, 50
- create_source_download_handler, 51
- create_source_download_handler(), 112, 134
- create_stat_annotations, 53
- create_stat_annotations(), 20, 21, 237, 239
- createModuleApp, 46
- createModuleApp(), 32, 59, 68, 77, 92, 126, 134, 143, 152, 157, 163, 171, 180, 186, 193, 199, 210, 250
- dataFilterServer, 55
- dataFilterServer(), 57, 58, 222
- dataFilterUI, 57
- dataFilterUI(), 56, 57
- default_palettes, 58
- default_palettes(), 135
- dittoViz::freqPlot(), 59–62, 65–67, 99
- dittoViz::scatterPlot(), 67–71, 74, 76
- dittoViz::yPlot(), 76–79, 83, 85
- dittoViz_freqPlotApp, 59
- dittoViz_freqPlotApp(), 65–67
- dittoViz_freqPlotInputsUI, 60
- dittoViz_freqPlotInputsUI(), 60, 66, 67
- dittoViz_freqPlotOutputUI, 66
- dittoViz_freqPlotOutputUI(), 60, 65, 67
- dittoViz_freqPlotServer, 66
- dittoViz_freqPlotServer(), 60, 65, 66
- dittoViz_scatterPlotApp, 67
- dittoViz_scatterPlotApp(), 74, 76
- dittoViz_scatterPlotInputsUI, 69
- dittoViz_scatterPlotInputsUI(), 68, 76
- dittoViz_scatterPlotOutputUI, 75
- dittoViz_scatterPlotOutputUI(), 68, 74, 76
- dittoViz_scatterPlotServer, 75
- dittoViz_scatterPlotServer(), 68, 74
- dittoViz_yPlotApp, 76
- dittoViz_yPlotApp(), 83, 85
- dittoViz_yPlotInputsUI, 78
- dittoViz_yPlotInputsUI(), 77, 85
- dittoViz_yPlotOutputUI, 83
- dittoViz_yPlotOutputUI(), 77, 83, 85
- dittoViz_yPlotServer, 84
- dittoViz_yPlotServer(), 77, 83
- downloadHandler(), 51
- draw_to_png, 85
- draw_to_png(), 52, 87, 112
- draw_to_svg, 86
- draw_to_svg(), 52, 85, 86, 112
- DT::datatable(), 222
- dumbbellPlot, 87
- dumbbellPlot(), 92, 93, 96, 97
- dumbbellPlotApp, 91
- dumbbellPlotApp(), 96, 97
- dumbbellPlotInputsUI, 92
- dumbbellPlotInputsUI(), 92, 97
- dumbbellPlotOutputUI, 96
- dumbbellPlotOutputUI(), 92, 96, 97
- dumbbellPlotServer, 97
- dumbbellPlotServer(), 92, 96
- empty_plot, 98
- example_bar, 98
- example_composition, 99
- example_demographics, 100
- example_heatmap_column_data, 101, 102
- example_heatmap_matrix, 101, 102
- example_iris, 103
- example_markers, 103
- example_mtcars, 104
- example_population, 105
- example_rnaseq, 106
- example_sales, 107
- example_school_earnings, 108
- example_skills, 108
- figureBuilderApp, 109
- figureBuilderApp(), 112, 113, 133, 134
- figureBuilderServer, 111
- figureBuilderServer(), 42, 86, 110, 113, 133
- figureBuilderUI, 113
- figureBuilderUI(), 110–112, 133
- finalize_manual_edits, 114
- finalize_manual_edits(), 219, 233, 234
- freezeReactiveValue(), 232
- generate_pair_strings, 115
- get_default, 116
- get_default(), 34, 235, 236
- get_documentation, 117
- get_model_backend, 117
- ggplot2::geom_text(), 98
- ggplot2::theme_void(), 98
- grDevices::svg(), 86

- heatmap_fit_width, 118
- hide_input, 119
- hide_input(), 237, 240
- InteractiveComplexHeatmap::HeatmapInfoOutput(), 32, 33
- InteractiveComplexHeatmap::InteractiveComplexHeatmapOutput(), 39, 40, 118
- InteractiveComplexHeatmap::makeInteractiveComplexHeatmap(), 41, 118
- InteractiveComplexHeatmap::originalHeatmapOutput(), 38, 39
- InteractiveComplexHeatmap::subHeatmapOutput(), 43, 44
- is_pure_type, 120
- linePlot, 121
- linePlot(), 125–127, 130, 131
- linePlotApp, 125
- linePlotApp(), 47, 130, 131
- linePlotInputsUI, 127
- linePlotInputsUI(), 126, 131
- linePlotOutputUI, 130
- linePlotOutputUI(), 126, 130, 131
- linePlotServer, 131
- linePlotServer(), 126, 130
- linetype_to_dash, 132
- list_model_backends, 132
- module_tack_ui, 134
- module_tack_ui(), 51, 52
- moduleGalleryApp, 133
- moduleGalleryApp(), 59, 68, 77, 92, 110–113, 126, 143, 152, 157, 163, 171, 180, 186, 193, 199, 210
- multiColorPicker, 135
- multiColorPicker(), 36, 137, 231, 232, 235
- multiDynamicInput, 137
- multiDynamicInput(), 27, 28, 36, 216, 247
- organize_inputs, 138
- organize_inputs(), 34, 37, 65, 74, 83, 96, 119, 120, 130, 145, 154, 161, 168, 177, 184, 190, 197, 204, 206, 213, 236, 237, 240
- parallelCoordinatesPlot, 140
- parallelCoordinatesPlot(), 142–145, 147
- parallelCoordinatesPlotApp(), 145, 147
- parallelCoordinatesPlotInputsUI, 143
- parallelCoordinatesPlotInputsUI(), 143, 147
- parallelCoordinatesPlotOutputUI, 145
- parallelCoordinatesPlotOutputUI(), 143, 145, 147
- parallelCoordinatesPlotServer, 146
- parallelCoordinatesPlotServer(), 143, 145
- parse_numeric_list, 147
- parse_pair_strings, 148
- piePlot, 148
- piePlot(), 152–154, 156
- piePlotApp, 151
- piePlotApp(), 154, 156
- piePlotInputsUI, 153
- piePlotInputsUI(), 152, 156
- piePlotOutputUI, 155
- piePlotOutputUI(), 152, 154, 156
- piePlotServer, 155
- piePlotServer(), 152, 154
- plotly::layout(), 21, 22
- plotly::plotly_data(), 29
- plotly::renderPlotly(), 114, 115, 233
- plotthis::AreaPlot(), 156, 158, 159, 161
- plotthis::BarPlot(), 163–166, 168
- plotthis::BoxPlot(), 170, 172, 174, 177
- plotthis::DensityPlot(), 181, 182, 184
- plotthis::DotPlot(), 186, 188–190
- plotthis::Histogram(), 194, 195, 197
- plotthis::SplitBarPlot(), 199, 201, 202, 204, 206
- plotthis_AreaPlotApp, 156
- plotthis_AreaPlotApp(), 105, 161
- plotthis_AreaPlotInputsUI, 157
- plotthis_AreaPlotOutputUI, 161
- plotthis_AreaPlotOutputUI(), 161
- plotthis_AreaPlotServer, 162
- plotthis_AreaPlotServer(), 161
- plotthis_BarPlotApp, 163
- plotthis_BarPlotApp(), 47, 98, 168
- plotthis_BarPlotInputsUI, 164
- plotthis_BarPlotInputsUI(), 47
- plotthis_BarPlotOutputUI, 169
- plotthis_BarPlotOutputUI(), 47, 168
- plotthis_BarPlotServer, 169
- plotthis_BarPlotServer(), 47, 168

- plotthis_BoxPlotApp, 170
- plotthis_BoxPlotApp(), 133, 177
- plotthis_BoxPlotInputsUI, 171
- plotthis_BoxPlotOutputUI, 178
- plotthis_BoxPlotOutputUI(), 177
- plotthis_BoxPlotServer, 178
- plotthis_BoxPlotServer(), 177
- plotthis_DensityPlotApp, 179
- plotthis_DensityPlotApp(), 184
- plotthis_DensityPlotInputsUI, 180
- plotthis_DensityPlotOutputUI, 184
- plotthis_DensityPlotOutputUI(), 184
- plotthis_DensityPlotServer, 185
- plotthis_DensityPlotServer(), 184
- plotthis_DotPlotApp, 186
- plotthis_DotPlotApp(), 103, 190
- plotthis_DotPlotInputsUI, 187
- plotthis_DotPlotOutputUI, 190
- plotthis_DotPlotOutputUI(), 190
- plotthis_DotPlotServer, 191
- plotthis_DotPlotServer(), 190
- plotthis_HistogramApp, 192
- plotthis_HistogramApp(), 197
- plotthis_HistogramInputsUI, 193
- plotthis_HistogramOutputUI, 197
- plotthis_HistogramOutputUI(), 197
- plotthis_HistogramServer, 198
- plotthis_HistogramServer(), 197
- plotthis_SplitBarPlotApp, 199
- plotthis_SplitBarPlotApp(), 98, 204, 206
- plotthis_SplitBarPlotInputsUI, 200
- plotthis_SplitBarPlotInputsUI(), 206
- plotthis_SplitBarPlotOutputUI, 205
- plotthis_SplitBarPlotOutputUI(), 204, 206
- plotthis_SplitBarPlotServer, 205
- plotthis_SplitBarPlotServer(), 204
- radarPlot, 206
- radarPlot(), 210, 212, 213, 215
- radarPlotApp, 209
- radarPlotApp(), 213, 215
- radarPlotInputsUI, 211
- radarPlotInputsUI(), 210, 215
- radarPlotOutputUI, 214
- radarPlotOutputUI(), 210, 213, 215
- radarPlotServer, 214
- radarPlotServer(), 210, 213
- recycle_line_style, 215
- register_model_backend, 216
- register_model_backend(), 132
- requireNamespace(), 41
- reset_annotation_inputs, 217
- reset_axes_inputs, 218
- reset_axis_title_text, 218
- reset_legend_inputs, 219
- reset_legend_inputs(), 17, 243
- reset_lines_inputs, 220
- reset_plotly_inputs, 221
- resolve_column_targets, 222
- resolve_column_targets(), 57
- resolve_facet_layout, 223
- resolve_facet_sharing, 224
- resolve_palette, 225
- resolve_palette(), 232
- safe_eval_filter, 226
- safe_eval_filter(), 37
- safe_resolve_adj_fxn, 227
- safe_resolve_adj_fxn(), 13
- setup_auto_update_logic, 228
- setup_auto_update_logic(), 232, 235, 236
- setup_axis_range, 229
- setup_group_colors, 231
- setup_group_colors(), 230
- setup_manual_edits, 233
- setup_manual_edits(), 114, 115, 219
- setup_reactive_defaults, 235
- setup_reactive_defaults(), 31, 34, 41, 47, 59, 60, 67–69, 76–78, 84, 91, 93, 97, 116, 126, 127, 131, 142, 144, 147, 152, 153, 156–158, 162–164, 170–172, 179–181, 185–187, 192, 194, 198–200, 206, 210, 211, 215, 228–230, 232
- shiny::downloadButton(), 51
- shiny::is.reactive(), 235
- shiny::isolate(), 34, 60, 69, 78, 93, 114, 116, 127, 144, 153, 158, 164, 172, 181, 187, 194, 200, 211
- shiny::moduleServer(), 233
- shiny::plotOutput(), 42
- shiny::reactive(), 31, 34, 41, 47, 59, 60, 67–69, 76–78, 84, 91, 93, 97, 116, 126, 127, 131, 142, 144, 147, 152, 153, 156–158, 162–164, 170–172, 179–181, 185–187, 192, 194, 198–200, 206, 210, 211, 215, 235

shiny::reactiveVal(), 29, 31, 34, 41, 47,
59, 60, 67–69, 76–78, 84, 91, 93, 97,
116, 126, 127, 131, 142, 144, 147,
152, 153, 156–158, 162–164,
170–172, 179–181, 185–187, 192,
194, 198–200, 206, 210, 211, 215,
229, 230, 232, 234, 235

shiny::reactiveValues(), 234

shiny::renderPlot(), 85, 86

shiny::selectInput(), 251, 252

shiny::shinyApp(), 48, 110, 133

shiny::updateSelectInput(), 248

shinyjqui::jquery_resizable(), 66, 75, 84,
96, 130, 146, 155, 161, 169, 178,
185, 191, 198, 205, 214

shinyjs::useShinyjs(), 113

shinyWidgets::updateVirtualSelect(),
248

shinyWidgets::virtualSelectInput(),
251, 252

show_input, 236

show_input(), 120, 240

stat_bracket_y_max, 237

stat_bracket_y_max(), 21, 230

stats::p.adjust(), 45

string_to_linetypes, 239

toggle_input_cell, 240

toggle_input_cell(), 120, 237

uniform_annotation_inputs_ui, 241

uniform_annotation_inputs_ui(), 217

uniform_axes_inputs_ui, 242

uniform_axes_inputs_ui(), 218

uniform_legend_inputs_ui, 243

uniform_legend_inputs_ui(), 16, 17, 219

uniform_lines_inputs_ui, 244

uniform_lines_inputs_ui(), 220

uniform_plotly_inputs_ui, 244

uniform_plotly_inputs_ui(), 19, 221

update_viz_select, 248

update_viz_select(), 252

updateMultiColorPicker, 245

updateMultiDynamicInput, 247

use_vizmodules_skills, 249

validate_expression, 250

viz_select_input, 251

viz_select_input(), 248