Package 'VizModules'

November 10, 2026

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

Version 0.6.0

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 colourpicker, dplyr, DT, readxl, shinyjs, scales, shinyjqui (>= 0.4.0), ggplot2, htmltools, jsonlite, methods, shinyWidgets (>= 0.7.0), htmlwidgets, rlang, 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 2026-11-10 21:50:16 UTC

Contents

add_ablines . . . 6
add_hlines . . . 8
add_marginal_plots . . . 9
add_plot_config . . . 11
add_reference_lines . . . 12
add_size_legend . . . 14
add_vlines . . . 16
adjusted_axis_label . . . 17
adjusted_values . . . 18
adjustment_fn . . . 19
adjust_column_values . . . 20
apply_axis_padding . . . 22
apply_axis_title_to_annotations . . . 23
apply_facet_subplot_spacing . . . 24
apply_highlight_styling . . . 25
apply_legend_inputs . . . 26
apply_legend_styling . . . 27
apply_level_order . . . 29
apply_plotly_newshape . . . 30
apply_render_margins . . . 31
apply_stat_annotations . . . 31
apply_subplot_axis_styling . . . 33
apply_tick_notation . . . 34
apply_title_layout . . . 35
apply_uniform_styling . . . 36
as_plotted . . . 38
auto_webgl . . . 39
axis_titles_as_annotations . . . 40
blank_to_null . . . 41
build_facet_annotations . . . 42
build_facet_panel_borders . . . 43
build_model_row_spec . . . 44
cached_structure . . . 45
clean_facet_dim . . . 47
collect_source_data . . . 48
ComplexHeatmap_HeatmapApp . . . 49
ComplexHeatmap_HeatmapInfoOutputUI . . . 51
ComplexHeatmap_HeatmapInputsUI . . . 52
ComplexHeatmap_HeatmapMainOutputUI . . . 56
ComplexHeatmap_HeatmapOutputUI . . . 58
ComplexHeatmap_HeatmapServer . . . 59
ComplexHeatmap_HeatmapStaticOutputUI . . . 61

ComplexHeatmap_HeatmapSubOutputUI . . . 62
compute_pairwise_stats . . . 63
createModuleApp . . . 65
create_axis_styles . . . 67
create_ggplot_axis_style . . . 68
create_highlight_annotations . . . 69
create_selected_annotations . . . 71
create_source_download_handler . . . 72
create_stat_annotations . . . 74
dataFilterServer . . . 77
dataFilterUI . . . 79
default_group_colors . . . 79
default_palettes . . . 80
default_stat_pairs . . . 81
discrete_choices . . . 82
dittoViz_freqPlotApp . . . 83
dittoViz_freqPlotInputsUI . . . 84
dittoViz_freqPlotOutputUI . . . 90
dittoViz_freqPlotServer . . . 91
dittoViz_scatterPlotApp . . . 92
dittoViz_scatterPlotInputsUI . . . 93
dittoViz_scatterPlotOutputUI . . . 101
dittoViz_scatterPlotServer . . . 101
dittoViz_yPlotApp . . . 103
dittoViz_yPlotInputsUI . . . 104
dittoViz_yPlotOutputUI . . . 110
dittoViz_yPlotServer . . . 111
draw_to_png . . . 112
draw_to_svg . . . 113
dumbbellPlot . . . 114
dumbbellPlotApp . . . 118
dumbbellPlotInputsUI . . . 119
dumbbellPlotOutputUI . . . 123
dumbbellPlotServer . . . 124
empty_plot . . . 125
error_bar_halfwidth . . . 126
example_bar . . . 127
example_composition . . . 128
example_demographics . . . 129
example_heatmap_column_data . . . 130
example_heatmap_matrix . . . 130
example_iris . . . 131
example_markers . . . 132
example_mtcars . . . 133
example_population . . . 134
example_rnaseq . . . 134
example_sales . . . 135
example_school_earnings . . . 136

example_skills . . . 137
facet_check . . . 137
figureBuilderApp . . . 138
figureBuilderServer . . . 140
figureBuilderUI . . . 142
finalize_manual_edits . . . 143
flatten_palette_options . . . 145
generate_pair_strings . . . 145
get_default . . . 146
get_documentation . . . 147
get_model_backend . . . 148
heatmap_fit_width . . . 149
hide_input . . . 150
is_pure_type . . . 151
level_order_choices . . . 152
level_order_input . . . 153
linePlot . . . 154
linePlotApp . . . 159
linePlotInputsUI . . . 161
linePlotOutputUI . . . 165
linePlotServer . . . 165
linetype_to_dash . . . 166
list_model_backends . . . 167
main_title_input_ids . . . 167
mark_overlay_axes . . . 168
merge_annotation_sets . . . 169
moduleGalleryApp . . . 170
module_tack_ui . . . 171
multiColorPicker . . . 172
multiDynamicInput . . . 174
na_to_null . . . 176
note_brackets_skipped . . . 176
nz_value . . . 177
organize_inputs . . . 178
parallelCoordinatesPlot . . . 179
parallelCoordinatesPlotApp . . . 181
parallelCoordinatesPlotInputsUI . . . 183
parallelCoordinatesPlotOutputUI . . . 185
parallelCoordinatesPlotServer . . . 185
parse_highlight_values . . . 186
parse_numeric_list . . . 187
parse_pair_strings . . . 188
piePlot . . . 189
piePlotApp . . . 191
piePlotInputsUI . . . 193
piePlotOutputUI . . . 195
piePlotServer . . . 196
plotthis_AreaPlotApp . . . 197

plotthis_AreaPlotInputsUI . . . 198
plotthis_AreaPlotOutputUI . . . 202
plotthis_AreaPlotServer . . . 202
plotthis_BarPlotApp . . . 203
plotthis_BarPlotInputsUI . . . 205
plotthis_BarPlotOutputUI . . . 210
plotthis_BarPlotServer . . . 210
plotthis_BoxPlotApp . . . 211
plotthis_BoxPlotInputsUI . . . 212
plotthis_BoxPlotOutputUI . . . 219
plotthis_BoxPlotServer . . . 219
plotthis_DensityPlotApp . . . 220
plotthis_DensityPlotInputsUI . . . 221
plotthis_DensityPlotOutputUI . . . 226
plotthis_DensityPlotServer . . . 226
plotthis_DotPlotApp . . . 227
plotthis_DotPlotInputsUI . . . 228
plotthis_DotPlotOutputUI . . . 232
plotthis_DotPlotServer . . . 233
plotthis_HistogramApp . . . 234
plotthis_HistogramInputsUI . . . 235
plotthis_HistogramOutputUI . . . 239
plotthis_HistogramServer . . . 240
plotthis_SplitBarPlotApp . . . 241
plotthis_SplitBarPlotInputsUI . . . 242
plotthis_SplitBarPlotOutputUI . . . 247
plotthis_SplitBarPlotServer . . . 247
radarPlot . . . 248
radarPlotApp . . . 251
radarPlotInputsUI . . . 253
radarPlotOutputUI . . . 256
radarPlotServer . . . 256
recycle_line_style . . . 257
register_model_backend . . . 258
require_data_frame . . . 259
reset_annotation_inputs . . . 260
reset_axes_inputs . . . 261
reset_axis_title_text . . . 261
reset_colorscale_inputs . . . 262
reset_group_colors . . . 263
reset_legend_inputs . . . 264
reset_lines_inputs . . . 265
reset_manual_edits . . . 266
reset_plotly_inputs . . . 267
reset_stats_inputs . . . 267
resolve_colorscale . . . 268
resolve_column_targets . . . 269
resolve_facet_layout . . . 271

resolve_facet_sharing . . . 272
resolve_palette . . . 272
safe_eval_filter . . . 274
safe_resolve_adj_fxn . . . 275
setup_auto_update_logic . . . 276
setup_axis_range . . . 277
setup_group_colors . . . 279
setup_level_order . . . 281
setup_manual_edits . . . 283
setup_reactive_defaults . . . 284
show_input . . . 286
split_color_shape_legend . . . 287
stat_bracket_headroom . . . 288
stat_bracket_y_max . . . 289
string_to_linetypes . . . 291
subplot_spacing_defaults . . . 292
toggle_facet_title_inputs . . . 293
toggle_input_cell . . . 294
uniform_annotation_inputs_ui . . . 295
uniform_axes_inputs_ui . . . 296
uniform_colorscale_inputs_ui . . . 297
uniform_legend_inputs_ui . . . 298
uniform_lines_inputs_ui . . . 299
uniform_plotly_inputs_ui . . . 300
uniform_stats_inputs_ui . . . 301
uniform_subplot_spacing_inputs_ui . . . 301
updateMultiColorPicker . . . 302
updateMultiDynamicInput . . . 304
update_viz_select . . . 305
use_vizmodules_skills . . . 306
validate_expression . . . 307
viz_select_input . . . 308
webgl_input . . . 310
with_stable_seed . . . 310

Index **312**

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

Description

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

Usage

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

Arguments

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

Details

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

Value

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

Author(s)

Jared Andrews

Examples

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

add_hlines *Build horizontal line shapes for a plotly figure*

Description

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

Usage

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

Arguments

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

Details

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

Value

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

Author(s)

Jared Andrews

Examples

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

`add_marginal_plots` *Add marginal distribution plots to a scatter plot*

Description

Draws the distribution of the plotted X values in a strip above a single-panel plotly figure, and of the Y values in a strip to its right, as a density curve, a histogram, a rug, or any combination of them. One trace is drawn per group, in that group's colour, and the rug gives each group its own lane so groups do not overplot.

Usage

```
add_marginal_plots(
  fig,
  x,
  y,
  groups = NULL,
  colors = "#000000",
  top = character(0),
  right = character(0),
  top.size = 0.15,
  right.size = 0.15,
  opacity = 0.5,
  bins = 30,
  binwidth = NULL,
  density.adjust = 1,
  rug.size = 0.3,
  rug.length = 0.8,
  rug.width = 1,
  gap = 0.02,
  show.axes = FALSE
)
```

Arguments

`fig`	A single-panel plotly figure.
`x`, `y`	Numeric vectors of the plotted X and Y values, one per point. Pass the values as drawn (after any data adjustment), not the raw data. Non-finite values are dropped.
`groups`	A vector or factor of the same length giving each point's group, or `NULL` for a single ungrouped distribution. Factor levels set the order of the rug lanes.
`colors`	Colours for the groups: a vector named by group, or an unnamed vector recycled over the groups in level order. Without groups, the first colour is used.
`top`, `right`	Character vectors of the marginals to draw above the plot (for `x`) and to its right (for `y`), any of `"density"`, `"histogram"` and `"rug"`. An empty vector draws nothing on that side.

`top.size`, `right.size`
: Fraction of the plot's height (`top.size`) or width (`right.size`) given to that side's marginals, from 0.02 to 0.8.

`opacity`
: Opacity of the density and histogram fills and of the rug lines, 0 to 1.

`bins`
: Number of histogram bins, spanning the range of all groups' values.

`binwidth`
: Histogram bin width. When given, it overrides `bins`, and the bins are aligned to multiples of it.

`density.adjust`
: Multiplier on the density bandwidth, as `adjust` in `stats::density()`. Larger values smooth more.

`rug.size`
: Share of a side's strip given to the rug when the side also has a density or histogram, 0.1 to 0.9. A rug on its own fills the side.

`rug.length`
: Length of each rug tick as a fraction of its group's lane, 0 to 1.

`rug.width`
: Line width of the rug ticks, in pixels.

`gap`
: Paper-space gap between the main panel and the marginals.

`show.axes`
: Logical. Whether each density or histogram strip shows its density or count axis, with a line, a few ticks, and their labels. The strips' value axes, which repeat the plot's own, and the rug strips stay bare. Default `FALSE`.

Details

The main panel is shrunk to make room, and each strip's value axis `matches` the main axis, so zooming or panning the plot moves its marginals with it. Paper-anchored shapes (such as the panel border ggplotly draws) and axis-title annotations from `axis_titles_as_annotations()` are moved into the shrunk panel.

Call this last, once everything else has been drawn: helpers that work per panel (fit lines, reference lines, axis styling) read the strips' extra axes as facet panels. It is not meant for a faceted figure.

A density and a histogram on the same side share one strip, the histogram drawn first and scaled to a density (count / (n * bin width)) so both share a scale; a histogram on its own shows counts. A rug sits in a strip of its own, between the main panel and any density or histogram. Densities, like ggplot2's, span each group's range of values. Every marginal is a line trace, so box and lasso selection pick only the plot's points, and each takes its group's `legendgroup`, so hiding a group from the legend hides its marginals too.

Value

The built plotly figure with the marginals added, or `fig` unchanged when `top` and `right` are both empty.

Author(s)

Jared Andrews

See Also

`stats::density()`, `axis_titles_as_annotations()`

Examples

```
library(plotly)
df <- example_iris
fig <- plot_ly(df,
    x = ~Sepal.Length, y = ~Sepal.Width, color = ~Species,
    type = "scatter", mode = "markers"
)
add_marginal_plots(fig,
    x = df$Sepal.Length, y = df$Sepal.Width, groups = df$Species,
    colors = c("#E69F00", "#56B4E9", "#009E73"),
    top = c("density", "rug"), right = "histogram"
)
```

---

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

Description

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

Usage

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

```
  abline.intercepts = NULL,
  abline.colors = NULL,
  abline.widths = NULL,
  abline.linetypes = NULL,
  abline.opacities = NULL
)
```

Arguments

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

Value

The modified plotly figure with all specified lines added.

Author(s)

Jared Andrews

Examples

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

---

add_size_legend *Add a point-size legend to a plotly figure*

Description

plotly drops the size legend when marker size encodes a numeric column (plotly.R#705), whether the figure comes from `plotly::ggplotly()` or is built by hand. This draws one in its place: a vertical column of circles with numeric labels, outside the right edge of the plot area in paper-referenced coordinates so it does not overlap the data. The breaks run evenly across the size scale's `limits`, and each circle is drawn at the size the scale gives its value, so the legend follows the plot's size scaling. The DotPlot and scatter plot modules use it, and a module in another package can too.

Usage

```
add_size_legend(
  fig,
  data,
  size.by,
  gap = 0.05,
  size.values = NULL,
  title = size.by,
  digits = NULL,
  title.size = NULL,
  text.size = NULL,
  start.y = 0.95,
  start.x = 1.02,
  font.family = NULL,
  font.color = NULL,
  limits = NULL,
  size.range = NULL,
  breaks = NULL
)
```

Arguments

`fig`	A plotly figure object.
`data`	A data frame containing the variable mapped to point size.
`size.by`	Character string, or `NULL`. Name of the column in `data` whose range determines the legend break labels. When `NULL` or empty (no size mapping is in effect), the figure is returned unchanged.

gap — Numeric. Vertical spacing (in paper units, 0–1) between consecutive legend entries. Defaults to 0.05.

size.values — Numeric vector of font sizes (px) used to render the circle glyphs, one per legend entry. When NULL (the default), the glyph sizes follow the plot's size scaling: from size.range when it is given, otherwise from the marker sizes in fig. The marker pixel diameters are converted to glyph font-sizes via .CIRCLE_GLYPH_DIAMETER_RATIO so the rendered circles match the plotted dots. When supplied, the vector is used verbatim as font sizes and its length determines the number of legend entries.

title — Character. Legend title. Defaults to size.by.

digits — Integer, or NULL. Decimal places the break labels are rounded to. When NULL (the default), the break values are printed as they are.

title.size — Numeric, or NULL. Font size (px) of the legend title annotation. When NULL, plotly's default is used.

text.size — Numeric, or NULL. Font size (px) of the numeric label annotations. Defaults to 12 when NULL.

start.y — Numeric. Paper-space y coordinate (0–1) at which the legend column begins; the title sits just above it and subsequent entries stack downward. Lower it to vertically offset the size legend from an overlapping color/shape legend. Invalid values fall back to the default. Defaults to 0.95.

start.x — Numeric. Paper-space x coordinate at which the legend column (circles, labels and title) is anchored. Values just above 1 place the legend to the right of the plot area; nudge it lower to pull the whole set inward when it would otherwise overflow a narrow plot, or higher to push it further out. Defaults to 1.02.

font.family — Character, or NULL. Font family of the title and label annotations. When NULL, plotly's default is used.

font.color — Character, or NULL. Font color of the title and label annotations. Defaults to "#000000" when NULL. The circles stay black.

limits — Numeric length-2 vector, or NULL. The size.by values drawn at the smallest and largest sizes, as given to the plot's size scale. The breaks run evenly between them. A NA end takes the data's minimum or maximum, and limits that are not in increasing order fall back to the data's range. Defaults to NULL, the data's range.

size.range — Numeric length-2 vector, or NULL. The size range the plot's size scale maps limits onto, in ggplot2 size units (the range of ggplot2::scale_size()). When given, each circle is drawn at the size the scale gives its break. When NULL (the default), the circles are read from the marker sizes in fig.

breaks — Numeric vector, or NULL. The size.by values to draw a circle for, in place of five spaced evenly between the limits. Values outside the limits are dropped. Useful for round labels, or to leave out the lower limit of a scale whose size.range starts at 0, which would draw an empty circle. When size.values is also given, it is recycled to one glyph size per break. Defaults to NULL.

Details

For a ggplot drawn with scale_size(range = r, limits = l), pass size.range = r and limits = l. Without size.range, the circles are read from the figure's own markers instead, taking its smallest and largest to be the data's minimum and maximum.

Call it after the figure's traces are complete. It builds the figure and appends its annotations to the built layout, so later `plotly::plotly_build()` calls (e.g. `apply_legend_inputs()` or `axis_titles_as_annotations()`) do not duplicate them. A module hiding its legend should pass `size.by = NULL`.

Value

The built plotly figure with size-legend annotations appended, or the unmodified figure when `size.by` is `NULL`/empty, not present in `data`, not numeric, or has no finite values.

Author(s)

Jacob Martin, Jared Andrews

Examples

```
library(plotly)
df <- data.frame(x = 1:5, y = c(2, 4, 3, 5, 1), n = c(10, 40, 25, 80, 55))
fig <- plot_ly(df, x = ~x, y = ~y, type = "scatter", mode = "markers",
    marker = list(size = sqrt(df$n) * 3))
add_size_legend(fig, df, size.by = "n", title = "Count", digits = 0)

# A ggplot with a fixed size scale: the legend runs 0, 25, ..., 100.
library(ggplot2)
p <- ggplot(df, aes(x, y, size = n)) +
    geom_point() +
    scale_size(range = c(2, 8), limits = c(0, 100))
add_size_legend(ggplotly(p), df, size.by = "n", limits = c(0, 100), size.range = c(2, 8))
```

---

add_vlines *Build vertical line shapes for a plotly figure*

Description

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

Usage

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

Arguments

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

Details

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

Value

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

Author(s)

Jared Andrews

Examples

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

---

adjusted_axis_label *Build an adjustment-aware axis label*

Description

Wraps a base column name with the names of any data adjustments that are applied to it before plotting, so that an axis title accurately describes the values displayed. The wrapping order mirrors how the 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

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

Details

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

Value

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

Author(s)

Jared Andrews

Examples

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

---

adjusted_values *Transform values the way the modules plot them*

Description

The adjustment function (adj.fxn, e.g. log10) is applied first, then the adjustment rescales the result ("z-score" or "relative.to.max"). That is the order a transform and a standardisation are normally combined in: log first, then z-score. dittoViz applies them the other way round when given both, which turns every below-average value into log() of a negative number, so the modules hand dittoViz this whole transform as its function instead (see adjustment_fn()).

Usage

```
adjusted_values(values, adjustment = NULL, adj.fxn = NULL)
```

Arguments

values	Vector of values, normally one column of the plotted data.
adjustment	NULL, "", "z-score" or "relative.to.max".
adj.fxn	NULL, "", a name accepted by safe_resolve_adj_fxn(), or a function.

Details

The rescaling is computed over the finite values only, so a value the function makes non-finite (`log(0)`) is dropped from the plot rather than turning the whole column into `NaN`.

Anything drawn over an adjusted plot (significance brackets, fit lines, axis limits) has to be computed from these values, not the raw column, or it lands in a different coordinate space from the data.

Value

The transformed values, unnamed.

Author(s)

Jared Andrews

See Also

`adjustment_fn()`, `as_plotted()`, `adjust_column_values()`

Examples

```
adjusted_values(c(1, 10, 100), adjustment = "z-score", adj.fxn = "log10")
adjusted_values(c(1, 2, 4), adjustment = "relative.to.max")
```

`adjustment_fn` *The whole plotted transform as one function, for dittoViz*

Description

dittoViz applies its `*.adjustment` before its `*.adj.fxn`. Passing it this function as `*.adj.fxn`, with no `*.adjustment`, has it plot `adjusted_values()` instead, while it still builds its adjusted columns, hover text and multi-variable reshape as usual.

Usage

```
adjustment_fn(adjustment = NULL, adj.fxn = NULL)
```

Arguments

`adjustment`, `adj.fxn`
: Passed to `adjusted_values()`.

Value

A function of one vector, or `NULL` when neither is set.

Author(s)

Jared Andrews

See Also

adjusted_values(), as_plotted()

Examples

```
fn <- adjustment_fn(adjustment = "z-score", adj.fxn = "log10")
fn(c(1, 10, 100))
adjustment_fn()
```

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

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 `safe_resolve_adj_fxn` (e.g., "log2", "log10", "sqrt"). If `NULL` or an empty string, x-axis values are left unchanged.

`y.adj.fun` Character scalar. Name of a transformation function to apply to y-axis values, as accepted by `safe_resolve_adj_fxn`. If `NULL` or an empty string, y-axis values are left unchanged.

`color.adj.fun` Character scalar. Name of a transformation function to apply to color values, as accepted by `safe_resolve_adj_fxn`. If `NULL` or an empty string, color values are left unchanged.

`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_padding *Set the padding between a figure's data and the edges of its plot area*

Description

Sets each x and y axis range to the extent of the data drawn on it, widened at each end by a fraction of that extent: `0` makes the data fill the plot area, `0.05` matches ggplot2's default expansion. The extent is read from the figure's traces as drawn: points and lines, bars from their base, error bars, and fills down to zero. A categorical axis (as `plotly::ggplotly()` draws one) spans its categories' bands, each one unit wide, so `0` puts the outer bands' edges on the plot area's.

Usage

```
apply_axis_padding(fig, x = NULL, y = NULL, swap = FALSE)
```

Arguments

`fig`	A plotly figure.
`x, y`	Padding for the x and y axes, as a fraction of the data's extent added at each end. `NULL` or `NA` leaves that axis as it is.
`swap`	Logical. Apply `x` to the figure's y axes and `y` to its x axes, for a plot drawn with its axes swapped (a rotated or flipped plot), so the padding follows the variable rather than the screen direction.

Details

Axes that share a scale (matched axes, or the identical ranges of a faceted plot's fixed scales) get the same range, so they still line up. Reversed axes stay reversed, and log axes are padded in log units. Axes holding dates or plotly categories, or with no numeric data on them, are left alone, as are axes marked with `mark_overlay_axes()`.

Only what the traces draw counts. Annotations and shapes, such as significance brackets or reference lines, do not widen the range and can be cut off, so use axis limits instead on an axis that carries them.

Value

The built plotly figure with the axis ranges set.

Author(s)

Jared Andrews

See Also

`apply_uniform_styling()`, which applies this from the Axes tab's "X Axis Padding" and "Y Axis Padding" inputs.

Examples

```
fig <- plotly::plot_ly(mtcars,
    x = ~wt, y = ~mpg, type = "scatter", mode = "markers"
)
# Points reaching the edges of the plot area.
apply_axis_padding(fig, x = 0, y = 0)$x$layout$xaxis$range

# A ggplot area plot with its default expansion removed along x only.
df <- data.frame(year = 2000:2020, n = cumsum(c(5, rep(1, 20))))
p <- ggplot2::ggplot(df, ggplot2::aes(year, n)) +
    ggplot2::geom_area()
apply_axis_padding(plotly::ggplotly(p), x = 0)
```

---

apply_axis_title_to_annotations

Apply axis title font styling to shared facet axis annotations

---

Description

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

Usage

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

Arguments

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

Value

The modified plotly figure with updated annotation fonts.

Author(s)

Jacob Martin

Examples

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

## End(Not run)
```

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

Description

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

Usage

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

Arguments

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

Details

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

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

Value

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

Author(s)

Jacob Martin

Examples

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

`apply_highlight_styling` *Restyle highlighted points in a plotly figure*

Description

Applies highlight marker styling (fill, size, border) to the points of a plotly figure whose `annotate.by` value appears in `highlight_vals`. Points are matched on their annotation value parsed out of the hover text rather than on coordinates, because `ggplotly()` encodes categorical axes as numeric positions that will not match the raw data values.

Usage

```
apply_highlight_styling(
  fig,
  annotate.by,
  highlight_vals,
  style,
  default.size = NULL,
  show.others = TRUE,
  require.markers = FALSE
)
```

Arguments

fig	Plotly figure object.
annotate.by	Character. Name of the hover field holding the values to match.
highlight_vals	Character vector. Values identifying the points to highlight.
style	A named list of styling values with elements color, size, border.color and border.width. Empty/NA elements are left unchanged.
default.size	Numeric, or NULL. Marker size to fall back on for traces that carry no explicit size.
show.others	Logical. Whether "show.others" was enabled in the plot.
require.markers	Logical. When TRUE, only traces drawing point markers are restyled (use it when other scatter traces, such as box outlines, come from the same data).

Value

The plotly figure with highlight styling applied.

Author(s)

Jared Andrews

See Also

uniform_annotation_inputs_ui(), parse_highlight_values(), create_highlight_annotations()

Examples

```
## Not run:
fig <- apply_highlight_styling(
    fig, annotate.by = "name", highlight_vals = c("Mazda RX4", "Fiat 128"),
   style = list(color = "#FF0000", size = 14, border.color = "#000000", border.width = 2)
)

## End(Not run)
```

---

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, position = NULL)
```

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.
position	Passed to apply_legend_styling(); NULL keeps the legend where it is.

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

fig	A plotly figure object.
title.size	Numeric font size for the legend (or colorbar) title, and for the titles of a legend's sections (legendgrouptitle, such as the colour and shape sections of a scatter plot mapping both), or NULL to leave unchanged.
text.size	Numeric font size for the legend entry labels (or colorbar tick labels), or NULL to leave unchanged.
position	Optional length-2 vector c(x, xanchor) placing the legend horizontally, e.g. c(1.02, "left"). NULL (the default) leaves the position unchanged.
font.family	Character font family for the legend title, section titles and entry labels (and colorbar title and ticks), or NULL to leave unchanged.
font.color	Character color for the legend title, section titles and entry labels (and colorbar title and ticks), or NULL to leave unchanged.
show	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_level_order *Reorder the levels of a categorical column*

Description

Re-levels `data[[column]]` as a factor whose levels follow `order`, which is how the plot modules set the order of a categorical axis: the plot functions, the statistics and their brackets all take the axis order from the column's levels, so they stay in step.

Usage

```
apply_level_order(data, column, order)
```

Arguments

`data`	A data frame.
`column`	The name of the column.
`order`	Character vector of levels in the order wanted, e.g. the value of a `level_order_input()`. Merged as in `level_order_choices()`, with no cap on the number of levels.

Value

`data`, with `column` re-levelled. It is returned unchanged when `order` is empty or already the default order, or `column` is missing or numeric.

Author(s)

Jared Andrews

See Also

`level_order_input()`, `level_order_choices()`, `setup_level_order()`

Examples

```
df <- apply_level_order(example_iris, "Species", c("virginica", "setosa"))
levels(df$Species)
```

apply_plotly_newshape *Apply Plotly newshape styling from uniform Plotly inputs*

Description

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

Usage

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

Arguments

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

Value

The modified plotly figure with updated newshape layout settings.

Author(s)

Jared Andrews

Examples

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

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

Description

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

Usage

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

Arguments

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

Value

The plotly figure with margins applied.

Author(s)

Jacob Martin

Examples

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

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

Description

Appends the shapes and annotations from `create_stat_annotations()` to an existing plotly figure's layout, 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

fig	A plotly figure object.
stat_result	List with annotations, shapes, and y.max as returned by create_stat_annotations().
y.min	Numeric or NULL; minimum y-axis value. If NULL (or not finite), the existing y-axis bottom is kept, falling back to the data minimum.
y.max	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

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

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.

Axes recorded as belonging to overlay panels drawn beside the plot, such as the marginal distributions `add_marginal_plots()` adds, are left alone.

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_tick_notation` *Set the notation of a figure's numeric tick labels*

Description

Tick labels of large or small numbers can come out in scientific notation (`1e+05`) or not depending only on which breaks an axis happens to get, since a ggplot converted with `plotly::ggplotly()` carries the labels R's `format()` gave its breaks as fixed tick text. This sets the notation explicitly instead, for every x and y axis of the figure.

Usage

```
apply_tick_notation(fig, scientific = FALSE)
```

Arguments

`fig`	A plotly figure.
`scientific`	Logical. Whether numeric tick labels use scientific notation. Default `FALSE`.

Details

Fixed tick text is rewritten only on axes whose labels are all numbers that match the tick positions (or are 10, 2, or e raised to them, for a log scale), so categorical axes, even ones whose categories look like numbers, and formatted labels (`"25%"`, `"$1,000"`) are left alone. With `scientific = FALSE`, only labels already in scientific notation are rewritten; with `TRUE`, every numeric label is written as a power of ten (1.5 times 10 to the 5th). Axes plotly labels itself get `exponentformat = "power"` when `scientific` is `TRUE`, and are otherwise left to plotly, whose default abbreviates large numbers (`100k`) rather than using scientific notation.

Axes marked with `mark_overlay_axes()` are left alone.

Value

The built plotly figure with its tick labels updated.

Author(s)

Jared Andrews

See Also

`apply_uniform_styling()`, which applies this from the Axes tab's "Scientific Notation Ticks" input.

Examples

```
p <- ggplot2::ggplot(data.frame(x = c("a", "b"), y = c(1e5, 3e5)), ggplot2::aes(x, y)) +
    ggplot2::geom_col()
fig <- plotly::ggplotly(p)
apply_tick_notation(fig)$x$layout$yaxis$ticktext
apply_tick_notation(fig, scientific = TRUE)$x$layout$yaxis$ticktext
```

---

apply_title_layout *Apply plot title styling to a plotly figure*

Description

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

Usage

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

Arguments

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

Value

The modified plotly figure with updated title styling.

Author(s)

Jacob Martin

Examples

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

## End(Not run)
```

`apply_uniform_styling` *Apply the uniform styling inputs to a module's figure*

Description

The plot modules build their figures in two steps. The first draws the plot itself from the data and every input that changes what is drawn; that is the expensive part (a ggplot built and converted with `plotly::ggplotly()`, say), and it is cached. This function is the second step: it applies the inputs that only style the figure – titles, fonts, ticks and their notation, gridlines, the padding around the data, reference lines, the modebar, the shape-drawing tool, and the legend – to the figure the first step returned. Changing one of those inputs re-runs only this step, which takes a few milliseconds rather than a full rebuild.

Usage

```
apply_uniform_styling(
  fig,
  input,
  isolate_fn = isolate,
  facet.by = NULL,
  title_y = 0.95,
  include.modebar.buttons = TRUE,
  fig.fn = NULL,
  ggplot.axis.styling = TRUE,
  legend.position = NULL,
  padding.axes = c("x", "y")
)
```

Arguments

`fig`	A plotly figure: the module's plot, before styling.
`input`	The module's Shiny `input`, holding the uniform inputs from `uniform_axes_inputs_ui()`, `uniform_lines_inputs_ui()`, `uniform_plotly_inputs_ui()`, and `uniform_legend_inputs_ui()`.
`isolate_fn`	The function from `setup_auto_update_logic()` that every input read goes through.
`facet.by`	The facetting column(s), or `NULL`. A faceted figure has shared axis titles drawn as annotations, which are styled to match, and no main title (see `add_plot_config()`).
`title_y`	Vertical position of the main title, in paper coordinates.
`include.modebar.buttons`	Passed to `add_plot_config()`.
`fig.fn`	Optional `function(fig, input, isolate_fn)` returning the figure, run after the legend is styled and before the axis titles become draggable annotations, so any axis title text it sets is the one made draggable. Read inputs in it as `isolate_fn(input$key)`. It runs after any overlay panels are drawn, so anything it places on `"paper"` coordinates can land on them; use `xref = "x domain"` and `yref = "y domain"` to stay within the plot's panel.

`ggplot.axis.styling`
: Passed to `create_axis_styles()`. `TRUE` for a figure whose axis lines come from its ggplot theme (see `create_ggplot_axis_style()`); `FALSE` sets them from the Axes inputs here.

`legend.position`
: Passed to `apply_legend_inputs()`; `NULL` keeps the legend where it is.

`padding.axes`
: Which of the module's axes, `"x"` and `"y"`, take the `axis.padding.x`/`axis.padding.y` inputs (see `apply_axis_padding()`). Leave out an axis with its own limit inputs, so its limits stand. When `input$rotate` is `TRUE` the padding follows the variables onto the swapped axes.

Details

The figure passed in is never modified (R copies on write), so each call styles the cached figure afresh and nothing appended here (reference lines, say) accumulates across calls.

Axes used only by overlay panels drawn alongside the plot (the scatter plot's marginal distributions, see `add_marginal_plots()`) keep their own styling and get no reference lines.

Value

The styled plotly figure.

Author(s)

Jared Andrews

See Also

`apply_title_layout()`, `apply_subplot_axis_styling()`, `apply_tick_notation()`, `apply_axis_padding()`, `add_reference_lines()`, `add_plot_config()`, `apply_plotly_newshape()`, `apply_legend_inputs()`, `axis_titles_as_annotations()`

Examples

```
fig <- plotly::plot_ly(mtcars, x = ~wt, y = ~mpg, type = "scatter", mode = "markers")
# The uniform inputs, as a module's `input` would hold them.
input <- list(
    title.font.size = 20, title.font.family = "Arial", title.font.color = "#000000",
    axis.title.horizontal.position = 0.5, axis.title.font.size = 16,
    axis.title.font.family = "Arial", axis.title.font.color = "#000000",
   axis.tickfont.size = 12, axis.tickfont.family = "Arial", axis.tickfont.color = "#000000",
    axis.tickangle.x = 0, axis.tickangle.y = 0, axis.ticks = "outside",
    axis.tickcolor = "#000000", axis.ticklen = 5, axis.tickwidth = 1,
    show.grid.x = TRUE, show.grid.y = FALSE, grid.color = "#CCCCCC",
    hline.intercepts = "20", hline.colors = "#D55E00",
    download.format = "svg", legend.text.size = 12, legend.title.size = 14
)
apply_uniform_styling(fig, input, isolate_fn = identity)
```

`as_plotted` *Replace data columns with the values the modules plot for them*

Description

Applies `adjusted_values()` to each of `cols` in place, keeping the column names. Each column is transformed on its own over the whole frame, as dittoViz does before any row subsetting or multi-variable reshape, so the result can be passed wherever the raw frame was (statistics, model formulas that name the raw columns, axis-range calculations).

Usage

```
as_plotted(df, cols, adjustment = NULL, adj.fxn = NULL)
```

Arguments

`df`	Data frame.
`cols`	Character vector of column names to transform. Names not in `df` are ignored.
`adjustment`, `adj.fxn`	Passed to `adjusted_values()`.

Value

`df` with `cols` transformed; unchanged when neither argument is set.

Author(s)

Jared Andrews

See Also

`adjusted_values()`, `adjust_column_values()`

Examples

```
as_plotted(data.frame(y = c(1, 10, 100)), "y", adjustment = "z-score", adj.fxn = "log10")
```

auto_webgl *Draw a figure's point layers with WebGL*

Description

SVG slows down badly with many points, where WebGL does not. This switches the figure's point traces (`scatter` traces drawn as markers only: jitter, the points of a scatter plot) to `scattergl` once, together, they hold at least `threshold` points. Lines, filled areas, boxes, and violins stay SVG, so they stay vector in SVG downloads; only the switched points become an embedded image there. Unlike `plotly::toWebGL()`, which switches every trace.

Usage

```
auto_webgl(fig, threshold = .webgl_threshold(), overlays = FALSE)
```

Arguments

`fig`	A plotly figure, built or as `plotly::ggplotly()` returns it.
`threshold`	Point count at and above which the points switch.
`overlays`	Logical; also switch unfilled line traces, for lines that must stay on top of the points.

Details

WebGL canvases sit above SVG traces, so a line drawn over the points (an ellipse, a contour) would end up beneath them. Set `overlays = TRUE` to switch a figure's unfilled line traces with its points, which keeps their order. The points alone decide whether the threshold is reached, and filled traces stay SVG either way.

Pair it with `webgl_input()`. The default threshold is 5,000 points, or `options(VizModules.webgl.threshold = n)`.

Value

The figure.

Author(s)

Jared Andrews

See Also

`webgl_input()`

Examples

```
fig <- plotly::plot_ly(x = rnorm(100), y = rnorm(100), type = "scatter", mode = "markers")
fig <- plotly::plotly_build(fig)
auto_webgl(fig, threshold = 50)$x$data[[1]]$type
```

`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)
```

`blank_to_null` *Normalize a module's "no selection" column input to* `NULL`

Description

Module selects use `""` for "none". Statistics helpers expect `NULL`, and a grouping column that is numeric is a gradient rather than a nesting, which the renders already treat as no grouping.

Usage

```
blank_to_null(value, df = NULL, numeric_is_null = FALSE)
```

Arguments

`value`	The input value.
`df`	Optional data frame, needed only for `numeric_is_null`.
`numeric_is_null`	Logical; when `TRUE`, a numeric column in `df` also yields `NULL`.

Details

Anything that is not a single, non-`NA`, non-empty value gives `NULL`: an input that has not reported yet (`NULL`), `NA`, `""` and a vector of more than one value alike. A multi-select therefore reads as "no selection", so handle those inputs explicitly.

Value

`value`, or `NULL`.

Author(s)

Jared Andrews

See Also

`nz_value()`, `na_to_null()`

Examples

```
blank_to_null("")
blank_to_null(NULL)
blank_to_null("Species")
blank_to_null("Sepal.Length", df = iris, numeric_is_null = TRUE)
```

`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

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

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

linecolor	Character, colour of the border lines. Default "black".
linewidth	Numeric, width of the border lines in pixels. Default 0.5.
ncol	Optional integer. Number of facet columns. If NULL or NA, detected from the distinct x-axis domain starts.
nrow	Optional integer. Number of facet rows. If NULL or NA, detected from the distinct y-axis domain starts.

Details

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

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

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

Value

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

Author(s)

Jacob Martin

Examples

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

---

build_model_row_spec *Build a dynamic row_spec from registered backends*

Description

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

Usage

```
build_model_row_spec()
```

Value

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

Author(s)

Jacob Martin

Examples

```
build_model_row_spec()
```

`cached_structure` *Cache a module's plot step*

Description

Draws a module's plot (everything before `apply_uniform_styling()`) and keeps it in the session's cache, keyed on the data, the module's other reactive values the step reads (`extra`), and the inputs the step reads, so a styling change never redraws the plot and going back to settings already drawn (toggling an option back, Reset) re-uses the figure. The figure is built here (`plotly::plotly_build()`), so the styling step does not repeat that work.

Usage

```
cached_structure(build, input, session, data, params = NULL, extra = NULL)
```

Arguments

`build`	A function of `isolate_fn` that draws the plot and returns a list with at least `fig`. See the rules above.
`input, session`	The module's `input` and `session`.
`data`	The module's `data` reactive.
`params`	The reactive-defaults store from `setup_reactive_defaults()`, or `NULL`.
`extra`	A function of no arguments returning the module's other reactive values `build` reads (axis-limit and colour stores, the selection, ...), or `NULL`.

Details

The inputs in the key are exactly those the step reads through the `isolate_fn` it is given: every `isolate_fn(input$<key>)` read is recorded, and the set only grows. So an input the step reads is always in the key (a stale plot otherwise), while inputs it never reads (the styling ones, widget bookkeeping, an input frozen by an update the step does not care about) are never in it. An entry stored before the set grew has a key of a different shape from every later one, so it can never be served for settings it was not drawn with.

On a miss the step is evaluated reactively, so it depends on what it reads in the usual way; on a hit the key's reads stand in for it. Errors are not cached. The key includes the module's namespace, so modules sharing a session never see each other's entries. Set `options(VizModules.cache = FALSE)` to turn caching off.

Value

A reactive returning `build`'s value, with `.build` added: a number unique to each time the plot was drawn, kept with the value in the cache. The figure's `layout.uirevision` is set from it, so a styling change keeps the user's zoom while a different plot resets it.

Three rules keep the cache honest

A step that breaks one of these does not fail: it draws a stale plot.

- Read every input in the literal form `isolate_fn(input$key)`. A wrapped read such as `isolate_fn(as.numeric(input$size))` cannot be recognised, and a bare `input$size` bypasses `isolate_fn` altogether, so that input never reaches the key. Convert outside the call.
- Name every other reactive the step reads in `extra`: colour and axis-limit stores, a debounced text input, a reactive argument. The exception is a reactive that only derives from `data`, which the key already covers.
- Give it no side effects (notifications, `reactiveVal()` writes), since a cache hit skips them. Return anything else the module needs in the list instead.

Author(s)

Jared Andrews

See Also

`apply_uniform_styling()` for the styling step, `finalize_manual_edits()` for sending a restyle as a layout-only update.

Examples

```
## Not run:
# Inside moduleServer(), with `params` from setup_reactive_defaults():
draw_myPlot <- function(isolate_fn) {
    p <- my_plot_fn(data(), x = isolate_fn(input$x.by), colors = palette_store())
    list(fig = plotly::ggplotly(p))
}
base_myPlot <- cached_structure(draw_myPlot, input, session, data,
```

```
    params = params, extra = function() list(palette_store())
)
generate_myPlot <- reactive({
    isolate_fn <- setup_auto_update_logic(input, params)
    apply_uniform_styling(base_myPlot()$fig, input, isolate_fn)
})

## End(Not run)
```

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

Description

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

Usage

```
clean_facet_dim(val)
```

Arguments

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

Details

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

Value

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

Author(s)

Jacob Martin

Examples

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

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

Description

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

Usage

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

Arguments

`plot_reactive`	A reactive expression returning a `plotly` plot object.
`stats_reactive`	Optional. A reactive expression (e.g. a `shiny::reactiveVal()`) returning a `data.frame` of statistical test results, or a named list of them for a plot with more than one table (a per-group summary and pairwise comparisons, say). 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, a named list of them (written to the archive as one CSV each), 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)
```

`ComplexHeatmap_HeatmapApp`

Create an example Modular ComplexHeatmap Shiny Application

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

data_list	An optional named list of data frames. If NULL (the default), example_heatmap_matrix is used as example data — paired with example_heatmap_column_data unless column_data says otherwise. When column_data is supplied it is attached to the first entry, which becomes list(matrix = , column_annotations =).
column_data	An optional data frame of per-sample metadata, enabling column annotations, column splitting, and metadata-aware column filtering. Defaults to example_heatmap_column_data when data_list is also NULL; pass data_list explicitly to opt out. Attached to the first data_list entry (see ComplexHeatmap_HeatmapServer()'s data 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 shinyApp() (no Data Import/Data Table sections) wiring data = list(matrix = <first element of data_list, or example_heatmap_matrix>, column_an directly into the module.
defaults	A named list of input IDs and their default values to apply on startup. An entry may also be a shiny::reactive() or shiny::reactiveVal() to have the input follow the parent app's state; see setup_reactive_defaults().
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 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_HeatmapServe` `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)
if (requireNamespace("ComplexHeatmap", quietly = TRUE) &&
    requireNamespace("InteractiveComplexHeatmap", quietly = TRUE) &&
    requireNamespace("circlize", quietly = TRUE)) {
    # 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_HeatmapMainOutpu` for how the separated output pieces fit together.

Usage

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

Arguments

id	The ID for the Shiny module. Must match the id used for ComplexHeatmap_HeatmapServer() and any other output pieces for the same heatmap.
title	Optional panel title. NULL (the default) omits the title.
width	Panel width in pixels.
...	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

id	The ID for the Shiny module.
data	The data frame used for plot generation, *or* a list with a matrix data frame (required) and a column_annotations data frame (optional) — see ComplexHeatmap_HeatmapServer()'s data parameter for details. Row-annotation choices come from matrix's columns; column-annotation choices (and the "Column Annotations" tab controls) only appear when column_annotations is supplied.
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

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_annotations

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_annotations` 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_HeatmapS`
`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()`'s 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

id	The ID for the Shiny module. Must match the id used for ComplexHeatmap_HeatmapServer() and any other output pieces for the same heatmap.
title	Optional panel title. NULL (the default) omits the title.
width, height	Panel dimensions in pixels.
fit.width	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().
...	Additional arguments passed to InteractiveComplexHeatmap::originalHeatmapOutput(), e.g. action, response, brush_opt.

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

`id`	The ID for the Shiny module.
`resizable`	Logical; accepted for signature parity with the other module output functions but ignored, since the InteractiveComplexHeatmap widget manages its own sizing.
`fit.width`	Logical; when `TRUE` (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**. `width1`/`width2` (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 `FALSE` for fixed pixel widths.
`...`	Additional arguments passed to `InteractiveComplexHeatmap::InteractiveComplexHeatmapOutput(`, e.g. `layout`, `compact`, `width1`/`height1`, `title1`/`title2`/`title3`.

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

id	The ID for the Shiny module.
data	A reactive yielding either a data frame (the matrix's columns plus any row-annotation columns, all together — the original, single-table behavior) or list(matrix = <data.frame>, to additionally enable column annotations, where column_annotations is a per-sample metadata table keyed by a column matching the matrix's selected column names (see the "Column Key" input). A NULL value (or a list missing matrix) 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 used, 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 used, but not shown 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 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().

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

`id`	The ID for the Shiny module. Must match the `id` used for `ComplexHeatmap_HeatmapServer()`.
`resizable`	Logical; whether to wrap the plot in a resizable container. Unlike `ComplexHeatmap_HeatmapOutputUI()`, this is honoured, since a `plotOutput` has no resize handle of its own.
`width, height`	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

`id`	The ID for the Shiny module. Must match the `id` used for `ComplexHeatmap_HeatmapServer()` and any other output pieces for the same heatmap.
`title`	Optional panel title. `NULL` (the default) omits the title.
`width, height`	Panel dimensions in pixels.

`fit.width`	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()`.
`...`	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_Heatma`
`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

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

Value

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

Author(s)

Jared Andrews, Jacob Martin

Examples

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

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

---

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

Description

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

Usage

```
createModuleApp(
  inputs_ui_fn,
  output_ui_fn,
  server_fn,
  data_list,
  defaults = NULL,
  hide.inputs = NULL,
  hide.tabs = NULL,
  show.table = TRUE,
  title = "VizModules App",
  primary.table = NULL,
  sidebar.width = 4
)
```

Arguments

inputs_ui_fn	A function with signature `function(id, data, ...)` that returns module input UI elements (e.g. `plotthis_BarPlotInputsUI()`).

output_ui_fn	A function with signature function(id) that returns the module's output UI (e.g. plotthis_BarPlotOutputUI()).
server_fn	A function with signature function(id, data, ...) that drives the module server logic (e.g. plotthis_BarPlotServer()).
data_list	A named list of 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. list(matrix = , column_annotations =) for ComplexHeatmap_Heatmap. Only the primary table (see primary.table) is filtered and shown in the Data Table; the rest are passed through to the module untouched.
defaults	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 shiny::reactive() or shiny::reactiveVal() to have the input follow app state; see setup_reactive_defaults().
hide.inputs	A character vector of input IDs to hide. These inputs are still initialized and their values passed to the plot, but are not shown in the UI. Passed through to server_fn when it accepts a hide.inputs argument.
hide.tabs	A character vector of tab names to hide. Inputs in these tabs are still initialized and their values passed to the plot, but are not shown in the UI. Passed through to server_fn when it accepts a hide.tabs argument.
show.table	Logical. When TRUE (default), a filterable DT table is shown below the plot and its row selection drives the data passed to the plot module. When FALSE, the table and filter controls are hidden and the full (unfiltered) dataset is passed directly to the plot module.
title	A character string used as the page title (default: "VizModules App").
primary.table	For a data_list 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.
sidebar.width	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. sidebar.width = 3 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 `title.font.family`, `text.colour`, `axis.showline`, `axis.mirror`, `axis.linecolor`, `axis.linewidth`, `axis.tickfont.size`, `axis.tickfont.color`, `axis.tickfont.family`, `axis.tickangle.x`, `axis.tickangle.y`, `axis.ticks`, `axis.tickcolor`, `axis.ticklen`, `axis.tickwidth`, `show.grid.x`, `show.grid.y`, and `grid.color`.

`axis_side` Character. Which axis to style, either `"x"` or `"y"`. Determines whether `axis.tickangle.x` or `axis.tickangle.y` is used for the tick angle, and which gridline inputs are applied.

`isolate_fn` Function. A function used to isolate Shiny inputs, typically `shiny::isolate`. Defaults to `isolate`.

`ggplot.axis.styling` Logical. Whether ggplot axis styling is applied. Defaults to `TRUE`.

Details

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

Value

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

Author(s)

Jacob Martin

Examples

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

---

create_ggplot_axis_style

Create ggplot axis styling theme arguments

Description

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

Usage

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

Arguments

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

Details

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

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

Value

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

Author(s)

Jacob Martin

Examples

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

---

create_highlight_annotations

Create annotations for highlighted points

Description

Creates plotly annotation objects for points highlighted by value matching. This function finds all points in the plot data that match the highlight values and creates annotations for them.

Usage

```
create_highlight_annotations(
  plot_data,
  fig,
  annotate.by,
  highlight_vals,
  x_col,
  y_col,
  annotation_params,
  show.others = TRUE,
  require.markers = FALSE
)
```

Arguments

plot_data	Data frame. The plot data from dittoViz scatterPlot (Target_data).
fig	Plotly figure object.
annotate.by	Character. Name of the field to use for annotation text.
highlight_vals	Character vector. Values to highlight from the annotate.by column.
x_col	Character. Name of the x-axis column.
y_col	Character. Name of the y-axis column.
annotation_params	List of annotation styling parameters.
show.others	Logical. Whether "show.others" was enabled in the plot.
require.markers	Logical. When TRUE, only traces drawing point markers are matched (use it when other scatter traces, such as box outlines, come from the same data).

Value

List of plotly annotation objects, or NULL if no valid annotations.

Author(s)

Jared Andrews

See Also

uniform_annotation_inputs_ui(), parse_highlight_values(), create_selected_annotations(), merge_annotation_sets()

Examples

```
## Not run:
# Inside a module's renderPlotly(), after building the plot:
annos <- create_highlight_annotations(
    plot_data,
```

```
    fig,
    annotate.by = "name",
    highlight_vals = parse_highlight_values(input$highlight.values),
    x_col = "wt",
    y_col = "mpg",
    annotation_params = params
)

## End(Not run)
```

`create_selected_annotations` *Create annotations for selected points*

Description

Creates plotly annotation objects for selected points from event_data("plotly_selected"). This function handles the complex logic of matching selected points to their corresponding traces and extracting annotation text.

Usage

```
create_selected_annotations(
  selected_data,
  fig,
  annotate.by,
  annotation_params,
  show.others = TRUE,
  require.markers = FALSE
)
```

Arguments

`selected_data`	Data frame from event_data("plotly_selected").
`fig`	Plotly figure object.
`annotate.by`	Character. Name of the field to use for annotation text.
`annotation_params`	List of annotation styling parameters (ax, ay, showarrow, etc.).
`show.others`	Logical. Whether "show.others" was enabled in the plot.
`require.markers`	Logical. When `TRUE`, only traces drawing point markers are matched (use it when other scatter traces, such as box outlines, come from the same data).

Value

List of plotly annotation objects, or NULL if no valid annotations.

Author(s)

Jared Andrews

See Also

uniform_annotation_inputs_ui(), create_highlight_annotations(), merge_annotation_sets()

Examples

```
## Not run:
# Inside a module's renderPlotly(), with the selection from event_data():
selected <- event_data("plotly_selected", source = plot_source)
annos <- create_selected_annotations(selected, fig, annotate.by = "name",
                                     annotation_params = params)

## End(Not run)
```

`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

`data_list`	A reactive returning either a single summary list produced by `collect_source_data()` (with elements `plot`, `plot_data`, `stats`, and `inputs`), or a named list of such summaries (one per plot). When a named list of summaries is supplied, each summary is written to its own set of files (prefixed with the list name) so several plots can be bundled into a single archive.
`filename_base`	`character(1)`. Base name for the downloaded `.zip` file without extension. The final filename takes the form `<filename_base>_<Sys.Date()>.zip`.
`images`	`logical(1)`. Whether to include `.svg` and `.png` images of each plot. Requires the button to carry the markup `module_tack_ui()` gives it; a hand-rolled `shiny::downloadButton()` without it simply gets no images.

output_id	character(1). 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 output$download.source.
session	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

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

`free.y` Logical; whether each facet panel has its own y scale (e.g. `scales = "free_y"`). 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 `FALSE`.

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)
```

`dataFilterServer` *Server logic for the dataFilter module*

Description

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

Usage

```
dataFilterServer(
  id,
  data,
  factor.char.cols = TRUE,
  page.length = 10,
  col.visibility = FALSE,
  hide.columns = NULL,
  filter.max.options = 50
)
```

Arguments

`id`	The ID for the Shiny module. Must match the `id` used in `dataFilterUI()`.
`data`	A `reactive` containing the data frame to display and filter. Values that are not data frames are coerced with `as.data.frame()`; a `NULL` value is treated as "not ready yet" and the table waits for data.
`factor.char.cols`	Logical. When `TRUE`, all character columns in `data` are converted to factors before the table is rendered. This causes DT to display select-box filters for those columns instead of free-text search boxes. 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 `TRUE`.
`page.length`	Integer. The default number of rows shown per page. Defaults to `10`.
`col.visibility`	Logical. When `TRUE`, adds a DT "Columns" button (Buttons extension `colvis`) so users can show/hide individual columns. Defaults to `FALSE`.
`hide.columns`	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 `col.visibility = TRUE` 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 `NULL` (show every column).

`filter.max.options`
: 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`, and rows keep their original order: sorting the table changes neither the rows returned nor their order, so it does not redraw a plot. The reactive only updates when the set of rows shown changes. 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_group_colors *Extract a caller-supplied group color mapping from* `defaults`

Description

Looks up `key` in a module's `defaults` and returns it as a named vector of hex colors suitable for `resolve_palette()` and `multiColorPicker()`. Values may be given as R color names (`"red"`) or hex codes; both are normalized to `#RRGGBB`. Anything that is not a fully named character vector is ignored, consistent with `get_default()`'s silent-fallback contract.

Usage

```
default_group_colors(defaults, key)
```

Arguments

defaults	A named list of default values, or NULL.
key	Character string — the color input's id, e.g. "palette.colours".

Value

A named character vector of hex colors, or NULL when defaults supplies no usable mapping.

Author(s)

Jared Andrews

See Also

resolve_palette(), get_default(), setup_group_colors(), reset_group_colors()

Examples

```
default_group_colors(list(palette.colours = c(A = "red", B = "#00FF00")), "palette.colours")
default_group_colors(list(palette.colours = c("red", "blue")), "palette.colours")
```

---

default_palettes *Color palette options for palettePicker*

Description

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

Usage

```
default_palettes()
```

Value

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

Author(s)

Jared Andrews

Examples

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

`default_stat_pairs` *Pick the comparisons a* `defaults` *list asks for from those on offer*

Description

The Comparisons selector is repopulated from the data whenever its grouping columns change, so a `stat.pairs` default cannot simply be written into the UI. The modules instead select, from each fresh set of choices, the ones `defaults$stat.pairs` names. A pair matches in either orientation, so `"Mid vs Entry"` selects the `"Entry vs Mid"` choice.

Usage

```
default_stat_pairs(defaults, pair_strings)
```

Arguments

`defaults`	A named list of module defaults, or `NULL`.
`pair_strings`	Character vector of `"A vs B"` choices on offer, as from `generate_pair_strings()`.

Value

The elements of `pair_strings` that `defaults$stat.pairs` names. When it names none, `""`, which is what the selector needs to show an empty selection (and means every pair is tested).

Author(s)

Jared Andrews

See Also

`generate_pair_strings()`, `parse_pair_strings()`, `reset_stats_inputs()`

Examples

```
choices <- c("Entry vs Mid", "Entry vs Late", "Mid vs Late")
default_stat_pairs(list(stat.pairs = "Mid vs Entry"), choices)
default_stat_pairs(NULL, choices)
```

`discrete_choices` *Choices for a categorical colour, shape, group or fill selector*

Description

The columns `facet_check()` accepts plus the logical columns, in data order, for any selector that maps one colour, shape or group to each level of a column. A column of IDs or gene names would ask for one level per row, so categorical columns with `max.levels` or more distinct values are left out, while a TRUE/FALSE column, which `facet_check()` leaves out of the split choices, is a sensible shape or group. Every built-in module builds its categorical Color By, Shape By, Group By and Fill By choices this way.

Usage

```
discrete_choices(data, max.levels = 50, keep = NULL)
```

Arguments

`data`	A data frame.
`max.levels`	A single positive number, passed to `facet_check()`. Categorical columns with this many or more distinct values are excluded.
`keep`	Column names to keep whatever their size, such as the column a caller named in `defaults`, so an explicit choice is always on offer. Names not in `data` are ignored.

Details

A selector that can also draw a numeric column as a gradient (the scatter module's Color By, say) should add its numeric columns to these itself.

Value

A character vector of column names, in the order they appear in `data`. `character(0)` when `data` is `NULL` or has no columns.

Author(s)

Jared Andrews

See Also

`facet_check()` for facet and split selectors.

Examples

```
df <- data.frame(
    sample = paste0("S", 1:60),
    group = rep(c("A", "B", "C"), 20),
    treated = rep(c(TRUE, FALSE), 30),
    score = rnorm(60)
)
# `sample` has 60 levels, so only `group` and `treated` are offered.
discrete_choices(df)

# A column named in `defaults` is offered regardless.
defaults <- list(group.by = "sample")
discrete_choices(df, keep = get_default(defaults, "group.by", NULL))
```

`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

`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 `shiny::reactive()` or `shiny::reactiveVal()` to have the input follow the parent app's state; see `setup_reactive_defaults()`.
`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_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

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. 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 (use x.order, the "Group Order" input, instead)
- 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.lineweight` - Boxplot line weight (UI: "Boxplot Line Weight", default: 0.5)
- `vlnplot.lineweight` - Violin line weight (UI: "Violin Line Weight", default: 0.5)
- `vlnplot.scaling` - Violin scaling method (UI: "Violin Scaling", default: "area")
- `vlnplot.width` - Violin width (derived from `boxgap`; not directly settable)
- `ridgeplot.lineweight` - Ridge line weight (UI: "Ridge Line Weight", default: 0.5)
- `ridgeplot.scale` - Ridge overlap scale (UI: "Ridge Scale (overlap)", default: 1.25)
- `ridgeplot.ymax.expansion` - Ridge Y-max expansion (UI: "Ridge Y-max Expansion", default: NA)
- `ridgeplot.shape` - Ridge shape (UI: "Ridge Shape", default: "smooth")
- `ridgeplot.bins` - Ridge bins (UI: "Ridge Bins", default: 30)
- `ridgeplot.binwidth` - Ridge binwidth (UI: "Ridge Binwidth", default: 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:

- `webgl` - Draw the points with WebGL once there are 5,000 or more, as set by the `VizModules.webgl.threshold` option (UI: "Plot with WebGL", default: TRUE)
- `x.order` - Order of the `group.by` groups along the x-axis (UI: "Group Order", drag and drop, default: the column's factor level order, else sorted). As a default, a character vector of levels of `group.by`; levels it leaves out follow in their default order. Not shown for a `group.by` with more than 20 groups
- `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)
- `axis.scientific` - Write numeric tick labels in scientific notation (UI: "Scientific Notation Ticks", default: FALSE; when off they never switch to it). See `apply_tick_notation()`
- `axis.padding.x` - Space between the data and the plot area's edges along X, as a fraction of the data's extent at each end (UI: "X Axis Padding", default: NA = the plot's own). See `apply_axis_padding()`
- `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

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

Value

A Shiny plotlyOutput for the 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

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 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

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 shiny::reactive() or shiny::reactiveVal() to have the input follow the parent app's state; see setup_reactive_defaults().
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_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

`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. 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); shown while `size.by` is unset
- `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
- `size.min` - Smallest point size a `size.by` column maps onto (UI: "Min Point Size", default: 1). Shown while `size.by` is set

- `size.max` - Largest point size a `size.by` column maps onto (UI: "Max Point Size", default: 6). Shown while `size.by` is set
- `size.scale.min` - `size.by` value drawn at the minimum point size, where the size legend starts (UI: "Size Scale Min", default: NA = the column's minimum); smaller values are drawn at the minimum size. Shown while `size.by` is set
- `size.scale.max` - `size.by` value drawn at the maximum point size, where the size legend ends (UI: "Size Scale Max", default: NA = the column's maximum); larger values are drawn at the maximum size. Shown while `size.by` is set
- `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: 1.03); 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.
- min.value - Minimum value (UI: "Min Value", default: NA)
- max.value - Maximum value (UI: "Max Value", default: NA)
- trajectory.group.by - Column whose groups the trajectories pass through (UI: "Trajectory Group By", default: "")
- add.trajectory.by.groups - Trajectories, each the groups it visits in order (UI: "Trajectories", a multiDynamicInput() with one row per trajectory, each picking its groups in order from the trajectory.group.by column; default: none). In defaults, a list of character vectors as dittoViz takes it, e.g. list(c("A", "B"), c("C", "D", "E")), applied while trajectory.group.by is its default column; the bracketed text "[A,B],[C,D,E]" also works. Changing the column clears the list. The module draws each as a path through its groups' medians, ending in an arrowhead (ggplotly drops dittoViz's). The medians are of the values as plotted, after any adjustment. On a split plot each panel's path runs through that panel's medians, and a path is left out of any panel, or plot, missing one of its groups. None are drawn while an adjustment makes an axis categorical. Zoomed in, a path stops at the panel's edge, but an arrowhead just past the edge is still drawn, as plotly does not clip annotations to the plot area
- trajectory.arrow.size - Arrowhead length in inches (UI: "Trajectory Arrow Size", default: 0.1)
- 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)
- `axis.scientific` - Write numeric tick labels in scientific notation (UI: "Scientific Notation Ticks", default: FALSE; when off they never switch to it). See `apply_tick_notation()`
- `axis.padding.x` - Space between the data and the plot area's edges along X, as a fraction of the data's extent at each end (UI: "X Axis Padding", default: NA = the plot's own). See `apply_axis_padding()`
- `axis.padding.y` - Space between the data and the plot area's edges along Y, as a fraction of the data's extent at each end (UI: "Y Axis Padding", default: NA = the plot's own). See `apply_axis_padding()`
- `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)
- model.legend.show - List the fit and model lines in the legend (UI: "Model Lines in Legend", default: FALSE). Off, a line fit to a Color By group hides and shows with that group's entry.
- marginal.x - Distributions of the X values drawn above the plot, any of "density", "histogram" and "rug" (UI: "Top Marginal (X)", default: none)
- marginal.y - Distributions of the Y values drawn to the right of the plot, any of "density", "histogram" and "rug" (UI: "Right Marginal (Y)", default: none)
- marginal.x.size - Share of the plot's height given to the top marginal (UI: "Top Marginal Height", default: 0.15). Shown while marginal.x is set
- marginal.y.size - Share of the plot's width given to the right marginal (UI: "Right Marginal Width", default: 0.15). Shown while marginal.y is set
- marginal.opacity - Opacity of the density and histogram fills and the rug lines (UI: "Marginal Opacity", default: 0.5). Shown while either marginal is set

- `marginal.bins` - Number of histogram bins (UI: "Histogram Bins", default: 30). Shown while either marginal has a histogram
- `marginal.binwidth` - Histogram bin width, overriding `marginal.bins` (UI: "Histogram Bin Width", default: NA = use `marginal.bins`). Shown while either marginal has a histogram
- `marginal.density.adjust` - Multiplier on the density bandwidth (UI: "Density Bandwidth Adjust", default: 1). Shown while either marginal has a density
- `marginal.rug.height` - Share of a marginal given to its rug when it sits alongside a density or histogram (UI: "Rug Height", default: 0.3). Shown while a marginal has both
- `marginal.rug.width` - Line width of the rug ticks, in pixels (UI: "Rug Line Width", default: 1). Shown while either marginal has a rug
- `marginal.axes` - Show the density or count axis, with its ticks and labels, of each density and histogram marginal (UI: "Show Marginal Axes", default: FALSE). Shown while either marginal has a density or histogram

Marginal plots show one distribution per Color By group, in the group's colour, or a single one in the Point color when the plot has no groups (including a numeric Color By). Like the fit lines, they are drawn from the plotted values, after any X/Y adjustment. A density and a histogram share their marginal, the histogram scaled to a density so the two match; a histogram on its own shows counts. The rug sits in its own strip next to the plot, with a lane for each group. Zooming the plot zooms its marginals, and hiding a group in the legend hides its marginals too. Marginals are not drawn on a split plot, nor along an axis an adjustment (`as.factor`) makes categorical.

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

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

Value

A Shiny plotlyOutput for the scatterplot

Author(s)

Jared Andrews

dittoViz_scatterPlotServer

Server logic for scatterPlot module

Description

Server logic for scatterPlot module

Usage

```
dittoViz_scatterPlotServer(
  id,
  data,
  hide.inputs = NULL,
  hide.tabs = NULL,
  defaults = NULL,
  fig.fn = 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().
fig.fn	An optional function function(fig, input, isolate_fn) returning the figure, for a wrapper module to add to or adjust the plot (extra layers, annotations, axis title text) before it is finalised. It runs in the figure's styling step (see apply_uniform_styling()), after the legend is styled and before the axis titles become draggable annotations, so whatever it adds reaches the rendered plot, manual-edit persistence and the source download alike. It sees the finished plot, marginal distributions included. That step is not cached, so it can read any reactive value. input is this module's input, which a wrapper sharing the module's id also writes its own inputs to. Read inputs as isolate_fn(input$key) so they respect the module's auto-update setting.

Value

The moduleServer function for the scatterPlot module.

Author(s)

Jared Andrews

See Also

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

`dittoViz_yPlotApp` *Create an example Modular yPlot Shiny Application*

Description

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

Usage

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

Arguments

`data_list`	An optional named list of data frames. If `NULL` (the default), 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 `shiny::reactive()` or `shiny::reactiveVal()` to have the input follow the parent app's state; see `setup_reactive_defaults()`.
`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_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

`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. 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 (use `x.order`, the "Group Order" input, 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)

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.lineweight` - Boxplot line weight (UI: "Boxplot Line Weight", default: 0.5)
- `vlnplot.lineweight` - Violin line weight (UI: "Violin Line Weight", default: 0.5)
- `vlnplot.scaling` - Violin scaling method (UI: "Violin Scaling", default: "area")
- `vlnplot.width` - Violin width (derived from `boxgap`; not directly settable)
- `ridgeplot.lineweight` - Ridge line weight (UI: "Ridge Line Weight", default: 0.5)
- `ridgeplot.scale` - Ridge overlap scale (UI: "Ridge Scale (overlap)", default: 1.25)
- `ridgeplot.ymax.expansion` - Ridge Y-max expansion (UI: "Ridge Y-max Expansion", default: NA)
- `ridgeplot.shape` - Ridge shape (UI: "Ridge Shape", default: "smooth")
- `ridgeplot.bins` - Ridge bins (UI: "Ridge Bins", default: 30)
- `ridgeplot.binwidth` - Ridge binwidth (UI: "Ridge Binwidth", default: NULL)
- `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:

- `webgl` - Draw the points with WebGL once there are 5,000 or more, as set by the `VizModules.webgl.threshold` option (UI: "Plot with WebGL", default: TRUE)
- `x.order` - Order of the `group.by` groups along the x-axis (UI: "Group Order", drag and drop, default: the column's factor level order, else sorted). As a default, a character vector of levels of `group.by`; levels it leaves out follow in their default order. Not shown for a `group.by` with more than 20 groups
- `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)
- `axis.scientific` - Write numeric tick labels in scientific notation (UI: "Scientific Notation Ticks", default: FALSE; when off they never switch to it). See `apply_tick_notation()`
- `axis.padding.x` - Space between the data and the plot area's edges along X, as a fraction of the data's extent at each end (UI: "X Axis Padding", default: NA = the plot's own). See `apply_axis_padding()`
- `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

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

Value

A Shiny plotlyOutput for the yPlot

Author(s)

Jared Andrews

`dittoViz_yPlotServer` *Server logic for yPlot module*

Description

Server logic for yPlot module

Usage

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

Arguments

`id`	The ID for the Shiny module.
`data`	A `reactive` containing the data frame to plot. 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 yPlot module.

Author(s)

Jared Andrews, Jacob Martin

See Also

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

`draw_to_png` *Render a grid or base drawing to PNG bytes*

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 `shiny::renderPlot()`'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 `draw_to_svg()`.
`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

`draw_fn`	A function of no arguments that draws onto the active device.
`width, height`	Size in pixels.
`res`	Pixels per inch used to convert `width`/`height` to the inches the SVG devices take. The default matches `shiny::renderPlot()`'s own `res`, 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 `ComplexHeatmap` 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.
`id_prefix`	Optional character scalar used to namespace the fragment's ids, so several fragments can share one document. See `.svg_namespace_ids()`.
`bg`	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

`data`	A data.frame or tibble containing the data to plot.
`x`	Character vector of column name(s) for x-axis values. Maximum 2 values allowed. If 1 value: creates single dot plot. If 2 values: creates dumbbell plot with connecting segments.
`y`	Character, column name for the y-axis (categorical variable recommended).
`colour.by`	Character, how to color the markers. Options: "X variables" (different colors for each x variable) or "Y variables" (different colors for each y category). Default: "X variables".
`palette.selection`	Character vector of hex colors for marker colors. A named vector is matched by name to the x variables (`colour.by = "X variables"`) 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.
`show.legend`	Logical, whether to display the legend. Default: TRUE.
`facet.by`	Optional character, column name to facet plots by. Creates subplots for each unique value. Default: NULL.
`line.colour`	Character, hex color for the connecting lines between dumbbell points. Default: "gray80".
`point.size`	Numeric, diameter of the markers in pixels. Default: 12.

`facet.scales` Character, controls axis scaling across facets. Options: "fixed" (same for all), "free" (independent), "free_x" (independent x-axis), "free_y" (independent y-axis). Default: "fixed".

`subplot.margin` Numeric, spacing between facet panels as a fraction of the plot area. May be a single value (applied to both directions) or a length-2 vector `c(horizontal, vertical)` to control the gap between columns and rows separately. Default: 0.06.

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

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

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

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

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

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

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

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

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

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

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

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

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

`axis.title.font.size`
Numeric, font size for the x/y axis titles. Default: 18.

`axis.title.font.color`
Character, hex color for the x/y axis titles. Default: "black".

`axis.title.font.family`
Character, font family for the x/y axis titles. Default: "Arial".

`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".

`facet.title.font.size`
Numeric, font size for the facet panel titles. Default: 18.

`facet.title.font.color`
Character, hex color for the facet panel titles. Default: "black".

`facet.title.font.family`
Character, font family for the facet panel titles. Default: "Arial".

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

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

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

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

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

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

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

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

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

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

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

Details

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

Modes:

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

Coloring options:

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

Value

A plotly object representing the interactive dumbbell plot.

Author(s)

Jacob Martin

Examples

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

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

---

dumbbellPlotApp *Create a Shiny App for Dumbbell Plots*

Description

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

Usage

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

Arguments

`data_list`	An optional named list of data frames. If `NULL` (the default), 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 `shiny::reactive()` or `shiny::reactiveVal()` to have the input follow the parent app's state; see `setup_reactive_defaults()`.
`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

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.

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)
- `axis.scientific` - Write numeric tick labels in scientific notation (UI: "Scientific Notation Ticks", default: FALSE; when off they never switch to it). See `apply_tick_notation()`
- `axis.padding.x` - Space between the data and the plot area's edges along X, as a fraction of the data's extent at each end (UI: "X Axis Padding", default: NA = the plot's own). See `apply_axis_padding()`
- `axis.padding.y` - Space between the data and the plot area's edges along Y, as a fraction of the data's extent at each end (UI: "Y Axis Padding", default: NA = the plot's own). See `apply_axis_padding()`
- `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

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

Value

A Shiny plotlyOutput for the dumbbellPlot

Author(s)

Jacob Martin, Jared Andrews

`dumbbellPlotServer` *Server logic for dumbbellPlot module*

Description

Server logic for dumbbellPlot module

Usage

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

Arguments

`id`	The ID for the Shiny module.
`data`	A `reactive` containing the data frame to plot. 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 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")
```

`error_bar_halfwidth` *Half-width of a group's error bar*

Description

What `linePlot()` draws either side of a group's mean for `error.type` = `"sd"`, `"sem"` or `"ci95"`. Missing values are dropped before `n` is counted, so the bar describes the values that went into the mean.

Usage

```
error_bar_halfwidth(
  x,
  type = c("sd", "sem", "ci95"),
  ci.method = c("normal", "t")
)
```

Arguments

`x`	Numeric vector of one group's y-values.
`type`	One of `"sd"` (standard deviation), `"sem"` (standard error of the mean, `sd / sqrt(n)`) or `"ci95"` (95% confidence interval for the mean).
`ci.method`	For `type` = `"ci95"`, `"normal"` scales the standard error by the 97.5th percentile of the normal distribution (1.96) and `"t"` by that of the t distribution on `n - 1` degrees of freedom. Ignored for the other types.

Details

A module that summarises its own groups (over a numeric x, say, which linePlot's own summaries do not cover) can work out each interval with this and hand `linePlot()` the bounds through `error.type` = `"columns"`, so its bars and ribbons match the line plot module's.

Value

A single number, or `NA_real_` for fewer than two non-missing values, which have no spread to draw.

Author(s)

Jared Andrews

See Also

`linePlot()`

Examples

```
x <- c(4.1, 5.3, 4.8, 6.0, 5.5)
error_bar_halfwidth(x, "sd")
error_bar_halfwidth(x, "sem")
# The t interval is wider for small groups.
error_bar_halfwidth(x, "ci95", "normal")
error_bar_halfwidth(x, "ci95", "t")
```

---

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` *Example single-cell-style composition data for the freqPlot module*

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-raw/generate_example_data.R`.

`example_demographics` *Example demographics dataset*

Description

A simulated employee survey dataset with 500 rows spanning six departments and four job levels. 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, Dise 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 *Example grouped iris dataset*

Description

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

Usage

```
example_iris
```

Format

A data frame with 150 rows and 6 columns:

Sepal.Length Sepal length in cm

Sepal.Width Sepal width in cm

Petal.Length Petal length in cm

Petal.Width Petal width in cm

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

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

Author(s)

Jared Andrews

Source

Generated from the classic iris dataset.

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

Description

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

Usage

```
example_markers
```

Format

A data frame with 104 rows and 4 columns:

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

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

avg_expression Average expression of the gene in the cell type

pct_expressed Percent of cells in the cell type expressing the gene

Author(s)

Jacob Martin

Source

Simulated in data-raw/generate_example_data.R.

`example_mtcars` *Example mtcars dataset with factors*

Description

The classic mtcars dataset with 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 *Example population dataset*

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` *Example sales dataset*

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

Example school earnings dataset for dumbbell plots

Description

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

Usage

```
example_school_earnings
```

Format

A data frame with 6 rows and 4 columns:

School University name

Women Median earnings for women (thousands of USD)

Men Median earnings for men (thousands of USD)

Group University type (STEM-heavy or Liberal Arts)

Author(s)

Jacob Martin

Source

Generated in data-raw/generate_example_data.R.

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

Description

A dataset of skill ratings across 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.

`facet_check` *Identify columns valid for faceting or splitting*

Description

Scans a data frame and returns the names of columns that are appropriate choices for a facet/split selector. A column qualifies when it is categorical (character or factor) and has fewer than `max.levels` unique values. This keeps facet/split inputs from offering numeric columns or high-cardinality categoricals that would produce an unwieldy number of panels. The modules also use it, through `discrete_choices()`, to limit their categorical colour, shape and group selectors, where a column of IDs would otherwise ask for one colour or shape per row.

Usage

```
facet_check(data, max.levels = 50)
```

Arguments

`data`	A data frame whose columns are evaluated.
`max.levels`	A single positive number. Columns with this many or more distinct values are excluded. Defaults to 50.

Details

Intended to populate the `choices` of a facet/split `viz_select_input()` via `update_viz_select()` inside a module server, so that only sensible faceting variables are exposed to the user.

A column is considered valid when both of the following hold:

- It is categorical: `is.character(col)` or `is.factor(col)`.
- It has fewer than `max.levels` unique values (`NA`s excluded).

Numeric columns and categorical columns with `max.levels` or more distinct values are always excluded.

Value

A character vector of column names suitable for faceting/splitting. Returns `character(0)` when no column qualifies.

Author(s)

Jacob Martin

Examples

```
facet_check(iris)
facet_check(mtcars)
# A lower cap drops the 3-level Species column.
facet_check(iris, max.levels = 3)
```

`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_annotations =)); in the latter case only the primary table is filtered and shown in the panel's table pane.
module_registry	An optional named list describing the plot modules to offer. If NULL (the default), all bundled VizModules modules are offered, each opening on the same example figure as in moduleGalleryApp(). Each entry is itself a list with components: label (character, shown in the picker), dataset (character, the dataset name its defaults were written for), inputs_ui, output_ui, and server_fn (the module's three functions), and defaults (a named list of input defaults applied only when dataset is the chosen dataset). An entry may also carry primary.table, 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 figureBuilderServer() for the vector_svg hook that lets a non-plotly module take part in the SVG figure export.
title	A character string used as the page title and header (default: "VizModules Figure Builder").
return_components	Logical. When FALSE (the default) a shiny::shinyApp() object is returned. When TRUE a named list with ui and server elements is returned instead, which is convenient for deployment scripts that need an explicit shinyApp(ui, server) call.

Details

Datasets are supplied via data_list and seed the "Add Plot" dialog; users can also upload additional datasets (CSV, TSV, 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

id	The ID for the Shiny module. Must match the id given to figureBuilderUI().

`data_list` An optional named list of data frames that seed the dataset registry. If `NULL` (the default), the bundled example datasets (plus a `sales_by_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_annotations = )`); in the latter case only the primary table is filtered and shown in the panel's table pane.

`module_registry`

An optional named list describing the plot modules to offer. If `NULL` (the default), all bundled VizModules modules are offered, each opening on the same example figure as in `moduleGalleryApp()`. Each entry is itself a list with components: `label` (character, shown in the picker), `dataset` (character, the dataset name its `defaults` were written for), `inputs_ui`, `output_ui`, and `server_fn` (the module's three functions), and `defaults` (a named list of input defaults applied only when `dataset` is the chosen dataset). `defaults` is passed to both `inputs_ui` and `server_fn`, so it can seed server-rendered controls such as the group color picker. An entry may also carry `primary.table`, 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 *UI component for the Figure Builder module*

Description

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

Usage

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

Arguments

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

Details

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

Usage

```
finalize_manual_edits(
  fig,
  plot_source,
  store,
  session,
  regen_keys = character(0)
)
```

Arguments

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

1. tags the figure with the module's plotly event `source` so its `plotly_relayout` events are captured by `setup_manual_edits()`;
2. restores any manually repositioned legend, annotations, axis titles, and colorbar captured so far;
3. records the figure so future relayout events can be matched to stable annotation keys (surviving re-ordering on rebuild);
4. attaches the JavaScript listener that forwards colorbar drags;
5. has the figure re-rendered in place: `layout.transition = TRUE` makes the plotly widget update its graph with `Plotly.react()` instead of purging and recreating it, which is faster and keeps the user's zoom and toggled legend entries whenever `layout.uirevision` is unchanged (the cached modules set it per build of the plot, so styling changes keep the view). A small script clears the previous render's event listeners first, which purging used to; and
6. sends only the layout when nothing else changed: if the figure's traces, config, `layout.uirevision`, and everything else but its layout match the figure last sent to this output, the new layout goes to the browser in a small message, applied with `Plotly.react()` over the traces already drawn, and the render is cancelled (`req(FALSE, cancelOutput = TRUE)`), so no code after this call runs. A styling change then costs kilobytes rather than the whole figure. Set `options(VizModules.layout.updates = FALSE)` to always send the whole figure.

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()`. Inside a render, when only the layout changed, it instead sends the layout and cancels the render.

Author(s)

Jared Andrews

See Also

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

Examples

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

## End(Not run)
```

flatten_palette_options *Flatten nested palette options*

Description

Converts a nested list of palette choices into a single-level named list where each entry is a character vector of colors.

Usage

```
flatten_palette_options(palettes)
```

Arguments

palettes	A named list of palettes or nested category lists.

Value

A flattened named list of palettes.

Author(s)

Jared Andrews

See Also

default_palettes(), resolve_palette()

Examples

```
pal_lookup <- flatten_palette_options(default_palettes()[["choices"]])
head(names(pal_lookup))
```

generate_pair_strings *Generate comparison pair strings from data columns*

Description

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

Usage

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

Arguments

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

Value

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

Author(s)

Jared Andrews

Examples

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

---

get_default *Resolve a default value from a named list*

Description

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

Usage

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

Arguments

defaults	A named list of default values, or NULL. Individual entries may be a reactive()/reactiveVal.
key	Character string — the name to look up.
fallback	The value to return when key is absent or fails validation.
validator	An optional single-argument predicate function (e.g., is.numeric, is.logical). When supplied, the stored value is returned only if validator(value) is TRUE. 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` *Extract parameter documentation from an R function help page*

Description

Reads the help page of a function and returns the descriptions of the parameters asked for, as plain text suitable for UI tooltips. Code is set in backticks and other Rd markup is reduced to its text.

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. Co-documented parameters (e.g., `x.by` and `y.by`) can be grouped together in a vector within the list; each name gets the shared description.
`cap`	Logical; if TRUE, capitalize the first letter of each description.

Details

Each package's help pages are read once per R session and kept, so building a module's UI again (another tab, another dataset, another Figure Builder panel) costs nothing. A package loaded with `pkgload::load_all()` is read from its source `man/` directory, so its own, current docs are used.

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

Examples

```
get_documentation("stats::lm", selected = list("formula", "data"), cap = TRUE)
```

`get_model_backend` *Get a registered model backend*

Description

Retrieve a model backend from the registry by name.

Usage

```
get_model_backend(name)
```

Arguments

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

Value

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

Author(s)

Jacob Martin

Examples

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

heatmap_fit_width *Fit a hand-built InteractiveComplexHeatmap widget to its container's width*

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::makeInteracti` say.

Usage

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

Arguments

`ui`	The UI returned by `InteractiveComplexHeatmap::InteractiveComplexHeatmapOutput()` (or the `originalHeatmapOutput()`/`subHeatmapOutput()` 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 `ComplexHeatmap_HeatmapOutputUI()` 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 *Check if column inputs contain mixed data types*

Description

This function validates that a vector of column names from a data frame contains columns of only one data type category: either all numeric OR all categorical (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)
```

`level_order_choices` *The levels a categorical column is plotted in, optionally reordered*

Description

Gives the levels of `data[[column]]` that are present, in the order a plot draws them by default - factor level order for a factor, sorted values otherwise - with any levels named in `order` moved to the front in that order.

Usage

```
level_order_choices(data, column, order = NULL, max.levels = 20)
```

Arguments

`data`	A data frame.
`column`	The name of the column.
`order`	Optional character vector of levels in the order wanted. Entries that are not present levels are ignored, and levels it leaves out follow in their default order, so an order kept from an earlier filtering of the data still applies to what is left.
`max.levels`	The most levels worth offering for reordering. A column with more gives `NULL`.

Value

A character vector of levels, or `NULL` when `column` is missing, numeric, or has no levels or more than `max.levels` of them.

Author(s)

Jared Andrews

See Also

`level_order_input()`, `apply_level_order()`, `setup_level_order()`

Examples

```
level_order_choices(example_iris, "Species")
level_order_choices(example_iris, "Species", order = "virginica")

# Numeric columns have no levels to order
level_order_choices(example_iris, "Sepal.Length")
```

level_order_input *Drag-and-drop control for the order of a categorical axis*

Description

A shinyjqui::orderInput() laid out to fill one organize_inputs() grid cell, listing the levels of a categorical column as buttons that can be dragged into the order they should be plotted in. Its value is the levels in their current order, a character vector.

Usage

```
level_order_input(
  inputId,
  label = "X Order",
  items = NULL,
  tooltip = NULL,
  width = "100%"
)
```

Arguments

inputId	The input ID, namespaced as for any other input (e.g. ns("x.order")).
label	The label shown above the control.
items	Character vector of levels to show, in order, typically from level_order_choices(). NULL or empty renders an empty, hidden control.
tooltip	Optional tooltip text shown on hover.
width	The CSS width of the control.

Details

The control hides its grid cell whenever it has no items, which is how setup_level_order() takes it off screen for a column it cannot order (a numeric one, or one with more levels than are practical to drag). A cell hidden with hide_input() stays hidden either way.

Value

A shiny.tag wrapping the order input.

Author(s)

Jared Andrews

See Also

level_order_choices(), apply_level_order(), setup_level_order()

Examples

```
level_order_input("x.order", "X Order", items = c("Healthy", "Disease"))

# The items for a column of a data frame, in the order they are drawn by default
level_order_input("x.order", "X Order",
    items = level_order_choices(example_iris, "Species")
)
```

`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", "columns"),
  error.ci.method = c("normal", "t"),
  error.lower = NULL,
  error.upper = NULL,
  error.ribbon = FALSE,
  error.ribbon.opacity = 0.25
)
```

Arguments

data	A data.frame or tibble containing the data to plot.
x	Character vector of column name(s) for the x-axis. Multiple columns create separate traces.
y	Character vector of column name(s) for the y-axis. Multiple columns create separate traces.
palette.selection	Character vector of hex colors for line colors. Used to assign colors to groups or traces.
plot.mode	Character, plotly mode for plot type. Options: "lines", "markers", "lines+markers". Default: "lines".
line.type	Character, line style. Options: "solid", "dot", "dash", "longdash", "dashdot", "longdashdot". Default: "solid".
colour.group.by	Character or formula, column name(s) to group lines by color. Can be a formula

like ~ column_name. Ignored if multiple x or y columns are provided. Default: NULL.

show.legend Logical, whether to display the legend. Default: TRUE.

facet.by Optional character, column name to facet plots by. Creates subplots for each unique value. Default: NULL.

facet.scales Character, controls axis scaling across facets. Options: "fixed" (same for all), "free" (independent), "free_x" (independent x-axis), "free_y" (independent y-axis). Default: "fixed".

facet.nrow Optional integer, number of rows in the faceted subplot grid. If NULL (default), a single row is used unless facet.ncol is supplied, in which case the number of rows is derived from the number of facet levels.

facet.ncol Optional integer, number of columns in the faceted subplot grid. If NULL (default), columns are derived from facet.nrow and the number of facet levels. Only one of facet.nrow / facet.ncol needs to be set; if both are provided, facet.nrow takes precedence.

subplot.margin Numeric, spacing between facet panels as a fraction of the plot area. May be a single value (applied to both directions) or a length-2 vector c(horizontal, vertical) to control the gap between columns and rows separately. Default: 0.05.

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

grid.color Character, hex color for gridlines. Default: "#CCCCCC".

axis.title.font.size
Numeric, font size for the x/y axis titles. Default: 18.

`axis.title.font.color`
: Character, hex color for the x/y axis titles. Default: "black".

`axis.title.font.family`
: Character, font family for the x/y axis titles. Default: "Arial".

`facet.title.font.size`
: Numeric, font size for the facet panel titles. Default: 18.

`facet.title.font.color`
: Character, hex color for the facet panel titles. Default: "black".

`facet.title.font.family`
: Character, font family for the facet panel titles. Default: "Arial".

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

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

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

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

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

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

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

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

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

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

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

`color.adjustment`
: Optional character or function, transformation to apply to color grouping variable. Same options as x.adjustment and y.adjustment. Default: NULL.

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

`error.colour`
: Hex colour of the error bars. Default: `NULL` (the series' own colour).

`error.width`
: Numeric, thickness of the error bars. Default: `NULL` (plotly's default).

`error.bar`
: Logical, whether to draw the interval `error.type` selects as error bars. Needs a single `y`, and a single categorical `x` unless `error.type = "columns"`. For `"sd"`,

"sem" and "ci95", each bar spans the plotted group mean plus or minus that amount, computed from the group's y-values, where a group is a single x category, split further by colour.group.by and facet.by when those are set. A group with fewer than two observations has no spread and is drawn without a bar. Default: FALSE.

error.type
: What the error bars and ribbon show, one of "sd" (one standard deviation, the default), "sem" (one standard error of the mean, sd / sqrt(n)), "ci95" (a 95% confidence interval for the mean) or "columns" (the interval between the error.lower and error.upper columns). Missing values are ignored when counting n. The first three summarise each group, so they need a categorical x. "columns" works with any x. It suits intervals worked out beforehand, such as a model's confidence or prediction interval, a forecast range or a min/max range.

error.ci.method
: How error.type = "ci95" 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.

error.lower, error.upper
: Optional character, names of the numeric columns holding each point's lower and upper bound, for error.type = "columns". Either order works. The bounds are on the y-axis's scale, so y.adjustment is applied to them too. With a categorical x, they are averaged per group, as y is, so supply one row per x position (and colour/facet group). A bound that lies on the wrong side of the line draws no bar on that side. Default: NULL.

error.ribbon
: Logical, whether to draw the interval error.type selects as a shaded band (a ribbon) behind each line, in the line's own colour, on its own or alongside the error bars. It has the same requirements as error.bar. A point without an interval leaves a gap in the band, so an isolated point's interval shows only as a bar. No ribbon is drawn for a numeric colour.group.by, which plotly does not split into separate lines. Hovering a band's edge shows that bound. Default: FALSE.

error.ribbon.opacity
: Numeric between 0 and 1, the ribbon's fill opacity. Default: 0.25.

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
)

# Each product line's yearly mean revenue, with a 95% confidence interval ribbon.
fig2 <- linePlot(
    data = example_sales,
    x = "year",
    y = "revenue",
    colour.group.by = "product_line",
    palette.selection = palette[1:3],
    error.type = "ci95",
    error.ribbon = TRUE
)

# A band from columns holding precomputed bounds, on a numeric x-axis.
fit <- data.frame(t = 1:20, est = sqrt(1:20))
fit$lo <- fit$est - 0.4
fit$hi <- fit$est + 0.4
fig3 <- linePlot(
    data = fit,
    x = "t",
    y = "est",
    palette.selection = "#1B9E77",
    error.type = "columns",
    error.lower = "lo",
    error.upper = "hi",
    error.ribbon = TRUE
)
```

`linePlotApp` *Create an example Modular linePlot Shiny Application*

Description

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

Usage

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

```
  hide.tabs = NULL
)
```

Arguments

data_list	An optional named list of data frames. If NULL (the default), 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 shiny::reactive() or shiny::reactiveVal() to have the input follow the parent app's state; see setup_reactive_defaults().
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_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` *Input UI components for the linePlot module*

Description

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

Usage

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

Arguments

`id`	The ID for the Shiny module.
`data`	The data frame used for plot generation.
`defaults`	A named list of default values for the inputs. 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. 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 the interval as error bars (UI: "Error Bars", default: TRUE; requires a single X and a single Y, and a categorical X unless the error type is "columns")
- `error.ribbon` - Show the interval as a shaded band behind each line (UI: "Error Ribbon", default: FALSE; same requirements as `error.bar`)
- `error.type` - What the error bars and ribbon show: "sd", "sem", "ci95" or "columns" (UI: "Error 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 type is "ci95")
- `error.lower`, `error.upper` - Numeric columns holding each point's lower and upper bound (UI: "Lower Bound" and "Upper Bound", default: ""; only shown while the error type is "columns")
- `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)
- `error.ribbon.opacity` - Ribbon fill opacity, 0 to 1 (UI: "Ribbon Opacity", default: 0.25)
- `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)
- `axis.scientific` - Write numeric tick labels in scientific notation (UI: "Scientific Notation Ticks", default: FALSE; when off they never switch to it). See `apply_tick_notation()`
- `axis.padding.x` - Space between the data and the plot area's edges along X, as a fraction of the data's extent at each end (UI: "X Axis Padding", default: NA = the plot's own). See `apply_axis_padding()`
- `axis.padding.y` - Space between the data and the plot area's edges along Y, as a fraction of the data's extent at each end (UI: "Y Axis Padding", default: NA = the plot's own). See `apply_axis_padding()`
- `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 *Output UI components for the linePlot module*

Description

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

Usage

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

Arguments

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

Value

A Shiny plotlyOutput for the linePlot

Author(s)

Jacob Martin

linePlotServer *Server logic for linePlot module*

Description

Server logic for linePlot module

Usage

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

Arguments

id	The ID for the Shiny module.
data	A reactive containing the data frame to plot. 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 *Convert linetype name to plotly dash style*

Description

Maps common linetype names to plotly dash specifications.

Usage

```
linetype_to_dash(linetype)
```

Arguments

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

Value

Character. Plotly-compatible dash specification.

Author(s)

Jared Andrews

Examples

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

list_model_backends *List registered model backends*

Description

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

main_title_input_ids *Input IDs that style the main plot title*

Description

The inputs a faceted plot has no use for, since `add_plot_config()` disables its main title. `toggle_facet_title_inputs()` hides them while a plot is faceted.

Usage

```
main_title_input_ids
```

Format

A character vector of input IDs (without namespacing).

Author(s)

Jared Andrews

See Also

toggle_facet_title_inputs()

Examples

```
main_title_input_ids
```

---

mark_overlay_axes *Exempt a figure's overlay axes from the shared styling*

Description

Marks axes as belonging to panels drawn beside the main plot, such as marginal distributions, annotation strips, or gene tracks. apply_uniform_styling() then leaves them as they were drawn: the Axes tab's fonts, ticks, gridlines, and lines are not applied to them, reference lines are not drawn on them, and they do not count as facets, so the plot's own axis titles stay draggable. This is the approach for a figure built with hand-set axis domains, where only the primary xaxis and yaxis follow the Axes tab.

Usage

```
mark_overlay_axes(fig, axes)
```

Arguments

fig	A plotly figure, already built with plotly::plotly_build().
axes	Character vector of layout axis names ("xaxis2", "yaxis3"), not trace references ("x2", "y3").

Details

Link an overlay panel to the plot with matches, as add_marginal_plots() does, so it zooms and pans with it.

Value

The figure, with axes added to any axes already marked.

Author(s)

Jared Andrews

See Also

add_marginal_plots(), apply_uniform_styling()

Examples

```
fig <- plotly::plot_ly()
fig <- plotly::add_trace(fig, x = 1:3, y = 1:3, type = "scatter", mode = "lines")
fig <- plotly::add_trace(fig,
    x = 1:3, y = 3:1, type = "scatter", mode = "lines", xaxis = "x2", yaxis = "y2"
)
fig <- plotly::layout(fig,
    xaxis = list(domain = c(0, 1)), yaxis = list(domain = c(0, 0.8)),
    xaxis2 = list(domain = c(0, 1), matches = "x", anchor = "y2"),
    yaxis2 = list(domain = c(0.85, 1))
)
fig <- plotly::plotly_build(fig)
mark_overlay_axes(fig, c("xaxis2", "yaxis2"))
```

---

merge_annotation_sets *Merge two sets of plotly point annotations*

Description

Appends `extra` annotations to `annos`, skipping any that already describe the same point and text so a point selected by hand and also matched by a highlight value is only labelled once.

Usage

```
merge_annotation_sets(annos, extra)
```

Arguments

`annos`	List of plotly annotation objects, or NULL.
`extra`	List of plotly annotation objects to append, or NULL.

Value

The combined list of annotation objects, or NULL when both are empty.

Author(s)

Jared Andrews

See Also

`create_selected_annotations()`, `create_highlight_annotations()`

Examples

```
a <- list(list(x = 1, y = 2, text = "A"))
b <- list(list(x = 1, y = 2, text = "A"), list(x = 3, y = 4, text = "B"))
length(merge_annotation_sets(a, b))
```

`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 `shiny::shinyApp()` object is returned. When `TRUE` a named list with `ui` and `server` elements is returned instead, which is convenient for deployment scripts that need an explicit `shinyApp(ui, server)` call.

Details

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` *Compact multi-group color picker input*

Description

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

Usage

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

Arguments

`inputId`	Character. Shiny input id.
`label`	Optional label displayed above the control.
`groups`	Character or factor vector of group names.
`palette_options`	Named list of palettes (each a character vector of colors). Defaults to the palettes from `default_palettes()`.
`selected_palette`	Optional name of the palette to preselect.
`colors`	Optional named vector of starting colors. Values are matched to `groups` by name when provided.
`width`	Optional CSS width for the container.
`show_text`	Logical. If `TRUE`, show editable hex text inputs beside the color pickers.

`compact`	Logical. If `TRUE`, renders a tighter layout with reduced spacing, smaller controls, and narrower palette selector.
`panel`	Logical. If `FALSE`, removes the surrounding panel/well styling (border, padding, background).

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)
}
```

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

Description

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

Usage

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

Arguments

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

Details

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

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

A `"select"` field given `multiple = TRUE` in its `args` reports a character vector of its items, in the order they were picked, which suits a field listing steps in order. Pre-fill one in `elements` with a character vector in the order wanted.

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)
}
```

na_to_null *Convert NA or empty string to NULL*

Description

A helper function to convert NA values or empty strings to NULL. Used to handle Shiny input default values which return NA or "" instead of NULL when the input field is empty. numericInput returns NA, textInput returns "".

Usage

```
na_to_null(x)
```

Arguments

x	A value that may be NA or an empty string.

Value

NULL if x is a single NA value or empty string, otherwise x unchanged.

Author(s)

Jared Andrews

See Also

blank_to_null(), nz_value()

Examples

```
na_to_null(NA)
na_to_null("")
na_to_null(5)
```

note_brackets_skipped *Tell the user why significance brackets are missing from the plot*

Description

Brackets are stacked above the data on the y-axis, so they cannot be drawn when the plotted values run along the x-axis (a rotated box plot, or a dittoViz plot containing a ridge plot). The tests are still run and shipped in the source-data download; this says so. The notification has a fixed id, so rebuilding the plot replaces it rather than stacking copies.

Usage

```
note_brackets_skipped(session = shiny::getDefaultReactiveDomain())
```

Arguments

session	The Shiny session, from inside moduleServer().

Value

Called for its side effect; NULL, invisibly.

Author(s)

Jared Andrews

See Also

apply_stat_annotations(), compute_pairwise_stats()

Examples

```
## Not run:
# Inside a module's renderPlotly(), when the values run along the x-axis:
note_brackets_skipped()

## End(Not run)
```

---

nz_value *Is an input's value a usable, non-empty string?*

Description

The check for the column-selecting inputs, whose "nothing chosen" state is the empty string rather than NULL. A Shiny input that has not reported yet is NULL, and both nzchar(NULL) and NULL == "" are logical(0), which makes if (...) an error rather than FALSE; this returns FALSE instead.

Usage

```
nz_value(x)
```

Arguments

x	A value from a Shiny input.

Value

TRUE for a length-1, non-NA, non-empty character scalar; FALSE for anything else, NULL included.

Author(s)

Jared Andrews

See Also

blank_to_null()

Examples

```
nz_value(NULL)
nz_value("")
nz_value(NA_character_)
nz_value("Species")
```

---

organize_inputs *Organize arbitrary Shiny inputs into a grid layout*

Description

Organize arbitrary Shiny inputs into a grid layout

Usage

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

Arguments

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

Value

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

Author(s)

Jared Andrews

Examples

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

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

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

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

---

parallelCoordinatesPlot

Create an Interactive Parallel Coordinates Plot with plotly

Description

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

Usage

```
parallelCoordinatesPlot(
  data,
```

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

Arguments

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

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

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

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

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

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

Details

plotly draws a parallel coordinates trace's lines with WebGL at a fixed width of one pixel and fully opaque: its `line` takes only a colour and colour scale, so line width and opacity cannot be set.

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"
)
```

---

parallelCoordinatesPlotApp

Create an example Modular parallelCoordinatesPlot Shiny Application

---

Description

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

Usage

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

Arguments

data_list	An optional named list of data frames. If NULL (the default), the module's example dataset is used, along with its showcase defaults. Each data frame should contain at least two numeric or categorical columns.
defaults	A named list of input IDs and their default values to apply on startup. An entry may also be a shiny::reactive() or shiny::reactiveVal() to have the input follow the parent app's state; see setup_reactive_defaults().
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_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

`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. 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)

plotly draws parallel coordinates lines one pixel wide and fully opaque, with no way to set their width or opacity, so there are no inputs for either.

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)
- `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()`, `parallelCoordinatesPlo`
`parallelCoordinatesPlotApp()`

Examples

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

parallelCoordinatesPlotOutputUI *Output UI components for the parallelCoordinatesPlot module*

Description

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

Usage

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

Arguments

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

Value

A Shiny plotlyOutput for the parallelCoordinatesPlot

Author(s)

Jacob Martin, Jared Andrews

parallelCoordinatesPlotServer *Server logic for parallelCoordinatesPlot module*

Description

Server logic for parallelCoordinatesPlot module

Usage

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

Arguments

id	The ID for the Shiny module.
data	A reactive containing the data frame to plot. 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 parallelCoordinatesPlot module.

Author(s)

Jacob Martin, Jared Andrews

See Also

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

---

parse_highlight_values

Parse a highlight string into the values it names

Description

Highlight values are typed as a comma- or newline-separated list, and may also be separated by spaces. A value containing a space ("CD4 T", "Player A") would be impossible to name that way, so each comma- or newline-delimited entry is kept whole when it is one of available, and split on whitespace otherwise.

Usage

```
parse_highlight_values(x, available = NULL)
```

Arguments

x	A single string, e.g. "CD4 T, B" or "P01 P07", or NULL.
available	Character vector of the values the entries name (the annotate.by column), or NULL to split every entry on whitespace.

Value

A character vector of unique, non-blank values, possibly empty.

Author(s)

Jared Andrews

See Also

apply_highlight_styling(), create_highlight_annotations(), uniform_annotation_inputs_ui()

Examples

```
parse_highlight_values("CD4 T, B", available = c("CD4 T", "B", "NK"))
parse_highlight_values("P01 P07")
```

---

parse_numeric_list *Parse comma-separated numeric string to vector*

Description

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

Usage

```
parse_numeric_list(text)
```

Arguments

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

Details

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

Value

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

Author(s)

Jared Andrews

Examples

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

---

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

Description

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

Usage

```
parse_pair_strings(pair_strings)
```

Arguments

`pair_strings`	Character vector of pair strings from UI input.

Value

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

Author(s)

Jared Andrews

Examples

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

piePlot *Create a plotly pie chart*

Description

Create a plotly pie chart

Usage

```
piePlot(
  df,
  labels,
  values,
  colors = NULL,
  palette = NULL,
  hole = 0,
  textinfo = "label+percent",
  textposition = "auto",
  insidetextorientation = "auto",
  sort = TRUE,
  direction = "counterclockwise",
  rotation = 0,
  show.legend = TRUE,
  legend.orientation = "h",
  legend.x = 0.5,
  legend.y = -0.1,
  legend.font.family = "Arial",
  legend.font.size = 12,
  legend.font.color = "#000000",
  title.text = "",
  title.font.family = "Arial",
  title.font.size = 18,
  title.font.color = "#000000",
  title.x = 0.5,
  text.font.family = "Arial",
  text.font.size = 12,
  text.font.color = "#000000",
  slice.line.color = "#FFFFFF",
  slice.line.width = 0
)
```

Arguments

df	A data frame where each row already represents a summarized slice (e.g., counts per category) with label and value columns.
labels	Character, name of the column to use for the slice labels.

`values` Character, name of the column to use for the aggregated values (slice sizes).

`colors` Optional character vector of hex colors for the slices. If named, values are matched to the values in `labels`; otherwise colours are recycled in data order.

`palette` Optional character vector of fallback colors used when `colors` is not supplied or missing values are present.

`hole` Numeric value between 0 and 1 for the hole size (0 for pie chart, >0 for donut chart). Default: 0.

`textinfo` Character string specifying the text info to show on slices. Any combination of "label", "text", "value", "percent" joined with a "+" (e.g., "label+percent") or "none" to hide text. Default: "label+percent".

`textposition` Character, position of the text relative to the slice. Options: "auto", "inside", "outside", or "none". Default: "auto".

`insidetextorientation`
Character, orientation for inside text. Options: "auto", "horizontal", "radial", or "tangential". Default: "auto".

`sort` Logical, whether to sort slices by their values in descending order. Default: TRUE.

`direction` Character, direction of slice progression. Options: "counterclockwise" or "clockwise". Default: "counterclockwise".

`rotation` Numeric, starting angle of the first slice in degrees (0-360). Default: 0.

`show.legend` Logical, whether to display the legend. Default: TRUE.

`legend.orientation`
Character, legend orientation. Options: "h" (horizontal) or "v" (vertical). Default: "h".

`legend.x` Numeric, horizontal legend position offset (0-1, where 0=left, 1=right). Default: 0.5.

`legend.y` Numeric, vertical legend position offset (-1 to 1). Default: -0.1.

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

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

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

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

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

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

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

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

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

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

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

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

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

Value

A plotly object.

Author(s)

Jacob Martin, Jared Andrews

Examples

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

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

---

piePlotApp *Create an example Modular piePlot Shiny Application*

Description

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

Usage

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

Arguments

data_list	An optional named list of 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.
defaults	A named list of input IDs and their default values to apply on startup. An entry may also be a shiny::reactive() or shiny::reactiveVal() to have the input follow the parent app's state; see setup_reactive_defaults().
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 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` *Input UI components for the piePlot module*

Description

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

Usage

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

Arguments

`id`	The ID for the Shiny module.
`data`	The data frame used for plot generation. Supply a summary table with one row per slice.
`defaults`	A named list of default values for the inputs. 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 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 *Output UI components for the piePlot module*

Description

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

Usage

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

Arguments

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

Value

A Shiny plotlyOutput for the piePlot

Author(s)

Jacob Martin, Jared Andrews

piePlotServer *Server logic for piePlot module*

Description

Server logic for piePlot module

Usage

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

Arguments

id	The ID for the Shiny module.
data	A reactive containing the data frame to plot. Provide a summarized table with columns for labels and aggregated values. 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 piePlot module.

Author(s)

Jacob Martin, Jared Andrews

See Also

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

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

Description

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

Usage

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

Arguments

`data_list`	An optional named list of data frames. If `NULL` (the default), 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 `shiny::reactive()` or `shiny::reactiveVal()` to have the input follow the parent app's state; see `setup_reactive_defaults()`.
`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_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 `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. 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)
- `axis.scientific` - Write numeric tick labels in scientific notation (UI: "Scientific Notation Ticks", default: FALSE; when off they never switch to it). See `apply_tick_notation()`

- `axis.padding.x` - Space between the data and the plot area's edges along X, as a fraction of the data's extent at each end (UI: "X Axis Padding", default: NA = the plot's own). See `apply_axis_padding()`
- `axis.padding.y` - Space between the data and the plot area's edges along Y, as a fraction of the data's extent at each end (UI: "Y Axis Padding", default: NA = the plot's own). See `apply_axis_padding()`
- `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

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

Value

A Shiny plotlyOutput for the AreaPlot

Author(s)

Jacob Martin

`plotthis_AreaPlotServer`

Server logic for AreaPlot module

Description

Server logic for AreaPlot module

Usage

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

Arguments

id	The ID for the Shiny module.
data	A reactive containing the data frame to plot. 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 AreaPlot module.

Author(s)

Jacob Martin, Jared Andrews

---

plotthis_BarPlotApp *Create an example Modular BarPlot Shiny Application*

Description

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

Usage

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

Arguments

data_list	An optional named list of data frames. If NULL (the default), 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 shiny::reactive() or shiny::reactiveVal() to have the input follow the parent app's state; see setup_reactive_defaults().
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_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

`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. 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:

- `x.order` - Order of the X values (UI: "X Order", drag and drop, default: the column's factor level order, else sorted). As a default, a character vector of levels of `x.data`; levels it leaves out follow in their default order. Not shown for an X with more than 20 values
- `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)
- `axis.scientific` - Write numeric tick labels in scientific notation (UI: "Scientific Notation Ticks", default: FALSE; when off they never switch to it). See `apply_tick_notation()`
- `axis.padding.x` - Space between the data and the plot area's edges along X, as a fraction of the data's extent at each end (UI: "X Axis Padding", default: NA = the plot's own). See `apply_axis_padding()`
- `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

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

Value

A Shiny plotlyOutput for the BarPlot

Author(s)

Jacob Martin, Jared Andrews

plotthis_BarPlotServer

Server logic for BarPlot module

Description

Server logic for BarPlot module

Usage

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

Arguments

id	The ID for the Shiny module.
data	A reactive containing the data frame to plot. 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 BarPlot module.

Author(s)

Jacob Martin, Jared Andrews

---

plotthis_BoxPlotApp *Create an example Modular BoxPlot Shiny Application*

Description

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

Usage

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

Arguments

data_list	An optional named list of data frames. If NULL (the default), 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 shiny::reactive() or shiny::reactiveVal() to have the input follow the parent app's state; see setup_reactive_defaults().
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_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 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. 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: ""). An expression evaluated once per x group over all the data, e.g. `mean(salary)`; a faceted plot shares the one order across its panels
- `y_max` - Maximum Y-axis value (UI: "Max Value of Y Axis", default: calculated)
- `y_min` - Minimum Y-axis value (UI: "Min Value of Y Axis", default: calculated)
- `add_point` - Add jitter points (UI: "Add Jitter Points", default: FALSE)
- `pt_size` - Point size (UI: "Point Size", default: 1)

- pt_alpha - Point transparency (UI: "Point Alpha", default: 1)
- jitter_width - Jitter width (UI: "Jitter Width", default: 0.3)
- pt_color - Point outline color (UI: "Point Outline Colour", default: "#000000")
- highlight - Highlight condition (UI: "Highlight", default: "")
- highlight_color - Highlight color (UI: "Highlight Colour", default: "#000000")
- highlight_size - Highlight size (UI: "Highlight Size", default: 1)
- highlight_alpha - Highlight transparency (UI: "Highlight Alpha", default: 1)
- facet_by - Faceting variable (UI: "Facet by", default: "")
- facet_scales - Facet scale behavior (UI: "Facet Scale", default: "fixed")
- facet_ncol - Number of facet columns (UI: "Columns", default: NULL)
- facet_nrow - Number of facet rows (UI: "Rows", default: NULL)
- facet_byrow - Facet ordering direction (UI: "Facet by Row", default: TRUE)
- palcolor - Custom color values (UI: palette picker, derived from palette)

Statistical annotation parameters

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

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

- `webgl` - Draw the points with WebGL once there are 5,000 or more, as set by the `VizModules.webgl.threshold` option (UI: "Plot with WebGL", default: TRUE)
- `boxplot.width` - Width of boxplot (UI: "Boxplot Width", default: 0.8)
- `x.order` - Order of the X groups (UI: "X Order", drag and drop, default: the column's factor level order, else sorted). As a default, a character vector of levels of `x.data`; levels it leaves out follow in their default order. Ignored while `sort_x` is set, and not shown for an X with more than 20 groups
- `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)
- `axis.scientific` - Write numeric tick labels in scientific notation (UI: "Scientific Notation Ticks", default: FALSE; when off they never switch to it). See `apply_tick_notation()`
- `axis.padding.x` - Space between the data and the plot area's edges along X, as a fraction of the data's extent at each end (UI: "X Axis Padding", default: NA = the plot's own). See `apply_axis_padding()`

- `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 shinyjqui::jqui_resizable() so it can be resized by dragging. Set to FALSE when embedding the output in a container that already provides resizing.

Value

A Shiny plotlyOutput for the boxPlot

Author(s)

Jacob Martin

plotthis_BoxPlotServer

Server logic for BoxPlot module

Description

Server logic for BoxPlot module

Usage

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

Arguments

id	The ID for the Shiny module.
data	A reactive containing the data frame to plot. 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 BoxPlot module.

Author(s)

Jacob Martin, Jared Andrews

---

plotthis_DensityPlotApp

Create an example Modular DensityPlot Shiny Application

Description

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

Usage

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

Arguments

data_list	An optional named list of data frames. If NULL (the default), 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 shiny::reactive() or shiny::reactiveVal() to have the input follow the parent app's state; see setup_reactive_defaults().
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_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 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. Nearly all parameters for plotthis::DensityPlot() can be set via these inputs, so see the help for that function for an exhaustive list.

Value

A Shiny tagList containing the UI elements

Plot parameters not implemented or with altered functionality

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

- xlab - X-axis label (plotly allows interactive editing)
- ylab - Y-axis label (plotly allows interactive editing)
- title - Plot title (plotly allows interactive editing)
- subtitle - Plot subtitle (not supported in plotly)
- legend.position - Legend positioning (plotly allows interactive repositioning)
- group_by_sep - Separator for group columns (not applicable in UI context)
- group_name - Group legend name (handled by plotly)
- xtrans - X-axis transformation (not implemented in UI)
- ytrans - Y-axis transformation (not implemented in UI)

- split_by - Split variable (returns a patchwork object, not supported in plotly), use facet_by instead
- split_by_sep - Only applies if split_by is used
- theme - ggplot2 theme (not applicable in plotly)
- theme_args - Theme arguments (not applicable in plotly)
- palette - Managed internally via the palette selection UI
- expand - Axis expansion (not implemented)
- seed - Random seed (not applicable)
- combine - Only applies if split_by is used
- nrow - Only applies if split_by is used
- ncol - Only applies if split_by is used
- byrow - Only applies if split_by is used
- axes - Only applies if split_by is used
- axis_titles - Only applies if split_by is used
- guides - Only applies if split_by is used
- design - Only applies if split_by is used
- palreverse - Reverse the color palette (not implemented)
- aspect.ratio - Aspect ratio control (handled by plotly layout)
- keep_empty - Keep empty factor levels (not implemented)
- keep_na - Keep NA values (not implemented)
- legend.direction - Managed position of legend however this can be handled via plotly

Plot parameters and defaults

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

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

Parameters controlling additional functionality

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

- `title.font.size` - Plot title font size (UI: "Title Size", default: 26)
- `title.font.family` - Font family for title text (UI: "Title Font", default: "Arial")
- `title.font.color` - Color for plot title (UI: "Title Color", default: "#000000")
- `axis.title.font.size` - Axis title font size (UI: "Axis Title Size", default: 18)
- `axis.title.font.color` - Axis title font color (UI: "Axis Title Color", default: "#000000")
- `axis.title.font.family` - Axis title font family (UI: "Axis Title Font", default: "Arial")
- `axis.showline` - Show axis border lines (UI: "Show axis lines", default: TRUE)
- `axis.mirror` - Mirror axis lines on opposite side (UI: "Mirror axis lines", default: TRUE)
- `show.grid.x` - Show X-axis major gridlines (UI: "Show X major gridlines", default: TRUE)
- `show.grid.y` - Show Y-axis major gridlines (UI: "Show Y major gridlines", default: TRUE)
- `axis.linecolor` - Color of axis lines (UI: "Axis line color", default: "black")
- `axis.linewidth` - Width of axis lines (UI: "Axis line width", default: 0.5)
- `axis.tickfont.size` - Size of tick labels (UI: "Tick label size", default: 12)
- `axis.tickfont.color` - Color of tick labels (UI: "Tick label color", default: "black")
- `axis.tickfont.family` - Font family for tick labels (UI: "Tick label font", default: "Arial")
- `axis.tickangle.x` - Rotation angle for X-axis tick labels (UI: "X-axis tick label angle", default: 0)
- `axis.tickangle.y` - Rotation angle for Y-axis tick labels (UI: "Y-axis tick label angle", default: 0)
- `axis.ticks` - Position of tick marks (UI: "Tick position", default: "outside")
- `axis.tickcolor` - Color of tick marks (UI: "Tick mark color", default: "black")
- `axis.ticklen` - Length of tick marks (UI: "Tick mark length", default: 5)
- `axis.tickwidth` - Width of tick marks (UI: "Tick mark width", default: 1)
- `axis.scientific` - Write numeric tick labels in scientific notation (UI: "Scientific Notation Ticks", default: FALSE; when off they never switch to it). See `apply_tick_notation()`
- `axis.padding.x` - Space between the data and the plot area's edges along X, as a fraction of the data's extent at each end (UI: "X Axis Padding", default: NA = the plot's own). See `apply_axis_padding()`
- `axis.padding.y` - Space between the data and the plot area's edges along Y, as a fraction of the data's extent at each end (UI: "Y Axis Padding", default: NA = the plot's own). See `apply_axis_padding()`
- `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

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

Value

A Shiny plotlyOutput for the DensityPlot

Author(s)

Jacob Martin

plotthis_DensityPlotServer

Density Plot Server Module

Description

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

Usage

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

Arguments

id	character unique ID for the shiny namespace.
data	reactive A reactive expression returning a data frame to be plotted. 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	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 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 DensityPlot module.

Author(s)

Jacob Martin, Jared Andrews

---

plotthis_DotPlotApp *Create an example Modular DotPlot Shiny Application*

Description

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

Usage

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

Arguments

data_list	An optional named list of data frames. If NULL (the default), 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 shiny::reactive() or shiny::reactiveVal() to have the input follow the parent app's state; see setup_reactive_defaults().

`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_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 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. 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)
- size.scale.min - size_by value drawn at the minimum dot size, where the size legend starts (UI: "Size Scale Min", default: NA = the column's minimum); smaller values are drawn at the minimum size. Shown while size_by is set
- size.scale.max - size_by value drawn at the maximum dot size, where the size legend ends (UI: "Size Scale Max", default: NA = the column's maximum); larger values are drawn at the maximum size. Shown while size_by is set
- 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)

Parameters controlling additional functionality

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

- webgl - Draw the points with WebGL once there are 5,000 or more, as set by the VizModules.webgl.threshold option (UI: "Plot with WebGL", default: TRUE)

Author(s)

Jacob Martin, Jared Andrews

See Also

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

Examples

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

---

plotthis_DotPlotOutputUI

Output UI components for the DotPlot module

Description

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

Usage

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

Arguments

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

Value

A Shiny plotlyOutput for the DotPlot

Author(s)

Jacob Martin, Jared Andrews

plotthis_DotPlotServer *Server logic for DotPlot module*

Description

Server logic for DotPlot module

Usage

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

Arguments

id	The ID for the Shiny module.
data	A reactive containing the data frame to plot. 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 DotPlot module.

Author(s)

Jacob Martin, Jared Andrews

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

Description

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

Usage

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

Arguments

`data_list`	An optional named list of data frames. If `NULL` (the default), 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 `shiny::reactive()` or `shiny::reactiveVal()` to have the input follow the parent app's state; see `setup_reactive_defaults()`.
`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_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

`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. Nearly all parameters for `plotthis::Histogram()` can be set via these inputs, so see the help for that function for an exhaustive list.

Value

A Shiny tagList containing the UI elements

Plot parameters not implemented or with altered functionality

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

- `xlab` - X-axis label (plotly allows interactive editing)
- `ylab` - Y-axis label (plotly allows interactive editing)
- `title` - Plot title (plotly allows interactive editing)
- `subtitle` - Plot subtitle (not supported in plotly)
- `legend.position` - Legend positioning (plotly allows interactive repositioning)
- `group_by_sep` - Separator for group columns (not applicable in UI context)
- `group_name` - Group legend name (handled by plotly)
- `xtrans` - X-axis transformation (not implemented in UI)
- `ytrans` - Y-axis transformation (not implemented in UI)
- `split_by` - Split variable (returns a patchwork object, not supported in plotly), use `facet_by` instead
- `split_by_sep` - Only applies if `split_by` is used
- `theme` - ggplot2 theme (not applicable in plotly)
- `theme_args` - Theme arguments (not applicable in plotly)
- `palette` - Managed internally via the palette selection UI
- `expand` - Axis expansion (not implemented)
- `seed` - Random seed (not applicable)
- `combine` - Only applies if `split_by` is used
- `nrow` - Only applies if `split_by` is used
- `ncol` - Only applies if `split_by` is used
- `byrow` - Only applies if `split_by` is used
- `axes` - Only applies if `split_by` is used
- `axis_titles` - Only applies if `split_by` is used
- `guides` - Only applies if `split_by` is used
- `design` - Only applies if `split_by` is used
- `palreverse` - Reverse the color palette (not implemented)
- `aspect.ratio` - Aspect ratio control (handled by plotly layout)
- `keep_empty` - Keep empty factor levels (not implemented)
- `keep_na` - Keep NA values (not implemented)
- `legend.direction` - Managed position of legend however this can be handled via plotly

Plot parameters and defaults

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

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

Parameters controlling additional functionality

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

- `title.font.size` - Plot title font size (UI: "Title Size", default: 26)
- `title.font.family` - Font family for title text (UI: "Title Font", default: "Arial")
- `title.font.color` - Color for plot title (UI: "Title Color", default: "#000000")
- `axis.title.font.size` - Axis title font size (UI: "Axis Title Size", default: 18)
- `axis.title.font.color` - Axis title font color (UI: "Axis Title Color", default: "#000000")
- `axis.title.font.family` - Axis title font family (UI: "Axis Title Font", default: "Arial")
- `axis.showline` - Show axis border lines (UI: "Show axis lines", default: TRUE)

- `axis.mirror` - Mirror axis lines on opposite side (UI: "Mirror axis lines", default: TRUE)
- `show.grid.x` - Show X-axis major gridlines (UI: "Show X major gridlines", default: TRUE)
- `show.grid.y` - Show Y-axis major gridlines (UI: "Show Y major gridlines", default: TRUE)
- `axis.linecolor` - Color of axis lines (UI: "Axis line color", default: "black")
- `axis.linewidth` - Width of axis lines (UI: "Axis line width", default: 0.5)
- `axis.tickfont.size` - Size of tick labels (UI: "Tick label size", default: 12)
- `axis.tickfont.color` - Color of tick labels (UI: "Tick label color", default: "black")
- `axis.tickfont.family` - Font family for tick labels (UI: "Tick label font", default: "Arial")
- `axis.tickangle.x` - Rotation angle for X-axis tick labels (UI: "X-axis tick label angle", default: 0)
- `axis.tickangle.y` - Rotation angle for Y-axis tick labels (UI: "Y-axis tick label angle", default: 0)
- `axis.ticks` - Position of tick marks (UI: "Tick position", default: "outside")
- `axis.tickcolor` - Color of tick marks (UI: "Tick mark color", default: "black")
- `axis.ticklen` - Length of tick marks (UI: "Tick mark length", default: 5)
- `axis.tickwidth` - Width of tick marks (UI: "Tick mark width", default: 1)
- `axis.scientific` - Write numeric tick labels in scientific notation (UI: "Scientific Notation Ticks", default: FALSE; when off they never switch to it). See `apply_tick_notation()`
- `axis.padding.x` - Space between the data and the plot area's edges along X, as a fraction of the data's extent at each end (UI: "X Axis Padding", default: NA = the plot's own). See `apply_axis_padding()`
- `axis.padding.y` - Space between the data and the plot area's edges along Y, as a fraction of the data's extent at each end (UI: "Y Axis Padding", default: NA = the plot's own). See `apply_axis_padding()`
- `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 `shinyjqui::jqui_resizable()` so it can be resized by dragging. Set to `FALSE` when embedding the output in a container that already provides resizing.

Value

A Shiny plotlyOutput for the histogramPlot

Author(s)

Jacob Martin

`plotthis_HistogramServer` *Histogram Plot Server Module*

Description

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

Usage

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

Arguments

`id`	`character` unique ID for the shiny namespace.
`data`	`reactive` A reactive expression returning a data frame to be plotted. 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`	`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 `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 Histogram module.

Author(s)

Jacob Martin, Jared Andrews

plotthis_SplitBarPlotApp *Create an example Modular SplitBarPlot Shiny Application*

Description

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

Usage

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

Arguments

`data_list`	An optional named list of data frames. If `NULL` (the default), 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 `shiny::reactive()` or `shiny::reactiveVal()` to have the input follow the parent app's state; see `setup_reactive_defaults()`.
`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_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

`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. 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)
- `axis.scientific` - Write numeric tick labels in scientific notation (UI: "Scientific Notation Ticks", default: FALSE; when off they never switch to it). See `apply_tick_notation()`
- `axis.padding.y` - Space between the data and the plot area's edges along Y, as a fraction of the data's extent at each end (UI: "Y Axis Padding", default: NA = the plot's own). See `apply_axis_padding()`
- `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

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

Value

A Shiny plotlyOutput for the SplitBarPlot

Author(s)

Jacob Martin

plotthis_SplitBarPlotServer

Server logic for SplitBarPlot module

Description

Server logic for SplitBarPlot module

Usage

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

Arguments

id	The ID for the Shiny module.
data	A reactive containing the data frame to plot. 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 SplitBarPlot module.

Author(s)

Jacob Martin, Jared Andrews

See Also

plotthis::SplitBarPlot(), organize_inputs(), plotthis_SplitBarPlotInputsUI(), plotthis_SplitBarPlotOutp
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

df	A data frame containing the data to plot. For a single trace, provide columns for categories (theta) and values (r). For multiple traces, include a grouping column. The function automatically closes the radar polygon by adding the first point to the end.
theta	Character, name of the column to use for the angular categories (axes).
r	Character, name of the column to use for the radial values.
group	Optional character, name of the column to use for grouping multiple traces. If NULL, a single trace is plotted. Default: NULL.
colors	Optional character vector of hex colors for the traces. If named, values are matched to the group values; otherwise colours are recycled.
palette	Optional character vector of fallback colors used when `colors` is not supplied or missing values are present.
fill	Logical or character, whether to fill the area under each trace. Use "toself" to fill to the first point, or FALSE for no fill. Default: "toself".
line.width	Numeric, width of the trace lines in pixels. Default: 2.

`line.dash` Character, line dash style. Options: "solid", "dot", "dash", "longdash", "dashdot", "longdashdot". Default: "solid".

`marker.size` Numeric, size of the markers on the trace. Default: 5.

`marker.symbol` Character, marker symbol. Options: "circle", "square", "diamond", "cross", "x", "triangle-up", etc. Default: "circle".

`opacity` Numeric, opacity of the traces (0-1). Default: 0.6.

`radial.visible` Logical, whether to show the radial axis. Default: TRUE.

`radial.range` Optional numeric vector of length 2 specifying the range of the radial axis (e.g., c(0, 100)). If NULL, automatically determined. Default: NULL.

`radial.showline`
Logical, whether to show the radial axis line. Default: TRUE.

`radial.linecolor`
Character, hex color for the radial axis line. Default: "#444444".

`radial.gridcolor`
Character, hex color for the radial grid lines. Default: "#EEEEEE".

`angular.direction`
Character, direction of angular axis. Options: "clockwise" or "counterclockwise". Default: "clockwise".

`angular.rotation`
Numeric, rotation angle for the angular axis in degrees. Default: 90.

`angular.gridcolor`
Character, hex color for the angular grid lines. Default: "#EEEEEE".

`show.legend` Logical, whether to display the legend. Default: TRUE.

`legend.orientation`
Character, legend orientation. Options: "h" (horizontal) or "v" (vertical). Default: "h".

`legend.x` Numeric, horizontal legend position offset (0-1). Default: 0.5.

`legend.y` Numeric, vertical legend position offset (-1 to 1). Default: -0.1.

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

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

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

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

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

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

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

`title.x` Numeric, horizontal position for the plot title (0-1). Default: 0.5.

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

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

Value

A plotly object.

Author(s)

Jacob Martin

Examples

```
# Single trace radar chart
# Note: Polygon is automatically closed by the function
skills <- data.frame(
    category = c("Speed", "Strength", "Defense", "Stamina"),
    value = c(8, 6, 7, 9)
)

radarPlot(
    df = skills,
    theta = "category",
    r = "value",
    title.text = "Player Stats"
)

# Multiple trace radar chart
# Note: Polygon is automatically closed for each trace
team_stats <- data.frame(
    category = rep(c("Speed", "Strength", "Defense", "Stamina"), 2),
    value = c(8, 6, 7, 9, 5, 9, 8, 6),
    player = rep(c("Player A", "Player B"), each = 4)
)

radarPlot(
    df = team_stats,
    theta = "category",
    r = "value",
    group = "player",
    title.text = "Team Comparison"
)
```

---

radarPlotApp *Create an example Modular radarPlot Shiny Application*

Description

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

Usage

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

Arguments

data_list	An optional named list of data frames. If NULL (the default), 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.
defaults	A named list of input IDs and their default values to apply on startup. An entry may also be a shiny::reactive() or shiny::reactiveVal() to have the input follow the parent app's state; see setup_reactive_defaults().
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_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 *Output UI components for the radarPlot module*

Description

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

Usage

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

Arguments

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

Value

A Shiny plotlyOutput for the radarPlot

Author(s)

Jared Andrews

radarPlotServer *Server logic for radarPlot module*

Description

Server logic for radarPlot module

Usage

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

Arguments

id	The ID for the Shiny module.
data	A reactive containing the data frame to plot. Provide data with columns for categories (theta) and values (r). For multiple traces, include a grouping column. 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 radarPlot module.

Author(s)

Jacob Martin

See Also

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

---

recycle_line_style *Recycle style vector to match line count*

Description

Extends or truncates a style vector to match the number of lines being drawn. If the input length doesn't match the target length, uses only the first value for all lines (as documented behavior).

Usage

```
recycle_line_style(values, n, default)
```

Arguments

values	Vector. Style values (colors, widths, etc.) to recycle.
n	Integer. Target length (number of lines).
default	The default value to use if values is NULL or empty.

Value

A vector of length n with recycled values.

Author(s)

Jared Andrews

Examples

```
recycle_line_style(c("red", "blue"), 2, "black")
recycle_line_style(NULL, 3, "black")
```

`register_model_backend` *Register a model backend*

Description

Add or replace a model backend in the registry. Once registered, the backend name appears in the model-type dropdown and the custom-model-lines pipeline dispatches through it automatically.

Usage

```
register_model_backend(name, backend)
```

Arguments

`name`	Character string. The name that will appear in the model-type dropdown (e.g. `"drm"`, `"gam"`).
`backend`	A named list with elements `fit`, `predict`, and `validate_classes` as described above.

Details

A backend is a named list with three required elements:

`fit` A function with signature `function(formula, data, ...)` that returns a fitted model object. Extra UI fields from the `multiDynamicInput()` row are forwarded as `...`.

`predict` A function with signature `function(model, newdata)` that returns a numeric vector of predicted y-values, one per row of `newdata`.

`validate_classes` Character vector of class names. After fitting, the pipeline checks `inherits(model, validate_classes)` and rejects the model if it fails.

Value

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

Author(s)

Jacob Martin

Examples

```
# Register a custom backend for dose-response curves (requires drc)
if (requireNamespace("drc", quietly = TRUE)) {
}
```

`require_data_frame` *Require a data frame from a module's data reactive*

Description

Wraps the `data` reactive every module server receives so that downstream readers always see a data frame. A `NULL` value (which a parent app can emit briefly while switching datasets) becomes a silent `shiny::req()` skip rather than an error cascade through every observer, and anything else is coerced with `as.data.frame()` before use.

Usage

```
require_data_frame(data)
```

Arguments

`data`	A `reactive` returning a data frame or an object coercible to one.

Value

A `reactive` returning a data frame with at least one column.

Author(s)

Jared Andrews

Examples

```
## Not run:
# First line of a module server, before anything reads `data()`:
data <- require_data_frame(data)

## End(Not run)
```

`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 *Reset uniform axes inputs to defaults*

Description

Resets all inputs created by `uniform_axes_inputs_ui()` to their default values. Call inside an `observeEvent(input$reset, ...)` block to avoid duplicating 20 `updateXxxInput` calls in every module server.

Usage

```
reset_axes_inputs(session, defaults = NULL)
```

Arguments

`session`	The Shiny session object (from `moduleServer`).
`defaults`	A named list of default values to reset to, or NULL to use hardcoded fallbacks. Typically the same list passed to the UI function.

Value

Called for side effects; returns `invisible(NULL)`.

Author(s)

Jared Andrews

Examples

```
## Not run:
# Call inside a module server's observeEvent(input$reset, ...) block:
reset_axes_inputs(session, defaults)

## End(Not run)
```

reset_axis_title_text *Drop persisted axis-title text edits*

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 setup_manual_edits().
keys	Character vector of axis annotation keys to clear text for. Defaults to both axis titles, c("axis:x", "axis:y"). The #<occurrence> suffix added by .annotation_edit_keys() 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_colorscale_inputs

Reset uniform Color Scale inputs to defaults

Description

Resets the colour scale inputs created by uniform_colorscale_inputs_ui() back to their default values. The picker goes back to the colours and preset it was built with, so the same defaults must reach both.

Usage

```
reset_colorscale_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_colorscale_inputs(session, defaults)

## End(Not run)
```

---

reset_group_colors *Restore a group color picker to its default mapping*

Description

Used by module Reset buttons. When `defaults` supplies a mapping for `inputId` the picker is set back to it; otherwise the widget resets itself to the stock palette it was built with.

Usage

```
reset_group_colors(session, inputId, defaults, groups, default_palette = NULL)
```

Arguments

session	The Shiny `session` object from inside `moduleServer()`.
inputId	Character string — the picker's id, without namespacing.
defaults	A named list of default values, or `NULL`.
groups	A character vector of the group levels currently in play.
default_palette	A character vector of fallback colors.

Value

Invisibly `NULL`; called for its side effect.

Author(s)

Jared Andrews

See Also

default_group_colors(), updateMultiColorPicker(), setup_group_colors()

Examples

```
## Not run:
# Inside a module's observeEvent(input$reset, ...):
reset_group_colors(session, "palette.colours", defaults, palette_groups(), default_palette_values)

## End(Not run)
```

---

reset_legend_inputs *Reset uniform Legend inputs to defaults*

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` *Reset uniform lines inputs to defaults*

Description

Resets all inputs created by `uniform_lines_inputs_ui()` to their default values. Call inside an `observeEvent(input$reset, ...)` block to avoid duplicating line-reset boilerplate in every module server.

Usage

```
reset_lines_inputs(session, include.fit.lines = FALSE, defaults = NULL)
```

Arguments

`session`	The Shiny session object (from `moduleServer`).
`include.fit.lines`	Logical; if TRUE, also resets the fit-line inputs (best.fit, line.best.smoothness, line.best.colour, linear.model, model.legend.show, custom.model.enable, custom.models). Default is FALSE.
`defaults`	A named list of default values to reset to, or NULL to use hardcoded fallbacks. Typically the same list passed to the UI function.

Value

Called for side effects; returns `invisible(NULL)`.

Author(s)

Jared Andrews

Examples

```
## Not run:
# Call inside a module server's observeEvent(input$reset, ...) block:
reset_lines_inputs(session, defaults = defaults)

## End(Not run)
```

`reset_manual_edits` *Discard every captured manual layout edit*

Description

Used by module Reset buttons, so a dragged legend, annotation, axis title or colorbar goes back to where the rebuilt figure puts it. The store's fields are reactive values, so they are cleared in place; reassigning the list returned by `setup_manual_edits()` would only change a local copy.

Usage

```
reset_manual_edits(store)
```

Arguments

`store`	The list returned by `setup_manual_edits()`.

Value

Invisibly `NULL`; called for its side effect.

Author(s)

Jared Andrews

See Also

`setup_manual_edits()`, `finalize_manual_edits()`

Examples

```
## Not run:
# Inside a module's observeEvent(input$reset, ...):
manual_edits <- setup_manual_edits(input, session, plot_source)
reset_manual_edits(manual_edits)

## End(Not run)
```

reset_plotly_inputs *Reset uniform Plotly inputs to defaults*

Description

Resets all inputs created by `uniform_plotly_inputs_ui()` to their default values. Call inside an `observeEvent(input$reset, ...)` block to avoid duplicating Plotly-reset boilerplate in every module server.

Usage

```
reset_plotly_inputs(session, defaults = NULL)
```

Arguments

`session`	The Shiny session object (from `moduleServer`).
`defaults`	A named list of default values to reset to, or NULL to use hardcoded fallbacks. Typically the same list passed to the UI function.

Value

Called for side effects; returns `invisible(NULL)`.

Author(s)

Jared Andrews

Examples

```
## Not run:
# Call inside a module server's observeEvent(input$reset, ...) block:
reset_plotly_inputs(session, defaults)

## End(Not run)
```

reset_stats_inputs *Reset uniform stats inputs to defaults*

Description

Resets all inputs created by `uniform_stats_inputs_ui()` to their default values. Call inside an `observeEvent(input$reset, ...)` block.

Usage

```
reset_stats_inputs(session, defaults = NULL, pair_strings = NULL)
```

Arguments

session	The Shiny session object (from moduleServer).
defaults	A named list of default values to reset to, or NULL to use hardcoded fallbacks. Typically the same list passed to the UI function.
pair_strings	Character vector of the comparisons currently on offer in the Comparisons selector. Those named by defaults$stat.pairs are reselected (see default_stat_pairs()); NULL clears the selection.

Value

Called for side effects; returns invisible(NULL).

Author(s)

Jared Andrews

Examples

```
## Not run:
# Inside a module server:
observeEvent(input$reset, {
    reset_stats_inputs(session, defaults, pair_strings = pair_choices())
})

## End(Not run)
```

---

resolve_colorscale *Resolve a continuous colour scale for plotly*

Description

Turns colour stops and optional limits into the colorscale, cmin and cmax of a plotly coloraxis, or of a trace once cmin and cmax are renamed to its zmin and zmax. The stops are spaced evenly from min to mid and from mid to max, so a middle stop sits exactly at mid and a midpoint off centre stretches one side of the scale rather than moving its ends.

Usage

```
resolve_colorscale(
  colors = NULL,
  min = NA,
  mid = NA,
  max = NA,
  reverse = FALSE,
  data.range = c(0, 1)
)
```

Arguments

colors	Character vector of colours, lowest value first. A named vector, as a multiColorPicker() reports, is taken in order. NULL uses the first preset of uniform_colorscale_inputs_ui().
min, mid, max	Numeric limits and midpoint. NA or NULL (a cleared numeric input) falls back to the ends of data.range and the centre of the limits. A mid outside the limits is held just inside them.
reverse	Logical; run the colours from high to low.
data.range	The values plotted, or their range, used for a blank limit. Non-finite values are ignored. With none left, the limits are 0 and 1.

Details

These are the values of the inputs from uniform_colorscale_inputs_ui(). A scale set on layout.coloraxis rather than on the trace can be changed without redrawing the plot: it is a layout-only update, as a styling step should be.

Value

A list with colorscale (a list of list(position, colour) stops running from 0 to 1), cmin, and cmax. cmin is always below cmax: limits given in the wrong order are swapped, and equal limits (constant data) are widened by 0.5 either way.

Author(s)

Jared Andrews

See Also

uniform_colorscale_inputs_ui(), reset_colorscale_inputs()

Examples

```
# Betas, with the middle colour at 0.3.
scale <- resolve_colorscale(c("blue", "white", "red"), min = 0, mid = 0.3, max = 1)
scale$colorscale

fig <- plotly::plot_ly(z = volcano, type = "heatmap", coloraxis = "coloraxis")
plotly::layout(fig, coloraxis = scale)
```

---

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

`data`	A data frame, or a character vector of the column names in the order they are passed to `DT::datatable()`.
`columns`	`NULL`, a character vector of column names, or a numeric vector of one-based column positions to resolve.
`rownames`	Logical. Whether the table is drawn with a row-names column (the `rownames` argument of `DT::datatable()`). 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

`n_facets`	Integer, number of facet panels.
`facet.nrow`	Optional integer, user-requested number of rows.
`facet.ncol`	Optional integer, user-requested number of columns.

Details

Resolution rules:

- Both `NULL`/`NA`: returns 1 (single row, preserves legacy behaviour).
- Only `facet.nrow` supplied: returns that value.
- Only `facet.ncol` supplied: returns `ceiling(n_facets / facet.ncol)`.
- Both supplied: `facet.nrow` wins.

The result is clamped to the range `[1, n_facets]`.

Value

A positive integer giving the number of rows for `plotly::subplot`.

Author(s)

Jared Andrews

Examples

```
resolve_facet_layout(6, facet.nrow = 2)
resolve_facet_layout(6, facet.ncol = 3)
```

resolve_facet_sharing *Resolve facet axis sharing from facet.scales*

Description

Converts a `facet.scales` string (one of `"fixed"`, `"free"`, `"free_x"`, `"free_y"`) into the `shareX` / `shareY` logical values expected by `plotly::subplot`.

Usage

```
resolve_facet_sharing(facet.scales = "fixed")
```

Arguments

`facet.scales`	Character, one of `"fixed"` (default), `"free"`, `"free_x"`, or `"free_y"`.

Value

A named list with logical elements `shareX` and `shareY`.

Author(s)

Jared Andrews

Examples

```
resolve_facet_sharing("fixed")
resolve_facet_sharing("free_x")
```

resolve_palette *Resolve a color palette for plot groups*

Description

Maps groups to colors using selected colors or a default palette. Handles named color vectors by matching to group names, fills in missing colors with fallback values, and ensures the output vector is named and matches group length.

Usage

```
resolve_palette(
  groups,
  selected_colors = NULL,
  default_palette = NULL,
  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"), "#CCCCCC", 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` *Safely resolve an adjustment function name to an actual function*

Description

Validates that the provided function name is in the allowed list before converting it to a function reference. Returns `NULL` for empty strings or unrecognized names.

Usage

```
safe_resolve_adj_fxn(fn_name)
```

Arguments

`fn_name`	Character string — name of the adjustment function. Currently allowed values: `"log2"`, `"log"`, `"log10"`, `"neg_log10"`, `"log1p"`, `"as.factor"`, `"abs"`, `"sqrt"`.

Details

Use this instead of `eval(str2expression())` **when resolving function names from user input** (e.g., a dropdown that selects a transformation like `"log2"` or `"sqrt"`).

Value

The corresponding function, or `NULL` if `fn_name` is empty or not in the allowed list.

Author(s)

Jared Andrews

Examples

```
safe_resolve_adj_fxn("log2") # returns log2
safe_resolve_adj_fxn("") # NULL
safe_resolve_adj_fxn("system") # warning + NULL
```

setup_auto_update_logic *Set up auto-update/isolate logic for reactive contexts*

Description

A helper function that encapsulates the common pattern of handling auto-update functionality in module servers. When auto-update is disabled, it adds a dependency on the update button. Returns a wrapper function that either isolates reactive expressions or passes them through unchanged.

Usage

```
setup_auto_update_logic(input, params = NULL, track = NULL)
```

Arguments

input	The Shiny input object from the module server, should have both auto.update (boolean) and update (button) inputs.
params	Optional reactive-defaults store from setup_reactive_defaults(), or NULL. When supplied, an input$<key> read whose key is backed by the store resolves from the store instead of the client input, so a parameter driven by the parent app updates in a single render.
track	Optional function, or NULL. When supplied, it is called with the id of every input$<key> read through the returned wrapper, so a caller can tell which inputs an expression depends on (the modules' cached plot step keys its cache on them).

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` and `track` both `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). Otherwise it is a wrapper with the same isolation semantics that additionally redirects store-backed `input$<key>` reads and reports the ids read to `track`.

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

`input`	The Shiny `input` object from inside `moduleServer()`.
`session`	The Shiny `session` object from inside `moduleServer()`.
`min_key, max_key`	Character strings — the limit controls' input ids, without namespacing.
`headroom`	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.
`params`	Optional reactive-defaults store from `setup_reactive_defaults()`, 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 *Track the group-to-color mapping a plot should draw with*

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 moduleServer().
key	Character string — the picker's input id, without namespacing, e.g. "palette.colours".
groups	A reactive() 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 setup_reactive_defaults(), or NULL. When key is backed by the store, the mapping follows it rather than the client input, matching what setup_auto_update_logic() would have done for a direct input$<key> 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_level_order` *Keep a level-order control in step with its column*

Description

The server half of `level_order_input()`. Refills the control with the levels of the selected column whenever the data or the column changes, holds the order the plot should use, and restores `defaults[[inputId]]` when the module's Reset button (`input$reset`) is pressed.

Usage

```
setup_level_order(
  input,
  session,
  data,
  column,
  defaults = NULL,
  inputId = "x.order",
  max.levels = 20
)
```

Arguments

`input, session`	The module's `input` and `session`.
`data`	A reactive holding the module's data frame.

column	A reactive (or function) giving the name of the column ordered, e.g. reactive(input$x.data).
defaults	The module's named list of defaults, or NULL.
inputId	The un-namespaced ID of the level_order_input().
max.levels	The most levels the control offers; see level_order_choices().

Details

A new column starts from defaults[[inputId]] (levels it does not have are ignored), and new data for the same column keeps the order the user dragged the levels into. A column the control cannot order - numeric, or with more than max.levels levels - empties the control, which hides it.

The returned store, not input[[inputId]], is what the plot should read. It changes only when the order to draw does, so neither the control reporting its initial value nor its echo of an update costs an extra render. A defaults[[inputId]] given as a shiny::reactive() is followed as the parent's state changes, as with setup_reactive_defaults().

Value

A shiny::reactiveVal() holding the levels in the order to draw, or NULL for the default order. Hand it to apply_level_order().

Author(s)

Jared Andrews

See Also

level_order_input(), apply_level_order()

Examples

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

    ui <- fluidPage(
        selectInput("x", "X", c("Species", "Petal.Size")),
        level_order_input("x.order", "X Order"),
        plotOutput("plot")
    )

    server <- function(input, output, session) {
        data <- reactive({
            df <- example_iris
            df$Petal.Size <- ifelse(df$Petal.Length > 4, "Large", "Small")
            df
        })
        x.order <- setup_level_order(input, session, data, reactive(input$x))

        output$plot <- renderPlot({
            df <- apply_level_order(data(), input$x, x.order())
            boxplot(df$Sepal.Length ~ df[[input$x]], xlab = input$x)
```

```
        })
    }

    shinyApp(ui, server)
}
```

`setup_manual_edits` *Set up persistent manual plot layout edits across re-renders*

Description

VizModules plots are fully rebuilt on every input change, which would normally discard any layout tweaks a user made by hand: dragging the legend, moving or editing annotations, repositioning the (draggable) axis titles, or sliding a continuous-colour legend (colorbar). `setup_manual_edits()` together with its companion `finalize_manual_edits()` add this persistence to a plotting module with two short lines of wiring, so manually repositioned elements survive subsequent rebuilds.

Usage

```
setup_manual_edits(input, session, plot_source)
```

Arguments

`input`	The module's `input` object.
`session`	The module's `session` object.
`plot_source`	Character scalar. A unique plotly event source id for this module instance, typically `session$ns("<plot-type>")` (e.g. `session$ns("scatter")`). It scopes the captured `plotly_relayout` events to this plot and **must match** the `plot_source` later passed to `finalize_manual_edits()`.

Details

Call `setup_manual_edits()` **once**, near the top of your module's `shiny::moduleServer()` body, so the observers it registers belong to the module's reactive domain. It creates a reactive store and registers the observers that capture `plotly_relayout` events (legend, annotation, and axis-title drags) plus the JavaScript-forwarded colorbar drag. Then, inside your `plotly::renderPlotly()`, pass the freshly built figure through `finalize_manual_edits()` before returning it.

Value

A list (the "edit store") to hand to `finalize_manual_edits()`, with components:

`edits` A `shiny::reactiveValues()` holding the captured `legend`, `annotations`, and `colorbar` edits.

`rendered_fig` A `shiny::reactiveVal()` holding the most recently rendered figure, used to map relayout annotation indices to stable, rebuild-proof keys.

`sent` An environment recording the figure last sent to the browser, so a render that changes only its layout sends just that.

`refresh` A `shiny::reactiveVal()` the browser bumps, through the `plot.refresh` input, when it cannot apply a layout-only update, which makes the next render send the whole figure.

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

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

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, and `keys`, the input ids the store backs.

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)
```

---

split_color_shape_legend

Give a plot's colour and shape their own legend sections

Description

A ggplot mapping both colour and shape to discrete columns has two legends, but plotly::ggplotly() merges them into one with an entry per combination, named "(colour,shape)" under a "colour
shape" title. This draws them as ggplot2 does instead: one legend in two titled sections, an entry per colour level and one per shape level (in black, as ggplot2 draws shape keys). Clicking an entry hides or shows every trace of its level, and double-clicking one shows only its level (inst/src/legendSections.js).

Usage

```
split_color_shape_legend(
  fig,
  color.levels,
  shape.levels,
  color.title = NULL,
  shape.title = NULL,
  marker.size = 8
)
```

Arguments

fig	A built plotly figure.
color.levels, shape.levels	The colour and shape levels as plotted.
color.title, shape.title	The sections' titles. NULL takes each from the legend title ggplotly wrote.
marker.size	The size, in px, of the entries' markers.

Details

The point traces drawn for each combination are kept, out of the legend and tagged with their levels (in `meta`). A trace drawn for a colour alone, such as a yPlot's boxes (`"(colour,1)"`, ggplot's placeholder shape for them), becomes its level's colour entry; a level without one gets a data-less legend-only trace, as does every shape level. Those are appended after the existing traces, so no trace's index (a selection's curve number) moves. Traces grouped under a colour level by name (the scatter plot's marginals) are hidden and shown with it.

Value

The figure, unchanged when no trace is drawn for a combination of a colour and a shape level.

Author(s)

Jared Andrews

See Also

`apply_legend_styling()`, whose title fonts style the section titles.

Examples

```
d <- transform(mtcars, am = factor(am), vs = factor(vs))
p <- ggplot2::ggplot(d, ggplot2::aes(wt, mpg, colour = am, shape = vs)) +
    ggplot2::geom_point()
fig <- plotly::plotly_build(plotly::ggplotly(p))
fig <- split_color_shape_legend(fig, levels(d$am), levels(d$vs))
```

`stat_bracket_headroom` *Bracket headroom for a module, read straight off its Stats tab*

Description

Wraps `stat_bracket_y_max()` with the geometry inputs every module's Stats tab carries, so the three modules that draw significance brackets share one call rather than three copies of the same argument list. Missing inputs fall back to the tab's own defaults, which matters while the tab has yet to be rendered.

Usage

```
stat_bracket_headroom(
  df,
  x,
  y,
  group.by = NULL,
  facet.by = NULL,
  per.facet = TRUE,
  input,
```

```
  dodge.width = 1
)
```

Arguments

df, x, y, group.by, facet.by, per.facet, dodge.width	Passed to stat_bracket_y_max().
input	The Shiny input object from inside moduleServer().

Value

A single number, or NULL when no brackets would be drawn.

Author(s)

Jared Andrews

See Also

stat_bracket_y_max(), uniform_stats_inputs_ui(), setup_axis_range()

Examples

```
## Not run:
# As the `headroom` of setup_axis_range(), inside a module server:
headroom <- function() {
    if (!isTRUE(input$stats.enabled)) return(NULL)
    stat_bracket_headroom(data(), x = input$x.data, y = input$y.data, input = input)
}

## End(Not run)
```

---

stat_bracket_y_max *Y-axis top needed to draw statistical annotation brackets in full*

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

df	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()`).
x	Character; x-axis column name.
y	Character; y-axis column name(s). Several may be given, in which case the data range spans all of them.
pairs	List of length-2 character vectors, or `NULL` for all pairwise combinations.
group.by	Character or `NULL`; nested grouping column.
facet.by	Character or `NULL`; faceting column.
per.facet	Logical; whether tests are run within each facet.
step.increase	Numeric; fraction of the y-range between successive bracket levels. Default 0.06.
text.bump	Numeric; fraction of the y-range between a bracket and its label. Default 0.04.
bracket.inset	Numeric; endpoint inset, which affects how tightly brackets pack onto a level. Default 0.025.
hide.ns	Logical; whether non-significant brackets are dropped before drawing. When `TRUE` the tests are run so only the surviving comparisons are counted. Default `FALSE`.
sig.threshold	Numeric; significance cutoff used with `hide.ns`.
test, p.adjust.method, paired	Passed to `compute_pairwise_stats()`, and used only when `hide.ns` is `TRUE`. Match them to the render's settings, or the wrong comparisons are counted.
dodge.width	Numeric; width the `group.by` 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` *Parse and validate linetype string to a vector*

Description

Parses a comma-separated string of linetypes and validates each element. Invalid linetypes are replaced with "solid" and a warning is issued.

Usage

```
string_to_linetypes(x)
```

Arguments

x	A string of linetypes delimited by commas, e.g. "solid, dashed, dotted". Valid linetypes are: "solid", "dashed", "dotted", "dotdash", "longdash", "twodash".

Value

A character vector of validated linetypes. If the input is "" or NULL, returns "solid".

Author(s)

Jared Andrews

Examples

```
string_to_linetypes("solid, dashed, dotted")
string_to_linetypes("")
```

---

subplot_spacing_defaults

Resolve the subplot spacing defaults

Description

Shared by uniform_subplot_spacing_inputs_ui() and reset_plotly_inputs() so the controls start and reset to the same values. A subplot.margin entry sets both directions at once (linePlot uses it for its tighter default); subplot.margin.x/subplot.margin.y override it per direction.

Usage

```
subplot_spacing_defaults(defaults)
```

Arguments

defaults	A named list of default values, or NULL.

Value

A list with numeric x and y.

Author(s)

Jared Andrews

Examples

```
subplot_spacing_defaults(NULL)
subplot_spacing_defaults(list(subplot.margin = 0.05, subplot.margin.y = 0.2))
```

toggle_facet_title_inputs *Match the title inputs to whether a plot is faceted*

Description

A faceted plot titles each panel and has no main title (add_plot_config() disables it, as its placeholder sits over the panel titles), so the facet title inputs are shown and the main title inputs hidden. An unfaceted plot is the reverse.

Usage

```
toggle_facet_title_inputs(session, faceted, extra = NULL, hidden = NULL)
```

Arguments

session	The module's Shiny session.
faceted	Logical, whether the plot is currently faceted.
extra	Additional facet-only input IDs shown and hidden with the facet title inputs.
hidden	Input IDs the app hid via hide.inputs, which are never shown again here.

Value

NULL, invisibly. Called for its side effect.

Author(s)

Jared Andrews

See Also

main_title_input_ids, hide_input(), show_input(), add_plot_config()

Examples

```
## Not run:
# Inside a module server, whenever the facet choice changes:
observe({
  toggle_facet_title_inputs(session, faceted = nz_value(input$facet.by), hidden = hide.inputs)
})

## End(Not run)
```

`toggle_input_cell` *Hide or show the grid cell wrapping a module input*

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 `session$ns`).
`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

ns	A namespace function, typically created by NS(id).
defaults	A named list of default values for the inputs.
choices	Character vector of column names offered by "Annotate By".
annotate.note	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,
  include.padding = c("x", "y")
)
```

Arguments

ns	A namespace function, typically created by NS(id).
defaults	A named list of default values for the inputs.
include.rotate	Logical; whether to include the "Rotate" input for swapping x and y axes (e.g., horizontal bar plots). Default is FALSE.
include.flip	Logical; whether to include the "Flip" input for flipping the axis. Default is FALSE.
include.padding	Which of "x" and "y" get an axis padding input (axis.padding.x, axis.padding.y; see apply_axis_padding()). Leave out an axis that has its own limit inputs. Default is both.

Details

Alongside the fonts, lines, gridlines, and ticks, the inputs include axis.scientific, which writes numeric tick labels in scientific notation (off by default, so they never switch to it on their own; see apply_tick_notation()), and axis.padding.x/axis.padding.y, the space between the data and the edges of the plot area as a fraction of the data's extent at each end (blank keeps the plot's own padding).

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)
# A plot with its own y-axis limit inputs: x padding only.
uniform_axes_inputs_ui(ns, include.padding = "x")
```

---

uniform_colorscale_inputs_ui

Generate uniform Color Scale input UI

Description

Creates a standardized tagList of the inputs that set a continuous colour scale, for a plotly figure with a colorbar such as a heatmap:

Usage

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

Arguments

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

Details

- `colorscale.colors` - The five colour stops, lowest value first, as a `multiColorPicker()` (UI: "Color Scale", default: the "RdYlBu" preset). In `defaults`, five colours, named by stop or in order
- `colorscale.palette` - The preset selected, one of "RdYlBu", "Jet", "Blue-White-Red", "Viridis", "Magma", or "Cividis". Set in `defaults` only
- `colorscale.reverse` - Run the colours from high to low (UI: "Reverse Colors", default: FALSE)
- `colorscale.min` - Value at the lowest colour (UI: "Scale Min", default: NA, the data's minimum)
- `colorscale.mid` - Value at the middle colour (UI: "Scale Mid", default: NA, the centre of the limits)
- `colorscale.max` - Value at the highest colour (UI: "Scale Max", default: NA, the data's maximum)
- `colorbar.title` - Title of the colorbar (UI: "Colorbar Title", default: "Value")

Pass the inputs to `resolve_colorscale()`, and restore them with `reset_colorscale_inputs()`. The picker's five stops are fixed, so it needs no `renderUI()` and no `setup_group_colors()` store: read it directly, as `isolate_fn(input$colorscale.colors)`.

The colorbar's fonts and visibility follow the Legend tab, through `apply_legend_styling()`.

Value

A `tagList` containing the colour scale input UI elements.

Author(s)

Jared Andrews

See Also

`resolve_colorscale()`, `reset_colorscale_inputs()`

Examples

```
ns <- shiny::NS("plot1")
uniform_colorscale_inputs_ui(ns)
uniform_colorscale_inputs_ui(ns, defaults = list(
    colorscale.palette = "Viridis", colorscale.min = 0, colorscale.max = 1,
    colorbar.title = "Beta value"
))
```

`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

`ns`	A namespace function, typically created by `NS(id)`.
`defaults`	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

`ns`	A namespace function, typically created by `NS(id)`.
`defaults`	A named list of default values for the inputs.
`include.fit.lines`	Logical; whether to include the fit and model line inputs: "line of best fit", "linear model line", custom model lines, and whether those lines are listed in the legend (`model.legend.show`, off by default). 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)
```

uniform_stats_inputs_ui

Generate uniform Stats input UI

Description

Creates a standardized tagList of statistical testing inputs for use across plot modules that support pairwise comparisons (BoxPlot, yPlot, freqPlot).

Usage

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

Arguments

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

Value

A tagList containing the stats input UI elements.

Author(s)

Jared Andrews

See Also

reset_stats_inputs(), default_stat_pairs(), stat_bracket_headroom(), create_stat_annotations(), apply_stat_annotations()

Examples

```
uniform_stats_inputs_ui(shiny::NS("stats"))
```

uniform_subplot_spacing_inputs_ui

Generate uniform subplot spacing input UI

Description

Creates a standardized tagList of the horizontal/vertical subplot spacing inputs. These controls only have an effect when faceting is active, so they are rendered within each module's "Facet" tab.

Usage

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

Arguments

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

Value

A tagList containing the subplot spacing input UI elements.

Author(s)

Jared Andrews

See Also

subplot_spacing_defaults(), apply_facet_subplot_spacing(), reset_plotly_inputs()

Examples

```
uniform_subplot_spacing_inputs_ui(shiny::NS("spacing"), defaults = list(subplot.margin = 0.05))
```

---

updateMultiColorPicker

Update a multiColorPicker input on the client

Description

Change the color values assigned to groups in an existing multiColorPicker input from the server side. You can supply explicit colors, apply a palette by name, or reset the widget back to its initial state.

Usage

```
updateMultiColorPicker(
  session,
  inputId,
  colors = NULL,
  palette = NULL,
  reset = FALSE
)
```

Arguments

session	The Shiny session object, typically session.
inputId	Character. The input id of the multiColorPicker to update.
colors	Optional named character vector of hex colors keyed by group name. Only groups present in the vector will be updated; others remain unchanged. Ignored when palette or reset is provided.
palette	Optional character string giving the name of a palette (as supplied in the widget's palette_options). The palette's colors are applied in order to the widget's groups' color pickers and the palette selector is updated to match. Ignored when reset is TRUE.
reset	Logical. If TRUE, reset the widget to its initial state (colors and selected palette). Overrides colors and palette.

Value

Invisibly returns NULL. Called for its side effect.

Author(s)

Jared Andrews

Examples

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

    ui <- fluidPage(
        multiColorPicker(
            "species_cols",
            "Species colors",
            groups = groups,
            selected_palette = "dittoColors"
        ),
        actionButton("randomize", "Randomize colors"),
        actionButton("apply_pal", "Apply ggplot2 palette"),
        actionButton("reset_cols", "Reset to initial"),
        verbatimTextOutput("chosen")
    )

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

        observeEvent(input$randomize, {
            new_colors <- setNames(
                sprintf("#%06X", sample(0xFFFFFF, length(groups))),
                groups
            )
            updateMultiColorPicker(session, "species_cols", colors = new_colors)
        })
```

```
        observeEvent(input$apply_pal, {
            updateMultiColorPicker(session, "species_cols", palette = "ggplot2")
        })

        observeEvent(input$reset_cols, {
            updateMultiColorPicker(session, "species_cols", reset = TRUE)
        })
    }

    shinyApp(ui, server)
}
```

`updateMultiDynamicInput` *Update a multiDynamicInput input on the client*

Description

Replace all rows of an existing `multiDynamicInput()` from the server, or clear it entirely.

Usage

```
updateMultiDynamicInput(session, inputId, elements = NULL, clear = FALSE)
```

Arguments

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

Value

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

Author(s)

Jacob Martin

`update_viz_select` *Update a select input created by viz_select_input*

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

`session`	The `session` object passed to the module server function.
`inputId`	The id of the input to update.
`choices`	New choices for the input, or `NULL` to leave them unchanged.
`selected`	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.
`...`	Further arguments passed to `shinyWidgets::updateVirtualSelect()`.

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

`path`	Directory of the project to install into. The skills are written under this directory, in the subdirectory determined by `client` (for example `.agents/skills`), which is created if needed.
`client`	Which client convention to install the skills for. `"agents"` (the default) writes to `.agents/skills`, discovered by OpenAI Codex and by GitHub Copilot's project skill locations. `"copilot"` writes to `.github/skills`, GitHub Copilot's repository-native location. `"claude"` writes to `.claude/skills`, 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 `client` values to install into several locations at once.
`overwrite`	Logical; if `FALSE` (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 *Validate a user-provided expression string for safety*

Description

Parses the expression text and walks the AST to ensure it only contains allowed operations (comparisons, logical operators, column references, and literals). Returns the original string if valid, or `NULL` if the input is empty, unparseable, or contains disallowed operations. This is useful when the expression string must be passed through to a downstream function (e.g., `plotthis::BoxPlot(highlight = ...)`) rather than evaluated directly.

Usage

```
validate_expression(expr_text, col_names)
```

Arguments

expr_text	Character string containing the expression to validate (e.g., "group == 'A' & value > 10").
col_names	Character vector of allowed column/symbol names (typically names(data)).

Details

Use this when a module passes a user-typed expression string to an external plotting function that will evaluate it internally. The string is validated but not executed by this function.

Value

The original expr_text string if safe, or NULL.

Author(s)

Jared Andrews

Examples

```
validate_expression("Sepal.Length > 5", names(iris))
validate_expression("system('echo pwned')", names(iris)) # NULL + warning
validate_expression("", names(iris)) # NULL
```

---

viz_select_input *Create a select input that scales to very large choice sets*

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

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

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)))
```

webgl_input *The "Plot with WebGL" toggle for a module's point layers*

Description

On (the default), the plot's points switch to WebGL once there are enough of them (see `auto_webgl()`); off, they always stay SVG, so SVG downloads stay fully vector. Read it in the module's plot step with `!isFALSE(isolate_fn(input$webgl))`, so a client that has not reported it yet counts as on.

Usage

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

Arguments

`ns`	The module's namespace function.
`defaults`	The module's `defaults`, whose `webgl` entry sets the value.

Value

A tooltipped checkbox input with id `webgl`.

Author(s)

Jared Andrews

See Also

`auto_webgl()`

Examples

```
webgl_input(shiny::NS("plot"))
webgl_input(shiny::NS("plot"), defaults = list(webgl = FALSE))
```

with_stable_seed *Evaluate a plot expression under a fixed random seed*

Description

Builds a plot with a reproducible random stream so randomised layers (jitter, most notably) land in the same place on every rebuild. Without this, any input change re-draws the jitter offsets, which makes points appear to jump around and leaves anything anchored to a point's position pointing at stale coordinates. The caller's random stream is restored afterwards.

Usage

```
with_stable_seed(expr, seed = 42L)
```

Arguments

expr	Expression producing the plot. Evaluated lazily, after the seed is set.
seed	Integer. Seed to build under.

Value

The value of expr.

Author(s)

Jared Andrews

Examples

```
with_stable_seed(runif(2))
with_stable_seed(runif(2))
```

Index

∗ **datasets**
example_bar, 127
example_composition, 128
example_demographics, 129
example_heatmap_column_data, 130
example_heatmap_matrix, 130
example_iris, 131
example_markers, 132
example_mtcars, 133
example_population, 134
example_rnaseq, 134
example_sales, 135
example_school_earnings, 136
example_skills, 137

add_ablines, 6
add_hlines, 8
add_marginal_plots, 9
add_marginal_plots(), *33*, *37*, *168*
add_plot_config, 11
add_plot_config(), *36, 37*, *167*, *293*
add_reference_lines, 12
add_reference_lines(), *37*
add_size_legend, 14
add_vlines, 16
adjust_column_values, 20
adjust_column_values(), *19*, *38*, *75*, *290*
adjusted_axis_label, 17
adjusted_values, 18
adjusted_values(), *19, 20*, *38*
adjustment_fn, 19
adjustment_fn(), *18, 19*
apply_axis_padding, 22
apply_axis_padding(), *37*, *88*, *98*, *108*, *121*, *163*, *201*, *208*, *217*, *224*, *238*, *245*, *296*
apply_axis_title_to_annotations, 23
apply_facet_subplot_spacing, 24
apply_facet_subplot_spacing(), *42, 43*, *302*
apply_highlight_styling, 25
apply_highlight_styling(), *187*
apply_legend_inputs, 26
apply_legend_inputs(), *16*, *37*, *299*
apply_legend_styling, 27
apply_legend_styling(), *26, 27*, *288*, *297*
apply_level_order, 29
apply_level_order(), *152, 153*, *282*
apply_plotly_newshape, 30
apply_plotly_newshape(), *37*
apply_render_margins, 31
apply_stat_annotations, 31
apply_stat_annotations(), *33*, *177*, *291*, *301*
apply_subplot_axis_styling, 33
apply_subplot_axis_styling(), *37*
apply_tick_notation, 34
apply_tick_notation(), *37*, *88*, *98*, *108*, *121*, *163*, *200*, *208*, *217*, *224*, *238*, *245*, *296*
apply_title_layout, 35
apply_title_layout(), *37*
apply_uniform_styling, 36
apply_uniform_styling(), *22*, *34*, *45, 46*, *102*, *168*
as.data.frame(), *77*, *91*, *102*, *111*, *124*, *165*, *186*, *196*, *203*, *211*, *220*, *227*, *233*, *240*, *248*, *257*, *259*
as_plotted, 38
as_plotted(), *19, 20*
auto_webgl, 39
auto_webgl(), *310*
axis_titles_as_annotations, 40
axis_titles_as_annotations(), *10*, *12*, *16*, *37*

blank_to_null, 41
blank_to_null(), *176*, *178*
build_facet_annotations, 42
build_facet_panel_borders, 43

build_model_row_spec, 44

cached_structure, 45
circlize::colorRamp2(), *53*
clean_facet_dim, 47
collect_source_data, 48
collect_source_data(), *72, 73*
ComplexHeatmap::columnAnnotation(), *55*, *60*
ComplexHeatmap::Heatmap(), *49*, *51*, *53*, *55, 56*, *60*, *130*
ComplexHeatmap::rowAnnotation(), *55*, *60*
ComplexHeatmap_HeatmapApp, 49
ComplexHeatmap_HeatmapApp(), *56*, *60*, *130*
ComplexHeatmap_HeatmapInfoOutputUI, 51
ComplexHeatmap_HeatmapInfoOutputUI(), *56–59*, *63*
ComplexHeatmap_HeatmapInputsUI, 52
ComplexHeatmap_HeatmapInputsUI(), *51*, *60*
ComplexHeatmap_HeatmapMainOutputUI, 56
ComplexHeatmap_HeatmapMainOutputUI(), *51, 52*, *58, 59*, *62, 63*
ComplexHeatmap_HeatmapOutputUI, 58
ComplexHeatmap_HeatmapOutputUI(), *51, 52*, *56, 57*, *60–63*, *149, 150*
ComplexHeatmap_HeatmapServer, 59
ComplexHeatmap_HeatmapServer(), *50–53*, *56, 57*, *61, 62*, *73*, *141*
ComplexHeatmap_HeatmapStaticOutputUI, 61
ComplexHeatmap_HeatmapSubOutputUI, 62
ComplexHeatmap_HeatmapSubOutputUI(), *52*, *56–59*
compute_pairwise_stats, 63
compute_pairwise_stats(), *74–76*, *177*, *188*, *290, 291*
create_axis_styles, 67
create_axis_styles(), *37*
create_ggplot_axis_style, 68
create_ggplot_axis_style(), *37*
create_highlight_annotations, 69
create_highlight_annotations(), *26*, *72*, *169*, *187*
create_selected_annotations, 71
create_selected_annotations(), *70*, *169*
create_source_download_handler, 72
create_source_download_handler(), *141*, *171*
create_stat_annotations, 74
create_stat_annotations(), *31, 32*, *289*, *291*, *301*
createModuleApp, 65
createModuleApp(), *51*, *84*, *92*, *103*, *119*, *160*, *171*, *182*, *192*, *197*, *204*, *212*, *221*, *228*, *234*, *241*, *252*, *307*

dataFilterServer, 77
dataFilterServer(), *79*, *270*
dataFilterUI, 79
dataFilterUI(), *77, 78*
default_group_colors, 79
default_group_colors(), *264*
default_palettes, 80
default_palettes(), *145*, *172*
default_stat_pairs, 81
default_stat_pairs(), *268*, *301*
discrete_choices, 82
discrete_choices(), *137*
dittoViz::freqPlot(), *83–86*, *90–92*, *128*
dittoViz::scatterPlot(), *92–95*, *100*, *102*
dittoViz::yPlot(), *103–106*, *110, 111*
dittoViz_freqPlotApp, 83
dittoViz_freqPlotApp(), *90–92*
dittoViz_freqPlotInputsUI, 84
dittoViz_freqPlotInputsUI(), *84*, *91, 92*
dittoViz_freqPlotOutputUI, 90
dittoViz_freqPlotOutputUI(), *84*, *90*, *92*
dittoViz_freqPlotServer, 91
dittoViz_freqPlotServer(), *84*, *90, 91*
dittoViz_scatterPlotApp, 92
dittoViz_scatterPlotApp(), *100*, *102*
dittoViz_scatterPlotInputsUI, 93
dittoViz_scatterPlotInputsUI(), *93*, *102*
dittoViz_scatterPlotOutputUI, 101
dittoViz_scatterPlotOutputUI(), *93*, *100*, *102*
dittoViz_scatterPlotServer, 101
dittoViz_scatterPlotServer(), *93*, *100*
dittoViz_yPlotApp, 103
dittoViz_yPlotApp(), *110, 111*
dittoViz_yPlotInputsUI, 104
dittoViz_yPlotInputsUI(), *104*, *111*
dittoViz_yPlotOutputUI, 110
dittoViz_yPlotOutputUI(), *104*, *110, 111*
dittoViz_yPlotServer, 111
dittoViz_yPlotServer(), *104*, *110*
downloadHandler(), *72*

draw_to_png, 112
draw_to_png(), *73*, *114*, *141*
draw_to_svg, 113
draw_to_svg(), *73*, *112*, *141*
DT::datatable(), *270*
dumbbellPlot, 114
dumbbellPlot(), *119, 120*, *123*, *125*
dumbbellPlotApp, 118
dumbbellPlotApp(), *123*, *125*
dumbbellPlotInputsUI, 119
dumbbellPlotInputsUI(), *119*, *125*
dumbbellPlotOutputUI, 123
dumbbellPlotOutputUI(), *119*, *123*, *125*
dumbbellPlotServer, 124
dumbbellPlotServer(), *119*, *123*

empty_plot, 125
error_bar_halfwidth, 126
example_bar, 127
example_composition, 128
example_demographics, 129
example_heatmap_column_data, *130*, 130
example_heatmap_matrix, *130*, 130
example_iris, 131
example_markers, 132
example_mtcars, 133
example_population, 134
example_rnaseq, 134
example_sales, 135
example_school_earnings, 136
example_skills, 137

facet_check, 137
facet_check(), *82*
figureBuilderApp, 138
figureBuilderApp(), *142, 143*, *170, 171*
figureBuilderServer, 140
figureBuilderServer(), *61*, *113*, *139, 140*, *142, 143*, *170*
figureBuilderUI, 142
figureBuilderUI(), *139–142*, *170*
finalize_manual_edits, 143
finalize_manual_edits(), *46*, *262*, *266*, *283, 284*
flatten_palette_options, 145
format(), *34*
freezeReactiveValue(), *280*

generate_pair_strings, 145
generate_pair_strings(), *81*
get_default, 146
get_default(), *53*, *79, 80*, *285, 286*
get_documentation, 147
get_model_backend, 148
ggplot2::geom_text(), *125*
ggplot2::scale_size(), *15*
ggplot2::theme_void(), *125*
grDevices::svg(), *113*

heatmap_fit_width, 149
hide_input, 150
hide_input(), *153*, *287*, *293, 294*

InteractiveComplexHeatmap::HeatmapInfoOutput(), *51, 52*
InteractiveComplexHeatmap::InteractiveComplexHeatmapOutput *58, 59*, *149*
InteractiveComplexHeatmap::makeInteractiveComplexHeatmap() *60*, *149*
InteractiveComplexHeatmap::originalHeatmapOutput(), *56, 57*
InteractiveComplexHeatmap::subHeatmapOutput(), *62, 63*
is_pure_type, 151

level_order_choices, 152
level_order_choices(), *29*, *153*, *282*
level_order_input, 153
level_order_input(), *29*, *152*, *281, 282*
linePlot, 154
linePlot(), *126*, *159–161*, *164*, *166*
linePlotApp, 159
linePlotApp(), *66*, *164*, *166*
linePlotInputsUI, 161
linePlotInputsUI(), *160*, *166*
linePlotOutputUI, 165
linePlotOutputUI(), *160*, *164*, *166*
linePlotServer, 165
linePlotServer(), *160*, *164*
linetype_to_dash, 166
list_model_backends, 167

main_title_input_ids, 167, *293*
mark_overlay_axes, 168
mark_overlay_axes(), *22*, *34*
merge_annotation_sets, 169
merge_annotation_sets(), *70*, *72*
module_tack_ui, 171

module_tack_ui(), *72, 73*
moduleGalleryApp, 170
moduleGalleryApp(), *84*, *92*, *103*, *119*, *139–143*, *160*, *182*, *192*, *197*, *204*, *212*, *221*, *228*, *234*, *241*, *252*
multiColorPicker, 172
multiColorPicker(), *55*, *79*, *174*, *269*, *279, 280*, *285*, *297*
multiDynamicInput, 174
multiDynamicInput(), *44, 45*, *55*, *97*, *258*, *304*

na_to_null, 176
na_to_null(), *41*
note_brackets_skipped, 176
nz_value, 177
nz_value(), *41*, *176*

organize_inputs, 178
organize_inputs(), *53*, *56*, *90*, *100*, *110*, *123*, *150, 151*, *153*, *164*, *184*, *195*, *201*, *209*, *218*, *225*, *232*, *239*, *246*, *248*, *255*, *286, 287*, *294*

parallelCoordinatesPlot, 179
parallelCoordinatesPlot(), *181–184*, *186*
parallelCoordinatesPlotApp, 181
parallelCoordinatesPlotApp(), *184*, *186*
parallelCoordinatesPlotInputsUI, 183
parallelCoordinatesPlotInputsUI(), *182*, *186*
parallelCoordinatesPlotOutputUI, 185
parallelCoordinatesPlotOutputUI(), *182*, *184*, *186*
parallelCoordinatesPlotServer, 185
parallelCoordinatesPlotServer(), *182*, *184*
parse_highlight_values, 186
parse_highlight_values(), *26*, *70*
parse_numeric_list, 187
parse_pair_strings, 188
parse_pair_strings(), *81*
piePlot, 189
piePlot(), *192–196*
piePlotApp, 191
piePlotApp(), *195, 196*
piePlotInputsUI, 193
piePlotInputsUI(), *192*, *196*
piePlotOutputUI, 195
piePlotOutputUI(), *192*, *195, 196*
piePlotServer, 196
piePlotServer(), *192*, *195*
plotly::ggplotly(), *14*, *22*, *34*, *36*, *39*, *287*
plotly::layout(), *32, 33*
plotly::plotly_build(), *16*, *45*, *168*
plotly::plotly_data(), *48*
plotly::renderPlotly(), *143, 144*, *283*
plotly::toWebGL(), *39*
plotthis::AreaPlot(), *197–199*, *201*
plotthis::BarPlot(), *203*, *205*, *207*, *209*
plotthis::BoxPlot(), *211*, *213*, *215*, *218*
plotthis::DensityPlot(), *222, 223*, *225*
plotthis::DotPlot(), *227*, *229, 230*, *232*
plotthis::Histogram(), *235–237*, *239*
plotthis::SplitBarPlot(), *241–244*, *246*, *248*
plotthis_AreaPlotApp, 197
plotthis_AreaPlotApp(), *134*, *201*
plotthis_AreaPlotInputsUI, 198
plotthis_AreaPlotOutputUI, 202
plotthis_AreaPlotOutputUI(), *201*
plotthis_AreaPlotServer, 202
plotthis_AreaPlotServer(), *201*
plotthis_BarPlotApp, 203
plotthis_BarPlotApp(), *66*, *127*, *209*
plotthis_BarPlotInputsUI, 205
plotthis_BarPlotInputsUI(), *65*
plotthis_BarPlotOutputUI, 210
plotthis_BarPlotOutputUI(), *66*, *209*
plotthis_BarPlotServer, 210
plotthis_BarPlotServer(), *66*, *209*
plotthis_BoxPlotApp, 211
plotthis_BoxPlotApp(), *170*, *218*
plotthis_BoxPlotInputsUI, 212
plotthis_BoxPlotOutputUI, 219
plotthis_BoxPlotOutputUI(), *218*
plotthis_BoxPlotServer, 219
plotthis_BoxPlotServer(), *218*
plotthis_DensityPlotApp, 220
plotthis_DensityPlotApp(), *225*
plotthis_DensityPlotInputsUI, 221
plotthis_DensityPlotOutputUI, 226
plotthis_DensityPlotOutputUI(), *225*
plotthis_DensityPlotServer, 226
plotthis_DensityPlotServer(), *225*
plotthis_DotPlotApp, 227
plotthis_DotPlotApp(), *132*, *232*

plotthis_DotPlotInputsUI, 228
plotthis_DotPlotOutputUI, 232
plotthis_DotPlotOutputUI(), *232*
plotthis_DotPlotServer, 233
plotthis_DotPlotServer(), *232*
plotthis_HistogramApp, 234
plotthis_HistogramApp(), *239*
plotthis_HistogramInputsUI, 235
plotthis_HistogramOutputUI, 239
plotthis_HistogramOutputUI(), *239*
plotthis_HistogramServer, 240
plotthis_HistogramServer(), *239*
plotthis_SplitBarPlotApp, 241
plotthis_SplitBarPlotApp(), *127*, *246*, *248*
plotthis_SplitBarPlotInputsUI, 242
plotthis_SplitBarPlotInputsUI(), *248*
plotthis_SplitBarPlotOutputUI, 247
plotthis_SplitBarPlotOutputUI(), *246*, *248*
plotthis_SplitBarPlotServer, 247
plotthis_SplitBarPlotServer(), *246*

radarPlot, 248
radarPlot(), *252*, *254, 255*, *257*
radarPlotApp, 251
radarPlotApp(), *255*, *257*
radarPlotInputsUI, 253
radarPlotInputsUI(), *252*, *257*
radarPlotOutputUI, 256
radarPlotOutputUI(), *252*, *255*, *257*
radarPlotServer, 256
radarPlotServer(), *252*, *255*
recycle_line_style, 257
register_model_backend, 258
register_model_backend(), *167*
require_data_frame, 259
requireNamespace(), *60*
reset_annotation_inputs, 260
reset_axes_inputs, 261
reset_axis_title_text, 261
reset_colorscale_inputs, 262
reset_colorscale_inputs(), *269*, *297, 298*
reset_group_colors, 263
reset_group_colors(), *80*
reset_legend_inputs, 264
reset_legend_inputs(), *27*, *299*
reset_lines_inputs, 265
reset_manual_edits, 266
reset_plotly_inputs, 267
reset_plotly_inputs(), *292*, *302*
reset_stats_inputs, 267
reset_stats_inputs(), *81*, *301*
resolve_colorscale, 268
resolve_colorscale(), *297, 298*
resolve_column_targets, 269
resolve_column_targets(), *78*
resolve_facet_layout, 271
resolve_facet_sharing, 272
resolve_palette, 272
resolve_palette(), *79, 80*, *145*, *280*

safe_eval_filter, 274
safe_eval_filter(), *56*
safe_resolve_adj_fxn, 275
safe_resolve_adj_fxn(), *18*, *20*
setup_auto_update_logic, 276
setup_auto_update_logic(), *36*, *280*, *285, 286*
setup_axis_range, 277
setup_axis_range(), *289*
setup_group_colors, 279
setup_group_colors(), *80*, *264*, *279*, *297*
setup_level_order, 281
setup_level_order(), *29*, *152, 153*
setup_manual_edits, 283
setup_manual_edits(), *143, 144*, *262*, *266*
setup_reactive_defaults, 284
setup_reactive_defaults(), *45*, *50*, *53*, *60*, *66*, *83*, *85*, *91, 92*, *94*, *102–104*, *111*, *118*, *120*, *124*, *146*, *160, 161*, *166*, *182, 183*, *186*, *192, 193*, *196–198*, *203–205*, *211–213*, *220–222*, *227*, *229*, *233–235*, *240–242*, *248*, *252, 253*, *257*, *276–278*, *280*, *282*
shiny::downloadButton(), *72*
shiny::is.reactive(), *285*
shiny::isolate(), *53*, *85*, *94*, *104*, *120*, *144*, *146*, *161*, *183*, *193*, *198*, *205*, *213*, *222*, *229*, *235*, *242*, *253*
shiny::moduleServer(), *283*
shiny::plotOutput(), *61*
shiny::reactive(), *50*, *53*, *60*, *66*, *83*, *85*, *91, 92*, *94*, *102–104*, *111*, *118*, *120*, *124*, *146*, *160, 161*, *166*, *182, 183*, *186*, *192, 193*, *196–198*, *203–205*, *211–213*, *220–222*, *227*, *229*,

233–235, *240–242*, *248*, *252*, *253*, *257*, *282*, *285*
shiny::reactiveVal(), *48*, *50*, *53*, *60*, *66*, *83*, *85*, *91*, *92*, *94*, *102–104*, *111*, *118*, *120*, *124*, *146*, *160*, *161*, *166*, *182*, *183*, *186*, *192*, *193*, *196–198*, *203–205*, *211–213*, *220–222*, *227*, *229*, *233–235*, *240–242*, *248*, *252*, *253*, *257*, *278*, *280*, *282–285*
shiny::reactiveValues(), *283*
shiny::renderPlot(), *112*, *113*
shiny::req(), *259*
shiny::selectInput(), *308*, *309*
shiny::shinyApp(), *66*, *139*, *140*, *170*
shiny::updateSelectInput(), *305*
shinyjqui::jqui_resizable(), *90*, *101*, *110*, *123*, *165*, *185*, *195*, *202*, *210*, *219*, *226*, *232*, *239*, *247*, *256*
shinyjqui::orderInput(), *153*
shinyjs::useShinyjs(), *142*
shinyWidgets::updateVirtualSelect(), *305*
shinyWidgets::virtualSelectInput(), *308*, *309*
show_input, 286
show_input(), *151*, *293*, *294*
split_color_shape_legend, 287
stat_bracket_headroom, 288
stat_bracket_headroom(), *301*
stat_bracket_y_max, 289
stat_bracket_y_max(), *32*, *278*, *279*, *288*, *289*
stats::density(), *10*
stats::p.adjust(), *64*
string_to_linetypes, 291
subplot_spacing_defaults, 292
subplot_spacing_defaults(), *302*

toggle_facet_title_inputs, 293
toggle_facet_title_inputs(), *167*, *168*
toggle_input_cell, 294
toggle_input_cell(), *151*, *287*

uniform_annotation_inputs_ui, 295
uniform_annotation_inputs_ui(), *26*, *70*, *72*, *187*, *260*
uniform_axes_inputs_ui, 296
uniform_axes_inputs_ui(), *36*, *261*
uniform_colorscale_inputs_ui, 297
uniform_colorscale_inputs_ui(), *262*, *269*
uniform_legend_inputs_ui, 298
uniform_legend_inputs_ui(), *26*, *27*, *36*, *264*
uniform_lines_inputs_ui, 299
uniform_lines_inputs_ui(), *36*, *265*
uniform_plotly_inputs_ui, 300
uniform_plotly_inputs_ui(), *30*, *36*, *267*
uniform_stats_inputs_ui, 301
uniform_stats_inputs_ui(), *267*, *289*
uniform_subplot_spacing_inputs_ui, 301
uniform_subplot_spacing_inputs_ui(), *292*
update_viz_select, 305
update_viz_select(), *138*, *309*
updateMultiColorPicker, 302
updateMultiColorPicker(), *264*
updateMultiDynamicInput, 304
use_vizmodules_skills, 306

validate_expression, 307
viz_select_input, 308
viz_select_input(), *305*

webgl_input, 310
webgl_input(), *39*
with_stable_seed, 310