

Package: ggpop (via r-universe)

October 29, 2026

Type Package

Title Icon-Based Population Charts and Plots for 'ggplot2'

Version 1.9.0

Date 2026-09-29

Description Create engaging population charts and point plots in R.
'ggpop' allows users to represent population data and points proportionally using customizable icons, facilitating the creation of circular representative population charts as well as any point-plots.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.3.3

Suggests testthat (>= 3.0.0), knitr, rmarkdown, cowplot, ggforce, gganimate, ggrepel, ggtext, scales, reactable, patchwork

Config/Needs/website sf, geofacet, ggtext, quarto, kcuilla/reactablefmr

Config/testthat/edition 3

Depends R (>= 4.0.5)

Imports ggplot2, dplyr, ggimage, magick, rlang, tidyr, purrr, fontawesome, rsvg, cli, tibble, grid, grDevices, stats, tools, utils

VignetteBuilder knitr

ByteCompile true

BugReports <https://github.com/jurjoraa/ggpop/issues>

URL <https://jurjoraa.github.io/ggpop/>

Config/pak/sysreqs libcairo2-dev cmake libfontconfig1-dev libfreetype6-dev make libmagick++-dev gsfons libicu-dev libpng-dev librsvg2-dev libuv1-dev libssl-dev

Repository <https://jurjoraa.r-universe.dev>

Date/Publication 2026-09-29 17:24:39 UTC
RemoteUrl <https://github.com/jurjoroa/ggpop>
RemoteRef HEAD
RemoteSha 575e39a79536eff92adeea1458631fe7268bb1e4

Contents

ggpop-package	2
fa_icons	5
fetch_df_coordinates	6
geom_icon_point	7
geom_pop	9
ggpop_markers	12
icon_grid	13
key_legend	14
legend_add_key	16
legend_box	17
legend_canvas	18
legend_composite	21
legend_ratios	23
legend_render	24
legend_spec	24
legend_strip	26
legend_subset	27
marker_encode	28
marker_legend	29
process_data	31
scale_legend_icon	32
theme_pop	33
theme_pop_dark	34
theme_pop_minimal	35
Index	37

ggpop-package	<i>ggpop: Icon-Based Population Charts for R</i>
---------------	--

Description

ggpop is a ggplot2 extension for creating icon-based population charts and pictogram plots. Use [geom_pop\(\)](#) and [geom_icon_point\(\)](#) to visualize proportion and population data with 2,000+ Font Awesome icons.

Main functions

- `geom_pop()` – proportional icon grids
- `geom_icon_point()` – icon scatter plots
- `process_data()` – prepare count data for plotting
- `fa_icons()` – search Font Awesome icon names
- `theme_pop()`, `theme_pop_dark()`, `theme_pop_minimal()` – built-in themes

`process_data()`

Converts count data to one row per icon. `group_var` and `sum_var` are unquoted; `high_group_var` takes a character string for faceted charts.

```
df_plot <- process_data(  
  data      = data.frame(sex = c("Female", "Male"), n = c(55, 45)),  
  group_var = sex,  
  sum_var   = n,  
  sample_size = 20  
)
```

`geom_pop()`

Draws icon grids. Add an icon column, map icon and color in `aes()`. Do not map x or y.

```
ggplot() +  
  geom_pop(data = df_plot, aes(icon = icon, color = type), size = 2) +  
  scale_color_manual(values = c(Female = "#C0392B", Male = "#2980B9")) +  
  theme_pop()
```

`geom_icon_point()`

Drop-in replacement for `geom_point()` using Font Awesome icons.

```
ggplot(iris, aes(x = Sepal.Length, y = Petal.Length, color = Species)) +  
  geom_icon_point(icon = "seedling", size = 1)
```

`fa_icons()`

Search the bundled Font Awesome icon list by keyword.

```
fa_icons(query = "person")
```

Themes

Three built-in themes optimized for icon charts: `theme_pop()`, `theme_pop_dark()`, `theme_pop_minimal()`.

Author(s)

Maintainer: Jorge A. Roa-Contreras <jorgeroa@stanford.edu> ([ORCID](#))

Authors:

- Ralitza Soultanova <Ralitza.soultanova@gmail.com> ([ORCID](#))
- Fernando Alarid-Escudero <falarid@stanford.edu> ([ORCID](#))
- Carlos Pineda-Antunez <cpinedaa@uw.edu> ([ORCID](#))

See Also

Useful links:

- <https://jurjoraa.github.io/ggpop/>
- Report bugs at <https://github.com/jurjoraa/ggpop/issues>

Examples

```
library(ggplot2)
library(dplyr)

## -----
## geom_pop(): population icon grid
## -----
df_plot <- process_data(
  data      = data.frame(sex = c("Female", "Male"), n = c(55, 45)),
  group_var = sex,
  sum_var   = n,
  sample_size = 20
) %>%
  mutate(icon = ifelse(type == "Female", "person-dress", "person"))

ggplot() +
  geom_pop(data = df_plot, aes(icon = icon, color = type), size = 2) +
  scale_color_manual(values = c(Female = "#C0392B", Male = "#2980B9")) +
  theme_pop() +
  labs(title = "Population by sex", color = NULL)

## -----
## geom_icon_point(): icon scatter plot
## -----
ggplot(iris, aes(x = Sepal.Length, y = Petal.Length, color = Species)) +
  geom_icon_point(icon = "seedling", size = 1) +
  scale_color_manual(values = c(
    setosa      = "#43A047",
    versicolor = "#1E88E5",
    virginica   = "#E53935"
  )) +
  labs(title = "Iris dataset", x = "Sepal Length", y = "Petal Length")
```

fa_icons

*Search and list Font Awesome icons***Description**

Retrieves Font Awesome icon names, optionally filtered by a search query or category. Results can be returned as a plain character vector or as a tibble with category classification.

Usage

```
fa_icons(
  query = NULL,
  category = NULL,
  regex = FALSE,
  classify = TRUE,
  include_unclassified = TRUE,
  class_map = NULL,
  primary_only = TRUE,
  as_vector = FALSE
)
```

Arguments

query	Character string. Filter icons whose names contain query. Set to NULL (default) to return all icons. If regex = TRUE, query is treated as a Perl-compatible regular expression.
category	Character vector. One or more category names to filter by. Run fa_categories() to see valid options. Setting category implies classify = TRUE.
regex	Logical. When TRUE, query is interpreted as a Perl-compatible regular expression. Default FALSE (fixed-string match).
classify	Logical. When TRUE (default), each icon is classified into categories using class_map and a primary_class column is included in the returned tibble. Ignored when as_vector = TRUE and category = NULL.
include_unclassified	Logical. When FALSE, icons that do not match any category pattern are dropped. Default TRUE.
class_map	A named list mapping category names to regex patterns. Defaults to the internal .fa_default_class_map().
primary_only	Logical. When TRUE (default), the tibble contains only the primary_class column and omits all_classes.
as_vector	Logical. When TRUE, return a plain sorted character vector of icon names instead of a tibble. If category = NULL, classification is skipped entirely. Default FALSE.

Value

When `as_vector = TRUE`, a sorted character vector of icon names. Otherwise a [tibble](#) with columns:

icon Icon name (character).

primary_class Primary category the icon belongs to, or NA when unclassified (character).

all_classes All matching categories (list-column of character vectors). Only present when `primary_only = FALSE`.

Examples

```
# All icons as a classified tibble
fa_icons()

# Quick lookup -- plain sorted vector
head(fa_icons(as_vector = TRUE), 10)

# Search for icons whose name contains "heart"
fa_icons(query = "heart")

# Filter by category
fa_icons(category = "animals")

# Regex search -- all icons starting with "arrow"
fa_icons(query = "^arrow", regex = TRUE)
```

fetch_df_coordinates	<i>Fetches the df_coordinates_final Dataset</i>
----------------------	---

Description

Downloads and caches the `df_coordinates_final` dataset if it is not already cached locally. This function ensures that the dataset is downloaded only once and loaded into memory without cluttering the global environment. The dataset is stored in a package-specific cache directory and retrieved efficiently for subsequent uses.

Usage

```
fetch_df_coordinates()
```

Details

The dataset is downloaded from GitHub. The file is cached in a directory specific to the package, which is determined using [R_user_dir](#). If the dataset is already cached, it will be loaded directly from the cache instead of downloading again.

Value

A data frame containing the `df_coordinates_final` dataset.

Examples

```
df <- fetch_df_coordinates()
head(df)
```

geom_icon_point	Create a scatter plot with Font Awesome icons instead of points
-----------------	---

Description

Works exactly like `geom_point()`, but renders Font Awesome icons instead of dots. Pass any data with x and y variables - no special formatting required.

Usage

```
geom_icon_point(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  icon = NULL,
  size = 1,
  dpi = 50,
  legend_icons = TRUE,
  stroke_width = NULL,
  icon_path = NULL,
  ...
)
```

Arguments

mapping	Set of aesthetic mappings created by <code>aes()</code> . If specified and <code>inherit.aes = TRUE</code> (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	The data to be displayed in this layer. There are three options: If <code>NULL</code> , the default, the data is inherited from the plot data as specified in the call to <code>ggplot()</code> .

A `data.frame`, or other object, will override the plot data. All objects will be fortified to produce a data frame. See `fortify()` for which variables will be created.

A function will be called with a single argument, the plot data. The return value must be a `data.frame`, and will be used as the layer data. A function can be created from a formula (e.g. `~ head(.x, 10)`).

<code>stat</code>	<p>The statistical transformation to use on the data for this layer. When using a <code>geom_*()</code> function to construct a layer, the <code>stat</code> argument can be used to override the default coupling between geoms and stats. The <code>stat</code> argument accepts the following:</p> <ul style="list-style-type: none"> • A <code>Stat</code> ggproto subclass, for example <code>StatCount</code>. • A string naming the stat. To give the stat as a string, strip the function name of the <code>stat_</code> prefix. For example, to use <code>stat_count()</code>, give the stat as <code>"count"</code>. • For more information and other ways to specify the stat, see the layer stat documentation.
<code>position</code>	<p>A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The <code>position</code> argument accepts the following:</p> <ul style="list-style-type: none"> • The result of calling a position function, such as <code>position_jitter()</code>. This method allows for passing extra arguments to the position. • A string naming the position adjustment. To give the position as a string, strip the function name of the <code>position_</code> prefix. For example, to use <code>position_jitter()</code>, give the position as <code>"jitter"</code>. • For more information and other ways to specify the position, see the layer position documentation.
<code>na.rm</code>	logical, whether remove NA values
<code>show.legend</code>	Logical. Should this layer be included in the legends? NA (default) includes the layer if any aesthetics are mapped. FALSE suppresses the layer's legend entries entirely.
<code>inherit.aes</code>	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. <code>annotation_borders()</code> .
<code>icon</code>	Default icon (default: NULL). Accepts a Font Awesome name, a bundled ggpop marker name (e.g. <code>"square-inset"</code> , <code>"circle-plus"</code> , <code>"diamond-hollow"</code>), or a path to a local <code>.svg</code> file. The same sources are valid in <code>aes(icon = ...)</code> ; SVG markers are recoloured by the mapped colour aesthetic.
<code>size</code>	Default icon size (default: 1).
<code>dpi</code>	Icon resolution (default: 50).
<code>legend_icons</code>	Show icons in legend (default: TRUE).
<code>stroke_width</code>	Numeric. Width of the icon outline/stroke.
<code>icon_path</code>	Optional path to a folder of your own SVG icons, referenced by file name (without <code>.svg</code>) through the icon aesthetic - just like a Font Awesome name. Defaults to <code>getOption("ggpop.icon.path")</code> . Monochrome SVGs (<code>fill="#000000"</code> or <code>currentColor</code>) are recoloured by the mapped colour. See ggpop_markers .
<code>...</code>	additional parameters

Value

A ggplot layer.

Aesthetics

geom_icon_point uses standard ggplot2 scatter plot aesthetics:

- **x** - Numeric variable for x-axis
- **y** - Numeric variable for y-axis
- **icon** - Font Awesome icon name (optional, column or mapped)
- **color/colour** - Color grouping
- **alpha** - Transparency
- **size** - Icon size

Examples

```
library(ggplot2)
data <- data.frame(
  x = rnorm(20),
  y = rnorm(20),
  category = sample(c("A", "B", "C"), 20, replace = TRUE),
  icon = sample(c("heart", "star", "circle"), 20, replace = TRUE)
)

# Map icon to a column
ggplot(data, aes(x = x, y = y, icon = icon, color = category)) +
  geom_icon_point()

# Use a fixed icon
ggplot(data, aes(x = x, y = y, color = category)) +
  geom_icon_point(icon = "star")
```

geom_pop

Create a circular representative population chart

Description

Draws a circular representative population chart based on group proportions, where each point (person) represents a fixed number of individuals. Each person is rendered as a Font Awesome icon.

Usage

```
geom_pop(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  na.rm = FALSE,
  show.legend = NA,
  inherit.aes = TRUE,
  icon = "ggmale",
  group_var = NULL,
  sample_size = NULL,
  arrange = FALSE,
  seed = NULL,
  sum_var = NULL,
  facet = NULL,
  size = 1,
  dpi = 50,
  legend_icons = TRUE,
  stroke_width = NULL,
  icon_path = NULL,
  ...
)
```

Arguments

mapping	Set of aesthetic mappings created by aes() . If specified and <code>inherit.aes = TRUE</code> (the default), it is combined with the default mapping at the top level of the plot. You must supply mapping if there is no plot mapping.
data	<p>The data to be displayed in this layer. There are three options:</p> <p>If <code>NULL</code>, the default, the data is inherited from the plot data as specified in the call to ggplot().</p> <p>A <code>data.frame</code>, or other object, will override the plot data. All objects will be fortified to produce a data frame. See fortify() for which variables will be created.</p> <p>A function will be called with a single argument, the plot data. The return value must be a <code>data.frame</code>, and will be used as the layer data. A function can be created from a formula (e.g. <code>~ head(.x, 10)</code>).</p>
stat	<p>The statistical transformation to use on the data for this layer. When using a <code>geom_*()</code> function to construct a layer, the <code>stat</code> argument can be used to override the default coupling between geoms and stats. The <code>stat</code> argument accepts the following:</p> <ul style="list-style-type: none"> • A Stat ggproto subclass, for example <code>StatCount</code>. • A string naming the stat. To give the stat as a string, strip the function name of the <code>stat_</code> prefix. For example, to use <code>stat_count()</code>, give the stat as <code>"count"</code>.

	<ul style="list-style-type: none"> For more information and other ways to specify the stat, see the layer stat documentation.
position	<p>A position adjustment to use on the data for this layer. This can be used in various ways, including to prevent overplotting and improving the display. The position argument accepts the following:</p> <ul style="list-style-type: none"> The result of calling a position function, such as <code>position_jitter()</code>. This method allows for passing extra arguments to the position. A string naming the position adjustment. To give the position as a string, strip the function name of the <code>position_</code> prefix. For example, to use <code>position_jitter()</code>, give the position as "jitter". For more information and other ways to specify the position, see the layer position documentation.
na.rm	logical, whether remove NA values
show.legend	logical. Should this layer be included in the legends? NA, the default, includes if any aesthetics are mapped. FALSE never includes, and TRUE always includes. It can also be a named logical vector to finely select the aesthetics to display. To include legend keys for all levels, even when no data exists, use TRUE. If NA, all levels are shown in legend, but unobserved levels are omitted.
inherit.aes	If FALSE, overrides the default aesthetics, rather than combining with them. This is most useful for helper functions that define both data and aesthetics and shouldn't inherit behaviour from the default plot specification, e.g. annotation_borders() .
icon	Default icon used when no icon column is mapped. Accepts a Font Awesome name, a bundled ggpop marker name (e.g. "square-inset", "circle-plus", "diamond-hollow"), or a path to a local .svg file. The same sources are valid in <code>aes(icon = ...)</code> ; SVG markers are recoloured by the mapped colour aesthetic.
group_var	(Deprecated) Use <code>aes(group = ...)</code> instead.
sample_size	The total number of individuals (points) to draw.
arrange	Logical; if TRUE, output data is arranged by group.
seed	Optional numeric seed used only when <code>arrange = FALSE</code> .
sum_var	Optional variable to sum over instead of counting.
facet	Optional faceting variable. If provided, final plot must be faceted with <code>ggplot2</code> (use <code>validate_geom_pop_faceting(p)</code>).
size	Icon size. If mapped in <code>aes(size = ...)</code> the parameter is ignored.
dpi	Height (in pixels) of the rendered PNG when using <code>fontawesome::fa_png()</code> . Higher values produce sharper icons.
legend_icons	Logical; if TRUE, legend displays the selected icons.
stroke_width	Numeric. Width of the icon outline in pixels (single value).
icon_path	Optional path to a folder of your own SVG icons, referenced by file name (without .svg) through the icon aesthetic - just like a Font Awesome name. Defaults to <code>getOption("ggpop.icon_path")</code> . Monochrome SVGs (<code>fill="#000000"</code> or <code>currentColor</code>) are recoloured by the mapped colour. See ggpop_markers .
...	additional parameters

Value

A ggplot layer that renders a circular population chart with icons.

Aesthetics

geom_pop understands the following aesthetics:

- **icon**: Font Awesome icon name (mapped column)
- **group**: grouping variable for raw data mode
- **color/colour**: icon color
- **alpha**: transparency (must be mapped)
- **size**: icon size (mapped or fixed)

See Also

[geom_icon_point](#), [process_data](#), [geom_image](#)

Examples

```
library(ggplot2)

df <- data.frame(
  sex = rep(c("F", "M"), each = 10),
  icon = rep(c("female", "male"), each = 10)
)

ggplot() +
  geom_pop(
    data = df,
    aes(icon = icon, group = sex, color = sex),
    size = 3,
    dpi = 80
  )
```

ggpop_markers

List the icon markers ggpop can render by name

Description

Returns the bundled ggpop marker names and, if an icon directory is given, the names of the user SVGs found there. These names (plus any Font Awesome name) are valid values for the icon aesthetic of `geom_pop()` and `geom_icon_point()`.

Usage

```
ggpop_markers(icon_path = getOption("ggpop.icon_path"))
```

Arguments

icon_path Optional path to a folder of user SVG icons. Defaults to `getOption("ggpop.icon.path")`.

Value

A list with element `bundled` (character vector of marker names) and, when `icon_path` resolves to a directory, `user`.

Examples

```
ggpop_markers()
```

icon_grid	<i>Build icon grid rows from plot data</i>
-----------	--

Description

Derives the icon row/column positions from the unique combinations in `df`, returning a plain data frame ready to `rbind()` with group and symbol rows before passing to [legend_canvas](#).

Usage

```
icon_grid(df, icon, label, row, col, label_fn = NULL, section = "grid")
```

Arguments

df Data frame used in the ggplot call.

icon Column name holding icon names.

label Column name holding cell labels.

row Column whose unique values define grid rows.

col Column whose unique values define grid columns.

label_fn Optional function applied to label values before display.

section Value for the section column (default "grid").

Value

A data frame with columns `section`, `type`, `label`, `color`, `icon`, `row`, `col`. Rows are sorted by row then column (factor level order respected).

See Also

[legend_canvas](#)

Examples

```
# df_icons <- icon_grid(
#   df, icon = "icon", label = "AgeLabel",
#   row = "StartAge", col = "StopAge",
#   label_fn = function(x) gsub(" ", "", x)
# )
```

key_legend

Add typed symbol entries to a legend canvas

Description

Adds a column of symbol + label entries to an existing ggplot canvas (typically the output of [marker_legend](#)) using the same `ggplot2::annotate()` approach. The coordinate system is shared with the base plot, so positions integrate seamlessly with the rest of the legend.

Four entry types are supported:

swatch A filled rectangle (colour bands, modality tiles).

line A horizontal segment (frontier or trend lines).

point A bold "*" glyph rendered as text.

text A label with no visible key symbol.

A fourth entry kind, **icon**, exists in [legend_canvas](#)'s `df_legend` vocabulary but is not a `key_legend()` type - icon rows are rendered separately via [marker_legend](#).

Two y-placement modes:

- `y_start = NULL` (default) - first entry is placed at the section-title row (`row_spacing * title_frac`), matching the position where `marker_legend()` puts section headers. Use this when entry\,1 is both the title and its own symbol (e.g. the "Efficient frontier" line entry).
- `y_start = 0` - first entry at row\,1, second at `-row_spacing`, etc. Use this with a text-only title.

Usage

```
key_legend(
  entries,
  x = 0,
  y_start = NULL,
  title = NULL,
  title_frac = 0.85,
  row_spacing = 1,
  key_width = 1.2,
  label_gap = 0.3,
  label_size = 2.8,
```

```

    label_color = "black",
    label_inside = FALSE,
    label_fontface = "plain",
    title_color = NULL,
    swatch_height = 0.45,
    point_size = 1.6
  )

```

Arguments

entries	A data frame with columns type ("swatch", "line", "point", or "text"), label, and color (or colour). Optional columns: linetype (default "solid"), linewidth (default 0.8), pch (default NA -> draws "*" for type = "point").
x	Left edge of the key-symbol column in plot coordinates.
y_start	Y coordinate of the first entry. NULL (default) places entry\,1 at row_spacing * title_frac.
title	Optional text-only section title drawn at row_spacing * title_frac, centred over the key column.
title_frac	Y-fraction used for the section-title row (default 0.85, matching marker_legend()).
row_spacing	Vertical distance between rows. Match the row_spacing passed to marker_legend .
key_width	Horizontal width of the key symbol area.
label_gap	Gap between the key symbol and the label text.
label_size	Text size (passed to ggplot2::annotate()).
label_color	Colour of label text.
label_inside	When TRUE, centres the label inside the key symbol instead of beside it. Only takes effect for swatch rows (default FALSE).
label_fontface	Font face for the title and entry labels (default "plain"). Common values: "plain", "bold", "italic". Does not affect the "point"-type "*" glyph, which is always bold.
title_color	Colour of the section title. Inherits label_color when NULL (the default).
swatch_height	Height of swatch rectangles as a fraction of row_spacing (default 0.45).
point_size	Size multiplier for point glyphs relative to label_size (default 1.6).

Value

A ggpop_key_legend object. Add it to any ggplot with + to inject the annotate layers onto that canvas. Print it (or use it standalone) to render a self-contained legend panel.

See Also

[marker_legend](#)

Examples

```
library(ggplot2)
s <- 0.78
rs <- 0.34 * s

ef <- data.frame(
  type = c("line", "swatch", "point"),
  label = c("Efficient frontier", "Grey zone", "Near efficient"),
  color = c("black", "grey50", "black"),
  stringsAsFactors = FALSE
)

# Standalone panel:
print(key_legend(ef, row_spacing = rs, key_width = 0.22))

# Added to a marker_legend canvas:
# p <- marker_legend(age_entries, ...) +
#   key_legend(ef, x = 1.10, row_spacing = rs, key_width = 0.22)
```

legend_add_key

Add a typed symbol row to a composite legend specification

Description

The new row is appended to the symbol section without changing spec. Render the returned specification to measure its border after the new row.

Usage

```
legend_add_key(
  spec,
  label,
  type = "line",
  colour = "black",
  linetype = "solid",
  linewidth = 0.8,
  pch = NA
)
```

Arguments

spec	A <code>legend_spec()</code> object.
label	Text beside the symbol.
type	One of "line", "point", "swatch", or "text".
colour	Colour of the symbol.

linetype, linewidth
Line style and width for a line row.

pch
Character or numeric point glyph for a point row.

Value

A new ggpop_legend_spec object.

See Also

[legend_spec\(\)](#), [legend_subset\(\)](#), [legend_render\(\)](#)

Examples

```
grid <- data.frame(row = 1, col = 1, icon = "square-inset", label = "A")
spec <- legend_spec(grid, symbols = c(Frontier = "line"))
legend_add_key(spec, "Capacity", type = "line", linetype = "dashed")
```

legend_box

Draw a border tightly around a composite legend's rendered content

Description

[legend_canvas](#) positions its sections from nominal geometry, but text labels overflow their anchor points by an amount that depends on the font, the label strings, and the final output size - none of which are known when the layers are built. A border drawn from that nominal geometry therefore clips the labels. `legend_box()` sidesteps this by rendering the legend once at the intended output size, measuring the true pixel extent of the drawn content, mapping it back to data coordinates, and adding a border rectangle around it.

Because it renders once to measure, the border reflects exactly what will be drawn - long labels, mixed fonts, any content - with no per-figure hand tuning.

Usage

```
legend_box(
  plot,
  width,
  height,
  padding = c(0.02, 0.12),
  colour = "black",
  linewidth = 0.7,
  fill = NA,
  threshold = 150,
  dpi = 150
)
```

Arguments

plot	A ggplot (typically legend_canvas output) whose coordinate system supplies fixed xlim/ylim (<code>legend_canvas()</code> always does). Add <code>legend_box()</code> last, after any other layers, so it encloses everything.
width, height	Physical size in inches of the region the legend will be drawn in for the final export - pass the same values used there so the measured text width matches. For a legend_strip legend this is the full figure width and the strip height.
padding	Length-2 numeric: gap between content and border, as fractions of the measured content width and height (default <code>c(0.02, 0.12)</code>).
colour	Border colour (default "black").
linewidth	Border line width (default <code>0.7</code>).
fill	Border fill (default NA, i.e. transparent).
threshold	Colour-channel cutoff (0-255) for detecting content; a pixel counts as content when its darkest red, green or blue channel is below this value (default 150). Testing the darkest channel, not the gray level, keeps light saturated colours such as yellow inside the border. Very pale fills (all channels above the cutoff) are not detected.
dpi	Resolution of the internal measurement render (default 150); higher is more precise but slower.

Value

plot with a border `ggplot2::annotate("rect", ...)` layer added.

See Also

[legend_canvas](#), [legend_strip](#)

legend_canvas	<i>Build a composite legend from a plain data frame</i>
---------------	---

Description

Renders a legend from a data frame that combines icon grid rows (built with [icon_grid](#)), colour tile rows, and typed-symbol rows via `rbind()`. Layout parameters determine where each section is positioned. The scale parameter multiplies all size and spacing values so the data frame can be written in round numbers.

Usage

```
legend_canvas(  
  df_legend,  
  grid_section = "grid",  
  grid_title = NULL,  
  group_section = NULL,
```

```

group_title = NULL,
group_width = NULL,
group_gap = 0.08,
group_label_size = NA_real_,
group_label_color = "white",
group_swatch_height = 0.44,
symbol_section = NULL,
symbol_right_gap = 0.3,
symbol_key_width = 0.2,
symbol_label_gap = NULL,
col_spacing,
row_spacing,
label_gap,
marker_size,
label_size,
label_fontface = "plain",
scale = 1,
label_scale = 1,
dpi = 300,
xlim = NULL,
ylim = NULL,
x_margin = c(1, 1),
align = NULL,
y_margin = c(1.1, 1.1),
valign = NULL,
clip = "off"
)

```

Arguments

df_legend	<p>Data frame with columns section, type, label, color, icon, row, col. Optional columns: label_size, lineheight. type selects how a row renders and which columns it needs:</p> <p>icon An icon marker (via geom_icon_point) placed at row/col. Only meaningful inside grid_section; requires icon, row, col.</p> <p>swatch A filled rectangle - colour tiles, frontier bands. Requires color.</p> <p>line A horizontal line segment - e.g. an efficient-frontier sample. Requires color.</p> <p>point A bold "*" glyph (or a custom character via an optional pch column). Requires color.</p> <p>swatch/line/point rows are rendered by key_legend - add a new type there, not here. icon rows go through marker_legend instead, a separate path.</p>
grid_section	Value of section identifying icon grid rows (default "grid").
grid_title	Title drawn above the icon grid (NULL = none).
group_section	Value of section identifying colour tile rows. NULL skips the group section.
group_title	Title drawn above the colour tiles. Inherits group_section when NULL.

group_width	Width of the colour tile section (scaled by scale).
group_gap	Gap between the tile right edge and $x = 0$ (scaled by scale).
group_label_size	Label size inside tiles; inherits label_size (after scaling) when NA.
group_label_color	Label colour inside tiles (default "white").
group_swatch_height	Tile height as fraction of row_spacing (default 0.44).
symbol_section	Value of section identifying typed-symbol rows. NULL skips the symbol section.
symbol_right_gap	Gap between icon grid right edge and symbol section (default 0.30; scaled by scale).
symbol_key_width	Width of the key symbol area (default 0.20; scaled by scale).
symbol_label_gap	Gap between key symbol and label; inherits label_gap (after scaling) when NULL.
col_spacing	Horizontal distance between icon grid columns.
row_spacing	Vertical distance between rows.
label_gap	Default gap between key symbol and label.
marker_size	Icon size for the grid.
label_size	Default label text size.
label_fontface	Font face for every title and label in the legend (grid title, group title/tile labels, symbol title/labels) - default "plain". Common values: "plain", "bold", "italic". Does not affect the "point"-type "*" glyph, which is always bold.
scale	Multiplier applied to every length and size: col_spacing, row_spacing, label_gap, marker_size, label_size (and its df_legend column), group_width, group_gap, group_label_size, symbol_right_gap, symbol_key_width, and symbol_label_gap. Because it scales the whole legend uniformly, every length can be written as a plain multiple of one base module and scale sizes the result (default 1).
label_scale	Extra multiplier applied on top of scale to the text sizes only - label_size, group_label_size, and the label_size column in df_legend - leaving marker and spacing sizes untouched. Use to enlarge/shrink every label relative to the markers from one place (default 1).
dpi	Icon render resolution (default 300).
xlim	Length-2 numeric; x limits of the canvas. Auto-computed when NULL.
ylim	Length-2 numeric; y limits of the canvas. Auto-computed when NULL to enclose whichever section reaches deepest - the grid rows, the group tiles, or the symbol keys - so a group/symbol block with more entries than the grid has rows is not clipped.

x_margin	Length-2 numeric: left/right padding added to auto x range (default c(1.00, 1.00), i.e. centred). Ignored when xlim is supplied directly; overridden when align is set. A symbol_section's labels extend right of x_right with no accounting for label width, so the default is sized generously enough to hold typical labels without clipping - not the smallest margin that centres the nominal content bounds.
align	Optional convenience for biasing x_margin: one of "center" (equal left/right padding), "left" (small left / large right padding), or "right" (large left / small right padding). Redistributes sum(x_margin) between the two sides - the total padding is unchanged, only its left/right split. NULL (default) leaves x_margin untouched.
y_margin	Length-2 numeric: top/bottom padding as multiples of (scaled) row_spacing (default c(1.1, 1.1), i.e. centred). Ignored when ylim is supplied directly; overridden when valign is set.
valign	Optional convenience for biasing y_margin: one of "center" (equal top/bottom padding), "top" (small top / large bottom padding), or "bottom" (large top / small bottom padding). Redistributes sum(y_margin) between the two sides, the same way align redistributes x_margin. NULL (default) leaves y_margin untouched.
clip	Passed to coord_cartesian (default "off").

Value

A ggplot ready to save or pass to [legend_strip](#).

See Also

[icon_grid](#), [key_legend](#), [legend_strip](#)

legend_composite	<i>Build a bordered composite legend in one call</i>
------------------	--

Description

Opinionated wrapper over [legend_canvas](#) + [legend_box](#) for the recurring three-section legend (an icon grid, a block of colour tiles, and a small typed-symbol key). You supply the *content* (df_legend) and the strip size; the wrapper applies a fixed proportion ladder ([legend_ratios](#)) driven by three base sizes, centres the content in the strip, and fits a border to the rendered result.

Every layout length is a multiple of one module; every text size is a multiple of one text base; icons use one marker size. A label_size column on the symbol-section rows is read as multiples of text.

The base sizes are calibrated for a base_width-inch strip and are scaled by width / base_width, so a legend keeps identical proportions at any output width - pass your width and the text, markers, and layout all follow. The group and symbol blocks may hold more entries than the grid has rows; the border grows to enclose whichever section runs deepest.

The group colour swatches are rectangles in data coordinates, so shrinking `content_range` to make sparse content fill a wide strip stretches them into banners while the fixed-size icons stay put. `legend_composite()` warns (class `"ggpop_swatch_aspect_warning"`) when the rendered swatch aspect ratio gets banner-like, pointing you to raise `content_range` or reduce width - keep sparse legends compact rather than stretched.

Usage

```
legend_composite(
  df_legend,
  width,
  height,
  grid_title = NULL,
  group_title = NULL,
  grid_section = "grid",
  group_section = "group",
  symbol_section = "symbol",
  module = 0.44294,
  text = 7.756,
  marker = 4.562,
  content_range = 7.21636,
  base_width = 27,
  swatch_height = 0.8,
  fontface = "plain",
  border = "#231F20",
  border_padding = c(0.018, 0.09),
  ratios = legend_ratios(),
  dpi = 300
)
```

Arguments

<code>df_legend</code>	Legend content, as for legend_canvas - grid rows plus optional group/symbol rows, identified by section.
<code>width, height</code>	Physical size (inches) of the strip the legend will fill (passed to legend_box and used to size/centre the content).
<code>grid_title, group_title</code>	Section titles (NULL = none / inherit).
<code>grid_section, group_section, symbol_section</code>	section values that identify each block (defaults "grid", "group", "symbol"). A missing group/symbol section is skipped.
<code>module</code>	Layout module: one row / one column pitch, in legend units.
<code>text</code>	Base text size (ggplot mm) - the primary label size.
<code>marker</code>	Icon marker size (geom_icon_point units).
<code>content_range</code>	Total data-x range the strip maps to; larger renders a smaller legend. The content is centred within it.

base_width	Output width (inches) at which the base sizes (module, text, marker, content_range) are calibrated. Every size is multiplied by width / base_width so the legend keeps identical proportions at any width (default 27). Set base_width = width to disable scaling and size in absolute units.
swatch_height	Colour-tile height as a fraction of a row.
fontface	Font face for all titles and labels.
border	Border colour (NA to skip the border).
border_padding	Length-2 padding passed to legend_box .
ratios	Proportion ladder; defaults to legend_ratios() .
dpi	Icon render resolution.

Value

A ggplot with a fitted border, ready for [legend_strip](#).

See Also

[legend_canvas](#), [legend_box](#), [legend_ratios](#), [legend_strip](#)

legend_ratios	<i>Default proportion ladder for legend_composite</i>
---------------	---

Description

Returns the fixed layout proportions used by [legend_composite](#) as a named list. Lengths are multiples of the layout module (see the module argument there); the two *_label entries are multiples of the text base (text there). Override individual entries and pass the result back via `legend_composite(ratios = ...)`.

Usage

```
legend_ratios()
```

Value

A named list of proportions.

See Also

[legend_composite](#)

legend_render	<i>Render a composite legend specification</i>
---------------	--

Description

Render a composite legend specification

Usage

```
legend_render(spec, width, height)
```

Arguments

spec	A legend_spec() object.
width, height	Physical legend size in inches.

Value

A ggplot with a border fitted to the final content.

See Also

[legend_spec\(\)](#)

Examples

```
grid <- data.frame(row = 1, col = 1, icon = "square-inset", label = "A")
spec <- legend_spec(grid, keys = c(Group = "#1090F3"))
legend_render(spec, width = 7, height = 1.5)
```

legend_spec	<i>Specify a composite legend with external colour keys</i>
-------------	---

Description

Stores icon-grid rows, named colour keys, and typed symbol rows as content. [legend_render\(\)](#) lays out the content and fits its border at the requested physical size. The specification can later be reused for related legends.

Usage

```

legend_spec(
  grid,
  keys = NULL,
  symbols = NULL,
  key_columns = 1L,
  grid_title = NULL,
  group_title = NULL,
  symbol_title = NULL,
  group_key_width = 0.2375,
  group_label_gap = 0.1187,
  group_gap = 0.591,
  key_column_gap = 0.317,
  group_title_nudge = 0,
  swatch_height = 0.56,
  xlim = NULL,
  label_size = 3.634,
  label_fontface = "plain",
  marker_size = 3.717,
  dpi = 120,
  ratios = legend_ratios(),
  border = "#231F20",
  border_padding = c(0.018, 0.09)
)

```

Arguments

<code>grid</code>	A data frame from <code>icon_grid()</code> .
<code>keys</code>	Named character vector of colour values. Names are key labels.
<code>symbols</code>	Named character vector of "line", "point", "swatch", or "text" types, or a data frame accepted by <code>key_legend()</code> . A text row has a label but no visible key.
<code>key_columns</code>	Number of colour-key columns, filled from top to bottom.
<code>grid_title, group_title</code>	Optional titles for the icon grid and colour keys.
<code>symbol_title</code>	Optional title for the symbol column. With no title the first symbol row sits on the title line, so it doubles as the column heading. With a title, the heading takes that line and the symbols start on the same rows as the icon grid and colour keys.
<code>group_key_width, group_label_gap, swatch_height</code>	Dimensions of the external colour keys in grid coordinates. <code>swatch_height</code> is a fraction of one row.
<code>group_gap</code>	Gap from the rightmost colour-key label to the icon grid.
<code>key_column_gap</code>	Gap between colour-key columns.
<code>group_title_nudge</code>	Horizontal adjustment to the centred group title.

xlim	Optional fixed horizontal canvas limits. NULL centres the measured content automatically.
label_size	Text size in millimetres.
label_fontface	Font face for every title and label: "plain", "bold", "italic", or "bold.italic".
marker_size	Icon size passed to geom_icon_point() .
dpi	Icon rendering resolution.
ratios	Layout proportions from legend_ratios() .
border, border_padding	Border colour and padding passed to legend_box() . Use border = NA to omit the border.

Value

A ggpop_legend_spec object to pass to [legend_render\(\)](#).

See Also

[legend_canvas\(\)](#), [legend_composite\(\)](#)

Examples

```
grid <- data.frame(
  row = c(1, 2), col = c(1, 1),
  icon = c("square-inset", "circle-solid"),
  label = c("A", "B")
)
spec <- legend_spec(grid, keys = c(Group1 = "#1090F3", Group2 = "#DF4601"),
  symbols = c(Frontier = "line"), key_columns = 2)
legend_render(spec, width = 7, height = 1.5)
```

legend_strip

Attach a legend strip below a ggplot

Description

Returns a ggpop_legend_strip object. When added to a ggplot with +, produces a ggpop_composite that stacks the main plot above the strip at the specified physical height. The composite works with ggplot2::ggsave() and print(). The main plot (the object legend_strip() is added to) may itself be a patchwork object (e.g. several panels combined with [+/plot_layout\(\)](#))

- this requires the **patchwork** package to be installed.

Usage

```
legend_strip(strip_plot, height)
```

Arguments

`strip_plot` A ggplot to render as the bottom strip (e.g. the output of [marker_legend](#)).
`height` Height of the strip in inches.

Value

A `ggplot2::ggplot` object; add it to a ggplot with `+`.

See Also

[marker_legend](#), [key_legend](#)

Examples

```
# p_legend <- marker_legend(entries, ...) + key_legend(...)
# p_full <- p_scatter + legend_strip(p_legend, height = 1.326)
# ggplot2::ggsave("out.png", p_full, width = 10.5, height = 8.826, dpi = 150)
```

legend_subset

Select content from a composite legend specification

Description

Creates a new specification without changing `spec`. Grid rows are renumbered from one so the selected grid stays compact. Render the result with [legend_render\(\)](#) after selecting its final content.

Usage

```
legend_subset(
  spec,
  grid_rows = NULL,
  keys = NULL,
  symbols = NULL,
  key_columns = NULL,
  group_gap = spec$group_gap,
  group_title_nudge = spec$group_title_nudge,
  xlim = spec$xlim
)
```

Arguments

`spec` A [legend_spec\(\)](#) object.
`grid_rows` Grid row positions to keep, in display order. `NULL` keeps every row.
`keys` Names of colour keys to keep, in display order. `NULL` keeps all.

symbols	Symbol labels to keep, in display order. NULL keeps all.
key_columns	Number of colour-key columns in the result. When keys is supplied, the default is one; otherwise the original count is kept.
group_gap, group_title_nudge, xlim	Optional layout overrides. Each inherits from spec when omitted. Set xlim = NULL to recompute the horizontal limits for the selected content.

Value

A new ggpop_legend_spec object.

See Also

[legend_spec\(\)](#), [legend_add_key\(\)](#), [legend_render\(\)](#)

Examples

```
grid <- data.frame(row = c(1, 2), col = c(1, 1),
                  icon = c("square-inset", "circle-solid"),
                  label = c("A", "B"))
spec <- legend_spec(grid, keys = c(One = "#1090F3", Two = "#DF4601"))
legend_subset(spec, grid_rows = 1, keys = "One")
```

marker_encode	<i>Encode data values as icon marker names</i>
---------------	--

Description

A marker scheme maps one value to a shape family and another to a variant. Regular markers combine the two names with a hyphen. One-time values map directly to a marker, while none uses the scheme’s fallback marker.

Usage

```
marker_encode(start, stop, once = FALSE, none = FALSE, scheme = "sda2028")
```

Arguments

start, stop	Vectors of values used to select shape family and variant. Numeric, character, and factor values are accepted. Values may be missing on rows selected by once or none when they are not needed.
once, none	Logical vectors marking one-time and no-screening rows. Scalars are recycled. The two flags cannot both be TRUE on a row.
scheme	"sda2028" or a list with named character vectors start, stop, and once, plus one character scalar none. The start and stop values are marker-name parts; once and none are complete icon names. Schemes may use bundled markers or user SVG names.

Details

The built-in "sda2028" scheme uses screening start ages 45/50/55 for square/circle/diamond and stop ages 70/75/80/85 for inset/hollow/cross/solid. One-time screening uses solid markers at ages 45/50/55, "plus-bold" at 60, and "triangle-down" at 65.

Value

A character vector of icon names, one per input row.

See Also

[ggpop_markers\(\)](#), [geom_icon_point\(\)](#)

Examples

```
marker_encode(c(45, 50, NA), c(70, 85, NA), none = c(FALSE, FALSE, TRUE))
marker_encode(c(45, 60), c(NA, NA), once = TRUE)

other <- list(
  start = c(A = "square"), stop = c(B = "hollow"),
  once = c(A = "square-solid"), none = "circle-solid"
)
marker_encode("A", "B", scheme = other)
```

marker_legend

Build a standalone composite legend of icon markers

Description

For an ordinary legend keyed to your plot data you do **not** need this function - map an aesthetic and let ggplot2 build the legend natively: `geom_icon_point(..., legend_icons = TRUE)` + `scale_legend_icon()`.

Use `marker_legend()` only for a *standalone composite* legend that ggplot2's guide system cannot express - a multi-column grouped legend decoupled from any plot, often combined with extra annotations and exported at fixed pixel dimensions (for example the screening-strategy Legend_*.png figures).

Usage

```
marker_legend(
  entries,
  layout = c("column", "grid"),
  ncol = 1,
  title = NULL,
  marker_size = 3,
  label_size = 2.8,
  dpi = 300,
```

```

    icon_path = NULL,
    col_spacing = 10,
    row_spacing = 1,
    label_gap = 0.6,
    label_colour = "black",
    label_fontface = "plain",
    default_color = "black"
  )

```

Arguments

entries	A data frame of legend rows. Must contain an icon column (icon source per row) and a label column (text shown beside the marker). An optional colour (or color) column sets the marker colour per row as a literal value. For layout = "grid" the data frame must also contain integer row and col columns.
layout	Legend arrangement. "column" (default) auto-arranges the rows into ncol columns, filling each column top to bottom. "grid" places each entry at its explicit row/col cell.
ncol	Number of columns for layout = "column". Ignored when an explicit column field is supplied in entries.
title	Optional bold title drawn centred above the legend.
marker_size	Icon size passed to geom_icon_point .
label_size	Text size for the labels.
dpi	Icon rendering resolution passed to geom_icon_point .
icon_path	Optional folder of user .svg markers, referenced by bare name in the icon column. See geom_icon_point .
col_spacing	Horizontal distance between columns.
row_spacing	Vertical distance between rows.
label_gap	Horizontal gap between a marker and its label.
label_colour	Text colour for the labels (default: "black").
label_fontface	Font face for the labels (default: "plain"). Common values: "plain", "bold", "italic".
default_color	Marker colour used for rows with no colour value.

Details

Lays out icon + label entries into a self-contained ggplot object. Each entry is drawn with [geom_icon_point](#), so any icon source is accepted - Font Awesome names, bundled ggpop markers, or user-supplied .svg paths (see [ggpop_markers](#)) - and the three may be mixed in a single legend. The result is a plain ggplot you can extend with further ggplot2::annotate() layers (frontier segments, colour bands, asterisks) and export at any size with ggplot2::ggsave().

Value

A ggplot object with theme_void() applied.

See Also

[geom_icon_point](#), [ggpop_markers](#)

Examples

```
# For a normal data-driven legend, prefer the native path instead:
#   geom_icon_point(aes(icon = icon, colour = group), legend_icons = TRUE) +
#   scale_legend_icon()

# marker_legend() is for a STANDALONE composite legend - here two semantic
# colour-columns, the kind ggplot2 guides cannot produce in one figure.
df_legend <- data.frame(
  column = c(1, 1, 2, 2),
  icon    = c("square-inset", "circle-solid", "square-hollow", "diamond-cross"),
  label   = c("Start 45y", "Start 50y", "Stop 75y", "Stop 80y"),
  colour  = c("#FF1493", "#FF1493", "#006400", "#006400"),
  stringsAsFactors = FALSE
)
marker_legend(df_legend, col_spacing = 12)
```

process_data

Process Population Data for Visualization

Description

The `process_data` function processes a dataset to calculate group proportions and generates a sampled dataset based on specified parameters. This processed data is suitable for creating visual representations, such as population charts, where each sample represents a group with associated counts and proportions.

Usage

```
process_data(
  data,
  high_group_var = NULL,
  group_var,
  sum_var = NULL,
  sample_size = 100
)
```

Arguments

<code>data</code>	A data frame containing the population data to be processed.
<code>high_group_var</code>	Character vector, optional. The variables used to group individuals hierarchically. This should be a categorical variable. If provided, the function samples individuals within each group defined by these variables.

group_var	Quosure. The variable used to group individuals in the dataset. This should be a categorical variable.
sum_var	Quosure, optional. The variable to sum over within each group. If NULL, the function counts the number of individuals per group.
sample_size	Integer. The total number of individuals to sample based on group proportions. Must be a positive integer.

Value

A tibble (data frame) with the following columns:

type The sampled group type.

group The group identifier.

n The count of individuals in the group.

prop The proportion of the group relative to the total population.

scale_legend_icon	<i>Legend helper for geom_pop/geom_icon_point legends</i>
-------------------	---

Description

A convenience function to set appropriate legend key sizes for icon-based legends. This is equivalent to using `theme(legend.key.size = ...)` but provides sensible defaults for population icon plots.

Usage

```
scale_legend_icon(
  size = 10,
  unit = "mm",
  spacing = 0.2,
  size_multiplier = 2,
  ...
)
```

Arguments

size	Numeric. Legend key size in specified units (default 10).
unit	Character. Unit for legend key sizing (default "mm").
spacing	Numeric. Spacing between legend items as fraction of size (default 0.2).
size_multiplier	Numeric. Multiplier to apply to the size for spacing calculations (default 2).
...	Additional theme arguments.

Value

A ggplot2 theme object that can be added to a plot.

Examples

```
library(ggplot2)
df <- data.frame(
  type = rep(c("A", "B"), each = 10),
  icon = rep(c("circle", "square"), each = 10)
)
ggplot(df, aes(icon = icon, color = type)) +
  geom_pop() +
  scale_legend_icon(size = 20)
```

theme_pop

*Population Plot Theme***Description**

A minimal theme optimized for icon-based population plots. Similar to `theme_void()` but with automatic legend key sizing, appropriate margins, and sensible defaults for population visualizations.

Usage

```
theme_pop(
  base_size = 11,
  base_family = "",
  base_line_size = base_size/22,
  base_rect_size = base_size/22,
  legend_icon_size = NULL,
  legend_spacing = NULL,
  plot_margin = NULL,
  legend_position = "right"
)
```

Arguments

<code>base_size</code>	Base font size in points (default: 11).
<code>base_family</code>	Base font family (default: "").
<code>base_line_size</code>	Base size for line elements (default: <code>base_size/22</code>).
<code>base_rect_size</code>	Base size for rect elements (default: <code>base_size/22</code>).
<code>legend_icon_size</code>	Size of legend icons in cm. If <code>NULL</code> (default), automatically calculated as <code>base_size/20</code> for proportional sizing.
<code>legend_spacing</code>	Spacing between legend items in cm (default: <code>0.3 * legend_icon_size</code>).
<code>plot_margin</code>	Plot margins. Default: <code>margin(5.5, 5.5, 5.5, 5.5, "pt")</code> . Can be a single numeric (applied to all sides) or <code>margin()</code> object.
<code>legend_position</code>	Position of legend: "none", "left", "right", "bottom", "top" (default: "right").

Value

A ggplot2 theme object.

Examples

```
library(ggplot2)
df <- data.frame(
  type = rep(c("A", "B"), each = 10),
  icon = rep(c("circle", "square"), each = 10)
)
ggplot(data = df, aes(icon = icon, color = type)) +
  geom_pop(size = 1) +
  theme_pop()
```

theme_pop_dark	<i>Dark Population Plot Theme</i>
----------------	-----------------------------------

Description

A dark variant of theme_pop() with white text on black background. Perfect for presentations or dark-mode visualizations.

Usage

```
theme_pop_dark(
  base_size = 11,
  base_family = "",
  base_line_size = base_size/22,
  base_rect_size = base_size/22,
  legend_icon_size = NULL,
  legend_spacing = NULL,
  plot_margin = NULL,
  legend_position = "right",
  bg_color = "black",
  text_color = "white"
)
```

Arguments

- base_size Base font size in points (default: 11).
- base_family Base font family (default: "").
- base_line_size Base size for line elements (default: base_size/22).
- base_rect_size Base size for rect elements (default: base_size/22).

legend_icon_size	Size of legend icons in cm. If NULL (default), automatically calculated as base_size/20 for proportional sizing.
legend_spacing	Spacing between legend items in cm (default: 0.3 * legend_icon_size).
plot_margin	Plot margins. Default: margin(5.5, 5.5, 5.5, 5.5, "pt"). Can be a single numeric (applied to all sides) or margin() object.
legend_position	Position of legend: "none", "left", "right", "bottom", "top" (default: "right").
bg_color	Background color (default: "black").
text_color	Text color (default: "white").

Value

A ggplot2 theme object.

Examples

```
library(ggplot2)
df <- data.frame(
  type = rep(c("A", "B"), each = 10),
  icon = rep(c("circle", "square"), each = 10)
)
ggplot(data = df, aes(icon = icon, color = type)) +
  geom_pop(size = 1) +
  theme_pop_dark(base_size = 40)
```

theme_pop_minimal	<i>Minimal Population Plot Theme</i>
-------------------	--------------------------------------

Description

An ultra-minimal variant with no margins or legend, perfect for icon arrays without annotations.

Usage

```
theme_pop_minimal(base_size = 11, base_family = "")
```

Arguments

base_size	Base font size in points (default: 11).
base_family	Base font family (default: "").

Value

A ggplot2 theme object.

Examples

```
library(ggplot2)
df <- data.frame(
  type = rep(c("A", "B"), each = 10),
  icon = rep(c("circle", "square"), each = 10)
)
ggplot(data = df, aes(icon = icon, color = type)) +
  geom_pop(size = 1) +
  theme_pop_minimal()
```

Index

`aes()`, [7](#), [10](#)
`annotation_borders()`, [8](#), [11](#)

`fa_icons`, [5](#)
`fa_icons()`, [3](#)
`fetch_df_coordinates`, [6](#)
`fortify()`, [8](#), [10](#)

`geom_icon_point`, [7](#), [12](#), [19](#), [22](#), [30](#), [31](#)
`geom_icon_point()`, [2](#), [3](#), [26](#), [29](#)
`geom_image`, [12](#)
`geom_pop`, [9](#)
`geom_pop()`, [2](#), [3](#)
`ggplot()`, [7](#), [10](#)
`ggpop` (ggpop-package), [2](#)
`ggpop-package`, [2](#)
`ggpop_markers`, [8](#), [11](#), [12](#), [30](#), [31](#)
`ggpop_markers()`, [29](#)

`icon_grid`, [13](#), [18](#), [21](#)
`icon_grid()`, [25](#)

`key_legend`, [14](#), [19](#), [21](#), [27](#)
`key_legend()`, [25](#)

`layer position`, [8](#), [11](#)
`layer stat`, [8](#), [11](#)
`legend_add_key`, [16](#)
`legend_add_key()`, [28](#)
`legend_box`, [17](#), [21–23](#)
`legend_box()`, [26](#)
`legend_canvas`, [13](#), [14](#), [17](#), [18](#), [18](#), [21–23](#)
`legend_canvas()`, [26](#)
`legend_composite`, [21](#), [23](#)
`legend_composite()`, [26](#)
`legend_ratios`, [21](#), [23](#), [23](#)
`legend_ratios()`, [26](#)
`legend_render`, [24](#)
`legend_render()`, [17](#), [24](#), [26–28](#)
`legend_spec`, [24](#)
`legend_spec()`, [16](#), [17](#), [24](#), [27](#), [28](#)

`legend_strip`, [18](#), [21](#), [23](#), [26](#)
`legend_subset`, [27](#)
`legend_subset()`, [17](#)

`marker_encode`, [28](#)
`marker_legend`, [14](#), [15](#), [19](#), [27](#), [29](#)

`plot_layout`, [26](#)
`process_data`, [12](#), [31](#)
`process_data()`, [3](#)

`R_user_dir`, [6](#)

`scale_legend_icon`, [32](#)

`theme_pop`, [33](#)
`theme_pop()`, [3](#)
`theme_pop_dark`, [34](#)
`theme_pop_dark()`, [3](#)
`theme_pop_minimal`, [35](#)
`theme_pop_minimal()`, [3](#)
`tibble`, [6](#)