

Package: sdvplotR (via r-universe)

October 12, 2026

Title Sports Team Logo Plots in 'ggplot2', 'gt' and 'reactable'

Version 0.1.0

Description A unified set of functions to visualize sports data across the 28 leagues of the SportsDataverse team index, including the National Football League, National Basketball Association, Women's National Basketball Association, Major League Baseball, National Hockey League, college football and basketball, and soccer, in 'ggplot2' plots, 'gt' tables and 'reactable' tables, and in interactive 'plotly', 'ggiraph' and 'leaflet' charts and maps. Team logos, wordmarks and player headshots can be drawn as plot layers, as axis text and inside table cells, in the design a team used in a given season, and team color scales map team identifiers to their official colors. Playing surfaces from 'sportyR' can be painted in a team's colors. Team identifiers from many data providers, including the codes of relocated franchises, are resolved to one canonical team id per league through the team index and logo manifest of the SportsDataverse logo archive
<<https://github.com/sportsdataverse/sdv-assets>>, read at run time with a copy bundled in the package. Built on 'ggpath' following the conventions of 'nflplotR' and designed to work with the companion SportsDataverse data packages. Also includes the 'gtUtils' table themes, color legends, cut lines, significance notation, outlier flags and cropping and saving helpers for 'gt' tables.

License MIT + file LICENSE

URL <https://sdvplotR.sportsdataverse.org/>,
<https://github.com/sportsdataverse/sdvplotR>

BugReports <https://github.com/sportsdataverse/sdvplotR/issues>

Depends R (>= 4.1.0)

Imports base64enc, cli (>= 3.0.0), commonmark, dplyr (>= 1.0.0),
fontawesome (>= 0.5.3), ggpath (>= 1.1.0), ggplot2 (>= 4.0.0),
glue (>= 1.4.0), grDevices, grid, gt (>= 0.10.0), htmltools,

magick, rlang ($\geq 1.0.0$), S7 ($\geq 0.2.0$), scales ($\geq 1.1.0$), stats, tidyselect, tools, utils

Suggests chromote, ggiraph, ggsoccer, ggtext ($\geq 0.1.1$), gridtext ($\geq 0.1.4$), leaflet, paletteer, plotly, reactable, rsvg, sportyR, testthat ($\geq 3.0.0$), webshot2, withr

Config/Needs/development devtools, httr, jsonlite, lintr, nflreadr, roxygen2, styler, usethis

Config/Needs/website r-lib/pkgdown, arrow, baseballr, cfbfastR, cfbseedR, fastRhockey, ggsoccer, ggiraph, ggrepel, ggsci, gtExtras, jsonlite, hoopR, leaflet, maps, nflfastR, nflseedR, nflreadr, oddsapiR, patchwork, plotly, kcuilla/reactablefmtr, sportyR, svglite, tidyverse, viridis, wehoop

Config/roxygen2/version 8.1.0

Config/testthat/edition 3

Encoding UTF-8

Language en-US

LazyData true

Roxygen list(markdown = TRUE)

Config/pak/sysreqs
cmake make libmagick++-dev gsfonts libuv1-dev libxml2-dev libssl-dev libnode-dev

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

Date/Publication 2026-10-12 22:09:31 UTC

RemoteUrl <https://github.com/sportsdataverse/sdvplotR>

RemoteRef HEAD

RemoteSha 31bb4e1ebd1014d5683a72a463aaa5b4c1571c28

Contents

add_sdv_logos	4
addSdvLogoMarkers	7
element_sdv_logo	8
geom_sdv_headshots	11
geom_sdv_logos	14
geom_sdv_logos_interactive	17
geom_sdv_wordmarks	19
ggtitle_image	23
gt_538_caption	24
gt_bold_rows	26
gt_border_bars_bottom	27
gt_border_bars_top	29
gt_border_grid	31
gt_color_pills	32
gt_color_ranks	34

gt_color_results	36
gt_column_subheaders	38
gt_outline	39
gt_delta	41
gt_fmt_rank	43
gt_fmt_tally	44
gt_grid	46
gt_group_stripes	49
gt_highlight_cells	50
gt_highlight_na	52
gt_indicator_boxes	53
gt_legend_continuous	56
gt_legend_discrete	59
gt_marginalia	62
gt_merge_stack_team_color	64
gt_outliers	65
gt_percentile_bar	67
gt_row_accent	70
gt_save_batch	71
gt_save_crop	73
gt_scale_note	75
gt_sdv_cols_label	76
gt_sdv_headshots	77
gt_sdv_logos	79
gt_sdv_wordmarks	80
gt_set_font	82
gt_significance	83
gt_snake	85
gt_snake_align	87
gt_social_crop	89
gt_social_tag	90
gt_spotlight	92
gt_stack_tables	94
gt_theme_almanac	97
gt_theme_athletic	98
gt_theme_booktabs	99
gt_theme_broadsheet	101
gt_theme_brutalist	102
gt_theme_drench	103
gt_theme_gtutils	105
gt_theme_kenpom	106
gt_theme_midnight	107
gt_theme_ncaa	109
gt_theme_pl	110
gt_theme_preview	111
gt_theme_savant	113
gt_theme_scoreboard	114
gt_theme_sdv	115

gt_theme_sdv_team	117
gt_theme_sofa	118
gt_theme_swiss	120
gt_theme_terminal	121
gt_theme_tier	122
gt_theme_tufte	124
gt_tiers	125
gt_title_header	127
gt_watermark	128
gt_wrap_labels	130
pal_midnight	131
reactable_sdv_cols_label	132
reactable_sdv_images	133
reactable_sdv_team_color	135
scale_axes_sdv	136
scale_sdv	139
sdv_axis_logos	140
sdv_clear_cache	142
sdv_color_palette	142
sdv_colorway	143
sdv_court_coords	144
sdv_example_standings	146
sdv_headshot_url	147
sdv_leagues	148
sdv_logo_url	149
sdv_marks	150
sdv_pitch_coords	151
sdv_resolve	153
sdv_suggest	154
sdv_surface	154
sdv_team_colors	156
sdv_team_factor	157
sdv_team_tiers	158
sdv_teams	160
sdv_versions	161
theme_bg	161
theme_sdv	162

Index**164**

add_sdv_logos

Team logos, wordmarks and headshots on plotly charts

Description

Places team marks on a `plotly::plot_ly()` chart at data coordinates, as layout images. The R counterpart of `sdvplot`'s `plotly` adapter. Works on native `plotly` charts; `plotly::ggplotly()` drops layout images, so draw a `ggplot` with `geom_sdv_logos()` or `geom_sdv_logos_interactive()` instead.

Usage

```
add_sdv_logos(  
  p,  
  x,  
  y,  
  team,  
  league,  
  size = 0.08,  
  season = NULL,  
  variant = "default",  
  id_system = "auto",  
  strict = FALSE,  
  xref = "x",  
  yref = "y"  
)
```

```
add_sdv_wordmarks(  
  p,  
  x,  
  y,  
  team,  
  league,  
  size = 0.08,  
  season = NULL,  
  variant = "default",  
  id_system = "auto",  
  strict = FALSE,  
  xref = "x",  
  yref = "y"  
)
```

```
add_sdv_headshots(  
  p,  
  x,  
  y,  
  player_id,  
  league,  
  size = 0.08,  
  id_system = "espn",  
  xref = "x",  
  yref = "y"
```

)

Arguments

<code>p</code>	A plotly object.
<code>x, y</code>	Positions: numbers, or category names on a categorical axis.
<code>team</code>	Team identifiers, one per position.
<code>league</code>	A league key from <code>sdv_leagues()</code> . Required: one abbreviation means different teams in different leagues.
<code>size</code>	Each image is fitted by height to <code>size</code> times the y-axis span (see Details), so <code>0.08</code> is 8% of the axis. The width follows the image's aspect ratio.
<code>season</code>	NULL for today's mark, or a season (or one per team): the mark the team wore that season where the archive dates its marks.
<code>variant</code>	"default", "dark", or a named variant from <code>sdv_marks()</code> ("scoreboard", "grayscale", "cap_on_light", ...). A team without it gets its closest mark of the same polarity; a name no mark in the archive has is an error listing the league's variants.
<code>id_system</code>	For headshots, as in <code>sdv_headshot_url()</code> .
<code>strict</code>	If TRUE, a value that does not resolve is an error; otherwise it becomes NA with one warning per call.
<code>xref, yref</code>	The axes the positions refer to ("x2" for a subplot's).
<code>player_id</code>	Player ids, one per position.

Details

Images are placed in data units and fitted by height: each is sized to `size` times the y-axis span, as `sdvplot`'s plotly adapter sizes them. The span is the axis range when one is set, and otherwise everything the chart's traces on that axis cover plus the new marks (the number of categories on a categorical axis), so marks for a subset of the plotted points are sized against the whole axis. Images already on the chart, including ones set with `plotly::layout()`, are kept. Date axes are not supported yet.

Value

The plotly object (an `htmlwidget`) with the images added.

Examples

```
if (requireNamespace("plotly", quietly = TRUE)) {
  d <- data.frame(
    team = c("KC", "BUF", "BAL"), off = c(0.12, 0.15, 0.18), def = c(-0.05, 0.02, -0.08)
  )
  plotly::plot_ly(d, x = ~off, y = ~def, type = "scatter", mode = "markers", text = ~team) |>
    add_sdv_logos(d$off, d$def, d$team, "nfl")
}
```

addSdvLogoMarkers	<i>Team logo markers on a leaflet map</i>
-------------------	---

Description

Adds a marker per row with the team's logo as its icon. A row whose team has no logo keeps leaflet's default marker, so no point is dropped silently.

Usage

```
addSdvLogoMarkers(
  map,
  lng,
  lat,
  team,
  league,
  size = 32,
  season = NULL,
  variant = "default",
  id_system = "auto",
  strict = FALSE,
  data = leaflet::getMapData(map),
  ...
)
```

Arguments

map	A leaflet map.
lng, lat	Longitudes and latitudes: vectors, or formulas read from data.
team	Team identifiers: a vector, or a formula read from data.
league	A league key from sdv_leagues() . Required: one abbreviation means different teams in different leagues.
size	Icon width and height, in pixels.
season	NULL for today's logo, or a season (or one per row): a vector, or a formula read from data.
variant	"default", "dark", or a named variant from sdv_marks() ("scoreboard", "grayscale", "cap_on_light", ...). A team without it gets its closest mark of the same polarity; a name no mark in the archive has is an error listing the league's variants.
id_system	"auto" (try every id system in priority order) or one id system: "team_id", "espn", "espn_abbr", "nhl", "nflverse", "mlbstats", "nba_api", "hockeytech", "ncaa", "pff", "cricinfo", "cfbd", "bref", "sportsipy", "fangraphs", "sdvplotr", "name", or "nhl_id" (NHL stats API ids, which overlap ESPN's, so "auto" never reads a number as one).

strict	If TRUE, a value that does not resolve is an error; otherwise it becomes NA with one warning per call.
data	The data formulas read; defaults to the map's.
...	Passed to <code>leaflet::addMarkers()</code> (label, popup, layerId, group, ...); a vector one per row is split with the rows.

Value

The leaflet map (an htmlwidget) with the markers added.

Examples

```
if (requireNamespace("leaflet", quietly = TRUE)) {
  d <- data.frame(team = c("KC", "BUF"), lon = c(-94.48, -78.79), lat = c(39.05, 42.77))
  leaflet::leaflet(d) |>
    leaflet::addTiles() |>
    addSdvLogoMarkers(~lon, ~lat, ~team, league = "nfl", label = ~team)
}
```

element_sdv_logo

Theme Elements for Image Grobs

Description

In conjunction with the `ggplot2::theme` system, the following `element_` functions enable images in non-data components of the plot, e.g. axis text.

- `element_sdv_logo()`: draws team logos instead of their abbreviations.
- `element_sdv_wordmark()`: draws team wordmarks instead of their abbreviations.
- `element_sdv_headshot()`: draws player headshots instead of their IDs.
- `element_sdv_raster()`: draws a single image in a rectangular theme element such as `plot.background`. A thin wrapper around `ggpath::element_raster()`.

Usage

```
element_sdv_logo(
  league,
  alpha = NULL,
  colour = NA,
  color = NULL,
  hjust = NULL,
  vjust = NULL,
  size = 0.5,
  season = NULL,
  id_system = "auto"
)
```

```

element_sdv_wordmark(
  league,
  alpha = NULL,
  colour = NA,
  color = NULL,
  hjust = NULL,
  vjust = NULL,
  size = 0.5,
  season = NULL,
  id_system = "auto"
)

element_sdv_headshot(
  league,
  alpha = NULL,
  colour = NA,
  color = NULL,
  hjust = NULL,
  vjust = NULL,
  size = 0.5,
  id_system = "espn"
)

element_sdv_raster(
  image_path,
  x = grid::unit(0.5, "npc"),
  y = grid::unit(0.5, "npc"),
  width = grid::unit(1, "npc"),
  height = grid::unit(1, "npc"),
  just = "centre",
  hjust = 0.5,
  vjust = 0.5,
  interpolate = TRUE,
  ...
)

```

Arguments

league	A league key from sdv_leagues() .
alpha	The alpha channel (transparency) between 0 and 1.
colour, color	The image will be colorized with this color. Use "b/w" for black and white.
hjust	Horizontal justification.
vjust	Vertical justification.
size	The output grob size in cm.
season	NULL for today's marks, or one season: the marks the teams wore that season. One element draws every label, so one season.

`id_system` "auto" (try every id system in priority order) or one id system: "team_id", "espn", "espn_abbr", "nhl", "nflverse", "mlbstats", "nba_api", "hockeytech", "ncaa", "pff", "cricinfo", "cfbd", "bref", "sportsipy", "fangraphs", "sdvplotr", "name", or "nhl_id" (NHL stats API ids, which overlap ESPN's, so "auto" never reads a number as one).

`image_path` A file path or url to an image.

`x, y, width, height, just, interpolate` Passed on to `ggpath::element_raster()`.

`...` Other arguments passed on to `ggpath::element_raster()`.

Details

The elements translate team abbreviations or player IDs into logo images or player headshots for the specified league. Rendering is delegated to `ggpath::element_path()`, which they extend. Set on a parent element such as `axis.text.x`, they also replace the position children (`axis.text.x.bottom`, `axis.text.y.left`, ...) that complete themes like `ggplot2::theme_minimal()` set, keeping the child's spacing.

Value

`element_sdv_logo()`, `element_sdv_wordmark()` and `element_sdv_headshot()` return a theme element extending `ggpath::element_path()`; `element_sdv_raster()` returns a `ggpath::element_raster()`.

See Also

`ggpath::element_path()`, `ggpath::element_raster()`

Examples

```
library(sdvplotR)
library(ggplot2)

team_abbr <- sdv_teams("nfl")$abbr[1:16]

df <- data.frame(
  random_value = runif(length(team_abbr), 0, 1),
  teams = team_abbr
)

# use logos for x-axis
ggplot(df, aes(x = teams, y = random_value)) +
  geom_col(aes(color = teams, fill = teams), width = 0.5) +
  scale_color_sdv(league = "nfl", which = "secondary") +
  scale_fill_sdv(league = "nfl", alpha = 0.4) +
  theme_minimal() +
  theme(axis.text.x = element_sdv_logo(league = "nfl"))
```

geom_sdv_headshots *ggplot2 Layer for Visualizing Player Headshots*

Description

This geom is used to plot player headshots instead of points in a ggplot. It requires x, y aesthetics as well as a valid player identifier: an ESPN athlete ID by default, in every league. Set `id_system` to plot IDs from another source (nflverse's gsis IDs, or the league's own player IDs). A team's ESPN athlete IDs come with its roster: `hoopR::espn_mbb_team_roster()` and `espn_nba_team_roster()`, `wehoop::espn_wbb_team_roster()` and `espn_wnba_team_roster()`, and `cfbfastR::espn_cfb_team_roster()` each return an `athlete_id` column.

Usage

```
geom_sdv_headshots(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  league,
  id_system = "espn",
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = TRUE
)
```

GeomSDVheadshot

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	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 ggplot(). 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. 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>).
stat	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:

	• A Stat ggproto subclass, for example StatCount. • 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 "count". • For more information and other ways to specify the stat, see the layer stat documentation.
position	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: • 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.
...	Other arguments passed on to <code>layer()</code>'s <code>params</code> argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the <code>position</code> argument, or aesthetics that are required can <i>not</i> be passed through ... Unknown arguments that are not part of the 4 categories below are ignored. • Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, <code>colour = "red"</code> or <code>linewidth = 3</code>. The geom's documentation has an Aesthetics section that lists the available options. The 'required' aesthetics cannot be passed on to the <code>params</code>. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data. • When constructing a layer using a <code>stat_*()</code> function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is <code>stat_density(geom = "area", outline.type = "both")</code>. The geom's documentation lists which parameters it can accept. • Inversely, when constructing a layer using a <code>geom_*()</code> function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is <code>geom_area(stat = "density", adjust = 0.5)</code>. The stat's documentation lists which parameters it can accept. • The <code>key_glyph</code> argument of <code>layer()</code> may also be passed on through ... This can be one of the functions described as key glyphs, to change the display of the layer in the legend.
league	A league key from <code>sdv_leagues()</code>. Required: one abbreviation means different teams in different leagues.
id_system	"espn" (ESPN athlete ids; the default, for every league with ESPN headshots: nfl, nba, wnba, mlb, nhl, cfb, mbb, wbb), "gsis" (nflverse's NFL gsis ids, such as "00-0033873", read through a published map) or "league" (the league's own player id: NBA / WNBA Stats PERSON_ID, MLBAM, NHL API; for the NFL, the gsis id).

<code>na.rm</code>	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
<code>show.legend</code>	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.
<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> .

Details

ESPN and league IDs are plain digits, so a mismatch draws the wrong player or no image rather than an error. League CDNs draw a silhouette for an unknown ID, and the NBA and WNBA CDNs refuse requests from datacenter IPs, so a plot drawn on CI or a server can come back without those headshots.

Value

A ggplot2 layer.

Aesthetics

`geom_sdv_headshots()` understands the following aesthetics (required aesthetics are in bold):

x - The x-coordinate.

y - The y-coordinate.

player_id - The player's ID: ESPN athlete ID, or as `id_system` says.

`alpha = NULL` - The alpha channel.

`colour = NULL` - The image will be colorized with this colour. Use "b/w" for black and white.

`angle = 0` - The angle of the image.

`hjust = 0.5` - The horizontal adjustment.

`vjust = 0.5` - The vertical adjustment.

`width = 1.0` - The desired width in npc.

`height = 1.0` - The desired height in npc.

Examples

```
library(sdvplotR)
library(ggplot2)

df <- data.frame(
  a = 1:3,
  b = 3:1,
  player_id = c("3139477", "12483", "3917315"),
  player_name = c("P.Mahomes", "M.Stafford", "K.Murray")
)
```

```

)

ggplot(df, aes(x = a, y = b)) +
  geom_sdv_headshots(aes(player_id = player_id), league = "nfl", height = 0.2) +
  geom_label(aes(label = player_name), nudge_y = -0.35, alpha = 0.5) +
  coord_cartesian(xlim = c(0.5, 3.5), ylim = c(0.5, 3.5)) +
  theme_void()

```

geom_sdv_logos

ggplot2 Layer for Visualizing Sports Team Logos

Description

This geom is used to plot sports team logos instead of points in a ggplot. It requires x, y aesthetics as well as a valid team identifier: anything `sdv_resolve()` accepts, or a conference or league key. A league's teams are listed by `sdv_teams()`.

Usage

```

geom_sdv_logos(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  league,
  id_system = "auto",
  strict = FALSE,
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = TRUE
)

```

GeomSDVlogo

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

	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>).
stat	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: • 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 "count". • For more information and other ways to specify the stat, see the layer stat documentation.
position	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: • 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.
...	Other arguments passed on to <code>layer()</code>'s <code>params</code> argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the <code>position</code> argument, or aesthetics that are required can <i>not</i> be passed through Unknown arguments that are not part of the 4 categories below are ignored. • Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, <code>colour = "red"</code> or <code>linewidth = 3</code>. The geom's documentation has an Aesthetics section that lists the available options. The 'required' aesthetics cannot be passed on to the <code>params</code>. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data. • When constructing a layer using a <code>stat_*()</code> function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is <code>stat_density(geom = "area", outline.type = "both")</code>. The geom's documentation lists which parameters it can accept. • Inversely, when constructing a layer using a <code>geom_*()</code> function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is <code>geom_area(stat = "density", adjust = 0.5)</code>. The stat's documentation lists which parameters it can accept. • The <code>key_glyph</code> argument of <code>layer()</code> may also be passed on through This can be one of the functions described as key glyphs, to change the display of the layer in the legend.

league	A league key from <code>sdv_leagues()</code> . Required: one abbreviation means different teams in different leagues.
id_system	"auto" (try every id system in priority order) or one id system: "team_id", "espn", "espn_abbr", "nhl", "nflverse", "mlbstats", "nba_api", "hockeytech", "ncaa", "pff", "cricinfo", "cfbd", "bref", "sportsipy", "fangraphs", "sdvplotr", "name", or "nhl_id" (NHL stats API ids, which overlap ESPN's, so "auto" never reads a number as one).
strict	If TRUE, a value that does not resolve is an error; otherwise it becomes NA with one warning per call.
na.rm	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
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. <code>annotation_borders()</code> .

Value

A ggplot2 layer (`ggplot2::layer()`) that can be added to a plot created with `ggplot2::ggplot()`.

Aesthetics

`geom_sdv_logos()` understands the following aesthetics (required aesthetics are in bold):

x - The x-coordinate.

y - The y-coordinate.

team - The team: an abbreviation, name or id, resolved by `sdv_resolve()`.

`alpha = NULL` - The alpha channel, i.e. transparency level, as a numerical value between 0 and 1.

`colour = NULL` - The image will be colored with this colour. Use the special character "b/w" to set it to black and white.

`angle = 0` - The angle of the image as a numerical value between 0 and 360 degrees.

`hjust = 0.5` - The horizontal adjustment relative to the given x coordinate.

`vjust = 0.5` - The vertical adjustment relative to the given y coordinate.

`width = 1.0` - The desired width of the image in npc (Normalised Parent Coordinates). A typical size is `width = 0.075`.

`height = 1.0` - The desired height of the image in npc. A typical size is `height = 0.1`.

`season = NA` - The season whose logo to draw: the ending year for the NHL (1990 for 1989-90), the year elsewhere. NA draws the team's logo today. See the Historical logos section.

Historical logos

Given a season, a team is drawn with the mark it wore that season where the SportsDataverse logo archive dates its marks, and with its current mark otherwise. The season also picks the team a reused code meant then (see [sdv_resolve\(\)](#)). The images are copies kept in the archive, so they don't change when a league reuses a file name. SVG marks are read with the 'rsvg' package. season takes single years: the ending year for split seasons (2005 for 2004-05).

Examples

```
library(sdvplotR)
library(ggplot2)

team_abbr <- sdv_teams("nfl")$abbr[1:8]

df <- data.frame(
  a = rep(1:4, 2),
  b = rep(2:1, each = 4),
  teams = team_abbr
)

ggplot(df, aes(x = a, y = b)) +
  geom_sdv_logos(aes(team = teams), league = "nfl", width = 0.075) +
  geom_label(aes(label = teams), nudge_y = -0.35, alpha = 0.5) +
  theme_void()

# the logo each team wore that season (NHL seasons are ending years;
# the NHL's marks are SVG files, read with the 'rsvg' package)
if (requireNamespace("rsvg", quietly = TRUE)) {
  old <- data.frame(
    x = 1:4, y = 1,
    team = c("QUE", "HFD", "ATL", "WPG"),
    season = c(1990, 1995, 2005, 2015)
  )
  ggplot(old, aes(x = x, y = y)) +
    geom_sdv_logos(aes(team = team, season = season), league = "nhl", width = 0.2) +
    theme_void()
}
```

geom_sdv_logos_interactive

Interactive team logos, wordmarks and headshots (ggiraph)

Description

[geom_sdv_logos\(\)](#) and its siblings with ggiraph's interactivity: map tooltip, data_id or onclick, then render with [ggiraph::girafe\(\)](#). Each returns two layers: the images, and an invisible interactive point over each that takes the hover.

Usage

```
geom_sdv_logos_interactive(
  mapping = NULL,
  data = NULL,
  ...,
  league,
  hover_size = 8,
  max_px = 160,
  id_system = "auto",
  strict = FALSE,
  inherit.aes = TRUE
)

geom_sdv_wordmarks_interactive(
  mapping = NULL,
  data = NULL,
  ...,
  league,
  hover_size = 8,
  max_px = 160,
  id_system = "auto",
  strict = FALSE,
  inherit.aes = TRUE
)

geom_sdv_headshots_interactive(
  mapping = NULL,
  data = NULL,
  ...,
  league,
  hover_size = 8,
  max_px = 160,
  id_system = "espn",
  inherit.aes = TRUE
)
```

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	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 ggplot(). 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.

	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>).
<code>...</code>	Passed to the image layer (width, height, alpha, ...).
<code>league</code>	A league key from <code>sdv_leagues()</code> . Required: one abbreviation means different teams in different leagues.
<code>hover_size</code>	The size of the invisible hover target, in ggplot2 point size.
<code>max_px</code>	The longest side, in pixels, of each image embedded in the page. ggiraph writes every image into the widget, and a source logo (500 px) or headshot is 20-300 KB, so 30 of them make a page of megabytes; 160 px stays sharp at the size these geoms draw. NULL embeds the source.
<code>id_system</code>	For headshots, as in <code>sdv_headshot_url()</code> .
<code>strict</code>	If TRUE, a value that does not resolve is an error; otherwise it becomes NA with one warning per call.
<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> .

Value

A list of two ggplot2 layers, to add with `+`.

Examples

```
if (requireNamespace("ggiraph", quietly = TRUE)) {
  library(ggplot2)
  d <- data.frame(x = 1:3, y = c(2, 3, 1), team = c("KC", "BUF", "BAL"))
  p <- ggplot(d, aes(x, y)) +
    geom_sdv_logos_interactive(
      aes(team = team, tooltip = team, data_id = team),
      league = "nfl", width = 0.1
    )
  ggiraph::girafe(ggobj = p)
}
```

geom_sdv_wordmarks

ggplot2 Layer for Visualizing Sports Team Wordmarks

Description

This geom is used to plot sports team wordmarks instead of points in a ggplot. It requires `x`, `y` aesthetics as well as a valid team identifier: anything `sdv_resolve()` accepts. A league's teams are listed by `sdv_teams()`.

Usage

```
geom_sdv_wordmarks(
  mapping = NULL,
  data = NULL,
  stat = "identity",
  position = "identity",
  ...,
  league,
  id_system = "auto",
  strict = FALSE,
  na.rm = FALSE,
  show.legend = FALSE,
  inherit.aes = TRUE
)
```

GeomSDVwordmark

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>. • 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 <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 position_ 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.
...	Other arguments passed on to <code>layer()</code>'s <code>params</code> argument. These arguments broadly fall into one of 4 categories below. Notably, further arguments to the position argument, or aesthetics that are required can <i>not</i> be passed through ... Unknown arguments that are not part of the 4 categories below are ignored. • Static aesthetics that are not mapped to a scale, but are at a fixed value and apply to the layer as a whole. For example, <code>colour = "red"</code> or <code>linewidth = 3</code>. The geom's documentation has an Aesthetics section that lists the available options. The 'required' aesthetics cannot be passed on to the <code>params</code>. Please note that while passing unmapped aesthetics as vectors is technically possible, the order and required length is not guaranteed to be parallel to the input data. • When constructing a layer using a <code>stat_*()</code> function, the ... argument can be used to pass on parameters to the geom part of the layer. An example of this is <code>stat_density(geom = "area", outline.type = "both")</code>. The geom's documentation lists which parameters it can accept. • Inversely, when constructing a layer using a <code>geom_*()</code> function, the ... argument can be used to pass on parameters to the stat part of the layer. An example of this is <code>geom_area(stat = "density", adjust = 0.5)</code>. The stat's documentation lists which parameters it can accept. • The <code>key_glyph</code> argument of <code>layer()</code> may also be passed on through ... This can be one of the functions described as key glyphs, to change the display of the layer in the legend.
league	A league key from <code>sdv_leagues()</code> . A team the logo archive has no wordmark for is not drawn (its path is NA, with a warning).
id_system	"auto" (try every id system in priority order) or one id system: "team_id", "espn", "espn_abbr", "nhl", "nflverse", "mlbstats", "nba_api", "hockeytech", "ncaa", "pff", "cricinfo", "cfbd", "bref", "sportsipy", "fangraphs", "sdvplotr", "name", or "nhl_id" (NHL stats API ids, which overlap ESPN's, so "auto" never reads a number as one).
strict	If TRUE, a value that does not resolve is an error; otherwise it becomes NA with one warning per call.
na.rm	If FALSE, the default, missing values are removed with a warning. If TRUE, missing values are silently removed.
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. <code>annotation_borders()</code> .

Value

A ggplot2 layer (`ggplot2::layer()`) that can be added to a plot created with `ggplot2::ggplot()`.

Aesthetics

`geom_sdv_wordmarks()` understands the following aesthetics (required aesthetics are in bold):

x - The x-coordinate.

y - The y-coordinate.

team - The team: an abbreviation, name or id, resolved by `sdv_resolve()`.

alpha = NULL - The alpha channel, i.e. transparency level, as a numerical value between 0 and 1.

colour = NULL - The image will be colorized with this colour. Use "b/w" for black and white.

angle = 0 - The angle of the image as a numerical value between 0 and 360 degrees.

hjust = 0.5 - The horizontal adjustment relative to the given x coordinate.

vjust = 0.5 - The vertical adjustment relative to the given y coordinate.

width = 1.0 - The desired width of the image in npc. A typical size is `width = 0.12`.

height = 1.0 - The desired height of the image in npc. A typical size is `height = 0.1`.

season = NA - The season whose wordmark to draw, as for `geom_sdv_logos()`.

Examples

```
library(sdvplotR)
library(ggplot2)

team_abbr <- sdv_teams("nfl")$abbr[1:8]

df <- data.frame(
  a = rep(1:4, 2),
  b = sort(rep(1:2, 4), decreasing = TRUE),
  teams = team_abbr
)

ggplot(df, aes(x = a, y = b)) +
  geom_sdv_wordmarks(aes(team = teams), league = "nfl", width = 0.12) +
  geom_label(aes(label = teams), nudge_y = -0.20, alpha = 0.5) +
  scale_x_continuous(expand = expansion(add = 0.5)) +
  theme_void()
```

Description

These functions work together to place an image to the left or right of the title in a ggplot. `ggtitle_image()` is the main function but must be used with `theme_title_image()`, which renders the title as mark-down and centers the image on the title text.

Usage

```
ggtitle_image(
  title_image = ggplot2::waiver(),
  title = ggplot2::waiver(),
  image_height = 15,
  image_side = c("left", "right"),
  subtitle = ggplot2::waiver(),
  league
)

theme_title_image(...)
```

Arguments

<code>title_image</code>	The URL of the image to add to the title. If a valid team identifier for league, the team logo will be used.
<code>title</code>	The text for the title.
<code>image_height</code>	The height of the image in pixels.
<code>image_side</code>	One of "left" or "right". Places the image on either side of the title text.
<code>subtitle</code>	Optional text for the subtitle.
<code>league</code>	A league key from <code>sdv_leagues()</code> .
<code>...</code>	Other arguments passed on to <code>ggtext::element_markdown()</code> , such as size, face and hjust.

Details

The title is markdown with an inline `<img>` tag. 'gridtext' draws inline images on the text baseline and ignores CSS vertical-align, so a logo taller than the text would rise above it (ESPN's marks, padded with transparent space, can sit wholly above the title). `theme_title_image()` lines the middle of each image up with the middle of the capital letters on its line. Setting `plot.title` to a plain `ggtext::element_markdown()` instead also renders the image, on the baseline.

Style the title through `theme_title_image(...)` (size, face, hjust, ...), never with a later `theme(plot.title = ...)`: adding `theme(plot.title = ggtext::element_markdown(hjust = 0.5))` after it replaces the centering element with a plain markdown one, so the image drops back to the baseline, and `theme(plot.title = element_text(...))` is an error. Add `theme_title_image()` after any complete theme such as `ggplot2::theme_minimal()`, which replaces every element.

Value

A ggplot2 labs object (for ggtitle_image) or theme object (for theme_title_image).

See Also

[theme_title_image\(\)](#)

Examples

```
library(sdvplotR)
library(ggplot2)

nfl <- subset(sdv_example_standings, league == "nfl")
p <- ggplot(nfl, aes(x = points_for, y = points_against)) +
  geom_point() +
  labs(
    title = "This Title will be overwritten",
    subtitle = "Points scored and allowed, 2025 regular season"
  )

if (requireNamespace("ggtext", quietly = TRUE)) {
  p +
    ggtitle_image(
      title_image = "SEA",
      title = "Seattle Seahawks Analysis",
      image_height = 20,
      image_side = "left",
      league = "nfl"
    ) +
    theme_title_image(size = 20, hjust = 0.5)
}
```

gt_538_caption

Add a 538-style caption to a gt table

Description

Adds a caption block beneath the table with a rule under its top line, in the style of FiveThirtyEight tables. The top_caption renders as a footnote with a border drawn under it, and the bottom_caption renders as a right-aligned source note below it. Both accept markdown.

Usage

```
gt_538_caption(
  gt_object,
  top_caption = NULL,
  bottom_caption = NULL,
  rule_color = NULL,
```

```

    rule_width = 1,
    size = 12,
    align = "right",
    ...
  )

```

Arguments

<code>gt_object</code>	A gt table object to modify.
<code>top_caption</code>	Character. Text rendered as a footnote beneath the table, with a rule drawn under it. Accepts markdown. Defaults to NULL, which leaves out the footnote and, with it, the rule.
<code>bottom_caption</code>	Character. Text rendered as a source note below the top caption, aligned by align. Accepts markdown. Defaults to NULL, which shows only the top caption.
<code>rule_color</code>	Optional. A hex color for the rule under the top caption. When NULL, the color is taken from the rendered table so it tracks the theme. Defaults to NULL.
<code>rule_width</code>	Numeric. The rule width in pixels. Defaults to 1.
<code>size</code>	Numeric. The font size of the top caption in pixels. Defaults to 12.
<code>align</code>	Character. The alignment of the bottom caption. Defaults to "right".
<code>...</code>	Additional arguments. Currently unused.

Details

The top caption is attached with `gt::tab_footnote()` on the column labels, which places it in the table footer. Scoped CSS then hides the footnote mark and draws a bottom border under the footnote, so it reads as a ruled caption rather than a numbered note. When `rule_color` is NULL the border color is taken from the first text color found in the rendered table (via `htmltools::as.tags()`), so it tracks the theme across light and dark color modes, and falls back to a neutral gray when the render carries no color. The rules are keyed on the table id. If the table has none, one is generated and assigned.

Apply the theme before this function. The rule color is read off the rendered table, so calling it first on a dark theme borrows a near-black (`#333333`) and the rule disappears into the background. Pass `rule_color` to sidestep the question entirely.

Value

Returns a modified gt table with the styled captions.

Examples

```

library(gt)

nfc_north <- subset(
  sdv_example_standings, division == "NFC North",
  c(team_name, wins:points_against)
)

```

```
gt(nfc_north) |>
  gt_538_caption(
    top_caption = "The NFC North in 2025",
    bottom_caption = "Source: *nflreadr* schedules, ranked by *nflseedR*"
  )
```

gt_bold_rows	<i>Bold rows in a gt table</i>
--------------	--------------------------------

Description

Bolds the body cells of chosen rows, with an option to recolor their text and fill their background. Rows are chosen by index or by a filter expression; with neither, every row is bolded.

Usage

```
gt_bold_rows(
  gt_object,
  rows = NULL,
  text_color = "black",
  highlight_color = NULL
)
```

Arguments

- gt_object A gt table object to modify.
- rows The rows to bold. Either an expression evaluated against the table’s data, such as wins > 10, or a numeric vector of row indices. If NULL, every row is bolded. Defaults to NULL.
- text_color Character. The text color for the bolded rows. Defaults to "black".
- highlight_color Character. The background fill for the bolded rows. Set to NULL for no fill. Defaults to NULL.

Details

rows is evaluated against the table’s underlying data, so rows = wins > 10 bolds the rows where that is TRUE; a numeric vector picks rows by index. The styling is applied with a single gt::tab_style() over gt::cells_body(), so it covers every column of the chosen rows.

Value

Returns a modified gt table with the chosen rows bolded.

Examples

```
library(gt)

nfc_west <- subset(
  sdv_example_standings, division == "NFC West",
  c(team_name, wins:points_against)
)

# bold every row
gt(nfc_west) |> gt_bold_rows()

# bold and fill the teams with a winning record
gt(nfc_west) |>
  gt_bold_rows(rows = win_pct > 0.5, highlight_color = "#FF3B0")
```

`gt_borderBars_bottom` *Add horizontal bars to the bottom of a gt table*

Description

Adds a row of horizontal color bars below a gt table from a vector of hex colors, optionally carrying an image and a line of text. The bars are attached as a source note, so they sit below everything else. It is the bottom-edge counterpart to [gt_borderBars_top\(\)](#).

Usage

```
gt_borderBars_bottom(
  gt_object,
  colors,
  bar_height = 10,
  bar_width = "100%",
  bar_align = "center",
  img = NULL,
  img_width = 30,
  img_height = 30,
  img_padding = 10,
  img_align = "right",
  text = NULL,
  text_weight = "bold",
  text_color = "#FFFFFF",
  text_size = 18,
  text_align = "left",
  text_padding = 10
)
```

Arguments

gt_object	A gt table object to modify.
colors	Character. Hex color codes, one per bar. When img or text is supplied only the first color is used.
bar_height	Numeric. The height of the bars in pixels. Defaults to 10.
bar_width	Character. The width of the bar block, as a CSS width. Defaults to "100%".
bar_align	Character. Alignment of the block when bar_width is under "100%". One of "left", "center", "right". Defaults to "center".
img	Optional. A URL for an image to render in the bar. It is decorative, so it carries an empty alt. Defaults to NULL.
img_width	Numeric. The image width in pixels. Defaults to 30.
img_height	Numeric. The image height in pixels. Defaults to 30.
img_padding	Numeric. Padding around the image in pixels, so it does not touch the edge. Defaults to 10.
img_align	Character. The side the image padding is applied to, "left" or "right"; any other value applies no padding. Defaults to "right".
text	Optional. Text to display in the bar. Defaults to NULL.
text_weight	Character. The font weight of the text. Defaults to "bold".
text_color	Character. The text color. Defaults to "#FFFFFF".
text_size	Numeric. The font size in pixels. Defaults to 18.
text_align	Character. The side the text padding is applied to, "left" or "right"; any other value applies no padding. Defaults to "left".
text_padding	Numeric. Padding around the text in pixels. Defaults to 10.

Details

The bars are added with `gt::tab_source_note()`, and a table id is resolved or generated so scoped CSS can zero the source-note padding. With neither `img` nor `text`, each entry in `colors` becomes its own full-width bar stacked in a block. When `img` or `text` is supplied, a single bar is drawn in the first color as a flex row with the text at one end and the image at the other. The text font is read from the table's source-note styling and imported as a Google Font, falling back to the inherited font when none is set, so it renders in an exported table.

Value

Returns a modified gt table with a row of bars below it.

Examples

```
library(gt)

afc_north <- subset(
  sdv_example_standings, division == "AFC North",
  c(team_name, wins:points_against)
```

```

)
gt(afc_north) |>
  gt_border_bars_bottom(c("#1B7837", "#FFFFFF", "#B2182B"))

# a single bar carrying a credit line
atlantic <- subset(
  sdv_example_standings, division == "Atlantic",
  c(team_name, wins, losses, win_pct:points_against)
)
gt(atlantic) |>
  gt_border_bars_bottom("#22223B", text = "Source: hoopR", bar_height = 28)

```

gt_border_bars_top	<i>Add horizontal bars to the top of a gt table</i>
--------------------	---

Description

Adds a row of horizontal color bars above a gt table from a vector of hex colors, optionally carrying an image and a line of text. The bars are attached as the table caption, so they sit above everything else.

Usage

```

gt_border_bars_top(
  gt_object,
  colors,
  bar_height = 10,
  bar_width = "100%",
  bar_align = "center",
  img = NULL,
  img_width = 30,
  img_height = 30,
  img_padding = 10,
  img_align = "right",
  text = NULL,
  text_weight = "bold",
  text_color = "#FFFFFF",
  text_size = 18,
  text_align = "left",
  text_padding = 10
)

```

Arguments

gt_object	A gt table object to modify.
colors	Character. Hex color codes, one per bar. When img or text is supplied only the first color is used.

bar_height	Numeric. The height of the bars in pixels. Defaults to 10.
bar_width	Character. The width of the bar block, as a CSS width. Defaults to "100%".
bar_align	Character. Alignment of the block when bar_width is under "100%". One of "left", "center", "right". Defaults to "center".
img	Optional. A URL for an image to render in the bar. It is decorative, so it carries an empty alt. Defaults to NULL.
img_width	Numeric. The image width in pixels. Defaults to 30.
img_height	Numeric. The image height in pixels. Defaults to 30.
img_padding	Numeric. Padding around the image in pixels, so it does not touch the edge. Defaults to 10.
img_align	Character. The side the image padding is applied to, "left" or "right"; any other value applies no padding. Defaults to "right".
text	Optional. Text to display in the bar. Defaults to NULL.
text_weight	Character. The font weight of the text. Defaults to "bold".
text_color	Character. The text color. Defaults to "#FFFFFF".
text_size	Numeric. The font size in pixels. Defaults to 18.
text_align	Character. The side the text padding is applied to, "left" or "right"; any other value applies no padding. Defaults to "left".
text_padding	Numeric. Padding around the text in pixels. Defaults to 10.

Details

The bars are added with `gt::tab_caption()`, and a table id is resolved or generated so scoped CSS can zero the caption padding. With neither `img` nor `text`, each entry in `colors` becomes its own full-width bar stacked in a block. When `img` or `text` is supplied, a single bar is drawn in the first color as a flex row with the text at one end and the image at the other. The text font is read from the table's title styling and imported as a Google Font, falling back to the inherited font when none is set, so it renders in an exported table.

Value

Returns a modified `gt` table with a row of bars above it.

Examples

```
library(gt)

afc_north <- subset(
  sdv_example_standings, division == "AFC North",
  c(team_name, wins:points_against)
)
gt(afc_north) |>
  gt_border_bars_top(c("#1B7837", "#FFFFFF", "#B2182B"))

# a single bar carrying a title
atlantic <- subset(
```

```

      sdv_example_standings, division == "Atlantic",
      c(team_name, wins, losses, win_pct:points_against)
    )
  gt(atlantic) |>
    gt_border_bars_top("#22223B", text = "Atlantic Division, 2025-26", bar_height = 34)

```

gt_border_grid	<i>Add a border grid to a gt table</i>
----------------	--

Description

Draws borders between every column and every row of a gt table, giving it a full grid, with an option to extend the borders around the column and row labels.

Usage

```
gt_border_grid(gt_object, color = "black", weight = 1, include_labels = FALSE)
```

Arguments

gt_object	A gt table object to modify.
color	Character. The border color. Defaults to "black".
weight	Numeric. The border thickness in pixels. Defaults to 1.
include_labels	Logical. Should the borders extend around the row and column labels? Defaults to FALSE.

Details

Column borders are drawn as right cell borders on every column but the last, and the row borders are added as scoped CSS on the .gt_row top border. A table id is resolved or generated first, since that CSS is keyed on #<table_id>.

Value

Returns a modified gt table with the grid borders applied.

Examples

```

library(gt)

afc_south <- subset(
  sdv_example_standings, division == "AFC South",
  c(team_name, wins:points_against)
)
gt(afc_south) |> gt_border_grid()

# heavier gray lines, including around the labels

```

```

pacific <- subset(
  sdv_example_standings, division == "Pacific",
  c(team_name, wins, losses, win_pct:points_against)
)
gt(pacific) |>
  gt_border_grid(color = "#BBBBBB", weight = 2, include_labels = TRUE)

```

gt_color_pills

Add color pills to gt table columns

Description

Renders values as rounded color pills, filled from a palette by either the raw value or the ordinal rank. Accepts a plain vector of hex colors or a paletteer palette, and an explicit domain or one taken from the data. [gt_color_ranks\(\)](#) fills the whole cell background instead of a pill.

Usage

```

gt_color_pills(
  gt_object,
  columns,
  rows = NULL,
  palette = c("#C84630", "#5DA271"),
  fill_type = "continuous",
  rank_order = "desc",
  digits = NULL,
  domain = NULL,
  format_type = "number",
  scale_percent = TRUE,
  suffix = "",
  reverse = FALSE,
  outline_color = NULL,
  outline_width = 0.25,
  pal_type = "discrete",
  pill_height = 25,
  text_color = NULL,
  na_color = NULL,
  ...
)

```

Arguments

gt_object	A gt table object to modify.
columns	The columns to fill with pills, using tidyselect.
rows	The rows to fill. Either an expression evaluated against the table's data, such as wins > 10, or a numeric vector of row indices. Rows left out keep their raw value. If NULL, every row is filled. Defaults to NULL.

palette	A vector of hex colors, or a paletteer palette given as <code>package::palette</code> . Defaults to <code>c("#C84630", "#5DA271")</code> .
fill_type	Character. Either "rank" or "continuous", choosing whether color follows ordinal rank or the raw values. Defaults to "continuous".
rank_order	Character. Either "asc" or "desc", used when fill_type is "rank". Defaults to "desc".
digits	Integer. Decimal places to round the printed value to. Defaults to NULL.
domain	Numeric. A length-2 vector giving the value range mapped onto the palette. If NULL, the observed range of the column is used and a warning is issued. Defaults to NULL.
format_type	Character. How to format the printed value. One of "number", "comma", "currency", or "percent". Defaults to "number".
scale_percent	Logical. When format_type is "percent", should values be multiplied by 100? Defaults to TRUE.
suffix	Character. A string appended to each formatted value, such as "M", "K", or "lbs". Defaults to "".
reverse	Logical. Should the palette be reversed? Defaults to FALSE.
outline_color	Optional. A hex color for a border around each pill. Defaults to NULL, no border.
outline_width	Numeric. The border width in pixels. Defaults to 0.25.
pal_type	Character. Which paletteer registry to look a <code>package::palette</code> string up in, "discrete" or "continuous". The other registry is tried as a fallback, so this rarely needs setting. Defaults to "discrete".
pill_height	Numeric. The height of each pill in pixels. Defaults to 25.
text_color	Optional. A hex color for the pill text. When NULL, the text color is chosen for contrast against each pill's fill. Defaults to NULL.
na_color	Optional. A hex color for the pill drawn over a missing value. When NULL, a missing value is left blank with no pill. Defaults to NULL.
...	Additional arguments passed to <code>scales::col_numeric</code> .

Details

Pills are drawn as HTML spans through `gt::text_transform()`, so they survive `gtsave()`. The fill is mapped with `scales::col_numeric()` over domain, and the text is set to black or white, whichever measures higher contrast against that fill, unless `text_color` is set. A missing value takes an `na_color` pill, or is left blank when `na_color` is NULL.

Selecting several columns maps them all onto **one** domain, taken from the selection as a whole when domain is unset, so their colors stay comparable. Pill width is worked out **per column**, in `ch` units, so each column's pills line up with each other rather than with the widest value in the selection. With `fill_type = "rank"`, each column is ranked against itself.

When `fill_type` is "rank", ranks are computed with `rank()` using averaged ties, then flipped when `rank_order` is "desc" so the top value anchors the high end of the palette. Leaving domain unset falls back to the observed range and warns, since the color mapping then depends on the data present.

Value

Returns a modified gt table with color pills in the selected columns.

See Also

[gt_color_ranks\(\)](#) for filling the whole cell, and [gt_legend_continuous\(\)](#) for a legend explaining the scale.

Examples

```
library(gt)

# the AFC's seven playoff teams
afc <- subset(
  sdv_example_standings, conference == "AFC" & conference_rank <= 7,
  c(team_name, wins, losses, points_for, points_against)
)

gt(afc) |> gt_color_pills(wins, domain = c(0, 17))

# several columns on one shared domain
gt(afc) |> gt_color_pills(c(points_for, points_against), domain = c(250, 550))

# tidyselect works too
gt(afc) |> gt_color_pills(starts_with("points"), domain = c(250, 550))

# only the rows that clear a threshold; the rest keep their raw value
gt(afc) |> gt_color_pills(wins, rows = wins >= 13, domain = c(0, 17))

# color by rank rather than value, with a paletteer palette
if (requireNamespace("paletteer", quietly = TRUE)) {
  gt(afc) |>
    gt_color_pills(points_for,
      fill_type = "rank", palette = "viridis::mako",
      domain = c(1, 7), digits = 0
    )
}
```

gt_color_ranks

Color ranked columns of a gt table

Description

A shorthand around `gt::data_color()` for filling columns that already hold rank values, where 1 is best. It takes either a plain vector of hex colors or a paletteer palette, and either an explicit domain or one derived from the data. Values are colored as they are; no ranking is computed for you.

Usage

```
gt_color_ranks(
  gt_object,
  columns,
  rows = NULL,
  palette = c("#3D8B6E", "#9DC5A7", "#EDE0CC", "#DB9070", "#BE4D3A"),
  domain = NULL,
  reverse = FALSE,
  na_color = "white",
  autocolour_text = TRUE,
  pal_type = "discrete",
  ...
)
```

Arguments

gt_object	A gt table object to modify.
columns	The column or columns to color.
rows	The rows to color. Either an expression evaluated against the table's data, such as <code>wins > 10</code> , or a numeric vector of row indices. Rows left out keep their default background. If <code>NULL</code> , every row is colored. Defaults to <code>NULL</code> .
palette	A color palette to use. If you want a palette from <code>paletteer</code> , specify it as <code>package::palette</code> . Defaults to a five-color green-to-red ramp.
domain	A length-2 numeric vector giving the value range mapped onto the palette. If <code>NULL</code> , the range is taken from the selected columns and shared across all of them, so rank 1 and the largest rank present anchor the two ends of the palette. Defaults to <code>NULL</code> .
reverse	Logical. Should the palette be reversed? Defaults to <code>FALSE</code> .
na_color	Character. The color used for NA values. Defaults to <code>"white"</code> .
autocolour_text	Logical. Should the text color be set automatically for contrast against the fill? Defaults to <code>TRUE</code> .
pal_type	Character. Either <code>"discrete"</code> or <code>"continuous"</code> , used when applying <code>paletteer</code> palettes. Defaults to <code>"discrete"</code> .
...	Additional arguments passed to <code>gt::data_color</code> .

Details

The whole cell background is filled. `gt_color_pills()` does the same job with rounded pill spans instead.

Value

Returns a modified gt table with the selected columns filled by value.

See Also

[gt_legend_continuous\(\)](#) for explaining the scale, and [gt_color_pills\(\)](#) for a pill treatment instead of a full-cell fill.

Examples

```
library(gt)

west <- subset(sdv_example_standings, league == "nba" & conference == "Western")
ranked <- data.frame(
  team = west$team_name,
  offense_rank = rank(-west$points_for, ties.method = "min"),
  defense_rank = rank(west$points_against, ties.method = "min")
)

gt(ranked) |> gt_color_ranks(c(offense_rank, defense_rank))

# a paletteer palette, reversed
if (requireNamespace("paletteer", quietly = TRUE)) {
  gt(ranked) |>
    gt_color_rank(
      offense_rank,
      palette = "viridis::mako", pal_type = "continuous",
      reverse = TRUE
    )
}
```

gt_color_results

Color rows of a gt table by win, loss, or tie result

Description

Fills and recolors each row according to a result read from a column. Results can be encoded as "W" and "L" or as 1 and 0, with an optional tie state. Rows holding none of the values are left unchanged.

Usage

```
gt_color_results(
  gt_object,
  result_column = "result",
  win_color = "#5DA271",
  loss_color = "#C84630",
  tie_color = NULL,
  wins_text_color = "white",
  loss_text_color = "white",
  tie_text_color = "white",
  tie_value = "T",
  result_type = "w1"
)
```

Arguments

gt_object	A gt table object to modify.
result_column	The column holding the result indicators, as a bare column name or a string. Defaults to "result".
win_color	Character. The background fill for winning rows. Defaults to "#5DA271".
loss_color	Character. The background fill for losing rows. Defaults to "#C84630".
tie_color	Optional. The background fill for tie rows. When NULL, ties are not colored. Defaults to NULL.
wins_text_color	Character. The text color for winning rows. Defaults to "white".
loss_text_color	Character. The text color for losing rows. Defaults to "white".
tie_text_color	Character. The text color for tie rows. Defaults to "white".
tie_value	The value in result_column marking a tie, used when tie_color is set. Defaults to "T".
result_type	Character. The encoding of result_column. Either "wl" for "W" and "L", or "binary" for 1 and 0. Defaults to "wl".

Details

A `gt::tab_style()` pass fills and recolors the body rows whose `result_column` equals the win value, then the loss value, and then, when `tie_color` is set, the `tie_value`. Under `result_type = "wl"` the win and loss values compared are "W" and "L"; under "binary" they are 1 and 0. Any row matching none of the values keeps its existing styling.

Value

Returns a modified gt table with winning and losing rows colored.

Examples

```
library(gt)

results <- data.frame(
  game = paste("Game", 1:4),
  pts = c(88, 74, 102, 65),
  result = c("W", "L", "W", "L")
)

gt(results) |> gt_color_results()

# binary encoding, with custom colors
results$result <- c(1, 0, 1, 0)
gt(results) |>
  gt_color_results(
    result_type = "binary", win_color = "#1B7837",
    loss_color = "#762A83"
  )
```

gt_column_subheaders *Add stacked column headers with subtitles to a gt table*

Description

Replaces each column label with a two-line header: a main heading stacked over a smaller subtitle. The heading and subtitle colors and font weights are set through arguments, and the per-column heading and subtitle text is supplied through . . .

Usage

```
gt_column_subheaders(
  gt_object,
  heading_color = "black",
  subtitle_color = "#808080",
  heading_weight = "bold",
  subtitle_weight = "normal",
  heading_size = 14,
  subtitle_size = 10,
  font = NULL,
  ...
)
```

Arguments

gt_object	A gt table object to modify.
heading_color	Character. Color for the main heading text. Defaults to "black".
subtitle_color	Character. Color for the subtitle text. Defaults to "#808080".
heading_weight	Character. Font weight for the main heading. Defaults to "bold".
subtitle_weight	Character. Font weight for the subtitle. Defaults to "normal".
heading_size	Numeric. Font size of the main heading in pixels. Defaults to 14.
subtitle_size	Numeric. Font size of the subtitle in pixels. Defaults to 10.
font	Optional. A font family applied to both lines. It is set as a CSS font-family and is not imported, so it must be available on the machine rendering the table or loaded by the theme. Defaults to NULL.
...	Named arguments where each name is a column in the gt table and each value is a list with two elements: heading (the main heading) and subtitle (the subtitle text). A column left out of . . . uses its column name as the heading and a non-breaking space () as the subtitle.

Details

Every column in the table is relabeled, not only the ones named in The loop walks the full set of column names, looks each one up in ..., and builds an HTML label holding the heading at heading_size over the subtitle at subtitle_size, joined by a line break. A column with no entry in ... falls back to its own name as the heading and a non-breaking space as the subtitle, so the second line still takes vertical space and the headers stay aligned. Labels are applied with `gt::cols_label()`, so call this after any other label changes or they will be overwritten.

Value

Returns a modified gt table with stacked headers and subtitles.

Examples

```
library(gt)

sdv_example_standings |>
  subset(division == "AFC West", c(team_name, wins, points_for)) |>
  gt() |>
  gt_column_subheaders(
    wins = list(heading = "Wins", subtitle = "W"),
    points_for = list(heading = "Points", subtitle = "PF"),
    heading_color = "blue", subtitle_color = "gray"
  )
```

gt_cutline

Draw a labeled cut line between rows of a gt table

Description

Draws a rule across the table after a given row, with an optional label on it. This is the cut line convention used in ranked and sorted tables to mark a cutoff or qualifying threshold.

Usage

```
gt_cutline(
  gt_object,
  after,
  label = NULL,
  color = "#A6081A",
  weight = 2,
  style = "dashed",
  label_color = NULL,
  label_size = 9,
  label_position = c("below", "above"),
  gap = 0
)
```

Arguments

gt_object	A gt table object to modify.
after	Numeric. The row number or numbers to draw a line after. <code>after = 4</code> draws between rows 4 and 5. Vectorized, so <code>c(4, 25)</code> draws two lines. <code>after = 0</code> draws a line above the first row, which is how you label the top section of a table sorted into a high group and a low group.
label	Optional. The label or labels for the lines, recycled against <code>after</code> . Use NA for an unlabeled line among labeled ones. Rendered in uppercase. Defaults to NULL.
color	Character. A hex color for the rule. Defaults to "#A6081A".
weight	Numeric. The thickness of the rule in pixels. Defaults to 2.
style	Character. The line style. One of "dashed", "solid", or "dotted". Defaults to "dashed".
label_color	Optional. A hex color for the label. Defaults to NULL, which uses color.
label_size	Numeric. The label size in pixels. Defaults to 9.
label_position	Character. Either "below" to put the label under the line or "above" to put it over. Defaults to "below".
gap	Numeric. Extra space in pixels added around the line to set the two sections further apart. A single number is applied above and below the line; a length-2 vector <code>c(above, below)</code> sets the two sides separately, so <code>c(0, 12)</code> opens space only under the line. Applies to labeled and unlabeled lines alike. Defaults to 0.

Details

The line is positional, not data-driven. `gt::tab_row_group()` splits a table on a value in the data and turns the result into labeled sections with heading rows. A cut line leaves the table flowing and marks a fixed row number.

The label is drawn as an inline SVG background image on the row, because gt's CSS inliner strips the `::before` and `::after` pseudo-elements this would normally be built with. That costs two things. An SVG background cannot use the page's webfonts, so the label renders in a system sans-serif rather than the theme's font. And a background image cannot straddle a border, so the labeled row is given extra padding for the label to sit in.

The labeled row's cell backgrounds are cleared, since row striping and themed body fills would otherwise cover the label. The row's own color is then reapplied to the row itself, so a striped theme such as `gt_theme_almanac()` keeps an unbroken stripe pattern. Colors set on individual cells in that row, such as `gt_color_ranks()` or `gt_color_pills()` fills, are cleared too; on a table with colored cells, draw the line unlabeled and describe it in a caption.

Labeled lines assume the table has no row groups, since group heading rows shift the row positions the label CSS targets. The rule itself is unaffected.

Apply the theme before this function. The stripe color is read from the table's options, so a line labeled first loses the stripe under its own row.

Value

Returns a modified gt table with the cut line or lines added.

See Also

[gt_spotlight\(\)](#) for drawing attention to a row.

Examples

```
library(gt)

east <- subset(
  sdv_example_standings, league == "nba" & conference == "Eastern",
  c(conference_rank, team_name, wins, losses, win_pct)
)
east <- east[order(east$conference_rank), ]

# mark where the six teams that skip the play-in end
gt(east) |>
  gt_theme_broadsheet() |>
  gt_cutline(after = 6, label = "Playoffs")

# the play-in line as well, unlabeled
gt(east) |>
  gt_cutline(after = c(6, 10), label = c("Playoffs", NA), color = "#0054AD")
```

 gt_delta

Add a computed change column to a gt table

Description

Takes two numeric columns and inserts a new column holding the change between them, signed and colored by direction. The change is to minus from, so a later period as to and an earlier one as from gives a positive number when the value has grown. The arithmetic, the formatting, and the sign coloring happen in one call.

Usage

```
gt_delta(
  gt_object,
  from,
  to,
  column_label = "Change",
  percent = FALSE,
  decimals = 1,
  arrows = FALSE,
  color = TRUE,
  color_positive = "#1B7837",
  color_negative = "#B2182B",
  color_neutral = NULL,
  force_sign = TRUE,
```

```

    after = NULL
  )

```

Arguments

gt_object	A gt table object to modify.
from	The starting column, a single numeric column.
to	The ending column, a single numeric column. The change is to - from.
column_label	Character. The label for the new column. Defaults to "Change".
percent	Logical. Should the change be shown as a percent of from rather than an absolute difference? Defaults to FALSE.
decimals	Integer. The number of decimal places. Defaults to 1.
arrows	Logical. Should an up or down triangle lead the value in place of a sign? Defaults to FALSE.
color	Logical. Should the values be colored by direction? Defaults to TRUE.
color_positive	Character. The color for an increase. Defaults to a green, "#1B7837".
color_negative	Character. The color for a decrease. Defaults to a red, "#B2182B".
color_neutral	Optional. The color for no change. Defaults to NULL, which leaves zero the table's normal text color.
force_sign	Logical. Should a plus be shown on an increase? Ignored when arrows is TRUE, since the arrow carries the direction. Defaults to TRUE.
after	Optional. The column the new one is placed after, a position or a name. Defaults to NULL, which places it after to.

Details

gt::fmt_number(force_sign = TRUE) formats a difference you have already worked out, and gtExtras::gt_fa_rank_change() handles movement in a rank column. This computes the delta from two columns, formats it, and colors it by sign together.

A row is left blank where either from or to is missing, and where a percent change divides by a from of zero. When arrows is TRUE the value is shown as a magnitude behind the triangle, so a decrease reads as a down triangle in front of a positive number.

Value

Returns a modified gt table with the change column added.

See Also

[gt_scale_note\(\)](#) for disclosing a divided scale.

Examples

```
library(gt)

nfc_north <- subset(
  sdv_example_standings, division == "NFC North",
  c(team_name, last_season_wins, wins)
)

# the change in wins from 2024, colored by sign
gt(nfc_north) |> gt_delta(last_season_wins, wins)

# as a percent of last season's wins, with arrows
gt(nfc_north) |> gt_delta(last_season_wins, wins, percent = TRUE, arrows = TRUE)
```

gt_fmt_rank	<i>Format numbers as ordinals in a gt table</i>
-------------	---

Description

Turns plain integers into ordinals: 1 becomes 1st, 2 becomes 2nd, 23 becomes 23rd. The suffix can be rendered as superscript, which is the usual treatment in a ranked column.

Usage

```
gt_fmt_rank(gt_object, columns, superscript = TRUE, suffix_size = "0.7em")
```

Arguments

gt_object	A gt table object to modify.
columns	The column or columns to format.
superscript	Logical. Should the suffix be rendered as superscript? Defaults to TRUE.
suffix_size	Character. The size of the suffix, as a CSS size. Defaults to "0.7em".

Details

Teens are handled correctly, so 11, 12 and 13 take "th" and not "st", "nd" and "rd". Values that are not numeric are left alone.

Value

Returns a modified gt table with ordinal formatting applied.

Examples

```
library(gt)

nfc <- subset(
  sdv_example_standings, conference == "NFC" & conference_rank <= 5,
  c(team_name, conference_rank)
)
nfc <- nfc[order(nfc$conference_rank), ]

gt(nfc) |> gt_fmt_rank(conference_rank)

# flat, without the superscript
gt(nfc) |> gt_fmt_rank(conference_rank, superscript = FALSE)
```

gt_fmt_tally	<i>Combine count columns into a single cell in a gt table</i>
--------------	---

Description

Merges two or more count columns into one "32-5" style cell, optionally with one of them shown as a share of the row total. Any set of tallies works, from wins and losses to tests passed and failed.

Usage

```
gt_fmt_tally(
  gt_object,
  columns,
  separator = "-",
  label = NULL,
  share = FALSE,
  share_of = 1,
  share_location = c("inline", "column"),
  share_decimals = 1,
  share_label = "%",
  share_prefix = " (",
  share_suffix = ")",
  ...
)
```

Arguments

gt_object	A gt table object to modify.
columns	The count columns to combine, in the order they should read. Two or more.
separator	Character. The string placed between the counts. Defaults to "-".
label	Optional. A new label for the combined column. Defaults to NULL, which leaves the existing label alone.

share	Logical. Should one of the counts be shown as a share of the row total? Defaults to FALSE.
share_of	The column the share is computed for, either a position or a name. Defaults to 1, the first column.
share_location	Character. Where the share goes. Either "inline" to append it to the combined cell, or "column" to reuse the last of columns for it. Defaults to "inline".
share_decimals	Integer. The number of decimal places for the share. Defaults to 1.
share_label	Character. The label for the share column when share_location is "column". Defaults to "%".
share_prefix	Character. The string placed before an inline share. Defaults to " (".
share_suffix	Character. The string placed after an inline share. Defaults to ")".
...	Additional arguments passed to <code>gt::vec_fmt_percent</code> .

Details

The counts are written into the first column and the rest are hidden, or the last one is reused to carry the share, so the table ends up narrower than it started.

`gt::cols_merge()` will join columns with a pattern and `gt::cols_merge_n_pct()` will pair a count with a percentage column you have already built. The difference here is that the share is computed for you from the counts themselves.

A row is left alone if any of its counts are missing, so a partial tally is never shown. The share is also left blank where the counts sum to zero.

Value

Returns a modified gt table with the counts combined.

See Also

[gt_fmt_rank\(\)](#) for ordinal formatting.

Examples

```
library(gt)

atlantic <- subset(
  sdv_example_standings, division == "Atlantic",
  c(team_name, wins, losses)
)

# renders as "56-26"
gt(atlantic) |> gt_fmt_tally(c(wins, losses), label = "Record")

# renders as "56-26 (68.3%)"
gt(atlantic) |> gt_fmt_tally(c(wins, losses), share = TRUE)

# the losing share instead, in its own column
gt(atlantic) |>
```

```

gt_fmt_tally(c(wins, losses),
  share = TRUE, share_of = "losses",
  share_location = "column", share_label = "Loss rate"
)

# three counts, as in an NFL standings table
nfc_east <- subset(
  sdv_example_standings, division == "NFC East",
  c(team_name, wins, losses, ties)
)

gt(nfc_east) |> gt_fmt_tally(c(wins, losses, ties), label = "W-L-T")

```

gt_grid

Arrange several gt tables in a grid

Description

Lays a list of tables out in rows and columns as small multiples, one table per region or per quarter. `gtExtras::gt_two_column_layout()` handles exactly two tables and `gt_stack_tables()` stacks any number of them vertically. This covers the rest.

Usage

```

gt_grid(
  tables = NULL,
  ncol = 2,
  labels = NULL,
  label_style = list(),
  title = NULL,
  subtitle = NULL,
  caption = NULL,
  source_note = NULL,
  caption_rule = FALSE,
  title_style = list(),
  subtitle_style = list(),
  caption_style = list(),
  source_note_style = list(),
  gap = 24,
  align = c("top", "center", "bottom"),
  file = NULL,
  bg = "white",
  whitespace = 50,
  zoom = 2
)

```

Arguments

<code>tables</code>	A list of gt table objects.
<code>ncol</code>	Integer. The number of tables across. The number of rows follows from the length of tables. Defaults to 2.
<code>labels</code>	Character. An optional caption above each table, recycled against tables. Rendered in a neutral style rather than each table's own, and placed outside the table so a wide label does not stretch its panel. Defaults to NULL.
<code>label_style</code>	A list of style options for the per-table captions. See <code>title_style</code> for the keys. Defaults to an empty list.
<code>title</code>	Character. An optional heading above the whole grid. Set this instead of giving each table its own <code>gt::tab_header()</code> . Defaults to NULL.
<code>subtitle</code>	Character. An optional line below <code>title</code> . Defaults to NULL.
<code>caption</code>	Character. An optional note below the grid. Defaults to NULL.
<code>source_note</code>	Character. An optional second line below caption, right-aligned by default. Set both, with <code>caption_rule = TRUE</code> , for the split caption gt_538_caption() gives a single table. Defaults to NULL.
<code>caption_rule</code>	Logical. Should a hairline rule sit between caption and source_note? Defaults to FALSE.
<code>title_style, subtitle_style, caption_style, source_note_style</code>	Lists of style options. Recognized keys are font (a Google font name), size, color, weight, italic, spacing (letter spacing), transform (such as "uppercase"), and align, plus line_height, margin_top, margin_bottom, padding_top, and padding_bottom for nudging the blocks closer together or further apart. Any key you leave out keeps its default, so <code>title_style = list(size = "34px")</code> changes only the size. Same convention as gt_title_header() . Lengths take a number, read as pixels, or a CSS string such as "2rem"; a negative margin pulls an element up tight against the one above it.
<code>gap</code>	Numeric. The space between tables in pixels. Defaults to 24.
<code>align</code>	Character. How tables of differing height line up within a row. Either "top", "center", or "bottom". Defaults to "top".
<code>file</code>	Optional. A path to write a PNG to. If NULL, the grid is returned for the viewer instead. Defaults to NULL.
<code>bg</code>	Character. The background color, used when saving. Defaults to "white".
<code>whitespace</code>	Numeric. Padding left around the grid when saving, in pixels. Defaults to 50.
<code>zoom</code>	Numeric. The rendering zoom factor used when saving. Defaults to 2.

Details

Unlike most of this package, this does not return a gt table. Separate tables have their own columns, widths, and headers, so there is no single table to hand back; the grid is assembled as HTML. That makes it a last step, after each table is themed and formatted, and it is why saving happens here through `file` rather than through [gt_save_crop\(\)](#).

Each table also keeps its own heading. Use `title` when the tables are blocks of a single thing, as with a long ranking split in two, and per-table `gt::tab_header()`s when they are genuinely separate exhibits.

Saving needs the `webshot2` package.

Value

An `htmltools::browsable()` tag list of the grid, which prints in the viewer; with `file`, invisibly file after writing the PNG to it.

See Also

`gt_stack_tables()` for a vertical stack, and `gt_snake()` for folding a single long table into blocks.

Examples

```
library(gt)

afc <- subset(sdv_example_standings, conference == "AFC")
by_division <- lapply(split(afc, afc$division), function(d) {
  gt(d[c("team_name", "wins", "losses", "points_for")]) |>
  gt_theme_broadsheet() |>
  tab_header(title = d$division[[1]])
})

gt_grid(by_division, ncol = 2)

# one heading over the whole sheet
gt_grid(by_division,
  ncol = 2, title = "The AFC by division",
  subtitle = "2025 NFL regular season", caption = "Data: nflreadr"
)

# a 538-style split caption under the whole sheet
gt_grid(by_division,
  ncol = 2,
  title = "The AFC by division",
  caption = gt::md("Teams are grouped by **division**."),
  source_note = "Data: nflreadr | Andrew Weatherman",
  caption_rule = TRUE
)

# tighten the subtitle up under the title, and open the gap to the grid
gt_grid(by_division,
  ncol = 2, title = "The AFC", subtitle = "2025 regular season",
  title_style = list(margin_bottom = -2),
  subtitle_style = list(margin_bottom = 24)
)

# styled without writing any css
gt_grid(by_division,
```

```

    ncol = 2,
    title = "The AFC by division",
    subtitle = "2025 NFL regular season",
    title_style = list(font = "Oswald", size = 34, transform = "uppercase"),
    subtitle_style = list(italic = TRUE, color = "#8A8A8A")
  )

# straight to an image; saving needs a headless Chrome (webshot2)

gt_grid(by_division, ncol = 2, file = tempfile(fileext = ".png"), bg = "#FBFAF7")

```

gt_group_stripes

*Band alternate row groups of a gt table***Description**

Shades every other row group, so each group reads as a block. `gt::opt_row_stripping()` bands every other row, which fights the grouping on a table that is already divided into sections.

Usage

```
gt_group_stripes(gt_object, color = "#F5F5F5", start = 2, include_stub = TRUE)
```

Arguments

<code>gt_object</code>	A gt table object to modify. It must have row groups.
<code>color</code>	Character. A hex color for the banded groups. Defaults to "#F5F5F5".
<code>start</code>	Integer. Which group to begin banding on, in the order the groups are rendered. 2 leaves the first group unshaded, 1 shades it. Defaults to 2.
<code>include_stub</code>	Logical. Should the stub column be banded along with the body? Defaults to TRUE.

Details

Groups are banded in the order they render, so this follows `gt::row_group_order()` rather than the order the groups happen to appear in the data.

The fill is applied to body cells rather than through CSS, because a group heading occupies a row of its own and shifts every `nth-child` count below it. Group heading rows are left alone; `gt::tab_options(row_group.background.color)` sets those, and it sets all of them at once.

Value

Returns a modified gt table with alternate groups banded.

Examples

```
library(gt)

afc <- subset(
  sdv_example_standings, conference == "AFC",
  c(team_name, division, wins, losses, points_for)
)

gt(afc, groupname_col = "division") |>
  gt_theme_broadsheet() |>
  gt_group_stripes()

# shade from the first group instead, in a warmer tone
gt(afc, groupname_col = "division") |>
  gt_group_stripes(color = "#FBF3E4", start = 1)
```

gt_highlight_cells	<i>Fill individual cells in a gt table by a condition</i>
--------------------	---

Description

Fills each cell in a block of columns that meets a condition and leaves the rest alone. It is the per-cell counterpart to `gtExtras::gt_highlight_cols()` and `gtExtras::gt_highlight_rows()`, which fill a whole column or a whole row. The condition is tested against every cell on its own, so a scattered set of cells can be filled in one call.

Usage

```
gt_highlight_cells(
  gt_object,
  columns,
  condition,
  fill = "#FFF3B0",
  text_color = NULL,
  bold = FALSE,
  ...
)
```

Arguments

gt_object	A gt table object to modify.
columns	The block of columns to test, the body of the grid.
condition	How a cell is chosen. Either a function or a right-hand-side formula applied to each column and returning a logical vector (<code>~ .x > 0.7</code> , <code>~ grepl("home", .x)</code> , <code>~ is.na(.x)</code>), or a logical matrix or data frame the same shape as columns for a mask computed ahead of time.

fill	Character. The cell fill color. Defaults to "#FFF3B0", a soft yellow.
text_color	Optional. A hex color for the text in filled cells. Defaults to NULL, which leaves the text color alone.
bold	Logical. Should filled cells be bolded? Defaults to FALSE.
...	Additional arguments passed to <code>gt::cell_text()</code> for the filled cells, such as <code>style = "italic"</code> .

Details

A single `gt::tab_style()` with `gt::cells_body()` cannot do this, since it reads rows and columns as a rectangle, every selected row crossed with every selected column. Filling a diagonal, a checker pattern, or any scatter otherwise takes a loop over the columns, one pass each. This wraps that loop. Each selected column is filled independently, so the pattern can be a diagonal, a checker, or any scatter of cells.

The condition reads the underlying data rather than the rendered cell, so a test like `~ grepl("home", .x)` sees whatever value the column holds, HTML and all. A formula or function must return one logical per row, and NA counts as not matched. For more than one color, layer the calls, one per color.

Value

Returns a modified gt table with the matching cells filled.

See Also

`gtExtras::gt_highlight_cols()` and `gtExtras::gt_highlight_rows()` for whole-column and whole-row fills.

Examples

```
library(gt)

# a correlation matrix: flag strong pairs, but not the diagonal of ones
nba <- subset(sdv_example_standings, league == "nba")
nba$point_diff <- nba$points_for - nba$points_against
vars <- c("wins", "point_diff", "points_for", "points_against", "conference_rank")
m <- round(cor(nba[vars]), 2)
cordf <- data.frame(var = rownames(m), m, row.names = NULL, check.names = FALSE)
cordf |>
  gt(rowname_col = "var") |>
  gt_highlight_cells(-var, ~ .x > 0.7 & .x < 1, fill = "#FFD1A9") |>
  gt_highlight_cells(-var, ~ .x < -0.7, fill = "#A9D0FF")

# a grid of team keys, one column per division in finishing order: playoff
# teams one color, the conference's top seed another
afc <- subset(sdv_example_standings, conference == "AFC")
grid <- data.frame(split(afc$team, afc$division), check.names = FALSE)
playoffs <- afc$team[afc$conference_rank <= 7]
top_seed <- afc$team[afc$conference_rank == 1]
```

```
gt(grid) |>
  gt_highlight_cells(everything(), ~ .x %in% playoffs, fill = "#CCE7F5") |>
  gt_highlight_cells(everything(), ~ .x == top_seed, fill = "#FCCF10")
```

gt_highlight_na

Style and relabel missing values in a gt table

Description

Highlights missing cells and can relabel them in the same call. `gt::sub_missing()` substitutes text only, and only for real NA values.

Usage

```
gt_highlight_na(
  gt_object,
  columns = gt::everything(),
  fill = "#F0F0F0",
  text_color = NULL,
  bold = FALSE,
  italic = FALSE,
  missing_text = NULL,
  na_strings = "NA",
  ignore_case = FALSE,
  ...
)
```

Arguments

gt_object	A gt table object to modify.
columns	The column or columns to check. Defaults to all of them.
fill	Character. A hex color for the cell fill behind missing values. Defaults to "#F0F0F0".
text_color	Optional. A hex color for the text of missing values. Defaults to NULL.
bold	Logical. Should missing values be bolded? Defaults to FALSE.
italic	Logical. Should missing values be italicized? Defaults to FALSE.
missing_text	Optional. Replacement text for missing values, such as "--" or "Not reported". Defaults to NULL, which leaves the text alone.
na_strings	A character vector of strings to treat as missing alongside real NA. Defaults to "NA".
ignore_case	Logical. Should na_strings be matched case-insensitively? Defaults to FALSE.
...	Additional arguments passed to <code>gt::cell_text</code> .

Details

Alongside real NA, this also catches values that are literally the string "NA", a common artifact of reading a CSV and a frequent reason `sub_missing()` appears to do nothing. Widen `na_strings` to catch other placeholders such as "-" or "N/A".

Value

Returns a modified gt table with missing values styled.

See Also

[gt_outliers\(\)](#) for flagging values that are present but suspect.

Examples

```
library(gt)

# playoff_wins is NA for the teams that missed the playoffs
afc <- subset(
  sdv_example_standings, conference == "AFC",
  c(team_name, wins, losses, conference_rank, playoff_wins)
)

gt(afc) |> gt_highlight_na(playoff_wins)

# relabel as well as highlight
gt(afc) |>
  gt_highlight_na(playoff_wins,
    missing_text = "missed playoffs",
    italic = TRUE, fill = "#FF8E1"
  )

# also catch placeholder strings left behind by a CSV import
gt(afc) |>
  gt_highlight_na(everything(), na_strings = c("NA", "N/A", "-"))
```

<code>gt_indicator_boxes</code>	<i>Add colored indicator boxes to a gt table</i>
---------------------------------	--

Description

Replaces column values with colored boxes, filled when a value meets a rule and left neutral otherwise. By default the box is colored when a value equals the second element of `indicator_vals` (1) and left color_no when it equals the first (0); supply `indicator_rule` for any other test. Name the columns to convert with `columns`, or the ones to leave alone with `key_columns`.

Usage

```
gt_indicator_boxes(
  gt_object,
  columns = NULL,
  key_columns = NULL,
  indicator_vals = c(0, 1),
  indicator_rule = function(x) x == indicator_vals[2],
  color_yes = "#FCCF10",
  color_no = "#EEEEEE",
  show_na_as_na = FALSE,
  show_text = FALSE,
  show_only = NULL,
  per_column_formats = NULL,
  color_na = NULL,
  border_color = NULL,
  border_width = 0.25,
  box_width = 20,
  box_height = 20,
  text_size = 12,
  text_weight = "bold"
)
```

Arguments

<code>gt_object</code>	A gt table object to modify.
<code>columns</code>	The columns to convert to boxes, using <code>tidyselect</code> . Defaults to <code>NULL</code> , which converts every column not named in <code>key_columns</code> .
<code>key_columns</code>	The columns to leave untouched, using <code>tidyselect</code> . Every other column is converted. Kept for the inverted way of saying the same thing; give this or <code>columns</code> , not both. Defaults to <code>NULL</code> .
<code>indicator_vals</code>	Numeric. A length-2 vector giving the "no" and "yes" values. Defaults to <code>c(0, 1)</code> .
<code>indicator_rule</code>	A function deciding when a box is colored. It receives the column values, and optionally the column name as a second argument, and returns a logical vector. Defaults to testing equality with <code>indicator_vals[2]</code> .
<code>color_yes</code>	Character. The fill for boxes meeting the rule. Defaults to <code>"#FCCF10"</code> .
<code>color_no</code>	Character. The fill for boxes not meeting the rule. Defaults to <code>"#EEEEEE"</code> .
<code>show_na_as_na</code>	Logical. Should NA be shown as NA rather than treated as "no"? Defaults to <code>FALSE</code> .
<code>show_text</code>	Logical. Should the formatted value be printed inside the box? Defaults to <code>FALSE</code> .
<code>show_only</code>	Character. Restrict printed text to one class of box, one of "yes", "no", or "NA". Defaults to <code>NULL</code> , which prints text for all.
<code>per_column_formats</code>	A named list keyed by column name, each element a list of formatting options (<code>digits</code> , <code>format_type</code> , <code>suffix</code>) for that column. Defaults to <code>NULL</code> .

color_na	Character. The fill for NA boxes. Defaults to NULL, which uses color_no.
border_color	Character. The border color around the boxes. Defaults to NULL, no border.
border_width	Numeric. The border width in pixels. Defaults to 0.25.
box_width	Numeric. The box width in pixels, used when show_text is FALSE. Defaults to 20.
box_height	Numeric. The box height in pixels. Defaults to 20.
text_size	Numeric. The font size of the box text in pixels, used when show_text is TRUE. Defaults to 12.
text_weight	Character. The font weight of the box text. Defaults to "bold".

Details

Every column outside `key_columns` is replaced with an HTML span through `gt::text_transform()`, and the transformed columns are then center-aligned. The rule is applied to the numeric coercion of each column, so text values become NA; NA cells take `color_na` (falling back to `color_no`) unless `show_na_as_na` keeps them labeled NA. Box text is set to black or white, whichever measures higher contrast against the fill.

When `show_text` is TRUE the box widens to fit the widest formatted value in the column; otherwise it is fixed at `box_width`. `indicator_rule` may accept a second argument, the column name, which allows a different test per column.

Value

Returns a modified gt table with the converted columns shown as colored boxes.

Examples

```
library(gt)

roster <- data.frame(
  player = c("A", "B", "C"),
  starter = c(1, 0, 1),
  injured = c(0, 0, 1),
  captain = c(1, 0, 0)
)

gt(roster) |> gt_indicator_boxes(key_columns = "player")

# print the underlying values and draw a border
gt(roster) |>
  gt_indicator_boxes(
    key_columns = "player", show_text = TRUE,
    border_color = "#333333"
  )
```

gt_legend_continuous *Add a color legend to a gt table*

Description

Draws a legend for a column colored with `gt_color_ranks()` or `gt::data_color()`. Pass it the same columns and palette you colored with and the legend will match the table, including the domain it takes from the data. For a hand-built key of labeled swatches, use `gt_legend_discrete()`.

Usage

```
gt_legend_continuous(
  gt_object,
  columns = NULL,
  palette = c("#3D8B6E", "#9DC5A7", "#EDE0CC", "#DB9070", "#BE4D3A"),
  domain = NULL,
  reverse = FALSE,
  pal_type = "discrete",
  type = c("continuous", "steps", "blocks"),
  n_bins = 5,
  labels = NULL,
  digits = 0,
  title = NULL,
  title_position = c("top", "bottom", "left", "right"),
  title_style = list(),
  labels_style = list(),
  labels_position = c("bottom", "top", "none"),
  location = c("bottom", "top"),
  align = c("center", "left", "right"),
  width = 200,
  height = 10,
  border_color = NULL,
  border_width = 1,
  radius = 2,
  gap = 3,
  title_gap = 4,
  block_gap = 2
)
```

Arguments

gt_object	A gt table object to modify.
columns	The column or columns the legend describes, used to derive domain. Defaults to NULL, which takes the columns from an earlier coloring call, or requires domain to be given instead. See Picking up the scale.
palette	A color palette to use. If you want a palette from paletteer, specify it as <code>package::palette</code> . Defaults to the same five-color green-to-red ramp as <code>gt_color_ranks()</code> .

domain	A length-2 numeric vector giving the value range the palette spans. If NULL, the range is taken from columns the same way <code>gt_color_ranks()</code> takes it, so the two agree. Defaults to NULL.
reverse	Logical. Should the palette be reversed? Set this to match the reverse used when coloring. Defaults to FALSE.
pal_type	Character. Either "discrete" or "continuous", used when applying paletteer palettes. Defaults to "discrete".
type	Character. The appearance of the bar. One of "continuous" for a smooth ramp, "steps" for <code>n_bins</code> flat steps that touch, or "blocks" for <code>n_bins</code> separated swatches. Defaults to "continuous".
n_bins	Integer. The number of bins when type is "steps" or "blocks". Defaults to 5.
labels	The tick labels. If NULL, the two ends of domain are used. The string "edges" uses the <code>n_bins</code> + 1 bin boundaries. Otherwise pass any character vector, which is spread evenly. Defaults to NULL.
digits	Integer. The number of decimal places used when deriving labels from domain. Defaults to 0.
title	Optional. A caption for the legend. Defaults to NULL.
title_position	Character. Which side of the bar the title sits on. One of "top", "bottom", "left", or "right". Defaults to "top".
title_style	A named list styling the title. See Styling. Defaults to an empty list.
labels_style	A named list styling the bound labels. See Styling. Defaults to an empty list.
labels_position	Character. One of "bottom", "top", or "none" to omit the labels. Defaults to "bottom".
location	Character. Where to place the legend. "bottom" adds it as a source note; "top" puts it in the header, keeping any existing title and subtitle. Defaults to "bottom".
align	Character. The horizontal alignment of the whole legend. One of "center", "left", or "right". Defaults to "center".
width	Numeric. The width of the bar in pixels. Defaults to 200.
height	Numeric. The height of the bar in pixels. Defaults to 10.
border_color	Optional. A hex color for a border around the bar, or around each block when type is "blocks". Defaults to NULL.
border_width	Numeric. The border width in pixels. Defaults to 1.
radius	Numeric. The corner radius of the bar in pixels. Defaults to 2.
gap	Numeric. The gap between the bar and its labels, in pixels. Defaults to 3.
title_gap	Numeric. The gap between the title and the bar, in pixels. Defaults to 4.
block_gap	Numeric. The gap between blocks when type is "blocks", in pixels. Defaults to 2.

Details

The bar can be a smooth gradient, a stepped ramp, or separated blocks. The title can sit on any side of it, and the title and bound labels are styled independently through `title_style` and `labels_style`.

The bar is drawn as a series of solid color segments instead of a CSS gradient. `gt`'s inline-CSS path, used by the RStudio Viewer, Quarto and R Markdown, strips gradient backgrounds, and the legend would disappear. Segment colors come from `scales::col_numeric`, the same function `gt::data_color()` uses, so the ramp matches the cells exactly.

`gt_title_header()` calls `gt::tab_header()`, which replaces the whole header. With `location = "top"`, call `gt_title_header()` first or it will overwrite the legend.

Value

Returns a modified `gt` table with the legend added.

Styling

`title_style` and `labels_style` are named lists following the same convention as `gt_grid()` and `gt_title_header()`. Recognized keys are `font` (a Google font name), `size`, `color`, `weight`, `italic`, `spacing` (letter spacing), `transform` (such as "uppercase"), and `align`, plus `line_height`, `margin_top`, `margin_bottom`, `padding_top`, and `padding_bottom`. Any length takes a number, read as pixels, or a CSS string.

Picking up the scale

`gt_color_ranks()`, `gt_color_pills()` and `gt_percentile_bar()` record the columns, palette, domain, reverse and `pal_type` they used on the table. This function reads any of those the caller did not supply, so a legend can be added with no arguments and cannot disagree with the cells it explains:

```
gt(df) |>
  gt_color_ranks(net, palette = "viridis::mako", domain = c(-10, 12)) |>
  gt_legend_continuous()
```

Anything passed explicitly wins over the recorded value. With several coloring calls, the most recent one is the one recorded. The record is an attribute on the table and survives the rest of a pipeline, apart from `gt_snake()`, which rebuilds the table from its data; add the legend before snaking, or pass the arguments directly.

See Also

`gt_color_ranks()`, and `gt_legend_discrete()` for a discrete key.

Examples

```
library(gt)

# the legend matches whatever gt_color_ranks() drew
```

```

nfc_east <- subset(
  sdv_example_standings, division == "NFC East",
  c(team_name, wins, losses, points_for)
)
gt(nfc_east) |>
  gt_color_ranks(columns = points_for) |>
  gt_legend_continuous(columns = points_for, title = "Points scored")

# discrete blocks, labeled at every bin edge, title to the left
west <- subset(
  sdv_example_standings, league == "nba" & conference == "Western",
  c(team_name, wins, losses)
)
gt(west) |>
  gt_color_ranks(columns = wins) |>
  gt_legend_continuous(
    columns = wins, type = "blocks", n_bins = 5, labels = "edges",
    title = "Wins", title_position = "left",
    title_style = list(
      weight = 600, transform = "uppercase",
      spacing = "0.08em", size = "10px"
    ),
    labels_style = list(size = "9px", color = "#999999")
  )

```

gt_legend_discrete	<i>Add a discrete color key to a gt table</i>
--------------------	---

Description

Draws a legend of labeled color swatches, the categorical counterpart to [gt_legend_continuous\(\)](#), which draws a continuous ramp. Give it the colors and labels you used elsewhere in the table, such as the fills from [gt_highlight_cells\(\)](#) or [gt_indicator_boxes\(\)](#), and it renders a color key. After [gt_tiers\(\)](#), which records its mapping on the table, `key_info` can be left out entirely.

Usage

```

gt_legend_discrete(
  gt_object,
  key_info = NULL,
  heading = NULL,
  subtitle = NULL,
  label_placement = c("outside", "inside"),
  location = c("top", "bottom"),
  shape = c("square", "rounded", "circle"),
  swatch_size = 14,
  border = TRUE,
  border_color = NULL,

```

```

border_width = 1,
gap = 14,
direction = c("horizontal", "vertical"),
align = c("center", "left", "right"),
heading_style = list(),
subtitle_style = list(),
label_style = list()
)

```

Arguments

gt_object	A gt table object to modify.
key_info	The key. Either a named character vector of label = color, or a data frame with a color column and a label column. Defaults to NULL, which takes the key <code>gt_tiers()</code> recorded on the table.
heading	Optional. A heading shown above the key. Defaults to NULL, no heading.
subtitle	Optional. A subtitle shown under the heading. Defaults to NULL, no subtitle.
label_placement	Character. Either "outside" to set each label beside its swatch, or "inside" to print it on the swatch. Defaults to "outside".
location	Character. Where the key goes. "top" places it in the header; "bottom" adds it as a source note. When location is "top" and no heading or subtitle is given, any existing title and subtitle are kept and the key is appended. Defaults to "top".
shape	Character. The swatch shape, one of "square", "rounded", or "circle". Defaults to "square".
swatch_size	Numeric. The swatch size in pixels. Defaults to 14.
border	Logical. Should a hairline be drawn around each swatch? This keeps a pale swatch visible against the page. Defaults to TRUE.
border_color	Optional. A hex color for the hairline, applied to every swatch. When NULL, each swatch takes its own edge, a soft darker shade of its fill, so pale swatches stay defined without a harsh uniform outline. Defaults to NULL.
border_width	Numeric. The hairline width in pixels. Defaults to 1.
gap	Numeric. The space between keys in pixels. Defaults to 14.
direction	Character. Whether the keys run "horizontal" in a row or "vertical" in a column. Defaults to "horizontal".
align	Character. The alignment of the whole key, one of "center", "left", or "right". Defaults to "center".
heading_style	A named list styling the heading. See Styling. Defaults to an empty list.
subtitle_style	A named list styling the subtitle. See Styling. Defaults to an empty list.
label_style	A named list styling the swatch labels. See Styling. For "inside" labels the color is chosen per swatch for contrast and any color here is ignored. Defaults to an empty list.

Details

The key can sit above the table or below it, run in a row or a column, and take square, rounded, or circular swatches. An optional heading and subtitle can ride along above the swatches, each styled through its own list.

The key is passed as either a named vector, `c("Home" = "#cce7f5", "Bye" = "#eeeeee")`, where the names are the labels, or a data frame with a color column and a label column.

When `label_placement` is "inside", each label takes black or white by whichever contrasts better with its swatch, so a color in `label_style` is ignored in that mode.

`gt_title_header()` and anything else calling `gt::tab_header()` replaces the whole header. With `location = "top"`, add those first or they will overwrite the key.

Value

Returns a modified gt table with the key added.

Styling

`heading_style`, `subtitle_style`, and `label_style` are named lists following the same convention as `gt_grid()` and `gt_title_header()`. Recognized keys are font (a Google font name), size, color, weight, italic, spacing (letter spacing), transform (such as "uppercase"), and align, plus `line_height`, `margin_top`, `margin_bottom`, `padding_top`, and `padding_bottom`. Any length takes a number, read as pixels, or a CSS string. Any key left out keeps its default. Colors left unset are derived from the table background, so the key reads on light and dark themes. Apply the theme first, since an unthemed table reports a white background and the heading and subtitle come out near-black on whatever ground the theme sets later.

See Also

`gt_legend_continuous()` for a continuous ramp.

Examples

```
library(gt)

afc_west <- subset(
  sdv_example_standings, division == "AFC West",
  c(team_name, wins, losses, playoff_wins)
)

# a named vector is the quickest way in
gt(afc_west) |>
  gt_legend_discrete(c("Playoffs" = "#CCE7F5", "Missed" = "#F5CCCC"))

# rounded swatches below the table, labels on the swatches
gt(afc_west) |>
  gt_legend_discrete(
    c("Yes" = "#FCCF10", "No" = "#EEEEEE"),
    heading = "Qualified",
    label_placement = "inside",
```

```

    location = "bottom",
    shape = "rounded"
  )

# style the heading and labels through lists, the same way gt_grid() does
gt(afc_west) |>
  gt_legend_discrete(
    c("Home" = "#CCE7F5", "Away" = "#FFFFFF", "Bye" = "#EEEEEE"),
    heading = "2025 Schedule",
    heading_style = list(font = "Oswald", size = 20, transform = "uppercase"),
    label_style = list(size = 13, color = "#444444")
  )

```

gt_marginalia

Style a column as margin notes

Description

Styles a text column as commentary instead of data. It applies a muted color, italics, a hairline separating the notes from the figures, a blank column label, and a width constraint so the prose wraps instead of stretching the table.

Usage

```

gt_marginalia(
  gt_object,
  columns,
  width = 220,
  label = "",
  italic = TRUE,
  color = NULL,
  size = "0.92em",
  rule = TRUE,
  rule_color = NULL,
  align = "left"
)

```

Arguments

gt_object	A gt table object to modify.
columns	The column or columns to treat as margin notes.
width	Numeric. The column width in pixels, which is what makes the prose wrap. Pass NULL to leave the width alone. Defaults to 220.
label	Character. The column label. Defaults to "", since a note column rarely needs one; pass a string to keep a heading.
italic	Logical. Should the notes be italicized? Defaults to TRUE.

color	Optional. A hex color for the text. If NULL, a muted color is derived from the table background and checked for legibility. Defaults to NULL.
size	Character. The font size, as a CSS size. Defaults to "0.92em", slightly smaller than the body and scaling with whatever size the theme sets.
rule	Logical. Should a hairline be drawn on the left edge, separating the notes from the data? Defaults to TRUE.
rule_color	Optional. A hex color for that hairline. Derived from the table background when NULL. Defaults to NULL.
align	Character. The text alignment. Defaults to "left".

Details

A text column with no width set runs to a single long line, so width is what wraps the prose. On a three-row example, that is the difference between a table 625 pixels wide and one 482 pixels wide.

Colors are derived from the table background instead of being hard-coded, so the treatment also reads correctly on a dark theme. That means the theme has to be applied first. An unthemed table reports a white background, and the notes come out too dark to read once a dark theme lands on top. Pass color and rule_color to fix them regardless of order.

Value

Returns a modified gt table with the selected columns styled as margin notes.

See Also

[gt_cutline\(\)](#) for a labeled break between rows.

Examples

```
library(gt)

nfc_west <- subset(
  sdv_example_standings, division == "NFC West",
  c(team_name, wins, losses, conference_rank, playoff_wins)
)

# a note on each team's postseason, written from its own row
nfc_west$note <- with(nfc_west, ifelse(
  is.na(playoff_wins),
  "Missed the playoffs.",
  paste0(
    "The No. ", conference_rank, " seed; ", playoff_wins, " playoff win",
    ifelse(playoff_wins == 1, "", "s"), "."
  )
))

gt(nfc_west) |>
  gt_theme_broadsheet() |>
  gt_marginalia(note)
```

```
# keep a heading, widen it, and drop the italics
gt(nfc_west) |>
  gt_marginalia(note, label = "Postseason", width = 280, italic = FALSE)
```

gt_merge_stack_team_color

Merge and Stack Text in gt Tables with Team Colors

Description

Takes an existing gt table and merges column 1 and column 2, stacking column 1's text on top of column 2's. Top text is in all caps with black bold text, while the lower text is smaller and colored by the team name.

Usage

```
gt_merge_stack_team_color(
  gt_object,
  col1,
  col2,
  team_col,
  league,
  font_size_top = 14,
  font_size_bottom = 12,
  color = "black",
  background = NULL,
  season = NULL,
  id_system = "auto"
)
```

Arguments

gt_object	An existing gt table object of class gt_tbl.
col1	The column to stack on top. Will be all caps, black bold text.
col2	The column to merge and place below. Will be smaller and colored.
team_col	The column of team names for the color of the bottom text.
league	A league key from sdv_leagues() .
font_size_top	Font size for the top text.
font_size_bottom	Font size for the bottom text.
color	The color for the top text.

background	The cell background the lower text is checked against. NULL (the default) reads the table's background color, so a table theme such as <code>gt_theme_midnight()</code> applied before this function is taken into account; a theme applied afterwards isn't seen, so set background then. A table with no background color set counts as white.
season	NULL for today's mark, or a season (or one per team): the mark the team wore that season where the archive dates its marks.
id_system	"auto" (try every id system in priority order) or one id system: "team_id", "espn", "espn_abbr", "nhl", "nflverse", "mlbstats", "nba_api", "hockeytech", "ncaa", "pff", "cricinfo", "cfbd", "bref", "sportsipy", "fangraphs", "sdvplotr", "name", or "nhl_id" (NHL stats API ids, which overlap ESPN's, so "auto" never reads a number as one).

Details

The lower text takes the team's primary color when it clears a 4.5:1 contrast ratio (WCAG AA) against the cell background, else the secondary color, else the primary darkened (or lightened, on a dark background) until it does, so a light primary such as Missouri's gold stays readable on a white table and keeps its gold on a dark one.

Value

An object of class `gt_tbl`.

Examples

```
library(gt)
library(sdvplotR)

df <- data.frame(
  team = c("KC", "BUF", "SF"),
  mascot = c("Chiefs", "Bills", "49ers"),
  wins = c(11, 10, 12)
)

df |>
  gt() |>
  gt_merge_stack_team_color(team, mascot, team, league = "nfl")
```

gt_outliers

Flag outlying values in a gt table

Description

Marks cells that fall outside a threshold, either beyond an interquartile fence, beyond a number of standard deviations, or outside bounds you supply.

Usage

```
gt_outliers(
  gt_object,
  columns,
  method = c("iqr", "sd", "bounds"),
  threshold = NULL,
  bounds = NULL,
  side = c("both", "high", "low"),
  fill = NULL,
  color = NULL,
  bold = TRUE,
  symbol = NULL,
  note = NULL
)
```

Arguments

gt_object	A gt table object to modify.
columns	The column or columns to test. Non-numeric columns are skipped.
method	Character. How to decide what counts as an outlier. One of "iqr", "sd", or "bounds". Defaults to "iqr".
threshold	Numeric. The cutoff used by "iqr" and "sd". If NULL, the convention for the method is used: 1.5 for "iqr" and 3 for "sd". Defaults to NULL.
bounds	A length-2 numeric vector giving c(lower, upper), required when method is "bounds". Use NA for an open end, such as c(NA, 100). Defaults to NULL.
side	Character. Which tail to flag. One of "both", "high", or "low". Defaults to "both".
fill	Optional. A hex color for the cell fill behind flagged values. Defaults to NULL, which applies no fill.
color	Optional. A hex color for flagged text. If NULL, a warning red is used, swapped for a readable alternative when it would not have enough contrast against fill. Defaults to NULL.
bold	Logical. Should flagged values be bolded? Defaults to TRUE.
symbol	Optional. A marker appended to flagged values, such as "†". Defaults to NULL.
note	Optional. A source note describing the rule that was applied. Pass TRUE for wording generated from method and threshold, a string for your own, or NULL for no note. Defaults to NULL.

Details

gt::data_color() shades a continuous scale; this flags a value as unusual. Thresholds are computed separately for each column, so every column is judged against its own distribution.

The default rule is the interquartile fence, not standard deviations, because an SD fence is built from a spread that the outlier itself inflates. On c(10.2, 10.4, 10.1, 19.8, 10.3, 10.0) the mean is 11.8 and the standard deviation 3.92, so a three-SD fence reaches 23.6 and misses the 19.8. The

quartiles barely move, so the IQR fence stops at 10.75 and catches it. The masking is worst in small samples. Use `method = "sd"` if you want it anyway.

Value

Returns a modified gt table with outlying values marked.

See Also

[gt_highlight_na\(\)](#) for missing values, and [gt_spotlight\(\)](#) for drawing attention to whole rows.

Examples

```
library(gt)

nba <- subset(
  sdv_example_standings, league == "nba",
  c(team_name, points_for, points_against)
)

# the default fence (1.5 times the IQR) catches the league's lowest scorer
gt(nba) |> gt_outliers(c(points_for, points_against), note = TRUE)

# an explicit acceptance range, flagging only the high side
gt(nba) |>
  gt_outliers(c(points_for, points_against),
    method = "bounds", bounds = c(NA, 9800),
    side = "high", fill = "#FDECEA", symbol = "+"
  )

# works on wider data too
nfl <- subset(
  sdv_example_standings, league == "nfl",
  c(team_name, wins, points_for, points_against)
)
gt(nfl) |> gt_outliers(c(wins, points_for, points_against))
```

gt_percentile_bar	<i>Draw percentile bars in a gt table</i>
-------------------	---

Description

Draws a percentile as a filled track with a circular marker at the tip of the fill, the value printed inside it. The bar, the marker, and the number are all CSS, so the whole thing stays vector at any export scale.

Usage

```
gt_percentile_bar(
  gt_object,
  columns,
  rows = NULL,
  domain = c(0, 100),
  scale = "auto",
  palette = c("#3661AD", "#C9C9C9", "#D22D49"),
  reverse = FALSE,
  pal_type = "discrete",
  track_color = "#E9E9E9",
  track_height = 6,
  marker_size = 22,
  text_color = "#FFFFFF",
  font_size = NULL,
  ring_color = NULL,
  ring_width = 2,
  full_track = TRUE,
  na_label = "-",
  na_track_color = NULL,
  na_text_color = "#9A9A9A",
  decimals = 0,
  width = 220
)
```

Arguments

gt_object	A gt table object to modify.
columns	The columns holding percentiles.
rows	The rows to draw bars in. Either an expression evaluated against the table's data, such as <code>pct > 50</code> , or a numeric vector of row indices. Rows left out keep their raw value. If <code>NULL</code> , every row is drawn. Defaults to <code>NULL</code> .
domain	A length-2 numeric vector giving the percentile range. Defaults to <code>c(0, 100)</code> .
scale	How to handle a column stored as 0-1 proportions rather than 0-100 percentiles. "auto" maps a column whose values all fall in <code>[0, 1]</code> onto domain, so <code>0.72</code> is treated and printed as <code>72</code> against the default range. "none" leaves the values untouched. A number multiplies every value by it, for any other conversion. Defaults to "auto".
palette	Colors mapped across domain and used for both the fill and the marker. Defaults to a blue-grey-red ramp.
reverse	Logical. Should the palette be reversed? Defaults to <code>FALSE</code> .
pal_type	Character. Which palette registry to look a <code>package::palette</code> string up in, "discrete" or "continuous". The other registry is tried as a fallback, so this rarely needs setting. Defaults to "discrete".
track_color	Character. The color of the unfilled track. Defaults to "#E9E9E9".
track_height	Numeric. The track thickness in pixels. Defaults to <code>6</code> .

marker_size	Numeric. The marker diameter in pixels. Defaults to 22.
text_color	Character. The color of the number inside the marker. Defaults to "#FFFFFF".
font_size	Numeric. The size of the number in pixels. If NULL, it is set from marker_size. Defaults to NULL.
ring_color	Character. A ring drawn around the marker, which separates it from the fill. NULL for none. Defaults to NULL.
ring_width	Numeric. The ring thickness in pixels. Defaults to 2.
full_track	Logical. Should the track run the full width, or stop at the marker? Defaults to TRUE.
na_label	Character. What to show for a missing percentile, such as "Not qualified" or "NR". The track is drawn in two segments with the label centered between them, so the row keeps its rhythm and cannot be read as a percentile of zero. NULL draws an unbroken empty track with no text. Defaults to an em dash.
na_track_color	Character. The track color for missing rows. If NULL, track_color is used. Defaults to NULL.
na_text_color	Character. The color of na_label. Defaults to "#9A9A9A".
decimals	Integer. Decimal places on the number. Defaults to 0.
width	Numeric. The column width in pixels. Defaults to 220.

Details

gtExtras::gt_plt_percentile() covers percentiles as a ggplot dot plot rendered to a raster image. This is the filled-bar-and-marker treatment instead.

The track is inset by half the marker so that a value at either extreme keeps the marker inside the cell rather than hanging over the edge. The fill and the marker take the same color, mapped from the value.

Value

Returns a modified gt table with percentile bars.

Examples

```
library(gt)

metrics <- data.frame(
  metric = c("Barrel %", "Exit velocity", "Chase rate", "Whiff %"),
  pct = c(94, 41, 72, 18)
)

gt(metrics) |> gt_percentile_bar(pct)

# the same values stored 0-1 render identically under the default scale = "auto"
metrics$pct <- metrics$pct / 100
gt(metrics) |> gt_percentile_bar(pct)

# a row that did not qualify
```

```
metrics$pct[2] <- NA
gt(metrics) |> gt_percentile_bar(pct, na_label = "Not qualified")
```

gt_row_accent

Put a colored bar on the leading edge of each row

Description

Draws a short color bar down the left edge of every row, keyed to a column in the data such as a team, category, or brand color. It serves as a per-row key without taking up a column.

Usage

```
gt_row_accent(
  gt_object,
  column,
  palette = NULL,
  rows = NULL,
  width = 4,
  side = c("left", "right"),
  hide = TRUE,
  na_color = "transparent"
)
```

Arguments

gt_object	A gt table object to modify.
column	The column the color is keyed to. Either a column already holding colors, or a grouping column used together with palette.
palette	Optional. Colors to map column onto. A named vector maps values explicitly, as <code>c(ACC = "#003366", SEC = "#B8232F")</code> . An unnamed vector is assigned across the sorted unique values and recycled. If NULL, column is taken to hold colors already. Defaults to NULL.
rows	The rows to accent. Either an expression evaluated against the table's data, such as <code>conf == "ACC"</code> or <code>net <= 10</code> , or a numeric vector of row indices, or a single row number. If NULL, every row is accented. Defaults to NULL.
width	Numeric. The bar width in pixels. Defaults to 4.
side	Character. Which edge the bar sits on, "left" or "right". Defaults to "left".
hide	Logical. Should column be hidden once the bar is drawn? Usually what you want when it holds hex codes. Defaults to TRUE.
na_color	Character. The color for rows where the key is missing. Defaults to "transparent", which draws no bar.

Details

It draws the same bar `gt_spotlight()` uses for emphasis, applied per row here. `gtExtras::gt_highlight_rows()` fills the whole row instead.

The bar is drawn as a cell border on the leftmost rendered column, or on the stub when the table has one, so it lines up with the row rather than sitting inside a column of its own.

Rows sharing a color are styled together rather than one at a time, so a long table with a handful of categories stays cheap.

Value

Returns a modified gt table with a color bar on each row.

See Also

`gt_spotlight()` for emphasis rather than a key.

Examples

```
library(gt)

teams <- data.frame(
  team = c("Duke", "Kansas", "Auburn", "Houston"),
  conf = c("ACC", "B12", "SEC", "B12"),
  net = c(10, 20, 5, 1)
)

# keyed to a grouping column
gt(teams) |>
  gt_row_accent(conf, palette = c(
    ACC = "#003366", B12 = "#C8102E",
    SEC = "#B8232F"
  ))

# or straight from a column of colors
teams$color <- c("#003366", "#C8102E", "#B8232F", "#C8102E")
gt(teams) |> gt_row_accent(color)

# only some rows: by expression, by indices, or by one row number
gt(teams) |> gt_row_accent(color, rows = net <= 10)
gt(teams) |> gt_row_accent(color, rows = c(1, 3))
gt(teams) |> gt_row_accent(color, rows = 2)
```

Description

Splits data by a column, builds a table for each group with a function you supply, and writes one image per group. The images are padded to a common width so a posted series does not come out at ragged sizes.

Usage

```
gt_save_batch(
  data,
  group,
  fn,
  file,
  dir,
  match_width = TRUE,
  bg = "white",
  whitespace = 50,
  zoom = 2,
  quiet = FALSE
)
```

Arguments

<code>data</code>	A data frame to split.
<code>group</code>	The column to split on.
<code>fn</code>	A function building one table. It is called as <code>fn(df, group)</code> , where <code>df</code> is that group's rows and <code>group</code> is its value, and it must return a <code>gt_tbl</code> .
<code>file</code>	Character. A file name pattern containing <code>{group}</code> , which is replaced by the group value, as <code>"net-{group}.png"</code> .
<code>dir</code>	Character. The directory to write into. Created if it does not exist. There is no default, so a batch never lands in the working directory unasked; pass <code> "." </code> for that.
<code>match_width</code>	Logical. Should every image be padded to the width of the widest one, so a posted series shares one width? Defaults to <code>TRUE</code> .
<code>bg</code>	Character. The background color of the padding. Defaults to <code>"white"</code> .
<code>whitespace</code>	Numeric. Padding left around each table, in pixels. Defaults to <code>50</code> .
<code>zoom</code>	Numeric. The rendering zoom factor. Defaults to <code>2</code> .
<code>quiet</code>	Logical. Suppress the per-group progress message. Defaults to <code>FALSE</code> .

Details

A group whose table fails to build does not abandon the run. The error is collected, the remaining groups are written, and a warning at the end names every group that failed.

Group values are used in file names with anything awkward replaced by a dash, so a group called `"North / East"` writes to `north-east`.

Without `match_width`, each image is trimmed to its own content, so a set posted together can come out at different widths. With it, every image is padded to the width of the widest before its border is added.

Value

Invisibly, a character vector of the files written.

See Also

`gt_grid()` for the same split composed into one image instead.

Examples

```
library(gt)

build <- function(df, group) {
  gt(df[c("team_name", "wins", "losses", "points_for")]) |>
    gt_theme_broadsheet() |>
    tab_header(title = group)
}

nfl <- subset(sdv_example_standings, league == "nfl")
gt_save_batch(nfl, conference, build, "nfl-{group}.png", dir = tempdir())
```

gt_save_crop

Save a gt table to an image and trim its whitespace

Description

Renders a table to a PNG, JPG, or JPEG, trims the whitespace the screenshot leaves around it, and pads a uniform border back on. Saving runs through headless Chrome (webshot2) and the trimming and padding through magick.

Usage

```
gt_save_crop(
  data,
  file = NULL,
  bg = "white",
  whitespace = 50,
  zoom = 2,
  expand = 5,
  width = NULL
)
```

Arguments

<code>data</code>	A gt table object to save.
<code>file</code>	Character. A path ending in <code>.png</code> , <code>.jpg</code> , or <code>.jpeg</code> to write the image to. Defaults to <code>NULL</code> , which returns the image without writing it.
<code>bg</code>	Character. The background color of the padded border. Defaults to <code>"white"</code> .
<code>whitespace</code>	Numeric. The border, in pixels, left around the trimmed table. Defaults to 50.
<code>zoom</code>	Numeric. The rendering zoom factor passed to the underlying save. A higher value gives a sharper image. Defaults to 2.
<code>expand</code>	Numeric. The pixel expansion passed to the underlying save. Defaults to 5.
<code>width</code>	Optional. A final output width in pixels. The image is scaled to it, height following, so a series of tables can share one width. Defaults to <code>NULL</code> , which leaves the rendered width alone.

Details

The table is first written to a temporary file at `zoom` and `expand`, then read back with `magick`, trimmed to its content, and given a `whitespace` border in `bg`. When `width` is set the finished image is resized to that width with the height following, so several tables saved at the same width line up when posted together. The temporary file is removed afterward.

Value

Invisibly, `file` after writing the cropped image to it, or the encoded image as a raw vector when `file` is `NULL`.

See Also

[gt_social_crop\(\)](#) for padding onto a fixed-ratio canvas.

Examples

```
library(gt)

afc_east <- subset(
  sdv_example_standings, division == "AFC East",
  c(team_name, wins:points_against)
)
gt(afc_east) |>
  gt_theme_broadsheet() |>
  gt_save_crop(tempfile(fileext = ".png"))

# pin a shared width across a series
atlantic <- subset(
  sdv_example_standings, division == "Atlantic",
  c(team_name, wins, losses, win_pct:points_against)
)
gt(atlantic) |>
```

```
gt_save_crop(tempfile(fileext = ".png"), width = 900, bg = "#FBFAF7")
```

gt_scale_note	<i>Scale columns and disclose the scaling</i>
---------------	---

Description

Divides the selected columns and records the scaling in the same call, so the numbers and the disclosure cannot drift apart.

Usage

```
gt_scale_note(
  gt_object,
  columns,
  divisor = 1000,
  note = NULL,
  where = c("source_note", "label", "both"),
  label_suffix = NULL,
  decimals = 0,
  ...
)
```

Arguments

gt_object	A gt table object to modify.
columns	The column or columns to scale.
divisor	Numeric. The amount to divide by. Defaults to 1000.
note	Optional. The disclosure text. If NULL, it is derived from divisor, giving "Figures in thousands." for 1e3, "Figures in millions." for 1e6, and so on. Divisors without a common name fall back to "Figures divided by 2,500." Defaults to NULL.
where	Character. Where the disclosure goes. One of "source_note", "label" to append a suffix to the column labels instead, or "both". Defaults to "source_note".
label_suffix	Optional. The suffix appended to column labels when where includes "label". If NULL, it is derived from divisor (for example, "(000s)"). Defaults to NULL.
decimals	Integer. The number of decimal places for the scaled values. Defaults to 0.
...	Additional arguments passed to <code>gt::fmt_number</code> .

Details

Scaling is applied through `gt::fmt_number()` and its `scale_by` argument, so the underlying data is left untouched.

Value

Returns a modified gt table with the columns scaled and the scaling disclosed.

Examples

```
library(gt)

# an NBA season's points, scored and allowed
atlantic <- subset(
  sdv_example_standings, division == "Atlantic",
  c(team_name, points_for, points_against)
)

# Boston's 9,418 points render as 9.4, with "Figures in thousands." beneath
gt(atlantic) |>
  gt_scale_note(c(points_for, points_against), divisor = 1e3, decimals = 1)

# disclose in the column labels instead
gt(atlantic) |>
  gt_scale_note(c(points_for, points_against),
    divisor = 1e3, decimals = 2, where = "label"
  )
```

gt_sdv_cols_label	<i>Render Logos in 'gt' Table Column Labels</i>
-------------------	---

Description

Translate team abbreviations into logos and render these images in column labels of 'gt' tables.

Usage

```
gt_sdv_cols_label(
  gt_object,
  columns = gt::everything(),
  league,
  height = 30,
  mark_type = c("logo", "wordmark", "headshot"),
  id_system = "auto",
  season = NULL,
  strict = FALSE
)
```

Arguments

gt_object	A table object created using <code>gt::gt()</code> .
columns	The columns whose labels should be replaced with logos.

league	A league key from sdv_leagues() .
height	The absolute height (px) of the image.
mark_type	One of "logo", "wordmark" or "headshot": whether the column names are teams (logo / wordmark) or player IDs (headshot).
id_system	How the column names are read. "auto" (the default) is sdv_logo_url() 's automatic reading for logo and wordmark labels, and means "espn" when mark_type = "headshot"; otherwise as in sdv_headshot_url() .
season, strict	For logo and wordmark labels only (headshots take neither): the season whose mark to show, and whether an unresolved label is an error. With strict = TRUE every label that is not a team errors, including the non-team columns the default columns = gt::everything() takes.

Value

An object of class gt_tbl.

See Also

[gt_sdv_logos\(\)](#), [gt_sdv_wordmarks\(\)](#), [gt_sdv_headshots\(\)](#)

Examples

```
library(gt)
library(sdvplotR)

df <- data.frame(
  KC = 1:3,
  BUF = 4:6,
  SF = 7:9
)

df |>
  gt() |>
  gt_sdv_cols_label(columns = c("KC", "BUF", "SF"), league = "nfl")
```

gt_sdv_headshots	<i>Render Player Headshots in 'gt' Tables</i>
------------------	---

Description

Translate player IDs to player headshots and render these images in html tables with the 'gt' package. IDs are ESPN athlete IDs in every league, or as id_system says. IDs that resolve to no headshot are left as text.

Usage

```
gt_sdv_headshots(
  gt_object,
  columns,
  league,
  height = 30,
  locations = NULL,
  id_system = "espn"
)
```

Arguments

gt_object	A table object created using <code>gt::gt()</code> .
columns	The columns for which the image translation should be applied. Argument has no effect if locations is not NULL.
league	A league key from <code>sdv_leagues()</code> .
height	The absolute height (px) of the image in the table cell.
locations	If NULL (the default), the function will render logos in argument columns. Otherwise, the cell or set of cells to be associated with the team name transformation. Only <code>gt::cells_body()</code> , <code>gt::cells_stub()</code> , <code>gt::cells_column_labels()</code> , and <code>gt::cells_row_groups()</code> helper functions can be used here.
id_system	"espn" (ESPN athlete ids; the default, for every league with ESPN headshots: nfl, nba, wnba, mlb, nhl, cfb, mbb, wbb), "gsis" (nflverse's NFL gsis ids, such as "00-0033873", read through a published map) or "league" (the league's own player id: NBA / WNBA Stats PERSON_ID, MLBAM, NHL API; for the NFL, the gsis id).

Value

An object of class `gt_tbl`.

See Also

[gt_sdv_logos\(\)](#), [gt_sdv_wordmarks\(\)](#), [gt_sdv_cols_label\(\)](#)

Examples

```
library(gt)
library(sdvplotR)

df <- data.frame(
  player_id = c("00-0033873", "00-0026498", "00-0035228"),
  player_name = c("P.Mahomes", "M.Stafford", "K.Murray")
)

df |>
  gt() |>
  gt_sdv_headshots(columns = "player_id", league = "nfl", id_system = "gsis")
```

Description

Translate team abbreviations into logos and render these images in html tables with the 'gt' package.

Usage

```
gt_sdv_logos(
  gt_object,
  columns,
  league,
  height = 30,
  locations = NULL,
  include_name = FALSE,
  season = NULL,
  id_system = "auto",
  strict = FALSE
)
```

Arguments

gt_object	A table object created using gt::gt() .
columns	The columns for which the image translation should be applied. Argument has no effect if locations is not NULL.
league	A league key from sdv_leagues() .
height	The absolute height (px) of the image in the table cell.
locations	If NULL (the default), the function will render logos in argument columns. Otherwise, the cell or set of cells to be associated with the team name transformation. Only gt::cells_body() , gt::cells_stub() , gt::cells_column_labels() , and gt::cells_row_groups() helper functions can be used here.
include_name	If TRUE, keep the cell's text after the logo, so a cell shows logo and name (what cbbplotR 's gt_cbb_teams() did), and the image gets an empty alt so the name is not read twice. Defaults to FALSE, the logo alone, whose alt is the team's full name.
season	NULL (the default) for today's marks, or one season whose marks every cell shows (the ending year for the NHL). See the Historical logos section.
id_system, strict	How the team values are read and what an unresolved one does, as in sdv_logo_url() .

Value

An object of class `gt_tbl`.

Historical logos

Given a season, a team is drawn with the mark it wore that season where the SportsDataverse logo archive dates its marks, and with its current mark otherwise. The season also picks the team a reused code meant then (see `sdv_resolve()`). The images are copies kept in the archive, so they don't change when a league reuses a file name. SVG marks are read with the 'rsvg' package. season takes single years: the ending year for split seasons (2005 for 2004-05).

See Also

`gt_sdv_wordmarks()`, `gt_sdv_headshots()`, `gt_sdv_cols_label()`

Examples

```
library(gt)
library(sdvplotR)

teams <- sdv_teams("nfl")$abbr[1:8]
df <- data.frame(
  team = teams,
  logo = teams,
  wins = sample(1:16, 8)
)

df |>
  gt() |>
  gt_sdv_logos(columns = "logo", league = "nfl")

# logo and name in one cell; NCAA.com school names resolve too
data.frame(team = c("Iowa St.", "St. John's (NY)"), net = c(8, 12)) |>
  gt() |>
  gt_sdv_logos(columns = "team", league = "mbb", height = 20, include_name = TRUE)

# the marks of the 1994-95 season
data.frame(team = c("QUE", "HFD", "WIN"), pts = c(65, 43, 39)) |>
  gt() |>
  gt_sdv_logos(columns = "team", league = "nhl", season = 1995)
```

gt_sdv_wordmarks

Render Wordmarks in 'gt' Tables

Description

Translate team abbreviations into wordmarks and render these images in html tables with the 'gt' package.

Usage

```
gt_sdv_wordmarks(
  gt_object,
  columns,
  league,
  height = 30,
  locations = NULL,
  season = NULL,
  id_system = "auto",
  strict = FALSE
)
```

Arguments

<code>gt_object</code>	A table object created using <code>gt::gt()</code> .
<code>columns</code>	The columns for which the image translation should be applied. Argument has no effect if <code>locations</code> is not <code>NULL</code> .
<code>league</code>	A league key from <code>sdv_leagues()</code> .
<code>height</code>	The absolute height (px) of the image in the table cell.
<code>locations</code>	If <code>NULL</code> (the default), the function will render logos in argument columns. Otherwise, the cell or set of cells to be associated with the team name transformation. Only <code>gt::cells_body()</code> , <code>gt::cells_stub()</code> , <code>gt::cells_column_labels()</code> , and <code>gt::cells_row_groups()</code> helper functions can be used here.
<code>season</code>	<code>NULL</code> (the default) for today's marks, or one season whose marks every cell shows (the ending year for the NHL). See the Historical logos section.
<code>id_system, strict</code>	How the team values are read and what an unresolved one does, as in <code>sdv_logo_url()</code> .

Value

An object of class `gt_tbl`.

See Also

`gt_sdv_logos()`, `gt_sdv_headshots()`, `gt_sdv_cols_label()`

Examples

```
library(gt)
df <- data.frame(team = c("KC", "BUF", "SF"), wins = c(14, 13, 12))
df |>
  gt() |>
  gt_sdv_wordmarks(columns = "team", league = "nfl")
```

gt_set_font

Set one font family across a whole gt table

Description

Applies a single font family to every styleable part of a gt table in one call: the title and subtitle, the stubhead, column spanners and labels, row groups, the stub, the body, footnotes, and source notes. Setting the same font through `gt::tab_options()` means naming each `*_font` option in turn.

Usage

```
gt_set_font(
  gt_object,
  font_family,
  from_google_font = TRUE,
  weight = NULL,
  style = NULL
)
```

Arguments

gt_object	A gt table object to modify.
font_family	Character. The font family to apply to the whole table.
from_google_font	Logical. Should the font be pulled from Google Fonts through <code>gt::google_font()</code> ? FALSE treats font_family as a font already installed on the local machine. Defaults to TRUE.
weight	The font weight applied to every part, passed to <code>gt::cell_text()</code> . Either a keyword such as "bold" or a numeric weight. Defaults to NULL, which leaves the weight alone.
style	Character. The font style applied to every part, one of "normal", "italic", or "oblique". Defaults to NULL, which leaves the style alone.

Details

The font is applied with a single `gt::tab_style()` over a list of cell locations covering each part of the table. Summary and grand-summary cells are not included, so a font set here does not reach them. When `from_google_font` is TRUE the family is wrapped in `gt::google_font()`, which adds the import so the font renders in an exported table without relying on it being installed locally.

Value

Returns a modified gt table with the font family applied to every covered part.

Examples

```
library(gt)

nfc_south <- subset(
  sdv_example_standings, division == "NFC South",
  c(team_name, wins:points_against)
)
gt(nfc_south) |> gt_set_font("Oswald")

# use a font already installed locally
gt(nfc_south) |> gt_set_font("Helvetica", from_google_font = FALSE)
```

gt_significance	<i>Add significance stars to a gt table</i>
-----------------	---

Description

Appends significance notation to an estimate column based on a column of p-values, then writes the matching legend into a source note. The p-value column is hidden by default, since the stars stand in for it. This is the usual convention for regression tables in the social and medical sciences.

Usage

```
gt_significance(
  gt_object,
  columns,
  p_columns,
  levels = c(0.01, 0.05, 0.1),
  symbols = c("***", "**", "*"),
  superscript = TRUE,
  size = "0.7em",
  legend = TRUE,
  legend_text = NULL,
  hide_p = TRUE
)
```

Arguments

gt_object	A gt table object to modify.
columns	The estimate column or columns to annotate.
p_columns	The column or columns holding the p-values, paired positionally with columns. Supply one p-value column per estimate column.
levels	A numeric vector of significance thresholds, in ascending order (strictest first). Defaults to c(0.01, 0.05, 0.1).
symbols	A character vector of notation for each level. Must be the same length as levels. Defaults to c("***", "**", "*").

<code>superscript</code>	Logical. Should the stars be rendered as superscript? Defaults to TRUE.
<code>size</code>	Character. The size of the stars, as a CSS size. Defaults to "0.7em".
<code>legend</code>	Logical. Should the legend be added as a source note? Defaults to TRUE.
<code>legend_text</code>	Optional. Custom legend text. If NULL, the legend is built from <code>levels</code> and <code>symbols</code> . Defaults to NULL.
<code>hide_p</code>	Logical. Should the p-value columns be hidden once the stars are applied? Defaults to TRUE.

Details

The legend is generated from the same `levels` and `symbols` used to place the stars, so changing one updates the other.

Each value takes the notation for the strictest threshold it satisfies, so with the defaults a p-value of 0.004 gets *** and not *. Values that meet none of the thresholds are left alone, as are NA p-values.

Value

Returns a modified gt table with significance notation applied.

See Also

[`gt\_fmt\_rank\(\)`](#), which uses the same superscript approach for ordinal suffixes.

Examples

```
library(gt)

# what a season's points scored and allowed say about its win percentage
nfl <- subset(sdv_example_standings, league == "nfl")
fit <- lm(win_pct ~ points_for + points_against + conference, data = nfl)
results <- data.frame(
  Term = rownames(summary(fit)$coefficients),
  Estimate = summary(fit)$coefficients[, 1],
  SE = summary(fit)$coefficients[, 2],
  p = summary(fit)$coefficients[, 4]
)

gt(results) |>
  fmt_number(c(Estimate, SE), decimals = 4) |>
  gt_significance(Estimate, p)

# daggers instead of stars, at a single threshold
gt(results) |>
  gt_significance(Estimate, p,
    levels = 0.05, symbols = "†",
    legend_text = "† p < .05"
  )
```

gt_snake

*Lay a long table out in side-by-side blocks***Description**

Breaks a long table into two or more blocks set beside each other, each with its own column labels. A sixty-row ranking becomes two columns of thirty, which is the usual treatment in print and the difference between an image that is unreadably tall and one that fits the frame.

Usage

```
gt_snake(
  gt_object,
  n_cols = 2,
  rows_per_col = NULL,
  gap = 20,
  fill = "",
  clean_gaps = TRUE
)
```

Arguments

gt_object	A gt table object to modify. Body-cell styling applied before this call is carried through the reshape; formatting and content transforms are not, so they still come after (see Details).
n_cols	Integer. The number of blocks to lay out. Defaults to 2.
rows_per_col	Integer. Rows in each block. Given this, n_cols is worked out from the data instead. Defaults to NULL.
gap	Numeric. The space between blocks in pixels, added as an empty spacer column so it does not disturb the alignment of the columns either side of it. Its borders, fill, and rules are scrubbed so it stays clean whitespace under any theme (see Details). 0 for no gap. Defaults to 20.
fill	Character. What to show in the trailing cells when the rows do not divide evenly. Defaults to "", an empty cell.
clean_gaps	Logical. Should the spacer columns be scrubbed of borders, fill, and rules so the gap reads as clean whitespace (see Details)? Set FALSE when you style the gap yourself, e.g. with a coloured border from <code>gt::tab_style()</code> and <code>gt::cell_borders()</code> . Defaults to TRUE.

Details

The table is rebuilt from its data. Body-cell **styles** set beforehand, with `gt_highlight_cells()` or `tab_style()` on `cells_body()`, are carried through: each is moved to its block's suffixed column and its row within that block, so a fill on the long table lands on the same cell once snaked. This means you can color the natural, un-snaked grid and snake afterwards.

Formatting and content transforms are not styles and are not remapped, so `fmt_*`, `text_transform()`, and `gtExtras::gt_img_rows()` still have to be applied after the snake, against the suffixed names above. Themes go afterwards too. Any heading and source notes already set are carried over.

To reshape a parallel frame (a precomputed mask, per-cell colors) to match the same blocks, see [gt_snake_align\(\)](#).

`gt::gt_split()` also divides a table, but it returns a group of separate tables to be rendered one after another, and it errors on any table carrying body-cell styles, which every theme in this package sets.

Value

Returns a gt table laid out in blocks.

Column names after snaking

Snaking is a reshape. The same columns appear once per block, so they cannot all keep the same name: each is suffixed with its block number. A four-column table split into two blocks becomes eight columns.

before	after gt_snake(n_cols = 2)
-----	-----
team wins losses win_pct	team_1 wins_1 losses_1 win_pct_1 <- rows 1-15
	team_2 wins_2 losses_2 win_pct_2 <- rows 16-30

The *labels* are untouched, so the table still reads team wins losses win_pct twice over. Only the names you address in code change.

That gives three ways to target columns afterwards:

```
# every block of one column, via a tidyselect helper
fmt_number(starts_with("win_pct"), decimals = 3)

# one block only, by its full name
tab_style(cell_text(weight = "bold"), cells_body(columns = team_2))

# the whole table, exactly as usual
cols_align(columns = everything(), align = "center")
```

Reach for `starts_with()` rather than listing `team_1`, `team_2` by hand: it keeps working if you change `n_cols` later. Watch out for prefixes that overlap, though. With columns named `pts` and `pts_pg`, `starts_with("pts")` catches both blocks of both; `matches("^pts_[0-9]+$")` catches only `pts`.

The gap between blocks is itself an empty column, named `.gap1`, `.gap2`, and so on. It holds nothing and carries no label. To target the spacers in a later call, use `starts_with(".gap")` or `matches("^\\.gap")`; `everything()` selects them too. Set `gap = 0` if you would rather it did not exist.

Because the gap is a real column, a theme or `gt_border_grid()` would otherwise draw its column borders and row rules straight through it, leaving the "whitespace" boxed in. When `clean_gaps =`

TRUE (the default), the body of each spacer and the inner edges of the columns either side are cleared with scoped !important CSS, so it wins even against borders applied after the snake; the spacer's own column label is cleared through gt directly, which holds up even when tab_spanner() merges header cells. The table's id is set (or carried over from the input) so the CSS can be scoped to it. Set clean_gaps = FALSE to style the gap yourself instead.

See Also

[gt_stack_tables\(\)](#) for stacking separate tables vertically.

Examples

```
library(gt)

nba <- subset(
  sdv_example_standings, league == "nba",
  c(team, wins, losses, win_pct)
)
nba <- nba[order(-nba$win_pct), ]

# three blocks of ten
gt(nba) |>
  gt_snake(n_cols = 3) |>
  gt_theme_broadsheet()

# or set the block length and let the count follow
gt(nba) |>
  gt_snake(rows_per_col = 8, gap = 30) |>
  gt_theme_swiss()

# formatting after the reshape: note the suffixed names
gt(nba) |>
  gt_snake(n_cols = 3) |>
  gt_theme_swiss() |>
  fmt_number(starts_with("win_pct"), decimals = 3) |>
  cols_align(columns = everything(), align = "center")

# and picking out a single block
gt(nba) |>
  gt_snake(n_cols = 3) |>
  gt_theme_swiss() |>
  tab_style(
    gt::cell_text(weight = "bold"),
    gt::cells_body(columns = team_1)
  )
```

Description

Lays a data frame out in side-by-side blocks the same way `gt_snake()` lays out the table, so a frame you computed against the long data still lines up after the snake. It is the escape hatch for a precomputed grid you do not want to rebuild as a condition: a logical mask for `gt_highlight_cells()`, a frame of per-cell colors or text, anything the same shape as the snaked body.

Usage

```
gt_snake_align(x, n_cols = 2, rows_per_col = NULL, fill = NA)
```

Arguments

<code>x</code>	A data frame or matrix with one row per row of the un-snaked table.
<code>n_cols</code>	Integer. The number of blocks, matching the <code>gt_snake()</code> call. Defaults to 2.
<code>rows_per_col</code>	Integer. Rows in each block. Given this, <code>n_cols</code> is worked out from the data instead, exactly as <code>gt_snake()</code> does it. Defaults to <code>NULL</code> .
<code>fill</code>	What to put in the trailing cells when the rows do not divide evenly. Defaults to <code>NA</code> , which <code>gt_highlight_cells()</code> reads as no match.

Details

Most of the time you do not need this. Body-cell styling applied before `gt_snake()` is carried through the reshape on its own, so coloring the long grid and then snaking already lands the fills in the right place. Reach for this only when the grid lives outside the table's styles.

Pass the same `n_cols` (or `rows_per_col`) you passed to `gt_snake()`. Each block's columns are suffixed with its block number, `_1`, `_2`, and so on, to match the names `gt_snake()` gives the table. The result has `ceiling(nrow(x) / n_cols)` rows and `ncol(x) * n_cols` columns, so it is the shape `gt_highlight_cells()` expects for the block of snaked columns.

Value

Returns a data frame of the reshaped blocks.

See Also

`gt_snake()`, which reshapes the table itself and carries body styling through.

Examples

```
library(gt)

nfc <- subset(
  sdv_example_standings, conference == "NFC",
  c(team, wins, losses)
)

# a logical grid the same shape as the data, built before the snake:
# double-digit wins or losses
mask <- data.frame(team = FALSE, nfc[c("wins", "losses")] >= 10)
```

```
# after the snake, each column carries its block number as a suffix
snaked <- unlist(lapply(1:2, function(i) paste0(names(nfc), "_", i)))

nfc |>
  gt() |>
  gt_snake(n_cols = 2) |>
  gt_highlight_cells(tidyselect::all_of(snaked),
    condition = gt_snake_align(mask, n_cols = 2),
    fill = "#EBD668"
  )
```

gt_social_crop

*Save a gt table onto a fixed-ratio canvas***Description**

A variant of [gt_save_crop\(\)](#) that centers the trimmed table on a canvas of a given aspect ratio, for posting to social platforms. The table itself is never cropped; the short side of the canvas is expanded until the ratio is met.

Usage

```
gt_social_crop(
  data,
  file = NULL,
  aspect_ratio = "1:1",
  bg = "white",
  whitespace = 60,
  gravity = "center",
  zoom = 2,
  expand = 5,
  width = NULL
)
```

Arguments

data	A gt table object to save.
file	Optional. A path to write the image to. Defaults to NULL.
aspect_ratio	Character or numeric. The target ratio, given as "1:1", "16:9", "4:5", or a number. Defaults to "1:1".
bg	Character. The background color of the canvas. Defaults to "white".
whitespace	Numeric. The amount of whitespace to leave around the table before padding out to the ratio. Defaults to 60.
gravity	Character. Where the table sits on the canvas, passed to <code>magick::image_extent</code> . Defaults to "center".

zoom	Numeric. The rendering zoom factor. Defaults to 2.
expand	Numeric. The pixel expansion passed to the underlying save. Defaults to 5.
width	Optional. A final output width in pixels. The finished canvas is scaled to it, keeping the aspect ratio, so a series of posts can share one width. Defaults to NULL.

Value

Invisibly, file after writing the cropped and padded image to it, or the encoded image as a raw vector when file is NULL.

See Also

[gt_save_crop\(\)](#) for a plain trimmed save.

Examples

```
library(gt)

nfc_west <- subset(
  sdv_example_standings, division == "NFC West",
  c(team_name, wins:points_against)
)

gt(nfc_west) |>
  gt_theme_broadsheet(density = "social") |>
  gt_social_crop(tempfile(fileext = ".png"), aspect_ratio = "4:5")

# match the canvas to a dark theme
gt(nfc_west) |>
  gt_theme_midnight() |>
  gt_social_crop(tempfile(fileext = ".png"), bg = "#0C0D10")
```

gt_social_tag	<i>Add social handles to a gt table</i>
---------------	---

Description

Renders a set of accounts, each an icon plus a handle, into the table’s source note, styled like the bottom line of [gt_538_caption\(\)](#). Icons come from Font Awesome through the fontawesome package.

Usage

```
gt_social_tag(
  gt_object,
  accounts,
  caption = NULL,
  stack = FALSE,
  separator = " | ",
  align = "right",
  icon_color = NULL,
  icon_height = "0.9em",
  text_size = NULL,
  text_weight = NULL,
  ...
)
```

Arguments

gt_object	A gt table object to modify.
accounts	A named character vector mapping platform to handle, such as <code>c(x = "@you", bluesky = "you.bsky.social")</code> . Names are matched to Font Awesome brand icons, and friendly aliases are accepted, including <code>x</code> and <code>twitter</code> , <code>ig</code> and <code>instagram</code> , <code>bsky</code> and <code>bluesky</code> , <code>gh</code> and <code>github</code> , <code>yt</code> and <code>youtube</code> , <code>web</code> and <code>website</code> , and <code>email</code> . An unrecognized name is used as the icon name directly.
caption	Optional. A caption rendered as the top footnote line. Defaults to <code>NULL</code> , which shows only the handle line.
stack	Logical. Should the accounts be stacked vertically, one per line, instead of sitting in a row? Defaults to <code>FALSE</code> .
separator	Character. The string placed between accounts when not stacked. Defaults to <code>" "</code> .
align	Character. The alignment of the handle line. One of <code>"right"</code> , <code>"center"</code> , or <code>"left"</code> . Defaults to <code>"right"</code> .
icon_color	Optional. A hex color for the icons. If <code>NULL</code> , they inherit the surrounding text color. Defaults to <code>NULL</code> .
icon_height	Character. The height of the icons, as a CSS size. Given in <code>em</code> it scales with <code>text_size</code> . Defaults to <code>"0.9em"</code> .
text_size	Optional. The font size for the handles, as a CSS size. If <code>NULL</code> , it inherits the source-note size. Defaults to <code>NULL</code> .
text_weight	Optional. The font weight for the handles. If <code>NULL</code> , it inherits the source-note weight. Defaults to <code>NULL</code> .
...	Additional arguments passed to <code>gt_538_caption()</code> , used only when a caption is supplied.

Details

If a caption is supplied it becomes the top line, a bordered footnote in the same manner as `gt_538_caption()`, with the handles beneath it. With no caption, only the handle line is shown.

If an icon name is not present in your installed version of fontawesome, the function stops with a message naming the icon and the version, instead of rendering a broken glyph.

Value

Returns a modified gt table with the handles, and any caption, added.

See Also

[gt_538_caption\(\)](#).

Examples

```
library(gt)

afc_east <- subset(
  sdv_example_standings, division == "AFC East",
  c(team_name, wins:points_against)
)

gt(afc_east) |>
  gt_social_tag(c(x = "@yourhandle", gh = "yourname"))

# with a caption line above, stacked and left-aligned
gt(afc_east) |>
  gt_social_tag(
    c(x = "@yourhandle", web = "example.com"),
    caption = "Data: nflreadr and nflseedR",
    stack = TRUE, align = "left"
  )
```

gt_spotlight

Focus on rows of a gt table by dimming the rest

Description

Emphasizes one or more rows and mutes everything else. `gtExtras::gt_highlight_rows()` makes a row louder; this leaves it as the only row at full strength.

Usage

```
gt_spotlight(
  gt_object,
  rows,
  columns = gt::everything(),
  fill = NULL,
  text_color = NULL,
  bold = TRUE,
```

```

    accent_color = NULL,
    accent_width = 4,
    accent_column = NULL,
    dim_color = "auto",
    if_none = c("warn", "dim", "ignore")
  )

```

Arguments

gt_object	A gt table object to modify.
rows	The rows to focus on. Either an expression evaluated against the table's data, such as <code>cyl == 8</code> , or a numeric vector of row indices.
columns	The columns the spotlight covers. Cells outside this selection are dimmed, in the focused rows as well as the others, so only the selected cells are left at full strength. Defaults to all of them, which lights the whole row.
fill	Optional. A hex color for the fill behind the focused rows. Defaults to <code>NULL</code> .
text_color	Optional. A hex color for the text of the focused rows. Defaults to <code>NULL</code> .
bold	Logical. Should the focused rows be bolded? Defaults to <code>TRUE</code> .
accent_color	Optional. A color for a bar drawn on the left edge of the focused rows. Supplying one is what turns the bar on, the same way fill and text_color work. Defaults to <code>NULL</code> , no bar.
accent_width	Numeric. The width of that bar in pixels, used only when accent_color is given. Defaults to 4.
accent_column	The column to draw the bar against, on its left edge. If <code>NULL</code> , the leftmost rendered column is used, which is what marks the start of the row. Name a column when the row you are lighting sits elsewhere, as with the second block of a <code>gt_snake()</code> table. More than one may be given, which draws a bar against each. Defaults to <code>NULL</code> .
dim_color	Character. The text color applied to every other row. "auto", the default, blends the table's text toward its background until it sits just above 4.5:1 contrast against it (WCAG AA for text), so the rows read as muted rather than disabled, on a light or a dark theme. Apply the theme first: the background is read from the table as set so far. Pass a color to choose it yourself, or <code>NULL</code> to leave the other rows alone, emphasizing without dimming.
if_none	Character. What to do when rows matches nothing. "warn" leaves the table alone and says so. "dim" dims the whole table, which is what you want when several tables are laid out together with <code>gt_grid()</code> and the highlighted row lives in only one of them. "ignore" leaves it alone silently. Defaults to "warn".

Details

Rows can be given as a filtering expression evaluated against the underlying data, or as plain row numbers.

Value

Returns a modified gt table with the chosen rows emphasized.

See Also

[gt_outline\(\)](#) for marking a threshold rather than a row.

Examples

```
library(gt)

nfc <- subset(
  sdv_example_standings, division %in% c("NFC North", "NFC West"),
  c(team_name, wins, losses, conference_rank)
)

# by expression: the playoff seeds
gt(nfc) |> gt_spotlight(rows = conference_rank <= 7)

# by position, with an accent bar
gt(nfc) |>
  gt_spotlight(rows = 2, accent_color = "#0054AD", fill = "#EEF3FA")

# narrowing the spotlight to some columns dims the rest of the row too
gt(nfc) |> gt_spotlight(wins:losses, rows = 3, accent_color = "darkblue")

# put the bar somewhere other than the first column
gt(nfc) |>
  gt_spotlight(losses, rows = 3, accent_color = "darkblue", accent_column = losses)

# across several tables: blocks without a 12-win team dim rather than stay lit
chunks <- split(nfc, ceiling(seq_len(nrow(nfc)) / 4))
tbls <- lapply(chunks, function(x) {
  gt(x) |> gt_spotlight(rows = wins >= 12, if_none = "dim")
})
gt_grid(tbls, ncol = 2)
```

gt_stack_tables

Stack several gt tables vertically

Description

Places a list of tables one above another in a single block, with an optional shared heading and footer. Each table keeps its own columns, widths, and header. [gt_snake\(\)](#) folds one table into blocks instead, and [gt_grid\(\)](#) arranges tables side by side.

Usage

```
gt_stack_tables(
  tables = NULL,
  gap = 16,
  align = c("center", "left", "right"),
```

```

    title = NULL,
    subtitle = NULL,
    caption = NULL,
    source_note = NULL,
    caption_rule = FALSE,
    title_style = list(),
    subtitle_style = list(),
    caption_style = list(),
    source_note_style = list(),
    file = NULL,
    bg = "white",
    whitespace = 50,
    zoom = 2
)

```

Arguments

tables	A list of gt table objects to stack.
gap	Numeric. The space between tables in pixels. Defaults to 16.
align	Character. How tables of differing width line up, one of "center", "left", or "right". Defaults to "center".
title	Character. An optional heading above the stack. Defaults to NULL.
subtitle	Character. An optional line below title. Defaults to NULL.
caption	Character. An optional note below the stack. Defaults to NULL.
source_note	Character. An optional second line below caption, right-aligned by default. Set both, with caption_rule = TRUE, for the split caption gt_538_caption() gives a single table. Defaults to NULL.
caption_rule	Logical. Should a hairline sit between caption and source_note? Defaults to FALSE.
title_style, subtitle_style, caption_style, source_note_style	Named lists of style options. See Styling. Each defaults to an empty list.
file	Optional. A path to write a PNG to. If NULL, the stack is returned for the viewer instead. Defaults to NULL.
bg	Character. The background color, used when saving. Defaults to "white".
whitespace	Numeric. Padding left around the stack when saving, in pixels. Defaults to 50.
zoom	Numeric. The rendering zoom factor used when saving. Defaults to 2.

Details

The stack is assembled as HTML rather than a gt table, since each table keeps its own columns and header. That makes it a last step, after every table is themed and formatted, and it means the output cannot be passed back into further gt calls. Saving happens here through webshot2 rather than through [gt_save_crop\(\)](#), so a file write needs the webshot2 package.

A shared heading and footer sit outside the stack in a shrink-to-fit wrapper, so they line up with the tables rather than the page. Google fonts named in a style list are loaded through a stylesheet link, since the composed HTML does not run through gt's own font machinery.

Value

An `htmltools::browsable()` tag list of the stacked tables, which prints in the viewer; with `file`, invisibly file after writing the PNG to it.

Styling

`title_style`, `subtitle_style`, `caption_style`, and `source_note_style` are named lists following the same convention as `gt_grid()` and `gt_title_header()`. Recognized keys are `font` (a Google font name), `size`, `color`, `weight`, `italic`, `spacing` (letter spacing), `transform` (such as "uppercase"), and `align`, plus `line_height`, `margin_top`, `margin_bottom`, `padding_top`, and `padding_bottom`. Any length takes a number, read as pixels, or a CSS string. Any key left out keeps its default.

See Also

[gt_grid\(\)](#) for a side-by-side grid and [gt_snake\(\)](#) for folding one long table into blocks.

Examples

```
library(gt)

t1 <- gt(subset(
  sdv_example_standings, division == "AFC East",
  c(team_name, wins, losses)
))
t2 <- gt(subset(
  sdv_example_standings, division == "Atlantic",
  c(team_name, wins, losses)
))

gt_stack_tables(list(t1, t2))

# one heading over the stack, saved straight to an image; saving needs a
# headless Chrome (webshot2)

gt_stack_tables(
  list(t1, t2),
  title = "Two leagues",
  subtitle = "Stacked into one block",
  title_style = list(font = "Oswald", size = 30, transform = "uppercase"),
  file = tempfile(fileext = ".png")
)
```

gt_theme_almanac	<i>Record-book theme for gt tables</i>
------------------	--

Description

The look of a printed statistical abstract. A slab body, narrow condensed labels, tight rows, and row banding to walk the eye across a wide table.

Usage

```
gt_theme_almanac(
  gt_object,
  accent = "#8C2F1E",
  density = c("compact", "comfortable", "social"),
  stripe = "#F1F1EF",
  ...
)
```

Arguments

gt_object	A gt table object to modify.
accent	Character. A hex color for the row-group labels and the rule above the table. Defaults to "#8C2F1E".
density	Character. The type and padding scale. One of "comfortable", "compact", or "social". See Density. Defaults to "compact".
stripe	Character. A hex color for the banded rows. Pass NA to switch banding off and keep the rest of the theme. Defaults to "#F1F1EF".
...	Additional arguments passed to <code>gt::tab_options</code> , applied last so they override anything the theme sets.

Details

It is the only theme in the package that bands its rows.

Value

Returns a modified gt table with the theme applied.

Density

density sets the type and padding scale together. "comfortable" uses a 14px body with roomy rows, "compact" a 12px body with tight rows, and "social" a 17px body with generous rows and a larger title, at the scale `gt_save_crop()` and `gt_social_crop()` export at.

Figures

Examples

```
library(gt)
nfc <- subset(
  sdv_example_standings, conference == "NFC",
  c(team_name, wins:points_against)
)
gt(head(nfc, 12)) |> gt_theme_almanac()

# banding off, cooler accent
east <- subset(
  sdv_example_standings, league == "nba" & conference == "Eastern",
  c(team_name, wins, losses, win_pct:points_against)
)
gt(east) |> gt_theme_almanac(stripe = NA, accent = "#1F3A5F")
```

gt_theme_athletic	<i>The Athletic theme for gt tables</i>
-------------------	---

Description

Modeled on The Athletic's tables. A monospaced Spline Sans Mono body, uppercase Work Sans labels and titles, dotted rules between rows, thin vertical rules separating the columns, and a solid black row-group band with knocked-out white labels. All columns are centered.

Usage

```
gt_theme_athletic(
  gt_object,
  density = c("comfortable", "compact", "social"),
  ...
)
```

Arguments

gt_object	A gt table object to modify.
density	Character. The type and padding scale. One of "comfortable", "compact", or "social". See Density. Defaults to "comfortable".
...	Additional arguments passed to <code>gt::tab_options</code> , applied last so they override anything the theme sets.

Details

The row rules are a dotted top border applied to every body row, and the column separators are a thin left border on every column except the first, so the stub reads without a leading rule. Both are drawn with `gt::cell_borders()` rather than table options. A table id is resolved (or generated) up front so the `gt::opt_css()` block can pin the closing row border and heading padding to this table alone. `density` rescales the finished table, since this theme sets its sizes directly rather than deriving them from a scale.

Value

Returns a modified gt table with the theme applied.

Density

density scales the theme's type and row padding together. "comfortable" leaves every size as the theme sets it, "compact" scales both down, and "social" scales both up, to the scale `gt_save_crop()` and `gt_social_crop()` export at.

Figures**Examples**

```
library(gt)
afc_north <- subset(
  sdv_example_standings, division == "AFC North",
  c(team_name, wins:points_against)
)
gt(afc_north) |> gt_theme_athletic()
gt(afc_north) |> gt_theme_athletic(density = "compact")
```

gt_theme_booktabs	<i>Academic booktabs theme for gt tables</i>
-------------------	--

Description

Three horizontal rules and nothing else, the way LaTeX booktabs draws them. A heavy rule sits above the column labels, a lighter one below them, and a heavy rule closes the body. There are no vertical rules, no rules between data rows, and no fills. The type is a Times-compatible serif.

Usage

```
gt_theme_booktabs(
  gt_object,
  accent = "#111111",
  density = c("comfortable", "compact", "social"),
  ...
)
```

Arguments

gt_object	A gt table object to modify.
accent	Character. A hex color for the three rules and the row-group labels. Defaults to "#111111", near-black, for the standard black rules.

density	Character. The type and padding scale. One of "comfortable", "compact", or "social". See Density. Defaults to "comfortable".
...	Additional arguments passed to <code>gt::tab_options</code> , applied last so they override anything the theme sets.

Details

accent recolors the three rules and the row-group labels.

Value

Returns a modified gt table with the theme applied.

Density

density sets the type and padding scale together. "comfortable" uses a 14px body with roomy rows, "compact" a 12px body with tight rows, and "social" a 17px body with generous rows and a larger title, at the scale `gt_save_crop()` and `gt_social_crop()` export at.

Figures

See Also

`gt_theme_tufte()` for a lighter, minimal-ink relative, and `gt_significance()` for significance notation.

Examples

```
library(gt)

afc_west <- subset(
  sdv_example_standings, division == "AFC West",
  c(team_name, wins:points_against)
)
gt(afc_west) |> gt_theme_booktabs()

# a regression table, with significance stars and a colored rule
nfl <- subset(sdv_example_standings, league == "nfl")
fit <- lm(win_pct ~ points_for + points_against, data = nfl)
data.frame(
  Term = names(coef(fit)),
  Estimate = coef(fit),
  p = summary(fit)$coefficients[, 4]
) |>
gt() |>
fmt_number(Estimate, decimals = 4) |>
gt_significance(Estimate, p) |>
gt_theme_booktabs(accent = "#1A3E6F")
```

gt_theme_broadsheet *Editorial serif theme for gt tables*

Description

Sets a serif body under small letterspaced sans column labels, with hairline rules and no fills anywhere. Row groups are an uppercase label over a rule rather than a colored band.

Usage

```
gt_theme_broadsheet(
  gt_object,
  accent = "#A6081A",
  density = c("comfortable", "compact", "social"),
  paper = "white",
  ...
)
```

Arguments

gt_object	A gt table object to modify.
accent	Character. A hex color for the rule above the table and for the row-group labels. Defaults to "#A6081A".
density	Character. The type and padding scale. One of "comfortable", "compact", or "social". See Density. Defaults to "comfortable".
paper	Character. The table background. Either "white" for a warm off-white, "salmon" for the financial-press pink, or any hex color, in which case the hairline color is left neutral. Defaults to "white".
...	Additional arguments passed to <code>gt::tab_options</code> , applied last so they override anything the theme sets.

Details

accent recolors the rule above the table and the row-group labels.

Value

Returns a modified gt table with the theme applied.

Density

density sets the type and padding scale together. "comfortable" uses a 14px body with roomy rows, "compact" a 12px body with tight rows, and "social" a 17px body with generous rows and a larger title, at the scale `gt_save_crop()` and `gt_social_crop()` export at.

Figures

See Also

`gt_title_header()` for a richer header block, and `gt_legend_continuous()` for explaining a colored column.

Examples

```
library(gt)

nfc_east <- subset(
  sdv_example_standings, division == "NFC East",
  c(team_name, wins:points_against)
)
gt(nfc_east) |> gt_theme_broadsheet()

# financial-press pink, sized for an image export
pacific <- subset(
  sdv_example_standings, division == "Pacific",
  c(team_name, wins, losses, win_pct:points_against)
)
gt(pacific) |>
  gt_theme_broadsheet(paper = "salmon", accent = "#0F5257", density = "social")
```

gt_theme_brutalist	<i>Heavy-rule display theme for gt tables</i>
--------------------	---

Description

Heavy black rules, a solid label bar, and a display face at a large size.

Usage

```
gt_theme_brutalist(
  gt_object,
  accent = "#FF3B00",
  density = c("comfortable", "compact", "social"),
  ...
)
```

Arguments

gt_object	A gt table object to modify.
accent	Character. The theme's single accent color, used on the row-group labels. Defaults to "#FF3B00".

density	Character. The type and padding scale. One of "comfortable", "compact", or "social". See Density. Defaults to "comfortable".
...	Additional arguments passed to <code>gt::tab_options</code> , applied last so they override anything the theme sets.

Details

The accent color is used in exactly one place, on the row-group labels.

Value

Returns a modified gt table with the theme applied.

Density

density sets the type and padding scale together. "comfortable" uses a 14px body with roomy rows, "compact" a 12px body with tight rows, and "social" a 17px body with generous rows and a larger title, at the scale `gt_save_crop()` and `gt_social_crop()` export at.

Figures

Examples

```
library(gt)
afc_south <- subset(
  sdv_example_standings, division == "AFC South",
  c(team_name, wins:points_against)
)
gt(afc_south) |> gt_theme_brutalist()
gt(afc_south) |> gt_theme_brutalist(accent = "#0033FF", density = "social")
```

gt_theme_drench

Saturated single-color theme for gt tables

Description

Colors the entire table surface instead of putting a colored detail on a neutral background, tying the table to a single brand or publication color.

Usage

```
gt_theme_drench(
  gt_object,
  color = "#123F5E",
  density = c("comfortable", "compact", "social"),
  ...
)
```

Arguments

<code>gt_object</code>	A gt table object to modify.
<code>color</code>	Character. The hex color the table is drenched in. Anything from a near-black to a mid-saturation brand color works, and very pale colors flip the type to dark automatically. Defaults to "#123F5E".
<code>density</code>	Character. The type and padding scale. One of "comfortable", "compact", or "social". See Density. Defaults to "comfortable".
<code>...</code>	Additional arguments passed to <code>gt::tab_options</code> , applied last so they override anything the theme sets.

Details

Text, rules, and secondary type are all derived from color. The type color is whichever of black or white measures higher contrast against the background, and the rules are built by shifting the background itself, since gray laid over a saturated ground looks washed out.

Value

Returns a modified gt table with the theme applied.

Legibility

The secondary color used for column labels, the subtitle and source notes is blended toward the type color until it clears a 4.5:1 contrast ratio against the background, so it stays readable whatever color you pass.

Density

`density` sets the type and padding scale together. "comfortable" uses a 14px body with roomy rows, "compact" a 12px body with tight rows, and "social" a 17px body with generous rows and a larger title, at the scale `gt_save_crop()` and `gt_social_crop()` export at.

Figures**See Also**

[gt_theme_midnight\(\)](#) for a restrained dark background instead.

Examples

```
library(gt)
southwest <- subset(
  sdv_example_standings, division == "Southwest",
  c(team_name, wins, losses, win_pct:points_against)
)
gt(southwest) |> gt_theme_drench()
```

```
# a brand color, with a matching export canvas; saving needs a headless
# Chrome (webshot2)

gt(southwest) |>
  gt_theme_drench(color = "#4B1E78", density = "social") |>
  gt_social_crop(tempfile(fileext = ".png"), bg = "#4B1E78")
```

gt_theme_gtutils	<i>gtUtils theme for gt tables</i>
------------------	------------------------------------

Description

The package's house theme. A warm cream background, an Almarai body, Signika Negative titles, column labels and row groups, hairline separators between rows, and gray row-group bands carrying cream labels. All columns are centered.

Usage

```
gt_theme_gtutils(
  gt_object,
  density = c("comfortable", "compact", "social"),
  ...
)
```

Arguments

gt_object	A gt table object to modify.
density	Character. The type and padding scale. One of "comfortable", "compact", or "social". See Density. Defaults to "comfortable".
...	Additional arguments passed to <code>gt::tab_options</code> , applied last so they override anything the theme sets.

Details

The row separators are a single bottom border in gray (#8A817C) applied to every body row except the last, so the table closes on the background rather than a rule. The background is a warm cream (#FFFD5), and horizontal rules are otherwise set transparent. A table id is resolved (or generated) up front so the `gt::opt_css()` block can pin the closing row border and heading padding to this table alone. `density` rescales the finished table, since this theme sets its sizes directly rather than deriving them from a scale.

Value

Returns a modified gt table with the theme applied.

Density

density scales the theme's type and row padding together. "comfortable" leaves every size as the theme sets it, "compact" scales both down, and "social" scales both up, to the scale `gt_save_crop()` and `gt_social_crop()` export at.

Figures

Examples

```
library(gt)
nfc_south <- subset(
  sdv_example_standings, division == "NFC South",
  c(team_name, wins:points_against)
)
gt(nfc_south) |> gt_theme_gtutils()
gt(nfc_south) |> gt_theme_gtutils(density = "compact")
```

gt_theme_kenpom

KenPom theme for gt tables

Description

Modeled on KenPom's tables. Helvetica Neue type, blue zebra-striped rows, blue column labels and row-group bands set over a light-blue fill, and underlined spanners. The heading is centered.

Usage

```
gt_theme_kenpom(
  gt_object,
  density = c("comfortable", "compact", "social"),
  ...
)
```

Arguments

<code>gt_object</code>	A gt table object to modify.
<code>density</code>	Character. The type and padding scale. One of "comfortable", "compact", or "social". See Density. Defaults to "comfortable".
<code>...</code>	Additional arguments passed to <code>gt::tab_options</code> , applied last so they override anything the theme sets.

Details

The striping is applied by row position, a pale blue (#F2FAFD) on odd rows and a slightly deeper blue (#e5ecf9) on even rows, so it follows the order the data is in. A thin black bottom border separates every body row except the last. To force the spanner row to render so it can be underlined, the theme adds a placeholder spanner and then hides it with `display: none` in the `gt::opt_css()` block. A table id is resolved (or generated) up front so that CSS binds to this table alone. `density` rescales the finished table, since this theme sets its sizes directly rather than deriving them from a scale.

Value

Returns a modified gt table with the theme applied.

Density

`density` scales the theme's type and row padding together. "comfortable" leaves every size as the theme sets it, "compact" scales both down, and "social" scales both up, to the scale `gt_save_crop()` and `gt_social_crop()` export at.

Figures

Examples

```
library(gt)
central <- subset(
  sdv_example_standings, division == "Central",
  c(team_name, wins, losses, win_pct:points_against)
)
gt(central) |> gt_theme_kenpom()
gt(central) |> gt_theme_kenpom(density = "compact")
```

gt_theme_midnight	<i>Dark theme for gt tables</i>
-------------------	---------------------------------

Description

A restrained dark theme. The background is a near-black instead of pure black, which avoids the smearing pure black causes on OLED screens. Column labels sit on a slightly raised band, rules are hairlines, and the body has no fills.

Usage

```
gt_theme_midnight(
  gt_object,
  accent = "#5B8DEF",
  density = c("comfortable", "compact", "social"),
  ...
)
```

Arguments

<code>gt_object</code>	A gt table object to modify.
<code>accent</code>	Character. A hex color for the row-group labels and the rule above the table. Defaults to "#5B8DEF".
<code>density</code>	Character. The type and padding scale. One of "comfortable", "compact", or "social". See Density. Defaults to "comfortable".
<code>...</code>	Additional arguments passed to <code>gt::tab_options</code> , applied last so they override anything the theme sets.

Details

Light text on a dark background reads as lighter weight than it is, so the theme opens up the line height to compensate.

Value

Returns a modified gt table with the theme applied.

Color scales on dark grounds

A green-to-red ramp built for white paper goes muddy on a near-black background, as its mid-tones collapse toward the background. Use [pal_midnight](#) with [gt_color_ranks\(\)](#) instead.

Density

`density` sets the type and padding scale together. "comfortable" uses a 14px body with roomy rows, "compact" a 12px body with tight rows, and "social" a 17px body with generous rows and a larger title, at the scale [gt_save_crop\(\)](#) and [gt_social_crop\(\)](#) export at.

Figures**See Also**

[pal_midnight](#) for a color scale that survives a dark ground, and [gt_theme_terminal\(\)](#) for a denser dark look.

Examples

```
library(gt)
nfc_west <- subset(
  sdv_example_standings, division == "NFC West",
  c(team_name, wins:points_against)
)
gt(nfc_west) |> gt_theme_midnight()

# color scales need lifting on a dark ground; see pal_midnight
west <- subset(
  sdv_example_standings, league == "nba" & conference == "Western",
  c(team_name, wins, losses, win_pct)
)
gt(west) |>
  gt_theme_midnight() |>
  gt_color_ranks(win_pct, palette = pal_midnight)
```

gt_theme_ncaa

*NCAA theme for gt tables***Description**

Open Sans throughout, with uppercase white column labels on a solid black band and zebra-striped rows, after the NCAA. Row groups render as white labels on a dark gray fill, column spanners are underlined, and every column is left aligned with a 25px left indent on each row.

Usage

```
gt_theme_ncaa(gt_object, density = c("comfortable", "compact", "social"), ...)
```

Arguments

gt_object	A gt table object to modify.
density	Character. The type and padding scale. One of "comfortable", "compact", or "social". See Density. Defaults to "comfortable".
...	Additional arguments passed to <code>gt::tab_options</code> , applied last so they override anything the theme sets.

Details

Column labels sit in a solid black band in white uppercase Open Sans, while source notes and footnotes switch to Almarai. Rows are zebra-striped through `gt::opt_row_striping()`, horizontal rules are hidden, and each row and heading carries a 25px left indent. The last body row's bottom border is painted white so it does not double the rule that closes the table.

Value

Returns a modified gt table with the theme applied.

Density

density scales the theme’s type and row padding together. "comfortable" leaves every size as the theme sets it, "compact" scales both down, and "social" scales both up, to the scale `gt_save_crop()` and `gt_social_crop()` export at.

Figures

Examples

```
library(gt)
atlantic <- subset(
  sdv_example_standings, division == "Atlantic",
  c(team_name, wins, losses, win_pct:points_against)
)
gt(atlantic) |> gt_theme_ncaa()
```

gt_theme_pl	<i>Premier League theme for gt tables</i>
-------------	---

Description

DM Sans throughout in the Premier League’s deep purple (#37003c), with muted-purple column labels and a purple rule bracketing the body. Row groups render as white labels on a pale lilac fill, and column spanners are bold and underlined.

Usage

```
gt_theme_pl(gt_object, density = c("comfortable", "compact", "social"), ...)
```

Arguments

- gt_object A gt table object to modify.
- density Character. The type and padding scale. One of "comfortable", "compact", or "social". See Density. Defaults to "comfortable".
- ... Additional arguments passed to `gt::tab_options`, applied last so they override anything the theme sets.

Details

A purple rule closes the column labels and another opens the body, and each body row is separated by a purple bottom border. The last body row’s bottom border is painted white so it does not double the rule that closes the table.

Value

Returns a modified gt table with the theme applied.

Density

density scales the theme's type and row padding together. "comfortable" leaves every size as the theme sets it, "compact" scales both down, and "social" scales both up, to the scale `gt_save_crop()` and `gt_social_crop()` export at.

Figures**Examples**

```
library(gt)
southeast <- subset(
  sdv_example_standings, division == "Southeast",
  c(team_name, wins, losses, win_pct:points_against)
)
gt(southeast) |> gt_theme_pl()
```

gt_theme_preview	<i>Preview data in every theme</i>
------------------	------------------------------------

Description

Renders the same few rows through each `gt_theme_*` function in the package and lays the results out in a grid, one panel per theme, labeled with its name.

Usage

```
gt_theme_preview(
  data,
  themes = NULL,
  n = 5,
  ncol = 3,
  density = "compact",
  file = NULL,
  ...
)
```

Arguments

data	A data frame. A gt table is also accepted, in which case its underlying data is used.
themes	Character. The theme functions to show, by name. If NULL, every gt_theme_* in the package is used. Defaults to NULL.
n	Integer. How many rows of data to show in each panel. Defaults to 5.
ncol	Integer. The number of panels across. Defaults to 3.
density	Character. A density passed to every theme, so the panels are comparable. If NULL, each theme uses its own default. Defaults to "compact".
file	Optional. A path to write a PNG to. If NULL, the grid is returned for the viewer. Defaults to NULL.
...	Further arguments passed to gt_grid() , such as gap or bg.

Details

Each panel is captioned with the theme's name in a neutral style, set outside the table. Using each theme's own heading instead would let a wide display title stretch its panel out of shape.

Themes that take extra arguments, such as style on [gt_theme_sofa\(\)](#), are shown at their defaults.

Value

What [gt_grid\(\)](#) returns: an `htmltools::browsable()` tag list of the previews, which prints in the viewer; with file, invisibly file after writing the PNG to it.

See Also

[gt_grid\(\)](#), which does the layout.

Examples

```
nfl <- subset(
  sdv_example_standings, league == "nfl",
  c(team_name, wins, losses, win_pct)
)
gt_theme_preview(
  nfl,
  themes = c("gt_theme_sdv", "gt_theme_kenpom", "gt_theme_athletic")
)

# a subset, saved as one wide image; saving needs a headless Chrome (webshot2)

nba <- subset(
  sdv_example_standings, league == "nba",
  c(team_name, wins, losses, win_pct)
)
gt_theme_preview(
  nba,
```

```

    themes = c("gt_theme_broadsheet", "gt_theme_swiss", "gt_theme_midnight"),
    ncol = 3, file = tempfile(fileext = ".png")
  )

```

gt_theme_savant

Baseball Savant theme for gt tables

Description

Roboto Condensed throughout, with bold column labels and zebra-striped rows under a centered heading, after Baseball Savant. Row groups render as off-white labels on a black fill, and column spanners are bold and underlined.

Usage

```

gt_theme_savant(
  gt_object,
  density = c("comfortable", "compact", "social"),
  ...
)

```

Arguments

gt_object	A gt table object to modify.
density	Character. The type and padding scale. One of "comfortable", "compact", or "social". See Density. Defaults to "comfortable".
...	Additional arguments passed to <code>gt::tab_options</code> , applied last so they override anything the theme sets.

Details

The heading is centered rather than left aligned. Rows are zebra-striped through `gt::opt_row_stripping()` and horizontal rules are hidden. The last body row's bottom border is painted white so it does not double the rule that closes the table.

Value

Returns a modified gt table with the theme applied.

Density

density scales the theme's type and row padding together. "comfortable" leaves every size as the theme sets it, "compact" scales both down, and "social" scales both up, to the scale `gt_save_crop()` and `gt_social_crop()` export at.

Figures

Examples

```
library(gt)
afc_east <- subset(
  sdv_example_standings, division == "AFC East",
  c(team_name, wins:points_against)
)
gt(afc_east) |> gt_theme_savant()
```

gt_theme_scoreboard	<i>Broadcast-graphics theme for gt tables</i>
---------------------	---

Description

A solid color band carrying condensed uppercase labels, over tight rows, after a television lower-third or a stadium scoreboard.

Usage

```
gt_theme_scoreboard(
  gt_object,
  accent = "#0E1621",
  density = c("compact", "comfortable", "social"),
  ...
)
```

Arguments

gt_object	A gt table object to modify.
accent	Character. A hex color for the header band. The label color adapts to it. Defaults to "#0E1621".
density	Character. The type and padding scale. One of "comfortable", "compact", or "social". See Density. Defaults to "compact".
...	Additional arguments passed to <code>gt::tab_options</code> , applied last so they override anything the theme sets.

Details

accent sets the band itself, so one argument restyles the whole table to a brand color. The label color is checked against the band and flipped to dark if the accent is pale.

Value

Returns a modified gt table with the theme applied.

Density

density sets the type and padding scale together. "comfortable" uses a 14px body with roomy rows, "compact" a 12px body with tight rows, and "social" a 17px body with generous rows and a larger title, at the scale `gt_save_crop()` and `gt_social_crop()` export at.

Figures**See Also**

`gt_spotlight()` for picking out a row, and `gt_fmt_rank()` for ordinals.

Examples

```
library(gt)
northwest <- subset(
  sdv_example_standings, division == "Northwest",
  c(team_name, wins, losses, win_pct:points_against)
)
gt(northwest) |> gt_theme_scoreboard()

# a brand color carries the whole table
gt(northwest) |> gt_theme_scoreboard(accent = "#0F766E")
```

gt_theme_sdv

SportsDataverse theme for gt tables

Description

The SportsDataverse house table: Chivo titles and column labels, a Lato body set in tabular figures, faint row dividers, and the SDV gradient (#3346F0 to #7FE6DC) drawn as a single horizon line under the column labels. `style = "dark"` sets the same table on the SportsDataverse navy, matching the dark mode of the package websites.

Usage

```
gt_theme_sdv(
  gt_object,
  style = c("light", "dark"),
  density = c("comfortable", "compact", "social"),
  ...
)
```

Arguments

gt_object	A gt table object to modify.
style	Character. "light" for a white table, or "dark" for the SportsDataverse navy. Defaults to "light".
density	Character. The type and padding scale. One of "comfortable", "compact", or "social". See Density. Defaults to "comfortable".
...	Additional arguments passed to <code>gt::tab_options</code> , applied last so they override anything the theme sets.

Details

The gradient line is the theme's one accent; everything else stays quiet so logos from `gt_sdv_logos()` and team colors carry the table. Titles and the heading are left-aligned. Column labels keep the case you give them rather than being forced to capitals. For a table dressed in one team's colors, see `gt_theme_sdv_team()`.

The line under the column labels is a CSS `::after` element. `gt`'s CSS inliner removes those, so `gt::as_raw_html(inline_css = TRUE)` output (as used for email) shows the table without it; knitted documents, websites and images saved with `gt_save_crop()` keep it.

Value

Returns a modified gt table with the theme applied.

Density

density scales the theme's type and row padding together. "comfortable" leaves every size as the theme sets it, "compact" scales both down, and "social" scales both up, to the scale `gt_save_crop()` and `gt_social_crop()` export at.

Figures**See Also**

`gt_theme_sdv_team()`, `theme_bg` for the background to pad a saved image with.

Examples

```
library(gt)
standings <- data.frame(
  team = c("KC", "LAC", "DEN", "LV"),
  w = c(15, 11, 10, 4), l = c(2, 6, 7, 13)
)
gt(standings) |>
  gt_sdv_logos(columns = "team", league = "nfl") |>
  tab_header("AFC West standings", "Through week 18") |>
  gt_theme_sdv()
```

```
gt(standings) |> gt_theme_sdv(style = "dark", density = "social")
```

gt_theme_sdv_team	<i>Team-colored SportsDataverse theme for gt tables</i>
-------------------	---

Description

`gt_theme_sdv()` dressed in one team's colors: the title block is filled with the team's primary color, the horizon line under the column labels takes the secondary color, and the column labels are set in the primary color. The colors come from `sdv_team_colors()`, so any abbreviation, alias or name `sdv_resolve()` resolves works here.

Usage

```
gt_theme_sdv_team(
  gt_object,
  team = NULL,
  league,
  density = c("comfortable", "compact", "social"),
  season = NULL,
  id_system = "auto",
  ...
)
```

Arguments

gt_object	A gt table object to modify.
team	Character. One team, as an abbreviation, alias or name for league. Defaults to NULL, which uses the SportsDataverse colors.
league	A league key from <code>sdv_leagues()</code> : the league team belongs to. Needed with a team.
density	Character. The type and padding scale. One of "comfortable", "compact", or "social". See Density. Defaults to "comfortable".
season	NULL for today's mark, or a season (or one per team): the mark the team wore that season where the archive dates its marks.
id_system	"auto" (try every id system in priority order) or one id system: "team_id", "espn", "espn_abbr", "nhl", "nflverse", "mlbstats", "nba_api", "hockeytech", "ncaa", "pff", "cricinfo", "cfbd", "bref", "sportsipy", "fangraphs", "sdvplotr", "name", or "nhl_id" (NHL stats API ids, which overlap ESPN's, so "auto" never reads a number as one).
...	Additional arguments passed to <code>gt::tab_options</code> , applied last so they override anything the theme sets.

Details

Text on the filled title block is black or white, whichever contrasts more with the team’s primary color, and the subtitle is blended toward it until it still clears a 4.5:1 contrast ratio. When the secondary color would vanish against the white table (a white or pale secondary), the horizon line uses the primary color instead, and column labels fall back to the SportsDataverse navy when the primary color is too light to read on white. With team = NULL the table wears the SportsDataverse navy and cyan.

Value

Returns a modified gt table with the theme applied.

Figures

Density

density scales the theme’s type and row padding together. "comfortable" leaves every size as the theme sets it, "compact" scales both down, and "social" scales both up, to the scale `gt_save_crop()` and `gt_social_crop()` export at.

See Also

`gt_theme_sdv()`, `sdv_team_colors()`.

Examples

```
library(gt)
leaders <- data.frame(
  player = c("Patrick Mahomes", "Travis Kelce", "Isiah Pacheco"),
  yards = c(4183, 984, 935)
)
gt(leaders) |>
  tab_header("Chiefs yardage leaders", "2023 regular season") |>
  gt_theme_sdv_team(team = "KC", league = "nfl")
```

gt_theme_sofa	<i>SofaScore theme for gt tables</i>
---------------	--------------------------------------

Description

Sofia Sans Condensed throughout on a warm cream ("light") or dark navy ("dark") ground, after SofaScore, with bold column labels and bold row-group labels. Column spanners are bold and underlined. Horizontal rules are hidden, so the ground alone separates the rows.

Usage

```
gt_theme_sofa(
  gt_object,
  style = "light",
  density = c("comfortable", "compact", "social"),
  ...
)
```

Arguments

gt_object	A gt table object to modify.
style	Character. The color scheme, "light" for a warm cream ground or "dark" for a dark navy ground. Defaults to "light".
density	Character. The type and padding scale. One of "comfortable", "compact", or "social". See Density. Defaults to "comfortable".
...	Additional arguments passed to <code>gt::tab_options</code> , applied last so they override anything the theme sets.

Details

The table background, its outer borders, and the last row's bottom border are all painted in the chosen ground color, so the rows read as separated by space rather than by rules. Row groups are closed with a black bottom border.

Value

Returns a modified gt table with the theme applied.

Density

density scales the theme's type and row padding together. "comfortable" leaves every size as the theme sets it, "compact" scales both down, and "social" scales both up, to the scale `gt_save_crop()` and `gt_social_crop()` export at.

Figures**Examples**

```
library(gt)
nfc_north <- subset(
  sdv_example_standings, division == "NFC North",
  c(team_name, wins:points_against)
)
gt(nfc_north) |> gt_theme_sofa()
gt(nfc_north) |> gt_theme_sofa(style = "dark")
```

gt_theme_swiss

International Typographic Style theme for gt tables

Description

Builds the table on whitespace. There are no fills, no banding, and exactly two rules in the whole table, one under the column labels and one closing the body. Everything else is done with space and alignment.

Usage

```
gt_theme_swiss(
  gt_object,
  accent = "#111111",
  density = c("comfortable", "social", "compact"),
  ...
)
```

Arguments

gt_object	A gt table object to modify.
accent	Character. A hex color, used once, on the rule beneath the column labels. Defaults to "#111111", which reads as no color at all.
density	Character. The type and padding scale. One of "comfortable", "compact", or "social". See Density. Defaults to "comfortable".
...	Additional arguments passed to <code>gt::tab_options</code> , applied last so they override anything the theme sets.

Details

Alignment is strict. Text sits flush left, numbers flush right, and nothing is centered.

Value

Returns a modified gt table with the theme applied.

Density

density sets the type and padding scale together. "comfortable" uses a 14px body with roomy rows, "compact" a 12px body with tight rows, and "social" a 17px body with generous rows and a larger title, at the scale `gt_save_crop()` and `gt_social_crop()` export at.

Figures

Examples

```
library(gt)
afc_west <- subset(
  sdv_example_standings, division == "AFC West",
  c(team_name, wins:points_against)
)
gt(afc_west) |> gt_theme_swiss()

pacific <- subset(
  sdv_example_standings, division == "Pacific",
  c(team_name, wins, losses, win_pct:points_against)
)
gt(pacific) |> gt_theme_swiss(accent = "#D33A2C")
```

gt_theme_terminal	<i>Data-terminal theme for gt tables</i>
-------------------	--

Description

A dense monospace theme in the register of a trading terminal. Dark background, amber labels, a rule on every row, and tight padding.

Usage

```
gt_theme_terminal(
  gt_object,
  accent = "#FFB86C",
  density = c("compact", "comfortable", "social"),
  ...
)
```

Arguments

gt_object	A gt table object to modify.
accent	Character. A hex color for the column labels, row groups, and the top rule. Defaults to amber "#FFB86C"; "#7EE787" gives a green-phosphor variant.
density	Character. The type and padding scale. One of "comfortable", "compact", or "social". See Density. Defaults to "compact".
...	Additional arguments passed to gt::tab_options, applied last so they override anything the theme sets.

Details

The monospace face has fixed-width digits already, so numeric columns line up without any extra formatting.

Value

Returns a modified gt table with the theme applied.

Density

density sets the type and padding scale together. "comfortable" uses a 14px body with roomy rows, "compact" a 12px body with tight rows, and "social" a 17px body with generous rows and a larger title, at the scale `gt_save_crop()` and `gt_social_crop()` export at.

Figures**See Also**

[pal_midnight](#) for a color scale that survives a dark ground.

Examples

```
library(gt)
afc <- subset(
  sdv_example_standings, conference == "AFC",
  c(team_name, wins:points_against)
)
gt(head(afc, 12)) |> gt_theme_terminal()

east <- subset(
  sdv_example_standings, league == "nba" & conference == "Eastern",
  c(team_name, wins, losses, win_pct:points_against)
)
gt(east) |> gt_theme_terminal(accent = "#7EE787")
```

gt_theme_tier

Tier list theme for gt tables

Description

Oswald throughout on a near-black ("dark") or white ("light") ground, with a bold title and every column center aligned. Rows are separated by thin black borders and horizontal rules are otherwise hidden.

Usage

```
gt_theme_tier(
  gt_object,
  style = "dark",
  density = c("comfortable", "compact", "social"),
  ...
)
```

Arguments

gt_object	A gt table object to modify.
style	Character. The color scheme, "dark" for a near-black ground or "light" for a white ground. Defaults to "dark".
density	Character. The type and padding scale. One of "comfortable", "compact", or "social". See Density. Defaults to "comfortable".
...	Additional arguments passed to <code>gt::tab_options</code> , applied last so they override anything the theme sets.

Details

Every body row except the last carries a black bottom border, and the last row's border is painted in the ground color so it does not double the edge of the table. Pairs with `gt_tiers()`, which builds the tier rows themselves.

Value

Returns a modified gt table with the theme applied.

Density

density scales the theme's type and row padding together. "comfortable" leaves every size as the theme sets it, "compact" scales both down, and "social" scales both up, to the scale `gt_save_crop()` and `gt_social_crop()` export at.

Figures**Examples**

```
library(gt)
afc <- subset(
  sdv_example_standings, conference == "AFC" & conference_rank <= 7,
  c(team_name, wins:points_against)
)
gt(afc) |> gt_theme_tier()
gt(afc) |> gt_theme_tier(style = "light")
```

gt_theme_tufte	<i>Minimal-ink theme for gt tables</i>
----------------	--

Description

A spare, Tufte-inspired look. An old-style serif on a warm white ground, italic column labels, and a single hairline under them. There are no vertical rules, no rules between rows, and no fills. A faint hairline closes the body.

Usage

```
gt_theme_tufte(  
  gt_object,  
  accent = "#111111",  
  density = c("comfortable", "compact", "social"),  
  ...  
)
```

Arguments

gt_object	A gt table object to modify.
accent	Character. A hex color for the header hairline and the row-group labels. Defaults to "#111111", near-black. A muted red or rust gives the Tufte marginal accent.
density	Character. The type and padding scale. One of "comfortable", "compact", or "social". See Density. Defaults to "comfortable".
...	Additional arguments passed to <code>gt::tab_options</code> , applied last so they override anything the theme sets.

Details

accent recolors the header hairline and the row-group labels.

Value

Returns a modified gt table with the theme applied.

Density

density sets the type and padding scale together. "comfortable" uses a 14px body with roomy rows, "compact" a 12px body with tight rows, and "social" a 17px body with generous rows and a larger title, at the scale [gt_save_crop\(\)](#) and [gt_social_crop\(\)](#) export at.

Figures

See Also

[gt_theme_booktabs\(\)](#) for a firmer, academic relative.

Examples

```
library(gt)

nfc_north <- subset(
  sdv_example_standings, division == "NFC North",
  c(team_name, wins:points_against)
)
gt(nfc_north) |> gt_theme_tufte()

# a muted rust accent on the hairline
atlantic <- subset(
  sdv_example_standings, division == "Atlantic",
  c(team_name, wins, losses, win_pct:points_against)
)
gt(atlantic) |> gt_theme_tufte(accent = "#7B3F2B")
```

gt_tiers

Color and bold the tier column of a gt table

Description

Fills the cells of a tier column by tier level and bolds them, pairing each level with a color. The remaining columns are rendered as images, since the intended layout is a tier list with logos or headshots ranked into colored bands.

Usage

```
gt_tiers(
  gt_object,
  levels,
  colors = NULL,
  style = "dark",
  img_height = "55px",
  tier_column = "tier",
  image_columns = NULL,
  alt = NULL
)
```

Arguments

gt_object A gt table object to modify.

levels	Character. The tier levels, matching the values in the table's tier column. May instead be a single named vector of level = color, in which case colors is left unset. That is the same shape <code>gt_legend_discrete()</code> takes, so one object can drive both.
colors	Character. Hex color codes paired with levels, in the same order. Must be the same length as levels. Defaults to NULL, which requires levels to carry the colors as names.
style	Character. The color scheme, passed to <code>gt_theme_tier()</code> . Either "dark" for a near-black ground or "light" for a white one. Defaults to "dark".
img_height	Character. The height of the images rendered in the image columns, as a CSS size. Defaults to "55px".
tier_column	Character. The name of the column holding the tier levels. Defaults to "tier".
image_columns	Optional. The columns to render as images. When NULL, every column other than tier_column is rendered as images. Defaults to NULL.
alt	Optional. A function that takes the image paths or URLs and returns their alt text, one string per image, such as <code>function(url) names_by_url[url]</code> . Defaults to NULL, which names a team logo or wordmark the package knows (any mark in the logo archive, and the season logos season draws) by the team, and any other image by its file name without the extension.

Details

The theme is applied once with `gt_theme_tier()`, the image columns are passed through `gt::fmt_image()` at `img_height`, and all column labels are cleared, so the input for those columns must be image paths or URLs. Each image gets its own alt text from alt, so a screen reader can tell the entries apart. The function then reduces over levels, and for each level fills the matching tier_column cells with the paired color and sets their text to black or white, whichever measures higher contrast against that fill, so each band keeps a legible label. Naming image_columns leaves the rest untouched, so a text column can sit alongside the images.

Value

Returns a modified gt table with the tier column colored and bolded.

Examples

```
library(gt)

# the AFC's seven playoff seeds: the top seed's bye, then the wild-card round
afc <- subset(sdv_example_standings, conference == "AFC" & conference_rank <= 7)
afc <- afc[order(afc$conference_rank), ]
seeds <- data.frame(
  tier = ifelse(afc$conference_rank == 1, "Bye", "Wild card"),
  logo = sdv_logo_url(afc$team, "nfl")
)

# the logos are URLs, which the browser fetches when the table is shown
gt(seeds) |>
```

```
gt_tiers(levels = c("Bye", "Wild card"), colors = c("#1B7837", "#B2182B"))
```

gt_title_header

Build a styled header block for a gt table

Description

Wraps `gt::tab_header()` to produce a fuller header: an optional kicker above the title, the title, an optional subtitle, and an optional date. Each of the four is styled independently through its own list, so the typography can be set without writing the `tab_style()` calls yourself.

Usage

```
gt_title_header(
  gt_object,
  title,
  subtitle = NULL,
  kicker = NULL,
  date = NULL,
  kicker_style = list(),
  title_style = list(),
  subtitle_style = list(),
  date_style = list()
)
```

Arguments

<code>gt_object</code>	A gt table object to modify.
<code>title</code>	Character. The title text.
<code>subtitle</code>	Optional. The subtitle text. Defaults to NULL.
<code>kicker</code>	Optional. A short line above the title. Defaults to NULL.
<code>date</code>	Optional. A date shown beneath the subtitle. A Date is formatted as, for example, "July 21, 2026"; anything else is shown as given. Defaults to NULL.
<code>kicker_style</code>	A named list styling the kicker. See Styling. Defaults to an empty list.
<code>title_style</code>	A named list styling the title. Defaults to an empty list.
<code>subtitle_style</code>	A named list styling the subtitle. Defaults to an empty list.
<code>date_style</code>	A named list styling the date. Defaults to an empty list.

Details

Google fonts named in the style lists are loaded through gt's own machinery, so they survive `gtsave()` as well as the HTML output.

`gt_title_header()` calls `gt::tab_header()`, which replaces the whole header. Paired with [gt_legend_continuous\(\)](#) at `location = "top"`, call this one first or it will overwrite the legend.

Value

Returns a modified gt table with the header block applied.

Styling

Each *_style argument takes a named list. Recognized keys are font (a Google font name), size, color, weight, italic, spacing (letter spacing), transform (such as "uppercase"), and align, plus line_height, margin_top, margin_bottom, padding_top, and padding_bottom for nudging the blocks closer together or further apart. Any key you leave out keeps its default. Lengths take a number, read as pixels, or a CSS string such as "2rem". This is the same convention used by [gt_grid\(\)](#), [gt_legend_continuous\(\)](#), and [gt_legend_discrete\(\)](#).

See Also

[gt_538_caption\(\)](#) for the footer equivalent.

Examples

```
library(gt)

nfc_west <- subset(
  sdv_example_standings, division == "NFC West",
  c(team_name, wins:points_against)
)

gt(nfc_west) |>
  gt_title_header(
    title = "The NFC West",
    subtitle = "Final standings of the 2025 regular season",
    kicker = "NFL",
    date = as.Date("2026-07-21"),
    kicker_style = list(
      color = "#0054AD", size = "0.8em",
      transform = "uppercase", spacing = "0.1em"
    ),
    title_style = list(font = "Libre Franklin", weight = 800, size = "26px"),
    subtitle_style = list(color = "#666666", italic = TRUE),
    date_style = list(color = "#B8232F", weight = 600)
  )
```

gt_watermark

Put a watermark behind a gt table

Description

Sets a wordmark or a logo behind the table body, faint enough to read through.

Usage

```
gt_watermark(
  gt_object,
  text = NULL,
  image = NULL,
  opacity = 0.06,
  size = "60%",
  position = "center",
  color = "#000000",
  angle = 0,
  font = "Helvetica, Arial, sans-serif"
)
```

Arguments

gt_object	A gt table object to modify.
text	Optional. Text to render as the watermark. Defaults to NULL.
image	Optional. A path to a PNG, JPEG, or SVG file to use instead of text. Defaults to NULL.
opacity	Numeric. How faint the watermark is, from 0 to 1. Defaults to 0.06.
size	Character. The width of the watermark relative to the table, as a CSS background size such as "60%". Defaults to "60%".
position	Character. Where it sits, as a CSS background position such as "center" or "right bottom". Defaults to "center".
color	Character. The text color, used only with text. Defaults to "#000000".
angle	Numeric. Rotation in degrees, used only with text. Defaults to 0.
font	Character. The font family for text. Defaults to a system sans-serif.

Details

A text watermark is drawn as an inline SVG rather than as HTML, because a background image is the only way to sit behind the table body without taking up a row. An SVG background cannot load the page's webfonts, so font has to name a face installed on the machine rendering the table.

The watermark sits behind the body cells, so it will be hidden wherever a cell has its own fill, from row striping or from `gt::data_color()`.

Value

Returns a modified gt table with a watermark behind the body.

See Also

[gt_social_tag\(\)](#) for visible attribution in the source note.

Examples

```
library(gt)

afc_south <- subset(
  sdv_example_standings, division == "AFC South",
  c(team_name, wins:points_against)
)

gt(afc_south) |> gt_watermark(text = "DRAFT", angle = -30, opacity = 0.08)

# any local png, jpeg, gif or svg; a small svg written on the spot here
logo <- tempfile(fileext = ".svg")
writeLines(paste0(
  '<svg xmlns="http://www.w3.org/2000/svg" width="40" height="40">',
  '<circle cx="20" cy="20" r="18"/></svg>'
), logo)
gt(afc_south) |>
  gt_watermark(image = logo, size = "40%", position = "right bottom")
```

 gt_wrap_labels

Break long column labels across lines in a gt table

Description

Wraps a column label onto several short lines so a wide heading stops forcing a narrow column wider than its data needs. "Strength of Schedule" over a two-digit column becomes three stacked lines instead of one long one.

Usage

```
gt_wrap_labels(
  gt_object,
  columns = gt::everything(),
  width = 12,
  balance = TRUE
)
```

Arguments

gt_object	A gt table object to modify.
columns	The columns whose labels should wrap. Defaults to <code>gt::everything()</code> .
width	Integer. The target line length in characters. Defaults to 12.
balance	Logical. Should the lines be evened out rather than filled greedily left to right? Defaults to TRUE.

Details

The hand-rolled version is `gt::cols_label(x = gt::html("A<br>B"))`, written out per column. This wraps at a width for you, and by default balances the lines so they come out close to even rather than a long first line over a short last one.

A label of one word, or one already shorter than width, is left alone. The wrap is on whitespace only, so a single long word is never split.

Value

Returns a modified `gt` table with the selected labels wrapped.

Examples

```
library(gt)

afc_east <- subset(
  sdv_example_standings, division == "AFC East",
  c(team_name, win_pct, conference_rank)
)

gt(afc_east) |>
  cols_label(
    win_pct = "Winning Percentage",
    conference_rank = "Final Conference Rank"
  ) |>
  gt_wrap_labels(c(win_pct, conference_rank), width = 10)
```

pal_midnight

A rank palette for dark backgrounds

Description

A five-color green-to-red ramp with its luminance range lifted so it still reads on a near-black background. The default palette in `gt_color_ranks()` is built for white paper, and its mid-tones collapse into a dark ground. Pass this instead when using `gt_theme_midnight()` or `gt_theme_terminal()`.

Usage

```
pal_midnight
```

Format

A character vector of five hex colors, running best to worst.

See Also

`gt_theme_midnight()`.

Examples

```
west <- subset(
  sdv_example_standings, league == "nba" & conference == "Western",
  c(team_name, wins, losses, win_pct)
)
gt::gt(west) |>
  gt_theme_midnight() |>
  gt_color_ranks(win_pct, palette = pal_midnight)
```

reactable_sdv_cols_label

Replace 'reactable' Column Headers with Team Logos

Description

Builds a named list of `reactable::colDef()` objects whose headers are team logos, for data frames whose column names are team abbreviations. Columns that cannot be resolved are left out of the list so reactable falls back to the plain column name.

Usage

```
reactable_sdv_cols_label(
  .data,
  ...,
  league,
  variant = "default",
  height = 30,
  season = NULL,
  id_system = "auto",
  strict = FALSE
)
```

Arguments

<code>.data</code>	A data frame whose column names are team abbreviations.
<code>...</code>	Additional arguments passed to every <code>reactable::colDef()</code> .
<code>league</code>	A league key from <code>sdv_leagues()</code> .
<code>variant</code>	Character. "default", "dark", or a named variant from <code>sdv_marks()</code> ("scoreboard", "grayscale", ...); see <code>sdv_logo_url()</code> . A team without it gets its closest mark, and in the browser the image falls back to the default mark when the variant's file fails to load.
<code>height</code>	Numeric. Image height in pixels.
<code>season</code>	NULL for today's mark, or a season (or one per team): the mark the team wore that season where the archive dates its marks.

<code>id_system</code>	"auto" (try every id system in priority order) or one id system: "team_id", "espn", "espn_abbr", "nhl", "nflverse", "mlbstats", "nba_api", "hockeytech", "ncaa", "pff", "cricinfo", "cfbd", "bref", "sportsipy", "fangraphs", "sdvplotr", "name", or "nhl_id" (NHL stats API ids, which overlap ESPN's, so "auto" never reads a number as one).
<code>strict</code>	If TRUE, a value that does not resolve is an error; otherwise it becomes NA with one warning per call.

Value

A named list of colDef objects for the columns argument of `reactable::reactable()`.

Examples

```
library(reactable)
library(sdvplotR)

df <- data.frame(KC = 1:3, BUF = 4:6, SF = 7:9)
reactable(df, columns = reactable_sdv_cols_label(df, league = "nfl"))
```

`reactable_sdv_images` *Render Team Logos, Wordmarks and Headshots in 'reactable' Tables*

Description

Cell renderers for `reactable::colDef()` that translate team abbreviations (or player IDs) into `<img>` tags. Values that cannot be resolved are returned unchanged so the original text is shown. Logos and wordmarks get the team's full name as alt text, headshots "Player <id> headshot".

Usage

```
reactable_sdv_logos(
  league,
  variant = "default",
  height = 30,
  default_img = NULL,
  season = NULL,
  id_system = "auto",
  strict = FALSE
)

reactable_sdv_wordmarks(
  league,
  variant = "default",
  height = 30,
  default_img = NULL,
  season = NULL,
```

```

    id_system = "auto",
    strict = FALSE
  )

  reactable_sdv_headshots(
    league,
    height = 40,
    default_img = NULL,
    id_system = "espn"
  )

```

Arguments

league	A league key from sdv_leagues() .
variant	Character. "default", "dark", or a named variant from sdv_marks() ("scoreboard", "grayscale", ...); see sdv_logo_url() . A team without it gets its closest mark, and in the browser the image falls back to the default mark when the variant's file fails to load.
height	Numeric. Image height in pixels.
default_img	Character. Fallback image URL used when the value cannot be resolved. If NULL (the default) the raw value is shown instead.
season	NULL for today's mark, or a season (or one per team): the mark the team wore that season where the archive dates its marks.
id_system	"auto" (try every id system in priority order) or one id system: "team_id", "espn", "espn_abbr", "nhl", "nflverse", "mlbstats", "nba_api", "hockeytech", "ncaa", "pff", "cricinfo", "cfbd", "bref", "sportsipy", "fangraphs", "sdvplotr", "name", or "nhl_id" (NHL stats API ids, which overlap ESPN's, so "auto" never reads a number as one).
strict	If TRUE, a value that does not resolve is an error; otherwise it becomes NA with one warning per call.

Value

A function with signature `function(value, index)` suitable for the cell argument of [reactable::colDef\(\)](#).

See Also

[reactable_sdv_cols_label\(\)](#), [reactable_sdv_team_color_bar\(\)](#), [gt_sdv_logos\(\)](#)

Examples

```

library(reactable)
library(sdvplotR)

df <- data.frame(team = c("KC", "BUF", "SF"), wins = c(13, 12, 11))

reactable(
  df,

```

```

    columns = list(
      team = colDef(cell = reactable_sdv_logos(league = "nfl"), html = TRUE)
    )
  )

```

reactable_sdv_team_color

Team-Colored Cell Styles for 'reactable' Tables

Description

Style functions for `reactable::colDef()` that color cells with a team's primary or secondary color: a horizontal bar whose length is proportional to the cell value, or a translucent background fill.

Usage

```

reactable_sdv_team_color_bar(
  data,
  team_col,
  league,
  which = c("primary", "secondary"),
  max_value = NULL,
  na_color = "grey70",
  season = NULL,
  id_system = "auto",
  strict = FALSE
)

```

```

reactable_sdv_team_color_bg(
  data,
  team_col,
  league,
  which = c("primary", "secondary"),
  alpha = 0.15,
  na_color = "grey70",
  season = NULL,
  id_system = "auto",
  strict = FALSE
)

```

Arguments

data	The data frame passed to <code>reactable::reactable()</code> . Needed to look up the team of each row.
team_col	Character. Name of the column in data holding team abbreviations.
league	A league key from <code>sdv_leagues()</code> .

which	Character. "primary" or "secondary" team color.
max_value	Numeric. Reference value that fills the whole cell. If NULL (the default), the maximum of the styled column is used.
na_color	Color used when a team cannot be resolved.
season	NULL for today's mark, or a season (or one per team): the mark the team wore that season where the archive dates its marks.
id_system	"auto" (try every id system in priority order) or one id system: "team_id", "espn", "espn_abbr", "nhl", "nflverse", "mlbstats", "nba_api", "hockeytech", "ncaa", "pff", "cricinfo", "cfbd", "bref", "sportsipy", "fangraphs", "sdvplotr", "name", or "nhl_id" (NHL stats API ids, which overlap ESPN's, so "auto" never reads a number as one).
strict	If TRUE, a value that does not resolve is an error; otherwise it becomes NA with one warning per call.
alpha	Numeric. Opacity of the background fill in $[0, 1]$.

Value

A function with signature `function(value, index, name)` suitable for the style argument of `reactable::colDef()`.

Examples

```
library(reactable)
library(sdvplotR)

df <- data.frame(team = c("KC", "BUF", "SF"), wins = c(13, 12, 11))

reactable(
  df,
  columns = list(
    team = colDef(style = reactable_sdv_team_color_bg(df, "team", league = "nfl")),
    wins = colDef(style = reactable_sdv_team_color_bar(df, "team", league = "nfl"))
  )
)
```

scale_axes_sdv

Axis Scales for Sports Team Logos

Description

These scale functions replace axis labels with team logos or player headshots. They work by modifying the axis text theme element to render images.

Usage

```
scale_x_sdv(  
  ...,  
  league,  
  expand = ggplot2::waiver(),  
  guide = ggplot2::waiver(),  
  position = "bottom",  
  size = 12,  
  season = NULL,  
  id_system = "auto"  
)  
  
scale_y_sdv(  
  ...,  
  league,  
  expand = ggplot2::waiver(),  
  guide = ggplot2::waiver(),  
  position = "left",  
  size = 12,  
  season = NULL,  
  id_system = "auto"  
)  
  
scale_x_sdv_headshots(  
  ...,  
  league,  
  expand = ggplot2::waiver(),  
  guide = ggplot2::waiver(),  
  position = "bottom",  
  size = 20,  
  id_system = "espn"  
)  
  
scale_y_sdv_headshots(  
  ...,  
  league,  
  expand = ggplot2::waiver(),  
  guide = ggplot2::waiver(),  
  position = "left",  
  size = 30,  
  id_system = "espn"  
)
```

Arguments

...	Other arguments passed to <code>ggplot2::scale_x_discrete()</code> or <code>ggplot2::scale_y_discrete()</code> .
league	A league key from <code>sdv_leagues()</code> .
expand	Expansion limits for the axis.

guide	Guide for the axis.
position	Position of the axis.
size	The logo size in pixels. It is applied as height for an x-scale and as width for a y-scale.
season	NULL for today's mark, or a season (or one per team): the mark the team wore that season where the archive dates its marks.
id_system	"auto" (try every id system in priority order) or one id system: "team_id", "espn", "espn_abbr", "nhl", "nflverse", "mlbstats", "nba_api", "hockeytech", "ncaa", "pff", "cricinfo", "cfbd", "bref", "sportsipy", "fangraphs", "sdvplotr", "name", or "nhl_id" (NHL stats API ids, which overlap ESPN's, so "auto" never reads a number as one).

Details

The scale translates team names into raw image HTML and places the HTML as axis labels. Because of the way ggplots are constructed, it is necessary to adjust the theme after calling this scale. This can be done by calling `theme_x_sdv()` or `theme_y_sdv()` after any complete theme such as `ggplot2::theme_minimal()`, which replaces every theme element. To set the theme by hand, change the relevant `axis.text` and its position children (`axis.text.x.bottom`, ...) to `ggtext::element_markdown()`: complete themes in 'ggplot2' 4 set those children themselves.

Value

A ggplot2 scale object.

See Also

`theme_x_sdv()`, `theme_y_sdv()`

Examples

```
library(sdvplotR)
library(ggplot2)

team_abbr <- sdv_teams("nfl")$abbr[1:8]

df <- data.frame(
  random_value = runif(length(team_abbr), 0, 1),
  teams = team_abbr
)

if (requireNamespace("ggtext", quietly = TRUE)) {
  ggplot(df, aes(x = teams, y = random_value)) +
    geom_col(aes(fill = teams), width = 0.5) +
    scale_fill_sdv(league = "nfl", alpha = 0.4) +
    scale_x_sdv(league = "nfl") +
    theme_minimal() +
    theme_x_sdv()
}
```

scale_sdv*Scales for Sports Team Colors*

Description

These functions map team names to their team colors in color and fill aesthetics.

Usage

```
scale_color_sdv(  
  league,  
  which = c("primary", "secondary"),  
  values = NULL,  
  ...,  
  aesthetics = "colour",  
  breaks = ggplot2::waiver(),  
  na.value = "grey50",  
  guide = NULL,  
  alpha = NA  
)
```

```
scale_colour_sdv(  
  league,  
  which = c("primary", "secondary"),  
  values = NULL,  
  ...,  
  aesthetics = "colour",  
  breaks = ggplot2::waiver(),  
  na.value = "grey50",  
  guide = NULL,  
  alpha = NA  
)
```

```
scale_fill_sdv(  
  league,  
  which = c("primary", "secondary"),  
  values = NULL,  
  ...,  
  aesthetics = "fill",  
  breaks = ggplot2::waiver(),  
  na.value = "grey50",  
  guide = NULL,  
  alpha = NA  
)
```

Arguments

league A league key from `sdv_leagues()`.

which	One of "primary" or "secondary": which team colour to use.
values	If NULL (the default) use the internal team color vectors. Otherwise a set of aesthetic values to map data values to.
...	Other arguments passed to <code>ggplot2::scale_color_manual()</code> or <code>ggplot2::scale_fill_manual()</code> .
aesthetics	The aesthetic to apply the scale to ("colour" or "fill").
breaks	Breaks for the scale.
na.value	Color for NA values.
guide	Guide function or name.
alpha	Factor to modify color transparency via <code>scales::alpha()</code> . If NA (the default) no transparency will be applied.

Value

A ggplot2 scale object.

Examples

```
library(sdvplotR)
library(ggplot2)

team_abbr <- sdv_teams("nfl")$abbr[1:16]

df <- data.frame(
  random_value = runif(length(team_abbr), 0, 1),
  teams = team_abbr
)

ggplot(df, aes(x = teams, y = random_value)) +
  geom_col(aes(color = teams, fill = teams), width = 0.5) +
  scale_color_sdv(league = "nfl", which = "secondary") +
  scale_fill_sdv(league = "nfl", alpha = 0.4) +
  theme_minimal() +
  theme(axis.text.x = element_text(angle = 45, hjust = 1))
```

sdv_axis_logos	<i>Team logos as plotly axis labels</i>
----------------	---

Description

Replaces a categorical axis's tick labels with team logos.

Usage

```
sdv_axis_logos(
  p,
  teams,
  league,
  axis = c("x", "y"),
  size = 0.08,
  season = NULL,
  variant = "default",
  id_system = "auto",
  strict = FALSE
)
```

Arguments

<code>p</code>	A plotly object.
<code>teams</code>	The axis categories, in the chart's order (the team values the chart plots on that axis).
<code>league</code>	A league key from sdv_leagues() . Required: one abbreviation means different teams in different leagues.
<code>axis</code>	"x" or "y".
<code>size</code>	Image size as a fraction of the plot area on the other axis.
<code>season</code>	NULL for today's mark, or a season (or one per team): the mark the team wore that season where the archive dates its marks.
<code>variant</code>	"default", "dark", or a named variant from sdv_marks() ("scoreboard", "grayscale", "cap_on_light", ...). A team without it gets its closest mark of the same polarity; a name no mark in the archive has is an error listing the league's variants.
<code>id_system</code>	For headshots, as in sdv_headshot_url() .
<code>strict</code>	If TRUE, a value that does not resolve is an error; otherwise it becomes NA with one warning per call.

Value

The plotly object (an htmlwidget).

Examples

```
if (requireNamespace("plotly", quietly = TRUE)) {
  teams <- c("KC", "BUF", "BAL")
  plotly::plot_ly(x = teams, y = c(11, 13, 12), type = "bar") |>
    sdv_axis_logos(teams, "nfl")
}
```

sdv_clear_cache	<i>Clear the sdvplotR caches</i>
-----------------	----------------------------------

Description

sdvplotR keeps the team index, the logo manifest and the NFL headshot map in memory for the session, keeps the index and manifest on disk (in `tools::R_user_dir("sdvplotR", "cache")`) for a day. This clears them, and the image cache of 'ggpath' when 'ggpath' provides a way to, so the next call reads the CDN again.

Usage

```
sdv_clear_cache()
```

Value

Invisibly NULL, called for its side effect.

See Also

[sdv_versions\(\)](#) for what is loaded now.

Examples

```
sdv_clear_cache()
```

sdv_color_palette	<i>Team colour palette</i>
-------------------	----------------------------

Description

A named colour vector for [ggplot2::scale_colour_manual\(\)](#) and friends. The R counterpart of sdvplot's `palette()`.

Usage

```
sdv_color_palette(
  league,
  teams = NULL,
  which = c("primary", "secondary"),
  season = NULL,
  id_system = "auto",
  strict = FALSE
)
```

Arguments

league	A league key from <code>sdv_leagues()</code> . Required: one abbreviation means different teams in different leagues.
teams	Team identifiers, or NULL for every team.
which	"primary" or "secondary".
season	NULL, one season year, or one per value: picks the team that used a reused code that season. Without one, a reused code means its current holder. Split seasons take their ending year (2021 for 2020-21).
id_system	"auto" (try every id system in priority order) or one id system: "team_id", "espn", "espn_abbr", "nhl", "nflverse", "mlbstats", "nba_api", "hockeytech", "ncaa", "pff", "cricinfo", "cfbd", "bref", "sportsipy", "fangraphs", "sdvplotr", "name", or "nhl_id" (NHL stats API ids, which overlap ESPN's, so "auto" never reads a number as one).
strict	If TRUE, a value that does not resolve is an error; otherwise it becomes NA with one warning per call.

Value

A named character vector of hex colours, unresolved teams dropped.

Examples

```
sdv_color_palette("nfl", c("KC", "BUF", "SF"))
```

sdv_colorway	<i>Team colours as a plotly colorway</i>
--------------	--

Description

Sets the chart's trace colours to the teams' colours, in order.

Usage

```
sdv_colorway(
  p,
  league,
  teams = NULL,
  which = c("primary", "secondary"),
  season = NULL,
  id_system = "auto"
)
```

Arguments

p	A plotly object.
league	A league key from <code>sdv_leagues()</code> . Required: one abbreviation means different teams in different leagues.
teams	Team identifiers, or NULL for every team.
which	"primary" or "secondary".
season	NULL for today's mark, or a season (or one per team): the mark the team wore that season where the archive dates its marks.
id_system	For headshots, as in <code>sdv_headshot_url()</code> .

Value

The plotly object (an htmlwidget).

Examples

```
if (requireNamespace("plotly", quietly = TRUE)) {
  plotly::plot_ly(type = "bar") |> sdv_colorway("nfl", c("KC", "BUF"))
}
```

sdv_court_coords

Convert Shot Locations to a sportyR Court Frame

Description

stats.nba.com / stats.wnba.com shot-chart data reports shot locations in the NBA's legacy frame: tenths of a foot, origin at the hoop, relative to the shooter's basket. hoopR's/wehoop's `load_nba_stats_shots()` / `load_wnba_stats_shots()` ship these columns as `x_legacy/y_legacy` (snake_case); the `Shot_Chart_Detail` element of hoopR's `nba_shotchartdetail()` uses the same frame, in the `LOC_X/LOC_Y` columns stats.nba.com returns (upper snake case). This converts them (feet, not tenths) into the frame `sportyR::geom_basketball("nba")` draws: origin at center court, baseline at `x = -47`, basket on the **left** at `x = -41.75`.

`provider = "euroleague"` converts the Euroleague shot frame of hoopR's `euroleague_game_points()` (`coord_x/coord_y`) instead, onto the FIBA court `sdv_surface("fiba")` draws, in meters.

Usage

```
sdv_court_coords(
  data,
  x_column = "x_legacy",
  y_column = "y_legacy",
  provider = "nba"
)
```

Arguments

data	A data frame with shot-location columns, e.g. from <code>hoopR::load_nba_stats_shots()</code> / <code>wehoop::load_wnba_stats_shots()</code> (<code>x_legacy/y_legacy</code>). <code>hoopR::nba_shotchartdetail()</code> returns a named list: pass its <code>Shot_Chart_Detail</code> data frame and name the <code>LOC_X/LOC_Y</code> columns, e.g. <code>sdv_court_coords(res\$Shot_Chart_Detail, "LOC_X", "LOC_Y")</code> .
x_column	String naming the column holding the stats-API <code>LOC_X</code> / <code>x_legacy</code> value (tenths of a foot), or Euroleague <code>coord_x</code> (centimeters). Default <code>"x_legacy"</code> .
y_column	String naming the column holding the stats-API <code>LOC_Y</code> / <code>y_legacy</code> value (tenths of a foot), or Euroleague <code>coord_y</code> (centimeters). Default <code>"y_legacy"</code> .
provider	The frame of data, which sets the output's units: <code>"nba"</code> (the default; stats.nba.com / stats.wnba.com, tenths of a foot in, feet out on the NBA/WNBA/NCAA court) or <code>"euroleague"</code> (<code>hoopR's euroleague_game_points()</code> <code>coord_x/coord_y</code> , centimeters in, meters out on the FIBA court). Case is ignored.

Details

The NBA legacy shot-location frame (`LOC_X/LOC_Y`, or `x_legacy/y_legacy`; tenths of a foot, hoop at the origin, relative to the shooter's basket) maps with negative `x` as the shooter's left corner: `court_y` is `x_column / 10` with no sign flip, so a left-corner three lands at negative `court_y` (the TV-bottom sideline), which is the shooter's left when facing the TV-left basket. Verified on real 2022-23 `shotchartdetail` data (Left Corner 3 `LOC_X` -249..-221, Right Corner 3 +221..+248).

Shots land on sportyR's TV-left half, so draw half-court charts with `sportyR::geom_basketball("nba", display_range = "defense")` or the full court; `display_range = "offense"` shows the TV-right half and none of the shots.

The converted points also fit sportyR's `"wnba"` and `"ncaa"` courts, which share the `"nba"` court's 94-foot floor and 5.25-foot basket offset. The input must still be in the NBA legacy frame (tenths of a foot, hoop at the origin: `x_legacy/y_legacy` or `LOC_X/LOC_Y`). Don't pass ESPN `coordinate_x/coordinate_y` from `hoopR/wehoop` play-by-play: they are already in feet on a center-court frame.

The Euroleague frame (`provider = "euroleague"`; measured on real games, 2026-10-06) is integer centimeters with the hoop at the origin, both teams mapped onto one basket, `coord_y` growing away from the baseline toward the court, and free throws encoded as `coord_x = coord_y = -1` (a sentinel, not a location), which become NA. The FIBA court is 28 x 15 m with its basket 1.575 m from the baseline, so `court_x = -12.425 + coord_y / 100` and `court_y = coord_x / 100`. Which sideline is positive `coord_x` is unverified, so a chart may be left-right mirrored; the court is symmetric, so distances and zones are unaffected.

Coordinate columns must be numeric, or character holding numbers, which is coerced (`nba_shotchartdetail()` returns every column as character). Factors, TRUE/FALSE and strings that aren't numbers (such as `" "` or `"NA"`) raise an error. Missing values stay NA, and an all-NA column (even a logical one) gives NA coordinates.

Value

data with `court_x` and `court_y` added (existing columns of those names are replaced); every other input column is kept as-is:

col_name	type	description
court_x	numeric	Along the court: the basket at -41.75 ft (nba) or -12.425 m (euroleague), half court at 0
court_y	numeric	Across the court: -25 to 25 ft (nba; negative = shooter's left) or -7.5 to 7.5 m (euroleague)

Examples

```
shots <- data.frame(x_legacy = c(-224, 240), y_legacy = c(39, 29))
sdv_court_coords(shots)

# Euroleague shots (hoopR::euroleague_game_points() columns) onto the FIBA court
euro <- data.frame(coord_x = c(0, 12, 650, -1), coord_y = c(0, 422, 50, -1))
sdv_court_coords(euro, "coord_x", "coord_y", provider = "euroleague")

if (requireNamespace("sportyR", quietly = TRUE)) {
  library(ggplot2)
  euro <- sdv_court_coords(euro, "coord_x", "coord_y", provider = "euroleague")
  sdv_surface("fiba", display_range = "defense") +
    geom_point(aes(court_x, court_y), data = euro, colour = "red", size = 3, na.rm = TRUE)
}
```

sdv_example_standings *Final regular-season standings for the examples*

Description

Real standings for one completed season in two leagues, one row per team: the 2025 NFL season and the 2025-26 NBA season. The reference examples run on it, so the table and plot helpers are shown on sports data. Each team is the abbreviation `sdv_teams()` lists, so it works as the team input of every logo, color and headshot helper as it is.

Usage

```
sdv_example_standings
```

Format

A tibble with 62 rows (32 NFL teams, then 30 NBA teams), ordered by league, conference, division and division finish, and 16 columns:

league "nfl" or "nba", the league value the helpers take.

season 2025 for the NFL; 2026 for the NBA, whose 2025-26 season goes by its ending year as in 'hoopR'.

team Team abbreviation.

espn_team_id ESPN team id.

team_name Full team name.

conference "AFC" or "NFC"; "Eastern" or "Western".

division Division, as "AFC East" or "Atlantic".

wins, losses, ties The regular-season record. NBA games can't end tied, so ties is 0 there.

win_pct Winning percentage, a tie counting as half a win, rounded to three decimals.

points_for, points_against Regular-season points scored and allowed.

conference_rank Final place in the conference with the league's tiebreakers applied: the NFL's playoff seeds are 1 to 7, the NBA's playoff places 1 to 6 and its play-in places 7 to 10.

playoff_wins Postseason wins, not counting the NBA's play-in; NA for a team that missed the playoffs.

last_season_wins Regular-season wins the season before: the NFL's 2024 and the NBA's 2024-25.

Source

Built by `data-raw/sdv_examples.R`. NFL: the `nflverse` schedules release, `nflreadr::load_schedules(2025)`, with the standings and conference ranks from `nflseedR::nfl_standings()`. NBA: the `sportsdataaverse-data` team box release, `hoopR::load_nba_team_box(2026)`, counting standings games only (not the NBA Cup final or the All-Star games), with divisions and conference ranks from ESPN's standings (the endpoint `hoopR::espn_nba_standings()` reads), whose records the script checks against the box scores. `last_season_wins` comes from the same loaders one season back. Team names and ESPN ids come from `sdvplotR`'s team reference.

Examples

```
# one division, in finishing order
subset(sdv_example_standings, division == "NFC West", c(team, wins:win_pct))

# the teams that reached the playoffs
playoffs <- subset(sdv_example_standings, !is.na(playoff_wins))
table(playoffs$league)

# with logos, which are downloaded

subset(sdv_example_standings, division == "Atlantic", c(team, wins, losses)) |>
  gt::gt() |>
  gt_sdv_logos(columns = "team", league = "nba")
```

sdv_headshot_url

Player headshot URLs

Description

The headshot image URL for player ids, as every `sdvplotR` headshot helper draws it. The R counterpart of `sdvplot`'s `headshot_url()`.

Usage

```
sdv_headshot_url(player_id, league, id_system = "espn")
```

Arguments

player_id	A vector of player ids.
league	A league key from sdv_leagues() . Required: one abbreviation means different teams in different leagues.
id_system	"espn" (ESPN athlete ids; the default, for every league with ESPN headshots: nfl, nba, wnba, mlb, nhl, cfb, mbb, wbb), "gsis" (nflverse's NFL gsis ids, such as "00-0033873", read through a published map) or "league" (the league's own player id: NBA / WNBA Stats PERSON_ID, MLBAM, NHL API; for the NFL, the gsis id).

Value

A character vector the length of player_id: the URL, or NA for a missing or malformed id or (gsis) one not in the map.

See Also

```
sdv\_logo\_url\(\).
```

Examples

```
sdv_headshot_url(3139477, "nfl")
sdv_headshot_url("2544", "nba", id_system = "league")

# nflverse gsis ids read the published headshot map
sdv_headshot_url("00-0033873", "nfl", id_system = "gsis")
```

sdv_leagues	<i>Leagues sdvplotR knows</i>
-------------	-------------------------------

Description

The SportsDataverse league keys every league argument takes: the leagues of the shared team index sdvplot (Python) and sdvplot-js use.

Usage

```
sdv_leagues()
```

Value

A sorted character vector.

See Also

[sdv_teams\(\)](#) for each league's teams.

Examples

```
sdv_leagues()
```

sdv_logo_url	<i>Team logo and wordmark URLs</i>
--------------	------------------------------------

Description

The image URL behind every sdvplotR geom, theme element and table helper: the SportsDataverse archive's copy of the team's mark (content-addressed and immutable), chosen for the season. The R counterpart of sdvplot's `logo_url()`.

Usage

```
sdv_logo_url(  
  team,  
  league,  
  season = NULL,  
  variant = "default",  
  mark_type = "logo",  
  id_system = "auto",  
  strict = FALSE  
)
```

Arguments

team	A vector of team identifiers, or conference / league keys ("SEC", "Big 12", "AFC"), read after every team alias.
league	A league key from sdv_leagues() . Required: one abbreviation means different teams in different leagues.
season	NULL for today's mark, or a season (or one per team): the mark the team wore that season where the archive dates its marks.
variant	"default", "dark", or a named variant from sdv_marks() ("scoreboard", "grayscale", "cap_on_light", ...). A team without it gets its closest mark of the same polarity; a name no mark in the archive has is an error listing the league's variants.
mark_type	"logo" or "wordmark".
id_system	"auto" (try every id system in priority order) or one id system: "team_id", "espn", "espn_abbr", "nhl", "nflverse", "mlbstats", "nba_api", "hockeytech", "ncaa", "pff", "cricinfo", "cfbd", "bref", "sportsipy", "fangraphs", "sdvplotr", "name", or "nhl_id" (NHL stats API ids, which overlap ESPN's, so "auto" never reads a number as one).

strict If TRUE, a value that does not resolve is an error; otherwise it becomes NA with one warning per call.

Value

A character vector the length of team: the URL, or NA (with a warning) where the team does not resolve or has no such mark.

See Also

[sdv_headshot_url\(\)](#), [sdv_marks\(\)](#).

Examples

```
sdv_logo_url(c("KC", "Buffalo Bills", "OAK"), "nfl")
sdv_logo_url("NYJ", "nfl", variant = "scoreboard")
sdv_logo_url("QUE", "nhl", season = 1990)
sdv_logo_url("KC", "nfl", mark_type = "wordmark")
sdv_logo_url("SEC", "cfb")
```

sdv_marks	<i>Every archived mark for a team</i>
-----------	---------------------------------------

Description

The logo manifest’s rows for one team (logos and wordmarks, every variant, source and era), best first: the rows the logo helpers choose from. The R counterpart of sdvplot’s marks().

Usage

```
sdv_marks(team, league, season = NULL, id_system = "auto")
```

Arguments

team	One team identifier, or a conference or league key ("SEC", "AFC").
league	A league key from sdv_leagues() . Required: one abbreviation means different teams in different leagues.
season	NULL, or one season: which team team names where a code was reused across eras. The rows returned cover every season.
id_system	"auto" (try every id system in priority order) or one id system: "team_id", "espn", "espn_abbr", "nhl", "nflverse", "mlbstats", "nba_api", "hockeytech", "ncaa", "pff", "cricinfo", "cfbd", "bref", "sportsipy", "fangraphs", "sdvplotr", "name", or "nhl_id" (NHL stats API ids, which overlap ESPN’s, so "auto" never reads a number as one).

Value

A data frame of manifest rows with team_id, variant, mark_type, valid_from, valid_to (the effective season range), source, source_rank and archive_url.

Examples

```
m <- sdv_marks("KC", "nfl")
table(m$variant)
```

sdv_pitch_coords

Convert Soccer Event Coordinates to the sdvplotR Pitch Frame

Description

Every soccer data provider uses its own coordinate frame. Opta and Wyscout run 0-100 on both axes and are not to scale, StatsBomb uses 120 x 80, tracking providers use meters or centimeters from the center spot, and ESPN reports the distance from the attacked goal as a fraction of half the pitch. `sdv_pitch_coords()` converts any of them to one frame: meters, origin at the center spot, a regulation 105 x 68 m pitch, attacking toward +x, with +y on the attacker's left. That is the pitch `sdv_surface()` draws for "soccer".

Usage

```
sdv_pitch_coords(
  data,
  provider,
  x_column = NULL,
  y_column = NULL,
  flip = NULL,
  pitch_length = NULL,
  pitch_width = NULL
)
```

Arguments

<code>data</code>	A data frame of events.
<code>provider</code>	The coordinate frame of data: "opta" (alias "statsperform"), "wyscout", "statsbomb", "uefa", "impect", "espn", "tracab", "skillcorner", "secondspectrum" or "metrica". Case is ignored.
<code>x_column, y_column</code>	Strings naming the coordinate columns. NULL (the default) uses the provider's usual names: "field_position_x" and "field_position_y" for ESPN (the output of <code>Python espn_soccer_game_plays()</code>), "x" and "y" otherwise.
<code>flip</code>	NULL (no row flipped), TRUE/FALSE for every row, or a string naming a logical column with no missing values; TRUE rows are turned half a turn.
<code>pitch_length, pitch_width</code>	The venue's size in meters, for tracking providers only (the Laws of the Game allow 90-120 by 45-90).

Details

The conversion is piecewise-linear between pitch landmarks: goal line, six-yard line, penalty spot, box edge and halfway line along the pitch; touchline, box side, six-yard side and post across it. A shot on the edge of an Opta box therefore lands on the edge of the regulation box, not 1.35 m outside it as a linear rescale would put it. Points beyond the outermost landmark are extrapolated from the nearest segment, not clamped. The landmark values come from 'mplsoccer' and ship in `system.file("extdata", "pitch_landmarks.csv", package = "sdvplotR")`.

Tracking providers ("tracab", "skillcorner", "secondspectrum", "metrica") are measured on the venue's real pitch, so they need `pitch_length` and `pitch_width`. ESPN frames are derived from the Opta frame: `opta_x = 100 - 50 * x` and `opta_y = 100 * (1 - y)`. ESPN marks an event with no location as `(0, 0)`, which becomes NA.

Event providers record every action as if the team attacked left to right. To draw two teams (or two halves of tracking data) attacking opposite ends, flip them: a flipped row is turned half a turn, so its `x` and `y` both change sign and each wing stays on its own side.

Value

data with `pitch_x` and `pitch_y` added (existing columns of those names are replaced); every other input column is kept as-is:

col_name	type	description
<code>pitch_x</code>	numeric	Meters along the pitch: -52.5 is the defended goal line, 52.5 the attacked one, 0 the halfway line
<code>pitch_y</code>	numeric	Meters across the pitch: -34 to 34; positive = the attacker's left

See Also

`sdv_surface()` draws the pitch; `ggsoccer::rescale_coordinates()` converts between provider frames without the regulation-pitch output.

Examples

```
shots <- data.frame(x = c(88.5, 83, 50), y = c(50, 21.1, 100))
sdv_pitch_coords(shots, "opta")

# two teams attacking opposite ends
shots$away <- c(FALSE, TRUE, FALSE)
sdv_pitch_coords(shots, "opta", flip = "away")

# tracking data, measured on the venue's pitch
tracking <- data.frame(x = c(-4150, 3600), y = c(0, -2016))
sdv_pitch_coords(tracking, "tracab", pitch_length = 105, pitch_width = 68)
```

sdv_resolve

*Resolve team identifiers to canonical team ids***Description**

Turns whatever team identifiers you have (abbreviations, full names, ESPN, nflverse, MLB Stats, nba_api, HockeyTech, CFBD, Sports Reference and other ids) into the index's canonical `team_id`, never guessing. Every `sdvplotR` helper that takes a team resolves it this way. The R counterpart of `sdvplot`'s `resolve()`, and it gives the same answers.

Usage

```
sdv_resolve(x, league, season = NULL, id_system = "auto", strict = FALSE)
```

Arguments

<code>x</code>	A vector of team identifiers (character, numeric or factor).
<code>league</code>	A league key from sdv_leagues() . Required: one abbreviation means different teams in different leagues.
<code>season</code>	NULL, one season year, or one per value: picks the team that used a reused code that season. Without one, a reused code means its current holder. Split seasons take their ending year (2021 for 2020-21).
<code>id_system</code>	"auto" (try every id system in priority order) or one id system: "team_id", "espn", "espn_abbr", "nhl", "nflverse", "mlbstats", "nba_api", "hockeytech", "ncaa", "pff", "cricinfo", "cfbd", "bref", "sportsipy", "fangraphs", "sdvplotr", "name", or "nhl_id" (NHL stats API ids, which overlap ESPN's, so "auto" never reads a number as one).
<code>strict</code>	If TRUE, a value that does not resolve is an error; otherwise it becomes NA with one warning per call.

Value

A character vector of `team_ids`, the length of `x`.

See Also

[sdv_suggest\(\)](#) for a value that did not resolve, [sdv_teams\(\)](#).

Examples

```
sdv_resolve(c("KC", "Kansas City Chiefs", "OAK"), "nfl")
sdv_resolve("QUE", "nhl", season = 1990)
sdv_resolve(c("Iowa St.", "Miami (OH)"), "mbb")
```

sdv_suggest

Suggest teams for a value that did not resolve

Description

Up to n candidate teams for a misspelled or unknown value, best first. It never picks one for you: similar names can be different teams. The R counterpart of sdvplot's suggest().

Usage

```
sdv_suggest(x, league, n = 5)
```

Arguments

x	One team value.
league	A league key from sdv_leagues() . Required: one abbreviation means different teams in different leagues.
n	The most candidates to return.

Value

A data frame with team_id and name, empty when nothing is close.

Examples

```
sdv_suggest("Kansas Cty Chiefs", "nfl", n = 2)
```

sdv_surface

Team-Styled Playing Surfaces

Description

Draws a regulation court, field or rink with 'sportyR' and, given a team, paints a few of its features in that team's colors from [sdv_team_colors\(\)](#). The result is a ggplot, so data layers go on top.

These are **stylized** surfaces built from team colors, not the team's real court, field or rink design: no dataset of real team surface designs (paint colors, center logos, end zone art, special-event floors) exists yet.

Usage

```
sdv_surface(league, team = NULL, season = NULL, center_logo = FALSE, ...)
```

Arguments

league	A league with a playing surface: one of names(surface_map), the leagues of sdv_leagues() that 'sportyR' draws, plus "fiba". Each maps to a 'sportyR' surface, after sdvplot's (Python) surfaces:
league	surface
"nfl", "aaf", "ufl", "usfl", "xfl"	sportyR::geom_football("nfl")
"cfb"	sportyR::geom_football("ncaa")
"nba"	sportyR::geom_basketball("nba")
"nbagl"	sportyR::geom_basketball("nba g league")
"wnba"	sportyR::geom_basketball("wnba")
"mbb", "wbb"	sportyR::geom_basketball("ncaa")
"fiba"	sportyR::geom_basketball("fiba") (28 x 15 m)
"mlb"	sportyR::geom_baseball("mlb")
"milb"	sportyR::geom_baseball("milb")
"ncaa_baseball"	sportyR::geom_baseball("ncaa")
"nhl", "whl"	sportyR::geom_hockey("nhl") (no WHL rink in 'sportyR')
"pwhl", "ahl", "echl", "ohl", "qmjhl", "ushl", "phf"	sportyR::geom_hockey() of the same league
"ncaa_mhockey", "ncaa_whockey"	sportyR::geom_hockey("ncaa")
"soccer"	sportyR::geom_soccer("fifa"), 105 x 68 m (pitch_updates)
	Python's sdvplot draws the PWHL on an NHL rink and the G League on its own court class; 'sportyR' 2.2.3 has both, so they use their own here.
team	NULL (the default) for the plain regulation surface, or one team identifier, resolved by sdv_resolve() .
season	A season for the team's colors and center logo (see sdv_team_colors()); NULL is the current identity.
center_logo	If TRUE, draw the team's logo at center court, center ice or midfield. Needs a team; ignored with a warning for "mlb", whose surface is an infield with no center.
...	Passed to the 'sportyR' geom: display_range, rotation, x_trans, y_trans, xlims, ylims, the unit arguments, and color_updates, which overrides the team colors feature by feature (see sportyR::cani_color_league_features() for the names).

Details

With a team, these features change; everything else keeps 'sportyR's regulation colors, so the court stays wood, the field green and the ice white:

- Basketball: the painted area and the court apron take the primary color. The lines drawn inside the paint (restricted arc, free-throw circle dashes, lower defensive boxes) turn black or white, whichever contrasts more with it.
- Football: both end zones take the primary color.
- Hockey: the center line, center faceoff circle and center faceoff spot take the primary color, or the secondary where the primary is too pale to read on white ice (under 3:1 contrast); the boards take the primary.

- Baseball: nothing. No part of a regulation infield is team-colored (the green background is the outfield grass), so the team is checked and the surface stays 'sportyR's.

Soccer and FIBA surfaces are drawn in meters. "soccer" defaults to a regulation 105 x 68 m pitch, the frame `sdv_pitch_coords()` converts to; 'sportyR's own "fifa" default is FIFA's 120 x 90 m maximum. Soccer takes a team for its center logo (18.3 m, the center circle); the pitch keeps its regulation colors. FIBA takes no team yet.

Surfaces use 'sportyR's coordinates: the origin at the center, in feet (yards for football, meters for soccer and FIBA). The center logo is sized in those units (12 feet on a court, 10 yards on a field, 24 feet on a rink, 18.3 m on a pitch) and follows `x_trans`, `y_trans`, `rotation` and the `unit` arguments.

Value

A ggplot object (`ggplot2::ggplot()`) with `coord_fixed()`; add layers with `+`.

Examples

```
if (requireNamespace("sportyR", quietly = TRUE)) {
  library(ggplot2)

  # in a team's colors, with its logo at center court, and shots on top
  shots <- data.frame(x = c(-40, -35, 30), y = c(5, -12, 0))
  sdv_surface("nba", "BOS", center_logo = TRUE) +
    geom_point(aes(x, y), data = shots, color = "red", size = 3)
}
```

sdv_team_colors

Team colours

Description

Primary and secondary colours (hex) for teams, resolved from any identifier `sdv_resolve()` accepts. The R counterpart of `sdvplot's team_colors()`.

Usage

```
sdv_team_colors(
  league,
  team = NULL,
  which = c("primary", "secondary", "all"),
  season = NULL,
  id_system = "auto",
  strict = FALSE,
  include_conferences = FALSE
)
```

Arguments

league	A league key from sdv_leagues() . Required: one abbreviation means different teams in different leagues.
team	Team identifiers, or NULL for every team in the league.
which	"primary", "secondary", or "all" (both).
season	NULL, one season year, or one per value: picks the team that used a reused code that season. Without one, a reused code means its current holder. Split seasons take their ending year (2021 for 2020-21).
id_system	"auto" (try every id system in priority order) or one id system: "team_id", "espn", "espn_abbr", "nhl", "nflverse", "mlbstats", "nba_api", "hockeytech", "ncaa", "pff", "cricinfo", "cfbd", "bref", "sportsipy", "fangraphs", "sdvplotr", "name", or "nhl_id" (NHL stats API ids, which overlap ESPN's, so "auto" never reads a number as one).
strict	If TRUE, a value that does not resolve is an error; otherwise it becomes NA with one warning per call.
include_conferences	With team = NULL, also list conference and league rows (see sdv_teams()).

Value

With team given, a character vector named by team (NA where a team does not resolve); for which = "all", "primary, secondary" strings. With team = NULL, a vector named by abbreviation (the team_id where a team has none), or for which = "all" a data frame:

col_name	type	description
team_id	character	Canonical team id
abbr	character	Abbreviation (NA where the league has none)
name	character	Full name
primary	character	Primary colour (hex)
secondary	character	Secondary colour (hex; NA where none)

Examples

```
sdv_team_colors("nfl", "KC")
sdv_team_colors("nfl", c("KC", "Buffalo Bills"), which = "all")
sdv_team_colors("ahl", which = "primary")
```

sdv_team_factor	<i>Order Team Names as a Factor</i>
-----------------	-------------------------------------

Description

A factor of teams as given, levels ordered by each team's abbreviation; values that do not resolve are NA (with the resolver's warning), so downstream ggplot2 scales drop them.

Usage

```
sdv_team_factor(teams, league, season = NULL, id_system = "auto")
```

Arguments

teams Character vector of team names.
 league A league key from `sdv_leagues()`.
 season, id_system Passed to `sdv_resolve()`.

Value

A factor of teams.

Examples

```
suppressWarnings(sdv_team_factor(c("KC", "BUF", "invalid"), league = "nfl"))
```

sdv_team_tiers	<i>Create Team Tier Plots</i>
----------------	-------------------------------

Description

This function sets up a ggplot to visualize team tiers across any league supported by sdvplotR. Teams are organized by performance or other metrics into labeled tiers.

Usage

```
sdv_team_tiers(
  data,
  league,
  title = glue::glue("{toupper(league)} Team Tiers"),
  subtitle = glue::glue("created with the #sdvplotR Tiermaker"),
  caption = NULL,
  tier_desc = c(`1` = "Elite", `2` = "Very Good", `3` = "Medium", `4` = "Bad", `5` =
    "What are they doing?", `6` = "", `7` = ""),
  presort = FALSE,
  alpha = 0.8,
  width = 0.075,
  no_line_below_tier = NULL,
  devel = FALSE,
  theme = c("dark", "light"),
  variant = "auto",
  season = NULL,
  id_system = "auto",
  strict = FALSE
)
```

Arguments

<code>data</code>	A data frame that must include the variables <code>tier_no</code> (the number of the tier starting from the top tier no. 1) and <code>team</code> (the team: anything sdv_resolve() accepts). If data includes the variable <code>tier_rank</code> , these ranks will be used within each tier. Otherwise, if <code>presort = FALSE</code> , the function will assume that data is already sorted and if <code>presort = TRUE</code> , teams will be sorted alphabetically within tiers.
<code>league</code>	A league key from sdv_leagues() .
<code>title</code>	The title of the plot. If <code>NULL</code> , it will be omitted.
<code>subtitle</code>	The subtitle of the plot. If <code>NULL</code> , it will be omitted.
<code>caption</code>	The caption of the plot. If <code>NULL</code> , it will be omitted.
<code>tier_desc</code>	A named vector of tier descriptions. Names must equal the tier numbers from <code>tier_no</code> .
<code>presort</code>	If <code>FALSE</code> (the default) the function assumes that the teams are already sorted within the tiers. Will otherwise sort alphabetically.
<code>alpha</code>	The alpha channel of the logos (transparency level).
<code>width</code>	The desired width of the logo in npc.
<code>no_line_below_tier</code>	Vector of tier numbers. The function won't draw tier separation lines below these tiers.
<code>devel</code>	Determines if logos shall be rendered. If <code>FALSE</code> (the default), logos will be rendered on each run. If <code>TRUE</code> the team abbreviations will be plotted instead of the logos. This is much faster and helps with plot development.
<code>theme</code>	"dark" (the default: a near-black background with white labels) or "light" (a white background with dark labels and lines).
<code>variant</code>	The logo variant to draw. "auto" (the default, as <code>sdvplot's team_tiers()</code>) draws each team's dark-background mark ("dark") on the dark theme and its default mark on the light theme, so dark logos such as the Capitals' or Penn State's don't fade into the dark background. Any other value is "default", "dark" or a named variant from sdv_marks() (see sdv_logo_url()), drawn on either theme. A team without the variant draws its closest mark, with no warning.
<code>season, id_system, strict</code>	Passed to sdv_resolve() and the logo lookup: a season for dated marks, the identifier system of team, and whether an unresolved team is an error.

Value

A ggplot object.

Examples

```
library(sdvplotR)
library(ggplot2)
```

```
# Build the team tiers data
df <- data.frame(
  tier_no = c(1, 1, 2, 2, 3, 3),
  team = c("KC", "DET", "BAL", "PHI", "BUF", "GB")
)

# Plot team tiers, each team's primary logo on white
sdv_team_tiers(df, league = "nfl", theme = "light")
```

sdv_teams

*Teams in the sdvplotR index***Description**

One row per team: its canonical team_id (the key every helper resolves to), names, abbreviation, conference and colours. The R counterpart of sdvplot's teams().

Usage

```
sdv_teams(league = NULL, include_conferences = FALSE)
```

Arguments

league A league key from [sdv_leagues\(\)](#), or NULL for every league.

include_conferences

Also list the conference and league rows (the college conferences, and the AFC, NFC and NFL). Their program is "conference" or "league" and their team_id is the short name ("SEC", "AFC"). The logo helpers draw them by that key; colour helpers and [sdv_resolve\(\)](#) never answer one.

Value

A data frame:

col_name	type	description
league	character	League key
team_id	character	Canonical team id (the ESPN id where ESPN has the team)
abbr	character	Abbreviation (NA where the league has none)
name	character	Full name
short_name	character	Short display name
location	character	City or school
program	character	NA for a team; "conference" or "league" for those rows
conference_id	character	The team's conference row team_id
conference	character	Conference name
color_primary	character	Primary colour (hex)
color_secondary	character	Secondary colour (hex)
color_source	character	"nflverse", "espn", "logo", "cbbplotR" or "fallback"

Examples

```
sdv_teams("nfl")
sdv_teams("cfb", include_conferences = TRUE)
```

sdv_versions	<i>Versions of the data sdvplotR is using</i>
--------------	---

Description

Which team index and logo manifest the session has loaded and where they came from: "live" (the SportsDataverse CDN, or the disk cache within a day of it), "cache" (an older disk copy, after a failed refresh) or "snapshot" (the copy bundled with the package: offline, with options(sdvplotR.offline = TRUE), or during a CRAN check). The R counterpart of sdvplot's versions().

Usage

```
sdv_versions()
```

Value

A list with package, index_version, index_source, manifest_rows and manifest_source.

Examples

```
sdv_versions()
```

theme_bg	<i>Background colors used by the gt_theme_*() table themes</i>
----------	--

Description

A lookup of the background color each gt_theme_* function sets, used to match a saved image's canvas to the table sitting on it. [gt_save_crop\(\)](#) and [gt_social_crop\(\)](#) pad the image using their own bg argument, and a mismatch shows up as a border around the table.

Usage

```
theme_bg
```

Format

A tibble with one row per theme, and one per style for the themes that take a style argument:

theme The theme function name.

has_style The style value the row applies to ("light" or "dark") for `gt_theme_sdv()`, `gt_theme_sofa()` and `gt_theme_tier()`, or "" for themes without one.

bg The background color the theme applies, as a hex code.

Source

Read back from each theme's `table.background.color` by `data-row/theme_bg.R.gt_theme_drench()` takes its background from its `color` argument and `gt_theme_broadsheet()` from its `paper` argument; their rows hold the default.

See Also

`gt_save_crop()`, `gt_social_crop()`.

Examples

```
# look up the background a theme uses, then match the canvas to it
bg <- theme_bg$bg[theme_bg$theme == "gt_theme_gtutils"]
bg

# saving needs a headless Chrome (webshot2)

nfc_west <- subset(
  sdv_example_standings, division == "NFC West",
  c(team_name, wins:points_against)
)
gt::gt(nfc_west) |>
  gt_theme_gtutils() |>
  gt_save_crop(tempfile(fileext = ".png"), bg = bg)
```

 theme_sdv

Theme Helpers for SDV Axis Labels

Description

These functions are convenience wrappers around a theme call that activates markdown in x-axis and y-axis labels using `ggtext::element_markdown()`.

Usage

```
theme_x_sdv()
```

```
theme_y_sdv()
```

Details

These functions set `ggtext::element_markdown()` on `axis.text.x` and its position children `axis.text.x.bottom/axis.text.x.top` (`theme_x_sdv()`), or on `axis.text.y`, `axis.text.y.left` and `axis.text.y.right` (`theme_y_sdv()`). Complete themes in 'ggplot2' 4 (`theme_minimal()`, ...) set those children themselves, and a markdown parent alone would not replace them. Add these after any complete theme, which replaces every element. They are made to be used in conjunction with `scale_x_sdv()` and `scale_y_sdv()` respectively.

Value

A ggplot2 theme object.

See Also

`scale_x_sdv()`, `scale_y_sdv()`

Examples

```
library(sdvplotR)
library(ggplot2)

team_abbr <- sdv_teams("nfl")$abbr[1:8]

df <- data.frame(
  random_value = runif(length(team_abbr), 0, 1),
  teams = team_abbr
)

if (requireNamespace("ggtext", quietly = TRUE)) {
  ggplot(df, aes(x = teams, y = random_value)) +
    geom_col(width = 0.5) +
    scale_x_sdv(league = "nfl") +
    theme_minimal() +
    theme_x_sdv()
}
```

Index

* datasets

- sdv_example_standings, 146
- theme_bg, 161
- add_sdv_headshots (add_sdv_logos), 4
- add_sdv_logos, 4
- add_sdv_wordmarks (add_sdv_logos), 4
- addSdvLogoMarkers, 7
- aes(), 11, 14, 18, 20
- annotation_borders(), 13, 16, 19, 21
- element_sdv_headshot
 - (element_sdv_logo), 8
- element_sdv_logo, 8
- element_sdv_raster (element_sdv_logo), 8
- element_sdv_wordmark
 - (element_sdv_logo), 8
- fortify(), 11, 14, 18, 20
- geom_sdv_headshots, 11
- geom_sdv_headshots_interactive
 - (geom_sdv_logos_interactive), 17
- geom_sdv_logos, 14
- geom_sdv_logos(), 5, 17, 22
- geom_sdv_logos_interactive, 17
- geom_sdv_logos_interactive(), 5
- geom_sdv_wordmarks, 19
- geom_sdv_wordmarks_interactive
 - (geom_sdv_logos_interactive), 17
- GeomSDVheadshot (geom_sdv_headshots), 11
- GeomSDVlogo (geom_sdv_logos), 14
- GeomSDVwordmark (geom_sdv_wordmarks), 19
- ggiraph::girafe(), 17
- ggpath::element_path(), 10
- ggpath::element_raster(), 8, 10
- ggplot(), 11, 14, 18, 20
- ggplot2::ggplot(), 16, 22, 156
- ggplot2::layer(), 16, 22
- ggplot2::scale_color_manual(), 140
- ggplot2::scale_colour_manual(), 142
- ggplot2::scale_fill_manual(), 140
- ggplot2::scale_x_discrete(), 137
- ggplot2::scale_y_discrete(), 137
- ggplot2::theme, 8
- ggplot2::theme_minimal(), 10, 23, 138
- ggsoccer::rescale_coordinates(), 152
- ggtxt::element_markdown(), 23, 138, 162, 163
- ggtitle_image, 23
- ggtitle_image(), 23
- gt::cells_body(), 78, 79, 81
- gt::cells_column_labels(), 78, 79, 81
- gt::cells_row_groups(), 78, 79, 81
- gt::cells_stub(), 78, 79, 81
- gt::gt(), 76, 78, 79, 81
- gt::opt_row_stripping(), 109, 113
- gt_538_caption, 24
- gt_538_caption(), 47, 90–92, 95, 128
- gt_bold_rows, 26
- gt_border_bars_bottom, 27
- gt_border_bars_top, 29
- gt_border_bars_top(), 27
- gt_border_grid, 31
- gt_color_pills, 32
- gt_color_pills(), 35, 36, 40, 58
- gt_color_ranks, 34
- gt_color_ranks(), 32, 34, 40, 56–58, 108, 131
- gt_color_results, 36
- gt_column_subheaders, 38
- gt_cutline, 39
- gt_cutline(), 63, 94
- gt_delta, 41
- gt_fmt_rank, 43
- gt_fmt_rank(), 45, 84, 115
- gt_fmt_tally, 44

- gt_grid, 46
- gt_grid(), 58, 61, 73, 93, 94, 96, 112, 128
- gt_group_strips, 49
- gt_highlight_cells, 50
- gt_highlight_cells(), 59, 88
- gt_highlight_na, 52
- gt_highlight_na(), 67
- gt_indicator_boxes, 53
- gt_indicator_boxes(), 59
- gt_legend_continuous, 56
- gt_legend_continuous(), 34, 36, 59, 61, 102, 127, 128
- gt_legend_discrete, 59
- gt_legend_discrete(), 56, 58, 126, 128
- gt_marginalia, 62
- gt_merge_stack_team_color, 64
- gt_outliers, 65
- gt_outliers(), 53
- gt_percentile_bar, 67
- gt_percentile_bar(), 58
- gt_row_accent, 70
- gt_save_batch, 71
- gt_save_crop, 73
- gt_save_crop(), 47, 89, 90, 95, 97, 99–101, 103, 104, 106–108, 110, 111, 113, 115, 116, 118–120, 122–124, 161, 162
- gt_scale_note, 75
- gt_scale_note(), 42
- gt_sdv_cols_label, 76
- gt_sdv_cols_label(), 78, 80, 81
- gt_sdv_headshots, 77
- gt_sdv_headshots(), 77, 80, 81
- gt_sdv_logos, 79
- gt_sdv_logos(), 77, 78, 81, 116, 134
- gt_sdv_wordmarks, 80
- gt_sdv_wordmarks(), 77, 78, 80
- gt_set_font, 82
- gt_significance, 83
- gt_significance(), 100
- gt_snake, 85
- gt_snake(), 48, 58, 88, 93, 94, 96
- gt_snake_align, 87
- gt_snake_align(), 86
- gt_social_crop, 89
- gt_social_crop(), 74, 97, 99–101, 103, 104, 106–108, 110, 111, 113, 115, 116, 118–120, 122–124, 161, 162
- gt_social_tag, 90
- gt_social_tag(), 129
- gt_spotlight, 92
- gt_spotlight(), 41, 67, 71, 115
- gt_stack_tables, 94
- gt_stack_tables(), 46, 48, 87
- gt_theme_almanac, 97
- gt_theme_almanac(), 40
- gt_theme_athletic, 98
- gt_theme_booktabs, 99
- gt_theme_booktabs(), 125
- gt_theme_broadsheet, 101
- gt_theme_broadsheet(), 162
- gt_theme_brutalist, 102
- gt_theme_drench, 103
- gt_theme_drench(), 162
- gt_theme_gtutils, 105
- gt_theme_kenpom, 106
- gt_theme_midnight, 107
- gt_theme_midnight(), 65, 104, 131
- gt_theme_ncaa, 109
- gt_theme_pl, 110
- gt_theme_preview, 111
- gt_theme_savant, 113
- gt_theme_scoreboard, 114
- gt_theme_sdv, 115
- gt_theme_sdv(), 117, 118, 162
- gt_theme_sdv_team, 117
- gt_theme_sdv_team(), 116
- gt_theme_sofa, 118
- gt_theme_sofa(), 112, 162
- gt_theme_swiss, 120
- gt_theme_terminal, 121
- gt_theme_terminal(), 108, 131
- gt_theme_tier, 122
- gt_theme_tier(), 126, 162
- gt_theme_tufte, 124
- gt_theme_tufte(), 100
- gt_tiers, 125
- gt_tiers(), 59, 60, 123
- gt_title_header, 127
- gt_title_header(), 47, 58, 61, 96, 102
- gt_watermark, 128
- gt_wrap_labels, 130
- key glyphs, 12, 15, 21
- layer position, 12, 15, 21
- layer stat, 12, 15, 20

layer(), [12](#), [15](#), [21](#)
 leaflet::addMarkers(), [8](#)

 pal_midnight, [108](#), [122](#), [131](#)
 plotly::ggplotly(), [5](#)
 plotly::layout(), [6](#)
 plotly::plot_ly(), [5](#)

 reactable::colDef(), [132–136](#)
 reactable::reactable(), [133](#), [135](#)
 reactable_sdv_cols_label, [132](#)
 reactable_sdv_cols_label(), [134](#)
 reactable_sdv_headshots
 (reactable_sdv_images), [133](#)
 reactable_sdv_images, [133](#)
 reactable_sdv_logos
 (reactable_sdv_images), [133](#)
 reactable_sdv_team_color, [135](#)
 reactable_sdv_team_color_bar
 (reactable_sdv_team_color), [135](#)
 reactable_sdv_team_color_bar(), [134](#)
 reactable_sdv_team_color_bg
 (reactable_sdv_team_color), [135](#)
 reactable_sdv_wordmarks
 (reactable_sdv_images), [133](#)

 scale_axes_sdv, [136](#)
 scale_color_sdv (scale_sdv), [139](#)
 scale_colour_sdv (scale_sdv), [139](#)
 scale_fill_sdv (scale_sdv), [139](#)
 scale_sdv, [139](#)
 scale_x_sdv (scale_axes_sdv), [136](#)
 scale_x_sdv(), [163](#)
 scale_x_sdv_headshots (scale_axes_sdv),
 [136](#)
 scale_y_sdv (scale_axes_sdv), [136](#)
 scale_y_sdv(), [163](#)
 scale_y_sdv_headshots (scale_axes_sdv),
 [136](#)
 scales::alpha(), [140](#)
 sdv_axis_logos, [140](#)
 sdv_clear_cache, [142](#)
 sdv_color_palette, [142](#)
 sdv_colorway, [143](#)
 sdv_court_coords, [144](#)
 sdv_example_standings, [146](#)
 sdv_headshot_url, [147](#)
 sdv_headshot_url(), [6](#), [19](#), [77](#), [141](#), [144](#), [150](#)
 sdv_leagues, [148](#)
 sdv_leagues(), [6](#), [7](#), [9](#), [12](#), [16](#), [19](#), [21](#), [23](#), [64](#),
 [77–79](#), [81](#), [117](#), [132](#), [134](#), [135](#), [137](#),
 [139](#), [141](#), [143](#), [144](#), [148–150](#),
 [153–155](#), [157–160](#)
 sdv_logo_url, [149](#)
 sdv_logo_url(), [77](#), [79](#), [81](#), [132](#), [134](#), [148](#),
 [159](#)
 sdv_marks, [150](#)
 sdv_marks(), [6](#), [7](#), [132](#), [134](#), [141](#), [149](#), [150](#),
 [159](#)
 sdv_pitch_coords, [151](#)
 sdv_pitch_coords(), [156](#)
 sdv_resolve, [153](#)
 sdv_resolve(), [14](#), [16](#), [17](#), [19](#), [22](#), [80](#), [117](#),
 [155](#), [156](#), [158–160](#)
 sdv_suggest, [154](#)
 sdv_suggest(), [153](#)
 sdv_surface, [154](#)
 sdv_surface(), [151](#), [152](#)
 sdv_team_colors, [156](#)
 sdv_team_colors(), [117](#), [118](#), [154](#), [155](#)
 sdv_team_factor, [157](#)
 sdv_team_tiers, [158](#)
 sdv_teams, [160](#)
 sdv_teams(), [14](#), [19](#), [146](#), [149](#), [153](#), [157](#)
 sdv_versions, [161](#)
 sdv_versions(), [142](#)

 theme_bg, [116](#), [161](#)
 theme_sdv, [162](#)
 theme_title_image (ggtitle_image), [23](#)
 theme_title_image(), [23](#), [24](#)
 theme_x_sdv (theme_sdv), [162](#)
 theme_x_sdv(), [138](#)
 theme_y_sdv (theme_sdv), [162](#)
 theme_y_sdv(), [138](#)