Package 'artpack'

September 13, 2025

Title Creates Generative Art Data

Version 0.2.0

Maintainer Meghan Harris <meghanha01@gmail.com>

Description Create data that displays generative art when mapped into a 'ggplot2' plot. Functionality includes specialized data frame creation for geometric shapes, tools that define artistic color palettes, tools for geometrically transforming data, and other miscellaneous tools that are helpful when using 'ggplot2' for generative art.

License MIT + file LICENSE

URL https://meghansaha.github.io/artpack/,
https://github.com/Meghansaha/artpack

BugReports https://github.com/Meghansaha/artpack/issues

Imports cli, dplyr (>= 1.0.8), grDevices, knitr, lifecycle, purrr (>= 0.3.4), rlang, sf, stringr (>= 1.5.0), tibble (>= 3.1.6)

Suggests covr, rmarkdown, spelling, testthat (>= 3.0.0), ggplot2 (>= 3.4.2), withr

Depends R (>= 4.1.0)

VignetteBuilder knitr

Config/testthat/edition 3

Config/testthat/parallel true

Encoding UTF-8

Language en-US

RoxygenNote 7.3.3

NeedsCompilation no

Author Meghan Harris [aut, cre, cph] (ORCID:
<https://orcid.org/0000-0003-3922-8101>)

Repository CRAN

Date/Publication 2025-09-13 18:40:02 UTC

Contents

`art_pals` *Custom-built artpack Color Palettes*

Description

The artpack palette picker. The `art_pals` function consists of 18 palettes: "arctic", "beach", "bw", "brood", "cosmos", "explorer", "gemstones", "grays", "icecream", "imagination", "majestic", "nature", "neon", "ocean", "plants", "rainbow", "sunnyside", "super"

Usage

```
art_pals(pal = NULL, n = 5, direction = "regular", randomize = FALSE)
```

Arguments

`pal` A character string of the desired artpack palette.
The 18 artpack palettes include:

- "arctic" - Icy blue and white colors
- "beach" - Sand-colored tans and ocean-colored blue colors
- "bw" - A gradient of black to white colors
- "brood" - A gradient of different shades of dark gray and black colors
- "cosmos" - Nebula-inspired blue, purple, and pink colors
- "explorer" - Pokemon-type inspired colors
- "gemstones" - Birthstone/Mineral-inspired colors
- "grays" - A gradient of dark, medium, and light gray colors
- "icecream" - A light pastel palette of cream, blue, brown, and pink colors

- "imagination" - 90's school supply-inspired colors
- "majestic" - Shades of majestic purple colors
- "nature" - A mix of tan, brown, green, and red colors
- "neon" - A neon spectrum of rainbow colors
- "ocean" - A gradient of dark to light blue colors
- "plants" - A gradient of dark to light green colors
- "rainbow" - A vibrant mix of rainbow colors
- "sunnyside" - A retro-inspired mix of pink, orange, and yellow colors
- "super" - A marveling mix of heroic colors

`n` The numbers of colors desired in the output.
Default is `5`. `n` must be a positive integer with a value greater than 0

`direction` The direction of the palette
Default is "regular". Only two options accepted: "regular" or "reverse"

`randomize` Determines if the colors in the palette appear in a randomized order.
Default is `FALSE`. Boolean where only `TRUE` or `FALSE` is accepted.

Value

A Character Vector.

Examples

```
library(ggplot2)
dots <- data.frame(x = c(1:10), y = 2.5)
dots$fills <- art_pals("rainbow", 10)

dots |>
  ggplot(aes(x, y)) +
  theme_void() +
  geom_point(
    shape = 21,
    fill = dots$fills,
    color = "#000000",
    size = 10,
    stroke = 2
  )


dots_rev <- data.frame(x = c(1:10), y = 2.5)
dots_rev$fills <- art_pals("rainbow", 10, "reverse")

dots_rev |>
  ggplot(aes(x, y)) +
  theme_void() +
  geom_point(
    shape = 21,
    fill = dots_rev$fills,
    color = "#000000",
```

```
    size = 10,
    stroke = 2
  )

dots_random <- data.frame(x = c(1:10), y = 2.5)
dots_random$fills <- art_pals("rainbow", 10, randomize = TRUE)

dots_random |>
  ggplot(aes(x, y)) +
  theme_void() +
  geom_point(
    shape = 21,
    fill = dots_random$fills,
    color = "#000000",
    size = 10,
    stroke = 2
  )
```

`circle_data` *Data Generation for Circles*

Description

A tool for creating a data frame of values that creates a circle with a specified radius when plotted.

The `geom_path` and `geom_polygon` geoms are recommended with this data for use in `ggplot2` for generative art.

Usage

```
circle_data(
  x,
  y,
  radius,
  color = NULL,
  fill = NULL,
  n_points = 100,
  group_var = FALSE,
  group_prefix = "circle_"
)
```

Arguments

`x`	Numeric value of length 1 - The center `x` coordinate value of the circle.
`y`	Numeric value of length 1 - The center `y` coordinate value of the circle.
`radius`	Numeric value of length 1 that must be greater than 0 - The radius of the circle.
`color`	Character value of length 1 - The intended color of the circle's border. A valid `R` color from `colors()` or a standard 6 digit hexadecimal webcolor like "#000000"

fill	Character value of length 1 - The intended color of the circle. A valid R color from colors() or a standard 6 digit hexadecimal webcolor like "#000000"
n_points	Numeric value. Default is 100. This determines how many points the circle will have. This option can come in handy when using jitter options or other texture/illusion methods. Must be of length 1 and at least a value of 100.
group_var	Logical. Default is FALSE. If TRUE, a group variable will be added to the dataframe. Useful in iterative data generation.
group_prefix	Character string of length 1 - The prefix used for the group variable. Default is "circle_"

Value

A Tibble

Examples

```
# Creating one circle

library(ggplot2)
one_circle <- circle_data(x = 0, y = 0, radius = 5)

# Plot The Data
one_circle |>
  ggplot(aes(x, y)) +
  geom_path(color = "green") +
  coord_equal()

# To create multiple circles, use your preferred method of iteration:
# Creating two circles

library(purrr)
library(dplyr)

# Make your specs
x_vals <- c(0, 10)
y_vals <- c(0, 0)
radi <- c(1, 3)
fills <- c("purple", "yellow")
circle_n <- 1:2

# Prep for your iteration
lst_circle_specs <-
  list(
    x_vals,
    y_vals,
    radi,
    fills,
    circle_n
  )
```

```
# Use `circle_data()` in your preferred iteration methods
two_circles <- pmap(lst_circle_specs, ~ circle_data(
  x = ..1,
  y = ..2,
  radi = ..3,
  fill = ..4,
  color = "#000000",
  group_var = TRUE
) |>
  # circle_data adds a `group` variable if `group_var` = TRUE.
  # For multiple circles, a unique identifier should be added/pasted in.
  mutate(group = paste0(group, ..5))) |>
  list_rbind()

# Plot the data

two_circles |>
  ggplot(aes(x, y, group = group)) +
  theme(legend.position = "none") +
  geom_polygon(
    color = two_circles$color,
    fill = two_circles$fill
  ) +
  coord_equal() #Always use coord_equal() or coord_fixed for circles!
```

`grid_maker` *Data Generation for A Custom-built Square Grid*

Description

Creates a dataframe of `x` and `y` points to visualize a square grid based on given `x` and `y` limits. Providing a color palette and fill style are optional.

Usage

```
grid_maker(
  xlim,
  ylim,
  size,
  fill_pal = NULL,
  fill_style = "range",
  color_pal = NULL,
  color_style = "range"
)
```

Arguments

`xlim`	A numeric vector with two X limits. A minimum and maximum limit for the X axis. Must be a length of 2.

ylim	A numeric vector with two Y limits. A minimum and maximum limit for the Y axis. Must be a length of 2.
size	A numeric input. The size of the grid. How many shapes will appear in a single row or column. Must be a length of 1, greater than 0, and less than or equal to the max xlim and max ylim.
fill_pal	Optional. A character vector of 6 digit hexadecimal webcolor code, or R colors() color strings to be applied to fill the grid.
fill_style	Optional. A character input. "range" or "random". Determines how the fill color palette is mapped.
color_pal	Optional. A character vector of 6 digit hexadecimal webcolor code, or R colors() color strings to be applied to borders of the grid.
color_style	Optional. A character input. "range" or "random". Determines how the border color palette is mapped.

Value

A Tibble

Examples

```
# Creating data for a grid:

library(ggplot2)
grid_data <- grid_maker(
  xlim = c(0, 50),
  ylim = c(0, 50),
  size = 10,
  fill_pal = c("turquoise", "black", "purple"),
  color_pal = c("black", "limegreen")
)

ggplot() +
  geom_polygon(
    data = grid_data,
    aes(x, y, group = group),
    fill = grid_data$fill,
    color = grid_data$color
  ) +
  coord_equal()
```

---

group_numbers *Convert Numbers into Padded Strings for Easier Group Numbering*

Description

Convert Numbers into Padded Strings for Easier Group Numbering

Usage

```
group_numbers(numbers, prefix = NULL, suffix = NULL, sep = NULL)
```

Arguments

numbers	A numeric vector with a length of at least 1.
prefix	A single string value that will be affixed in front of the numbers provided.
suffix	A single string value that will be affixed behind the numbers provided.
sep	A single string value that will be used to separate the prefix and/or suffix from the numbers provided. prefix or suffix is required to use sep.

Value

A Character Vector

Examples

```
# Useful for easier group numbering so groups are ordered as intended
# Expects a numeric vector of numbers to convert to padded numbers
regular_numbers <- 1:19
padded_numbers <- group_numbers(regular_numbers)

# The padding matters when creating labels for groupings
# as numbers will be converted to characters if attached to strings.
# Sorts as expected:
sort(regular_numbers)

# Does not as a character:
sort(paste0("group_", regular_numbers))

# Will sort as expected when padded:
sort(paste0("group_", padded_numbers))
```

---

group_sample *Sample Data Frames by a Group Variable*

Description

Sample Data Frames by a Group Variable

Usage

```
group_sample(
  data,
  group,
  n = 1,
  prop = NULL,
```

```
  prob = NULL,
  group_output = FALSE
)
```

Arguments

data	A data frame or tibble with at least 1 variable.
group	A variable in data that will be used for groupings.
n, prop	Supply either n, the number of groups, or prop, the proportion of groups to select. n must be a positive integer that is greater than or equal to 1. prop must be a positive numeric value that is greater than 0 and less than or equal to 1. Default is n = 1.
prob	Optional. A vector of probability weights for obtaining the elements of the group being sampled. Must be the same length as the total unique values in data's group variable.
group_output	A logical boolean TRUE or FALSE. If TRUE, returns a grouped tibble. Default is FALSE.

Value

A sampled dataframe

Examples

```
vec_coords <- 1:10
df_data <-
 data.frame(
   "x" = vec_coords,
   "y" = vec_coords,
   "group_col" = group_numbers(1:5) |> rep(each = 2)
 )

df_sampled_data_prop <-
 df_data |>
 group_sample(group_col, prop = .2)

df_sampled_data_prop

df_sampled_data_n <-
 df_data |>
 group_sample(group_col, n = 2)

df_sampled_data_n
```

group_slice *Subset Data Frames by a Group Variable Using Their Positions*

Description

Subset Data Frames by a Group Variable Using Their Positions

Usage

```
group_slice(
  data,
  group,
  n = 1,
  prop = NULL,
  position = "head",
  group_output = FALSE
)
```

Arguments

data	A data frame or tibble with at least 1 variable.
group	A variable in data that will be used for groupings.
n, prop	Supply either n, the number of groups, or prop, the proportion of groups to select. n must be a positive integer that is greater than or equal to 1. prop must be a positive numeric value that is greater than 0 and less than or equal to 1. Default is n = 1.
position	A character string of "head" or "tail". Determines if the first group or last group in the data frame is selected where "head" will select the first group in the dataframe and "tail" will select the last group.
group_output	A logical boolean TRUE or FALSE. If TRUE, returns a grouped tibble. Default is FALSE

Value

A sliced dataframe

Examples

```
vec_coords <- 1:10
df_data <-
 data.frame(
   "x" = vec_coords,
   "y" = vec_coords,
   "group_col" = group_numbers(1:5) |> rep(each = 2)
 )
```

```
df_sliced_data_head <-
 df_data |>
 group_slice(group_col, n = 2, position = "head")

df_sliced_data_head

df_sliced_data_tail <-
 df_data |>
 group_slice(group_col, n = 2, position = "tail")

df_sliced_data_tail

df_sliced_data_prop <-
 df_data |>
 group_slice(group_col, prop = .80)

df_sliced_data_prop
```

---

packer *Data Generation for Circle Packing*

Description

[Experimental]

A tool for creating a data frame of values that create a circle packing design when plotted. When the default `circle_type` "whole" is used, the output should mapped with `geom_polygon` in a ggplot. When "swirl" is used, the output should be mapped with `geom_path` for the best results.

Usage

```
packer(
  n,
  min_x = 0,
  max_x = 100,
  min_y = 0,
  max_y = 100,
  big_r = 5,
  med_r = 3,
  small_r = 1,
  color_pal = NULL,
  color_type = "regular",
  circle_type = "whole"
)
```

Arguments

n	The total number of circles you would like the function to attempt to create. A single numeric value with a minimum value of 10.
min_x	The minimum limit of the x-axis - the left 'border' of the canvas A single numeric value.
max_x	The maximum limit of the x-axis - the right 'border' of the canvas A single numeric value.
min_y	The minimum limit of the y-axis - the bottom 'border' of the canvas A single numeric value.
max_y	The maximum limit of the y-axis - the top 'border' of the canvas A single numeric value.
big_r	The radius used for your 'big' sized circles A single numeric value.
med_r	The radius used for your 'medium' sized circles. A single numeric value.
small_r	The radius used for your 'small' sized circles. A single numeric value.
color_pal	A vector of hex color codes that will be mapped to the data.
color_type	Default is "regular" - The colors will be mapped in order from big circles to small circles. "reverse" - The colors will be mapped in reversed order from small to big circles. "random" - The colors will be mapped randomly to any sized circle.
circle_type	Default is "whole" - Regular circles. "swirl" - circles are replaced with spirals. Spirals should be mapped with geom_path in a ggplot for the best results.

Value

A Tibble

Examples

```
library(ggplot2)
set.seed(0310)
packed_circles <- packer(
  n = 50, big_r = 5, med_r = 3, small_r = 1,
  min_x = 0, max_x = 100, min_y = 0, max_y = 100
)
packed_circles

packed_circles |>
  ggplot(aes(x, y, group = group)) +
  theme_void() +
  theme(plot.background = element_rect(fill = "black")) +
  geom_polygon(fill = "white", color = "red") +
  coord_equal()
```

point_in_polygon *Point in Polygon Test for Generative Art*

Description

[Experimental]

Tests whether points are inside a polygon using Cartesian coordinates. Distinguishes between points that are inside, outside, or on the polygon boundary.

Usage

```
point_in_polygon(point_x = NULL, point_y = NULL, poly_x = NULL, poly_y = NULL)
```

Arguments

point_x	Numeric vector of x coordinates for points to test
point_y	Numeric vector of y coordinates for points to test
poly_x	Numeric vector of x coordinates defining polygon vertices
poly_y	Numeric vector of y coordinates defining polygon vertices

Details

The function uses the sf package's geometry operations with Cartesian coordinates. If the provided polygon's first and last vertices are different, it is not considered to be a closed polygon. If this occurs, it will be automatically closed. All input vectors must be numeric, and point_x/point_y must have matching lengths, as must poly_x/poly_y.

Value

Integer vector: 0 = outside polygon, 1 = inside polygon -OR- on polygon boundary

Examples

```
# point_in_polygon can be used with basic vectors#
# X points we want to check the position of relevant to the polygon
point_x <- c(0.5, 1.5, 2.5, 1.0, 3.0)
# Y points we want to check the position of relevant to the polygon
point_y <- c(0.5, 1.5, 2.5, 0.0, 1.0)
# X points of the polygon we want to test against
poly_x <- c(0, 2, 2, 0, 0)
# Y points of the polygon we want to test against
poly_y <- c(0, 0, 2, 2, 0)

# 0 = outside polygon, 1 = inside polygon or on boundary
point_in_polygon(point_x, point_y, poly_x, poly_y)
```

```
# point_in_polygon can also be used within a data frame#
library(dplyr)

test_data <- data.frame(px = c(0.5, 1.5, 2.5), py = c(0.5, 1.5, 2.5))
polygon_x <- c(0, 2, 2, 0, 0)
polygon_y <- c(0, 0, 2, 2, 0)

test_data |>
  mutate(logic = point_in_polygon(px, py, polygon_x, polygon_y))

# You can also see the results of the function visualized on a ggplot#
library(ggplot2)

# Create test points and polygon for visualization
df_points_prep <-
  tibble(
    x = c(0.5, 1.5, 2.5, 1.0, 0.0, 3.0),
    y = c(0.5, 1.5, 2.5, 0.0, 1.0, 1.0)
  )
df_polygon <-
  tibble(
    x = c(0, 2, 2, 0, 0),
    y = c(0, 0, 2, 2, 0)
  )

# Test the points and add labels for the plot
df_points <-
  df_points_prep |>
  mutate(
    position = point_in_polygon(x, y, df_polygon$x, df_polygon$y),
    position_string = case_match(position,
                                 0 ~ "Outside",
                                 1 ~ "Inside/On Boundary"
                                 ) |> factor(),
    color = case_match(position,
                       0 ~ "#e31a1c",
                       1 ~ "#33a02c"
                       )
  )

# Pull out the colors for the plot points
unique_df <- unique(df_points[c("position_string", "color")])
vec_colors <- setNames(unique_df$color, unique_df$position_string)

# Plot it
ggplot() +
  geom_polygon(data = df_polygon, aes(x = x, y = y),
               fill = "#104d70", alpha = 0.3, color = "black", linewidth = 1) +
  geom_point(data = df_points, aes(x = x, y = y, color = position_string), size = 4) +
  scale_color_manual(values = vec_colors) +
  labs(color = "Position:") +
  coord_equal()
```

`resizer` *Transforms and Scales Numeric Points in a Data Frame by a Provided Factor and Direction*

Description

Transforms and Scales Numeric Points in a Data Frame by a Provided Factor and Direction

Usage

```
resizer(
  data = NULL,
  x,
  y,
  x_anchor = NULL,
  y_anchor = NULL,
  factor = NULL,
  direction = "up",
  drop = FALSE,
  ...
)
```

Arguments

`data`	A data frame or tibble with at least `x` and `y` variables.
`x`	A numeric variable in `data`. The variable intended to be plotted on the x axis in a `ggplot`.
`y`	A numeric variable in `data`. The variable intended to be plotted on the y axis in a `ggplot`.
`x_anchor`	A numeric value. The `x` coordinate point that the resized polygon will be scaled and anchored from. Default is the first `x` value in `data`.
`y_anchor`	A numeric value. The `y` coordinate that the resized polygon will be scaled and anchored from. Default is the first `y` value in `data`.
`factor`	A numeric value. The factor that will be used to resize the existing polygon in `data`.
`direction`	A string value of either `"up"` or `"down`. Data that is scaled `"up"` (default) will increase in size when plotted. Data that is scaled `"down"` will decrease in size.
`drop`	Logical `TRUE` or `FALSE` that determines if all other variables that are not being resized are removed from the final output. Default is `FALSE`.
`...`	Additional arguments passed to methods. Currently unused but reserved for future extensibility.

Value

A data frame

Examples

```
library(ggplot2)

# Resize a simple square "up" by a factor of 6
# Start with data that makes a shape#
df_square <-
  data.frame(
    x = c(0,1,1,0,0),
    y = c(0,0,1,1,0)
  )

# Resize the shape#
df_square_resized <-
  df_square |>
  resizer(x, y, factor = 6)

# Plot them
df_square |>
  ggplot(aes(x,y)) +
  # resized square - red dashed line
  geom_path(data = df_square_resized, color = "#a83246", linewidth = 2, linetype = 2) +
  # original square - black solid line
  geom_path(color = "#000000", linewidth = .8) +
  coord_equal()

# Resize a circle "down" by a factor of 3
df_circle <-
  circle_data(x = 5, y = 5, radius = 5, group_var = TRUE)

# Set then anchor point as the middle of the circle c(5,5)
# Although the point 5,5 is in the circle's bounds
# it's not actually a row in `df_circle`
# A message will display in cases like these and is "fine" to ignore.

df_circle_resized <-
  df_circle |>
  resizer(x,y, x_anchor = 5, y_anchor = 5, direction = "down", factor = 3)

# Plot them
df_circle |>
  ggplot(aes(x,y)) +
  # resized square - red dashed line
  geom_path(data = df_circle_resized, color = "#a83246", linewidth = 2, linetype = 2) +
  # original square - black solid line
  geom_path(color = "#000000", linewidth = .8) +
  coord_equal()
```

rotator *Rotate Points in a Data Frame Based on an Anchor Point*

Description

Rotates the x and y points in a given data frame by a given angle based on a designated anchor point.

Usage

```
rotator(data, x, y, angle = 5, anchor = "center", drop = FALSE)
```

Arguments

data	A data frame or tibble with at least x and y variables
x	A numeric variable in data. The variable intended to be plotted on the x axis in a ggplot.
y	A numeric variable in data. The variable intended to be plotted on the y axis in a ggplot.
angle	The angle (in degrees) the points in data will be rotated around it's anchor
anchor	The anchor point for the rotation. Default is "center". Options include:"center", "bottom", "top", "left", and "right"
drop	Logical TRUE or FALSE that determines if all other variables that are not being rotated are removed from the final output. Default is FALSE.

Value

A data frame

Examples

```
library(ggplot2)
original_square <- data.frame(
  x = c(0, 3, 3, 0, 0),
  y = c(0, 0, 3, 3, 0)
)
rotated_square <- rotator(data = original_square,
                          x = x,
                          y = y,
                          angle = 45,
                          anchor = "center")

ggplot()+
  geom_path(data = original_square,
                     aes(x,y),
                     color = "red")+
```

```
  geom_polygon(data = rotated_square,
                       aes(x,y),
                       fill = "purple")+
  coord_equal()
```

---

seq_bounce *Bouncing Sequence Generation*

Description

Generate a regular sequence that 'bounces' between the provided `start_n` and `end_n` values in increments by the `by` value for the length of the `length` value provided.

Usage

```
seq_bounce(start_n = NULL, end_n = NULL, length = NULL, by = 1)
```

Arguments

`start_n`	The lower (min) numeric bound of the sequence to be generated. Must be `< end_n`.
`end_n`	The upper (max) numeric bound of the sequence to be generated. Must be `> start_n`.
`length`	The desired length of the generated sequence.
`by`	The number increment of the sequence.

Value

A numeric vector

Examples

```
#By default, seq_bounce creates sequences by increments of 1
#The length argument accepts any positive integer
seq_bounce(start_n = 1, end_n = 5, length = 15)

#The by argument accepts any positive numeric
seq_bounce(start_n = 0, end_n = 10, length = 30, by = .247)

#The end_n value must be greater than the start_n value
#This will give you an error
try(seq_bounce(start_n = 0, end_n = -10, length = 15))

#Instead, reverse the values
seq_bounce(start_n = -10, end_n = 0, length = 15)
```

set_brightness *Change the brightness of a hexadecimal color value*

Description

[Experimental]

Usage

```
set_brightness(color = NULL, percentage = NULL)
```

Arguments

color	Character value of length 1 - The color that will have its brightness set. A standard 6 digit hexadecimal webcolor like "#000000" or a valid R color from colors() is accepted.
percentage	A numeric value of length 1. The percentage of which the brightness should be set. Values from 0 - 1 are accepted.

Value

A character string (hexadecimal color)

Examples

```
# Load in ggplot2 so we can see the colors
library(ggplot2)

# Create color values
original_color <- "#7755aa" #(original brightness == .5)
darker_color <- set_brightness(original_color, .3) #(brightness == %30)
lighter_color <- set_brightness(original_color, .7) #(brightness == %70)

# Make a data frame with the color values
df_colors <-
  data.frame(
    x = 0:2,
    y = 1,
    color = c(darker_color, original_color, lighter_color)
  )

# Add a label for clarity
df_colors$label <- paste(c("Darker", "Original", "Lighter"), ":", df_colors$color)

# Plot to see the brightness changes
df_colors |>
  ggplot(aes(x,y)) +
```

```
  geom_label(aes(x = 0:2), y = 2, label = df_colors$label) +
  geom_point(color = df_colors$color, shape = 15, size = 50) +
  coord_cartesian(xlim = c(-1,3), ylim = c(0,3)) +
  theme_void()
```

`set_saturation` *Change the saturation of a hexadecimal color value*

Description

[Experimental]

Usage

```
set_saturation(color = NULL, percentage = NULL)
```

Arguments

`color`	Character value of length 1 - The color that will have its saturation set. A standard 6 digit hexadecimal webcolor like "#000000" or a valid `R` color from `colors()` is accepted.
`percentage`	A numeric value of length 1. The percentage of which the saturation should be set. Values from 0 - 1 are accepted.

Value

A character string (hexadecimal color)

Examples

```
# Load in ggplot2 so we can see the colors
library(ggplot2)

# Create color values
original_color <- "#7340bf" #(original saturation == .5)
desaturated_color <- set_saturation(original_color, .2) #(saturation == %20)
saturated_color <- set_saturation(original_color, .9) #(saturation == %90)

# Make a data frame with the color values
df_colors <-
  data.frame(
    x = 0:2,
    y = 1,
    color = c(desaturated_color, original_color, saturated_color)
  )
```

```
# Add a label for clarity
df_colors$label <- paste(c("Desaturated", "Original", "Saturated"), ":", df_colors$color)

# Plot to see the saturation changes
df_colors |>
  ggplot(aes(x,y)) +
  geom_label(aes(x = 0:2), y = 2, label = df_colors$label) +
  geom_point(color = df_colors$color, shape = 15, size = 50) +
  coord_cartesian(xlim = c(-1,3), ylim = c(0,3)) +
  theme_void()
```

`square_data` *Data Generation for Squares*

Description

A tool for creating a data frame of values that create a square with a specified size when plotted.

The `geom_path` and `geom_polygon` geoms are recommended with this data for use in `ggplot2` for generative art.

Usage

```
square_data(
  x,
  y,
  size,
  color = NULL,
  fill = NULL,
  n_points = 100,
  group_var = FALSE,
  group_prefix = "square_"
)
```

Arguments

`x`	Numeric value of length 1 - The bottom left `x` value of the square.
`y`	Numeric value of length 1 - The bottom left `y` value of the square.
`size`	Numeric value of length 1 that must be greater than 0 - The size of the square.
`color`	Character value of length 1 - The color of the square's border. A valid `R` color from `colors()` or a standard 6 digit hexadecimal webcolor like "#000000"
`fill`	Character value of length 1 - The color of the square. A valid `R` color from `colors()` or a standard 6 digit hexadecimal webcolor like "#000000"

n_points	Numeric value. Default is 100. This determines how many points the square will have. This option can come in handy when using jitter options or other texture/illusion methods. Must be of length 1 and at least a value of 4.
group_var	Logical. Default is FALSE. If TRUE, a group variable will be added to the dataframe. Useful in iterative data generation.
group_prefix	Character string of length 1 - The prefix used for the group variable. Default is "square_"

Value

A Tibble

Examples

```
# Creating one square
library(ggplot2)
one_square <- square_data(x = 0, y = 0, size = 5)

# Plot The Data
one_square |>
  ggplot(aes(x,y))+
  geom_path(color = "green")+
  coord_equal()

# To create multiple squares, use your preferred method of iteration:
# Creating two squares

library(purrr)
library(dplyr)

# Make your specs
x_vals <- c(0,4)
y_vals <- c(0,0)
sizes <- c(1,3)
fills <- c("purple", "yellow")
square_n <- 1:2

# Prep for your iteration
lst_square_specs <-
  list(
    x_vals,
    y_vals,
    sizes,
    fills,
    square_n
  )

# Use `square_data()` in your preferred iteration methods
two_squares <- pmap(lst_square_specs, ~square_data(
  x = ..1,
  y = ..2,
```

```
  size = ..3,
  fill = ..4,
  color = "#000000",
  group_var = TRUE
) |>
  # square_data adds a `group` variable if `group_var` = TRUE.
  # For multiple squares, a unique identifier should be added/pasted in.
  mutate(group = paste0(group,..5))
) |>
  list_rbind()

# Plot the data

two_squares |>
  ggplot(aes(x, y, group = group))+
  theme(legend.position = "none")+
  geom_polygon(color = two_squares$color,
               fill = two_squares$fill) +
  coord_equal()
```

---

wave_data *Data Generation for 2D Sine and Cosine Waves*

Description

[Experimental]

A tool for making data frames filled with data that displays sine or cosine waves when graphed.

The geom_path and geom_polygon geoms are recommended with this data for use in ggplot2 for generative art.

Usage

```
wave_data(
  start,
  end,
  size = 1,
  type = "sin",
  orientation = "horizontal",
  freq = 3,
  n_points = 500,
  color = NULL,
  fill = NULL,
  group_var = FALSE,
  dampen = NULL,
  amplify = NULL
)
```

Arguments

start	Numeric value. The starting point of the wave on the coordinate system. By default refers to the x-axis. Will refer to the y-axis if orientation is set to vertical. Must be of length 1.
end	Numeric value. The ending point of the wave on the coordinate system. By default refers to the x-axis. Will refer to the y-axis if orientation is set to vertical Must be of length 1.
size	Numeric value. The height or width of the wave. Orientation is set to horizontal by default, thus size will affect height by default. When orientation is set to vertical, size controls the width of the wave. Must be a positive numeric value. Must be of length 1.
type	String value. "sin" or "cos" for sine or cosine waves. sin is default. Must be of length 1.
orientation	String value. Default is horizontal which will draw the wave from left to right (x-axis) on the coordinate system. vertical will draw the wave from bottom to top (y-axis) on the coordinate system. Must be of length 1.
freq	Numeric value. Default is 3 cycles per second. This affects how many "peaks" are created in the wave. Must be a positive numeric value. Must be of length 1.
n_points	Numeric value. Default is 500. This determines how many points each half of the wave will have. This option can come in handy when using jitter options or other texture/illusion methods. Must be of length 1.
color	Optional String Value. A 6 digit hexadecimal webcolor code, or R colors() color string for the border color of the wave. Must be of length 1.
fill	Optional String Value. A 6 digit hexadecimal webcolor code, or R colors() color string for the fill color of the wave. Must be of length 1.
group_var	Logic value. TRUE or FALSE. Default is FALSE. If TRUE, Adds a group variable to the data frame. Useful for iterative work to make multiple waves in a single data frame.
dampen	Optional. A factor in which to dampen the wave (make "flatter"). Must be of length 1.
amplify	Optional. A factor in which to amplify the wave (make "sharper"). Must be of length 1.

Value

A Tibble

Examples

```
library(ggplot2)
wave_df <- wave_data(
  start = 0, end = 10,
  fill = "purple",
  color = "green"
```

```
)

wave_df |>
  ggplot(aes(x, y)) +
  theme_void() +
  geom_polygon(
    fill = wave_df$fill,
    color = wave_df$color,
    linewidth = 3
  ) +
  coord_equal()
```

Index