Package 'artpack'

March 19, 2023

Title Provides a comprehensive tool set for creating Rtistry

Version 0.0.0.9000

Description aRtpack is a package that contains a comprehensive tool set for aRtists of all skill levels. aRtpack seeks to make life easier for more seasoned aRtists by providing helper functions to save time on more complex pieces. aRtpack also seeks to introduce more beginners into Rtistry by providing quick and easy ways to create aRt right away.

License MIT + file LICENSE

Encoding UTF-8

Roxygen list(markdown = TRUE)

RoxygenNote 7.2.3

Depends ggplot2 (>= 3.4.1)

Suggests testthat (>= 3.0.0),

Config/testthat/edition 3

Imports purrr (>= 0.3.4),
tibble (>= 3.1.6),
dplyr (>= 1.0.8),
grDevices (>= 4.1.1)

URL `https://meghansaha.github.io/artpack/`

R topics documented:

`art_pals` *Custom-built aRtpack Color Palettes*

Description

The aRtpack palette picker. The `art_pals` function consists of 17 palettes: "arctic", "beach", "bw", "brood", "cosmos", "explorer", "gemstones", "grays", "icecream", "imagination", "majestic", "nature", "neon", "ocean", "plants", "rainbow", "sunnyside"

Usage

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

Arguments

`pal`	A character string of the desired aRtpack palette. The 17 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
`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"

Value

A character vector.

Examples

```
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)
```

`circle_packer` *Data Generation for Circle Packing*

Description

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

```
circle_packer(
  n,
  min_x = 0,
  max_x = 10,
  min_y = 0,
  max_y = 10,
  big_r = 5,
  med_r = 2,
  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.
`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_polygon` in a ggplot for the best results.

Value

A Tibble

Examples

```
packed_circles <- circle_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()
```

`grid_maker` *Create Data 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, pal = NULL, fill_style = "range")
```

Arguments

`xlim`	A numeric vector with two X limits. A minimum and maximum limit for the X axis.
`ylim`	A numeric vector with two Y limits. A minimum and maximum limit for the Y axis.
`size`	A numeric input. The size of the grid. How many shapes will appear in a single row or column.
`pal`	Optional. A character vector of color codes to be applied to the grid.
`fill_style`	Optional. A character input. "range" or "random". Determines how the color palette is mapped.

Value

a tibble

Examples

```
grid_data <- grid_maker(xlim = c(0,1),
                       ylim = c(0,1),
                       size = 2,
                       pal = c("red", "black", "purple"))

ggplot()+
  geom_polygon(data = grid_data,
                       aes(x,y, group = group),
                       fill = grid_data$fill)+
  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 = NULL, angle = 5, anchor = "center")
```

Arguments

`data`	A data frame or tibble with at least `x` and `y` variables
`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"

Value

#A data frame

Examples

```
original_square <- data.frame(x = c(0,3,3,0,0),
                              y = c(0,0,3,3,0))
rotated_square <- rotator(data = original_square,
                          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()
```

`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.

Usage

```
square_data(x, y, size, group_var = FALSE)
```

Arguments

`x`	Numeric - The bottom left `x` value of the square.
`y`	Numeric - The bottom left `y` value of the square.
`size`	Numeric - The size of the square. Must be a value greater than zero.
`group_var`	`TRUE`/`FALSE`. Default is `FALSE`. If `TRUE`, adds a grouping variable to the data frame. Default is switched to `TRUE` when more than one `x`, `y`, and `size` values are present.

Value

A Tibble

Examples

```
one_square <- square_data(x = 0, y = 0, size = 5)

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

x_vals <- c(0,4)
y_vals <- c(0,0)
sizes <- c(1,3)
sq_cols <- c("purple", "yellow")

two_squares <- square_data(x = x_vals, y = y_vals, size = sizes)

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

Index