**# DO NOT ERASE. COMPARE WITH BAR MEAN BEFORE AND RECOVER WHAT HAS BEEN MODIFIED**

**fun\_gg\_line <- function(data1, y, categ, categ.class.order = NULL, categ.legend.name = NULL, categ.color = NULL, line.size = 1, error.disp = NULL, error.whisker.width = 0.5, dot.color = "same", dot.tidy = FALSE, dot.bin.nb = 30, dot.jitter = 0.25, dot.size = 3, dot.border.size = 0.5, dot.alpha = 0.5, ylim = NULL, ylog = FALSE, y.tick.nb = NULL, y.include.zero = FALSE, y.top.extra.margin = 0.05, y.bottom.extra.margin = 0, stat.disp = NULL, stat.size = 4, stat.dist = 2, xlab = NULL, ylab = NULL, vertical = TRUE, title = "", text.size = 12, text.angle = 0, classic = FALSE, grid = FALSE, return = FALSE, lib.path = NULL){**

*# AIM*

*# ggplot2 vertical barplot representing mean values with the possibility to add error bars and to overlay dots*

*# for ggplot2 specifications, see: https://ggplot2.tidyverse.org/articles/ggplot2-specs.html*

*# WARNINGS*

*# rows containing NA in data1[, c(y, categ)] will be removed before processing, with a warning (see below)*

*# if ever bars disappear, see the end of https://github.com/tidyverse/ggplot2/issues/2887*

*# to have a single bar, create a factor column with a single class and specify the name of this column in categ argument as unique element (no categ2 in categ argument). For a single set of grouped bars, create a factor column with a single class and specify this column in categ argument as first element (categ1). See categ below*

*# with several single bars (categ argument with only one element), bar.width argument (i.e., width argument of ggplot2::geom\_bar()) defines each bar width. The bar.width argument also defines the space between bars by using (1 - bar.width). In addition, xmin and xmax of the fun\_gg\_bar() output report the bar boundaries (around x-axis unit 1, 2, 3, etc., for each bar)*

*# with several sets of grouped bars (categ argument with two elements), bar.width argument defines each set of grouped bar width. The bar.width argument also defines the space between set of grouped bars by using (1 - bar.width). In addition, xmin and xmax of the fun\_gg\_bar() output report the bar boundaries (around x-axis unit 1, 2, 3, etc., for each set of grouped bar)*

*# to manually change the 0 base bar into this code, see https://stackoverflow.com/questions/35324892/ggplot2-setting-geom-bar-baseline-to-1-instead-of-zero*

*# ARGUMENTS*

*# data1: a dataframe containing one column of values (see y argument below) and one or two columns of categories (see categ argument below). Duplicated column names not allowed*

*# y: character string of the data1 column name for y-axis (containing numeric values). Numeric values will be averaged by categ to generate the bars and will also be used to plot the dots*

*# categ: vector of character strings of the data1 column name for categories (column of characters or factor). Must either be one or two column names. If a single column name (further refered to as categ1), then one bar per class of categ1. If two column names (further refered to as categ1 and categ2), then one bar per class of categ2, which form a group of bars in each class of categ1. BEWARE, categ1 (and categ2 if it exists) must have a single value of y per class of categ1 (and categ2). To have a single bar, create a factor column with a single class and specify the name of this column in categ argument as unique element (no categ2 in categ argument). For a single set of grouped bars, create a factor column with a single class and specify this column in categ argument as first element (categ1)*

*# categ.class.order: list indicating the order of the classes of categ1 and categ2 represented on the barplot (the first compartment for categ1 and and the second for categ2). If categ.class.order = NULL, classes are represented according to the alphabetical order. Some compartment can be NULL and other not*

*# categ.legend.name: character string of the legend title for categ2. If categ.legend.name = NULL, then categ.legend.name <- categ1 if only categ1 is present and categ.legend.name <- categ2 if categ1 and categ2 are present. Write "" if no legend required*

*# categ.color: vector of character color string for bar filling. If categ.color = NULL, default colors of ggplot2, whatever categ1 and categ2. If categ.color is non null and only categ1 in categ argument, categ.color can be either: (1) a single color string (all the bars will have this color, whatever the classes of categ1), (2) a vector of string colors, one for each class of categ1 (each color will be associated according to categ.class.order of categ1), (3) a vector or factor of string colors, like if it was one of the column of data1 data frame (beware: a single color per class of categ1 and a single class of categ1 per color must be respected). Integers are also accepted instead of character strings, as long as above rules about length are respected. Integers will be processed by fun\_gg\_palette() using the max integer value among all the integers in categ.color. If categ.color is non null and categ1 and categ2 specified, all the rules described above will apply to categ2 instead of categ1 (colors will be determined for bars inside a group of bars)*

*# bar.width: numeric value (from 0 to 1) of the bar or set of grouped bar width (see WARNINGS above)*

*# error.disp: either "SD", "SD.TOP", "SEM" or "SEM.TOP". If NULL, no error bars added*

*# error.whisker.width: numeric value (from 0 to 1) of the whisker (error bar extremities) width, with 0 meaning no whiskers and 1 meaning a width equal to the corresponding bar width*

*# dot.color: vector of character string. Idem as categ.color but for dots, except that in the possibility (3), the rule "a single color per class of categ1 and a single class of categ1", cannot be respected (each dot can have a different color). If NULL, no dots plotted*

*# dot.tidy: logical. Nice dot spreading? If TRUE, use the geom\_dotplot() function for a nice representation. If FALSE, dots are randomly spread, using the dot.jitter argument (see below)*

*# dot.bin.nb: positive integer indicating the number of bins (i.e., nb of separations) of the ylim range. Each dot will then be put in one of the bin, with the size the width of the bin. Not considered if dot.tidy is FALSE*

*# dot.jitter: numeric value (from 0 to 1) of random dot horizontal dispersion, with 0 meaning no dispersion and 1 meaning a dispersion in the corresponding bar width interval. Not considered if dot.tidy is TRUE*

*# dot.size: numeric value of dot size. Not considered if dot.tidy is TRUE*

*# dot.border.size: numeric value of border dot size. Write zero for no dot border. If dot.tidy is TRUE, value 0 remove the border. Another one leave the border without size control (geom\_doplot() feature)*

*# dot.alpha: numeric value (from 0 to 1) of dot transparency (full transparent to full opaque, respectively)*

*# ylim: 2 numeric values for y-axis range. If NULL, range of y in data1*

*# ylog: logical. Log scale for the y-axis? BEWARE: do not tranform the data, but just display ticks in a log scale manner. BEWARE: if TRUE, ylim must not contain null or negative values. In addition, will be automatically set to FALSE if vertical argument is set to FALSE, to prevent a bug in ggplot2 (see https://github.com/tidyverse/ggplot2/issues/881)*

*# y.tick.nb: number of desired values on the y-axis*

*# y.include.zero: logical. Does ylim range include 0? BEWARE: if ylog = TRUE, will be automately set to FALSE with a warning message*

*# y.top.extra.margin: single proportion (between 0 and 1) indicating if extra margins must be added to ylim. If different from 0, add the range of the axis \* y.top.extra.margin (e.g., abs(ylim[2] - ylim[1]) \* y.top.extra.margin) to the top of y-axis. BEWARE with ylog = TRUE, the range result must not overlap zero or negative values*

*# y.bottom.extra.margin: idem as y.top.extra.margin but to the bottom of y-axis*

*# stat.disp: add the mean number above the corresponding bar. Either NULL (no number shown), "top" (at the top of the figure region) or "above" (above each bar)*

*# stat.size: numeric value of the stat size (in points). Increase the value to increase text size*

*# stat.dist: numeric value of the stat distance. Increase the value to increase the distance*

*# xlab: a character string for x-axis legend. If NULL, character string of categ1*

*# ylab: a character string y-axis legend. If NULL, character string of the y argument*

*# vertical: logical. Vertical bars? BEWARE: cannot have horizontal bars with a log axis, i.e., ylog = TRUE & vertical = FALSE (see ylog above)*

*# title: character string of the graph title*

*# text.size: numeric value of the text size (in points)*

*# text.angle: integer value of the text angle for the x-axis labels. Positive values for counterclockwise rotation: 0 for horizontal, 90 for vertical, 180 for upside down etc. Negative values for clockwise rotation: 0 for horizontal, -90 for vertical, -180 for upside down etc.*

*# classic: logical. Use the classic theme (article like)?*

*# grid: logical. draw horizontal lines in the background to better read the bar values? Not considered if classic = FALSE*

*# return: logical. Return the graph parameters?*

*# lib.path: absolute path of the required packages, if not in the default folders*

**}**