

**# Check NOT DONE**

**fun\_gg\_bar <- function(data1, y, categ, categ.class.order = NULL, categ.legend.name = NULL, categ.color = NULL, bar.width = 0.5, 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 = "no", y.tick.nb = NULL, y.inter.tick.nb = NULL, y.include.zero = FALSE, y.top.extra.margin = 0, y.bottom.extra.margin = 0, stat.disp = NULL, stat.size = 4, stat.dist = 2, xlab = NULL, ylab = NULL, vertical = TRUE, text.size = 12, title = "", title.text.size = 8, text.angle = 0, classic = FALSE, grid = FALSE, return = FALSE, plot = TRUE, add = NULL, warn.print = 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. Order of the 2 values matters (for inverted axis). BEWARE: values of the ylim must be already in the corresponding log if ylog argument is not "no" (see below)*

*# ylog: Either "no" (values in the y argument column of the data1 data frame are not log), "log2" (values in the y argument column of the data1 data frame are log2 transformed) or "log10" (values in the y argument column of the data1 data frame are log10 transformed). BEWARE: do not tranform the data, but just display ticks in a log scale manner. Thus, negative or zero values allowed. BEWARE: not possible to have horizontal bars with a log axis, due to a bug in ggplot2 (see https://github.com/tidyverse/ggplot2/issues/881)*

*# y.tick.nb: approximate number of desired label values on the y-axis (n argument of the the fun\_scale() function)*

*# y.inter.tick.nb: number of desired secondary ticks between main ticks. Not considered if ylog is other than "no". In that case, play with the ylim and y.tick.nb arguments*

*# y.include.zero: logical. Does ylim range include 0? Ok even if ylog = TRUE because ylim must already be log transformed values*

*# 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*

*# 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 or expression for x-axis legend. If NULL, character string of categ1*

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

*# vertical: logical. Vertical bars? BEWARE: will be automatically set to TRUE if ylog argument is other than "no". Indeed, not possible to have horizontal bars with a log axis, due to a bug in ggplot2 (see https://github.com/tidyverse/ggplot2/issues/881)*

*# text.size: numeric value of the size of the (1) axis numbers and axis legends, (2) texts in the graphic legend, (3) stats above bars (in points)*

*# title: character string of the graph title*

*# title.text.size: numeric value of the title 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?*

*# plot: logical. Plot the graphic? If FALSE and return argument is TRUE, graphical parameters and associated warnings are provided without plotting*

*# add: character string allowing to add more ggplot2 features (dots, lines, themes, etc.). BEWARE: (1) must start with "+" just after the simple or double opening quote (no space, end of line, carriage return, etc., allowed), (2) must finish with ")" just before the simple or double closing quote (no space, end of line, carriage return, etc., allowed) and (3) each function must be preceded by "ggplot2::" (for instance: "ggplot2::coord\_flip()). If the character string contains the "ggplot2::theme" string, then internal ggplot2 theme() and theme\_classic() functions will be inactivated to be reused by add. BEWARE: handle this argument with caution since added functions can create conflicts with the preexisting internal ggplot2 functions*

*# warn.print: logical. Print warnings at the end of the execution? No print if no warning messages*

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

*# REQUIRED PACKAGES*

*# ggplot2*

*# scales*

*# REQUIRED FUNCTIONS FROM CUTE\_LITTLE\_R\_FUNCTION*

*# fun\_comp\_2d()*

*# fun\_gg\_just()*

*# fun\_gg\_palette()*

*# fun\_name\_change()*

*# fun\_pack()*

*# fun\_check()*

*# fun\_round()*

*# fun\_scale()*

*# RETURN*

*# a barplot if plot argument is TRUE*

*# a list of the graph info if return argument is TRUE:*

*# $stat: the graphic statistics*

*# $removed.row.nb: which rows have been removed due to NA detection in y and categ columns (NULL if no row removed)*

*# $removed.rows: removed rows containing NA (NULL if no row removed)*

*# $data: the graphic bar and dot coordinates*

*# $axes: the x-axis and y-axis info*

*# $warn: the warning messages. Use cat() for proper display. NULL if no warning*

*# EXAMPLES*

*### nice representation (1)*

*# obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), categ.class.order = list(NULL, c("B", "A")), categ.legend.name = "LEGEND", categ.color = NULL, bar.width = 0.3, error.disp = "SD.TOP", error.whisker.width = 0.8, dot.color = "same", dot.jitter = 0.5, dot.size = 3.5, dot.border.size = 0.2, dot.alpha = 0.5, ylim = c(10, 25), y.include.zero = TRUE, stat.disp = "above", stat.size = 4, xlab = "GROUP", ylab = "VALUE", text.size = 12, title = "GRAPH1", title.text.size = 8, text.angle = 0, classic = TRUE, grid = TRUE)*

*### nice representation (2)*

*# set.seed(1) ; obs1 <- data.frame(Time = c(rnorm(24, 0), rnorm(24, -10), rnorm(24, 10), rnorm(24, 20)), Group1 = rep(c("CAT", "DOG"), times = 48), Group2 = rep(c("A", "B", "C", "D"), each = 24)) ; set.seed(NULL) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), categ.class.order = list(NULL, c("B", "A", "D", "C")), categ.legend.name = "LEGEND", categ.color = NULL, bar.width = 0.8, dot.color = "grey50", dot.tidy = TRUE, dot.bin.nb = 60, dot.size = 3.5, dot.border.size = 0.2, dot.alpha = 0.5, ylim= c(-20, 30), stat.disp = "above", stat.size = 4, stat.dist = 1, xlab = "GROUP", ylab = "VALUE", vertical = FALSE, text.size = 12, title = "GRAPH1", title.text.size = 8, text.angle = 45, classic = FALSE)*

*### simple example*

*# obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = "Group1")*

*### separate bars. Example (1) of modification of bar color using a single value*

*# obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = "Group1", categ.color = "white")*

*### separate bars. Example (2) of modification of bar color using one value par class of categ2*

*# obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = "Group1", categ.color = c("coral", "lightblue"))*

*### separate bars. Example (3) of modification of bar color using the bar.color data frame column, with respect of the correspondence between categ2 and bar.color columns*

*# obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10), bar.color = rep(c("coral", "lightblue"), time = 10)) ; obs1 ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = "Group1", categ.color = obs1$bar.color)*

*### separate bars. Example (1) of modification of dot color, using the same dot color as the corresponding bar*

*# obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = "Group1", dot.color = "same")*

*### separate bars. Example (2) of modification of dot color, using a single color for all the dots*

*# obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = "Group1", dot.color = "green")*

*### separate bars. Example (3) of modification of dot color, using one value par class of categ2*

*# obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = "Group1", dot.color = c("green", "brown"))*

*### separate bars. Example (4) of modification of dot color, using different colors for each dot*

*# obs1 <- data.frame(Time = 1:10, Group1 = rep(c("G", "H"), times = 5)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = "Group1", dot.color = hsv(h = (1:nrow(obs1)) / nrow(obs1)))*

*### grouped bars. Simple example*

*# obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"))*

*### grouped bars. More grouped bars*

*# obs1 <- data.frame(Time = 1:24, Group1 = rep(c("G", "H"), times = 12), Group2 = rep(c("A", "B", "C", "D"), each = 6)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"))*

*### grouped bars. Example (1) of modification of bar color, using a single value*

*# obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), categ.color = "white")*

*### grouped bars. Example (2) of modification of bar color, using one value par class of categ2*

*# obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), categ.color = c("coral", "lightblue"))*

*### grouped bars. Example (3) of modification of bar color, using one value per line of obs1, with respect of the correspondence between categ2 and bar.color columns*

*# obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10), bar.color = rep(c("coral", "lightblue"), each = 10)) ; obs1 ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), categ.color = obs1$bar.color)*

*### grouped bars. Example (1) of modification of dot color, using the same dot color as the corresponding bar*

*# obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), dot.color = "same")*

*### grouped bars. Example (2) of modification of dot color, using a single color for all the dots*

*# obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), dot.color = "green")*

*### grouped bars. Example (3) of modification of dot color, using one value par class of categ2*

*# obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), dot.color = c("green", "brown"))*

*### grouped bars. Example (4) of modification of dot color, using different colors for each dot*

*# obs1 <- data.frame(Time = 1:10, Group1 = rep(c("G", "H"), times = 5), Group2 = rep(c("A", "B"), each = 5)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), dot.color = hsv(h = (1:nrow(obs1)) / nrow(obs1)))*

*### no dots (y.include.zero set to TRUE to see the lowest bar):*

*# obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), dot.color = NULL, y.include.zero = TRUE)*

*### bar width. Example (1) with bar.width = 0.25 -> three times more space between single bars than the bar width (y.include.zero set to TRUE to see the lowest bar)*

*# obs1 <- data.frame(Time = 1:1000, Group1 = rep(c("G", "H"), each = 500)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = "Group1", dot.color = NULL, y.include.zero = TRUE, bar.width = 0.25)*

*### bar width. Example (2) with bar.width = 1, no space between single bars*

*# obs1 <- data.frame(Time = 1:1000, Group1 = rep(c("G", "H"), each = 500)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = "Group1", dot.color = NULL, y.include.zero = TRUE, bar.width = 1)*

*### bar width. Example (3) with bar.width = 0.25 -> three times more space between sets of grouped bars than the set width*

*# obs1 <- data.frame(Time = 1:1000, Group1 = rep(c("G", "H"), times = 500), Group2 = rep(LETTERS[1:5], each = 200)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), dot.color = NULL, y.include.zero = TRUE, bar.width = 0.25)*

*### bar width. Example (4) with bar.width = 0 -> no space between sets of grouped bars*

*# obs1 <- data.frame(Time = 1:1000, Group1 = rep(c("G", "H"), times = 500), Group2 = rep(LETTERS[1:5], each = 200)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), dot.color = NULL, y.include.zero = TRUE, bar.width = 1)*

*### error bars*

*# obs1 <- data.frame(Time = 1:1000, Group1 = rep(c("G", "H"), times = 500), Group2 = rep(LETTERS[1:5], each = 200)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), dot.color = NULL, error.disp = "SD.TOP")*

*### whisker width. Example (1) with error.whisker.width = 1 -> whiskers have the width of the corresponding bar*

*# obs1 <- data.frame(Time = 1:1000, Group1 = rep(c("G", "H"), times = 500), Group2 = rep(LETTERS[1:5], each = 200)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), dot.color = NULL, error.disp = "SD", error.whisker.width = 1)*

*### whisker width. Example (2) error bars with no whiskers*

*# obs1 <- data.frame(Time = 1:1000, Group1 = rep(c("G", "H"), times = 500), Group2 = rep(LETTERS[1:5], each = 200)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), dot.color = NULL, error.disp = "SD", error.whisker.width = 0)*

*### tidy dot distribution. Example (1)*

*# obs1 <- data.frame(Time = 1:1000, Group1 = rep(c("G", "H"), times = 500), Group2 = rep(LETTERS[1:5], each = 200)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), dot.color = "same", dot.tidy = TRUE, dot.bin.nb = 100)*

*### tidy dot distribution. Example (2) reducing the dot size with dot.bin.nb*

*# obs1 <- data.frame(Time = 1:1000, Group1 = rep(c("G", "H"), times = 500), Group2 = rep(LETTERS[1:5], each = 200)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), dot.color = "same", dot.tidy = TRUE, dot.bin.nb = 150)*

*### dot jitter. Example (1)*

*# obs1 <- data.frame(Time = 1:1000, Group1 = rep(c("G", "H"), times = 500), Group2 = rep(LETTERS[1:5], each = 200)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), dot.color = "same", dot.tidy = FALSE, dot.jitter = 1, dot.size = 2)*

*### dot jitter. Example (2) with dot.jitter = 1 -> dispersion around the corresponding bar width*

*# obs1 <- data.frame(Time = 1:1000, Group1 = rep(c("G", "H"), times = 500), Group2 = rep(LETTERS[1:5], each = 200)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), dot.color = "grey", dot.size = 3, dot.alpha = 1, dot.jitter = 1)*

*### dot jitter. Example (3) with no dispersion*

*# obs1 <- data.frame(Time = 1:100, Group1 = rep(c("G", "H"), times = 50), Group2 = rep(LETTERS[1:5], each = 20)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), dot.color = "grey", dot.size = 3, dot.alpha = 1, dot.jitter = 0)*

*### dot size, dot border size and dot transparency*

*# obs1 <- data.frame(Time = 1:100, Group1 = rep(c("G", "H"), times = 50), Group2 = rep(LETTERS[1:5], each = 20)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), dot.color = "grey", dot.size = 4, dot.border.size = 0, dot.alpha = 0.6)*

*### y-axis limits. Example (1)*

*# obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), ylim = c(-1, 25))*

*### y-axis limits. Example (2) showing that order matters in ylim argument*

*# obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), ylim = c(25, -1))*

*### log scale. Example (1). BEWARE: y column must be log, otherwise incoherent scale (see below warning message with the return argument)*

*# obs1 <- data.frame(Time = log10((1:20) \* 100), Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), ylog = "log10")*

*### log scale. Example (2). BEWARE: values of the ylim must be in the corresponding log*

*# obs1 <- data.frame(Time = log10((1:20) \* 100), Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), ylog = "log10", ylim = c(1,4))*

*### tick number. Example (1)*

*# obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), y.tick.nb = 10)*

*### tick number. Example (2) using a log2 scale*

*# obs1 <- data.frame(Time = log2((1:20) \* 100), Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), ylog = "log2", y.tick.nb = 10, ylim = c(1, 16))*

*### tick number. Example (3) using a log10 scale*

*# obs1 <- data.frame(Time = log10((1:20) \* 100), Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), ylog = "log10", y.tick.nb = 10)*

*### tick number. Example (4) using a log10 scale: the reverse y-axis correctly deal with log10 scale*

*# obs1 <- data.frame(Time = log10((1:20) \* 100), Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), ylog = "log10", y.tick.nb = 10, ylim = c(4, 1))*

*### secondary tick number. Example (1)*

*# obs1 <- data.frame(Time = 1:20, Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), y.inter.tick.nb = 2)*

*### secondary ticks. Example (2) not for log2 and log10 scales (see below warning message with the return argument)*

*# obs1 <- data.frame(Time = log10((1:20) \* 100), Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), ylog = "log10", y.inter.tick.nb = 2)*

*### include zero in the y-axis*

*# obs1 <- data.frame(Time = (1:20), Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), y.include.zero = TRUE)*

*### extra margins. To avoid dot cuts*

*# obs1 <- data.frame(Time = (1:20), Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), y.top.extra.margin = 0.25, y.bottom.extra.margin = 0.25)*

*### mean diplay. Example (1) at the top of the plot region*

*# obs1 <- data.frame(Time = (1:20), Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), y.top.extra.margin = 0.1, stat.disp = "top", stat.size = 4, stat.dist = 2)*

*### mean diplay. Example (2) above bars*

*# obs1 <- data.frame(Time = (1:20), Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), y.top.extra.margin = 0.1, stat.disp = "above", stat.size = 4, stat.dist = 2)*

*### bar orientation. Example (1) without log scale, showing that the other arguments are still operational*

*# obs1 <- data.frame(Time = (1:20), Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), y.tick.nb = 10, y.inter.tick.nb = 2, y.include.zero = TRUE, vertical = FALSE)*

*### bar orientation. Example (2) with log scale. Horizontal orientation is blocked with log2 and log10 scales because of a bug in ggplot2 (https://github.com/tidyverse/ggplot2/issues/881)*

*# obs1 <- data.frame(Time = log10((1:20) \* 100), Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), ylog = "log10", vertical = FALSE)*

*### classic representation (use grid = TRUE to display the background lines of the y axis ticks)*

*# obs1 <- data.frame(Time = (1:20), Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), classic = TRUE, grid = FALSE)*

*### graphic info. Example (1)*

*# obs1 <- data.frame(Time = log10((1:20) \* 100), Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), return = TRUE)*

*### graphic info. Example (2) of assignation and warning message display*

*# obs1 <- data.frame(Time = log10((1:20) \* 100), Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; warn <- fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), ylog = "log10", return = TRUE) ; cat(warn$warn)*

*### add ggplot2 functions*

*# obs1 <- data.frame(Time = log10((1:20) \* 100), Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar(data1 = obs1, y = "Time", categ = c("Group1", "Group2"), add = "+ggplot2::theme\_classic()")*

*### all the arguments*

*# obs1 <- data.frame(x = 1:20, Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_bar(data1 = obs1, y = "x", categ = c("Group1", "Group2"), categ.class.order = list(NULL, c("B", "A")), categ.legend.name = "", categ.color = c("red", "blue"), bar.width = 0.25, error.disp = "SD", error.whisker.width = 0.8, dot.color = "grey", dot.tidy = FALSE, dot.bin.nb = 30, dot.jitter = 1, dot.size = 4, dot.border.size = 0, dot.alpha = 1, ylim = c(0, 25), ylog = "no", y.tick.nb = NULL, y.inter.tick.nb = NULL, y.include.zero = FALSE, y.top.extra.margin = 0.05, y.bottom.extra.margin = 0, stat.disp = "above", stat.size = 4, stat.dist = 2, xlab = "GROUP", ylab = "VALUE", vertical = FALSE, text.size = 12, title = "", title.text.size = 8, text.angle = 45, classic = TRUE, grid = TRUE, return = TRUE, plot = TRUE, add = NULL, warn.print = TRUE, lib.path = NULL)*

*# DEBUGGING*

*# data1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10), group2 = rep(c("A", "B"), each = 10), bar.color = rep(c("brown", "orange"), each = 10)) ; data1[2:3, 1] <- NA ; data1[7:8, 2] <- NA ; y = names(data1)[1] ; categ = c(names(data1)[2], names(data1)[3]) ; categ.class.order = list(L1 = NULL, L2 = c("B", "A")) ; categ.legend.name = NULL ; categ.color = na.omit(data1)$bar.color ; bar.width = 0.5 ; error.disp = "SD" ; 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 = 1 ; ylim = NULL ; ylog = "no" ; y.tick.nb = NULL ; y.inter.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 ; text.size = 12 ; title = "" ; title.text.size = 8 ; text.angle = 0 ; classic = FALSE ; grid = FALSE ; return = FALSE ; plot = TRUE ; add = NULL ; warn.print = TRUE ; lib.path = NULL*

*# data1 <-data.frame(a = rep(1:20, 5), group1 = rep(c("G", "H"), times = 50), group2 = rep(LETTERS[1:5], each = 20)) ; y = names(data1)[1] ; categ = c(names(data1)[2], names(data1)[3]) ; categ.class.order = list(L1 = NULL, L2 = c("B", "A", "E", "D", "C")) ; categ.legend.name = NULL ; categ.color = NULL ; bar.width = 0.5 ; error.disp = "SD" ; error.whisker.width = 0.5 ; dot.color = "same" ; dot.tidy = TRUE ; dot.bin.nb = 30 ; dot.jitter = 0.25 ; dot.size = 3 ; dot.border.size = 0.5 ; dot.alpha = 1 ; ylim = NULL ; ylog = "no" ; y.tick.nb = NULL ; y.inter.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 ; text.size = 12 ; title = "" ; title.text.size = 8 ; text.angle = 0 ; classic = FALSE ; grid = FALSE ; return = FALSE; plot = TRUE ; add = NULL ; warn.print = TRUE ; lib.path = NULL*

*# data1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10), group2 = rep(c("A", "B"), each = 10), bar.color = rep(c("brown", "orange"), each = 10)) ; data1[2:3, 1] <- NA ; data1[7:8, 2] <- NA ; y = names(data1)[1] ; categ = c(names(data1)[2], names(data1)[3]) ; categ.class.order = list(L1 = NULL, L2 = c("B", "A")) ; categ.legend.name = NULL ; categ.color = na.omit(data1)$bar.color ; bar.width = 0.5 ; error.disp = "SD" ; error.whisker.width = 0.5 ; dot.color = "same" ; dot.tidy = TRUE ; dot.bin.nb = 30 ; dot.jitter = 0.25 ; dot.size = 3 ; dot.border.size = 0.5 ; dot.alpha = 1 ; ylim = NULL ; ylog = "no" ; y.tick.nb = NULL ; y.inter.tick.nb = NULL ; y.include.zero = FALSE ; y.top.extra.margin = 0.05 ; y.bottom.extra.margin = 0 ; stat.disp = "above" ; stat.size = 4 ; stat.dist = 2 ; xlab = NULL ; ylab = NULL ; vertical = TRUE ; text.size = 12 ; title = "" ; title.text.size = 8 ; text.angle = 0 ; classic = FALSE ; grid = FALSE ; return = FALSE; plot = TRUE ; add = NULL ; warn.print = TRUE ; lib.path = NULL*

*# set.seed(1) ; data1 <- data.frame(a = c(rnorm(25, 0), rnorm(25, -10), rnorm(25, 10), rnorm(25, 20)), group1 = rep(c("G", "H"), times = 50), group2 = rep(c("A", "B", "C", "D"), each = 25)) ; set.seed(NULL) ; y = "Time" ; categ = c("group1", "group2") ; categ.class.order = list(NULL, c("B", "A", "D", "C")) ; categ.legend.name = "LEGEND" ; categ.color = NULL ; bar.width = 0.8 ; error.disp = "SD" ; error.whisker.width = 0.5 ; dot.color = "same" ; dot.tidy = TRUE ; dot.bin.nb = 60 ; dot.jitter = 0.25 ; dot.size = 3.5 ; dot.border.size = 0 ; dot.alpha = 1 ; ylim= c(-15, 25) ; ylog = "no" ; y.tick.nb = NULL ; y.inter.tick.nb = NULL ; y.include.zero = "no" ; y.top.extra.margin = 0.05 ; y.bottom.extra.margin = 0 ; stat.disp = "above" ; stat.size = 4 ; stat.dist = 2 ; xlab = "GROUP" ; ylab = "VALUE" ; vertical = FALSE ; text.size = 12 ; title = "" ; title.text.size = 8 ; text.angle = -200 ; classic = FALSE ; grid = FALSE ; return = FALSE; plot = TRUE ; add = NULL ; warn.print = TRUE ; lib.path = NULL*

*# set.seed(1) ; data1 <- data.frame(x = 1:1000, group1 = rep(c("G", "H"), times = 500), group2 = rep(LETTERS[1:5], each = 200)) ; set.seed(NULL) ; y = "x" ; categ <- c("group1", "group2") ; categ.class.order = list(NULL, c("B", "A", "D", "C", "E")) ; categ.legend.name = "LEGEND" ; categ.color = NULL ; bar.width = 0.8 ; error.disp = "SD" ; error.whisker.width = 1 ; dot.color = NULL ; dot.tidy = FALSE ; dot.bin.nb = 60 ; dot.jitter = 0.25 ; dot.size = 3.5 ; dot.border.size = 0.2 ; dot.alpha = 1 ; ylim= c(1, 4) ; ylog = "log10" ; y.tick.nb = NULL ; y.inter.tick.nb = NULL ; y.include.zero = FALSE ; y.top.extra.margin = 0 ; y.bottom.extra.margin = 0 ; stat.disp = "above" ; stat.size = 4 ; stat.dist = 1 ; xlab = "GROUP" ; ylab = "VALUE" ; vertical = TRUE ; text.size = 12 ; title = "" ; title.text.size = 8 ; text.angle = -200 ; classic = FALSE ; grid = FALSE ; return = FALSE; plot = TRUE ; add = NULL ; warn.print = TRUE ; lib.path = NULL*

*# function name*

**function.name <- paste0(as.list(match.call(expand.dots=FALSE))[[1]], "()")**

*# end function name*

*# required function checking*

**if(length(utils::find("fun\_comp\_2d", mode = "function")) == 0){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": REQUIRED fun\_comp\_2d() FUNCTION IS MISSING IN THE R ENVIRONMENT\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**if(length(utils::find("fun\_gg\_just", mode = "function")) == 0){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": REQUIRED fun\_gg\_just() FUNCTION IS MISSING IN THE R ENVIRONMENT\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**if(length(utils::find("fun\_gg\_palette", mode = "function")) == 0){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": REQUIRED fun\_gg\_palette() FUNCTION IS MISSING IN THE R ENVIRONMENT\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**if(length(utils::find("fun\_name\_change", mode = "function")) == 0){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": REQUIRED fun\_name\_change() FUNCTION IS MISSING IN THE R ENVIRONMENT\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**if(length(utils::find("fun\_pack", mode = "function")) == 0){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": REQUIRED fun\_pack() FUNCTION IS MISSING IN THE R ENVIRONMENT\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**if(length(utils::find("fun\_check", mode = "function")) == 0){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": REQUIRED fun\_check() FUNCTION IS MISSING IN THE R ENVIRONMENT\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**if(length(utils::find("fun\_round", mode = "function")) == 0){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": REQUIRED fun\_round() FUNCTION IS MISSING IN THE R ENVIRONMENT\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**if(length(utils::find("fun\_scale", mode = "function")) == 0){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": REQUIRED fun\_scale() FUNCTION IS MISSING IN THE R ENVIRONMENT\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

*# end required function checking*

*# reserved words to avoid bugs (used in this function)*

**reserved.words <- c("categ.check", "categ.color", "dot.color", "dot.max", "dot.min", "ERROR.INF", "ERROR.SUP", "group", "group.check", "max.dot.error", "MEAN", "min.dot.error", "SD", "SEM", "tempo.categ1", "tempo.categ2", "text.max.pos", "text.min.pos", "x", "x.y", "y", "y.check", "y\_from.dot.max", "ymax")**

*# end reserved words to avoid bugs (used in this function)*

*# argument checking (and modification for proper color management)*

**warn <- NULL**

**warn.count <- 0**

**arg.check <- NULL** *#*

**text.check <- NULL** *#*

**checked.arg.names <- NULL** *# for function debbuging: used by r\_debugging\_tools*

**ee <- expression(arg.check <- c(arg.check, tempo$problem) , text.check <- c(text.check, tempo$text) , checked.arg.names <- c(checked.arg.names, tempo$fun.name))**

**tempo <- fun\_check(data = data1, class = "data.frame", na.contain = TRUE, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & any(duplicated(names(data1)))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": DUPLICATED COLUMN NAMES OF data1 ARGUMENT NOT ALLOWED:\n", paste(names(data1)[duplicated(names(data1))], collapse = " "))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**tempo <- fun\_check(data = y, class = "vector", mode = "character", length = 1, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & ! (y %in% names(data1))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": y ARGUMENT MUST BE A COLUMN NAME OF data1")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}else if(tempo$problem == FALSE){**

**tempo <- fun\_check(data = data1[, y], data.name = "y COLUMN OF data1", class = "vector", mode = "numeric", na.contain = TRUE, fun.name = function.name) ; eval(ee)**

**}**

**tempo <- fun\_check(data = categ, class = "vector", mode = "character", fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & length(categ) > 2){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": categ ARGUMENT CANNOT HAVE MORE THAN 2 COLUMN NAMES OF data1\n\n================\n\n")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}else if(tempo$problem == FALSE & ! all(categ %in% names(data1))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": categ ARGUMENT MUST BE COLUMN NAMES OF data1. HERE IT IS:\n", paste(categ, collapse = " "))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

*# reserved word checking*

**if(any(names(data1) %in% reserved.words)){**

**if(any(duplicated(names(data1)))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": DUPLICATED COLUMN NAMES OF data1 ARGUMENT NOT ALLOWED:\n", paste(names(data1)[duplicated(names(data1))], collapse = " "))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**tempo.output <- fun\_name\_change(names(data1), reserved.words)**

**for(i2 in 1:length(tempo.output$ini)){** *# a loop to be sure to take the good ones*

**names(data1)[names(data1) == tempo.output$ini[i2]] <- tempo.output$post[i2]**

**if(any(y == tempo.output$ini[i2])){**

**y[y == tempo.output$ini[i2]] <- tempo.output$post[i2]**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": IN y ARGUMENT (COLUMN NAMES OF data1 ARGUMENT),\n", tempo.output$ini[i2], " HAS BEEN REPLACED BY ", tempo.output$post[i2], "\nBECAUSE RISK OF BUG AS SOME NAMES IN y ARGUMENT ARE RESERVED WORD USED BY THE ", function.name, " FUNCTION")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**if(any(categ == tempo.output$ini[i2])){**

**categ[categ == tempo.output$ini[i2]] <- tempo.output$post[i2]**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": IN categ ARGUMENT (COLUMN NAMES OF data1 ARGUMENT),\n", tempo.output$ini[i2], " HAS BEEN REPLACED BY ", tempo.output$post[i2], "\nBECAUSE RISK OF BUG AS SOME NAMES IN categ ARGUMENT ARE RESERVED WORD USED BY THE ", function.name, " FUNCTION")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**}**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": REGARDING COLUMN NAMES REPLACEMENT, THE NAMES\n", paste(tempo.output$ini, collapse = " "), "\nHAVE BEEN REPLACED BY\n", paste(tempo.output$post, collapse = " "))**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

*# end reserved word checking*

*# na detection and removal (done now to be sure of the correct length of categ)*

**if(any(is.na(data1[, c(y, categ)]))){**

**removed.row.nb <- unlist(lapply(lapply(c(data1[c(y, categ)]), FUN = is.na), FUN = which))**

**removed.rows <- data1[removed.row.nb, ]**

**data1 <- data1[-removed.row.nb, ]**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": NA DETECTED IN COLUMN ", paste(c(y, categ), collapse = " "), " OF data1 AND CORRESPONDING ROWS REMOVED (SEE $removed.row.nb AND $removed.rows)")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}else{**

**removed.row.nb <- NULL**

**removed.rows <- NULL**

**}**

*# end na detection and removal (done now to be sure of the correct length of categ)*

**for(i1 in 1:length(categ)){**

**if(any(is.na(data1[, categ[i1]]))){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": IN categ NUMBER ", i1, " IN data1, THE CATEGORY COLUMN ", categ[i1], " CONTAINS NA")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**tempo1 <- fun\_check(data = data1[, categ[i1]], data.name = paste0("categ NUMBER ", i1, " OF data1"), class = "vector", mode = "character", na.contain = TRUE, fun.name = function.name, print = FALSE)**

**tempo2 <- fun\_check(data = data1[, categ[i1]], data.name = paste0("categ NUMBER ", i1, " OF data1"), class = "factor", na.contain = TRUE, fun.name = function.name, print = FALSE)**

**if(tempo1$problem == TRUE & tempo2$problem == TRUE){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ", paste0("categ NUMBER ", i1, " OF data1"), " MUST BE A FACTOR OR CHARACTER VECTOR\n\n================\n\n")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}else if(tempo1$problem == FALSE){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": IN categ NUMBER ", i1, " IN data1, THE CHARACTER COLUMN HAS BEEN CONVERTED TO FACTOR")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**data1[, categ[i1]] <- factor(data1[, categ[i1]])** *# if already a factor, change nothing, if characters, levels according to alphabetical order*

**}**

**if( ! is.null(categ.class.order)){**

**tempo <- fun\_check(data = categ.class.order, class = "list", fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & length(categ.class.order) > 2){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": categ.class.order ARGUMENT MUST BE A LIST OF MAX LENGTH 2\n\n================\n\n")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}else if(tempo$problem == FALSE){**

**for(i3 in 1:length(categ.class.order)){**

**if(is.null(categ.class.order[[i3]])){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": THE categ.class.order COMPARTMENT ", i3, " IS NULL. ALPHABETICAL ORDER WILL BE APPLIED")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**data1[, categ[i3]] <- factor(as.character(data1[, categ[i3]]))** *# if already a factor, change nothing, if characters, levels according to alphabetical order*

**}else if(any(duplicated(categ.class.order[[i3]]))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": COMPARTMENT ", i3, " OF categ.class.order ARGUMENT CANNOT HAVE DUPLICATED CLASSES: ", paste(categ.class.order[[i3]], collapse = " "))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}else if( ! (all(categ.class.order[[i3]] %in% unique(data1[, categ[i3]])) & all(unique(data1[, categ[i3]]) %in% categ.class.order[[i3]]))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": COMPARTMENT ", i3, " OF categ.class.order ARGUMENT MUST BE CLASSES OF ELEMENT ", i3, " OF categ\nHERE IT IS:\nCOMPARTMENT ", i3, " OF categ.class.order:", paste(categ.class.order[[i3]], collapse = " "), "\nCOLUMN ", categ[i3], " OF data1: ", paste( unique(data1[, categ[i3]]), collapse = " "))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}else{**

**data1[, categ[i3]] <- factor(data1[, categ[i3]], levels = categ.class.order[[i3]])** *# reorder the factor*

**}**

**}**

**}**

**}**

**if( ! is.null(categ.legend.name)){**

**tempo <- fun\_check(data = categ.legend.name, class = "vector", mode = "character", fun.name = function.name) ; eval(ee)**

**}else{**

**categ.legend.name <- categ[length(categ)]** *# if only categ1, then legend name of categ1, if length(categ) == 2, then legend name of categ2*

**}**

**if( ! is.null(categ.color)){**

*# check the nature of color*

**tempo1 <- fun\_check(data = categ.color, class = "vector", mode = "character", na.contain = TRUE, fun.name = function.name, print = FALSE)**

**tempo2 <- fun\_check(data = categ.color, class = "factor", na.contain = TRUE, fun.name = function.name, print = FALSE)**

**if(tempo1$problem == TRUE & tempo2$problem == TRUE){**

*# integer colors into gg\_palette*

**tempo.check.color <- fun\_check(data = categ.color, class = "integer", double.as.integer.allowed = TRUE, na.contain = TRUE, fun.name = function.name, print = FALSE)$problem**

**if(tempo.check.color == TRUE){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": categ.color MUST BE A FACTOR OR CHARACTER VECTOR OR INTEGER VECTOR\n\n================\n\n")** *# integer possible because dealt above*

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}else{** *# convert integers into colors*

**categ.color <- fun\_gg\_palette(max(categ.color, na.rm = TRUE))**

**}**

*# end integer colors into gg\_palette*

**}**

**if( ! (all(categ.color %in% colors() | grepl(pattern = "^#", categ.color)))){** *# check that all strings of low.color start by #*

**tempo.cat <- paste0("ERROR IN ", function.name, ": categ.color ARGUMENT MUST BE A HEXADECIMAL COLOR VECTOR STARTING BY # AND/OR COLOR NAMES GIVEN BY colors(): ", paste(unique(categ.color), collapse = " "))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**if(any(is.na(categ.color))){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": categ.color ARGUMENT CONTAINS NA")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

*# end check the nature of color*

*# check the length of color*

*# No problem of NA management by ggplot2 because already removed*

**categ.len <- length(categ)** *# if only categ1, then colors for classes of categ1, if length(categ) == 2, then colors for classes of categ2*

**if(length(categ.color) == length(unique(data1[, categ[categ.len]]))){** *# here length(categ.color) is equal to the different number of categ*

**data1[, categ[categ.len]] <- factor(data1[, categ[categ.len]])** *# if already a factor, change nothing, if characters, levels according to alphabetical order*

**data1 <- data.frame(data1, categ.color = data1[, categ[categ.len]])**

**levels(data1$categ.color) <- categ.color**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": IN ", categ[categ.len], " OF categ ARGUMENT, THE FOLLOWING COLORS:\n", paste(categ.color, collapse = " "), "\nHAVE BEEN ATTRIBUTED TO THESE CLASSES:\n", paste(levels(factor(data1[, categ[categ.len]])), collapse = " "))**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}else if(length(categ.color) == length(data1[, categ[categ.len]])){***# here length(categ.color) is equal to nrow(data1) -> Modif to have length(categ.color) equal to the different number of categ (length(categ.color) == length(levels(data1[, categ[categ.len]])))*

**data1 <- data.frame(data1, categ.color = categ.color)**

**tempo.check <- unique(data1[ , c(categ[categ.len], "categ.color")])**

**if( ! (nrow(tempo.check) == length(unique(categ.color)) & nrow(tempo.check) == length(unique(data1[ , categ[categ.len]])))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": categ.color ARGUMENT HAS THE LENGTH OF data1 ROW NUMBER\nBUT IS INCORRECTLY ASSOCIATED TO EACH CLASS OF categ ", categ[categ.len], ":\n", paste(unique(mapply(FUN = "paste", data1[ ,categ[categ.len]], data1[ ,"categ.color"])), collapse = "\n"))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}else{**

**data1[, categ[categ.len]] <- factor(data1[, categ[categ.len]])** *# if already a factor, change nothing, if characters, levels according to alphabetical order*

**categ.color <- unique(categ.color[order(data1[, categ[categ.len]])])** *# Modif to have length(categ.color) equal to the different number of categ (length(categ.color) == length(levels(data1[, categ[categ.len]])))*

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": categ.color ARGUMENT HAS THE LENGTH OF data1 ROW NUMBER\nCOLORS HAVE BEEN RESPECTIVELY ASSOCIATED TO EACH CLASS OF categ ", categ[categ.len], " AS:\n", paste(levels(factor(data1[, categ[categ.len]])), collapse = " "), "\n", paste(categ.color, collapse = " "))**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**}else if(length(categ.color) == 1){**

**data1[, categ[categ.len]] <- factor(data1[, categ[categ.len]])** *# if already a factor, change nothing, if characters, levels according to alphabetical order*

**data1 <- data.frame(data1, categ.color = categ.color)**

**categ.color <- rep(categ.color, length(levels(data1[, categ[categ.len]])))**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": categ.color ARGUMENT HAS LENGTH 1, MEANING THAT ALL THE DIFFERENT CLASSES OF ", categ[categ.len], "\n", paste(levels(factor(data1[, categ[categ.len]])), collapse = " "), "\nWILL HAVE THE SAME COLOR\n", paste(categ.color, collapse = " "))**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}else{**

**tempo.cat <- paste0("ERROR IN ", function.name, ": categ.color ARGUMENT MUST BE (1) LENGTH 1, OR (2) THE LENGTH OF data1 NROWS, OR (3) THE LENGTH OF THE CLASSES IN THE categ ", categ[categ.len], " COLUMN. HERE IT IS COLOR LENGTH ", length(categ.color), " VERSUS CATEG LENGTH ", length(data1[, categ[categ.len]]), " AND CATEG CLASS LENGTH ", length(unique(data1[, categ[categ.len]])), "\nPRESENCE OF NA COULD BE THE PROBLEM\n\n================\n\n")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}else{**

**categ.len <- length(categ)** *# if only categ1, then colors for classes of categ1, if length(categ) == 2, then colors for classes of categ2*

**data1[, categ[categ.len]] <- factor(data1[, categ[categ.len]])** *# if already a factor, change nothing, if characters, levels according to alphabetical order*

**categ.color <- fun\_gg\_palette(length(levels(data1[, categ[categ.len]])))**

**data1 <- data.frame(data1, categ.color = data1[, categ[categ.len]])**

**levels(data1$categ.color) <- categ.color**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": NULL categ.color ARGUMENT -> COLORS RESPECTIVELY ATTRIBUTED TO EACH CLASS OF ", categ[categ.len], " IN data1:\n", paste(categ.color, collapse = " "), "\n", paste(levels(data1[, categ[categ.len]]), collapse = " "))**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**tempo <- fun\_check(data = bar.width, prop = TRUE, length = 1, fun.name = function.name) ; eval(ee)**

**if( ! is.null(error.disp)){**

**tempo <- fun\_check(data = error.disp, options = c("SD", "SD.TOP", "SEM", "SEM.TOP"), length = 1, fun.name = function.name) ; eval(ee)**

**}**

**tempo <- fun\_check(data = error.whisker.width, prop = TRUE, length = 1, fun.name = function.name) ; eval(ee)**

**if( ! is.null(dot.color)){**

*# check the nature of color*

**tempo1 <- fun\_check(data = dot.color, class = "vector", mode = "character", na.contain = TRUE, fun.name = function.name, print = FALSE)**

**tempo2 <- fun\_check(data = dot.color, class = "factor", na.contain = TRUE, fun.name = function.name, print = FALSE)**

**if(tempo1$problem == TRUE & tempo2$problem == TRUE){**

*# integer colors into gg\_palette*

**tempo.check.color <- fun\_check(data = dot.color, class = "integer", double.as.integer.allowed = TRUE, na.contain = TRUE, fun.name = function.name, print = FALSE)$problem**

**if(tempo.check.color == TRUE){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": dot.color MUST BE A FACTOR OR CHARACTER VECTOR OR INTEGER VECTOR\n\n================\n\n")** *# integer possible because dealt above*

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}else{** *# convert integers into colors*

**dot.color <- fun\_gg\_palette(max(dot.color, na.rm = TRUE))**

**}**

*# end integer colors into gg\_palette*

**}**

**if(all(dot.color == "same") & length(dot.color) == 1){**

**dot.color <- categ.color** *# same color of the dots as the corresponding bar color*

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": dot.color ARGUMENT HAS BEEN SET TO \"SAME\"\nTHUS, DOT COLORS HAVE BEEN RESPECTIVELY ASSOCIATED TO EACH CLASS OF categ ", categ[categ.len], " AS:\n", paste(levels(factor(data1[, categ[categ.len]])), collapse = " "), "\n", paste(levels(factor(dot.color)), collapse = " "))**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}else if( ! (all(dot.color %in% colors() | grepl(pattern = "^#", dot.color)))){** *# check that all strings of low.color start by #*

**tempo.cat <- paste0("ERROR IN ", function.name, ": dot.color ARGUMENT MUST BE (1) A HEXADECIMAL COLOR VECTOR STARTING BY #, OR (2) COLOR NAMES GIVEN BY colors(), OR (3) INTEGERS, OR THE STRING\"same\"\nHERE IT IS: ", paste(unique(dot.color), collapse = " "))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**if(any(is.na(dot.color))){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": dot.color ARGUMENT CONTAINS NA")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

*# end check the nature of color*

*# check the length of color*

*# No problem of NA management by ggplot2 because already removed*

**categ.len <- length(categ)** *# if only categ1, then colors for classes of categ1, if length(categ) == 2, then colors for classes of categ2*

**if(length(dot.color) == length(unique(data1[, categ[categ.len]]))){** *# here length(dot.color) is equal to the different number of categ*

**data1[, categ[categ.len]] <- factor(data1[, categ[categ.len]])** *# if already a factor, change nothing, if characters, levels according to alphabetical order*

**data1 <- data.frame(data1, dot.color = data1[, categ[categ.len]])**

**levels(data1$dot.color) <- dot.color**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": IN ", categ[categ.len], " OF categ ARGUMENT, THE FOLLOWING COLORS:\n", paste(dot.color, collapse = " "), "\nHAVE BEEN ATTRIBUTED TO THESE CLASSES:\n", paste(levels(factor(data1[, categ[categ.len]])), collapse = " "))**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}else if(length(dot.color) == length(data1[, categ[categ.len]])){***# here length(dot.color) is equal to nrow(data1) -> Modif to have length(dot.color) equal to the different number of categ (length(dot.color) == length(levels(data1[, categ[categ.len]])))*

**data1 <- data.frame(data1, dot.color = dot.color)**

**}else if(length(dot.color) == 1 & ! all(dot.color == "same")){**

**data1[, categ[categ.len]] <- factor(data1[, categ[categ.len]])** *# if already a factor, change nothing, if characters, levels according to alphabetical order*

**data1 <- data.frame(data1, dot.color = dot.color)**

**dot.color <- rep(dot.color, length(levels(data1[, categ[categ.len]])))**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": dot.color ARGUMENT HAS LENGTH 1, MEANING THAT ALL THE DIFFERENT CLASSES OF ", categ[categ.len], "\n", paste(levels(factor(data1[, categ[categ.len]])), collapse = " "), "\nWILL HAVE THE SAME COLOR\n", paste(dot.color, collapse = " "))**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}else{**

**tempo.cat <- paste0("ERROR IN ", function.name, ": dot.color ARGUMENT MUST BE (1) LENGTH 1, OR (2) THE LENGTH OF data1 NROWS, OR (3) THE LENGTH OF THE CLASSES IN THE categ ", categ[categ.len], " COLUMN. HERE IT IS COLOR LENGTH ", length(dot.color), " VERSUS CATEG LENGTH ", length(data1[, categ[categ.len]]), " AND CATEG CLASS LENGTH ", length(unique(data1[, categ[categ.len]])), "\nPRESENCE OF NA COULD BE THE PROBLEM\n\n================\n\n")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**tempo <- fun\_check(data = dot.tidy, class = "vector", mode = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = dot.bin.nb, class = "vector", typeof = "integer", length = 1, double.as.integer.allowed = TRUE, neg.values = FALSE, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = dot.jitter, prop = TRUE, length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = dot.size, class = "vector", mode = "numeric", length = 1, neg.values = FALSE, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = dot.border.size, class = "vector", mode = "numeric", length = 1, neg.values = FALSE, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = dot.alpha, prop = TRUE, length = 1, fun.name = function.name) ; eval(ee)**

**if( ! is.null(ylim)){**

**tempo <- fun\_check(data = ylim, class = "vector", mode = "numeric", length = 2, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & any(ylim %in% c(Inf, -Inf))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ylim ARGUMENT CANNOT CONTAIN -Inf OR Inf VALUES\n\n================\n\n")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**tempo <- fun\_check(data = ylog, options = c("no", "log2", "log10"), length = 1, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & ylog != "no"){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": ylog ARGUMENT SET TO ", ylog, ".\nVALUES FROM THE y ARGUMENT COLUMN OF THE data1 DATA FRAME MUST BE ALREADY ", toupper(ylog), " TRANSFORMED, AS THE ylog ARGUMENT JUST MODIFIES THE AXIS SCALE")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**if( ! is.null(ylim)){**

**if(any(ylim <= 0)){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": ylim ARGUMENT CAN SPAN ZERO OR NEGATIVE VALUES IF ylog ARGUMENT IS SET TO ", ylog, " BECAUSE ylim DOES NOT TRANSFORM DATA, JUST MODIFIES THE AXIS SCALE")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}else if(any( ! is.finite(if(ylog == "log10"){10^ylim}else{2^ylim}))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": ylim ARGUMENT RETURNS INF WITH THE ylog ARGUMENT SET TO ", ylog, "\nAS SCALE COMPUTATION IS ", ifelse(ylog == "log10", 10, 2), "^ylim:\n", paste(ifelse(ylog == "log10", 10, 2)^ylim, collapse = " "), "\nARE YOU SURE THAT ylim ARGUMENT HAS BEEN SPECIFIED WITH VALUES ALREADY IN LOG SCALE?\n", paste(ylim, collapse = " "))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**}**

**if( ! is.null(y.tick.nb)){**

**tempo <- fun\_check(data = y.tick.nb, class = "vector", typeof = "integer", length = 1, double.as.integer.allowed = TRUE, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & y.tick.nb < 0){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": y.tick.nb ARGUMENT MUST BE A NON NULL POSITIVE INTEGER\n\n================\n\n")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**if( ! is.null(y.inter.tick.nb)){**

**tempo <- fun\_check(data = y.inter.tick.nb, class = "vector", typeof = "integer", length = 1, double.as.integer.allowed = TRUE, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & y.inter.tick.nb < 0){**

**tempo.cat <- paste0("ERROR IN ", function.name, ": y.inter.tick.nb ARGUMENT MUST BE A NON NULL POSITIVE INTEGER\n\n================\n\n")**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**tempo <- fun\_check(data = y.include.zero, class = "vector", mode = "logical", length = 1, fun.name = function.name) ; eval(ee)**

*# inactivated because xlim and ylim already log transformed*

*# if(tempo$problem == FALSE & ylog != "no" & y.include.zero == TRUE){*

*# tempo.warn <- paste0("FROM FUNCTION ", function.name, ": ylog ARGUMENT SET TO ", ylog, " AND y.include.zero ARGUMENT SET TO TRUE -> y.include.zero ARGUMENT RESET TO FALSE BECAUSE NO 0 ALLOWED IN LOG SCALE")*

*# warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))*

*# }*

**tempo <- fun\_check(data = y.top.extra.margin, prop = TRUE, length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = y.bottom.extra.margin, prop = TRUE, length = 1, fun.name = function.name) ; eval(ee)**

**if( ! is.null(stat.disp)){**

**tempo <- fun\_check(data = stat.disp, options = c("top", "above"), length = 1, fun.name = function.name) ; eval(ee)**

**}**

**tempo <- fun\_check(data = stat.size, class = "vector", mode = "numeric", length = 1, neg.values = FALSE, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = stat.dist, class = "vector", mode = "numeric", length = 1, fun.name = function.name) ; eval(ee)**

**if( ! is.null(xlab)){**

**if(all(class(xlab) %in% "expression")){** *# to deal with math symbols*

**tempo <- fun\_check(data = xlab, class = "expression", length = 1, fun.name = function.name) ; eval(ee)**

**}else{**

**tempo <- fun\_check(data = xlab, class = "vector", mode = "character", length = 1, fun.name = function.name) ; eval(ee)**

**}**

**}**

**if( ! is.null(ylab)){**

**if(all(class(ylab) %in% "expression")){** *# to deal with math symbols*

**tempo <- fun\_check(data = ylab, class = "expression", length = 1, fun.name = function.name) ; eval(ee)**

**}else{**

**tempo <- fun\_check(data = ylab, class = "vector", mode = "character", length = 1, fun.name = function.name) ; eval(ee)**

**}**

**}**

**tempo <- fun\_check(data = vertical, class = "vector", mode = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & ylog != "no" & vertical == FALSE){**

**vertical <- TRUE**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": BECAUSE OF A BUG IN ggplot2, CANNOT FLIP BARS HORIZONTALLY WITH A YLOG SCALE -> vertical ARGUMENT RESET TO TRUE")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**tempo <- fun\_check(data = text.size, class = "vector", mode = "numeric", length = 1, neg.values = FALSE, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = title, class = "vector", mode = "character", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = title.text.size, class = "vector", mode = "numeric", length = 1, neg.values = FALSE, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = text.angle, class = "vector", typeof = "integer", double.as.integer.allowed = TRUE, length = 1, neg.values = TRUE, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = classic, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = grid, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = return, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = plot, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**if( ! is.null(add)){**

**tempo <- fun\_check(data = add, class = "vector", mode = "character", length = 1, fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE & ! grepl(pattern = "^\\+", add)){** *# check that the add string start by +*

**tempo.cat <- paste0("ERROR IN ", function.name, ": add ARGUMENT MUST START WITH \"+\": ", paste(unique(add), collapse = " "))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}else if(tempo$problem == FALSE & ! grepl(pattern = "ggplot2::", add)){** *#*

**tempo.cat <- paste0("ERROR IN ", function.name, ": add ARGUMENT MUST CONTAIN \"ggplot2::\" IN FRONT OF EACH GGPLOT2 FUNCTION: ", paste(unique(add), collapse = " "))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}else if(tempo$problem == FALSE & ! grepl(pattern = ")$", add)){** *# check that the add string finished by )*

**tempo.cat <- paste0("ERROR IN ", function.name, ": add ARGUMENT MUST FINISH BY \")\": ", paste(unique(add), collapse = " "))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**tempo <- fun\_check(data = warn.print, class = "logical", length = 1, fun.name = function.name) ; eval(ee)**

**if( ! is.null(lib.path)){**

**tempo <- fun\_check(data = lib.path, class = "vector", mode = "character", fun.name = function.name) ; eval(ee)**

**if(tempo$problem == FALSE){**

**if( ! all(dir.exists(lib.path))){** *# separation to avoid the problem of tempo$problem == FALSE and lib.path == NA*

**tempo.cat <- paste0("ERROR IN ", function.name, ": DIRECTORY PATH INDICATED IN THE lib.path ARGUMENT DOES NOT EXISTS:\n", paste(lib.path, collapse = "\n"))**

**text.check <- c(text.check, tempo.cat)**

**arg.check <- c(arg.check, TRUE)**

**}**

**}**

**}**

**if(any(arg.check) == TRUE){**

**stop(paste0("\n\n================\n\n", paste(text.check[arg.check], collapse = "\n"), "\n\n================\n\n"), call. = FALSE)** *#*

**}**

*# source("C:/Users/Gael/Documents/Git\_versions\_to\_use/debugging\_tools\_for\_r\_dev-v1.2/r\_debugging\_tools-v1.2.R") ; eval(parse(text = str\_basic\_arg\_check\_dev)) ; eval(parse(text = str\_arg\_check\_with\_fun\_check\_dev)) # activate this line and use the function (with no arguments left as NULL) to check arguments status and if they have been checked using fun\_check()*

*# end argument checking (and modification for proper color management)*

*# package checking*

**fun\_pack(req.package = c("ggplot2"), lib.path = lib.path)**

**fun\_pack(req.package = c("scales"), lib.path = lib.path)**

*# end package checking*

*# main code*

**if(length(categ) == 1){**

*# new data frames for bar and error bars*

**mean.dataframe <- aggregate(x = data1[y], by = {x.env <- list(data1[, categ[1]]) ; names(x.env) <-categ[1] ; x.env}, FUN = mean, na.rm = TRUE)**

**sd.dataframe <- aggregate(x = data1[y], by = {x.env <- list(data1[, categ[1]]) ; names(x.env) <-categ[1] ; x.env}, FUN = sd, na.rm = TRUE)**

**nb.dataframe <- aggregate(x = data1[y], by = {x.env <- list(data1[, categ[1]]) ; names(x.env) <- categ[1] ; x.env}, FUN = function(x.env2){length(x.env2[ ! is.na(x.env2)])})**

**if( ! all(identical(mean.dataframe[, categ[1]], sd.dataframe[, categ[1]]) & identical(mean.dataframe[, categ[1]], nb.dataframe[, categ[1]]))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": aggregate OUTPUT IS DIFFERENT IN TERM OF CLASS ORDER FOR mean.dataframe, sd.dataframe AND nb.dataframe. CODE HAS TO BE MODIFIED\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}else{**

**sem.dataframe <- sd.dataframe**

**sem.dataframe[, y] <- sd.dataframe[, y] / (nb.dataframe[, y])^0.5**

**}**

*# end new data frames for bar and error bars*

*# data1 check categ order for dots coordinates recovery*

**data1 <- data.frame(data1, categ.check = data1[, categ[1]])**

**data1$categ.check <- as.integer(data1$categ.check)** *# to check that data1[, categ[1]] and dot.coord$group are similar, during merging*

*# end data1 check categ order for dots coordinates recovery*

*# per bar dots coordinates recovery*

**tempo.gg.name <- "gg.indiv.plot."**

**tempo.gg.count <- 0**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::ggplot(data = data1, mapping = ggplot2::aes\_string(x = categ[1], y = y, fill = categ[1])))** *# fill because this is what is used with geom\_bar*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_point(stroke = dot.border.size, size = dot.size, alpha = dot.alpha, pch = 21))**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_boxplot())** *# to easily have the equivalent of the grouped bars*

**dot.coord <- ggplot2::ggplot\_build(eval(parse(text = paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + "))))$data[[1]]**

**if( ! is.null(dot.color)){**

**dot.coord <- data.frame(dot.coord[order(dot.coord$group, dot.coord$y), ], y.check = as.double(data1[order(data1$categ.check, data1[, y]), y]), categ.check = data1[order(data1$categ.check, data1[, y]), "categ.check"], dot.color = data1[order(data1$categ.check, data1[, y]), "dot.color"], tempo.categ1 = data1[order(data1$categ.check, data1[, y]), categ[1]])** *# y.check to be sure that the order is the same between the y of data1 and the y of dot.coord*

**names(dot.coord)[names(dot.coord) == "tempo.categ1"] <- categ[1]**

**if( ! identical(dot.coord$y, dot.coord$y.check)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": (dot.coord$y AND dot.coord$y.check) AS WELL AS (dot.coord$group AND dot.coord$categ.check) MUST BE IDENTICAL. CODE HAS TO BE MODIFIED\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**}**

*# end per bar dots coordinates recovery*

**}else if(length(categ) == 2){**

*# new data frames for bar and error bars*

**mean.dataframe <- aggregate(x = data1[y], by = {x.env <- list(data1[, categ[1]], data1[, categ[2]]) ; names(x.env) <- c(categ[1], categ[2]) ; x.env}, FUN = mean, na.rm = TRUE)**

**sd.dataframe <- aggregate(x = data1[y], by = {x.env <- list(data1[, categ[1]], data1[, categ[2]]) ; names(x.env) <- c(categ[1], categ[2]) ; x.env}, FUN = sd, na.rm = TRUE)**

**nb.dataframe <- aggregate(x = data1[y], by = {x.env <- list(data1[, categ[1]], data1[, categ[2]]) ; names(x.env) <- c(categ[1], categ[2]) ; x.env}, FUN = function(x.env2){length(x.env2[ ! is.na(x.env2)])})**

**tempo.check.mean <- mapply(FUN = "paste", mean.dataframe[, categ[1]], mean.dataframe[, categ[2]], sep = "\_")**

**tempo.check.sd <- mapply(FUN = "paste", sd.dataframe[, categ[1]], sd.dataframe[, categ[2]], sep = "\_")**

**tempo.check.nb <- mapply(FUN = "paste", nb.dataframe[, categ[1]], nb.dataframe[, categ[2]], sep = "\_")**

**if( ! all(identical(tempo.check.mean, tempo.check.sd) & identical(tempo.check.mean, tempo.check.nb))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": aggregate OUTPUT IS DIFFERENT IN TERM OF CLASS ORDER FOR mean.dataframe, sd.dataframe AND nb.dataframe. CODE HAS TO BE MODIFIED\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}else{**

**sem.dataframe <- sd.dataframe**

**sem.dataframe[, y] <- sd.dataframe[, y] / (nb.dataframe[, y])^0.5**

**}**

*# end new data frames for bar and error bars*

*# data1 check categ order for dots coordinates recovery*

**tempo.factor <- paste0(data1[order(data1[, categ[2]], data1[, categ[1]]), categ[2]], "\_", data1[order(data1[, categ[2]], data1[, categ[1]]), categ[1]])**

**data1 <- data.frame(data1[order(data1[, categ[2]], data1[, categ[1]]), ], categ.check = factor(tempo.factor, levels = unique(tempo.factor)))**

**data1$categ.check <- as.integer(data1$categ.check)**

*# end data1 check categ order for dots coordinates recovery*

*# per bar dots coordinates recovery*

**tempo.gg.name <- "gg.indiv.plot."**

**tempo.gg.count <- 0**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::ggplot(data = data1, mapping = ggplot2::aes\_string(x = categ[1], y = y, fill = categ[2])))** *# fill because this is what is used with geom\_bar*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_point(stroke = dot.border.size, size = dot.size, alpha = dot.alpha, pch = 21))**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_boxplot())** *# to easily have the equivalent of the grouped bars*

**dot.coord <- ggplot2::ggplot\_build(eval(parse(text = paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + "))))$data[[1]]**

**if( ! is.null(dot.color)){**

**dot.coord <- data.frame(dot.coord[order(dot.coord$group, dot.coord$y), ], y.check = as.double(data1[order(data1$categ.check, data1[, y]), y]), categ.check = data1[order(data1$categ.check, data1[, y]), "categ.check"], dot.color = data1[order(data1$categ.check, data1[, y]), "dot.color"], tempo.categ1 = data1[order(data1$categ.check, data1[, y]), categ[1]], tempo.categ2 = data1[order(data1$categ.check, data1[, y]), categ[2]])** *# y.check to be sure that the order is the same between the y of data1 and the y of dot.coord*

**names(dot.coord)[names(dot.coord) == "tempo.categ1"] <- categ[1]**

**names(dot.coord)[names(dot.coord) == "tempo.categ2"] <- categ[2]**

**if( ! (identical(dot.coord$y, dot.coord$y.check) & identical(dot.coord$group, dot.coord$categ.check))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": (dot.coord$y AND dot.coord$y.check) AS WELL AS (dot.coord$group AND dot.coord$categ.check) MUST BE IDENTICAL. CODE HAS TO BE MODIFIED\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**}**

**}else{**

**tempo.cat <- paste0("\n\n============\n\nERROR IN ", function.name, ": CODE INCONSISTENCY 2\n\n============\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**data2 <- mean.dataframe**

**if( ! is.null(error.disp)){**

**if(error.disp == "SD"){**

**data2 <- data.frame(data2, SD = sd.dataframe[, y], ERROR.INF = mean.dataframe[, y] - sd.dataframe[, y], ERROR.SUP = mean.dataframe[, y] + sd.dataframe[, y])**

**}else if(error.disp == "SD.TOP"){**

**data2 <- data.frame(data2, SD = sd.dataframe[, y], ERROR.INF = mean.dataframe[, y], ERROR.SUP = mean.dataframe[, y] + sd.dataframe[, y])**

**}else if(error.disp == "SEM"){**

**data2 <- data.frame(data2, SEM = sem.dataframe[, y], ERROR.INF = mean.dataframe[, y] - sem.dataframe[, y], ERROR.SUP = mean.dataframe[, y] + sem.dataframe[, y])**

**}else if(error.disp == "SEM.TOP"){**

**data2 <- data.frame(data2, SEM = sem.dataframe[, y], ERROR.INF = mean.dataframe[, y], ERROR.SUP = mean.dataframe[, y] + sem.dataframe[, y])**

**}else{**

**tempo.cat <- paste0("\n\n============\n\nERROR IN ", function.name, ": CODE INCONSISTENCY 3\n\n============\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**}**

*# stat output*

**stat <- data2**

**names(stat)[names(stat) == y] <- "MEAN"**

*# end stat output*

*# range depending on means and error bars*

**if(is.null(ylim)){**

**if(is.null(dot.color)){** *# no dots plotted*

**if( ! is.null(error.disp)){**

**if(any(c(data2[, "ERROR.INF"], data2[, "ERROR.SUP"]) %in% c(Inf, -Inf))){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": THE data2 ARGUMENT CONTAINS -Inf OR Inf VALUES IN THE ERROR.INF OR ERROR.SUP COLUMN, THAT WILL NOT BE CONSIDERED IN THE PLOT RANGE")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**ylim <- range(c(data2[, "ERROR.INF"], data2[, "ERROR.SUP"]), na.rm = TRUE, finite = TRUE)** *# finite = TRUE removes all the -Inf and Inf except if only this. In that case, whatever the -Inf and/or Inf present, output -Inf;Inf range. Idem with NA only*

**}else{**

**if(any(data2[, y] %in% c(Inf, -Inf))){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": THE data2 ARGUMENT CONTAINS -Inf OR Inf VALUES IN THE y COLUMN, THAT WILL NOT BE CONSIDERED IN THE PLOT RANGE")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**ylim <- range(data2[, y], na.rm = TRUE, finite = TRUE)** *# finite = TRUE removes all the -Inf and Inf except if only this. In that case, whatever the -Inf and/or Inf present, output -Inf;Inf range. Idem with NA only*

**}**

**}else{**

**if(any(data1[, y] %in% c(Inf, -Inf))){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": THE data1 ARGUMENT CONTAINS -Inf OR Inf VALUES IN THE y COLUMN, THAT WILL NOT BE CONSIDERED IN THE PLOT RANGE")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

**ylim <- range(data1[, y], na.rm = TRUE, finite = TRUE)** *# finite = TRUE removes all the -Inf and Inf except if only this. In that case, whatever the -Inf and/or Inf present, output -Inf;Inf range. Idem with NA only*

**}**

**}**

**if(suppressWarnings(all(ylim %in% c(Inf, -Inf)))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, " COMPUTED YLIM CONTAINS Inf VALUES, BECAUSE VALUES FROM data2 ARGUMENTS ARE NA OR Inf ONLY\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

*# end range depending on means and error bars*

**ylim.order <- order(ylim)** *# to deal with inverse axis*

**ylim <- sort(ylim)**

**ylim[1] <- ylim[1] - abs(ylim[2] - ylim[1]) \* ifelse(diff(ylim.order) > 0, y.bottom.extra.margin, y.top.extra.margin)** *# diff(ylim.order) > 0 means not inversed axis*

**ylim[2] <- ylim[2] + abs(ylim[2] - ylim[1]) \* ifelse(diff(ylim.order) > 0, y.top.extra.margin, y.bottom.extra.margin)** *# diff(ylim.order) > 0 means not inversed axis*

**if(y.include.zero == TRUE){** *# no need to check ylog != "no" because done before*

**ylim <- range(c(ylim, 0), na.rm = TRUE, finite = TRUE)** *# finite = TRUE removes all the -Inf and Inf except if only this. In that case, whatever the -Inf and/or Inf present, output -Inf;Inf range. Idem with NA only*

**}**

**ylim <- ylim[ylim.order]**

**if(any(is.na(ylim))){**

**tempo.cat <- paste0("\n\n============\n\nERROR IN ", function.name, ": CODE INCONSISTENCY 4\n\n============\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

*# width commputations*

**if(length(categ) == 2){**

**bar.width2 <- bar.width / length(unique(data1[, categ[length(categ)]]))** *# real width of each bar in x-axis unit, among the set of grouped bar. Not relevant if no grouped bars length(categ) == 1*

**}else if(length(categ) == 1){**

**bar.width2 <- bar.width**

**}else{**

**tempo.cat <- paste0("\n\n============\n\nERROR IN ", function.name, ": CODE INCONSISTENCY 5\n\n============\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**error.whisker.width <- bar.width \* error.whisker.width** *# real error bar width*

**dot.jitter <- bar.width2 \* dot.jitter** *# real dot.jitter*

*# end width commputations*

*# barplot*

*# constant part*

**tempo.gg.name <- "gg.indiv.plot."**

**tempo.gg.count <- 0**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::ggplot())**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::xlab(if(is.null(xlab)){categ[1]}else{xlab}))**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::ylab(if(is.null(ylab)){y}else{ylab}))**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::ggtitle(title))**

*# text angle management*

**tempo.just <- fun\_gg\_just(angle = text.angle, axis = ifelse(vertical == TRUE, "x", "y"))**

*# end text angle management*

**add.check <- TRUE**

**if( ! is.null(add)){** *# if add is NULL, then = 0*

**if(grepl(pattern = "ggplot2::theme", add) == TRUE){**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": \"ggplot2::theme\" STRING DETECTED IN THE add ARGUMENT -> INTERNAL GGPLOT2 THEME FUNCTIONS theme() AND theme\_classic() HAVE BEEN INACTIVATED, TO BE USED BY THE USER")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**add.check <- FALSE**

**}**

**}**

**if(add.check == TRUE & classic == TRUE){**

*# BEWARE: not possible to add several times theme(). NO message but the last one overwrites the others*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::theme\_classic(base\_size = text.size))**

**if(grid == TRUE){**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), m.gg <- ggplot2::theme(**

**text = ggplot2::element\_text(size = text.size),**

**plot.title = ggplot2::element\_text(size = title.text.size),** *# stronger than text*

**line = ggplot2::element\_line(size = 0.5),**

**axis.line.y.left = ggplot2::element\_line(colour = "black"),** *# draw lines for the y axis*

**axis.line.x.bottom = ggplot2::element\_line(colour = "black"),** *# draw lines for the x axis*

**panel.grid.major.x = if(vertical == TRUE){NULL}else{ggplot2::element\_line(colour = "grey75")},**

**panel.grid.major.y = if(vertical == TRUE){ggplot2::element\_line(colour = "grey75")}else{NULL},**

**axis.text.x = if(vertical == TRUE){ggplot2::element\_text(angle = tempo.just$angle, hjust = tempo.just$hjust, vjust = tempo.just$vjust)}else{NULL},**

**axis.text.y = if(vertical == TRUE){NULL}else{ggplot2::element\_text(angle = tempo.just$angle, hjust = tempo.just$hjust, vjust = tempo.just$vjust)}**

**))**

**}else{**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), m.gg <- ggplot2::theme(**

**text = ggplot2::element\_text(size = text.size),**

**plot.title = ggplot2::element\_text(size = title.text.size),** *# stronger than text*

**line = ggplot2::element\_line(size = 0.5),**

**axis.line.y.left = ggplot2::element\_line(colour = "black"),**

**axis.line.x.bottom = ggplot2::element\_line(colour = "black"),**

**axis.text.x = if(vertical == TRUE){ggplot2::element\_text(angle = tempo.just$angle, hjust = tempo.just$hjust, vjust = tempo.just$vjust)}else{NULL},**

**axis.text.y = if(vertical == TRUE){NULL}else{ggplot2::element\_text(angle = tempo.just$angle, hjust = tempo.just$hjust, vjust = tempo.just$vjust)}**

**))**

**}**

**}else if(add.check == TRUE & classic == FALSE){**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), m.gg <- ggplot2::theme(**

**text = ggplot2::element\_text(size = text.size),**

**plot.title = ggplot2::element\_text(size = title.text.size),** *# stronger than text*

**line = ggplot2::element\_line(size = 0.5),**

**panel.background = ggplot2::element\_rect(fill = "grey95"),**

**axis.line.y.left = ggplot2::element\_line(colour = "black"),**

**axis.line.x.bottom = ggplot2::element\_line(colour = "black"),**

**panel.grid.major.x = ggplot2::element\_line(colour = "grey75"),**

**panel.grid.major.y = ggplot2::element\_line(colour = "grey75"),**

**panel.grid.minor.x = ggplot2::element\_blank(),**

**panel.grid.minor.y = ggplot2::element\_blank(),**

**strip.background = ggplot2::element\_rect(fill = "white", colour = "black"),**

**axis.text.x = if(vertical == TRUE){ggplot2::element\_text(angle = tempo.just$angle, hjust = tempo.just$hjust, vjust = tempo.just$vjust)}else{NULL},**

**axis.text.y = if(vertical == TRUE){NULL}else{ggplot2::element\_text(angle = tempo.just$angle, hjust = tempo.just$hjust, vjust = tempo.just$vjust)}**

**))**

**}**

*# end constant part*

*# barplot and error bars*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_bar(data = data2, mapping = ggplot2::aes\_string(x = categ[1], y = y, fill = categ[length(categ)]), stat = "identity", position = ggplot2::position\_dodge(width = NULL), color = "black", width = bar.width))** *# stat = "identity" because already counted, position = position\_dodge(width = NULL) for grouped bars (width = NULL means no overlap between grouped bars). Please, see explanation in https://stackoverflow.com/questions/34889766/what-is-the-width-argument-in-position-dodge/35102486#35102486*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_discrete\_manual(aesthetics = "fill", name = categ.legend.name, values = as.character(categ.color), guide = ggplot2::guide\_legend(override.aes = list(fill = categ.color))))** *# values are the values of color (which is the border color in geom\_bar. BEWARE: values = categ.color takes the numbers to make the colors if categ.color is a factor*

**if( ! is.null(error.disp)){**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_errorbar(data = data2, mapping = ggplot2::aes\_string(x = categ[1], group = categ[length(categ)], ymin = "ERROR.INF", ymax = "ERROR.SUP"), position = ggplot2::position\_dodge(width = bar.width), color = "black", width = error.whisker.width))** *# cannot use fill = categ[length(categ)] because not an aesthetic of geom\_errorbar, but if only x = categ[1], wrong x coordinates with grouped bars*

**}**

*# end barplot and error bars*

*# coordinates management (for random plotting and for stat display)*

*# bars*

**bar.coord <- ggplot2::ggplot\_build(eval(parse(text = paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + "))))$data[[1]]** *# to have the summary statistics of the plot. Here because can be required for stat.disp when just bar are plotted*

*# end bars*

**if( ! is.null(dot.color)){**

*# random dots*

**if(dot.tidy == FALSE){**

**dot.coord.rd1 <- merge(dot.coord, bar.coord[c("fill", "group", "x")], by = intersect("group", "group"), sort = FALSE)** *# rd for random. Send the coord of the bars into the coord data.frame of the dots (in the column x.y). BEWARE: by = intersect("group", "group") because group is enough as only one value of x per group number in bar.coord. Thus, no need to consider fill*

**if(nrow(dot.coord.rd1) != nrow(dot.coord)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": THE merge() FUNCTION DID NOT RETURN A CORRECT dot.coord.rd1 DATA FRAME. CODE HAS TO BE MODIFIED\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**set.seed(1)**

**sampled.dot.jitter <- if(nrow(dot.coord.rd1) == 1){runif(n = nrow(dot.coord.rd1), min = - dot.jitter / 2, max = dot.jitter / 2)}else{sample(x = runif(n = nrow(dot.coord.rd1), min = - dot.jitter / 2, max = dot.jitter / 2), size = nrow(dot.coord.rd1), replace = FALSE)}**

**dot.coord.rd2 <- data.frame(dot.coord.rd1, dot.x = dot.coord.rd1$x.y + sampled.dot.jitter)** *# set the dot.jitter thanks to runif and dot.jitter range. Then, send the coord of the bars into the coord data.frame of the dots (in the column x.y)*

**set.seed(NULL)**

**if(length(categ) == 1){**

**tempo.data1 <- unique(data.frame(data1[categ[1]], group = as.integer(factor(as.numeric(data1[, categ[1]])))))** *# categ[2] first if categ[2] is used to make the categories in ggplot and categ[1] is used to make the x-axis*

**names(tempo.data1)[names(tempo.data1) == categ[1]] <- paste0(categ[1], ".check")**

**verif <- paste0(categ[1], ".check")**

**}else if(length(categ) == 2){**

**tempo.data1 <- unique(data.frame(data1[c(categ[1], categ[2])], group = as.integer(factor(paste0(as.numeric(data1[, categ[2]]), ".", as.numeric(data1[, categ[1]]))))))** *# categ[2] first if categ[2] is used to make the categories in ggplot and categ[1] is used to make the x-axis*

**names(tempo.data1)[names(tempo.data1) == categ[1]] <- paste0(categ[1], ".check")**

**names(tempo.data1)[names(tempo.data1) == categ[2]] <- paste0(categ[2], ".check")**

**verif <- c(paste0(categ[1], ".check"), paste0(categ[2], ".check"))**

**}else{**

**tempo.cat <- paste0("\n\n============\n\nERROR IN ", function.name, ": CODE INCONSISTENCY 6\n\n============\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**dot.coord.rd3 <- merge(dot.coord.rd2, tempo.data1, by = "group", sort = FALSE)** *# send the factors of data1 into coord*

**if(nrow(dot.coord.rd3) != nrow(dot.coord) | ( ! fun\_comp\_2d(dot.coord.rd3[categ], dot.coord.rd3[verif])$identical.content)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": THE merge() FUNCTION DID NOT RETURN A CORRECT dot.coord.rd3 DATA FRAME. CODE HAS TO BE MODIFIED\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

*# end random dots*

**}**

*# tidy dots*

*# coordinates are recover during plotting (see dot.coord.tidy1 below)*

*# end tidy dots*

**}**

*# end coordinates management (for random plotting and for stat display)*

*# dot display*

**if( ! is.null(dot.color)){**

**if(dot.tidy == FALSE){**

**if(isTRUE(all.equal(dot.border.size, 0))){** *# similar to dot.border.size == 0 but deals with floats (approx is enough)*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_point(data = dot.coord.rd3, mapping = ggplot2::aes\_string(x = "dot.x", y = "y", group = categ[length(categ)]), size = dot.size, color = dot.coord.rd3$dot.color, alpha = dot.alpha, pch = 16))** *# group used in aesthetic to do not have it in the legend. Here ggplot2::scale\_discrete\_manual() cannot be used because of the group easthetic*

**}else{**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_point(data = dot.coord.rd3, mapping = ggplot2::aes\_string(x = "dot.x", y = "y", group = categ[length(categ)]), stroke = dot.border.size, size = dot.size, fill = dot.coord.rd3$dot.color, alpha = dot.alpha, pch = 21))** *# group used in aesthetic to do not have it in the legend. Here ggplot2::scale\_discrete\_manual() cannot be used because of the group easthetic*

**}**

**}else if(dot.tidy == TRUE){**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_dotplot(data = dot.coord, mapping = ggplot2::aes\_string(x = categ[1], y = "y", color = categ[length(categ)]), position = ggplot2::position\_dodge(width = bar.width), binaxis = "y", stackdir = "center", alpha = dot.alpha, fill = dot.coord[rev(order(dot.coord[, categ[1]], decreasing = TRUE)), "dot.color"], show.legend = FALSE, binwidth = (ylim[2] - ylim[1]) / dot.bin.nb))** *# very weird behavior of geom\_dotplot, because data1 seems reorderer according to x = categ[1] before plotting. Thus, I have to use fill = dot.coord[rev(order(dot.coord[, categ[1]], decreasing = TRUE)), "dot.color"] to have the good corresponding colors # show.legend option do not remove the legend, only the aesthetic of the legend (dot, line, etc.)*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_discrete\_manual(aesthetics = "color", name = categ.legend.name, values = if(isTRUE(all.equal(dot.border.size, 0))){as.character(levels(dot.coord[rev(order(dot.coord[, categ[1]], decreasing = TRUE)), "dot.color"]))}else{rep("black", length(categ.color))}))** *# values = rep("black", length(categ.color)) are the values of color (which is the border color of dots), and this modify the border color on the plot. BEWARE: values = categ.color takes the numbers to make the colors if categ.color is a factor. BEWARE: , guide = ggplot2::guide\_legend(override.aes = list(fill = levels(dot.color))) here*

*# coordinates of tidy dots*

**tempo.coord <- ggplot2::ggplot\_build(eval(parse(text = paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + "))))$data** *# to have the tidy dot coordinates*

**if(length(which(sapply(tempo.coord, FUN = nrow) == nrow(data1))) > 1){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": MORE THAN 2 COMPARTMENT WITH NROW EQUAL TO nrow(data1) IN THE tempo.coord LIST (FOR TIDY DOT COORDINATES). CODE HAS TO BE MODIFIED\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}else{**

**dot.coord.tidy1 <- tempo.coord[[which(sapply(tempo.coord, FUN = nrow) == nrow(data1))]]**

**}**

**tempo.bar.coord <- merge(bar.coord, unique(dot.coord[, c("group", categ)]), by = intersect("group", "group"), sort = FALSE)** *# add the categ in bar.coord. BEWARE: by = intersect("group", "group") because group is enough as only one value of x per group number in bar.coord. Thus, no need to consider fill*

**if(nrow(tempo.bar.coord) != nrow(bar.coord)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": THE merge() FUNCTION DID NOT RETURN A CORRECT tempo.bar.coord DATA FRAME. CODE HAS TO BE MODIFIED\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**dot.coord.tidy2 <- merge(dot.coord.tidy1, tempo.bar.coord[c("fill", "group", "x", categ)], by = intersect("group", "group"), sort = FALSE)** *# send the coord of the bars into the coord data.frame of the dots (in the column x.y). BEWARE: by = intersect("group", "group") because group is enough as only one value of x per group number in bar.coord. Thus, no need to consider fill*

**if(nrow(dot.coord.tidy2) != nrow(dot.coord)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": THE merge() FUNCTION DID NOT RETURN A CORRECT dot.coord.tidy2 DATA FRAME. CODE HAS TO BE MODIFIED\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**if(length(categ) == 1){**

**tempo.data1 <- unique(data.frame(data1[categ[1]], group = as.integer(factor(as.numeric(data1[, categ[1]])))))** *# categ[2] first if categ[2] is used to make the categories in ggplot and categ[1] is used to make the x-axis*

**names(tempo.data1)[names(tempo.data1) == categ[1]] <- paste0(categ[1], ".check")**

**verif <- paste0(categ[1], ".check")**

**}else if(length(categ) == 2){**

**tempo.data1 <- unique(data.frame(data1[c(categ[1], categ[2])], group = as.integer(factor(paste0(as.numeric(data1[, categ[2]]), ".", as.numeric(data1[, categ[1]]))))))** *# categ[2] first if categ[2] is used to make the categories in ggplot and categ[1] is used to make the x-axis*

**names(tempo.data1)[names(tempo.data1) == categ[1]] <- paste0(categ[1], ".check")**

**names(tempo.data1)[names(tempo.data1) == categ[2]] <- paste0(categ[2], ".check")**

**verif <- c(paste0(categ[1], ".check"), paste0(categ[2], ".check"))**

**}else{**

**tempo.cat <- paste0("\n\n============\n\nERROR IN ", function.name, ": CODE INCONSISTENCY 7\n\n============\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**dot.coord.tidy3 <- merge(dot.coord.tidy2, tempo.data1, by = "group", sort = FALSE)** *# send the factors of data1 into coord*

**if(nrow(dot.coord.tidy3) != nrow(dot.coord) | ( ! fun\_comp\_2d(dot.coord.tidy3[categ], dot.coord.tidy3[verif])$identical.content)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": THE merge() FUNCTION DID NOT RETURN A CORRECT dot.coord.tidy3 DATA FRAME. CODE HAS TO BE MODIFIED\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

*# end coordinates of tidy dots*

**}**

**}**

*# end dot display*

*# stat display*

*# layer after dots but ok, behind dots on the plot*

**if( ! is.null(stat.disp)){**

**if(stat.disp == "top"){**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::annotate(geom = "text", x = bar.coord$x, y = ylim[2], label = fun\_round(bar.coord$y, 2), size = stat.size, color = "black", hjust = ifelse(vertical == TRUE, 0.5, 1.1), vjust = ifelse(vertical == TRUE, 1.1, 0.5)))** *# beware: no need of order() for labels because bar.coord$x set the order. For justification, see https://stackoverflow.com/questions/7263849/what-do-hjust-and-vjust-do-when-making-a-plot-using-ggplot*

**}else if(stat.disp == "above"){**

*# stat coordinates*

**if( ! is.null(dot.color)){** *# for text just above max dot*

**if(dot.tidy == FALSE){**

**tempo.stat.ini <- dot.coord.rd3**

**}else if(dot.tidy == TRUE){**

**tempo.stat.ini <- dot.coord.tidy3**

**}**

**stat.coord1 <- aggregate(x = tempo.stat.ini["y"], by = {x.env <- if(length(categ) == 1){list(tempo.stat.ini$group, tempo.stat.ini$x.y, tempo.stat.ini[, categ[1]])}else if(length(categ) == 2){list(tempo.stat.ini$group, tempo.stat.ini$x.y, tempo.stat.ini[, categ[1]], tempo.stat.ini[, categ[2]])} ; names(x.env) <- if(length(categ) == 1){c("group", "x.y", categ[1])}else if(length(categ) == 2){c("group", "x.y", categ[1], categ[2])} ; x.env}, FUN = min, na.rm = TRUE)**

**names(stat.coord1)[names(stat.coord1) == "y"] <- "dot.min"**

**stat.coord2 <- aggregate(x = tempo.stat.ini["y"], by = {x.env <- if(length(categ) == 1){list(tempo.stat.ini$group, tempo.stat.ini$x.y, tempo.stat.ini[, categ[1]])}else if(length(categ) == 2){list(tempo.stat.ini$group, tempo.stat.ini$x.y, tempo.stat.ini[, categ[1]], tempo.stat.ini[, categ[2]])} ; names(x.env) <- if(length(categ) == 1){c("group", "x.y", categ[1])}else if(length(categ) == 2){c("group", "x.y", categ[1], categ[2])} ; x.env}, FUN = max, na.rm = TRUE)**

**names(stat.coord2) <- paste0(names(stat.coord2), "\_from.dot.max")**

**names(stat.coord2)[names(stat.coord2) == "y\_from.dot.max"] <- "dot.max"**

**stat.coord3 <- cbind(bar.coord[order(bar.coord$x), ], stat.coord1[order(stat.coord1$x.y), ], stat.coord2[order(stat.coord2$x.y), ])** *# should be ok to use bar.coord$x and stat.coord$x.y to assemble the two data frames because x coordinates of the bars. Thus, we cannot have identical values*

**if( ! all(identical(round(stat.coord3$x, 9), round(stat.coord3$x.y, 9)))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": FUSION OF bar.coord, stat.coord1 AND stat.coord2 ACCORDING TO bar.coord$x, stat.coord1$x.y AND stat.coord2$x.y IS NOT CORRECT. CODE HAS TO BE MODIFIED\n\n================\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**dot.text.coord <- stat.coord3[, c("x", "group", "dot.min", "dot.max")]**

**names(dot.text.coord)[names(dot.text.coord) == "dot.min"] <- "text.min.pos"**

**names(dot.text.coord)[names(dot.text.coord) == "dot.max"] <- "text.max.pos"**

**}**

**if( ! is.null(error.disp)){** *# for text just above error bars*

**if(length(categ) == 1){**

**tempo.data1 <- unique(data.frame(data1[categ[1]], group = as.integer(factor(as.numeric(data1[, categ[1]])))))** *# categ[2] first if categ[2] is used to make the categories in ggplot and categ[1] is used to make the x-axis*

**if( ! identical(stat[order(stat[, categ[1]]), categ[1]], tempo.data1[order(tempo.data1[, categ[1]]), categ[1]])){**

**tempo.cat <- paste0("\n\n============\n\nERROR IN ", function.name, ": CODE PROBLEM IN TRYING TO ASSEMBLE stat AND tempo.data1\n\n============\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}else{**

**names(tempo.data1)[names(tempo.data1) == categ[1]] <- paste0(categ[1], ".check")**

**names(tempo.data1)[names(tempo.data1) == "group"] <- "group.check"**

**stat.coord4 <- cbind(stat[order(stat[, categ[1]]), ], tempo.data1[order(tempo.data1[, paste0(categ[1], ".check")]), ])**

**}**

**}else if(length(categ) == 2){**

**tempo.data1 <- unique(data.frame(data1[c(categ[1], categ[2])], group = as.integer(factor(paste0(as.numeric(data1[, categ[2]]), ".", as.numeric(data1[, categ[1]]))))))** *# categ[2] first if categ[2] is used to make the categories in ggplot and categ[1] is used to make the x-axis*

**if( ! fun\_comp\_2d(stat[order(stat[, categ[1]], stat[, categ[2]]), c(categ[1], categ[2])], tempo.data1[order(tempo.data1[, categ[1]], tempo.data1[, categ[2]]), c(categ[1], categ[2])])$identical.content){**

**tempo.cat <- paste0("\n\n============\n\nERROR IN ", function.name, ": CODE PROBLEM IN TRYING TO ASSEMBLE stat AND tempo.data1\n\n============\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}else{**

**names(tempo.data1)[names(tempo.data1) == categ[1]] <- paste0(categ[1], ".check")**

**names(tempo.data1)[names(tempo.data1) == categ[2]] <- paste0(categ[2], ".check")**

**names(tempo.data1)[names(tempo.data1) == "group"] <- "group.check"**

**stat.coord4 <- cbind(stat[order(stat[, categ[1]], stat[, categ[2]]), ], tempo.data1[order(tempo.data1[, paste0(categ[1], ".check")], tempo.data1[,paste0(categ[2], ".check")]), ])**

**}**

**}else{**

**tempo.cat <- paste0("\n\n============\n\nERROR IN ", function.name, ": CODE INCONSISTENCY 8\n\n============\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**if( ! identical(bar.coord$group[order(bar.coord$group)], stat.coord4$group.check[order(stat.coord4$group.check)])){**

**tempo.cat <- paste0("\n\n============\n\nERROR IN ", function.name, ": CODE PROBLEM IN TRYING TO ASSEMBLE bar.coord AND stat.coord4\n\n============\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}else{**

**stat.coord5 <- cbind(bar.coord[order(bar.coord$group), ], stat.coord4[order(stat.coord4$group.check), ])**

**error.text.coord <- stat.coord5[, c("x", "group", "ERROR.INF", "ERROR.SUP")] #**

**names(error.text.coord)[names(error.text.coord) == "ERROR.INF"] <- "text.min.pos"**

**names(error.text.coord)[names(error.text.coord) == "ERROR.SUP"] <- "text.max.pos"**

**}**

**}**

**if(( ! is.null(dot.color)) & ! is.null(error.disp)){** *# for text above max dot or error bar*

**stat.coord3 <- stat.coord3[order(stat.coord3$x), ]**

**stat.coord5 <- stat.coord5[order(stat.coord5$x), ]**

**if( ! identical(stat.coord3$group, stat.coord5$group)){**

**tempo.cat <- paste0("\n\n============\n\nERROR IN ", function.name, ": CODE PROBLEM IN TRYING TO ASSEMBLE stat.coord3 AND stat.coord5\n\n============\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}else{**

**stat.coord6 <- data.frame(stat.coord3, min.dot.error = mapply(FUN = min, stat.coord3$dot.min, stat.coord5$ERROR.INF, na.rm = TRUE))**

**stat.coord7 <- data.frame(stat.coord6, max.dot.error = mapply(FUN = max, stat.coord3$dot.max, stat.coord5$ERROR.SUP, na.rm = TRUE))**

**both.text.coord <- stat.coord7[, c("x", "group", "min.dot.error", "max.dot.error")] #**

**names(both.text.coord)[names(both.text.coord) == "min.dot.error"] <- "text.min.pos"**

**names(both.text.coord)[names(both.text.coord) == "max.dot.error"] <- "text.max.pos"**

**}**

**}**

**if(( ! is.null(dot.color)) & is.null(error.disp)){**

**text.coord <- dot.text.coord**

**}else if(is.null(dot.color) & ! is.null(error.disp)){**

**text.coord <- error.text.coord**

**}else if(( ! is.null(dot.color)) & ! is.null(error.disp)){**

**text.coord <- both.text.coord**

**}**

**if( ! (is.null(dot.color) & is.null(error.disp))){**

**bar.coord <- bar.coord[order(bar.coord$x), ]**

**text.coord <- text.coord[order(text.coord$x), ]** *# to be sure to have the two objects in the same order for x. BEWARE: cannot add identical(as.integer(text.coord$group), as.integer(bar.coord$group)) because with error, the correspondence between x and group is not the same*

**if( ! identical(text.coord$x, bar.coord$x)){**

**tempo.cat <- paste0("\n\n============\n\nERROR: text.coord AND bar.coord DO NOT HAVE THE SAME x COLUMN CONTENT\n\n============\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**}**

*# end stat coordinates*

*# stat display*

**if(is.null(dot.color) & is.null(error.disp)){** *# text just above bars*

*# performed twice: first for y values >=0, then y values < 0, because only a single value allowed for hjust anf vjust*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::annotate(geom = "text", x = bar.coord$x[bar.coord$y >= 0], y = bar.coord$y[bar.coord$y >= 0], label = fun\_round(bar.coord$y, 2)[bar.coord$y >= 0], size = stat.size, color = "black", hjust = ifelse(vertical == TRUE, 0.5, 0.5 - stat.dist), vjust = ifelse(vertical == TRUE, 0.5 - stat.dist, 0.5)))** *# beware: no need of order() for labels because bar.coord$x set the order*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::annotate(geom = "text", x = bar.coord$x[bar.coord$y < 0], y = bar.coord$y[bar.coord$y < 0], label = fun\_round(bar.coord$y, 2)[bar.coord$y < 0], size = stat.size, color = "black", hjust = ifelse(vertical == TRUE, 0.5, 0.5 + stat.dist), vjust = ifelse(vertical == TRUE, 0.5 + stat.dist, 0.5)))** *# beware: no need of order() for labels because bar.coord$x set the order*

**}else{** *# text just above error bars or dots*

*# I checked that text.coord and bar.coord have the same x and group column content. Thus, ok to use them together*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::annotate(geom = "text", x = text.coord$x[bar.coord$y >= 0], y = text.coord$text.max.pos[bar.coord$y >= 0], label = fun\_round(bar.coord$y, 2)[bar.coord$y >= 0], size = stat.size, color = "black", hjust = ifelse(vertical == TRUE, 0.5, 0.5 - stat.dist), vjust = ifelse(vertical == TRUE, 0.5 - stat.dist, 0.5)))** *# beware: no need of order() for labels because bar.coord$x set the order*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::annotate(geom = "text", x = text.coord$x[bar.coord$y < 0], y = text.coord$text.min.pos[bar.coord$y < 0], label = fun\_round(bar.coord$y, 2)[bar.coord$y < 0], size = stat.size, color = "black", hjust = ifelse(vertical == TRUE, 0.5, 0.5 + stat.dist), vjust = ifelse(vertical == TRUE, 0.5 + stat.dist, 0.5)))** *# beware: no need of order() for labels because bar.coord$x set the order*

**}**

*# end stat display*

**}else{**

**tempo.cat <- paste0("\n\n============\n\nERROR IN ", function.name, ": CODE INCONSISTENCY 9\n\n============\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**}**

*# end stat display*

*# y scale management (cannot be before dot plot management)*

**tempo.coord <- ggplot2::ggplot\_build(eval(parse(text = paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + "))))$layout$panel\_params[[1]]**

**tempo.scale <- fun\_scale(lim = ylim, n = ifelse(is.null(y.tick.nb), length(tempo.coord$y.major\_source), y.tick.nb))**

*# for the ggplot2 bug with ylog, this does not work: eval(parse(text = ifelse(vertical == FALSE & ylog == "log10", "ggplot2::scale\_x\_continuous", "ggplot2::scale\_y\_continuous")))*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_y\_continuous(**

**breaks = tempo.scale,**

**labels = if(ylog == "log10"){scales::trans\_format("identity", scales::math\_format(10^.x))}else if(ylog == "log2"){scales::trans\_format("identity", scales::math\_format(2^.x))}else if(ylog == "no"){ggplot2::waiver()}else{tempo.cat <- paste0("\n\n============\n\nERROR IN ", function.name, ": CODE INCONSISTENCY 10\n\n============\n\n") ; stop(tempo.cat, call. = FALSE)},**

**expand = c(0, 0),**

**limits = NA,**

**trans = ifelse(diff(ylim) < 0, "reverse", "identity")** *# equivalent to ggplot2::scale\_y\_reverse()*

**))**

**if(vertical == TRUE){**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::coord\_cartesian(ylim = ylim))** *# clip = "off" to have secondary ticks outside plot region does not work*

**}else{**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::coord\_flip(ylim = ylim))** *# clip = "off" to have secondary ticks outside plot region does not work*

**}**

*# secondary ticks (after ggplot2::coord\_cartesian() or ggplot2::coord\_flip())*

**tempo.coord <- ggplot2::ggplot\_build(eval(parse(text = paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + "))))$layout$panel\_params[[1]]**

*# no secondary ticks for log2. Play with ylim*

**if(ylog == "log10"){**

**ylim.order <- order(ylim)** *# to deal with inverse axis*

**ini.scipen <- options()$scipen**

**options(scipen = -1000)** *# force scientific format*

**power10.exp <- as.integer(substring(text = 10^ylim, first = (regexpr(pattern = "\\+|\\-", text = 10^ylim))))** *# recover the power of 10. Example recover 08 from 1e+08*

**mantisse <- as.numeric(substr(x = 10^ylim, start = 1, stop = (regexpr(pattern = "\\+|\\-", text = 10^ylim) - 2)))** *# recover the mantisse. Example recover 1.22 from 1.22e+08*

**options(scipen = ini.scipen)** *# restore the initial scientific penalty*

**tempo.tick.pos <- as.vector(outer(log10(2:10), 10^((power10.exp[1] - ifelse(diff(ylim.order) > 0, 1, -1)):(power10.exp[2] + ifelse(diff(ylim.order) > 0, 1, -1)))))**

**tempo.tick.pos <- sort(tempo.tick.pos, decreasing = ifelse(diff(ylim.order) > 0, FALSE, TRUE))**

**tempo.tick.pos <- log10(tempo.tick.pos[tempo.tick.pos >= min(10^ylim) & tempo.tick.pos <= max(10^ylim)])**

**if(any(is.na(tempo.tick.pos) | ! is.finite(tempo.tick.pos))){**

**tempo.cat <- paste0("\n\n============\n\nERROR IN ", function.name, ": CODE INCONSISTENCY 11\n\n============\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

*# if(vertical == TRUE){ # do not remove in case the bug is fixed*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::annotate(geom = "segment", y = tempo.tick.pos, yend = tempo.tick.pos, x = tempo.coord$x.range[1], xend = tempo.coord$x.range[1] + diff(tempo.coord$x.range) / 80))**

*# }else{ # not working because of the ggplot2 bug*

*# assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::annotate(geom = "segment", x = tempo.tick.pos, xend = tempo.tick.pos, y = tempo.coord$y.range[1], yend = tempo.coord$y.range[1] + diff(tempo.coord$y.range) / 80))*

*# }*

**}else if(( ! is.null(y.inter.tick.nb)) & ylog == "no"){**

**if(y.inter.tick.nb > 0){**

**if(vertical == TRUE){**

**ticks.pos <- suppressWarnings(as.numeric(tempo.coord$y.labels))** *# too difficult to predict the behavior of tempo.coord$x.major\_source depending on ylim neg or not, inv or not*

**if(any(is.na(ticks.pos))){**

**tempo.cat <- paste0("\n\n============\n\nERROR IN ", function.name, ": CODE INCONSISTENCY 12\n\n============\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**tick.dist <- mean(diff(ticks.pos), na.rm = TRUE)**

**minor.tick.dist <- tick.dist / (y.inter.tick.nb + 1)**

**minor.tick.pos <- seq(ticks.pos[1] - tick.dist, ticks.pos[length(ticks.pos)] + tick.dist, by = minor.tick.dist)**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::annotate(geom = "segment", y = minor.tick.pos, yend = minor.tick.pos, x = tempo.coord$x.range[1], xend = tempo.coord$x.range[1] + diff(tempo.coord$x.range) / 80))**

**}else{**

**ticks.pos <- suppressWarnings(as.numeric(tempo.coord$x.labels))***# too difficult to predict the behavior of tempo.coord$x.major\_source depending on ylim neg or not, inv or not*

**if(any(is.na(ticks.pos))){**

**tempo.cat <- paste0("\n\n============\n\nERROR IN ", function.name, ": CODE INCONSISTENCY 13\n\n============\n\n")**

**stop(tempo.cat, call. = FALSE)**

**}**

**tick.dist <- mean(diff(ticks.pos), na.rm = TRUE)**

**minor.tick.dist <- tick.dist / (y.inter.tick.nb + 1)**

**minor.tick.pos <- seq(ticks.pos[1] - tick.dist, ticks.pos[length(ticks.pos)] + tick.dist, by = minor.tick.dist)**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::annotate(geom = "segment", y = minor.tick.pos, yend = minor.tick.pos, x = tempo.coord$y.range[1], xend = tempo.coord$y.range[1] + diff(tempo.coord$y.range) / 80))**

**}**

**}**

**}**

*# end secondary ticks (after ggplot2::coord\_cartesian() or ggplot2::coord\_flip())*

*# end y scale management (cannot be before dot plot management)*

**if(plot == TRUE){**

**suppressWarnings(print(eval(parse(text = paste(paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + "), if(is.null(add)){NULL}else{add})))))**

**}else{**

**warn.count <- warn.count + 1**

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": PLOT NOT SHOWN AS REQUESTED")**

**warn <- paste0(ifelse(is.null(warn), tempo.warn, paste0(warn, "\n\n", tempo.warn)))**

**}**

*# end barplot*

**if(warn.print == TRUE & ! is.null(warn)){**

**warning(warn, call. = FALSE)**

**cat("\n\n")**

**}**

**if(return == TRUE){**

**output <- ggplot2::ggplot\_build(eval(parse(text = paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + "))))**

**output <- list(stat = stat, removed.row.nb = removed.row.nb, removed.rows = removed.rows, data = output$data, axes = output$layout$panel\_params[[1]], warn = paste0("\n", warn, "\n\n"))**

**return(output)**

**}**

**}**