**fun\_gg\_boxplot <- function(data1, y, categ, categ.class.order = NULL, categ.legend.name = NULL, categ.color = NULL, box.fill = FALSE, box.width = 0.5, box.notch = FALSE, box.mean = TRUE, whisker.kind = "std", whisker.width = 0.5, dot.color = "same", dot.tidy = FALSE, dot.bin.nb = 30, dot.jitter = 0.25, dot.size = 3, dot.alpha = 0.5, dot.border.size = 0.5, dot.border.color = NULL, 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 = 0.05, 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, path.lib = NULL){**

*# AIM*

*# ggplot2 vertical boxplot with the possibility to add background 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)*

*# hinges are not computed like in the classical boxplot() function of R.*

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

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

*# 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 boxs 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 box per class of categ1. If two column names (further refered to as categ1 and categ2), then one box per class of categ2, which form a group of boxs 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 box, 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 boxs, 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 boxplot (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 box frame. 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 boxs 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 boxs inside a group of boxs)*

*# box.fill: logical. Fill the box? If TRUE, the categ.color argument will be used to generate filled boxplot (the box frames being black) and if ever dot plotted, they will be over the boxes. If FALSE, the categ.color argument will be used to color the box frames, and if ever dot plotted, they will be beneath the boxes*

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

*# box.notch: logical. Notched boxplot? It TRUE, display notched boxplot, the notches corresponding approximately to the 95% confidence interval of the median (the notch interval is exactly 1.58 x Inter Quartile Range (IQR) / sqrt(n), with n the number of values that made the box). If notch intervals between two boxes do not overlap, it can be interpreted as significant median differences*

*# box.mean: logical. Add mean value? It TRUE, a dashed bar, additional to the solid median bar and corresponding to the mean value, is incorporated into each boxplot*

*# whisker.kind: length of the whiskers. Either "no" (no whiskers), or "std" (length of each whisker equal to 1.5 x Inter Quartile Range (IQR)), or "max" (length of the whiskers up or down to the most distant dot)*

*# whisker.width: numeric value (from 0 to 1) of the whisker width, with 0 meaning no whiskers and 1 meaning a width equal to the corresponding boxplot 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 box width interval. Not considered if dot.tidy is TRUE*

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

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

*# 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.border.color: single character color string defining the color of the dot border (same color for all the dots, whatever their categories). If dot.border.color = NULL, the border color will be the same as the dot color. A single integer is also accepted instead of a character string, that will be processed by fun\_gg\_palette()*

*# 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 boxs 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 median number above the corresponding box. Either NULL (no number shown), "top" (at the top of the figure region) or "above" (above each box)*

*# 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 from the box plot*

*# 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 boxs? BEWARE: will be automatically set to TRUE if ylog argument is other than "no". Indeed, not possible to have horizontal boxs 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 boxs (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 box 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*

*# path.lib: 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 boxplot 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 box and dot coordinates*

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

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

*# EXAMPLES*

*# DEBUGGING*

*# data1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10), group2 = rep(c("A", "B"), each = 10), box.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)$box.color ; box.fill = TRUE ; box.width = 0.5 ; box.notch = FALSE ; box.mean = TRUE ; whisker.kind = "std" ; 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 ; path.lib = 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 ; box.width = 0.5 ; 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 ; path.lib = NULL*

*# data1 <- data.frame(a = 1:20, group1 = rep(c("G", "H"), times = 10), group2 = rep(c("A", "B"), each = 10), box.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)$box.color ; box.width = 0.5 ; 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 ; path.lib = 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 ; box.width = 0.8 ; 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 = "MEAN" ; 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 ; path.lib = 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 ; box.width = 0.8 ; 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 = "MEAN" ; 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 ; path.lib = NULL*

*# function name*

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

*# end function name*

*# required function checking*

**if(length(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)**

**}**

**if(length(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)**

**}**

**if(length(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)**

**}**

**if(length(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)**

**}**

**if(length(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)**

**}**

**if(length(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)**

**}**

**if(length(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)**

**}**

**if(length(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)**

**}**

*# 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", "group", "group.check", "MEAN", "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)*

**warning <- NULL**

**arg.check <- NULL** *# for function debbuging*

**checked.arg.names <- NULL** *# for function debbuging*

**ee <- expression(arg.check <- c(arg.check, tempo$problem) , checked.arg.names <- c(checked.arg.names, tempo$param.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("\n\n================\n\nERROR IN ", function.name, ": DUPLICATED COLUMN NAMES OF data1 ARGUMENT NOT ALLOWED:\n", paste(names(data1)[duplicated(names(data1))], collapse = " "), "\n\n================\n\n")**

**cat(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("\n\n================\n\nERROR IN ", function.name, ": y ARGUMENT MUST BE A COLUMN NAME OF data1\n\n================\n\n")**

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

**cat(tempo.cat)**

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

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

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

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

**cat(tempo.cat)**

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

**}**

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

**for(i3 in 1:length(tempo.output$ini)){** *# a loop to be sure to take the good ones*

**names(data1)[names(data1) == tempo.output$ini[i3]] <- tempo.output$post[i3]**

**if(any(y == tempo.output$ini[i3])){**

**y[y == tempo.output$ini[i3]] <- tempo.output$post[i3]**

**tempo.warning <- paste0("FROM FUNCTION ", function.name, ": IN y ARGUMENT (COLUMN NAMES OF data1 ARGUMENT),\n", tempo.output$ini[i3], " HAS BEEN REPLACED BY ", tempo.output$post[i3], "\nBECAUSE RISK OF BUG AS SOME NAMES IN y ARGUMENT ARE RESERVED WORD USED BY THE ", function.name, " FUNCTION")**

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

**}**

**if(any(categ == tempo.output$ini[i3])){**

**categ[categ == tempo.output$ini[i3]] <- tempo.output$post[i3]**

**tempo.warning <- paste0("FROM FUNCTION ", function.name, ": IN categ ARGUMENT (COLUMN NAMES OF data1 ARGUMENT),\n", tempo.output$ini[i3], " HAS BEEN REPLACED BY ", tempo.output$post[i3], "\nBECAUSE RISK OF BUG AS SOME NAMES IN categ ARGUMENT ARE RESERVED WORD USED BY THE ", function.name, " FUNCTION")**

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

**}**

**}**

**tempo.warning <- paste0("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 = " "))**

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

**}**

*# 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, ]**

**tempo.warning <- paste0("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)")**

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

**}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]]))){**

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

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

**}**

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

**cat(tempo.cat)**

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

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

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

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

**}**

**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("\n\n================\n\nERROR IN ", function.name, ": categ.class.order ARGUMENT MUST BE A LIST OF MAX LENGTH 2\n\n================\n\n")**

**cat(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]])){**

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

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

**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("\n\n================\n\nERROR IN ", function.name, ": COMPARTMENT ", i3, " OF categ.class.order ARGUMENT CANNOT HAVE DUPLICATED CLASSES: ", paste(categ.class.order[[i3]], collapse = " "), "\n\n================\n\n")**

**cat(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("\n\n================\n\nERROR 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 = " "), "\n\n================\n\n")**

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

**cat(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("\n\n================\n\nERROR 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 = " "), "\n\n================\n\n")**

**cat(tempo.cat)**

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

**}**

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

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

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

**}**

*# end check the nature of color*

*# check the length of color*

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

**i0 <- 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[i0]]))){** *# here length(categ.color) is equal to the different number of categ*

**data1[, categ[i0]] <- factor(data1[, categ[i0]])** *# if already a factor, change nothing, if characters, levels according to alphabetical order*

**data1 <- data.frame(data1, categ.color = data1[, categ[i0]])**

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

**tempo.warning <- paste0("FROM FUNCTION ", function.name, ": IN ", categ[i0], " OF categ ARGUMENT, THE FOLLOWING COLORS:\n", paste(categ.color, collapse = " "), "\nHAVE BEEN ATTRIBUTED TO THESE CLASSES:\n", paste(levels(factor(data1[, categ[i0]])), collapse = " "))**

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

**}else if(length(categ.color) == length(data1[, categ[i0]])){***# 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[i0]])))*

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

**tempo.check <- unique(data1[ , c(categ[i0], "categ.color")])**

**if( ! (nrow(tempo.check) == length(unique(categ.color)) & nrow(tempo.check) == length(unique(data1[ , categ[i0]])))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": categ.color ARGUMENT HAS THE LENGTH OF data1 ROW NUMBER\nBUT IS INCORRECTLY ASSOCIATED TO EACH CLASS OF categ ", categ[i0], ":\n", paste(unique(mapply(FUN = "paste", data1[ ,categ[i0]], data1[ ,"categ.color"])), collapse = "\n"), "\n\n================\n\n")**

**cat(tempo.cat)**

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

**}else{**

**data1[, categ[i0]] <- factor(data1[, categ[i0]])** *# if already a factor, change nothing, if characters, levels according to alphabetical order*

**categ.color <- unique(categ.color[order(data1[, categ[i0]])])** *# Modif to have length(categ.color) equal to the different number of categ (length(categ.color) == length(levels(data1[, categ[i0]])))*

**tempo.warning <- paste0("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[i0], " AS:\n", paste(levels(factor(data1[, categ[i0]])), collapse = " "), "\n", paste(categ.color, collapse = " "))**

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

**}**

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

**data1[, categ[i0]] <- factor(data1[, categ[i0]])** *# 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[i0]])))**

**tempo.warning <- paste0("FROM FUNCTION ", function.name, ": categ.color ARGUMENT HAS LENGTH 1, MEANING THAT ALL THE DIFFERENT CLASSES OF ", categ[i0], "\n", paste(levels(factor(data1[, categ[i0]])), collapse = " "), "\nWILL HAVE THE SAME COLOR\n", paste(categ.color, collapse = " "))**

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

**}else{**

**tempo.cat <- paste0("\n\n================\n\nERROR 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[i0], " COLUMN. HERE IT IS COLOR LENGTH ", length(categ.color), " VERSUS CATEG LENGTH ", length(data1[, categ[i0]]), " AND CATEG CLASS LENGTH ", length(unique(data1[, categ[i0]])), "\nPRESENCE OF NA COULD BE THE PROBLEM\n\n================\n\n")**

**cat(tempo.cat)**

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

**}**

**}else{**

**i0 <- length(categ)** *# if only categ1, then colors for classes of categ1, if length(categ) == 2, then colors for classes of categ2*

**data1[, categ[i0]] <- factor(data1[, categ[i0]])** *# if already a factor, change nothing, if characters, levels according to alphabetical order*

**categ.color <- fun\_gg\_palette(length(levels(data1[, categ[i0]])))**

**data1 <- data.frame(data1, categ.color = data1[, categ[i0]])**

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

**tempo.warning <- paste0("FROM FUNCTION ", function.name, ": NULL categ.color ARGUMENT -> COLORS RESPECTIVELY ATTRIBUTED TO EACH CLASS OF ", categ[i0], " IN data1:\n", paste(categ.color, collapse = " "), "\n", paste(levels(data1[, categ[i0]]), collapse = " "))**

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

**}**

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

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

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

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

**tempo <- fun\_check(data = whisker.kind, options = c("no", "std", "max"), length = 1, fun.name = function.name) ; eval(ee)**

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

**cat(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 box color*

**tempo.warning <- paste0("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[i0], " AS:\n", paste(levels(factor(data1[, categ[i0]])), collapse = " "), "\n", paste(levels(factor(dot.color)), collapse = " "))**

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

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

**tempo.cat <- paste0("\n\n================\n\nERROR 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 = " "), "\n\n================\n\n")**

**cat(tempo.cat)**

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

**}**

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

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

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

**}**

*# end check the nature of color*

*# check the length of color*

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

**i0 <- 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[i0]]))){** *# here length(dot.color) is equal to the different number of categ*

**data1[, categ[i0]] <- factor(data1[, categ[i0]])** *# if already a factor, change nothing, if characters, levels according to alphabetical order*

**data1 <- data.frame(data1, dot.color = data1[, categ[i0]])**

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

**tempo.warning <- paste0("FROM FUNCTION ", function.name, ": IN ", categ[i0], " OF categ ARGUMENT, THE FOLLOWING COLORS:\n", paste(dot.color, collapse = " "), "\nHAVE BEEN ATTRIBUTED TO THESE CLASSES:\n", paste(levels(factor(data1[, categ[i0]])), collapse = " "))**

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

**}else if(length(dot.color) == length(data1[, categ[i0]])){***# 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[i0]])))*

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

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

**data1[, categ[i0]] <- factor(data1[, categ[i0]])** *# 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[i0]])))**

**tempo.warning <- paste0("FROM FUNCTION ", function.name, ": dot.color ARGUMENT HAS LENGTH 1, MEANING THAT ALL THE DIFFERENT CLASSES OF ", categ[i0], "\n", paste(levels(factor(data1[, categ[i0]])), collapse = " "), "\nWILL HAVE THE SAME COLOR\n", paste(dot.color, collapse = " "))**

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

**}else{**

**tempo.cat <- paste0("\n\n================\n\nERROR 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[i0], " COLUMN. HERE IT IS COLOR LENGTH ", length(dot.color), " VERSUS CATEG LENGTH ", length(data1[, categ[i0]]), " AND CATEG CLASS LENGTH ", length(unique(data1[, categ[i0]])), "\nPRESENCE OF NA COULD BE THE PROBLEM\n\n================\n\n")**

**cat(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.alpha, prop = TRUE, length = 1, 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)**

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

**tempo1 <- fun\_check(data = dot.border.color, class = "vector", mode = "character", length = 1, , fun.name = function.name, print = FALSE)**

**tempo2 <- fun\_check(data = dot.border.color, class = "vector", typeof = "integer", double.as.integer.allowed = TRUE, length = 1, , fun.name = function.name, print = FALSE)**

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

*# integer colors into gg\_palette*

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": dot.border.color MUST BE A SINGLE CHARACTER STRING OF COLOR OR A SINGLE INTEGER VALUE\n\n================\n\n")** *# integer possible because dealt above*

**cat(tempo.cat)**

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

**}else if(tempo1$problem == FALSE & tempo2$problem == TRUE & ! (all(dot.border.color %in% colors() | grepl(pattern = "^#", dot.border.color)))){** *# check that all strings of low.color start by #*

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": dot.border.color ARGUMENT MUST BE (1) A HEXADECIMAL COLOR STRING STARTING BY #, OR (2) A COLOR NAME GIVEN BY colors(), OR (3) AN INTEGER VALUE\nHERE IT IS: ", paste(unique(dot.border.color), collapse = " "), "\n\n================\n\n")**

**cat(tempo.cat)**

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

**}else if(tempo1$problem == TRUE & tempo2$problem == FALSE){** *# convert integers into colors*

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

**}**

*# end integer colors into gg\_palette*

**}**

**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("\n\n================\n\nERROR IN ", function.name, ": ylim ARGUMENT CANNOT CONTAIN -Inf OR Inf VALUES\n\n================\n\n")**

**cat(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"){**

**tempo.warning <- paste0("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")**

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

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

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

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

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

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

**tempo.cat <- paste0("\n\n================\n\nERROR 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 = " "), "\n\n================\n\n")**

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

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

**cat(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.warning <- 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")*

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

*# }*

**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**

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

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

**}**

**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("\n\n================\n\nERROR IN ", function.name, ": add ARGUMENT MUST START WITH \"+\": ", paste(unique(add), collapse = " "), "\n\n================\n\n")**

**cat(tempo.cat)**

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

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

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

**cat(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("\n\n================\n\nERROR IN ", function.name, ": add ARGUMENT MUST FINISH BY \")\": ", paste(unique(add), collapse = " "), "\n\n================\n\n")**

**cat(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(path.lib)){**

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

**if(tempo$problem == FALSE & ! all(dir.exists(path.lib))){**

**cat(paste0("\n\n============\n\nERROR IN ", function.name, ": \nDIRECTORY PATH INDICATED IN THE path.lib PARAMETER DOES NOT EXISTS: ", path.lib, "\n\n============\n\n"))**

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

**}**

**}**

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

**stop()** *# nothing else because print = TRUE by default in fun\_check()*

**}**

*# 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"), path.lib = path.lib)**

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

*# end package checking*

*# main code*

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

*# 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 box 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\_box*

**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 boxs*

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

**box.coord <- ggplot2::ggplot\_build(eval(parse(text = paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + "))))$data[[2]]**

**tempo.mean <- aggregate(x = dot.coord$y, by = list(dot.coord$group), FUN = mean, na.rm = TRUE)**

**names(tempo.mean)[names(tempo.mean) == "x"] <- "MEAN"**

**names(tempo.mean)[names(tempo.mean) == "Group.1"] <- "BOX"**

**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)**

**}else{**

**if( ! identical(tempo.mean[order(tempo.mean$BOX), ]$BOX, unique(dot.coord[order(dot.coord$group), ]$group))){**

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

**stop(tempo.cat)**

**}else{**

**tempo.mean <- data.frame(tempo.mean[order(tempo.mean$BOX), ], unique(dot.coord[order(dot.coord$group), categ[1], drop = FALSE]))**

**}**

**}**

**}**

*# end per box dots coordinates recovery*

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

*# 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 box 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\_box*

**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 boxs*

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

**box.coord <- ggplot2::ggplot\_build(eval(parse(text = paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + "))))$data[[2]]**

**tempo.mean <- aggregate(x = dot.coord$y, by = list(dot.coord$group), FUN = mean, na.rm = TRUE)**

**names(tempo.mean)[names(tempo.mean) == "x"] <- "MEAN"**

**names(tempo.mean)[names(tempo.mean) == "Group.1"] <- "BOX"**

**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)**

**}else{**

**if( ! identical(tempo.mean[order(tempo.mean$BOX), ]$BOX, unique(dot.coord[order(dot.coord$group), ]$group))){**

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

**stop(tempo.cat)**

**}else{**

**tempo.mean <- data.frame(tempo.mean[order(tempo.mean$BOX), ], unique(dot.coord[order(dot.coord$group), c(categ[1], categ[2])]))**

**}**

**}**

**}**

**}else{**

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

**stop(tempo.cat)**

**}**

*# stat output (will also serve for boxplot and mean display)*

**stat <- data.frame(X = box.coord$x, MIN = box.coord$ymin, QUART1 = box.coord$lower, MEDIAN = box.coord$middle, MEAN = tempo.mean$MEAN, QUART3 = box.coord$upper, MAX = box.coord$ymax, tempo.mean[colnames(tempo.mean) != "MEAN"])** *# add notch, outliers, group*

*# end stat output (will also serve for boxplot and mean display)*

*# range depending on medians and error boxs*

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

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

**tempo.warning <- paste0("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")**

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

**}**

**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 data1 ARGUMENTS ARE NA OR Inf ONLY\n\n================\n\n")**

**stop(tempo.cat)**

**}**

*# end range depending on medians and error boxs*

**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 medians 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 medians 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)**

**}**

*# width commputations*

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

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

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

**box.width2 <- box.width**

**}else{**

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

**stop(tempo.cat)**

**}**

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

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

*# end width commputations*

*# boxplot*

*# 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){**

**tempo.warning <- paste0("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")**

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

**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)}**

**))**

**}**

*# Contrary to fun\_gg\_bar(), cannot plot the boxplot right now, because I need the dots plotted first*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_boxplot(data = data1, mapping = ggplot2::aes\_string(x = categ[1], y = y, group = categ[length(categ)]), position = ggplot2::position\_dodge(width = NULL), color = NA, width = box.width, fill = NA))** *# this is to set the graph (i.e., a blanck boxplot to be able to use x coordinates to plot dots before boxes)*

*# end constant part*

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

*# boxs*

**box.coord <- ggplot2::ggplot\_build(eval(parse(text = paste0(paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + "), ' + ggplot2::geom\_boxplot(data = data1, mapping = ggplot2::aes\_string(x = categ[1], y = y, fill = categ[length(categ)]), position = ggplot2::position\_dodge(width = NULL), width = box.width)'))))$data[[2]] # box.coord <- ggplot2::ggplot\_build(eval(parse(text = paste0(paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + "), ' + ggplot2::geom\_boxplot(data = data1, mapping = ggplot2::aes\_string(x = categ[1], y = y, fill = categ[length(categ)]), position = ggplot2::position\_dodge(width = NULL), color = "black", width = box.width) + ggplot2::scale\_discrete\_manual(aesthetics = "fill", name = categ.legend.name, values = as.character(categ.color), guide = ggplot2::guide\_legend(override.aes = list(fill = categ.color)))'))))$data[[2]]** *# to have the summary statistics of the plot. Here because can be required for stat.disp when just box are plotted*

*# end boxs*

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

*# random dots*

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

**dot.coord.rd1 <- merge(dot.coord, box.coord[c("fill", "group", "x")], by = intersect("group", "group"), sort = FALSE)** *# rd for random. Send the coord of the boxs 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 box.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)**

**}**

**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 boxs 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)**

**}**

**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)**

**}**

*# 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)*

*# boxplot display before dot display if box.fill = TRUE*

**if(box.fill == TRUE){**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_boxplot(data = data1, mapping = ggplot2::aes\_string(x = categ[1], y = y, color = categ[length(categ)], fill = categ[length(categ)]), position = ggplot2::position\_dodge(width = NULL), width = box.width, outlier.shape = if( ! is.null(dot.color)){NA}else{21}, outlier.color = if( ! is.null(dot.color)){NA}else{dot.border.color}, outlier.fill = if( ! is.null(dot.color)){NA}else{NULL}, outlier.size = if( ! is.null(dot.color)){NA}else{dot.size}, outlier.stroke = if( ! is.null(dot.color)){NA}else{dot.border.size}, outlier.alpha = if( ! is.null(dot.color)){NA}else{dot.alpha}))** *# the color, size, etc. of the outliers are dealt here. outlier.color = NA to do not plot outliers when dots are already plotted*

**}**

*# end boxplot display before dot display if box.fill = TRUE*

*# dot display*

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

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

**if(dot.border.size == 0){**

**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, color = if(is.null(dot.border.color)){NULL}else{rep(dot.border.color, nrow(dot.coord.rd3))}, 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", linetype = categ[length(categ)]), position = ggplot2::position\_dodge(width = box.width), binaxis = "y", stackdir = "center", alpha = dot.alpha, fill = dot.coord[rev(order(dot.coord[, categ[1]], decreasing = TRUE)), "dot.color"], stroke = dot.border.size, color = if(is.null(dot.border.color)){NULL}else{rep(dot.border.color, nrow(dot.coord))}, show.legend = FALSE, binwidth = (ylim[2] - ylim[1]) / dot.bin.nb))** *# very weird behavior of geom\_dotplot, (1) because with aes group = (to avoid legend), the dot plotting is not good in term of coordinates, and (2) 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 = "linetype", name = categ.legend.name, values = rep(1, 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)**

**}else{**

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

**}**

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

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

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

**stop(tempo.cat)**

**}**

**dot.coord.tidy2 <- merge(dot.coord.tidy1, tempo.box.coord[c("fill", "group", "x", categ)], by = intersect("group", "group"), sort = FALSE)** *# send the coord of the boxs 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 box.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)**

**}**

**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)**

**}**

**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)**

**}**

*# end coordinates of tidy dots*

**}**

**}**

*# end dot display*

*# boxplot display (if box.fill = FALSE, otherwise, already plotted above)*

**if(box.fill == TRUE){**

**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, color = "black"))))** *# values are the values of color (which is the border color in geom\_box. BEWARE: values = categ.color takes the numbers to make the colors if categ.color is a factor*

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

**}else{**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_boxplot(data = data1, mapping = ggplot2::aes\_string(x = categ[1], y = y, color = categ[length(categ)], fill = categ[length(categ)]), position = ggplot2::position\_dodge(width = NULL), width = box.width, outlier.color = if( ! is.null(dot.color)){NA}else{dot.border.color}, outlier.fill = if( ! is.null(dot.color)){NA}else{NULL}, outlier.size = if( ! is.null(dot.color)){NA}else{dot.size}, outlier.stroke = if( ! is.null(dot.color)){NA}else{dot.border.size}, outlier.alpha = if( ! is.null(dot.color)){NA}else{dot.alpha})) # the color, size, etc. of the outliers are dealt here. outlier.color = NA to do not have outliers**

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

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

**}**

*# end boxplot display (if box.fill = FALSE, otherwise, already plotted above)*

*# 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 = box.coord$x, y = ylim[2], label = fun\_round(box.coord$middle, 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 box.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(box.coord[order(box.coord$x), ], stat.coord1[order(stat.coord1$x.y), ], stat.coord2[order(stat.coord2$x.y), ])** *# should be ok to use box.coord$x and stat.coord$x.y to assemble the two data frames because x coordinates of the boxs. 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 box.coord, stat.coord1 AND stat.coord2 ACCORDING TO box.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)**

**}**

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

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

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

**box.coord <- box.coord[order(box.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(box.coord$group)) because with error, the correspondence between x and group is not the same*

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

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

**stop(tempo.cat)**

**}**

**}**

*# end stat coordinates*

*# stat display*

**if(is.null(dot.color)){** *# text just above boxs*

*# 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 = box.coord$x[box.coord$middle >= 0], y = box.coord$middle[box.coord$middle >= 0], label = fun\_round(box.coord$middle, 2)[box.coord$middle >= 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 box.coord$x set the order*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::annotate(geom = "text", x = box.coord$x[box.coord$middle < 0], y = box.coord$middle[box.coord$middle < 0], label = fun\_round(box.coord$middle, 2)[box.coord$middle < 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 box.coord$x set the order*

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

*# I checked that text.coord and box.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$x[box.coord$middle >= 0], y = text.coord$text.max.pos[box.coord$middle >= 0], label = fun\_round(box.coord$middle, 2)[box.coord$middle >= 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 box.coord$x set the order*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::annotate(geom = "text", x = text.coord$x[box.coord$middle < 0], y = text.coord$text.min.pos[box.coord$middle < 0], label = fun\_round(box.coord$middle, 2)[box.coord$middle < 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 box.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)**

**}**

**}**

*# 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)},**

**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)**

**}**

*# 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)**

**}**

**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)**

**}**

**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{**

**tempo.warning <- paste0("FROM FUNCTION ", function.name, ": PLOT NOT SHOWN AS REQUESTED")**

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

**}**

*# end boxplot*

**if(warn.print == TRUE & ! is.null(warning)){**

**warning(warning)**

**}**

**if(return == TRUE){**

**output <- ggplot2::ggplot\_build(eval(parse(text = paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + "))))**

**output$data <- output$data[-1]** *# remove the first data because corresponds to the initial empty boxplot*

**output <- list(stat = stat, removed.row.nb = removed.row.nb, removed.rows = removed.rows, data = output$data, axes = output$layout$panel\_params[[1]], warnings = paste0("\n", warning, "\n\n"))**

**return(output)**

**}**

**}**