**# add legend width from scatter. Ok with facet?**

**# transfert the 2nd tick part to scatter**

**# improve grid -> put secondary grids. Then trasfert to scatter**

**# replace .categ.legend.name by box.legend.name**

**# replace dot.categ.legend.name by dot.legend.name**

**# facet in bold and with variable name https://github.com/rstudio/cheatsheets/blob/master/data-visualization-2.1.pdf**

**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.space = 0.1,**

**box.line.size = 0.5,**

**box.notch = FALSE,**

**box.alpha = 1,**

**box.mean = TRUE,**

**box.whisker.kind = "std",**

**box.whisker.width = 0,**

**dot.color = "black",**

**dot.categ = NULL,**

**dot.categ.class.order = NULL,**

**dot.categ.legend.name = NULL,**

**dot.tidy = FALSE,**

**dot.tidy.bin.nb = 50,**

**dot.jitter = 0.5,**

**dot.size = 3,**

**dot.alpha = 0.5,**

**dot.border.size = 0.5,**

**dot.border.color = NULL,**

**x.lab = NULL,**

**y.lab = NULL,**

**y.lim = NULL,**

**y.log = "no",**

**y.tick.nb = NULL,**

**y.second.tick.nb = NULL,**

**y.include.zero = FALSE,**

**y.top.extra.margin = 0.05,**

**y.bottom.extra.margin = 0.05,**

**stat.disp = NULL,**

**stat.disp.mean = FALSE,**

**stat.size = 4,**

**stat.dist = 2,**

**vertical = TRUE,**

**text.size = 12,**

**text.angle = 0,**

**title = "",**

**title.text.size = 8,**

**legend.show = TRUE,**

**legend.width = 0.5,**

**article = TRUE,**

**grid = FALSE,**

**return = FALSE,**

**return.ggplot = FALSE,**

**plot = TRUE,**

**add = NULL,**

**warn.print = FALSE,**

**lib.path = NULL**

**){**

*# AIM*

*# ggplot2 boxplot with the possibility to add background or foreground 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 box, create a factor column with a single class and specify the name of this column in the 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 (i.e., as categ1, knowing that categ2 must also be specified in this situation). See categ below*

*# with separated boxs (categ argument with only one element), box.width argument 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 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)*

*# The dot.alpha argument can alter the display of the color boxes when using pdf output*

*# ARGUMENTS*

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

*# y: character string of the data1 column name for y-axis (column containing numeric values). Numeric values will be split according to the classes of the column names indicated in the categ argument 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 be either 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: no empty classes allowed. To have a single box, create a factor column with a single class and specify the name of this column in the categ argument (here, 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 (i.e., as categ1, knowing that categ2 must also be specified in this situation)*

*# 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 color character 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 number of 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) as well as filled outlier dots (the dot border being controled by the dot.border.color argument) and if all the dots are plotted (argument dot.color other than NULL), they will be over the boxes. If FALSE, the categ.color argument will be used to color the box frames and the outlier dot borders, and if all the dots are 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.space: numeric value (from 0 to 1) indicating the box separation in grouped boxes. 0 means no space and 1 means boxes shrinked to a vertical line. Ignored if no grouped boxes*

*# box.line.size: numeric value of line size of boxes and whiskers (in mm)*

*# 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.alpha: numeric value (from 0 to 1) of box transparency (full transparent to full opaque, respectively). BEWARE: work only for the fill of boxplots, not for the frame. See https://github.com/tidyverse/ggplot2/issues/252*

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

*# box.whisker.kind: range 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)*

*# box.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 color character string for color of dots.*

*# If NULL, no dots plotted*

*# If "same", the dots will have the same colors as the respective boxplots*

*# Otherwise, colors will depend on the dot.categ argument. If dot.categ is NULL, then colors will be applied to each class of the last column name specified in categ. If dot.categ is non NULL, colors will be applied to each class of the column name specified in dot.categ. Color strings can be (1), (2) or (3) of categ.color argument, except that in the possibility (3), the rule "a single color per class of categ and a single class of categ per color", does not have to be respected (for instance, each dot can have a different color). See examples Put in example or leave here? -> Example: with categ = "Group1", dot.color = "red" and dot.categ = NULL, all the dots will be red, whatever the classes in Group1 column of data1, and no legend will be display for dots. With categ = c("Group1", "Group2"), dot.color = c("red", "blue") and dot.categ = NULL, the dots will be red for first class of Group2 and blue for the 2nd class of Group2, and no legend will be display for dots. With categ = c("Group1", "Group2"), dot.color = c("red", "blue") and dot.categ = "Group1", the dots will be red for first class of Group1 and blue for the 2nd class of Group1, and a legend will be display for dots*

*# dot.categ: optional single character string of a data1 column name (further refered to as categ3), which is associated to the dot.color argument to generate a legend for dots. If non NULL, then a legend will be created for the dots, in addition to the legend for the boxes. If NULL, no legend created and the colors of dot will depend on dot.color and categ arguments (see the explanation in dot.color)*

*# dot.categ.class.order: optional vector of character strings indicating the order of the classes of categ3. If dot.categ is non NULL and dot.categ.class.order is NULL, classes are displayed in the legend according to the alphabetical order. Ignored if dot.categ is NULL*

*# dot.categ.legend.name: optional character string of the legend title for categ3. If dot.categ.legend.name == NULL, categ3 value is used (name of the column in data1). Write "" if no legend required. Ignored if dot.categ is NULL*

*# dot.tidy: logical. Nice dot spreading? If TRUE, use the geom\_dotplot() function for a nice representation. BEWARE: change the true coordinates of dots that are aligned. Thus the gain in aestheticism is associated with a loss in precision that can be very important. If FALSE, dots are randomly spread, using the dot.jitter argument (see below) keeping the true dot coordinates*

*# dot.tidy.bin.nb: positive integer indicating the number of bins (i.e., nb of separations) of the y.lim range. Each dot will then be put in one of the bin, with the size the width of the bin. In other words, increase the number to have smaller dots. 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 (in mm). 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 (in mm). 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()*

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

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

*# y.lim: 2 numeric values indicating the range of the y-axis*

*# y.log: 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 transform the data, but just display ticks in a log scale manner. Thus, negative or zero values allowed in the y argument when y.log is "log2" or "log10". 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 (i.e., main ticks) on the y-axis (n argument of the the cute::fun\_scale() function). BEWARE: provide this number even if y.log is "log2" or "log10", which can be difficult to read (e.g., ..., 2^2, 2^2.5, 2^3, ...). If NULL and if y.log is "no", then the number of label values is set by ggplot2. If NULL and if y.log is "log2" or "log10", then the number of label values correspond to integer units between y.lim (e.g., ..., 2^1, 2^2, 2^3, ...)*

*# y.second.tick.nb: number of desired secondary ticks between main ticks. Ignored if y.log is other than "no" (log scale plotted). Use argument return = TRUE and see $plot$y.second.tick.values to have the values associated to secondary ticks. IF NULL, no secondary ticks*

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

*# y.top.extra.margin: single proportion (between 0 and 1) indicating if extra margins must be added to y.lim. If different from 0, add the range of the axis \* y.top.extra.margin (e.g., abs(y.lim[2] - y.lim[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.disp.mean: logical. Diplay means instead of medians ?*

*# 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*

*# vertical: logical. Vertical boxs? BEWARE: will be automatically set to TRUE if y.log 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)*

*# 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.*

*# title: character string of the graph title*

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

*# legend.show: logical. Show legend? Not considered if categ argument is NULL, because this already generate no legend, excepted if legend.width argument is non NULL. In that specific case (categ is NULL, legend.show is TRUE and legend.width is non NULL), an empty legend space is created. This can be useful when desiring graphs of exactly the same width, whatever they have legends or not*

*# legend.width: single proportion (between 0 and 1) indicating the relative width of the legend sector (on the right of the plot) relative to the width of the plot. Value 1 means that the window device width is split in 2, half for the plot and half for the legend. Value 0 means no room for the legend which will overlay the plot region. Write NULL to inactivate the legend sector. In such case, ggplot2 will manage the room required for the legend display, meaning that the width of the plotting region can vary between graphs, depending on the text in the legend*

*# article: logical. If TRUE, use a article theme (article like). If FALSE, use a classic related ggplot theme. Use the add argument (add = "+ggplot2::theme\_classic()" for the exact classic ggplot theme*

*# grid: logical. Draw horizontal lines in the background to better read the box values? Not considered if article == FALSE*

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

*# return.ggplot: logical. Return the ggplot object in the output list? Ignored if return is FALSE. See $ggplot in the RETURN section below for more details*

*# 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, facet, etc.). Ignored if NULL*

*# BEWARE: (1) the string must start with "+", (2) the string must finish with ")" and (3) each function must be preceded by "ggplot2::". Example: "+ ggplot2::coord\_flip() + ggplot2::theme\_bw()"*

*# If the character string contains the "ggplot2::theme" string, then the article argument of fun\_gg\_boxplot() (see above) is ignored with a warning*

*# Handle the add 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? ? If FALSE, warning messages are never printed, but can still be recovered in the returned list. Some of the warning messages (those delivered by the internal ggplot2 functions) are not apparent when using the argument plot = FALSE*

*# lib.path: character string indicating the absolute path of the required packages (see below). if NULL, the function will use the R library default folders*

*# REQUIRED PACKAGES*

*# ggplot2*

*# scales*

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

*# fun\_check()*

*# fun\_comp\_1d()*

*# fun\_comp\_2d()*

*# fun\_gg\_just()*

*# fun\_gg\_palette()*

*# fun\_inter\_ticks()*

*# fun\_name\_change()*

*# fun\_pack()*

*# fun\_round()*

*# fun\_scale()*

*# RETURN*

*# a boxplot if plot argument is TRUE*

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

*# $data: the initial data*

*# $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)*

*# $plot: the graphic box and dot coordinates*

*# $dots: dot coordinates*

*# $main.box: coordinates of boxes*

*# $median: median coordinates*

*# $sup.whisker: coordinates of top whiskers (y for base and y.end for extremities)*

*# $inf.whisker: coordinates of bottom whiskers (y for base and y.end for extremities)*

*# $sup.whisker.edge: coordinates of top whisker edges (x and xend)*

*# $inf.whisker.edge: coordinates of bottom whisker edges(x and xend)*

*# $mean: diamon mean coordinates (only if box.mean argument is TRUE)*

*# $stat.display.positive: coordinates of stat numbers when positive (only if stat.disp argument is TRUE)*

*# $stat.display.negative: coordinates of stat numbers when negative (only if stat.disp argument is TRUE)*

*# y.second.tick.positions: coordinates of secondary ticks (only if y.second.tick.nb argument is non NULL or if y.log argument is different from "no")*

*# y.second.tick.values: values of secondary ticks. NULL except if y.second.tick.nb argument is non NULL or if y.log argument is different from "no")*

*# $panel: the variable names used for the panels (NULL if no panels). BEWARE: NA can be present according to ggplot2 upgrade to v3.3.0*

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

*# $warn: the warning messages. Use cat() for proper display. NULL if no warning. BEWARE: some of the warning messages (those delivered by the internal ggplot2 functions) are not apparent when using the argument plot = FALSE*

*# $ggplot: ggplot object that can be used for reprint (use print($ggplot) or update (use $ggplot + ggplot2::...). NULL if return.ggplot argument is FALSE. Indeed, a non NULL $ggplot in the output list is sometimes annoying as the manipulation of the list print the plot*

*# EXAMPLE*

*# obs1 <- data.frame(x = 1:20, Group1 = rep(c("G", "H"), times = 10), Group2 = rep(c("A", "B"), each = 10)) ; fun\_gg\_boxplot(data1 = obs1, y = "x", categ = c("Group1", "Group2"), categ.class.order = list(NULL, c("B", "A")), categ.legend.name = "", categ.color = c("red", "blue"),box.fill = FALSE, box.width = 0.5, box.space = 0.1, box.line.size = 0.5, box.notch = FALSE, box.alpha = 1, box.mean = TRUE, box.whisker.kind = "std", box.whisker.width = 0, dot.color = "black", dot.categ = NULL, dot.categ.class.order = NULL, dot.categ.legend.name = NULL, dot.tidy = TRUE, dot.tidy.bin.nb = 50, dot.jitter = 0.5, dot.size = 3, dot.alpha = 0.5, dot.border.size = 0.5, dot.border.color = NULL, x.lab = NULL, y.lab = NULL, y.lim = NULL, y.log = "no", y.tick.nb = NULL, y.second.tick.nb = NULL, y.include.zero = FALSE, y.top.extra.margin = 0.05, y.bottom.extra.margin = 0.05, stat.disp = NULL, stat.disp.mean = FALSE, stat.size = 4, stat.dist = 2, vertical = TRUE, text.size = 12, text.angle = 0, title = "", title.text.size = 8, legend.width = 0.5, article = TRUE, grid = FALSE, return = FALSE, return.ggplot = FALSE, plot = TRUE, add = NULL, warn.print = TRUE, lib.path = NULL)*

*# DEBUGGING*

*# set.seed(1) ; obs1 <- data.frame(Time = c(rnorm(10), rnorm(10) + 2), Group1 = rep(c("G", "H"), each = 10)) ; set.seed(NULL) ; obs1$Time[1:10] <- NA ; data1 = obs1 ; y = "Time" ; categ = c("Group1") ; categ.class.order = NULL ; categ.legend.name = NULL ; categ.color = c("green") ; box.fill = FALSE ; box.width = 0.5 ; box.space = 0.1 ; box.notch = FALSE ; box.line.size = 0.5 ; box.alpha = 0.5 ; box.mean = TRUE ; box.whisker.kind = "std" ; box.whisker.width = 0.5 ; dot.color = "black" ; dot.categ = "Group1"; dot.categ.class.order = c("G", "H") ; dot.categ.legend.name = NULL ; dot.tidy = TRUE ; dot.tidy.bin.nb = 50 ; dot.jitter = 0.25 ; dot.size = 3 ; dot.alpha = 0.5 ; dot.border.size = 0.5 ; dot.border.color = NULL ; y.lim = NULL ; y.log = "no" ; y.tick.nb = NULL ; y.second.tick.nb = NULL ; y.include.zero = FALSE ; y.top.extra.margin = 0.05 ; y.bottom.extra.margin = 0.05 ; stat.disp = NULL ; stat.disp.mean = FALSE ; stat.size = 4 ; stat.dist = 2 ; x.lab = NULL ; y.lab = NULL ; vertical = TRUE ; text.size = 12 ; title = "" ; title.text.size = 8 ; legend.width = 0.5 ; text.angle = 0 ; article = FALSE ; grid = FALSE ; return = TRUE ; return.ggplot = FALSE ; plot = TRUE ; add = NULL ; warn.print = FALSE ; lib.path = NULL*

*# set.seed(1) ; obs1 <- data.frame(Time = c(rnorm(10), rnorm(10) + 2), Group1 = rep(c("G", "H"), each = 10), Group2 = rep(c("A", "B"), time = 10), Group3 = rep(c("I", "J"), time = 10)) ; set.seed(NULL) ; obs1$Time[1:10] <- NA ; data1 = obs1 ; y = "Time" ; categ = c("Group1", "Group2") ; categ.class.order = list(c("G", "H"), c("A", "B")); categ.legend.name = NULL ; categ.color = c("green", "blue") ; box.fill = FALSE ; box.width = 0.5 ; box.space = 0.1 ; box.notch = FALSE ; box.line.size = 0.5 ; box.alpha = 0.5 ; box.mean = TRUE ; box.whisker.kind = "std" ; box.whisker.width = 0.5 ; dot.color = "black" ; dot.categ = "Group1" ; dot.categ.class.order = NULL ; dot.categ.legend.name = NULL ; dot.tidy = TRUE ; dot.tidy.bin.nb = 30 ; dot.jitter = 0.25 ; dot.size = 3 ; dot.alpha = 0.5 ; dot.border.size = 0.5 ; dot.border.color = NULL ; y.lim = NULL ; y.log = "no" ; y.tick.nb = NULL ; y.second.tick.nb = NULL ; y.include.zero = FALSE ; y.top.extra.margin = 0.05 ; y.bottom.extra.margin = 0.05 ; stat.disp = NULL ; stat.disp.mean = FALSE ; stat.size = 4 ; stat.dist = 2 ; x.lab = NULL ; y.lab = NULL ; vertical = TRUE ; text.size = 12 ; title = "" ; title.text.size = 8 ; legend.width = 0.5 ; text.angle = 0 ; article = FALSE ; grid = FALSE ; return = FALSE ; return.ggplot = FALSE ; plot = TRUE ; add = NULL ; warn.print = FALSE ; lib.path = NULL*

*# set.seed(1) ; obs1 <- data.frame(Time = c(rnorm(10), rnorm(10) + 2), Group1 = rep(c("G", "H"), each = 10), Group2 = rep(c("A", "B"), time = 10)) ; set.seed(NULL) ; data1 = obs1 ; y = "Time" ; categ = c("Group1") ; categ.class.order = list(c("H", "G")); categ.legend.name = NULL ; categ.color = c("blue") ; box.fill = FALSE ; box.width = 0.5 ; box.space = 0.1 ; box.notch = TRUE ; box.line.size = 1 ; box.alpha = 1 ; box.mean = FALSE ; box.whisker.kind = "max" ; box.whisker.width = 0 ; dot.color = "black" ; dot.categ =* **"Group1"** *; dot.categ.class.order = NULL ; dot.categ.legend.name = NULL ; dot.tidy = TRUE ; dot.tidy.bin.nb = 30 ; dot.jitter = 0.25 ; dot.size = 3 ; dot.alpha = 0.5 ; dot.border.size = 0.5 ; dot.border.color = NULL ; y.lim = NULL ; y.log = "log10" ; y.tick.nb = NULL ; y.second.tick.nb = NULL ; y.include.zero = FALSE ; y.top.extra.margin = 0.05 ; y.bottom.extra.margin = 0.05 ; stat.disp = NULL ; stat.disp.mean = FALSE ; stat.size = 4 ; stat.dist = 2 ; x.lab = NULL ; y.lab = NULL ; vertical = TRUE ; text.size = 12 ; title = "" ; title.text.size = 8 ; legend.width = 0.5 ; text.angle = 0 ; article = FALSE ; grid = FALSE ; return = FALSE ; return.ggplot = FALSE ; plot = TRUE ; add = NULL ; warn.print = FALSE ; lib.path = NULL*

*# function name*

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

*# end function name*

*# required function checking*

**req.function <- c(**

**"fun\_comp\_2d",**

**"fun\_gg\_just",**

**"fun\_gg\_palette",**

**"fun\_name\_change",**

**"fun\_pack",**

**"fun\_check",**

**"fun\_round",**

**"fun\_scale",**

**"fun\_inter\_ticks"**

**)**

**for(i1 in req.function){**

**if(length(find(i1, mode = "function")) == 0){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, "\nREQUIRED ", i1, "() FUNCTION IS MISSING IN THE R ENVIRONMENT\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**}**

*# end required function checking*

*# reserved words to avoid bugs (names of dataframe columns used in this function)*

**reserved.words <- c("categ.check", "categ.color", "dot.color", "dot.categ", "dot.max", "dot.min", "group", "PANEL", "group.check", "MEAN", "tempo.categ1", "tempo.categ2", "text.max.pos", "text.min.pos", "x", "x.y", "y", "y.check", "y\_from.dot.max", "ymax", "tidy\_group")**

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

*# argument primary checking*

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

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

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

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

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

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

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

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

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

**}**

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

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

**}**

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

**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("categ.color ARGUMENT MUST BE A FACTOR OR CHARACTER VECTOR OR INTEGER VECTOR")** *# integer possible because dealt above*

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

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

**}**

**}**

**}**

**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.space, prop = TRUE, length = 1, fun.name = function.name) ; eval(ee)**

**tempo <- fun\_check(data = box.line.size, class = "vector", mode = "numeric", length = 1, neg.values = FALSE, 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.alpha, prop = TRUE, 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 = box.whisker.kind, options = c("no", "std", "max"), length = 1, fun.name = function.name) ; eval(ee)**

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

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

**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("dot.color MUST BE A FACTOR OR CHARACTER VECTOR OR INTEGER VECTOR")** *# integer possible because dealt above*

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

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

**}**

**}**

**}**

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

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

**}**

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

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

**}**

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

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

**}**

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

**tempo <- fun\_check(data = dot.tidy.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("dot.border.color MUST BE A SINGLE CHARACTER STRING OF COLOR OR A SINGLE INTEGER VALUE")** *# integer possible because dealt above*

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

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

**}**

**}**

**if( ! is.null(x.lab)){**

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

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

**}else{**

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

**}**

**}**

**if( ! is.null(y.lab)){**

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

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

**}else{**

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

**}**

**}**

**if( ! is.null(y.lim)){**

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

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

**tempo.cat <- paste0("y.lim ARGUMENT CANNOT CONTAIN -Inf OR Inf VALUES")**

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

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

**}**

**}**

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

**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("y.tick.nb ARGUMENT MUST BE A NON NULL POSITIVE INTEGER")**

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

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

**}**

**}**

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

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

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

**tempo.cat <- paste0("y.second.tick.nb ARGUMENT MUST BE A NON NULL POSITIVE INTEGER")**

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

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

**}**

**}**

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

**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.disp.mean, class = "logical", 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)**

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

**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 = 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 = 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)**

**if( ! is.null(legend.width)){**

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

**}**

**tempo <- fun\_check(data = article, 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 = return.ggplot, 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 = "^\\s\*\\+", add)){** *# check that the add string start by +*

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

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

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

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

**tempo.cat <- paste0("FOR EASIER FUNCTION DETECTION, add ARGUMENT MUST CONTAIN \"ggplot2::\" OR \"lemon::\" IN FRONT OF EACH GGPLOT2 FUNCTION: ", paste(unique(add), collapse = " "))**

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

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

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

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

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

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

**}**

**}**

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

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

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

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

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

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

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

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

**}**

**}**

**}**

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

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

**}**

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

*# end argument primary checking*

*# second round of checking and data preparation*

**warn <- NULL**

**warn.count <- 0**

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}**

**if( ! (y %in% names(data1))){**

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}else{**

**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)**

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

**tempo.cat <- paste0("ERROR IN ", function.name, "\ny ARGUMENT MUST BE NUMERIC COLUMN IN data1")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}**

**}**

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}else if( ! all(categ %in% names(data1))){**

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}**

*# reserved word checking*

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

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}**

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

**if(dot.categ %in% categ){**

**reserved.words <- c(reserved.words, paste0(dot.categ, "\_DOT"))** *# paste0(dot.categ, "\_DOT") is added to the reserved words because in such situation, a new column will be added to data1 that is named paste0(dot.categ, "\_DOT")*

**}**

**}**

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

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

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

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

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

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

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

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

**}**

**# BEWARE: names of y argument potentially replaced**

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

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

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

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

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

**}**

**# BEWARE: names of categ argument potentially replaced**

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

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

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

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

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

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

**}**

**}**

**# BEWARE: names of dot.categ argument potentially replaced**

**}**

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

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

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

**if( ! (is.null(add) | is.null(tempo.output$ini))){**

**if(grepl(x = add, pattern = paste(tempo.output$ini, collapse = "|"))){**

**tempo.cat <- paste0("ERROR IN ", function.name, "\nDETECTION OF COLUMN NAMES OF data1 IN THE add ARGUMENT STRING, THAT CORRESPOND TO RESERVED STRINGS FOR ", function.name, "\nCOLUMN NAMES HAVE TO BE CHANGED\nTHE PROBLEMATIC COLUMN NAMES ARE SOME OF THESE NAMES:\n", paste(tempo.output$ini, collapse = " "), "\nIN THE DATA FRAME OF data1 AND IN THE STRING OF add ARGUMENT, TRY TO REPLACE NAMES BY:\n", paste(tempo.output$post, collapse = " "), "\n\nFOR INFORMATION, THE RESERVED WORDS ARE:\n", paste(reserved.words, collapse = "\n"))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}**

**}**

**}**

*# end reserved word checking*

*# management of add containing facet*

**facet.categ <- NULL**

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

**facet.check <- TRUE**

**tempo <- unlist(strsplit(x = add, split = "\\s\*\\+\\s\*(ggplot2|lemon)\\s\*::\\s\*"))** *#*

**tempo <- sub(x = tempo, pattern = "^facet\_wrap", replacement = "ggplot2::facet\_wrap")**

**tempo <- sub(x = tempo, pattern = "^facet\_grid", replacement = "ggplot2::facet\_grid")**

**tempo <- sub(x = tempo, pattern = "^facet\_rep", replacement = "lemon::facet\_rep")**

**if(any(grepl(x = tempo, pattern = "ggplot2::facet\_wrap|lemon::facet\_rep\_wrap"))){**

**tempo1 <- suppressWarnings(eval(parse(text = tempo[grepl(x = tempo, pattern = "ggplot2::facet\_wrap|lemon::facet\_rep\_wrap")])))**

**facet.categ <- names(tempo1$params$facets)**

**tempo.text <- "facet\_wrap OR facet\_rep\_wrap"**

**facet.check <- FALSE**

**}else if(grepl(x = add, pattern = "ggplot2::facet\_grid|lemon::facet\_rep\_grid")){**

**tempo1 <- suppressWarnings(eval(parse(text = tempo[grepl(x = tempo, pattern = "ggplot2::facet\_grid|lemon::facet\_rep\_grid")])))**

**facet.categ <- c(names(tempo1$params$rows), names(tempo1$params$cols))**

**tempo.text <- "facet\_grid OR facet\_rep\_grid"**

**facet.check <- FALSE**

**}**

**if(facet.check == FALSE & ! all(facet.categ %in% names(data1))){** *# BEWARE: all(facet.categ %in% names(data1)) is TRUE when facet.categ is NULL*

**tempo.cat <- paste0("ERROR IN ", function.name, "\nDETECTION OF \"", tempo.text, "\" STRING IN THE add ARGUMENT BUT PROBLEM OF VARIABLE DETECTION (COLUMN NAMES OF data1)\nTHE DETECTED VARIABLES ARE:\n", paste(facet.categ, collapse = " "), "\nTHE data1 COLUMN NAMES ARE:\n", paste(names(data1), collapse = " "), "\nPLEASE REWRITE THE add STRING AND RERUN")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}**

**}**

*# end management of add containing facet*

*# conversion of categ columns in data1 into factors*

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

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

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

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

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

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

**tempo.warn <- paste0("(", warn.count,") IN categ NUMBER ", i1, " IN data1, THE CHARACTER COLUMN HAS BEEN CONVERTED TO FACTOR, WITH LEVELS ACCORDING TO THE ALPHABETICAL ORDER")**

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

**}**

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

**}**

**# OK: all the categ columns of data1 are factors from here**

*# end conversion of categ columns in data1 into factors*

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

**if(length(categ.class.order) != length(categ)){**

**tempo.cat <- paste0("ERROR IN ", function.name, "\ncateg.class.order ARGUMENT MUST BE A LIST OF LENGTH EQUAL TO LENGTH OF categ\nHERE IT IS LENGTH: ", length(categ.class.order), " VERSUS ", length(categ))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}else{**

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

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

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

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

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

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

**categ.class.order[[i3]] <- levels(data1[, categ[i3]])** *# character vector that will be used later*

**}else{**

**tempo <- fun\_check(data = categ.class.order[[i3]], data.name = paste0("COMPARTMENT ", i3 , " OF categ.class.order ARGUMENT"), class = "vector", mode = "character", length = length(levels(data1[, categ[i3]])), fun.name = function.name) ; eval(ee)** *# length(data1[, categ[i1]) -> if data1[, categ[i1] was initially character vector, then conversion as factor after the NA removal, thus class number ok. If data1[, categ[i1] was initially factor, no modification after the NA removal, thus class number ok*

**}**

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}else{**

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

**}**

**names(categ.class.order)[i3] <- categ[i3]**

**}**

**}**

**}else{**

**categ.class.order <- vector("list", length = length(categ))**

**tempo.categ.class.order <- NULL**

**for(i2 in 1:length(categ.class.order)){**

**categ.class.order[[i2]] <- levels(data1[, categ[i2]])**

**names(categ.class.order)[i2] <- categ[i2]**

**tempo.categ.class.order <- c(tempo.categ.class.order, ifelse(i2 != 1, "\n", ""), categ.class.order[[i2]])**

**}**

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

**tempo.warn <- paste0("(", warn.count,") THE categ.class.order SETTING IS NULL. ALPHABETICAL ORDER WILL BE APPLIED FOR BOX ORDERING:\n", paste(tempo.categ.class.order, collapse = " "))**

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

**}**

**# categ.class.order not NULL anymore (list)**

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

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

**tempo.warn <- paste0("(", warn.count,") THE categ.legend.name SETTING IS NULL. NAMES OF categ WILL BE USED: ", paste(categ, collapse = " "))**

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

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

**}**

**# categ.legend.name not NULL anymore (character string)**

*# management of categ.color*

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

*# check the nature of color*

*# 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 == FALSE){**

*# convert integers into colors*

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

**}**

*# end integer colors into gg\_palette*

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}**

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

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

**tempo.warn <- paste0("(", warn.count,") categ.color ARGUMENT CONTAINS NA")**

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

**}**

*# end check the nature of color*

*# check the length of color*

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

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

*# data1[, categ[categ.len]] <- factor(data1[, categ[categ.len]]) # not required because sure that is is a factor*

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

**data1$categ.color <- factor(data1$categ.color, labels = categ.color)**

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

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

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

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

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

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

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

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}else{**

*# data1[, categ[categ.len]] <- factor(data1[, categ[categ.len]]) # not required because sure that is is a factor*

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

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

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

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

**}**

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

*# data1[, categ[categ.len]] <- factor(data1[, categ[categ.len]]) # not required because sure that is is a factor*

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

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

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

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

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

**}else{**

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}**

**}else{**

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

*# data1[, categ[categ.len]] <- factor(data1[, categ[categ.len]]) # not required because sure that is is a factor*

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

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

**data1$categ.color <- factor(data1$categ.color, labels = categ.color)**

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

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

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

**}**

**# categ.color not NULL anymore**

**categ.color <- as.character(categ.color)**

**# categ.color is a character string representing the diff classes**

**data1$categ.color <- factor(data1$categ.color, levels = unique(categ.color))** *# ok because if categ.color is a character string, the order make class 1, class 2, etc. unique() because no duplicates allowed*

**# data1$categ.color is a factor with order of levels -> categ.color**

*# end management of categ.color*

*# management of dot.color*

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

*# optional legend of dot colors*

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

**ini.dot.categ <- dot.categ**

**if( ! dot.categ %in% names(data1)){** *# no need to use all() because length(dot.categ) = 1*

**tempo.cat <- paste0("ERROR IN ", function.name, "\ndot.categ ARGUMENT MUST BE A COLUMN NAME OF data1. HERE IT IS:\n", dot.categ)**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}else if(dot.categ %in% categ){** *# no need to use all() because length(dot.categ) = 1. Do not use dot.categ %in% categ[length(categ)] -> error*

*# management of dot legend if dot.categ %in% categ (because legends with the same name are joined in ggplot2***)**

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

**tempo.warn <- paste0("(", warn.count,") THE COLUMN NAME OF data1 INDICATED IN THE dot.categ ARGUMENT (", dot.categ, ") HAS BEEN REPLACED BY ", paste0(dot.categ, "\_DOT"), " TO AVOID MERGED LEGEND BY GGPLOT2")**

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

**data1 <- data.frame(data1, dot.categ = data1[, dot.categ])** *# dot.categ is not a column name of data1 (checked above with reserved words)*

**dot.categ <- paste0(dot.categ, "\_DOT")**

**names(data1)[names(data1) == "dot.categ"] <- dot.categ** *# paste0(dot.categ, "\_DOT") is not a column name of data1 (checked above with reserved words)*

*# tempo.cat <- paste0("ERROR IN ", function.name, "\ndot.categ ARGUMENT CANNOT BE A COLUMN NAME OF data1 ALREADY SPECIFIED IN THE categ ARGUMENT:\n", dot.categ, "\nINDEED, dot.categ ARGUMENT IS MADE TO HAVE MULTIPLE DOT COLORS NOT RELATED TO THE BOXPLOT CATEGORIES")*

*# stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)*

**}**

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

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

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

**tempo.cat <- paste0("ERROR IN ", function.name, "\ndot.categ COLUMN MUST BE A FACTOR OR CHARACTER VECTOR")** *#*

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}**

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

**# dot.categ column of data1 is factor from here**

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

**if(any(duplicated(dot.categ.class.order))){**

**tempo.cat <- paste0("ERROR IN ", function.name, "\ndot.categ.class.order ARGUMENT CANNOT HAVE DUPLICATED CLASSES: ", paste(dot.categ.class.order, collapse = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

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

**tempo.cat <- paste0("ERROR IN ", function.name, "\ndot.categ.class.order ARGUMENT MUST BE CLASSES OF dot.categ ARGUMENT\nHERE IT IS:\n", paste(dot.categ.class.order, collapse = " "), "\nFOR dot.categ.class.order AND IT IS:\n", paste(levels(data1[, dot.categ]), collapse = " "), "\nFOR dot.categ COLUMN (", ini.dot.categ, ") OF data1")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}else{**

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

**}**

**}else{**

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

**dot.categ.class.order <- unlist(categ.class.order[length(categ)])**

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

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

**tempo.warn <- paste0("(", warn.count,") THE dot.categ.class.order SETTING IS NULL AND dot.color IS \"same\". ORDER OF categ.class.order WILL BE APPLIED FOR LEGEND DISPLAY: ", paste(dot.categ.class.order, collapse = " "))**

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

**}else{**

**dot.categ.class.order <- sort(levels(data1[, dot.categ]))**

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

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

**tempo.warn <- paste0("(", warn.count,") THE dot.categ.class.order SETTING IS NULL. ALPHABETICAL ORDER WILL BE APPLIED FOR LEGEND DISPLAY: ", paste(dot.categ.class.order, collapse = " "))**

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

**}**

**}**

**# dot.categ.class.order not NULL anymore (character string) if dot.categ is not NULL**

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

**if( ! identical(ini.dot.categ, categ[length(categ)])){**

**tempo.cat <- paste0("ERROR IN ", function.name, "\nWHEN dot.color ARGUMENT IS \"same\", THE COLUMN NAME IN dot.categ ARGUMENT MUST BE IDENTICAL TO THE LAST COLUMN NAME IN categ ARGUMENT. HERE IT IS:\ndot.categ: ", paste(ini.dot.categ, collapse = " "), "\ncateg: ", paste(categ, collapse = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}else if( ! fun\_comp\_1d(unlist(categ.class.order[length(categ)]), dot.categ.class.order)$identical.content){**

**tempo.cat <- paste0("ERROR IN ", function.name, "\nWHEN dot.color ARGUMENT IS \"same\",\nLAST COMPARTMENT OF categ.class.order ARGUMENT AND dot.categ.class.order ARGUMENT CANNOT BE DIFFERENT:\nLAST COMPARTMENT OF categ.class.order: ", paste(unlist(categ.class.order[length(categ)]), collapse = " "), "\ndot.categ.class.order: ", paste(dot.categ.class.order, collapse = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}**

**}**

**for(i3 in 1:length(categ)){**

**if(identical(categ[i3], ini.dot.categ) & ! identical(unlist(categ.class.order[i3]), dot.categ.class.order) & identical(sort(unlist(categ.class.order[i3])), sort(dot.categ.class.order))){**

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

**tempo.warn <- paste0("(", warn.count,") THE dot.categ ARGUMENT SETTING IS PRESENT IN THE categ ARGUMENT SETTING, BUT ORDER OF THE CLASSES IS NOT THE SAME:\ncateg.class.order: ", paste(unlist(categ.class.order[i3]), collapse = " "), "\ndot.categ.class.order: ", paste(dot.categ.class.order, collapse = " "), "\nNOTE THAT ORDER OF categ.class.order IS THE ONE USED FOR THE AXIS REPRESENTATION")**

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

**}**

**}**

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

**dot.categ.legend.name <- if(ini.dot.categ %in% categ[length(categ)]){dot.categ}else{ini.dot.categ}** *#*

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

**tempo.warn <- paste0("(", warn.count,") THE dot.categ.legend.name SETTING IS NULL -> ", dot.categ.legend.name, " WILL BE USED AS LEGEND TITLE OF DOTS")**

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

**}**

**# dot.categ.legend.name not NULL anymore (character string)**

**}else{**

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

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

**tempo.warn <- paste0("(", warn.count,") THE dot.categ.class.order ARGUMENT IS NOT NULL, BUT IS THE dot.categ ARGUMENT\n-> dot.categ.class.order NOT CONSIDERED AS NO LEGEND WILL BE DRAWN")**

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

**}**

**# But dot.categ.class.order will be converted to NULL below (not now)**

**}**

*# end optional legend of dot colors*

*# check the nature of color*

*# 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 == FALSE){**

*# convert integers into colors*

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

**}**

*# 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*

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

**tempo.warn <- paste0("(", warn.count,") dot.color ARGUMENT HAS BEEN SET TO \"same\"\nTHUS, DOTS WILL HAVE THE SAME COLORS AS THE CORRESPONDING BOXPLOT")**

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

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

**tempo.cat <- paste0("ERROR IN ", function.name, "\ndot.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 = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}**

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

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

**tempo.warn <- paste0("(", warn.count,") dot.color ARGUMENT CONTAINS NA")**

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

**}**

*# end check the nature of color*

*# check the length of color*

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

*# optional legend of dot colors*

**if(length(dot.color) > 1 & length(unique(data1[, dot.categ])) != length(dot.color)){**

**tempo.cat <- paste0("ERROR IN ", function.name, "\ndot.color ARGUMENT IS NOT THE SAME LENGTH AS LEVELS OF dot.categ COLUMN (", dot.categ, "):\ndot.color: ", paste(dot.color, collapse = " "), "\ndot.categ LEVELS: ", paste(levels(data1[, dot.categ]), collapse = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}else if(length(dot.color) == 1 & length(dot.categ.class.order) > 1){** *# to deal with single color*

**dot.color <- rep(dot.color, length(dot.categ.class.order))**

**}**

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

**data1$dot.color <- factor(data1$dot.color, labels = dot.color)** *# do not use labels = unique(dot.color). Otherwise, we can have green1 green2 when dot.color is c("green", "green")*

**dot.color <- as.character(unique(data1$dot.color[order(data1[, dot.categ])]))** *# reorder the dot.color character vector*

**if(length(dot.color) == 1 & length(dot.categ.class.order) > 1){** *# to deal with single color*

**dot.color <- rep(dot.color, length(dot.categ.class.order))**

**}**

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

**if(length(unique(data1[ , "dot.color"])) > 1 & ( ! (nrow(tempo.check) == length(unique(data1[ , "dot.color"])) & nrow(tempo.check) == length(unique(data1[ , dot.categ]))))){** *# length(unique(data1[ , "dot.color"])) > 1 because if only one color, can be attributed to each class of dot.categ*

**tempo.cat <- paste0("ERROR IN ", function.name, "\ndot.color ARGUMENT IS INCORRECTLY ASSOCIATED TO EACH CLASS OF dot.categ (", dot.categ, ") COLUMN:\n", paste(unique(mapply(FUN = "paste", data1[ , dot.categ], data1[ ,"dot.color"])), collapse = "\n"))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}else{**

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

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

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

**}**

**# dot.color is a character string representing the diff classes of dot.categ**

**# data1$dot.color is a factor with order of levels -> dot.categ**

*# end optional legend of dot colors*

**}else{**

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

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

*# data1[, categ[categ.len]] <- factor(data1[, categ[categ.len]]) # not required because sure that is is a factor*

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

**data1$dot.color <- factor(data1$dot.color, labels = dot.color)**

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

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

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

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

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

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

*# data1[, categ[categ.len]] <- factor(data1[, categ[categ.len]]) # not required because sure that is is a factor*

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

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

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

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

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

**}else{**

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}**

*# end check the length of color*

**dot.color <- as.character(dot.color)**

**# dot.color is a character string representing the diff classes**

**data1$dot.color <- factor(data1$dot.color, levels = unique(dot.color))** *# ok because if dot.color is a character string, the order make class 1, class 2, etc. If dot.color is a column of data1, then levels will be created, without incidence, except if dot.categ specified (see below). unique() because no duplicates allowed*

**# data1$dot.color is a factor with order of levels -> dot.color**

**}**

*# end optional legend of dot colors*

**}else if(is.null(dot.color) & ! (is.null(dot.categ) & is.null(dot.categ.class.order) & is.null(dot.categ.legend.name))){**

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

**tempo.warn <- paste0("(", warn.count,") dot.categ OR dot.categ.class.order OR dot.categ.legend.name ARGUMENT HAS BEEN SPECIFIED BUT dot.color ARGUMENT IS NULL (NO DOT PLOTTED)")**

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

**}**

**# dot.color either NULL (no dot plotted) or character string (potentially representing the diff classes of dot.categ)**

**# data1$dot.color is either NA or a factor (with order of levels -> depending on dot.categ or categ[length(categ)], or other**

**if(is.null(dot.categ)){**

**dot.categ.class.order <- NULL # because not used anyway**

**}**

**# dot.categ.class.order either NULL if dot.categ is NULL (no legend displayed) or character string (potentially representing the diff classes of dot.categ)**

*# end management of dot.color*

**if(is.null(dot.color) & box.fill == FALSE & dot.alpha <= 0.025){**

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

**tempo.warn <- paste0("(", warn.count,") THE FOLLOWING ARGUMENTS WERE SET AS:\ndot.color = NULL (NOT ALL DOTS BUT ONLY POTENTIAL OUTLIER DOTS DISPLAYED)\nbox.fill = FALSE (NO FILLING COLOR FOR BOTH BOXES AND POTENTIAL OUTLIER DOTS)\ndot.alpha = ", fun\_round(dot.alpha, 4), "\n-> POTENTIAL OUTLIER DOTS MIGHT NOT BE VISIBLE BECAUSE ALMOST TRANSPARENT")**

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

**}**

**if(is.null(dot.color) & box.fill == FALSE & dot.border.size == 0){**

**tempo.cat <- paste0("ERROR IN ", function.name, "\nTHE FOLLOWING ARGUMENTS WERE SET AS:\ndot.color = NULL (NOT ALL DOTS BUT ONLY POTENTIAL OUTLIER DOTS DISPLAYED)\nbox.fill = FALSE (NO FILLING COLOR FOR BOTH BOXES AND POTENTIAL OUTLIER DOTS)\ndot.border.size = 0 (NO BORDER FOR POTENTIAL OUTLIER DOTS)\n-> THESE SETTINGS ARE NOT ALLOWED BECAUSE THE POTENTIAL OUTLIER DOTS WILL NOT BE VISIBLE")**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}**

**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$problem == FALSE & tempo2$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("ERROR IN ", function.name, "\ndot.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 = " "))**

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}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(y.log != "no"){**

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

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

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

**if( ! is.null(y.lim)){**

**if(any(y.lim <= 0)){**

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

**tempo.warn <- paste0("(", warn.count,") y.lim ARGUMENT CAN SPAN ZERO OR NEGATIVE VALUES IF y.log ARGUMENT IS SET TO ", y.log, " BECAUSE y.log DOES NOT TRANSFORM DATA, JUST MODIFIES THE AXIS SCALE")**

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

**}else if(any( ! is.finite(if(y.log == "log10"){10^y.lim}else{2^y.lim}))){**

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

**stop(paste0("\n\n================\n\n", tempo.cat, "\n\n================\n\n"), call. = FALSE)**

**}**

**}**

**}**

*# inactivated because y must already be log transformed data*

*# if(y.log != "no" & y.include.zero == TRUE){*

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

*tempo.warn <- paste0("(", warn.count,") y.log ARGUMENT SET TO ", y.log, " AND y.include.zero ARGUMENT SET TO TRUE -> y.include.zero ARGUMENT RESET TO FALSE BECAUSE NO 0 ALLOWED IN LOG SCALE")*

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

*# }*

**if(y.log != "no" & vertical == FALSE){**

**vertical <- TRUE**

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

**tempo.warn <- paste0("(", warn.count,") BECAUSE OF A BUG IN ggplot2, CANNOT FLIP BOXS HORIZONTALLY WITH A Y.LOG SCALE -> vertical ARGUMENT RESET TO TRUE")**

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

**}**

*# end second round of checking and data preparation*

*# package checking*

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

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

*# end package checking*

*# main code*

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

**column.check <- c(y, categ, "categ.color", if( ! is.null(dot.color)){"dot.color"}, if( ! is.null(dot.color) & ! is.null(dot.categ)){if(dot.categ != ini.dot.categ){dot.categ}}, if( ! is.null(facet.categ)){facet.categ})** *# dot.categ because can be a 3rd column of data1*

**if(any(is.na(data1[, column.check]))){**

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

**tempo.warn <- paste0("(", warn.count,") NA DETECTED IN COLUMNS ", paste(column.check, collapse = " "), " OF data1 AND CORRESPONDING ROWS REMOVED (SEE $removed.row.nb AND $removed.rows)")**

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

**for(i2 in 1:length(column.check)){**

**if(any(is.na(data1[, column.check[i2]]))){**

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

**tempo.warn <- paste0("(", warn.count,") NA REMOVAL DUE TO COLUMN ", column.check[i2], " OF data1")**

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

**}**

**}**

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

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

**column.check <- column.check[ ! column.check == y]** *# remove y to keep quali columns*

**if(length(removed.row.nb) != 0){**

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

**for(i3 in 1:length(column.check)){**

**if(any( ! unique(removed.rows[, column.check[i3]]) %in% unique(data1[, column.check[i3]]))){**

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

**tempo.warn <- paste0("(", warn.count,") IN COLUMN ", column.check[i3], " OF data1, THE FOLLOWING CLASSES HAVE DISAPPEARED AFTER NA REMOVAL (IF COLUMN USED IN THE PLOT, THIS CLASS WILL NOT BE DISPLAYED):\n", paste(unique(removed.rows[, column.check[i3]])[ ! unique(removed.rows[, column.check[i3]]) %in% unique(data1[, column.check[i3]])], collapse = " "))**

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

**}**

**}**

**}**

**count.categ <- 0**

**for(i2 in 1:length(column.check)){**

**if(column.check[i2] %in% categ){**

**count.categ <- count.categ + 1**

**}**

**if(column.check[i2] == categ[count.categ]){**

**categ.class.order[count.categ] <- list(levels(data1[, column.check[i2]])[levels(data1[, column.check[i2]]) %in% unique(data1[, column.check[i2]])]) # remove the absent color in the character vector**

**data1[, column.check[i2]] <- factor(as.character(data1[, column.check[i2]]), levels = unique(categ.class.order[[count.categ]]))**

**}**

**if( ! is.null(dot.color) & ! is.null(dot.categ)){** *# reminder : dot.categ cannot be a column name of categ anymore (because in that case dot.categ name is changed into "...\_DOT"*

**if(column.check[i2] == ini.dot.categ){**

**dot.categ.class.order <- levels(data1[, column.check[i2]])[levels(data1[, column.check[i2]]) %in% unique(data1[, column.check[i2]])] # remove the absent color in the character vector**

**data1[, column.check[i2]] <- factor(as.character(data1[, column.check[i2]]), levels = unique(dot.categ.class.order))**

**}**

**}**

**if(column.check[i2] == "categ.color"){**

**categ.color <- levels(data1[, column.check[i2]])[levels(data1[, column.check[i2]]) %in% unique(data1[, column.check[i2]])] # remove the absent color in the character vector**

**if(length(categ.color) == 1 & length(unlist(categ.class.order[length(categ)])) > 1){ # to deal with single color**

**categ.color <- rep(categ.color, length(unlist(categ.class.order[length(categ)])))**

**}**

**data1[, column.check[i2]] <- factor(as.character(data1[, column.check[i2]]), levels = unique(categ.color))**

**}**

**if(column.check[i2] == "dot.color"){**

**dot.color <- levels(data1[, column.check[i2]])[levels(data1[, column.check[i2]]) %in% unique(data1[, column.check[i2]])] # remove the absent color in the character vector**

**if(length(dot.color) == 1 & length(dot.categ.class.order) > 1){** *# to deal with single color. If dot.categ.class.order == NULL (which is systematically the case if dot.categ == NULL), no rep(dot.color, length(dot.categ.class.order)*

**dot.color <- rep(dot.color, length(dot.categ.class.order))**

**}**

**data1[, column.check[i2]] <- factor(as.character(data1[, column.check[i2]]), levels = unique(dot.color))**

**}**

**if(column.check[i2] %in% facet.categ){** *# works if facet.categ == NULL this method should keep the order of levels when removing some levels*

**tempo.levels <- levels(data1[, column.check[i2]])[levels(data1[, column.check[i2]]) %in% unique(as.character(data1[, column.check[i2]]))]**

**data1[, column.check[i2]] <- factor(as.character(data1[, column.check[i2]]), levels = tempo.levels)**

**}**

**}**

**}else{**

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

**removed.rows <- NULL**

**}**

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

*# y coordinates recovery (create ini.box.coord, dot.coord and modify data1)*

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

*# width commputations*

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

**box.space <- 0 # to inactivate the shrink that add space between grouped boxes, because no grouped boxes here**

*# end width commputations*

*# 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), eval(parse(text = paste0("ggplot2::ggplot()", if(is.null(add)){""}else{add}))))** *# add added here to have the facets*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_point(data = data1, mapping = ggplot2::aes\_string(x = categ[1], y = y, color = categ[1]), stroke = dot.border.size, size = dot.size, alpha = dot.alpha, shape = 21))**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_discrete\_manual(aesthetics = "color", name = categ.legend.name, values = if(is.null(dot.color)){rep(NA, length(unique(data1[, categ[1]])))}else if(length(dot.color) == 1){rep(dot.color, length(unique(data1[, categ[1]])))}else{dot.color}))**

**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, fill = categ[1]), coef = if(box.whisker.kind == "no"){0}else if(box.whisker.kind == "std"){1.5}else if(box.whisker.kind == "max"){Inf}))** *# fill because this is what is used with geom\_box # to easily have the equivalent of the grouped boxs*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_discrete\_manual(aesthetics = "fill", name = categ.legend.name, values = if(length(categ.color) == 1){rep(categ.color, length(unique(data1[, categ[1]])))}else{categ.color}))**

*# end per box dots coordinates recovery*

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

*# width commputations*

**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*

*# end width commputations*

*# 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), eval(parse(text = paste0("ggplot2::ggplot()", if(is.null(add)){""}else{add}))))** *# add added here to have the facets*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_point(data = data1, mapping = ggplot2::aes\_string(x = categ[1], y = y, color = categ[2]), stroke = dot.border.size, size = dot.size, alpha = dot.alpha, shape = 21))**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_discrete\_manual(aesthetics = "color", name = categ.legend.name, values = if(is.null(dot.color)){rep(NA, length(unique(data1[, categ[2]])))}else if(length(dot.color) == 1){rep(dot.color, length(unique(data1[, categ[2]])))}else{dot.color}))**

**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, fill = categ[2]), coef = if(box.whisker.kind == "no"){0}else if(box.whisker.kind == "std"){1.5}else if(box.whisker.kind == "max"){Inf}))** *# fill because this is what is used with geom\_box # to easily have the equivalent of the grouped boxs*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_discrete\_manual(aesthetics = "fill", name = categ.legend.name, values = if(length(categ.color) == 1){rep(categ.color, length(unique(data1[, categ[2]])))}else{categ.color}))**

*# end per box dots coordinates recovery*

**}else{**

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

**stop(tempo.cat)**

**}**

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

**dot.coord <- tempo.graph.info.ini$data[[1]]**

**ini.box.coord <- tempo.graph.info.ini$data[[2]]**

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

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

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

**names(tempo.mean)[names(tempo.mean) == "Group.2"] <- "PANEL"**

**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 = if(is.null(dot.color)){NA}else{data1[order(data1$categ.check, data1[, y]), "dot.color"]},**

**data1[order(data1$categ.check, data1[, y]), ][categ]** *# avoid the renaming below*

**)** *# 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( ! is.null(dot.categ)){**

**dot.coord <- data.frame(dot.coord, data1[order(data1$categ.check, data1[, y]), ][dot.categ])** *# avoid the renaming*

**}**

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

**dot.coord <- data.frame(dot.coord, data1[order(data1$categ.check, data1[, y]), ][facet.categ])** *# for facet panels*

**tempo.test <- NULL**

**for(i2 in 1:length(facet.categ)){**

**tempo.test <- paste0(tempo.test, ".", formatC(as.numeric(dot.coord[, facet.categ[i2]]), width = nchar(max(as.numeric(dot.coord[, facet.categ[i2]]), na.rm = TRUE)), flag = "0"))** *# convert factor into numeric with leading zero for proper ranking # merge the formatC() to create a new factor. The convertion to integer should recreate the correct group number. Here as.numeric is used and not as.integer in case of numeric in facet.categ (because comes from add and not checked by fun\_check, contrary to categ)*

**}**

**tempo.test <- as.integer(factor(tempo.test))**

**if( ! identical(as.integer(dot.coord$PANEL), tempo.test)){**

**tempo.cat <- paste0("\n\n================\n\nINTERNAL CODE ERROR IN ", function.name, "\nas.integer(dot.coord$PANEL) AND tempo.test MUST BE IDENTICAL. CODE HAS TO BE MODIFIED\n\n================\n\n")**

**stop(tempo.cat)**

**}**

**}**

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

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

**dot.coord <- data.frame(dot.coord, tidy\_group = data1[order(data1$categ.check, data1[, y]), ][, dot.categ])** *# avoid the renaming*

*# tidy\_group\_coord is to be able to fuse table when creating the table for dot coordinates*

**if(dot.categ %in% categ){**

**dot.coord <- data.frame(dot.coord, tidy\_group\_coord = dot.coord$group)**

**}else{**

**dot.coord <- data.frame(dot.coord, tidy\_group\_coord = as.integer(factor(paste0(**

**formatC(as.integer(dot.coord[, categ[1]]), width = nchar(max(as.integer(dot.coord[, categ[1]]), na.rm = TRUE)), flag = "0"),** *# convert factor into numeric with leading zero for proper ranking*

**".",**

**if(length(categ) == 2){formatC(as.integer(dot.coord[, categ[2]]), width = nchar(max(as.integer(dot.coord[, categ[2]]), na.rm = TRUE)), flag = "0")},** *# convert factor into numeric with leading zero for proper ranking*

**if(length(categ) == 2){"."},**

**formatC(as.integer(dot.coord[, dot.categ]), width = nchar(max(as.integer(dot.coord[, dot.categ]), na.rm = TRUE)), flag = "0")** *# convert factor into numeric with leading zero for proper ranking*

**)))** *# merge the 2 or 3 formatC() to create a new factor. The convertion to integer should recreate the correct group number*

**)** *# for tidy dot plots*

**}**

**}else{**

**dot.coord <- data.frame(dot.coord, tidy\_group = if(length(categ) == 1){**

**dot.coord[, categ]}else{as.integer(factor(paste0(**

**formatC(as.integer(dot.coord[, categ[1]]), width = nchar(max(as.integer(dot.coord[, categ[1]]), na.rm = TRUE)), flag = "0"),** *# convert factor into numeric with leading zero for proper ranking*

**".",**

**formatC(as.integer(dot.coord[, categ[2]]), width = nchar(max(as.integer(dot.coord[, categ[2]]), na.rm = TRUE)), flag = "0")***# convert factor into numeric with leading zero for proper ranking*

**)))** *# merge the 2 formatC() to create a new factor. The convertion to integer should recreate the correct group number*

**})** *# for tidy dot plots*

*# tidy\_group\_coord is to be able to fuse table when creating the table for dot coordinates*

**dot.coord <- data.frame(dot.coord, tidy\_group\_coord = dot.coord$group)**

**}**

**}**

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

**tempo.cat <- paste0("\n\n================\n\nINTERNAL CODE ERROR IN ", function.name, "\n(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$BOX, tempo.mean$PANEL), ]$BOX, unique(dot.coord[order(dot.coord$group, dot.coord$PANEL), c("group", "PANEL")])$group)){**

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

**stop(tempo.cat)**

**}else{**

**tempo <- unique(dot.coord[order(dot.coord$group, dot.coord$PANEL), c(categ, if( ! is.null(dot.color) & ! is.null(dot.categ)){if(dot.categ != ini.dot.categ){dot.categ}}, if( ! is.null(facet.categ)){facet.categ}), drop = FALSE])**

*# names(tempo) <- paste0(names(tempo), ".mean")*

**tempo.mean <- data.frame(tempo.mean[order(tempo.mean$BOX, tempo.mean$PANEL), ], tempo)**

**}**

**}**

**# at that stage, categ color and dot color are correctly attributed in data1, box.coord and dot.coord**

*# end y dot coordinates recovery (create ini.box.coord, dot.coord and modify data1)*

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

**ini.box.coord <- ini.box.coord[order(ini.box.coord$group), ]**

**stat <- data.frame(MIN = ini.box.coord$ymin, QUART1 = ini.box.coord$lower, MEDIAN = ini.box.coord$middle, QUART3 = ini.box.coord$upper, MAX = ini.box.coord$ymax, NOTCHUPPER = ini.box.coord$notchupper, NOTCHLOWER = ini.box.coord$notchlower, OUTLIERS = ini.box.coord["outliers"], COLOR = ini.box.coord$fill, stringsAsFactors = TRUE)** *# ini.box.coord["outliers"] written like this because it is a list. X coordinates not put now because several features to set*

**names(stat)[names(stat) == "outliers"] <- "OUTLIERS"**

**tempo.mean <- tempo.mean[order(tempo.mean$BOX), ]**

**if( ! fun\_comp\_2d(ini.box.coord[c("PANEL", "group")], tempo.mean[c("PANEL", "BOX")])$identical.content){**

**tempo.cat <- paste0("\n\n================\n\nINTERNAL CODE ERROR IN ", function.name, "\nini.box.coord[c(\"PANEL\", \"group\")] AND tempo.mean[c(\"PANEL\", \"BOX\")] MUST BE IDENTICAL. CODE HAS TO BE MODIFIED\n\n================\n\n")**

**stop(tempo.cat)**

**}else{**

**stat <- data.frame(stat[c("MIN", "QUART1", "MEDIAN")], MEAN = tempo.mean$MEAN, stat[c("QUART3", "MAX", "NOTCHUPPER", "NOTCHLOWER", "OUTLIERS")], tempo.mean[colnames(tempo.mean) != "MEAN"], stat["COLOR"], stringsAsFactors = TRUE)** *# ini.box.coord["outliers"] written like this because it is a list*

**}**

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

*# ylim range*

**if(is.null(y.lim)){**

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

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

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

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

**}**

**y.lim <- 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(y.lim %in% c(Inf, -Inf)))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, " COMPUTED y.lim CONTAINS Inf VALUES, BECAUSE VALUES FROM data1 ARGUMENTS ARE NA OR Inf ONLY\n\n================\n\n")**

**stop(tempo.cat)**

**}**

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

**y.lim <- sort(y.lim)**

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

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

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

**y.lim <- range(c(y.lim, 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*

**}**

**y.lim <- y.lim[y.lim.order]**

**if(any(is.na(y.lim))){**

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

**stop(tempo.cat)**

**}**

*# end ylim range*

*# drawing*

*# constant part*

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

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

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), eval(parse(text = paste0("ggplot2::ggplot()", if(is.null(add)){""}else{add}))))** *# add is directly put here to deal with additional variable of data, like when using facet\_grid. No problem if add is a theme, will be dealt below*

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

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

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

*# text angle management*

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

*# end text angle management*

**add.check <- TRUE**

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

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

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

**tempo.warn <- paste0("(", warn.count,") \"ggplot2::theme\" STRING DETECTED IN THE add ARGUMENT -> article ARGUMENT WILL BE IGNORED")**

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

**add.check <- FALSE**

**}**

**}**

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

*# BEWARE: not possible to add theme()several times. 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),**

**legend.key = ggplot2::element\_rect(color = "white", size = 1.5), # size of the frame of the legend**

**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 = "grey85")},**

**panel.grid.major.y = if(vertical == TRUE){ggplot2::element\_line(colour = "grey85")}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)},**

**strip.background = ggplot2::element\_rect(fill = NA, colour = NA) # for facet background**

**))**

**}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),**

**legend.key = ggplot2::element\_rect(color = "white", size = 1.5), # size of the frame of the legend**

**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)},**

**strip.background = ggplot2::element\_rect(fill = NA, colour = NA)**

**))**

**}**

**}else if(add.check == TRUE & article == 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),**

**legend.key = ggplot2::element\_rect(color = "white", size = 1.5), # size of the frame of the legend**

**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 = "grey85"),**

**panel.grid.major.y = ggplot2::element\_line(colour = "grey85"),**

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

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

**strip.background = ggplot2::element\_rect(fill = NA, colour = NA),**

**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*

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

*# boxs*

**tempo.graph.info <- 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, notch = box.notch, coef = if(box.whisker.kind == "no"){0}else if(box.whisker.kind == "std"){1.5}else if(box.whisker.kind == "max"){Inf}) + ggplot2::scale\_discrete\_manual(aesthetics = "fill", name = categ.legend.name, values = if(length(categ.color) == 1){rep(categ.color, length(unique(data1[, categ[length(categ)]])))}else{categ.color})'))))** *# will be recovered later again, when ylim will be considered*

**tempo.yx.ratio <- (tempo.graph.info$layout$panel\_params[[1]]$y.range[2] - tempo.graph.info$layout$panel\_params[[1]]$y.range[1]) / (tempo.graph.info$layout$panel\_params[[1]]$x.range[2] - tempo.graph.info$layout$panel\_params[[1]]$x.range[1])**

**box.coord <- tempo.graph.info$data[[2]]** *# to have the summary statistics of the plot. Contrary to ini.box.plot, now integrates ylim Here because can be required for stat.disp when just box are plotted*

**box.coord <- box.coord[order(box.coord$group), ]**

**if(stat.disp.mean == TRUE){ # for mean display**

**if( ! identical(tempo.mean$BOX, box.coord$group)){**

**tempo.cat <- paste0("\n\n============\n\nINTERNAL CODE ERROR IN ", function.name, "\ntempo.mean$BOX AND box.coord$group DO NOT HAVE THE SAME VALUE ORDER\n\n============\n\n")**

**stop(tempo.cat)**

**}else{**

**tempo <- c(categ, if( ! is.null(dot.color) & ! is.null(dot.categ)){if(dot.categ != ini.dot.categ){dot.categ}}, if( ! is.null(facet.categ)){facet.categ})**

**for(i3 in tempo){**

**names(tempo.mean)[names(tempo.mean) == i3] <- paste0(i3, ".mean")**

**}**

**box.coord <- data.frame(box.coord, tempo.mean)**

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

**tempo.warn <- paste0("(", warn.count,") MEAN VALUES INSTEAD OF MEDIAN VALUES DISPLAYED")**

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

**}**

**}**

*# width commputations*

**width.ini <- c(box.coord$xmax - box.coord$xmin)[1]** *# all the box widths are equal here. Only the first one taken*

**width.correct <- width.ini \* box.space / 2**

**if( ! identical(box.coord$group, stat$BOX)){**

**tempo.cat <- paste0("\n\n================\n\nINTERNAL CODE ERROR IN ", function.name, "\n(box.coord$group AND stat$BOX) MUST BE IDENTICAL. CODE HAS TO BE MODIFIED\n\n================\n\n")**

**stop(tempo.cat)**

**}else{**

**stat <- data.frame(**

**stat,**

**X = box.coord$x,**

**X\_BOX\_INF = box.coord$xmin + width.correct,**

**X\_BOX\_SUP = box.coord$xmax - width.correct,**

**X\_NOTCH\_INF = box.coord$x - (box.coord$x - (box.coord$xmin + width.correct)) / 2,**

**X\_NOTCH\_SUP = box.coord$x + (box.coord$x - (box.coord$xmin + width.correct)) / 2,**

**X\_WHISK\_INF = box.coord$x - (box.coord$x - (box.coord$xmin + width.correct)) \* box.whisker.width,**

**X\_WHISK\_SUP = box.coord$x + (box.coord$x - (box.coord$xmin + width.correct)) \* box.whisker.width,**

**# tempo.mean[colnames(tempo.mean) != "MEAN"], # already added above**

**stringsAsFactors = TRUE**

**)**

**stat$COLOR <- factor(stat$COLOR, levels = unique(categ.color))**

**if( ! all(stat$NOTCHUPPER < stat$QUART3 & stat$NOTCHLOWER > stat$QUART1) & box.notch == TRUE){**

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

**tempo.warn <- paste0("(", warn.count,") SOME NOTCHES ARE BEYOND BOX HINGES. TRY ARGUMENT box.notch = FALSE")**

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

**}**

**}**

**dot.jitter <- c((box.coord$xmax - width.correct) - (box.coord$xmin + width.correct))[1] \* dot.jitter** *# real dot.jitter. (box.coord$xmin + width.correct) - (box.coord$xmax - width.correct))[1] is the width of the box. Is equivalent to (box.coord$x - (box.coord$xmin + width.correct))[1] \* 2*

*# end width commputations*

*# end boxs*

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

*# random dots*

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

**dot.coord.rd1 <- merge(dot.coord, box.coord[c("fill", "PANEL", "group", "x")], by = c("PANEL", "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 = c("PANEL", "group") without fill column because PANEL & group columns are enough as only one value of x column per group number in box.coord. Thus, no need to consider fill column*

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

**tempo.cat <- paste0("\n\n================\n\nINTERNAL CODE ERROR IN ", function.name, "\nTHE 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(data1[, categ[1]])))** *# categ[1] is factor*

**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(**

**formatC(as.integer(data1[, categ[2]]), width = nchar(max(as.integer(data1[, categ[2]]), na.rm = TRUE)), flag = "0"),** *# convert factor into numeric with leading zero for proper ranking*

**".",**

**formatC(as.integer(data1[, categ[1]]), width = nchar(max(as.integer(data1[, categ[1]]), na.rm = TRUE)), flag = "0")***# convert factor into numeric with leading zero for proper ranking*

**)))** *# merge the 2 formatC() to create a new factor. The convertion to integer should recreate the correct group number*

**)**

**)** *# 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\nINTERNAL CODE ERROR IN ", function.name, "\nCODE INCONSISTENCY 6\n\n============\n\n")**

**stop(tempo.cat)**

**}**

**dot.coord.rd3 <- merge(dot.coord.rd2, tempo.data1, by = intersect("group", "group"), sort = FALSE)** *# send the factors of data1 into coord. BEWARE: I have replaced by = "group" by intersect("group", "group") because of an error due to wrong group group merging in dot.coord.rd3*

**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\nINTERNAL CODE ERROR IN ", function.name, "\nTHE 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 recovered during plotting (see dot.coord.tidy1 below)*

*# end tidy dots*

**}**

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

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

**coord.names <- NULL**

*# creation of the data frame for (main box + legend) and data frame for means*

**if(box.notch == FALSE){**

**for(i3 in 1:length(categ)){**

**if(i3 == 1){**

**tempo.polygon <- data.frame(GROUPX = c(t(stat[, rep(categ[i3], 5)])), stringsAsFactors = TRUE)**

**}else{**

**tempo.polygon <- cbind(tempo.polygon, c(t(stat[, rep(categ[i3], 5)])), stringsAsFactors = TRUE)**

**}**

**}**

**names(tempo.polygon) <- categ**

**tempo.polygon <- data.frame(X = c(t(stat[, c("X\_BOX\_INF", "X\_BOX\_SUP", "X\_BOX\_SUP", "X\_BOX\_INF", "X\_BOX\_INF")])), Y = c(t(stat[, c("QUART1", "QUART1", "QUART3", "QUART3", "QUART1")])), COLOR = c(t(stat[, c("COLOR", "COLOR", "COLOR", "COLOR", "COLOR")])), BOX = as.character(c(t(stat[, c("BOX", "BOX", "BOX", "BOX", "BOX")]))), tempo.polygon, stringsAsFactors = TRUE)**

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

**for(i4 in 1:length(facet.categ)){**

**tempo.polygon <- data.frame(tempo.polygon, c(t(stat[, c(facet.categ[i4], facet.categ[i4], facet.categ[i4], facet.categ[i4], facet.categ[i4])])), stringsAsFactors = TRUE)**

**names(tempo.polygon)[length(names(tempo.polygon))] <- facet.categ[i4]**

**}**

**}**

**}else{**

**for(i3 in 1:length(categ)){**

**if(i3 == 1){**

**tempo.polygon <- data.frame(GROUPX = c(t(stat[, rep(categ[i3], 11)])), stringsAsFactors = TRUE)**

**}else{**

**tempo.polygon <- cbind(tempo.polygon, c(t(stat[, rep(categ[i3], 11)])), stringsAsFactors = TRUE)**

**}**

**}**

**names(tempo.polygon) <- categ**

**tempo.polygon <- data.frame(X = c(t(stat[, c("X\_BOX\_INF", "X\_BOX\_SUP", "X\_BOX\_SUP", "X\_NOTCH\_SUP", "X\_BOX\_SUP", "X\_BOX\_SUP", "X\_BOX\_INF", "X\_BOX\_INF", "X\_NOTCH\_INF", "X\_BOX\_INF", "X\_BOX\_INF")])), Y = c(t(stat[, c("QUART1", "QUART1", "NOTCHLOWER", "MEDIAN", "NOTCHUPPER", "QUART3", "QUART3", "NOTCHUPPER", "MEDIAN", "NOTCHLOWER", "QUART1")])), COLOR = c(t(stat[, c("COLOR", "COLOR", "COLOR", "COLOR", "COLOR", "COLOR", "COLOR", "COLOR", "COLOR", "COLOR", "COLOR")])), BOX = as.character(c(t(stat[, c("BOX", "BOX", "BOX", "BOX", "BOX", "BOX", "BOX", "BOX", "BOX", "BOX", "BOX")]))), tempo.polygon, stringsAsFactors = TRUE)**

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

**for(i4 in 1:length(facet.categ)){**

**tempo.polygon <- data.frame(tempo.polygon, c(t(stat[, c(facet.categ[i4], facet.categ[i4], facet.categ[i4], facet.categ[i4], facet.categ[i4], facet.categ[i4], facet.categ[i4], facet.categ[i4], facet.categ[i4], facet.categ[i4], facet.categ[i4])])), stringsAsFactors = TRUE)**

**names(tempo.polygon)[length(names(tempo.polygon))] <- facet.categ[i4]**

**}**

**}**

**}**

**tempo.polygon$COLOR <- factor(tempo.polygon$COLOR, levels = unique(categ.color))**

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

**for(i3 in 1:length(categ)){**

**tempo.polygon[, categ[i3]] <- factor(tempo.polygon[, categ[i3]], levels = categ.class.order[[i3]])**

**}**

**}**

*# modified name of dot.categ column (e.g., "Group1\_DOT") must be included for boxplot using ridy dots*

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

**if(dot.categ != ini.dot.categ){**

**tempo.polygon <- data.frame(tempo.polygon, GROUPX = tempo.polygon[, ini.dot.categ])**

**names(tempo.polygon)[names(tempo.polygon) == "GROUPX"] <- dot.categ**

**}**

**}**

**tempo.diamon.mean <- data.frame(X = c(t(stat[, c("X", "X\_NOTCH\_INF", "X", "X\_NOTCH\_SUP", "X")])), Y = c(t(cbind(stat["MEAN"] - (stat[, "X"] - stat[, "X\_NOTCH\_INF"]) \* tempo.yx.ratio, stat["MEAN"], stat["MEAN"] + (stat[, "X"] - stat[, "X\_NOTCH\_INF"]) \* tempo.yx.ratio, stat["MEAN"], stat["MEAN"] - (stat[, "X"] - stat[, "X\_NOTCH\_INF"]) \* tempo.yx.ratio))), COLOR = c(t(stat[, c("COLOR", "COLOR", "COLOR", "COLOR", "COLOR")])), GROUP = c(t(stat[, c("BOX", "BOX", "BOX", "BOX", "BOX")])), stringsAsFactors = TRUE)**

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

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

**tempo.diamon.mean <- data.frame(tempo.diamon.mean, c(t(stat[, c(facet.categ[i3], facet.categ[i3], facet.categ[i3], facet.categ[i3], facet.categ[i3])])), stringsAsFactors = TRUE)**

**names(tempo.diamon.mean)[length(names(tempo.diamon.mean))] <- facet.categ[i3]**

**}**

**}**

**tempo.diamon.mean$COLOR <- factor(tempo.diamon.mean$COLOR, levels = unique(categ.color))**

*# end creation of the data frame for (main box + legend) and data frame for means*

**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, size = box.line.size, notch = box.notch, coef = if(box.whisker.kind == "no"){0}else if(box.whisker.kind == "std"){1.5}else if(box.whisker.kind == "max"){Inf}, alpha = box.alpha, 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. Finally, boxplot redrawn (see below)*

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

**data = tempo.polygon,**

**mapping = ggplot2::aes\_string(x = "X", y = "Y", group = "BOX", fill = categ[length(categ)], color = categ[length(categ)]),**

**size = box.line.size,**

**alpha = box.alpha**

**))**

**coord.names <- c(coord.names, "main.box")**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_segment(data = stat, mapping = ggplot2::aes(x = X, xend = X, y = QUART3, yend = MAX, group = categ[length(categ)]), color = "black", size = box.line.size, alpha = box.alpha))** *#*

**coord.names <- c(coord.names, "sup.whisker")**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_segment(data = stat, mapping = ggplot2::aes(x = X, xend = X, y = QUART1, yend = MIN, group = categ[length(categ)]), color = "black", size = box.line.size, alpha = box.alpha))** *#*

**coord.names <- c(coord.names, "inf.whisker")**

**if(box.whisker.width > 0){**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_segment(data = stat, mapping = ggplot2::aes(x = X\_WHISK\_INF, xend = X\_WHISK\_SUP, y = MAX, yend = MAX, group = categ[length(categ)]), color = "black", size = box.line.size, alpha = box.alpha, lineend = "round"))** *#*

**coord.names <- c(coord.names, "sup.whisker.edge")**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_segment(data = stat, mapping = ggplot2::aes(x = X\_WHISK\_INF, xend = X\_WHISK\_SUP, y = MIN, yend = MIN, group = categ[length(categ)]), color = "black", size = box.line.size, alpha = box.alpha, lineend = "round"))** *#*

**coord.names <- c(coord.names, "inf.whisker.edge")**

**}**

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

*# assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_point(data = stat, mapping = ggplot2::aes\_string(x = "X", y = "MEAN", group = categ[length(categ)]), shape = 23, stroke = box.line.size \* 2, fill = stat$COLOR, size = box.mean.size, color = "black", alpha = box.alpha)) # group used in aesthetic to do not have it in the legend*

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

**data = tempo.diamon.mean,**

**mapping = ggplot2::aes(x = X, y = Y, group = GROUP),**

**fill = tempo.diamon.mean[, "COLOR"],**

**color = hsv(0, 0, 0, alpha = box.alpha),** *# outline of the polygon in black but with alpha*

**size = box.line.size \* 2,**

**alpha = box.alpha**

**))**

**coord.names <- c(coord.names, "mean")**

**}**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_segment(data = stat, mapping = ggplot2::aes(x = if(box.notch == FALSE){X\_BOX\_INF}else{X\_NOTCH\_INF}, xend = if(box.notch == FALSE){X\_BOX\_SUP}else{X\_NOTCH\_SUP}, y = MEDIAN, yend = MEDIAN, group = categ[length(categ)]), color = "black", size = box.line.size \* 2, alpha = box.alpha))** *#*

**coord.names <- c(coord.names, "median")**

**}**

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

*# dot display*

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

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

**if(is.null(dot.categ)){**

**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,**

**shape = 19,**

**color = dot.coord.rd3$dot.color,**

**alpha = dot.alpha**

**))** *# 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)]),**

**shape = 21,**

**stroke = dot.border.size,**

**color = if(is.null(dot.border.color)){dot.coord.rd3$dot.color}else{rep(dot.border.color, nrow(dot.coord.rd3))},**

**size = dot.size,**

**fill = dot.coord.rd3$dot.color,**

**alpha = dot.alpha**

**))** *# 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.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", alpha = dot.categ),**

**size = dot.size,**

**shape = 19,**

**color = dot.coord.rd3$dot.color**

**))** *# 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", alpha = dot.categ),**

**size = dot.size,**

**shape = 21,**

**stroke = dot.border.size,**

**color = if(is.null(dot.border.color)){dot.coord.rd3$dot.color}else{rep(dot.border.color, nrow(dot.coord.rd3))},**

**fill = dot.coord.rd3$dot.color**

**))** *# 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*

**}**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_discrete\_manual(aesthetics = "alpha", name = dot.categ.legend.name, values = rep(dot.alpha, length(dot.categ.class.order)), guide = ggplot2::guide\_legend(override.aes = list(fill = dot.color, color = if(is.null(dot.border.color)){dot.color}else{dot.border.color}, stroke = dot.border.size, alpha = dot.alpha))))** *# 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*

**}**

**coord.names <- c(coord.names, "dots")**

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

*# here plot using group -> no scale*

**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", group = "group"),** *# not dot.categ here because the classes of dot.categ create new separations*

**position = ggplot2::position\_dodge(width = box.width),**

**binaxis = "y",**

**stackdir = "center",**

**alpha = dot.alpha,**

**fill = dot.coord$dot.color,**

**stroke = dot.border.size,**

**color = if(is.null(dot.border.color)){dot.coord$dot.color}else{rep(dot.border.color, nrow(dot.coord))},**

**show.legend = FALSE,** *# BEWARE: do not use show.legend = TRUE because it uses the arguments outside aes() as aesthetics (here color and fill). Thus I must find a way using ggplot2::scale\_discrete\_manual()*

**binwidth = (y.lim[2] - y.lim[1]) / dot.tidy.bin.nb**

**))** *# geom\_dotplot ggplot2 v3.3.0: I had to remove rev() in fill and color # very weird behavior of geom\_dotplot ggplot2 v3.2.1, (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.)*

**coord.names <- c(coord.names, "dots")**

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

**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", alpha = dot.categ),** *# not dot.categ here because the classes of dot.categ create new separations*

**position = ggplot2::position\_dodge(width = box.width),**

**binaxis = "y",**

**stackdir = "center",**

**fill = NA,**

**stroke = NA,**

**color = NA,**

*# BEWARE: do not use show.legend = TRUE because it uses the arguments outside aes() as aesthetics (here color and fill). Thus I must find a way using ggplot2::scale\_discrete\_manual()*

**binwidth = (y.lim[2] - y.lim[1]) / dot.tidy.bin.nb**

**))** *# geom\_dotplot ggplot2 v3.3.0: I had to remove rev() in fill and color # very weird behavior of geom\_dotplot ggplot2 v3.2.1, (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 = dot.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*

**coord.names <- c(coord.names, "bad\_remove")**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_discrete\_manual(aesthetics = "alpha", name = dot.categ.legend.name, values = rep(dot.alpha, length(dot.categ.class.order)), labels = dot.categ.class.order, guide = ggplot2::guide\_legend(title = if(ini.dot.categ == categ[length(categ)]){dot.categ}else{ini.dot.categ}, override.aes = list(fill = levels(dot.coord$dot.color), color = if(is.null(dot.border.color)){levels(dot.coord$dot.color)}else{dot.border.color}, stroke = dot.border.size, alpha = dot.alpha))))** *# 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*

**}**

*# 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))) > if(is.null(dot.categ)){1}else{2}){** *# if(is.null(dot.categ)){1}else{2} because 1 dotplot if dot.categ is NULL and 2 dotplots is not, with the second being a blank dotplot with wrong coordinates. Thus take the first in that situation*

**tempo.cat <- paste0("\n\n================\n\nINTERNAL CODE ERROR IN ", function.name, "\nMORE THAN ", if(is.null(dot.categ)){1}else{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))[1]]]** *# the second being a blank dotplot with wrong coordinates. Thus take the first whatever situation*

**}**

**tempo.box.coord <- merge(box.coord, unique(dot.coord[, c("PANEL", "group", categ)]), by = c("PANEL", "group"), sort = FALSE)** *# do not add dot.categ and tidy\_group\_coord here because the coordinates are for stats. Add the categ in box.coord. BEWARE: by = c("PANEL", "group") without fill column because PANEL & group columns are enough as only one value of x column per group number in box.coord. Thus, no need to consider fill column*

*# below inactivated because not true when dealing with dot.categ different from categ*

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

*tempo.cat <- paste0("\n\n================\n\nINTERNAL CODE ERROR IN ", function.name, "\nTHE 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", "PANEL", "group", "x", categ)], by = c("PANEL", "group"), sort = FALSE)** *# send the coord of the boxs into the coord data.frame of the dots (in the column x.y).BEWARE: by = c("PANEL", "group") without fill column because PANEL & group columns are enough as only one value of x column per group number in tempo.box.coord. Thus, no need to consider fill colum # DANGER: from here the fill.y and x.y (from tempo.box.coord) are not good in dot.coord.tidy2. It is ok because Group1 Group2 from tempo.box.coord are ok with the group column from dot.coord.tidy1. This is due to the fact that dot.coord.tidy resulting from geom\_dotplot does not make the same groups as the other functions*

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

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

**stop(tempo.cat)**

**}**

*# From here, check for dot.coord.tidy3 which wil be important for stat over the plot. BEWARE: dot.categ has nothing to do here for stat coordinates. Thus, not in tempo.data1*

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

**tempo.data1 <- unique(data.frame(data1[categ[1]], group = as.integer(data1[, categ[1]])))** *# categ[1] is factor*

**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(**

**formatC(as.integer(data1[, categ[2]]), width = nchar(max(as.integer(data1[, categ[2]]), na.rm = TRUE)), flag = "0"),** *# convert factor into numeric with leading zero for proper ranking*

**".",**

**formatC(as.integer(data1[, categ[1]]), width = nchar(max(as.integer(data1[, categ[1]]), na.rm = TRUE)), flag = "0")***# convert factor into numeric with leading zero for proper ranking*

**)))** *# merge the 2 formatC() to create a new factor. The convertion to integer should recreate the correct group number*

**)**

**)** *# 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\nINTERNAL CODE ERROR IN ", function.name, "\nCODE INCONSISTENCY 7\n\n============\n\n")**

**stop(tempo.cat)**

**}**

**dot.coord.tidy3 <- merge(dot.coord.tidy2, tempo.data1, by = intersect("group", "group"), sort = FALSE)** *# send the factors of data1 into coord. BEWARE: I have tested intersect("group", "group") instead of by = "group". May be come back to by = "group" in case of error. But I did this because of an error in dot.coord.rd3 above*

**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\nINTERNAL CODE ERROR IN ", function.name, "\nTHE 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 = if(length(categ.color) == 1){rep(categ.color, length(unique(data1[, categ[length(categ)]])))}else{categ.color})) #, guide = ggplot2::guide\_legend(override.aes = list(fill = levels(tempo.polygon$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(hsv(0, 0, 0, alpha = box.alpha), length(unique(data1[, categ[length(categ)]]))))) # , guide = ggplot2::guide\_legend(override.aes = list(color = "black", alpha = box.alpha))))** *# 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 # outline of the polygon in black but with alpha*

**}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, size = box.line.size, notch = box.notch, alpha = box.alpha, coef = if(box.whisker.kind == "no"){0}else if(box.whisker.kind == "std"){1.5}else if(box.whisker.kind == "max"){Inf}, outlier.shape = if( ! is.null(dot.color)){NA}else{21}, outlier.color = if( ! is.null(dot.color)){NA}else{if(dot.border.size == 0){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*

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

**data = tempo.polygon,**

**mapping = ggplot2::aes\_string(x = "X", y = "Y", group = "BOX", color = categ[length(categ)]),**

**size = box.line.size,**

**alpha = box.alpha,**

**lineend = "round",**

**linejoin = "round"**

**))**

**coord.names <- c(coord.names, "main.box")**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_segment(data = stat, mapping = ggplot2::aes(x = if(box.notch == FALSE){X\_BOX\_INF}else{X\_NOTCH\_INF}, xend = if(box.notch == FALSE){X\_BOX\_SUP}else{X\_NOTCH\_SUP}, y = MEDIAN, yend = MEDIAN, group = categ[length(categ)]), color = stat$COLOR, size = box.line.size \* 2, alpha = box.alpha))** *#*

**coord.names <- c(coord.names, "median")**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_segment(data = stat, mapping = ggplot2::aes(x = X, xend = X, y = QUART3, yend = MAX, group = categ[length(categ)]), color = stat$COLOR, size = box.line.size, alpha = box.alpha))** *#*

**coord.names <- c(coord.names, "sup.whisker")**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_segment(data = stat, mapping = ggplot2::aes(x = X, xend = X, y = QUART1, yend = MIN, group = categ[length(categ)]), color = stat$COLOR, size = box.line.size, alpha = box.alpha))** *#*

**coord.names <- c(coord.names, "inf.whisker")**

**if(box.whisker.width > 0){**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_segment(data = stat, mapping = ggplot2::aes(x = X\_WHISK\_INF, xend = X\_WHISK\_SUP, y = MAX, yend = MAX, group = categ[length(categ)]), color = stat$COLOR, size = box.line.size, alpha = box.alpha, lineend = "round"))** *#*

**coord.names <- c(coord.names, "sup.whisker.edge")**

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_segment(data = stat, mapping = ggplot2::aes(x = X\_WHISK\_INF, xend = X\_WHISK\_SUP, y = MIN, yend = MIN, group = categ[length(categ)]), color = stat$COLOR, size = box.line.size, alpha = box.alpha, lineend = "round"))** *#*

**coord.names <- c(coord.names, "inf.whisker.edge")**

**}**

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

*# assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::geom\_point(data = stat, mapping = ggplot2::aes\_string(x = "X", y = "MEAN", group = categ[length(categ)]), shape = 23, stroke = box.line.size \* 2, color = stat$COLOR, size = box.mean.size, fill = NA, alpha = box.alpha)) # 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*

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

**data = tempo.diamon.mean,**

**mapping = ggplot2::aes(x = X, y = Y, group = GROUP),**

**color = tempo.diamon.mean[, "COLOR"],**

**size = box.line.size \* 2,**

**alpha = box.alpha,**

**lineend = "round",**

**linejoin = "round"**

**))**

**coord.names <- c(coord.names, "mean")**

**}**

**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(unique(data1[, categ[length(categ)]])))))** *#, 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 = if(length(categ.color) == 1){rep(categ.color, length(unique(data1[, categ[length(categ)]])))}else{categ.color}, guide = ggplot2::guide\_legend(override.aes = list(alpha = if(plot == TRUE & ((length(dev.list()) > 0 & names(dev.cur()) == "windows") | (length(dev.list()) == 0 & Sys.info()["sysname"] == "Windows"))){1}else{box.alpha}))))** *# , 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*

**if(plot == TRUE & ((length(dev.list()) > 0 & names(dev.cur()) == "windows") | (length(dev.list()) == 0 & Sys.info()["sysname"] == "Windows"))){** *# if any Graph device already open and this device is "windows", or if no Graph device opened yet and we are on windows system -> prevention of alpha legend bug on windows using value 1*

*# to avoid a bug on windows: if alpha argument is different from 1 for lines (transparency), then lines are not correctly displayed in the legend when using the R GUI (bug https://github.com/tidyverse/ggplot2/issues/2452). No bug when using a pdf*

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

**tempo.warn <- paste0("(", warn.count,") FROM FUNCTION ", function.name, ": GRAPHIC DEVICE USED ON A WINDOWS SYSTEM ->\nTRANSPARENCY OF THE LINES IS INACTIVATED IN THE LEGEND TO PREVENT A WINDOWS DEPENDENT BUG (SEE** *https://github.com/tidyverse/ggplot2/issues/2452***)\nTO OVERCOME THIS ON WINDOWS, USE ANOTHER DEVICE (pdf() FOR INSTANCE)")**

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

**}**

**}**

*# 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 = stat$X, y = y.lim[2], label = if(stat.disp.mean == FALSE){fun\_round(stat$MEDIAN, 2)}else{fun\_round(stat$MEAN, 2)}, size = stat.size, color = "black", hjust = ifelse(vertical == TRUE, 0.5, 1.1), vjust = ifelse(vertical == TRUE, 1.1, 0.5)))** *# beware: no need of order() for labels because box.coord$x set the order. For justification, see https://stackoverflow.com/questions/7263849/what-do-hjust-and-vjust-do-when-making-a-plot-using-ggplot*

**coord.names <- c(coord.names, "stat.display")**

**}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**

**tempo.stat.ini$x.y <- tempo.stat.ini$x.x** *# this is just to be able to use tempo.stat.ini$x.y for untidy or tidy dots (remember that dot.coord.tidy3$x.y is not good, see above)*

**}**

**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\nINTERNAL CODE ERROR IN ", function.name, "\nFUSION 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\nINTERNAL CODE ERROR IN ", function.name, "\ntext.coord AND box.coord DO NOT HAVE THE SAME x COLUMN CONTENT\n\n============\n\n")**

**stop(tempo.cat)**

**}**

**}**

*# end stat coordinates*

*# stat display*

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

**if(stat.disp.mean == FALSE){**

**tempo.log.high <- if(diff(y.lim) > 0){box.coord$middle >= 0}else{box.coord$middle < 0}**

**tempo.log.low <- if(diff(y.lim) > 0){box.coord$middle < 0}else{box.coord$middle >= 0}**

**}else{**

**tempo.log.high <- if(diff(y.lim) > 0){box.coord$MEAN >= 0}else{box.coord$MEAN < 0}**

**tempo.log.low <- if(diff(y.lim) > 0){box.coord$MEAN < 0}else{box.coord$MEAN >= 0}**

**}**

**if(any(tempo.log.high) == TRUE){**

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

**geom = "text",**

**x = get(if(is.null(dot.color)){"box.coord"}else{"text.coord"})$x[tempo.log.high],** *# get(if(is.null(dot.color)){"box.coord"}else{"text.coord"}) for text just above error boxs or dots*

**y = get(if(is.null(dot.color)){"box.coord"}else{"text.coord"})[tempo.log.high, if(is.null(dot.color)){"middle"}else{"text.max.pos"}],**

**label = if(stat.disp.mean == FALSE){fun\_round(box.coord$middle, 2)[tempo.log.high]}else{fun\_round(box.coord$MEAN, 2)[tempo.log.high]},**

**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*

**}**

**if(any(tempo.log.low) == TRUE){**

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

**geom = "text",**

**x = get(if(is.null(dot.color)){"box.coord"}else{"text.coord"})$x[tempo.log.low],** *# get(if(is.null(dot.color)){"box.coord"}else{"text.coord"}) for text just above error boxs or dots*

**y = get(if(is.null(dot.color)){"box.coord"}else{"text.coord"})[tempo.log.low, if(is.null(dot.color)){"middle"}else{"text.min.pos"}],**

**label = if(stat.disp.mean == FALSE){fun\_round(box.coord$middle, 2)[tempo.log.low]}else{fun\_round(box.coord$MEAN, 2)[tempo.log.low]},**

**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*

**coord.names <- c(coord.names, "stat.display.positive", "stat.display.negative")**

**}else{**

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

**stop(tempo.cat)**

**}**

**}**

*# end stat display*

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

*# the rescaling aspect is complicated and not intuitive. See:*

*# explaination: https://github.com/tidyverse/ggplot2/issues/3948*

*# the oob argument of scale\_y\_continuous() https://ggplot2.tidyverse.org/reference/scale\_continuous.html*

*# see also https://github.com/rstudio/cheatsheets/blob/master/data-visualization-2.1.pdf*

*# secondary ticks*

**final.plot <- ggplot2::ggplot\_build(eval(parse(text = paste(paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + "), ' + if(vertical == TRUE){ggplot2::coord\_cartesian(ylim = if(diff(y.lim) < 0){rev(y.lim)}else{y.lim})}else{ggplot2::coord\_flip(ylim = y.lim)}'))))**

**tempo.coord <- final.plot$layout$panel\_params[[1]]**

*# y.second.tick.positions: coordinates of secondary ticks (only if y.second.tick.nb argument is non NULL or if y.log argument is different from "no")*

**if(y.log != "no"){** *# integer main ticks for log2 and log10*

**tempo.scale <- (as.integer(min(y.lim, na.rm = TRUE)) - 1):(as.integer(max(y.lim, na.rm = TRUE)) + 1)**

**}else{**

**tempo <- if(is.null(attributes(tempo.coord$y$breaks))){tempo.coord$y$breaks}else{unlist(attributes(tempo.coord$y$breaks))}**

**if(all(is.na(tempo))){**

**tempo.cat <- paste0("\n\n============\n\nINTERNAL CODE ERROR IN ", function.name, "\nONLY NA IN tempo.coord$y$breaks\n\n============\n\n")**

**stop(tempo.cat)**

**}**

**tempo.scale <- fun\_scale(lim = y.lim, n = ifelse(is.null(y.tick.nb), length(tempo[ ! is.na(tempo)]), y.tick.nb))** *# in ggplot 3.3.0, tempo.coord$y.major\_source replaced by tempo.coord$y$breaks. If fact: n = ifelse(is.null(y.tick.nb), length(tempo[ ! is.na(tempo)]), y.tick.nb)) replaced by n = ifelse(is.null(y.tick.nb), 4, y.tick.nb))*

**}**

**y.second.tick.values <- NULL**

**y.second.tick.pos <- NULL**

**if(y.log != "no"){**

**tempo <- fun\_inter\_ticks(lim = y.lim, log = y.log)**

**y.second.tick.values <- tempo$values**

**y.second.tick.pos <- tempo$coordinates**

*# 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 = y.second.tick.pos, yend = y.second.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 = y.second.tick.pos, xend = y.second.tick.pos, y = tempo.coord$y.range[1], yend = tempo.coord$y.range[1] + diff(tempo.coord$y.range) / 80))*

*# }*

**coord.names <- c(coord.names, "y.second.tick.positions")**

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

*# if(y.second.tick.nb > 0){ #inactivated because already checked before*

**tempo <- fun\_inter\_ticks(lim = y.lim, log = y.log, breaks = tempo.scale, n = y.second.tick.nb)**

**y.second.tick.values <- tempo$values**

**y.second.tick.pos <- tempo$coordinates**

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

**geom = "segment",**

**y = y.second.tick.pos,**

**yend = y.second.tick.pos,**

**x = if(vertical == TRUE){tempo.coord$x.range[1]}else{tempo.coord$y.range[1]},**

**xend = if(vertical == TRUE){tempo.coord$x.range[1] + diff(tempo.coord$x.range) / 80}else{tempo.coord$y.range[1] + diff(tempo.coord$y.range) / 80}**

**))**

**coord.names <- c(coord.names, "y.second.tick.positions")**

**}**

*# end secondary ticks*

*# for the ggplot2 bug with y.log, this does not work: eval(parse(text = ifelse(vertical == FALSE & y.log == "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,**

**minor\_breaks = y.second.tick.pos,**

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

**expand = c(0, 0),** *# remove space after after axis limits*

**limits = sort(y.lim),** *# NA indicate that limits must correspond to data limits but ylim() already used*

**oob = scales::rescale\_none,**

**trans = ifelse(diff(y.lim) < 0, "reverse", "identity")** *# equivalent to ggplot2::scale\_y\_reverse() but create the problem of y-axis label disappearance with y.lim decreasing. Thus, do not use. Use ylim() below and after this*

**))**

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

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::coord\_cartesian(ylim = y.lim))** *#problem of ggplot2::ylim() is taht it redraw new breaks # coord\_cartesian(ylim = y.lim)) not used because bug -> y-axis label disappearance with y.lim decreasing # 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 = y.lim))** *# clip = "off" to have secondary ticks outside plot region does not work # create the problem of y-axis label disappearance with y.lim decreasing*

**}**

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

*# legend management*

**if( ! is.null(legend.width)){**

**legend.final <- fun\_gg\_get\_legend(ggplot\_built = final.plot, fun.name = function.name, lib.path = lib.path) # get legend**

**}**

*# end legend management*

*# drawing*

**fin.plot <- suppressMessages(suppressWarnings(eval(parse(text = paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + ")))))**

**if(plot == TRUE){**

*# following lines inactivated because of problem in warn.recov and message.recov*

*# assign("env\_fun\_get\_message", new.env())*

*# assign("tempo.gg.name", tempo.gg.name, envir = env\_fun\_get\_message)*

*# assign("tempo.gg.count", tempo.gg.count, envir = env\_fun\_get\_message)*

*# assign("add", add, envir = env\_fun\_get\_message)*

*# two next line: for the moment, I cannot prevent the warning printing*

*# warn.recov <- fun\_get\_message(paste(paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + "), if(is.null(add)){NULL}else{add}), kind = "warning", header = FALSE, print.no = FALSE, env = env\_fun\_get\_message) # for recovering warnings printed by ggplot() functions*

*# message.recov <- fun\_get\_message('print(eval(parse(text = paste(paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + "), if(is.null(add)){NULL}else{add}))))', kind = "message", header = FALSE, print.no = FALSE, env = env\_fun\_get\_message) # for recovering messages printed by ggplot() functions*

**# reactivate the 4 next lines to incorporate legend.width**

**# if(is.null(legend.width)){**

**suppressMessages(suppressWarnings(print(fin.plot)))**

**# }else{**

**# suppressMessages(suppressWarnings(gridExtra::grid.arrange(fin.plot, legend.final, ncol=2, widths=c(1, legend.width))))**

**# }**

*# suppressMessages(suppressWarnings(print(eval(parse(text = paste(paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + "), if(is.null(add)){NULL}else{add}))))))*

**}else{**

*# following lines inactivated because of problem in warn.recov and message.recov*

*# message.recov <- NULL*

*# warn.recov <- NULL*

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

**tempo.warn <- paste0("(", warn.count,") PLOT NOT SHOWN AS REQUESTED")**

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

**}**

*# end drawing*

*# outputs*

*# following lines inactivated because of problem in warn.recov and message.recov*

*# if( ! (is.null(warn) & is.null(warn.recov) & is.null(message.recov))){*

*# warn <- paste0(warn, "\n\n", if(length(warn.recov) > 0 | length(message.recov) > 0){paste0(paste0("MESSAGES FROM ggplot2 FUNCTIONS: ", ifelse( ! is.null(warn.recov), unique(message.recov), ""), ifelse( ! is.null(message.recov), unique(message.recov), ""), collapse = "\n\n"), "\n\n")})*

*# }else if( ! (is.null(warn) & is.null(warn.recov)) & is.null(message.recov)){*

*# warn <- paste0(warn, "\n\n", if(length(warn.recov) > 0){paste0(paste0("MESSAGES FROM ggplot2 FUNCTIONS: ", unique(warn.recov), collapse = "\n\n"), "\n\n")})*

*# }else if( ! (is.null(warn) & is.null(message.recov)) & is.null(warn.recov)){*

*# warn <- paste0(warn, "\n\n", if(length(message.recov) > 0){paste0(paste0("MESSAGES FROM ggplot2 FUNCTIONS: ", unique(message.recov), collapse = "\n\n"), "\n\n")})*

*# }*

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

**warning(paste0("FROM ", function.name, " FUNCTION:\n\n", warn), call. = FALSE)** *# to recover the warning messages, use return = TRUE*

**}**

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

**tempo.output <- ggplot2::ggplot\_build(fin.plot)**

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

**if(length(tempo.output$data) != length(coord.names)){**

**tempo.cat <- paste0("\n\n================\n\nINTERNAL CODE ERROR IN ", function.name, "\nlength(tempo.output$data) AND length(coord.names) MUST BE IDENTICAL. CODE HAS TO BE MODIFIED\n\n================\n\n")**

**stop(tempo.cat)**

**}else{**

**names(tempo.output$data) <- coord.names**

**tempo.output$data <- tempo.output$data[coord.names != "bad\_remove"]**

**}**

**tempo <- tempo.output$layout$panel\_params[[1]]**

**output <- list(**

**data = data1,**

**stat = stat,**

**removed.row.nb = removed.row.nb,**

**removed.rows = removed.rows,**

**plot = c(tempo.output$data, y.second.tick.values = list(y.second.tick.values)),**

**panel = facet.categ,**

**axes = list(**

**x.range = tempo$x.range,**

**x.labels = if(is.null(attributes(tempo$x$breaks))){tempo$x$breaks}else{tempo$x$scale$get\_labels()},** *# is.null(attributes(tempo$x$breaks)) test if it is number (TRUE) or character (FALSE)*

**x.positions = if(is.null(attributes(tempo$x$breaks))){tempo$x$breaks}else{unlist(attributes(tempo$x$breaks))},**

**y.range = tempo$y.range,**

**y.labels = if(is.null(attributes(tempo$y$breaks))){tempo$y$breaks}else{tempo$y$scale$get\_labels()},**

**y.positions = if(is.null(attributes(tempo$y$breaks))){tempo$y$breaks}else{unlist(attributes(tempo$y$breaks))}**

**),**

**warn = paste0("\n", warn, "\n\n"),**

**ggplot = if(return.ggplot == TRUE){fin.plot}else{NULL}** *# fin.plot plots the graph if return == TRUE*

**)**

**return(tempo <- output)**

**}**

*# end outputs*

*# end main code*

**}**