**# https://ggplot2-book.org/scales.html**

**#remain do do:**

**# add names to the output. NULL vector, and name added for echa plotted feature**

**# add legend if dots have specific colors**

**# add message recov**

**# 3) quanti variable for categ**

**# to solve:**

**# fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group2"), categ.color = c("green", "red"), categ.class.order = list(c("A", "B")), return = TRUE, box.fill = TRUE)**

**# fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group2"), categ.color = c("green", "red"), categ.class.order = list(c("A", "B")), return = TRUE, box.fill = FALSE)**

**# fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = c("Group2"), dot.color=c("green", "blue"), dot.categ = "Group3", dot.border.color = "black", dot.alpha = 1, dot.border.size = 2, dot.categ.class.order = c("J", "I"))**

**# fun\_gg\_boxplot(data1 = obs1, y = "Time", categ = "Group1", categ.color= c("blue"), categ.class.order=list(c("H", "G")), box.width=0.5, box.line.size=1, box.notch=TRUE, box.alpha = 1, box.fill = FALSE, box.whisker.kind = "max", box.whisker.width = 0, dot.color=1:2, dot.categ = "Group2")**

**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 = 0.5,**

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

**dot.categ = NULL,**

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

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

**dot.tidy = FALSE,**

**dot.tidy.bin.nb = 30,**

**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.inter.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,**

**classic = TRUE,**

**grid = FALSE,**

**return = 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)) ; data1 = obs1 ; y = "Time" ; categ = "Group1" ; categ.class.order = list(c("G", "H")) ; 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 = NULL ; dot.categ = NULL ; 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.inter.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 ; text.angle = 0 ; classic = FALSE ; grid = FALSE ; return = TRUE ; 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)) ; 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 = NULL ; dot.categ = NULL ; dot.categ.class.order = NULL ; dot.categ.legend.name = NULL ; dot.tidy = FALSE ; 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.inter.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 ; text.angle = 0 ; classic = FALSE ; grid = FALSE ; return = 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)) ; 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 = 1:2 ; dot.categ =* **"Group2"** *; dot.categ.class.order = NULL ; dot.categ.legend.name = NULL ; dot.tidy = FALSE ; 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.inter.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 ; text.angle = 0 ; classic = FALSE ; grid = FALSE ; return = 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*

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

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

**stop(tempo.cat)**

**}**

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

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

**stop(tempo.cat)**

**}**

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

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

**stop(tempo.cat)**

**}**

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

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

**stop(tempo.cat)**

**}**

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

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

**stop(tempo.cat)**

**}**

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

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

**stop(tempo.cat)**

**}**

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

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

**stop(tempo.cat)**

**}**

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

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

**stop(tempo.cat)**

**}**

*# end required function checking*

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

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

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

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

**warning <- NULL**

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

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

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

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

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

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

**cat(tempo.cat)**

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

**}**

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

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

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

**cat(tempo.cat)**

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

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

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

**}**

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

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

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

**cat(tempo.cat)**

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

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

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

**cat(tempo.cat)**

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

**}**

*# reserved word checking*

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

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

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

**cat(tempo.cat)**

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

**}**

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

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

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

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

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

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

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

**}**

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

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

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

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

**}**

**}**

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

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

**}**

*# end reserved word checking*

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

**cat(tempo.cat)**

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

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

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

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

**}**

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

**}**

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

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

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

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

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

**cat(tempo.cat)**

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

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

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

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

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

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

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

**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[i1]])), 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("\n\n================\n\nERROR IN ", function.name, ": COMPARTMENT ", i3, " OF categ.class.order ARGUMENT CANNOT HAVE DUPLICATED CLASSES: ", paste(categ.class.order[[i3]], collapse = " "), "\n\n================\n\n")**

**cat(tempo.cat)**

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

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

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": COMPARTMENT ", i3, " OF categ.class.order ARGUMENT MUST BE CLASSES OF ELEMENT ", i3, " OF categ 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\n\n================\n\n")**

**cat(tempo.cat)**

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

**}else{**

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

**}**

**}**

**}**

**}else{**

**tempo.warning <- paste0("FROM ", function.name, " FUNCTION: THE categ.class.order SETTING IS NULL. ALPHABETICAL ORDER WILL BE APPLIED FOR ", paste(categ, collapse = " "))**

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

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

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

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

**}**

**}**

**# categ.class.order not NULL anymore**

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

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

**}else{**

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

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

**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**

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

*# check the nature of color*

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

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

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

*# integer colors into gg\_palette*

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

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

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

**cat(tempo.cat)**

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

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

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

**}**

*# end integer colors into gg\_palette*

**}**

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

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

**cat(tempo.cat)**

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

**}**

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

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

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

**}**

*# end check the nature of color*

*# check the length of color*

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

**if(length(categ.color) == length(levels(data1[, categ[i0]]))){** *# here length(categ.color) is equal to the different number of categ*

*# data1[, categ[i0]] <- factor(data1[, categ[i0]]) # not required because sure that is is a factor*

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

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

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

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

**}else if(length(categ.color) == length(data1[, categ[i0]])){***# here length(categ.color) is equal to nrow(data1) -> Modif to have length(categ.color) equal to the different number of categ (length(categ.color) == length(levels(data1[, categ[i0]])))*

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

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

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

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

**cat(tempo.cat)**

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

**}else{**

*# data1[, categ[i0]] <- factor(data1[, categ[i0]]) # not required because sure that is is a factor*

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

**tempo.warning <- paste0("FROM ", function.name, " FUNCTION: categ.color ARGUMENT HAS THE LENGTH OF data1 ROW NUMBER\nCOLORS HAVE BEEN RESPECTIVELY ASSOCIATED TO EACH CLASS OF categ ", categ[i0], " AS:\n", paste(levels(factor(data1[, categ[i0]])), collapse = " "), "\n", paste(categ.color, collapse = " "))**

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

**}**

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

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

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

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

**}else{**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": categ.color ARGUMENT MUST BE (1) LENGTH 1, OR (2) THE LENGTH OF data1 NROWS, OR (3) THE LENGTH OF THE CLASSES IN THE categ ", categ[i0], " COLUMN. HERE IT IS COLOR LENGTH ", length(categ.color), " VERSUS CATEG LENGTH ", length(data1[, categ[i0]]), " AND CATEG CLASS LENGTH ", length(unique(data1[, categ[i0]])), "\nPRESENCE OF NA COULD BE THE PROBLEM\n\n================\n\n")**

**cat(tempo.cat)**

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

**}**

**}else{**

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

*# data1[, categ[i0]] <- factor(data1[, categ[i0]]) # not required because sure that is is a factor*

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

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

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

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

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

**}**

**# 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**

**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)){**

*# optional legend of dot colors*

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

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

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

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

**cat(tempo.cat)**

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

**}else if(tempo$problem == FALSE & all(dot.categ %in% categ)){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": dot.categ ARGUMENT CANNOT BE A COLUMN NAME OF data1 ALREADY SPECIFIED IN THE categ ARGUMENT:\n", dot.categ, "\n\n================\n\n")**

**cat(tempo.cat)**

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

**}**

**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("\n\n================\n\nERROR IN ", function.name, ": dot.categ COLUMN MUST BE A FACTOR OR CHARACTER VECTOR\n\n================\n\n")** *#*

**cat(tempo.cat)**

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

**}**

**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)){**

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

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

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

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

**cat(tempo.cat)**

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

**}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("\n\n================\n\nERROR IN ", function.name, ": dot.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 (", dot.categ, ") OF data1\n\n================\n\n")**

**cat(tempo.cat)**

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

**}else{**

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

**}**

**}**

**}else{**

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

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

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

**}**

**# dot.categ.class.order not NULL anymore**

**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)**

**}else{**

**dot.categ.legend.name <- dot.categ** *#*

**tempo.warning <- paste0("FROM ", function.name, " FUNCTION: THE dot.categ.legend.name SETTING IS NULL. VALUES OF dot.categ WILL BE USED: ", paste(dot.categ, collapse = " "))**

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

**}**

**# dot.categ.legend.name not NULL anymore**

**}**

*# end optional legend of dot colors*

*# check the nature of color*

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

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

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

*# integer colors into gg\_palette*

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

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

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

**cat(tempo.cat)**

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

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

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

**}**

*# end integer colors into gg\_palette*

**}**

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

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

**tempo.warning <- paste0("FROM ", function.name, " FUNCTION: dot.color ARGUMENT HAS BEEN SET TO \"SAME\"\nTHUS, DOT COLORS HAVE BEEN RESPECTIVELY ASSOCIATED TO EACH CLASS OF categ ", categ[i0], " AS:\n", paste(levels(factor(data1[, categ[i0]])), collapse = " "), "\n", paste(dot.color, collapse = " "))**

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

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

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

**cat(tempo.cat)**

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

**}**

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

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

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

**}**

*# end check the nature of color*

*# check the length of color*

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

*# optional legend of dot colors*

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

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

**cat(tempo.cat)**

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

**}**

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

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

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

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

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

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

**cat(tempo.cat)**

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

**}else{**

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

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

**}**

**# 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{**

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

**if(length(dot.color) == length(levels(data1[, categ[i0]]))){** *# here length(dot.color) is equal to the different number of categ*

*# data1[, categ[i0]] <- factor(data1[, categ[i0]]) # not required because sure that is is a factor*

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

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

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

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

**}else if(length(dot.color) == length(data1[, categ[i0]])){***# here length(dot.color) is equal to nrow(data1) -> Modif to have length(dot.color) equal to the different number of categ (length(dot.color) == length(levels(data1[, categ[i0]])))*

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

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

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

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

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

**}else{**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": dot.color ARGUMENT MUST BE (1) LENGTH 1, OR (2) THE LENGTH OF data1 NROWS, OR (3) THE LENGTH OF THE CLASSES IN THE categ ", categ[i0], " COLUMN. HERE IT IS COLOR LENGTH ", length(dot.color), " VERSUS CATEG LENGTH ", length(data1[, categ[i0]]), " AND CATEG CLASS LENGTH ", length(unique(data1[, categ[i0]])), "\nPRESENCE OF NA COULD BE THE PROBLEM\n\n================\n\n")**

**cat(tempo.cat)**

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

**}**

*# 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))){**

**tempo.warning <- paste0("FROM ", function.name, " FUNCTION: 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)")**

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

**}**

**# 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**

**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)**

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

**tempo.warning <- paste0("FROM ", function.name, " FUNCTION: 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")**

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

**}**

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

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

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ": 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.border.size = 0 (NO BORDER FOR POTENTIAL OUTLIER DOTS)\n-> THESE SETTINGS ARE NOT ALLOWED BECAUSE THE POTENTIAL OUTLIER DOTS WILL NOT BE VISIBLE\n\n================\n\n")**

**cat(tempo.cat)**

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

**}**

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

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

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

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

*# integer colors into gg\_palette*

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

**cat(tempo.cat)**

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

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

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

**cat(tempo.cat)**

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

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

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

**}**

*# end integer colors into gg\_palette*

**}**

**if( ! is.null(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("\n\n================\n\nERROR IN ", function.name, ": y.lim ARGUMENT CANNOT CONTAIN -Inf OR Inf VALUES\n\n================\n\n")**

**cat(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(tempo$problem == FALSE & y.log != "no"){**

**tempo.warning <- paste0("FROM ", function.name, " FUNCTION: 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")**

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

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

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

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

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

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

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

**cat(tempo.cat)**

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

**}**

**}**

**}**

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

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

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

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

**cat(tempo.cat)**

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

**}**

**}**

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

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

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

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

**cat(tempo.cat)**

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

**}**

**}**

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

*# inactivated because xlim and y.lim already log transformed*

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

*# tempo.warning <- paste0("FROM ", function.name, " FUNCTION: 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")*

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

*# }*

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

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

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

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

**}**

**tempo <- fun\_check(data = stat.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)**

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

**vertical <- TRUE**

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

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

**}**

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

**tempo <- fun\_check(data = 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)**

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

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

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

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

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

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

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

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

**cat(tempo.cat)**

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

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

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

**cat(tempo.cat)**

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

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

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

**cat(tempo.cat)**

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

**}**

**}**

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

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

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

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

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

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

**}**

**}**

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

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

**}**

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

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

*# package checking*

**fun\_pack(req.package = c("ggplot2"), 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"}, dot.categ)**

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

**tempo.warning <- paste0("FROM ", function.name, " FUNCTION: NA DETECTED IN COLUMNS ", paste(column.check, collapse = " "), " OF data1 AND CORRESPONDING ROWS REMOVED (SEE $removed.row.nb AND $removed.rows)")**

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

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

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

**tempo.warning <- paste0("FROM ", function.name, " FUNCTION: COLUMN ", column.check[i2], " OF data1 CONTAINS NA")**

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

**}**

**}**

**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){**

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

**if(any( ! unique(removed.rows[, column.check[i1]]) %in% unique(data1[, column.check[i3]]))){**

**tempo.warning <- paste0("FROM ", function.name, " FUNCTION: IN COLUMN ", column.check[i3], " OF data1, THE FOLLOWING CLASSES HAVE BEEN LOST DUE TO NA REMOVAL IN data1:\n", paste(unique(removed.rows[, column.check[i3]])[ ! unique(removed.rows[, column.check[i3]]) %in% unique(data1[, column.check[i3]])], collapse = " "))**

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

**}**

**}**

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

**}**

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

**if(any( ! levels(data1[, column.check[i2]]) %in% unique(data1[, column.check[i2]]))){**

**tempo.warning <- paste0("FROM ", function.name, " FUNCTION: IN COLUMN ", column.check[i2], " OF data1, , THE FOLLOWING LEVELS ARE NOT REPRESENTED IN THE COLUMN:\n", paste(levels(data1[, column.check[i2]])[ ! levels(data1[, column.check[i2]]) %in% unique(data1[, column.check[i2]])], collapse = " "))**

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

**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**

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

**}else 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**

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

**}else{**

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

**}**

**}**

**}**

**}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), ggplot2::ggplot())**

**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(categ.color))}else{as.character(dot.color)})) # rep(NA, length(categ.color)) used because dot.color is NULL**

**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 = as.character(categ.color)))**

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

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

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

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

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

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

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

**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"]},**

**tempo.categ1 = data1[order(data1$categ.check, data1[, y]), categ[1]]**

**)** *# y.check to be sure that the order is the same between the y of data1 and the y of dot.coord*

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

**}else{**

**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"]},**

**tempo.categ1 = data1[order(data1$categ.check, data1[, y]), categ[1]],**

**tempo.categ3 = data1[order(data1$categ.check, data1[, y]), dot.categ]**

**)** *# y.check to be sure that the order is the same between the y of data1 and the y of dot.coord*

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

**names(dot.coord)[names(dot.coord) == "tempo.categ3"] <- dot.categ**

**}**

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

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

**stop(tempo.cat)**

**}else{**

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

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

**stop(tempo.cat)**

**}else{**

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

**}**

**}**

**# }**

*# end per box dots coordinates recovery*

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

*# 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), ggplot2::ggplot())**

**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(categ.color))}else{as.character(dot.color)})) # rep(NA, length(categ.color)) used because dot.color is NULL**

**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 = as.character(categ.color)))**

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

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

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

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

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

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

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

**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"]},**

**tempo.categ1 = data1[order(data1$categ.check, data1[, y]), categ[1]],**

**tempo.categ2 = data1[order(data1$categ.check, data1[, y]), categ[2]]**

**)** *# y.check to be sure that the order is the same between the y of data1 and the y of dot.coord*

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

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

**}else{**

**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"]},**

**tempo.categ1 = data1[order(data1$categ.check, data1[, y]), categ[1]],**

**tempo.categ2 = data1[order(data1$categ.check, data1[, y]), categ[2]],**

**tempo.categ3 = data1[order(data1$categ.check, data1[, y]), dot.categ]**

**)** *# y.check to be sure that the order is the same between the y of data1 and the y of dot.coord*

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

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

**names(dot.coord)[names(dot.coord) == "tempo.categ3"] <- dot.categ**

**}**

**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, ": (dot.coord$y AND dot.coord$y.check) AS WELL AS (dot.coord$group AND dot.coord$categ.check) MUST BE IDENTICAL. CODE HAS TO BE MODIFIED\n\n================\n\n")**

**stop(tempo.cat)**

**}else{**

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

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

**stop(tempo.cat)**

**}else{**

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

**}**

**}**

**# }**

**}else{**

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

**stop(tempo.cat)**

**}**

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

**tempo.cat <- paste0("\n\n================\n\nINTERNAL CODE ERROR IN ", function.name, ": (ini.box.coord$group AND tempo.mean$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))){**

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

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

**}**

**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, ": CODE 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), ggplot2::ggplot())**

**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){**

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

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

**add.check <- FALSE**

**}**

**}**

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**))**

**}else{**

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

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

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

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

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

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

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

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

**))**

**}**

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

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

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

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

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

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

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

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

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

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

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

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

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

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

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

**))**

**}**

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

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

*# end constant part*

*# 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 = as.character(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, ": tempo.mean$BOX AND box.coord$group DO NOT HAVE THE SAME VALUE ORDER\n\n============\n\n"))**

**stop(tempo.cat)**

**}else{**

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

**tempo.warning <- paste0("FROM ", function.name, " FUNCTION: MEAN VALUES INSTEAD OF MEDIAN VALUES DISPLAYED")**

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

**}**

**}**

*# 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, ": (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"],**

**stringsAsFactors = TRUE**

**)**

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

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

**tempo.warning <- paste0("FROM ", function.name, " FUNCTION: SOME NOTCHES ARE BEYOND BOX HINGES. TRY ARGUMENT box.notch = FALSE")**

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

**}**

**}**

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

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

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

**stop(tempo.cat)**

**}**

**set.seed(1)**

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

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

**set.seed(NULL)**

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

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

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

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

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

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

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

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

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

**}else{**

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

**stop(tempo.cat)**

**}**

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

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

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

**stop(tempo.cat)**

**}**

*# end random dots*

**}**

*# tidy dots*

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

*# end tidy dots*

**}**

*# end 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(i2 in 1:length(categ)){**

**if(i2 == 1){**

**tempo.polygon <- data.frame(GROUPX = c(t(stat[, c(categ[i2], categ[i2], categ[i2], categ[i2], categ[i2])])), stringsAsFactors = TRUE)**

**}else{**

**tempo.polygon <- cbind(tempo.polygon, c(t(stat[, c(categ[i2], categ[i2], categ[i2], categ[i2], categ[i2])])), 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)**

**}else{**

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

**if(i2 == 1){**

**tempo.polygon <- data.frame(GROUPX = c(t(stat[, c(categ[i2], categ[i2], categ[i2], categ[i2], categ[i2], categ[i2], categ[i2], categ[i2], categ[i2], categ[i2], categ[i2])])), stringsAsFactors = TRUE)**

**}else{**

**tempo.polygon <- cbind(tempo.polygon, c(t(stat[, c(categ[i2], categ[i2], categ[i2], categ[i2], categ[i2], categ[i2], categ[i2], categ[i2], categ[i2], categ[i2], categ[i2])])), 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)**

**}**

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

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

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

**tempo.polygon[, categ[i2]] <- factor(tempo.polygon[, categ[i2]], levels = categ.class.order[[i2]])**

**}**

**}**

**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)**

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

**}**

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

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

**data = dot.coord,**

**mapping = ggplot2::aes\_string(x = categ[1], y = "y", linetype = categ[length(categ)]),**

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

**binaxis = "y",**

**stackdir = "center",**

**alpha = dot.alpha,**

**fill = dot.coord[rev(order(dot.coord[, categ[1]], decreasing = TRUE)), "dot.color"],**

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

**color = if(is.null(dot.border.color)){dot.coord[rev(order(dot.coord[, categ[1]], decreasing = TRUE)), "dot.color"]}else{rep(dot.border.color, nrow(dot.coord))},**

**show.legend = FALSE,**

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

**))** *# very weird behavior of geom\_dotplot, (1) because with aes group = (to avoid legend), the dot plotting is not good in term of coordinates, and (2) because data1 seems reorderer according to x = categ[1] before plotting. Thus, I have to use fill = dot.coord[rev(order(dot.coord[, categ[1]], decreasing = TRUE)), "dot.color"] to have the good corresponding colors # show.legend option do not remove the legend, only the aesthetic of the legend (dot, line, etc.)*

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::scale\_discrete\_manual(aesthetics = "linetype", name = categ.legend.name, values = rep(1, length(categ.color))))** *# values = rep("black", length(categ.color)) are the values of color (which is the border color of dots), and this modify the border color on the plot. BEWARE: values = categ.color takes the numbers to make the colors if categ.color is a factor. BEWARE: , guide = ggplot2::guide\_legend(override.aes = list(fill = levels(dot.color))) here*

*# coordinates of tidy dots*

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

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

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

**stop(tempo.cat)**

**}else{**

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

**}**

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

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

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

**stop(tempo.cat)**

**}**

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

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

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

**stop(tempo.cat)**

**}**

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

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

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

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

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

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

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

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

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

**}else{**

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

**stop(tempo.cat)**

**}**

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

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

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

**stop(tempo.cat)**

**}**

*# end coordinates of tidy dots*

**}**

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

**}**

*# end dot display*

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

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

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

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

**}**

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

*# stat display*

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

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

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

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::annotate(geom = "text", x = 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*

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

*# stat coordinates*

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

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

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

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

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

**}**

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

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

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

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

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

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

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

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

**stop(tempo.cat)**

**}**

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

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

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

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

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

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

**tempo.cat <- (paste0("\n\n============\n\nINTERNAL CODE ERROR IN ", function.name, ": text.coord AND box.coord DO NOT HAVE THE SAME x COLUMN CONTENT\n\n============\n\n"))**

**stop(tempo.cat)**

**}**

**}**

*# end stat coordinates*

*# stat display*

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

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

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

**geom = "text",**

**x = box.coord$x[box.coord$middle >= 0],**

**y = box.coord$middle[box.coord$middle >= 0],**

**label = if(stat.disp.mean == FALSE){fun\_round(box.coord$middle, 2)[box.coord$middle >= 0]}else{fun\_round(box.coord$MEAN, 2)[box.coord$MEAN >= 0]},**

**size = stat.size,**

**color = "black",**

**hjust = ifelse(vertical == TRUE, 0.5, 0.5 - stat.dist),**

**vjust = ifelse(vertical == TRUE, 0.5 - stat.dist, 0.5)**

**))** *# beware: no need of order() for labels because box.coord$x set the order*

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

**geom = "text",**

**x = box.coord$x[box.coord$middle < 0],**

**y = box.coord$middle[box.coord$middle < 0],**

**label = if(stat.disp.mean == FALSE){fun\_round(box.coord$middle, 2)[box.coord$middle < 0]}else{fun\_round(box.coord$MEAN, 2)[box.coord$MEAN < 0]},**

**size = stat.size,**

**color = "black",**

**hjust = ifelse(vertical == TRUE, 0.5, 0.5 + stat.dist),**

**vjust = ifelse(vertical == TRUE, 0.5 + stat.dist, 0.5)**

**))** *# beware: no need of order() for labels because box.coord$x set the order*

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

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

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

**geom = "text",**

**x = text.coord$x[box.coord$middle >= 0],**

**y = text.coord$text.max.pos[box.coord$middle >= 0],**

**label = if(stat.disp.mean == FALSE){fun\_round(box.coord$middle, 2)[box.coord$middle >= 0]}else{fun\_round(box.coord$MEAN, 2)[box.coord$MEAN >= 0]},**

**size = stat.size,**

**color = "black",**

**hjust = ifelse(vertical == TRUE, 0.5, 0.5 - stat.dist),**

**vjust = ifelse(vertical == TRUE, 0.5 - stat.dist, 0.5)**

**))** *# beware: no need of order() for labels because box.coord$x set the order*

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

**geom = "text",**

**x = text.coord$x[box.coord$middle < 0],**

**y = text.coord$text.min.pos[box.coord$middle < 0],**

**label = if(stat.disp.mean == FALSE){fun\_round(box.coord$middle, 2)[box.coord$middle < 0]}else{fun\_round(box.coord$MEAN, 2)[box.coord$MEAN < 0]},**

**size = stat.size,**

**color = "black",**

**hjust = ifelse(vertical == TRUE, 0.5, 0.5 + stat.dist),**

**vjust = ifelse(vertical == TRUE, 0.5 + stat.dist, 0.5)**

**))** *# beware: no need of order() for labels because box.coord$x set the order*

**}**

*# end stat display*

**}else{**

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

**stop(tempo.cat)**

**}**

**}**

*# end stat display*

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

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

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

*# 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,**

**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, ": CODE INCONSISTENCY 10\n\n============\n\n")) ; stop(tempo.cat)},**

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

**limits = NA,**

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

**))**

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

**assign(paste0(tempo.gg.name, tempo.gg.count <- tempo.gg.count + 1), ggplot2::coord\_cartesian(ylim = y.lim))** *# 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*

**}**

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

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

*# no secondary ticks for log2. Play with y.lim*

**if(y.log == "log10"){**

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

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

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

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

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

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

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

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

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

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

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

**stop(tempo.cat)**

**}**

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

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

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

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

*# }*

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

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

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

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

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

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

**stop(tempo.cat)**

**}**

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

**minor.tick.dist <- tick.dist / (y.inter.tick.nb + 1)**

**minor.tick.pos <- seq(ticks.pos[1] - tick.dist, ticks.pos[length(ticks.pos)] + tick.dist, by = minor.tick.dist)**

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

**}else{**

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

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

**tempo.cat <- (paste0("\n\n============\n\nINTERNAL CODE ERROR IN ", function.name, ": CODE INCONSISTENCY 13\n\n============\n\n"))**

**stop(tempo.cat)**

**}**

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

**minor.tick.dist <- tick.dist / (y.inter.tick.nb + 1)**

**minor.tick.pos <- seq(ticks.pos[1] - tick.dist, ticks.pos[length(ticks.pos)] + tick.dist, by = minor.tick.dist)**

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

**}**

**}**

**}**

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

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

*# drawing*

**if(plot == TRUE){**

**message.recov <- utils::capture.output(suppressWarnings(print(eval(parse(text = paste(paste(paste0(tempo.gg.name, 1:tempo.gg.count), collapse = " + "), if(is.null(add)){NULL}else{add}))))), type = "message")** *# for recovering messages printed by ggplot() functions*

**}else{**

**message.recov <- NULL**

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

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

**}**

*# end drawing*

*# outputs*

**if( ! (is.null(warning) & is.null(message.recov))){**

**warning <- paste0("Warning message:\n\n", warning, "\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(warning)){**

**tempo <- base::warning(warning)** *# to recover the warning messages, use return = TRUE*

**}**

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

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

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

**if(length(output$data) != length(coord.names)){**

**tempo.cat <- paste0("\n\n================\n\nINTERNAL CODE ERROR IN ", function.name, ": length(output$data) AND length(coord.names) MUST BE IDENTICAL. CODE HAS TO BE MODIFIED\n\n================\n\n")**

**stop(tempo.cat)**

**}else{**

**names(output$data) <- coord.names**

**}**

**output <- list(data = data1, stat = stat, removed.row.nb = removed.row.nb, removed.rows = removed.rows, plot = output$data, axes = output$layout$panel\_params[[1]], warnings = paste0("\n", warning, "\n\n"))**

**return(output)**

**}**

*# end outputs*

*# end main code*

**}**