######## fun\_merge() #### merge the columns of two 2D objects, by common rows

**fun\_merge <- function(data1, data2, name1, name2, factor.as = "numeric", warn.print = FALSE){**

*# AIM*

*# merge the columns of 2 data frames or 2 matrices or 2 tables, by associating the rows according to 1 or several common colums that must be strictly similar between the 2 objects*

*# contrary to the classical merge() function of R, fun\_merge() orders the rows of the 2 objects according to the common columns, and merge only and only if the ordered common columns are strictly identical. Otherwise return an error*

*# keep row names of data1 in the merged object if they exist. Do not consider row names of data2*

*# keep the intial row order of data1 after merging*

*# BEWARE:*

*# see: https://stackoverflow.com/questions/1299871/how-to-join-merge-data-frames-inner-outer-left-right*

*# see: https://en.wikipedia.org/wiki/Join\_(SQL) and the french version*

*# REQUIRED PACKAGES*

*# none*

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

*# fun\_comp\_2d()*

*# fun\_check()*

*# ARGUMENTS*

*# data1: matrix or data frame or table*

*# data2: same class of object as data1 (data frame for data1 data frame, matrix for data1 matrix and table for data1 table) with same number of rows as in data1*

*# name1: either a vector of character strings or a vector of integer. If character strings, they must be the name of the columns in data1 that are common to the columns in data2. If integers, they must be the column numbers in data1 that are common to column numbers in data2. name1 can be strings and name2 (below) integers, and vice-versa. BEWARE: order of the elements in data1 are important as ordering is according to the first element, then the second, etc.*

*# name2: as in name1 but for data2. Order in name2 is not important as order in name1 is used for the ordering*

*# factor.as: either "numeric" (sort factors according to levels order, i.e., class number) or "character" (sort factors according to alphabetical order)*

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

*# RETURN*

*# a list containing:*

*# $data: the merged data frame or matrix or table*

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

*# EXAMPLES*

*# obs1 = matrix(1:10, ncol = 5, dimnames = list(letters[1:2], LETTERS[1:5])) ; obs2 = as.data.frame(matrix(1:10, ncol = 5, dimnames = list(letters[1:2], LETTERS[1:5]))) ; obs1 ; obs2 ; fun\_comp\_2d(obs1, obs2)*

*# DEBUGGING*

*# data1 = matrix(1.0001:21, ncol = 4) ; dimnames(data1) <- list(LETTERS[1:5], letters[1:4]); data2 = matrix(1.0001:31, ncol = 6) ; dimnames(data2) <- list(NULL, c("a", "aa", "c", "d", "aaa", "aaaa")) ; set.seed(1) ; data2[, "c"] <- sample(data2[, "c"]) ; data2[, "d"] <- sample(data2[, "d"]) ; set.seed(NULL) ; data1 ; data2 ; name1 = c("c", "d") ; name2 = c("d", "c") ; factor.as = "numeric" # for function debugging*

*# function name*

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

*# end function name*

*# required function checking*

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

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

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

**}**

*# end required function checking*

*# argument checking using fun\_check()*

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

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

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

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

**tempo1 <- fun\_check(data = data1, class = "matrix", print = FALSE)**

**tempo2 <- fun\_check(data = data1, class = "data.frame", print = FALSE)**

**tempo3 <- fun\_check(data = data1, class = "table", print = FALSE)**

**if(tempo1$problem == TRUE & tempo2$problem == TRUE & tempo3$problem == TRUE){**

**tempo.cat <- paste0("ERROR IN ", function.name, ":\ndata1 ARGUMENT MUST BE A 2D OBJECT (MATRIX, DATA FRAME OR TABLE)\nHERE IT IS: ", paste(class(data1), collapse = " "))** *#*

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

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

**}**

**tempo1 <- fun\_check(data = data2, class = "matrix", print = FALSE)**

**tempo2 <- fun\_check(data = data2, class = "data.frame", print = FALSE)**

**tempo3 <- fun\_check(data = data2, class = "table", print = FALSE)**

**if(tempo1$problem == TRUE & tempo2$problem == TRUE & tempo3$problem == TRUE){**

**tempo.cat <- paste0("ERROR IN ", function.name, ":\ndata2 ARGUMENT MUST BE A 2D OBJECT (MATRIX, DATA FRAME OR TABLE)\nHERE IT IS: ", paste(class(data2), collapse = " "))** *#*

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

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

**}**

**if( ! identical(class(data1), class(data2))){**

**tempo.cat <- paste0("ERROR IN ", function.name, ":\ndata1 and data2 ARGUMENTS MUST BE A 2D OBJECT (MATRIX, DATA FRAME OR TABLE) OF SAME CLASS\nHERE IT IS RESPECTIVELY: ", paste(class(data1), collapse = " "), " AND ", paste(class(data2), collapse = " "))** *#*

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

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

**}**

**tempo1 <- fun\_check(data = name1, class = "vector", typeof = "integer", double.as.integer.allowed = TRUE, print = FALSE)**

**tempo2 <- fun\_check(data = name1, class = "vector", typeof = "character", print = FALSE)**

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

**tempo.cat <- paste0("ERROR IN ", function.name, ":\nname1 ARGUMENT MUST BE A UNIQUE CHARACTER STRING OR INTEGER\nHERE IT IS: ", paste(name1, collapse = " "))** *#*

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

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

**}**

**tempo1 <- fun\_check(data = name2, class = "vector", typeof = "integer", double.as.integer.allowed = TRUE, print = FALSE)**

**tempo2 <- fun\_check(data = name2, class = "vector", typeof = "character", print = FALSE)**

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

**tempo.cat <- paste0("ERROR IN ", function.name, ":\nname2 ARGUMENT MUST BE A UNIQUE CHARACTER STRING OR INTEGER\nHERE IT IS: ", paste(name2, collapse = " "))** *#*

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

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

**}**

**tempo <- fun\_check(data = factor.as, options = c("numeric", "character"), length = 1, fun.name = function.name) ; eval(ee)**

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

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

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

**}**

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

*# end argument checking using fun\_check()*

*# other argument checking*

*# column existence*

**if(mode(name1) == "character"){**

**if( ! all(name1 %in% colnames(data1))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ":\nTHE CHARACTER STRINGS IN name1 ARGUMENT ARE NOT ALL COLUMN NAMES OF data1:\n", paste(name1, collapse = " "), "\n", colnames(data1), "\n\n================\n\n")** *#*

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

**}**

**}else if(mode(name1) == "numeric"){**

**if( ! all((name1 > ncol(data1) & name1 <= 0))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ":\nINTEGERS IN name1 ARGUMENT ARE NOT ALL COLUMN NUMBERS OF data1:\n", paste(name1, collapse = " "), "\n1:", ncol(data1), "\n\n================\n\n")** *#*

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

**}**

**}else{**

**tempo.cat <- paste0("\n\n============\n\nERROR IN ", function.name, ": CODE INCONSISTENCY 1\n\n============\n\n")**

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

**}**

**if(mode(name2) == "character"){**

**if( ! all(name2 %in% colnames(data2))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ":\nTHE CHARACTER STRINGS IN name2 ARGUMENT ARE NOT ALL COLUMN NAMES OF data2:\n", paste(name2, collapse = " "), "\n", colnames(data2), "\n\n================\n\n")** *#*

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

**}**

**}else if(mode(name2) == "numeric"){**

**if( ! all((name2 > ncol(data2) & name2 <= 0))){**

**tempo.cat <- paste0("\n\n================\n\nERROR IN ", function.name, ":\nINTEGERS IN name2 ARGUMENT ARE NOT ALL COLUMN NUMBERS OF data2:\n", paste(name2, collapse = " "), "\n1:", ncol(data2), "\n\n================\n\n")** *#*

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

**}**

**}else{**

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

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

**}**

**if(length(name1) != length(name2)){**

**tempo.cat <- paste0("\n\n============\n\nERROR IN ", function.name, ":\nLENGTH OF name1 ARGUMENT (", length(name1), ") IS NOT THE SAME AS LENGTH OF name2 ARGUMENT (", length(name2), "):\n", paste(name1, collapse = " "), "\n", paste(name2, collapse = " "), "\n\n============\n\n")**

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

**}**

*# end column existence*

*# end other argument checking*

*# main code*

*# definition of set1 and set2: common columns*

**set1 <- data1[, name1, drop = FALSE]** *# set1 will be the reference for merging, drop = FALSE to keep the 2D structure*

**if(any(apply(set1, 2, FUN = "%in%", "factor"))){**

**if(factor.as == "numeric"){**

**set1[, apply(set1, 2, FUN = "%in%", "factor")] <- as.numeric(set1[, apply(set1, 2, FUN = "%in%", "factor")])**

**}**

**}**

**set2 <- data2[, name2, drop = FALSE]** *# set2 will be the reference for merging, drop = FALSE to keep the 2D structure*

**if(any(apply(set2, 2, FUN = "%in%", "factor"))){**

**if(factor.as == "numeric"){**

**set2[, apply(set2, 2, FUN = "%in%", "factor")] <- as.numeric(set2[, apply(set2, 2, FUN = "%in%", "factor")])**

**}**

**}**

*# end definition of set1 and set2: common columns*

*# conversion as character to avoid floating point problems*

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

**options(digits = 22)**

**set1 <- as.matrix(set1)**

**set2 <- as.matrix(set2)**

**mode(set1) <- "character"**

**mode(set2) <- "character"**

**options(digits = options.ini)**

*# end conversion as character to avoid floating point problems*

*# recovering initial order of set1*

**ini.set1.order <- eval(parse(text = paste("order(", paste("set1[, ", 1:ncol(set1), "]", sep = "", collapse = ", "), ")")))**

**set1 <- set1[ini.set1.order, ]**

**ini.set2.order <- eval(parse(text = paste("order(", paste("set2[, ", 1:ncol(set2), "]", sep = "", collapse = ", "), ")")))**

**set2 <- set2[ini.set2.order, ]**

*# end recovering initial order of set1*

*# check non identical columns*

**if(length(name1) > 1){**

**for(i2 in 1:(length(name1) - 1)){**

**for(i3 in (i2 + 1):length(name1)){**

**if(identical(set1[, i2], set1[, i3])){**

**tempo.cat <- paste0("\n\n============\n\nERROR IN ", function.name, ":\nCOLUMN ", i2, " OF data1 CORRESPONDING TO ELEMENT ", name1[i2], " OF name1 ARGUMENT IS IDENTICAL TO COLUMN ", i3, " OF data1 CORRESPONDING TO ELEMENT ", name1[i3], " OF name1 ARGUMENT\n\n============\n\n")**

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

**}**

**}**

**}**

**}**

**if(length(name2) > 1){**

**for(i2 in 1:(length(name2) - 1)){**

**for(i3 in (i2 + 1):length(name2)){**

**if(identical(set2[, i2], set2[, i3])){**

**tempo.cat <- paste0("\n\n============\n\nERROR IN ", function.name, ":\nCOLUMN ", i2, " OF data2 CORRESPONDING TO ELEMENT ", name2[i2], " OF name2 ARGUMENT IS IDENTICAL TO COLUMN ", i3, " OF data2 CORRESPONDING TO ELEMENT ", name2[i3], " OF name2 ARGUMENT\n\n============\n\n")**

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

**}**

**}**

**}**

**}**

*# end check non identical columns*

*# warning duplicates*

**# repositioning of the column in set2 as in set1 by comparing the two sorted column**

**#deal with identical col names when merging -> .x for data1, .y for data2**

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

**warning(warn, call. = FALSE)**

**}**

**# output <- list()**

**return(output)**

**}**