Standard Functions for Basic Statistical Analysis

R code in standardfunctions V10.R

Bruno Fischer Colonimos

16 mars 2017

Abstract

This is the code of the standard functions v10. (make-results= modified)

Contents

1 Constants and default options settings

```
# sfinitdefaults sets some constants in the data structure and returns a
# function that gets and sets global options

sfinitdefaults <- function () {
        # structure definition, with some constants already set up
        defaultvalue <- list( #namesum
                namesumeng = c("n", "Mean", "St.dev",
                               "Min.", "1st Qu.","Median", "3rd Qu.",  "Max.",
                               " NA's"),
                namesumfrench  = c("n", "Moyenne", "Ecart-type",
                                   "Min.", "Q1","Mediane", "Q3",  "Max.",
                                   " NA's"),
                namesum ="",
                language = "",
                filldefault = "steelblue")
        # Info function
        info <- function() {names(defaultvalue)}
        # access function
        function(name = NULL, value = NULL, setnull = FALSE){
                if(is.null(name)) {
                        warning("sfdefaults : You did not supply a name. Is it normal?",
                                immediate. = TRUE, call. = TRUE)
                        message(paste0( "Syntax sfdefault:\n",
                                        "sfdefault() : displays syntax\n",
                                        "sfdefautlt('?') : vector of defined  names\n",
                                        "sfdefautlt('name') : value of 'name'\n",
                                        "sfdefautlt('name', non_null_value) : set value of 'name'"))
                } else if (name=="?") {
                        info()
                } else if (!is.null(value)) {
                        defaultvalue[[name]] <<- value
                        defaultvalue[[name]]
                } else if (setnull == TRUE) {
                        defaultvalue[[name]] <<- NULL
                } else {defaultvalue[[name]]}

        }
}
```

set up access function and some defaults

```
# begin
sfdefault <- sfinitdefaults()

# language
sfdefault("language","french")
if(sfdefault("language") == "french") {
        sfdefault("namesum", sfdefault("namesumfrench"))
        sfdefault("percentlabel", sfdefault("pourcent"))
} else {
        sfdefault("namesum", sfdefault("namesumeng"))
        sfdefault("percentlabel", sfdefault("percent"))
}
```

Programming default options

```
sfdefault("reportNA", FALSE) # report number of NA's in a variable?
sfdefault("orderfreq", TRUE) # Should we order the levels of a factor before graphing?
```

display options

```
sfdefault("digits", 2)
sfdefault("sumdigits" , 2)
sfdefault("filldefault", "steelblue")
sfdefault("colorannots1", "red")
# max number of plots
sfdefault("maxplots", 3)
# options discrete charts
sfdefault("cat1plots", c("pie", "bar")) # which plots to store for 1 cat var #c("bar", "pie")

# options bar chart
sfdefault("discretebarwidth", 0.5)

# piechart options
sfdefault("scaletitle" , "") # title for the legend of the pie
sfdefault("dolabel" , TRUE) # label the slices with %
sfdefault("minperc" , 5) # minimal % value for showing a lable
sfdefault("labpos" , 1.15) # label position relative to the center (0=center, 1 = plot radius)

# verification
# sfdefault("?")
```

2 Structure for each set of statistical results

```
# make a result list. unsupplied elements assigned default=NULL and not included in result list
make.result <- function(name = NULL,
                        funname = NULL, # name of the function which produced the result
                        varnames =NULL, # vector (or better = named vector) of variable names
                        numcases = NULL, # number of non-NA cases
                        summaries = NULL, # numerical summaries (quantitative variables)
                        levels = NULL,
                        levels2 = NULL,
                        breaks = NULL,
                        closed= NULL,
                        table = NULL, # default table
                        table1 = NULL,
                        table2 = NULL,
                        table3 = NULL,
                        ptable = NULL, # printable table
                        details = NULL, # additional info (mostly in table form)
                        chi2 = NULL,
                        anova = NULL,
                        test1 = NULL,
                        test2 = NULL,
                        test3 = NULL,
                        plot = NULL, # default plot
                        plot1 = NULL,
                        plot2 = NULL,
```

```
                        plot3 = NULL
                        ) {
        # get arg - values list
        lenv <- as.list(environment())
        # remove NULL values from list
        if (length(which(sapply(lenv,is.null))) == 0) {
                lenv
        } else {
                lenv[-(which(sapply(lenv,is.null), arr.ind=FALSE))]
        }
}

# result retrieval:
# use result$name or result[[name]]
```

3 Recording a collection of sets of statistical results

```
# initialization : create the recording structure and the accessor/modifier
# function
initresult <- function() {
        allresults <- list(.whatsit = "resultlist") # encapsulated list of results
        function(rname, rval) {
                if (missing(rval)) {
                        allresults[[rname]]
                } else {
                        allresults[[rname]] <<- rval
                        rname
                }
        }
}


# assigning the modifier and accessor function to 'result
result <- initresult
```

3.0.1 usage:

for each statistical function that returns a set of results, call result(, funname (arguments...)) to **store** the results set under the name . result() **retrieves and returns** the results set stored under the name

4 Helper functions

4.1 General utility functions

assoc.op : associative operator function: apply a binary operator or function to a list of (many) arguments

opname the operator/function name (string)
listargs the list of arguments to apply the operator to

returns: a single result (of any type)

```
assoc.op <- function(opname, listargs) {
        xfun <- function(listargs, res) {
                if (length(listargs) == 0) {
                        res
                } else{
                        xfun(listargs[-1],
                             do.call(opname,
                                     list(res,  listargs[[1]])))
                }
        }

        if (length(listargs) > 0) {
                xfun(listargs[-1], res = listargs[[1]])
        } else {warning("argument listargs has lenth 0")
                logical(0)
        }
}
```

REM: This is likely not necessary. Check 'reduce'

```
# identify a  warning
is.warning <- function(x) {"warning" %in% class(x)}
```

vlookup (as in Excel). each column may be described by its rank or its name

```
vlookup <- function(value, searchtable, searchcol = 1, returncol= 2){
        searchtable[match(value, searchtable[[searchcol]]), returncol]
}
```

4.2 Functions for filtering out NA's

- from a dataframe/tbl

... is a succession of variable names which we want to filter out the NAs from (ex: nonadf(dataframe, "age", "revenue"))

```
nonadf <- function(dataf, ..., useNA = "no") {
        lvar = list(...)
        lseq = seq_along(lvar)
        if (useNA == "no") {
                # make list of logical vectors
                llogicals <- lapply(
                        X = lvar,
                        FUN = function(nomvar) {
                                !is.na(dataf[[nomvar]])
                        }
                )
                # combine all with 'and' operator (&)
                andlogicals <- assoc.op("&", llogicals)
                dataf[which(andlogicals),]
        } else {
                dataf
        }
}
```

- from a vector

```
nonavect <- function(vect) {vect[which(!is.na(vect))]}
```

4.3 Simple and multiple summary tables

```
# (nb of cases, mean, stdev, five-number-summary, optionally nb of NA's)
```

vector of summaries for 1 quant variable

```
sumvector <- function (var, dnames = sfdefault("namesum"),
                       reportNA = sfdefault("reportNA")) {
        if (length(var) == 0) {
                sapply(numeric(length = 9), function(x) NA)
        }else {# construct a more complete summary vector
                s <- summary(var)
                if (length(s) < 7) {s <- c(s, rep(0, times=7-length(s)))}
                ret <- numeric(3)
                ret[1] <- sum(!is.na(var))
                ret[2] <-  s["Mean"]
                ret[3] <- sd(var, na.rm = TRUE)
                s <- c(ret, s[-4])
                names(s) <- dnames
                if (reportNA) {s} else {s[1:(length(s) - 1)] }
        }
}

#
# sumvector <- function (var, dnames = sfdefault("namesum"),
#                        reportNA = sfdefault("reportNA")) {
#         if (length(var) == 0) {
#                 sapply(numeric(length = 9), function(x) NA)
#         }else {# construct a more complete summary vector
#                 nonavar <- nonavect(var)
#                 numcases <-  length(nonavar)
#                 numna <- length(var) - numcases
#                 meanval <- mean(nonavar)
#                 sdval <- sd(nonavar)
#
#                 res <- c(numcases, meanval, sdval, quantile(nonavar))
#
#                 if (reportNA) {
#                         names(res) <- dnames[1:8]
#                 } else {
#                         res <- c(res,numna)
#                         names(res) <- dnames
#                 }
#                 res
#         }
# }
```

Combined summaries for different variables in a dataframe, for all individuals

```
cbsummaries <- function (dataf, vnames) {
        # vnames = a vector of variable names (each a numeric variable of dataf)
        lsum = lapply(vnames, function(nam) sumvector(dataf[[nam]]))
        df <- do.call(what = data.frame, args = lsum)
        colnames(df) <- vnames
        # rownames(df) <- namesum
        df
}
```

Combined summaries for one variable, *conditional* to the values of a factor

```
condsummaries <- function(dataf, vname, fname) {
        # vname = the variable name
        # fname = the factor name
        # levels: if not factor, make it a factor and take the levels
        if (is.factor(dataf[[fname]])) {
                lv <- levels(dataf[[fname]])
        } else {
                lv <- levels(factor(dataf[[fname]]))
        }
        lsum = lapply(lv ,
                      FUN=function(lev) {
                              dt <- dataf[dataf[[fname]]==lev , ]
                              sumvector(dt[[vname]])
                      } )
        df <- do.call(what = data.frame, args = lsum)
        colnames(df) <- lv
        # rownames(df) <- namesum # rownames are preserved
        df
}
```

4.4 Frequency tables

```
# joint frequency table
jointfreqtable <- function(dataf, nomfact1, nomfact2, useNA = "no") {
        if (useNA == "no") {
                dataf <- dataf[!is.na(dataf[[nomfact1]]) & !is.na(dataf[[nomfact2]]) , ]
        }
        table(dataf[[nomfact1]], dataf[[nomfact2]] , useNA = useNA)
}


# new . fonctionne avec des tbl_df aussi
condfreqtable <- function(dataf, nomfact1, nomfact2, useNA = "no") {
        if (useNA == "no") {
                dataf <- dataf[!is.na(dataf[[nomfact1]]) & !is.na(dataf[[nomfact2]]) , ]
        }
        dt <-prop.table(table(dataf[[nomfact1]], dataf[[nomfact2]] , useNA = useNA),
                        margin = 1)
        dt2 <- as.data.frame(dt)
        names(dt2) <- c(nomfact1, nomfact2, "perc") # compatibilite avec la def ancienne
        dt2
}
```

4.5 reordering factors

```
# new definition seems ok for both data.frame and tbl_df
orderfact <- function(dataf, nomfact, orderfreq = TRUE, orderdesc = TRUE,
                      ordervar = "c..nt", orderval = NA, orderfun = sum,
                      nlevels = NULL) {
        if (is.null(nlevels)) {
                direction <- ifelse(orderdesc,-1, 1)

                if (orderfreq & ordervar == "c..nt") {
                        dataf$c..nt <- c(1)
                }
                if (is.na(orderval) & ordervar == "c..nt") {
                        dataf$c..nt <- c(1)
                } else if (is.na(orderval) & ordervar != "c..nt") {
                        # dataf$c..nt <- ifelse(is.na(dataf[, ordervar]), 0, 1)
                        #
                        # ordervar <- "c..nt"
                        # ne rien faire ??
                } else {
                        dataf$c..nt <-
                                ifelse(is.na(dataf[[ordervar]]),
                                       0 ,
                                       ifelse(dataf[[ordervar]] == orderval,
                                              1, 0))
                        ordervar <- "c..nt"
                }
                # reordonner le facteur
                if (orderfreq) {
                        xx <- dataf[[nomfact]]
                        xxx <- direction * dataf[[ordervar]]
                        resfact <- reorder(xx, xxx, orderfun, na.rm = TRUE)
                } else {
                        resfact <- dataf[[nomfact]]
                }
        } else {
                resfact <- factor(dataf[[nomfact]], levels = nlevels) ### modfié ????????????
        }
        # retour
        resfact
}
```

4.6 Statistical Testing functions

```
# try.chisq.test ==> essaye un test du chi2, et si il genere un warning
# (conditions approximation du chi2 non satisfaites), alors, calculer la
# p-valeur par simulation
# si keep-all, retourne les 2 tests (chi2 et
# simulation, une valeur logique indiquant le warning, et le warning lui-m?me).
# Le test prefere est alors list? comme test1

try.chisq.test <- function(..., keep.all = TRUE) {
```

```
        ww <- tryCatch(chisq.test(...),
                       error = function(e) {e},
                       warning = function(w) w )

        if (is.warning(ww)) {
                if (keep.all) {
                        list(test1 = chisq.test(..., simulate.p.value = TRUE),
                             test2 = chisq.test(...),
                             warning = TRUE,
                             warningmsg = ww )
                } else {
                        list(test1 = chisq.test(..., simulate.p.value = TRUE))
                }
        } else {
                if (keep.all) {
                        list(test1 = chisq.test(...),
                             test2 = chisq.test(..., simulate.p.value = TRUE),
                             warning = FALSE,
                             warningmsg = "" )
                } else {
                        list(test1 = chisq.test(...))
                }
        }
}
```

---

5 Graphing functions

5.1 Simple bar chart

```
#
# sfdefault("percentlabel")

barchart <- function(dataf,
                     nomvar,
                     useNA = "no",
                     #digits = sfdefault("digits"),
                     rfreq = TRUE,
                     barwidth = sfdefault("discretebarwidth"),
                     cfill = sfdefault("filldefault"),
                     percentlabel = sfdefault("percentlabel") ) {
        # data+aes
        if (useNA == "no") {
                dataf <- dataf[which(!is.na(dataf[[nomvar]])),]
        }
        if (rfreq) {
                pt <- ggplot(dataf,
                             aes_(as.name(nomvar),
                                  quote(
                                          100 * ..count.. / sum(..count..)
```

```
                                    )))
        } else {
                pt <- ggplot(dataf,
                             aes_(as.name(nomvar)))
        }
        # geom
        pt <- pt + geom_bar(width = barwidth, fill = cfill)
        # ylabel
        if (rfreq) {
                pt <- pt + ylab(percentlabel)
        }
        pt
}
```

5.2 Pie chart

```
# sfdefault("scaletitle" , "") # title for the legend of the pie
# sfdefault("dolabel" , TRUE) # label the slices with %
# sfdefault("minperc" , 8) # minimal % value for showing a lable
# sfdefault("labpos" , 1.1)

piechart <- function(data, var,
                     scaletitle = sfdefault("scaletitle"),
                     dolabel = sfdefault("dolabel"),
                     minperc = sfdefault("minperc"),
                     labpos = sfdefault("labpos") ) {
        # define local function
        piechart1 <- function(data, mapping, scaletitle, dolabel, minperc, labpos) {
                pie <- ggplot(data) +
                        geom_bar(mapping, width = 1, color="black") +
                        scale_x_continuous(labels=NULL, breaks=NULL) +
                        scale_y_continuous(labels=NULL, breaks=NULL) +
                        scale_fill_discrete(guide = guide_legend(title = scaletitle)) +
                        coord_polar(theta = "y") +
                        xlab(NULL) +
                        ylab(NULL)
                st <- ggplot_build(pie)
                lbldf <- with(st$data[[1]], {
                        xl <- xmin+ labpos * (xmax-xmin)
                        yl <- (ymax + ymin)/2
                        perc <- 100 * round(count/sum(count) , 2)
                        perclabs <- ifelse( perc > minperc, paste0(as.character(perc),"%"), "")
                        data.frame(xl, yl,perc, perclabs)
                } )
                if (dolabel) {pie <- pie + geom_text(data = lbldf, aes(x=xl, y = yl, label = perclabs))
                pie

        }
        # use local function (passing mapping)
        piechart1(data, aes_(1, fill = as.name(var)), scaletitle, dolabel, minperc, labpos)
}
```

```
# Try
# piechart(mpg, "manufacturer", minperc = 1)
# piechart(mpg, "model", minperc = 3)
# piechart(mpg, "drv", minperc = 3)
```

5.3 Simple Histogram + optional density

```
chistodens <- function(dataf, nomvar,
                       usedensity = FALSE, usendensity =FALSE, plot_density = FALSE,
                       fillhist = sfdefault("filldefault"), color_density = "red", digits = 2, # ? modi
                       bins = NULL, closed = NULL, ...) {  # ... = addtl arguments for geom_hist
        if (plot_density) {
                usedensity <- TRUE
        } # plot_density overrides usedensity, density overrides ndensity
        if (usedensity){
                usendensity <- FALSE
        }
        # bins = Null, integer, or a function name : "nclass.Sturges", "nclass.FD" , "nclass.scott"
        # get or compute bins (as integer)
        if (!is.null(bins)) {
                if ("character" %in% class(bins) ) {
                        bins <-  do.call(bins, list(nonavect(dataf[[nomvar]])))
                } else {bins <- NULL
                warning("bins is not a function", call. = TRUE)}
        }
        # make histogram
        p <- ggplot(dataf, aes_(as.name(nomvar))) +
                if (usedensity) {geom_histogram(aes(y=..density..),
                                                bins = bins, fill = fillhist,...)
                } else if (usendensity) {geom_histogram(aes(y=..ndensity..),
                                                        bins = bins, fill = fillhist,...)
                } else {geom_histogram(bins = bins, fill = fillhist, ...)}

        if (plot_density) {p <- p + geom_density(color=color_density) }
        p
}
```

5.4 continuous x factor boxplot & jitter plot

```
cbyfboxjit <- function(dataf, varf, varc, useNA = "no",
                       labellayer = "", labelall = "All values", labelgroups = "by goup") {

        if (useNA == "no") {
                dataf <- dataf[!is.na(dataf[[varf]]) & !is.na(dataf[[varc]]), ]
        }
        ggplot(dataf,aes_(as.name(varf) , as.name(varc))) +
                geom_boxplot(aes(group = 1, fill =  labelall),
                             outlier.colour = "gray", outlier.size = 0) +
                geom_boxplot(aes(fill = labelgroups), varwidth = TRUE,
                             outlier.colour = "gray", outlier.size = 0) +
```

```
                geom_jitter( aes_(color=as.name(varf)), width =.5, alpha=.5) +
                labs(fill = labellayer)
}
```

5.5 continuous x discrete boxplot & jitter plot

```
#
cbydboxjit <- function(dataf, vard, varc, useNA = "no",
                       labellayer = "", labelall = "All values", labelgroups = "by goup") {
        # dataf <- as.data.frame(dataf) # same problem with tbl_df
        if (useNA == "no") {
                dataf <- dataf[!is.na(dataf[[vard]]) & !is.na(dataf[[varc]]), ]
        }
        dataf$fact_vard. <- factor(dataf[[vard]])
        ggplot(dataf,aes_(as.name(vard) , as.name(varc), color=quote(fact_vard.))) +
                geom_boxplot(aes(group = 1, fill =  labelall), outlier.colour = "gray") +
                geom_boxplot(aes(fill = labelgroups), varwidth = TRUE, outlier.colour = "gray") +
                geom_jitter( width =.5, alpha=.5) +
                labs(fill = labellayer)
}
```

5.6 continuous by factor density plot

```
cbyfdensity <- function(dataf, varf, varc, useNA = "no") {
        if (useNA == "no") {
                dataf <- dataf[!is.na(dataf[[varf]]) & !is.na(dataf[[varc]]), ]
        }
        if (!is.factor(dataf[[varf]])) {dataf[[varf]] <- factor(dataf[[varf]])}
        ggplot(dataf,aes_(as.name(varc), y=quote(..density..), fill=as.name(varf))) +
                geom_density(alpha = 0.3)
}
```

5.7 continuous by factor freqpoly

```
cbyffreqpoly <- function(dataf, varf, varc, useNA = "no", size = 1) {
        if (useNA == "no") {
                dataf <- dataf[!is.na(dataf[[varf]]) & !is.na(dataf[[varc]]), ]
        }
        if (!is.factor(dataf[[varf]])) {dataf[[varf]] <- factor(dataf[[varf]])}
        ggplot(dataf,aes_(as.name(varc), y=quote(..ndensity..), color=as.name(varf))) +
                geom_freqpoly(size = size)
}
```

5.8 continuous by factor dodged histogram

```
cbyfhistogram <- function(dataf, varf, varc, useNA = "no",
                          usedensity = FALSE, usendensity = FALSE, ...) {
```

```
        if (useNA == "no") {
                dataf <- dataf[!is.na(dataf[[varf]]) & !is.na(dataf[[varc]]), ]
        }
        if (!is.factor(dataf[[varf]])) {dataf[[varf]] <- factor(dataf[[varf]])}

        # s <- condsummaries(dataf,vname = varc, fname = varf )

        p <- if (usedensity) {ggplot(dataf,aes_(as.name(varc), y=quote(..density..), fill=as.name(varf))
        } else if (usendensity) {ggplot(dataf,aes_(as.name(varc), y=quote(..ndensity..), fill=as.name(va
        } else {ggplot(dataf,aes_(as.name(varc), fill=as.name(varf)))}
        p <- p+ geom_histogram(..., position = "dodge")
        p
}
```

5.9 continuous by factor faceted histogram

```
cbyffachistogram <- function(dataf, varf, varc, useNA = "no",
                             usedensity = FALSE, usendensity = FALSE, ...) {
        if (useNA == "no") {
                dataf <- dataf[!is.na(dataf[[varf]]) & !is.na(dataf[[varc]]), ]
        }
        if (!is.factor(dataf[[varf]])) {dataf[[varf]] <- factor(dataf[[varf]])}

        p <- if (usedensity) {
                ggplot(dataf,aes_(as.name(varc), y=quote(..density..), fill=as.name(varf)))
        } else if (usendensity){
                ggplot(dataf,aes_(as.name(varc), y=quote(..ndensity..), fill=as.name(varf)))
        } else {
                ggplot(dataf,aes_(as.name(varc), fill=as.name(varf)))
        }
        p <- p+ geom_histogram(...)

        form <- as.formula(paste0(varf,"~."))
        p+ facet_grid(form)
}
```

5.10 Multiple plot function

```
#
# ggplot objects can be passed in ..., or to plotlist (as a list of ggplot objects)
# - cols:   Number of columns in layout
# - layout: A matrix specifying the layout. If present, 'cols' is ignored.
#
# If the layout is something like matrix(c(1,2,3,3), nrow=2, byrow=TRUE),
# then plot 1 will go in the upper left, 2 will go in the upper right, and
# 3 will go all the way across the bottom.
#
multiplot <- function(..., plotlist=NULL, file, cols=1, layout=NULL) {
        library(grid)
```

```
        # Make a list from the ... arguments and plotlist
        plots <- c(list(...), plotlist)

        numPlots = length(plots)

        # If layout is NULL, then use 'cols' to determine layout
        if (is.null(layout)) {
                # Make the panel
                # ncol: Number of columns of plots
                # nrow: Number of rows needed, calculated from # of cols
                layout <- matrix(seq(1, cols * ceiling(numPlots/cols)),
                                 ncol = cols, nrow = ceiling(numPlots/cols))
        }

        if (numPlots==1) {
                print(plots[[1]])

        } else {
                # Set up the page
                grid.newpage()
                pushViewport(viewport(layout = grid.layout(nrow(layout), ncol(layout))))

                # Make each plot, in the correct location
                for (i in 1:numPlots) {
                        # Get the i,j matrix positions of the regions that contain this subplot
                        matchidx <- as.data.frame(which(layout == i, arr.ind = TRUE))

                        print(plots[[i]], vp = viewport(layout.pos.row = matchidx$row,
                                                        layout.pos.col = matchidx$col))
                }
        }
}
```

---

6 Main analysis functions

Resultat desires: * **une variable non structurée** : question ouverte * verbatim : liste des réponses * verbatim2 : liste : col 1 = valeurs d'un facteur, col2 = réponses correspondantes

- **une variable**
  - cat1 1 facteur
  - num1c 1 variable continue
  - num1d 1 variables discrete
- **deux variables**
  - cat2 2 facteurs
  - cat1num1
  - num2

6.1 verbatim + verbatim2 = list of text values (as in "other..." answers)

```
verbatim <- function(dataf, nomfact, useNA = "no"){
        dataf <- if (useNA == "no") {
                nonadf(dataf, nomfact)
        } else {
                dataf
        }
        numc <- length(nonavect(dataf[[nomfact]]))
        ptable <- as.data.frame(dataf[[nomfact]])
        colnames(ptable) <- "Verbatim"
        make.result(name = paste0( nomfact, "(verbatim)" ), #modifié
                    funname = verbatim,
                    varnames = c(nomfact = nomfact), # ??? is named vector ok ??
                    ptable = ptable,
                    numcases = numc)
}
```

verbatim2 lists another variable alongside to "explain" verbatims (bynomfact is not filtered for NA values: we want all responses to nomfact)

```
verbatim2 <- function(dataf, nomfact, bynomfact,  useNA = "no"){
        dataf <- if (useNA == "no") {
                nonadf(dataf, nomfact)
        } else {
                dataf
        }
        numc <- nrow(dataf)
        ptable <- dataf[order(dataf[[bynomfact]]), c(bynomfact, nomfact)]
        colnames(ptable) <- c( bynomfact, "Verbatim")
        make.result(name = paste0( nomfact, "(verbatim2) by ", bynomfact), #modifié
                    funname = verbatim2,
                    varnames = c(nomfact = nomfact, bynomfact = bynomfact),
                    ptable = ptable,
                    numcases = numc)
}
```

6.2 cat1 : 1 categorical variable

```
cat1 <- function(dataf, nomfact, useNA = "no",
                 orderfreq = sfdefault("orderfreq"),
                 orderdesc = TRUE, ordervar = "c..nt",
                 orderval = NA, orderfun = sum,
                 dotest = TRUE, # make a chi2 test?
                 digits = 2,
                 plots = sfdefault("cat1plots"),
                 # options for barchart
                 rfreq = TRUE, cfill = sfdefault("filldefault"),
                 # options for piechart
                 scaletitle = sfdefault("scaletitle"),
                 dolabel = sfdefault("dolabel"),
                 minperc = sfdefault("minperc"),
                 labpos = sfdefault("labpos") ) {
```

```
        # useNA = "always, "ifany" or "no",
        # orderfreq = TRUE  or FALSE,
        # descorder =TRUE or FALSE
        # ordervar = variable to use for ordering,
        # orderval = value if the ordering variable is the frequency of ordervar == value

        # reordering the levels:
        dataf[[nomfact]] <-
                orderfact(dataf, nomfact,
                        orderfreq, orderdesc, ordervar, orderval, orderfun)
        # getting rid of NA's ?
        dataf <- if (useNA == "no") {
                nonadf(dataf, nomfact)
        } else {
                dataf
        }
        # make table as dataframe
        tbl <- table(dataf[[nomfact]], useNA = useNA)
        tbl <- data.frame(num = tbl, rfreq = tbl / sum(tbl))
        tbl <- tbl[, c(1,2,4)]
        names(tbl) <- c(nomfact, "num", "rfreq")
        tbl$numlabs <- paste0("n=" ,tbl$num)
        tbl$perclabs <- paste0(100 * round(tbl$rfreq, digits),"%")
        tbl$index <- ave(1:nrow(tbl),  FUN = function(x) 1:length(x)) # rank
        num <- sum(tbl$num)
        # printable table
        ptb <- tbl[1:3]
        colnames(ptb) <- c(nomfact, "Freq.", "Rel.Freq")
        # put relative freq in % units ****************************************Edit
        ptb[[3]] <- 100 * ptb[[3]]

        # Goodness-of-Fit chi-square test for a uniform distribution
        if (dotest) {uchisq <- try.chisq.test(tbl[["num"]])
        } else {uchisq <- NULL
        }

        # bar chart with ggplot2
        # # the data
        # dataf1 <- if (useNA == "no") {
        #         dataf[which(!is.na(dataf[[nomfact]])), ]
        # } else {
        #         dataf
        # }
        # # base ggplot
        # pt <- if (rfreq) {
        #         ggplot(dataf1,
        #                aes_(as.name(nomfact), quote(100 * ..count.. / sum(..count..))))
        # } else {
        #         ggplot(dataf1,
        #                aes_(as.name(nomfact)))
        # }
        # # geom
        # pt <- pt + geom_bar(fill = cfill)
```

```
        # # ylabel
        # if (rfreq) {pt <- pt + ylab(label = "percent")}
        bar <- barchart(dataf, nomvar=nomfact, useNA = useNA, rfreq = rfreq,
                              barwidth = sfdefault("discretebarwidth"),
                              cfill = cfill, percentlabel = sfdefault("percentlabel") )

        pie <- piechart(dataf, nomfact,
                              scaletitle = scaletitle,
                              dolabel = dolabel,
                              minperc = minperc,
                              labpos = labpos )
        # prepare plots for storage
        plot0 <- NULL
        plot1 <- NULL
        plot2 <- NULL
        plot3 <- NULL
        maxplots <- sfdefault("maxplots")
        for (i in 1:min(length(plots), maxplots)) {
                pname <- paste("plot", i-1, sep = "")
                assign(pname, eval(as.name(plots[i])))
        }

        # return
        make.result(name = paste0( nomfact, " (cat1)"), #modifié
                    funname = cat1,
                    varnames = c(nomfact = nomfact),
                    numcases = num,
                    levels = levels(dataf[[nomfact]]),
                    table = tbl,
                    ptable = ptb,
                    chi2 = uchisq,
                    plot = plot0,
                    plot1 = plot1,
                    plot2 = plot2,
                    plot3 = plot3)
}
```

6.3 mocat1 : Multiple Ordered Categorical

```
mocat1 <- function(dataf, prefix, valvect = NULL, valshort = NULL, valname = NULL) {
        variables <- grep(prefix, colnames(dataf), value = TRUE) # => the relevant cols names
        if (is.null(valvect)) {valvect <- variables}
        if (is.null(valshort)) {valshort <- valvect}
        if (is.null(valname)) {valname <- prefix}
        # verify
        if (length(variables) != length(valvect) |
            length(variables) != length(valshort)){
                error(" mocat1 : argument lengths mismatch")
        }
        corrtable <- data.frame(variable = variables,
                                valvect,
                                valshort) # correspondance table for use in graphs and tables
```

```
# Data: keep only useful cols
dataf <- dataf[ , variables]
# keep only useful rows
isuseful <- rep(TRUE, nrow(dataf)) #initialisation
for(i in 1:nrow(dataf))
{isuseful[i] <- !all(is.na(dataf[i, ]))}
dataf <- dataf[isuseful, ]

ncases <- nrow(dataf) # nombre de cas

## make the graph(s): long format for the ranks dfrm
lresdf <- melt(dataf)
lresdf <- nonadf(lresdf,"value") # get rid of NA's

# order the factor in reverse because of coord_flip. useful ?
#lresdf$variable <- orderfact(lresdf , "variable", orderdesc = FALSE) ## ? useful? NO

# compute % of individuals and citations explicitly, record the variable values in lims
restable <- group_by(lresdf, variable) %>%
        summarise(nbcit = n(),
                  rangmed = median(value)) %>%
        arrange(desc(nbcit)) %>%
        mutate(percases = 100 * nbcit / ncases,
               percit = 100 * nbcit / sum(nbcit))

restable$valnames <- vlookup(restable$variable, searchtable = corrtable,
                             searchcol = "variable", returncol = "valvect")
restable$shortname <- vlookup(restable$variable, searchtable = corrtable,
                              searchcol = "variable", returncol = "valshort")

# printable table
ptable <- select(restable, valnames, nbcit, percases, percit, rangmed)
colnames(ptable) <- c(valname, "citations", "% individus", "% citations", "rang median")

lims <- restable$variable # to ensure both plots have the same category order
graphlabels <- as.character(restable$shortname) # short names, in the same order as lims !! as
names(graphlabels) <- lims # (to be sure and not to depend on order later)

p1 <- ggplot(restable, aes(variable, percases)) +
        geom_bar(stat="identity") +
        scale_x_discrete(limits = rev(lims), labels = graphlabels) + #labels = graphlabels
        labs(y = "% individus", x = valname) +
        coord_flip()

p2 <- ggplot(lresdf, aes(variable, value)) +
        geom_violin() +
        geom_jitter(height = 0.3, width = 0.5,
                    alpha = 0.4, color = "steelblue") +
        scale_x_discrete(labels = NULL,
                         limits=rev(lims)) +
        labs(x = NULL, y = 'Rang citation') +
        coord_flip()
```

```
        make.result( name = paste0( prefix, "(mocat)"), #modifié
                     funname = mocat,
                     varnames = c(prefix = prefix),
                     numcases = ncases,
                     ptable = ptable,
                     # plot = quote(multiplot(plot1, plot2, cols = 2)), # autre code possible
                     plot = quote(multiplot(plot1, plot2,
                                            layout = matrix(c(1, 1, 2), nrow = 1, byrow = TRUE))), # co
                     plot1 = p1,
                     plot2 = p2 )
}



# setresult("situation_difficultes") # code that should be in 'depouillement
#
# Variant for multiplot
# layout <- matrix(c(1, 1, 2, 3, 4, 5), nrow = 2, byrow = TRUE)
# multiplot(plotlist = plots, layout = layout)
```

6.4 num1d = 1 numeric d(iscrete)

```
# new definition:
num1d <- function(dataf, nomvar, useNA ="no",
                  digits = sfdefault("digits"), sumdigits = sfdefault("sumdigits"),
                  rfreq = TRUE, width = sfdefault("discretebarwidth", 0.5), cfill = "steelblue") {
        # make a table (with Frequency = nb of rows)
        tb <- table(dataf[[nomvar]])
        num <- sum(tb)
        tbf <- tb/sum(tb)
        tbflabs <- paste0(100* round(tbf,digits), "%")
        tbl <- data.frame(tb, tbf, tbflabs)
        tbl <- tbl[ , c(1,2,4,5)]
        colnames(tbl) <-  c(nomvar, "num", "rfreq", "perclabs")
        tbl$numlabs  <-  paste0("n=", tbl$num)
        tbl$index <- ave(1:nrow(tbl),  FUN = function(x) 1:length(x)) # rank
        # printable table
        ptb <- tbl[1:3]
        ptb[[3]] <- 100 * round(ptb[[3]], digits)
        colnames(ptb) <- c(nomvar, "Freq.", "Rel.Freq")

        s <- sumvector(dataf[[nomvar]])

        # Goodness-of-Fit chi-square test for a uniform distribution
        uchisq <- try.chisq.test(tbl[["num"]])

        # bar chart
        # # data+aes
        # if (useNA == "no") {dataf <- dataf[which(!is.na(dataf[[nomvar]])), ]}
        # if (rfreq) {
        #         pt <- ggplot( dataf,
        #                       aes_(as.name(nomvar),
```

```
        #                             quote(100 * ..count.. / sum(..count..))) )
        # } else {
        #         pt <- ggplot( dataf,
        #                       aes_(as.name(nomvar)) )
        # }
        # # geom
        # pt <- pt + geom_bar(width = width, fill = cfill )
        # # ylabel
        # if (rfreq) {pt <- pt + ylab("percent")}
        bar <- barchart(dataf, nomvar, useNA = useNA, rfreq = rfreq,
                        barwidth = sfdefault("discretebarwidth"),
                        cfill = cfill, percentlabel = sfdefault("percentlabel") )

        # return values
        make.result(name = nomvar,
                    name = paste0( nomvar, "(num1d)"), #modifié
                    funname = num1d,
                    varnames = c(nomvar = nomvar),
                    summaries = s,
                    table = tbl,
                    ptable = ptb,
                    numcases = num,
                    chi2 = uchisq,
                    plot = bar  # pt
                    )
}
```

6.5 num1c = 1 numeric c(ontinuous)

```
# another helper function
# make class labels from bins vector
mkclabs <- function(breaks, sep = " - ", closed = NULL) {
        if (is.null(closed)) {closed <- "right"} # default close="right"
        # closed
        if (closed == "right") {
                bchar <- "]"
        } else if (closed == "left") {
                bchar <- "["
        } else {
                bchar <- "|"
                warning("Invalid 'closed' argument in mkclabs")
        }

        left <- head(breaks, length(breaks) - 1)
        right <- tail(breaks, length(breaks) - 1)
        mapply(function(x,y){paste0(bchar,x, sep , y, bchar)},
               left, right, SIMPLIFY =TRUE)
}
```

```
# num1c
num1c <- function(dataf, nomvar, usedensity = FALSE, plot_density = FALSE,
                  fillhist = sfdefault("filldefault"), color_density = "red", digits = 2, # ? modifier
                  bins = NULL, closed = NULL, ...) {  # ... = addtl arguments for geom_hist
        if (plot_density) {usedensity <- TRUE} # plot_density overrides usedensity
        # bins = Null, integer, or a function name : "nclass.Sturges", "nclass.FD" , "nclass.scott"
        # get or compute bins (as integer)
        if (!is.null(bins)) {
                if ("character" %in% class(bins) ) {
                        bins <-  do.call(bins, list(nonavect(dataf[[nomvar]])))
                } else {bins <- NULL
                warning("bins is not a function", call. = TRUE)}
        }
        # make histogram
        p <- ggplot(dataf, aes_(as.name(nomvar))) +
                if (usedensity) {geom_histogram(aes(y=..density..),
                                                bins = bins, fill = fillhist,...)
                } else {geom_histogram(bins = bins, fill = fillhist, ...)}

        if (plot_density) {p <- p + geom_density(color=color_density) }

        # make summaries vector + get number of cases
        s = sumvector(dataf[[nomvar]])
        num = s["n"] # number of cases

        # get the frequency table from ggplot
        tb <- ggplot_build(p)$data[[1]][ , 1:8]
        # add  columns to it
        tb$rfreq <- tb$count/num
        tb$numlabs <-  paste0("n=", tb$count)
        tb$perclabs <- paste0(100* round(tb$rfreq, digits), "%")
        tb$index <- ave(1:nrow(tb),  FUN = function(x) 1:length(x)) # rank
        # done, compute more info
        cbinw <- unique(round(tb$xmax-tb$xmin,digits)) # get binwidth
        cbreaks <- with(tb, c(xmin[1],xmax)) # get breaks vector from table
        clabs <- mkclabs(cbreaks, closed = closed) # make class lablels
        # make a printable table
        ptb <- data.frame(
                class = clabs,
                center = tb$x,
                freq = tb$count,
                rfreq = tb$rfreq * 100
        )

        # Uniform Chi2 test
        uchisq <- try.chisq.test(tb$count)
        # warn if different class widths
        if (length(cbinw) >= 2) {
                warning(paste0("Unif chi2 test ",
                               nomvar,
                               " called with different class widths!",
                               call. = TRUE)) }
```

```
        # return values
        make.result( name = paste0( nomvar, "(num1c)"), #modifié
                     funname = num1c,
                     varnames = c(nomvar = nomvar),
                     numcases = num,
                     summaries = s,
                     table = tb,
                     ptable = ptb,
                     details =list(binwidths = cbinw,
                                   breaks = cbreaks,
                                   closed = closed),
                     chi2 = uchisq,
                     plot = p)
}
```

6.6 cat2 = 2 categorical vars

```
# definition
cat2 <- function(dataf, nomfact1, nomfact2,  useNA = "no",
                 orderfreq1 = sfdefault("orderfreq"), orderdesc1 = TRUE,
                 ordervar1 = "c..nt",
                 orderval1 = NA, orderfun1 = sum, nlevel1 =NULL,
                 orderfreq2 = sfdefault("orderfreq"), orderdesc2 = TRUE,
                 ordervar2 = "c..nt",
                 orderval2 = NA, orderfun2 = sum, nlevel2 =NULL,
                 rfreq = TRUE, digits = 2, cfill = sfdefault("filldefault") ) {
        # useNA = "always, "ifany" or "no", orderfreq = TRUE  or FALSE,
        # descorder =TRUE or FALSE
        # ordervar = variable to use for ordering

        # reordering the levels:
        # nomfact2 first
        dataf[[nomfact2]] <-  orderfact(dataf, nomfact2, orderfreq2, orderdesc2,
                                        ordervar2, orderval2, orderfun2, nlevel2)
        # nomfact1
        if(orderfreq1 == TRUE &
           ordervar1 == nomfact2 & !is.na(orderval1)){ # fr?quences conditionnelles!
                #print("Frequ cond")
                tbl <- condfreqtable(dataf, nomfact1, nomfact2,  useNA = "no")
                #print("apres Frequ cond table")
                tbl <- tbl[tbl[[nomfact2]] == orderval1, ]
                # print("tbl") #dbg
                # print(tbl) #dbg
                tbl[[nomfact1]] <- orderfact(tbl, nomfact1,
                                             orderfreq1, orderdesc1,
                                             ordervar = "perc",
                                             orderfun = orderfun1) #***************
                dataf[[nomfact1]] <- orderfact(dataf, nomfact1,
                                               nlevels = levels(tbl[,nomfact1]))
        } else { # autres cas
                dataf[[nomfact1]] <- orderfact(dataf, nomfact1, orderfreq1,
                                               orderdesc1, ordervar1, orderval1,
```

```
                                        orderfun1, nlevel1)
}

#         print(levels(dataf[[nomfact1]])) #debug
#         print(levels(dataf[[nomfact2]])) #debug
# make table as dataframe
tblcrois <- table(dataf[[nomfact1]], dataf[[nomfact2]], useNA = useNA)

tbl <- as.data.frame(tblcrois)
colnames(tbl) <- c(nomfact1,nomfact2,"num")
# print(tbl) #debug
num <- sum(tbl$num)

tbl1 <- summarize_(group_by_(tbl,as.name(nomfact1)),
                   num=quote(sum(num))) # shit with non-standard eval
tbl2 <- summarize_(group_by_(tbl,as.name(nomfact2)),
                   num=quote(sum(num))) # shit with non-standard eval

# supplement tbl1
tbl1$numlabs = paste0("n=", tbl1$num)
if (!is.na(orderval1)){
        tbl1$numval <- tblcrois[ ,orderval1] # keep it, not a df
        tbl1$percval <- tbl1$numval / tbl1$num
        tbl1$perclabs <- paste0(100 * round(tbl1$percval, digits), "%")
}
tbl1$index <- ave(1:nrow(tbl1),  FUN = function(x) 1:length(x)) # rank

# Chi-square test for independence
ichisq <- try.chisq.test(tblcrois)

#  bar chart with ggplot2
#  data
dataf2 <- if (useNA == "no") {
        dataf[which(!is.na(dataf[[nomfact1]]) &
                            !is.na(dataf[[nomfact2]])), ]
} else {dataf
}
#  plot
pt <- ggplot(dataf2) +
        geom_bar(aes_(as.name(nomfact1), fill = as.name(nomfact2)),
                 position = "Fill") +
        guides(fill = guide_legend(reverse = TRUE)) +
        ylab("percent")

make.result(
        name = paste0( nomfact1, " (cat2) by ", nomfact2 ), #modifié
        funname = cat2,
        varnames = c(nomfact1 = nomfact1, nomfact2 = nomfact2),
        numcases = num,
        levels =levels(dataf[[nomfact1]]),
        levels2 =levels(dataf[[nomfact2]]),
        table = tbl1,
        table1 = tblcrois,
```

```
                details = list(tbl=tbl,tbl2=tbl2),
                chi2 = ichisq,
                plot = pt
        )
}
```

6.7 cat1num1 (cat1num1c, cat1num1d)

```
#
# fonctions de d?termination du nombre de classes dabs un histogramme
# nclass.Sturges(mpg$hwy)
# nclass.FD(mpg$hwy)
# nclass.scott(mpg$hwy)
#
#
```

6.7.1 fonctions de generation de graphiques

useful

```
# gets the bins number using a function 'bins' == 'nclass.???'
get.bins <- function(dataf, nomvar, bins) {
        if (!is.null(bins)) {
                if ("character" %in% class(bins) ) {
                        bins <-  do.call(bins, list(nonavect(dataf[[nomvar]])))
                } else if ("numeric" %in% class(bins) | "integer" %in% class(bins) ) {
                } else {
                        bins <- NULL
                        warning("bins is not a function", call. = TRUE)
                }
        }
        bins
}


get.breaks <- function(dataf,nomvar,bins) {
        pretty(range(mpg$hwy,
                     n =  get.bins(dataf, nomvar,bins)))
}

# # ex
# range(mpg$cty)
# get.bins(mpg, "cty", "nclass.Sturges")
# get.breaks(mpg, "cty", "nclass.Sturges")
#
# get.bins(mpg, "cty", "nclass.FD")
# get.breaks(mpg, "cty", "nclass.FD")
#
# get.breaks(mpg, "cty", 17)
# get.breaks(mpg, "cty", 3)
# get.breaks(mpg, "cty", 20)
```

```r
bins <- "nClass.Sturges"
class(bins)
```

```
## [1] "character"
```

```r
bins <- 25
class(bins)
```

```
## [1] "numeric"
```

```r
("numeric" %in% class(bins) | "integer" %in% class(bins))
```

```
## [1] TRUE
```

```r
#
# nb <- get.bins(mpg,"hwy","nclass.Sturges")
# r <- range(mpg$hwy)
# w <- r[2]-r[1]
#
# bn <- seq(r[1], r[2], by = w/nb)
# pretty(bn)
# pretty(c(r[1], r[2]), n = nb)
```

6.7.2 cat1num1c (in progress)

```r
catnum1c <- function(dataf, nomfact, nomvar,  useNA = "no",
                     orderfreq = TRUE, orderdesc = TRUE, ordervar = "c..nt",
                     orderval = NA, orderfun = sum, nlevel =NULL,
                     labellayer = "", labelall = "All values", labelgroups = "by goup",
                     breaks = NULL, closed = NULL,
                     rfreq = TRUE, digits = sfdefault("digits"), cfill = sfdefault("filldefault")){
        #ordering the factor if needed
        dataf[[nomfact]] <-  orderfact(dataf, nomfact, orderfreq, orderdesc,
                                       ordervar, orderval, orderfun, nlevel)
        # make a plot (box-jitter)
        pt1 <- cbyfboxjit(dataf, varf=nomfact, varc=nomvar, useNA = useNA,
                          labellayer = labellayer, labelall = labelall, labelgroups = labelgroups)
        # faceted histogram
        pt2 <- cbyffachistogram(dataf, varf=nomfact, varc=nomvar, useNA = useNA,
                                usedensity = FALSE, usendensity = FALSE,
                                breaks = breaks, closed = closed)
        # summaries
        # c <- condsummaries(dataf = dataf,vname = nomvar,fname = nomfact)

        # tables
        # make factor with cut
        # tb1 =

        # Planned:
        # breaks = NULL,
        # closed= NULL,
        # table = NULL,
        # tabledf = NULL,
        # ptable = NULL,
```

```
        # chi2 = NULL,
        # anova = NULL,
        # plot = NULL )

        # name
        name=c(nomvar, nomfact) # inutile maintenant ?
        numcases = length(!is.na(dataf[[nomvar]] & !is.na(dataf[[nomfact]])))
        # summaries
        #
        s <- try(condsummaries(dataf, nomvar, nomfact))
        # levels
        nlevels = levels(dataf[[nomfact]]) # see if reorder
        # breaks = breaks # include in output, nothing to compute ??? depends
        # table
        # table <-
        make.result(
                name = paste0( nomvar, " (catnum1c) by ", nomfact ), #modifié
                funname = catnum1c,
                varnames = c(nomvar = nomvar, nomfact = nomfact),
                numcases = numcases,
                summaries = s,
                levels = nlevels,
                plot1 = pt1,
                plot2 = pt2
        )

}
```