bibliotheque.fonctions.paie.R

Fab

Wed Nov 26 13:44:03 2014

```
##
#  Fonctions auxiliaires
##

library(MASS)

chemin <-  function(fichier)
  file.path(chemin.dossier.données, fichier)

file2utf8 <- function(nom, encodage.in = encodage.entrée)  {

 chem <- chemin(nom)
 err <- system2(iconv, c("-f", encodage.in, "-t", "UTF-8", shQuote(chem), "-o", "temp"))
 if (! err)  err <- system2("mv", c("temp", shQuote(chem))) else stop("Erreur d'encodage avec iconv")
 if (! err)  message("Conversion réussie") else stop("Erreur de copie fichier après encodage avec iconv
}

en.séparateurs <- function(chem)  {

  commande <- sed %+% " -e s/,/\\./g -e s/;/,/g -i " %+% shQuote(chem)
  shell(commande)
}

fr.séparateurs <- function(chem)  {

  commande <- sed %+% " -e s/,/;/g -e s/\\./,/g -i " %+% shQuote(chem)
  shell(commande)
}


# Trouve le numéro de la ligne à laquelle se situe la liste des noms de variables
# en recherchant soit le mot "Matricule" soit une expression du type "Code..."
# Il faudra déduire ce "skip" du read.csv pour récupérer proprement les noms de variable

# Pour cela on scanne les 25 premières lignes de la table une première fois


trouver.valeur.skip <-  function(chemin.table, encodage, classes = NA, séparateur.liste = séparateur.lis
  max(
    sapply(
      read.csv(chemin.table, sep=séparateur.liste, dec=séparateur.décimal, nrows = 25, fileEncoding = e
      function(x)
      {
        m <- match(champ.détection.1, x, nomatch = 0 )
        if (m == 0)
          m <- pmatch(champ.détection.2, x, nomatch = 0, duplicates.ok = FALSE )
        return(m)
```

```
      }
    ))


# selectionner.cle.matricule <-  function(Base1, Base2)
# {
#   if (fusionner.nom.prénom) {
#      subset(Base1,
#            select = c("Nom", "Prénom", étiquette.matricule, setdiff(names(Base1),names(Base2))))
#     } else {
#      subset(Base1,
#          select = c(étiquette.matricule, setdiff(names(Base1), names(Base2))))
#     }
# }

sélectionner.clé <-  function(base1, base2)
{
  Base1 <- get(base1)
  Base2 <- get(base2)

  if (fusionner.nom.prénom) {

    Set1 <- c("Mois", "Année", étiquette.matricule, setdiff(names(Base1), names(Base2)))
    Set2 <- setdiff(names(Base2), c("Nom", "Prénom", étiquette.matricule))

    assign(base1,
           subset(Base1, select = c("Nom", "Prénom", Set1)),
           envir = .GlobalEnv)

    assign(base1,
           cbind(as.data.frame(convertir.nom.prénom.majuscules(Base1[, c("Nom", "Prénom")])),
                 Base1[, Set1]),
           envir = .GlobalEnv)

    assign(base2,
           cbind(as.data.frame(convertir.nom.prénom.majuscules(Base2[, c("Nom", "Prénom")])),
                 Base2[, Set2]),
           envir = .GlobalEnv)



  } else {

    assign(base1, subset(Base1,
                         select = c(étiquette.matricule,"Mois","Année",
                         setdiff(names(Base1), names(Base2)))), envir = .GlobalEnv)

  }
}

#chem.dot <- paste0("\'",chem, ".dot", "\'")
#system(paste0("sed -e s/,/\\./g < \'", chem,"\' > ", chem.dot), wait = TRUE)
```

```
read.csv.skip <- function(x, encodage = encodage.entrée, classes = NA, étiquettes = NULL, drop = NULL,
                          rapide = FALSE, séparateur.liste = séparateur.liste.entrée, séparateur.décimal
                          convertir.encodage = TRUE)
{
  chem <- chemin(x)
  if (! rapide) {

    T <- read.csv(chem,
                  comment.char = "",
                  sep =séparateur.liste,
                  dec = séparateur.décimal,
                  colClasses = classes,
                  skip = trouver.valeur.skip(chem, encodage, séparateur.liste = séparateur.liste, sépa
                  encoding = encodage)

    if (!is.null(drop)) { T <- T[-(drop)] }

  } else {

    if (encodage != "UTF-8" && convertir.encodage) {
      message("La table en entrée doit être encodée en UTF-8")
      if (convertir.encodage) message("Conversion via iconv du format " %+% encodage %+% " au format UT
      file2utf8(x, encodage.in = encodage)
    }

    if (is.na(classes)) classes = NULL
    T <- try(data.table::fread(chem,
                     sep = séparateur.liste,
                     header = TRUE,
                     verbose = FALSE,
                     skip = champ.détection.1,
                     colClasses = classes,
                     showProgress = FALSE))

    if (inherits(T, "try-error") && grepl("The supplied 'sep' was not found", T, fixed = TRUE)) {
      message("Conversion des séparateurs...")
      en.séparateurs(chem)
      message("Séparateurs convertis.")
      T <- read.csv.skip (x,
                    encodage,
                    classes,
                    étiquettes,
                    drop,
                    rapide,
                    séparateur.liste,
                    séparateur.décimal)
    }
  }

if (!is.null(étiquettes)) names(T) <- étiquettes


return(T)
```

```
}

Sauv.base <- function(chemin.dossier, nom, nom.sauv, encodage = encodage.sortie, sep = séparateur.liste
{
  message("Sauvegarde de ", nom)
  write.table(get(nom),
              paste0(chemin.dossier, "/", iconv(nom.sauv, to = encodage, mark = FALSE), ".csv"),
              quote = FALSE,
              sep = sep,
              dec = dec,
              row.names = FALSE,
              fileEncoding = encodage)
}

sauv.bases <- function(dossier, ...)
{
  if (!see_if(is.dir(dossier)))
  {
    stop("Pas de dossier de travail spécifié")
  }

  tmp <- as.list(match.call())
  tmp[1] <- NULL

  message("Dans le dossier ", dossier," :")
  invisible(lapply(tmp[-1], function(x) if (exists(x)) Sauv.base(dossier, x, x)))
}

# Utiliser une assignation globale
# car la fonction anonyme ne comporte que de variables locales

Read.csv <- function(base.string, vect.chemin, charger = charger.bases, colClasses = NA, colNames = NULL
                     drop = NULL, séparateur.liste = séparateur.décimal.entrée, séparateur.décimal = sép
                     rapide = FALSE, convertir.encodage = TRUE, encodage = encodage.entrée)  {

    if (charger.bases) {

        assign(base.string,
               do.call(rbind, lapply(vect.chemin,
                                     read.csv.skip,
                                        classes = colClasses,
                                        étiquettes = colNames,
                                        séparateur.liste = séparateur.liste,
                                        séparateur.décimal = séparateur.décimal,
                                        drop = drop,
                                        convertir.encodage = convertir.encodage,
                                        encodage = encodage,
                                        rapide = rapide)),
               envir = .GlobalEnv)
    }
}

pretty.print <- function(x) cat(gsub(".", " ",deparse(substitute(x)), fixed = TRUE), "   ", x,"\n")
```

```
Résumé <- function(x,y, align = 'r', extra = 0, ...)  {

     Y <- na.omit(y)

     S <- cbind(c("Minimum", "1er quartile", "Médiane", "Moyenne", "3ème quartile", "Maximum"),
                prettyNum(sub("[M13].*:", "", summary(Y, ...)), big.mark = " "))

     if (! missing(extra))
        if (extra == "length") {
          L <- if (is.vector(Y)) length(Y) else nrow(Y)
          S <- cbind(S, c("", "", "", L, "", ""))
        } else {
        if (is.numeric(extra))
                S <- cbind(S, c("", "", "", as.character(extra), "", ""))
        }

     dimnames(S)[[2]] <- c("Statistique", x)

     kable(S, row.names = FALSE, align = align)
}

Tableau <- function(x, ...)
{
  V <- c(...)
  if ("sep.milliers" %in% names(V))
  {
    sep.milliers <- V["sep.milliers"]
    V$sep.milliers <- NULL
  }  else
  sep.milliers <- " "

  T <- t(prettyNum(V, big.mark = sep.milliers))
  T <- as.data.frame(T)
  names(T) <- x
  kable(T, row.names = FALSE, align = "c", booktabs=TRUE)
}

Tableau.vertical <- function(colnames, rownames, extra = "", ...)
{
    tmp <- c(...)

    if (! all(lapply(tmp, is.function))) {
      message("all arguments must be functions")
      return("")
    }

    lr <- length(rownames)

    h <- function(x) as.numeric(sub(",",".", sub(" ", "", x, fixed=T), fixed=T))

    g <- function(f) {
        S <- rep("", lr)
```

```
        S[ceiling(lr/2)] <- as.character(prettyNum((h(f(rownames[lr]))/h(f(rownames[1])) - 1) * 100, dig

        S
    }


    if (! missing(extra) && (is.character(extra)) && (extra == "variation")) {
      T <- data.frame(rownames)
      NT <- colnames[1]
      ltmp <- length(tmp)
      for (x in seq_len(ltmp)) {
        T <- cbind(T, sapply(rownames, tmp[[x]]), g(tmp[[x]]))
        NT <- c(NT, colnames[[x + 1]], "Variation (%)")
      }

      names(T) <- NT
    } else {
      T <- data.frame(rownames, lapply(tmp, function(f) sapply(rownames, f)))
      names(T) <- colnames
    }

    kable(T, row.names = FALSE, align = "c", booktabs=TRUE)
}

Tableau.vertical2 <- function(colnames, rownames, ...)
{
  tmp <- list(...)

  T <- data.frame(rownames,
                  lapply(tmp, function(y) formatC(y,
                                                  big.mark=" ",
                                                  width="12",
                                                  format="f",
                                                  digits=1,
                                                  decimal.mark=",",
                                                  preserve.width="common")))
  names(T) <- colnames

  kable(T, row.names = FALSE, align = NULL, booktabs=TRUE)
}


#   julian.date.début.période <- julian(as.Date(paste0("01/01/", début.période.sous.revue), date.format
#   julian.exercice.suivant.premier <- julian(as.Date(paste0("01/01/",(début.période.sous.revue+1)), da
#   julian.date.fin.période   <- julian(as.Date(paste0("01/01/", fin.période.sous.revue+1), date.format
#   julian.exercice.dernier <- julian(as.Date(paste0("01/01/",fin.période.sous.revue), date.format))
#
# calcul.nb.jours <- function(entrée, sortie)
# {
#
#   julian.entrée <-
#     ifelse(entrée == "",
#            julian.date.début.période,
```

```
#           max(julian.date.début.période, julian(as.Date(entrée, date.format))))
#
#   julian.sortie <-
#     ifelse(sortie == "",
#            julian.date.fin.période,
#            min(julian.date.fin.période, julian(as.Date(sortie, date.format))))
#
#   return (julian.sortie - julian.entrée)
# }
#
# calcul.nb.jours.mois.deprecated <- function(mois.entrée, mois.sortie, année)
# {
#
#   # calcul exact pour une période continue
#
#   if (mois.sortie < mois.entrée) return(0);
#
#   if (mois.sortie == 12)
#   {
#      année.sortie <- année +1
#      mois.sortie = 1
#   }
#   else
#   {
#     année.sortie <- année
#     mois.sortie <- mois.sortie + 1
#   }
#
#    as.numeric(as.Date(paste0("01",
#                                    formatC(mois.sortie, width = 2, flag = "0"),
#                                    année.sortie),
#                       "%d%m%Y")
#               - as.Date(paste0("01",
#                                    formatC(mois.entrée, width = 2, flag = "0"),
#                                    année),
#                              "%d%m%Y"))
# }

v.jmois  <-  c(31, 28, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)
v.jmois.leap  <-  c(31, 29, 31, 30, 31, 30, 31, 31, 30, 31, 30, 31)

calcul.nb.jours.mois <- function(Mois, année)   if ((année - 2008) %%4 == 0) {
    return(sum(v.jmois.leap[Mois]))
    } else {
    return(sum(v.jmois[Mois]))
  }


positive <- function(X) X[!is.na(X) & X > 0]
non.null <- function(X) X[!is.na(X) & X != 0]
significatif <- function(x) !is.na(x) & abs(x) > 0.01

installer.paquet <- function(paquet, rigoureusement = FALSE)
```

```
{
  if (missing(paquet))  return(NA_character_)
  Paquet <- deparse(paquet)
  if (length(find.package(Paquet, quiet = TRUE)) == 0)
  {
    install.packages(Paquet)
    if (length(find.package(Paquet, quiet = TRUE)) !=0 )
    {
      message(Paquet, " a été installé.")
      return(invisible(1))
    }
    else
    {
      message(Paquet, " n'a pas été installé.")
      if (rigoureusement == TRUE)
      {
        message("Arrêt: le paquet ", Paquet, " n'a pas pu être installé.")
        stop("Fin")
      }
      return(invisible(0))
    }
  }
  else
    message(Paquet, " est déjà installé.")
  return(invisible(0))
}

installer.paquets <- function(..., rigoureusement = FALSE)
{
  tmp <- as.list(match.call())
  tmp[1] <- NULL
  if (!missing(rigoureusement)) tmp[length(tmp)] <- NULL
  if (length(tmp) == 0) return(0)

  invisible(do.call(sum, lapply(tmp, function(x) installer.paquet(x, rigoureusement))))
}

convertir.nom.prénom.majuscules <- function(S)
{

  S[ , c("Nom", "Prénom")] <- apply(S[ , c("Nom", "Prénom")],
                                    2,
                                    function(x)
                                      toupper(chartr("éèôâçë","eeoaice", x)))


}

#Age fin décembre de l'Année en années révolues si né au XXème siècle
# On trouve quelques valeurs aberrantes correspondant à des NIr non conventionnels par ex 8041620130028
extraire.nir <- function(Base, Année)  {
  age <- Année - (as.numeric(substr(as.character(
  format(Base[ , Nir], scientific = FALSE)),
```

```
  2, 3))
  + 1900)
  ifelse(age < 80, age, NA)
}


# tester.homogeneite.matricules(Base)

#  Teste si, dans une base, la proportion d'enregistrements Noms-Prénoms dont les matricules ne sont pa
#  reste inférieure à une marge de tolérance fixée (taux.tolérance.homonymie)
#  utilité : tester si l'appariement sur Nom-Prénom au lieu de matricule sera acceptable

tester.homogeneite.matricules <- function(Base) {

  message("Contrôle sur la cohérence de l'association Nom-Prénom-Matricule (homonymies et changements d
  S <- convertir.nom.prénom.majuscules(Base[ , c("Nom", "Prénom", "Matricule")])

  with.matr    <-   nrow(unique(S))
  without.matr <-   nrow(unique(S[ , c("Nom", "Prénom")]))

  message("Matricules distincts: ", with.matr)
  message("Noms-Prénoms distincs: ", without.matr)

  if (with.matr  >   (1 + taux.tolérance.homonymie/100) * without.matr)
  {
     message(paste0("Résultats trop différents (", taux.tolérance.homonymie, " % de marge tolérée). Cha
     if (fusionner.nom.prénom == FALSE)
       stop("Vous pouvez essayer de fusionner sur Nom, Prénom en spécifiant fusionner.nom.prénom <- TRU
  }
}

longueur.non.na <- function(v) length(v[!is.na(v)])

# opérateurs infixe

# concaténer deux strings

`%+%` <- function(x, y) paste0(x,  y)

`%*%` <- function(x, y) if (is.na(x) | is.na(y)) return(0) else return(x*y)

# numérotation des tableaux

numéro.tableau <- 0

incrément <- function() {
  numéro.tableau <<- numéro.tableau +1
  numéro.tableau
}
```