Fonctions

Daniel Rakotomalala

Les fonctions sont présentées par ordre alphabétiques. Toutefois, certaines fonctions dont le nom commence par une lettre donnée peuvent dépendre de fonctions dont le nom commence par une lettre se plaçant à une ordre plus élevé dans l'alphabet.

bealer()

```
# Applique la méthode décrite dans Boucher et al. 2014.
# Limitée à l'instrument JG~2X.


bealer <- function (data, depvar, pmeandepvar, controls, pmeancontrols,
                peerlevel, peerlevelsize,
                felevel, first_step = FALSE) {
  data2 <- cbind(lapply(c(depvar, pmeandepvar, controls, pmeancontrols),
         function (x) transmute(group_by(data, eval(parse(text = felevel))),
                                eval(parse(text = x)) -
                                  mean(eval(parse(text = x)), na.rm = TRUE)) %>%
           ungroup %>% (function (d) d[, - 1])) %>%
    do.call(what = cbind) %>%
    set_colnames(paste("wfe_", c(depvar, pmeandepvar, controls, pmeancontrols), sep = "")),



    lapply(controls,
           function (c) transmute(group_by(data, eval(parse(text = felevel))),
                                  (eval(parse(text = c)) -
                                     mean(eval(parse(text = c)), na.rm = TRUE)) /
                                    ((eval(parse(text = peerlevelsize)) - 1) ^ 2)) %>%
             ungroup %>% (function (d) d[, - 1])) %>%
      do.call(what = cbind) %>%
      set_colnames(paste("jgs_", controls, sep = ""))) %>%
    (function (d) cbind(data, d))



  e1 <- ivreg(as.formula(paste(
    "wfe_", depvar, " ~ ", "wfe_", pmeandepvar,
    " + ",
    paste(paste("wfe_", controls, sep = ""), collapse = " + "),
    " + ",
    paste(paste("wfe_", pmeancontrols, sep = ""), collapse = " + "),
    " - 1 | . - ",
    "wfe_", pmeandepvar,
```

```
   " + ",
   paste(paste("jgs_", controls, sep = ""), collapse = " + ")
 , sep = "")),

 data = data2)


 betahat <- coef(e1)[paste("wfe_", pmeandepvar, sep = "")]

 for (i in controls) assign(paste("gammahat_wfe_", i, sep = ""),
                            coef(e1)[paste("wfe_", i, sep = "")])

 for (j in pmeancontrols) assign(paste("deltahat_wfe_", j, sep = ""),
                                 coef(e1)[paste("wfe_", j, sep = "")])




datafe <- lapply(controls,
                 function (c) transmute(group_by(data2, eval(parse(text = felevel))),
                                        mean(eval(parse(text = c)), na.rm = TRUE)) %>%
                   ungroup %>%
                   (function (d) d[, - 1])) %>%
  do.call(what = cbind) %>%
  set_colnames(paste("mfe_", controls, sep = ""))



datapeer <- lapply(controls,
                   function (c) transmute(group_by(data2, eval(parse(text = peerlevel))),

                                          eval(parse(text = c)) - mean(eval(parse(text = c)), na.rm = T

                                          mean(eval(parse(text = c)), na.rm = TRUE)

                                          ) %>%
                     ungroup %>%
                     (function (d) d[, - 1])) %>%
  do.call(what = cbind) %>%
  set_colnames(as.vector(sapply(controls,
                                function (c) c(paste("wpeer_", c, sep = ""),
                                               paste("mpeer_", c, sep = ""))))




data2 <- cbind(data2, datafe, datapeer)

data2 <- mutate(data2,

                jyhat = eval(parse(text = paste(
                  "((gammahat_wfe_", controls, " - ",
                  "deltahat_wfe_", pmeancontrols, " / ",
```

```
                  " (", peerlevelsize, " - 1", "))", " / ",
                  "(1 + ", "betahat / ",
                  "(", peerlevelsize, " - 1))) * ",
                  "wpeer_", controls, " + ",

                  "((gammahat_wfe_", controls, " + ",
                  "deltahat_wfe_", pmeancontrols, ") / ",
                  "(1 - betahat)) * ",
                  "(mpeer_", controls, " - ",
                  "mfe_", controls, ")",

                sep = "", collapse = " + ")))

                ) %>%
   (function (d) group_by(d, eval(parse(text = peerlevel))) %>%
      mutate(jgyhat = (sum(jyhat) - jyhat) / (eval(parse(text = peerlevelsize)) - 1)) %>%
      ungroup)



   e2 <- ivreg(as.formula(paste(
     "wfe_", depvar, " ~ ", "wfe_", pmeandepvar,
     " + ",
     paste(paste("wfe_", controls, sep = ""), collapse = " + "),
     " + ",
   paste(paste("wfe_", pmeancontrols, sep = ""), collapse = " + "),
   " - 1 | . - ",
   "wfe_", pmeandepvar,
   " + jgyhat",

   sep = ""

  )),

  data = data2)

if (first_step == TRUE) {
  list(e1, e2) %>% setNames(c("step1", "step2"))
} else e2




}
```

bealer2()

```
# une version augmentée pour rajouter une ou plusieurs variables endogènes à instrumenter à chaque étap

bealer2 <- function (data, depvar, pmeandepvar, controls, pmeancontrols,
                peerlevel, peerlevelsize,
                felevel, first_step = FALSE,
```

```
              varendo = NULL, instvarendo = NULL) {
data2 <- cbind(lapply(c(depvar, pmeandepvar, controls, pmeancontrols, varendo, instvarendo),
       function (x) transmute(group_by(data, eval(parse(text = felevel))),
                              eval(parse(text = x)) -
                                mean(eval(parse(text = x)), na.rm = TRUE)) %>%
         ungroup %>% (function (d) d[, - 1])) %>%
  do.call(what = cbind) %>%
  set_colnames(paste("wfe_", c(depvar, pmeandepvar, controls, pmeancontrols,
                               varendo, instvarendo),
                     sep = "")),



  lapply(controls,
         function (c) transmute(group_by(data, eval(parse(text = felevel))),
                                (eval(parse(text = c)) -
                                   mean(eval(parse(text = c)), na.rm = TRUE)) /
                                  ((eval(parse(text = peerlevelsize)) - 1) ^ 2)) %>%
           ungroup %>% (function (d) d[, - 1])) %>%
    do.call(what = cbind) %>%
    set_colnames(paste("jgs_", controls, sep = ""))) %>%
  (function (d) cbind(data, d))



e1 <- ivreg(as.formula(paste(
  "wfe_", depvar, " ~ ", "wfe_", pmeandepvar,
  " + ",
  paste(paste("wfe_", controls, sep = ""), collapse = " + "),
  " + ",
  paste(paste("wfe_", pmeancontrols, sep = ""), collapse = " + "),
  ifelse(is.null(varendo), NULL, paste("+ wfe_", varendo, sep = "")),
  " - 1 | . - ",
  "wfe_", pmeandepvar,
  " + ",
  paste(paste("jgs_", controls, sep = ""), collapse = " + "),
  ifelse(is.null(varendo), NULL, paste("- wfe_", varendo,
                                       " + wfe_", instvarendo, sep = ""))
, sep = "")),

data = data2)


betahat <- coef(e1)[paste("wfe_", pmeandepvar, sep = "")]

for (i in controls) assign(paste("gammahat_wfe_", i, sep = ""),
                           coef(e1)[paste("wfe_", i, sep = "")])

for (j in pmeancontrols) assign(paste("deltahat_wfe_", j, sep = ""),
                                coef(e1)[paste("wfe_", j, sep = "")])
```

```
datafe <- lapply(controls,
                 function (c) transmute(group_by(data2, eval(parse(text = felevel))),
                                        mean(eval(parse(text = c)), na.rm = TRUE)) %>%
                   ungroup %>%
                   (function (d) d[, - 1])) %>%
  do.call(what = cbind) %>%
  set_colnames(paste("mfe_", controls, sep = ""))



datapeer <- lapply(controls,
                   function (c) transmute(group_by(data2, eval(parse(text = peerlevel))),

                                          eval(parse(text = c)) - mean(eval(parse(text = c)), na.rm = 

                                          mean(eval(parse(text = c)), na.rm = TRUE)

                                          ) %>%
                     ungroup %>%
                     (function (d) d[, - 1])) %>%
  do.call(what = cbind) %>%
  set_colnames(as.vector(sapply(controls,
                                function (c) c(paste("wpeer_", c, sep = ""),
                                               paste("mpeer_", c, sep = ""))))) 



data2 <- cbind(data2, datafe, datapeer)

data2 <- mutate(data2,

                jyhat = eval(parse(text = paste(
                  "((gammahat_wfe_", controls, " - ",
                  "deltahat_wfe_", pmeancontrols, " / ",
                  " (", peerlevelsize, " - 1", "))", " / ",
                  "(1 + ", "betahat / ",
                  "(", peerlevelsize, " - 1))) * ",
                  "wpeer_", controls, " + ",

                  "((gammahat_wfe_", controls, " + ",
                  "deltahat_wfe_", pmeancontrols, ") / ",
                  "(1 - betahat)) * ",
                  "(mpeer_", controls, " - ",
                  "mfe_", controls, ")",

                sep = "", collapse = " + ")))

                ) %>%
  (function (d) group_by(d, eval(parse(text = peerlevel))) %>%
     mutate(jgyhat = (sum(jyhat) - jyhat) / (eval(parse(text = peerlevelsize)) - 1)) %>%
     ungroup)
```

```
   e2 <- ivreg(as.formula(paste(
     "wfe_", depvar, " ~ ", "wfe_", pmeandepvar,
     " + ",
     paste(paste("wfe_", controls, sep = ""), collapse = " + "),
     " + ",
   paste(paste("wfe_", pmeancontrols, sep = ""), collapse = " + "),
   ifelse(is.null(varendo), NULL, paste("+ wfe_", varendo, sep = "")),
   " - 1 | . - ",
   "wfe_", pmeandepvar,
   " + jgyhat",
   ifelse(is.null(varendo), NULL, paste(" - wfe_", varendo,
                                        " + wfe_", instvarendo, sep = ""))
   ,

   sep = ""

  )),

  data = data2)

if (first_step == TRUE) {
  list(e1, e2) %>% setNames(c("step1", "step2"))
} else e2




}
```

binedmeaner()

```
# Moyennes agrégées d'une variable continue y sur des intervalles d'une variable continue x.
# La longueur des intervalles doit être spécifiée par l'utilisateur.

binedmeaner <- function(data, x, y, bin) {
  df <- rename(data, x = x, y = y)
  df$bvar <- cut(df$x, seq(min(df$x, na.rm = TRUE), round(max(df$x, na.rm = TRUE)), bin))

  df$low <- as.numeric(sub("\\((.+),.*", "\\1", df$bvar))
  df$up <- as.numeric(sub("[^,]*,([^]]*)\\]", "\\1", df$bvar))
  df$mp <- (df$up + df$low) / 2

  df2 <- summarise(group_by(df, bvar, mp), "my" = mean(y, na.rm = TRUE))

  df2 <- left_join(df, df2,  by = "bvar") %>% rename(mp = mp.x) %>% select(- mp.y)
  df2$my[duplicated(df2$my)] <- NA

  df2
}
```

cut2num()

```
# Obtient les midpoints d'une variable continue lorsqu'on la découpe en intervalles.
# Les intervalles sont de longueurs égales.

cut2num <- function(obj, bin){
    labs <- levels(cut(obj, seq(min(obj), max(obj), bin)))
    d <- data.frame(int = as.character(labs),
                    lower = as.numeric( sub("\\((.+),.*", "\\1", labs) ),
                    upper = as.numeric( sub("[^,]*,([^]]*)\\]", "\\1", labs) ))
    d$midpoints <- rowMeans(d[, -1])
    d[, c(1, 4)] %>% as_tibble %>% mutate_if(is.factor, as.character)
  }
```

cut3()

```
# Version augmentée de cut2num.

cut3 <- function(obj, cut, bin) {
    df <- data.frame(var = obj)
    df$id <- rownames(df)

    df1 <- df[df$var < cut, ]
    df1$bvar <- cut(df1$var, seq(max(df1$var), min(df1$var), - bin))

    dfr <- df[df$var >= cut, ]
    dfr$bvar <- cut(dfr$var, seq(min(dfr$var), max(dfr$var), bin))

    df2 <- rbind(df1, dfr)
    df <- left_join(df, df2[, c("bvar", "id")], by = "id")

    df$low <- as.numeric(sub("\\((.+),.*", "\\1", df$bvar))
    df$up <- as.numeric(sub("[^,]*,([^]]*)\\]", "\\1", df$bvar))
    df$mp <- (df$up + df$low) / 2

    df
  }
```

DCdensity2()

```
# DCdensity(), mais avec extraction du tableau afin de personnaliser le graphique.

DCdensity2 <- function(runvar,
           cutpoint,
           bin = NULL,
           bw = NULL,
           verbose = FALSE,
           ext.out = FALSE,
           htest = FALSE) {
    runvar <- runvar[complete.cases(runvar)]
    #Grab some summary vars
```

```
  rn <- length(runvar)
  rsd <- sd(runvar)
  rmin <- min(runvar)
  rmax <- max(runvar)
  if (missing(cutpoint)) {
    if (verbose)
      cat("Assuming cutpoint of zero.\n")
    cutpoint <- 0
  }

  if (cutpoint <= rmin | cutpoint >= rmax) {
    stop("Cutpoint must lie within range of runvar")
  }

  if (is.null(bin)) {
    bin <- 2 * rsd * rn ^ (-1 / 2)
    if (verbose)
      cat("Using calculated bin size: ", sprintf("%.3f", bin), "\n")
  }

  l <-
    floor((rmin - cutpoint) / bin) * bin + bin / 2 + cutpoint #Midpoint of lowest bin
  r <-
    floor((rmax - cutpoint) / bin) * bin + bin / 2 + cutpoint #Midpoint of highest bin
  lc <- cutpoint - (bin / 2) #Midpoint of bin just left of breakpoint
  rc <- cutpoint + (bin / 2) #Midpoint of bin just right of breakpoint
  j <- floor((rmax - rmin) / bin) + 2

  binnum <-
    round((((
      floor((runvar - cutpoint) / bin) * bin + bin / 2 + cutpoint
    ) - l) / bin) + 1)

  cellval <- rep(0, j)
  for (i in seq(1, rn)) {
    cnum <- binnum[i]
    cellval[cnum] <- cellval[cnum] + 1
  }
  cellval <- (cellval / rn) / bin

  cellmp <- seq(from = 1, to = j, by = 1)
  cellmp <-
    floor(((l + (cellmp - 1) * bin) - cutpoint) / bin) * bin + bin / 2 + cutpoint

  #If no bandwidth is given, calc it
  if (is.null(bw)) {
    #bin number just left of breakpoint
    leftofc <-
      round((((
        floor((lc - cutpoint) / bin) * bin + bin / 2 + cutpoint
      ) - l) / bin) + 1)
    #bin number just right of breakpoint
    rightofc <-
```

```
      round((((
        floor((rc - cutpoint) / bin) * bin + bin / 2 + cutpoint
      ) - l) / bin) + 1)
    if (rightofc - leftofc != 1) {
      stop("Error occurred in bandwidth calculation")
    }
    cellmpleft <- cellmp[1:leftofc]
    cellmpright <- cellmp[rightofc:j]

    #Estimate 4th order polynomial to the left
    P.lm <- lm(cellval ~ poly(cellmp, degree = 4, raw = T),
               subset = cellmp < cutpoint)
    mse4 <- summary(P.lm)$sigma ^ 2
    lcoef <- coef(P.lm)
    fppleft <- 2 * lcoef[3] +
      6 * lcoef[4] * cellmpleft +
      12 * lcoef[5] * cellmpleft * cellmpleft
    hleft <-
      3.348 * (mse4 * (cutpoint - l) / sum(fppleft * fppleft)) ^ (1 / 5)

    #And to the right
    P.lm <- lm(cellval ~ poly(cellmp, degree = 4, raw = T),
               subset = cellmp >= cutpoint)
    mse4 <- summary(P.lm)$sigma ^ 2
    rcoef <- coef(P.lm)
    fppright <- 2 * rcoef[3] +
      6 * rcoef[4] * cellmpright +
      12 * rcoef[5] * cellmpright * cellmpright
    hright <-
      3.348 * (mse4 * (r - cutpoint) / sum(fppright * fppright)) ^ (1 / 5)


    bw = .5 * (hleft + hright)
    if (verbose)
      cat("Using calculated bandwidth: ", sprintf("%.3f", bw), "\n")
  }
  if (sum(runvar > cutpoint - bw & runvar < cutpoint) == 0 |
      sum(runvar < cutpoint + bw & runvar >= cutpoint) == 0)
    stop("Insufficient data within the bandwidth.")

    #estimate density to either side of the cutpoint using a triangular kernel
    d.l <-
      data.frame(
        cellmp = cellmp[cellmp < cutpoint],
        cellval = cellval[cellmp < cutpoint],
        dist = NA,
        est = NA,
        lwr = NA,
        upr = NA
      )
    pmin <- cutpoint - 2 * rsd
    pmax <- cutpoint + 2 * rsd
    for (i in 1:nrow(d.l)) {
```

```
    d.l$dist <- d.l$cellmp - d.l[i, "cellmp"]
    w <- kernelwts(d.l$dist, 0, bw, kernel = "triangular")
    newd <- data.frame(dist = 0)
    pred <-
      predict(
        lm(cellval ~ dist, weights = w, data = d.l),
        interval = "confidence",
        newdata = newd
      )
    d.l$est[i] <- pred[1]
    d.l$lwr[i] <- pred[2]
    d.l$upr[i] <- pred[3]
  }
  d.r <-
    data.frame(
      cellmp = cellmp[cellmp >= cutpoint],
      cellval = cellval[cellmp >= cutpoint],
      dist = NA,
      est = NA,
      lwr = NA,
      upr = NA
    )
  for (i in 1:nrow(d.r)) {
    d.r$dist <- d.r$cellmp - d.r[i, "cellmp"]
    w <- kernelwts(d.r$dist, 0, bw, kernel = "triangular")
    newd <- data.frame(dist = 0)
    pred <-
      predict(
        lm(cellval ~ dist, weights = w, data = d.r),
        interval = "confidence",
        newdata = newd
      )
    d.r$est[i] <- pred[1]
    d.r$lwr[i] <- pred[2]
    d.r$upr[i] <- pred[3]
  }

cmp <- cellmp
cval <- cellval
padzeros <- ceiling(bw / bin)
jp <- j + 2 * padzeros
if (padzeros >= 1) {
  cval <- c(rep(0, padzeros),
            cellval,
            rep(0, padzeros))
  cmp <- c(seq(l - padzeros * bin, l - bin, bin),
           cellmp,
           seq(r + bin, r + padzeros * bin, bin))
}

#Estimate to the left
dist <- cmp - cutpoint
w <- 1 - abs(dist / bw)
```

```
  w <- ifelse(w > 0, w * (cmp < cutpoint), 0)
  w <- (w / sum(w)) * jp
  fhatl <-
    predict(lm(cval ~ dist, weights = w), newdata = data.frame(dist = 0))[[1]]

  #Estimate to the right
  w <- 1 - abs(dist / bw)
  w <- ifelse(w > 0, w * (cmp >= cutpoint), 0)
  w <- (w / sum(w)) * jp
  fhatr <-
    predict(lm(cval ~ dist, weights = w), newdata = data.frame(dist = 0))[[1]]

  #Calculate and display dicontinuity estimate
  thetahat <- log(fhatr) - log(fhatl)
  sethetahat <-
    sqrt((1 / (rn * bw)) * (24 / 5) * ((1 / fhatr) + (1 / fhatl)))
  z <- thetahat / sethetahat
  p <- 2 * pnorm(abs(z), lower.tail = FALSE)

  if (verbose) {
    cat(
      "Log difference in heights is ",
      sprintf("%.3f", thetahat),
      " with SE ",
      sprintf("%.3f", sethetahat),
      "\n"
    )
    cat("  this gives a z-stat of ", sprintf("%.3f", z), "\n")
    cat("  and a p value of ", sprintf("%.3f", p), "\n")
  }
  if (ext.out)
    return(
      list(
        theta = thetahat,
        se = sethetahat,
        z = z,
        p = p,
        binsize = bin,
        bw = bw,
        cutpoint = cutpoint,
        data = data.frame(cellmp, cellval),
        "d.l" = d.l,
        "d.r" = d.r
      )
    )
  else if (htest) {
    # Return an htest object, for compatibility with base R test output.
    structure(
      list(
        statistic   = c(`z` = z),
        p.value     = p,
        method      = "McCrary (2008) sorting test",
        parameter   = c(
```

```
            `binwidth`   = bin,
            `bandwidth`  = bw,
            `cutpoint`   = cutpoint
          ),
          alternative = "no apparent sorting"
        ),
        class = "htest"
      )
    }
    else
      return(p)
}
```

ext_adjrsq()

```
# Extrait le R^2 depuis un modèle de type lm.
# Prend en compte la méthode spécifique d'extraction de plm.

ext_adjrsq <- function (x) {
  if ("plm" %in% class(x)) {
    summary(x)$r.squared[2]
  } else {
    summary(x)$adj.r.squared
  }
}
```

ext_ct()

```
# Extracteur de la matrice de type coef summary.lm à partir, pour écart-types estimés par wild bootstra
# Ne marche que sur des objets outputs de wboot_altFIN et wboot_clu_altFIN.

ext_ct <- function(sdcf, obj) as.data.frame(sdcf[[which(names(sdcf) == obj)]][[1]])
```

frencherstats()

```
# Simple traducteur FR de libellés de stats desc issus de summary().

frencherstats <- function (x) {str_replace(x, "1st Qu.", "1er Qu.") %>%
    str_replace("Median", "Médiane") %>% str_replace("Mean", "Moyenne") %>%
    str_replace("sd", "Écart-type") %>% str_replace("3rd Qu.", "3ème Qu.") %>%
    str_replace("NA's", "Manquantes") %>%
    str_replace_na("Manquante")}
```

fsignif()

```
# ne marche qu'avec les objets plm
fsignif <- function (obj) {
  f <- summary(obj)$fstatistic
  paste(f$statistic %>% round(2), "$^{",
```

```
        signif(f$p.value), "}$", sep = "")
}
```

hrestr()

```
# Très spécifique aux données RDD de la thèse.
# Restreint les données par rapport à une fenêtre dist.

hrestr <- function(data, h) {
    filter(data, dist >= - h &
             dist <= h)
  }
```

kstester()

```
# Se base sur ks.test()
# Effectue ks.test() sur les classes, dans une école

kstester <- function (data, gsup, ginf, var, mvar, r,
                      detailed = FALSE) {
  # les ginf simulés aléatoirement
  sims <- lapply(sort(unique(data[[gsup]])), function (x)
    lapply(1:r, function (y) {
      set.seed(y)
      filter(data, eval(parse(text = gsup)) == x) %>%
        transmute(sim = sample(eval(parse(text = ginf))))
    }) %>%
      do.call(what = cbind)) %>%
    do.call(what = rbind)



  mean_var_ginf_sim <- lapply(1:r, function (x) {
    select(arrange(data, eval(parse(text = gsup))),
           gsup, ginf, var) %>%
      cbind(matrix(sims[, x], ncol = 1) %>%
              set_colnames("sim")) %>%
      group_by("gsup2" = eval(parse(text = gsup)),
               sim) %>%
      mutate(mean_var_ginf_sim = mean(eval(parse(text = var)), na.rm = TRUE)) %>%
      ungroup %>%
      group_by(gsup2,
               sim,
               mean_var_ginf_sim) %>%
      summarise %>%
      ungroup %>%
      select(gsup2, mean_var_ginf_sim)
  })

  kslist <- lapply(sort(unique(data[[gsup]])), function (x)
    ks.test(filter(data, eval(parse(text = gsup)) == x) %>%
```

```
              group_by(eval(parse(text = ginf)),
                       "mvar2" = eval(parse(text = mvar))) %>%
              summarise %>%
              pull(mvar2),

            unlist(lapply(mean_var_ginf_sim, function (y)
              filter(y, gsup2 == x) %>%
                pull(mean_var_ginf_sim)))
            ))

  ksdata <- data.frame("gsup" = sort(unique(data[[gsup]])),
                       "D" = unlist(lapply(kslist, function (x) x$statistic)),
                       "pval" = unlist(lapply(kslist, function (x) x$p.value)))

  if (detailed == FALSE)
         ksdata else
         listN(sims, mean_var_ginf_sim, kslist, ksdata)

}
```

listN()

```
# Assigne des noms aux éléments d'une liste tels qu'ils sont renseignés.

listN <- function(...){
    anonList <- list(...)
    names(anonList) <- as.character(substitute(list(...)))[-1]
    anonList
}
```

mimicer()

```
# Base pour effectuer des tableaux de régression personnalisées.
# La personnalisation concerne uniquement la mise en page pdf. Aucune modification sur les coef, sd ou

mimicer <- function (ctlist) {

  mimiclist <- lapply(lapply(ctlist, psignifsev3), nameadjuster)

  mimictab <- plyr::join_all(
    list(list(as.data.frame(Reduce(union, lapply(mimiclist, function (x) x[["covars"]])))) %>%
           set_colnames("covars") %>%
           as.data.frame),

         mimiclist
         ) %>% flatten,
    by = "covars", type = "left"
  ) %>%
    apply(2, function (x) psignifse_corrector(x) %>%
            str_replace("_", " "))
```

```
  mimictab

}
```

normalizer()

```
normalizer <- function (data, group, var) {
    group <- enquo(group)
    var <- enquo(var)
    left_join(data, group_by(data, !! group) %>%
                  summarise(varmean = mean(!! var, na.rm = TRUE),
                            varsd = sd(!! var, na.rm = TRUE)),
              by = as.character(group)[2]) %>%
        transmute(varnorm = (!! var - varmean) / varsd) %>%
        pull(varnorm) }
```

normalizer2

```
# normaliseur avec vecteur uniquement comme input
normalizer2 <- function (x) (x - mean(x, na.rm = TRUE)) / sd(x, na.rm = TRUE)
```

nameadjuster()

```
# Composante spécifique à mimicer.

nameadjuster <- function (x) {
  cbind(names(x), unname(x)) %>% as.data.frame %>%
    set_colnames(c("covars", "coefse"))
}
```

not_one_one_finder()

```
# Trouve, s'il y en a, les correspondances non-uniques entre les valeurs de deux variables.
# Utile pour trouver les anomalies dans les données.

not_one_one_finder <- function(data, var1, var2) {
  var1 <- enquo(var1)
  var2 <- enquo(var2)

 res <- filter(counter(summarise(group_by(filter(data, ! is.na(!! var1) &
                                                   ! is.na(!! var2)),
                                          !! var1, !! var2)), !! var1),
       n > 1)
 if (dim(res)[1] == 0) print("var1 and var2 form one-one matches") else
   as.character(pull(res, !! var1))
}
```

Les psignif()

```
# Nécessité d'épuration considérable ici. On pourrait idéalement ne garder que psignifsev3.
# Laissé pour plus tard.

pse_cfv2 <- function(ct,
                indvar = 1, indse = 1) {
  paste("(", round(ct[indvar, indse], 3), ")", sep = "")
}


psev2 <- function(ct,
                indvar = 1, indse = 2) {
  paste("(", round(ct[indvar, indse], 3), ")", sep = "")
}


psignif_cfv2 <- function(ct, obj,
                         indvar = 1, indpval = 3) {
  coef <- coef(get(obj))[indvar]
  ifelse(ct[indvar, indpval] <= .01, paste(round(coef, 3), "$^{***}$",
                                           sep = ""),
         if_else(ct[indvar, indpval] <= .05, paste(round(coef, 3), "$^{**}$",
                                           sep = ""),
                 if_else(ct[indvar, indpval] <= .1, paste(round(coef, 3), "$^{*}$",
                                           sep = ""),
                         paste(round(coef, 3),
                               sep = ""))))
}


psignifse_corrector <- function (x) {
  lapply(x, function (y) {
    str_replace(y, "-", "$-$")
  } )%>%
      data.frame %>%
      as.matrix %>% as.vector
}


psignifsev2 <- function (x) {
  x1 <-list()
  for (i in 1:length(x)) {
  nobs <- dim(model.frame(get((substring(names(x)[[i]], 4)))))[1]
  x1[[i]] <- c(psignifv2(x[[i]]), psev2(x[[i]]), paste("N = ", nobs, sep = ""))
  }
  unlist(x1)
}


psignifsev3 <- function (ct, indcoef = 1, indse = 2, indpval = 4) {
  apply(ct, 1, function (x) ifelse(x[indpval] <= .01,
                                   lapply(x, function (y) c(paste(round(x[indcoef], 3),
                                                                  "$^{***}$", sep = ""),
```

```
                                                        paste("(", round(x[indse], 3), ")", sep = "
                                  ifelse(x[indpval] <= .05,
                                         lapply(x, function (y) c(paste(round(x[indcoef],
                                                                              3), "$^{**}$", sep = "")
                                                                  paste("(", round(x[2], 3), ")", sep =
                                                                        "")),
                                         ifelse(x[indpval] <= .1, lapply(x, function (y)
                                           c(paste(round(x[indcoef], 3), "$^{*}$", sep = ""),
                                             paste("(", round(x[2], 3), ")", sep = ""))),
                                                lapply(x, function (y) c(paste(round(x[indcoef], 3), se
                                                                               ""),
                                                                         paste("(", round(x[indse], 3)
                                                                               sep = "")))))))) %>%
    unlist %>%
    setNames(rownames(ct) %>% sapply(function (x) c(x, paste("sd", x))) %>% as.vector)

}


psignifv2 <- function(ct,
                      indvar = 1, indcoef = 1, indpval = 4) {
  ifelse(ct[indvar, indpval] <= .01, paste(round(ct[indvar, indcoef], 3), "$^{***}$",
                                           sep = ""),
         ifelse(ct[indvar, indpval] <= .05, paste(round(ct[indvar, indcoef], 3), "$^{**}$",
                                           sep = ""),
                ifelse(ct[indvar, indpval] <= .1, paste(round(ct[indvar, indcoef], 3), "$^{*}$",
                                           sep = ""),
                       paste(round(ct[indvar, indcoef], 3),
                             sep = ""))))
                  }
```

rdsenser()

```
# Fonction d'analyse de sensibilité RDD selon la fenêtre.

rdsenser <- function (obj, hopt, maxdev, by, data, inference = FALSE) {
  if(inference == TRUE) {

      hseq <- seq(hopt, hopt + maxdev, by)
      rselist <- lapply(hseq, function (x) rsearellano(update(obj, data = hrestr(data, x)))) # une list

      sens <- data.frame("dev" = rep(NA, length(hseq)),
                         "est" = rep(NA, length(hseq)),
                         "sd" = rep(NA, length(hseq)),
                         "up" = rep(NA, length(hseq)),
                         "low" = rep(NA, length(hseq)))



        sens[["dev"]] <- hseq - hopt
        sens[["est"]] <- sapply(hseq, function (x) round(coef(update(obj, data = hrestr(data, x)))[2], 3
        sens[["sd"]] <- sapply(rselist, function (x) round(x[, 2][2], 3))
```

```
        sens[["up"]] <- round(sens[["est"]] + 1.96*sens[["sd"]], 3)
        sens[["low"]] <- round(sens[["est"]] - 1.96*sens[["sd"]], 3)

    } else {
      hseq <- seq(hopt, hopt + maxdev, by)

      sens <- data.frame("dev" = rep(NA, length(hseq)),
                         "est" = rep(NA, length(hseq)))

      sens[["dev"]] <- hseq - hopt
      sens[["est"]] <- sapply(hseq, function (x) round(coef(update(obj, data = hrestr(data, x)))[2], 3)
    }
    sens
}
```

rsearellano()

```
# Donne les écart-types d'Arellano (1987).
# Aucune création supplémentaire par le doctorant. Juste pour compresser l'écriture.

rsearellano <- function (obj) {
  coeftest(obj, vcov = vcovHC(obj))
}
```

rsewhite()

```
# Donne les écart-types de White (1980).
# Aucune création supplémentaire par le doctorant. Juste pour compresser l'écriture.

rsewhite <- function(obj) {
  coeftest(obj, vcov = vcovHC(obj, method = "white1", type = "HC0"))
}
```

str_betw()

```
# Compression d'un code de recherche de caractères entre un pattern donné.
str_betw <- function(char, pattern) {
  str_remove_all(str_extract(char, paste(pattern, ".*",
                                         pattern, sep = "")),
                 pattern)
}
```

sumer()

```
# Très spécifique à la correction des items dans cm2.
# Aucun traffic. L'auteur se met à disposition pour une vérification.

sumer <- function(data, i) {
```

```
  apply(select(data, paste("i", i, sep = "")), 1, sum)
}
```

Les autres versions de summary()

```
# Versions alternatives de summary.
# Pour extraire plus d'informations

summary2 <- function (x) {

  summ <- summary(x) %>% round(2)
  summ2 <- rep(NA, length(summ) + 1)
  summ2[ -5 ] <- summ
  summ2[ 5 ] <- sd(x, na.rm = TRUE) %>% round(2)
  summ2[8] <- paste(sum(is.na(x)), "(", round(sum(is.na(x)) / length(x), 2),
                    ")", sep = "")

  names <- names(summ)
  names2 <- rep(NA, length(names) + 1)
  names2[ -5 ] <- names
  names2[ 5 ] <- "sd"
  names2[8] <- "NA's"

  names(summ2) <- names2

  summ2[c(1, 4, 5, 7, 8)]
}

summary3 <- function (x) {

  summ <- summary(x) %>% round(2)
  summ2 <- rep(NA, length(summ) + 1)
  summ2[ -5 ] <- summ
  summ2[ 5 ] <- sd(x, na.rm = TRUE) %>% round(2)
  summ2[8] <- paste(sum(is.na(x)), "(", round(sum(is.na(x)) / length(x), 2),
                    ")", sep = "")

  names <- names(summ)
  names2 <- rep(NA, length(names) + 1)
  names2[ -5 ] <- names
  names2[ 5 ] <- "sd"
  names2[8] <- "NA's"

  names(summ2) <- names2

  summ2[c(1:3, 6:8)]
}

summary4 <- function (x) {

  summ <- summary(x) %>% round(2)
  summ2 <- rep(NA, length(summ) + 1)
```

```
  summ2[ -5 ] <- summ
  summ2[ 5 ] <- sd(x, na.rm = TRUE) %>% round(2)
  summ2[8] <- paste(sum(is.na(x)), "(", round(sum(is.na(x)) / length(x), 2),
                    ")", sep = "")

  names <- names(summ)
  names2 <- rep(NA, length(names) + 1)
  names2[ -5 ] <- names
  names2[ 5 ] <- "sd"
  names2[8] <- "NA's"

  names(summ2) <- names2

  summ2
}
```

pmeaner()

```
pmeaner <- function (data, group, var) {
    group <- enquo(group)
    var <- enquo(var)
    left_join(data, group_by(data, !! group) %>%
                summarise(groupsize = n(),
                          varsum = sum(!! var, na.rm = TRUE)),
              by = as.character(group)[2]) %>%
      transmute(pmeanvar = (varsum - !! var) / (groupsize - 1)) %>%
      pull(pmeanvar)
  }
```

signif()

```
signif <- function (x)
  ifelse(x <= .01, "***",
         ifelse(x <= .05, "**",
                ifelse(x <= .1, "*", "")))
```

table2()

```
# Version de table avec l'option useNA = "always" toujours

table2 <- function (...) table(..., useNA = "always")
```

tableproper()

```
# Un table à deux dimension mais avec les proportions et les valeurs manquantes en plus.

tableproper <- function (...) {
  table <- table2(...)
  tableprop <- apply(table, 2, function (x) paste(
```

```
    x, " (", as.character(round(x/sum(x, na.rm  = TRUE), 2)), ")",
    sep = ""
    )) %>% set_rownames(rownames(table))
  tableprop
}
```

tablepropernona()

```
# table proper mais en ne considérant pas les NA

tablepropernona <- function (...) {
  table <- table(...)
  tableprop <- apply(table, 2, function (x) paste(
    x, " (", as.character(round(x/sum(x, na.rm  = TRUE), 2)), ")",
    sep = ""
    )) %>% set_rownames(rownames(table))
  tableprop
}
```

unaccent()

```
# Enlève tout accent dans les chaînes de caratères.

unaccent <- function(text) {
  text <- gsub("['`^~\"]", " ", text)
  text <- iconv(text, to="ASCII//TRANSLIT//IGNORE")
  text <- gsub("['`^~\"]", "", text)
  return(text)
}
```

wboot()

```
# Wild bootstrap, à un objet de type lm donné.
# Créé par le doctorant, optimisé par le directeur de thèse.

wboot <- function(obj, B, seed = 1, form = formula(obj),
                  detailed = TRUE){
  set.seed(seed)
  N <- nobs(obj)
  e <- resid(obj, model = "pooling")
  bresid <- matrix(sample(c(-1, 1), size = N * B, replace = TRUE), nrow = N)
  yfit <- fitted(obj, model = "pooling")
  X <- model.matrix(obj, model = "within")
  bcoef <- apply(bresid, 2, function(x){
    set.seed(seed)
    bystar <- Within(yfit + x)
    coef(lm.fit(X, bystar))
  })[names(coef(obj)), ]
  bsigma <- apply(bcoef, 1, sd)
  btvalue <- unname(coef(obj) / bsigma)
```

```
  bpvalue <- 2 * pnorm(abs(btvalue), lower.tail = FALSE)
  quants <- t(apply(bcoef, 1, quantile, c(0.025, 0.975)))
  result <- cbind(coef(obj) %>% matrix(ncol = 1) %>% set_colnames("coef"),
                  'sigma' = bsigma , 't-stat' = btvalue, 'p-value' = bpvalue, quants)
  rownames(result) <- names(coef(obj))
  l <- list(result, bcoef)
  names(l) <- c("result", "bcoef")
  if (detailed) l else result
}
```

wboot_clu()

```
# Wild bootstrap avec cluster déjà indiqué dans le modèle.

wboot_clu <- function(obj, B, seed = 1, form = formula(obj),
                      detailed = TRUE) {
  set.seed(seed)
  ind <- as.character(index(obj)[[1]]) # la différence ici
  N <- nobs(obj)
  C <- length(unique(ind))
  e <- resid(obj, model = "pooling")
  mp1 <- matrix(sample(c(-1, 1), size = C * B, replace = TRUE), nrow = C)
  rownames(mp1) <- unique(ind)
  mp1 <- mp1[ind, ]
  bresid <- mp1 * e
  yfit <- fitted(obj, model = "pooling")
  X <- model.matrix(obj, model = "within")
  bcoef <- apply(bresid, 2, function(x){
    bystar <- Within(yfit + x)
    coef(lm.fit(X, bystar))
  })[names(coef(obj)), ]
  bsigma <- apply(bcoef, 1, sd)
  btvalue <- unname(coef(obj) / bsigma)
  bpvalue <- 2 * pnorm(abs(btvalue), lower.tail = FALSE)
  quants <- t(apply(bcoef, 1, quantile, c(0.025, 0.975)))
  result <- cbind(coef(obj) %>% matrix(ncol = 1) %>% set_colnames("coef"),
                  'sigma' = bsigma , 't-stat' = btvalue, 'p-value' = bpvalue, quants)
  rownames(result) <- names(coef(obj))
  l <- list(result, bcoef)
  names(l) <- c("result", "bcoef")
  if (detailed) l else result
}
```

Sauvegarde

```
save(list = ls(), file = "fonctions.rda", version = 2)
```