```
# Imputations methods used in the article
# the given functions estimate the values of the variables for a given time
# all functions take as input:
# - df: a data.frame in the long format containing the register data
# - time: a string: the name of the column containing the time elapsed since tretament onset. must be numeric
# - id : a string: the names of the column containing the patient-treatment identifiers
# - variables. a vector of strings, containing the names of the column with the data to be imputed.
# - t0: a numerical value: the imputation time, in the same unit as the time column

# check if the packages are installed, and install it if not
list.of.packages <- c("data.table","mice","zoo","lme4")
new.packages <- list.of.packages[!(list.of.packages %in% installed.packages()[,"Package"])]
if(length(new.packages)) install.packages(new.packages)

# load the libraries
library(data.table)
library(mice)
library(zoo)
library(lme4)

# function to perform basic checks on the data provided
basic_checks = function(df,time,id,variables,t0){

  # check if arguments are strings
  for(var  in  c(variables,time,id)){if(!is.character(var))stop(paste0(deparse(substitute(var)),"  argument is not a character"))}
  # check if time, id and variables are column names
  for(var in c(variables,time,id)){if(!var %in% names(df))stop(paste0(deparse(substitute(var))," is not a column name of df"))}
  # check if times are numeric
  if(!is.numeric(df[[time]]))stop('time column is not numeric')
  if(!is.numeric(t0))stop('t0 is not numeric')

  # create a temporary data.table
  temp_df <- setDT(df)[,.SD,.SDcols = c(id,time,variables)]
  # check for duplicated time measures
  if(length(unique(temp_df[,.N,by = c(time,id)]$N)) != 1 )stop('there are duplicated time measures for the same id')
  # check for the value of t0
  if(t0 < min(temp_df[[time]],na.rm = T) | t0 > max(df[[time]],na.rm = T))warning("t0 is out of the range of values present in the time column")

  setnames(temp_df,c(time,id),c("time","id"))
  temp_df[,difft := abs(time - t0)] # to get the closest value to t0
  temp_df[,difftposneg := (time - t0)] # to have time direction
  temp_df[,time2 := time^2/10] # variable for regressions
  temp_df[,time3 := time^3/100] # variable for regressions

  return(temp_df)
}
```

```
LOCF = function(df,time,id,variables,t0){
  temp_df <- basic_checks(df,time,id,variables,t0)
  # order(difft) classify the data frame in order from closest to t0 to furthest
  plouf <- lapply(variables,function(x){
    temp_df[order(difft)][difftposneg < 0 & !is.na(get(x)),setNames(list( get(x)[1]),
paste0(x,"_LOCF")),by = id]
  })
  return(Reduce(function(x,y){merge(x,y,all = T,by = "id")},plouf))
}


NAO = function(df,time,id,variables,t0){
  temp_df <- basic_checks(df,time,id,variables,t0)
  plouf <- lapply(variables,function(x){
    temp_df[order(difft)][!is.na(get(x)),setNames(list( get(x)[1]), paste0(x,"_NAO")),by = id]
  })
  return(Reduce(function(x,y){merge(x,y,all = T,by = "id")},plouf))
}


LE = function(df,time,id,variables,t0){
  temp_df <- basic_checks(df,time,id,variables,t0)
  plouf <-  lapply(variables,function(x){
      temp_df[order(difft)][!is.na(get(x)),
                  setNames(list( (get(x)[1]-get(x)[2] )/(time[1]-time[2])*(t0 - time[1]) + get(x)[1] ),
paste0(x,"_LE")),by = id]
    })
  return(Reduce(function(x,y){merge(x,y,all = T,by = "id")},plouf))
}

LFE = function(df,time,id,variables,t0){
    temp_df <- basic_checks(df,time,id,variables,t0)
    plouf <-  lapply(variables,function(x){
      temp_df[order(difft)][difftposneg < 0 & !is.na(get(x)),
                  setNames(list( (get(x)[1]-get(x)[2] )/(time[1]-time[2])*(t0 - time[1]) + get(x)[1] ),
paste0(x,"_LFE")),by = id]
    })
    output <- Reduce(function(x,y){merge(x,y,all = T,by = "id")},plouf)
    NAresults <- data.table(id = setdiff(unique(temp_df$id),output$id))
    NAresults[,paste0(variables,"_LFE") := NA]
    return(rbind(NAresults,output))
}


PE = function(df,time,id,variables,t0){
  temp_df <- basic_checks(df,time,id,variables,t0)

  plouf <-  lapply(variables,function(x){
    formula <- c(as.formula(paste0(x," ~ 1")),
            as.formula(paste0(x," ~ time")),
            as.formula(paste0(x," ~ time + time2")),
         as.formula(paste0(x," ~ time + time2 + time3")))
  impute <- data.frame(time = t0,time2 = t0^2/10, time3 = t0^3/100)
  temp_df[!is.na(get(x)), setNames(list(
      if(.N <= 4){predict(lm(formula[.N][[1]],data = .SD),newdata = impute)} else
      if(.N > 4) {predict(lm(formula[4][[1]],data = .SD),newdata = impute)}
      ),paste0(x,"_PE")),by = id  ]
 })
 output <- Reduce(function(x,y){merge(x,y,all = T,by = "id")},plouf)
 NAresults <- data.table(id = setdiff(unique(temp_df$id),output$id))
 NAresults[,paste0(variables,"_PE") := NA]
 return(rbind(NAresults,output))
}

LME3 = function(df,time,id,variables,t0){
 temp_df <- basic_checks(df,time,id,variables,t0)
 plouf <-  lapply(variables,function(x){
  imp <- predict(lmer(paste0(x,"~ time + time2 + time3 + ( time + time2  + time3| id)"),
     data=temp_df,
     control = lmerControl(calc.derivs = FALSE,optimizer = "nloptwrap")),
     newdata = temp_df[,.(time = t0, time2 = t0^2/10, time3 = t0^3/100),by = id])
  data.frame(  setNames(list( unique(temp_df$id), imp ),c("id",paste0(x,"_LME3")) ) )

 })
 return(Reduce(function(x,y){merge(x,y,all = T,by = "id")},plouf))
}


# to test
df <- data.table(id = rep(LETTERS[1:5],each = 10))
df[,time := sample(seq(0,30,0.1),.N),by = id ] # create time variable
df[,c("var1","var2") := lapply(1:2,function(x){sample(1:40,.N,replace = T)}),by = id] # create variables to
impute

for(var in c("var1","var2")){df[sample(1:.N,10),c(var) := NA]} # create missing data

# exemple for each method
test <- NAO(df,"time","id",c("var1","var2"),6)
test2 <- LOCF(df,"time","id",c("var1","var2"),6)
test3 <- LFE(df,"time","id",c("var1","var2"),6)
test4 <- PE(df,"time","id",c("var1","var2"),6)
test5 <- LME3(df,"time","id",c("var1","var2"),6)
```