# PriorT2DRisk package

Using the codes highlighted in grey on pages 4-9, run the package.

## Usage

### One-off Calculator:

Each function can be used as either a calculator or vectoriser. When used as a one-off calculator, a dataset is not required. This can be handy when checking the risk estimate for an individual person or for quickly seeing the effect of changing a single parameter value.

PriorT2DRisk(sex="M", age=35, eth=2, nzdep=5, smoker=1, af=1, familyhx=1, lld=0,

athrombi=0, bpl=0, oral=0, insulin=0, sbp=120, tchdl=3.3, bmi=27, egfr=78, acr=1, hba1c=48, years=1)

# [1] 0.0483

All parameter values can be numeric or encoded. Values for some parameters such as BMI can be NA if the value is unknown. See values in R documentation. Binary variables can take TRUE/FALSE boolean values.

round(PriorT2DRisk(sex=0, age=75, eth=PI, nzdep=3, smoker=0, af=F, familyhx=0, lld=Y,athrombi=0, bpl=F, oral=T, insulin=0, sbp=130, tchdl=4, bmi=31, egfr=92, acr=1.4, hba1c=56, years=3)\*100, 0)

# [1] 5

R documentation:

library(PredictRiskScores)

??PriorT2DRisk

Binary parameters can take a variety of input values. For example, the value for TRUE can be T, Y, Yes, or 1, while the value for FALSE can be F, N, No, or 0. Quotations are not required and case sensitivity is ignored.

### Vectorise dataset:

When a dataset is supplied, a risk score is produced for each row of data. Each argument requires the variable name from the dataset. This can be handy when risk estimates are required for each row of data, or when datasets require vectorisation.

# Fake example

DF = data.frame(sex = c("F", "F", "F", "F", "F", "M", "M", "M"),

 age = c(55, 35, 75, 55, 75, 55, 55, 75),

 ethnic\_labels = c(1, 3, 2, 1, 4, 2, 2, 4),

 nzdep\_quintiles = c(1, 5, 5, 5, 5, 1, 5, 5),

 smoker\_status = c(0, 0, 0, 0, 1, 0, 0, 0),

 fam\_hx = c(0, 0, 0, 0, 0, 0, 0, 1),

 af = c(0, 0, 0, 0, 0, 0, 0, 1),

 lld = c(0, 0, 0, 0, 0, 0, 0, 0),

 bpl = c(0, 0, 0, 0, 0, 0, 0, 0),

 athrombi = c(0, 0, 0, 0, 0, 0, 0, 0),

 oral = c(0, 0, 0, 0, 0, 0, 0, 0),

 insulin = c(0, 0, 0, 0, 0, 0, 0, 0),

 systolic\_bp = c(132, 120, 138, 138, 132, 132, 138, 132),

 tchdl\_ratio = c(4, 4, 4.8, 4.8, 4, 4, 4.8, 4),

 years = c(5, 5, 7, 2, 5, 5, 9, 5),

 egfr = c(90, 90, 65, 25, 90, 90, 105, 90),

 acr = c(1.5, 1.5, 1, 1, 1.5, 1.5, 2, 1.5),

 bmi = c(34, 34, 34, 27, 34, 34, 34, 34),

 hba1c = c(62, 62, 62, 62, 62, 62, 62, 62))

dim(DF)

# 8 observations 19 variables

# As a vectoriser

PriorT2DRisk(dat=DF, sex=sex, age=age, eth=ethnic\_labels, nzdep=nzdep\_quintiles, smoker=smoker\_status, af=af, familyhx=fam\_hx, sbp=systolic\_bp, tchdl=tchdl\_ratio, bmi=bmi, years=years, egfr=egfr, acr=acr, hba1c=hba1c, oral=oral, insulin=insulin, bpl=bpl, lld=lld, athrombi=athrombi)

# [1] 0.0322 0.0123 0.1440 0.0746 0.1036 0.0536 0.0746 0.2224

When a dataset is supplied, the function returns a numeric vector of risk scores. Each element of the vector is positioned as per row index. As such, the resulting numeric vector can be assigned back to the dataset as a new variable.

DF$riskscores\_perc <- PriorT2DRisk(dat=DF, sex=sex, age=age, eth=ethnic\_labels, nzdep=nzdep\_quintiles, smoker=smoker\_status, af=af, familyhx=fam\_hx, sbp=systolic\_bp, tchdl=tchdl\_ratio, bmi=bmi, years=years, egfr=egfr, acr=acr, hba1c=hba1c, oral=oral, insulin=insulin, bpl=bpl, lld=lld, athrombi=athrombi)\*100

 DF$riskscores\_perc # %

 # [1] 3.22 1.23 14.40 7.46 10.36 5.36 7.46 22.24

 round(DF$riskscores\_perc, 0) # rounded %

 # [1] 3 1 14 7 10 5 7 22

#### Integration with data.table and dplyr

The suite of functions in this package can be integrated into both data.table and dplyr. For example, when datasets are extremely large, consider using data.table along with the := notation. In the example below, a new column called riskscore is created. The data.table syntax might seem confusing at first but it offers fast and efficient performance.

##### Data.table

install.packages("data.table")

library(data.table)

setDT(DF)

DF[, riskscore := PriorT2DRisk(dat=DF, sex=sex, age=age, eth=ethnic\_labels, nzdep=nzdep\_quintiles, smoker=smoker\_status, af=af, familyhx=fam\_hx, sbp=systolic\_bp, tchdl=tchdl\_ratio, bmi=bmi, years=years, egfr=egfr, acr=acr, hba1c=hba1c, oral=oral, insulin=insulin, bpl=bpl, lld=lld, athrombi=athrombi)]

# OR

DF[, riskscores\_perc := round(PriorT2DRisk(dat=DF, sex=sex, age=age, eth=ethnic\_labels, nzdep=nzdep\_quintiles, smoker=smoker\_status, af=af, familyhx=fam\_hx, sbp=systolic\_bp, tchdl=tchdl\_ratio, bmi=bmi, years=years, egfr=egfr, acr=acr, hba1c=hba1c, oral=oral, insulin=insulin, bpl=bpl, lld=lld, athrombi=athrombi)\*100, 1)] # %, 1 decimal place

##### Dplyr

Users who are more comfortable with dplyr will find that the functions works with mutate. In the example below, a new column called risk score is created.

install.packages("dplyr")

library(dplyr)

DF <- DF %>%

mutate(riskscore = PriorT2DRisk(dat=DF, sex=sex, age=age, eth=ethnic\_labels, nzdep=nzdep\_quintiles, smoker=smoker\_status, af=af, familyhx=fam\_hx, sbp=systolic\_bp, tchdl=tchdl\_ratio, bmi=bmi, years=years, egfr=egfr, acr=acr, hba1c=hba1c, oral=oral, insulin=insulin, bpl=bpl, lld=lld, athrombi=athrombi),

# %, 1 decimal place

riskscore\_perc = round(riskscore \* 100, 1)) %>%

as.data.frame()

“PriorT2DRisk”

# --- Please run the following codes highlighted in grey ---

**PriorT2DRisk** <- function(dat, sex, age, eth, nzdep, smoker, af, familyhx, lld, athrombi, bpl, oral, insulin, sbp, tchdl, bmi, egfr, acr, hba1c, years, ...){

 # Params

 demo.vars <- c("sex", "age", "eth", "nzdep")

 smk.vars <- c("smoker")

 bin.vars <- c("af", "familyhx", "lld", "athrombi", "bpl", "oral", "insulin")

 num.vars <- c("sbp", "tchdl", "bmi", "egfr", "acr", "hba1c", "years")

 # Calls

 call <- gsub("()", "", match.call()[1])

 is.table <- deparse(substitute(dat))!=""

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

 if(length(list(...)) == 0){

 dp <- 4

 allow.age <- TRUE

 allow.na <- TRUE

 } else {

 default <- setdiff(c("dp", "allow.age", "allow.na"), names(list(...)))

 if(length(default) %in% 1:2){

 lapply(default,

 function(x){

 if(x == "dp"){

 val <- 4

 } else if(x == "allow.na") {

 val <- TRUE

 } else {

 val <- TRUE

 }

 assign(x, val, envir = parent.frame(2))

 })

 }

 lapply(names(list(...)),

 function(x)

 assign(x, unlist(list(...)[x]),

 envir = parent.frame(2)))

 }

 # ParamCheck

 vars <- c(demo.vars, bin.vars, smk.vars, num.vars)

 ParamCheck(input, vars, call, is.table, allow.age, allow.na)

 # Values

 f.ind <- which(tolower(input$sex) %in% ok.female)

 m.ind <- which(tolower(input$sex) %in% ok.male)

 demo.vals <- list(age = input$age,

 maori = +(tolower(input$eth) %in% ok.maori),

 pacific = +(tolower(input$eth) %in% ok.pi),

 indian = +(tolower(input$eth) %in% ok.indian),

 asian = +(tolower(input$eth) %in% ok.asian),

 smoker = +(tolower(input$smoker) %in% ok.smoker),

 nzdep = input$nzdep)

 bin.vals <- sapply(bin.vars,

 function(x){

 +(tolower(input[[x]]) %in% ok.true)

 },

 USE.NAMES = TRUE,

 simplify = FALSE)

 num.vals <- sapply(num.vars,

 function(x){

 as.numeric(input[[x]])

 },

 USE.NAMES = TRUE,

 simplify = FALSE)

 values <- c(demo.vals, bin.vals, num.vals) # Order sensitive!

 # Adjustments

 if(allow.age){

 values$age[which(values$age < 30)] <- 30

 values$age[which(values$age > 79)] <- 80

 }

 if(!allow.na){

 vars <- c(smk.vars, bin.vars)

 values[vars] <- sapply(vars,

 function(x){

 input[[x]] <- if(is.name(input[[x]])){

 as.character(input[[x]])

 }

 replace(values[[x]],

 which(is.na(input[[x]])),

 NA)

 },

 USE.NAMES = TRUE,

 simplify = FALSE)

 }

 # Recentering

 values$age[f.ind] <- values$age[f.ind] - 53.598009

 values$age[m.ind] <- values$age[m.ind] - 53.738152

 values$nzdep[f.ind] <- values$nzdep[f.ind] - 3.657006

 values$nzdep[m.ind] <- values$nzdep[m.ind] - 3.410281

 values$sbp[f.ind] <- values$sbp[f.ind] - 131.380365

 values$sbp[m.ind] <- values$sbp[m.ind] - 131.662168

 values$tchdl[f.ind] <- values$tchdl[f.ind] - 3.970698

 values$tchdl[m.ind] <- values$tchdl[m.ind] - 4.330372

 values$bmi[f.ind] <- values$bmi[f.ind] - 33.515572

 values$bmi[m.ind] <- values$bmi[m.ind] - 31.338254

 values$egfr[f.ind] <- values$egfr[f.ind] - 89.558866

 values$egfr[m.ind] <- values$egfr[m.ind] - 88.788314

 values$acr[f.ind] <- log((**values$acr[f.ind]** + 0.0099999997764826) / 1000) + 4.314302355

 values$acr[m.ind] <- log((**values$acr[m.ind]** + 0.0099999997764826) / 1000) + 4.275179379

 values$hba1c[f.ind] <- values$hba1c[f.ind] - 63.618622

 values$hba1c[m.ind] <- values$hba1c[m.ind] - 63.889441

 values$years[f.ind] <- values$years[f.ind] - 5.406364

 values$years[m.ind] <- values$years[m.ind] - 5.183025

 # Coefficients

 fem.coeff <- list(age = 0.0424465,

 maori = 0.0770441,

 pacific = -0.253300,

 indian = 0.138371,

 asian = -0.3611259,

 smoker = 0.4391752,

 nzdep = 0.0699105,

 af = 0.7864886,

 familyhx = 0.1063846,

 lld = -0.1595083,

 athrombi = 0.0605766,

 bpl = 0.0988141,

 oral = 0.1248604,

 insulin = 0.3535548,

 sbp = 0.0127053,

 tchdl = 0.1139678,

 bmi = 0.0073966,

 egfr = -0.0090784,

 acr = 0.1842885,

 hba1c = 0.0076733,

 years = 0.0163962

 )

 male.coeff <- list(age = 0.0472422,

 maori = -0.0553093,

 pacific = -0.210811,

 indian = 0.1522338,

 asian = -0.3852922,

 smoker = 0.3509447,

 nzdep = 0.0413719,

 af = 0.5284553,

 familyhx = 0.2093793,

 lld = -0.0344494,

 athrombi = 0.0474684,

 bpl = 0.1532122,

 oral = 0.0051476,

 insulin = 0.1846547,

 sbp = 0.0054797,

 tchdl = 0.0805627,

 bmi = 0.0117137,

 egfr = -0.0025889,

 acr = 0.1815067,

 hba1c = 0.0074805,

 years = 0.0162351

 )

 value.score <- mapply(function(val, f.coeff, m.coeff){

 effect <- rep(0, length(input$sex))

 effect <- replace(effect, f.ind, val[f.ind] \* f.coeff)

 effect <- replace(effect, m.ind, val[m.ind] \* m.coeff)

 return(effect)

 },

 val = values,

 f.coeff = fem.coeff,

 m.coeff = male.coeff,

 SIMPLIFY = F)

 sum.score <- Reduce("+", value.score)

 estimate <- rep(0, length(sum.score))

 estimate <- replace(estimate, f.ind, 1 - 0.9455710 ^ exp(sum.score[f.ind]))

 estimate <- replace(estimate, m.ind, 1 - 0.9121175 ^ exp(sum.score[m.ind]))

 rounded.val <- as.numeric(formatC(round(estimate, dp),

 format = 'f',

 digits = dp))

 if(length(ls(pattern = "inval.")) >= 1){

 rounded.val <- replace(rounded.val,

 unlist(mget(ls(pattern = "inval."))),

 NA)

 }

 return(rounded.val)

}

##################################################################################

**ParamCheck** <- function(input, vars, call, is.table, allow.age, allow.na){

 # Set vars by call type

 demo.vars <- get("demo.vars", parent.frame())

 if(call == "NoPriorCVDRisk\_Policy"){

 num.vars <- character(0)

 smk.vars <- character(0)

 if(!is.table){

 warning(call. = FALSE, paste0("Using this function as a calculator for an individual is not recommended! See ?", call))

 }

 } else {

 num.vars <- get("num.vars", parent.frame())

 smk.vars <- get("smk.vars", parent.frame())

 }

 numNA.vars <- if(call %in% c("NoPriorCVDRisk\_Policy", "NoPriorCVDBleedRisk", "PriorT2DRisk", "NoPriorCVDRisk")){

 character(0)

 } else {

 get("numNA.vars", parent.frame())

 }

 lvl.vars <- if(call == "PostACSRisk"){

 get("lvl.vars", parent.frame())

 }

 # 1. Missing argument check

 if(is.table){

 input <- get("input", parent.frame())

 }

 if(!all(vars %in% names(input))) {

 stop(call. = F,

 paste("Missing parameter(s):",

 paste(setdiff(vars, names(input)),

 collapse = ", ")))

 }

 # 2. Missing input check

 if(is.table){ # Dataset provided (but missing correct input columns)

 dat <- as.data.frame(get("dat", parent.frame()), row.names = NULL)

 input <- input[!names(input) %in% c("dat", "dp", "allow.age", "allow.na")]

 colnames <- as.vector(sapply(input, as.character))

 is.missing <- any(!colnames %in% names(dat))

 if(is.missing){

 to.check <- colnames[!input %in% names(dat)]

 stop(call. = F,

 paste("Check input(s) names:",

 paste(sQuote(to.check),

 collapse = ", ")))

 }

 input[] <- dat[, colnames]

 } else { # One-off calculator (but missing correct input values)

 to.check <- names(input)[!sapply(input, class) %in% c("character", "numeric", "name", "logical")]

 if(length(to.check) > 0){

 stop(call. = F,

 paste("multiple input values detected for",

 paste(sQuote(to.check),

 collapse = ", ")))

 }

 }

 # 3. Compulsary input check

 vars <- c(demo.vars, num.vars)

 has.na <- sapply(vars,

 function(x){

 if(class(input[[x]]) == "name"){

 input[[x]] <- as.character(input[[x]])

 }

 any(is.na(input[[x]]) | input[[x]] == "NA")

 }, USE.NAMES = TRUE)

 if(any(has.na)){

 stop(call. = F,

 paste(paste(sQuote(names(which(has.na))),

 collapse = ", "),

 "cannot contain NA! For acceptable values,", paste0("see ?", call)))

 }

 # If all clear, put data to parent.frame

 assign("input", input, parent.frame())

 # Accepted Values

 ok.ages <- if(allow.age){

 as.numeric(18:110)

 } else {

 as.numeric(30:74)

 }

 na <- if(allow.na){

 c("na", "NA", NA)

 } else {

 character(0)

 }

 ok.female <- tolower(c("f", "female", 0))

 ok.male <- tolower(c("m", "male", 1))

 ok.nzdeps <- as.numeric(1:5)

 ok.nzeo <- tolower(c("NZ European", "European", "NZEO", "Euro", "E", "1", "10", "11", "12"))

 ok.maori <- tolower(c("Maori", "NZMaori", "NZ Maori", "M", "2", "21"))

 ok.pi <- tolower(c("Pacific", "Pacific Islander", "PI", "P", "3", "30", "31", "32", "33", "34", "35", "36", "37"))

 ok.asian <- tolower(c("Asian", "Other Asian", "SE Asian", "East Asian", "Chinese", "ASN", "A", "4", "40", "41", "42", "44"))

 ok.indian <- tolower(c("Indian", "Fijian Indian", "IN", "I", "43"))

 ok.other <- tolower(c("Other", "Middle Eastern", "African", "Latin American", "South American", "Latin", "MELAA", "ME", "5", "51", "52", "53", "54", "61"))

 ok.smoker <- tolower(c("Y", "Yes", "Smoker", "Current", "S", 1, "T", TRUE))

 ok.exsmkr <- tolower(c("Y", "Yes", "Ex", "Ex-smoker", "Exsmoker", "E", 1, "T", TRUE))

 ok.nonsmk <- tolower(c("N", "No", "Non", "Non-smoker", "N", 0, "F", FALSE))

 ok.true <- tolower(c("Y", "Yes", 1, "T", TRUE))

 ok.false <- tolower(c("N", "No", 0, "F", FALSE))

 ok.stemi <- tolower(c("STEMI", "ST-Elevation", "S", "2"))

 ok.nstemi <- tolower(c("NSTEMI", "NONSTEMI", "NON-STEMI", "N", "1"))

 # 4. Input check

 lapply(ls(pattern = "ok."),

 function(x)

 assign(x, get(x), envir = parent.frame(3L))

 )

 # Demographic

 lapply(c(demo.vars, smk.vars),

 function(x){

 if(x == "sex"){

 invalid <- which(!tolower(input[[x]]) %in% c(ok.female, ok.male))

 }

 if(x == "age"){

 invalid <- which(!as.numeric(input[[x]]) %in% ok.ages | is.na(input$age))

 }

 if(x == "eth"){

 invalid <- if(call == "NoPriorCVDRisk\_Policy"){

 which(!tolower(input[[x]]) %in% c(ok.nzeo, ok.maori, ok.pi, ok.indian, ok.asian, ok.other))

 } else {

 which(!tolower(input[[x]]) %in% c(ok.nzeo, ok.maori, ok.pi, ok.indian, ok.asian))

 }

 }

 if(x == "nzdep"){

 invalid <- which(!as.numeric(input[[x]]) %in% ok.nzdeps)

 }

 if(x == "exsmoker"){

 invalid <- which(!tolower(input[[x]]) %in% c(ok.exsmkr, ok.nonsmk, na))

 }

 if(x == "smoker"){

 invalid <- which(!tolower(input[[x]]) %in% c(ok.smoker, ok.nonsmk, na))

 }

 assign(paste("inval", x, sep = "."),

 invalid,

 envir = parent.frame(2L))

 })

 # Binary vars

 lapply(get("bin.vars", parent.frame()),

 function(x){

 invalid <- which(!tolower(input[[x]]) %in% c(ok.true, ok.false, na))

 assign(paste("inval", x, sep = "."),

 invalid,

 envir = parent.frame(2L))

 })

 if(call != "NoPriorCVDRisk\_Policy"){

 lapply(c(num.vars, numNA.vars),

 function(x){

 invalid <- which(!class(input[[x]]) %in% c("numeric", "integer", "logical"))

 assign(paste("inval", x, sep = "."),

 invalid,

 envir = parent.frame(2L))

 })

 }

 if(call == "PostACSRisk"){

 lapply(lvl.vars,

 function(x){

 if(x == "acstype"){

 invalid <- which(!tolower(input[[x]]) %in% c(ok.stemi, ok.nstemi, c(na, "0")))

 }

 assign(paste("inval", x, sep = "."),

 invalid,

 envir = parent.frame(2L))

 })

 }

 # warning message

 inval.vars <- ls(pattern = "inval.")

 inval.vars <- inval.vars[sapply(inval.vars,

 function(x){

 length(get(x)) >= 1

 }, USE.NAMES = T)]

 if(length(inval.vars) >= 1){

 warning(call. = FALSE,

 paste0(paste(sQuote(gsub("inval.", "", inval.vars)),

 collapse = ", "),

 " contain(s) one or more un-calculatable value(s). See R documentation using ", "?", call))

 }

 # Invalid params return NA score

 lapply(inval.vars,

 function(x)

 assign(x, get(x), envir = parent.frame(3L))

 )

}