# Installation

To install package:

remotes::install\_github("billy-nz/PredictRiskScores")

To load the package:

library(PredictRiskScores)

## 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.

NoPriorCVDRisk\_BMI(sex="F", age=65, eth="Indian", exsmoker=1 , smoker=0,

nzdep=5, diabetes=0, af=0, familyhx=1, lld=1,

athrombi=1, bpl=1, sbp=118, tchdl=3.3, bmi=32)

# [1] 0.0701

#--- exsmoker=1 & smoker==0 --> Ex-smoker

#--- exsmoker=0 & smoker==1 --> Current smoker

#--- exsmoker=0 & smoker==0 --> Never smoker

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(NoPriorCVDRisk\_BMI(sex=F, age=55, eth=IN, exsmoker=Y, smoker=0, nzdep=5,

diabetes=T,af=Y, familyhx=T, lld=1, athrombi=Y,

bpl=T, sbp=120, tchdl=3.2, bmi=NA)\*100, 0)

# [1] 14

R documentation:

library(PredictRiskScores)

??NoPriorCVDRisk\_BMI

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", "F", "M", "M", "M", "M"),

age = c(35, 35, 55, 55, 35, 82, 35, 35, 55, 75),

ethnic\_labels = c(1, 43, 4, 2, 1, 2, 3, 4, 2, 43),

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

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

nzdep\_quintiles = c(1, 1, 5, 5, 5, 1, 3, 5, 5, 5),

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

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

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

lipidlowering = c(0, 0, 0, 0, 1, 0, 0, 0, 0, 0),

antithrombics = c(0, 0, 0, 0, 1, 0, 0, 0, 0, 0),

bplowering = c(0, 0, 0, 0, 1, 0, 0, 0, 0, 0),

systolic\_bp = c(128, 128, 160, 138, 138, 128, 128, 120, 138, 120),

tchdl\_ratio = c(4, 4, 8.2, 4.8, 4.8, 4, 4, 3.2, 4.8, 3.2),

bmi = c(18, 18, 18, 27, 37, 18, 18, 18, 27, 32))

dim(DF)

# 10 observations 15 variables

# As a vectoriser

NoPriorCVDRisk\_BMI(dat=DF, sex=sex, age=age, eth=ethnic\_labels, exsmoker=exsmoker\_status, smoker=smoker\_status, nzdep=nzdep\_quintiles,

diabetes=diab\_status, af=af, familyhx=fam\_hx, lld=lipidlowering, athrombi=antithrombics, bpl=bplowering, sbp=systolic\_bp, tchdl=tchdl\_ratio, bmi=bmi)

# [1] 0.0061 0.0075 0.0899 0.1525 0.0588 0.2246 0.0206 0.0091 0.0704 0.1809

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 <- NoPriorCVDRisk\_BMI(dat=DF, sex=sex, age=age, eth=ethnic\_labels, exsmoker=exsmoker\_status, smoker=smoker\_status, nzdep=nzdep\_quintiles, diabetes=diab\_status, af=af, familyhx=fam\_hx, lld=lipidlowering, athrombi=antithrombics, bpl=bplowering, sbp=systolic\_bp, tchdl=tchdl\_ratio, bmi=bmi)\*100

DF$riskscores\_perc # %

# [1] 0.61 0.75 8.99 15.25 5.88 22.46 2.06 0.91 7.04 18.09

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

# [1] 1 1 9 15 6 22 2 1 7 18

#### 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 := NoPriorCVDRisk\_BMI(dat=DF, sex=sex, age=age, eth=ethnic\_labels, exsmoker=exsmoker\_status, smoker=smoker\_status, nzdep=nzdep\_quintiles, diabetes=diab\_status, af=af, familyhx=fam\_hx, lld=lipidlowering, athrombi=antithrombics, bpl=bplowering, sbp=systolic\_bp, tchdl=tchdl\_ratio, bmi=bmi)]

# OR

DF[, riskscores\_perc := round(NoPriorCVDRisk\_BMI(dat=DF, sex=sex, age=age, eth=ethnic\_labels, exsmoker=exsmoker\_status, smoker=smoker\_status, nzdep=nzdep\_quintiles, diabetes=diab\_status, af=af, familyhx=fam\_hx, lld=lipidlowering, athrombi=antithrombics, bpl=bplowering, sbp=systolic\_bp, tchdl=tchdl\_ratio, bmi=bmi)\*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 = NoPriorCVDRisk\_BMI(dat=DF, sex=sex, age=age, eth=ethnic\_labels, exsmoker=exsmoker\_status, smoker=smoker\_status, nzdep=nzdep\_quintiles, diabetes=diab\_status, af=af, familyhx=fam\_hx, lld=lipidlowering, athrombi=antithrombics, bpl=bplowering, sbp=systolic\_bp, tchdl=tchdl\_ratio, bmi=bmi), # %, 1 decimal place

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

as.data.frame()

Having trouble installing the “PredictRiskScores” package?

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

NoPriorCVDRisk\_BMI <- function(dat, sex, age, eth, nzdep, exsmoker, smoker, diabetes, af, familyhx, sbp, tchdl, bmi, bpl, lld, athrombi,...){

# Params

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

smk.vars <- c("exsmoker", "smoker")

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

num.vars <- c("sbp", "tchdl")

numNA.vars <- c("bmi")

# 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, numNA.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),

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

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)

bmi <- list(bmilt185 = +(input$bmi < 18.5 & !is.na(input$bmi)),

bmi25\_30 = +(input$bmi %in% 25:29.9 & !is.na(input$bmi)),

bmi30\_35 = +(input$bmi %in% 30:34.9 & !is.na(input$bmi)),

bmi35\_40 = +(input$bmi %in% 35:39.9 & !is.na(input$bmi)),

bmige40 = +(input$bmi >= 40 & !is.na(input$bmi)),

bmimiss = +(input$bmi == "" | is.na(input$bmi)))

values <- c(demo.vals, bin.vals, num.vals, bmi) # 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)

}

values$exsmoker[which(values$smoker == 1)] <- 0

# Recentering

values$age[f.ind] <- values$age[f.ind] - 56.05801

values$age[m.ind] <- values$age[m.ind] - 51.59444

# browser()

values$nzdep[f.ind] <- values$nzdep[f.ind] - 2.994877

values$nzdep[m.ind] <- values$nzdep[m.ind] - 2.975732

values$sbp[f.ind] <- values$sbp[f.ind] - 128.6736

values$sbp[m.ind] <- values$sbp[m.ind] - 128.8637

values$tchdl[f.ind] <- values$tchdl[f.ind] - 3.715383

values$tchdl[m.ind] <- values$tchdl[m.ind] - 4.385853

# Interaction

values$int\_age\_diab <- ifelse(values$diabetes == 0, 0, values$age)

values$int\_age\_sbp <- values$age \* values$sbp

values$int\_sbp\_bplt <- ifelse(values$bpl == 0, 0, values$sbp)

# Coefficients

fem.coeff <- list(age = 0.0734393,

maori = 0.4164622,

pacific = 0.2268597,

indian = 0.2086713,

asian = -0.2680559,

ex\_smoke = 0.1444243,

cur\_smoke = 0.6768396,

nzdep = 0.0957229,

diabetes = 0.4967444,

af = 0.9293084,

familyhx = 0.0645588,

lld = -0.0568366,

athrombi = 0.1393368,

bpl = 0.3487781,

sbp = 0.0176523,

tchdl = 0.1361335,

bmilt185 = 0.6277962,

bmi25\_30 = 0.0018215,

bmi30\_35 = -0.0169324,

bmi35\_40 = 0.0343351,

bmige40 = 0.3196519,

bmimiss = 0.0213595,

int\_age\_diab = -0.0189779,

int\_age\_sbp = -0.000471,

int\_sbp\_bplt = -0.0054002)

male.coeff <- list(age = 0.0669484,

maori = 0.3166164,

pacific = 0.2217931,

indian = 0.3666816,

asian = -0.4131973,

ex\_smoke = 0.0748648,

cur\_smoke = 0.5317607,

nzdep = 0.0631146,

diabetes = 0.4107586,

af = 0.6250334,

familyhx = 0.1275721,

lld = -0.0256429,

athrombi = 0.0701999,

bpl = 0.2847596,

sbp = 0.0179827,

tchdl = 0.1296756,

bmilt185 = 0.5488212,

bmi25\_30 = -0.033177,

bmi30\_35 = -0.0025986,

bmi35\_40 = 0.1202739,

bmige40 = 0.3799261,

bmimiss = -0.073928,

int\_age\_diab = -0.0124356,

int\_age\_sbp = -0.0004931,

int\_sbp\_bplt = -0.0049226)

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 - 0.9845026 ^ exp(sum.score[f.ind]))

estimate <- replace(estimate, m.ind, 1 - 0.9712501 ^ 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))

)

}