Package 'heaven'

November 20, 2026

Type Package

Title Fast Data Preparation for Danish Registry Data

Version 2026.10.30

Description Prepares and analyses large Danish administrative and health registry datasets. Provides memory-efficient routines for cohort construction, time splitting, code-based condition identification, prescription exposure periods, risk-set matching, and direct standardisation. The package is designed for secure research environments such as Statistics Denmark and builds on the data structures provided by the 'data.table' package.

Depends R (>= 3.5.0)

Encoding UTF-8

Imports data.table(>= 1.9.6), stats, parallel, foreach, utils, Rcpp (>= 1.0.14), lava, lubridate, tools

LinkingTo Rcpp, RcppArmadillo

Suggests arrow, dplyr, ggplot2, scales, zoo, testthat, doParallel, survival, Publish, sas7bdat, prodlim, riskRegression, knitr, rmarkdown

License GPL (>= 3)

URL https://github.com/tagteam/heaven

BugReports https://github.com/tagteam/heaven/issues

VignetteBuilder knitr

LazyData true

Collate 'RcppExports.R' 'averageIncome.R' 'charlsonIndex.R' 'createProject.R' 'riskSetMatch.R' 'exposureMatch.R' 'findConditions.R' 'getAdmLimits.R' 'groupAges.R' 'guessRawdataDirectory.R' 'heaven-data.R' 'heaven-package.R' 'hypertensionMedication.R' 'importDREAM.R' 'importSAS.R' 'incidenceMatch.R' 'kommuneRegion.R' 'lexisFromTo.R' 'lexisSeq.R' 'lexisTwo.R' 'listRawdata.R' 'matchReport.R' 'medicinMacro.R' 'mm1.R' 'mm2.R' 'patientProfiles.R' 'poissonRisk.R' 'poissonregression.R' 'sasWhereConstructors.R'

'scanRawdata.R' 'simAdmissionData.R' 'simPop.R'
'simPrescriptionData.R' 'splitFromTo.R' 'splitSeq.R'
'splitTwo.R' 'standardize.prodlim.R' 'standardize.proportion.R'
'standardize.rate.R' 'view_parquet.R' 'xrecepter.R'

Config/testthat/edition 3

Config/roxygen2/version 8.1.1

NeedsCompilation yes

Author Christian Tobias Torp-Pedersen [aut, cre],
Helene Charlotte Rytgaard [aut],
Anders Munch [aut],
Matthew David Phelps [aut],
Thomas Alexander Gerds [aut]

Maintainer Christian Tobias Torp-Pedersen <ctp@heart.dk>

Repository CRAN

Date/Publication 2026-11-20 12:00:02 UTC

Contents

atccodes . . . 3
averageIncome . . . 4
charlson.codes . . . 5
charlsonIndex . . . 5
createProject . . . 6
diseasecode . . . 7
dsr . . . 8
edu_code . . . 10
exposureMatch . . . 10
findCondition . . . 14
getAdmLimits . . . 16
groupAges . . . 17
hypertensionATC . . . 18
hypertensionMedication . . . 19
icdcodes . . . 20
importDREAM . . . 21
importSAS . . . 22
incidenceMatch . . . 27
innerMedicinMacro . . . 31
kommuneRegion . . . 32
kommune_codes . . . 33
lexisFromTo . . . 33
lexisSeq . . . 35
lexisTwo . . . 37
listRawdata . . . 39
matchReport . . . 40
medicinMacro . . . 41
mm1 . . . 45

mm2 . . . 47
patientProfile . . . 48
poissonregression . . . 50
poissonRisk . . . 50
rDateToSas . . . 51
sasDateRange . . . 51
sasSubstr . . . 52
sasWhereConstructor . . . 53
scanRawdata . . . 54
simAdmissionData . . . 55
simPop . . . 56
simPrescriptionData . . . 57
sks_codes . . . 58
splitFromTo . . . 58
splitSeq . . . 60
splitTwo . . . 62
standardize.prodlim . . . 65
standardize.proportion . . . 66
standardize.rate . . . 67
view_parquet . . . 69
xrecepter . . . 71

Index **74**

`atccodes` *atccodes*

Description

data.table with unique atc codes

Usage

```
atccodes
```

Format

A data.table

averageIncome *averageIncome*

Description

This function calculates average income for a range of years prior to a date. The function expects a data.table with yearly incomes from which to calcu- late the average.

The function can include multiple occurrences of a patient ID, but output is restricted to one average income calculated for each combination of person and year.

If the input data includes income from many years not relevant to the project, then it is wise to trim these data prior to using the function.

Usage

```
averageIncome(data,income,datvars,incomevars,numyears=5)
```

Arguments

data	- A dataframe/table with with an identification variable and a date from which the previous years' income should be averaged.
income	- A dataframe/table with at least three variables which are the identification of individual, the year and the income
datvars	- A character vector of two: identification and the date representing the year that is just after the years used to calculate average income.
incomevars	- A character with three variables: identification variable, year and income - in that order
numyears	- the number of years to include in the average income, default is 5.

Details

The function can in particular be used with the income data provided by Statistics Denmark. These files include a record per year and sometimes a record by quarter. The function will assume that repeated incomes for a year should be removed leaving a single income per year. The actual income selected for averaging is chosen by the user when selecting variables.

Value

A data.table with one row per person and index year for which income is available in the preceding numyears years. Its three columns are the identifier named by datvars[1], year (the index year) and income (the mean of the available yearly incomes).

Author(s)

Christian Torp-Pedersen

Examples

```
require(data.table)
dat <- data.table(pnr=c(1,2,3,3), dato=as.Date(paste0(2018:2021,"-11-11")))
indkomst <- data.table(pnr=rep(c(1,2,3,3),5),year=rep(2015:2020,5),
income=rep(200:205,5))
averageIncome(dat,indkomst,c("pnr","dato"),c("pnr","year","income"))
```

---

charlson.codes *charlson.codes*

Description

List with pre-specified codes for Charlson Index

Usage

```
charlson.codes
```

Format

A list

---

charlsonIndex *charlsonindex - define charlson index at a selected time*

Description

This function uses 19 disease categories and their weights to establish charlson index at a particular time indicated by a variable

The suggested list of Charlson definitions comes partly from Quann (Am J Epidemiol.2011 173:676-82), partly from DMCG.dk benchmarking consortium and with further minor adjustments provided by Peter Ascanius Jacobsen. It is unlikely that any international list of definitions will apply for a Danish population and the listing used should be either cited or provided in a supplement for publication. The data included in this package "charlson.codes" includes a selection of ICD8 and ICD10 codes. ICD9 has never been used in Denmark.

If there are repeats of patient id with different index date (data.date), then Charlson index is calculated independently for each of these dates.

Note: Only cases that have at least one of the relevant diseases will be given values with this function. The remaining should manually be provided zeroes.

Usage

```
charlsonIndex(data,ptid='pnr',vars,data.date,charlson.date,look.back=5
,icd.codes)
```

Arguments

data	- A dataframe/table with disease codes and dates
ptid	- Variable defining individual
vars	- variables where disease codes should be searched for
data.date	- variable in data with date of disease
charlson.date	- variable defining index date for determination of charlson index
look.back	- numnber of years to look back from charlson.date for diseases
icd.codes	- named list of charlson codes. If not specified the package internal object charlson.codes is used.

Details

The charlson weights and selection of disease codes are from DCMG.dk

Note: The function only calculates a charlson index for those observations where the index is at least "1". Cases with zero appears as NA and need to be converted to zero after the function.

Value

A list with 2 data.tables - index holding charlson index and elements holding presence of each category

Author(s)

Christian Torp-Pedersen

Examples

```
require(data.table)
set.seed(211)
adm <- simAdmissionData(10)
adm[,charlson.date:=as.Date("2017-01-01")]
adm2 <- adm[1:5]
adm2[,charlson.date:=as.Date("2015-01-01")]
adm <- rbind(adm,adm2)
ci <- charlsonIndex(data=adm,ptid='pnr',vars='diag',data.date='inddto',
  charlson.date="charlson.date")
```

---

createProject *ordnung muss sein*

Description

create new empty folder for project organisation

Usage

```
createProject(directory)
```

Arguments

directory	Full path of the new project. Must be supplied explicitly.

Details

ordnung muss sein

Value

No return value (invisible NULL). Called for the side effect of creating the project directory, subdirectories, README and script templates.

Author(s)

Thomas A. Gerds <tag@biostat.ku.dk>

Examples

```
project <- tempfile("heaven-project-")
createProject(project)
list.files(project)
unlink(project, recursive = TRUE)
```

---

diseasecode *diseasecode*

Description

List with pre-specified disease codes (ATC, ICD10, ICD8)

Usage

```
diseasecode
```

Format

A list

dsr *Confidence intervals for age standardized rates and rate ratios*

Description

Function to Compute confidence interval for directly standardized rates and rate ratios

Usage

```
dsr(
  count1,
  pop1,
  count0,
  pop0,
  stdpop,
  conf.level = 0.95,
  method = "gamma",
  crude = TRUE
)
```

Arguments

count1	counts for group 1 (e.g. exposed)
pop1	number of person-years in group 1
count0	counts for group 1 (e.g. exposed)
pop0	number of person-years in group 0
stdpop	number of person-years in stdpop population
conf.level	confidence level of confidence intervals
method	method for calculating confidence intervals
crude	logical. if TRUE also calculate crude rates

Details

Function to Compute confidence interval for directly standardized rates and rate ratios for sparse data. Method implemented include gamma confidence intervals (for DSR), exact confidence intervals (for crude rates), the inverse of the F distribution (for DSR ratio) and some Wald confidence interval (also on log-scale) for comparison purpose.

Value

A data.table with one row per group (0 or 1), containing the standardized rate, its rate.lower and rate.upper confidence limits, and the rate ratio relative to group 0 with ratio.lower and ratio.upper confidence limits. When crude = TRUE, there are two rows per group and a type column distinguishes "crude" and "standardized" estimates.

Author(s)

Paul F Blanche <pabl@sund.ku.dk> and Thomas A. Gerds <tag@biostat.ku.dk>

References

Fay, Michael P., and Eric J. Feuer. "Confidence intervals for directly standardized rates: a method based on the gamma distribution." Statistics in Medicine 16.7 (1997): 791-801.

Fay, Michael P. "Approximate confidence intervals for rate ratios from directly standardized rates with sparse data." Communications in Statistics-Theory and Methods 28.9 (1999): 2141-2160.

Fay, Michael P., et al. "Estimating average annual percent change for disease rates without assuming constant change." Biometrics 62.3 (2006): 847-854.

Fay, Michael P., Michael A. Proschan, and Erica Brittain. Combining one-sample confidence procedures for inference in the two-sample case. Biometrics 71.1 (2015): 146-156.

Interesting other approaches that avaoid inconsistency between exact CI and p-values are implemented in exactci and exact2x2, see

Fay, Michael P. "Two-sided exact tests and matching confidence intervals for discrete data." R journal 2.1 (2010): 53-58.

See Also

epitools::ageadjust.direct

Examples

```
library(data.table)
set.seed(84)
 n=160
d <- data.table(e1=rpois(n,lambda=9),
                rt1=rpois(n,lambda=1880),
                e2=rpois(n,lambda=123),
                rt2=rpois(n,lambda=80000))
d[,agegroups:=factor(rep(c("40-50","45-50","50-55","55-60","60-65","65-70","70-75","75-80"),n/8))]
d[,sex:=factor(rep(c("f","m"),c(n/2,n/2)))]
d[,year:=rep(2001:2010,n/10)]
D=d[,list(e1=sum(e1),rt1=sum(rt1),e2=sum(e2),rt2=sum(rt2)),by=c("sex","agegroups")]
D[sex=="m",e1:=e1+rpois(.N,lambda=as.numeric(agegroups)*17)]
D[sex=="m",rt1:=rt1-rpois(.N,lambda=as.numeric(agegroups)*1600)]
dsr(count1=D[sex=="m",e1], pop1=D[sex=="m",rt1],
count0=D[sex=="f",e1], pop0=D[sex=="f",rt1],
stdpop=D[sex=="f",rt1])
```

edu_code *"edu_code"*

Description

data.table with education codes

Usage

```
edu_code
```

Format

A data.table

exposureMatch *Exact matching on time-dependent exposure*

Description

A case is a person who initiates treatment at index date. A control is a person who has not yet initiated treatment at the index date of the case. The index date of the case can also be the onset of a disease (or other exposure), in which case controls are persons who do not yet have the disease (are not yet exposed). See `incidenceMatch` for situations where a case has a time to event outcome at the index date and controls do not have the outcome at the case index date.

Within a cohort or registry study, a number of controls are matched to each case on demographics, comorbidity, concomitant medication and other pre-treatment person characteristics measured at the index date of the case.

Matching may be useful in non-randomized studies when the aim is to estimate a

causal parameter: The average effect of treatment in a hypothetical study which randomizes treatment allocation. E.g., the difference or ratio of the absolute risk of developing an outcome within the first 6 months after the index date.

regression parameter: The conditional effect of treatment (exposure) on an outcome given pre-treatment (pre-exposure) covariates. E.g., a hazard ratio in a Cox regression model, or an odds ratio in a logistic regression model. In particular, when it is difficult or expensive or time-consuming to measure pre-exposure variables or to remove model dependency. See Details and references.

Usage

```
exposureMatch(
  ptid,
  event,
  terms,
  data,
  n.controls,
  case.index = NULL,
  end.followup = NULL,
  date.terms = NULL,
  duration.terms = NULL,
  output.count.controls = TRUE,
  cores = 1,
  seed = 0,
  progressbar = TRUE,
  verbose = TRUE
)
```

Arguments

ptid	Personal ID variable defining participant
event	Name of variable that defines cases. MUST be numeric 0/1 where 0 codes for never-case, and 1 for case.
terms	Vector of variable names specifying the variables that should be matched on. Make sure that appropriate classification is in place for truly continuous variables, such as age. This is to ensure a sufficient number of controls for each case. For example it may be difficult to find controls for cases of very high and very low ages and extreme ages should therefor further aggregated.
data	The single dataset with all information - coerced to data.table if data.frame
n.controls	Number of controls for each case
case.index	Name of the variable which contains the case index dates. This can be a calendar date variable or a numeric variable, i.e., the time to outcome event from a well defined baseline date. Missing values are interpreted as no event at the end of followup.
end.followup	Name of the variable which defines the date (as date or time) from which a control can no longer be selected as a control due in general to. **death:** Nothing happens thereafter **exposure of interest:** a control can be selected among those that later develop condition of interest, but not after the date of exposure **other competing risks:** Event after which we are not interested in the subject anymore. E.g., the date of an outcome event. **censoring:** Event after which we do not observe anymore. E.g., emmigration, end of study period, end of registry, drop-out The end.followup must be larger or equal to the `case.index` date.

`date.terms` A vector of variable names (character) in `data` specifying dates of conditions such as comorbidities. Controls are considered as matched only when the `date.terms` of the control are on the same chronological side of case index as for the case. Missing values are interpreted as absence of comorbidity at `end.followup`. Many datasets have comorbidities as time dependent variables. Matching on these requires that the comorbidity date is not (yet) reached for a corresponding variables for cases if the case does not have the comorbidity and similarly that the date has been reached when the case does have that co-morbidity.

`duration.terms` A list where each element defines a time duration term with two elements:

start Name of a variable which defines a date or time which defines the duration as the difference between this variable and the `case.index`.

min Numeric value given in days when `case.index` is a date and in the same time unit as `case.index` otherwise. Cases and controls have either both a duration as least as long as `min` or both a duration shorter than min.

Useful to prepare to summarize the history of exposure for cases and controls in an equally long period looking back in time from the `case.index`.

`output.count.controls`

Logical. If `TRUE` add number of found controls to each case/control set.

`cores` number of cores to use in the calculation.

`seed` Random seed to make results reproducible

`progressbar` set to `FALSE` to avoid progressbar

`verbose` Logical. Set to `TRUE` to see messages about the progress.

Details

The function performs exact matching and hence all matching variables must be factor variables or character.

In some cases there is a necessity to account for delayed entry. An example could be to ensure that people are not selected as controls prior to immigration to the country. This can be handled with "date.terms". Cases and controls can as an example be provided with a date.term that is maximum of the date of birth and the date of immigration.

In some cases it makes sense to think about the study population of a hypothetical study with randomized treatment allocation. Then, the aim of the function is similar to that of propensity score analysis and exact matching may be preferred as it does not dependent on a propensity score model (see King & Nielsen 2019).

It may appear tempting always to use multiple cores, but this comes at the cost of copying the data to the cores.

The function matchReport may afterwards be used to provide simple summaries of use of cases and controls

Value

data.table with cases and controls. After matching, a the variable "case.id" identifies sets which include 1 case and x matched controls.

Variables in the original dataset are preserved. The final dataset includes all original cases but only the controls that were selected.

Author(s)

Christian Torp-Pedersen & Thomas Alexander Gerds

References

Paul R Rosenbaum. Risk set matching. Design of observational studies, Springer, 2010.

Yunfei Paul Li, Kathleen J Propert, and Paul R Rosenbaum. Balanced risk set matching. Journal of the American Statistical Association, 96(455):870– 882, 2001.

Mohammad A Mansournia, Miguel A Herna'n, and Sander Greenland. Matched designs and causal diagrams. International journal of epidemiology, 42(3): 860–869, 2013.

King, G., & Nielsen, R. Why Propensity Scores Should Not Be Used for Matching. Political Analysis, 1-20. doi:10.1017/pan.2019.11

See Also

riskSetMatch matchReport Matchit

Examples

```
require(data.table)
case <- c(rep(0,40),rep(1,15))
ptid <- paste0("P",1:55)
sex <- c(rep("fem",20),rep("mal",20),rep("fem",8),rep("mal",7))
byear <- c(rep(c(2020,2030),20),rep(2020,7),rep(2030,8))
case.Index <- c(seq(1,40,1),seq(5,47,3))
startDisease <- rep(10,55)
control.Index <- case.Index
diabetes <- seq(2,110,2)
heartdis <- seq(110,2,-2)
diabetes <- c(rep(1,55))
heartdis <- c(rep(100,55))
library(data.table)
dat <- data.table(case,ptid,sex,byear,diabetes,heartdis,case.Index,
control.Index,startDisease)
# Exposure density matching
matchdat <- exposureMatch(ptid="ptid",event="case",
            terms=c("byear","sex"),data=dat,n.controls=2,
            case.index="case.Index",end.followup="control.Index")
matchdat
# Same with 2 cores
if (requireNamespace("doParallel", quietly = TRUE)) {
library(parallel)
library(foreach)
matchdat2 <- exposureMatch("ptid","case",c("byear","sex"),data=dat,2,case.index=
  "case.Index",end.followup="control.Index"
  ,cores=2)
matchdat2
all.equal(matchdat,matchdat2)
}
#Time dependent matching. In addition to fixed matching parameters there are
```

```
#two other sets of dates where it is required that if a case has that condi-
#tion prior to index, then controls also need to have the condition prior to
#the case index to be eligible - and if the control does not have the condi-
#tion prior to index then the same is required for the control.
matchdat <- exposureMatch("ptid","case",c("byear","sex"),data=dat,n.controls=2,
  case.index="case.Index",end.followup="control.Index",cores=1,
  date.terms=c("diabetes","heartdis"))
matchdat
```

`findCondition` *Find conditions in registry data*

Description

This functions is useful for the very common task of selecting cases based on a code which has complete or partial match to a vector of character vari- ables.

The function is designed to search a group of variables (character) for multiple conditions defined in a list of named character vectors. The func- tion will produce a data.table with selected variables for cases where a match is found. In addition a list of names character vectors can have exclusions from the search. This last facility is useful if e.g. all cancer except non melanoma skin cancer is sought. In that case inclusion can have all cancers and the exclusions just the non-melanoma skin cancer.

See examples for common use of the output

Usage

```
findCondition(data, vars, keep, conditions, exclusions=NULL,
match="contain",condition.name="X")
```

Arguments

`data`	Data in which to search for conditions
`vars`	Name(s) of variable(s) in which to search.
`keep`	a character vector of the columns in Data.table to keep in output
`conditions`	A named list of (vectors of) search strings. See examples.
`exclusions`	A names list of (vectors of) search strings to exclude from the output.
`match`	A variable to tell how to use the character vectors: "exact"=exactly matches the search string, "contains"=contains the search string, "start"=Starts with the search string, "end"=Ends with the search string
`condition.name`	Name of variable(s) where values define conditions. The values of this variable are the names from parameter "conditions".

Value

A data.table containing the columns selected by keep and a column named by condition.name, whose values identify the matching elements of conditions. Each row represents a matched condition in an input record.

Author(s)

Christian Torp-Pedersen <ctp@heart.dk>, Thomas A. Gerds <tag@biostat.ku.dk>

Examples

```
library(heaven)
library(data.table)

# find all diagnoses that start with "DT"
set.seed(800); adm <- simAdmissionData(800)
x <- findCondition(adm,vars=c("diag"),
        keep=c("pnr","inddto","uddto"),
        conditions=list(THIS=c("DT")),
        match="start",condition.name="THAT")
x
# restrict to first by pnr
x[x[,.I[1],by=list(pnr)]$V1]
# restrict to last by pnr
x[x[,.I[.N],by=list(pnr)]$V1]

opr <- data.table(
  pnr=1:100,opr=paste0(rep(c('A','B'),50),seq(0,100,10)),
  oprtil=paste0(rep(c('A','C'),50),seq(0,100,10)),
  odto=101:200
)
search <- list(Cond1=c('A1','A2'),Cond2=c('B10','B40','B5'),
Cond3=c('A1','C20','B2'))

excl <- list(Cond2='B100')

out <- findCondition(opr,vars=c("opr","oprtil"),
        keep=c("pnr","odto"),
        conditions=search, exclusions=excl,
        match="start",condition.name="cond")
### And how to use the result:
# Find first occurence of each condition and then use "dcast" to create
# a data.table with vectors corresponding to each condition.
test <- out[,list(min=min(odto)),by=c("pnr","cond")]
# provide a list of variables with one value each
test2 <- dcast(pnr~cond,data=test,value.var="min")
test2 # A datatable with first dates of each condition for each pnr, but only
      # for pnr with at least one condition
# Define a condition as present when before a certain index date
dates <- data.table (pnr=1:100,basedate=sample(0:200,size=100,replace=TRUE))
test3 <- merge(out,dates,by="pnr")
test3[,before:=as.numeric(odto<=basedate)] # 1 when condition fulfille
```

```
test3 <- test3[,list(before=max(before)),by=c("pnr","cond")]
test4 <- dcast(pnr~cond,value.var="before",data=test3)
test4[is.na(test4)] <- 0 # Converts NAs to zero
test4[]
```

`getAdmLimits` *Get Admission time*

Description

Adds first admission date and last discharge date to a sequence of admissions. The default input data.table is the current standard by which admissions are available in Statistics Denmark.

Usage

```
getAdmLimits(
  dt,
  pnr = "pnr",
  inddto = "inddto",
  uddto = "uddto",
  keep = NULL,
  collapse = FALSE,
  error = "warn"
)
```

Arguments

`dt`	data.table containing the variables
`pnr`	Variable with ID for each subject/group (default is `pnr`).
`inddto`	Variable with entry times. Must be date or numeric (default is `inddto`).
`uddto`	Variable with exit times. Must be date or numeric (default is `uddto`).
`keep`	Vector of names of variables of `dt` to keep in the output
`collapse`	if `TRUE` return only the lines with non-overlapping admissions
`error`	Character. One of `"warn"`, `"remove"` remove lines with errors, `"flag"` add a new variable called error. An error is defined as either a missing value in either inddto or uddto or when inddto>uddto.

Value

A data.table with additional class `admlimits`, sorted by subject and admission date. The `Date` columns `first.indate` and `last.outdate` give the start and end of each group of overlapping admissions. With `collapse = TRUE`, one row per group is returned and the original admission and discharge columns are removed. A logical `error` column identifies invalid input rows when `error = "flag"`, or when `error = "warn"` finds invalid rows.

Author(s)

Thomas A. Gerds <tag@biostat.ku.dk>

Examples

```
set.seed(8)
lpr <- simAdmissionData(10)
## Variables have default names
adm <- getAdmLimits(lpr)

## with errors
lpr1 <- lpr
lpr1 <- lpr1[uddto>as.Date("2010-01-01"),uddto:=NA]
adm1 <- getAdmLimits(lpr1)
adm1 <- getAdmLimits(lpr1,error="remove")
adm1 <- getAdmLimits(lpr1,error="flag")

## case where Variables have custom names
data.table::setnames(lpr, c('pnr','inddto','uddto'), c('personid', 'Entrydate','Outdate'))
adm2 <- getAdmLimits(lpr,pnr='personid',inddto='Entrydate',uddto='Outdate')

## duplicated and overlap
set.seed(8)
lpr3 <- simAdmissionData(10)
lpr3 <- lpr3[sample(1:NROW(lpr3),replace=TRUE,size=round(NROW(lpr3)*1.5))]
adm3 <- getAdmLimits(lpr3,collapse=TRUE)
```

`groupAges` *groupAges*

Description

Helper function to bin a vector of continuous ages into groups and automatically generate labels.

Usage

```
groupAges(
  x,
  lower,
  upper,
  by,
  sep = " - ",
  include.below = TRUE,
  include.above = TRUE,
  below.char = "<",
  above.char = "+"
)
```

Arguments

x	Vector to be grouped (usually ages in years)
lower	Lower bound (inclusive) of age
upper	Upper bound (exclusive) of age
by	How many years to group by
sep	Controls the serperator in the label between the the lower and upper bound for each age group
include.below	If TRUE (default) value under the lower bound will be included and labeled accordingly. If false, values under will be returned as NA
include.above	If TRUE (default) value over the upper bound will be included and labeled accordingly. If false, values over will be returned as NA
below.char	The character to use for labeling the group of value under the lower bound
above.char	The character to use for labeling the group of value above the lower bound

Value

Returns a factor vector of the same length as x.

Author(s)

Matthew Phelps

Examples

```
library(heaven)
set.seed(1)
x <- floor(runif(20, min = 0, max = 105))
groupAges(
 x,
 lower = 15,
 upper = 50,
 by = 10,
 below.char = "<",
 above.char = "+"
)
```

---

hypertensionATC *hypertensionATC*

Description

List with pre-specified ATC codes for classes of antihypertensive medication

Usage

```
hypertensionATC
```

Format

A list

`hypertensionMedication`

hypertensionMedication - define presence of hypertension based on prescriptions

---

Description

This function attempts to define presence of hypertension based on the use of at least two drugs for hypertension. The basis for the function is a validation study performed in relation to Olesen et al. (BMJ 2011;342:d124) where the sensitivity was 80.0 defined index date and using medication 180 days before that date. Not directly validated, this function can also calculate the date of start of hypertension as a date where an individual has picked up at least two classes of antihypertensive medication for two consecutive periods of 3 months.

Note: If there are repeats of ptid with separate index.date, then the presence of hypertension is evaluated for each date.

Usage

```
hypertensionMedication(data, vars=c("ptid","ATCcode",
 "prescription.date"), index.date=NULL,medication.definition=hypertensionATC)
```

Arguments

`data`	- A data frame/table containing prescription data
`vars`	- names of variables containing person identification, ATC code and date of prescription, and in that order
`index.date`	- name of a variable in data containing a date where the question is whether hypertension was present prior to this date. This will usually require merging of that variable to the medication data.
`medication.definition`	- a named list of character vectors defining the first part of the ATC codes for classes of antihypertensive medication. The default version is shown in details.

Details

This function uses a list of named character vectors to define classes of antihypertensive medication. The default versionis

AntiAdrenerg: 'C02A','C02B','C02C'

Diuretic: 'C02DA','C03A','C03B','C03D','C03E','C03X','C02DA','C07B','C07C','C07D','C08G','C09BA','C09DA','C

Vasodilator: 'C02DB','C02DD','C02DG'

BetaBlocker: 'C07A','C07B','C07C','C07D','C07F'

CalciumAntagonist: 'C07FB','C08','C08G','C09BB','C09DB'

ReninAngiotensinInhibitor; 'C09AA','C09BA','C09BB','C09CA','C09DA','C09DB','C09XA02','C09XA52'

Value

A data.table with the following variables

ptid: The variable from input defining the individual

hypertension: if an index variable is presen then it is a 0/1 variable as to wheter hypertension is present. If there is no index.date it is the date of hypertension

numDrugs: Number of drugs at time of hypertension or index.date

Author(s)

Christian Torp-Pedersen

See Also

findCondition

Examples

```
if (requireNamespace("zoo", quietly = TRUE)) {
require(data.table)
dat <-data.table(
 ATC=sample(c('C02A','C03A','C02DD','C07A','C07F','C09AA','C09BA'),100,
            replace=TRUE),
 EKSD=as.Date("2012-01-01")+(sample(1:600,100,replace=TRUE)),
 ptid=1:10,
 index=c(rep(as.Date("2013-06-01"),50),rep(as.Date("2013-06-02"),50)))
setkey(dat,"ptid")
# Hypertension with indexdate:
index <- hypertensionMedication(dat,vars=c("ptid","ATC","EKSD"),
 index.date="index")
index
noindex <-  hypertensionMedication(dat,vars=c("ptid","ATC","EKSD"))
noindex
}
```

`icdcodes` *icdcodes*

Description

data.table with unique icd codes

Usage

```
icdcodes
```

Format

A data.table

importDREAM *importDREAM*

Description

The DREAM dataset is a SAS dataset where all in the Danish population that have received public support are recorded. A number of variables are presented. For a detailed description of DREAM refer to www.dreammodel.dk

The current function can output 2 datasets from the DREAM database

support A long version data.table providing a sequential list of time periods and the codes for public support. Description of codes can be identified a number of places and for those working on Statistics Denmark there are SAS format codes available

branche A long version data.table providing a record per month for type of profession with start/end/work-code. SAS formats are also available for these codes

Usage

```
importDREAM(dreamData,type="support",pnr="pnr",explData=NULL)
```

Arguments

dreamData	Name of dataset holding DREAM data. Typically product of the importSAS function. In most cases it will be useful to add a set.hook to the importSAS function - set.hook="keep PNR branch:" for type of work and set.hook="keep=PNR y_:" for support
type	- "support" extracts weekly public support and "branche" supplies montly type of work. Other names cause error.
pnr	- name of person identifier in DREAM dataset, typically "PNR"
explData	Name of data holding relation between codes and explanatory text. Typically read from "//srvfsenas1/formater/SAS formater i Danmarks Statistik/txt_filer" using "read.csv2". The data is expected to hold two variables: the code and the explanation - in that order. The variable holding the code MUST be named "branche" or "support" as appropriate.

Details

DREAM data are organized in a special way. From the start of recording there is a weekly recording of receipt of support from the state. These are in se- parate variables. Then there is a montly recording of type of work also in separate variables. Other data are in DREAM, but not extracted by current function which either selects public support or type of work. The result is either occupation (branche) or support-type with start and end dates and in long format.

Value

the function returns a data.table with the following variables:

- 'pnr' - person identifier from call
- start - start of each period
- end - end of each period
- code - coding of period
- explanation - text to eplain coding if a text dataset was specified

Author(s)

Christian Torp-Pedersen

See Also

importSAS

Examples

```
# The following is a minute version of DREAM
library(data.table)
microDREAM <-data.table(PNR=c(1,2),branche_2008_01=c("","3"),
  branche_2008_02=c("1","3"), branche_2008_03=c("1",""),
  y_9201=c("","1"),y_9202=c("","2"),y_9203=c("","2"))
# Explanation of codes for "branche"
branche <- data.table(branche=c("1","2","3"),
            text=c("school","gardening","suage"))
support <- data.table(support=c("1","2"),text=c("education","sick"))
temp <- importDREAM(microDREAM,type="branche",pnr="PNR",explData=branche)
temp[]
temp2 <- importDREAM(microDREAM,type="support",pnr="PNR",explData=support)
temp2[]
```

`importSAS` *importSAS*

Description

Selective import of sas7bdat files into R data.table format

Usage

```
importSAS(filename,wd=NULL,keep=NULL,drop = NULL,
                  where = NULL,obs = NULL,filter = NULL,
                  filter.by = NULL, filter.negative = FALSE,
                  set.hook=NULL,step.hook=NULL,pre.hook=NULL,
                  post.hook=NULL,savefile = NULL,overwrite = TRUE,
```

```
                show.sas.code=FALSE,save.tmp = FALSE,content=FALSE,
                na.strings=c("dot"),date.vars=NULL,datetime.vars=NULL,
                character.vars="pnr", numeric.vars = NULL,sas.program,
                sas.switches, sas.runner, use.colClasses=TRUE,
                skip.date.conversion=FALSE, force.numeric=TRUE,
                sas.data.extension="sas7bdat", verbose=FALSE,...)
contentSAS(filename,wd=NULL)
```

Arguments

filename	Path to the SAS dataset to import. Relative paths are resolved before the function enters its temporary working directory.
wd	Writable parent directory for temporary SAS scripts, logs and CSV files. When NULL, the session temporary directory tempdir() is used. Files are created in a unique subdirectory and removed on exit, unless save.tmp = TRUE.
keep	A vector of variable names, i.e., the variables (columns) to read and keep from the dataset. Default is to read and keep all variables.
drop	Specifies the variables (columns) to leave out from the dataset. Default is to leave out no variables.
where	Specifies which conditions the observations (rows) from the dataset should fulfil. Default is no conditions. Use SAS syntax (see examples).
obs	Number of observations to read from the dataset. Setting this to Inf has the same effect as not setting it, i.e, read all observations.
filter	Alternative or in addition to the where statement it is possible to filter the rows of filename based on a data.table. E.g., filter can be a data.table with one column consisting of *unique* PNRs to specify that only the matching rows should be imported from filename.
filter.by	Vector of variable names to filter by. By default all variables present in the filter data are used.
filter.negative	Vector of length two that defines how to merge the filter data with the SAS dataset filename: • c(1,1) means observation are only included if they are present both the filter data and the SAS data. • c(-1,1) means observation are only included if they are not present filter data but present in the SAS data.
set.hook	Quoted SAS statments (within use single quotes) to be placed in addition to set options (where, keep, drop, obs) when setting the data set filename. See examples.
step.hook	Quoted SAS statments (within use single quotes) to be placed after setting the data set filename. See examples.
pre.hook	Quoted SAS code (within use single quotes) to be set in the beginning of the SAS program. For example, it maybe useful to specify options such as 'options obs=100;' in combination with a where statement.

post.hook — Quoted SAS code (within use single quotes) to be set at the end of the SAS program. For example, it maybe useful to specify 'proc print data=df;' in case of trouble.

savefile — Optional output CSV path, supplied explicitly to retain the imported data. Relative paths are resolved under wd (or tempdir() when wd = NULL); absolute paths are used as supplied.

overwrite — Logical. Determines whether or not to overwrite files already existing with the same name as files generated by this function. This is the temporary SAS file, log file and csv file, and possibly a permanent file with the name given under "savefile". If the value is FALSE and some files already exist, the function will abort and print the name of the problematic files. The default value is TRUE.

show.sas.code — Logical. If TRUE show sas code in R console before running it.

save.tmp — Logical. Option to save all temporary files. Even though this is set to FALSE, the csv file will be saved if there is given a filename in "savefile". Default value is FALSE.

content — Logical. If true, the function will only read and return the content of the import file. Together with save.tmp=TRUE, this can be used to generate the SAS file without running it.

na.strings — A vector of strings to interpret values of character variables as NA. Each element should be a regular expression that can be understood by grepl. For example, the default value "^\.$" matches fields that contain a single dot and nothing else than a dot.

date.vars — Vector of variables to be converted to date variables. For these variables a SAS format statement yymmdd10. is used to force the correct order of year months and days and the conversion is done with lubridate::ymd. Conversion can be skipped with argument skip.date.conversion.

datetime.vars — Vector of variables to be converted to datetime variables. For these variables a SAS format statement datetime. is used and the conversion is done with lubridate::dmy_hms. Conversions in both SAS and R programs can be skipped with argument skip.date.conversion.

character.vars — names of variables that should be converted to character. Case does not matter. Default is "pnr".

numeric.vars — character.vars names of variables that should be converted to numeric. case does not matter.

sas.program — sas program. On linux where .Platform$OS.type=="unix" this defaults to "sas" on any other system to "C:/Program Files/SASHome/SASFoundation/9.4/sas.exe"

sas.switches — On linux this defaults to "" on any other system to "-batch -nosplash -noenhancededitor -sysin"

sas.runner — How sas is invoked. On linux this defaults to "system" on any other system to "shell".

use.colClasses — Logical. If TRUE learn about the variable types from SAS's proc contents and pass to fread as argument colClasses.

skip.date.conversion — if TRUE do not try to convert any dates or datetime variables. If "SAS" only skip the format statements but format via lubridate (see date.vars and datetime.vars). If "R" skip lubridate conversion but keep the format statements.

force.numeric if TRUE force numeric format on numeric variables specified by argument numeric.vars.

sas.data.extension
String to be checked against the file extenstion of filename. Default is "sas7bdat".

verbose Logical. When TRUE warnings and errors are shown, otherwise not shown. This can be useful to turn on when there are problems with the result.

... Arguments passed to fread for reading the created .csv file. OBS: try to avoid specifying colClasses and instead use the arguments of importSAS: character.vars, date.vars, datetime.vars and numeric.vars.

Details

This function first runs SAS proc contents in the background and uses the result to write temporary SAS code that writes the data in csv format to a temporary file. The data are then read with data.table::freads into R. The function tries to help with the formatting of numeric and date variables and with the understanding of missing values. However, in some cases this help is counterproductive and can therefore be turned off, to some extent at least, with arguments na.strings, force.numeric, skip.date.conversion. Also, the user selects which columns and rows to import. See examples.

As R is case-sensitive while SAS is not, to avoid confusion all variable names are converted to lower case.

Value

For data import, a data.table with the columns requested in keep (or all columns) and rows selected by where (or all rows), up to obs rows. Variable names are converted to lower case and types are converted according to the supplied arguments and SAS formats. With content = TRUE, or for contentSAS(), a data.table of variable metadata is returned instead: SAS variable name Variable, storage Type, available Format and Informat, and the inferred R target.type.

Author(s)

Anders Munch <a.munch@sund.ku.dk> and Thomas A Gerds <tag@biostat.ku.dk>

References

This function is based on pioneering work by Jesper Lindhardsen.

Examples

```
# Scratch files are written under tempdir() and removed after each call.

## Not run:
# Requires a SAS installation and access to the private registry files.

# Before importing a SAS data file it is useful to look into the
# format of the variables:
contentSAS(filename="X:/Data/Rawdata_Hurtig/704791/diag_indl")
contentSAS(filename="x:/Data/Rawdata_Hurtig/704791/pop.sas7bdat")

# Also, it is often a good idea to initially only read a limited amount of data
```

```
df101 <- importSAS(filename="X:/Data/Rawdata_Hurtig/704791/diag_indl",obs=101,save.tmp=TRUE)
# and to examine the result
str(df101)
df101

# Format, dates, numeric, character
df101 <- importSAS(filename="X:/Data/Rawdata_Hurtig/704791/diag_indl",obs=101,
                   save.tmp=TRUE,date.vars="inddto",
                   numeric.vars="pnr",character.vars="packsize")

# we can also use the pre.hook to limit the number of observations via sas options:
importSAS(filename="X:/Data/Rawdata_Hurtig/704791/diag_indl",
       pre.hook="options obs=17;",where="diag='DN899'",keep=c("PNR","diag"),show.sas.code=1L)

# NOTE: In combination with a where statement SAS will find the first 101 observations that
#       satisfy the where statement. When the where statement finds nothing then SAS
#       will run through the whole file without finding anything

# To import the whole file just remove the limitations:
# df <- importSAS(filename="X:/Data/Rawdata_Hurtig/704791/diag_indl",show.sas.code=TRUE)
# But: this is not useful when files are huge and you should to look
# at the examples below which show how to read selected columns and rows only

# In huge data it is efficient to select only the required columns
# and to read only the rows of the data set that meet a criterion.
# The following example shows how to select columns with the keep option
# and how to specify the "where statement" of the SAS data step:
df0 <- importSAS(filename="X:/Data/Rawdata_Hurtig/704791/diag_indl",
                 obs=101,
                 keep=c("diag","diagtype"),
                 where="diagtype = 'A' and diag = '45490'")

# Examine the imported data.table
str(df0)
df0
# using where contains is powerful
importSAS(filename="X:/Data/Rawdata_Hurtig/704791/diag_indl",
                 obs=10,
                 save.tmp=TRUE,
                 where="diag contains 'DI2'",
                 keep=c("PNR","diag","inddto","uddto"))


# Another typical task is to import only the rows which correspond to
# a set of pnr numbers or a set of diagnoses or both.
# To achieve this the function importSAS is merging files during the import.
# The feature is called 'filter' and illustrated in the following examples:
# Example 1: import only rows of diag_indl which correspond to one of the
#            first 23 pnr's in pop
pop <- importSAS(filename="x:/Data/Rawdata_Hurtig/704791/pop.sas7bdat",
                 obs=23,keep="PNR")
df1 <- importSAS(filename="X:/Data/Rawdata_Hurtig/704791/diag_indl",
                 obs=8501,filter=pop)
```

```
str(df1)
df1

# Example 2: import only rows of diag_indl which correspond the
#            first pnr's in pop for which also diag is equal to DK409
pop2 <- importSAS(filename="x:/Data/Rawdata_Hurtig/704791/pop.sas7bdat",
                  obs=4223,keep="PNR")
pop2 <- pop2[,diag:="DZ508"]
df2 <- importSAS(filename="X:/Data/Rawdata_Hurtig/704791/diag_indl",
                  obs=8501,keep=c("pnr","diag","inddto","uddto","pattype"),
                  where="pattype ne 3",filter=pop2)
str(df2)
df2

## Sometimes the sas data file cannot be read due to unknown formats
## here is how to solve this:
df2a <- importSAS(filename="X:/Data/Rawdata_Hurtig/704791/diag_indl",
                  obs=101,
                  pre.hook="options nofmterr;")


# The hooks set.hook and step.hook can be used as follows:
df3 <- importSAS(filename="X:/Data/Rawdata_Hurtig/704791/diag_indl",
                  obs=101,
                  keep=c("PNR","diag","diagtype"),
                  set.hook="firstobs=5",
                  step.hook="by PNR; firstPNR=first.PNR;",
                  where="diag contains 'I'")
str(df3)
df3

# if you are more fluent in SAS than in R it may help to
# know how to communicate e.g., a keep statement, instead of
# using the keep argument:

df4 <-importSAS(filename="X:/Data/Rawdata_Hurtig/999999/Dream201801",
                 set.hook="keep=pnr branch;",
                 obs=1000)

# Because the "overwrite" argument is FALSE, running the above code again will abort the import
# to not overwrite the temporary files.
# Setting "overwrite=TRUE" will allow the function to overwrite the files.


## End(Not run)
```

---

incidenceMatch *incidenceMatch - Risk set matching for nested case-control designs*

Description

A case is a person who has an outcome event at the index date. A control is a person who does not yet have the event at the index date of the case. See `exposureMatch` for situations where treatment start or onset of a comorbidity define the case and the index date.

Risk set matching or incidence density sampling for nested case-control designs targets a a Cox regression model. with time-dependent covariates. In large-scale registry data the main purpose for risk set matching, instead of standard Cox regression on all data, is to save computation time. See Details and references.

Note that the parameter estimates of a conditional logistic regression analysis applied to the output of `incidenceMatch` are hazard ratios which should be interpreted in terms of a Cox regression model.

To provide necessary speed for large samples the general technique used is work with data.table and to create a series of match groups that have the fixed matching variables identical (such as birthyear and gender).

Usage

```
incidenceMatch(
  ptid,
  event,
  terms,
  data,
  n.controls,
  case.index = NULL,
  end.followup = NULL,
  date.terms = NULL,
  duration.terms = NULL,
  output.count.controls = TRUE,
  cores = 1,
  seed = 0,
  progressbar = TRUE,
  verbose = TRUE
)
```

Arguments

`ptid`	Personal ID variable defining participant
`event`	Name of variable that defines cases. MUST be numeric 0/1 where 0 codes for never-case, and 1 for case.
`terms`	Vector of variable names specifying the variables that should be matched on. Make sure that appropriate classification is in place for truly continuous variables, such as age. This is to ensure a sufficient number of controls for each case. For example it may be difficult to find controls for cases of very high and very low ages and extreme ages should therefor further aggregated.
`data`	The single dataset with all information - coerced to data.table if data.frame
`n.controls`	Number of controls for each case

`case.index` Name of the variable which contains the case index dates. This can be a calendar date variable or a numeric variable, i.e., the time to outcome event from a well defined baseline date. Missing values are interpreted as no event at the end of followup.

`end.followup` Name of the variable which defines the date (as date or time) from which a control can no longer be selected due to

death: Nothing happens thereafter

other competing risks: Event after which we are not interested in the subject anymore. E.g., the date of an outcome event.

censoring: Event after which we do not observe anymore. E.g., emmigration, end of study period, end of registry, drop-out

The end.followup must be larger or equal to the `case.index` date.

`date.terms` Unclear if useful in this context. But, see description for `exposureMatch`.

`duration.terms` A list where each element defines a time duration term with two elements:

start Name of a variable which defines a date or time which defines the duration as the difference between this variable and the `case.index`.

min Numeric value given in days when `case.index` is a date and in the same time unit as `case.index` otherwise. Cases and controls have either both a duration as least as long as `min` or both a duration shorter than min.

Useful to prepare to summarize the history of exposure for cases and controls in an equally long period looking back in time from the `case.index`.

`output.count.controls`

Logical. If `TRUE` add number of found controls to each case/control set.

`cores` number of cores to use in the calculation.

`seed` Random seed to make results reproducible

`progressbar` set to `FALSE` to avoid progressbar

`verbose` Logical. Set to `TRUE` to see messages about the progress.

Details

The function performs exact matching and hence all matching variables must be factor variables or character.

It may appear tempting always to use multiple cores, but this comes at the cost of copying the data to the cores.

This function prepares the data for fitting a Cox regression model via `survival::clogit` or directly via `survival::coxph` or equivalent routine. The regression parameters are hazard ratios. The matching variables are allowed to have a time-dependent non-proportional effect on the hazard rate of the outcome very much in the same way as would be obtained without matching by a strata statement to stratify the baseline hazard function. The original motivation for the nested case-control design is when it is difficult, expensive or time-consuming to measure the exposure variables.

The function matchReport may afterwards be used to provide simple summaries of use of cases and controls

Value

data.table with cases and controls. After matching, a the variable "case.id" identifies sets which include 1 case and x matched controls.

Variables in the original dataset are preserved. The final dataset includes all original cases but only the controls that were selected.

Author(s)

Christian Torp-Pedersen & Thomas Alexander Gerds

References

Bryan Langholz and Larry Goldstein. Risk set sampling in epidemiologic cohort studies. Statistical Science, pages 35–53, 1996.

Ornulf Borgan, Larry Goldstein, Bryan Langholz, et al. Methods for the analysis of sampled cohort data in the cox proportional hazards model. The Annals of Statistics, 23(5):1749–1778, 1995.

Vidal Essebag, Robert W Platt, Michal Abrahamowicz, and Louise Pilote. Comparison of nested case-control and survival analysis methodologies for analysis of time-dependent exposure. BMC medical research methodology, 5 (1):5, 2005.

See Also

exposureMatch clogit matchReport Matchit

Examples

```
require(data.table)
case <- c(rep(0,40),rep(1,15))
ptid <- paste0("P",1:55)
sex <- c(rep("fem",20),rep("mal",20),rep("fem",8),rep("mal",7))
byear <- c(rep(c(2020,2030),20),rep(2020,7),rep(2030,8))
case.Index <- c(seq(1,40,1),seq(5,47,3))
startDisease <- rep(10,55)
control.Index <- case.Index
diabetes <- seq(2,110,2)
heartdis <- seq(110,2,-2)
diabetes <- c(rep(1,55))
heartdis <- c(rep(100,55))
library(data.table)
dat <- data.table(case,ptid,sex,byear,diabetes,heartdis,case.Index,
control.Index,startDisease)
# Risk set matching
matchdat <- incidenceMatch(ptid="ptid",event="case",
            terms=c("byear","sex"),data=dat,n.controls=2,
            case.index="case.Index",
            end.followup="control.Index",seed=8)
matchdat
matchReport(matchdat)
# Same with 2 cores
library(parallel)
```

```
library(foreach)
if (requireNamespace("doParallel", quietly = TRUE)) {
matchdat2 <- incidenceMatch("ptid","case",c("byear","sex"),data=dat,
   n.controls=2,case.index="case.Index",end.followup="control.Index"
  ,cores=2,seed=8)
matchdat2
all.equal(matchdat,matchdat2)
}

# Case control matching with requirement of minimum exposure time in each
# group
ew <- incidenceMatch(ptid="ptid",event="case",terms=c("byear","sex"),
      data=dat,n.controls=2,case.index="case.Index",
      end.followup="control.Index",cores=1,
      duration.terms=list(list(start="startDisease",min=15)))
ew
```

`innerMedicinMacro` *The heart of the medicin macro*

Description

Inner process of medicin macro

Usage

```
innerMedicinMacro(
  dat,
  admdat,
  doses,
  idunique,
  index,
  prescriptionwindow,
  maxdepot,
  verbose
)
```

Arguments

`dat`	Prescription data.frame with numeric columns `pnr`, `eksd`, `strnum`, `packsize` and `apk`.
`admdat`	Admission data.frame with numeric columns `pnr`, `inddto` and `uddto`. All dates must use the same scale as `dat$eksd`.
`doses`	List of numeric vectors `value`, `min`, `max` and `def`; see `medicinMacro`.
`idunique`	Unique subject identifiers, one per row of `index`.
`index`	Data.frame with zero-based start and end row indices: `id.start`, `id.end` for prescriptions and `id.start.adm`, `id.end.adm` for admissions.

prescriptionwindow
: prescription window

maxdepot
: see medicine macro

verbose
: Set to FALSE or 0 to suppress diagnostic output, or TRUE or 1 to print debugging information.

Value

An unnamed list with one data.frame per subject, in the order of the rows of index, with length length(idunique). Each data.frame contains numeric columns pnr (subject identifier), X (estimated daily dose; zero denotes no exposure), B (start of the exposure interval) and E (end of the interval). The interval boundaries use the same numeric date scale as the input. When called by medicinMacro, this is days since 1995-01-01; that wrapper converts the boundaries to class Date. Intervals are interpreted as [B, E) by medicinMacro.

Author(s)

Helene Charlotte Rytgaard and Thomas Alexander Gerds

Examples

```
prescriptions <- data.frame(pnr = 1, eksd = 0, strnum = 1,
                            packsize = 5, apk = 1)
admissions <- data.frame(pnr = 1, inddto = 2, uddto = 4)
doses <- list(value = 1, min = 1, max = 1, def = 1)
index <- data.frame(id.start = 0, id.end = 0,
                    id.start.adm = 0, id.end.adm = 0)
innerMedicinMacro(prescriptions, admissions, doses, idunique = 1,
                  index = index, prescriptionwindow = 2,
                  maxdepot = 0, verbose = FALSE)
```

---

kommuneRegion *kommuneRegion - Converts kommune number to Region*

Description

Converts a 3-digit kommune number to text representing one of the five administrative regions.

Usage

```
kommuneRegion(kommune)
```

Arguments

kommune
: 3-digit kommune number.

Details

This function converts a 3-digit kommune number to text corresponding to one of the five regions in Denmark. The conversion is adapted from a DST SAS format and is valid from 2007. There is no easy way to convert older kommune numbers to the current regions, since some municipalities were split in the process of creating the five regions.

Value

A character vector of the same length as `kommune`, giving the Danish region for each municipality. Unrecognised or missing codes are represented by an empty string.

Author(s)

Christian Torp-Pedersen

Examples

```
kommuneRegion(410) # Sydjylland
```

---

kommune_codes *kommune_codes*

Description

data.table with the names and codes of every kommune, and the region they belong to

Format

A data.table

---

lexisFromTo *lexisFromTo*

Description

lexixFromTo is a specialised version of lexis splitting. In handling drug treatment, employment and similar factors the individual may move in and out of conditions. This function splits according such a situation. The input is first "base data" id/in/out/event (and other variables) that may already have been split by other functions. The other data is a sequence of id/from/to/value/name - that may represent multiple conditions with from/to and with "name" to distinguish.

Usage

```
lexisFromTo(indat,splitdat,invars,splitvars,default="0",datacheck=TRUE)
```

Arguments

indat	- base data with id, start, end, event and other data - possibly already split
splitdat	- Data with splitting guide - id/from/to/value/name
invars	- vector of colum names for id/entry/exit/event - in that order, example: c("id","start","end","event")
splitvars	- vector of column names containing dates to split by. example: c("id","start","end","value","name") - must be in that order!
default	- Value given to intervels not given a value by the function.
datacheck	- This function can crash or produce incorrect results if input data have overlapping intervals or negative intervals in any of the two input datasets. Thic is checked and error produced by datacheck. Can be omitted if data are checked otherwise

Details

The input to this function are two data.tables and two lists of the critical variables. The BASE data it the data to be split. This data must have a variable to identify participants, start/end times and a variable to indicate event after last interval. The other table (SPLITTINGUIDE) contains possibly multiple records for each participants with id/from/to/value/name. The program checks that intervals are not negative and that intervals for each "name" and each individual do not overlap. Violation stops the program with error. Overlaps may occur in real situations, but the user needs to make decisions regarding this prior to this function.

It is required that the splittingguide contains at least one record. Missing data for key variables are not allowed and will cause errors.

Value

The function returns a new data table where records have been split according to the splittingguide dataset. Variables unrelated to the splitting are left unchanged.

Author(s)

Christian Torp-Pedersen

Examples

```
library(data.table)
dat <- data.table(id=c("A","A","B","B","C","D"),
                  start=as.Date(c(0,100,0,100,200,400),origin='1970-01-01'),
                  end=as.Date(c(100,200,100,200,300,500),origin='1970-01-01'),
                  event=c(0,1,0,0,1,1))
split <- data.table (id=c("A","A","A","A","B","B","B","D","D"),
                   start=as.Date(c(0,50,25,150,110,150,400,300,500),origin='1970-01-01'),
                  end= as.Date(c(25,75,150,151,120,250,500,400,500),origin='1970-01-01'),
                     value=c(1,2,3,4,1,2,3,6,7),
                     name=c("d1","d1","d2","d2","d1","d1","d2","d3","d4"))
#Show the dataset:
dat[]
split[]
```

```
temp <- lexisFromTo(dat # inddato with id/in/out/event
                    ,split # Data with id and dates
                  ,c("id","start","end","event") #names of id/in/out/event - in that order
                   ,c("id","start","end","value","name")) #Nmes var date-vars to split by
```

---

lexisSeq *lexisSeq*

Description

LexisSeq is one of three split functions defined in heaven. The purpose is to split according a vector of dates. Typical situations are age (e.g. 5 year periods), calender time (e.g. 2 year periods) and selected times after a si- tuation of interest (e.g. three selected time periods after onset of a disease). The input is a data.table and splitting guide. The "base" data are the data to be split. They may contain much information, but the key is "id", "start","end" and "event". These describe the participant's id, start of time period, end of time period and the event of interest (must be 0/1).

The other input is data to define splitvector and name. The splitvector may be a fixed vector (format="vector", e.g. a series of fixed calender dates) or a list of 3 integers defining start, end and intervel to split by (format="seq", for a split on age between 20 and 80 by 5 years a splitvector could be defined as: splitvector <- c(20,80,5)*365.25 and provided to the function as a variable). "varname" is a name of a variable in the data.table the defines a value to be added to the splitvector. For the age split just used as an example it would be a variable containing the birthdate. For a split after onset of a conditi- on it should be the date of the condition and NA when the condition does not occur. When no value should be added to the vector (e.g. split by calender time) "varname" should keep its default value of NULL.

On output a new variable with default name "value" defines the result of splitting. The variable can be renamed to a user defined name (e.g. value="myvalue"). This variable will contain zero when time is before the first value of the splitting vector (added the "varname") and then increased by one as each value of the splitting vector is reached.

Overall the function provides identical usefulness as the SAS lexis macro

Usage

```
lexisSeq(indat,invars,varname=NULL,splitvector,format,value="value",
datacheck=TRUE)
```

Arguments

indat	base data with id, start, end, event and other data - possibly already split
invars	colum names for id,entry,exit,event - in that order, example: c("id","start","end","event")
varname	name of variable to be added to vector
splitvector	A vector of calender times (integer). Splitvector is a sequence of fixed dates (or other time scala).
format	String with two possible values: • "vector" a series of fixed calender dates

- "seq" see description

value	0 to the left of the vector, increase of 1 as each element of vector is passed
datacheck	- Checks that data are in appropriate format and that intervals are neighter negative or overlapping. Can be omitted if checked elsewhere.

Details

The input must be data.table. This data.table is assumed already to be split by other functions with multiple records having identical participant id. The function extracts those variables necessary for splitting, splits by the provided vector and finally merges other variable onto the final result.

A note of caution: This function works with dates as integers. R has a de- fault origina of dates as 1 January 1970, but other programs have different default origins - and this includes SAS and Excell. It is therefor important for decent results that care is taken that all dates are defined similarly.

The output will always have the "next" period starting on the day where the last period ended. This is to ensure that period lengths are calculated pro- perly. The program will also allow periods of zero lengths which is a conse- quence when multiple splits are made on the same day. When there is an event on a period with zero length it is important to keep that period not to loose events for calculations. Whether other zero length records should be kept in calcul ations depend on context.

Value

The function returns a new data table where records have been split according to the values in splitvector. Variables unrelated to the splitting are left unchanged.

Author(s)

Christian Torp-Pedersen

See Also

lexis2 lexisFromTo

Examples

```
library(data.table)

dat <- data.table(ptid=c("A","A","B","B","C","C","D","D"),
                  start=as.Date(c(0,100,0,100,0,100,0,100),origin="1970-01-01"),
                  end=as.Date(c(100,200,100,200,100,200,100,200),origin="1970-01-01"),
                  dead=c(0,1,0,0,0,1,0,1),
                Bdate=as.Date(c(-5000,-5000,-2000,-2000,0,0,100,100),origin="1970-01-01"))
#Example 1 - Splitting on a vector with 3 values to be added to "Bdate"
out <- lexisSeq(indat=dat,invars=c("ptid","start","end","dead"),
                varname="Bdate",
                splitvector=as.Date(c(0,150,5000),origin="1970-01-01"),
                format="vector")
out[]
#Example 2 - splitting on a from-to-by sequence with no adding (calender time?)
out2 <- lexisSeq(indat=dat,invars=c("ptid","start","end","dead"),
                  varname=NULL,splitvector=c(0,200,50),
```

```
                format="seq",value="myvalue")
out2[]
```

---

lexisTwo *lexisTwo*

Description

Splitting is about collecting person specific exposure-outcome-confounder pattern over time in start-stop-event format. lexixTwo is one of 3 splitting functions in heaven. lexixTwo is useful to add the time-dynamic information about comorbidities and other events in binary (yes/no) format to an existing data set which readily contains person specific information in start-stop-event format. A person specific time interval (start-stop) of the existing data set is split according to the occurrence dates of the comorbidities and other events whenever the comorbidity status (event status) changes within the time interval.

The "base" data are the data to be split. They may contain much information, but the key is "id","start","end" and "event". These describe the participant's id, start of time interval, end of time interval and the event of interest (must be 0/1).

The other input is a data.table with the splitting guide. This can be supplied in two formats: wide and long Wide format: This requires one record pr. individual that have dates to be split on. One column defines the same id as in the "base" table. The other columns contain dates for each condition where the split should occur. These column names will also appear in the output data, but on output the values are zero before the dates and 1 after. When dates are NA output has zero.

Long format: This requires one record per data where a possible split should occur. The columns should contain id, name of condition and the data to split on

Usage

```
lexisTwo(indat,splitdat,invars,splitvars,format="wide",datacheck=TRUE)
```

Arguments

indat	A data.table or data.frame whose first 4 columns are in that order: **id** Person identification variable such as PNR. The data may contain multiple lines per subject. **start** Start of time interval. Either a date or an integer/numeric. **end** End of time interval. Either in date format or given as numeric/integer. **event** Binary 0-1 variable indicating if an event occurred at end of interval
splitdat	The splitting guide. A data.table which contains person specific information about the onset dates of comorbidities and other events. Wide format: **id** Person identification variable such as PNR. The data may contain multiple lines per subject. **Date 1** Either a date or an integer/numeric. Format must match that of the start and stop of arguments indat. The onset date of comorbidity 1 or other event. If integer/numeric it can be time since a baseline date on project specific scale (e.g., days or months).

Date 2 Either a date or an integer/numeric. The onset date of comorbidity 2 or other event. If integer/numeric it can be time since a baseline date on project specific scale (e.g., days or months).

Dat3 ... Additional date variables to split on.

Long format:

id Person identification variable such as PNR. The data may contain multiple lines per subject.

Condition name Character providing the variable name of condition

Date Either a date or an integer/numeric. The onset date of comorbidity or other event. If integer/numeric it can be time since a baseline date on project specific scale (e.g., days or months).

`invars` vector of column names for id/entry/exit/event - in that order, example: c("id","start","end","event")

`splitvars` For wide format: - vector of column names of columns containing dates to split by. example: c("date1","date2","date3","date4") For long format: - vector of the 3 columns in the data.table: id/name/date, example: c("id","name","date") The name of the id column must be the same in both datasets

`format` - format of splitting guide - "wide" or "long"

`datacheck` - This program may crash if intervals are overlapping or negative. Datachecking produces an error in such cases. This can be omitted if the data have been checked by other means. For the splitting guide this options checks that there is only one entry for each variable to split by for each person identifier.

Details

The program checks that intervals are not negative. Violation results in an error. Overlap may occur in real data, but the user needs to make decisions regarding this prior to using this function.

It is required that the splitting guide contains at least one record. Missing data in the person id variables are not allowed and will cause errors.

A note of caution: This function works with dates as numeric. R has a default original of dates as 1 January 1970, but other programs have different default origins - and this includes SAS and Excel. It is therefor important for decent results that care is taken that all dates are defined similarly.

The output will always have the "next" period starting on the day where the last period ended. This is to ensure that period lengths are calculated pro- perly. The program will also allow periods of zero lengths which is a conse- quence when multiple splits are made on the same day. When there is an event on a period with zero length it is important to keep that period not to loose events for calculations. Whether other zero length records should be kept in calculations depends on the context.

Value

The function returns a new data table where records have been split according to the splittingguide dataset. Variables unrelated to the splitting are left unchanged. The names of columns from "splitvars" are also in output data, but now they have the value zero before the dates and 1 after.

Author(s)

Christian Torp-Pedersen

See Also

lexisSeq lexisFromTo

Examples

```
library(data.table)

dat <- data.table(pnr=c("123456","123456","234567","234567","345678","345678"
,"456789","456789"),
                start=as.Date(c(0,100,0,100,0,100,0,100),origin="1970-01-01"),
                end=as.Date(c(100,200,100,200,100,200,100,200),origin="1970-01-01"),
                event=as.integer(c(0,1,0,0,0,1,0,1)))

split <- data.table (pnr=c("123456","234567","345678","456789"),
como1.onset=as.integer(c(0,NA,49,50)), como2.onset=as.integer(c(25,75,49,49)),
como3.onset=as.integer(c(30,NA,49,48)), como4.onset=as.integer(c(50,49,49,47)))
#Show the datasets:
dat[]
split[]
lexisTwo(dat # inddato with id/in/out/event
   ,split # Data with id and dates
   ,c("pnr","start","end","event") #names of id/in/out/event - in that order
   ,c("pnr","como1.onset","como2.onset","como3.onset","como4.onset"))
   #Names of date-vars to split by
# And with splittingguide in long format
splitvars <- c("como1.onset","como2.onset","como3.onset","como4.onset")
split <- data.table::melt(data=split,id.vars="pnr",measure.vars=splitvars,
  variable.name="name",value.name="value")
split[,value:=as.Date(value,origin="1970-01-01")]
split[]
split <- split[!is.na(value)] # remove missing values
lexisTwo(dat # in-data with id/in/out/event
   ,split # Data with id/name/date
   ,c("pnr","start","end","event") #names of id/in/out/event - in that order
   ,c("pnr","name","value")
   ,format="long")
```

`listRawdata` *List sas data files*

Description

List the contents of a directory which contains sas data files

Usage

```
listRawdata(dir, pattern = "\\.sas7bdat$", full.names = FALSE, ...)
```

Arguments

dir	directory which contains the sas files
pattern	passed to list.files
full.names	passed to list.files
...	passed to list.files

Details

List the contents of a directory which contains sas data files

Value

A character vector of matching SAS file names, including their paths when full.names = TRUE.

Author(s)

Thomas A. Gerds <tag@biostat.ku.dk>

See Also

scanRawdata, addUsualRegisters

Examples

```
rawdir <- tempfile("heaven-rawdata-")
dir.create(rawdir)
file.create(file.path(rawdir, "pop.sas7bdat"))
listRawdata(rawdir)
unlink(rawdir, recursive = TRUE)
```

---

matchReport — *matchReport - report of matching from incidenceMatch and exposureMatch*

Description

The function provides very simple tables of the success of finding controls and the reuse of cases and controls. Designed to deal with the results of function riskSetMatch

Usage

```
matchReport(x, verbose = TRUE)
```

Arguments

x	result of exposureMatch or incidenceMatch
verbose	Logical. Print the report when TRUE; set to FALSE to retrieve the tables without console output.

Details

This function can be helpful to reevaluate matching options after exact matching. If many cases do not find controls it may be desirable to reduce the number of matching terms or to do further rounding of continuous matching variables.

Value

Invisibly returns a named list of two objects of class `table`. `matching.success` counts cases by the number of matched controls; `control.reuse` counts control subjects by the number of times each was selected. The report is also printed when `verbose = TRUE`.

Author(s)

Christian Torp-Pedersen & Thomas Alexander Gerds

See Also

incidenceMatch exposureMatch

Examples

```
require(data.table)
case <- c(rep(0,40),rep(1,15))
ptid <- paste0("P",1:55)
sex <- c(rep("fem",20),rep("mal",20),rep("fem",8),rep("mal",7))
byear <- c(rep(c(2020,2030),20),rep(2020,7),rep(2030,8))
caseIndex <- c(seq(1,40,1),seq(5,47,3))
controlIndex <- caseIndex
library(data.table)
dat <- data.table(ptid,case,sex,byear,caseIndex,controlIndex)
# Very simple match without reuse - no dates to control for
matched.data <- incidenceMatch(ptid="ptid",event="case",terms=c("byear","sex"),
data=dat,n.controls=2,
case.index="caseIndex",end.followup="controlIndex")
matchReport(matched.data)
```

`medicinMacro` *Medicin macro to estimate prescription lengths and averages*

Description

Processing function, to perform calculations according to the data and variables specified in preprocessing object.

Usage

```
medicinMacro(
  drugs,
  drugdb,
  admdb,
  datesdb,
  type = "dynamic",
  drugdb.datevar = "eksd",
  admdb.datevars = c("first.indate", "last.outdate"),
  datesdb.datevars,
  method,
  cap.values,
  window,
  pre.window,
  id = "pnr",
  strength.var = "strnum",
  packsize.var = "packsize",
  atc.var = "atc",
  apk.var = "apk",
  splitting = FALSE,
  verbose = FALSE
)
```

Arguments

`drugs` A named list of drugs. Each element of this list should be a list with the following elements:

atc A vector of ATC codes which should match the components of the drug exactly.

maxdepot The maximum total dose that a single patient can possibly stack.

period A vector of dates to limit the period in which to estimate the daily dose.

prescriptionwindow Default is 2 prescriptions.

doses A named list with the elements `value`, `min`, `max` and `def`. Here `value` is a vector of strengths of one unit (e.g. pill) of the drug. The vector should have one such strength for each of the different packages that occur in the data. `min` is a vector of the same length as `value` where each element is the assumed minimum total dose that a patient can consume on one day. `max` is a vector of the same length as `value` where each element is the assumed maximum total dose that a patient can consume on one day. `def` is a vector of the same length as `value` where each element is the assumed default dose that an average patient would consume on one day. See examples.

`drugdb` data.table with (subset of) medical drugs registry

`admdb` data.table with (subset of) hospital admission registry. The data.table should be prepared such that it contains only overnight hospital stay (i.e., pattype=2) and non-overlapping hospital stay periods, i.e., as obtained with `getAdmLimits` (SAS-AKA: code-from-hell).

datesdb	data.table with individual periods or time points of interest. Only used when the type argument is different from "dynamic". Should contain one start and end date for each individual when type equal to "period" and just one date for each individual when type equal to "cross-sectional" (see examples).
type	string, specifying which analysis to perform. Can be "dynamic", "period", or "cross-sectional". Default is "dynamic". (see examples).
drugdb.datevar	name of the date variable in drugdb. Default is "eksd".
admdb.datevars	names of the date variables in admdb. Default is c("inddto","uddto").
datesdb.datevars	name(s) of the date variables in datesdb. Should be a single string when type="cross-sectional" and a vector of two string when type="period". Default is "time" in the first case and c("start", "end") in the second.
method	Which method to use to estimate the daily dose. Can be "default.doses" or "number.of.days". Default is "default.doses". See documentation for more details. Only used when the type is different from "dynamic".
cap.values	Boolean, specifying whether or not to restrict the estimated daily dose to lie within the specified minimum and maximum dose. Default is TRUE Only used when the type is different from "dynamic".
window	Only used when the type = "cross-sectional". Specify how many days back from the given time point we should go to find purchases to estimate the daily dose.
pre.window	Only used when the type = "period". Specify how many days back from the individual end periods we should go to find purchases to estimate the daily dose for the first part of the periods.
id	Name of variable to identify individuals. Default is "pnr".
strength.var	Name of variable to identify strength. Default is "strnum".
packsize.var	Name of variable to identify size of packages. Default is "packsize".
atc.var	Name of variable to identify drugs. Default is "atc".
apk.var	Name of variable to identify number of packages. Default is "apk".
splitting	Split the data into 10 chunks to estimate remaining time (and possibly speed up computation).
verbose	Logical. Set to TRUE to print diagnostic information; the default FALSE suppresses this output.

Value

A named list of class medicinmacro, containing a data.table for each drug with exposure estimates. For type = "dynamic", each table contains the subject identifier, estimated daily dose, exposure interval dates firstday and lastday, and interval length exposure.days. Intervals run from firstday (inclusive) to lastday (exclusive). For type = "period", tables contain prescription intervals and estimated dose, drug supply, leftover drug and proportions of time covered. For type = "cross-sectional", tables contain the assessment time, dose and supply estimates, and the logical exposure indicator exposed.at.time. Drugs with no matching prescriptions in dynamic mode, or with unsupported stockpiling settings in the other modes, are omitted.

Author(s)

Helene Charlotte Rytgaard, Anders Munch and Thomas Alexander Gerds based on the famous SAS macro by Christian Tobias Torp-Pedersen

Examples

```
set.seed(05021992)
library(data.table)
N=20
packs = list("R03AK11"=list(c(750,75),c(500,200),c(400,200)),
             "R03AL03"=list(c(750,75),c(500,200),c(400,200)),
             "C01CA01"=list(c(200,100),c(750,30)))
lmdb=simPrescriptionData(N,packages=packs)
lpr=simAdmissionData(N)
lpr <- getAdmLimits(lpr,collapse=TRUE)
drug1 = list(atc=c("R03AK11","R03AL03","R03AC02","R03AC04","R03AC19",
                   "R03AL02","R03AA01","R03AC18","R03AL01"),
             maxdepot=4000,
             period=as.Date(c("1995-01-01", "2012-12-31")),
             prescriptionwindow=2,
             doses=list(value=c(750,500,400,200,75),
                        min = c(250,200,200,100,25),
                        max = c(1000,600,800,600,100),
                        def = c(750,500,400,200,75)))
drug2=list(atc=c("C01CA01","C01AA05"),
           maxdepot=4000,
           period=as.Date(c("1995-01-01", "2012-12-31")),
           prescriptionwindow=2,
           doses=list(value=c(200, 400, 500,750),
                      min = c(100, 200, 250,750),
                      max = c(300, 800, 1000,750),
                      def = c(200, 400, 500,750)))
x=medicinMacro(drugs=list("drug1"=drug1,"drug2"=drug2),drugdb=lmdb,admdb=lpr)
x$drug1

## without admdata
x1=medicinMacro(drugs=list("drug1"=drug1,"drug2"=drug2),drugdb=lmdb,admdb=NULL)
x1$drug1

## Examples with other types
## NB: Calculations with maxdepot>0 is not implemented yet for type different from "dynamic".
## Not specifying maxdepot is the same at setting maxdepot=0.
library(data.table)
drug1 = list(atc=c("R03AK11","R03AL03","R03AC02","R03AC04","R03AC19",
                   "R03AL02","R03AA01","R03AC18","R03AL01"),
             doses=list(value=c(750,500,400,200,75),
                        min = c(250,200,200,100,25),
                        max = c(1000,600,800,600,100),
                        def = c(750,500,400,200,75)))
drug2=list(atc=c("C01CA01","C01AA05"),
           doses=list(value=c(200, 400, 500,750),
                      min = c(100, 200, 250,750),
```

```
                      max = c(300, 800, 1000,750),
                      def = c(200, 400, 500,750)))
## Note that we define the periods separately, as they are now individual:
ind.periods <- data.table(pnr=1:N,
                          start=rep(as.Date("1996-01-01"), N)+floor(rnorm(N,sd=100)),
                          end=rep(as.Date("1998-05-30"), N)+floor(rnorm(N,sd=100)))
z=medicinMacro(drugs=list("drug1"=drug1,"drug2"=drug2),drugdb=lmdb,admdb=lpr,
               type="period",datesdb=ind.periods)
z$drug1
## Using different method:
z=medicinMacro(drugs=list("drug1"=drug1,"drug2"=drug2),drugdb=lmdb,admdb=lpr,
               type="period",datesdb=ind.periods, method="number.of.days")
z$drug1

## Using type = "cross-sectional" to check if individuals are exposed at a given time
## Note that we now only give one timepoint but then also need to supply the window argument
ind.periods2 <- copy(ind.periods)
ind.periods2[,time:=start][,c("start","end"):=NULL]
y=medicinMacro(drugs=list("drug1"=drug1,"drug2"=drug2),drugdb=lmdb,admdb=lpr,
               type="cross-sectional",datesdb=ind.periods2, window=365.25)
y$drug1
## Changing the window size of course affects our estimates:
y=medicinMacro(drugs=list("drug1"=drug1,"drug2"=drug2),drugdb=lmdb,admdb=lpr,
               type="cross-sectional",datesdb=ind.periods2, window=10)
y$drug1
## Above, we renamed the date variables before plugging it in, but we could also have used the
## datesdb.datevars argument:
y=medicinMacro(drugs=list("drug1"=drug1,"drug2"=drug2),drugdb=lmdb,admdb=lpr,
               type="cross-sectional",datesdb=ind.periods, window=365.25,
               datesdb.datevars="start")
y$drug1
```

`mm1` *Calculates whether patients are exposed or not at a given time*

Description

Part of the `medicinMacro`

Usage

```
mm1(
  drugs,
  drugdb,
  admdb,
  time.points,
  window,
  method = "default.doses",
  stash = 0,
```

```
  cap.values = TRUE,
  verbose = FALSE
)
```

Arguments

drugs	A named list of drugs. Each element of this list should be a list with the following elements: **atc** A vector of ATC codes which should match the components of the drug exactly. **maxdepot** The maximum total dose that a single patient can possibly stack. **period** A vector of dates to limit the period in which to estimate the daily dose. **prescriptionwindow** Default is 2 prescriptions. **doses** A named list with the elements `value`, `min`, `max` and `def`. Here `value` is a vector of strengths of one unit (e.g. pill) of the drug. The vector should have one such strength for each of the different packages that occur in the data. `min` is a vector of the same length as `value` where each element is the assumed minimum total dose that a patient can consume on one day. `max` is a vector of the same length as `value` where each element is the assumed maximum total dose that a patient can consume on one day. `def` is a vector of the same length as `value` where each element is the assumed default dose that an average patient would consume on one day. See examples.
drugdb	data.table with (subset of) medical drugs registry
admdb	data.table with (subset of) hospital admission registry. The data.table should be prepared such that it contains only overnight hospital stay (i.e., pattype=2) and non-overlapping hospital stay periods, i.e., as obtained with `getAdmLimits` (SAS-AKA: code-from-hell).
time.points	Time points
window	Only used when the `type = "cross-sectional"`. Specify how many days back from the given time point we should go to find purchases to estimate the daily dose.
method	Default doses by default
stash	Depot from prior prescriptions
cap.values	Overwrite a last period with previous
verbose	Logical. Set to `TRUE` to see messages about the progress.

Value

Exposed or not at a given time

Author(s)

Anders Munch NB: _potential_ purchases, becuase a ptt's might not purchase anything in the time frame – this gives a total.drug and drug.supply.days of 0

mm2 *Calculates exposure periods in a given time span*

Description

Part of the medicinMacro

Usage

```
mm2(
  drugs,
  drugdb,
  admdb,
  periods,
  method = "default.doses",
  stash = 0,
  cap.values = TRUE,
  pre.window = 365.25 * 2,
  print.time = FALSE,
  verbose = FALSE
)
```

Arguments

drugs — A named list of drugs. Each element of this list should be a list with the following elements:

atc A vector of ATC codes which should match the components of the drug exactly.

maxdepot The maximum total dose that a single patient can possibly stack.

period A vector of dates to limit the period in which to estimate the daily dose.

prescriptionwindow Default is 2 prescriptions.

doses A named list with the elements value, min, max and def. Here value is a vector of strengths of one unit (e.g. pill) of the drug. The vector should have one such strength for each of the different packages that occur in the data. min is a vector of the same length as value where each element is the assumed minimum total dose that a patient can consume on one day. max is a vector of the same length as value where each element is the assumed maximum total dose that a patient can consume on one day. def is a vector of the same length as value where each element is the assumed default dose that an average patient would consume on one day. See examples.

drugdb — data.table with (subset of) medical drugs registry

admdb — data.table with (subset of) hospital admission registry. The data.table should be prepared such that it contains only overnight hospital stay (i.e., pattype=2) and non-overlapping hospital stay periods, i.e., as obtained with getAdmLimits (SAS-AKA: code-from-hell).

periods	Time periods
method	Default dose if no other selected
stash	Depot
cap.values	Overwrite last period with previous
pre.window	Time window for prescriptions
print.time	Provide time printing
verbose	Logical. Set to TRUE to see messages about the progress.

Value

Exposure period in a given time span

Author(s)

Anders Munch Helper functions restricts x to lie within min and max Find smallest value in the grid, that is larger than x #' 1: Take out the relevant period and calculate non-hopsitalized days Probably should be a better way than this?: Restrict the data Make periods unique Only keep latest purchase before start Add end of each period (= beginning of the next so far) Add ptt.'s with no purhcases (which might still be hospitalized in the period!): Also use this place to add 0-exposure periods to patient with a gap from start to first purchase The following is for handling edge case with hospitalization after start date within first purchase period. If supplied, calculate non-hospitalized days and add the info TODO: maybe remove the hospitalization dates outside the periods before forming the cartesian product Add on the drug info Overwrite/special case for no purchasers: #' 2: Calculate end points and daily dose in different cases NB: non-purchases or total.drug==0 is might be given default doses here, but this is restored below. The last period should be based on the previous estimate, as we don't know, when the period ends (so can't give a proper daily dose estimate in this case) So overwrite previous estimate from last period. Handle case with only one period (set the estimate to the default dosis) NB: non-purchases or total.drug==0 might be given default doses here, but this is restored below. Clean up for the edge case of the pre.period: Re-estimate the total drug at start point by substracting the estimated amount of drug spend before start date. Then overwrite and drop .after.start info for the pre.periods. Also overwrite drug info for the cases with no drug left at the start date to not cause confusion. From this we calculate other summary measures: Set n.purchases==0 to estimated dose and days.supply = 0 (as they are capped or set to default above above): Also leave out period with 0 days (appears because of the ending dates):

---

patientProfile *Profile of common covariate combinations among patients*

Description

Profile of common covariate combinations among patients

Usage

```
patientProfile(dt, primary.cov, ...)
```

Arguments

dt	Data.table object where each row represents a patient and each row represents a covariates (or risk factor)
primary.cov	An integer, character, or factor vector. The primary covariate to group by and calculate proportions for (usually age)
...	Additional covariates to group by - e.g. gender, status of previous exposures/events, etc. Accepts vectors of integers, characters or factors

Details

The function is designed to show how many patients exists in the dataset that have each combination of risk factors.

Covariate combinations will that conatin 0 patients will return a 0 value for that covariate combination. However, this assumes that all possible outcomes for each variable are found in at least one patient in the dataset. For example, using "age" as a covariate, if the dataset ranges from 20 to 50 years olds, it is assumed that at least one patient from every interger age is present i.e. 20, 21, 22....50. If there are no patients aged 22, for example, then no covariate combinations with patients aged 22 will be created.

Value

Returns a data.table object where each row is a covariate combination. `N.in.primary.cov` records the number of patients in the data set with the specified primary covariate. E.g. using the "age" example from the "details" section, this column would record how many patients are 20, 21, 22....50. `prop.by.primary.cov` records the proportion of patients with the same primary covariate (`primary.cov`) with that parameter combination.

Author(s)

Matthew Phelps

Examples

```
n <- 1e2
set.seed(13)
x <- data.table::data.table(
  age = sample(20:35, size = n, replace = TRUE),
  v2 = sample(c("yes", "no"), n, replace = TRUE),
  v3 = sample(c("yes", "no"), n, replace = TRUE, prob = c(0.2, 0.8))
)
patientProfile(dt = x, primary.cov = "age", "v2", "v3")
```

poissonregression *Poisson regression by aggregated data using glm*

Description

Aggregates data and performs poisson regression using GLM.

Usage

```
poissonregression(formula, data, aggvars, byvars)
```

Arguments

formula	The formula as specified in glm.
data	dataset.
aggvars	vector of variables in data to be aggregated (summarized).
byvars	vector of variables in data to aggregate by.

Value

An object of class `c("glm", "lm")`: the fitted Poisson regression model after summing `aggvars` within groups defined by `byvars`. It contains coefficients, fitted values and model diagnostics and can be used with `summary` and `predict`.

poissonRisk *Predicting absolute risk from poisson regression model*

Description

Predicting absolute risks from poisson regression

Usage

```
poissonRisk(model, interval, newdata)
```

Arguments

model	glm object of fitted poisson model
interval	risk interval within time scale of the poisson model
newdata	values of the covariates in the poisson model for which the absolute risks are calculated.

Value

A data frame (or data.table when `newdata` is a data.table) with one row per row of `newdata`. The input covariates are followed by the coefficient summary from `lava::estimate`, containing the estimated absolute risk over `interval`, its standard error, confidence limits and test statistics.

Author(s)

Thomas A. Gerds <tag@biostat.ku.dk>, Regitze Kuhr Skals <r.skals@rn.dk>

---

rDateToSas *rDateToSas*

Description

rDateToSas

Usage

```
rDateToSas(rdate)
```

Arguments

`rdate`	Date object to be converted into a format SAS can read

Value

A character vector of the same length as `rdate`, with dates formatted as day, abbreviated month and year (for example, `"01Jan2015"` in an English locale), suitable for SAS date literals.

---

sasDateRange *sasDateRange*

Description

Helper function for making sas "Where" statements for use in importSAS(). Creates date range (inclusive) in SAS formate

Usage

```
sasDateRange(var.date, date.start, date.end, inclusive = TRUE)
```

Arguments

`var.date`	Name of the date variable in the SAS dataset
`date.start`	The start date of the desired date range in r date format
`date.end`	The end date of the desired date range in r date format
`inclusive`	Logical. If `TRUE` do greater or equal instead of greater.

Value

Function returns a string that can be passed to the "where" statement of importSAS. Alternatinvely it can be passed to makeSasWhereStatment function to be combined with further SAS where statements

Author(s)

Matthew Phelps

Examples

```
sasDateRange("inddto", as.Date("2015-01-01"), as.Date("2016-01-01"))
```

`sasSubstr` *sasSubstr*

Description

Helper function to construct a SAS 'where' statement to search through substrings in a SAS dataset via importSAS

Usage

```
sasSubstr(str.vec, target.var)
```

Arguments

`str.vec`	Character vector where each element is the start of the substring that should be searched for. Fx. c("DI05", "DI06) will search for any recrods that start with DI05 or DI06.
`target.var`	Character name of the variable in the SAS dataset that is to be searched

Details

The function allows a user to search for strings matching those in 'str.vec' in SAS datasets. Fx it can be used to find records in the LPR that match an ICD code

Value

Returns a string that can be passed to the "where" arguments of importSAS, or combined with other conditions in sasWhereConstructor

Author(s)

Matthew Phelps

See Also

importSAS

Examples

```
library(heaven)
x <- c("DI05", "DI06")
sasSubstr(x, target.var="A")
```

---

sasWhereConstructor *sasWhereConstructor*

Description

Simple helper function that combines multiple SAS where statements with "AND"

Usage

```
sasWhereConstructor(..., collapse = " AND ")
```

Arguments

...	A vector of valid SAS where statements.
collapse	Argment passed to paste0. default is " AND ".

Value

Returns a string that can be passed to the "where" argument of importSAS

Author(s)

Matthew Phelps

scanRawdata *Scan sas data files*

Description

Scan the contents of a directory which contains sas data files

Usage

```
scanRawdata(
  dir,
  which = c("pop", "doede", "diag", "lmdb", "opr"),
  verbose = FALSE
)
```

Arguments

dir	directory which contains the sas files
which	vector of expressions to match the file names
verbose	Logical. Print file contents and pass verbosity to importSAS.

Details

Scan the contents of a directory which contains sas data files

Value

A named list of data.tables describing the variables in each matching SAS file. Each table is returned by `importSAS(content = TRUE)`; list names are the SAS file names. An empty list is returned if no files match.

Author(s)

Thomas A. Gerds <tag@biostat.ku.dk>

See Also

listRawdata

Examples

```
## Not run:
# Requires SAS and access to the registry files at the supplied path.
scanRawdata("x:/Data/Rawdata_Hurtig/706818/",which=c("pop","lmdb"))

## End(Not run)
```

`simAdmissionData` *Admission data simulation function*

Description

Simulate admission data

Usage

```
simAdmissionData(n, m = 5, diagnoses, startDate = "1995-01-01")
```

Arguments

`n`	Number of patients
`m`	Maximal number of admission dates per patient
`diagnoses`	List of diagnoses. Defaults to all possible ICD10 codes: http://www.icd10data.com/ICD10CM/Codes
`startDate`	starting date

Details

Simulate admission data alike the Danish medical registry

Value

A data.table with one row per simulated admission, sorted by `pnr` and `inddto`. Columns are patient identifier `pnr`, record identifier `recnum`, admission and discharge dates `inddto` and `uddto` (class `Date`), a simulated `indexdate` (class `Date`), diagnosis code `diag` and patient type `pattype`.

Author(s)

Helene Charlotte Rytgaard <hely@sund.ku.dk>, Thomas Alexander Gerds <tag@biostat.ku.dk>

Examples

```
## a single subject
set.seed(1)
simAdmissionData(1)
## 7 subjects
set.seed(28)
simAdmissionData(7)
```

`simPop` *Population simulation*

Description

Simulate population with birthdate and gender

Usage

```
simPop(n, min.age = 20, max.age = 90, sex = 0.5, mortality = 0.05)
```

Arguments

`n`	Number of subjects
`min.age`	minimal age
`max.age`	maximal age
`sex`	gender distribution: value between 0 and 1
`mortality`	risk of death: value between 0 and 1

Details

Sample data for illustration, i.e., vignettes and example sections of the package.

Value

data.table with columns: pnr, sex, birthdate, status, doddate where doddate is the date of death for subjects with status = 1 and the current date for subjects with status = 0.

Author(s)

Thomas A. Gerds <tag@biostat.ku.dk>

See Also

simPrescriptionData, simAdmissionData

Examples

```
set.seed(7)
simPop(7)
```

simPrescriptionData *Prescription data simulation function*

Description

Simulate prescription data

Usage

```
simPrescriptionData(
  n,
  max.prescriptions = 37,
 packages = list(list(c(200, 30), c(400, 100), c(400, 300), c(500, 60)), list(c(750,
    100), c(750, 250), c(75, 500))),
  max.packages = 3,
  startDate = "1995-01-01"
)
```

Arguments

n	Number of patients
max.prescriptions	Maximal number of prescription dates per patient
packages	Named list of association lists. The names of the list are ATC codes, the value associates strengths of one unit and package sizes.
max.packages	Integer. Upper bound for number of packages purchased on a single date
startDate	start date

Details

Simulate prescription data alike the Danish medical registry

http://medinfo.dk/sks/brows.php http://www.medicinpriser.dk/ http://pro.medicin.dk/ https://www.whocc.no/atc_ddd_index/

A07: strength 200, number of pills 30 strength 400, number of pills 100 etc

A12: strength 750, number of pills 100 strength 750, number of pills 250 etc

The association list which is used as the default value # for the package argument:

```
list("A07"=list(c(200,30),c(400,100),c(400,300),c(500,60)), "A12B"=list(c(750,100),c(750,250),c(75,5
```

Value

A data.table with one row per simulated prescription, sorted by patient, ATC code and purchase date. Columns are patient identifier pnr, drug code atc, purchase date eksd (class Date), unit strength strnum, package size packsize and number of packages purchased apk.

Author(s)

Helene Charlotte Rytgaard <hely@sund.ku.dk>, Thomas Alexander Gerds <tag@biostat.ku.dk>

Examples

```
## see https://www.dst.dk/da/TilSalg/Forskningsservice/Data/Andre_Styrelser
## for details of the contents of lmdb

## a single subject
set.seed(1)
simPrescriptionData(1)
## 7 subjects
set.seed(29)
simPrescriptionData(7)
```

`sks_codes` *SKS codes*

Description

Several sections of the Sygehusvaesnets Klassifikationssystem (SKS). Retrieved from SKS in October 2018. Consists of B, D, K, M, and U codes.

Usage

```
sks_codes
```

Format

A data table with 27864 rows and 2 variables

kode A character vector of the code

tekst A description of the code (in Danish)

Source

http://medinfo.dk/sks/

`splitFromTo` *splitFromTo*

Description

splitFromTo is devised to split a series of records in multiple records based on entry/exit dates provided from another splitting guide dataset. The function can handle multiple different events with each type of event characterized by the variable "value".

The function is useful for analysis of data where temporary changes occur. These may be drugs treatment, pregnancy etc. For a situation of pregnancy for example it may be useful for analysis to split a person into three records representing time before, during and after pregnancy.

The input is first "base data" id/in/out (and other variables) that may already have been split by other functions. The other data is a sequence of id/from/to/value/name - that may represent multiple conditions with from/to and with "name" to distinguish.

When no value is provided by the splitting guide the "default" is used. If this is not provided the value is set to "0"

Usage

```
splitFromTo(indat,splitdat,invars,splitvars,default="0",datacheck=TRUE)
```

Arguments

indat	- base data with id, start, end and other data - possibly already split
splitdat	- Data with splitting guide - id/from/to/value/name
invars	- vector of column names for id/entry/exit - in that order, example: c("id","start","end")
splitvars	- vector of column names containing dates to split by. example: c("id","start","end","value","name") - must be in that order!
default	- Value given to intervals not given a value by the function.
datacheck	- This function would crash or produce incorrect results if input data have overlapping intervals or negative intervals in any of the two input datasets. Ts is checked and error produced by datacheck. Can be set to FALSE if data are checked otherwise

Details

The input to this function are two data.tables and two lists of the critical variables. The BASE data it the data to be split. This data must have a variable to identify participants and start/end times.

The other table (SPLITTINGUIDE) contains possibly multiple records for each participants with id/from/to/value/name - or just id/from/to

The program checks that intervals are not negative and that intervals for each "name" and each individual do not overlap. Violation stops the program with error. Overlaps may occur in real situations, but the user needs to make decisions regarding this prior to using this function.

It is required that the splitting guide contains at least one record. Missing data for key variables are not allowed and will cause errors.

Value

The function returns a new data table where records have been split according to the splitting guide dataset. Variables unrelated to the splitting are left unchanged.

Author(s)

Christian Torp-Pedersen

Examples

```
library(data.table)
dat <- data.table(id=c("A","A","B","B","C","D"),
                  start=as.Date(c(0,100,0,100,200,400),origin='1970-01-01'),
                  end=as.Date(c(100,200,100,200,300,500),origin='1970-01-01'))
split <- data.table (id=c("A","A","A","A","B","B","B","D","D"),
                   start=as.Date(c(0,50,25,150,110,150,400,300,500),origin='1970-01-01'),
                  end= as.Date(c(25,75,150,151,120,250,500,400,500),origin='1970-01-01'),
                     value=c(1,2,3,4,1,2,3,6,7),
                     name=c("d1","d1","d2","d2","d1","d1","d2","d3","d4"))
#Show the dataset:
dat[]
split[]
temp <- splitFromTo(dat # inddato with id/in/out/event
                    ,split # Data with id and dates
                    ,c("id","start","end") #names of id/in/out/event - in that order
                  ,c("id","start","end","value","name")) #Names var date-vars to split by
temp[]
```

---

splitSeq *splitSeq*

Description

splitSeq is a function which can split records according to a vector of selected times. At the outset each record has two variables representing start and end on a time scale. A vector of time points is supplied and each record is replaced by as many records as the number of times points from the vector that occurs in the interval. After splitting the variable representing end of time is replaced by the splitting-time and the next record has this splitting-time as the start of time variable.

This function is particularly useful to split variables according to variables that change continuously. Typical situations are age(e.g. 5 year periods), calender time (e.g. 2 year periods) and selected times after a situation of interest (e.g. fixed sized time periods after a starting date). The input is a data.table and splitting guide. The "base" data are the data to be split. They may contain much information, but the key is "id", "start" and "end". These describe the participant's id, start of time period and end of time period.

The other input is data to define splitvector and name. The splitvector may be a fixed vector (format="vector", e.g. a series of fixed calender dates) or a list of 3 integers defining start, end and interval to split by (format="seq", for a split on age between 20 and 80 by 5 years a splitvector could be defined as: splitvector <- c(20,80,5)*365.25 and provided to the function as a variable). "varname" is a name of a variable in the data.table the defines a value to be added to the splitvector. For the age split just used as an example it would be a variable containing the birth date. For a split after onset of a conditi on it should be the date of the condition and NA when the condition does not

occur. When no value should be added to the vector (e.g. split by calender time) "varname" should keep its default value of NULL.

On output a new variable with default name "value" defines the result of splitting. The variable can be renamed to a user defined name (e.g. value="myvalue"). This variable will contain zero when time is before the first value of the splitting vector (added the "varname") and then increased by one as each value of the splitting vector is reached.

Usage

```
splitSeq(indat,invars,varname=NULL,splitvector,format,value="value",
datacheck=TRUE)
```

Arguments

indat	base data with id, start, end and other data - possibly already split
invars	column names for id,entry,exit - in that order, example: c("id","start","end")
varname	name of variable to be added to vector
splitvector	A vector of calender times (integer). Splitvector is a sequence of fixed dates (or other time scala).
format	String with two possible values: • "vector" a series of fixed calender dates • "seq" see description
value	0 to the left of the vector, increase of 1 as each element of vector is passed
datacheck	- Checks that data are in appropriate format and that intervals are neihter negative or overlapping. Can be set to FALSE if checked elsewhere.

Details

The input must be data.table. This data.table is assumed already to be split by other functions with multiple records having identical participant id. The function extracts those variables necessary for splitting, splits by the provided vector and finally merges other variable onto the final result.

A note of caution: This function works with dates as integers. R has a de- fault origina of dates as 1 January 1970, but other programs have different default origins - and this includes SAS and Excell. It is therefor important for decent results that care is taken that all dates are defined similarly.

The output will always have the "next" period starting on the day where the last period ended. This is to ensure that period lengths are calculated pro- perly. The program will also allow periods of zero lengths which is a conse- quence when multiple splits are made on the same day. When there is an event on a period with zero length it is important to keep that period not to loose events for calculations. Whether other zero length records should be kept in calculations depend on context.

This function is identical to the lexisSeq function with the change that "event" is not considered.

Value

The function returns a new data table where records have been split according to the values in splitvector. Variables unrelated to the splitting are left unchanged.

Author(s)

Christian Torp-Pedersen

See Also

lexisSeq

Examples

```
library(data.table)

dat <- data.table(ptid=c("A","A","B","B","C","C","D","D"),
                  start=as.Date(c(0,100,0,100,0,100,0,100),origin="1970-01-01"),
                  end=as.Date(c(100,200,100,200,100,200,100,200),origin="1970-01-01"),
                Bdate=as.Date(c(-5000,-5000,-2000,-2000,0,0,100,100),origin="1970-01-01"))
#Example 1 - Splitting on a vector with 3 values to be added to "Bdate"
out <- splitSeq(indat=dat,invars=c("ptid","start","end"),
                varname="Bdate",
                splitvector=as.Date(c(0,150,5000),origin="1970-01-01"),
                format="vector")
out[]
#Example 2 - splitting on a from-to-by sequence with no adding (calender time?)
out2 <- splitSeq(indat=dat,invars=c("ptid","start","end"),
                 varname=NULL,splitvector=c(0,200,50),
                 format="seq",value="myvalue")
out2[]
```

---

splitTwo *splitTwo*

Description

This function can split records from a dataset in two records using information from a "splitting-guide". At start each record has variables representing stand and end of time - after the split time of a split record end at the time of a split one one record and represent the start of time on the next record.

The dataset to be split needs to have variables that define start and end of a time interval (start/end). Also necessary is a variable (id) that defines the relation between individuals and records in both the dataset to be split and the splitting guide.

The dataset to be split can have any other variable than those necessary for the split - these appear on each of the split records after the split.

The splitting-guide provides the information on which times to split records. This guide can be provide in a wide or a long format:

Wide format: This requires one record pr. individual that have dates to be split on. One column defines the same id as in the "base" table. The other columns contain dates for each condition where the split should occur. These column names will also appear in the output data, but on output the values are zero before the dates and 1 after. When dates are NA output has zero.

Long format: This requires one record per data where a possible split should occur. The columns should contain id, name of condition and the data to split on. Output is identical to that provide with the wide format.

Usage

```
splitTwo(indat,splitdat,invars,splitvars,format="wide",datacheck=TRUE)
```

Arguments

indat	A data.table or data.frame whose first 3 columns are: **id** Person identification variable such as PNR. The data may contain multiple lines per subject. **start** Start of time interval. Either a date or an integer/numeric. **end** End of time interval. Either in date format or given as numeric/integer.
splitdat	The splitting guide. A data.table which contains person specific information about the onset dates of comorbidities and other events. Wide format: **id** Person identification variable such as PNR. The data may contain multiple lines per subject. **Date 1** Either a date or an integer/numeric. Format must match that of the start and stop of arguments indat. The onset date of comorbidity 1 or other event. If integer/numeric it can be time since a baseline date on project specific scale (e.g., days or months). **Date 2** Either a date or an integer/numeric. The onset date of comorbidity 2 or other event. If integer/numeric it can be time since a baseline date on project specific scale (e.g., days or months). **Dat3 ...** Additional date variables to split on. Long format: **id** Person identification variable such as PNR. The data may contain multiple lines per subject. **Condition name** Character providing the variable name of condition. **Date** Either a date or an integer/numeric. The onset date of comorbidity or other event. If integer/numeric it can be time since a baseline date on project specific scale (e.g., days or months).
invars	vector of column names for id/entry/exit - in that order, example: c("id","start","end")
splitvars	For wide format: - vector of column names of columns containing dates to split by. example: c("date1","date2","date3","date4") For long format: - vector of the 3 columns in the data.table: id/name/date, example: c("id","name","date") The name of the id column must be the same in both datasets
format	- format of splitting guide - "wide" or "long"
datacheck	- This program may crash if intervals are overlapping or negative. Data checking produces an error in such cases. This can be omitted if the data have been checked by other means. For the splitting guide this options checks that there is only one entry for each variable to split by for each person identifier.

Details

The program checks that intervals are not negative and that intervals from one individual do not overlap.

It is required that the splitting guide contains at least one record. Missing data in the person id variables are not allowed and will cause errors.

A note of caution: This function works with dates as numeric. R has a default original of dates as 1 January 1970, but other programs have different default origins - and this includes SAS and Excel. It is therefor important for decent results that care is taken that all dates are defined similarly.

The output will always have the "next" period starting on the day where the last period ended. This is to ensure that period lengths are calculated pro- perly. The program will also allow periods of zero lengths which is a conse- quence when multiple splits are made on the same day.

Value

The function returns a new data table where records have been split according to the splitting guide dataset. Variables unrelated to the splitting are left unchanged. The names of columns from "splitvars" are also in output data, but now they have the value zero before the dates and 1 after.

Author(s)

Christian Torp-Pedersen

Examples

```
library(data.table)

dat <- data.table(pnr=c("123456","123456","234567","234567","345678","345678"
,"456789","456789"),
                start=as.Date(c(0,100,0,100,0,100,0,100),origin="1970-01-01"),
                end=as.Date(c(100,200,100,200,100,200,100,200),origin="1970-01-01"))

split <- data.table (pnr=c("123456","234567","345678","456789"),
como1.onset=as.integer(c(0,NA,49,50)), como2.onset=as.integer(c(25,75,49,49)),
como3.onset=as.integer(c(30,NA,49,48)), como4.onset=as.integer(c(50,49,49,47)))
#Show the datasets:
dat[]
split[]
splitTwo(dat # inddato with id/in/out
   ,split # Data with id and dates
   ,c("pnr","start","end") #names of id/in/out - in that order
   ,c("pnr","como1.onset","como2.onset","como3.onset","como4.onset"))
   #Names of date-vars to split by
# And with splittingguide in long format
splitvars <- c("como1.onset","como2.onset","como3.onset","como4.onset")
split <- data.table::melt(data=split,id.vars="pnr",measure.vars=splitvars,
  variable.name="name",value.name="value")
split[,value:=as.Date(value,origin="1970-01-01")]
split[]
split <- split[!is.na(value)] # remove missing values
splitTwo(dat # in-data with id/in/out
```

```
,split # Data with id/name/date
,c("pnr","start","end") #names of id/in/out - in that order
,c("pnr","name","value")
,format="long")
```

`standardize.prodlim` *Standardize absolute risks*

Description

Compute standardized absolute risks based on a stratified Kaplan-Meier or Aalen-Johansen estimate

Usage

```
standardize.prodlim(object, var1, var2, times, data, weights, ...)
```

Arguments

`object`	Object obtained with `prodlim` stratified with two categorical covariates.
`var1`	Name of one of the two categorical variables. The absolute risks are standardized within the levels of this variables to the distribution of the other categorical variable in the full data.
`var2`	Name of the other of the two categorical variables, i.e., the one on which the weights are based.
`times`	Compute absolute risks at these time points
`data`	Only if weights are not given. Data set in which to compute the weights for standardization as percentage in each of the levels of `var2`.
`weights`	A numeric vector of weights that sum to 1 and has length equal to the number of strata defined by `var2`. If omitted, weights are calculated using `data`.
`...`	not used

Details

Fit prodlim with two categorical covariates, say age and sex. Then standardize the absolute risk within each of the two gender to the age distribution in whole data set.

Value

A data.table with the grouping column named by `var1` and numeric columns named `risk<time>` for the requested time points. Each row is a cumulative-incidence estimate for a `var2` stratum multiplied by its standardization weight. Sum these contributions within each `var1` group to obtain the standardized absolute risks.

Author(s)

Thomas A. Gerds <tag@biostat.ku.dk>

Examples

```
if (requireNamespace("survival", quietly = TRUE) &&
    requireNamespace("prodlim", quietly = TRUE)) {
library(survival)
library(prodlim)
library(data.table)
data(pbc)
d <- na.omit(pbc[,c("time","status","trt","age","sex")])
d$agegroup <- cut(d$age,breaks=c(0,40,50,60,100))
fit <- prodlim(Hist(time,status==1)~trt+agegroup,data=d)
standardize.prodlim(fit,var1="trt",var2="agegroup",data=d,times=1000)
standardize.prodlim(fit,var1="trt",var2="agegroup",data=d,times=c(1000,1500))
}
```

`standardize.proportion` *Standardize proportions and absolute risks to a given age distribution*

Description

Standardize proportions and absolute risks to a given age distribution

Usage

```
standardize.proportion(
  vars,
  agevar,
  byvar,
  data,
  ref = 2,
  formula,
  cause,
  times,
  level = 0.95,
  ...
)
```

Arguments

`vars`	Name vector of categorical variables
`agevar`	Name of categorical age variable
`byvar`	Name of categorical strata variable
`data`	Data set
`ref`	For categorical variable that are not factors the level for which we calculate the standardized proportion
`formula`	Passed to prodlim for calculation of absolute risks

cause	In case of competing risks the cause of interest
times	The time point(s) for evaluation of absolute risks
level	Confidence level
...	Not (yet) used

Details

Standardize proportions and absolute risks to a given age distribution

Value

A data.table with one row per byvar stratum. Columns prop.<variable> contain age-standardized proportions as percentages. When formula is supplied, risk.<time> columns contain standardized absolute risks as percentages. Corresponding lower. and upper. columns contain confidence limits at the requested level.

Author(s)

Thomas A. Gerds <tag@biostat.ku.dk>

See Also

standardize.prodlim

Examples

```
set.seed(84)
d <- data.table::data.table(
  X1 = factor(sample(0:1, 120, replace = TRUE)),
  X2 = factor(sample(0:1, 120, replace = TRUE)),
  X4 = rep(c("a", "b"), each = 60),
  agegroups = factor(rep(c("young", "middle", "old"), 40)))
standardize.proportion(vars=c("X1","X2"),agevar="agegroups",byvar="X4",data=d)
```

---

standardize.rate *Standardize proportions and absolute risks to a given age distribution*

Description

Standardize proportions and absolute risks to a given age distribution

Usage

```
standardize.rate(
  x,
  age = "agegroups",
  exposure,
  by,
  standardize.to = "ref.level",
  data,
  method = "gamma",
  level = 0.95,
  crude = TRUE,
  ...
)
```

Arguments

x	List of names of variable names used to calculate the rates. Each element of the list contains the names of two variables in the dataset: the first variable contains the number of events and the second variable contains the number of subjects or person years.
age	Name of categorical age variable.
exposure	Name of the exposure variable for rate ratios.
by	Vector of names of further categorical strata variables
standardize.to	what population to use for standardization.
data	Data set which contains all the variables
method	Character. The method for calculating confidence intervals. If "gamma" use gamma distribution (see Fay et al.). If "wald" or "wald-log" use normal or log-normal approximation.
level	Confidence level
crude	Logical. If TRUE calculate crude rates too.
...	Not (yet) used

Details

Standardize proportions and absolute risks to a given age distribution

Value

A data.table with exposure group, estimated rate, rate.lower and rate.upper confidence limits, and ratio relative to the first exposure group, with ratio.lower and ratio.upper confidence limits. The x column identifies the event and denominator pair. Columns selected by by identify strata. When crude = TRUE, type distinguishes crude and standardized estimates; otherwise only standardized estimates are returned.

Author(s)

Thomas A. Gerds <tag@biostat.ku.dk>, Jeppe E. H. Madsen <jehm@sund.ku.dk>

References

Michael P Fay. Approximate confidence intervals for rate ratios from directly standardized rates with sparse data. Communications in Statistics- Theory and Methods, 28(9):2141–2160, 1999.

Niels Keiding, David Clayton, et al. Standardization and control for confounding in observational studies: a historical perspective. Statistical Science, 29(4):529–558, 2014.

See Also

standardize.prodlim standardize.proportion epitools::ageadjust.direct

Examples

```
library(data.table)
set.seed(84)
n=160
d <- data.table(e1=rpois(n,lambda=9),
                rt1=rpois(n,lambda=1880),
                e2=rpois(n,lambda=123),
                rt2=rpois(n,lambda=80000))
d[,agegroups:=factor(rep(c("40-50","45-50","50-55","55-60","60-65","65-70","70-75","75-80"),n/8))]
d[,sex:=factor(rep(c("f","m"),c(n/2,n/2)))]
d[,year:=rep(2001:2010,n/10)]
D=d[,list(e1=sum(e1),rt1=sum(rt1),e2=sum(e2),rt2=sum(rt2)),by=c("sex","agegroups")]
D[sex=="m",e1:=e1+rpois(.N,lambda=as.numeric(agegroups)*17)]
D[sex=="m",rt1:=rt1-rpois(.N,lambda=as.numeric(agegroups)*1600)]
standardize.rate(x=list(c("e1","rt1")),
                 age="agegroups",exposure="sex",data=D,standardize.to="f")
standardize.rate(x=list(c("e1","rt1")),
                 age="agegroups",exposure="sex",data=D,standardize.to="m")
standardize.rate(x=list(c("e1","rt1")),
                 age="agegroups",exposure="sex",data=D,standardize.to="mean")
standardize.rate(x=list("rate1"=c("e1","rt1"),"rate2"=c("e2","rt2")),
                 age="agegroups",exposure="sex",data=d,by="year")

# For more than two exposure groups, select the pair to compare.
d[,groups:=factor(rep(paste0("G",1:4),rep(n/4,4)))]
D=d[,list(e1=sum(e1),rt1=sum(rt1),e2=sum(e2),rt2=sum(rt2)),by=c("groups","agegroups")]
D[groups=="G3",e1:=e1+rpois(.N,lambda=as.numeric(agegroups)*17)]
D[groups=="G3",rt1:=rt1-rpois(.N,lambda=as.numeric(agegroups)*1600)]
pair <- D[groups %in% c("G3", "G4")]
pair[,groups:=droplevels(groups)]
standardize.rate(x=list(c("e1","rt1")),
                 age="agegroups",exposure="groups",
                 data=pair,standardize.to="mean")
```

`view_parquet` *View a Parquet File or Directory*

Description

Quickly inspect the contents of a Parquet file or a directory of Parquet files without reading the whole thing into memory. ‘action = "head"‘ shows the first ‘num_rows‘ rows (all columns), while ‘action = "str"‘ prints a transposed structure summary via [dplyr::glimpse()].

Usage

```
view_parquet(path, action = c("head", "str"), num_rows = 5, quiet = FALSE)
```

Arguments

path	Path to a Parquet file or a directory of Parquet files. Windows backslash paths are accepted and normalised automatically.
action	Either ‘"head"‘ to show the first few rows, or ‘"str"‘ for a structure summary. Defaults to ‘"head"‘.
num_rows	Number of rows to display when ‘action = "head"‘. Default 5.
quiet	Logical; if ‘TRUE‘, suppress both the path message and the data display. The collected data are still returned invisibly.

Details

The function uses [arrow::open_dataset()], which is lazy: only the row groups needed to satisfy the request are read from disk. This makes it fast even on very large files or multi-file directories.

Value

Invisibly returns the collected ‘tibble‘ (the head, or the sample used for the structure summary), so it can be captured if needed. Output is printed as a side effect unless ‘quiet = TRUE‘.

Examples

```
if (requireNamespace("arrow", quietly = TRUE) &&
    requireNamespace("dplyr", quietly = TRUE)) {
  path <- tempfile(fileext = ".parquet")
  arrow::write_parquet(data.frame(id = 1:3, value = c(4, 5, 6)), path)
  view_parquet(path, action = "head")
  view_parquet(path, action = "str")
  unlink(path)
}
```

`xrecepter` *wrapper for medicin macro*

Description

wrapper for medicin macro

Usage

```
xrecepter(
  drugdb,
  adm,
  atc,
  window = 5,
  value,
  min,
  max,
  def,
  maxdepot,
  period = c("'01jan1995'd", "'31dec2015'd"),
  name = "adrug",
  user,
  wd,
  sas.program = "/usr/local/bin/sas ",
  sas.switches,
  sas.runner,
  macro,
  save.tmp = FALSE,
  verbose = TRUE,
  server = "doob",
  remote = FALSE,
  remote.home,
  left_only = 1
)
```

Arguments

`drugdb`	drugdb data
`adm`	admission data
`atc`	atc kode specifying drugdb of interest
`window`	number of purchases that enter the calculations
`value`	is a vector of strengths of one unit (e.g., pill) of the drug. The vector should have one such strength for each of the different packages that occur in the data.
`min`	is a vector of the same length as `value` where each element is the assumed minimum total dosis that a patient can consume on one day. For example, if the

	value is 50 mg and the pills of this strength can be cut (in halves) the minimum total dosis is 25 mg.
max	is a vector of the same length as value where each element is the assumed maximum total dosis that a patient can consume on one day. For example, if the value is 50 mg and one can at most consume 4 pills a day the maximum total dosis is 200 mg.
def	is a vector of the same length as value where each element is the assumed default dosis that an average patient would consume on one day. For example, if the value is 50 mg and usually a patient would consume 2 pills the default is 100 mg.
maxdepot	Maximum number of units that a patient can stash
period	Vector of two dates that SAS can eat: c("'01jan1995'd","'31dec2015'd"). Can also be numeric values: number of days since 1960-01-01.
name	working name of output file (only relevant when save.tmp is set)
user	user name on remote (unix like) machine
wd	Writable parent directory for SAS scratch files. When omitted, tempdir() is used. A unique subdirectory is created for each call.
sas.program	path to sas program
sas.switches	On Linux this defaults to "-batch -nosplash -noenhancededitor -sysin".
sas.runner	How sas is invoked. On linux this defaults to "system" on any other system to "shell".
macro	Path to the SAS macro. Must be supplied explicitly.
save.tmp	if TRUE do not delete the temporary SAS files
verbose	like bla bla?
server	name of remote (unix like) machine
remote	if TRUE run on a remote (unix like) machine
remote.home	home folder on remote (unix like) machine
left_only	- when 1 only use prior prescription, when 0 use prior and future prescriptions.

Details

wrapper for medicin macro

Value

A data.table with one row per estimated exposure interval, including patient identifier pnr, dates firstday and lastday, daily dose and interval length exposure.days. Returns invisible NULL if SAS does not produce an output file.

Author(s)

Thomas A. Gerds <tag@biostat.ku.dk>

Examples

```
library(heaven)
## Not run:
# Requires SAS and a local copy of the external SAS medication macro.
 library(heaven)
library(data.table)
set.seed(05021992)
N=18
packs = list("R03AK11"=list(c(10,1)))
lmdb=simPrescriptionData(N,packages=packs,max.packages=1)
lpr=simAdmissionData(N)
## very simple data
lmdb <- lmdb[c(1,4,18)]
R03 = list(atc=c("R03AK11"),
           maxdepot=100,
           period=as.Date(c("1995-01-01", "2012-12-31")),
            prescriptionwindow=2,
           doses=list(value=c(5,10),
                      min = c(.5,.5),
                      max = c(2,2),
                      def = c(1,1)))
xrecepter(drugdb=lmdb,adm=lpr,window=5,remote=FALSE,wd=tempdir(),save.tmp=FALSE,
          value=rep(R03$doses$value,2),min=rep(R03$doses$min,2),max=rep(R03$doses$max,2),
           def=rep(R03$doses$def,2),maxdepot=100,period=c(12784,20089),atc="R03AK11",
           name="blaupill",sas.program="/usr/local/bin/sas",
           macro="/path/to/medicin_macro_version_31.sas",
           verbose=FALSE)

## End(Not run)
```

Index

∗ **datasets**
  atccodes, 3
  charlson.codes, 5
  diseasecode, 7
  edu_code, 10
  hypertensionATC, 18
  icdcodes, 20
  kommune_codes, 33
  sks_codes, 58

atccodes, 3
averageIncome, 4

charlson.codes, 5
charlsonIndex, 5
contentSAS *(*importSAS*)*, 22
createProject, 6

diseasecode, 7
dsr, 8

edu_code, 10
exposureMatch, 10

findCondition, 14

getAdmLimits, 16
groupAges, 17

hypertensionATC, 18
hypertensionMedication, 19

icdcodes, 20
importDREAM, 21
importSAS, 22
incidenceMatch, 27
innerMedicinMacro, 31

kommune_codes, 33
kommuneRegion, 32

lexisFromTo, 33
lexisSeq, 35
lexisTwo, 37
listRawdata, 39

matchReport, 40
medicinMacro, 41
mm1, 45
mm2, 47

patientProfile, 48
poissonregression, 50
poissonRisk, 50

rDateToSas, 51

sasDateRange, 51
sasSubstr, 52
sasWhereConstructor, 53
scanRawdata, 54
simAdmissionData, 55
simPop, 56
simPrescriptionData, 57
sks_codes, 58
splitFromTo, 58
splitSeq, 60
splitTwo, 62
standardize.prodlim, 65
standardize.proportion, 66
standardize.rate, 67

view_parquet, 69

xrecepter, 71