Package ‘sdcMicro’

May 16, 2018

Type Package

Title Statistical Disclosure Control Methods for Anonymization of Microdata and Risk Estimation

Version 5.3.0

Date 2018-05-15

Maintainer Matthias Templ <matthias.templ@gmail.com>

Description Data from statistical agencies and other institutions are mostly confidential. This package can be used for the generation of anonymized (micro)data, i.e. for the creation of public- and scientific-use files. In addition, various risk estimation methods are included. Note that the package includes a graphical user interface that allows to use various methods of this package.

LazyData TRUE

ByteCompile TRUE

LinkingTo Rcpp

Depends R (>= 2.10)

Suggests laeken, testthat

Imports utils, stats, graphics, car, carData, rmarkdown, knitr, data.table, xtable, robustbase, cluster, MASS, e1071, tools, Rcpp, methods, sets, ggplot2, shiny, haven, rhandsontable, DT, shinyBS, prettydoc, VIM(>= 4.7.0)

License GPL-2

URL https://github.com/sdcTools/sdcMicro

R topics documented:

'shuffle.R' 'sdcMicro-package.R' 'suda2.R'
'timeEstimation.R' 'topBotCoding.R' 'valTable.R' 'zzz.R'
'printFunctions.R' 'mafast.R' 'maG.R' 'sdcApp.R'
'show_sdcMicroObj.R'

RoxygenNote 6.0.1
VignetteBuilder knitr
NeedsCompilation yes
Author Matthias Templ [aut, cre],
Bernhard Meindl [aut],
Alexander Kowarik [aut]
Repository CRAN
Date/Publication 2018-05-16 15:28:41 UTC

R topics documented:

sdcMicro-package .................................................. 4
addGhostVars ....................................................... 8
addNoise ............................................................ 10
argus_microaggregation ............................................ 12
argus_rankswap .................................................... 13
calcRisks ............................................................ 13
casc1 ................................................................. 14
CASCreffmicrodata .................................................. 15
createNewID .......................................................... 16
dataGen .............................................................. 16
dRisk ................................................................. 18
dRiskRMD .............................................................. 19
dUtility ................................................................. 21
EIA ...................................................................... 22
extractManipData .................................................... 25
francdat ............................................................... 26
free1 ................................................................. 27
freq ................................................................. 28
freqCalc .............................................................. 29
generateStrata ..................................................... 31
get.sdcMicroObj ................................................... 31
globalRecode ...................................................... 32
groupAndRename ................................................... 34
importProblem ...................................................... 35
indivRisk ............................................................ 35
kAnon_violations ................................................... 37
LLmodGlobalRisk ................................................... 38
LocalRecProg ....................................................... 39
localSupp ............................................................ 41
localSuppression .................................................. 42
mafast ............................................................... 45
Statistical Disclosure Control (SDC) for the generation of protected microdata for researchers and for public use.

**Description**

This package includes all methods of the popular software mu-Argus plus several new methods. In comparison with mu-Argus the advantages of this package are that the results are fully reproducible even with the included GUI, that the package can be used in batch-mode from other software, that the functions can be used in a very flexible way, that everybody could look at the source code and that there are no time-consuming meta-data management is necessary. However, the user should have a detailed knowledge about SDC when applying the methods on data.

**Details**

The package is programmed using S4-classes and it comes with a well-defined class structure. The implemented graphical user interface (GUI) for microdata protection serves as an easy-to-handle tool for users who want to use the sdcMicro package for statistical disclosure control but are not used to the native R command line interface. In addition to that, interactions between objects which results from the anonymization process are provided within the GUI. This allows an automated recalculation and displaying information of the frequency counts, individual risk, information loss and data utility after each anonymization step. In addition to that, the code for every anonymization step carried out within the GUI is saved in a script which can then be easily modified and reloaded.

<table>
<thead>
<tr>
<th>Package</th>
<th>sdcMicro</th>
</tr>
</thead>
<tbody>
<tr>
<td>Type</td>
<td>Package</td>
</tr>
<tr>
<td>Version</td>
<td>2.5.9</td>
</tr>
<tr>
<td>Date</td>
<td>2009-07-22</td>
</tr>
<tr>
<td>License</td>
<td>GPL 2.0</td>
</tr>
</tbody>
</table>

**Author(s)**

Matthias Templ, Alexander Kowarik, Bernhard Meindl
Maintainer: Matthias Templ <templ@statistik.tuwien.ac.at>

**References**


sdcMicro-package

2010. doi: 10.1007/9781849962384_3


Examples

```r
## example from Capobianchi, Polettini and Lucarelli:
data(francdat)
f <- freqCalc(francdat, keyVars=c(2,4,5,6), w=8)
f
f$fk
f$Fk
## with missings:
x <- francdat
x[3,5] <- NA
x[4,2] <- x[4,4] <- NA
x[5,6] <- NA
x[6,2] <- NA
f2 <- freqCalc(x, keyVars=c(2,4,5,6), w=8)
f2$Fk
## individual risk calculation:
indivf <- indivRisk(f)
indivf$rk
## Local Suppression
localS <- localSupp(f, keyVar=2, threshold=0.25)
f2 <- freqCalc(localS$freqCalc, keyVars=c(2,4,5,6), w=8)
indivf2 <- indivRisk(f2)
indivf2$rk

## select another keyVar and run localSupp once again,
#if you think the table is not fully protected
data(free1)
free1 <- as.data.frame(free1)
f <- freqCalc(free1, keyVars=1:3, w=30)
ind <- indivRisk(f)
## and now you can use the interactive plot for individual risk objects:
## plot(ind)

## example from Capobianchi, Polettini and Lucarelli:
data(francdat)
l1 <- localSuppression(francdat, keyVars=c(2,4,5,6), importance=c(1,3,2,4))
l1
```
```r
l1x
l2 <- localSuppression(frencdat, keyVars=c(2,4,5,6), k=2)
l3 <- localSuppression(frencdat, keyVars=c(2,4,5,6), k=4)

## Data from mu-Argus:
## Global recoding:
data(free1)
free1 <- as.data.frame(free1)
free1[, "AGE"] <- globalRecode(free1[,"AGE"], c(1,9,19,29,39,49,59,69,100), labels=1:8)

## Top coding:
topBotCoding(free1[,"DEBTS"], value=9000, replacement=9100, kind="top")

## Numerical Rank Swapping:
## do not use the mu-Argus test data set (free1)
# since the numerical variables are (probably) faked.
data(tarragona)
Tarragonal <- rankSwap(tarragona, P=10)

## Microaggregation:
m1 <- microaggregation(tarragona, method="onedims", aggr=3)
m2 <- microaggregation(tarragona, method="pca", aggr=3)
# summary(m1)
## approx. 1 minute computation time
## valTable(tarragona, method=c("simple","onedims","pca"))
data(microData)
microData <- as.data.frame(microData)
m1 <- microaggregation(microData, method="mdav")
x <- m1$x  ### fix me
summary(m1)
plotMicro(m1, 1, which.plot=1) # too less observations...
data(free1)
free1 <- as.data.frame(free1)
plotMicro(microaggregation(free1[,31:34], method="ondenims"), 1, which.plot=1)

## disclosure risk (interval) and data utility:
m1 <- microaggregation(tarragona, method="ondenims", aggr=3)
dRisk(obj=tarragona, xm=m1$mx)
dRisk(obj=tarragona, xm=m2$mx)
dUtility(obj=tarragona, xm=m1$mx)
dUtility(obj=tarragona, xm=m2$mx)

## S4 class code for Adding Noise methods will be included
# in the next version of sdcMicro.

## Fast generation of synthetic data with aprox.
# the same covariance matrix as the original one.
data(mtcars)
cov(mtcars[,4:6])
```
```r
## PRAM

```
```
addGhostVars

---

```r
head(sdc@risk$individual)
sdc@risk$global

### microaggregation
sdc <- undolast(sdc)
head(get.sdcmicroObj(sdc, type="manipNumVars"))
sdc <- microaggregation(sdc)
head(get.sdcmicroObj(sdc, type="manipNumVars"))

### pram
sdc <- undolast(sdc)
head(sdc@risk$individual)
sdc@risk$global
sdc <- pram(sdc, keyVar="water")
head(sdc@risk$individual)
sdc@risk$global

### rankSwap
sdc <- undolast(sdc)
head(sdc@risk$individual)
sdc@risk$global
head(get.sdcmicroObj(sdc, type="manipNumVars"))
sdc <- rankSwap(sdc)
head(get.sdcmicroObj(sdc, type="manipNumVars"))
head(sdc@risk$individual)
sdc@risk$global

### suda2
sdc <- suda2(sdc)
sdc@risk$suda2

### topBotCoding
head(get.sdcmicroObj(sdc, type="manipNumVars"))
sdc@risk$numeric
sdc <- topBotCoding(sdc, value=60000000, replacement=62000000, column="income")
head(get.sdcmicroObj(sdc, type="manipNumVars"))
sdc@risk$numeric

### LocalRecProg
data(testdata2)
sdc <- createSdcObj(testdata2,
  keyVars=c("urbrur", "roof", "walls", "water", "sex", "relat"))
sdc@risk$global
sdc <- LocalRecProg(sdc)
sdc@risk$global

### llmodGlobalRisk
sdc <- undolast(sdc)
sdc <- llmodGlobalRisk(sdc, inclProb=0.001)
sdc@risk$model

# End(Not run)
```

---

addGhostVars
**Description**

specify variables that are linked to a key variable. This results in all suppressions of the key-variable being also applied on the corresponding 'ghost'-variables.

**Usage**

```r
addGhostVars(obj, keyVar, ghostVars)
```

**Arguments**

- **obj**: an object of class `sdcmicr0Obj-class`
- **keyVar**: character-vector of length 1 referring to a categorical key variable within `obj`.
- **ghostVars**: a character vector specifying variables that are linked to `keyVar`. Variables listed here must not be listed in either slots `@keyvars`, `@numvars`, `@pramvars`, `@weightvar`, `@hhid` or `@strataVar` in `obj`.

**Value**

a modified `sdcmicr0Obj-class` object.

**Author(s)**

Bernhard Meindl

**References**


**Examples**

```r
data(testdata2)
sdc <- createSdcObj(testdata2,
  keyVars=c('urbrur','roof','walls','water','electcon','relat','sex'),
  numVars=c('expend','income','savings'), w='sampling_weight')
## we want to link the anonymization status of key variable 'urbrur' to 'hhcivil'
sdc <- addGhostVars(sdc, keyVar="urbrur", ghostVars=c("hhcivil"))
## we want to link the anonymization status of key variable 'roof' to 'represent'
sdc <- addGhostVars(sdc, keyVar="roof", ghostVars=c("represent"))
```
addNoise

Adding noise to perturb data

Description

Various methods for adding noise to perturb continuous scaled variables.

Usage

addNoise(obj, variables = NULL, noise = 150, method = "additive", ...)

Arguments

- `obj` either a `data.frame` or a `sdcmicroObj-class` that should be perturbed
- `variables` vector with names of variables that should be perturbed
- `noise` amount of noise (in percentages)
- `...` see possible arguments below

Details

If `obj` is of class `sdcmicroObj-class`, all continuous key variables are selected per default. If `obj` is of class “data.frame” or “matrix”, the continuous variables have to be specified.

Method ‘additive’ adds noise completely at random to each variable depending on its size and standard deviation. ‘correlated’ and method ‘correlated2’ adds noise and preserves the covariances as described in R. Brand (2001) or in the reference given below. Method ‘restr’ takes the sample size into account when adding noise. Method ‘ROMM’ is an implementation of the algorithm ROMM (Random Orthogonalized Matrix Masking) (Fienberg, 2004). Method ‘outdect’ adds noise only to outliers. The outliers are indentified with univariate and robust multivariate procedures based on a robust mahalanobis distances calculated by the MCD estimator.

Value

If `obj` was of class `sdcmicroObj-class` the corresponding slots are filled, like manipNumVars, risk and utility.

If `obj` was of class “data.frame” or “matrix” an object of class “micro” with following entities is returned:

- `x` the original data
- `xm` the modified (perturbed) data
- `method` method used for perturbation
- `noise` amount of noise
Author(s)
Matthias Templ and Bernhard Meindl

References


See Also
sdcmicroObj-class, summary.micro

Examples

data(Tarragona)
a1 <- addNoise(Tarragona)
a1

data(testdata)
testdata[, c('expend','income','savings')] <-
addNoise(testdata[,c('expend','income','savings')])$xm

## for objects of class sdcMicroObj:
data(testdata2)
sdc <- createSdcObj(testdata2, keyVars=c('urbrur','roof','walls','water','electcon','relat','sex'), numVars=c('expend','income','savings'), w='sampling_weight')
argus_microaggregation

Description

calls microaggregation code from mu-argus. In case only one variable should be microaggregated
and useOptimal is TRUE, Hansen-Mukherjee polynomial exact method is applied. In any other
case, the Mateo-Domingo method is used.

Usage

argus_microaggregation(df, k, useOptimal = FALSE)

Arguments

df a data.frame with only numerical columns
k required group size
useOptimal (logical) should optimal microaggregation be applied (only possible in case of
one variable)

Value

a list with two elements

• original: the originally provided input data
• microaggregated: the microaggregated data.frame

See Also


Examples

mat <- matrix(sample(1:100, 50, replace=TRUE), nrow=10, ncol=5)
df <- as.data.frame(mat)
res <- argus_microaggregation(df, k=5, useOptimal=FALSE)

sdc <- addNoise(sdc)
argus_rankswap

Description

argus_rankswap

Usage

argus_rankswap(df, perc)

Arguments

df  
a data.frame with only numerical columns
perc  
a number defining the swapping percentage

Value

a list with two elements

• original: the originally provided input data
• swapped: the data.frame containing the swapped values

See Also


Examples

mat <- matrix(sample(1:100, 50, replace=TRUE), nrow=10, ncol=5)
df <- as.data.frame(mat)
res <- argus_rankswap(df, perc=10)

calcRisks

Recompute Risk and Frequencies for a sdcMicroObj

Description

Recomputation of Risk should be done after manual changing the content of an object of class sdcMicroObj-class

Usage

calcRisks(obj, ...)

Arguments

obj an object of class \texttt{sdcMicroObj-class}

... no arguments at the moment

Details

By applying this function, the disclosure risk is re-estimated and the corresponding slots of an object of class \texttt{sdcMicroObj-class} are updated. This function mostly used internally to automatically update the risk after an sdc method is applied.

See Also

\texttt{sdcMicroObj-class}

Examples

data(testdata2)
sdc <- createSdcObj(testdata2,
  keyVars=c('urbrur','roof','walls','water','electcon','relat','sex'),
  numVars=c('expend','income','savings'), w='sampling_weight')
sdc <- calcRisks(sdc)

casc1

\textit{Small Artificial Data set}

Description

Small Toy Example Data set which was used by Sanz-Mateo et.al.

Format

The format is: int [1:13, 1:7] 10 12 17 21 9 12 12 14 13 15 ... - attr(*, "dimnames")=List of 2 ..$ : 
chr [1:13] "1" "2" "3" "4" ... ..$ : chr [1:7] "1" "2" "3" "4" ...

Examples

data(casc1)
casc1
Description

This test data set was obtained on July 27, 2000 using the public use Data Extraction System of the U.S. Bureau of the Census.

Format

A data frame sampled from year 1995 with 1080 observations on the following 13 variables.

- **AFNLWGT**: Final weight (2 implied decimal places)
- **AGI**: Adjusted gross income
- **EMCONTRB**: Employer contribution for hlth insurance
- **FEDTAX**: Federal income tax liability
- **PTOTVAL**: Total person income
- **STATETAX**: State income tax liability
- **TAXINC**: Taxable income amount
- **POTHVAL**: Total other persons income
- **INTVAL**: Amt of interest income
- **PEARNVAL**: Total person earnings
- **FICA**: Soc. sec. retirement payroll deduction
- **WSALVAL**: Amount: Total Wage and salary
- **ERNVAL**: Business or Farm net earnings

Source

Public use file from the CASC project. More information on this test data can be found in the paper listed below.

References


Examples

data(CASCrefmicrodata)
str(CASCrefmicrodata)
createNewID  
*Creates new randomized IDs*

**Description**

This is useful if the record IDs consist, for example, of a geo identifier and the household line number. This method can be used to create new, random IDs that cannot be reconstructed.

**Usage**

```r
createNewID(obj, newID, withinVar)
```

**Arguments**

- `obj`  
an *sdcmicroobj-class*-object
- `newID`  
a character specifying the desired variable name of the new ID
- `withinVar`  
if not NULL a character vector specifying a variable (e.g. an existing household ID) which will be used when calculating the new IDs. If specified, the same IDs will be assigned to the same values of the given variable.

**Value**

an *sdcmicroobj-class*-object with updated slot origData

---

dataGen  
*Fast generation of synthetic data*

**Description**

Fast generation of (primitive) synthetic multivariate normal data.

**Usage**

```r
dataGen(obj, ...)```

**Arguments**

- `obj`  
an *sdcmicroobj-class*-object or a *data.frame*
- `...`  
see possible arguments below

- `n`: amount of observations for the generated data, defaults to 200
- `use`: howto compute covariances in case of missing values, see also argument `use` in `cov`. The default choice is `everything`, other possible choices are `all.obs`, `complete.obs`, `na.or.complete` or `pairwise.complete.obs`.
Details
Uses the cholesky decomposition to generate synthetic data with approx. the same means and covariances. For details see at the reference.

Value
the generated synthetic data.

Note
With this method only multivariate normal distributed data with approximatively the same covariance as the original data can be generated without reflecting the distribution of real complex data, which are, in general, not follows a multivariate normal distribution.

Author(s)
Matthias Templ

References

See Also
sdcmicroObj-class, shuffle

Examples
```r
data(mtcars)
cov(mtcars[,4:6])
cov(dataGen(mtcars[,4:6]))
pairs(mtcars[,4:6])
pairs(dataGen(mtcars[,4:6]))

## for objects of class sdcMicro:
data(testdata2)
sdc <- createSdcObj(testdata2,
  keyVars=c('urbrur','roof','walls','water','electcon','relat','sex'),
  numVars=c('expend','income','savings'), w='sampling_weight')
sdc <- dataGen(sdc)
```
**dRisk**

*overall disclosure risk*

**Description**

Distance-based disclosure risk estimation via standard deviation-based intervals around observations.

**Usage**

```
dRisk(obj, ...)```

**Arguments**

- `obj` a data.frame or object of class `sdcMicroObj-class`
- `...` possible arguments are:
  - `xm`: perturbed data
  - `k`: percentage of the standard deviation

**Details**

An interval (based on the standard deviation) is built around each value of the perturbed value. Then we look if the original values lay in these intervals or not. With parameter `k` one can enlarge or down scale the interval.

**Value**

The disclosure risk or/and the modified `sdcMicroObj-class`

**Author(s)**

Matthias Templ

**References**


**See Also**

`dUtility`
Examples

data(free1)
free1 <- as.data.frame(free1)
m1 <- microaggregation(free1[, 31:34], method="onedims", aggr=3)
m2 <- microaggregation(free1[, 31:34], method="pca", aggr=3)
dRisk(obj=free1[, 31:34], xm=m1$mx)
dRisk(obj=free1[, 31:34], xm=m2$mx)
dUtility(obj=free1[, 31:34], xm=m1$mx)
dUtility(obj=free1[, 31:34], xm=m2$mx)

## for objects of class sdcMicro:
data(testdata2)
sdc <- createSdcObj(testdata2,
  keyVars=c('urbrur', 'roof', 'walls', 'water', 'electcon', 'relat', 'sex'),
  numVars=c('expend', 'income', 'savings'), w='sampling_weight')
## this is already made internally: sdc <- dRisk(sdc)
## and already stored in sdc

dRiskRMD

RMD based disclosure risk

Description

Distance-based disclosure risk estimation via robust Mahalanobis Distances.

Usage

dRiskRMD(obj, ...)

Arguments

obj an sdcMicroObj-class-object or a data.frame

... see possible arguments below

  • xm masked data
  • kweight for adjusting the influence of the robust Mahalanobis distances, i.e. to increase or decrease each of the disclosure risk intervals.
  • k2parameter for method RMDID2 to choose a small interval around each masked observation.

Details

This method is an extension of method SDID because it accounts for the “outlyingness” of each observations. This is a quite natural approach since outliers do have a higher risk of re-identification and therefore these outliers should have larger disclosure risk intervals as observations in the center of the data cloud.

The algorithm works as follows:
1. Robust Mahalanobis distances are estimated in order to get a robust multivariate distance for each observation.

2. Intervals are estimated for each observation around every data point of the original data points where the length of the interval is defined/weighted by the squared robust Mahalanobis distance and the parameter $k$. The higher the RMD of an observation the larger the interval.

3. Check if the corresponding masked values fall into the intervals around the original values or not. If the value of the corresponding observation is within such an interval the whole observation is considered unsafe. So, we get a whole vector indicating which observation is save or not, and we are finished already when using method RMDID1).

4. For method RMDID1w: we return the weighted (via RMD) vector of disclosure risk.

5. For method RMDID2: whenever an observation is considered unsafe it is checked if $m$ other observations from the masked data are very close (defined by a parameter $k2$ for the length of the intervals as for SDID or RSDID) to such an unsafe observation from the masked data, using Euclidean distances. If more than $m$ points are in such a small interval, we conclude that this observation is “safe”.

Value

The disclosure risk or the modified \texttt{sdcMicroObj-class}

<table>
<thead>
<tr>
<th>Value</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td>risk1</td>
<td>percentage of sensitive observations according to method RMDID1.</td>
</tr>
<tr>
<td>risk2</td>
<td>standardized version of risk1</td>
</tr>
<tr>
<td>wrisk1</td>
<td>amount of sensitive observations according to RMDID1 weighted by their corresponding robust Mahalanobis distances.</td>
</tr>
<tr>
<td>wrisk2</td>
<td>RMDID2 measure</td>
</tr>
<tr>
<td>indexRisk1</td>
<td>index of observations with high risk according to risk1 measure</td>
</tr>
<tr>
<td>indexRisk2</td>
<td>index of observations with high risk according to wrisk2 measure</td>
</tr>
</tbody>
</table>

Author(s)

Matthias Templ

References


See Also

\texttt{dRisk}
dUtility

Examples

data(Tarragona)
x <- Tarragona[, 5:7]
y <- addNoise(x)$xm
dRiskRMD(x, xm=y)
dRisk(x, xm=y)

data(testdata2)
sdc <- createSdcObj(testdata2,
  keyVars=c('urbanr', 'roof', 'walls', 'water', 'electcon', 'relat', 'sex'),
  numVars=c('expend', 'income', 'savings'), w='sampling_weight')
## this is already made internally:
## sdc <- dRiskRMD(sdc)
## and already stored in sdc

---

dUtility  data utility

Description

IL1s data utility.

Usage

dUtility(obj, ...)

Arguments

obj  original data or object of class sdcMicroObj-class

...  see arguments below

  • xm: perturbed data
  • method: method IL1, IL1s or eigen. More methods are implemented in
    summary.micro()

Details

The standardised distances of the perturbed data values to the original ones are measured. Measure
IL1 measures the distances between the original values and the perturbed ones, scaled by the stan-
dard deviation. Method ‘eigen’ and ‘robeigen’ compares the eigenvalues and robust eigenvalues
form the original data and the perturbed data.

Value

data utility or modified entry for data utility the sdcMicroObj-class.

Author(s)

Matthias Templ
References

for IL1 and IL1s: see http://vneumann.etse.urv.es/webCrises/publications/isijcr/lncs3050Outlier.pdf,


See Also
dRisk, dRiskRMD

Examples

data(free1)
free1 <- as.data.frame(free1)
m1 <- microaggregation(free1[, 31:34], method="onedims", aggr=3)
m2 <- microaggregation(free1[, 31:34], method="pca", aggr=3)

# for objects of class sdcMicro:
data(testdata2)
sdc <- createSdcObj(testdata2,
keyVars=c("urbbr",'roof','walls','water','electcon','relat','sex'),
numVars=c("expend","income","savings"), w='sampling_weight')

# this is already made internally:
sdc <- dUtility(sdc)

EIA data set

Description

Data set obtained from the U.S. Energy Information Authority.
**Format**

A data frame with 4092 observations on the following 15 variables.

**UTILITYID** UNIQUE UTILITY IDENTIFICATION NUMBER


**STATE** STATE FOR WHICH THE UTILITY IS REPORTING. A factor with levels AK AL AR AZ CA CO CT DC DE FL GA HI IA ID IL IN KS KY LA MA MD ME MI MN MO MS MT NC ND NE NH NJ NM NV NY OH OK OR PA RI SC SD TN TX UT VA VT WA WI WV WY

**YEAR** REPORTING YEAR FOR THE DATA

**MONTH** REPORTING MONTH FOR THE DATA

**RESREVENUE** REVENUE FROM SALES TO RESIDENTIAL CONSUMERS

**RESSALES** SALES TO RESIDENTIAL CONSUMERS

**COMREVENUE** REVENUE FROM SALES TO COMMERCIAL CONSUMERS

**COMSALES** SALES TO COMMERCIAL CONSUMERS

**INDREVENUE** REVENUE FROM SALES TO INDUSTRIAL CONSUMERS

**INDSALES** SALES TO INDUSTRIAL CONSUMERS
**extractManipData**

**OTHREVENUE**  REVENUE FROM SALES TO OTHER CONSUMERS  
**OTHRSALES**  SALES TO OTHER CONSUMERS  
**TOTREVENUE**  REVENUE FROM SALES TO ALL CONSUMERS  
**TOTSALS**  SALES TO ALL CONSUMERS

**Source**

Public use file from the CASC project.

**References**


**Examples**

data(EIA)  
head(EIA)

---

**extractManipData**  
Remove certain variables from the data set inside a sdc object.

**Description**

Extract the manipulated data from an object of class `sdcmicroobj-class`.

**Usage**

```r
extractManipData(obj, ignoreKeyVars = FALSE, ignorePramVars = FALSE, ignoreNumVars = FALSE, ignoreGhostVars = FALSE, ignoreStrataVar = FALSE, randomizeRecords = "no")
```

**Arguments**

- `obj`  
  object of class `sdcmicroobj-class`

- `ignoreKeyVars`  
  If manipulated KeyVariables should be returned or the unchanged original variables

- `ignorePramVars`  
  if manipulated PramVariables should be returned or the unchanged original variables

- `ignoreNumVars`  
  if manipulated NumericVariables should be returned or the unchanged original variables

- `ignoreGhostVars`  
  if manipulated Ghost (linked) Variables should be returned or the unchanged original variables
ignoreStrataVar

if manipulated StrataVariables should be returned or the unchanged original variables

randomizeRecords

(logical) specifies, if the output records should be randomized. The following options are possible:

- 'no' default, no randomization takes place
- 'simple' records are just randomly swapped.
- 'byHH' if slot 'hhId' is not NULL, the clusters defined by this variable are randomized across the dataset. If slot 'hhId' is NULL, the records or the dataset are randomly changed.
- 'withinHH' if slot 'hhId' is not NULL, the clusters defined by this variable are randomized across the dataset and additionally, the order of records within the clusters are also randomly changed. If slot 'hhId' is NULL, the records or the dataset are randomly changed.

Value

a data.frame containing the anonymized data set

Author(s)

Alexander Kowarik, Bernhard Meindl

Examples

```r
## for objects of class sdcMicro:
data(testdata2)
sdc <- createSdcObj(testdata,
  keyVars=c('urbrur','roof'),
  numVars=c('expend','income','savings'), w='sampling_weight')
sdc <- removeDirectID(sdc, var="age")
dataM <- extractManipData(sdc)
```

francdat  

*data from the casc project*

Description

Small synthetic data from Capobianchi, Polettini, Lucarelli

Format

A data frame with 8 observations on the following 8 variables.

Num1 a numeric vector

Key1 Key variable 1. A numeric vector
Num2  a numeric vector
Key2  Key variable 2. A numeric vector
Key3  Key variable 3. A numeric vector
Key4  Key variable 4. A numeric vector
Num3  a numeric vector
w    The weight vector. A numeric vector

Details
This data set is very similar to that one which are used by the authors of the paper given below. We need this data set only for demonstration effect, i.e. that the package provides the same results as their software.

Source

Examples

data(francdat)
francdat

Demo data set from mu-Argus

Description
The public use toy demo data set from the mu-Argus software for SDC.

Format
The format is: num [1:4000, 1:34] 36 36 36 36 36 36 36 36 36 36 ... - attr(*, "dimnames")=List of 2 ..$ : NULL ..$ : chr [1:34] "REGION" "SEX" "AGE" "MARSTAT" ...

Details
Please, see at the link given below. Please note, that the correlation structure of the data is not very realistic, especially concerning the continuous scaled variables which drawn independently from are a multivariate uniform distribution.

Source
Public use file from the CASC project.
Examples

data(freq)
head(freq)

---

freq     Freq
---

Description

Extract sample frequency counts (fk) or estimated population frequency counts (Fk)

Usage

freq(obj, type = "fk")

Arguments

- **obj**: an `sdcmicroobj-class`-object
- **type**: either 'fk' or 'FK'

Value

A vector containing sample frequencies or weighted frequencies

Examples

data(testdata)
sdc <- createSdcObj(testdata,
  keyVars=c('urbrur','roof','walls','relat','sex'),
  pramVars=c('water','electcon'),
  numVars=c('expend','income','savings'), w='sampling_weight')
head(freq(sdc, type="fk"))
head(freq(sdc, type="Fk"))
freqCalc

Frequencies calculation for risk estimation

Description

Computation and estimation of the sample and population frequency counts.

Usage

freqCalc(x, keyVars, w = NULL, alpha = 1)

Arguments

- **x**: data frame or matrix
- **keyVars**: key variables
- **w**: column index of the weight variable. Should be set to NULL if one deal with a population.
- **alpha**: numeric value between 0 and 1 specifying how much keys that contain missing values (NAs) should contribute to the calculation of \( f_k \) and \( F_k \). For the default value of 1, nothing changes with respect to the implementation in prior versions. Each wildcard-match would be counted while for \( \alpha \neq 0 \) keys with missing values would be basically ignored.

Details

The function considers the case of missing values in the data. A missing value stands for any of the possible categories of the variable considered. It is possible to apply this function to large data sets with many (categorical) key variables, since the computation is done in C.

freqCalc() does not support sdcMicro S4 class objects.

Value

Object from class freqCalc.

freqCalc: data set
keyVars: variables used for frequency calculation
w: index of weight vector. NULL if you do not have a sample.
alpha: value of parameter alpha
fk: the frequency of equal observations in the key variables subset sample given for each observation.
Fk: estimated frequency in the population
n1: number of observations with \( f_k = 1 \)
n2: number of observations with \( f_k = 2 \)
Author(s)
Bernhard Meindl

References


See Also

`indivRisk`, `measure_risk`

Examples

data(francdat)
f <- freqCalc(francdat, keyVars=c(2,4,5,6), w=8)
f$f$freqCalc
f$f$k
f$f$K
## with missings:
x <- francdat
x[3,5] <- NA
x[4,2] <- x[4,4] <- NA
x[5,6] <- NA
x[6,2] <- NA
f2 <- freqCalc(x, keyVars=c(2,4,5,6), w=8)
cbind(f2$f$k, f2$f$K)

## test parameter 'alpha'
f3a <- freqCalc(x, keyVars=c(2,4,5,6), w=8, alpha=1)
f3b <- freqCalc(x, keyVars=c(2,4,5,6), w=8, alpha=0.5)
f3c <- freqCalc(x, keyVars=c(2,4,5,6), w=8, alpha=0.1)
data.frame(fka=f3a$f$k, fkb=f3b$f$k, fkc=f3c$f$k)
data.frame(Fka=f3a$F$k, Fkb=f3b$F$k, Fkc=f3c$F$k)
### generateStrata

*Generate one strata variable from multiple factors*

**Description**

For strata defined by multiple variables (e.g. sex, age, country) one combined variable is generated.

**Usage**

```r
generateStrata(df, stratavars, name)
```

**Arguments**

- `df`: a data.frame
- `stratavars`: character vector with variable name
- `name`: name of the newly generated variable

**Value**

The original data set with one new column.

**Author(s)**

Alexander Kowarik

**Examples**

```r
x <- testdata
dx <- generateStrata(x, c("sex", "urbrur", "strataIDvar"))
head(x)
```

---

### get.sdcMicroObj

*get.sdcMicroObj*

**Description**

extract information from `sdCMicroObj-class`-objects depending on argument type

**Usage**

```r
get.sdcMicroObj(object, type)
```
Arguments

object  a sdcMicroObj-class-object

type    a character vector of length 1 defining what to calculate/return/modify. Allowed
types are the all slotNames of obj.

Value

a slot of a sdcMicroObj-class-object depending on argument type

Examples

sdc <- createSdcObj(testdata2,
  keyVars=c('urbur','roof','walls','water','electcon','relat','sex'),
  numVars=c('expend','income','savings'), w='sampling_weight')
sl <- slotNames(sdc)
res <- sapply(sl, function(x) get.sdcMicroObj(sdc, type=x))
str(res)

---

Global Recoding

Description

Global recoding of variables

Usage

globalRecode(obj, ...)

Arguments

obj       a numeric vector, a data.frame or an object of class sdcMicroObj-class

...       see possible arguments below

  • column: which keyVar should be changed. Character vector of length 1
    specifying the variable name that should be recoded (required if obj is a
data.frame or an object of class sdcMicroObj-class.

  • breaks: either a numeric vector of cut points or number giving the number
    of intervals which x is to be cut into.

  • labels: labels for the levels of the resulting category. By default, labels
    are constructed using "(a,b]" interval notation. If labels = FALSE, simple
    integer codes are returned instead of a factor.

  • method: The following arguments are supported:
    – “equidistant:” for equal sized intervalls
    – “logEqui:” for equal sized intervalls for log-transformed data
    – “equalAmount:” for intervalls with approximatively the same amount
      of observations
Details
If a labels parameter is specified, its values are used to name the factor levels. If none is specified, the factor level labels are constructed.

Value
the modified `sdcmicroObj-class` or a factor, unless labels = FALSE which results in the mere integer level codes.

Note
globalRecode can not be applied to vectors stored as factors from sdcMicro >= 4.7.0!

Author(s)
Matthias Templ and Bernhard Meindl

References

See Also
cut

Examples
data(free1)
free1 <- as.data.frame(free1)

## application to a vector
head(globalRecode(free1$AGE, breaks=c(1,9,19,29,39,49,59,69,100), labels=1:8))
table(globalRecode(free1$AGE, breaks=c(1,9,19,29,39,49,59,69,100), labels=1:8))

## application to a data.frame
# automatic labels
table(globalRecode(free1, column="AGE", breaks=c(1,9,19,29,39,49,59,69,100))$AGE)

## calculation of break-points using different algorithms
table(globalRecode(free1$AGE, breaks=6))
table(globalRecode(free1$AGE, breaks=6, method="logEqui"))
table(globalRecode(free1$AGE, breaks=6, method="equalAmount"))

## for objects of class sdcMicro:
data(testdata2)
sdc <- createSdcObj(testdata2, 
   keyVars=c('urbrur','roof','walls','water','electcon','relat','sex'),
   numVars=c('expend','income','savings'), w='sampling_weight')
sdc <- globalRecode(sdc, column="water", breaks=3)
```r

table(get.sdcMicroObj(sdc, type="manipKeyVars")$water)
```

<table>
<thead>
<tr>
<th>groupAndRename</th>
<th>Join levels of a variables in an object of class <code>sdcmicroobj-class</code> or <code>factor</code> or <code>data.frame</code></th>
</tr>
</thead>
</table>

**Description**

If the input is an object of class `sdcmicroobj-class`, the specified factor-variable is recoded into a factor with less levels and risk-measures are automatically recomputed.

**Usage**

```r
groupAndRename(obj, var, before, after, addNA = FALSE)
```

**Arguments**

- `obj` object of class `sdcmicroObj-class`
- `var` name of the key variable to change
- `before` vector of levels before recoding
- `after` name of new level after recoding
- `addNA` logical, if TRUE missing values in the input variables are added to the level specified in argument after.

**Details**

If the input is of class `data.frame`, the result is a `data.frame` with a modified column specified by `var`.

If the input is of class `factor`, the result is a `factor` with different levels.

**Value**

the modified `sdcmicroObj-class`

**Author(s)**

Bernhard Meindl

**References**

**importProblem**

**Examples**

```r
## for objects of class sdcMicro:
data(testdata2)
testdata2$urbrur <- as.factor(testdata2$urbrur)
sdc <- createSdcObj(testdata2,
  keyVars=c('urbrur', 'roof', 'walls', 'water', 'electcon', 'relat', 'sex'),
  numVars=c('expend', 'income', 'savings'), w='sampling_weight')
sdc <- groupAndRename(sdc, var="urbrur", before=c("1","2"), after=c("1"))
```

**Description**

reads an sdcProblem with code that has been exported within `sdcApp`.

**Usage**

`importProblem(path)`

**Arguments**

- `path` a file path

**Value**

an object of class `sdcMicro_GUI_export` or an object of class `simple.error`

**Author(s)**

Bernhard Meindl

---

**indivRisk**

**Individual Risk computation**

**Description**

Estimation of the risk for each observation. After the risk is computed one can use e.g. the function `localSuppr()` for the protection of values of high risk. Further details can be found at the link given below.

**Usage**

`indivRisk(x, method = "approx", qual = 1, survey = TRUE)`
Arguments

- **x**: object from class freqCalc
- **method**: approx (default) or exact
- **qual**: final correction factor
- **survey**: TRUE, if we have survey data and FALSE if we deal with a population.

Details

S4 class sdcMicro objects are only supported by function `measure_risk` that also estimates the individual risk with the same method.

Value

- **rk**: base individual risk
- **method**: method
- **qual**: final correction factor
- **fk**: frequency count
- **knames**: colnames of the key variables

Note

The base individual risk method was developed by Benedetti, Capobianchi and Franconi

Author(s)

Matthias Templ. Bug in method “exact” fixed since version 2.6.5. by Youri Baeyens.

References


See Also

`measure_risk`, `freqCalc`
Examples

```r
## example from Capobianchi, Polettini and Lucarelli:
data(francdat)
f <- freqCalc(francdat, keyVars=c(2,4,5,6),w=8)
f
f$fk
f$Fk
## individual risk calculation:
indivf <- indivRisk(f)
indivf$rk
```

Description

returns the number of observations violating k-anonymity.

Usage

```r
kAnon_violations(object, weighted, k)
```

## S4 method for signature 'sdcMicroObj,logical,numeric'
```r
canon_violations(object, weighted, k)
```

Arguments

- `object` a `sdcMicroObj-class` object
- `weighted` TRUE or FALSE defining if sampling weights should be taken into account
- `k` a positive number defining parameter k

Value

the number of records that are violating k-anonymity based on unweighted sample data only (in case parameter weighted is FALSE) or computing the number of observations that are estimated to violate k-anonymity in the population in case parameter weighted equals TRUE.
Global risk using log-linear models.

**Description**

The sample frequencies are assumed to be independent and following a Poisson distribution. The parameters of the corresponding parameters are estimated by a log-linear model including the main effects and possible interactions.

**Usage**

```r
LLmodGlobalRisk(obj, method = "IPF", inclProb = NULL, form = NULL,
                 modOutput = FALSE)
```

**Arguments**

- `obj`: `sdcmicroObj-class` object or a `data.frame` containing the categorical key variables.
- `method`: At this time, only iterative proportional fitting ("IPF") can be used.
- `inclProb`: Inclusion probabilities (experimental)
- `form`: A formula specifying the model.
- `modOutput`: If TRUE, additional output is given.

**Details**

This measure aims to (1) calculate the number of sample uniques that are population uniques with a probabilistic Poisson model and (2) to estimate the expected number of correct matches for sample uniques.

ad 1) this risk measure is defined over all sample uniques (SU) as

\[ \tau_1 = \sum_{SU} P(F_k = 1 | f_k = 1), \]

i.e. the expected number of sample uniques that are population uniques.

ad 2) this risk measure is defined over all sample uniques (SU) as

\[ \tau_2 = \sum_{SU} P(F_k = 1 | f_k = 1), CORRECT! \]

Since population frequencies \( F_k \) are unknown, they has to be estimated.

The iterative proportional fitting method is used to fit the parameters of the Poisson distributed frequency counts related to the model specified to fit the frequency counts. The obtained parameters are used to estimate a global risk, defined in Skinner and Holmes (1998).

**Value**

Two global risk measures or the modified risk in the `sdcmicroObj-class` object.
Note

LLmodGlobalRisk is deprecated for modRisk and is only provided for compatibility with older versions of this package. It may be removed in future versions.

Author(s)

Matthias Templ

References


See Also

loglm, measure_risk
modRisk

LocalRecProg

Local recoding via Edmond’s maximum weighted matching algorithm

Description

To be used on both categorical and numeric input variables, although usage on categorical variables is the focus of the development of this software.

Usage

LocalRecProg(obj, ancestors = NULL, ancestor_setting = NULL, k_level = 2,
              FindLowestK = TRUE, weight = NULL, lowMemory = FALSE,
              missingValue = NA, ...)

Arguments

obj a data.frame or a sdcMicroObj-class-object
ancestors Names of ancestors of the categorical variables
ancestor_setting For each ancestor the corresponding categorical variable
k_level Level for k-anonymity
FindLowestK requests the program to look for the smallest k that results in complete matches of the data.

weight A weight for each variable (Default=1)

lowMemory Slower algorithm with less memory consumption

missingValue The output value for a suppressed value.

... see arguments below

- categoricalNames of categorical variables
- numericalNames of numerical variables

Details

Each record in the data represents a category of the original data, and hence all records in the input data should be unique by the N Input Variables. To achieve bigger category sizes (k-anonymity), one can form new categories based on the recoding result and repeatedly apply this algorithm.

Value

dataframe with original variables and the supressed variables (suffix _lr). / the modified sdcMicroObj-class

Methods

list("signature(obj="sdcMicroObj")")

Author(s)

Alexander Kowarik, Bernd Prantner, IHSN C++ source, Akimichi Takemura

References


Examples

# LocalRecProg
data(testdata2)
r1=LocalRecProg(testdata2,
categorical=c("urbrur", "roof", "walls", "water", "sex", "relat"),
missingValue=-99)
r2=LocalRecProg(testdata2,
categorical=c("urbrur", "roof", "walls", "water", "sex", "relat"),
ancestor=c("water2", "water3", "relat2"),
ancestor_setting=c("water","water","relat"),missingValue=-99)
r3=LocalRecProg(testdata2,
categorical=c("urbrur", "roof", "walls", "water", "sex", "relat"),
ancestor=c("water2", "water3", "relat2"),
ancestor_setting=c("water","water","relat"),missingValue=-99,
localSupp

FindLowestK=FALSE)

## for objects of class sdcMicro:
data(testdata2)
sdc <- createSdcObj(testdata2,
  keyVars=c('urbrur','roof','walls','water','electcon','relat','sex'),
  numvars=c('expend','income','savings'), w='sampling_weight')
sdc <- LocalRecProg(sdc)

---

**Local Suppression**

**Description**

A simple method to perform local suppression.

**Usage**

localSupp(obj, threshold = 0.15, keyVar)

**Arguments**

- **obj**: object of class freqCalc or sdcMicroObj-class.
- **threshold**: threshold for individual risk
- **keyVar**: Variable on which some values might be suppressed

**Details**

Values of high risk (above the threshold) of a certain variable (parameter keyVar) are suppressed.

**Value**

an updated object of class freqCalc or the sdcMicroObj-class object with manipulated data.

**Author(s)**

Matthias Templ and Bernhard Meindl

**References**


**See Also**

freqCalc, indivRisk
Examples

```r
## example from Capobianchi, Polettini and Lucarelli:
data(frcndat)
keyVars <- paste0("Key", 1:4)
f <- freqCalc(frcndat, keyVars = keyVars, w = 8)
f
f$ck
f$k
## individual risk calculation:
 indivf <- indivRisk(f)
 indivf$rk
## Local Suppression
localS <- localSupp(f, keyVar = "Key", threshold = 0.15)
f2 <- freqCalc(localS$freqCalc, keyVars = keyVars, w = 8)
 indivf2 <- indivRisk(f2)
 indivf2$rk
 identical(indivf$rk, indivf2$rk)
## select another keyVar and run localSupp once again, 
# if you think the table is not fully protected
## for objects of class sdcMicro:
data(testdata)
sdc <- createSdObj(testdata,
   keyVars = c('urbrur', 'roof', 'walls', 'water', 'electcon', 'relat', 'sex'),
   numVars = c('expend', 'income', 'savings'), w = 'sampling_weight')
sdc <- localSupp(sdc, keyVar = 'urbrur', threshold = 0.045)
print(sdc, type = "ls")
```

---

### localSuppression

**Local Suppression to obtain k-anonymity**

#### Description

Algorithm to achieve k-anonymity by performing local suppression.

#### Usage

```r
localSuppression(obj, k = 2, importance = NULL, combs = NULL, ...)
```

```r
kAnon(obj, k = 2, importance = NULL, combs = NULL, ...)
```

#### Arguments

- **obj**: A `sdcMicroObj-class`-object or a `data.frame`
- **k**: Threshold for k-anonymity
importance numeric vector of numbers between 1 and n (n=length of vector keyVars). This vector represents the "importance" of variables that should be used for local suppression in order to obtain k-anonymity. key-variables with importance=1 will - if possible - not suppressed, key-variables with importance=n will be used whenever possible.

combs numeric vector. if specified, the algorithm will provide k-anonymity for each combination of n key variables (with n being the value of the ith element of this parameter. For example, if combs=c(4,3), the algorithm will provide k-anonymity to all combinations of 4 key variables and then k-anonymity to all combinations of 3 key variables. It is possible to apply different k to these subsets by specifying k as a vector. If k has only one element, the same value of k will be used for all subgroups.

Details

The algorithm provides a k-anonymized data set by suppressing values in key variables. The algorithm tries to find an optimal solution to suppress as few values as possible and considers the specified importance vector. If not specified, the importance vector is constructed in a way such that key variables with a high number of characteristics are considered less important than key variables with a low number of characteristics.

The implementation provides k-anonymity per strata, if slot 'strataVar' has been set in sdcMicroObj-class or if parameter 'strataVar' is used when applying the data.frame method. For details, have a look at the examples provided.

Value

Manipulated data set with suppressions that has k-anonymity with respect to specified key-variables or the manipulated data stored in the sdcMicroObj-class.

Note

Deprecated methods 'localSupp2' and 'localSupp2Wrapper' are no longer available in sdcMicro > 4.5.0. kAnon is a more intuitive term for localSuppression because the aim is always to obtain k-anonymity for some parts of the data.
Author(s)

Bernhard Meindl, Matthias Templ

References


Examples

data(francdat)
## Local Suppression
locals <- localSuppression(francdat, keyVar=c(4,5,6))
plot(locals)
## Not run:
## for objects of class sdcMicro, no stratification
data(testdata2)
sdc <- createSdcObj(testdata2,
keyVars=c('urbrur','roof','walls','water','electcon','relat','sex'),
numVars=c('expend','income','savings'), w='sampling_weight')
sdc <- localSuppression(sdc)

data(testdata2$ageG) <- cut(testdata2$age, 5, labels=paste0("AG",1:5))
sdc <- createSdcObj(testdata2,
keyVars=c('urbrur','roof','walls','water','electcon','relat','sex'),
numVars=c('expend','income','savings'), w='sampling_weight',
strataVar='ageG')
sdc <- localSuppression(sdc)

## it is also possible to provide k-anonymity for subsets of key-variables
## with different parameter k!
## in this case we want to provide 10-anonymity for all combinations
## of 5 key variables, 20-anonymity for all combinations with 4 key variables
## and 30-anonymity for all combinations of 3 key variables.
## note: stratas are automatically considered!
combs <- 5:3
k <- c(10,20,30)
sdc <- localSuppression(sdc, k=k, combs=combs)

## data.frame method (no stratification)
keyVars <- c("urbrur","roof","walls","water","electcon","relat","sex")
strataVars <- c("ageG")
inp <- testData2[,c(keyVars, strataVars)]
ls <- localSuppression(inp, keyVars=1:7)
print(ls)
plot(ls)
## mafast

**Fast and Simple Microaggregation**

### Description

Function to perform a fast and simple (primitive) method of microaggregation. (for large datasets)

### Usage

```r
mafast(obj, variables = NULL, by = NULL, aggr = 3, measure = mean)
```

### Arguments

- **obj**: either a `sdcmicroobj-class`-object or a `data.frame`
- **variables**: variables to microaggregate. If `obj` is of class `sdcmicroObj` the numerical key variables are chosen per default.
- **by**: grouping variable for microaggregation. If `obj` is of class `sdcmicroObj` the strata variables are chosen per default.
- **aggr**: aggregation level (default=3)
- **measure**: aggregation statistic, mean, median, trim, onestep (default = mean)

### Value

If `obj` was of class `sdcmicroObj-class` the corresponding slots are filled, like `manipNumVars`, `risk` and `utility`. If `obj` was of class “data.frame” or “matrix” an object of the same class is returned.

### Author(s)

Alexander Kowarik

### See Also

`microaggregation`
Examples

data(Tarragona)
m1 <- mafast(Tarragona, variables=c("GROSS.PROFIT","OPERATING.PROFIT","SALES"),aggr=3)
data(testdata)
m2 <- mafast(testdata,variables=c("expend","income","savings"),aggr=50,by="sex")
summary(m2)

## for objects of class sdcMicro:
data(testdata2)
sdc <- createSdcObj(testdata2,
    keyVars=c('urbrur','roof','walls','water','electcon','relat','sex'),
    numVars=c('expend','income','savings'), w='sampling_weight')
sdc <- dRisk(sdc)
sdc@risk$numeric
sdc1 <- mafast(sdc,aggr=4)
sdc1@risk$numeric

sdc2 <- mafast(sdc,aggr=10)
sdc2@risk$numeric
## Not run:
##### Performance tests
x <- testdata
for(i in 1:20){
    x <- rbind(x,testdata)
}
system.time(xx <- mafast(x,variables=c("expend","income","savings"),aggr=50,by="sex"))

## End(Not run)

---

**measure_risk**

*Disclosure Risk for Categorical Variables*

**Description**

The function measures the disclosure risk for weighted or unweighted data. It computes the individual risk (and household risk if reasonable) and the global risk. It also computes a risk threshold based on a global risk value.

Prints a `measure_risk`-object

Prints a `ldiversity`-object

**Usage**

```r
measure_risk(obj, ...)

ldiversity(obj, ldiv_index = NULL, l_recurs_c = 2, missing = -999, ...)
```

## S3 method for class `measure_risk'
Arguments

obj Object of class sdcMicroObj-class

... see arguments below

- data: Input data, a data.frame.
- keyVars: names (or indices) of categorical key variables (for data-frame method)
- w: name of variable containing sample weights
- hid: name of the clustering variable, e.g. the household ID
- max_global_risk: Maximal global risk for threshold computation
- fast_hier: If TRUE a fast approximation is computed if household data are provided.

- ldiv_index: indices (or names) of the variables used for l-diversity
- l_recurc_c: L-Diversity Constant
- missing: a integer value to be used as missing value in the C++ routine
- x: Output of measure_risk() or ldiversity()

Details

To be used when risk of disclosure for individuals within a family is considered to be statistical independent.

Internally, function freqCalc() and indivRisk are used for estimation.

Measuring individual risk: The individual risk approach based on so-called super-population models. In such models population frequency counts are modeled given a certain distribution. The estimation procedure of sample frequency counts given the population frequency counts is modeled by assuming a negative binomial distribution. This is used for the estimation of the individual risk. The extensive theory can be found in Skinner (1998), the approximation formulas for the individual risk used is described in Franconi and Polettini (2004).

Measuring hierarchical risk: If "hid" - the index of variable holding information on the hierarchical cluster structures (e.g., individuals that are clustered in households) - is provided, the hierarchical risk is additional estimated. Note that the risk of re-identifying an individual within a household may also affect the probability of disclosure of other members in the same household. Thus, the household or cluster-structure of the data must be taken into account when estimating disclosure risks. It is commonly assumed that the risk of re-identification of a household is the risk that at least one member of the household can be disclosed. Thus this probability can be simply estimated from individual risks as 1 minus the probability that no member of the household can be identified.

Global risk: The sum of the individual risks in the dataset gives the expected number of re-identifications that serves as measure of the global risk.
l-Diversity: If “ldiv_index” is unequal to NULL, i.e. if the indices of sensible variables are specified, various measures for l-diversity are calculated. l-diversity is an extension of the well-known k-anonymity approach where also the uniqueness in sensible variables for each pattern spanned by the key variables are evaluated.

Value

A modified `sdcmicroobj-class` object or a list with the following elements:

- `global_risk_ER`: expected number of re-identification.
- `global_risk`: global risk (sum of individual risks).
- `global_risk_pct`: global risk in percent.
- `Res`: matrix with the risk, frequency in the sample and grossed-up frequency in the population (and the hierarchical risk) for each observation.
- `global_threshold`: for a given `max_global_risk` the threshold for the risk of observations.
- `max_global_risk`: the input `max_global_risk` of the function.
- `hier_risk_ER`: expected number of re-identification with household structure.
- `hier_risk`: global risk with household structure (sum of individual risks).
- `hier_risk_pct`: global risk with household structure in percent.
- `ldiversity`: Matrix with Distinct_Ldiversity, Entropy_Ldiversity and Recursive_Ldiversity for each sensitivity variable.

Prints risk-information into the console
Information on L-Diversity Measures in the console

Author(s)

Alexander Kowarik, Bernhard Meindl, Matthias Templ, Bernd Prantner, minor parts of IHSN C++ source

References


See Also

`freqCalc`, `indivrisk`

`measure_risk`
Examples

```r
## measure_risk with sdcMicro objects:
data(testdata)
sdc <- createSdcObj(testdata,
 keyVars=c('urbrur','roof','walls','water','electcon'),
 numVars=c('expend','income','savings'), w='sampling_weight')

## risk is already estimated and available in...
names(sdc@risk)

## measure risk on data frames or matrices:
res <- measure_risk(testdata,
 keyVars=c("urbrur","roof","walls","water","sex"))
print(res)
head(res$Res)
resw <- measure_risk(testdata,
 keyVars=c("urbrur","roof","walls","water","sex"),w="sampling_weight")
print(resw)
head(resw$Res)
res1 <- ldiversity(testdata,
 keyVars=c("urbrur","roof","walls","water","sex"),ldiv_index="electcon")
print(res1)
head(res1)
res2 <- ldiversity(testdata,
 keyVars=c("urbrur","roof","walls","water","sex"),ldiv_index=c("electcon","relat"))
print(res2)
head(res2)

# measure risk with household risk
resh <- measure_risk(testdata,
 keyVars=c("urbrur","roof","walls","water","sex"),w="sampling_weight",hid="ori_hid")
print(resh)

# change max_global_risk
rest <- measure_risk(testdata,
 keyVars=c("urbrur","roof","walls","water","sex"),
 w="sampling_weight",max_global_risk=0.0001)
print(rest)

## for objects of class sdcMicro:
data(testdata2)
sdc <- createSdcObj(testdata2,
 keyVars=c('urbrur','roof','walls','water','electcon','relat','sex'),
 numVars=c('expend','income','savings'), w='sampling_weight')
## already interally applied and available in object sdc:
## sdc <- measure_risk(sdc)
```
mergeHouseholdData  
Replaces the raw household-level data with the anonymized household-level data in the full dataset for anonymization of data with a household structure (or other hierarchical structure). Requires a matching household ID in both files.

Description
Replaces the raw household-level data with the anonymized household-level data in the full dataset for anonymization of data with a household structure (or other hierarchical structure). Requires a matching household ID in both files.

Usage
mergeHouseholdData(dat, hhId, dathh)

Arguments
- dat: a data.frame with the full dataset
- hhId: name of the household (cluster) ID (identical in both datasets)
- dathh: a dataframe with the treated household level data (generated for example with selectHouseholdData)

Value
a data.frame with the treated household level variables and the raw individual level variables

Author(s)
Thijs Benschop and Bernhard Meindl

Examples
```r
# Load data
x <- testdata
# Create household level dataset
x_hh <- selectHouseholdData(dat=x, hhId="ori_hid",
  hhVars=c("urbrur", "roof", "walls", "water", "electcon", "household_weights"))
# Anonymize household level dataset and extract data
sdc_hh <- createSdcObj(x_hh, keyVars=c("urbrur", 'roof'), w='household_weights')
sdc_hh <- kAnon(sdc_hh, k = 3)
x_hh_anon <- extractManipData(sdc_hh)

# Merge anonymized household level data back into the full dataset
x_anonhh <- mergeHouseholdData(x, "ori_hid", x_hh_anon)

# Anonymize full dataset and extract data
sdc_full <- createSdcObj(x_anonhh, keyVars=c('sex', 'age', 'urbrur', 'roof'), w='sampling_weight')
sdc_full <- kAnon(sdc_full, k = 3)
x_full_anon <- extractManipData(sdc_full)
```
Description

Function to perform various methods of microaggregation.

Usage

microaggregation(obj, variables = NULL, aggr = 3, strata_variables = NULL,
    method = "mdav", weights = NULL, nc = 8, clustermethod = "clara",
    measure = "mean", trim = 0, varsort = 1, transf = "log")

Arguments

obj either an object of class sdcMicroObj-class or a data.frame
variables variables to microaggregate. For NULL: If obj is of class sdcMicroObj, all numerical key variables are chosen per default. For data.frames, all columns are chosen per default.
aggr aggregation level (default=3)
strata_variables for data.frames, by-variables for applying microaggregation only within strata defined by the variables. For sdcMicroObj-class-objects, the stratification-variable defined in slot @strataVar is used. This slot can be changed any time using strataVar<-. method pca, rmd, onedims, single, simple, clustpca, pppca, clustpppcma, mdav, clustmcdpca, influence, mcdpca weights sampling weights. If obj is of class sdcMicroObj the vector of sampling weights is chosen automatically. If determined, a weighted version of the aggregation measure is chosen automatically, e.g. weighted median or weighted mean.
nc number of cluster, if the chosen method performs cluster analysis
clustermethod clustermethod, if necessary
measure aggregation statistic, mean, median, trim, onestep (default=mean)
trim trimming percentage, if measure=trim
varsort variable for sorting, if method=single
transf transformation for data x

Details

On http://neon.vb.cbs.nl/casc/Glossary.htm one can found the “official” definition of microaggregation:

Records are grouped based on a proximity measure of variables of interest, and the same small groups of records are used in calculating aggregates for those variables. The aggregates are released instead of the individual record values.
The recommended method is “rmd” which forms the proximity using multivariate distances based on robust methods. It is an extension of the well-known method “mdav”. However, when computational speed is important, method “mdav” is the preferable choice.

While for the proximity measure very different concepts can be used, the aggregation itself is naturally done with the arithmetic mean. Nevertheless, other measures of location can be used for aggregation, especially when the group size for aggregation has been taken higher than 3. Since the median seems to be unsuitable for microaggregation because of being highly robust, other measures which are included can be chosen. If a complex sample survey is microaggregated, the corresponding sampling weights should be determined to either aggregate the values by the weighted arithmetic mean or the weighted median.

This function contains also a method with which the data can be clustered with a variety of different clustering algorithms. Clustering observations before applying microaggregation might be useful. Note, that the data are automatically standardised before clustering.

The usage of clustering method ‘Mclust’ requires package mclust02, which must be loaded first. The package is not loaded automatically, since the package is not under GPL but comes with a different licence.

The are also some projection methods for microaggregation included. The robust version ‘pppca’ or ‘clustpppca’ (clustering at first) are fast implementations and provide almost everytime the best results.

Univariate statistics are preserved best with the individual ranking method (we called them ‘onedsims’, however, often this method is named ‘individual ranking’), but multivariate statistics are strongly affected.

With method ‘simple’ one can apply microaggregation directly on the (unsorted) data. It is useful for the comparison with other methods as a benchmark, i.e. replies the question how much better is a sorting of the data before aggregation.

Value

If ‘obj’ was of class sdcMicroObj-class the corresponding slots are filled, like manipNumVars, risk and utility. If ‘obj’ was of class “data.frame”, an object of class “micro” with following entities is returned:

- x: original data
- mx: the microaggregated dataset
- method: method
- aggr: aggregation level
- measure: proximity measure for aggregation

Note

if only one variable is specified, mfast is applied and argument method is ignored.

Author(s)

Matthias Templ, Bernhard Meindl
For method “mdav”: This work is being supported by the International Household Survey Network (IHSN) and funded by a DGF Grant provided by the World Bank to the PARIS21 Secretariat at the Organisation for Economic Co-operation and Development (OECD). This work builds on previous work which is elsewhere acknowledged.

Author for the integration of the code for mdav in R: Alexander Kowarik.

References

http://www.springerlink.com/content/v257655u88w2/?sortorder=asc&p_o=20


See Also

summary.micro, plotMicro, valTable

Examples

data(Tarragona)
m1 <- microaggregation(Tarragona, method='onedims', aggr=3)
## summary(m1)
data(testdata)
m2 <- microaggregation(testdata[1:100,c('expend','income','savings')],
   method='mdav', aggr=4)
summary(m2)

## for objects of class sdcMicro:
## no stratification because @strataVar is NULL
data(testdata2)
sdc <- createSdcObj(testdata2,
   keyVars=c('urbrur','roof','walls','water','electcon','sex'),
   numVars=c('expend','income','savings'), w='sampling_weight')
sdc <- microaggregation(sdc, variables=c("expend","income"))

## with stratification by 'relat'
strataVar(sdc) <- "relat"
sdc <- microaggregation(sdc, variables=c("savings"))
microaggrGower  

Microaggregation for numerical and categorical key variables based on a distance similar to the Gower Distance

Description

The microaggregation is based on the distances computed similar to the Gower distance. The distance function makes distinction between the variable types factor, ordered, numerical and mixed (semi-continuous variables with a fixed probability mass at a constant value e.g. 0)

Usage

microaggrGower(obj, variables = NULL, aggr = 3, dist_var = NULL, by = NULL, mixed = NULL, mixed.constant = NULL, trace = FALSE, weights = NULL, numFun = mean, catFun = VIM::sampleCat, addRandom = FALSE)

Arguments

- **obj**  
  sdcMicroObj-class-object or a data.frame
- **variables**  
  character vector with names of variables to be aggregated (Default for sdcMicroObj is all keyVariables and all numeric key variables)
- **aggr**  
  aggregation level (default=3)
- **dist_var**  
  character vector with variable names for distance computation
- **by**  
  character vector with variable names to split the dataset before performing microaggregation (Default for sdcMicroObj is strataVar)
- **mixed**  
  character vector with names of mixed variables
- **mixed.constant**  
  numeric vector with length equal to mixed, where the mixed variables have the probability mass
- **trace**  
  TRUE/FALSE for some console output
- **weights**  
  numerical vector with length equal the number of variables for distance computation
- **numFun**  
  function: to be used to aggregated numerical variables
- **catFun**  
  function: to be used to aggregated categorical variables
- **addRandom**  
  TRUE/FALSE if a random value should be added for the distance computation.

Details

The function sampleCat samples with probabilities corresponding to the occurrence of the level in the NNs. The function maxCat chooses the level with the most occurrences and random if the maximum is not unique.

Value

The function returns the updated sdcMicroObj or simply an altered data frame.
Note

In each by group all distance are computed, therefore introducing more by-groups significantly decreases the computation time and memory consumption.

Author(s)

Alexander Kowarik

See Also

sampleCat and maxCat

Examples

data(testdata, package = "sdcMicro")
testdata <- testdata[1:200,]
for(i in c(1:7, 9)) testdata[, i] <- as.factor(testdata[, i])
test <- microaggrGower(testdata, variables = c("relat", "age", "expend"),
dist_var = c("age", "sex", "income", "savings"), by = c("urbrur", "roof"))

sdc <- createSdcObj(testdata,
keyVars = c("urbrur", "roof", "walls", "water", "electcon", "relat", "sex"),
numVars = c("expend", "income", "savings"), w = "sampling_weight")

sdc <- microaggrGower(sdc)

data(microData)
microData <- as.data.frame(microData)
m1 <- microaggregation(microData, method = "mdav")
summary(m1)

microData
**modRisk**

*Global risk using log-linear models.*

**Description**

The sample frequencies are assumed to be independent and following a Poisson distribution. The parameters of the corresponding parameters are estimated by a log-linear model including the main effects and possible interactions.

**Usage**

```r
modRisk(obj, method = "default", weights, formulaM, bound = Inf, ...)
```

**Arguments**

- `obj`: An `sdcMicroObj-class`-object or a numeric matrix or data.frame containing all variables required in the specified model.
- `method`: chose method for model-based risk-estimation. Currently, the following methods can be selected:
  - "default": the standard log-linear model.
  - "CE": the Clogg Eliason method, additionally, considers survey weights by using an offset term.
  - "PML": the pseudo maximum likelihood method.
  - "weightedLLM": the weighted maximum likelihood method, considers survey weights by including them as one of the predictors.
  - "IPF": iterative proportional fitting as used in deprecated method 'LLmodGlobalRisk'.
- `weights`: a variable name specifying sampling weights
- `formulaM`: A formula specifying the model.
- `bound`: a number specifying a threshold for 'risky' observations in the sample.
- `...`: additional parameters passed through, currently ignored.

**Details**

This measure aims to (1) calculate the number of sample uniques that are population uniques with a probabilistic Poisson model and (2) to estimate the expected number of correct matches for sample uniques.

ad 1) this risk measure is defined over all sample uniques as

\[
\tau_1 = \sum_{j:f_j = 1} P(F_j = 1|f_j = 1),
\]

i.e. the expected number of sample uniques that are population uniques.
ad 2) this risk measure is defined over all sample uniques as

\[ \tau_2 = \sum_{j: f_j = 1} P(1/F_j | f_j = 1) \]

Since population frequencies \( F_k \) are unknown, they need to be estimated.
The iterative proportional fitting method is used to fit the parameters of the Poisson distributed frequency counts related to the model specified to fit the frequency counts. The obtained parameters are used to estimate a global risk, defined in Skinner and Holmes (1998).

Value

Two global risk measures and some model output given the specified model. If this method is applied to an sdcmicroObj-class-object, the slot 'risk' in the object ist updated with the result of the model-based risk-calculation.

Author(s)

Matthias Templ, Marius Totter, Bernhard Meindl

References


See Also

loglm, measure_risk

Examples

```r
## data.frame method
data(testdata2)
form <- ~sex+water+roof
w <- "sampling_weight"
(modRisk(testdata2, method="default", formulaM=form, weights=w))
(modRisk(testdata2, method="CE", formulaM=form, weights=w))
(modRisk(testdata2, method="PML", formulaM=form, weights=w))
(modRisk(testdata2, method="weightedLLM", formulaM=form, weights=w))
(modRisk(testdata2, method="1PF", formulaM=form, weights=w))

## application to a sdcMicroObj
data(testdata2)
sdc <- createSdcObj(testdata2,
  keyVars=c('urbrur','roof','walls','electcon','relat','sex'),
```
plot.localSuppression

numVars=c('expend','income','savings'), w='sampling_weight')
sdc <- modRisk(sdc,form=~sex+water+roof)
slot(sdc, "risk")$model

nextSdcObj

Description

internal function used to provide the undo-functionality.

Usage

nextSdcObj(obj)

Arguments

obj a sdcMicroObj-class object

Value

a modified sdcMicroObj-class object

plot.localSuppression

Description

Barplot for objects from class localSuppression.

Usage

## S3 method for class 'localSuppression'
plot(x, ...)

Arguments

x object of class ‘localSuppression’

Details

Just look at the resulting plot.
plot.sdcMicroObj

Plot functions for objects of class `sdcmicroobj-class`

Description

Descriptive plot function for `sdcmicroobj-class`-objects. Currently only visualization of local suppression is implemented.

Usage

```r
## S3 method for class 'sdcmicroobj'
plot(x, type = "ls", ...)
```

Arguments

- `x` An object of class `sdcmicroobj-class`
- `type` specified what kind of plot will be generated
  - 'ls': plot of local suppressions in key variables
- `...` currently ignored
Author(s)
Bernhard Meindl

Examples

data(testdata)
sdc <- createSdcObj(testdata,
  keyVars=c('urbrur', 'roof', 'walls', 'relat', 'sex'),
  pramVars=c('water', 'electcon'),
  numVars=c('expend', 'income', 'savings'), w='sampling_weight')
sdc <- kAnon(sdc, k=5)
plot(sdc, type="ls")

plotMicro                  Comparison plots

Description
Plots for the comparison of the original data and perturbed data.

Usage
plotMicro(x, p, which.plot = 1:3)

Arguments

  x     object from class micro
  p     necessary parameter for the box cox transformation (lambda)
  which.plot

which.plot which plot should be created?
  • 1: density traces
  • 2: parallel boxplots
  • 3: differences in totals

Details
Univariate and multivariate comparison plots are implemented to detect differences between the perturbed and the original data, but also to compare perturbed data which are produced by different methods.

Author(s)
Matthias Templ
References

See Also
microaggregation

Examples

```r
data(free1)
free1 <- as.data.frame(free1)
m1 <- microaggregation(free1[, 31:34], method="onedims", aggr=3)
m2 <- microaggregation(free1[, 31:34], method="pca", aggr=3)
plotMicro(m1, p=1, which.plot=1)
```

---

**pram**

*Post Randomization*

Description
To be used on categorical data stored as factors. The algorithm randomly changes the values of variables in selected records (usually the risky ones) according to an invariant probability transition matrix or a custom-defined transition matrix.

Usage

```r
pram(obj, variables = NULL, strata_variables = NULL, pd = 0.8, alpha = 0.5)
```

Arguments

- **obj**
  
  Input data. Allowed input data are objects of class `data.frame`, `factor` or `sdcmicroobj-class`.

- **variables**
  
  Names of variables in 'obj' on which post-randomization should be applied. If obj is a factor, this argument is ignored. Please note that pram can only be applied to factor-variables.

- **strata_variables**
  
  Names of variables for stratification (will be set automatically for an object of class `sdcmicroObj-class`). One can also specify an integer vector or factor that specifies that desired groups. This vector must match the dimension of the input data set, however. For a possible use case, have a look at the examples.
minimum diagonal entries for the generated transition matrix \( P \). Either a vector of length \( 1 \) (which is recycled) or a vector of the same length as the number of variables that should be postrandomized. It is also possible to set \( pd \) to a numeric matrix. This matrix will be used directly as the transition matrix. The matrix must be constructed as follows:

- the matrix must be a square matrix
- the rownames and colnames of the matrix must match the levels (in the same order) of the factor-variable that should be postrandomized.
- the rowSums and colSums of the matrix need to equal 1

It is also possible to combine the different ways. For details have a look at the examples.

\( \alpha \) amount of perturbation for the invariant Pram method. This is a numeric vector of length \( 1 \) (that will be recycled if necessary) or a vector of the same length as the number of variables. If one specified as transition matrix directly, \( \alpha \) is ignored.

... further input, currently ignored.

**Value**

a modified \texttt{sdcmicroObj-class} object or a new object containing original and post-randomized variables (with suffix "\_pram").

**Note**

Deprecated method ‘pram\_strata’ is no longer available in \texttt{sdcMicro} > 4.5.0

**Author(s)**

Alexander Kowarik, Matthias Templ, Bernhard Meindl

**References**

http://www.gnu.org/software/glpk


Examples

data(testdata)

## application on a factor-variable
res <- pram(as.factor(testdata$roof))
print(res)
summary(res)

## application on a data.frame
## pram can only be applied to factors, thus we have to recode
## to factors before the method can be applied
testdata$roof <- factor(testdata$roof)
testdata$walls <- factor(testdata$walls)
testdata$water <- factor(testdata$water)

## pram() is applied within subgroups defined by
## variables "urbrur" and "sex"
res <- pram(testdata, variables="roof",
strata_variables=c("urbrur","sex"))
print(res)
summary(res)

## default parameters (pd=0.8 and alpha=0.5) for the generation
## of the invariant transition matrix will be used for all variables
res1 <- pram(testdata, variables=c("roof","walls","water"))
print(res1)

## specific parameters for each variable
res2 <- pram(testdata,variables=c("roof","walls","water"),
pd=c(0.95,0.8,0.9), alpha=0.5)
print(res2)

## detailed information on pram-parameters (such as the transition matrix 'Rs')
## is stored in the output, eg. for variable 'roof'
attr(res2, "pram_params")$roof

## we can also specify a custom transition-matrix directly
# for variable roof; matrix must have rownames and colnames that match
# the levels of the variable that should be post-randomized
# rowSums() and colSums() must equal 1 too!
mat <- diag(length(levels(testdata$roof)))
rownames(mat) <- colnames(mat) <- levels(testdata$roof)
res3 <- pram(testdata,variables="roof", pd=mat)
print(res3) # of course, nothing has changed!

## it is possible use a transistion matrix for a variable and use the 'traditional' way
## of specifying a number for the minimal diagonal entries of the transition matrix
## for other variables. In this case we must supply \code{pd} as list.
res4 <- pram(testdata,variables=c("roof","walls"), pd=list(mat,0.5), alpha=c(NA, 0.5))
print(res4)
summary(res4)
attr(res4, "pram_params")
## application to objects of class sdcMicro with default parameters

data(testdata2)
testdata2$urbrur <- factor(testdata2$urbrur)
sdc <- createSdcObj(testdata2,
  keyVars=c("roof", "walls", "water", "electcon", "relat", "sex"),
  numVars=c("expend", "income", "savings"), w= sampling_weight)
sdc <- pram(sdc, variables=c("urbrur"))
print(sdc, type="pram")

## this is equal to the previous application. If argument 'variables' is NULL,
## all variables from slot 'pramVars' will be used if possible.
sdc <- createSdcObj(testdata2,
  keyVars=c("roof", "walls", "water", "electcon", "relat", "sex"),
  numVars=c("expend", "income", "savings"), w= sampling_weight,
  pramVars="urbrur")
sdc <- pram(sdc)
print(sdc, type="pram")

## we can specify transition matrices for sdcMicroObj-objects too
testdata2$roof <- factor(testdata2$roof)
sdc <- createSdcObj(testdata2,
  keyVars=c("roof", "walls", "water", "electcon", "relat", "sex"),
  numVars=c("expend", "income", "savings"), w= sampling_weight)
mat <- diag(length(levels(testdata2$roof)))
rownames(mat) <- colnames(mat) <- levels(testdata2$roof)
mat[1,] <- c(0.9, 0.0, 0.0, 0.05, 0.05)
sdc <- pram(sdc, variables="roof", pd=mat)
print(sdc, type="pram")
# we can also have a look at the transitions
get.sdcMicroObj(sdc, "pram")$transitions

### print.freqCalc

Print method for objects from class freqCalc.

#### Description

Print method for objects from class freqCalc.

#### Usage

```r
## S3 method for class 'freqCalc'
print(x, ...)
```

#### Arguments

- `x` object from class `freqCalc`
- `...` Additional arguments passed through.
Value

information about the frequency counts for key variables for object of class `freqCalc`.

Author(s)

Matthias Templ

See Also

`freqCalc`

Examples

```r
## example from Capobianchi, Polettini and Lucarelli:
data(francdat)
f <- freqCalc(francdat, keyVars=c(2,4,5,6), w=8)
f
```

---

**print.indivRisk** *Print method for objects from class indivRisk*

### Description

Print method for objects from class indivRisk

### Usage

```r
## S3 method for class 'indivRisk'
print(x, ...)
```

### Arguments

- `x` : object from class indivRisk
- `...` : Additional arguments passed through.

### Value

few information about the method and the final correction factor for objects of class ‘indivRisk’.

### Author(s)

Matthias Templ

### See Also

`indivRisk`
print.localSuppression

Print method for objects from class localSuppression

Usage

```
## S3 method for class 'localSuppression'
print(x, ...)  
```

Arguments

- `x` object from class localSuppression
- `...` Additional arguments passed through.

Value

Information about the frequency counts for key variables for object of class 'localSuppression'.

Author(s)

Matthias Templ

See Also

localSuppression

Examples

```
## example from Capobianchi, Polettini and Lucarelli:
data(francdat)
f1 <- freqCalc(francdat, keyVars=c(2,4,5,6), w=8)
data.frame(fk=f1$fk, Fk=f1$Fk)
## individual risk calculation:
indivRisk(f1)
```
print.micro  Print method for objects from class micro

Description

printing an object of class micro

Usage

```r
## S3 method for class 'micro'
print(x, ...)
```

Arguments

- `x`: object from class micro
- `...`: Additional arguments passed through.

Value

information about method and aggregation level from objects of class micro.

Author(s)

Matthias Templ

See Also

microaggregation

Examples

```r
data(free1)
free1 <- as.data.frame(free1)
m1 <- microaggregation(free1[, 31:34], method='onedims', aggr=3)
m1
```
print.modrisk

**Description**

Print method for objects from class modrisk

**Usage**

```r
## S3 method for class 'modrisk'
print(x, ...)
```

**Arguments**

- `x`: an object of class `modrisk`
- `...`: Additional arguments passed through.

**Value**

Output of model-based risk estimation

**Author(s)**

Bernhard Meindl

**See Also**

`modRisk`

---

print.pram

**Description**

Print method for objects from class pram

**Usage**

```r
## S3 method for class 'pram'
print(x, ...)
```

**Arguments**

- `x`: an object of class `pram`
- `...`: Additional arguments passed through.
Value

absolute and relative frequencies of changed observations in each modified variable

Author(s)

Bernhard Meindl, Matthias Templ
Matthias Templ and Bernhard Meindl

See Also

pram

Description

Descriptive print function for Frequencies, local Supression, Recoding, categorical risk and numerical risk.

Usage

```r
## S4 method for signature 'sdcMicroObj'
print(x, type = "kAnon", docat = TRUE, ...)
```

Arguments

- `x` An object of class `sdcMicroObj-class`
- `type` Selection of the content to be returned or printed
- `docat` logical, if TRUE (default) the results will be actually printed
- `...` the type argument for the print method, currently supported are:
  - `general`: basic information on the input obj such as the number of observations and variables.
  - `kAnon`: displays information about 2- and 3-anonymity
  - `ls`: displays various information if local suppression has been applied.
  - `pram`: displays various information if post-randomization has been applied.
  - `recode`: shows information about categorical key variables before and after recoding
  - `risk`: displays information on re-identification risks
  - `numrisk`: displays risk- and utility measures for numerical key variables
Details

Possible values for the type argument of the print function are: "freq": for Frequencies, "ls": for Local Supression output, "pram": for results of post-randomization "recode":for Recodes, "risk": for Categorical risk and "numrisk": for Numerical risk.

Possible values for the type argument of the freq function are: "fk": Sample frequencies and "Fk": weighted frequencies.

Author(s)

Alexander Kowarik, Matthias Templ, Bernhard Meindl

Examples

data(testdata)
sdc <- createSdcObj(testdata,
keyVars=c('urbrur','roof','walls','relat','sex'),
pramVars=c('water','electcon'),
numVars=c('expend','income','savings'), w='sampling_weight')
sdc <- microaggregation(sdc, method="mdav", aggr=3)
print(sdc)
print(sdc, type="general")
print(sdc, type="ls")
print(sdc, type="recode")
print(sdc, type="risk")
print(sdc, type="numrisk")
print(sdc, type="pram")
print(sdc, type="kAnon")
print(sdc, type="comp_numvars")

print.suda2  

Print method for objects from class suda2

Description

Print method for objects from class suda2.

Usage

## S3 method for class 'suda2'
print(x, ...)

Arguments

x an object of class suda2
...
additional arguments passed through.

Value

Table of dis suda scores.
**rankSwap**

**Author(s)**
Matthias Templ

**See Also**
suda2

**Examples**

```r
## Not run:
data(testdata)
data_suda2 <- suda2(testdata,variables=c("urbrur","roof","walls","water","sex"))
data_suda2

## End(Not run)
```

---

**Description**

Swapping values within a range so that, first, the correlation structure of original variables are preserved, and second, the values in each record are disturbed. To be used on numeric or ordinal variables where the rank can be determined and the correlation coefficient makes sense.

**Usage**

```r
rankSwap(obj, variables = NULL, TopPercent = 5, BottomPercent = 5,
         K0 = -1, R0 = 0.95, P = 0, missing = NA, seed = NULL)
```

**Arguments**

- **obj**
  a `sdcMicroObj-class`-object or a `data.frame`

- **variables**
  names or index of variables for that rank swapping is applied. For an object of class `sdcMicroObj-class`, all numeric key variables are selected if `variables=NULL`.

- **TopPercent**
  Percentage of largest values that are grouped together before rank swapping is applied.

- **BottomPercent**
  Percentage of lowest values that are grouped together before rank swapping is applied.

- **K0**
  Subset-mean preservation factor. Preserves the means before and after rank swapping within a range based on K0. K0 is the subset-mean preservation factor such that \[ |X_1 - X_2| \leq \frac{2K_0X_1}{\sqrt{N_S}} \], where \(X_1\) and \(X_2\) are the subset means of the field before and after swapping, and \(N_S\) is the sample size of the subset.
**rankSwap**

- **R0**
  Multivariate preservation factor. Preserves the correlation between variables within a certain range based on the given constant R0. We can specify the preservation factor as $R_0 = \frac{R_1}{R_2}$, where $R_1$ is the correlation coefficient of the two fields after swapping, and $R_2$ is the correlation coefficient of the two fields before swapping.

- **P**
  Rank range as percentage of total sample size. We can specify the rank range itself directly, noted as $P$, which is the percentage of the records. So two records are eligible for swapping if their ranks, $i$ and $j$ respectively, satisfy $|i - j| \leq \frac{PN}{100}$, where $N$ is the total sample size.

- **missing**
  Missing - the value to be used as missing value in the C++ routine instead of NA. If NA, a suitable value is calculated internally. Note that in the returned dataset, all NA-values (if any) will be replaced with this value.

- **seed**
  Seed.

**Details**

Rank swapping sorts the values of one numeric variable by their numerical values (ranking). The restricted range is determined by the rank of two swapped values, which cannot differ, by definition, by more than $P$ percent of the total number of observations. R0 and K0 are only used if positive. Only one of the two are used (R0 is preferred if both are positive).

**Value**

The rank-swapped data set or a modified `sdcmicroObj-class` object.

**Author(s)**

Alexander Kowarik for the interface, Bernhard Meindl for improvements.

For the underlying C++ code: This work is being supported by the International Household Survey Network (IHSN) and funded by a DGF Grant provided by the World Bank to the PARIS21 Secretariat at the Organisation for Economic Co-operation and Development (OECD). This work builds on previous work which is elsewhere acknowledged.

**References**


**Examples**

```r
data(testdata2)
data_swap <- rankSwap(testdata2, variables=c("age","income","expend","savings"))
```

## for objects of class `sdcmicro`:
data(testdata2)
sdc <- createSdcObj(testdata2,
  keyVars=c('urbrur','roof','walls','water','electcon','relat','sex'),
  numVars=c('expend','income','savings'), w='sampling_weight')
sdc <- rankSwap(sdc)

Description

reads data from various formats into R. Used in sdcApp.

Usage

readMicrodata(path, type, convertCharToFac = TRUE, drop_all_missings = TRUE, ...)

Arguments

path a file path

which format does the file have. Currently allowed values are

• sas
• spss
• stata
• R
• rdf
• csv

convertCharToFac (logical) if TRUE, all character vectors are automatically converted to factors

drop_all_missings (logical) if TRUE, all variables that contain NA-values only will be dropped

... additional parameters. Currently used only if type='csv' to pass arguments to read.table().

Value

a data.frame or an object of class 'simple.error'. If a stata file was read in, the resulting data.frame has an additional attribute lab in which variable and value labels are stored.

Note

if type is either 'sas', 'spss' or 'stata', values read in as NaN will be converted to NA.

Author(s)

Bernhard Meindl
removeDirectID  Remove certain variables from the data set inside a sdc object.

Description
Delete variables without changing anything else in the sdcObject (writing NAs).

Usage
removeDirectID(obj, var)

Arguments
- obj: object of class `sdcmicroObj-class`
- var: name of the variable(s) to be remove

Value
the modified `sdcmicroObj-class`

Author(s)
Alexander Kowarik

Examples
```r
# for objects of class sdcMicro:
data(testdata2)
sdc <- createSdcObj(testdata, keyVars=c('urbrur','roof'),
umVars=c('expend','income','savings'), w='sampling_weight')
sdc <- removeDirectID(sdc, var="age")
```

report  Generate an Html-report from an sdcMicroObj

Description
Summary statistics of the original and the perturbed data set

Usage
```r
report(obj, outfile = getwd(), filename = "SDC-Report",
title = "SDC-Report", internal = FALSE, verbose = FALSE)
```
Arguments

- **obj**: an object of class `sdcMicroObj-class` or `reportObj`
- **outdir**: output folder
- **filename**: output filename
- **title**: Title for the report
- **internal**: TRUE/FALSE, if TRUE a detailed internal report is produced, else a non-disclosive overview
- **verbose**: TRUE/FALSE, if TRUE, some additional information is printed.

Details

The application of this function provides you with an html-report for your sdcMicro object that contains useful summaries about the anonymization process.

Author(s)

Matthias Templ, Bernhard Meindl

Examples

```r
## Not run:
data(testdata2)
sdc <- createSdcObj(testdata2,
    keyVars=c('urbrur','roof','walls','water','electcon','relat','sex'),
    numVars=c('expend','income','savings'), w='sampling_weight')
report(sdc)
```

Description

Allows to compute risky (unweighted) combinations of key variables either up to a specified dimension or using identification level. This mimics the approach taken in mu-argus.

Usage

```r
riskyCells(obj, useIdentificationLevel = FALSE, threshold, ...)
```
Arguments

obj a data.frame, data.table or an object of class `sdcmicroObj-class`

useIdentificationLevel (logical) specifies if tabulation should be done up to a specific dimension (useIdentificationLevel=FALSE using argument `maxDim`) or taking identification levels (useIdentificationLevel=FALSE using argument level) into account.

threshold a numeric vector specifying the thresholds at which cells are considered to be unsafe. In case a tabulation is done up to a specific level (useIdentificationLevel=FALSE), the thresholds may be specified differently for each dimension. In the other case, the same threshold is used for all tables.

... see possible arguments below

- keyVars: index or variable-names within obj that should be used for tabulation. In case obj is of class `sdcmicroObj-class`, this argument is not used and the pre-defined key-variables are used.
- level: in case useIdentificationLevel=FALSE, this numeric vector specifies the importance of the key variables. The construction of output tables follows the implementation in mu-argus, see e.g. [http://neon.vb.cbs.nl/casc/Software/MUmanual5.1.pdf](http://neon.vb.cbs.nl/casc/Software/MUmanual5.1.pdf). The length of this numeric vector must match the number of key variables.
- maxDim: in case useIdentificationLevel=FALSE, this number specifies maximal number of variables to tabulate.

Value

a data.table showing the number of unsafe cells, thresholds for any combination of the key variables. If the input was a `sdcmicroObj-class` object and some modifications have been already applied to the categorical key variables, the resulting output contains the number of unsafe cells both for the original and the modified data.

Author(s)

Bernhard Meindl

Examples

```r
# Not run:
# data.frame method / all combinations up to maxDim
riskyCells(testdata2, keyVars=c(1:5), threshold=c(50,25,10,5),
           useIdentificationLevel=FALSE, maxDim=4)
riskyCells(testdata2, keyVars=c(1:5), threshold=10,
           useIdentificationLevel=FALSE, maxDim=3)

# data.frame method / using identification levels
riskyCells(testdata2, keyVars=c(1:6), threshold=20,
           useIdentificationLevel=TRUE, level=c(1,1,2,3,3,5))
riskyCells(testdata2, keyVars=c(1,3,4,6), threshold=10,
           useIdentificationLevel=TRUE, level=c(1,2,2,4))
```
sdcApp

## Description

starts the graphical user interface developed with *shiny*.

## Usage

```r
sdcApp(maxRequestSize = 50, debug = FALSE, theme = "IHSN", ..., 
  shiny.server = FALSE)
```

## Arguments

- **maxRequestSize** (numeric) number defining the maximum allowed filesize (in megabytes) for uploaded files, defaults to 50MB
- **debug** logical if TRUE, set shiny-debugging options
- **theme** select stylesheet for the interface. Supported choices are
  - 'yeti'
  - 'flatly'
  - 'journal'
  - 'IHSN'
- **...** arguments (e.g. host) that are passed through `runApp` when starting the shiny application
- **shiny.server** Setting this parameter to TRUE will return the app in the form of an object rather than invoking it. This is useful for deploying sdcApp via `shiny-server`. 
Value

starts the interactive graphical user interface which may be used to perform the anonymisation process.

Examples

```r
## Not run:
sdcApp(theme="flatly")

## End(Not run)
```

---

`sdcMicroObj-class`` Class "sdcMicroObj"

---

Description

Class to save all information about the SDC process
undo last changes to `sdcMicroObj`-objects if possible note that this will only work if the user makes use of the `prev` slot or uses the `sdcMicroObj` functions
`strataVar<-` allows to modify the variable which is used if anonymization limitation techniques are applied independent for each characteristic of the defined strata.

Usage

```r
createSdcObj(dat, keyVars, numVars = NULL, pramVars = NULL,
ghostVars = NULL, weightVar = NULL, hhID = NULL, strataVar = NULL,
sensibleVar = NULL, excludeVars = NULL, options = NULL, seed = NULL,
randomizeRecords = FALSE, alpha = 1)

undolast(object)

strataVar(object) <- value
```

```r
## S4 replacement method for signature 'sdcMicroObj,characterOrNULL'
strataVar(object) <- value
```

Arguments

<table>
<thead>
<tr>
<th>Argument</th>
<th>Description</th>
</tr>
</thead>
<tbody>
<tr>
<td><code>dat</code></td>
<td>The microdata set. A numeric matrix or data frame containing the data.</td>
</tr>
<tr>
<td><code>keyVars</code></td>
<td>Indices or names of categorical key variables. They must, of course, match with the columns of <code>dat</code>.</td>
</tr>
<tr>
<td><code>numVars</code></td>
<td>Index or names of continuous key variables.</td>
</tr>
<tr>
<td><code>pramVars</code></td>
<td>Indices or names of categorical variables considered to be pramed.</td>
</tr>
</tbody>
</table>
ghostVars: if specified a list which each element being a list of exactly two elements. The first element must be a character vector specifying exactly one variable name that was also specified as a categorical key variable (keyVars), while the second element is a character vector of valid variable names (that must not be listed as keyVars). If localSuppression or kanon was applied, the resulting suppression pattern for each key-variable is transferred to the depending variables.

weightVar: Indices or name determining the vector of sampling weights.

hhid: Index or name of the cluster ID (if available).

strataVar: Indices or names of stratification variables.

sensibleVar: Indices or names of sensible variables (for l-diversity)

excludeVars: which variables of dat should not be included in result-object? Users may specify a vector of variable-names available in dat that were not specified in either keyVars, numVars, pramVars, ghostVars, hhid, strataVar or sensibleVar.

options: additional options (if specified, a list must be used as input)

seed: (numeric) number specifying the seed which will be set to allow for reproducability. The number will be rounded and saved as element seed in slot options.

randomizeRecords: (logical) if TRUE, the order of observations in the input microdata set will be randomized.

alpha: numeric between 0 and 1 specifying the fraction on how much keys containing NAs should contribute to the frequency calculation which is also crucial for risk-estimation.

object: a sdcMicroObj-class object

value: NULL or a character vector of length 1 specifying a valid variable name

Value

a sdcMicroObj-class object

an object of class sdcMicroObj with modified slot @strataVar

Objects from the Class

Objects can be created by calls of the form new("sdcMicroObj", ...).

Author(s)

Bernhard Meindl, Alexander Kowarik, Matthias Templ, Elias Rut

Examples

showClass("sdcMicroObj")
## Not run:
data(testdata)
sdc <- createSdcObj(testdata,
  keyVars=c('urbrur','roof','walls','water','electcon','relat','sex'),
numVars=c('expend','income','savings'), w='sampling_weight')
head(sdc@manipNumVars)
### Display Risks
sdc@risk$global
sdc <- dRisk(sdc)
sdc@risk$numeric
### use addNoise without Parameters
sdc <- addNoise(sdc, variables=c("expend","income"))
head(sdc@manipNumVars)
sdc@risk$numeric
### undolast
sdc <- undolast(sdc)
head(sdc@manipNumVars)
sdc@risk$numeric
### redo addNoise with Parameter
sdc <- addNoise(sdc, noise=0.2)
head(sdc@manipNumVars)
sdc@risk$numeric
### dataGen
sdc <- undolast(sdc)
#head(sdc@risk$individual)
#sdc@risk$global
sdc <- dataGen(sdc)
#head(sdc@risk$individual)
#sdc@risk$global
### LocalSuppression
sdc <- undolast(sdc)
head(sdc@risk$individual)
sdc@risk$global
sdc <- localSuppression(sdc)
head(sdc@risk$individual)
sdc@risk$global
### microaggregation
sdc <- undolast(sdc)
head(get.sdcMicroObj(sdc, type="manipNumVars"))
sdc <- microaggregation(sdc)
head(get.sdcMicroObj(sdc, type="manipNumVars"))
### pram
sdc <- undolast(sdc)
head(sdc@risk$individual)
sdc@risk$global
sdc <- pram(sdc, keyVar="water")
head(sdc@risk$individual)
sdc@risk$global
### rankSwap
sdc <- undolast(sdc)
head(sdc@risk$individual)
sdc@risk$global
head(get.sdcMicroObj(sdc, type="manipNumVars"))
sdc <- rankSwap(sdc)
head(get.sdcMicroObj(sdc, type="manipNumVars"))
head(sdc@risk$individual)
sdc@risk$global
```r
### suda2
sdc <- suda2(sdc)
sdc@risk$suda2
### topBotCoding
head(get.sdcMicroObj(sdc, type="manipNumVars"))
sdc@risk$numeric
sdc <- topBotCoding(sdc, value=60000000, replacement=62000000, column="income")
head(get.sdcMicroObj(sdc, type="manipNumVars"))
sdc@risk$numeric
### LocalRecProg
data(testdata2)
sdc <- createSdcObj(testdata2,
   keyVars=c("urbrur", "roof", "walls", "water", "sex", "relat"))
sdc@risk$global
sdc <- LocalRecProg(sdc)
sdc@risk$global
### llmodGlobalRisk
sdc <- undolast(sdc)
sdc <- llmodGlobalRisk(sdc, inclProb=0.001)
sdc@risk$model

## End(Not run)

## we can also specify ghost (linked) variables
## these variables are linked to some categorical key variables
## and have the same suppression pattern as the variable that they
## are linked to after \code{\link{localsuppression}} has been applied
data(testdata)
testdata$electcon2 <- testdata$electcon
testdata$electcon3 <- testdata$electcon
testdata$water2 <- testdata$water

keyVars <- c("urbrur", "roof", "walls", "water", "electcon", "relat", "sex")
umVars <- c("expend", "income", "savings")
w <- "sampling_weight"

## we want to make sure that some variables not used as key-variables
## have the same suppression pattern as variables that have been
## selected as key variables. Thus, we are using 'ghost'-variables.
ghostVars <- list()

## we want variables 'electcon2' and 'electcon3' to be linked
## to key-variable 'electcon'
ghostVars[[1]] <- list()
ghostVars[[1]][[1]] <- "electcon"
ghostVars[[1]][[2]] <- c("electcon2","electcon3")

## we want variable 'water2' to be linked to key-variable 'water'
ghostVars[[2]] <- list()
ghostVars[[2]][[1]] <- "water"
ghostVars[[2]][[2]] <- "water2"

## create the sdcMicroObj
```
selectHouseholdData creates a household level file from a dataset with a household structure.

Description

It removes individual level variables and selects one record per household based on a household ID. The function can also be used for other hierarchical structures.

Usage

selectHouseholdData(dat, hhId, hhVars)

Arguments

dat a data.frame with the full dataset
hhId name of the variable with the household (cluster) ID
hhVars character vector with names of all household level variables

Value

A data.frame with only household level variables and one record per household

Note

It is of great importance that users select a variable with containing information on household-ids and weights in hhVars.

Author(s)

Thijs Benschop and Bernhard Meindl
Examples

```r
## ori-hid: household-ids; household_weights: sampling weights for households
x_hh <- selectHouseholdData(dat=testdata, hhId="ori_hid",
    hhVars=c("urbrur", "roof", "walls", "water", "electcon", "household_weights"))
```

Description

modify `sdcMicroObj-class`-objects depending on argument type

Usage

```r
set.sdcMicroObj(object, type, input)
```

Arguments

- **object**: a `sdcMicroObj-class`-object
- **type**: a character vector of length 1 defining what to calculate/return/modify. Allowed types are listed below and the slot with the corresponding name will be replaced by the content of `input`.
  - origData:
  - keyVars:
  - pramVars:
  - numVars:
  - weightVar:
  - hhId:
  - strataVar:
  - sensibleVar:
  - manipPramVars:
  - manipNumVars:
  - manipGhostVars:
  - manipStrataVar:
  - risk:
  - utility:
  - pram:
  - localsuppression:
  - options:
  - prev:
  - set:
  - additionalResults:
  - deletedVars:
- **input**: a list depending on argument type. The content of the list must match the allowed data-type of the slot in the `sdcMicroObj-class`-object that should be replaced.
Value

a `sdcMicroObj-class` object

Examples

```r
sdc <- createSdcObj(testdata2, 
  keyVars=c('urbrur', 'roof', 'walls', 'water', 'electcon', 'relat', 'sex'), 
  numVars=c('expend', 'income', 'savings'), w='sampling_weight')
ind_pram <- match(c("sex"), colnames(testdata2))
get.sdcMicroObj(sdc, type="pramVars")
sdc <- set.sdcMicroObj(sdc, type="pramVars", input=list(ind_pram))
get.sdcMicroObj(sdc, type="pramVars")
```

```
show, sdcMicroObj-method

Show
```

Description

show a sdcMicro object

Usage

```r
## S4 method for signature 'sdcMicroObj'
show(object)
```

Arguments

object an sdcmicro obj

Value

a sdcMicro object

shuffle \hspace{1cm} Shuffling and EGADP

Description

Data shuffling and General Additive Data Perturbation.

Usage

```r
shuffle(obj, form, method = "ds", weights = NULL, covmethod = "spearman", 
  regmethod = "lm", gadp = TRUE)
```
Arguments

obj
An object of class `sdcMicroObj` or a `data.frame` including the data.

form
An object of class “formula” (or one that can be coerced to that class): a symbolic description of the model to be fitted. The responses have to consists of at least two variables of any class and the response variables have to be of class numeric. The response variables belongs to numeric key variables (quasi-identifiers of numeric scale). The predictors are can be distributed in any way (numeric, factor, ordered factor).

method
currently either the original form of data shuffling (“ds” - default), “mvn” or “mlm”, see the details section. The last method is in experimental mode and almost untested.

weights
Survey sampling weights. Automatically chosen when obj is of class `sdcmicroobj-class`.

covmethod
Method for covariance estimation. “spearman”, “pearson” and `dQuotemcd` are possible. For the latter one, the implementation in package robustbase is used.

regmethod
Method for multivariate regression. “lm” and “MM” are possible. For method “MM”, the function “rlm” from package MASS is applied.

gadp
TRUE, if the egadp results from a fit on the original data is returned.

Details

Perturbed values for the sensitive variables are generated. The sensitive variables have to be stored as responses in the argument ‘form’, which is the usual formula interface for regression models in R.

For method “ds” the EGADP method is applied on the norm inverse percentiles. Shuffling then ranks the original values according to the GADP output. For further details, please see the references.

Method “mvn” uses a simplification and draws from the normal Copulas directly before these draws are shuffled.

Method “mlm” is also a simplification. A linear model is applied the expected values are used as the perturbed values before shuffling is applied.

Value

If ‘obj’ is of class `sdcmicroObj-class` the corresponding slots are filled, like `manipNumVars`, risk and utility. If ‘obj’ is of class “data.frame” an object of class “micro” with following entities is returned:

shConf the shuffled numeric key variables

egadp the perturbed (using gadp method) numeric key variables

Note

In this version, the covariance method chosen is used for any covariance and correlation estimations in the whole gadp and shuffling function.
Author(s)  
Matthias Templ, Alexander Kowarik, Bernhard Meindl

References  

See Also  
`rankSwap`, `lm`

Examples  
data(Prestige, package="carData")
form <- formula(income + education ~ women + prestige + type, data=Prestige)
sh <- shuffle(obj=Prestige, form)
plot(Prestige[,c("income", "education")])
plot(sh$sh)
colMeans(Prestige[,c("income", "education")])
colMeans(sh$sh)
cor(Prestige[,c("income", "education")], method="spearman")
cor(sh$sh, method="spearman")

## for objects of class sdcMicro:
data(testdata2)
sdc <- createSdcObj(testdata2,  
  keyVars=c("urbrur", 'roof', 'walls', 'water', 'electcon', 'relat', 'sex'),  
  numVars=c('expend', 'income', 'savings'), w='sampling_weight')
sdc <- shuffle(sdc, method=c('ds'), regmethod= c('lm'), covmethod=c('spearman'),  
  form=savings+expend ~ urbrur+walls)

Description  
allows to restrict original data to only a subset. This may be useful to test some anonymization methods. This function will only be used in the graphical user interface `sdcApp`.

Usage  
`subsetMicrodata(obj, type, n)`
Arguments

- **obj**: an object of class `data.frame` containing micro data
- **type**: algorithm used to sample from original microdata. Currently supported choices are
  - `n_perc`: the restricted microdata will be a `n`-percent sample of the original microdata.
  - `first_n`: only the first `n` observations will be used.
  - `every_n`: the restricted microdata set consists of every `n`-th record.
  - `size_n`: a total of `n` observations will be randomly drawn.
- **n**: numeric vector of length 1 specifying the specific parameter with respect to argument `type`.

Value

an object of class `sdcMicroObj-class` with modified slot `@origData`.

Author(s)

Bernhard Meindl

---

**suda2**

*Suda2: Detecting Special Uniques*

**Description**

SUDA risk measure for data from (stratified) simple random sampling.

**Usage**

`suda2(obj, ...)`

**Arguments**

- **obj**: object of class `data.frame` or a `sdcMicroObj-class` object
- **...**: see arguments below
  - **variables**: Categorical (key) variables. Either the column names or and index of the variables to be used for risk measurement.
  - **missing**: Missing value coding in the given data set.
  - **DisFraction**: It is the sampling fraction for the simple random sampling, and the common sampling fraction for stratified sampling. By default, it’s set to 0.01.
  - **original_scores**: if this argument is `TRUE` (the default), the suda-scores are computed as described in paper "SUDA: A Program for Detecting Special Uniques" by Elliot et al., if `FALSE`, the computation of the scores is slightly different as it was done in the original implementation of the algorithm by the IHSN.
Details

Suda 2 is a recursive algorithm for finding Minimal Sample Uniques. The algorithm generates all possible variable subsets of defined categorical key variables and scans them for unique patterns in the subsets of variables. The lower the amount of variables needed to receive uniqueness, the higher the risk of the corresponding observation.

Value

A modified sdcMicroObj-class object or the following list

- ContributionPercent: The contribution of each key variable to the SUDA score, calculated for each row.
- score: The suda score.
- disscore: The dis suda score
- attribute_contributions: data.frame showing how much of the total risk is contributed by each variable. This information is stored in a data.frame in two variables:
  - variable: containing the name of the variable
  - contribution: contains how much risk a variable contributes to the total risk.
- attribute_level_contributions: shows risks of each attribute-level. this is saved in a data.frame with three columns.
  - variable: containing the name of the variable
  - attribute: holding relevant level-codes and
  - contribution: contains the risk of this level within the variable.)

Note

Since version >5.0.2, the computation of suda-scores has changed and is now by default as described in the original paper by Elliot et al.

Author(s)

Alexander Kowarik and Bernhard Meindl (based on the C++ code from the Organisation For Economic Co-Operation And Development.

For the C++ code: This work is being supported by the International Household Survey Network and funded by a DGF Grant provided by the World Bank to the PARIS21 Secretariat at the Organisation for Economic Co-operation and Development (OECD). This work builds on previous work which is elsewhere acknowledged.

References


M. J. Elliot, A. Manning, K. Mayes, J. Gurd and M. Bane (20xx) SUDA: A Program for Detecting Special Uniques, Using DIS to Modify the Classification of Special Uniques

Examples

```r
## Not run:
data(testdata2)
data_suda2 <- suda2(testdata2, variables=c("urbrur","roof","walls","water","sex"))
data_suda2
str(data_suda2)
sdcmicro(data_suda2)

## for objects of class sdcMicro:
data(testdata2)
sdc <- createSdcObj(testdata2,
  keyvars=c('urbrur','roof','walls','water','electcon','relat','sex'),
  numvars=c('expend','income','savings'), w='sampling_weight')
sdc <- suda2(sdc, original_scores=FALSE)

## End(Not run)
```

---

**summary.freqCalc**

*Summary method for objects from class freqCalc*

**Description**

Summary method for objects of class ‘freqCalc’ to provide information about local suppressions.

**Usage**

```r
## S3 method for class 'freqCalc'
summary(object, ...)```

**Arguments**

- **object**: object from class freqCalc
- **...**: Additional arguments passed through.

**Details**

Shows the amount of local suppressions on each variable in which local suppression was applied.

**Value**

Information about local suppression in each variable (only if a local suppression is already done).
Summary method for objects from class 'micro'.

Usage

## S3 method for class 'micro'
summary(object, ...)

Arguments

object objects from class micro
...

Additional arguments passed through.

Details

This function computes several measures of information loss, such as
**Value**

- **meanx**: A conventional summary of the original data
- **meanxm**: A conventional summary of the microaggregated data
- **amean**: average relative absolute deviation of means
- **amedian**: average relative absolute deviation of medians
- **aonestep**: average relative absolute deviation of onestep from median
- **devvar**: average relative absolute deviation of variances
- **amad**: average relative absolute deviation of the mad
- **acov**: average relative absolute deviation of covariances
- **arcov**: average relative absolute deviation of robust (with mcd) covariances
- **acor**: average relative absolute deviation of correlations
- **arcor**: average relative absolute deviation of robust (with mcd) correlations
- **acors**: average relative absolute deviation of rank-correlations
- **adlm**: average absolute deviation of lm regression coefficients (without intercept)
- **adlts**: average absolute deviation of lts regression coefficients (without intercept)
- **apcaload**: average absolute deviation of pca loadings
- **apppacaload**: average absolute deviation of robust (with projection pursuit approach) pca loadings
- **atotals**: average relative absolute deviation of totals
- **pmtotals**: average relative deviation of totals

**Author(s)**

Matthias Templ

**References**


**See Also**

*microaggregation, valTable*

**Examples**

data(Tarragona)
m1 <- microaggregation(Tarragona, method='onedims', aggr=3)
## summary(m1)
Summary method for objects from class 'pram'

Description

Summary method for objects from class 'pram' to provide information about transitions.

Usage

```r
## S3 method for class 'pram'
summary(object, ...)
```

Arguments

- `object` object from class 'pram'
- `...` Additional arguments passed through.

Details

Shows various information about the transitions.

Value

The summary of object from class 'pram'.

Author(s)

Matthias Templ and Bernhard Meindl

References


See Also

- `pram`

Examples

```r
data(free1)
x <- as.factor(free1[,"MARSTAT")
x2 <- pram(x)
x2
summary(x2)
```
Tarragona data set

Description

A real data set comprising figures of 834 companies in the Tarragona area. Data correspond to year 1995.

Format

A data frame with 834 observations on the following 13 variables.

- **FIXED.ASSETS**  a numeric vector
- **CURRENT.ASSETS**  a numeric vector
- **TREASURY**  a numeric vector
- **UNCOMMITTED.FUNDS**  a numeric vector
- **PAID.UP.CAPITAL**  a numeric vector
- **SHORT.TERM.DEBT**  a numeric vector
- **SALES**  a numeric vector
- **LABOR.COSTS**  a numeric vector
- **DEPRECIATION**  a numeric vector
- **OPERATING.PROFIT**  a numeric vector
- **FINANCIAL.OUTCOME**  a numeric vector
- **GROSS.PROFIT**  a numeric vector
- **NET.PROFIT**  a numeric vector

Source

Public use data from the CASC project.

References


Examples

data(Tarragona)
head(Tarragona)
dim(Tarragona)
Description

A concise (1-5 lines) description of the dataset.

Format

testdata: a data frame with 4580 observations on the following 15 variables.

- `urbrur` a numeric vector
- `roof` a numeric vector
- `walls` a numeric vector
- `water` a numeric vector
- `electcon` a numeric vector
- `relat` a numeric vector
- `sex` a numeric vector
- `age` a numeric vector
- `hhcivil` a numeric vector
- `expend` a numeric vector
- `income` a numeric vector
- `savings` a numeric vector
- `ori_hid` a numeric vector
- `sampling_weight` a numeric vector
- `household_weights` a numeric vector

testdata2: A data frame with 93 observations on the following 19 variables.

- `urbrur` a numeric vector
- `roof` a numeric vector
- `walls` a numeric vector
- `water` a numeric vector
- `electcon` a numeric vector
- `relat` a numeric vector
- `sex` a numeric vector
- `age` a numeric vector
- `hhcivil` a numeric vector
- `expend` a numeric vector
- `income` a numeric vector
Function for Top and Bottom Coding.

Usage

topBotCoding(obj, value, replacement, kind = "top", column = NULL)

Arguments

obj a numeric vector, a data.frame or a sdcMicroObj-class-object
value limit, from where it should be top- or bottom-coded
replacement replacement value.
kind top or bottom
column variable name in case the input is a data.frame or an object of class sdcMicroObj-class.

Details

Extreme values larger or lower than value are replaced by a different value (replacement in order to reduce the disclosure risk.

Value

Top or bottom coded data or modified sdcMicroObj-class.
A Function for the comparison of different microaggregation methods.

Usage

valTable(x, method = c("simple", "onedims", "clustppca", "addNoise: additive", "swappNum"), measure = "mean", clustermethod = "clara", aggr = 3, nc = 8, transf = "log", p = 15, noise = 15, w = 1:dim(x)[2], delta = 0.1)
Arguments

- **x**: data frame or matrix
- **method**: microaggregation methods or adding noise methods or rank swapping.
- **measure**: FUN for aggregation. Possible values are mean (default), median, trim, onestep.
- **clustermethod**: clustermethod, if a method will need a clustering procedure
- **aggr**: aggregation level (default=3)
- **nc**: number of clusters. Necessary, if a method will need a clustering procedure
- **transf**: Transformation of variables before clustering.
- **p**: Swapping range, if method swappNum has been chosen
- **noise**: noise addition, if an addNoise method has been chosen
- **w**: variables for swapping, if method swappNum has been chosen
- **delta**: parameter for adding noise method ‘correlated2’

Details

Tabelarise the output from summary.micro. Will be enhanced to all perturbation methods in future versions.

Value

Measures of information loss splitted for the comparison of different methods.

Methods for adding noise should be named via “addNoise: method”, e.g. “addNoise: correlated”, i.e. the term ‘at first’ then followed by a ‘:’ and a blank and then followed by the name of the method as described in function ‘addNoise’.

Author(s)

Matthias Templ

References


See Also

`microaggregation`, `summary.micro`

Examples

data(Tarragona)
## Not run:
valTable(Tarragona[100:200,],
method=c("simple","onedims","pca","addNoise: additive"))
valTable(Tarragona,
method=c("simple","onedims","pca","clustppca", "mdav","addNoise: additive","swappNum")
## clustppca in combination with Mclust outperforms
## the other algorithms for this data set...

## End(Not run)

---

**varToFactor**

*Change the a keyVariable of an object of class sdcMicroObj-class from Numeric to Factor or from Factor to Numeric*

### Description

Change the scale of a variable

### Usage

```r
varToFactor(obj, var)
varToNumeric(obj, var)
```

### Arguments

- **obj**
  - object of class `sdcMicroObj-class`
- **var**
  - name of the keyVariable to change

### Value

the modified `sdcMicroObj-class`

### Examples

```r
## for objects of class sdcMicro:
data(testdata2)
sdc <- createSdcObj(testdata2,
  keyVars=c('urbrur','roof','walls','water','electcon','relat','sex'),
  numVars=c('expend','income','savings'), w='sampling_weight')
sdc <- varToFactor(sdc, var="urbrur")
```
writeSafeFile

Description

writes an anonymized dataset to a file. This function should be used in the graphical user interface sdcApp only.

Usage

writeSafeFile(obj, format, randomizeRecords, fileOut, ...)

Arguments

obj an object of class data.frame containing micro data
format (character) specifies the output file format. Accepted values are:
  • 'rdata' output will be saved in the R binary file-format.
  • 'sav' output will be saved as SPSS-file.
  • 'dta' output will be saved as STATA-file.
  • 'csv' output will be saved as comma seperated (text)-file.
  • 'sas' output will be saved as SAS-file (sas7bdat).
randomizeRecords (logical) specifies, if the output records should be randomized. The following options are possible:
  • 'no' default, no randomization takes place
  • 'simple' records are just randomly swapped.
  • 'byHH' if slot 'hhId' is not NULL, the clusters defined by this variable are randomized across the dataset. If slot 'hhId' is NULL, the records or the dataset are randomly changed.
  • 'withinHH' if slot 'hhId' is not NULL, the clusters defined by this variable are randomized across the dataset and additionally, the order of records within the clusters are also randomly changed. If slot 'hhId' is NULL, the records or the dataset are randomly changed.
fileOut (character) file to which output should be written
... optional arguments used for write.table if argument format equals csv

Author(s)

Bernhard Meindl
Index

*Topic **aplot**
  plot.localSuppression, 58
  plotMicro, 60

*Topic **classes**
  plot.sdcMicroObj, 59
  print.sdcMicroObj, 69
  sdcMicroObj-class, 78

*Topic **datasets**
  casc1, 14
  CASCrefmicrodata, 15
  EIA, 22
  francdat, 26
  free1, 27
  microData, 55
  Tarragona, 93
  testData, 94

*Topic **manip**
  addGhostVars, 8
  addNoise, 10
  dataGen, 16
  dRisk, 18
  dRiskRMD, 19
  dUtility, 21
  freqCalc, 29
  globalRecode, 32
  indivRisk, 35
  LLmodGlobalRisk, 38
  LocalRecProg, 39
  localSupp, 41
  localSuppression, 42
  mafast, 45
  measure_risk, 46
  microaggregation, 51
  modRisk, 56
  pram, 61
  riskyCells, 75
  shuffle, 84
  suda2, 87
  topBotCoding, 95

*Topic **methods**
  groupAndRename, 34
  removeDirectID, 74
  report, 74
  varToFactor, 98

*Topic **package**
  sdcMicro-package, 4

*Topic **print**
  measure_risk, 46
  print.freqCalc, 64
  print.indivRisk, 65
  print.localSuppression, 66
  print.micro, 67
  print.modRisk, 68
  print.pram, 68
  print.suda2, 70
  summary.freqCalc, 89
  summary.micro, 90
  summary.pram, 92
  valTable, 96

  addGhostVars, 8
  addNoise, 10
  argus_microaggregation, 12
  argus_rankswap, 13
  calcRisks, 13
  casc1, 14
  CASCrefmicrodata, 15
  cov, 16
  createNewID, 16
  createSdCObj(sdcMicroObj-class), 78
  cut, 33
  
data.frame, 87, 99
  dataGen, 16
  dRisk, 18, 20, 22
  dRiskRMD, 19, 22
  dUtility, 18, 21

dEIA, 22
INDEX

101

extractManipData, 25
francdat, 26
free1, 27
freq, 28
freqCalc, 29, 36, 41, 48, 64, 65, 90
generateStrata, 31
globalRecode, 32
groupAndRename, 34
importProblem, 35
indivRisk, 30, 35, 41, 48, 65, 96
kAnon, 79
kAnon (localSuppression), 42
kAnon_violations, 37
kAnon_violations, sdcMicroObj, logical, numeric
(kAnon_violations), 37
ldiversity (measure_risk), 46
LLmodGlobalRisk, 38
lm, 86
LocalRecProg, 39
localSupp, 41
localSuppression, 42, 59, 66, 79
loglm, 39, 57
mafast, 45, 52
maxCat, 55
measure_risk, 30, 36, 39, 46, 48, 57
mergeHouseholdData, 49
microaggregation, 45, 51, 61, 67, 91, 97
microaggregOver, 54
microData, 55
modRisk, 39, 56, 68
modrisk, 68
modrisk (print.modrisk), 68
nextSdcObj, 58
plot.localSuppression, 58
plot.sdcMicroObj, 59
plotMicro, 53, 60
pram, 61, 68, 69, 92
print, sdcMicroObj-method
(print.sdcMicroObj), 69
print.freqCalc, 64
print.indivRisk, 65
print.ldiversity (measure_risk), 46
print.localSuppression, 66
print.measure_risk (measure_risk), 46
print.micro, 67
print.modrisk, 68
print.pram, 68
print.sdcMicroObj, 69
print.suda2, 70
rankSwap, 71, 86
readMicrodata, 73
removeDirectID, 74
report, 74
riskyCells, 75
runApp, 77
sampleCat, 55
sdcApp, 35, 73, 77, 86, 99
sdcMicro(sdcMicro-package), 4
sdcMicro-package, 4
sdcMicroObj-class, 34, 59, 69, 78, 98
selectHouseholdData, 50, 82
set.sdcMicroObj, 83
show, sdcMicroObj-method, 84
shuffle, 17, 84
strataVar<- (sdcMicroObj-class), 78
strataVar<-, sdcMicroObj, characterOrNULL-method
(sdcMicroObj-class), 78
subsetMicrodata, 86
suda2, 71, 87
summary.freqCalc, 89
summary.micro, 11, 53, 90, 97
summary.pram, 92
Tarragona, 93
testdata, 94
testdata2 (testdata), 94
topBotCoding, 95
undolast (sdcMicroObj-class), 78
valTable, 53, 91, 96
varToFactor, 98
varToNumeric (varToFactor), 98
writeSafeFile, 99