Chapter 1
Software

Abstract The methods used in this book are exclusively available in the software environment R (R Development Core Team 2014). A very brief introduction to some functionalities of R is given. This introduction does not replace a general introduction to R but shows some points that are important in order to understand the examples and the R code in the book. The package **sdcMicro** (Templ et al. 2015) forms a basis for this book and it includes all presented SDC methods. It is free, open-source and available from the comprehensive R archive network (CRAN). This package implements popular statistical disclosure methods for risk estimation such as the suda2-algorithm, the individual risk approach or risk measurement using log-linear models. In addition, perturbation methods such as global recoding, local suppression, post-randomization, microaggregation, adding correlated noise, shuffling and various other methods are integrated. With the package **sdcMicro**, statistical disclosure control methods can be applied in an exploratory, interactive and user-friendly manner. All results are saved in a structured manner and these results are updated automatically as soon a method is applied. Print and summary methods allow to summarize the status of disclosure risk and data-utility as well as reports can be generated in automated manner. In addition, most applications/anonymizations can be carried out with the point-and-click graphical user interface (GUI) **sdcMicroGUI** (Kowarik et al. 2013) without knowledge in the software environment R or the newer version of **sdcMicroGUI**, an app that is available within the package **sdcMicro** as function `sdcApp`. The new version runs in a browser and is based on **shiny** (Chang et al. 2016). A software package with a similar concept as **sdcMicro**—the **simPop** package (Templ et al. 2017)—is used to generate synthetic data sets.

1.1 Prerequisites

R can be used as an overgrown calculator. All operations of an calculator can be very easily used also in R, e.g. addition is done with +, subtraction with -, division with /, exponential with `exp()`, logarithm with `log()`, square root `sqrt()`, sinus `sin()`, …... All operations work as expected, e.g. the following expression is parsed by R, inner brackets are solved first, multiplication and division operators have precedence over the addition and subtraction operators, etc.

© Springer International Publishing AG 2017
M. Templ, *Statistical Disclosure Control for Microdata*,
DOI 10.1007/978-3-319-50272-4_1

```
5 + 2 * log(3 * 3)

   ## [1] 9.394449
```

R is a function and object-oriented language. Functions can be applied to objects. The syntax is as shown in the following example

```
mean(rnorm(10))

   ## [1] -0.2763877
```

With the function `rnorm` 10 numbers are drawn from a standard normal distribution. Afterwards the `mean` is calculated for these 10 numbers. Functions typically have function arguments that can be set. The syntax for calling a function is in general

```
res1 <- name_of_function(v1) # an input argument
res2 <- name_of_function(v1, v2) # two input arguments
res3 <- name_of_function(v1, v2, v3) # three input arguments
# ...
```

Functions often have additional function arguments with default values You get access to all function arguments with `args()`

```
require(sdcMicro)
args(addNoise)

   ## function (obj, variables = NULL, noise = 150, method = "additive",
   ##     ...)
   ## NULL
```

The help for a function can be found with

```
?addNoise
```

Allocation to objects are made by <- or = and the generated object can be printed via object name followed by typing `ENTER`

```
x <- rnorm(10)
x

   ##  [1]  1.1908289  0.4773820 -1.0320670  2.5720002 -0.3302985
   ##  [6]  0.1175549  0.7655485 -0.4642652  0.3261664  0.1262540
```

Please note that R is case sensitive.

1.1.1 Installation and Updates

If R is already installed on the computer, ensure that you work with the current version of R. If the software is not installed, go to http://cran.r-project.org/bin/ and choose your platform. For Windows, just download the executable file and follow the on-screen instructions.

1.1.2 Install sdcMicro and Its Browser-Based Point-and-Click App

Open R on your computer and type:

```
install.packages("sdcMicro")
install.packages("simPop")
```

Installation is needed only once. Note that (only) the GUI requires the `GTK+` package to draw windows. When installing **sdcMicroGUI** (`install.packages("sdcMicroGUI")`), all required packages (including `GTK+`) are automatically installed if the user have sufficient system administration rights. From version 4.7.0 onwards, the GUI is included in package **sdcMicro** and started with `sdcGUI()`. From version 5.0.0 onwards the GUI is replaced by a shiny app. The browser-based app can be started via

```
sdcApp()
```

1.1.3 Updating the SDC Tools

Typing `update.packages()` into R searches for possible updates and installs new versions of packages if any are available. For the **sdcMicroGUI**, users can also click on the menu item *GUI* → *Check for Updates*; this should be done regularly.

The previous information was related to install the stable CRAN version of the packages. However, latest changes are only available in the development version of the package. This is hosted on https://github.com/alexkowa/sdcMicro and includes test batteries to ensure that the package keeps stable when modifying parts of the package. From time to time, a new version is uploaded to CRAN.

To install the latest development version, the installation of the package **devtools** (Wickham and Chang 2015) is needed. After calling the **devtools** package, the development versions can be installed via `install_github()`

```
require("devtools")
install_github("alexkowa/sdcMicro")
```

1.1.4 Help

It is crucial to have basic knowledge about getting help. With

```
help.start()
```

your browser opens and help (and more) is available.

The clickable help index of the package can be accessed by typing the following command into R.

```
help(package=sdcMicro)
```

To find a specific help of a function, one can use `help(name)` or `?name`. As an example, we look at the help file of function `rankSwap`, which is included in the package **sdcMicro**

```
library(sdcMicro)
?rankSwap
```

Data in the package can be loaded via the `data()` function.

```
data(testdata)
```

`help.search()` can be used to find functions for which you don't know its exact name, e.g.

```
help.search("adding noise")
```

will search your local R installation for functions matching the character string `"adding noise"` in the (file) name, alias, title, concept or keyword entries. With function `apropos` one can find and list objects by (partial) name. For example, to list all objects with partial name match `risk`:

```
apropos("risk")

   ## [1] "calcRisks"        "dRisk"            "dRiskRMD"
   ## [4] "indivRisk"        "LLmodGlobalRisk"  "measure_risk"
   ## [7] "modRisk"          "risk"
```

It can be seen that some useful function names such as `measure_risk` or `calcRisks` are listed.

To search help pages, vignettes or task views, using the search engine at http://search.r-project.org and to view them in your web browser, you can use

```
RSiteSearch("adding noise")
```

which reports all search results for the character string `"adding noise"`.

1.1.5 The R Workspace and the Working Directory

Created objects are available in the workspace of R and loaded in the memory of your computer. The collection of all created objects is called *workspace*. To list the objects in the workspace

```
ls()

   ## character(0)
```

When importing or exporting data, the working directory must be defined. To show the current working directory, the function `getwd` can be used

```
getwd()

   ## [1] "/Users/teml/workspace/sdc-springer/book"
```

To change the working directory, the function `setwd` is the choice

```
# paste creates a string
p <- paste(getwd(), "/data", sep="")
p

   ## [1] "/Users/teml/workspace/sdc-springer/book/data"

# now change the working directory
setwd(p)
# has it changed? Yes...
getwd()

   ## [1] "/Users/teml/workspace/sdc-springer/book/data"
```

1.1.6 Data Types

The objective is to know the most important data types: numeric, character and logical as well as the data structures

- vectors/factors
- lists
- data frames
- special data types: missing values, NULL-objects, NaN, $-/+$ Inf

Vectors are the simplest data structure in R. A vector is a sequence of elements of the same type such as numerical vectors, character vectors, logical vectors Vectors are often created with the function `c()`, e.g.

```
v.num <- c(1,3,5.9,7)
v.num

   ## [1] 1.0 3.0 5.9 7.0

is.numeric (v.num)

   ## [1] TRUE
```

`is.numeric` query if the vector is of class numeric. Note that characters are written with parenthesis.

Logical vectors are often created indirectly from numerical/character vectors

```
v.num > 3

   ## [1] FALSE FALSE  TRUE  TRUE
```

Many operations on vectors are performed element-wise, e.g. logical comparisons or arithmetic operations with vectors. A common error source is when the length of vectors differ. Then the shorter one is repeated (*recycling*)

```
v1 <- c(1,2,3)
v2 <- c(4,5)
v1 + v2

   ## [1] 5 7 7
```

One should also be aware that R coerces internally to meaningful data types automatically. For example,

```
v2 <- c (100, TRUE, "A", FALSE)
v2

   ## [1] "100"   "TRUE"  "A"     "FALSE"

is.numeric (v2)

   ## [1] FALSE
```

Here the lowest common data type is a string and therefore all entries of the vector are coerced to character. Note, to create vectors, the functions `seq` and `rep` are very useful.

Often it is necessary to subset vectors. The selection is made using the `[]` operator. A selection can be done in three ways

positive: a vector of positive integers that specifies the position of the desired elements
negative: a vector with negative integers indicating the position of the non-required elements
logical: a logic vector in which the elements are to be the selected (TRUE), and those who are not selected (FALSE).

```
require("laeken")
data("eusilc")
# extract for 10 observations of variable age from eusilc data
age <- eusilc[1:10, "age"]
age

   ##  [1] 34 39  2 38 43 11  9 26 47 28

# positive indexing:
age[c(3,6,7)]

   ## [1]  2 11  9

# negative indexing:
age[-c(1,2,4,5,8:10)]

   ## [1]  2 11  9

# logical indexing:
age < 15

   ##  [1] FALSE FALSE  TRUE FALSE FALSE  TRUE  TRUE FALSE FALSE FALSE

# a logical expression can be written directly in []
age[age < 15]

   ## [1]  2 11  9
```

A list in R is a *ordered* collection of objects whereas each object is part of the list and where the data types of the individual list elements can be different (vectors, matrices, data.frames, lists, etc.). The dimension of each list item can be different. Lists can be used to group and summarize various objects in an object. There are (at least) three ways accessing elements of a list, (a) the `[]` -operator, the operator `[[]]`, the `$` -operator and the name of a list item. With `str())`, you can view the structure of a list, with `names()` you get the names of the list elements

```
## measure risk on data frames
data(Tarragona)
x <- Tarragona[, 5:7]
## add noise
y <- addNoise(x)
## estimate the risk, result is a list
risk <- dRiskRMD(x, xm=y$xm)
class(risk)

   ## [1] "list"

str(risk)

   ## List of 8
   ##  $ risk1     : num 0.0048
   ##  $ risk2     : num 0.0048
   ##  $ wrisk1    : num 0.0117
   ##  $ wrisk2    : num 0.0117
   ##  $ indexRisk1: Named int [1:4] 154 160 744 812
   ##   ..- attr(*, "names")= chr [1:4] "154" "160" "744" "812"
   ##  $ indexRisk2: int [1:4] 154 160 744 812
   ##  $ riskvec1  : num [1:834] 0 0 0 0 0 0 0 0 0 0 ...
   ##  $ riskvec2  : num [1:834] 0 0 0 0 0 0 0 0 0 0 ...

names(risk)

   ## [1] "risk1"      "risk2"      "wrisk1"     "wrisk2"
   ## [5] "indexRisk1" "indexRisk2" "riskvec1"   "riskvec2"

## access elements from the named list
risk$wrisk2

   ## [1] 0.01172644
```

Factors in R are of special importance. They are used to represent nominal or ordinal data. More precisely, unordered factors for nominally scaled data and ordered factors for ordinal scaled data. Factors can be seen as special vectors. They are internally coded integers from 1 to n (# of occurrences) which are all associated with a name (label). So when and why variables should be stored as class factor? Basically, factors has to be used for categorical information to get the correct number of degrees of freedom and correct design matrices in statistical modeling. In addition, the implementation of graphics for factors versus numerical/character vectors differ. Moreover, factors are more efficient in storing of character vectors However, factors have a more complex data structure since factors include a numerically coded data vector and labels for each level/category.

```
## access a vector with the dollar operator
gender <- eusilc$rb090
## first six values:
head(gender)

   ## [1] female male   male   female male   male
   ## Levels: male female
```

Data frames (in R `data.frame`) are the most important data type. This corresponds to the well-known from other software packages rectangle data format with *rows* correspond to observation units and *columns* to variables. A `data.frame` is like a `list` whereas all list elements are vector/factors but with the restriction that all list elements have the same number of elements (equal length). For example, data from external sources to be read are often stored as data frames, i.e. data frames are usually created by reading data but they can also be constructed with function `data.frame()`.

A lot of opportunities exist to subset a data frame, e.g. with syntax: `[`*index row*`,` *index columns*`]`. Again positive, negative and logical indexing is possible and the type of indexing may be different for row index and column index. To access to individual columns is most easy with the `$` -operator (like lists).

```
## babies of age 1 living in households of size 2:
babies1 <- eusilc$age == 1 & eusilc$hsize == 2
str(babies1)

   ##  logi [1:14827] FALSE FALSE FALSE FALSE FALSE FALSE ...

## select some variables, e.g. including
## family/children related allowances
cn <- colnames(eusilc) %in% c("rb090", "db040", "hy050n")
str(cn)

   ##  logi [1:28] FALSE FALSE TRUE FALSE FALSE TRUE ...

eusilc[babies1, cn]

   ##         db040  rb090  hy050n
   ## 6333  Styria female 2009.54
   ## 13860 Vienna   male 1897.37

## or for short:
eusilc[eusilc$age == 1 & eusilc$hsize == 2, c("rb090", "db040", "hy050n")]

   ##         rb090  db040  hy050n
   ## 6333  female Styria 2009.54
   ## 13860   male Vienna 1897.37
```

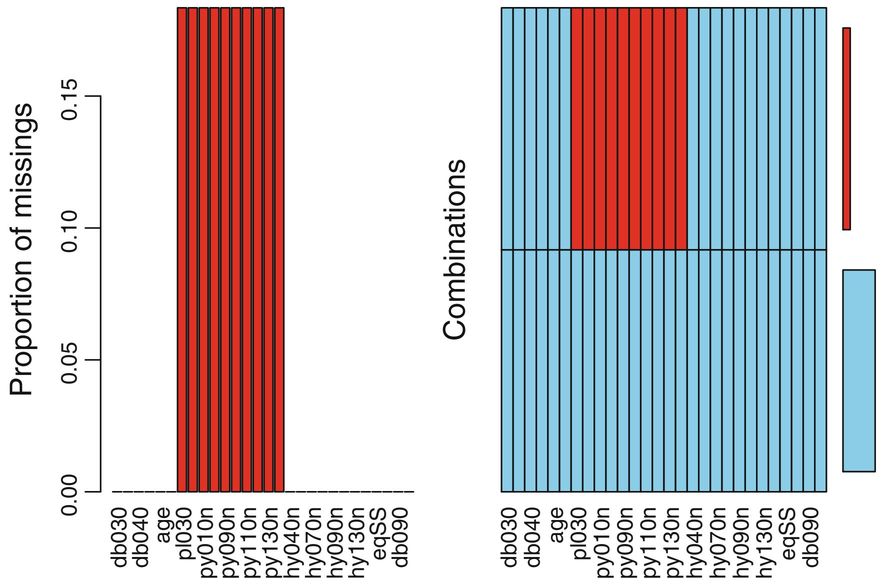


Fig. 1.1 Missing patterns in the EU-SILC data set. *Left* proportion of missing values in each variable. *Right* Missing value patterns

A few helpful functions that can be used in conjunction with data frames are `dim()`, reporting the dimension (number of rows and columns), `head()`, the first (default 6) rows of a data frame, `colnames()`, the columns/variable names

Missing values are almost always present in the data. The default representation of missing value in R is the symbol `NA`. A very useful function to check if data values are missing is `is.na`. It returns a logical vector or data.frame depending if the input is a vector or data.frame indicating missingness. To calculate the number of missing values, we could sum the TRUE's (interpreted as 1 while FALSE is interpreted as 0).

```
sum(is.na(eusilc))

   ## [1] 27200
```

All in all, 27200 values are missing, this is approx. 6.55% of the data values.

```
27200 / (nrow(eusilc) * ncol(eusilc)) * 100

   ## [1] 6.551754
```

To analyse the structure of missing values, the R package **VIM** (Templ et al. 2012). For the EU-SILC data set we immediately see the source of missingness, see Fig. 1.1.

```r
require("VIM")
aggr(eusilc)
```

More precisely, we see the monotone missingness in the personal income components of the European Union Statistics on Income and Living Conditions (EU-SILC) data set. The proportion of missings in this variables is almost 20% (see left plot). In the right plot of Fig. 1.1 the patterns of missingness are visible. Most of the observations have no missings, the rest have missing values in each of the personal income components. In fact, these are children that have no personal income.

Before applying SDC methods it is recommended to look at the structure of missingness since missing values has effect on risk measurement. More information on missing values can be found in Templ et al. (2012).

1.1.7 Generic Functions, Methods and Classes

These topics are only discussed very briefly since they are more advanced. However, since **sdcMicro** is highly object-oriented some basics are good to know. The use of object-orientation makes implementations in R very user-friendly. First we replicate some basic concepts about classes in R.

R has different class systems, the most important ones are S3 and S4 classes. Programming with S3 classes is lazy living, it is simpler to program in S3 than in S4. However, S4 is more *clean* and the use of S4 can make packages very user-friendly.

In any case, in R each object is assigned to a class (attribute *class*) Classes allow object-oriented programming and *overloading* of *generic functions*. Generic functions produce different output for objects different classes as soon as methods are written for such classes.

This sounds complex, but with the following example it should get clearer.

As an example of a generic function, we use the function `summary`. `summary` is a generic function used to produce result summaries. The function invokes particular methods which depend on the class of the first argument.

```
## how often summary is overloaded with methods
## on summary for certain classes
length(methods(summary))

   ## [1] 175

class(eusilc$hsize)

   ## [1] "integer"

summary(eusilc$hsize)

   ##    Min. 1st Qu.  Median    Mean 3rd Qu.    Max.
   ##   1.000   2.000   3.000   3.234   4.000   9.000

## convert it to a factor
eusilc$hsize <- as.factor(eusilc$hsize)
class(eusilc$hsize)

   ## [1] "factor"

summary(eusilc$hsize)

   ##    1    2    3    4    5    6    7    8    9
   ## 1745 3624 3147 3508 1815  630  252   88   18
```

From this previous example one can see that the summary is different, depending on the class of the object. R internally looks if a method is implemented for the given class of the object. If yes, this function is used, if not, the function `summary.default` is used. This procedure is called *method dispatch*. In the previous example, last line, R looked if a function `summary.factor` is available, which was true.

Note that generic functions can be defined, and we also can define print, summary and plot methods for objects of certain classes.

As mentioned before, S4 classes are different and more formal. The important fact to know when working with **sdcMicro** is how to access elements of S4 class objects.

In the next code, a S4 class object of class *sdcMicroObj* is generated.

```
sdc <- createSdcObj(eusilc,
  keyVars=c('age','rb090','db040','hsize','pb220a'),
  numVars=c('eqIncome'), w='rb050')
class(sdc)

   ## [1] "sdcMicroObj"
   ## attr(,"package")
   ## [1] "sdcMicro"
```

The defined functions for S3 classes, such as `names()` or `head()`, does not longer work. For example, the replacement for `names()` is `slotNames()`

```
slotNames(sdc)

   ##  [1] "origData"          "keyVars"           "pramVars"
   ##  [4] "numVars"           "ghostVars"         "weightVar"
   ##  [7] "hhId"              "strataVar"         "sensibleVar"
   ## [10] "manipKeyVars"      "manipPramVars"     "manipNumVars"
   ## [13] "manipGhostVars"    "manipStrataVar"    "originalRisk"
   ## [16] "risk"              "utility"           "pram"
   ## [19] "localSuppression"  "options"           "additionalResults"
   ## [22] "set"               "prev"              "deletedVars"
```

In addition, the $ operator do not longer work, but its S4 counterpart–the slot–must be used. For example to access the slot `risk` we use `sdc@risk` This slot contains a list and this list contains again a list, that can be also accessed

```
str(sdc@risk)

   ## List of 3
   ##  $ global    :List of 5
   ##   ..$ risk      : num 0.00224
   ##   ..$ risk_ER   : num 33.1
   ##   ..$ risk_pct  : num 0.224
   ##   ..$ threshold: logi NA
   ##   ..$ max_risk  : num 0.01
   ##  $ individual: num [1:14827, 1:3] 0.001961 0.012359 0.000495 ...
   ##   ..- attr(*, "dimnames")=List of 2
   ##   .. ..$ : NULL
   ##   .. ..$ : chr [1:3] "risk" "fk" "Fk"
   ##  $ numeric    : num 1

str(sdc@risk$global)


   ## List of 5
   ##  $ risk      : num 0.00224
   ##  $ risk_ER   : num 33.1
   ##  $ risk_pct  : num 0.224
   ##  $ threshold: logi NA
   ##  $ max_risk  : num 0.01

sdc@risk$global$risk_pct


   ## [1] 0.2235023
```

To make the life easier, there are accessor functions defined for working with objects of class `sdcMicro`, see Sect. 1.4.3.

1.2 Brief Overview on SDC Software Tools

μ-Argus

Under the 5-th framework programme of the European Union, a general tool for the anonymization of micro-data, μ-**Argus**, has been improved. The software is still developed by Statistics Netherlands and other partners, and some of those extensions are being subsidized by Eurostat. It features a graphical point and click user interface, which was based on Visual Basic until version 4.2. and is now (Version 5.1 and onwards) written using Java and can be downloaded for free from the CASC website (http://neon.vb.cbs.nl/casc/mu.htm). Currently, only 32-bit versions have been built, and there is no command line interface available.

C++ Code from the International Household Survey Network

Previous efforts from the International Household Survey Network (IHSN) include the development of microdata anonymization software. IHSN developed `C++` code for microdata anonymization in order to support the safe dissemination of confidential data. In addition, plug-ins were developed to call the code from statistical software, namely from `Stata`, `SPSS` and `SAS`. While the software developed from the IHNS is free and open-source, the use of the code from the previous mentioned statistical software is restricted for commercial use since the users have to buy a license for the statistical software. The IHSN code is fully integrated (and improved) in **sdcMicro**.

sdcMicro and sdcMicroGUI

Package **sdcMicroGUI** (Kowarik et al. 2013) provides a graphical user interface for **sdcMicro** and it serves as an easy-to-handle, highly interactive tool for users who want to use the **sdcMicro** package for statistical disclosure control but are not familiar with the native R command line interface. The GUI performs automated recalculation and display of frequency counts, individual and global risk measures, information loss and data utility after each anonymization step. Changes to risk and utility measurements of the original data are also conveniently displayed in the graphical user interface (GUI) and the code is saved in a script, which can easily be exported, modified and re-used, making it possible to reproduce any results. Currently, a new re-implementation of **sdcMicroGUI** will be made using **shiny** (Chang et al. 2016). The aim is that methods can be used in a browser. The package should be available in autumn 2016.

Other tools for data from biomedical sciences

Biomedical data sets have (usually) a simple structure, i.e. categorical data, few key variables, no complex designs as well as no hierarchical or cluster structures. Therefore, only simple tools for measuring the disclosure risk and simple tools for perturbing the data sets are in use.

With **TIAMAT** (Dai et al. 2009) different k-anonymization techniques can be compared. Also the **eCPC toolkit** from the Swedish eScience Research Center (SeRC) and the **UTD Anonymization ToolBox** developed by the Data Security and Privacy Lab at the University of Texas at Dallas allows investigations in k-anonymity. They all are based on the Mondrian algorithm (LeFevre et al. 2006) to achieve k-anonymity. **AnonTool**, developed by the department of Department of Informatics Systems at the Alpen-Adria-Universität Klagenfurt, provides two methods: k-anonymity and l-diversity (see for both Sect. 3.3). All specifications are given by an XML file. **Arx** (Kohlmayer et al. 2012) is implemented in Java. The anonymization in **Arx** consists of three basic steps, first to configure the anonymization process. Second, to explore the so-called solution space and third, analyzing the perturbed data. The tool is useful for k-anonymity, l-diversity and similar approaches such as t-closeness or δ-presence (Nergiz et al. 2007). It provides interactive features to investigate in the information loss based on univariate summaries of the original and perturbed data.

1.3 Differences Between SDC Tools

Table 1.1 gives an overview of software and available methods of four concrete software products—the μ-**Argus** software (Hundepool et al. 2008) from Statistics Netherlands, the R packages **sdcMicro** (Templ et al. 2015) and **sdcMicroGUI** (Kowarik et al. 2013) and C++ code that was written by the IHSN (see http://www.ihsn.org/home/node/32). In addition, the tools available from the biomedical area (such as **Arx**) are summarized in the field “Biomed Tools”.

From the “Biomed” tools, **Arx** is the most powerful one due to its well-designed point and click graphical user-interface. In addition, it has more methods integrated as other tools in the biomedical area. However, it cannot deal with data from complex designs and has limited features apart from frequency-based procedures.

The difference of μ-**Argus** and **sdcMicro** is not only with the amount of supported methods. The main advantages of **sdcMicro** is its user-friendly object-oriented application, the ease of importing data (in μ-**Argus** an import script which determines the hierarchical structure of data has to be written) and its flexibility to use. Note that reproducibility, a lot of interactive features (e.g., automatic code generation) and computational optimized code as well as automatic recalculations are not provided by μ-**Argus**. In addition, many methods are missing in μ-**Argus** and some methods are implemented in an oversimple manner. For example, SUDA, shuffling or model-based risk estimation are not available in μ-**Argus**. Or one very central method is local suppression with the aim to provide data that fulfills k-anonymity (see Sect. 3.3). When having for example seven key variables defined, in μ-**Argus** subsets of key variables (typically up to all combinations of four variables) are made k-anonym, but k-anonymity is usually not fulfilled for the given seven key variables. In **sdcMicro** this subset approach can be applied, but it is also possible to ensure k-anonymity for all seven variables. Since methods can not be applied using a command line interface in μ-**Argus** we can not compare computation speed of **sdcMicro** but we can state that μ-**Argus** is not suitable for large data sets. It becomes slow and

Table 1.1 List of methods supported by different statistical disclosure control software. Ticks in brackets indicate only limited implementation of a method. The table is originally published in Templ et al. (2015). Published with kind permission of ©Matthias Templ, Alexander Kowarik and Bernhard Meindl 2015 under the terms of the creative commons attribution license which permits any use, distribution, and reproduction in any medium provided the original author(s) and source are credited.

Method | Software	μ-**Argus** 4.2	**BioMed tools**	**sdcMicro** > 4.3.0	**sdcMicroGUI** > 1.1.0	**IIHSN**
frequency counts	✔	✔	✔	✔	✔
individual risk (IR)	✔		✔	✔	✔
IR on households	✔		✔	✔	✔
l-diversity		✔	✔	✔	✔
suda2			✔		✔
global risk (GR)	✔		✔	✔	✔
GR with log-lin mod.			✔		
recoding	✔	✔	✔	✔	(✔)
local suppression	(✔)	(✔), Arx: ✔	✔	✔	(✔)
swapping	(✔)		✔		✔
pram	✔		✔	✔	✔
adding correlated noise			✔	✔	✔
microaggregation	✔		✔	✔	✔
shuffling			✔	✔	
utility measures	(✔)	Arx: (✔)	✔	✔	
GUI	(✔)	✔		✔	
CLI		Arx: ✔	(✔)		✔
reporting	✔		✔	✔	
platform independent		Arx: ✔	✔	✔	✔
free and open-source		Arx: ✔	✔	✔	✔

runs out-of-memory even with medium-sized data sets while with **sdcMicro** huge data sets up to millions of observations can be processed.

An exercise in computation time is shown in Fig. 1.2. We investigate in the computation time of the most computer-intense methods, local suppression and risk measurement. **sdcMicro** and IHSN code is compared, other software either do not have a command line interface for comparison (so as μ-**Argus**), the tested procedures are not available (**Arx**) or the procedures are such slow (also μ-**Argus**) that it makes no sense to put it in this figure. Figure 1.2 the performance increase with respect to computation time of the current version of **sdcMicro**, 5.0.0 compared to the previous implementation in **sdcMicro** (<version 4.1.0) using IHSN C++ code is shown. To measure the computation time, the Bangladesh Survey on Income (I2D2) with 48969 observations on 41 variables was used. However, to increase the size of the data, the observations were randomly replicated to enlarge the data set up to

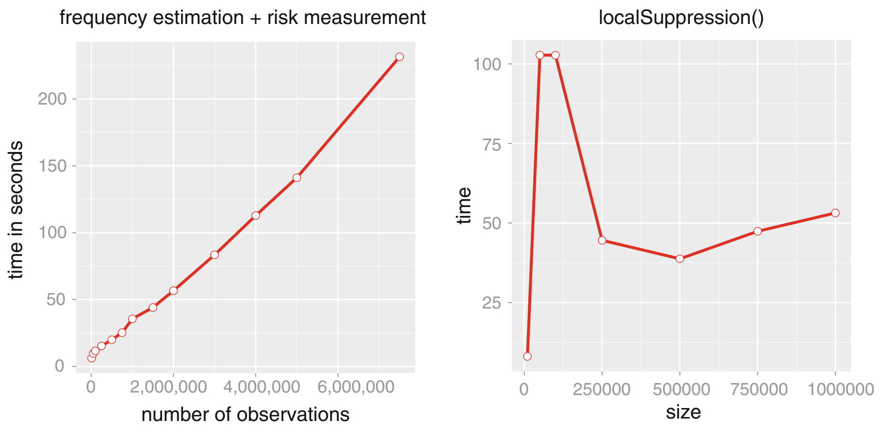


Fig. 1.2 Computation time of IHSN C++ code (equals sdcMicro version $< 4.1.0$) and sdcMicro (version $\geq 4.1.0$)

7,500,000 observations. For different numbers of observations, frequency counts and risk (left plot in Fig. 1.2) as well as (heuristic "optimal") local suppression (right plot in Fig. 1.2) were applied independently for each data set.

While for the IHSN C++ solutions the computation time is exponential regarding the number of observations, the new implementation features a "lower than linear growth" for local suppression. The decrease in computation time between 100.000 and 500.000 observations can be explained with the internal use of measuring k-anonymity (see Sect. 3.3) in the implementation of the local suppression method in **sdcMicro**. The size of the data set is reduced internally, because observations with frequencies larger k can be omitted from the analysis since they do not have any risk. Thus, the larger the data set the more observation can be omitted for calculations for this method.

The figures clearly show that the **sdcMicro** can also be applied on very large data sets. Of course, there are also some experimental methods included with much higher demand on computation time. However, the main features of **sdcMicro** are efficiently implemented.

1.4 Working with sdcMicro

For each method discussed we additionally show its usage via the command line interface of **sdcMicro**. The application of (most) methods is also possible using the graphical user interface of package **sdcMicroGUI**. To open the GUI type

```
require("sdcMicroGUI")
sdcGUI()
```

This will load the **sdcMicroGUI** package into R and display the point-and-click GUI. If you have not installed **sdcMicroGUI**, you will see an error message; follow the steps described in Sect. 1.1.1 to install the package.

However, the GUI is not in focus of the book but it can be used to replace most of the command line instructions in this book just by point-and-click in the GUI.

1.4.1 General Information About sdcMicro

The first version, version 1.0.0, of the **sdcMicro** package was released in 2007 on the comprehensive R archive network (`http://cran.r-project.org`) and introduced in Templ (2008). However, this version only included few methods and the package consisted of a collection of some functions that were only applicable to small data sets. The current release, version 5.0.0, is a huge step forward. Almost all methods are implemented in an object-oriented manner (using S4 classes) and have been written internally either by implementations in C++ or by using the package **data.table** (Dowle et al. 2013). This allows for high-performance computations. The IHSN provided C++ code for many methods which were rewritten from scratch (except `suda2` and `rankSwap`) and integrated into **sdcMicro**.

1.4.2 S4 Class Structure of the sdcMicro Package

This section is mainly based on Templ et al. (2015) that shows the implementation of **sdcMicro** in detail. The following list gives an overview about the general aim of **sdcMicro** (see also Templ et al. 2015):

- **sdcMicro** includes the most comprehensive collection of micro-data protection methods;
- the well-defined S4-class implementation provides a user-friendly implementation and makes it easy to use its functionalities from **sdcMicroGUI**;
- utility functions extract information from well-defined S4 class objects;
- certain slots are automatically updated after application of a method;
- an undo function allows to return to a previous state of the anonymization process without the need to do additional calculations;
- for performance reasons, the methods are internally implemented either in C++ or by using the **data.table** package (Dowle et al. 2013);
- dynamic reports about results of the anonymization process can be generated.

Complex survey data needs special attention, but also for data collected without complex sampling designs some information is crucial. Of course, the software tool must be told which variables are important to protect and–if available in the data–the information about the variables holding the information on sampling weights and possible cluster structures (like household ID's) have to be reported. The idea is to generate an object in R that contains all these information, and in addition, stores further information that is estimated from the data. Moreover the information should update as soon a method is applied on such an object. To create such an object, the function `createSdcObj` is of central importance.

Of course, the add-on R package **sdcMicro** has to be loaded first in R to make the needed functions available. Parameters for `createSdcObj` are for example categorical and continuous key variables, the vector of sampling weights and optionally stratification and cluster IDs. The following code shows how to generate such an object using test data from a survey of the Philippines that are also included in the **sdcMicro** package.

```
require("sdcMicro")
data("testdata", package="sdcMicro")
sdc <- createSdcObj(testdata,
          keyVars=c('urbrur','water','sex','age'),
          numVars=c('expend','income','savings'),
          pramVars=c("walls"),
          w='sampling_weight',
          hhId='ori_hid')
```

This shows how to define the categorical and continuous key variables, the index of variables that are selected for PRAM (see Sect. 4.2.3), the vector of weights and the household IDs.

As mentioned before, by calling `createSdcObj` not only the previous mentioned information is stored, but many other slots are filled, e.g. slots holding information on risk and utility. Some of them are empty and only be filled as soon a certain method is applied, e.g. a slot called `pram` is only filled if the pram method is applied explicitly. The information contained in the slots update update when new methods applied on the object, e.g. the slot holding the information on risk and utility updates automatically with new risk and utility estimates as soon as an anonymization method is applied.

The following slots of an object of class `sdcMicroObj` are available:

```
slotNames(sdc)

   ##  [1] "origData"           "keyVars"            "pramVars"
   ##  [4] "numVars"            "ghostVars"          "weightVar"
   ##  [7] "hhId"               "strataVar"          "sensibleVar"
   ## [10] "manipKeyVars"       "manipPramVars"      "manipNumVars"
   ## [13] "manipGhostVars"     "manipStrataVar"     "originalRisk"
   ## [16] "risk"               "utility"            "pram"
   ## [19] "localSuppression"   "options"            "additionalResults"
   ## [22] "set"                "prev"               "deletedVars"
```

Slot origData contains the original data, keyVars the index of categorical key variables. pramVars contains an index indicating variables that are pramed (see Sect. 4.2.3 for details), slot numVars specifies indices of continuous key variables and the vector defining sampling weights is contained in slot weightVar. A possible index determining the cluster variable (slot hhId), the stratification variable (slot strataVar) and sensible variables (slot sensibleVar) can also be used. In addition, manipulated variables are saved in the slots beginning with manip. The risk measures (see Chap. 3) are stored in slots originalRisk (for the original unmodified data) and risk (for the manipulated data). Slot utility collects all information on data utility, further information on pramed variables, local suppressions are stored in slots pram and localSuppression while additional results (e.g., self-defined utility measures) are saved in slot additionalResults. Optionally, under the slot prev previous results are saved. Wrapper functions to extract relevant information from the sdcMicroObj are available (see Sect. 1.4.3). For more details on the structure of the sdcMicroObj have a look at the help file (help("createSdcObj")).

The show (print) method shows some basic properties of objects of class sdcMicroObj. The output of this print method is discussed in Sect. 3.3.

```
print(sdc)

   ## Infos on 2/3-Anonymity:
   ##
   ## Number of observations violating
   ##   - 2-anonymity: 330 (7.205%)
   ##   - 3-anonymity: 674 (14.716%)
   ##   - 5-anonymity: 1288 (28.122%)
   ## ## -----------------------------------------------------------------
```

Methods are applied on the sdcMicroObj and all related computations are done automatically. E.g, individual risks are re-estimated whenever a protection method is applied and the related slots of the sdcMicroObj are updated. A method to an sdcMicroObj can be applied by

method(sdcMicroObj),

where method is a placeholder for a specific method.

We note that **sdcMicro** also supports the straightforward application of methods to micro-data. For example, microaggregation of three continuous key variables (*expend*, *income* and *savings*) on the data set testdata can be achieved with

```
microaggregation(testdata[,c("expend","income","savings")])

   ##
   ##  Object created with method mdav and aggregation level 3
   ##  -------------------------
   ## x ... original values
   ##      expend              income              savings
   ##  Min.    :    3377   Min.    :2.897e+03   Min.    :   2975
   ##  1st Qu.:25610225   1st Qu.:2.510e+07   1st Qu.:2434823
   ##  Median :50462300   Median :5.075e+07   Median :4982921
   ##  Mean   :50499785   Mean   :5.012e+07   Mean   :4964039
   ##  3rd Qu.:75513585   3rd Qu.:7.500e+07   3rd Qu.:7487258
   ##  Max.   :99962044   Max.   :1.000e+08   Max.   :9997808
   ##
   ##  -------------------------
   ## mx ... microaggregated values
   ##      expend              income             savings
   ##  Min.    : 1151705   Min.    :  621527   Min.    :  85812
   ##  1st Qu.:25359582   1st Qu.:25066667   1st Qu.:2373826
   ##  Median :50285163   Median :50766667   Median :4990754
   ##  Mean   :50499785   Mean   :50115690   Mean   :4964039
   ##  3rd Qu.:75370230   3rd Qu.:75100000   3rd Qu.:7522210
   ##  Max.   :99174303   Max.   :99200000   Max.   :9928146
   ##
   ##  -------------------------
   ## Try names(your object from class micro) for more details
```

From the previous code line it is visible that an object of class *sdcMicroObj* is not mandatory to have. The methods can directly applied on data frames.

However, it is more convenient to first create an object of class *sdcMicroObj* using `createSdcObj()` and apply methods on objects of this class. To apply microaggregation not on a data frame but on such a class, we use

Table 1.2 Functions in R package **sdcMicro** for SDC disclosure risk and utility methods

Function	Aim	Updates slots ...
`freqCalc()`	Sample and population frequency estimation (used by `measureRisk()`)	–
`suda2()`	Frequency calculation on subsets	`@risk$suda2`
`ldiversity()`	*l*-diversity	`@risk$ldiversity`
`measureRisk()`	Individul, household and global risk estimation	`@risk*`
`modRisk()`	Global risk estimation using log-linear models	`@risk$model`
`dRisk()`	Disclosure risk for continuous scaled variables	`@risk$numeric`
`dRiskRMD()`	Advanced disclosure risk measures for continuous scaled variables	`@risk$numericRMD` (risk on cont. variables)
`dUtility()`	Data utiltiy measures	`@utility$*`

```
sdc <- microaggregation(sdc)
```

with `sdc` being an object of class `sdcMicroObj`. Note that the continuous key variables are already defined via `createSdcObj()`. To speak in more abstract terms: a method be applied to an object of class *sdcMicroObj* will extract the needed information, apply the method and put back the results as well as it will update several slots (e.g. update risk estimates).

In Table 1.2, the currently available methods and its function calls are listed. A brief aim of the function as well as an information which slots get updated in the `sdcMicroObj` object are outlined (see also Templ et al. 2015).

Table 1.3 Functions in R package **sdcMicro** for anonyimzation/perturbation methods

Function	Aim	Updates slots ...
`globalRecode()`	Anonyization of categorical key variables	`@risk$*`, `@manipKeyVars`, `@prev`
`groupVars()`	Anonymization of categorical key variables	`@risk$*`, `@manipKeyVars`, `@prev`
`localSupp()`	Univariate local suppression of high risky values	`@risk$*`, `@localSuppression`, `@manipKeyVars`, `@prev`
`kAnon()`	Local suppression to achieve k-anonymity	`@risk$*`, `@localSuppression`, `@manipKeyVars`, `@prev`
`pram()`	Swapping values using the post randomisation method	`@pram`, `@manipPramVars`, `@prev`
`microaggrGower()`	Microaggregation on categorical and continuous key variables	`@risk$*`, `@utility$*`, `@manipNumVars`, `@prev`
`topBottomCoding()`	Top and bottom coding	`@risk$*`, `@utility$*`, `@manipNumVars`, `@prev`
`addNoise()`	Perturbation of continuous variables	`@risk$numeric`, `@utility$*`, `@manipNumVars`, `@prev`
`rankSwapp()`	Perturbation of continuous variables	`@risk$numeric`, `@utility$*`, `@manipNumVars`, `@prev`
`mafast()`	Perturbation of continuous variables	`@risk$numeric`, `@utility$*`, `@manipNumVars`, `@prev`
`microaggregation()`	Perturbation of continuous variables, wrapper for various methods	`@risk$numeric`, `@utility$*`, `@manipNumVars`, `@prev`
`shuffle()`	Perturbation of continuous variables	`@risk$numeric`, `@utility$*`, `@manipNumVars`, `@prev`

For example, the application of `localSuppression()` on an `sdcMicroObj` suppress certain values in the data set and afterwards it updates the slots `risk` and `localSuppression`. In the slot `risk` all relevant list elements are replaced with new estimates and in the slot `localSuppression` the information of suppressed values gets updated. Another example is to apply `microaggregation`. In the above code, method microaggregation is applied using default values (to change these, see `help("microaggregation")`) to an object `sdc` of class `sdcMicroObj`. Since function `microaggregation()` is only suitable for continuous scaled variables (use `microaggrGower()` for other cases), the categorical variables remain untouched and microaggregation is applied on all continuous key variables. Finally, slot `@risk$numeric` (disclosure risk for continuous key variables) and `@utility$*` (data utility for continuous key variables) and `@manipNumVars` (the perturbed variables) are updated or filled. In addition, information on the previous state of the anonymization is saved in `@prev` (see Table 1.3).

1.4.3 Utility Functions

Available help functions of **sdcMicro** are listed in Table 1.4 (see also Templ et al. 2015). Functions to extract information from different slots are implemented. Helpful (especially when working with **sdcMicroGUI**), is the `undolast()` function that allows to undo the last step(s) of the anonymization.

Table 1.4 Utility functions

Function	Aim
`show()`	Print method for objects of class `sdcMicroObj`
`print.*()`	Print methods showing information on k-anonymity, l-diversity, local suppressions, recoding, disclosure risk, suda2 and pram
`get.sdcMicroObj()`	Directly return slots from `sdcMicroObj` objects. Alternatively, `slot` can also be used.
`generateStrata()`	Generate single variable defining a strata from multiple variables
`undolast()`	Revert the last modification of an object of class `sdcMicroObj`
`extractManipData()`	Extracts manipulated data from `sdcMicroObj` objects
`calcRisks()`	Recomputes the disclosure risk on objects of class `sdcMicroObj`
`varToFactor()` and `varToNumeric()`	Change a key variable of an object of class `sdcMicroObj` from `numeric` to `factor` or from `factor` to `numeric`

The slots of the `sdcMicroObj` can be accessed also using function `get.sdcMicroObj()` or `slot` as well as the current state of the data (with all anonymizations done so far) can be extracted, see the following code where the data utility and the categorical key variables are extracted from the *sdcMicroObj* `sdc`.

```
## data utility:
ut <- slot(sdc, "utility")
## index of categorical key variables:
cat <- slot(sdc, "keyVars")
## key variables, orginal data
head(testdata[, cat], 3)

   ##   urbrur water sex age
   ## 1      2     3   1  46
   ## 2      2     3   2  41
   ## 3      2     3   1   9
```

Using the function `extractManipData`, the manipulated data can also be extracted from the object `sdc`.

```
dat <- extractManipData(sdc)
str(dat)

   ## 'data.frame': 4580 obs. of 15 variables:
   ## $ urbrur : Factor w/ 2 levels "1","2": 2 2 2 2 2 2 2 2 2 2 ...
   ## $ roof : int 4 4 4 4 4 4 4 4 4 4 ...
   ## $ walls : int 3 3 3 3 2 2 2 2 2 2 ...
   ## $ water : Factor w/ 8 levels "1","2","3","4",..: 3 3 3 3 3 3 3 3 3 3
   ...
   ## $ electcon : int 1 1 1 1 1 1 1 1 1 1 ...
   ## $ relat : int 1 2 3 3 1 2 3 3 3 3 ...
   ## $ sex : Factor w/ 2 levels "1","2": 1 2 1 1 1 2 2 2 1 2 ...
   ## $ age : Factor w/ 88 levels "0","1","2","3",..: 47 42 10 7 53 48 14
   20 10 17 ...
   ## $ hhcivil : int 2 2 1 1 2 2 1 1 1 1 ...
   ## $ expend : int 90929693 27338058 26524717 18073948 6713247 49057636
   63386309 1106874 32659507 34347609 ...
   ## $ income : num 5.78e+07 2.53e+07 6.92e+07 7.96e+07 9.03e+07 ...
   ## $ savings : num 116258 279345 5495381 8695862 203620 ...
   ## $ ori_hid : int 1 1 1 1 2 2 2 2 2 2 ...
   ## $ sampling_weight : int 100 100 100 100 100 100 100 100 100 100 ...
   ## $ household_weights: num 25 25 25 25 16.7 ...
```

Print methods are available to show relevant information. The following code prints results about risk currently stored in object `sdc`. For explanations about risk-measures, see Chap. 3 for the theory on these measures.

```
print(sdc, "risk")

   ## Risk measures:
   ##
   ## Number of observations with higher risk than the main part of the
   data: 0
   ## Expected number of re-identifications: 24.78 (0.54  %)
   ##
   ## Information on hierarchical risk:
   ## Expected number of re-identifications: 117.20 (2.56 %)
   ##
   -----------------------------------------------------------------------
```

Other print methods report the number and percentage of observations violating 2 and 3-anonymity, the number of local suppressions, give information on recoding, the individual and cluster risk, the risk on continuous key variables and give information on pramed variables (output is suppressed):

```
print(sdc)
print(sdc, "ls")
print(sdc, type="recode")
print(sdc, type="risk")
print(sdc, type="numrisk")
print(sdc, type="pram")
```

These print methods will be practically applied in Chap. 8 as soon we anonymize data sets.

More information on **sdcMicro** and its facilities can be found in the manual of **sdcMicro**, and especially in Templ et al. (2015).

1.4.4 Reporting Facilities

Using the function `report()` it is possible to generate reports about the results of the anonymization process. The reports are internally generated using packages **brew** (Horner 2011) and **knitr** (Xie 2014a).

```
args(report)

   ## function (obj, outdir = getwd(), filename = "SDC-Report", title =
   "SDC-Report",
   ## internal = FALSE, verbose = FALSE)
   ## NULL
```

shows the possible function arguments. It is obvious that the first argument should be an object of class *sdcMicroObj*. Since the report is save on the hard disk in a certain format (function argument `format`), the directory and the file name can be specified. Two different types, internal (setting function argument `internal=TRUE`) and external (setting function argument `internal=FALSE`) reports, can be produced and exported as html, pdf or plain text files.

Internal Reports:

They include information about selected key variables, performed actions, disclosure risk and data utility and session information about the package versions used. This detailed report is suitable and useful for the organization that holds the data for internal use and documentation of the anonymization procedure.

External Reports:

They contain less information than an internal report. For example, all information about disclosure risks and information loss is suppressed. This report is suitable for external users of the anonymized micro-data set.

`?report` gives more information since it gives access to the help file.

All the information that is included in the report always depends on the anonymization process that has been applied and reflects the current values in a given object of class `sdcMicroObj`. For example, if PRAM was not applied, no specific summary for variables subjected to PRAM is available because this information is not available. However, if PRAM was applied, the entire disclosure risk summary is presented differently because usual risk measures are not valid if categorical key-variables have been modified using this procedure.

We stop the discussion on reports here, but show the exact structure of a SDC report in Sect. 8 after discussing the SDC methods in the next sections and anonymizing real-world data sets.

1.5 The Point-and-Click App sdcApp

The function `sdcApp` mainly written by Bernhard Meindl see `sdcApp` envokes an easy-to-handle, highly interactive browser-based anonymization tool for users who want to use the **sdcMicro** package for statistical disclosure control but are not familiar with the native R command line interface. The software performs automated recalculation and display of frequency counts, individual and global risk measures, information loss and data utility after each anonymization step. Changes to risk and utility measurements of the original data are also conveniently displayed in the graphical user interface (GUI). Furthermore, the code of every anonymization step carried out within the GUI is saved in a script, which can easily be exported, modified and re-used, making it possible to reproduce any results.

The developed app has the following capabilities:

Link to sdcMicro: The GUI uses the functionality of the **sdcMicro** package. It allows high performance and fast computations, since all basic operations are written in computationally efficient manner.

Import/Export: Datasets exported from other statistical software, such as SAS, SPSS, Stata, can easily be imported. It is also possible to use `.csv` files as well as data stored in R binary format. An interactive preview and selection of import parameters (such as delimiters or separators) are provided for the import of `.csv` files, which allows users to read the data correctly into the GUI. Export facilities are provided for the same formats from which data can be read into the GUI.

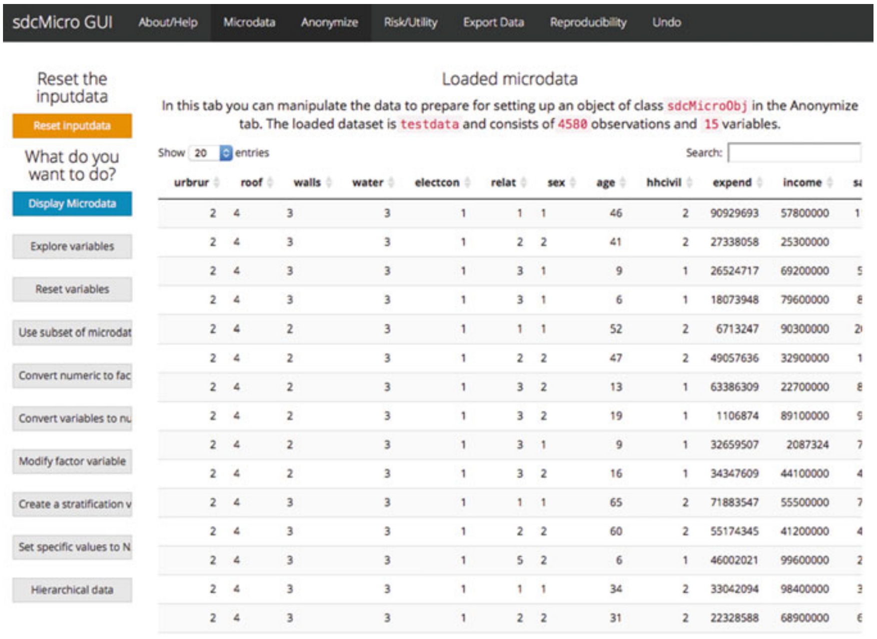
Usability: The app is an easy-to-use tool for anonymization of microdata, and all methods are easily accessible.

Recoding: Facilities to rename and regroup categories and change values of a variable are included.

Interactivity: Risk und utility measures are automatically estimated and displayed whenever users apply a disclosure limitation technique. In this way, users can immediately check the effects of any action. In addition, the risk and utility of the original unmodified data are displayed, which helps the user assess the effectiveness of the anonymization.

Undo: Because users can undo the last step completed in the app, they can try out several methods with different parameters and get instant feedback until the best result is achieved. Currently, users can reverse exactly one step in the history.

Reporting: Automatically generated, standardized reports in various output formats can be produced directly from the user interface. Reports can be exported to `html`, LaTeXor plain text files, see Sect. 1.4.4.

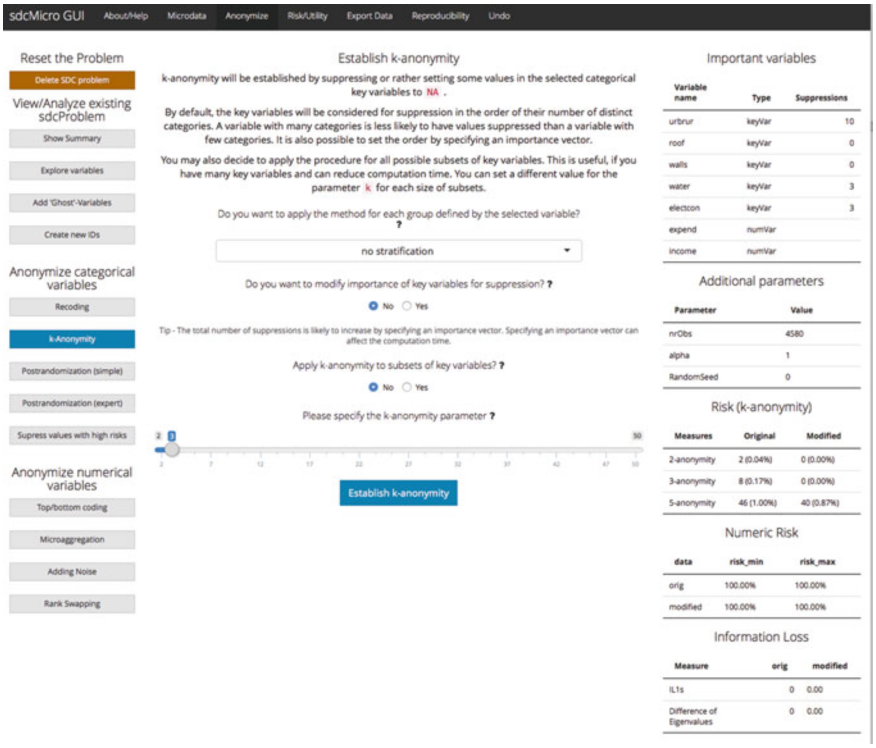


Loaded microdata

In this tab you can manipulate the data to prepare for setting up an object of class sdcMicroObj in the Anonymize tab. The loaded dataset is testdata and consists of 4580 observations and 15 variables.

urbrur	roof	walls	water	electcon	relat	sex	age	hhcivil	expend	income	sa
2	4	3	3	1	1	1	46	2	90929693	57800000	1
2	4	3	3	1	2	2	41	2	27338058	25300000	
2	4	3	3	1	3	1	9	1	26524717	69200000	5
2	4	3	3	1	3	1	6	1	18073948	79600000	8
2	4	2	3	1	1	1	52	2	6713247	90300000	2
2	4	2	3	1	2	2	47	2	49057636	32900000	1
2	4	2	3	1	3	2	13	1	63386309	22700000	8
2	4	2	3	1	3	2	19	1	1106874	89100000	9
2	4	2	3	1	3	1	9	1	32659507	2087324	7
2	4	2	3	1	3	2	16	1	34347609	44100000	4
2	4	3	3	1	1	1	65	2	71883547	55500000	7
2	4	3	3	1	2	2	60	2	55174345	41200000	4
2	4	3	3	1	5	2	6	1	46002021	99600000	2
2	4	3	3	1	1	1	34	2	33042094	98400000	3
2	4	3	3	1	2	2	31	2	22328588	68900000	6

Fig. 1.3 Data manipulation view after importing a data set. Users can modify variables, generate strata variables, etc



Establish k-anonymity

k-anonymity will be established by suppressing or rather setting some values in the selected categorical key variables to NA .

By default, the key variables will be considered for suppression in the order of their number of distinct categories. A variable with many categories is less likely to have values suppressed than a variable with few categories. It is also possible to set the order by specifying an importance vector.

You may also decide to apply the procedure for all possible subsets of key variables. This is useful, if you have many key variables and can reduce computation time. You can set a different value for the parameter k for each size of subsets.

Do you want to apply the method for each group defined by the selected variable? ?

no stratification

Do you want to modify importance of key variables for suppression? ?

[✓ No] Yes

Tip - The total number of suppressions is likely to increase by specifying an importance vector. Specifying an importance vector can affect the computation time.

Apply k-anonymity to subsets of key variables? ?

[✓ No] Yes

Please specify the k-anonymity parameter ?

Important variables

Variable name	Type	Suppressions
urbrur	keyVar	10
roof	keyVar	0
walls	keyVar	0
water	keyVar	3
electcon	keyVar	3
expend	numVar	
income	numVar	

Additional parameters

Parameter	Value
nrObs	4580
alpha	1
RandomSeed	0

Risk (k-anonymity)

Measures	Original	Modified
2-anonymity	2 (0.04%)	0 (0.00%)
3-anonymity	8 (0.17%)	0 (0.00%)
5-anonymity	46 (1.00%)	40 (0.87%)

Numeric Risk

data	risk_min	risk_max
orig	100.00%	100.00%
modified	100.00%	100.00%

Information Loss

Measure	orig	modified
IL1s	0	0.00
Difference of Eigenvalues	0	0.00

Fig. 1.4 Applying a SDC method using the app. Here, local suppression is applied to achieve 3-anonymity. Important informations are displayed on the *right*

Reproducibility: With the app, users can save, load or edit scripts for later re-use. Within the GUI, each step of the anonymization procedure is recorded and stored in a script. The script includes valid R expressions that can be copied into R . Thus, any anonymization procedure can be reproduced either by loading a script into the GUI or pasting the script directly into an R console.

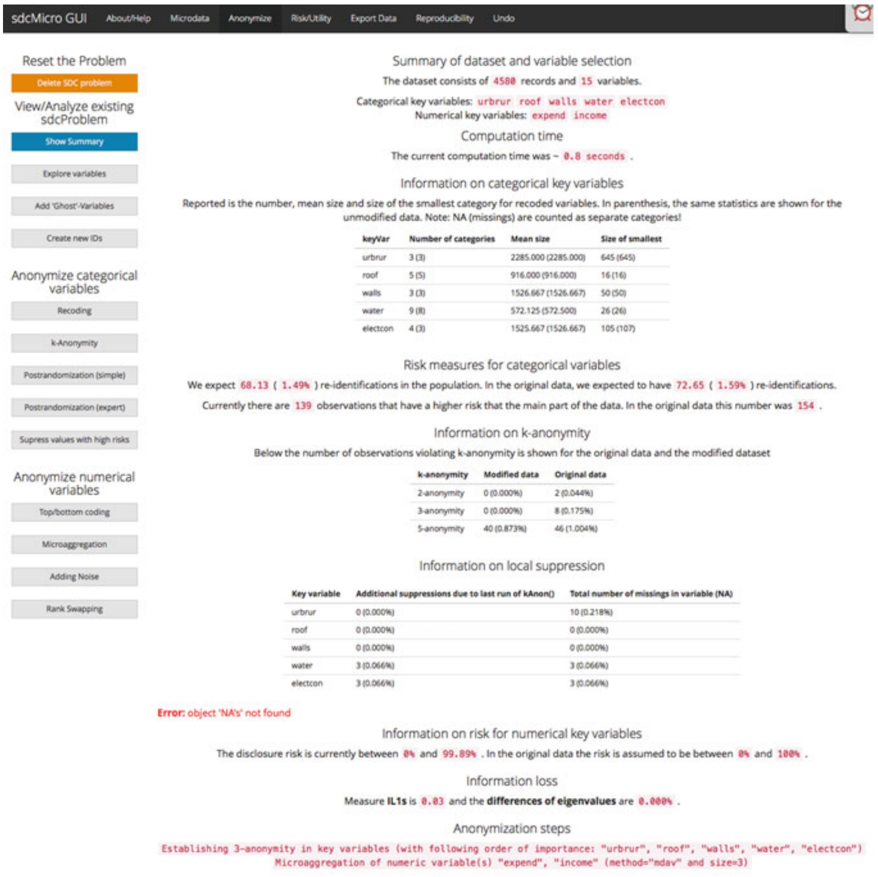
The user can then interact with the interface by using standard control inputs such as buttons, drop-down menus, sliders, radio buttons or file-upload fields.

The app can be opened with

```r
sdcApp()
```

In this browser window, the main sections are accessible with tabs listed in the top-navigation bar. The first step in the anonymization process is always to either upload microdata (from various formats) in tab *Microdata* or by importing a previously exported problem instance.

Once micro data have been uploaded (which are only stored locally of course), users can apply a range on methods to the currently active dataset. Figure 1.3 shows



Summary of dataset and variable selection

The dataset consists of 4580 records and 15 variables.

Categorical key variables: urbrur roof walls water electcon
Numerical key variables: expend income

Computation time

The current computation time was ~ 0.8 seconds .

Information on categorical key variables

Reported is the number, mean size and size of the smallest category for recoded variables. In parenthesis, the same statistics are shown for the unmodified data. Note: NA (missings) are counted as separate categories!

keyVar	Number of categories	Mean size	Size of smallest
urbrur	3 (3)	2285.000 (2285.000)	645 (645)
roof	5 (5)	916.000 (916.000)	16 (16)
walls	3 (3)	1526.667 (1526.667)	50 (50)
water	9 (8)	572.125 (572.500)	26 (26)
electcon	4 (3)	1525.667 (1526.667)	105 (107)

Risk measures for categorical variables

We expect 68.13 (1.49%) re-identifications in the population. In the original data, we expected to have 72.65 (1.59%) re-identifications.

Currently there are 139 observations that have a higher risk that the main part of the data. In the original data this number was 154 .

Information on k-anonymity

Below the number of observations violating k-anonymity is shown for the original data and the modified dataset

k-anonymity	Modified data	Original data
2-anonymity	0 (0.000%)	2 (0.044%)
3-anonymity	0 (0.000%)	8 (0.175%)
5-anonymity	40 (0.873%)	46 (1.004%)

Information on local suppression

Key variable	Additional suppressions due to last run of kAnon()	Total number of missings in variable (NA)
urbrur	0 (0.000%)	10 (0.218%)
roof	0 (0.000%)	0 (0.000%)
walls	0 (0.000%)	0 (0.000%)
water	3 (0.066%)	3 (0.066%)
electcon	3 (0.066%)	3 (0.066%)

Error: object 'NA's' not found

Information on risk for numerical key variables

The disclosure risk is currently between 0% and 99.89% . In the original data the risk is assumed to be between 0% and 100% .

Information loss

Measure **IL1s** is 0.03 and the **differences of eigenvalues** are 0.000% .

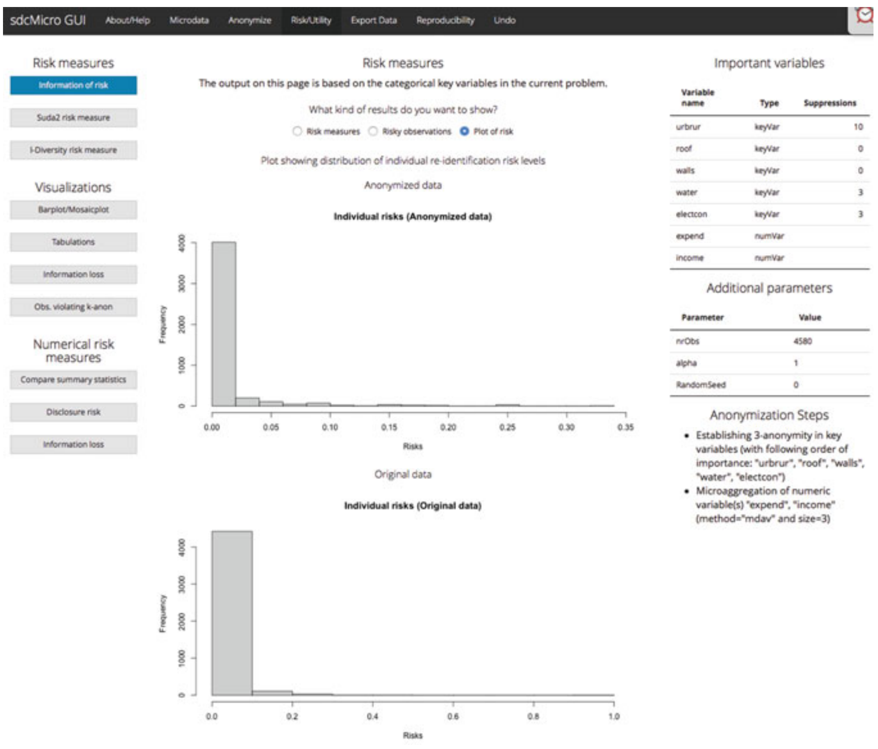
Anonymization steps

```
Establishing 3-anonymity in key variables (with following order of importance: "urbrur", "roof", "walls", "water", "electcon")
Microaggregation of numeric variable(s) "expend", "income" (method="mdav" and size=3)
```

Fig. 1.5 Summary after anonymization

the data manipulation view. After microdata are ready to use, the SDC problem can be specified in the *Microdata* tab. Exemplarely, we took *urbrur*, *roof*, *walls*, *water* and *electcon* as categorical variables in the SDC problem, and variables *expend* and *income* as continuous scaled variables. Note that also sampling weights and stratification variables can be chosen as well as various other kind of variable specification can be done.

After specification of the SDC problem, a summary of the SDC problem is presented (Fig. 1.4). A bunch of SDC methods can then be applied to the SDC problem. We applied here local suppression and microaggregation and show its summary in Fig. 1.5.



Risk measures

The output on this page is based on the categorical key variables in the current problem.

What kind of results do you want to show?

Risk measures Risky observations [✓ Plot of risk]

Plot showing distribution of individual re-identification risk levels

Anonymized data

Original data

Important variables

Variable name	Type	Suppressions
urbrur	keyVar	10
roof	keyVar	0
walls	keyVar	0
water	keyVar	3
electcon	keyVar	3
expend	numVar	
income	numVar	

Additional parameters

Parameter	Value
nrObs	4580
alpha	1
RandomSeed	0

Anonymization Steps

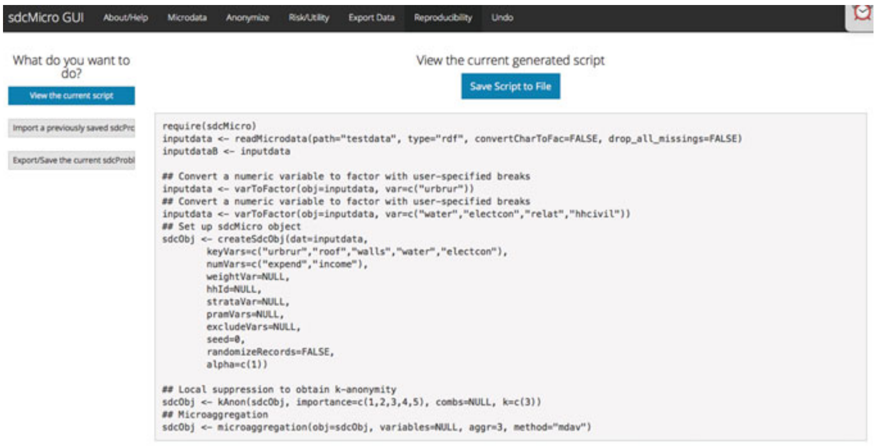
- Establishing 3-anonymity in key variables (with following order of importance: "urbrur", "roof", "walls", "water", "electcon")
- Microaggregation of numeric variable(s) "expend", "income" (method="mdav" and size=3)

Fig. 1.6 Displaying the disclosure risk. Here, the distribution of the individual risk is compared with the individual risk of the original (unmodified) data

Various disclosure risk measures can be displayed, e.g., see Fig. 1.6 where the individual risk (Sect. 3.5) is shown using histograms.

Every action of the user in the app (click on buttons, selection from drop-down menus, etc.) is memorized and the underlying code is shown in the tab *Reproducibility*, see also Fig. 1.7. This script can be saved and used for later use, e.g. by importing it to the app.

The app `sdcApp` is not further used in the book. However, (almost) any method and code shown in the book can also be applied in the app.



```
require(sdcMicro)
inputdata <- readMicrodata(path="testdata", type="rdf", convertCharToFac=FALSE, drop_all_missings=FALSE)
inputdataB <- inputdata

## Convert a numeric variable to factor with user-specified breaks
inputdata <- varToFactor(obj=inputdata, var=c("urbrur"))
## Convert a numeric variable to factor with user-specified breaks
inputdata <- varToFactor(obj=inputdata, var=c("water","electcon","relat","hhcivil"))
## Set up sdcMicro object
sdcObj <- createSdcObj(dat=inputdata,
        keyVars=c("urbrur","roof","walls","water","electcon"),
        numVars=c("expend","income"),
        weightVar=NULL,
        hhId=NULL,
        strataVar=NULL,
        pramVars=NULL,
        excludeVars=NULL,
        seed=0,
        randomizeRecords=FALSE,
        alpha=c(1))

## Local suppression to obtain k-anonymity
sdcObj <- kAnon(sdcObj, importance=c(1,2,3,4,5), combs=NULL, k=c(3))
## Microaggregation
sdcObj <- microaggregation(obj=sdcObj, variables=NULL, aggr=3, method="mdav")
```

Fig. 1.7 The tab *Reproducibility* shows the code that has internally be applied to **sdcMicro** by point-and-click in the app. This code is stored for later use and reproducibility issues

1.6 The simPop package

The R package **simPop** (Templ et al. 2017) can be used to generate synthetic data. The package can be downloaded from the Comprehensive R Archive Network (CRAN). Similar to **sdcMicro**, the package supports an object-oriented S4 class structure (Chambers 2008). Parallelization is applied by default, taking into account the number of available CPUs and the structure of the data set to be generated.

The main functions are listed in Table 1.5 (see also Templ et al. 2017). All functionalities are described in the help manual, and executable examples are provided. Although most of **simPop** functions are applicable to data frames, their implementation will typically make use of objects of specific classes, in particular Templ et al. (2017):

dataObj: objects of this class contain information on the population census and survey data to be used as input for the generation of the synthetic population. They are automatically created with `specifyInput()`. They hold information on the variables containing the household and person IDs, household size, sampling weights, stratification information, and type of data (i.e., sample or a population);

simPopObj: a *simPopObj* object is created with function `createSdcObj`. Objects of this class hold information on the sample (in slot `sample`), the population (slot `pop`), optionally information on some margins in the form of a table (slot `table`), and many other information such as disclosure risk, data utility, key variables, etc. Objects in slot `sample` and `pop` must be objects of class *dataObj*. Most methods that are required to create synthetic populations can be applied directly to objects of this class.

Table 1.5 Most important functions of R-package **simPop** listed in order of a typical workflow for simulation of synthetic data

Function	Aim
`manageSimPopObj`	Accessor to objects from class *simPopObj*
`tableWt`	Weighted cross-tabulation
`calibSample`	Generalized raking procedures (iterative proportional fitting); calibrates the sample to known margins
`addWeights`	Add weights (e.g., calibrated weights) to an object of class *simPopObj*
`ipu`	Iterative proportional updating
`sampHH`	Households are drawn/sampled and new IDs are generated
`specifyInput`	Create an object of class *dataObj*
`simStructure`	Simulate the basic household structure
`simCategorical`	Simulates categorical variables
`simContinuous`	Simulates continuous variables
`simRelation`	Simulation of categorical variables
`simComponents`	Simulates components of (semi-)continuous variables
`addKnownMargins`	Add known margins (table) to objects of class *simPopObj*
`calibPop`	Calibrate a synthetic population to known margins using simulated annealing
`sampleObj`	Query or replace slot'sample' of a *simPopObj*
`popObj`	Query or replace slot'pop' of a *simPopObj*
`tableObj`	Query slot'table' of a *simPopObj*
`spCdf`, `spTable`	Weighted cumulative distribution function and cross-tabulation of expected and realized population sizes
`spCdfplot`, `spMosaic`	Plots of expected and realized population sizes

Special functionality (and not listed in Table 1.5) is available in **simPop** to apply corrections for heaping in age or income variables. In particular, the Whipple-Index (Shryock et al. 1976) and the Sprague-Index (see, e.g., Calot and Sardon 2004) are implemented with functions `whipple()` and `sprague()`, respectively. With the function `correctHeap` heaping can be corrected, whereby using truncated (log-)normal distributions. A detailed description of this functionality is out of the scope of this work, but the help pages available at `?whipple`, `?sprague` and `?correctHeap` provide details and running examples.

Applications of **simPop** are shown in the case study of Sect. 8.7. Further details on the R package is given in Templ et al. (2017).

Exercises:

Question 1.1 **Study help files**
(a) Load the **sdcMicro** package in R. Have a look at the help index.
(b) Open the help of function `createSdcObj()` and execute the examples in the example section.

Question 1.2 **Getting familiar with some data sets**
(a) Load the data set `testdata` which is available in **sdcMicro**. Look at the structure of the data using functions `str()`, `head()`, `colnames()` and `?testdata`.
(b) Load the data set `eusilc` from package **laeken**. Note that the package might need to be installed first using `install.packages("laeken")` and loaded via `library(laeken)` before you can access the data.

Question 1.3 **Getting familiar with the S4 class structure**
(a) Run the example of `?globalRecode`.
(b) List the names of all slots from the object `sdc`.
(c) Access the slot called `risk` and search for the individual risk.

References

Calot, G., & Sardon, J.P. Methodology for the calculation of Eurostat's demographic indicators. In *Population and social conditions 3/2003/F/n, Eurostat, European Demographic Observatory*, Luxembourg.

Chambers, J. M. (2008). *Software for data analysis: programming with*R. New York: Springer.

Chang, W., Cheng, J., Allaire, J. J., Xie, Y., & McPherson, J. (2016). Shiny: Web Application Framework for R.

Dai, C., Ghinita, G., Bertino, E., Byun, J.-W., & Li, N. (2009). TIAMAT: A tool for interactive analysis of microdata anonymization techniques. *PVLDB*, *2*(2), 1618–1621.

Dowle, M., Short, T., Lianoglou, A., & Saporta, R. (2013). *data.table: Extension of data.frame for Fast Indexing, Fast Ordered Joins, Fast Assignment, Fast Grouping and List Columns*, R package version 1.8.10. Retrieved from http://CRAN.R-project.org/package=data.table. With contributions from Srinivasan.

Horner, J. (2011). *brew: Templating Framework for Report Generation*, R package version 1.0-6. Retrieved from http://CRAN.R-project.org/package=brew.

Hundepool, A., Van de Wetering, A., Ramaswamy, R., Franconi, L., Polettini, S., Capobianchi, A. et al. (2008). *μ-Argus. User Manual*, version 4.2.

Kohlmayer, F., Prasser, F., Eckert, C., Kemper, A., Kuhn, K.A. (2012). Flash: Efficient, stable and optimal k-anonymity. In *2012 International Conference on Privacy, Security, Risk and Trust (PASSAT), 2012 International Confernece on Social Computing (SocialCom)* (pp. 708–717).

Kowarik, A., Templ, M., Meindl, B., & Fonteneau, F. (2013). sdcMicroGUI: Graphical user interface for package sdcMicro, R package version 1.1.1. Retrieved from http://CRAN.R-project.org/package=sdcMicroGUI

LeFevre, K., DeWitt, D. J., & Ramakrishnan, R. (2006). Mondrian multidimensional k-anonymity. In *Proceedings of the 22nd International Conference on Data Engineering*, ICDE 2006 (p. 25). Washington, DC, USA: IEEE Computer Society. http://dx.doi.org/10.1109/ICDE.2006.101. ISBN 0-7695-2570-9. 10.1109/ICDE.2006.101.

Nergiz, M.E., Atzori, M., & Clifton, C. (2007). Hiding the presence of individuals from shared databases. In *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data, SIGMOD 2007* (pp. 665–676). New York, NY, USA: ACM. ISBN 978-1-59593-686-8. doi:10.1145/1247480.1247554.

R Development Core Team. (2014). R: A language and environment for statistical computing. R Foundation for Statistical Computing, Vienna, Austria. Retrieved from http://www.R-project.org/. ISBN 3-900051-07-0.

Shryock, H. S., Stockwell, E. G., & Siegel, J. S. (1976). *The methods and materials of demography*. New York: Academic Press.

Templ, M., Meindl, B., Kowarik, A., & Dupriez, O. (2015). Simulation of synthetic complex data: The R-package simPop. *Journal of Statistical Software*, 1–38 (2017). (Accepted for publication in December 2015).

Templ, M. (2008). Statistical disclosure control for microdata using the R-package sdcMicro. *Transactions on Data Privacy*, *1*(2), 67–68. Retrieved from http://www.tdp.cat/issues/abs.a004a08.php.

Templ, M., Alfons, A., & Filzmoser, P. (2012). Exploring incomplete data using visualization techniques. *Advances in Data Analysis and Classification*, *6*(1), 29–47.

Templ, M., Meindl, B., & Kowarik, A. (2015). Statistical disclosure control for micro-data using the R package sdcMicro. *Journal of Statistical Software*, *67*(1), 1–37.

Wickham, H., & Chang, W. (2015). devtools: tools to make developing R packages easier, R package version 1.7.0. Retrieved from http://CRAN.R-project.org/package=devtools.

Xie, Y. (2014a). *knitr: A General-purpose Package for Dynamic Report Generation in R*, R package version 1.6. Retrieved from http://yihui.name/knitr/.