Matthias Templ

Statistical Disclosure Control for Microdata

Methods and Applications in R

Springer

Statistical Disclosure Control for Microdata

Matthias Templ

Statistical Disclosure Control for Microdata

Methods and Applications in R

Springer

Matthias Templ
Institute of Data Analysis and Process Design (IDP)
School of Engineering (SoE)
Zurich University of Applied Sciences (ZHAW)
Winterthur, Switzerland

and

data-analysis OG
Vienna, Austria
www.data-analysis.at

ISBN 978-3-319-50270-0 ISBN 978-3-319-50272-4 (eBook)
DOI 10.1007/978-3-319-50272-4

Library of Congress Control Number: 2017937274

Mathematics Subject Classification (2010): 62D99, 62D05, 62-07, 62J05, 62P25

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer International Publishing AG
The registered company address is: Gewerbestrasse 11, 6330 Cham, Switzerland

To all who raised my interest in writing this book by not believing in our work and the success of free and open-source philosophy.

Preface

Introduction

An increasing amount of data on persons and establishments are collected by statistical organizations. Also, the demand for microdata for researchers is increasing since economic or empirical analysis, and to make statements about our society on empirical basis is often only possible when investigating in data with detailed information. Moreover, a considerable increase in the production of socioeconomic data and their accessibility by researchers have been observed in recent years. Statistical agencies are making more of their survey and census microdata available, government agencies are publishing more of their administrative data, and the private sector has become a major provider of big data.

This, however, comes with a variety of legal, ethical, and technical challenges. Privacy protection principles and regulations impose restrictions on access and use of individual data. Statistical producers are faced with the challenge of ensuring respondents' confidentiality when making microdata files accessible. Not only data producers are obligated to protect confidentiality, but also confidentiality is crucial for maintaining the trust of respondents and ensuring the honesty and validity of their responses. Statistical disclosure control (SDC) is thus an emerging field of research. Proper and secure microdata dissemination requires statistical agencies to establish policies and procedures that formally define the conditions for accessing microdata (Dupriez and Boyko 2010) and to apply statistical disclosure control (SDC) methods to data before release.

Also, the demand for complex microdata for training purposes for students seems to increase. Research projects under the 5th, 6th, or 7th framework program for research of the European Union generated confidential microdata for public or scientific use. Public-use files are also generated for researchers to be able to run complex simulation studies to compare methods. The *BLUE-ETS*, *AMELI*, *DACSEIS*, and *EUREDIT* research projects of this framework program are examples using anonymized data for simulation tasks. Outside Europe, there is a long tradition to anonymize data sets. For example, the US CENSUS Bureau provides

public-use data since the 1970s, and current releases of public-use data set include the current population survey, the survey of income and program participation, the American housing survey, the survey of program dynamics, the American community survey, and the consumer expenditure survey [see, e.g., Task Force on the Conference of European Statisticians 2007]. In addition, almost every statistical agency in the world and institutions holding confidential data, e.g., on health statistics, are at least interested in providing public- or scientific-use files of high quality and low disclosure risk. In any case, due to national laws on privacy, microdata cannot be distributed to the public or to researchers whenever re-identification of persons or establishments is possible.

This book gives a detailed view on well-known SDC methods for data, complex sample surveys, censuses, and administrative sources. It discusses the traditional approach of data anonymization by perturbation of the data, the disclosure risk, and data utility of anonymized data sets. It also includes methods to simulate synthetic data.

This is not a book about the software environment `R` and related packages for statistical disclosure control. The aim of the book was to explain the theory of statistical disclosure control methods. However, in order to better understand the theory, a lot of practical examples are given. Almost every example can be reproduced by the readers using `R` and the `R` packages **sdcMicro** and **simPop**. The code for each example is included in the book and provided as supplementary files at

http://www.statistik.tuwien.ac.at/public/templ/sdcbook

The free and open-source software provided with this book allows the reader to also apply the presented methods on their own data sets provided by the mentioned software.

Overview of the Book

This book is intended for statistical producers at National Statistical Offices (NSOs), data holders, and international and national agencies, as well as data users who are interested in the subject. It does not require no prior knowledge of statistical disclosure control (SDC). This book is focused on SDC methods for microdata. It does not cover SDC methods for protecting tabular outputs [see Castro 2010, for more details].

Chapter 1 includes a very brief introduction to R. It is intended to make further `R` code in the book understandable, but it does not replace a conventional introduction to R. If readers want to learn `R` in a broader context, we refer to the official manuals on `R` on the CRAN Webpage or to further contributed documentation on CRAN or books about R. After this short introduction, SDC tools and the differences between each of them are discussed. This section closes with a general discussion about the `R` package **sdcMicro**. The **sdcMicro** and the **simPop** packages are applied in various other chapters, especially in the case studies of Chap. 8.

The methods' parts start with an introduction to the basic concepts regarding statistical disclosure in Chap. 2. These concepts include the definition of terms used in the area of SDC and briefly present different kinds of disclosure scenarios. Chapter 2 finishes with risk-utility maps that show the trade-off between disclosure risk and data utility.

Chapter 3 includes methods for measuring disclosure risk. The aim is not only to discuss one concept of measuring/estimating disclosure risk, but also to describe several concepts. The chapter starts with basic methods such as *k*-anonymity and *l*-diversity. To measure risk on subsets of key variables, the SUDA method is explained. A large part of this chapter is dedicated to measuring the individual risk and the global risk with log-linear models.

In Chap. 4, the most common SDC methods are presented. The chapter is basically divided into two parts: one for categorical data and the other for continuous scaled data. Not only popular, but also new methods, such as shuffling, are discussed.

An introduction to common approaches for assessing information loss and data utility is given in Chap. 5. This chapter includes more than just typical comparisons, such as comparing means and covariances, since these methods—even regularly applied—are often too general and not suitable for specific data sets. Therefore, further concepts based on estimating certain indicators are presented as well.

Even if this book is mainly focused on traditional methods for SDC, Chap. 6 is dedicated to the simulation of synthetic data sets. A lot of different concepts exist in literature, but we focus on only one particular approach. We point out some advantages of the approach of model-based simulation of synthetic population data and describe the framework of this concept.

Chapter 7 provides practical guidelines on how to implement SDC on data sets in practice. Basically, the work flow for anonymizing data is shown. All the learned lessons are included in this workflow.

This workflow is also applied in Chap. 8 on several very popular data sets such as the Structure of Earnings Statistics (SES), the European Statistics on Income and Living Conditions (EU-SILC), the Family Income and Expenditure Survey (FIES), the International Income Distribution Database (I2D2), the Purchase Power Parity Data set (P4), and the Survey-based Harmonized Indicators Data (SHIP) that are also harmonized data files from household surveys.

Note that almost all methods introduced in this book can be implemented using **sdcMicroGUI**, an `R` -based, user-friendly, point-and-click graphical user interface (Kowarik et al. 2013). However, all methods are shown in applications using the `R`-package **sdcMicro** (Templ et al. 2015) that differs in that sense that working with the package is command line only and more methods are included than in **sdcMicroGUI** . Note that a successor, a new version of **sdcMicroGUI** that works in a browser and is included in **sdcMicro**, version > 4.7. The app can be opened via the function call `sdcApp()` in R. For simulating synthetic data, the package

simPop is used. Readers are encouraged to explore the implementation of methods using this book along with the detailed user manuals of **sdcMicroGUI** (Kowarik et al. 2013), **sdcMicro** (Templ et al. 2015), and **simPop** (Templ et al. 2017).

Winterthur, Switzerland Matthias Templ
March 2017

References

Castro, J. (2010). Statistical disclosure control in tabular data. In J. Nin & J. Herranz (Eds.), *Privacy and anonymity in information management systems, Advanced information and knowledge processing* (pp. 113–131). London: Springer.

Dupriez, O., & Boyko, E. (2010). Dissemination of microdata files. *Formulating policies and procedures*. IHSN Working Paper No 005, International Household Survey Network, Paris.

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

Task Force on the Conference of European Statisticians. (2007). Managing statistical confidentiality & microdata access. *Principles and guidelines of good practice*. Technical report, United Nations Economic Commission for Europe, New York and Geneva.

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.

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

Acknowledgements

The author benefited from collaborations and discussions with **Bernhard Meindl** (Statistics Austria, data-analysis OG) and **Alexander Kowarik** (Statistics Austria, data-analysis OG). Many publications on this topic were jointly made and important input on parts of the book was given by them, especially on Chap. 8 but this is also true for some other chapters. The book also profited from previous work with Shuang Chen (OECD/Paris 21), Olivier Dupriez (World Bank), François Fonteneau (OECD/PARIS21), Thijs Benschop and Marius Totter (Vienna University of Technology). I would also like to thank the reviewers, especially Reviewer 1, for the numerous comments and suggestions. For English proofreading of parts of this book, my gratitude goes to Karin Zorn. The underlying software to this book was developed with financial support from the International Household Network, the World Bank (several projects, mentioned explicitly in the references), Google, Statistics Austria, data-analysis OG, and Vienna University of Technology.

The programming of the software was carried out in cooperation with Alexander Kowarik and Bernhard Meindl. Without their great contribution and professional work, the software would not be as powerful as it is at this stage. The code on model-based global risk measures was revised by Marius Totter while he was working on his master thesis, which I supervised. I would also like to thank my contact person and editor at Springer, Veronika Rosteck. She was always supportive and constructive, but most importantly, she was/is always in a good mood which motivated me to keep writing this book. Finally, my gratitude goes also to my wife Barbara for her supportive and understanding smile related to more intensive writing phases of the book.

Contents

Acronyms

CRAN	Comprehensive R Archive Network
CSS	Complex Survey Samples
EU-SILC	European Statistics on Income and Living Conditions
FIES	Family Income and Expenditure Survey
GUI	Point-and-click graphical user interface
I2D2	International Income Distribution Database
IHSN	International Household Survey Network
MCD	Minimum Covariance Determinant
MDAV	Maximum Distance to Average Vector
NSI	National Statistical Institute
P4	Purchase Power Parity Data
PRAM	Post Randomization Method
RMD	Robust Mahalanobis Distance
ROMM	Random Orthogonal Matrix Masking
SDC	Statistical Disclosure Control
SES	Structural Earning Statistics
SHIP	Survey-Based Harmonited Indicator Program
SUDA	Special Uniqueness Detection Algorithm

The book does not contain a glossary on Statistical Disclosure Control, because the important keywords are explained in Chap. 2. Moreover, glossaries on this topic can be found on the Internet, e.g., at http://neon.vb.cbs.nl/casc/Glossary.htm.

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.

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
```

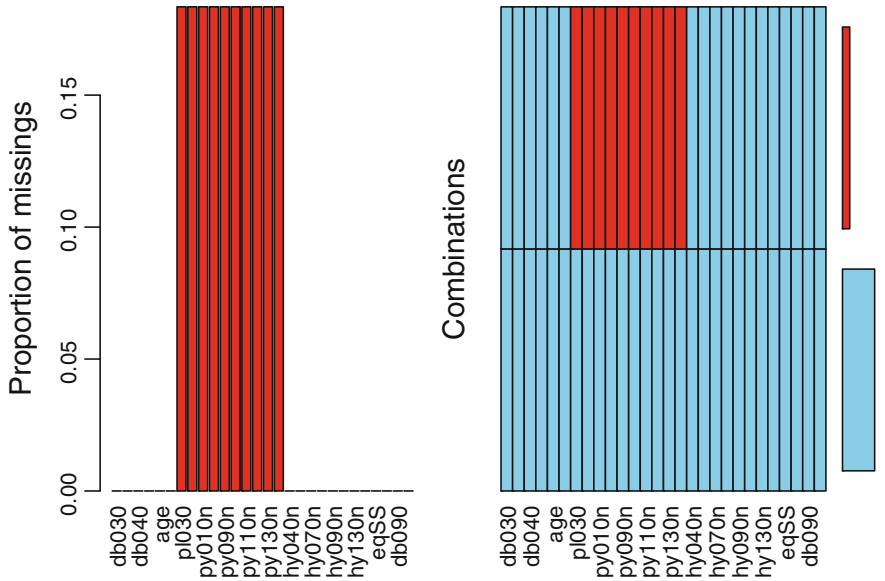

Fig. 1.1 Missing patterns in the EU-SILC data set. *Left* proportion of missing values in each variable. *Right* Missing value patterns

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.

```
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