Chapter 4
Methods for Data Perturbation

Abstract Methods for perturbation of data differ for categorical and continuous variables. The risk for categorical key variables is dependent on the frequency counts of keys, whereas keys with only few observations are problematic. Categories of categorical key variables with low frequency counts are therefore often recoded and combined with other categories. However, a still too high disclosure risk may be present for some individuals. Local suppression is one method to further reduce the disclosure risk. In order to find a well-balanced, suitable solution, global recoding is usually applied in an explorative manner to observe with which (reasonable) recodings one achieves the best effect in terms of reducing the disclosure risk and providing high data utility. Especially with a large amount of key variables, swapping methods, such as PRAM, are good alternatives. Methods for continuous scaled variables are combining values (microaggregation) or adding noise to the values. Advanced methods such as shuffling allow to preserve certain statistics.

4.1 Kind of Methods

The SDC techniques may be categorized into three kind of methods:

- non-perturbative techniques, such as recoding and local suppression, which suppress or reduce the detail without altering the original data;
- perturbative techniques, such as adding noise, Post-Randomization Method (PRAM), micro-aggregation and shuffling, which distort the original micro-data set before release;
- techniques that generate a synthetic microdata file that preserves certain statistics or relationships of the original files.

© Springer International Publishing AG 2017
M. Templ, *Statistical Disclosure Control for Microdata*,
DOI 10.1007/978-3-319-50272-4_4

This chapter focuses on the non-perturbative and perturbative techniques. As with disclosure risk measures, different SDC methods are applicable to categorical variables and continuous variables.

Note that generating synthetic data is a more complicated approach and the methods differ completely. This is the reason why these methods are not discussed in this chapter, and we refer to Chap. 6 that is dedicated to synthetic data simualation.

4.2 Methods for Categorical Key Variables

4.2.1 Recoding

Global recoding is a non-perturbative method that can be applied to both categorical and continuous key variables. For a categorical variable, the idea of recoding is to combine several categories into fewer categories with higher frequency counts and less detailed information. In other words, a global recoding achieves anonymity by mapping the values of the categorical key variables to generalized or altered categories. For example, one could combine multiple levels of schooling (e.g., secondary, tertiary, postgraduate) into one (e.g., secondary and above). For a continuous variable, recoding means to discretize the variable; for example, recoding a continuous income variable into a categorical variable of income levels. In both cases, the goal is to reduce the total number of possible values of a variable. Typically, recoding is applied to categorical variables to collapse categories with few observations into a single category with larger frequency counts. For example, if there are only two respondents with tertiary level of education, the tertiary can be combined with the secondary level into a single category of "secondary and above".

Remember, we have the following variables in our `sdcMicroObj`

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

The categorical key-variable `age` selected before has a lot of categories

```
table(testdata$age)
```

```
##
##   0   1   2   3   4   5   6   7   8   9  10  11  12  13  14
##  98  90 134 112 128 133 136 125 144 126 151 127 108 143 103
##  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29
## 111 106 101  96  64  88  61  64  55  55  69  56  68  69  50
##  30  31  32  33  34  35  36  37  38  39  40  41  42  43  44
##  90  51  72  49  59  86  61  68  51  42  67  43  44  49  40
##  45  46  47  48  49  50  51  52  53  54  55  56  57  58  59
##  65  37  31  43  28  44  28  28   8  19  31  28  17  27  20
##  60  61  62  63  64  65  66  67  68  69  70  71  72  73  74
##  36  24  28  14  12  40   8  16  14   8  20   6  14   4   6
##  75  76  77  78  79  80  82  83  84  85  88  90  95
##   6   4   5   3   2   5   1   1   1   1   1   2   1
```

and some categories are sparse. Recoding `age` into `age classes` will cause that the number of observations in the keys increase and less observations are violating the 2- and 3-anonymity assumption.

```
labs <- c("1-9","10-19","20-29","30-39",
          "40-49","50-59","60-69","70-79","80-130")
sdc <- globalRecode(sdc, column="age",
                    breaks=c(0,9,19,29,39,49,59,69,79,130),
                    labels=labs)
print(sdc)
```

```
## Infos on 2/3-Anonymity:
##
## Number of observations violating
## - 2-anonymity: 111 (2.424%) | in original data: 653 (14.258%)
## - 3-anonymity: 184 (4.017%) | in original data: 1087 (23.734%)
## - 5-anonymity: 345 (7.533%) | in original data: 1781 (38.886%)
##
##
----------------------------------------------------------------------
```

Let's have a look at the frequency counts of the categorized age variable. Note that we use function `extractManipData()` to extract the current data from the *sdcMicroObj* object.

```
table(extractManipData(sdc)$age)
```

```
##
##     1-9   10-19   20-29   30-39   40-49   50-59   60-69   70-79
##    1128    1110     635     629     447     250     200      70
## 80-130
##      13
```

Still some categories have only few cases, so we further modify the age variable by joining the last two groups. Here a general function could be applied, called `groupAndRename()`.

```
sdc <- groupAndRename(sdc, var = "age",
                     before = c("60-69", "70-130"),
                     after = "60-130")
print(sdc)

   ## Infos on 2/3-Anonymity:
   ##
   ## Number of observations violating
   ##   - 2-anonymity: 85 (1.856%) | in original data: 653 (14.258%)
   ##   - 3-anonymity: 162 (3.537%) | in original data: 1087 (23.734%)
   ##   - 5-anonymity: 294 (6.419%) | in original data: 1781 (38.886%)
   ##
   ## ----------------------------------------------------------------------

table(extractManipData(sdc)$age)

   ##
   ##    1-9  10-19  20-29  30-39  40-49  50-59 60-130
   ##   1128   1110    635    629    447    250    283
```

Alternatively we could also re-run the code using function `undolast()` and performing `globalRecode()` again with new breaks.

A special case of global recoding is top and bottom coding. This method can be applied to ordinal or continuous variables. The idea for this approach is that all values above (i.e., top coding) and/or below (i.e., bottom coding) a pre-specified threshold value are combined into a new category. Top coding sets an upper limit on all values of a variable and replaces any value greater than this limit by the upper limit; for example, top coding would replace the age value for any individual aged above 80 with 80. Similarly, bottom coding replaces any value below a pre-specified lower limit by the lower limit; for example, bottom coding would replace the age value for any individual aged under 5 with 5. Instead of 80 or 5, respectively, one can also choose any number, e.g. the arithmetic mean of the high (or low) scores so that the arithmetic mean is not changed after top and bottom coding. As already seen in previous code blocks, in **sdcMicro** function `globalRecode()` can be used to perform both global recoding and the top/bottom coding, but there is also a specialized function called `topBotCoding` available. A help file with more examples as seen above is accessible using `?globalRecode` and `?topBotCoding`.

To replace high incomes in our test data set with the mean of the high incomes we can use the following code. Again the *sdcMicroObj* `sdc` updates its slots (e.g. risk and utility) after applying an anonymization method.

```
highIncomes <- testdata$income[testdata$income > 100000000]
sdc <- topBotCoding(sdc,
                    value = 100000000,
                    replacement = mean(highIncomes),
                    column = "income")
```

We note that **sdcMicroGUI** and the forthcoming browser-based version of the GUI offers a more user-friendly way of applying global recoding in general.

Exercises:

Question 4.1 **Global recoding**:

Take the `eusilc` data set from package **laeken** again. Assume the following disclosure scenario that defines `age`, `pb220a` (citizenship), `pl030` (education level), `rb090` (gender) and `hsize` (household size) as categorical key variables. Use the package **sdcMicro** to create an object of class *sdcMicroObj* considering the sampling weights (function argument `weightVar` in `createSdcObj`) and the household ID (function argument `hhId`). Reduce the disclosure risk through some reasonable recodings. Apply the recoding carefully, i.e. the analytical quality of the data should mostly remain the same while the disclosure risk should lower considerably.

4.2.2 Local Suppression

If unique combinations of categorical key variables remain after recoding, local suppression could be applied to the data with the aim to achieve k-anonymity. For the detailed explanation of k-anonymity, have a look in Sect. 3.3. Local suppression is a non-perturbative method typically applied to categorical variables. In this approach, missing values are created to replace certain values of individual variables to increase the number of key variables sharing the same pattern, thus reducing the record-level disclosure risks. There are two approaches to implementing local suppression. One approach sets the parameter k and tries to achieve k-anonymity (typically 3-anonymity) with minimum suppression of values. For example, in sdcMicroGUI (Kowarik et al. 2013), the user sets the value for k and orders key variables by the likelihood they will be suppressed. Then the application calls a heuristic algorithm to suppress a minimum number of values in the key variables to achieve k-anonymity. The second approach sets a record-level risk threshold. This method involves examining unsafe records with individual disclosure risks higher than the threshold and suppressing values of the selected key variable(s) for all the unsafe records.

Local suppression can be done univariate such as in Sect. 4.2.2.2, but in general it is a multivariate problem that is NP-hard, not solvable optimally in a reasonable time.

Say we have a problem P. It consists of achieving k-anonymity with the constraint to suppress as few values. In addition, a weighting determining the importance of key variables may just involve another additional parameter that can be incorporated in the constraints.

Some algorithms has been written which provide heuristic solutions.

Mondrian one solution is the algorithm Mondrian (LeFevre et al. 2006). This algorithm tries to combine categories to achieve k-anonymity, i.e. it is a sort of recoding based on the median counts of categories. However, this is a too over-simplistic approach of sorting and splitting, we do not obtained very promising results. More theoretical descriptions on the Mondrian algorithm can be found in Sect. 4.2.2.6.

all-M approach in μ-Argus and **sdcMicro** an algorithm is implemented that is often refered as the *all-M* approach. Using this algorithm, k-anonymity cannot be provided, but k-anonymity in all subsets of size M of the key variables only (in μ-Argus with the maximum of $M = 4$ variables). More precisely, the algorithm will provide k-anonymity for each combination of M key variables.

k-anonymity approach the default approach of **sdcMicro** in function `kAnon` ensures k-anonymity for the combination of all selected key variables. Naturally, this lead to k-anonymity but also to oversuppressions whenever the amount of key variables is too high, say larger than 7–10 key variables. Then often it is suitable to provide k-anonymity on a subset of variables (e.g. 7 variables) and apply a swapping method such as PRAM to the other key variables. Practical applications are shown in Sect. 4.2.2.3.

The real problem is the computation time and to find a good heuristic algorithm that solves our problem P to suppress as few values. Computation time increases a lot as soon as missing values are present in the data, since one has to treat them adequately. To decrease computation time, a lot of tricks must be used, based on (1) filtering the data in advance (2) ordering the data and (3) using C++ code and (4) thinking of good heuristics to find a good local optimal solution for this multivariate problem.

4.2.2.1 Illustrative Examples for Frequency Calculation with Missing Values

Here we show the whole problem of sample frequency calculation in case of missing values in the key variables, achieving k-anonymity and local suppression.

Remember, in Sect. 3.2.2 five possible methods how to calculate frequencies have been noted. For readability these five methods are defined once more:

1. (default method) Missing values increase frequencies in other categories.
2. (conservative method) Misssing values do not increase frequencies in other categories but in those observations where a missing occurs.
3. (category size) Missing values do increase frequencies in other categories by a factor c.
4. (conservative method 2) Misssing values do not increase frequencies in other categories.
5. (own category) Same as method 4, but missings are treated like an own category.

Table 4.1 Simple toy data set to explain different methods for sample frequency counts including missings/local suppressions in the following tables. Different solutions are displayed that show how to achieve 2-anonymity with local suppression based on different methods to calculate sample frequency counts

Original data					Method 1 (default)				
ID	Region	Status	Age group	f_k	ID	Region	Status	Age group	f_k
1	A	Single	30–49	2	1	A	Single	30–49	3
2	A	Married	30–49	2	2	A	Married	30–49	3
3	A	Married	30–49	2	3	A	Married	30–49	3
4	A	Single	30–49	2	4	A	Single	30–49	3
5	A	Widow	30–49	1	5	A	*	30–49	5
Method 2 (conservative)					Method 3 (category size)				
ID	Region	Status	Age group	f_k	ID	Region	Status	Age group	f_k
1	A	Single	30–49	2	1	A	Single	30–49	2.4
2	A	Married	30–49	2	2	A	Married	30–49	2.4
3	A	Married	30–49	2	3	A	Married	30–49	2.4
4	A	Single	30–49	2	4	A	Single	30–49	2.4
5	A	*	30–49	5	5	A	*	30–49	5 (or 3.2)
					$2.4 = \text{ratio single} \cdot 1 \text{ missing}$ $2.4 = \text{ratio married} \cdot 1 \text{ missing}$				
Method 4 (conservative 2)					Method 5 (own category)				
ID	Region	Status	Age group	f_k	ID	Region	Status	Age group	f_k
1	A	*	30–49	5	1	A	*	30–49	3
2	A	Married	30–49	2	2	A	Married	30–49	2
3	A	Married	30–49	2	3	A	Married	30–49	2
4	A	*	30–49	5	4	A	*	30–49	3
5	A	*	30–49	5	5	A	*	30–49	3

To motivate and explain these different concepts, a toy data example is used in the following, see Table 4.1 (upper left), with different values only in the second variable. It can be seen that 2-anonymity is not reached in the original data. For our scenarios, we assume that 2-anonymity should be achieved by local suppressions, i.e. the necessary suppressions under each frequency count method are made and sample frequencies are calculated. The results are displayed in Table 4.1. The default method, the conservative method and the category size method lead to the lowest numbers of necessary suppression to achieve 2-anonymity, i.e. with one suppression

Table 4.2 Simple toy data set to explain different methods for sample frequency counts including missings/local suppressions in the following tables. Different solutions are displayed to achieve 3-anonymity with local suppression based on different methods to calculate sample frequency counts

Original data					Method 1 (default)				
ID	Region	Status	Age group	f_k	ID	Region	Status	Age group	f_k
1	A	Single	30–49	2	1	A	Single	30–49	3
2	A	Married	30–49	2	2	A	Married	30–49	3
3	A	Married	30–49	2	3	A	Married	30–49	3
4	A	Single	30–49	2	4	A	Single	30–49	3
5	A	Widow	30–49	1	5	A	*	30–49	5
Method 2 (conservative)					Method 3 (category size)				
ID	Region	Status	Age group	f_k	ID	Region	Status	Age group	f_k
1	A	*	30–49	5	1	A	*	30–49	5
2	A	*	30–49	5	2	A	Married	30–49	3.2
3	A	*	30–49	5	3	A	Married	30–49	3.2
4	A	*	30–49	5	4	A	*	30–49	5
5	A	*	30–49	5	5	A	*	30–49	5
					$3.2 = 2 +$ ratio of married $\cdot$ 3 missings				
Method 4 (conservative 2)					Method 5 (own category)				
ID	Region	Status	Age group	f_k	ID	Region	Status	Age group	f_k
1	A	*	30–49	5	1	A	*	30–49	5
2	A	*	30–49	5	2	A	*	30–49	5
3	A	*	30–49	5	3	A	*	30–49	5
4	A	*	30–49	5	4	A	*	30–49	5
5	A	*	30–49	5	5	A	*	30–49	5

2-anonymity is reached. However, the sample frequencies are different. The method using an own category for the missing values is most strict. It leads to the lowest frequency counts with three suppressions.

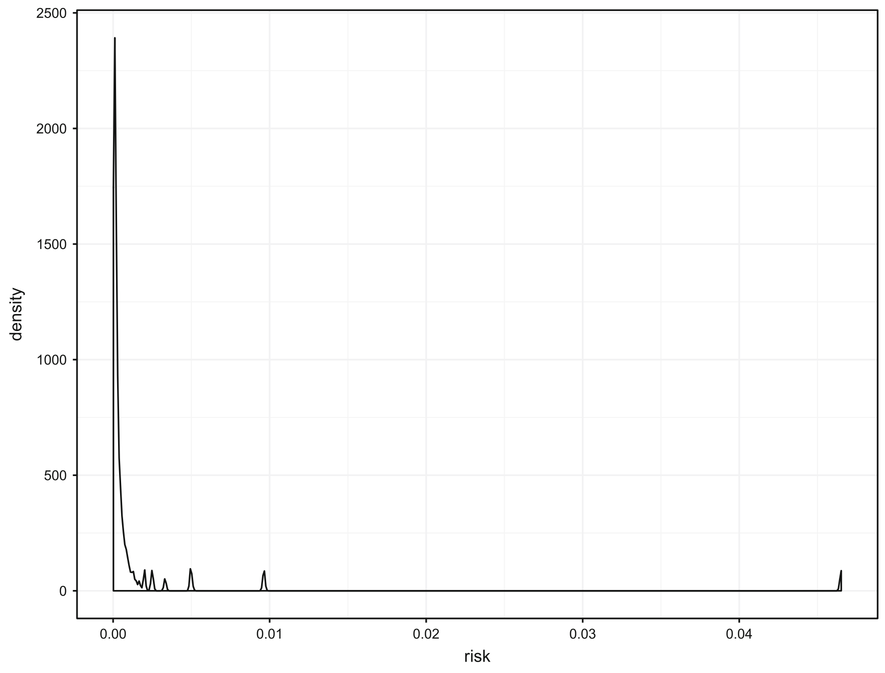
As from the previous table, from the upper left table in Table 4.2 it can be seen that 3-anonymity is not reached. For our scenarios, we assume that this time 3-anonymity should be achieved by local suppressions. The results are displayed in Table 4.2. The default method count a missing as a any possible category. Thus one suppression is already enough to achieve 3-anonymity. The category size methods counts the amount of values of category *married* plus the ratio of category *married* to the amount of categories times the number of missing values. All other approaches are more conservative and all values of *status* have to be suppressed to achieve 3-anonymity.

The choice of method for sample frequency calculation is best done with discussion with the law department or the management of the data holder. In the following, the so called default method is used. This method is used for years at Statistics Austria and every organisation who uses the package **sdcMicro**. The default method is amongst possible choices the less conservative one. It assumes that an intruder cannot be sure about the correct category of the suppressed value.

4.2.2.2 Univariate Local Suppression for Observations with High Risk

Using function `localSupp()` of **sdcMicro**, it is possible to suppress values of a key variable for all units with individual risks above a pre-defined threshold, given a disclosure scenario. This procedure requires user intervention by setting a threshold. This is illustrated in the following code listing. First, a density plot of the individual risks is plotted. This plot helps to determine the threshold. The majority of the data have very low risk, below 0.005, which already may serve as the value of the threshold. For those observations with higher individual risk than 0.005, the values of variable “urbur” are suppressed.

```r
## extract vector of individual risks
risk <- get.sdcMicroObj(sdc, "risk")$individual[, "risk"]
## have a look on the risk
ggplot(data.frame(risk), aes(x=risk)) + geom_density() + theme_bw()
```



```
## suppress values in urbrur for risk higher than 0.05
sdc <- localSupp(sdc, keyVar = "urbrur", threshold = 0.05)
```

4.2.2.3 Ensuring *k*-Anonymity—The Optimal Approach

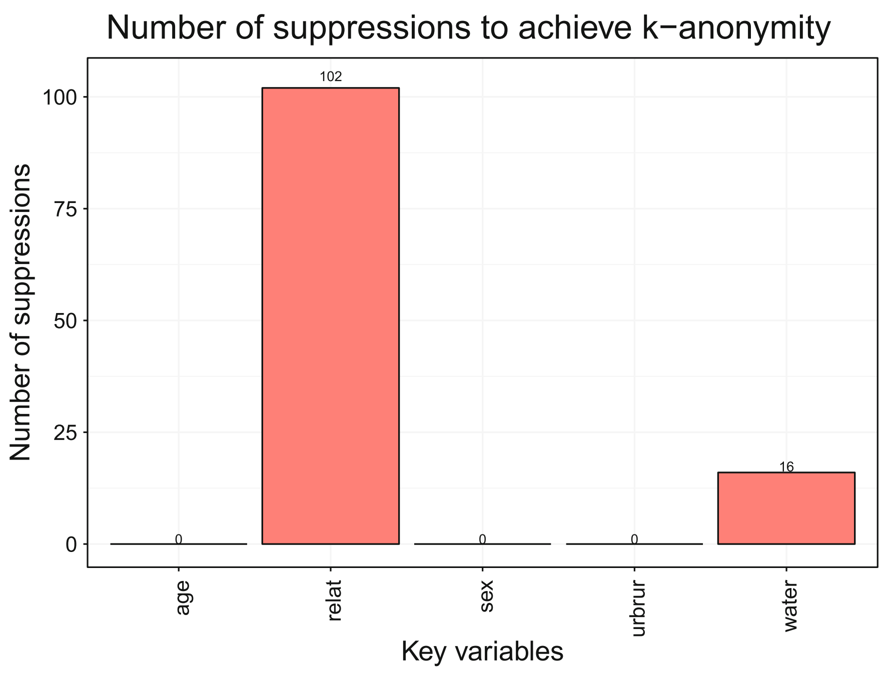
To automatically suppress a minimum amount of values in the key variables to achieve *k*-anonymity, one can use function `kAnon()`. In the code listing below, first the previous local suppression is reverted. Local suppression to achieve 3-anonymity is then done by setting parameter `k = 3`.

```
## undo last step (localSupp(...))
sdc <- undolast(sdc)

## local suppression to ensure 3-anonymity
## for given key variables
sdc <- kAnon(sdc, k=3)
```

The amount of suppression can be reported via a plot of the object `sdc`.

```
plot(sdc, "ls")
```



Also the print of this object gives information about the number and percentages of suppressions for each variable.

```
print(sdc, "ls")

   ## Local suppression:
   ##  KeyVar | Suppressions (#) | Suppressions (%)
   ##  urbrur |                0 |            0.000
   ##   water |               16 |            0.349
   ##     sex |                0 |            0.000
   ##     age |                0 |            0.000
   ##   relat |              102 |            2.227
   ## ---------------------------------------------------------------------
```

In this implementation, a heuristic algorithm is called to suppress as few values as possible. We see that 3-anonymity is ensured. In brackets the number of observations violating 2- and 3-anonymity in the original data set are reported.

```
print(sdc, "kAnon")

   ## Infos on 2/3-Anonymity:
   ##
   ## Number of observations violating
   ##   - 2-anonymity: 0 (0.000%) | in original data: 653 (14.258%)
   ##   - 3-anonymity: 0 (0.000%) | in original data: 1087 (23.734%)
   ##   - 5-anonymity: 138 (3.013%) | in original data: 1781 (38.886%)
   ##
   ## ---------------------------------------------------------------------
```

Importance of variables: It is possible to specify a desired ordering of key variables. If you have a look at the functions arguments in `kAnon()`, it can be seen that the function argument `importance` is responsible for it. The aim is that the higher the importance of a variable, the less suppressions are taken for this variable.

In other words, the algorithm allows the specification of a preference-vector that determines which key variables should be preferred when selecting values to be suppressed. By specifying this importance of variables as a parameter in `kAnon()`, for key variables with high importance, suppression will only take place if no other choices are possible. Still, it is possible to achieve *k*-anonymity for selected key variables.

```
## undo last local suppression
sdc <- undolast(sdc)
```

Remember, the key variables are

```
str(testdata[, sdc@keyVars])

   ## 'data.frame':    4580 obs. of  5 variables:
   ##  $ urbrur: int  2 2 2 2 2 2 2 2 2 2 ...
   ##  $ water : int  3 3 3 3 3 3 3 3 3 3 ...
   ##  $ sex   : int  1 2 1 1 1 2 2 2 1 2 ...
   ##  $ age   : int  46 41 9 6 52 47 13 19 9 16 ...
   ##  $ relat : int  1 2 3 3 1 2 3 3 3 3 ...
```

Another importance is assigned to those variables. In general, the lower the number, the less local suppressions. For any reason, we assume that variable *relat* is very important, giving the importance of 1. Variable *age* seems more important than *urbrur*, *water* and *sex*, thus giving age importance of 2. The rest of the variables are assigned by the numbers 5 (*urbrur*), 4 *water* and 3 *sex*. The result is now different from before.

```
sdc <- kAnon(sdc, importance = c(5,4,3,2,1), k = 3)
print(sdc, "ls")
```

```
   ## Local suppression:
   ##  KeyVar | Suppressions (#) | Suppressions (%)
   ##  urbrur |               44 |            0.961
   ##   water |               52 |            1.135
   ##     sex |               13 |            0.284
   ##     age |                9 |            0.197
   ##   relat |                4 |            0.087
   ## ---------------------------------------------------------------------
```

It can be seen that now other variables are mainly used for suppressions. In variable *relat*, for example, only 3 instead of 114 local suppression are made.

Stratification:

The methods can also be applied on each strata separately as long as the strata is specified in `createSdcObj`. Automatically then the algorithm ensures k-anonymity in all strata.

4.2.2.4 An Approach for Ensuring k-anonymity in Subsets

Due to computational issues another approach exists in μ-Argus, often refered as all-M approach. It is also implemented in **sdcMicro** (with no computational constraints), but mainly for comparison reasons. With this approach k-anonymity is usually not reached considering all key variables, but it can ensure k-anonymity on subsets of key variables.

It is also possible to provide k-anonymity for subsets of key-variables with varying parameter k. In the follow-up example we want to provide 10-anonymity for all combinations of 4 key variables, 20-anonymity for all combinations with 3 key variables and 30-anonymity for all combinations of 2 key variables. Note that strata are automatically considered.

```
sdc <- undolast(sdc) combs
<- 4:2 k <-
c(5,10,20)
sdc <- kAnon(sdc,
k=k, combs=combs)
print(sdc, "ls")

   ## Local suppression:
   ##  KeyVar | Suppressions (#) | Suppressions (%)
   ##  urbrur |                0 |            0.000 ##   water |
   96 |            2.096 ##     sex |                0 |
   0.000 ##     age |               98 |            2.140 ##   relat |
   147 |            3.210 ##
   ----------------------------------------------------------------------
```

```
print(sdc, "kAnon")
```

```
## Infos on 2/3-Anonymity:
##
## Number of observations violating
##   - 2-anonymity: 0 (0.000%) | in original data: 653 (14.258%)
##   - 3-anonymity: 3 (0.066%) | in original data: 1087 (23.734%)
##   - 5-anonymity: 30 (0.655%) | in original data: 1781 (38.886%)
##
## --------------------------------------------------------------------
```

Since both, k-anonymity is not ensured by this approach and a lot of parameters have to be set (k's and combinations), we strongly suggest to use the approach from the previous Sect. 4.2.2.3. In addition, more suppressions are made using this approach based on subsets.

However, if for any reason the SDC specialist decides to have a large amount of key variables this approach becomes important.

4.2.2.5 Choice of the Local Suppression Algorithm

The "Argus" subset approach and the optimal approach are equal if `combs` = # number of key variables. In any other case, the optimal approach ensures k-anonymity while the subset approach cannot ensure this. In addition, several parameters have to be determined when using the subset approach.

In general, it is advisable to provide k-anonymity for the most important indirect identifiers, i.e. to apply the optimal approach. The number of key variables is usually a number between 4 and 8. Scenarios with more than 8 key variables are seldom and rarely reported in literature. The optimal approach is preferable in this case since it ensures k-anonymity and does not lead to much over-suppression.

With a very high number of key variables, say 15, the optimal approach will reach k-anonymity but, because the high number of possible keys, many values might be suppressed.

With such a large number of key variables, one alternative is to apply PRAM (see Sect. 4.2.3) on some (e.g. 7 variables) and the optimal local suppression approach guarantees k-anonymity for the remaining variables (e.g. 8 variables). However, since it is not easy to determine the disclosure risk for prammed variables, alternatives are suggested in the following.

If PRAM should not be applied, k anonymity should be ensured in subsets of variables. Ideally, the number of variables in a subset should be high, e.g. 8. Then, 6435 combinations of 8 out of 15 variables exist. Thus, k-anonymity is ensured in each of the 6435 combinations by using the subset local suppression approach.

In the previous section, it can be seen that three values of k for three combinations are given. But using `combs <- 4:2` and `k <- c(5, 10, 20)` was somehow a subjective decision. On the one hand, one wants to guarantee low disclosure risk. With this aim, the values for `combs` should be high as well as the corresponding values of k. On the other hand, one wants to result in a low number of suppressions, thus low numbers of `comb` and `k` should be chosen. Currently, there is no algorithm available which would solve this optimization problem.

4.2.2.6 Mondrian

A very simple approach which is often cited is the Mondrian algorithm (LeFevre et al. 2006) which is based on multidimensional partitioning. For each split/partitioning, the key variable with the highest number of categories is chosen. The data are split according to the median in each partition whenever a split is possible. A split is possible when the splitted data contains enough observations. More precisely, this greedy (strict) median-partitioning algorithm results in a set of multidimensional regions, each containing between k and $2p(k-1)+m$, with p the number of key variables and m the number of distinct categories (for more details, see LeFevre et al. 2006).

This algorithm ensures k-anonymity but due to its simple procedure it may lead to over-suppressions. The algorithm is not included in **sdcMicro** but available upon request.

4.2.2.7 Linked Variables in Local Suppression

As mentioned in Chap. 2 after applying local suppression (`kAnon` or `localSuppression`) ghost/linked variables should have the same suppression pattern as the variable that they are linked to.

We give an illustrative artificial example to show how this concept works. Another variable with the same content is produced, in the following this is variable `electcon2`, `electcon3` (linked to `electcon` and `water2` (linked to `water`).

```
data("testdata", package = "sdcMicro")
testdata$electcon2 <- testdata$electcon
testdata$electcon3 <- testdata$electcon
testdata$water2 <- testdata$water
keyVars <- c("urbrur","roof","walls","water","electcon","relat","sex")
numVars <- c("expend","income","savings")
w <- "sampling_weight"
```

We want to make sure that some variables not used as key-variables will have the same suppression pattern as variables that have been selected as key variables. Thus, we are using *ghost*-variables. As indicated above, in our example we want variables `electcon2` and `electcon3` to be linked to key-variable `electcon` and we want variable`water2` to be linked to key-variable `water`.

```
ghostVars <- list()
ghostVars[[1]] <- list()
ghostVars[[1]][[1]] <- "electcon"
ghostVars[[1]][[2]] <- c("electcon2","electcon3")
ghostVars[[2]] <- list()
ghostVars[[2]][[1]] <- "water"
ghostVars[[2]][[2]] <- "water2"
```

Next we create the *sdcMicroObj* object.

```
obj <- createSdcObj(testdata, keyVars=keyVars,
  numVars=numVars, w=w, ghostVars=ghostVars)
```

We apply 3-anonymity to selected key variables

```
obj <- kAnon(obj, k=3)
obj
```

```
obj <- kAnon(obj, k=3)
obj
```

```
## Data set with 4580 rows and 17 columns.
##  --> Categorical key variables: urbrur, roof, walls, water, electcon,
##                                relat, sex
##  --> Numerical key variables: expend, income, savings
##  --> Weight variable: sampling_weight
## ----------------------------------------------------------------------
##
## 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!
##
##  Key Variable Number of categories     Mean size
##        urbrur                    2 (2)  2290.000 (2290.000)
##          roof                    6 (5)   913.600  (916.000)
##         walls                    3 (3)  1526.667 (1526.667)
##         water                    9 (8)   569.250  (572.500)
##      electcon                    4 (3)  1525.333 (1526.667)
##         relat                    9 (9)   554.000  (508.889)
##           sex                    2 (2)  2290.000 (2290.000)
##  Size of smallest
##               646  (646)
##                15   (16)
##                50   (50)
##                25   (26)
##               103  (107)
##                 3    (1)
##              2284 (2284)
## ----------------------------------------------------------------------
##
## Infos on 2/3-Anonymity:
##
## Number of observations violating
##  - 2-anonymity: 0 (original data: 157)
##  - 3-anonymity: 0 (original data: 281)
##
## Percentage of observations violating
##  - 2-anonymity: 0.000 % (original data: 3.428 %)
##  - 3-anonymity: 0.000 % (original data: 6.135 %)
## ----------------------------------------------------------------------
##
## Numerical key variables: expend, income, savings
##
## Disclosure risk (~100.00% in original data):
##  modified data: [0.00%; 100.00%]
##
## Current Information Loss in modified data (0.00% in original data):
##  IL1: 0.00
##  Difference of Eigenvalues: 0.000%
## ----------------------------------------------------------------------
##
## Local Suppression:
##    KeyVar | Suppressions (#) | Suppressions (%)
##    urbrur |                0 |            0.000
##      roof |               12 |            0.262
##     walls |                0 |            0.000
##     water |               26 |            0.568
##  electcon |                4 |            0.087
##     relat |              148 |            3.231
##       sex |                0 |            0.000
## ----------------------------------------------------------------------
```

In the following we can see that the suppression patterns of the key variable and its ghost variables are identical.

```
manipGhostVars <- get.sdcMicroObj(obj, "manipGhostVars")
manipKeyVars <- get.sdcMicroObj(obj, "manipKeyVars")
all(is.na(manipKeyVars$electcon) == is.na(manipGhostVars$electcon2))


   ## [1] TRUE

all(is.na(manipKeyVars$electcon) == is.na(manipGhostVars$electcon3))


   ## [1] TRUE

all(is.na(manipKeyVars$water) == is.na(manipGhostVars$water2))


   ## [1] TRUE
```

Exercises:

Question 4.2 **Local suppression**:
Take the `eusilc` data set from package **laeken** and the *sdcMicroObj* object produced in the previous example including all already done recodings. The task is now to ensure k-anonymity.

4.2.3 Post-randomization Method (PRAM)

If there are a larger number of categorical key variables (e.g., more than 5), recoding might not sufficiently reduce disclosure risks, or local suppression might lead to great information loss. In this case, the PRAM (Gouweleeuw et al. 1998) may be a more efficient alternative.

PRAM (Gouweleeuw et al. 1998) is a probabilistic, perturbative method for protecting categorical variables. The method swaps the categories for selected variables based on a pre-defined transition matrix, which specifies the probabilities for each category to be swapped with other categories.

To illustrate, consider the variable location, with three categories: $location = 1$ "east", $location = 2$ "middle", $location = 3$ "west". We define a 3-by-3 transition matrix, where p_{ij} is the probability of changing category i to j. For example, in the following matrix,

$$\mathbf{P} = \begin{pmatrix} 0.1 & 0.9 & 0 \\ 0.2 & 0.1 & 0.7 \\ 0.9 & 0 & 0.1 \end{pmatrix}$$

the probability that the value of the variable will stay the same after perturbation is 0.1, since we set $p_{11} = p_{22} = p_{33}$. The probability of east being changed into middle is p_{12}, while east will not be changed into west because p_{13} is set to be 0. PRAM protects the records by perturbing the original data file, while at the same time, since the probability mechanism used is known, the characteristics of the original data can be estimated from the perturbed data file. PRAM can be applied to each record independently, allowing the flexibility to specify the transition matrix as a function parameter according to desired effects. For example, it is possible to prohibit changes from one category to another by setting the corresponding probability in the transition matrix to 0, as shown in the example above. It is also possible to apply PRAM to subsets of the microdata independently.

```
set.seed(1234)
A <- as.factor(rep(c("A1","A2","A3"), each=5)); A


   ##  [1] A1 A1 A1 A1 A1 A2 A2 A2 A2 A2 A3 A3 A3 A3 A3
   ## Levels: A1 A2 A3
```

We apply `pram()` on vector `A` and print the result:

```
Apramed <- pram(A); Apramed


   ## Number of changed observations:
   ## - - - - - - - - - - -
   ## x != x_pram : 1 (6.67%)
```

The summary provides more detail. It shows a table of original frequencies and the corresponding table after applying PRAM. All transitions that took place are also listed:

```
summary(Apramed)

   ## Variable:  x
   ##  ----------------------
   ## Frequencies in original and perturbed data:
   ##                                  x A1 A2 A3 NA
   ## 1:           Original Frequencies  5  5  5  0
   ## 2: Frequencies after Perturbation  6  5  4  0
   ##
   ## Transitions:
   ##    transition Frequency
   ## 1:     1 --> 1         5
   ## 2:     2 --> 2         5
   ## 3:     3 --> 1         1
   ## 4:     3 --> 3         4
```

PRAM is applied to each observation independently and randomly. This means that different solutions are obtained for every run of PRAM if no seed is specified for the random number generator. A main advantage of the PRAM procedure is the flexibility of the method. Since the transition matrix can be specified freely as a function parameter, all desired effects can be modeled. For example, it is possible to prohibit changes from one category to another by setting the corresponding probability in the transition matrix to 0.

In **sdcMicro**, `pram()` allows PRAM to be performed. The corresponding help file can be accessed by typing `?pram` into an R console. When using the function it is possible to apply PRAM to sub-groups of the micro-data set independently. In this case, the user needs to select the stratification variable defining the sub-groups. If the specification of this variable is omitted, the PRAM procedure is applied to all observations of the data set.

We again create an object of class *sdcMicroObj*, also specifying a strata variable to apply PRAM within these strata independently. Note that when applying PRAM to variables, these variables should be saved as a `factor`.

```
require("sdcMicro")
data(testdata, package="sdcMicro")
# categorical variables should be saved as a factor
vars <- c('urbrur', 'water', 'sex', 'age', 'relat', 'walls', 'roof')
testdata[, vars] <- lapply(testdata[, vars], as.factor)

# setting up the SDC problem
sdc <- createSdcObj(testdata,
          keyVars = c('urbrur', 'water', 'sex', 'age', 'relat'),
          numVars = c('expend', 'income', 'savings'),
          pramVars = c("walls"),
          w = 'sampling_weight',
          hhId = 'ori_hid',
          strataVar = 'hhcivil'
          )
```

```
sdc <- pram(sdc)
print(sdc, "pram")



   ## Post-Randomization (PRAM):
   ## Variable: walls
   ## --> final Transition-Matrix:
   ##            2         3           9
   ## 2 0.84447887 0.1499160 0.005605102
   ## 3 0.05420769 0.9410131 0.004779201
   ## 9 0.13485876 0.3180081 0.547133184
   ##
   ## Changed observations:
   ##   variable nrChanges percChanges
   ## 1    walls       398        8.69
   ## ---------------------------------------------------------------------
```

Sometimes it is useful to apply pram not on all categories. To give an illustrative example, children with age below 16 should not be prammed in variable education. Or governmental organisations should not be prammed in variable ownership.

We show an example for our testdata set. We want to apply pram to variable *urbrur* for each group of variable *urbrur*. However, no value should be changed where *roof* equals 4. Thus we are creating a new value for these observations. First the previous application of PRAM is undone, then a new category is made for those observations that should not change.

```
sdc <- undolast(sdc)
sv <- testdata$urbrur
levels(sv) <- c("1","2","3")
sv[testdata$roof == 4] <- 3
```

Next PRAM is applied. For a later check, the original data as well as the prammed variable *roof* is extracted.

```
sdc <- pram(sdc, variables = "roof", strata_variables = sv)
orig <- get.sdcMicroObj(sdc, "origData")$roof
pramed <- get.sdcMicroObj(sdc, "manipPramVars")$roof
```

Now it can be validated if any of the category 4 in the prammed variable differs from the original category.

```
all(pramed[orig == 4] == 4)

   ## [1] FALSE
```

It can be seen that nothing is changed for this category.

4.3 Methods for Continuous Key Variables

4.3.1 Microaggregation

Micro-aggregation (Defays and Anwar 1998) is a perturbing method typically applied to continuous variables. It is also a natural approach to achieving k-anonymity. The method first partitions records into groups, then assigns an aggregate value (typically the arithmetic mean, but other robust methods are also possible) to each variable in the group. As an example, in Table 4, records are first partitioned into groups of two, and then the values are replaced by the group means. Note that in the example, by setting group size of two, micro-aggregation automatically achieves 2-anonymity with respect to the three key variables.

Individual values of the records for each variable are replaced by the group aggregation value, which is often the mean; as an example, see Table 4.3, where two values that are most similar are replaced by their column-wise means.

To preserve the multivariate structure of the data, the most challenging part of micro-aggregation is grouping records by how "similar" they are. The simplest method is to sort data based on a single variable in ascending or descending order. Another option is to cluster data first, and sort by the most influential variable in each cluster. These methods, however, might not be optimal for multivariate data (Templ and Meindl 2008).

Table 4.3 Example of micro-aggregation. Columns 1–3 contain the original variables, columns 4–6 the micro-aggregated values (rounded on two digits)

	Num1	Num2	Num3	Mic1	Mic2	Mic3
1	0.30	0.400	4	0.65	0.85	8.5
2	0.12	0.220	22	0.15	0.51	15.0
3	0.18	0.800	8	0.15	0.51	15.0
4	1.90	9.000	91	1.45	5.20	52.5
5	1.00	1.300	13	0.65	0.85	8.5
6	1.00	1.400	14	1.45	5.20	52.5
7	0.10	0.010	1	0.12	0.26	3.0
8	0.15	0.500	5	0.12	0.26	3.0

Individual ranking method

The individual ranking methods (Defays and Anwar 1998) is often applied because of its simplicity. The method replaces values by its aggregates column by column independently. First, the first column is sorted and the index of sorting is memorized to be able to sort the values back in the original order. Then the first k values are replaced by their aggregate (usually the arithmetic mean), the next k values are replaced by their aggregate, and so on, until all values are aggregated from the first variable. The variable is then back-sorted. This procedure is then applied on the other variables independently.

Microaggregation based on PCA

The Principle Component Analysis method sorts data on the first principal components (see, e.g.,Templ and Meindl 2008). A robust version of this method can be applied to clustered data for small- or medium-sized datasets (see, e.g.,Templ and Meindl 2008). This approach is fast and performs well whenever the first principal component explains a high percentage of the variance for the variables considered for micro-aggregation. If this is not the case, other algorithms are preferable, such as the MDAV algorithm explained next.

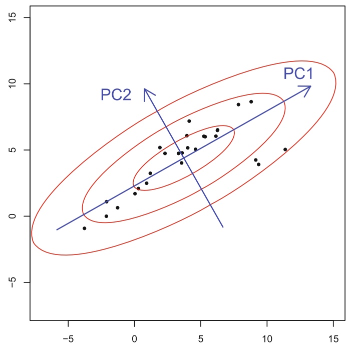
Figure 4.1 shows the procedure. First, the first principal component must be estimated and along the first principal component the values are then aggregated. As for almost any microaggregation procedure, the remaining $(2k - 1)$ are microaggregated whenever $nmodulok$ is unequal zero.

Microaggregation by MDAV

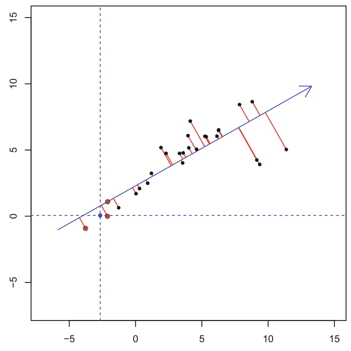
The Maximum Distance to Average Vector (MDAV) method is a standard method that groups records based on classical Euclidean distances in a multivariate space (Domingo-Ferrer and Mateo-Sanz 2002).

Figure 4.2 shows the MDAV procedure, which works as follows.

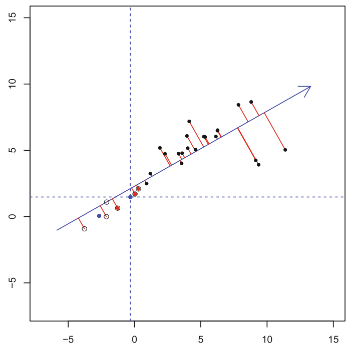
1. The center of the data is estimated using column-wise arithmetic means.
2. The farthest observation (Euclidean distance), say $\mathbf{x}_r$ from the center is then chosen.



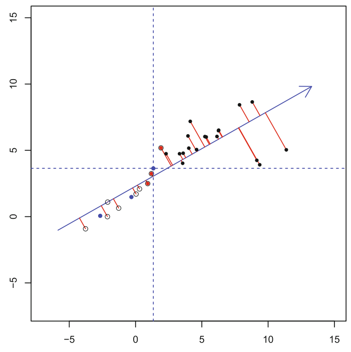
(a) First two principal components.



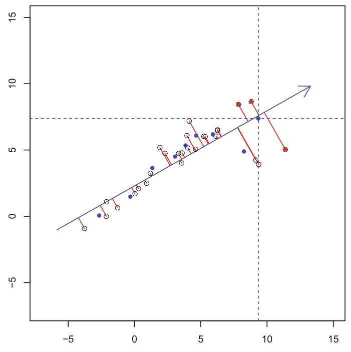
(b) Projection/sorting on first component. Aggregation along the sorted values (first three observations).



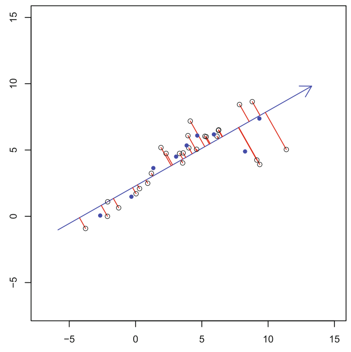
(c) Aggregation along the sorted values (next three observations).



(d) Aggregation along the sorted values (next three observations).

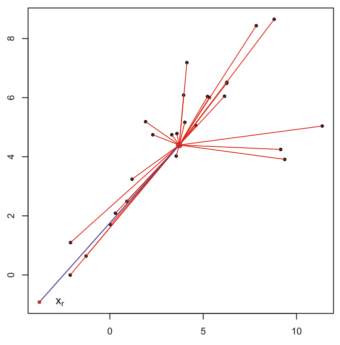


(e) Aggregation along the sorted values (remaining last observations).

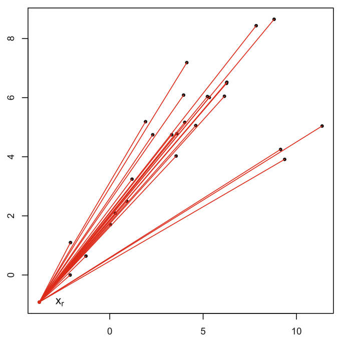


(f) Microaggregated values in blue.

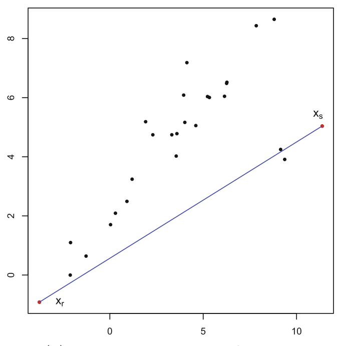
Fig. 4.1 Schematic workflow of microaggregation using principal component analysis on two-dimensional toy data. Aggregation is done along the first principal component



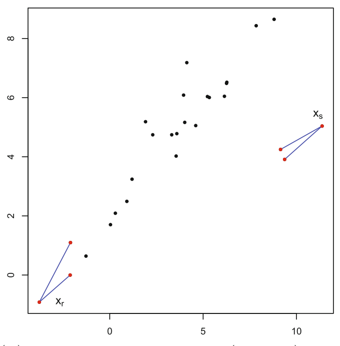
(a) Distances to the center.



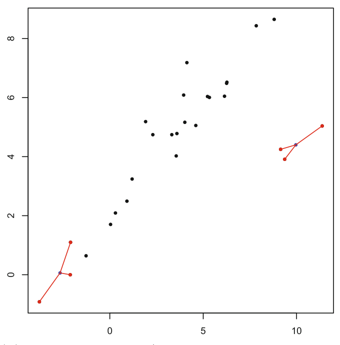
(b) Distances from observation $\mathbf{x}_r$ that is farthest from center to all other observations.



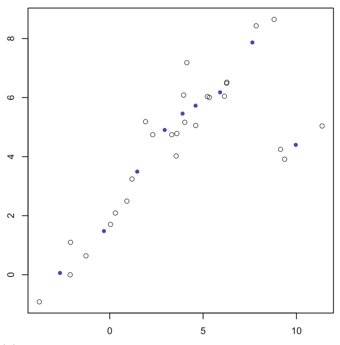
(c) Farthest point $\mathbf{x}_s$ from $\mathbf{x}_r$.



(d) Determining the $k-1$ (here: 2) nearest neighbors of $\mathbf{x}_r$ and $\mathbf{x}_s$.



(e) Replacement/aggregation with the mean.



(f) Resulting microaggregated observations (in blue) after applying the same procedure as long as all observations are microaggregated.

Fig. 4.2 Schematic workflow of microaggregation using MDAV on two-dimensional toy data. Aggregation is stepwise applied after all observations are microaggregated. The first step is shown in (**b**)–(**e**)

3. The farthest observation (Euclidean distance) from $\mathbf{x}_s$ is also selected.
4. $k - 1$ nearest neighbors (using Euclidean distances) are chosen for $\mathbf{x}_r$, and also for $\mathbf{x}_s$ and the arithmetic column-wise means are estimated for both groups of observations.
5. The corresponding observations are replaced by their arithmetic column-wise means.
6. Start at (1) for the remaining observations. A variant of the procedure is to start not at step (1) but at step (2), i.e. not to estimate the centers in each step but held them fixed. This iteration is done until less than or equal $2k - 1$ observations are left.
7. Aggregate the remaining observations.

Microaggregation by robust Mahalanobis distances

The MDAV method was further improved by replacing Euclidean distances with robust multivariate (Mahalanobis) distance measures (Templ and Meindl 2008). All of these methods are implemented in **sdcMicro** (Templ et al. 2015) and **sdcMicroGUI** (Kowarik et al. 2013).

Microaggregation for mixed scale data

To deal not only with continuous key variables but also with categorical (and ordered categorical or semi-continuous) in the same time, two central issues have to be changed. First, a more general definition of distance is required and secondly, for non-continuous variables a strategy for aggregation has to be developed.

For the first task, this microaggregation method is using distances computed similar to the Gower distance (Gower 1971). The distance function makes distinction between the variable types categorical, ordered categorical, continuous and semi-continuous (variables with a fixed probability mass at a constant value, e.g. 0). The distance calculation is explained in Sect. 5.2.1.

After calculating the distances and to microaggregate, it is necessary to sample a category in each group whenever a variable is non-(semi-)continuous. This probabilities corresponding to the occurrence of the level in the groups. The level with the most occurrences is then chosen, or if the maximum is not unique it is selected randomly.

Applying microaggregation in **sdcMicro** is straightforward, we again apply the corresponding function to our *sdcMicroObj* object, and the utility and risk will be automatically updated as well as other slots such as `manipData`.

```
sdc <- microaggregation(sdc)
print(sdc, "numrisk")


   ## Numerical key variables: expend, income, savings
   ##
   ## Disclosure risk is currently between [0.00%; 36.09%]
   ##
   ## Current Information Loss:
   ##   - IL1: 0.11
   ##   - Difference of Eigenvalues: 0.060%
   ## ---------------------------------------------------------------------
```

Depending on the chosen method in function `microaggregation()`, additional parameters can be specified. For example, it is possible to specify the number of observations that should be aggregated (parameter `aggr`) as well as the statistics used to calculate the aggregation (parameter `measure`). This statistics defaults is the arithmetic mean. It is also possible to perform micro-aggregation independently to pre-defined strata (the parameter is called `strata_variables`) or to use cluster methods to achieve the grouping (depending on parameter `method`). Let us undo the last action and perform microaggregation using different parameters. As mentioned before, the disclosure risk updates automatically.

```
sdc <- undolast(sdc)
sdc <- microaggregation(sdc,
                        aggr = 4,
                        strata_variables="age",
                        method="mdav")
print(sdc, "numrisk")

   ## Numerical key variables: expend, income, savings
   ##
   ## Disclosure risk is currently between [0.00%; 25.35%]
   ##
   ## Current Information Loss:
   ##   - IL1: 0.13
   ##   - Difference of Eigenvalues: 0.100%
   ## ----------------------------------------------------------------------
```

We can observe that the disclosure risk decreased considerably – ∼ only 2.88% of the original values do not fall within intervals calculated around the perturbed values, compare Sect. 3.11 on distance-based risk estimation. We can see that the information loss criteria increased slightly. All of the previous settings (and many more) can be applied in **sdcMicro**. The corresponding help file can be viewed with command `?microaggregation`.

Not very commonly used is the microaggregtion of categorical and continuous variables. This can be achieved with function `microaggrGower` that uses the Gower distance (Gower 1971) to measure the similarity between observations. For details have a look at the help function `?microaggrGower` in R. its application is also straightforward, just apply the function to an object of class *sdcMicroObj*. Note that in such an object, every important information is already stored and the variables to be microaggregated are chosen automatically.

```
sdc <- microaggrGower(sdc)
```

4.3.2 Noise Addition

Adding noise is a perturbative method typically applied to continuous variables. The idea is to add or multiply a stochastic or randomized number to the original values to protect data from exact matching with external files. While this approach sounds simple in principle, many different algorithms can be used. In this section, we introduce the uncorrelated and correlated additive noise methods.

Adding noise should be used with caution, as the results depend greatly on the amount of noise chosen.

4.3.2.1 Uncorrelated Additive Noise

Uncorrelated additive noise (see, e.g., Hundepool et al. 2007) can be expressed as the following:

$$\mathbf{z}_j = \mathbf{x}_j + \epsilon_j \quad , \tag{4.1}$$

where vector $\mathbf{x}_j$ represents the original values of variable j, $\mathbf{z}_j$ represents the perturbed values of variable j and ϵ_j (uncorrelated noise, or white noise) denotes normally distributed errors with $Cov(\epsilon_l, \epsilon_k) = 0$ for all $k \neq l$. In matrix notation this looks like

$$\mathbf{Z} = \mathbf{X} + \epsilon \quad , \tag{4.2}$$

with $X \sim (\mu, \Sigma)$, $\epsilon \sim N(0, \Sigma_\epsilon)$ and $\Sigma_\epsilon = c \cdot diag(\sigma_1^2, \sigma_2^2, \ldots, \sigma_p^2)$, for a constant $c > 0$.

In the following code, synthetic toy data are simulated and noise is added. The original but also the perturbed data are displayed in Fig. 4.3. Noise can be added to variables using function `addNoise()` and its parameter `noise`. The corresponding help file can be accessed with `?addNoise`. In addition, a 97.5% tolerance ellipse showing the covariance structure is drawn for both the original and the perturbed data. More noise than probably needed is added to the original variables in order to show the flaws of this method. It is clearly visible that the covariance of the data set is not respected in the noise addition. The perturbed data have a different covariance structure than the original data.

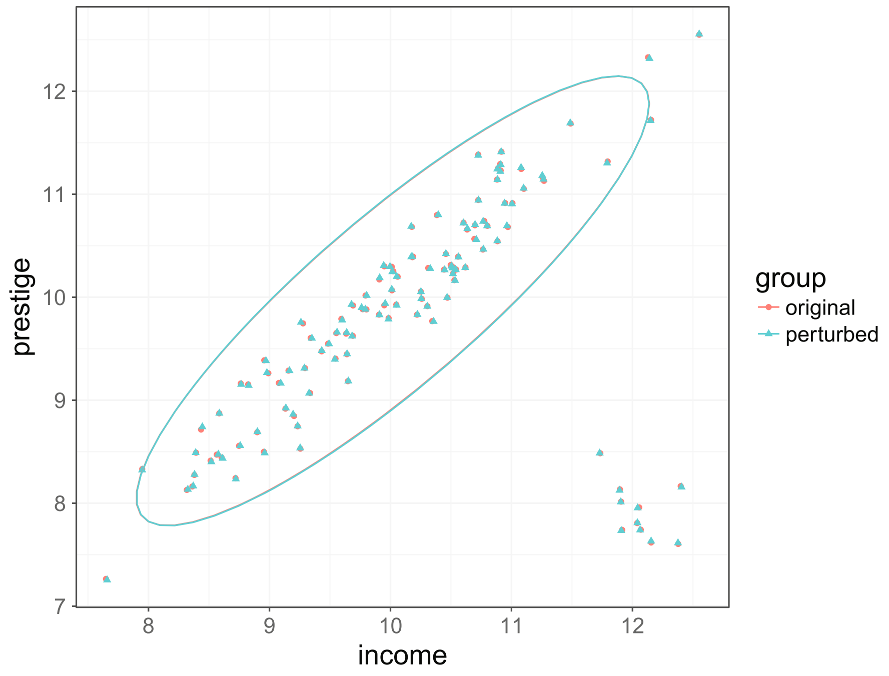


Fig. 4.3 Adding additive noise. More noise than probably needed is added to see the flaws of the method

```
## we generate some synthetic bivariate toy data to illustrate noise addition
X <- MASS::mvrnorm(100,
                   mu = c(10,10),
                   Sigma = matrix(c(1,0.95,0.95,1),
                              byrow = TRUE, ncol=2))
X2 <- MASS::mvrnorm(10,
                    mu = c(12,8),
                    Sigma = matrix(c(0.1,0,0,0.1),
                              byrow = TRUE, ncol=2))
X <- data.frame(rbind(X, X2))
colnames(X) <- c("income","prestige")
head(X)

   ##      income  prestige
   ## 1 10.887193 11.249357
   ## 2 11.101999 11.053242
   ## 3 12.128663 12.329934
   ## 4  8.585157  8.873169
   ## 5  8.899352  8.689755
   ## 6  8.564348  8.472442

## now add noise:
set.seed(123)
Y <- addNoise(X, method="additive", noise=0.7)$xm
```

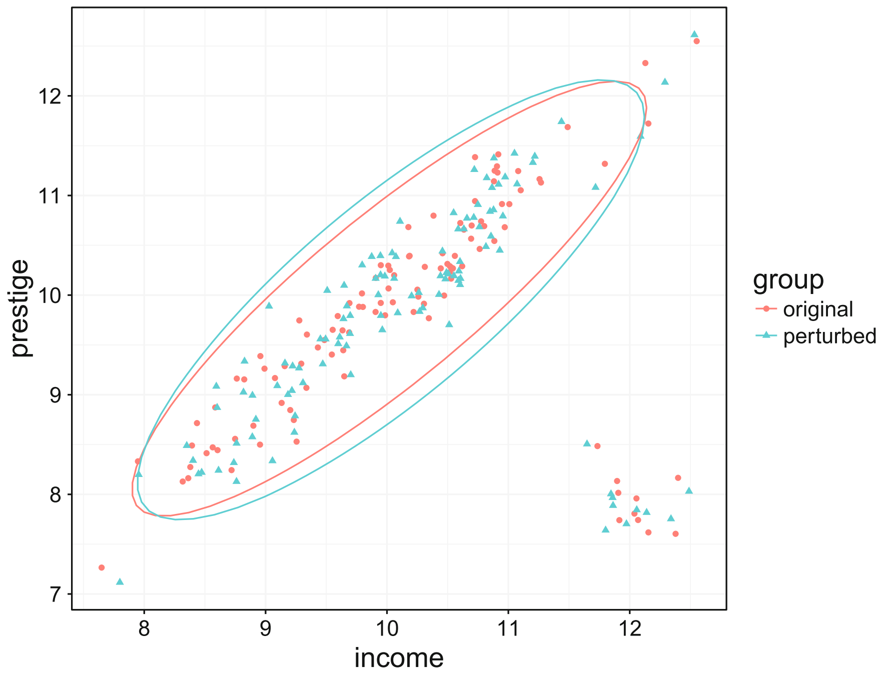


Fig. 4.4 Adding correlated noise

4.3.2.2 Correlated Additive Noise

While adding uncorrelated additive noise preserves the means and variance of the original data, co-variances and correlation coefficients are not preserved. It is preferable to apply correlated noise because the co-variance matrix of the errors is proportional to the co-variance matrix of the original data (Brand 2002).

The difference to the uncorrelated noise method is that the covariance matrix of the errors is now designed to be proportional to the covariance of the original data, i.e. $\epsilon \sim N(0, \Sigma_\epsilon = c\Sigma)$. The following holds

$$\Sigma_Z = \Sigma + c\Sigma = (1 + \alpha)\Sigma \quad . \tag{4.3}$$

Whenever the constant c is known, the covariance of the original data can be estimated from the perturbed data set.

Figure 4.4 presents the original data as well as the perturbed data set with the additive correlated noise method. Again, more noise as probably necessary is added to better see the effect of the method. Nevertheless, at least the covariance structure of this toy data set is well preserved.

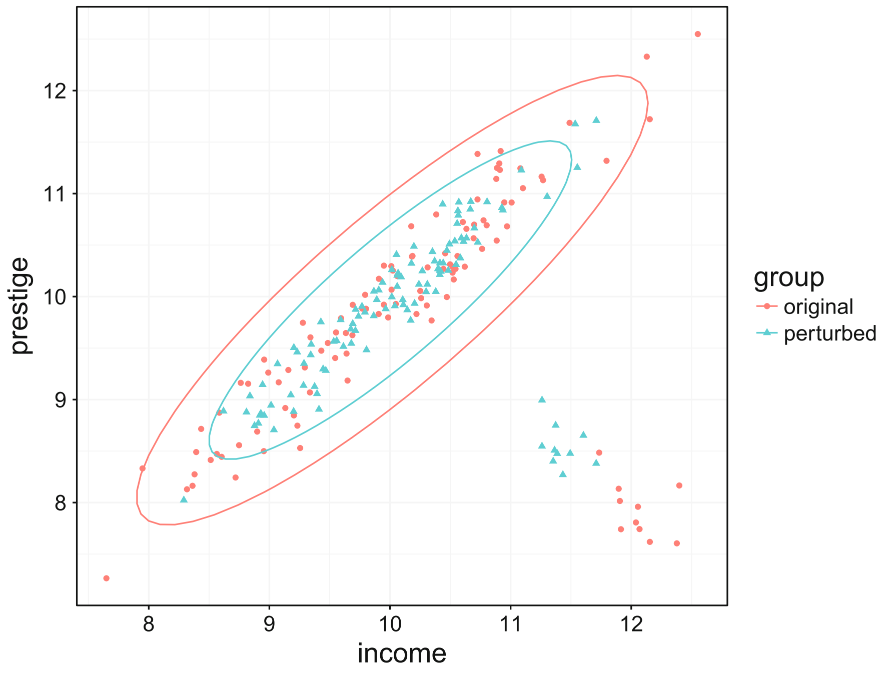


Fig. 4.5 Adding correlated noise based on transformations

4.3.2.3 Adding Correlated Noise Based on Transformations

Adding correlated noise based on transformations (Kim 1986) is another method that is also implemented in **sdcMicro**. Here we calculate $d = (1 - c^2)\epsilon$ and then $x_j d + c z_j$ whereas y_j are random numbers from $N(\frac{(1-d)\bar{x}_j}{c}, s_j)$, with $\bar{x}_j$ and s_j the mean standard deviation of x_j.

Another similar method which takes the sample size into account is described, for example, in Brand (2002). It is a method which is often denoted as the restricted correlated noise method and which is implemented in sdcMicro as well (method `restr.`).

Figure 4.5 presents the original data as well as the perturbed data set with the transformation based additive correlated noise method. Again, more noise as probably necessary is added to better see the effect of the method. The tails of the distribution are wider for the perturbed data and also a bias is introduced for the outlying group in the bottom-right of the graphic.

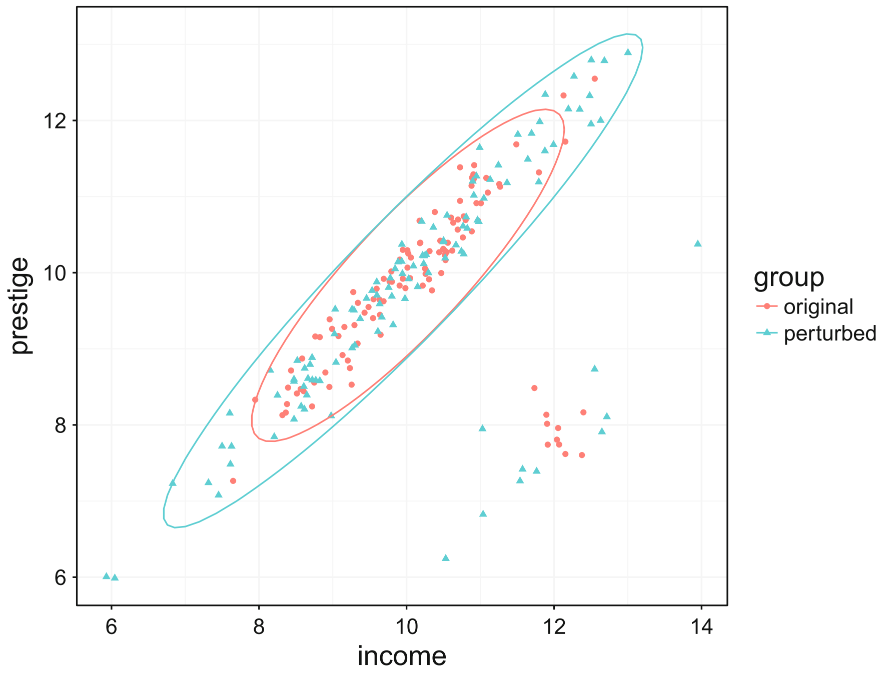


Fig. 4.6 Adding noise by Random Orthogonal Matrix Masking

4.3.2.4 Random Orthogonal Matrix Masking

Furthermore, for method ROMM (Random Orthogonal Matrix Masking) (Ting et al. 2005) perturbed data are obtained by $\mathbf{Z} = \mathbf{AX}$, whereas $\mathbf{A}$ is randomly generated and fulfils $\mathbf{A}^{-1} = \mathbf{A}^T$ (orthogonality condition). To obtain an orthogonal matrix as described in Ting et al. (2005) the Gram-Schmidt procedure (see, e.g., Golub and Van Loan 1996) is used.

Figure 4.6 presents the original data as well as the perturbed data set using ROMM. Again, more noise as probably necessary is added (parameter `p`) to better see the potential flaws of the method. The tails of the distribution are more narrow for the perturbed data.

4.3.2.5 Robustness Issues

The distribution of the original variables $\mathbf{x}_j$ may not follow a normal distribution and, especially the correlated noise methods may be influenced by this fact.

In this case, a robust version of the correlated noise method is described in detail by Templ and Meindl (2008). Hereby, the covariance matrices are estimated with robust covariance estimators such as the MCD estimator (Rousseeuw and Van Driessen 1999).

In **sdcMicro**, many algorithms are implemented that can be used to add noise to continuous variables. For example, it is possible to add noise only to outlying observations. In this case it is assumed that such observations possess higher risks than non-outlying observations.

4.3.2.6 Applying Adding Noise Methods

Of course, any mentioned noise methods can be applied to objects from class *sdcMicroObj* using function `addNoise`, e.g.

We now want to apply a method for adding correlated noise based on non-perturbed data after we undo microaggregation on our previously generated `sdc` object:

```
sdc <- undolast(sdc)
sdc <- addNoise(sdc, method="correlated2")
print(sdc, "numrisk")

   ## Numerical key variables: expend, income, savings
   ##
   ## Disclosure risk is currently between [0.00
   ##
   ## Current Information Loss:
   ##    - IL1: 0.11
   ##    - Difference of Eigenvalues: 0.060
   ## -------------------------------------------------------------------
```

We see that the data utility measure is comparable with the microaggregation on strata in the previous code chunk but the risk is higher than before using microaggregation.

Other methods ensure that the amount of noise added takes into account the underlying sample size and sampling weights.

4.3.3 Shuffling

Shuffling (Muralidhar and Sarathy 2006) generates new values for selected sensitive variables based on the conditional density of sensitive variables given non-sensitive variables. As a rough illustration, assume we have two sensitive variables, income and savings, which contain confidential information. We first use age, occupation, race and education variables as predictors in a regression model to simulate a new set of values for income and savings. We then apply reverse mapping (i.e., shuffling) to replace ranked new values with the ranked original values for income and savings. This way, the shuffled data consists of the original values of the sensitive variables. Moreover, Muralidhar and Sarathy (2006) showed that since we only need the rank of the perturbed value in this approach, instead of generating a new value, shuffling can be implemented using only the rank order correlation matrix (measuring the strength of the association between the ranked sensitive variables and ranked non-sensitive variables) and the ranks of values of non-sensitive variables.

In the following code we do not use default values because we want to show how to specify the form of the model. We first restore the previous results and remove the effect of adding noise using `undolast()`.

```r
sdc <- undolast(sdc)
form <- formula(expend + income + savings ~ age + urbrur + water +
  electcon + age + sex, data=testdata)
sdc <- shuffle(sdc, form)
print(sdc, "numrisk")

   ## Numerical key variables: expend, income, savings
   ##
   ## Disclosure risk is currently between [0.00%; 0.22%]
   ##
   ## Current Information Loss:
   ##   - IL1: 1.63
   ##   - Difference of Eigenvalues: 2.090%
   ## ---------------------------------------------------------------------
```

To find a good model is essential to provide good results. If we would for example add variable `walls` to the model, the results are no longer of high quality.

Exercises:

Question 4.3 **Adding noise**:

Take the data set `EIA`. Assume that variables `RESSALES`, `COMSALES`, `INDSALES`, and `OTHRSALES` are continuous key variables, and `UTILNAME` and `STATE` the categorical key variables. Define your *sdcMicroObj* object. Compare additive noise and the method called `correlated2` for adding noise in terms of disclosure risk. Remark: In the next chapter, we will also compare the data utility.

Question 4.4 **Microaggregation**

Apply microaggregation on each state independently.

Question 4.5 **Microaggregation**

Assume that in general your management defines the disclosure risk on the basis of 3-anonymity for your categorical key variables. Is your data set *safe* if you apply microaggregation on your continuous key variables and ensure 3-anonymity for your categorical key variables?

References

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

LeFevre, K., DeWitt, D. J., & Ramakrishnan, R. (2006). Mondrian multidimensional k-anonymity. In *ICDE '06: Proceedings of the 22nd international conference on data engineering* (p. 25).

Gouweleeuw, J., Kooiman, P., Willenborg, L., & De Wolf, P.-P. (1998). Post randomisation for statistical disclosure control: Theory and implementation. *Journal of Official Statistics14*(4), 463–478.

Defays, D., & Anwar, M. N. (1998). Masking microdata using micro-aggregation. *Journal of Official Statistics*, *14*(4), 449–461.

Templ, M., & Meindl, B. (2008b). Robustification of microdata masking methods and the comparison with existing methods. In *Privacy in statistical databases*. Lecture Notes in Computer Science (Vol. 5262, pp. 113–126). Springer.

Domingo-Ferrer, J., & Mateo-Sanz, J. M. (2002). Practical data-oriented microaggregation for statistical disclosure control. *IEEE Transactions on Knowledge and Data Engineering*, *14*(1), 189–201.

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.

Gower, J. C. (1971a). A general coefficient of similarity and some of its properties. *Biometrics*, *27*(4), 857–871.

Hundepool, A., et al. (2007). *Handbook on statistical disclosure control*.

Brand, R. (2002). *Microdata protection through noise addition*. Lecture Notes in Computer Science London: Springer.

Kim, J. J. (1986). A method for limiting disclosure in microdata based on random noise and transformation. In *Proceedings of the section on survey research methods* (pp. 303–308). American Statistical Association.

Ting, D., Fienberg, S., & Trottini, M. (2005). Romm methodology for microdata release. In *Monographs of official statistics: Work session on statistical data confidentiality*. Luxembourg: Eurostat.

Golub, G. H., & Van Loan, C. F. (1996). *Matrix computations* (3rd ed.). Baltimore: Johns Hopkins University Press.

Rousseeuw, P. J., & Van Driessen, K. (1999). A fast algorithm for the minimum covariance determinant estimator. *Technometrics*, *41*, 212–223.

Muralidhar, K., & Sarathy, R. (2006). Data shuffling—A new masking approach for numerical data. *Management Science*, *52*(2), 658–670.