A Graphical User Interface for Microdata Protection Which Provides Reproducibility and Interactions: the sdcMicro GUI

Matthias Templ[*,**], **Thomas Petelin**[*]

[*]Department of Statistics and Probability Theory, Vienna University of Technology, Wieder Hauptstr. 8-10, 1040 Vienna, Austria. [**]Department of Methodology, Statistics Austria, Guglgasse 13, 1110 Vienna, Austria.

E-mail: templ@tuwien.ac.at

Abstract. The proposed graphical user interface (GUI) for microdata protection serves as an easy-to-handle tool for users who want to use the sdcMicro package for statistical disclosure control but are not familiar with the native R command line interface. In addition to that, interactions between objects that result from the anonymization process are provided within this GUI. This allows an automated recalculation and display of frequency counts, individual risk, information loss and data utility after each anonymization step. Furthermore, the code of every anonymization step carried out within the GUI is saved in a script, which can easily be modified and re-used.

The tool is open-source and can be downloaded for all platforms from the comprehensive R archive network (CRAN). A detailed description of the design of this GUI is given, which allows researchers and users to enhance or modify the tool.

Keywords. Official Statistics, Microdata Protection, R, Graphical User Interface.

1 Introduction

Microdata is information at the level of individual respondents. For instance, information about the address, hometown, education level, employment status, age per person in a survey. This kind of information is needed for research on various topics.

In fact, an increasing demand for microdata among researchers has been noted over the last few years. However, microdata must keep up the required statistical privacy, for example, when data is provided to researchers coming from other institutions as the data holders.

Although data privacy is regulated differently in almost all countries, the privacy of any statistical unit must be preserved in general, i.e. the risk of re-identification should be low.

To eliminate or at least minimize the risk of re-identification, perturbation methods for the anonymization of microdata could be applied to the raw data. Perturbation methods can be separated into two groups due to the distribution of the (key) variables.

`sdcMicro` [10, 9] is a highly flexible package to generate anonymized microdata files [for applications, see, e.g., 6]. It includes all methods of the popular closed-source μ-Argus software [5] plus several new ones [see, e.g., 13, 9, 11, 12] and it is open source and distributed via CRAN.

1.1 Categorical Variables

The main goal is to reach both a low re-identification risk and a minimum modification of the data. The general approach to reach anonymity of categorical variables is an explanatory one [9], i.e. by trying out different recodings, considering limitations which may given by subject matter specialists. To achieve this goal one can use the package `sdcMicro` [10] and the easy-to-use GUI of this package.
At the end of such an anonymization process, some specific values may be suppressed in an optimal way (minimizing a cost function, see the `sdcMirco` manual [10], for example).

1.2 Numerical Variables

In comparison to categorical variables, almost all numerical observations are unique. A set of numerical variables could be used to match an observation to a specific individual. The aim is to keep the structure of the dataset while minimizing the risk of re-identification by applying perturbation to the data.

The outline of the paper is as follows: Section 2 reports all functions of `sdcMicro` which are fully supported by the GUI. The advantages of this GUI are outlined as well as the main features of the GUI are presented. Finally, in Section 4 details about the implementation are provided. Since the software is an open-source project, it needs such accompanying information to allow for enhancement or modification of the GUI by other institutions, users and researchers.

2 Supported Functionality from **`sdcMicro`**

Although every function and command can be applied within the GUI script window (see Figure 7), Figure 1 and the following list give an overview of the implemented `sdcMicro` functions which are explicitly supported by the GUI.

- `freqCalc()`: Fast computation of frequency counts on unit level. One of the two basic functions needed for this package to do further operations.

- `indivRisk()`: Estimation of the individual risk for each unit using the Benedetti-Franconi model [see, e.g., 4]. After the risk is computed, the function `localSupp()` could be used for the protection of values at high individual risk, for example.

- `plot.indivRisk()`: A plot of an interactive histogram or empirical cumulative distribution function curve with interactive sliders. For objects of class *"indivRisk"* only the command `plot()` is needed.

- `localSupp()`: A simple algorithm to perform local suppression. This function suppresses values with a high risk of re-identification.

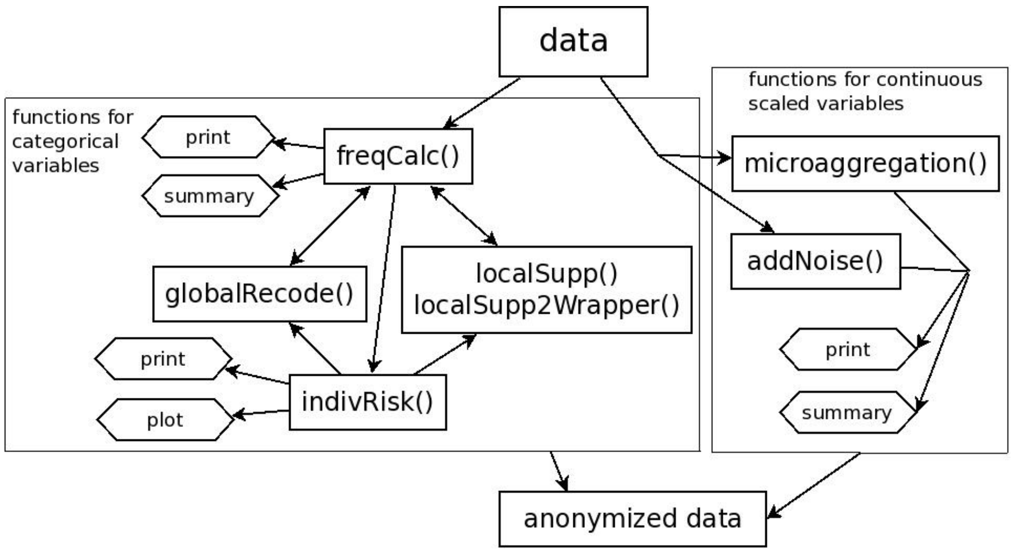


Figure 1: Functions which are supported by the GUI and their relationship.

- `localSupp2Wrapper()`: This is a wrapper for `localSupp2()` which is a more sophisticated version of `localSupp()`. It guarantees k-anonymity [7] and the importance of variables can be determined (the more important, the fewer suppressions).
- `globalRecode()`: A simple procedure to group and (re)label a variable.
- `microaggregation()`: Five methods (out of more than 10 in `sdcMicro`), namely RMD ([8]), pca (see, e.g., [1]), clustppca [8], influence [8] are supported by the GUI.
- `addNoise()`: This function supports different methods to perform perturbation of numerical variables (e.g., by correlated noise [2], [the implemented methods are listed in 9]).
- Certain print and summary methods for specific objects.

Some of these functions are designed to work with categorical variables and others with numerical variables. Figure 1 shows also which function is assigned to what kind of variable type and the relations between the functions.

3 The `sdcMicro` GUI

3.1 Advantages of the GUI Compared to the CLI Version and to Other GUI's for Microdata Protection

It is well known that command-line interfaces imply a steep learning curve, especially when using a object-oriented programming language as R. Often, a command-line interface may allow for greater efficiency and productivity once the language and specific commands are learnt, but reaching this level may take a long time. While experienced researchers may be able to work with the `sdcMicro` package using the command-line interface of R, many users prefer and may need a software which provides a practical easy-to-use tool to anonymize their data sets.

In fact, the popular software μ-Argus has provided a graphical user interface for many years (but it does not allow access to a command-line interface). However, users may be handicapped by restrictions of this graphical user interface because results are not reproducible.

The proposed GUI provides full functionality of the main functions of `sdcMicro` and offers some more features to the users, for example:

- comprehensive overview of the main functions and their output (see, e.g., Section 3.2)
- facilities to rename and regroup categories and to change values of a variable (see Section 3.5.1)
- automated recalculation of frequencies and individual risk after each step (see Section 3.5.1)
- display of the output of the frequency and risk estimation interactively within the GUI (see Section 3.5.1)
- recording of the code and the selected values after each step (see Section 4.4)
- possibility to save/load/edit a script for later re-use (see Section 4.4)
- (R specific) no need to manually re-assign computed data to a data frame
- simplified load / save data option

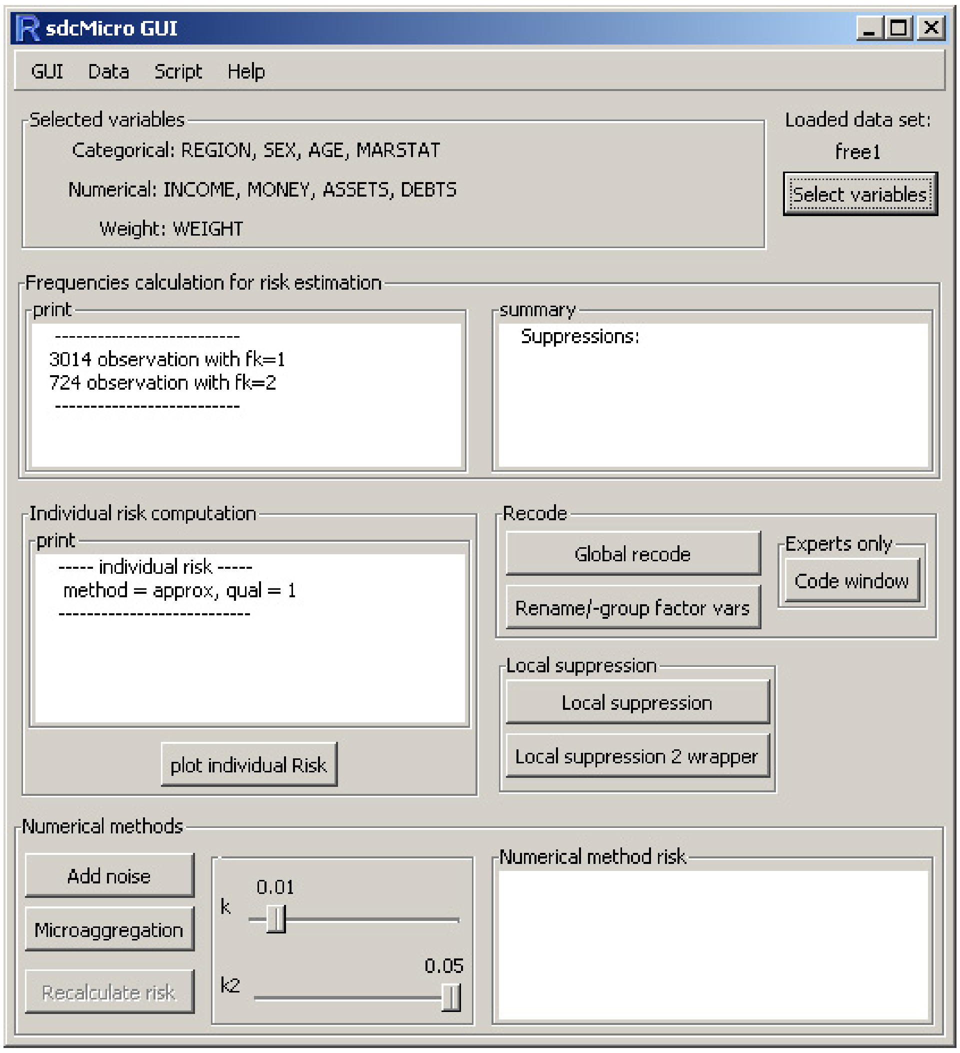
In statistics, it is very important that graphical user interfaces allow to reproduce any result easily. This is very important during the process of data anonymization. If the data has changed it is often necessary to undo the last few operations on the data without applying all the previous mouse clicks on the data set again. This is solved within the GUI by recording all steps and parameters in a script.

Compared to the well-known μ-Argus software for microdata protection [5], the `sdcMicro` GUI provides additionally flexibility due to the automated recalculation and display of the most important estimations results, and the possibility to re-run all the previous mouse clicks by loading a saved script (which can also be modified) without any repetition of clicks.

The anonymization of data is a highly interactive process [see, e.g., 6, 9]; for example, recodings are tried out but restored whenever the information loss is too high or the re-identification risk is not reduced. Thus, the interactive features of the GUI are absolutely necessary for an effective anonymization of microdata.

3.2 Main Window

The main window (see Figure 2) offers direct access to all available functions. It is designed to give a brief summary of the frequency calculation and risk estimation and provides two sets of buttons for operations to use with categorical and numerical variables.



Categorical: REGION, SEX, AGE, MARSTAT

Numerical: INCOME, MONEY, ASSETS, DEBTS

Weight: WEIGHT

free1

```
 -------------------------
3014 observation with fk=1
724 observation with fk=2
 -------------------------
```

```
Suppressions:
```

```
 ----- individual risk -----
  method = approx, qual = 1
 --------------------------
```

k 0.01

k2 0.05

Figure 2: Main window of the sdcMicro GUI. Specific variables of the μ-Argus test data set are already selected. This data set is accessible within the sdcMicro package but can also be downloaded from the website of the CASC project (*http://neon.vb.cbs.nl/casc/*)

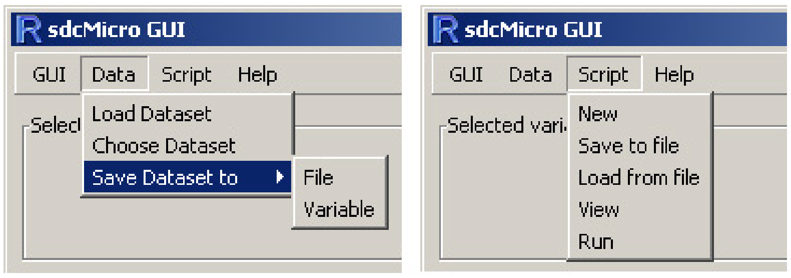


Figure 3: Menu options

3.3 Menu

As mentioned before, by using the GUI a user does not have to use R commands to load, choose and save a dataset. Also the possibility to load a script including the code is available via the menu. For a detailed description of this part, see 3.7.

3.4 Variable Selection

After choosing a dataset to work on, the variable selection functionality must be used to determine those variables to be anonymized. This can be done by clicking on the `Select Variables` menu. Whenever this button is clicked, another window pops up (Figure 4) for variable selection. Most functions only apply to categorical or numerical variables; therefore a pre-selection is advantageous (the same concept is used by the μ-Argus software). The user has to choose which variables belong to which groups of variables and if the data provides sampling weights the user has to choose a weight variable as well.

3.5 Categorical Variables

3.5.1 Recoding

By clicking on the `Global Recode` button (see Figure 2) a window pops up (Figure 5) in which the user is able to modify categorical variables. Whenever a recoding is done, the two frames automatically refresh and a summary is displayed. `Global Recode` offers the option to convert a categorical variable directly to the type factor or split it up, reassign labels for the single groups and create a factor out of it.

By clicking on the `Rename/-group factor vars`[1] button (see Figure 2), a window to rename and group categories of a variable pops up (see Figure 6). For example, standard recodings such as *municipality* ⇒ *Länder* can be easily made after clicking on the button `rename`. In Figure 6 the category *adult* is replaced by *age18plus*. Even groupings can easily be made when selecting categories and hitting the `group` button. The same window as in Figure 6 (window at the right hand side) pops up and a new name of this group can be assigned. These functionalities allow for convenient modification of categorical variables without using the powerful R command-line interface.

The `code window` (see Figure 7, accessible by the "experts only" frame, see Figure 2) gives users with R knowledge the possibility to use the GUI and additionally manipulate and work with the data using the R command-line interface. The user can access the current

[1]A factor type vector contains a set of numeric codes with character-valued labels.

modified dataset from the GUI in the code window, apply any R command (working in an specific environment, see Section 4.2) and continue to use the GUI with the modified dataset.

3.5.2 Local Suppression

Figure 8 is accessible by clicking on the button for `Local suppression` (picture on the left-hand side of Figure 8) and `Local Suppression 2 wrapper` (picture on the right-hand side) on the main window. Local suppression is designed to suppress values in one variable in according to a chosen threshold of risk. The other function, *localSupp2Wrapper()*, tries to achieve k-anonymity with all selected categorical variables by a certain importance. The more important a variable, the lower the amount of suppressions it suffers. If the importance for a variable is set to 1, no additional suppressions are carried out for this variable.

3.6 Numerical Methods

The proposed GUI provides two functions (`addNoise()` for adding noise to variables and `microaggregation()`) for perturbation and aggregation of numerical variables, but also functionality to measure the disclosure risk [see, e.g., 11]. By clicking on the `Add noise` and the `Microaggregation` button (see Figure 2), the information given in Figure 9 pops up.

Both functions supply different methods and it is up to the user to select which variables he wants to modify.

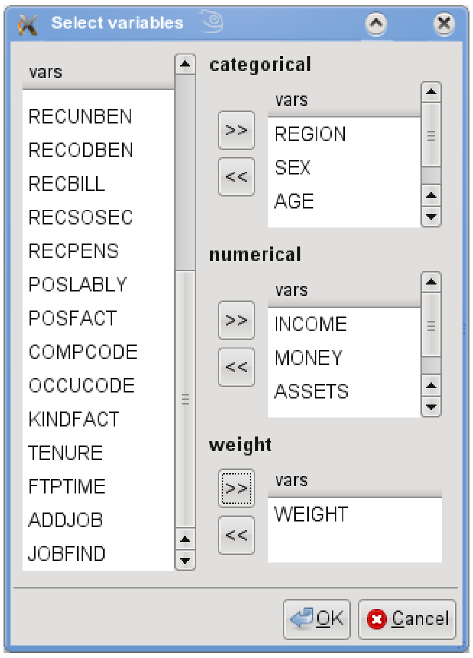
3.7 Script Window

This is one of the major improvements of the `sdcMicro` GUI in comparison to μ-Argus, because every action the user performed in the GUI (with parameters) including code is recorded and listed in the *code window* (see 3.5.1). To access the window, the user just has to navigate to it through the menu. One has also the option to save the actual status or to load an old script to reproduce output or modify some steps or alter the output. Another possibility is that the user can delete single steps from the script or execute it to a certain point to start his work from there.

4 Details About the Implementation

4.1 Software

`gWidgetsRGtk2` [14] is used to generate the GUI. This package allows the gWidgets API to use the `RGtk2` package allowing the use of the `GTK` libraries within R. It provides a lot of useful templates for dialogs. `GTK` is written in `C` and is a highly usable, feature-rich toolkit for creating graphical user interfaces in cross platform compatibility manner. The `Gtk2` libraries usually come with a standard Linux installation, contrary to Windows. This shortcoming for the Windows user, however, is not a serious problem, because the `gWidgetsRGtk2` R package for Windows comes with a very easy-to-use built-in install routine if `Gtk2` is not installed. After installation, R must be restarted once.



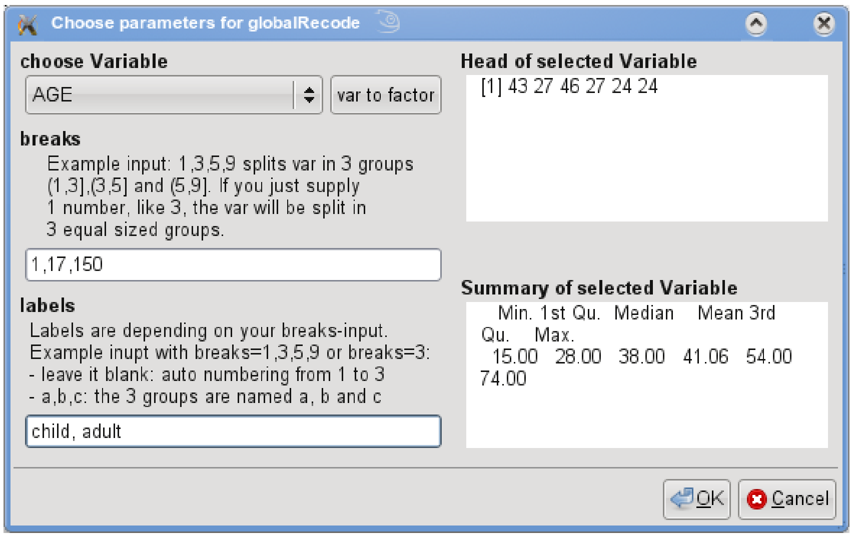
vars: RECUNBEN, RECODBEN, RECBILL, RECSOSEC, RECPENS, POSLABLY, POSFACT, COMPCODE, OCCUCODE, KINDFACT, TENURE, FTPTIME, ADDJOB, JOBFIND

categorical — vars: REGION, SEX, AGE

numerical — vars: INCOME, MONEY, ASSETS

weight — vars: WEIGHT

Figure 4: Variable selection window



choose Variable: AGE

breaks

Example input: 1,3,5,9 splits var in 3 groups (1,3],(3,5] and (5,9]. If you just supply 1 number, like 3, the var will be split in 3 equal sized groups.

1,17,150

labels

Labels are depending on your breaks-input.
Example inupt with breaks=1,3,5,9 or breaks=3:
- leave it blank: auto numbering from 1 to 3
- a,b,c: the 3 groups are named a, b and c

child, adult

Head of selected Variable

```
[1] 43 27 46 27 24 24
```

Summary of selected Variable

```
   Min. 1st Qu.  Median    Mean 3rd
Qu.    Max.
  15.00   28.00   38.00   41.06   54.00
74.00
```

Figure 5: Global recode window. Information of variable AGE is presented in the two boxes on the right. Modifications can be done in the left boxes whereas the information on the right side is updated after a recoding.

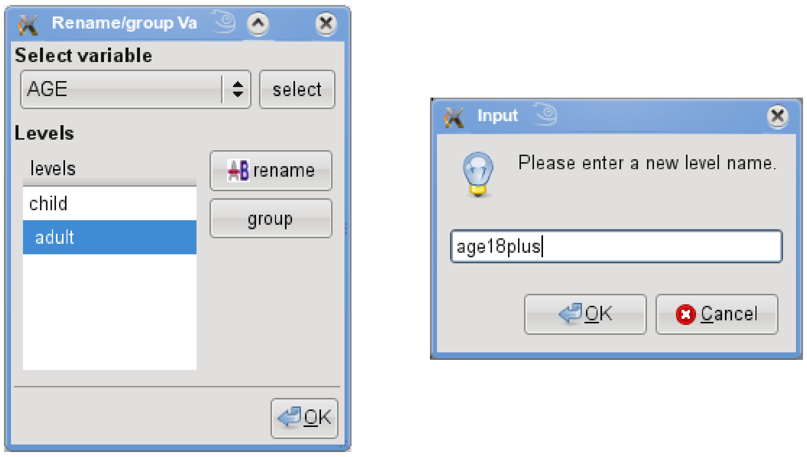


Figure 6: Left: Rename/-group factor window. Levels of categorical variables can be grouped and/or renamed. Right: Renaming after the `rename` button was clicked.

The use of the `Gtk` tools is advantageous since other possible packages for GUI generation in R (for example, `rJava` or the `tcltk` package) come with less pre-written functionality and their documentation is limited.

4.2 Internal Object Handling

The function `sdcGUI`, which generates the GUI, creates its own environment when the GUI is loading. All variables and parameters needed are saved in that environment, and therefore, the global environment of R is not flooded with internal objects from the GUI. As Figure 11 shows, `sdcGUI()` provides three functions (*existd()*, *getd()* and *putd()*) which access the GUI environment `sdcGUIenv`. If an operation is applied to the data, e.g. *addNoise*, it calls *getd()* with the appropriate argument of the needed stored variable to retrieve it. When the calculation is finished, the function *putd()* writes the result back to the environment.

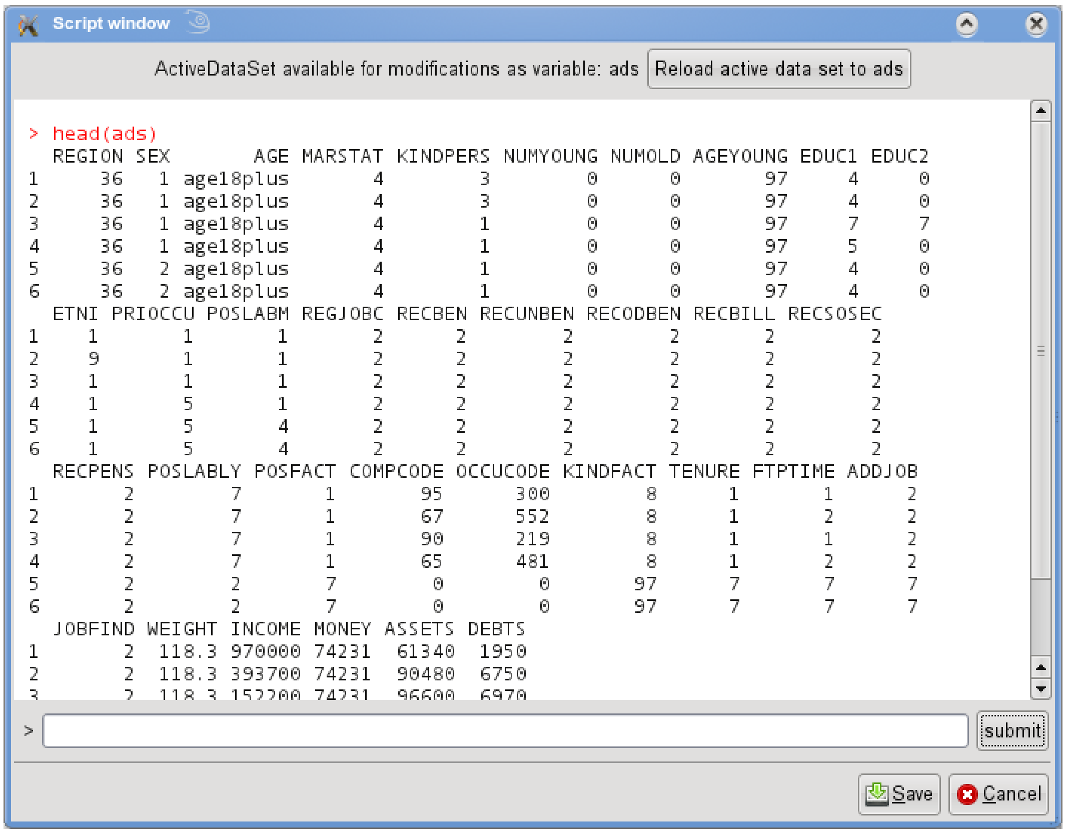
Important internal variables that can be accessed by *getd()* and *putd()* are presented in Figure 11.

4.3 Operation Workflow

This subsection describes the workflow based on the `sdcMicro` function *localSupp()* (used for local suppression). This pattern is applicable for all functions for categorical and numerical variables.

4.3.1 Workflow for Implemented `sdcMicro` Functions

Based on the GUI every modification is applied to the dataset initialized by clicking on a button. Assuming a user wants to perform local suppression on the loaded data set, he must click on the *Local suppression* button. A function (`ls1()`, see Figure 12) is called which asks for the parameters via a popup window (see Figure 8), validates and passes them on to a worker function. This worker function (Figure 12, `localSupp_tmp()`) manages


Script window
ActiveDataSet available for modifications as variable: ads
Reload active data set to ads
submit
Save
Cancel

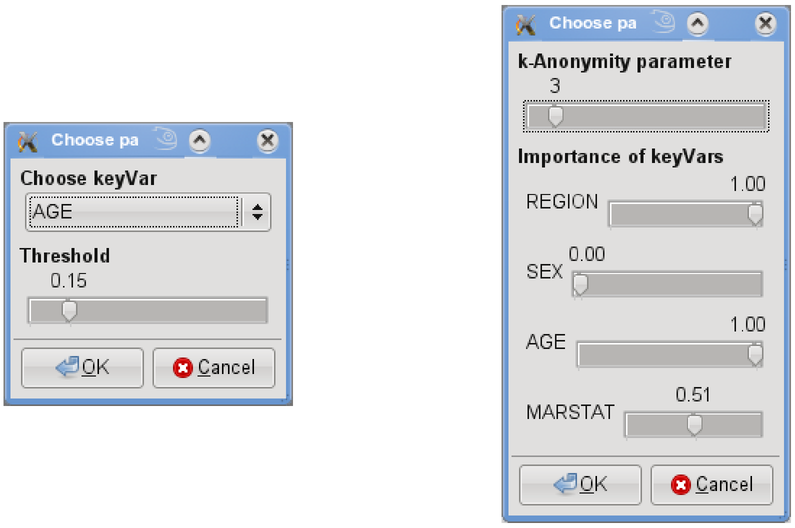
```
> head(ads)
  REGION SEX       AGE MARSTAT KINDPERS NUMYOUNG NUMOLD AGEYOUNG EDUC1 EDUC2
1     36   1 age18plus       4        3        0      0       97     4     0
2     36   1 age18plus       4        3        0      0       97     4     0
3     36   1 age18plus       4        1        0      0       97     7     7
4     36   1 age18plus       4        1        0      0       97     5     0
5     36   2 age18plus       4        1        0      0       97     4     0
6     36   2 age18plus       4        1        0      0       97     4     0
  ETNI PRIOCCU POSLABM REGJOBC RECBEN RECUNBEN RECODBEN RECBILL RECSOSEC
1    1       1       1       2      2        2        2       2        2
2    9       1       1       2      2        2        2       2        2
3    1       1       1       2      2        2        2       2        2
4    1       5       1       2      2        2        2       2        2
5    1       5       4       2      2        2        2       2        2
6    1       5       4       2      2        2        2       2        2
  RECPENS POSLABLY POSFACT COMPCODE OCCUCODE KINDFACT TENURE FTPTIME ADDJOB
1       2        7       1       95      300        8      1       1      2
2       2        7       1       67      552        8      1       2      2
3       2        7       1       90      219        8      1       1      2
4       2        7       1       65      481        8      1       2      2
5       2        2       7        0        0       97      7       7      7
6       2        2       7        0        0       97      7       7      7
  JOBFIND WEIGHT INCOME MONEY ASSETS DEBTS
1       2  118.3 970000 74231  61340  1950
2       2  118.3 393700 74231  90480  6750
3       2  118.3 152200 74231  96600  6970
>
```

Figure 7: Code window which supports the usual R command line interface in an specific R environment.

all further operations. It assures that the operation name as well as all given parameters are saved by calling `script.add()` (for a detailed description of the functionality of the script see Section 4.4). After passing all the information needed to `script.add()` the actual `sdcMicro` function is called in order to modify the data (Figure 12, *localSupp()*) and the return values are gathered by the worker function. When a `sdcMicro` function returns the processed data, the `activeDataSet` environment variable needs to be updated. This action is done by an update function (Figure 12, `updateActiveDataSet()`) which furthermore initializes the recalculation of the frequencies and the individual risk (Figure 12, `freqCalcIndivRisk()`). This function also updates all the displayed results in the main window.

4.3.2 Workflow for Utility Functions

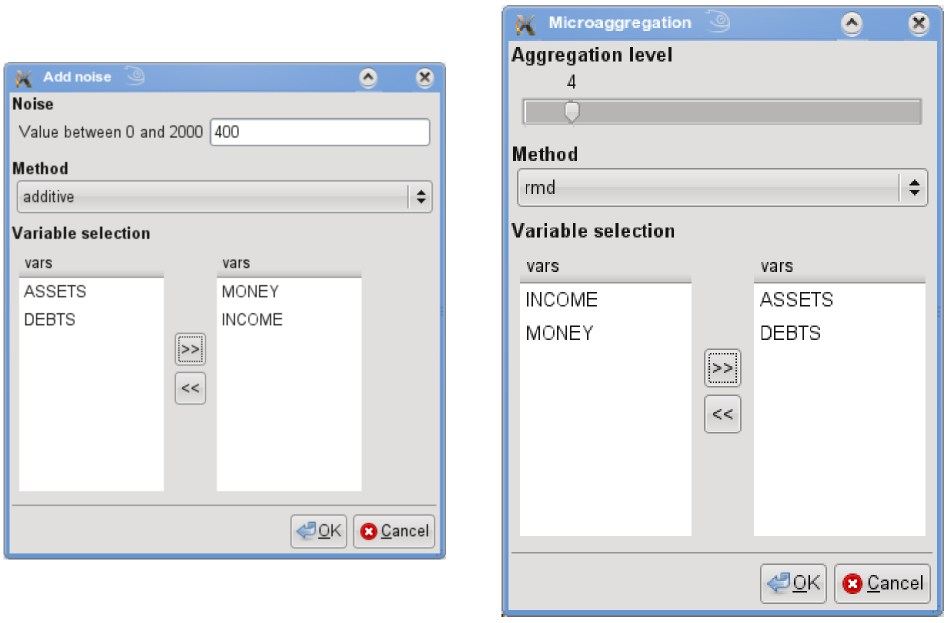
The operation workflow for utility functions is basically the same as for the implemented `sdcMicro` functions with the exception of the `sdcMicro` function call and the collection of the result done by the worker function (Figure 12, *localSupp_tmp()* without doing operations 3. and 4.). In this case the worker function fulfills all the operations on the dataset.



Choose keyVar
AGE
Threshold
0.15

k-Anonymity parameter
3
Importance of keyVars
REGION 1.00
SEX 0.00
AGE 1.00
MARSTAT 0.51

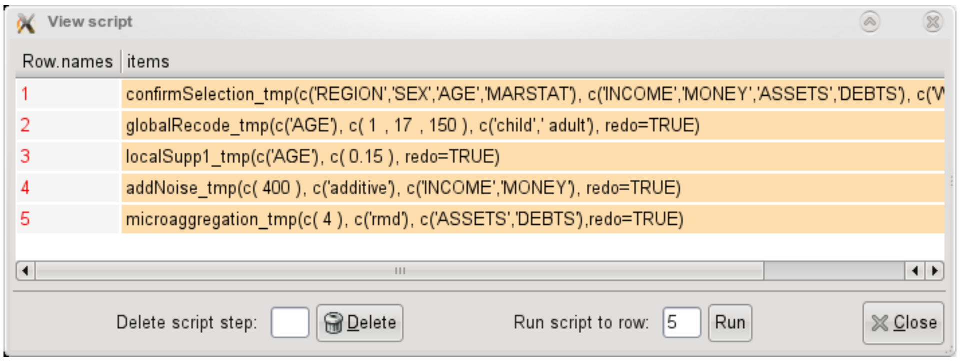
Figure 8: Local suppression (left) and Local suppression 2 wrapper (right) window. The higher the importance of a key variable, the lower the amount of suppression in the variable.



Noise
Value between 0 and 2000 400
Method
additive
Variable selection
vars: ASSETS, DEBTS
vars: MONEY, INCOME

Aggregation level
4
Method
rmd
Variable selection
vars: INCOME, MONEY
vars: ASSETS, DEBTS

Figure 9: Add noise (left) and microaggregation (right) window for the anonymization of continuous scaled variables.



View script

Row.names	items
1	confirmSelection_tmp(c('REGION','SEX','AGE','MARSTAT'), c('INCOME','MONEY','ASSETS','DEBTS'), c('V
2	globalRecode_tmp(c('AGE'), c(1 , 17 , 150), c('child',' adult'), redo=TRUE)
3	localSupp1_tmp(c('AGE'), c(0.15), redo=TRUE)
4	addNoise_tmp(c(400), c('additive'), c('INCOME','MONEY'), redo=TRUE)
5	microaggregation_tmp(c(4), c('rmd'), c('ASSETS','DEBTS'),redo=TRUE)

Delete script step: [ ] Delete   Run script to row: 5 Run   Close

Figure 10: Script window which allows to delete, run or modify any steps.

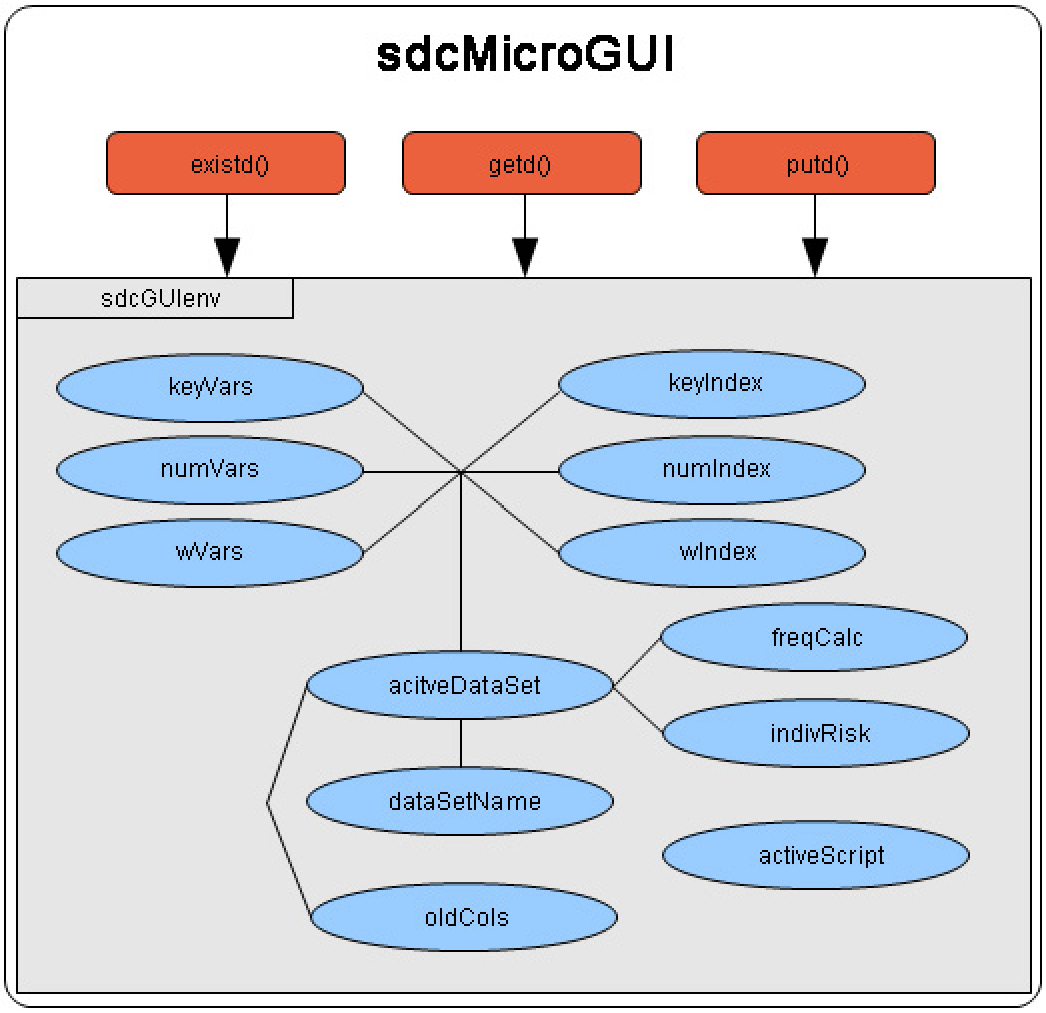


Figure 11: Internal object handling

Table 1: `sdcGUIenv` environment variables

Name	Description
activeDataSet:	The dataset chosen for anonymization
dataSetName:	Name of the loaded dataset
oldCols:	If an operations is performed on the dataset, the last version is stored for risk estimation
freqCalc:	The result of the latest frequency calculation is saved because it is needed for some sdcMicro functions
indivRisk:	The result of the latest individual risk estimation is also saved because it is needed for some sdcMicro functions
keyVars:	Variable names of selected categorical variables
keyIndex:	Column index of selected categorical variables
numVars:	Variable names of selected numerical variables
numIndex:	Column index of selected numerical variables
wVars:	Variable names of selected weight variable
wIndex:	Column index of selected weight variable
activScript:	All used functions are saved in this variable

4.4 Script Functionality and GUI Enhancement

This section shows how to include a new function and how to include a new method into an existing class.

The script below simulates a user-made function call. Therefore, all modifications of the dataset beginning with the selection of categorical and numerical variables as well as the weight variable must be stored. As shown in Figure 12, for the selection and validation of the parameters the worker function `localSupp_tmp()` acts as the main part of the workflow. To illustrate the record process take a look at the following code snippet from `localSupp1_tmp()` (comments are given after the # sign):

```
localSupp1_tmp <- function(keyVar, threshold, redo=FALSE) {
  if( !redo ) {
    Script.add(paste("localSupp1_tmp(", parseVarStr(keyVar), ", ",
      parseVar(threshold), ", redo=TRUE)", sep=""))
        # stores the function call for later re-use
  }
  x <- getd("freqCalc")
    # get frequency calculation of active data set
  keyVar <- getIndex(keyVar)
    # get index of categorical variables
  indivRisk <- getd("indivRisk")$rk
    # get the needed value from the individual risk estimation
  l1 <- localSupp(x, keyVar, indivRisk, threshold)
    # execute the sdcMicro function and get result
  updateActiveDataSet(l1$freqCalc)
    # update active data set
}
```

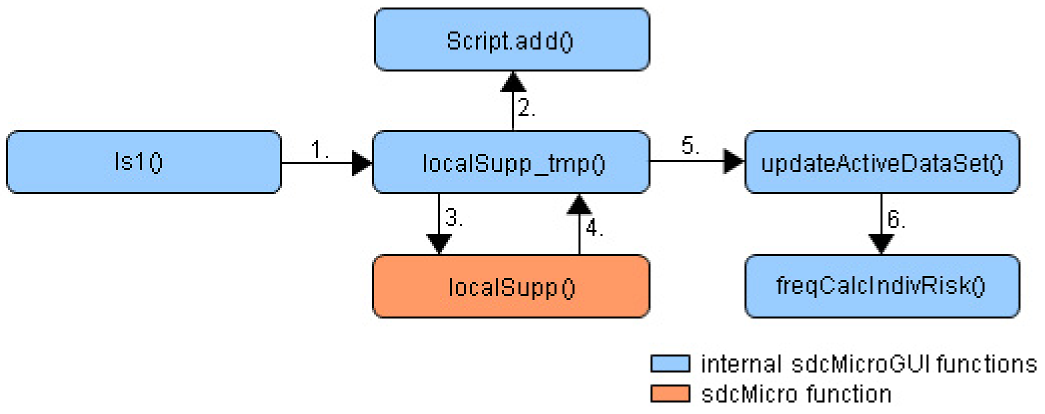


Figure 12: Operation workflow for implemented `sdcMicro` functions

The *redo* value is needed for every worker function and is usually set to `FALSE` to store the function call. Nevertheless, often one part of the GUI is made in an iterative manner. For such situations usually `redo` is set to `FALSE` for computational efficiency reasons (the code window functionality serves as a typical example, see also Section 4.5).

The script is stored in a list in the `sdcGUIenv` environment which can be viewed via the menu (see Figure 3). An option to save the script is provided as well. It is also possible to load the script from a file to reproduce output or re-run the script on a dataset for which more observations or variables are included.

Function `ls1` (see Figure 12) includes commands from the `gWidgetsRGtk2` package to include the `sdcMicro` function `localSupp` in the GUI.

In order to give an impression how to enhance the graphical user interface by an additional method that uses a pre-defined class structure, a simple example is given in the following.

Suppose the aim is to include the popular microaggregation method MDAV ([3]) and suppose that the code of this method is already included in function `microaggregation()`[2]. The corresponding code snippet might look like as follows:

```
if (method == "mdav") {
    #----
    body
    #----
    res <- list(..., method = 'mdav',
        aggr = aggr, blowxm = blowxm, ...)
  }
```

The calculation is done in the section denoted `body`. The output must be of class `micro`, i.e. the method, the aggregation level and the microaggregation results must be provided using pre-defined parameter names (`method`, `aggr`, `blowxm`). Others parameters might be passed trough (with the `...` argument).

[2]Also a new function can be written but, for simplicity, the output should be of class `micro`.

Look for the following line in function `sdcGUI.R`[3], which indicates that a window for microaggregation is generated in the GUI.

```
nm2_window = gwindow("Microaggregation", width=230, parent=window)
```

To include MDAV in the GUI only a string with the name of the procedure (`"mdav"`) must be included in function `sdcGUI.R`. Adding the string `"mdav"` to the corresponding list (in `sdcGUI.R`)

```
methodSel = gdroplist(c("rmd", "pca", "clustppca",
                         "influence", "mdav"))
```

enables the use of MDAV within the GUI.

Conceptual modifications of the GUI are possible with knowledge of the software environment R and the `gWidgetsRGtk2` R package.

4.5 Code Window

To use the `Code window` the user needs at least basic R knowledge to be able to work with it. This functionality was implemented because experienced users may want to have additional flexibility such as the standard R command-line interface version provides. The concept of `sdcGUI` is that all variables which are needed to operate are stored in a separate environment so that the global R environment stays "clean". Assuming an experienced user works with the GUI and wants to manipulate the dataset by hand he would have to export the dataset to the global environment, modify it there and open it again to continue to work with it in the GUI. Basically, that would work, but the actions done while using the CLI version of R will not get recorded, so the output would not be reproducible. This problem is solved by the `Code window` (see Figure 7).

By opening the `Code window` a new environment is created and the `activeDataSet` is made available in there under the shortcut `ads`. A simple input line and an output box equivalent to the standard R CLI is provided.

Every user's command is evaluated and sent to R for processing and the output is captured. The dataset in the `sdcGUIenv` environment is not changed because it is changed in the code window environment and the executed command is stored in a separate list, until the `Save` button of the window is pressed. Whenever this happens, the modified dataset in the code window environment is stored in the `sdcGUIenv` environment and the saved commands copied to the *activeScript* variable.

A sample input in the *Code window* to alter the following dataset

```
> ads
  Num1 Key1 Num2 Key2 Key3 Key4 Num3     w
1 0.30    1 0.40    2    5    1    4  18.0
2 0.12    1 0.22    2    1   14   22  45.5
3 0.18    1 0.80    2    1    1    8  39.0
4 1.90    3 9.00    3    1    5   91  17.0
5 1.00    4 1.30    3    1    4   13 541.0
6 1.00    4 1.40    3    1    1   14   8.0
7 0.10    6 0.01    2    1   23    1   5.0
8 0.15    1 0.50    2    5    1    5  92.0
```

[3]This function can be found in the the `/R` directory of the installation path of the package.

could look like this:

```
ads[2,4] <- 5
ads[,6] <- ifelse(ads[,6]>10,10,ads[,6])
```

The resulting dataset of the *Code window* is

```
> ads
  Num1 Key1 Num2 Key2 Key3 Key4 Num3     w
1 0.30    1 0.40    2    5    1    4  18.0
2 0.12    1 0.22    5    1   10   22  45.5
3 0.18    1 0.80    2    1    1    8  39.0
4 1.90    3 9.00    3    1    5   91  17.0
5 1.00    4 1.30    3    1    4   13 541.0
6 1.00    4 1.40    3    1    1   14   8.0
7 0.10    6 0.01    2    1   10    1   5.0
8 0.15    1 0.50    2    5    1    5  92.0
```

The added entries in the *activeScript* variable look as follows:

```
[1] "confirmSelection_tmp(c('Key1','Key2','Key3','Key4'), c('Num3'
          ,'Num2','Num1'), c('w'), redo=TRUE)"
[2] "localSupp3_tmp(c( 2 ), c( 0.7 , 0.9 , 0.4 , 0.9 ), redo=TRUE)"
[3] "localSupp1_tmp(c('Key1'), c( 0.15 ), redo=TRUE)"
[4] "ads <- activeDataSet()"
[5] "ads[2,4] <- 5"
[6] "ads[,6] <- ifelse(ads[,6]>10,10,ads[,6])"
[7] "updateActiveDataSet(ads)"
```

Usually the script just needs the commands previously used, with the `redo` parameter set to `TRUE`, because every predefined function automatically operates on the `activeDataSet`. If the `Code window` is used, every command executed uses the temporary code window dataset. Therefore line 4 simulates the generation of the temporary dataset that the following commands can be run on and line 7 updates the actual dataset `activeDataSet` in the `sdcGUIenv` environment.

5 Conclusion

This GUI provides an extension to the package `sdcMicro` [10]. The developed GUI makes `sdcMicro` accessible to a wider range of users including those not familiar with the R command-line interface. The user can access all basic functions for microdata protection by using this GUI.

Flexibility is provided by displaying the main results (e.g., the summary of frequency counts and the estimated disclosure risk), which are updated after a user interaction automatically. Additional flexibility is provided by storing all the user operations with all parameters in a script which can then be saved, modified and/or reloaded. Thus, full reproducibility is provided also when using this GUI instead of the CLI version.

It is also possible to extend the GUI whenever new functions in the core `sdcMicro` package are included. With the help of the detailed description of internal object handling in Section 4 it is possible that other researchers and users modify and enhance this GUI. However, basic knowledge of both R and the `gWidgetsRGtk2` R package is required.

References

[1] N. Anwar. Micro-aggregation - the small aggregates method. In *Internal report*. Luxembourg: Eurostat, 1993.

[2] R. Brand. Microdata protection through noise addition. In *Privacy in Statistical Databases. Lecture Notes in Computer Science. Springer*, pages 347–359, 2004.

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

[4] L. Franconi and S. Polettini. Individual risk estimation in μ-ARGUS: a review. In *Privacy in Statistical Databases. Lecture Notes in Computer Science. Springer*, pages 262–272, 2004.

[5] A. Hundepool, A. Van deWetering, Ramaswamy R., L. Franconi, A. Capobianchi, P-P. DeWolf, J. Domingo-Ferrer, V. Torra, R. Brand, and S. Giessing. μ-ARGUS version 4.2 software and users manual, 2008. ⟨`http://neon.vb.cbs.nl/casc`⟩.

[6] B. Meindl and M. Templ. The anonymisation of the CVTS2 and income tax dataset. an approach using R-package sdcMicro. In *to appear in: Joint UNECE/Eurostat Work Session on Statistical Data Confidentiality. Monographs of Official Statistics.*, 2007.

[7] P. Samarati and L. Sweeney. Protecting privacy when disclosing information: k-anonymity and its enforcement through generalization and suppression. Technical report, SRI Intl. Tech. Rep., 1998.

[8] M. Templ. *Privacy in Statistical Databases, Lecture Notes in Computer Science, Vol. 4302*, chapter Software Development for SDC in R, pages 347 – 359. Springer, Berlin Heidelberg, 2006. ISBN: 978-3-540-49330-3.

[9] M. Templ. Statistical disclosure control for microdata using the R-package sdcMicro. *Transactions on Data Privacy*, 1(2):67–85, 2008.

[10] M. Templ. sdcmicro: Statistical disclosure control methods for the generation of public- and scientific-use files. version 2.6.0. software and users manual. published online, 2009. ⟨`http://cran.r-project.org/web/packages/sdcMicro/index.html`⟩.

[11] M. Templ and B. Meindl. Robust statistics meets SDC: New disclosure risk measures for continuous microdata masking. *Privacy in Statistical Databases. Lecture Notes in Computer Science. Springer*, 5262:113–126, 2008. ISBN 978-3-540-87470-6, DOI 10.1007/978-3-540-87471-3_10.

[12] M. Templ and B. Meindl. Robustification of microdata masking methods and the comparison with existing methods. *Privacy in Statistical Databases. Lecture Notes in Computer Science. Springer*, 5262:177–189, 2008. ISBN 978-3-540-87470-6, DOI 10.1007/978-3-540-87471-3_15.

[13] D. Ting, S. Fienberg, and M. Trottini. Romm methodology for microdata release. In *Monographs of official statistics, Work session on statistical data confidentiality*. Eurostat, Luxembourg, 2005.

[14] J. Verzani and M. Lawrence. gWidgetsRGtk2: Toolkit implementation of gWidgets for RGtk2. published online, 2009. ⟨`http://cran.r-project.org/web/packages/gWidgetsRGtk2/index.html`⟩.