# synthpop : Bespoke creation of synthetic data in R

**Beata Nowok**
University of Edinburgh

**Gillian M Raab**
University of Edinburgh

**Chris Dibben**
University of Edinburgh

**Abstract**

In many contexts, confidentiality constraints severely restrict access to unique and valuable microdata. Synthetic data which mimics the real data and preserves the relationships between variables but do not contain any disclosive records is one possible solution to this problem. The **synthpop** package, introduced in this paper, provides routines to generate a synthetic version of real data sets. We describe the methodology and its consequences for the data characteristics. We illustrate the package features using a real data example.

*Keywords*: synthetic data, disclosure control, CART, R.

## 1. Introduction and background

### 1.1. Synthetic data for disclosure control

National statistical agencies and other institutions gather large amounts of information about individuals and organisations. Such data can be used to understand population processes so as to inform policy and planning. The cost of such data can be considerable, both for the collectors and the subjects who provide their data. Because of confidentiality constraints and guarantees issued to data subjects the full access to such data is often restricted to the staff of the collection agencies. Traditionally, data collectors have used anonymization along with simple perturbation methods such as aggregation, recoding, record-swapping, suppression of sensitive values or adding random noise to prevent the identification of data subjects. Advances in computer technology have shown that such measures may not prevent disclosure Ohm (2010) and in addition they may compromise the conclusions one can draw from such data (Elliot and Purdam (2007), Winkler (2007)).

In response to these limitations there have been several initiatives, most of them centred around the U.S. Census Bureau, to generate synthetic data which can be released to users outside the setting where the original data are held. The basic idea of the synthetic data is to replace some or all of the real values by sampling from appropriate probability distributions so that the essential statistical features of the original data are preserved. The monograph by Drechsler (2011) summarises some of the theoretical and policy developments. The approach has been developed along similar lines to recent practical experience with multiple imputation methods. The data collection agency generates multiple synthetic data sets and inferences are obtained by combining the results of models fitted to each of them. The methods are detailed in Raghunathan, Reiter, and Rubin (2003) and have been further discussed and exemplified in a series of papers (Reiter (2002), Reiter (2005a), Reiter (2005b), Caiola and Reiter (2010) and Dreschler and Reiter (2010) among others).

The original aim of these methods have been to provide publicly available data sets that can be used for inference in place of the real data. However, such inferences will only be valid if the model used to construct the synthetic data is the true mechanism that has generated the real data. At best, this is an approximation and published results comparing estimates from real and synthetic data have been mixed.

Our aim in writing the **synthpop** package is a more modest one of providing test data for users of confidential data sets. These test data should resemble the actual data as closely as possible, but would never be used in any final analyses. The users carry out exploratory analyses and test models on the synthetic data, but they, or perhaps staff of the data collection agencies, would use the code developed on the synthetic data to run their final analyses on the real data. This approach recognises the limitations of synthetic data produced by these methods. It is interesting to note that a similar approach is currently being used for both of the synthetic products made available by the U.S. Census Bureau[1], where results obtained from the synthetic data are validated on the real data ("gold standard files").

## 1.2. Motivation for the development of synthpop

The England and Wales Longitudinal Study (ONS LS), the Scottish Longitudinal Study (SLS) and the Northern Ireland Longitudinal Study (NILS) are rich micro-datasets linking samples from the national Census in each country to administrative data (births, deaths, marriages, cancer registrations and other sources) for individuals and their immediate families across several decades. Whilst unique and valuable resources, the sensitive nature of the information they contain means that access to the microdata is restricted to approved researchers and longitudinal study (LS) support staff, who can only view and work with the data in safe settings controlled by the national statistical agencies. Consequently, compared to other census data products such as the aggregate statistics or samples of anonymised records, the three longitudinal studies (LSs) are used by a small number of researchers, a situation which limits their potential impact. Given that confidentiality constraints and legal restrictions mean that open access is not possible with the real microdata, alternative options are needed to allow academics and other users to carry out their research more freely. To address this the SYLLS (Synthetic Data Estimation for UK Longitudinal Studies) project[2] has been funded by the Economic and Social Research Council to develop techniques to produce synthetic data which mimics the real data and preserves the relationships between variables and transitions of individuals over time, but can be made available to accredited researchers to analyse on their own computers. The **synthpop** package for R has been written as part of the SYLLS project to allow LS support staff to produce synthetic data for users of the LSs, that are tailored to the needs of each individual user.

We will use the term "synthesiser" for someone who is producing the synthetic data from the real data and hence has access to both. The term "analyst" will refer to someone who has no access to the real data and will be using the synthetic data for exploratory analyses. After the exploratory analysis the analyst will develop confirmatory models and can send the syntax

[1] see `http://www.census.gov/programs-surveys/sipp/methodology/sipp-synthetic-beta-data-product.html` and `https://www.census.gov/ces/dataproducts/synlbd/`

[2] see `http://www.lscs.ac.uk/projects/synthetic-data-estimation-for-uk-longitudinal-studies/`

to a synthesiser to carry out the gold standard analyses. As well as providing routines to generate the synthetic data the **synthpop** package contains routines that can be used by the analyst to summarise synthetic data and fitted models from synthetic data and those that can be used by the synthesiser to compare gold standard analyses with those from the synthetic data. Although primarily targeted to the data from the LSs, the package is written in a form that should make it applicable to other confidential data where the resource of synthetic data would be valuable. It allows the user to extend the types of data that can be handled by writing their own routines that can be added to the package.

### 1.3. Structure of this paper

The structure of this paper is as follows. The next section introduces the notation, terminology and the main theoretical results needed for the simplest and, we expect, the most common use of the package. More details of the theoretical results for the general case can be found in Raab, Nowok, and Dibben (2014). Readers not interested in the theoretical details can now proceed directly to Section 3 which presents the package and its basic functionality. Section 4 that follows provides some illustrative examples. The concluding Section 5 indicates directions for future developments.

## 2. Overview of method

Real data from a survey or a sample from a census are available to the synthesiser. They consist of a sample of $n$ units consisting of $(x_{obs}, y_{obs})$ where $x_{obs}$, which may be null, is a matrix of data that can be released unchanged to the analyst and $y_{obs}$ is an $n$ x $p$ matrix of $p$ variables that require to be synthesised. We consider here the simple case when the synthetic data sets (syntheses) will each have the same number of records as the original data and the method of generating the synthetic sample (e.g. simple random sampling or a complex sample design) matches that of the real data.

### 2.1. Generating synthetic data

The observed data are assumed to be a sample from a population with parameters that can be estimated by the synthesiser, specifically $y_{obs}$ is assumed to be a sample from $f(y|x_{obs}, \theta)$ where $\theta$ is a vector of parameters. This could be a hypothetical infinite super-population or a finite population which is large enough for finite population corrections to be ignored. The synthesiser fits the data to the assumed distribution and obtains estimates of its parameters. In most implementations of synthetic data generation, including **synthpop**, the joint distribution is defined in terms of a series of conditional distributions. A column of $y_{obs}$ is selected and the distribution of this variable, conditional on $x_{obs}$ is estimated. Then the next column is selected and its distribution is estimated conditional on $x_{obs}$ and the column of $y_{obs}$ already selected. The distribution of subsequent columns of $y_{obs}$ are estimated conditional on $x_{obs}$ and all previous columns of $y_{obs}$.

The generation of the synthetic data sets can proceed in parallel to the fitting of each conditional distribution. Each column of the synthetic data is generated from the assumed distribution, conditional on $x_{obs}$, the fitted parameters of the conditional distribution (simple synthesis) and the synthesised values of all the previous columns of $y_{obs}$. Alternatively the synthetic values can be generated from the posterior distribution of the parameters (proper synthesis). In both cases, a total of $m$ synthetic data sets are generated.

### 2.2. Inference from the synthetic data

If the analyst wants to estimate a model from the synthetic data, she will fit the model to each of the $m$ synthetic data sets and obtain an estimate of its vector of parameters $\beta$ from each synthetic data set as $(\hat{\beta}_1, \cdots, \hat{\beta}_i, \cdots, \hat{\beta}_m)$. If the model for the data is correct the $m$ estimates from the synthetic data will be centred around the estimate $\hat{\beta}$ that would have been obtained from the observed data. We are assuming that it is the goal of the analyst to use the synthetic data to estimate $\hat{\beta}$ and its estimated variance-covariance matrix $V_{\hat{\beta}}$. The mean of $m$ synthetic estimates, $\bar{\hat{\beta}} = \sum \hat{\beta}_i / m$ provides an unbiased estimate of $\hat{\beta}$. Provided the observed and synthetic data are generated by the same sampling scheme then $V_{\bar{\hat{\beta}}} = \sum V_{\hat{\beta}_i} / m$ will be an unbiased estimate of $V_{\hat{\beta}}$. The variance-covariance matrix of $\bar{\hat{\beta}}$, conditional on $\hat{\beta}$ and $V_{\hat{\beta}}$ becomes $V_{\hat{\beta}}/m$ which can be estimated from $V_{\bar{\hat{\beta}}}/m$. Thus the stochastic error in the mean of the synthetic estimates about the values from the observed data can be reduced to a negligible quantity by increasing $m$. It must be remembered, however that the unbiasedness of $\bar{\hat{\beta}}$ only applies when observed data are a sample from the distribution used for synthesis. In practical applications differences between the analyses on the observed data and those from the mean of the syntheses will be found because the data do not conform to the model used for synthesis. Such differences will not be reduced by increasing $m$. The synthesiser, with access to the observed data, can estimate $\bar{\hat{\beta}} - \hat{\beta}$ and compare it to its standard error in order to judge the extent that this model mismatch affects the estimates.

Note that this result is different from most other papers which aim to use the results of the synthetic data to make inference about the population from which the real gold standard data have been generated. But our aim, in the simplest case we describe above, is only to make inferences to the results that would have been obtained by the gold standard analysis, with the expectation that the analyst will run final models on the real data. Also, unlike most of the literature above, in the simplest case we do not sample from the predictive distribution of the parameters to create the synthetic data but an option to do so is available in **synthpop**. This approach has been proposed recently by Reiter and Kinney (2012) for partially synthetic data. The justification for this approach for fully synthetic data is in Raab *et al.* (2014) along with the details of how the **synthpop** package can be used to make inferences to the population.

## 3. The synthpop package in practice

### 3.1. Obtaining the software

The **synthpop** package is an add-on package to the statistical software R. It is freely available from the Comprehensive R Archive Network at `http://CRAN.R-project.org/package=synthpop`. It requires a non-standard package **party** (Hothorn, Hornik, and Zeileis (2006)) which needs to be installed prior to loading package **synthpop** from a library. The namespace of package **party** is loaded automatically when package **synthpop** is loaded.

The **synthpop** package utilises the structure and some functions of the **mice** multiple imputation package (van Buren and Groothuis-Oudshoorn (2011)) but adopts and extends it for the specific purpose of generating synthetic data.

### 3.2. Basic functionality

The **synthpop** package aims to provide a user with an easy way of generating synthetic versions of real data sets. Via the function `syn()` a synthetic data set is produced using a single command. The only required argument is `data` which is a data frame or a matrix containing the data to be synthesised. By default, a single synthetic data set is produced using simple synthesis. Multiple data sets can be obtained by setting parameter `m` to a desired number and proper synthesis is conducted when argument `proper` is set to `TRUE`. Data synthesis can be further customized with other optional parameters. Below, we only present the salient features of the `syn()` function. See examples in Section 4 and the R documentation for the function `syn()` for more details (command `?syn` at the R console).

*Choice of synthesising method*
The synthesising models are defined by a parameter `method` which can be a single string or a vector of strings. Providing a single method name assumes the same synthesising method for each variable, unless a variable's data type precludes it. Note that a variable to be synthesised first that has no predictors is a special case and its synthetic values are generated by random sampling with replacement from the original data (`"sample"` method). In general, a user can choose between parametric and non-parametric methods. The latter are based on classification and regression trees (CART) that can handle any type of data. By default the `ctree` implementation of the CART technique is used for all variables to be synthesised. Setting the parameter `method` to `"parametric"` assigns default parametric methods to variables to be synthesised based on their types. The default parametric methods for numeric, binary, unordered factor and ordered factor data type are specified in vector `defaultMethod` which may be customised if desired. Alternatively a method can be chosen out of the available methods for each variable separately. The methods currently implemented are listed in Table 1. A new synthesising method can be easily introduced by writing a function named `syn.newmethod()` and then specifying `method` parameter of `syn()` function as `"newmethod"`.

*Controlling the predictions*
The synthetic values of the variables are generated sequentially using variables already synthesised and conditional distributions fitted to the same variables in the real data. Next to synthesising model specification, a user may choose the order in which variables should be synthesised (`visitSequence` parameter) and also the set of variables to include as predictors in the synthesising model (`predictorMatrix` parameter). As mentioned above, the choice of explanatory variables is restricted by the synthesis sequence and variables that are not synthesised yet cannot be used in prediction models. There is a possibility, however, to include as predictors variables that do not belong to the data set to be synthesised.

*Handling data with missing or restricted values*
The aim of producing a synthetic version of a real data here is to mimic their characteristics in all possible ways, which may include missing and restricted values data. Values representing missing data in categorical variables are treated as additional categories and reproducing

| Method | Description | Data type |
|---|---|---|
| *Non-parametric* | | |
| `ctree, cart` | Classification and regression trees | any |
| `surv.ctree` | Classification and regression trees | duration |
| *Parametric* | | |
| `norm` | Normal linear regression | numeric |
| `normrank`* | Normal linear regression preserving the marginal distribution | numeric |
| `logreg`* | Logistic regression | binary |
| `polyreg`* | Polytomous logistic regression | factor, >2 levels |
| `polr`* | Ordered polytomous logistic regression | ordered factor, >2 levels |
| `pmm` | Predictive mean matching | numeric |
| *Other* | | |
| `sample` | Random sample from the observed data | any |
| `passive` | Function of other synthesised data | any |

Table 1: Built-in synthesising methods. * indicates default parametric methods

them is straightforward. Continuous variables with missing data are modelled in two steps. In the first step, we synthesise an auxiliary binary variable specifying whether a value is missing or not. Depending on the method specified by a user for the original variable a logit or CART model is used for synthesis. If there are different types of missing values an auxiliary categorical variable is created to reflect this and an appropriate model is used for synthesis (a polytomous or CART model). In the second step, a synthesising model is fitted to the non-missing values in the original variable and then used to generate synthetic values for the non-missing category records in our auxiliary variable. The auxiliary variable and a variable with non-missing values and zeros for remaining records are used instead of the original variable for prediction of other variables. The missing data codes have to be specified by a user in `contNA` parameter of the `syn()` function if they differ from the `R` missing data code `NA`.

Restricted values are those where the values for some cases are determined explicitly by those of other variables. In such cases the rules and the corresponding values should be specified using `rules` and `rvalues` parameters. The variables used in `rules` have to be synthesised prior to the variable they refer to. In the synthesis process the restricted values are assigned first and then only the records with not restricted values are synthesised.

## 4. Illustrative examples

### 4.1. Data

The **synthpop** package includes a data frame `SD2011` with individual microdata that will be used for illustration. The data set is a subset of survey data collected in 2011 within the Social Diagnosis project (Council for Social Monitoring (2011)) which aims to investigate objective and subjective quality of life in Poland. The complete data set is freely available at http://www.diagnoza.com/index-en.html along with a detailed documentation. The SD2011 subset contains 35 selected variables of various type for a sample of 5,000 individuals aged 16 and over.

## 4.2. Simple example

To get access to **synthpop** functions and SD2011 data set we need to load the package via

```
> library(synthpop)
```

For our illustrating examples of syn() function we use seven variables of various data types which are listed in Table 2.

| Variable name | Description | Data type |
|---|---|---|
| sex | Sex | binary |
| age | Age | numeric |
| edu | Highest educational qualification | factor, >2 levels |
| marital | Marital status | factor, >2 levels |
| income | Personal monthly net income | numeric |
| ls | Overall life satisfaction | factor, >2 levels |
| wkabint | Plans to go abroad to work in the next two years | factor, >2 levels |

Table 2: Variables to be synthesised.

Although function syn() allows synthesis of a subset of variables (see Section 4.3), for ease of presentation here we extract variables of interest from SD2011 data set and store them in a data frame called rds which stands for 'real data set'. The structure of rds data can be investigated using the head() function which prints the first rows of a data frame.

```
> vars <- c("sex","age","edu","marital","income","ls","wkabint")
> rds <- SD2011[,vars]
> head(rds)
```

```
     sex age                  edu marital income                ls wkabint
1 FEMALE  57   VOCATIONAL/GRAMMAR MARRIED    800           PLEASED      NO
2   MALE  20   VOCATIONAL/GRAMMAR  SINGLE    350 MOSTLY SATISFIED      NO
3 FEMALE  18   VOCATIONAL/GRAMMAR  SINGLE     NA           PLEASED      NO
4 FEMALE  78 PRIMARY/NO EDUCATION WIDOWED    900             MIXED      NO
5 FEMALE  54   VOCATIONAL/GRAMMAR MARRIED   1500 MOSTLY SATISFIED      NO
6   MALE  20            SECONDARY  SINGLE     -8           PLEASED      NO
```

To run a default synthesis only data to be synthesised have to be provided as a function argument. Note that in order to reproduce the results presented below, before running the syn() function the pseudo random number generator seed has to be fixed

```
> set.seed(17914709)
> sds.default <- syn(rds)
```

```
syn  variables
1    sex age edu marital income ls wkabint
```

The resulting object of class `synds` called here `sds.default`, where `sds` stands for 'synthesised data set', is a list. The `print` method displays its selected components (see below). An element `syn` contains a synthesised data set which can be accessed using a standard list referencing (`sds.default$syn`).

```
> sds.default

Call:
($call) syn(data = rds)

Number of synthesised data sets:
($m)  1

First rows of synthesised data set:
($syn)
     sex age                  edu marital income               ls wkabint
1 FEMALE  65   VOCATIONAL/GRAMMAR MARRIED    610          PLEASED      NO
2 FEMALE  60 PRIMARY/NO EDUCATION MARRIED     NA MOSTLY SATISFIED      NO
3 FEMALE  45   VOCATIONAL/GRAMMAR MARRIED    800 MOSTLY SATISFIED      NO
4 FEMALE  38   VOCATIONAL/GRAMMAR MARRIED   1500          PLEASED      NO
5   MALE  56   VOCATIONAL/GRAMMAR MARRIED   1000          PLEASED      NO
6   MALE  62            SECONDARY MARRIED   2000 MOSTLY SATISFIED      NO
...

Synthesising methods:
($method)
     sex      age      edu  marital   income       ls  wkabint
"sample"  "ctree"  "ctree"  "ctree"  "ctree"  "ctree"  "ctree"

Order of synthesis:
($visitSequence)
    sex     age     edu marital  income      ls wkabint
      1       2       3       4       5       6       7

Matrix of predictors:
($predictorMatrix)
        sex age edu marital income ls wkabint
sex       0   0   0       0      0  0       0
age       1   0   0       0      0  0       0
edu       1   1   0       0      0  0       0
marital   1   1   1       0      0  0       0
income    1   1   1       1      0  0       0
ls        1   1   1       1      1  0       0
wkabint   1   1   1       1      1  1       0
```

The remaining (undisplayed) list elements include other `syn()` function parameters used in the synthesis. Their names can be listed via `names()` function. For a complete description see the `syn()` function help page (`?syn`).

```
> names(sds.default)
```

```
 [1] "call"            "m"               "syn"             "method"
 [5] "visitSequence"   "predictorMatrix" "event"           "smoothing"
 [9] "denom"           "minbucket"       "proper"          "n"
[13] "k"               "rules"           "rvalues"         "contNA"
[17] "drop.not.used"   "drop.pred.only"
```

By default, all variables except for the first one in the visit sequence (`visitSequence`) are synthesised using `ctree` implementation of CART models. First variable to be synthesised cannot have predictors that are to be synthesised later on and therefore a random sample (with replacement) is drawn from its observed values. The default visit sequence reflects the order of variables in the real data set - columns are synthesised from left to right.

The default matrix of predictors (`predictorMatrix`) is defined by the visit sequence. All variables that are earlier in the visit sequence are used as predictors. A value of 1 in a predictor matrix means that the column variable is used as a predictor for the target variable in the row. Since the order of variables is exactly the same as in the real data, for the default visit sequence the default predictor matrix has values of 1 in the lower triangle.

Synthesising data with default parametric methods is run with the methods listed below. Values of the other `syn()` arguments remain the same as for the default synthesis.

```
> set.seed(17914709)
> sds.parametric <- syn(rds, method = "parametric")
```

```
> sds.parametric$method
```

```
       sex        age        edu    marital     income         ls    wkabint
  "sample" "normrank"  "polyreg"  "polyreg" "normrank"  "polyreg"  "polyreg"
```

## 4.3. Extended example

To extend the simple example presented in Section 4.2 we change order of synthesis, synthesise only selected variables, customise selection of predictors, handle missing values in a continuous variable and apply some rules that a variable has to follow.

*Sequence and scope of synthesis*
The default algorithm of synthesising variables in columns from left to right can be changed via the `visitSequence` argument. The vector `visitSequence` should include indices of columns in an order desired by a user. In addition if we do not want to synthesise some variables we can exclude them from visit sequence. To synthesise variables `sex`, `age`, `ls`, `marital` and `edu` in this order we run `syn()` function with the following specification

```
> set.seed(17914709)
> sds.selection <- syn(rds, visitSequence = c(1, 2, 6, 4, 3))

Variable(s): income, wkabint not synthesised or used in prediction.
The variable(s) will be removed from data and not saved in synthesised data.

syn  variables
1    sex age ls marital edu
```

An appropriate prediction matrix is created automatically. However, despite the change of visit sequence the variables in `predictorMatrix` are arranged as in the original real data. The same refers to `method` and synthesised data set `syn`.

```
> sds.selection

Call:
($call) syn(data = rds, visitSequence = c(1, 2, 6, 4, 3))

Number of synthesised data sets:
($m)  1

First rows of synthesised data set:
($syn)
     sex age                       edu marital               ls
1 FEMALE  65                 SECONDARY MARRIED            MIXED
2 FEMALE  60       VOCATIONAL/GRAMMAR MARRIED          PLEASED
3 FEMALE  45 POST-SECONDARY OR HIGHER MARRIED MOSTLY SATISFIED
4 FEMALE  38       VOCATIONAL/GRAMMAR  SINGLE            MIXED
5   MALE  56       VOCATIONAL/GRAMMAR MARRIED MOSTLY SATISFIED
6   MALE  62       VOCATIONAL/GRAMMAR MARRIED          PLEASED
...

Synthesising methods:
($method)
     sex      age      edu  marital       ls
"sample"  "ctree"  "ctree"  "ctree"  "ctree"

Order of synthesis:
($visitSequence)
    sex     age      ls marital     edu
      1       2       5       4       3

Matrix of predictors:
($predictorMatrix)
        sex age edu marital ls
sex       0   0   0       0  0
age       1   0   0       0  0
```

```
edu        1   1   0       1  1
marital    1   1   0       0  1
ls         1   1   0       0  0
```

Note that a user-defined `method` vector (setting method for each variable separately) and a specified `predictorMatrix` both have to include information for all variables present in the original real data set regardless of whether they are in `visitSequence` or not. This allows changes in `visitSequence` without adjustments to the `method` and `predictorMatrix`. For variables not to be synthesised but still to be used as a predictor, which needs to be reflected in a `predictorMatrix`, an empty `method` (`""`) should be set.

*Selection of predictors*
The most important rule when selecting predictors is that independent variables in a prediction model have to be already synthesised. The only exception is when a variable is used only as a predictor and is not going to be synthesised at all. Assume we want to:

- exclude life satisfaction (`ls`) from the predictors of marital status (`marital`);
- use monthly income (`income`) as a predictor of life satisfaction (`ls`), education (`edu`) and marital status (`marital`) but do not synthesise income variable itself;
- use polytomous logistic regression (`polyreg`) to generate marital status (`marital`) instead of a default `ctree` method.

In order to build an adequate predictor matrix, instead of doing it from scratch we can define an initial `visitSequence` and corresponding `method` vector and run `syn()` function with parameter `drop.not.used` set to `FALSE` (otherwise `method` and `predictorMatrix` will miss information on `wkabint`), parameter `m` indicating number of synthesis set to zero and other arguments left as defaults. Then we can adjust the predictor matrix used in this synthesis and rerun the function with new parameters. The R code for this is given below.

```
> visitSequence.ini <- c(1, 2, 5, 6, 4, 3)
> method.ini <- c("sample", "ctree", "ctree", "polyreg", "", "ctree", "")
> set.seed(17914709)
> sds.ini <- syn(data = rds, visitSequence = visitSequence.ini,
+   method = method.ini, m = 0, drop.not.used = FALSE)

> sds.ini$predictorMatrix

        sex age edu marital income ls wkabint
sex       0   0   0       0      0  0       0
age       1   0   0       0      0  0       0
edu       1   1   0       1      1  1       0
marital   1   1   0       0      1  1       0
income    0   0   0       0      0  0       0
ls        1   1   0       0      1  0       0
wkabint   0   0   0       0      0  0       0
```

```
> predictorMatrix.corrected <- sds.ini$predictorMatrix
> predictorMatrix.corrected["marital","ls"] <- 0
> predictorMatrix.corrected

        sex age edu marital income ls wkabint
sex       0   0   0       0      0  0       0
age       1   0   0       0      0  0       0
edu       1   1   0       1      1  1       0
marital   1   1   0       0      1  0       0
income    0   0   0       0      0  0       0
ls        1   1   0       0      1  0       0
wkabint   0   0   0       0      0  0       0

> set.seed(17914709)
> sds.corrected <- syn(data = rds, visitSequence = visitSequence.ini,
+   method = method.ini, predictorMatrix = predictorMatrix.corrected)
```

*Handling missing values in continuous variables*
By default, numeric missing data codes for a continuous variable are treated as non-missing values. This may lead to erroneous synthetic values, especially when standard parametric models are used or when synthetic values are smoothed to decrease disclosure risk. The problem refers not only to the variable in question, but also to variables predicted from it. The parameter `contNA` of the `syn()` function allows to define missing-data codes for continuous variables in order to model them separately (see Section 3.2). In our simple example a continuous variable `income` has two types of missing values (`NA` and `-8`) and the `contNA` argument should be defined as follows

```
> contNA.income <- as.list(rep(NA, ncol(rds)))
> contNA.income[[5]] <- c(NA,-8)
```

*Rules for restricted values*
To illustrate application of rules for restricted values consider marital status. According to Polish law males have to be at least 18 to get married. Thus, in our synthesised data set all male individuals younger than 18 should have marital status `SINGLE` which is the case in the real data set. Running without rules gives incorrect results, which is particularly problematic for synthesis with parametric methods, where most of the males under 18 are classified as `MARRIED` (see summary table below).

```
> maritalM18.rds <- table(rds[rds$age < 18 & rds$sex == 'MALE' ,"marital"])
> maritalM18.default <- table(sds.default$syn[sds.default$syn$age < 18 &
+   sds.default$syn$sex == 'MALE',"marital"])
> maritalM18.parametric <- table(sds.parametric$syn[sds.default$syn$age < 18 &
+   sds.parametric$syn$sex == 'MALE',"marital"])
> cbind("Real data" = maritalM18.rds, CART = maritalM18.default,
+   Parametric = maritalM18.parametric)
```

```
                   Real data CART Parametric
SINGLE                    57   60         16
MARRIED                    0    2         44
WIDOWED                    0    0          0
DIVORCED                   0    0          0
LEGALLY SEPARATED          0    0          1
DE FACTO SEPARATED         0    0          1
```

Application of a rule, as specified below, leads to the correct results

```
> rules.marital <- list("","","","age < 18 & sex == 'MALE'","","","")
> rvalues.marital <- list(NA,NA,NA,'SINGLE',NA,NA,NA)
> set.seed(17914709)
> sds.rmarital <- syn(rds, rules = rules.marital, rvalues = rvalues.marital)
> set.seed(17914709)
> sds.rmarital.param <- syn(rds, rules = rules.marital,
+   rvalues = rvalues.marital, method = "parametric")
> rmaritalM18.default <- table(sds.rmarital$syn[sds.rmarital$syn$age < 18
+   & sds.rmarital$syn$sex == 'MALE',"marital"])
> rmaritalM18.parametric <- table(sds.rmarital.param$syn[
+   sds.rmarital.param$syn$age < 18
+   & sds.rmarital.param$syn$sex == 'MALE',"marital"])

> cbind("Real data" = maritalM18.rds, CART = rmaritalM18.default,
+   Parametric = rmaritalM18.parametric)
```

```
                   Real data CART Parametric
SINGLE                    57   62         52
MARRIED                    0    0          0
WIDOWED                    0    0          0
DIVORCED                   0    0          0
LEGALLY SEPARATED          0    0          0
DE FACTO SEPARATED         0    0          0
```

## 4.4. Synthetic data analysis

Optimally an analysis based on the synthesised data should lead to the same statistical inferences as an analysis based on the real data. For illustration we estimate here a simple logistic regression model where our dependant variable is a probability of having intention to work abroad. We use `wkabint` variable which specifies the intentions of work migration but we adjust it to disregard the destination country group. Besides we recode current missing data code of variable `income` (`'-8'`) into `R` missing data code `NA`.

```
> rds$wkabint <- as.character(rds$wkabint)
> rds$wkabint[rds$wkabint=='YES, TO EU COUNTRY' |
+   rds$wkabint=='YES, TO NON-EU COUNTRY'] <- 'YES'
> rds$wkabint <- factor(rds$wkabint)
> rds$income[rds$income==-8] <- NA
```

We generate five synthetic data sets.

```
> set.seed(17914709)
> sds <- syn(rds, m = 5)
```

Before running the models let us compare some descriptive statistics of the real and synthetic data sets. A very useful function in R for this purpose is `summary()`. When a data frame is provided as an argument, here our real data set `rds`, it produces summary statistics of each variable.

```
> summary(rds)

     sex             age                                edu
 MALE  :2182   Min.   :16.0   PRIMARY/NO EDUCATION    : 962
 FEMALE:2818   1st Qu.:32.0   VOCATIONAL/GRAMMAR      :1613
               Median :49.0   SECONDARY               :1482
               Mean   :47.7   POST-SECONDARY OR HIGHER: 936
               3rd Qu.:61.0   NA's                    :   7
               Max.   :97.0

               marital          income                       ls
 SINGLE             :1253   Min.   :  100   PLEASED            :1947
 MARRIED            :2979   1st Qu.:  970   MOSTLY SATISFIED   :1692
 WIDOWED            : 531   Median : 1350   MIXED              : 827
 DIVORCED           : 199   Mean   : 1641   MOSTLY DISSATISFIED: 274
 LEGALLY SEPARATED  :   7   3rd Qu.: 2000   DELIGHTED          : 191
 DE FACTO SEPARATED:  22   Max.   :16000   (Other)            :  61
 NA's               :   9   NA's   :1286    NA's               :   8
 wkabint
 NO  :4646
 YES : 318
 NA's:  36
```

The `summary()` function with the `synds` object as an argument gives summary statistics of the variables in the synthesised data set. If more than one synthetic data set has been generated, as default a summary of the first one is displayed. It can be changed using `msel` parameter which can be a single number or a vector.

```
> summary(sds)

Synthetic object with 5 syntheses using methods:
     sex      age      edu  marital   income       ls  wkabint
"sample"  "ctree"  "ctree"  "ctree"  "ctree"  "ctree"  "ctree"

Summary for synthetic data set 1:
     sex             age                                edu
```

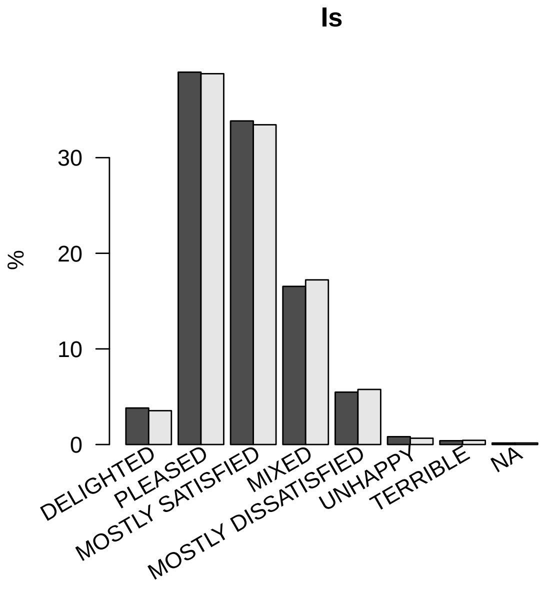
```
 MALE  :2180   Min.   :16.0   PRIMARY/NO EDUCATION    : 959
 FEMALE:2820   1st Qu.:32.0   VOCATIONAL/GRAMMAR      :1611
               Median :49.0   SECONDARY               :1486
               Mean   :47.7   POST-SECONDARY OR HIGHER: 936
               3rd Qu.:62.0   NA's                    :   8
               Max.   :97.0

               marital          income                       ls
 SINGLE             :1279   Min.   :  100   PLEASED            :1939
 MARRIED            :2985   1st Qu.:  968   MOSTLY SATISFIED   :1672
 WIDOWED            : 521   Median : 1350   MIXED              : 861
 DIVORCED           : 176   Mean   : 1613   MOSTLY DISSATISFIED: 288
 LEGALLY SEPARATED :   6   3rd Qu.: 2000   DELIGHTED          : 177
 DE FACTO SEPARATED:  24   Max.   :16000   (Other)            :  55
 NA's               :   9   NA's   :1246    NA's               :   8
 wkabint
 NO  :4637
 YES : 335
 NA's:  28
```

```
> summary(sds, msel = 2)
> summary(sds, msel = 1:5)
```

To more easily compare the synthesised variables with the real ones the synthesiser can use a `compare.synds()` function. It takes a synthetic data object and a data frame with original data as its arguments and compares relative frequency distributions of each variable in tabular and graphic form. Alternatively it can be used for a subset of variables specified by a `vars` argument. For quantitative variables it produces relative frequency distribution of various missing data categories and a histogram of non-missing values. The below function call is followed by an exemplary output for a factor (`ls`) and a numeric variable (`income`). Note that if a synthetic data object contains multiple synthetic data sets only the first one is used for comparison.

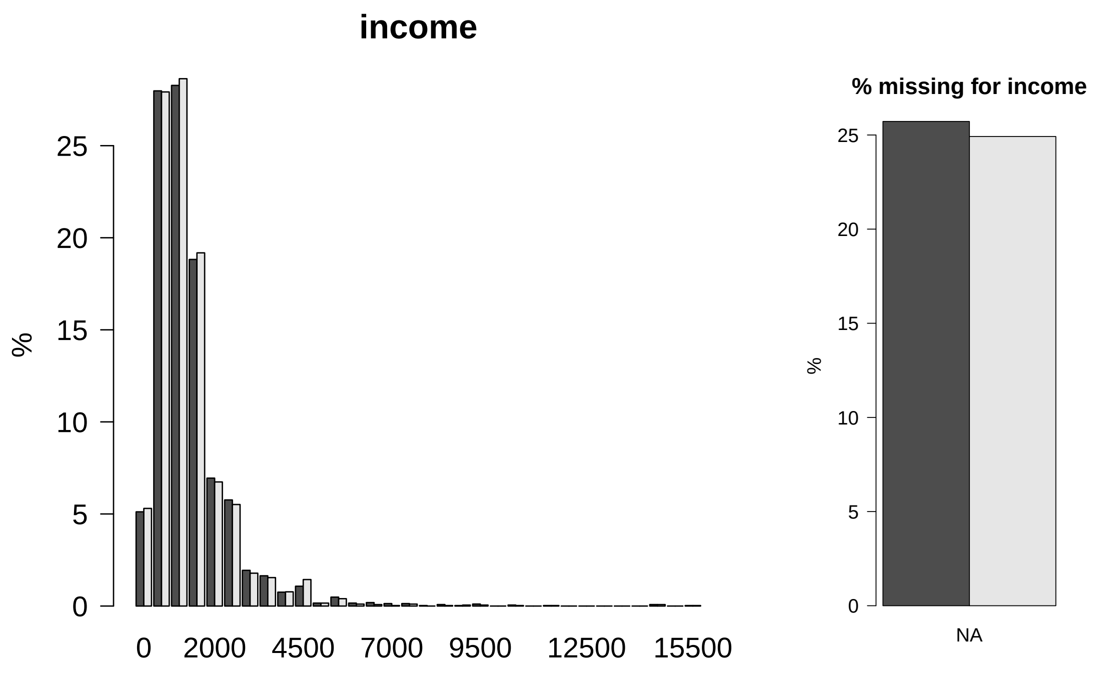
```
> compare.synds(sds,rds)

> compare.synds(sds,rds,vars="ls")

Data types of variables selected for comparison:

 ls
 "factor"

Comparing percentages actual (black) with synthetic (grey) for  ls

          DELIGHTED PLEASED MOSTLY SATISFIED MIXED MOSTLY DISSATISFIED
real           3.82   38.94            33.84 16.54                5.48
```

```
synthetic      3.54   38.78            33.44 17.22                5.76
          UNHAPPY TERRIBLE   NA
real         0.82     0.40 0.16
synthetic    0.66     0.44 0.16
```



```
> compare.synds(sds,rds,vars="income")

Data types of variables selected for comparison:

 income
 "numeric"

Comparing actual (black) with synthetic (grey) for % missing data in  income

             NA
real      25.72
synthetic 24.92

Comparing actual (black) with synthetic (grey) for  income

              0   500  1000  1500  2000  2500  3000  3500   4000  4500
real      5.116 27.98 28.27 18.82 6.947 5.762 1.939 1.642 0.7539 1.077
synthetic 5.301 27.92 28.64 19.18 6.739 5.514 1.785 1.545 0.7725 1.438
            5000   5500   6000    6500    7000   7500    8000    8500
real      0.1616 0.4847 0.1616 0.18848 0.13463 0.1346 0.02693 0.08078
```

```
synthetic 0.1598 0.3996 0.1066 0.07991 0.02664 0.1066 0.00000 0.02664
             9000    9500 10000   10500 11000   11500 12000 12500 13000
real      0.02693 0.10770     0 0.05385     0 0.02693     0     0     0
synthetic 0.05328 0.05328     0 0.02664     0 0.02664     0     0     0
          13500 14000   14500 15000   15500
real          0     0 0.08078     0 0.02693
synthetic     0     0 0.07991     0 0.02664
```



We estimate the real data model using generalised linear model implemented in R `glm()` function. A **synthpop** package function `glm.synds()` is an equivalent function for estimating models for each of the `m` synthesised data sets. A similar function called `lm.synds()` is available for a standard linear regression model. Note that the `glm.synds()` and `lm.synds()` functions have a parameter `object` rather than `data` as it is the case in `glm()` and `lm()` functions. An outcome of `glm.synds()` and `lm.synds()` function is an object of class `fit.synds`. If `m>1`, printing a `fit.synds` object gives estimates for the first synthesised data set only but it can be changed via an `msel` argument of a `print` method.

```
> model.rds <- glm(wkabint ~ sex + age + edu + log(income),
+   family = "binomial", data = rds)
> summary(model.rds)

Call:
glm(formula = wkabint ~ sex + age + edu + log(income), family = "binomial",
    data = rds)
```

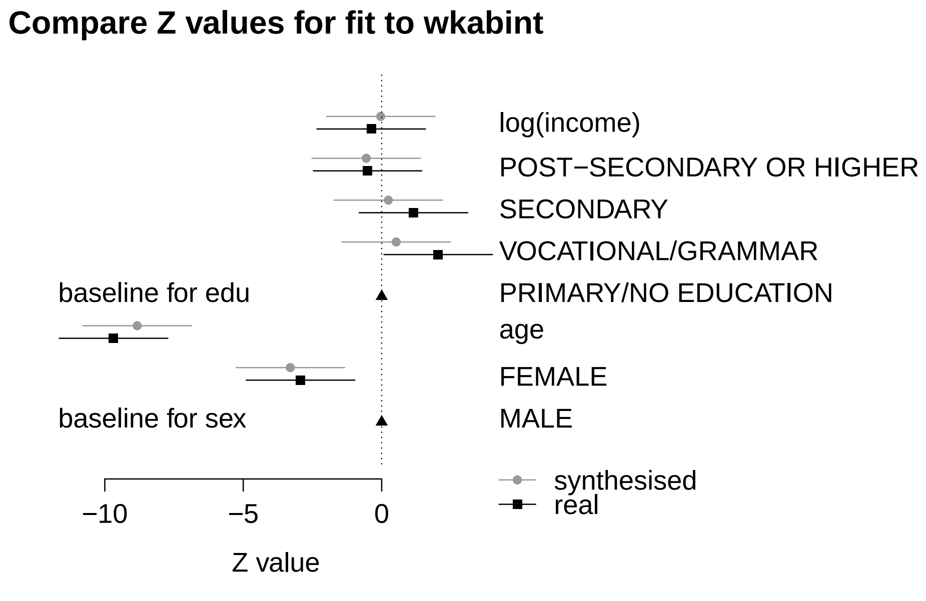
```
Deviance Residuals:
   Min      1Q  Median      3Q     Max
-0.873  -0.369  -0.250  -0.163   3.078

Coefficients:
                            Estimate Std. Error z value Pr(>|z|)
(Intercept)                 -0.21052    0.89125   -0.24   0.8133
sexFEMALE                   -0.47387    0.16182   -2.93   0.0034 **
age                         -0.05384    0.00556   -9.68   <2e-16 ***
eduVOCATIONAL/GRAMMAR        0.62753    0.30758    2.04   0.0413 *
eduSECONDARY                 0.36839    0.32125    1.15   0.2515
eduPOST-SECONDARY OR HIGHER -0.18697    0.36696   -0.51   0.6104
log(income)                 -0.04610    0.12224   -0.38   0.7061
---
Signif. codes:  0 '***' 0.001 '**' 0.01 '*' 0.05 '.' 0.1 ' ' 1

(Dispersion parameter for binomial family taken to be 1)

    Null deviance: 1543.6  on 3690  degrees of freedom
Residual deviance: 1371.7  on 3684  degrees of freedom
  (1309 observations deleted due to missingness)
AIC: 1386

Number of Fisher Scoring iterations: 7

> model.sds <- glm.synds(wkabint ~ sex + age + edu + log(income),
+   family = "binomial", object = sds)
> model.sds

Call:
glm.synds(formula = wkabint ~ sex + age + edu + log(income),
    family = "binomial", object = sds)

Coefficients:

syn = 1
                            Estimate Std. Error z value  Pr(>|z|)
(Intercept)                 -1.31506   0.991080 -1.3269 1.845e-01
sexFEMALE                   -0.45939   0.157634 -2.9143 3.565e-03
age                         -0.04565   0.005224 -8.7398 2.335e-18
eduVOCATIONAL/GRAMMAR        0.46426   0.293437  1.5822 1.136e-01
eduSECONDARY                 0.40394   0.298597  1.3528 1.761e-01
eduPOST-SECONDARY OR HIGHER  0.35048   0.324374  1.0805 2.799e-01
log(income)                  0.05152   0.130632  0.3944 6.933e-01

> print(model.sds, msel = 3)
```

The `summary()` function of a `fit.synds` object can be used by the analyst to combine estimates based on all the synthesised data sets. For inference to original data quantities it includes coefficients (`beta syn`) and their standard errors (`se beta syn`), Z scores (`Z syn`) and synthesising errors for coefficients (`syn err beta`) and Z-scores (`syn err Z`). For inference to population quantities it includes coefficients (`est`), synthesising errors (`se`) and Z scores (`Z`). The mean of the estimates from each of the $m$ synthetic data sets yields unbaised estimates of the coefficients. The variance is estimated differently depending whether inference is made to the real data quantities or the population parameters and whether synthetic data were produced using simple or proper synthesis (for details see Raab *et al.* (2014); expressions used to calculate variance for different cases are presented in Table 1). By default a simple synthesis is conducted and inference is made to original data quantities.

```
> summary(model.sds)

Fit to synthetic data set with 5 syntheses

Call:
glm.synds(formula = wkabint ~ sex + age + edu + log(income),
    family = "binomial", object = sds)

Combined estimates:
                            beta syn se beta syn    Z syn syn err beta
(Intercept)                 -0.50547    0.984396 -0.51348     0.440235
sexFEMALE                   -0.52156    0.158069 -3.29954     0.070691
age                         -0.04627    0.005241 -8.82821     0.002344
eduVOCATIONAL/GRAMMAR        0.14027    0.267637  0.52412     0.119691
eduSECONDARY                 0.06621    0.275793  0.24006     0.123339
eduPOST-SECONDARY OR HIGHER -0.17350    0.312116 -0.55589     0.139583
log(income)                 -0.00493    0.130486 -0.03778     0.058355
                            syn err Z
(Intercept)                    0.4472
sexFEMALE                      0.4475
age                            0.4495
eduVOCATIONAL/GRAMMAR          0.4472
eduSECONDARY                   0.4472
eduPOST-SECONDARY OR HIGHER    0.4472
log(income)                    0.4472
```

Function `compare.fit.synds()` allows the synthesiser to compare the estimates based on the synthesised data sets with those based on the real data and presents the results in both tabular and graphical form.

```
> compare.fit.synds(model.sds,rds)

Call used to fit models to synthetised data sets:
glm.synds(formula = wkabint ~ sex + age + edu + log(income),
    family = "binomial", object = sds)

Coefficients (combined results for synthetised data sets):
                             beta real se beta real  Z real beta syn
(Intercept)                   -0.21052     0.891248 -0.2362 -0.50547
sexFEMALE                     -0.47387     0.161820 -2.9284 -0.52156
age                           -0.05384     0.005559 -9.6836 -0.04627
eduVOCATIONAL/GRAMMAR          0.62753     0.307576  2.0402  0.14027
eduSECONDARY                   0.36839     0.321252  1.1467  0.06621
eduPOST-SECONDARY OR HIGHER   -0.18697     0.366956 -0.5095 -0.17350
log(income)                   -0.04610     0.122240 -0.3772 -0.00493
                             se beta syn    Z syn syn err beta syn err Z
(Intercept)                     0.984396 -0.51348     0.440235    0.4472
sexFEMALE                       0.158069 -3.29954     0.070691    0.4475
age                             0.005241 -8.82821     0.002344    0.4495
eduVOCATIONAL/GRAMMAR           0.267637  0.52412     0.119691    0.4472
eduSECONDARY                    0.275793  0.24006     0.123339    0.4472
eduPOST-SECONDARY OR HIGHER     0.312116 -0.55589     0.139583    0.4472
log(income)                     0.130486 -0.03778     0.058355    0.4472
```



## 5. Concluding remarks

In this paper we presented the basic functionality of R package **synthpop** for generating synthetic data. Interested readers can consult the package documentation for additional features currently implemented which can be used to influence the disclosure risk and the utility of the synthesised data. Note that **synthpop** is under continual development and future vesrsions will include, among others, appropriate procedures for synthesising multiple event data, conducting stratified synthesis and generating partially synthetic data. The ultimate aim of **synthpop** is to provide a comprehensive, flexible and easy to use tool for generating bespoke synthetic data that can be safely released to interested data users. Since there are many different options to synthesise data, developing general guidelines for best practice remains an open issue to be addressed in our future research.

## References

Caiola G, Reiter JP (2010). "Random forests for generating partially synthetic, categorical data." *Transactions on Data Privacy*, **3**(1), 27–42.

Council for Social Monitoring (2011). "Social Diagnosis 2000-2011: Integrated Database." URL `http://www.diagnoza.com/index-en.html`.

Drechsler J (2011). *Synthetic Data Sets for Statistical Disclosure Control*. Springer, New York.

Dreschler J, Reiter JP (2010). "Sampling with synthesis: a new approach for releasing public use census midrodata." *Journal of the American Statistical Association*, **105**(492), 1347–1357.

Elliot M, Purdam K (2007). "A case study of the impact of statistical disclosure control on data quality in the individual UK Samples of Anonymized Records." *Environment and Planning A*, **39**(5), 1101–1118.

Hothorn T, Hornik K, Zeileis A (2006). "Unbiased recursive partitioning: A conditional inference framework." *Journal of Computational and Graphical Statistics*, **15**(3), 651–674. URL `http://www.tandfonline.com/doi/abs/10.1198/106186006X133933`.

Ohm P (2010). "Broken promises of privacy: Responding to the surprising failure of anonymization." *UCLA Law Review*, **57**(6), 1701–1775.

Raab G, Nowok B, Dibben C (2014). "A simplified approach to synthetic data." *Submitted*.

Raghunathan TE, Reiter JP, Rubin DB (2003). "Multiple imputation for statistical disclosure limitation." *Journal of Official Statistics*, **19**(1), 1–17.

Reiter JP (2002). "Satisfying disclosure restrictions with synthetic data sets." *Journal of Official Statistics*, **18**(4), 531–544.

Reiter JP (2005a). "Releasing multiply imputed, synthetic public use microdata: An illustration and empirical study." *Journal of the Royal Statistical Society. Series A: Statistics in Society*, **168**(1), 185–205.

Reiter JP (2005b). "Using CART to generate partially synthetic, public use microdata." *Journal of Official Statistics*, **21**, 441–462.

Reiter JP, Kinney SK (2012). "Inferentially valid, partially synthetic data: Generating from posterior predictive distributions not necessary." *Journal of Official Statistics*, **28**(4), 583–590.

van Buren S, Groothuis-Oudshoorn K (2011). "mice: Multivariate Imputation by Chained Equations in R." *Journal of Statistical Software*, **45**(3), 1–67. URL http://www.jstatsoft.org/v45/i03/.

Winkler WE (2007). "Examples of easy-to-implement, widely used methods of masking for which analytical properties are not justified." *Technical Report Series 2007 21*, Statistical Research Division, U.S. Census Bureau, Washington, DC 20233. URL http://www.census.gov.edgekey.net/srd/papers/pdf/rrs2007-21.pdf.

**Affiliation:**

Beata Nowok
Institute of Geography
School of GeoSciences
University of Edinburgh
Drummond Street
Edinburgh EH8 9XP
E-mail: beata.nowok@ed.ac.uk