Generating synthetic data with the synthpop package for R

Introduction & background

Gillian Raab
Administrative Data Research Centre – Scotland
Objectives of the workshop?

- To enable you to create synthetic data sets with the `synthpop` package
- To understand the different options available in `synthpop` to define the models used for creating the synthetic data and to customise your synthesis to specific characteristics of your data
- To know how to evaluate your synthetic data using general and specific utility measures
- To compare the utility of different synthesis options
- To understand the SDC options available in `synthpop` for post-processing your synthetic data
- To know what strategies you might use to handle large and complex data sets
Origins of the synthpop package

- Desire to make synthetic data available to users of the Scottish Longitudinal Study (SLS)
- The SLS holds Scottish Census data, linked over time as well as much linked administrative data
- Each user as his or her own extract prepared and needs to visit the safe setting
- US Census Bureau products
  - each produced by a whole team of analysts
- SLS
  - a new synthesis is needed for each user
- Hence the synthpop package we hope you will learn today
- A trained and accredited user can request a bespoke a synthetic data set for preliminary analysis
- The final analysis will be run on the real data by visiting the safe setting, or by users submitting code to be run by SLS staff
Generating synthetic data: synthpop

library(synthpop)
mydata <- read.csv("mydata.csv")
mysyndata <- syn(mydata)
mysyndata
Help file for syn

Most important parameters

- **m** the number of synthetic data sets to be produced
- method vector of method to be used for each variable
- visit.sequence
- predictor.matrix
- cont.na defines missing value codes
- rules/ rvalues restrictions on values
- smoothing

```r
syn.strata(data, strata = NULL, 
   m = 1, k = nrow(data), proper = FALSE, 
   mminlevels = 1, mmaxlevels = 60, 
   rules = NULL, rvalues = NULL, 
   cont.na = NULL, semicont = NULL, 
   smoothing = NULL, event = NULL, denom = NULL, 
   drop.not.used = FALSE, drop.pred.only = FALSE, 
   default.method = c("normrank", "logreg", "polyreg", "polr"), 
   numtocat = NULL, catgroups = rep(5, length(numtocat)), 
   models = FALSE, print.flag = TRUE, seed = "sample", ...) 
```

```r
# S3 method for class 'syns'
print(x, ...) 
```

**Arguments**

- **data**
  - a data frame or a matrix (x, y) containing the original data. Observations are in rows and variables are in columns.
- **method**
  - a single string or a vector of strings of length ncol(data) specifying the synthesising method to be used for each variable in the data. Order of variables is exactly the same as in data. If specified as a single string, the same method is used for all variables in a visit sequence unless a data type or a position in a visit sequence requires a different method. If method is set to "parametric" the default synthesising method specified by the default.method argument are applied. Variables that are transformations of other variables can be synthesised using a passive method that is specified as a string starting with s (see syn.passive). Variables that need not to be synthesised have the empty method "". By default all variables are synthesised using "cart" method, which is short implementation of a CART model (see syn.cart). See details for more information on method.
**mysyndata** is an object of class **synds**

A list with many components

- `syn` the synthetic data set(s)
- `visit.sequence`
- `predictor matrix`
- `cont.na`
- `rules/rvalues`
- `Smoothing`
- ..........

Methods for evaluating your synthesis compared to the original can be applied to the synds object.
compare(object, data)
What we will not cover today

- Creating and evaluating multiple synthetic data sets ($m>1$)

- Parameter `proper = TRUE`
  - Synthetic data are created by fitting models to the original and generating the synthetic data from a model
  - Defaults to `proper = FALSE`
  - With `proper = TRUE` the parameters used to generate the synthetic data are samples from the predictive distribution of the parameters given the data

- When does this matter?
  - When making inferences from synthetic data with functions like `glm.synds()` – fitting models to synthetic data and its summary function with parameter `population.inference = TRUE`

- By default we assume that `population.inference = FALSE`
  - This is appropriate when the data are being created for exploratory analysis and the final results will be run on the original data

- The vignette `synthpop : inference` goes into these details and the methods we implement will be discussed by Jorg Dreschler on Friday
Find it here

Or here
https://cran.r-project.org/web/packages/synthpop/index.html
Overview of methods

- We fit a model to the joint distribution of all the variables
- Then generate a synthetic data set from the chosen model with parameters defined by this fit
- Standard parametric models are seldom useful. Very little real survey data follows standard parametric distribution e.g. Normal or transformed Normal
- Some of the methods provided are empirical rather than parametric, e.g. “sample”, “CART”, “random.forest”, “normrank” (regression on Normal scores, with back transformation)
- These methods can preserve the exact original observations which may be disclosive
- So smoothing of continuous variables is available and recommended
Building up the model from conditional distributions

- Suppose we have a data set with 
  - **age**, **sex**, and **marital status**

- Sequence of models
  - First we take a bootstrap sample of **age** to make the first column of the synthetic data `syn$age`
  - Then we fit a logistic model to predict **sex** from **age**, using the real data and make the next column of the synthetic data by predicting **sex** from `syn$age` to get `sex$syn`
  - Then we fit a multinomial model of **marital status** in terms of **age** and **sex** with the real data and make the next column of the synthetic data by predicting from `syn$age` and `syn$sex` to get `syn$maritalstatus`

- Since version 1.5.0 we now have two possible multivariate methods
  - “ipf” (iterative proportional fitting) log-linear models defined by the margins that are preserved
  - “catall” saturated log linear models
  - Variables with these methods must be synthesised as a group either for all variables or before further conditional models
  - They can be used for numeric variables by grouping them into categories with parameter `numtocat`. Synthetic numeric variables are then created from bootstrap samples of the original within categories
  - Can be time and memory hungry
Default values of parameters

- `visit.sequence`  1: `length(data)` or `names(data)`
- `method = "cart"`  gives a set of conditional models with all conditional distributions defined as CART models
  Note this must be in the order of the variables in the original data
  ```
  sex  agegr  edu  socprof
  "sample"  "cart"  "cart"  "cart"
  ```
- `predictor.matrix`  all previously synthesised variables
  ```
  sex  agegr  edu  socprof
  sex  0  0  0  0
  agegr  1  0  0  0
  edu  1  1  0  0
  socprof  1  1  1  0
  ```
  - Best way to change this is to do a trial run with `m = 0`
  - `syn0 <- syn(data,m=0)`
  - `pm <- syn0$predictor.matrix`
  - Then alter elements of `pm` e.g. `pm[4,2:3] <- 0`
**syn() method**

**Single string**
- same method for each variable, except the first one to be synthesised ("sample")
- default "cart" ("ctree" different implementation of CART)
- if "parametric" default parametric methods assigned to variables to be synthesised based on their types (default.method) but "normrank" not "norm"
- method for deterministic relations: "~I(weight/height^2)"
- new method: write `syn.newmethod()` and use "newmethod" in `syn()`

**Vector of strings**
- method has to be specified for each variable in the data set
- for variables to be left unchanged use an empty method ("")

See `?syn` for all methods currently available and `?syn.cart`, `?syn.norm`, etc. for details about specific method
Other parameters for syn

- **Seed**
  - Set to allow the same synthetic data to be created again. `seed = 9642` Assigned and stored if not set

- **cont.na**
  - Indicates a missing value for a continuous variable, other than NA. E.g. -8 for income not relevant or -9 not answered would be coded as `cont.na = list(income = c(-8,-9))`

- **Missing values (including these)**
  - Are synthesised as additional variables that are dependent on other variables, or predict them (i.e. missing at random)

- **rules and rvalues**
  - To enforce relationships. E.g. if all single people should have missing year of marriage
  - E.g. `rules = list(ymarr = "marital =='SINGLE' ")`, `rvalues = list(ymarr =NA)`

Unexpected values (not obeying the rules) found for variable(s): ymarr. Rules have been applied but make sure they are correct.
Evaluating your synthesis

Graphically

- Univariate distributions
  - Use compare
- Bivariate distributions
  - Use multi-compare

Statistically

This will follow after you have had a practice
compare(object, data)
multi.compare(object, data)

Two categorical variables
multi.compare(syn1, test1, var="marital", by="sex")
multi.compare(object, data)

One categorical one numeric
multi.compare(syn1, test1 var="income", by="marital")
multi.compare(object, data)

One categorical one numeric
multi.compare(syn1, test1 var="income", by="marital", cont.type = "boxplot")
Finally

Examining models used in creating the synthetic data

Synthesis by “ctree” method

income trustneigh trustfam trust *depress* marital socprof edu agegr sex
Health warnings and disclaimers

- Synthetic data are only as good as the models used to create them and should always be checked.
- To be able to synthesise any of the features of real data is a big challenge.
- As synthpop is open source it is being used by others beyond the LSs.
- Several groups we know of have used it to provide data sets to be used for teaching.
- Now over to you to practice – follow notes.