dsSynthetic: Synthetic data generation for the DataSHIELD federated analysis system

Soumya Banerjee [1]*, Tom R.P. Bishop[1] * $

1 Medical Research Council Epidemiology Unit, University of Cambridge School of Clinical Medicine, Cambridge, United Kingdom

* **Equal contribution**

$ **Corresponding author**

E-mail: Corresponding author Tom.Bishop@mrc-epid.cam.ac.uk

Abstract

Objective Platforms such as DataSHIELD allow users to analyse sensitive data remotely, without having full access to the detailed data items (federated analysis). While this feature helps to overcome difficulties with data sharing, it can make it challenging to write code without full visibility of the data. One solution is to generate realistic, non-disclosive synthetic data that can be transferred to the analyst so they can perfect their code without the access limitation. When this process is complete, they can run the code on the real data.

Results We have created a package in DataSHIELD (dsSynthetic) which allows generation of realistic synthetic data, building on existing packages. In our paper and accompanying tutorial we demonstrate how the use of synthetic data generated with our package can help DataSHIELD users with tasks such as writing analysis scripts and harmonising data to common scales and measures.

Keywords

Synthetic data, data harmonisation, federated analysis.

Introduction

Research is increasingly dependent on the analysis and interpretation of sensitive, personal data (or data that cannot be easily shared) or on the co-analysis of such data from several sources simultaneously. Making data available so that it can be used, may be impeded by ethico-legal processes or by fear of loss

of control of data. This results in a lot of work to get permission to gather several datasets together in a central location.

DataSHIELD (built on R) provides a novel solution that can circumvent some of these challenges [1] [2]. The key feature of DataSHIELD is that data stay on a server at each of the institutions that are responsible for the data. Each institution has control over who can access their data. The platform allows a user to pass commands to each server and the analyst receives results that are designed to only disclose the summary data. For example, the user can fit a linear model to the data but not see the residuals. Thus DataSHIELD can be used to analyse data without physically sharing it with the users. We refer to this as federated analysis.

A challenge of this approach is that it is harder for data analysts to write, debug and check code that uses the data when the data are not tangibly in front of them. This is because DataSHIELD prevents access to the data at a detailed level. It is similar to trying to build a Lego model while blindfolded. While it is possible for the user to understand what is happening by asking for summary statistics, this process can still be difficult. For example, to confirm that a subset operation into male and female groups has been successful, the analyst could ask for a summary of the original gender column and check whether the counts of male and female participants match the length of the subset objects. With the datasets in front of them it is immediately obvious if a problem has occurred.

Hypothesis for using synthetic data

R packages like *synthpop* [3] have been developed to generate realistic synthetic data that is not disclosive. That is, the synthetic data is representative of the real data, but it could not be used to recover any of the real data. *synthpop* uses classification and regression trees (CART) [4] to generate synthetic data. *synthpop* fits a sequence of regression models. Once the models are fit, synthetic values are generated by drawing from the predictive distributions.

In DataSHIELD, the user ("client") passes analytical commands to multiple datasets held on remote servers. If the user could have a client side synthetic copy of the real data that is held on each of the servers they are using, this could make code writing, checking and debugging easier. They could have full access to the synthetic data, and confirm that their code is working correctly before applying it to the real data held on the servers.

The user therefore has the benefit of being able to see the data they are working with, but without

the need to go through laborious data transfer processes.

Other packages that provide synthetic data generation are *simstudy* [5] and *gcipdr* [6]. *simstudy* [5] requires the user to define the characteristics of variables and their relationships. Non-disclosive access via DataSHIELD can provide these summary statistics.

Likewise, *gcipdr* [6] requires users to extract features such as mean, standard deviation and correlations via DataSHIELD, and uses these to provide automated generation of the synthetic data.

We introduce a package (*dsSynthetic* 1.0) and an associated bookdown which provides functionality for generating a local, non-disclosive synthetic copy of data held on remote servers via DataSHIELD. We build on the *synthpop* [3] and *simstudy* packages [5]. Our objective is to generate synthetic data that is realistic enough to allow users to more easily design, implement and test code. This code can then be applied on the real data.

Therefore we have written the dsSynthetic and partner dsSyntheticClient R packages (hereafter dsSynthetic) to help users to write code for data that is available in a DataSHIELD setting. A tutorial covering the examples in this text with executable code is available here:

https://tombisho.github.io/synthetic_bookdown

Main text

Overview of steps

We describe the steps for generating synthetic data with the dsSynthetic package below:

1. The data custodian uploads the raw data to the server side and installs the server side package *dsSynthetic*.

2. The user installs the package *dsSyntheticClient* on the client side.

3. The user calls functions in the *dsSyntheticClient* package to generate a synthetic but non-disclosive data set which is transferred from the server to the client side.

   The generation of the synthetic data can use methods from: a) synthpop, where the synthetic data are generated on the server side and returned to the client, or b) simstudy where non-disclosive summary characteristics of and relationships between variables are generated on the server side, returned to the client, and the synthetic data are generated

4. With the synthetic data on the client side, the user can view the data and write code. They will be able to see the results of the code for the whole dataset.

5. When the code is complete, it can be implemented on the server using the real data.

These variations are shown in Fig. 1.

The computational steps are outlined below.

Algorithm 1: Workflow for generating synthetic data in dsSynthetic.

```
%Connecting to server(s)
library(dsSyntheticClient)
library(DSOpal)
library(dsBaseClient)
library(DSI)
builder = DSI::newDSLoginBuilder()
builder$append(server="server1", url= "https://opal-sandbox.mrc-epid.cam.ac.uk", user="dsuser",
 password="P@ssw0rd", table = "DASIM.DASIM1")
logindata = builder$build()
%Login to server
connections = datashield.login(logins=logindata, assign = TRUE)
%Generate synthetic data
synth_data = dsSyntheticClient::ds.syn(data = "D", method = "cart", m = 1, seed = 123)$server1$syn
```

Using synthetic data to build a DataSHIELD analysis script

A key use-case for *dsSynthetic* is to aid analysts in writing DataSHIELD analysis scripts in the standard DataSHIELD scenario where the real data cannot be fully accessed. We assume that the data sets are ready for analysis at different sites, the *dsSynthetic* package is installed on the servers and the *dsSyntheticClient* package is installed on the analyst's computer (client). This use-case is also demonstrated in Fig. 2.

1. Generate the synthetic data so that it is available on the client (this could be data from multiple servers).

2. Load the data into *DSLite* [7].

To test our DataSHIELD script locally on the client, we need to replicate the server environment(s) (which only accepts DataSHIELD function input) on the client. This is provided by the *DSLite* package.

A DSLite instance exists in a user's local R session and behaves the same as a server, except that it allows full access to the data held on it.

Therefore the script can be developed against the DSLite instance and at any time the user can view what is happening on DSLite. This makes it easier for the user to correct any problems.

3. Once finished, the script is run on real data on the remote server(s). This step should now run smoothly as any problems with the code have been identified when working with the synthetic data.

   Once finished, the script is run on real data. A worked example of this is detailed in the tutorial.

Using synthetic data to write harmonisation code

Prior to analysis with DataSHIELD, data must be harmonised so that variables have the same definition across datasets [8]. Common solutions to achieving harmonisation are to:

1. Have each data custodian harmonise their data and host it on the server. The challenge with this is that it requires a lot of coordination across groups and there can be a lack of visibility of how harmonisation was achieved.

2. Make a one-off transfer to a central group which does the harmonising work before the data are returned to the custodian for the analysis phase by multiple parties. This suffers the same challenges of a data transfer for analysis.

To avoid these challenges a third way is for a central group to write harmonisation code in Opal [9] which can be used as the hosting data warehouse on each of the participating servers. This code acts on the raw data on each server to generate datasets which contain harmonised variables. That is, all the variables are on common scales and measures (e.g. all measures of height in metres). To eliminate the need for a data transfer or full data access agreement, the users log into Opal to write the harmonisation code. This means they may not have full access to the data, only to summary statistics.

Again, this makes it challenging to write and test the code. A further complication is that Opal requires the code to be written in MagmaScript, which is based on JavaScript. This language is generally

unfamiliar to users that do not have a background in software development. dsSynthetic allows users to write and test MagmaScript on a local synthetic copy of data, before implementing it on the server running Opal.

As with the analysis use case, this testing phase is performed in the R environment in DataSHIELD, and implemented on the server once the code is working properly. Our package enables this by generating synthetic data. We then use the V8 package to run MagmaScript and JavaScript within R. This tested MagmaScript code can then be pasted into the Opal server to run against the real data.

A schematic of this workflow is shown in Fig. 3.

In detail, the steps proposed are:

1. Generate synthetic data as described previously.
2. Start a JavaScript session on the client side.
3. Load the synthetic data into the session.
4. Write and test MagmaScript code in the session using synthetic data.
5. When testing is complete, copy the code into the Opal server to generate the harmonised version of the real data.

A worked example of this is detailed in the tutorial.

Minimising risk of disclosure

A concern with generating a synthetic dataset is that it might reveal too much information about the real data. For example, that a real record from the real data is somehow included in the synthetic data. In the case of using *synthpop*, existing work has assessed this risk to be low [10].

We have also disabled built-in features of *synthpop* where certain options allow the return of real data. When generating synthetic data via simstudy, this relies on non-disclosive results such as mean and standard deviation that can already be obtained via DataSHIELD, therefore there is no additional risk of disclosure.

Discussion

We introduce a package (*dsSynthetic*) which provides functionality for generating synthetic data in DataSHIELD. This can aid writing and testing of code. Synthetic data generation is also available as part of other privacy preserving frameworks like OpenSAFELY [11] and other packages [12] [13]. Its purpose in OpenSAFELY is to reduce the number of queries that are performed on the real data and its infrastructure by making sure that code runs on local synthetic data first. *dsSynthetic* is simpler to use and generates synthetic data that is more realistic. A version of *synthpop* for DataSHIELD is offered in [12] but the *simstudy* method is not available, nor are the detailed workflows for generating and using synthetic data.

Our package offers more flexibility and power in generating synthetic data by offering different options to generate synthetic data (using packages like *simstudy* and *synthpop*) and comprehensive workflows for writing DataSHIELD scripts and data harmonisation

We hope our package and associated tutorial will help users write DataSHIELD code and harmonise data in the field of federated analysis by providing them with synthetic data to simplify the task.

Limitations

We inherit the limitations of the underlying packages *simstudy* and *synthpop*. In particular, the user should exercise judgment about the number of variables requested in a synthetic dataset, so as to not overwhelm the server performing the generation.

Abbreviations

DSLite: DataSHIELD Lite.

Declarations

Acknowledgements

We acknowledge the help and support of the DataSHIELD technical team. We are especially grateful to Paul Burton, Stuart Wheater, Eleanor Hyde, Yannick Marcon and Demetris Avraam for fruitful

discussions and feedback.

Funding statement

This work was funded by EUCAN-Connect under the European Union's Horizon 2020 research and innovation programme (grant agreement no. 824989). The funders had no role in study design, data collection and analysis, decision to publish, or preparation of the manuscript. The views expressed are those of the authors and not necessarily those of the funders. The views expressed are those of the authors and not necessarily those of the funders.

Competing interests

All authors declare they have no conflicts of interest to disclose.

Ethics approval and consent to participate

No ethics approval and consent to participate was necessary.

Consent to participate

Not applicable.

Data accessibility

No data was generated in this study.

Author contributions

SB and TB wrote the manuscript. TB carried out the analysis and implementation, and participated in the design of the study. TB directed the study. All authors gave final approval for publication.

Availability of data and materials

No data was generated in this study.

All code and a tutorial are available here as an R bookdown format with executable code:

https://tombisho.github.io/synthetic_bookdown

All code is available from:

https://github.com/tombisho/dsSynthetic

https://github.com/tombisho/dsSyntheticClient

References

1. Gaye A, Marcon Y, Isaeva J, Laflamme P, Turner A, et al. (2014) DataSHIELD: Taking the analysis to the data, not the data to the analysis. Int J Epidemiol 43: 1929–1944.

2. DataSHIELD — DataSHIELD — Newcastle University. URL `http://www.datashield.ac.uk/`.

3. Nowok B, Raab GM, Dibben C (2016) Synthpop: Bespoke creation of synthetic data in R. J Stat Softw 74: 1–26.

4. Gareth J, Daniela W, Trevor H, Robert T (2015) Introduction to statistical learning. doi:10.1016/j.peva.2007.06.006. URL `http://faculty.marshall.usc.edu/gareth-james/ISL/`. `arXiv:1011.1669v3`.

5. Goldfeld K, Wujciak-Jens J (2020). CRAN - Package simstudy. URL `https://cran.r-project.org/web/packages/simstudy/index.html`.

6. Bonofiglio F, Schumacher M, Binder H (2020) Recovery of original individual person data (IPD) inferences from empirical IPD summaries only: Applications to distributed computing under disclosure constraints. Stat Med 39: 1183–1198.

7. datashield/DSLite: Server-less implementation of the DataSHIELD interface. URL `https://github.com/datashield/DSLite`.

8. Fortier I, Doiron D, Little J, Ferretti V, L'Heureux F, et al. (2011) Is rigorous retrospective harmonization possible? Application of the dataSHaPER approach across 53 large studies. Int J Epidemiol 40: 1314–1328.

9. Doiron D, Marcon Y, Fortier I, Burton P, Ferretti V (2017) Software application profile: Opal and mica: Open-source software solutions for epidemiological data management, harmonization and dissemination. Int J Epidemiol 46: 1372–1378.

10. Elliott M (2014) Final Report on the Disclosure Risk Associated with the Synthetic Data Produced by the SYLLS Team. Technical report. URL https://hummedia.manchester.ac.uk/institutes/cmist/archive-publications/reports/2015-02-Report on disclosure risk analysis of synthpop synthetic versions of LCF_final.pdf.

11. Mathur R, Rentsch CT, Morton CE, Hulme WJ, Schultze A, et al. (2021) Ethnic differences in SARS-CoV-2 infection and COVID-19-related hospitalisation, intensive care unit admission, and death in 17 million adults in England: an observational cohort study using the OpenSAFELY platform. Lancet 397: 1711–1724.

12. Dragan I, Sparsø T, Kuznetsov D, Slieker R, Ibberson M (2020) dsSwissKnife: An R package for federated data analysis. bioRxiv : 2020.11.17.386813.

13. Lenz S, Hess M, Binder H (2021) Deep generative models in DataSHIELD. BMC Med Res Methodol 21: 64.

Figures

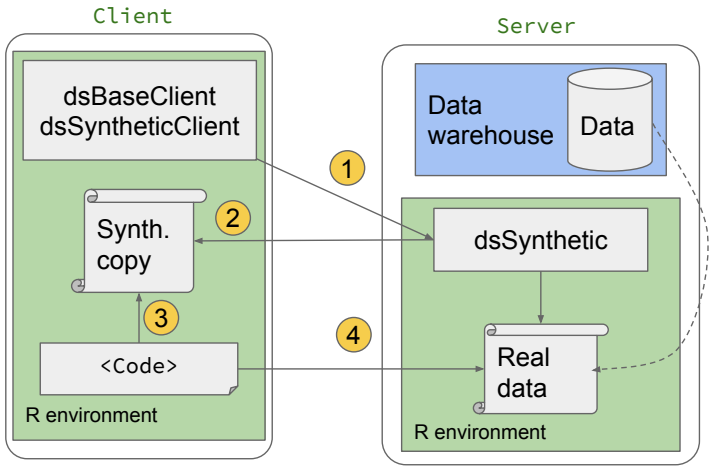

Figure 1. A schematic of generating synthetic data with dsSynthetic, and using it to write, debug and test code. The final code is then used on real data on the server 1. User requests synthetic copy of real data 2. Synthetic copy of data generated on server side and passed to client 3. User writes code against synthetic data on client side 4. Code run on server side with real data.

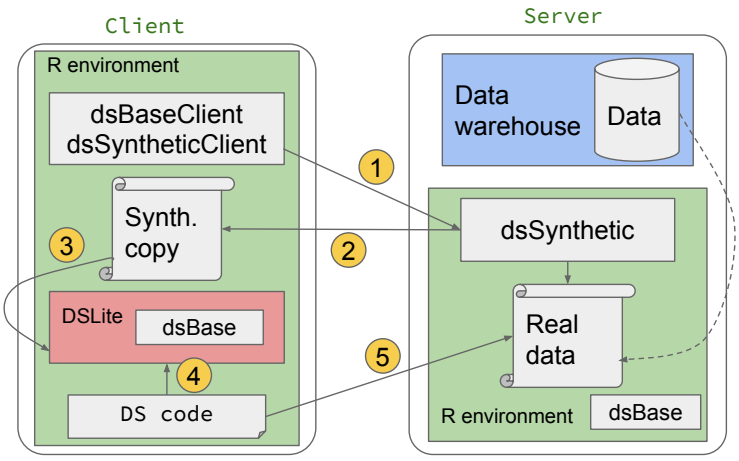

Figure 2. Writing an analysis script via synthetic data without full access. The steps are the following: 1. User requests synthetic copy of real data 2. Synthetic data generated on server side and passed to client 3. Synthetic data held in DSLite environment that simulates server side DataSHIELD environment 4. Develop code against synthetic data in DSLite (with full access) 5. Run code against real data.

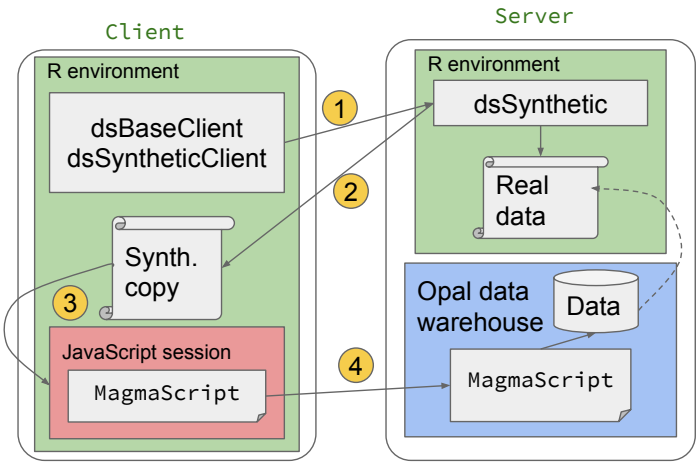

Figure 3. Central harmonisation via synthetic data without full access. The steps are the following: 1. User requests synthetic copy of real data 2. Synthetic copy generated on server side and passed to client 3. Synthetic data loaded into JavaScript session. User writes harmonisation code in (MagmaScript) on client side 4. MagmaScript code implemented on server side to run on real data.