The Journal of Open Source Software

Metasyn: Transparent Generation of Synthetic Tabular Data with Privacy Guarantees

Raoul Schram [1*], **Samuel Spithorst** [1], **and Erik-Jan van Kesteren** [1,2*¶]

1 Utrecht University, The Netherlands **2** ODISSEI: Open Data Infrastructure for Social Science and Economic Innovations, The Netherlands ¶ Corresponding author * These authors contributed equally.

DOI: 10.xxxxxx/draft

Software

- Review
- Repository
- Archive

Editor: Open Journals

Reviewers:

- @openjournals

Submitted: 01 January 1970
Published: unpublished

Summary

Synthetic data is a promising tool for improving the accessibility of datasets that are otherwise too sensitive to be shared publicly. To this end, we introduce `metasyn`, a Python package for generating synthetic data from tabular datasets. Unlike existing synthetic data generation software, `metasyn` is built on a simple generative model with a "naïve" marginal independence assumption — an explicit choice that removes multivariate information from the synthetic data. It makes this trade-off in order to maintain transparency and auditability, to keep information leakage to a minimum, and even to enable privacy or disclosure risk guarantees through a plug-in system. While the analytical validity of the generated data is thus intentionally limited, its potential uses are broad, including exploratory analyses, code development and testing, and external communication and teaching (van Kesteren, 2024). `Metasyn` is flexible, scalable, and easily extended to meet diverse privacy needs.

Figure 1: Logo of the `metasyn` project.

Statement of need

`Metasyn` is a python package for generating synthetic data with a focus on privacy and disclosure control. It is aimed at owners of sensitive datasets such as public organisations, research groups, and individual researchers who want to improve the accessibility of their data for research and reproducibility by others. The goal of `metasyn` is to make it easy for data owners to share the structure and an approximation of the content of their data with others while keeping privacy concerns to a minimum.

With this goal in mind, `metasyn` distinguishes itself from existing software for generating synthetic data (e.g., Nowok et al., 2016; Ping et al., 2017; Templ et al., 2017) by strictly limiting the statistical information from the real data in the produced synthetic data. This choice enables the software to generate synthetic data with **privacy and disclosure guarantees** through a plug-in system. Moreover, our system provides an **auditable and editable intermediate representation** in the form of a human- and machine-readable `.json` metadata file from which new data can be synthesized.

Through our focus on privacy and transparency, metasyn explicitly avoids generating synthetic data with high analytical validity. The data generated by our system is realistic in terms of data structure and plausible in terms of values for each variable — the "augmented plausible" category of synthetic data (Bates et al., 2019) — but multivariate relations or conditional patterns not learnt from the real data. This has implications for how this synthetic data can be used: not for statistical analysis and inference, but rather for initial exploration, analysis script development, and communication outside the data owner's institution. In the intended use case, an external researcher can make use of the synthetic data to assess the feasibility of their intended research before making the (often time-consuming) step of requesting access to the sensitive source data for the final analysis.

As mentioned before, the privacy capacities of metasyn are extensible through a plug-in system, recognizing that different data owners have different needs and definitions of privacy. A data owner can define under which conditions they would accept open distribution of their synthetic data — be it based on differential privacy (Dwork, 2006), statistical disclosure control (Wolf, 2012), k-anonymity (Sweeney, 2002), or another specific definition of privacy. As part of the initial release of metasyn, we publish a plugin following the disclosure control guidelines from Eurostat (Bond et al., 2015).

Software features

At its core, metasyn is designed for three functions, which are briefly described in this section:

1. **Estimation**: Automatically select univariate distributions and fit them to a properly formatted tabular dataset, optionally with additional privacy guarantees.
2. **(De)serialization**: Create an intermediate representation of the fitted model for auditing, editing, and exporting.
3. **Generation**: Generate new synthetic datasets based on the fitted model or its serialized representation.

Estimation

The generative model for multivariate datasets in metasyn makes the simplifying assumption of marginal independence: each column is considered separately, just as is done in e.g., naïve Bayes classifiers (Hastie et al., 2009). Formally, this leads to the following generative model for the K-variate data $\mathbf{x}$:

$$p(\mathbf{x}) = \prod_{k=1}^{K} p(x_k) \tag{1}$$

There are many advantages to this naïve approach when compared to more advanced generative models: it is transparent and explainable, it is able to flexibly handle data of mixed types, and it is computationally scalable to high-dimensional datasets. As mentioned before, the tradeoff is the limited analytical validity when the independence assumption does not hold: in the synthetic data, the expected value of correlations, regression parameters, and other measures of association is 0.

Model estimation starts with an appropriately pre-processed data frame. For metasyn, this means the data frame is tidy (Wickham, 2014), each column has the correct data type, and missing data are represented by a missing value. Internally, our software uses the polars data frame library (Vink et al., 2024), as it is performant, has consistent data types, and natively supports missing data (i.e., null values). A simple example source table could look like this (note that categorical data has the appropriate cat data type, not str):

```
┌─────┬────────┬─────┬────────┬──────────┐
│ ID  ┆ fruits ┆ B   ┆ cars   ┆ optional │
```

```
│ ---  ┆ ---    ┆ ---  ┆ ---    ┆ ---         │
│ i64  ┆ cat    ┆ i64  ┆ cat    ┆ i64         │
╞══════╪════════╪══════╪════════╪═════════════╡
│ 1    ┆ banana ┆ 5    ┆ beetle ┆ 28          │
│ 2    ┆ banana ┆ 4    ┆ audi   ┆ 300         │
│ 3    ┆ apple  ┆ 3    ┆ beetle ┆ null        │
│ 4    ┆ apple  ┆ 2    ┆ beetle ┆ 2           │
│ 5    ┆ banana ┆ 1    ┆ beetle ┆ -30         │
└──────┴────────┴──────┴────────┴─────────────┘
```

For each data type supported by `metasyn`, there is a set of candidate distributions that can be fitted to that data type (see Table Table 1). To estimate the generative model of Equation Equation 1, for each variable the software fits all compatible candidate distributions — by default with maximum likelihood estimation — and then selects the one with the lowest BIC (Neath & Cavanaugh, 2012). For distributions where this is not possible, such as those for the string data type, a pseudo-BIC is created that trades off fit and complexity of the underlying models.

Table 1: Candidate distributions associated with data types in the core `metasyn` package.

Variable type	Example	Candidate distributions
categorical	yes/no, country	Categorical (Multinoulli), Constant
continuous	1.0, 2.1, …	Uniform, Normal, LogNormal, TruncatedNormal, Exponential, Constant
discrete	1, 2, …	Poisson, Uniform, Normal, TruncatedNormal, Categorical, Constant
string	A108, C122, some words	Regex, Categorical, Faker, FreeText, Constant
date/time	2021-01-13, 01:40:12	Uniform, Constant

From this table, the string distributions deserve special attention as they are not commonly encountered as probability distributions. Regex (regular expression) inference is performed on structured strings using the companion package RegexModel. It is able to automatically detect structure such as room numbers (A108, C122, B109), e-mail addresses, websites, and more, which it summarizes using a probabilistic variant of regular expressions. Another option, should Regex inference fail for lack of structure, is to detect the language (using lingua) and randomly pick words from that language. We call this approach FreeText. The final alternative is for the data owner to specify that a certain variable should be synthesized using the popular Faker package, which can generate specific data types such as localized addresses.

Generative model estimation with `metasyn` can be performed as follows:

```python
from metasyn import MetaFrame, VarSpec

# "ID" column is the primary key,
# thus should generate unique values.
# "B" column is not, despite unique
# values in the dataframe
specs = [
  VarSpec("ID", unique=True),
  VarSpec("B", unique=False),
]
```

```python
# create metaframe
mf = MetaFrame.fit_dataframe(df, var_specs=specs)
```

Serialization and deserialization

After a fitted model object is created, `metasyn` allows it to be transparently stored in a human- and machine-readable `.json` file. This file can be considered as metadata: it contains dataset-level descriptive information as well as variable-level information. The metadata format has a specific structure, which we call the `generative metadata format`, or `gmf`. The header contains the following dataset-level information:

```json
"n_rows": 5,
"n_columns": 5,
"provenance": {
    "created by": {
        "name": "metasyn",
        "version": "1.0.1"
    },
    "creation time": "2024-08-07T12:20:36.022017"
}
```

Then, for each variable the `gmf` file contains information the name, the data type, the proportion of missing values, and the distribution fitted on the data. For example, a table column containing different types of fruits could result in the following `.json`:

```json
{
  "name": "fruits",
  "type": "categorical",
  "dtype": "Categorical(ordering='physical')",
  "prop_missing": 0.0,
  "distribution": {
    "implements": "core.multinoulli",
    "version": "1.0",
    "provenance": "builtin",
    "class_name": "MultinoulliDistribution",
    "unique": false,
    "parameters": {
      "labels": ["apple", "banana"],
      "probs": [0.4, 0.6]
    }
  },
  "creation_method": { "created_by": "metasyn" }
}
```

There are several advantages to creating such a serialized representation. First, it can be audited: the data owner can see exactly what information from the real data is made public through exporting the synthetic data, namely, the parameters of the distribution. Second, the file can be edited. For example, if a data owner thinks some of the labels of the "fruit" column contain sensitive information, these can simply be pseudonymized in the metadata file. Third, after exporting this file, an unlimited number of synthetic records can be created without incurring additional privacy risks, because the original data is no longer part of the synthetization process.

Serialization and deserialization with `metasyn` can be performed as follows:

```python
# write a fitted MetaFrame to json
mf.export("fruits_gmf.json")
```

```python
# then, audit and export json from secure environment

# outside the secure environment, load json into MetaFrame
mf_out = MetaFrame.from_json("fruits_gmf.json")
```

Data generation

After creating the fitted model object, either from the original data or by deserializing a model object from a `.json` file, new data can be generated by the object. For each variable in the model object, the software randomly samples from the fitted distribution to create a synthetic version of the data. Data generation (or synthetization) in `metasyn` can be performed as follows:

```python
from metasyn import MetaFrame

# load json into a metadataset object
mf = MetaFrame.from_json("metasyn_example.json")

# create a fake dataset
df_syn = mf.synthesize(10)
```

This may result in the following `polars` data frame[1]. Note that missing values in the `optional` column are appropriately reproduced as well, courtesy of the "prop_missing" entry in the metadata format.

```
shape: (10, 5)
```

ID --- i64	fruits --- cat	B --- i64	cars --- cat	optional --- i64
1	banana	4	beetle	null
2	banana	3	audi	null
3	banana	1	beetle	223
4	banana	0	beetle	258
…	…	…	…	…
7	banana	3	beetle	298
8	banana	2	beetle	67
9	banana	4	beetle	-30
10	banana	2	beetle	172

Plug-ins and automatic privacy

In addition to the core features described above, the `metasyn` package allows for plug-ins: add-on packages that alter the behaviour of the parameter estimation. Through this system, privacy guarantees can be built into `metasyn` (privacy plugin template) and additional distributions can be supported (distribution plugin template). For example, a plugin package called `metasyn-disclosure-control` implements the disclosure control output guidelines from Eurostat (Bond et al., 2015) by re-implementing the `fit()` method of the candidate distributions shown in Table Table 1 to include a micro-aggregation step. In this way, information transfer from the sensitive real data to the synthetic public data can be further reduced.

This plug-in system is user-friendly: the user only needs to `pip install` the package and then `metasyn` can automatically find it to make the methods accessible:

[1] This `polars` dataframe can be easily converted to a `pandas` dataframe using `df_syn.to_pandas()`

```python
from metasyn import MetaDataset
from metasyncontrib.disclosure import DisclosurePrivacy

mf = MetaFrame.fit_dataframe(df, privacy=DisclosurePrivacy())
```

Conclusion

Synthetic data is a valuable tool for communicating about sensitive datasets. In this work, we have presented the software metasyn, which allows data owners to generate a synthetic version of their sensitive tabular data with a focus on privacy and transparency. Unlike existing tools for generating synthetic data, we choose to aim for low analytic validity to enable strong privacy guarantees: the underlying model makes a simplifying independence assumption, resulting in few parameters and thus a very limited information transfer. This approach additionally allows for disclosure guarantees through a plug-in system.

Further documentation and examples can be found on metasyn.readthedocs.io.

Acknowledgements

This research was conducted in whole or in part using ODISSEI, the Open Data Infrastructure for Social Science and Economic Innovations (https://ror.org/03m8v6t10)

The metasyn project is supported by the FAIR Research IT Innovation Fund of Utrecht University (March 2023)

References

Bates, A., Spakulová, I., Dove, I., & Mealor, A. (2019). *ONS methodology working paper series number 16—synthetic data pilot.*

Bond, S., Brandt, M., & Wolf, P. de. (2015). *Guidelines for output checking. eurostat.*

Dwork, C. (2006). Differential privacy. *International Colloquium on Automata, Languages, and Programming*, 1–12.

Hastie, T., Tibshirani, R., Friedman, J. H., & Friedman, J. H. (2009). *The elements of statistical learning: Data mining, inference, and prediction* (Vol. 2). Springer.

Neath, A. A., & Cavanaugh, J. E. (2012). The bayesian information criterion: Background, derivation, and applications. *Wiley Interdisciplinary Reviews: Computational Statistics*, *4*(2), 199–203.

Nowok, B., Raab, G. M., & Dibben, C. (2016). Synthpop: Bespoke creation of synthetic data in r. *Journal of Statistical Software*, *74*, 1–26.

Ping, H., Stoyanovich, J., & Howe, B. (2017). Datasynthesizer: Privacy-preserving synthetic datasets. *Proceedings of the 29th International Conference on Scientific and Statistical Database Management*, 1–5.

Sweeney, L. (2002). K-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, *10*(05), 557–570.

Templ, M., Meindl, B., Kowarik, A., & Dupriez, O. (2017). Simulation of synthetic complex data: The r package simPop. *Journal of Statistical Software*, *79*(10), 1–38.

van Kesteren, E.-J. (2024). To democratize research with sensitive data, we should make synthetic data more accessible. *arXiv Preprint arXiv:2404.17271*.

Vink, R., Gooijer, S. de, Beedie, A., Gorelli, M. E., Guo, W., Zundert, J. van, Peters, O., Hulselmans, G., nameexhaustion, Grinstead, C., Marshall, Burghoorn, G., chielP, Turner-Trauring, I., Santamaria, M., Heres, D., Mitchell, L., Magarick, J., ibENPC, … Brannigan, L. (2024). *Pola-rs/polars: Python polars* (py-1.4.1). Zenodo. https://doi.org/10.5281/zenodo.7697217

Wickham, H. (2014). Tidy data. *Journal of Statistical Software*, *59*(10), 1–23. https://doi.org/10.18637/jss.v059.i10

Wolf, P.-P. de. (2012). *Statistical disclosure control*. Wiley & Sons, Chichester.