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Published: unpublished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. `Metasyn` explicitly avoids generating synthetic data with high analytical validity; instead, the synthetic data has realistic structure and plausible values, but multivariate relations are omitted ("augmented plausible synthetic data"; (Bates et al., 2019)). Moreover, our system provides an **auditable**

The Journal of Open Source Software

and editable intermediate representation in the form of a `.json` metadata file from which new data can be synthesized.

These choices enable the software to generate synthetic data with **privacy and disclosure guarantees** 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 (Hundepool et al., 2012), k-anonymity (Sweeney, 2002), or another specific definition of privacy. As part of the initial release of `metasyn`, we publish a plug-in following the disclosure control guidelines from Eurostat (Bond et al., 2015).

Software features

At its core, `metasyn` has three main functions:

1. **Estimation**: Automatically select 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 a fitted model.

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.

Model estimation starts with an appropriately pre-processed data frame, meaning it 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, a set of candidate distributions is fitted (see Table Table 1), and then `metasyn` selects the one with the lowest BIC (Neath & Cavanaugh, 2012). For distributions where BIC computation is impossible (e.g., 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. The FreeText distribution detects the language (using lingua) and randomly picks words from that language. The Faker distribution can generate specific data types such as localized addresses, when pre-specified by the user.

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),
]

# 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 header contains the following dataset-level information:

```json
"n_rows": 5,
"n_columns": 5,
"provenance": {
    "created by": {
        "name": "metasyn",
        "version": "1.0.2"
```

```
    },
    "creation time": "2024-08-12T12:20:36.022017"
}
```

Then, for each variable the file contains descriptive and statistical information, as in the following example:

```
{
  "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. It can be manually audited, edited, and after exporting this file, an unlimited number of synthetic records can be created without incurring additional privacy risks.

Serialization and deserialization with metasyn can be performed as follows:

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

# then, audit and export json from secure environment

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

Data generation

For each variable in a fitted or deserialized model object, metasyn can randomly sample synthetic datapoints. 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.

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

The Journal of Open Source Software

```
shape: (10, 5)
┌─────┬────────┬─────┬────────┬──────────┐
│ ID  ┆ fruits ┆ B   ┆ cars   ┆ optional │
│ --- ┆ ---    ┆ --- ┆ ---    ┆ ---      │
│ i64 ┆ cat    ┆ i64 ┆ cat    ┆ 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 plug-in template) and additional distributions can be supported (distribution plug-in template). The `metasyn-disclosure-control` plug-in implements output guidelines from Eurostat (Bond et al., 2015) by including a micro-aggregation step in the `fit()` method. In this way, information transfer from the sensitive real data to the synthetic public data can be further limited. Disclosure control is done as follows:

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

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


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*. https://www.ons.gov.uk/methodology/methodologicalpublications/generalmethodology/onsworkingpaperseries/onsmethodologyworkingpaperseriesnumber16syntheticdatapilot

Bond, S., Brandt, M., & Wolf, P. de. (2015). *Guidelines for the checking of output based on microdata research*. Eurostat. https://web.archive.org/web/20160408145718/http://dwbproject.org/export/sites/default/about/public_deliveraples/dwb_d11-8_synthetic-data_cta-ecta_output-checking-guidelines_final-reports.zip

Dwork, C. (2006). Differential privacy. *International Colloquium on Automata, Languages, and Programming*, 1–12. https://doi.org/10.1007/11787006_1

Hastie, T., Tibshirani, R., Friedman, J. H., & Friedman, J. H. (2009). *The elements of statistical learning: Data mining, inference, and prediction* (Vol. 2). Springer. https://doi.org/10.1007/978-0-387-84858-7

Hundepool, A., Domingo-Ferrer, J., Franconi, L., Giessing, S., Nordholt, E. S., Spicer, K., & De Wolf, P.-P. (2012). *Statistical disclosure control*. Wiley & Sons, Chichester. https://doi.org/10.1002/9781118348239

Neath, A. A., & Cavanaugh, J. E. (2012). The bayesian information criterion: Background, derivation, and applications. *Wiley Interdisciplinary Reviews: Computational Statistics*, *4*(2), 199–203. https://doi.org/10.1002/wics.199

Nowok, B., Raab, G. M., & Dibben, C. (2016). Synthpop: Bespoke creation of synthetic data in r. *Journal of Statistical Software*, *74*, 1–26. https://doi.org/10.18637/jss.v074.i11

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. https://doi.org/10.1145/3085504.3091117

Sweeney, L. (2002). K-anonymity: A model for protecting privacy. *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, *10*(05), 557–570. https://doi.org/10.1142/S0218488502001648

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. https://doi.org/10.18637/jss.v079.i10

van Kesteren, E.-J. (2024). To democratize research with sensitive data, we should make synthetic data more accessible. *arXiv Preprint arXiv:2404.17271*. https://doi.org/10.48550/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