PyDPLib: Python Differential Privacy Library for Private Medical Data Analytics

Sana Imtiaz*§, Philipp Matthies†, Francisco Pinto†, Màtè Maros‡, Holger Wenz‡,
Ramin Sadre§, Vladimir Vlassov*
*KTH Royal Institute of Technology, Stockholm, Sweden
†Smart Reporting GmbH, Munich, Germany
‡Medical Faculty Mannheim, Heidelberg University, Mannheim, Germany
§Université catholique de Louvain, Louvain-la-Neuve, Belgium
Email: *{sanaim,vladv}@kth.se, †{p.matthies,f.pinto}@smart-reporting.com,
‡{mate.maros@medma.uni-heidelberg.de, holger.wenz@umm.de}, §ramin.sadre@uclouvain.be

2021 IEEE International Conference on Digital Health (ICDH) | 978-1-6654-1685-6/21/$31.00 ©2021 IEEE | DOI: 10.1109/ICDH52753.2021.00034

***Abstract*—Pharmaceutical and medical technology companies accessing real-world medical data are not interested in personally identifiable data but rather in cohort data such as statistical aggregates, patterns, and trends. These companies cooperate with medical institutions that collect medical data and want to share it but they need to protect the privacy of individuals on the shared data. We present PyDPLib, a Python Differential Privacy library for private medical data analytics. We illustrate an application of differential privacy using PyDPLib in our platform for visualizing private statistics on a database of prostate cancer patients. Our experimental results show that PyDPLib allows creating statistical data plots without compromising patients' privacy while preserving underlying data distributions. Even though PyDPLib has been developed to be used in our platform for reporting the radiological examinations and procedures, it is general enough to be used to provide differential privacy on data in any data analytics and visualization platform, service or application.**

***Index Terms*—Differential privacy, python library, private visual statistics, electronic data capture, prostate cancer dataset**

1. Introduction

Pharmaceutical and medical technology companies are interested in accessing real-world medical data for driving their business models and decisions, which aids in the provision of continuously improving personalized services. However, these companies are not interested in personally identifiable data or protected health information (PHI) of the individuals but rather the cohort data. These companies cooperate with medical institutions who collect the medical data and want to share it, but they need to protect the privacy of the participating individuals.

Sharing data with PHI raises privacy concerns due to the threat of misuse or re-identification. Data protection laws like the EU's General Data Protection Regulation (GDPR) ensure higher public trust in data sharing, and enforce informed use of collected user data by the companies. GDPR enforces *privacy-by-design*, which means that the technology is designed with data privacy preservation [1]. One solution is to use information flow control [2] and design privacy-centric platforms that possess data flow models with respective permissions for every type of user to ensure user privacy and transparent accountability. However, a privacy-centric platform alone cannot solve the problem of sharing medical data with medical technology companies.

Data anonymization is commonly employed for privacy preservation in the health care industry, although anonymization alone does not guarantee sufficient privacy preservation due to the risk of re-identification and attribute disclosure [3]. A possible solution could be the use of realistic synthetic datasets for enhanced user privacy with reduced risk of re-identification. However, generation of mass-scale synthetic datasets might not always be reasonable due to: 1) the need for continuous integration and generation of new data, and 2) fixed privacy preservation levels for all parties. Mechanisms for secure multiparty computation [4] are also proposed in literature for collectively computing private statistics without sharing the sensitive data. However, these solutions might be computationally expensive and may not always cater to solving the problem of sharing statistical patterns with third parties.

Differential privacy (DP) [5] allows computing statistical patterns in a dataset while withholding information about individuals in the dataset. DP provides provable privacy guarantees and ensures the minimal impact of participation of a single individual in a database, by adding calculated noise depending on the queried statistical patterns. DP can also cater to flexible noise addition based on the user type and privilege. Numerous DP libraries exist in literature such as: Google's DP library [6] and its python wrapper *PyDP* [7], *Diffprivlib* IBM's DP library [8] and open-source implementation [9], *dp-stats* library for differentially private statistics and machine learning algorithms [10], mechanisms for DP histogram publication [11], and *OpenDP* collection of tools for statistical analysis of sensitive private data [12]. However, to the best of our knowledge, most of these tools are end-to-end solutions for computing DP statistical measurements, which makes it difficult to integrate them

978-1-6654-1685-6/21/$31.00 ©2021 IEEE
DOI 10.1109/ICDH52753.2021.00034

in an already-existing system without making significant changes to the system internals. Using these DP tools also requires an understanding of privacy preservation techniques which might be difficult for professionals in the medical domain. Moreover, these tools cannot often be used in conjunction with data visualization software, which makes it hard to integrate them in end-to-end systems that only require implementation of a privacy preservation layer.

We have developed PyDPLib, a platform-independent differential privacy library in Python. We have also developed a privacy-centric platform for structured data collection that employs PyDPLib, and we show our results on a database of prostate cancer patients. Our reporting software *SmartReports* and a PI-RADS [13] template was used to collect this dataset. Moreover, by using a software with interactive visualization such as plotly [14], PyDPLib is used to create interactive and private statistical plots.

The main contributions of this paper are as follows.

- We have developed a differential privacy library PyDPLib for computing private statistics with medical data as a use case.
- PyDPLib provides different levels of privacy preservation with noise addition guarantees, depending on the user type and privilege.
- Our reporting software *SmartReports* uses information flow control, and when used with PyDPLib, ensures 2-layer privacy preservation.
- PyDPLib supports a variety of statistical operations on continuous and discrete data, and can be used in conjunction with any data visualization software.
- We have developed a template for structured clinical data collection and curated a human expert labeled dataset based on our template. We demonstrate our results for PyDPLib on this dataset.

2. Preliminaries

Differential Privacy. Differential Privacy offers a provable and quantifiable amount of privacy protection by means of a privacy-loss budget ε. The most popular mathematical tool used to express DP is as follows [15]: A randomized algorithm A is ε-differentially private (ε-DP), if for the set of all datasets D and D' that differ on at most one row (i.e. the data of one individual), and any subset $S \subseteq range(A)$,

$$Pr[A(D) \in S] \leq e^{\varepsilon} Pr[A(D') \in S]. \quad (1)$$

The loss of privacy is quantified using ε, which is used to determine the noise addition for ensuring DP. Achieving higher levels of privacy preservation (small ε) involves adding more noise to the data which leads to a decrease in the output accuracy of the algorithm and vise versa. Therefore, a trade-off must be found between keeping the information private and achieving meaningful results, depending on the data and nature of the randomized algorithm. The post-processing theorem in DP [5] states:

Theorem 1. *If a mechanism M satisfies ε-DP, and g be any function, then $g(M(X))$ also satisfies ε-DP.*

DP mechanisms leverage this theorem as they mostly focus on perturbing the distribution by noise addition. This perturbation is done either on the input data points or on the output of the querying statistical function. Applying the post-processing theorem, any data drawn from a noisy distribution that satisfies DP will also be ε-DP. We leverage this theorem to provide differentially private visual representations of statistics on sensitive PHI.

Laplacian Differential Privacy. The Laplacian mechanism is one of the most popular noise addition mechanisms in DP [16]. A standard approach is adding random noise with the Laplacian distribution proportional to the sensitivity S_f of the queried function to ensure DP-queries. Random noise is drawn from a Laplacian distribution with mean 0 and variance S_f/ε to achieve ε-differential privacy [5].

Sensitivity. According to [5], sensitivity S_f captures the magnitude by which a single individual's data can change the output of the function f in the worst case. It helps to quantify the uncertainty in the response that needs to be introduced in order to hide the participation of a single individual. Mathematically, for any function f over the set of all datasets D and D' that differ on at most one row,

$$S_f = \max||f(D) - f(D')|| \quad (2)$$

where $||.||$ denotes Manhattan distance or $L1$ norm [15]. We use Laplacian differential privacy by adding noise directly to the data records (with aggregated or non aggregated data points) in order to ensure easy integration into any data analytics system using any data visualization software. Each data point x is individually noised as x' by picking a random noise sample from a Laplacian distribution given by:

$$x' = x + Lap(0, S_f/\varepsilon) \quad (3)$$

where S_f represents the sensitivity of the statistical query. Sensitivity of supported statistical operations is elaborated in Section 4.5. We now describe the structured data collection as well as our collection template.

3. Structured Clinical Data Collection

SmartReports Reporting Software. SmartReports enables structured reporting of radiological examinations and procedures evaluated using flexible decision trees. Furthermore, the underlying data is machine-readable and the generated reports are exported to our electronic data capture (EDC) software, and accessible for processing by PyDPLib.

Electronic Data Capture. Our EDC web application is a privacy-centric platform implementing *privacy-by-design* principles of the GDPR. State-of-the-art data protection measures have been adopted based on the Open Web Application Security Project (OWASP) Top Ten list, which represents a broad consensus about the most critical security risks. The clinical report data is imported and stored in this application and the DP analyses are generated server-side and can be accessed by users based on their data flow policy.

PI-RADS Template. Clinical report data is captured for prostate cancer patients who underwent MR-imaging using

the Prostate Imaging Reporting & Data System (PI-RADS) v1 [13]. The PI-RADS v1 definition of clinically significant cancer (based on pathology/histology) is: Gleason score ≥ 7 (including $3+4$ with prominent, but not predominant Gleason 4 component) and/or volume $\geq 0.5ml$ and/or extra prostatic extension (EPE). Table 1 shows the 5-point scale for the PI-RADS score based on the likelihood of combination of mpMRI findings correlated with the presence of a clinically significant cancer. It is applied to each lesion. In addition to the PI-RADS score, information regarding age, Gleason score, prostatitis, benign prostatic hyperplasia (BPH), number of lesions and therapy is also collected.

TABLE 1. 5-POINT SCALE OF PI-RADS ASSESSMENT SCORE

Score	Assessment
1	Very low (clinically significant cancer is highly unlikely)
2	Low (clinically significant cancer is unlikely)
3	Intermediate (clinically significant cancer is equivocal)
4	High (clinically significant cancer is likely)
5	Very high (clinically significant cancer is highly likely)

4. Differential Privacy Library

PyDPLib provides ε-DP with Laplacian noise addition. The most common usages of DP mechanisms add DP-noise to the output of the computed statistical measures. However, since our platform provides a visual representation of both the raw and aggregated statistics, PyDPLib adds noise to the input data points which may or may not be aggregated. PyDPLib uses *Numpy* for dependencies, and supports a wide range of statistical queries and data types.

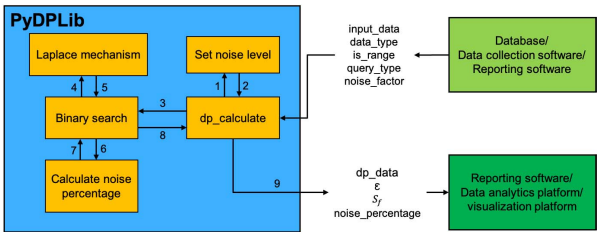


Figure 1. PyDPLib methods and interaction with other modules

Figure 1 shows a high-level diagram displaying inputs/outputs and the interactions between different methods in PyDPLib. Input from the database or reporting system is passed to $dp\_calculate$ method. The $dp\_calculate$ first sets the noise level based on the input `noise_factor`. Afterwards, $dp\_calculate$ calls the $Binary\ Search$ method to find a suitable ε for the desired noise settings and S_f, by using the $Laplace\ mechanism$ for noise addition. This binary search is done in recursion until we find an ε that gives us the appropriate noise percentage. Finally, ε is used to noise the input data and PyDPLib returns the differentially private data dp_data, ε, S_f as well as the percentage noise.

4.1. Threat Model and Types of Users

In our approach, users are divided into three types:

Owner: Hospital that own the data, and possess the full right to information disclosure;

Collaborator: Collaborating hospitals that access the data for research and other collaborative services;

Third party: includes commercial partners that need access to the medical data without information disclosure.

For our threat model, the data owners are trusted parties. They are also data curators and assign the data flow policies. The collaborators are trusted but can be *honest-but-curious* parties in the worst case. Therefore, we recommend using privacy preservation for collaborators. Third parties should not have access to the data but could visualize the underlying information to make general observations on the data patterns. In summary, third parties should not be able to extract the information of a single user based on the visualizations.

4.2. Setting the Appropriate Noise Factor

Depending on the type of user as mentioned earlier, PyDPLib selects respective noise addition settings. The Laplacian noise addition is controlled by the `noise_factor`, which offers three settings for noise addition to input data:

- 0: **low**. $0-5\%$ noise addition. Suitable for data owners (privileged users).
- 1: **medium**. $5-10\%$ noise addition. Suitable for collaborators requiring a low loss of accuracy.
- 2: **high**. $10-20\%$ noise addition. Suitable for third parties requiring statistics without data disclosure.

PyDPLib sets the appropriate noise level based on `noise_factor`. Afterwards, PyDPLib selects the appropriate ε for the Laplacian noise mechanism depending on the range, data type, and sensitivity of the statistical query S_f. In case of `is_range` or categorical data, the data is first fit to the input range and afterwards, noise percentage is calculated. We use a binary search algorithm for ε selection.

4.3. Binary Search Algorithm for ε Selection

We perform a binary search for ε based on the `noise_factor` and the sensitivity S_f of the statistical query. low_{idx} and hi_{idx} are used to query low and high indexes in the epsilons array respectively. The binary search algorithm also takes additional parameters for determining data fitting mechanisms and noise calculation: `is_bool` and `is_range` are used for data fitting, and $noise\_level$ indicates the noise margins for selected noise factor. Algorithm 1 shows the complete Binary Search algorithm. Once an appropriate ε is found, the data points are noised according to S_f of the selected statistical query, and forwarded to the output along with selected value of ε.

Algorithm 1: Binary Search for ε

Data: epsilons, low_{idx}, hi_{idx}, $noise\_level$, $input\_data$, is_bool, is_range, S_f
Result: ε
Initialize sum to 0. Set $noise\_lvl\_low$ and $noise\_lvl\_high$ depending on $noise\_level$.

```
if (hi_idx ≥ low_idx) then
    mid = (hi_idx + low_idx)/2
    for 5 times do
        for datum in input_data do
            Calculate noised data using Laplacian
              noise with S_f and ε = epsilons[mid]
        if is_range then fit noised data to range;
        Calculate percentage noise and add to sum.
    Set noise_mid to the average of sum.
    if noise_lvl_low ≤ noise_mid ≤
      noise_lvl_high then
        return ε = epsilons[mid]; /* found */
    else if noise_mid < noise_lvl_low then
        Repeat Binary Search in [low_idx, mid]
    else
        Repeat Binary Search in [mid, hi_idx]
else
    return 0 ;                    /* not found */
```

4.4. Supported Data Types

PyDPlib supports a vast variety of data formats as well as continuous and discrete (or categorical) data. The categorical variables must be mapped to a numeric format. `data_type` could be char, string, int, numpy.int, float, numpy.float, bool, and numpy.bool. `is_range` (True/False) specifies if data values lie within a specific range. The output data is fitted to the range of input data after noise addition.

Fitting Data to Range. PyDPLib features the `fit_data_to_range` and `data_range` methods to handle categorical variables or range-bound data. The noising mechanism must return valid values regardless of the chosen noising levels. When `is_range` is `True`, the range of valid data points is inferred from the input data (non-noised) using the method `data_range`. This method returns all the unique data points as `range_values`. Afterwards, `fit_data_to_range` takes the following input parameters.

- `data`: noised data to fit to range;
- `range_values`: unique and valid data point values inferred from input data;
- `data_type`: could be int, numpy.int, float, numpy.float, bool, and numpy.bool.

The fitted result is interpreted as:

$$x'_{fitted} = \frac{(max_r - min_r)(x' - min_{data})}{max_{data} - min_{data}} + min_r \quad (4)$$

where max_r and min_r are the maximum and minimum input values inferred from `range_values`, and x' is a noised data point in `data` with maximum and minimum noised values given by max_{data} and min_{data} respectively.

Continuous data is represented by float or numpy.float, and the fitting mechanism is as in Equation 4. `fitted_data` is used to compute the noise percentage as compared to the input data. For boolean data, the noised data is fitted as: `True` if noised data > 0.5, else `False`. This is in agreement with the randomized response for DP in boolean attributes as observed in [17]. An alternative approach for noising the boolean values independently of the statistical query is to use the coin flip algorithm, where the reported boolean value is dependent on the outcome of the successive coin flips. However, repeated queries may expose the original probabilities of the underlying data.

4.5. Supported Statistical Queries and Sensitivity

PyDPLib offers four statistical `query_type`: 1) count or histogram, 2) average or mean, 3) median, and 4) variance. The noise addition mechanism (Equation 3) uses sensitivity S_f of the statistical operation to compute the margin of added noise, and each data point is individually noised. If the input data is categorical or range-bound, the noised data point is then fitted to the input range accordingly. Sensitivity of statistical queries for calibrating the noise addition is a well studied problem in literature [5], [15], [18], [19]. We now describe the S_f of each statistical query in detail.

Count or Histogram. Count or histogram queries are the simplest of statistical operations as the addition or deletion of a single individual or record can change the count by at most 1 [5]. According to Equation 2, the sensitivity for count or histogram query is given as $S_{f(hist)} = 1$. Therefore, noise is sampled from $Lap(0, S_{f(hist)}/\varepsilon)$ distribution.

Average or Mean. For computing the sensitivity of average or mean, we need to compute the upper and lower bounds of the input data. For n input data points with lower bound a and upper bound b, the sensitivity is given by [20]:

$$S_{f(avg)} = |b - a|/n \quad (5)$$

Each data point is individually noised with a randomly picked sample from $Lap(0, S_{f(avg)}/\varepsilon)$ distribution, and a noisy average is computed by the data analytics software.

Median. Although the median and mean queries are statistically different, they exhibit the same sensitivity. The sensitivity of median query on an input data with n entries with lower bound a and upper bound b is given by [18]:

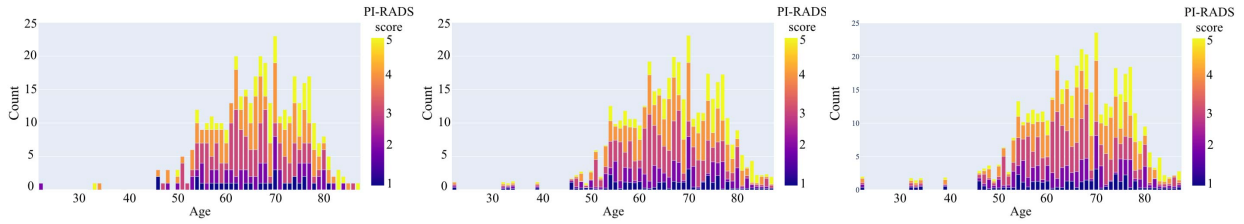
$$S_{f(med)} = |b - a|/n \quad (6)$$

x' is computed by adding a randomly picked sample from $Lap(0, S_{f(med)}/\varepsilon)$ distribution to x data point.

Variance. Sensitivity of variance for input data with n entries with lower bound a and upper bound b is given by:

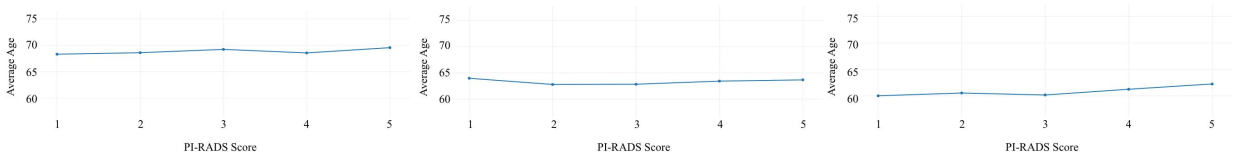
$$S_{f(var)} = (b - a)^2/n \quad (7)$$

Here x' is computed by adding a randomly picked sample from $Lap(0, S_{f(var)}/\varepsilon)$ distribution to x data point. The noised data is then fitted to the input range, if applicable.



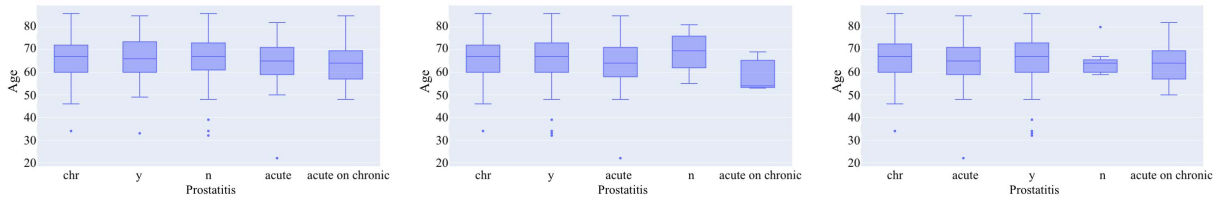
a) Low b) Medium c) High

Figure 2. Histograms for patient age vs. PI-RADS scores with different `noise_factor` settings (low – high). Example for x-versus-y plot with noised y-axis, `query_type` = 1, `is_range = True`, and `data_type = int`.



a) Low b) Medium c) High

Figure 3. PI-RADS scores vs. average patient age with different `noise_factor` settings (low – high). Example for x-versus-y plot with noised x-axis, `query_type` = 2, `is_range = True`, and `data_type = float`.



a) Low b) Medium c) High

Figure 4. Prostatitis vs. median age with different `noise_factor` settings (low – high). Example for x-versus-y plot with noised x-axis, `query_type` = 3, `is_range = True`, and `data_type = int`.

5. Experiments and Results

920 prostate cancer patient reports based on the PI-RADS template have been collected within the years 2017-18 by board-certified radiologists, and manually labelled by domain experts. Some of the collected and manually extracted attributes are shown in Table 2. We illustrate a variety of private statistical plots on the PI-RADS dataset using PyDPLib and visualized with Plotly [14]. Since PyDPLib perturbs the data distribution according to the data type, range and desired statistical query; and is independent of the used plotting mechanism, any data analytics or visualization software can be used to compute and display these plots.

`query_type` 1: Histograms. Figure 5 shows the histogram of the patient ages. Due to privacy concerns, we only illustrate the histogram in high noise settings. `is_range` is True as age can only be positive. Ages are directly noised using PyDPLib and displayed as a histogram. An alternative approach would be to compute the counts first and pass them to PyDPLib, as we discuss for the following histograms.

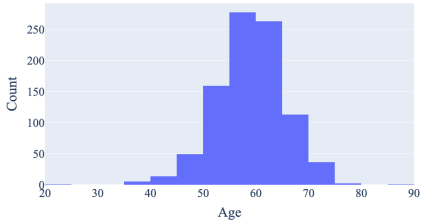


Figure 5. Age histogram with `noise_factor = 2` (high). Example for `query_type` = 1, `is_range = True`, and `data_type = float`.

Figure 2 shows the histogram plots for age versus PI-RADS score in low, medium and high privacy settings. Absolute ages are used, and `is_range = True` as count can only be positive. Moreover, PI-RADS score can only have valid values in the range $1-5$ as described in Section 3. The counts for PI-RADS score for each age are computed on raw data, and passed to the PyDPLib. This is an example of

TABLE 2. PI-RADS DATASET

Attribute	Type	Data type
Anonym_ID	numerical	int
RequestRadiologyDepartmentCode	categorical	string
ProcedureValidatingPhysicianName	free text	string
LastModified	categorical	date
Age	numerical	float
Year	numerical	int
Gleason score	categorical	int
PCA_no_PIRADS_defined	categorical	bool
Follow up	categorical	string
PIRADS score	categorical	int
Number_of_Lesions	numerical	int/string
Prostatitis	categorical	string
BPH	categorical	bool
Primary_TU_LocalRecurrence_or_Progress	categorical	string
Therapy	categorical	string
TURP surgery	categorical	string

x-versus-y plot, where only y-axis is noised. As can be seen, the overall distribution of samples is maintained in all noise settings. However, the individual frequencies recorded in the histograms are noised differently depending on the selected noise factor for increased user privacy. This makes it hard to single out an individual from the visual plot. Prostate cancer occurrence becomes high in ages $50-80$, with the highest levels of PI-RADS scores in $60-70$'s. This behaviour is generally observed regardless of the noise factor.

`query_type` 2: Average or mean. Figure 3 shows the PI-RADS score versus the average patient age in low, medium and high privacy settings. Ages are interpreted as continuous data with float data type, and `is_range` is set to `True` since the ages can only be positive. Patient ages are individually noised by PyDPLib and noisy average is computed. This is an example of x-versus-y plot where only x-axis is noised. As can be seen, the average age for all PI-RADS scores lies in the $60-70$'s range. Addition of a new record with an extreme value will alter the query sensitivity, and PyDPLib will select an ε that offers the desired noise addition with low impact on overall distribution.

`query_type` 3: Median. Figure 4 shows the median patient age versus Prostatitis. As shown in Table 2, Prostatitis is a categorical attribute (`is_range = True`) with values: `chr` (chronic), `y` (yes), `n` (no), `acute` and `acute on chronic`. These categorical variables are mapped to numerical values and passed to PyDPLib. After noising, these numerical values are mapped to original categories and visualized as a box plot with median bars. The resultant plot is an example of x-versus-y plot with noised x-axis. Alternatively, we can noise the ages and visualize them with respect to Prostatitis. Similarly, we can create many interesting statistical plots for various data types by using PyDPLib with any visualization software.

6. Conclusion

Medical institutions that want to share cohort statistics with external parties, e.g. pharmaceutical companies, have to protect the privacy of the individual patients that contributed to the underlying data. We have presented PyDPLib, a differential privacy library for private medical data analytics that greatly simplifies this task. PyDPLib offers different common statistical operations, such as histograms and mean, and noises the input data according to the type of operation, the data type, and the privilege level of the intended end user.

Based on a use case with real data, we have shown that PyDPLib allows creating statistical data visualizations that preserve the underlying data distributions without compromising the privacy of the individuals. Although developed for concrete use in our reporting platform, PyDPLib is general enough to be used in any data analytics or visualization application that requires differential privacy.

References

[1] Privacy by Design — General Data Protection Regulation (GDPR). [Online]. Available: https://gdpr-info.eu/issues/privacy-by-design/

[2] D. Hedin and A. Sabelfeld, "A perspective on information-flow control." *Software safety and security*, vol. 33, pp. 319–347, 2012.

[3] S. Imtiaz, R. Sadre, and V. Vlassov, "On the case of privacy in the IoT ecosystem: A survey," in *International Conference on Internet of Things (iThings)*. IEEE, 2019, pp. 1015–1024.

[4] R. Tso *et al.*, "Privacy-preserving data communication through secure multi-party computation in healthcare sensor cloud," *Journal of Signal Processing Systems*, vol. 89, no. 1, pp. 51–59, 2017.

[5] C. Dwork, A. Roth *et al.*, "The algorithmic foundations of differential privacy," *FnT-TCS*, vol. 9, no. 3-4, pp. 211–407, 2014.

[6] Google's differential privacy libraries. [Online]. Available: https://github.com/google/differential-privacy

[7] Openmined/pydp: A python wrapper for google's differential privacy libraries. [Online]. Available: https://github.com/OpenMined/PyDP

[8] N. Holohan *et al.*, "Diffprivlib: the ibm differential privacy library," *arXiv preprint arXiv:1907.02444*, 2019.

[9] Diffprivlib: The IBM Differential Privacy Library. [Online]. Available: https://github.com/IBM/differential-privacy-library/

[10] dp-stats: A library for differentially private statistics and machine learning algorithms. [Online]. Available: https://gitlab.com/dp-stats/

[11] J. Xu, Z. Zhang, X. Xiao, Y. Yang, and G. Yu, "Differentially private histogram publication," in *IEEE ICDE*, 2012, pp. 32–43.

[12] OpenDP tools. [Online]. Available: https://opendp.org/

[13] American College of Radiology. (2019) Prostate Imaging Reporting & Data System (PI-RADS). [Online]. Available: https://www.acr.org/Clinical-Resources/Reporting-and-Data-Systems/PI-RADS

[14] Plotly python graphing library. [Online]. Available: https://plotly.com/python/is-plotly-free/

[15] C. Dwork, F. McSherry, K. Nissim, and A. Smith, "Calibrating noise to sensitivity in private data analysis," in *Theory of cryptography conference*. Springer, 2006, pp. 265–284.

[16] Q. Geng and P. Viswanath, "The optimal noise-adding mechanism in differential privacy," *IEEE Transactions on Information Theory*, vol. 62, no. 2, pp. 925–951, 2015.

[17] Y. Wang, X. Wu, and D. Hu, "Using randomized response for differential privacy preserving data collection." in *EDBT/ICDT Workshops*, vol. 1558, 2016.

[18] R. Cummings and D. Durfee, "Individual sensitivity preprocessing for data privacy," in *Proceedings of the 14th Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2020, pp. 528–547.

[19] M. Bun and T. Steinke, "Average-case averages: Private algorithms for smooth sensitivity and mean estimation," *arXiv preprint arXiv:1906.02830*, 2019.

[20] S. Tu. (2013) Introduction to Differential Privacy. [Online]. Available: https://stephentu.github.io/writeups/6885-lec20-b.pdf